Contents

Marketing
Finally

Erlang Tutorial

Joe Armstrong (joe.armstrong@telia.com)

Florence

2 September 2001

(version 1.0)

Plan

- Erlang.
- Plan.
- What is Erlang?
- History.
- Essential Characteristics.
- Erlang - Background.
- Erlang - Properties.
- Sequential Erlang in 5 examples.
- Primitives for concurrency and distribution.
- Concurrent Erlang in 3 examples.
- Distributed Erlang in 1 example.
- Fault tolerant Erlang in 3 examples.
- Hot code replacement Erlang in 1 example.
- Generic Client-Server.
- Paramaterising the Server.
- Comments.
- Technique.

- Products in Erlang.
- OTP.
- Open Source Erlang.
- Development.
- Thoughts.
- The Bluetail Story.
- Marketing.
- Finally.

What is Erlang?

- The result of a technology transfer effort to transfer some of the best ideas in FP/Logic programming into an industrial context.
- A language for programming *distributed fault-tolerant soft real-time non-stop applications.*
- A set of well-tested libraries for programming *distributed...*
- A set of programming patterns for programming *distributed...*
- A set of routines for programming *distributed...*
- An application OS for delivering *distributed...*
- A rapid application delivery platform for programming *distributed...*
- A functional programming language.

"functional" is deliberately last in this list :-)

What it is not

- A research vehicle.
- A language for efficient sequential computation.

History

- Pre 1986 - Programming experiments - how to program a telephone exchange.
- 1986 - Erlang emerges as dialect of Prolog. Implementation is a Prolog interpretor - 1 developer (Joe).
- 1989 - 3 developers (Mike, Robert, Joe), 10 Users. Own abstract machine (JAM)
- 1993 - Erlang systems founded (25 people).
- 1996 - OTP formed. AXD301 development starts.
- 1998 - Erlang banned within Ericsson for new products.
- 1998 - Open source Erlang.
- 1998 - Erlang "fathers" quit Ericsson. Starts Bluetail.
- 2000 - Blutail sold to Alteon Web systems.
- 2000 - Alteon web systems sold to Nortel Networks
- 2001 - Nortel produces SSL accelerator (best in test),

  http://www.networkcomputing.com/1212/1212f46.html + ISD platform.
- 2001 - Erlang (Alteon) group is "down-sized".

Essential Characteristics

These are essential:

- Change code in a running system.
- Dynamic sizes of all objects.
- Fast context switching/message passing.
- Low memory overhead per process/task.
- Thousands of processes.
- No memory leaks/fragmentation.
- No "global" errors. Stop errors propagating.
- Methods to be able to recover from SW and HW errors.
- Simple language, easy to learn.
- Predictable performance.
- Easy to port/implement.

Non essential

- Static type system.
- "Pure".
- Lazy.

Erlang - Background

Background:

- Computer Science Lab founded 1983.
- Experiments with:Ada, C, concurrent Euclid, Eri-Pascal, CLU, ML, CML, LPL, PFL, Hope, Prolog, OPS5, *with real telecom hardware* .
- Solve "essential characteristics".
- Use standard OS.
- Use standard processors.
- Distributed system.
- High level language.

Erlang - Properties

- Functional/single assignment.
- Light weight processes.
- Asynchronous message passing (send and pray).
- OS independent.
- Special error handling primitives.
- Lists, tuples, binaries.
- Dynamic typing (an optional soft typing system is being developed).
- Real-time GC.

Sequential Erlang in 5 examples

1 - Factorial

```
-module(math).
-export([fac/1]).

fac(N) when N > 0 ->  N * fac(N-1);
fac(0)            ->  1.

> math:fac(25).
15511210043330985984000000
```

2 - Binary Trees

```
lookup(Key, {Key, Val, _, _}) ->
    {ok, Val};
lookup(Key,{Key1,Val,S,B}) when Key < Key1 ->
    lookup(Key, S);
lookup(Key, {Key1,Val,S,B}) ->
    lookup(Key, B);
lookup(Key, nil) ->
    not_found.
```

3 - Append

```
append([H|T], L) -> [H|append(T, L)];
append([],    L) -> L.
```

4 - Sort

```
sort([Pivot|T]) ->
    sort([X||X <- T, X <- Pivot]) ++
    [Pivot] ++
    sort([X||X <- T, X >= Pivot]);
sort([]) -> [].
```

5 - Adder

```
> Adder = fun(N) -> fun(X) -> X + N end end.
#Fun
> G = Adder(10).
#Fun
> G(5).
15
```

Primitives for concurrency and distribution

spawn

```
Pid = spawn(fun() -> loop(0) end).
```

send and receive

```
Pid ! Message,
.....

receive
   Message1 ->
      Actions1;
   Message2 ->
      Actions2;
   ...
after Time ->
   TimeOutActions
end
```

Distribution

```
...
Pid = spawn(Fun@Node)
...
alive(Node)
...
not_alive(Node)
```

Concurrent Erlang in 3 examples

1 - "area" server

```
-module(math).
-export([fac/1]).

start() ->
    spawn(fun() -> loop(0) end).

loop(Tot) ->
    receive
        {Pid, {square, X}} ->
            Pid ! X*X,
            loop(Tot + X*X);
        {Pid, {rectangle,[X,Y]}} ->
            Pid ! X*Y,
            loop(Tot + X*Y);
        {Pid, areas} ->
            Pid ! Tot,
            loop(Tot)
    end.
```

2 - Area client

```
Pid ! {self(), {square, 10}},
receive
    Area ->
        ...
end
```

3 - Global Server

```
...
Pid = spawn(Fun),
register(bank, Pid),
...
bank ! ...
```

Distributed Erlang in 1 example

```
...
Pid = spawn(Fun@Node)
...
alive(Node)
...
not_alive(Node)
```

Fault tolerant Erlang in 3 examples

1 - catch

```
> X = 1/0.
** exited: {badarith, divide_by_zero} **
> X = (catch 1/0).
{'EXIT',{badarith, divide_by_zero}}
> b().
X = {'EXIT',{badarith, divide_by_zero}}
```

2 - Catch and throw

```
case catch f(X) ->
    {exception1, Why} ->
        Actions;
    NormalReturn ->
        Actions;
end,

f(X) ->
   ...
   Normal_return_value;
f(X) ->
   ...
   throw({exception1, ...}).
```

3 - Links and trapping exits

```
process_flag(trap_exits, true),
P = spawn_link(Node, Mod, Func, Args),
receive
    {'EXIT', P, Why} ->
        Actions;
    ...
end
```

Hot code replacement Erlang in 1 example

Here's the server:

```
loop(Data, F) ->
    receive
        {request, Pid, Q} ->
            {Reply, Data1} = F(Q, Data),
            Pid ! Reply,
            loop(Data1, F);
        {change_code, F1} ->
            loop(Data, F1)
    end
```

To do a code replacement operation do something like:

```
Server ! {change_code, fun(I, J) ->
                         do_something(...)
                       end}
```

The (real-time) garbage collector removes F!

Generic Client-Server

```
start(Name, Data, Fun) ->
    register(Name,
             spawn(fun() ->
                       loop(Data, Fun)
                   end)).

rpc(Name, Q) ->
    Tag = ref(),
    Name ! {query, self(), Tag, Q},
    receive
        {Tag, Reply} -> Reply
    end.

loop(Data, F) ->
    receive
        {query, Pid, Tag, Q} ->
            {Reply, Data1} = F(Q, Data),
            Pid ! {Tag, Reply},
            loop(Data1, F)
end.
```

Paramaterising the Server

```
start() -> cs:start(keydb, [], fun handler/2).

add(Key, Val) -> cs:rpc(keydb, {add, Key, Val}).
lookup(Key)   -> cs:rpc(keydb, {lookup, Key}).

handler({add, Key, Val}, Data) ->
    {ok, add(Key,Data)}.
handler({lookup, Key}, Data)   ->
    {find(Key, Data), Data}.

add(Key,Val,[{Key, _}|T]) -> [{Key,Val}|T];
add(Key,Val,[_|T])        -> [H|add(Key,Val,T)];
add(Key,Val,[])           -> [{Key,Val}].

find(Key,[{Key,Val}|_]) -> {found, Val};
find(Key,[H|T])         -> find(Key, T);
find(Key,[])            -> error.
```

- Sequential.
- Can be typed.
- Isolates (concurrent + error handling + ...) code from sequential code.

Comments

Why is this nice?

- We can structure the system so that 95% of the code is written as client code and 5% as “concurrency patterns”.
- We could type check the client code.
- We cannot type check the generic code.
- The generic code is written and tested by “experts”.
- Client code written by applications programmers.
- 10 patterns suffice for almost all know patterns of concurrency. Client-server, Worker-supervisor, event-handler, upgrade-handler, keep-me-alive, hot-standby.

Technique

- 1986 - 1989 Prolog interpretor.
- 1988 - JAM.
- 1989 - Vee.
- 1992 - Beam.
- 1995 - Types.
- 1996 - Hype.
- 1997 - Erlang97, Standard.
- 1998? - FPGA.

Products in Erlang

- 1986 - 1988 ACS/Dunder (Ericsson).
- 1988 - 1993 Many small projects.
- 1992 - 1995 MOB (Ericsson).
- 1992 - 1994 A few medium projects (NetSim, Teletrain, ..) (Ericsson).
- 1996 - ATM, Elvira (MOB2)(Ericsson).
- 1998 - AXD301 (Ericsson).
- 1999 - GPRS (Ericsson).
- 2000 - Mail robustifier (Bluetail).
- 2001 - ISD platform, SSL accelerator (Nortel/Alteon).

OTP

What is OTP?

OTP stands for **Open Telecom Platform** . OTP is a "middleware platform for building high-availability, fault-tolerant, distributed, soft real-time, applications.

- A large number of libraries.
- A collection of *behaviors* (programming patterns) which encapsulate common behavorial patterns. For example, client-server, supervision-tree, ...
- A set of *applications* - completed software components that can be plugged together to perform complex tasks. For example, eva - a distributed event and history logging infrastructure, Corba, ...
- Similar in scope to **.NET** - but limited to one language (Erlang).
- Available from *http://www.erlang.org/*.
- "Open source" license *(do what you want)*.

Open Source Erlang

- OSE - is the *same* Erlang release that Ericsson uses in its products. For example, AXD301 GRPS etc.
- Produced by Ericsson OTP group - with external inputs :-).
- Used in several Ericsson products (AXD301, GRPS etc.) - and in a number of new Nortel products (SSL accelerator, ISD platform etc.).
- Highly mature implementation - i.e. the *first* public Erlang release (1998) had already been proved in several commercial products (Mobility server etc.) - The ERTS (Erlang Run Time System) might inspire anyone interested in implementation issues for systems offering concurrency together with garbage collected languages (for example Java or CIL compiled languages in **.NET**).
- Has demonstrated long term performance reliability. Possibly years of non-stop operation (nobody really knows :-).

Development

- 1986 – 1 developer, 0 users, 0 support.
- 1989 – 3 developers, 10 users, 0.5 support.
- 1991 – 4 developers, 40 users, 1.0 support.
- 1993 – Erlang systems founded. ES grows from 3 - 25 people in 3 years.
- 1996 – OTP founded. Grows to 30 in 2 years.
- 1997 – 10 developers. 300 Erlang programmers (1000 total project employed). 5 big (100+) projects. Many small (< 20) projects.

Needed Erlang Systems to expand. Courses/consulting vital for first phase of expansion.

Needed OTP to get into Ericsson mainstream. Needed good documentation, professional project management and revision control.

If it hasn't got a part number it doesn't exist.

We still did everything ourselves but we got more help.

Thoughts

- The “gap” – the best that research has to offer and the minimum acceptable by industry is too large.
- You need good support. e-mail, telephone, consulting (days - years).
- Good documentation costs money.
- *To displace an existing technology you have to wait for something to fail* .
- Step into the vacuum after a crisis has occurred – look for the gaps.
- Use “satisfied users” to sell to new users (credibility).
- Don’t fight, you never win, you only loose.
- Ditch committees, pre-studies, reports – find the hero programmer.
- Talk, talk, talk, talk to the hero programmer (not telephone, e-mail etc.
- Put all development on one site (corridor).

The Bluetail Story

- 1998 - Bluetail was formed by the Erlang "fathers" (except Mike Williams, who stayed on in Ericsson and in now a "big boss") + Jane Walerud.
- Business idea - *Bringing reliability to the Internet* .
- 1999 - First product (BMR = Bluetail Mail Robustifier) - programmed in Erlang in three months from scratch.
- BMR programmed using a generic "reliable, high-availability" behavior - a behavior that can be paramaterised with 17 different funs.
- 1999 - BMR sold to Telnordia (Swedens 3’rd biggest ISP) handles all Telnordia e-mail.
- 2000 - Bluetail sold to Alteon web systems for 1.4B SEK. They were after the technology (Erlang).
- 2000 - Alteon sold to Nortel networks for 7.8B USD.
- 2000 - Jane Walerud - Swedish "IT person of the year".
- 2001 - The death the the dotcoms - downsized. Nortel writes off 8B USD for the Alteon purchase.

Marketing

Don’t

- tell them its a PL.
- use the word declarative (they might ask you what it means!).
- use the word functional.
- confuse them with measurements and facts.
- claim you can do everything (you can’t).

Emphase

- time to market (it’s shorter).
- total life cycle costs (reduced).
- total cost of ownership (reduced).
- the IPSE, or IDE (don’t use the word “emacs”).
- the re-usable components, or API’s (don’t call them libraries).

Use latest buzzwords

There is a “performance gap” – but we try to run on the fastest available processors, then the gap is less of a problem. We are “sufficiently fast”

Finally

- Concentrate on *essential features* .
- You will never displace an existing technology if it works – *Wait for the failures* .
- Move *quickly* into the vacuum after a failure.
- Develop new unchallenged application areas.
- 5% of all real system software sucks – don't worry. Ship it and improve it later.
- FP is a *here and now technology* – companies using FP will demonstrate real commercial advantage over those using conventional technology
- You need a *business infrastructure* (People expert in Business development, Marketing, Sales, Lawyers, ...) to succeed.
- Writing a business plan is just like writing a research proposal.
- Writing a patent plan is just like writing a conference paper.
- Move towards the mainstream.
- Don't be shy asking for money - remember it is the programmers who are the heros - we invented the Internet.
- Nurture your VCs, lawyers, business people. Explain to them how it works, in terms that they can understand. Be very patient.
- Do fun stuff.
- Have fun.