Erlang

Bits of History
Words of Advice
of
Ericsson
Products

Joe Armstrong

Computer Science Laboratory
Ericsson Telecommunications AB
Sweden
`joe@cslab.ericsson.se`

22 October 1998

Plan

- Me
- What is Erlang
- The dark ages
- AXE10 features
- Essential characteristics
- Erlang examples
- The Erlang “wars”
- Other attempts
- Technical milestones
- Users
- The Business cycle
- Openness
- Marketing Erlang (trends)
- Growth
- Thoughts/Future/Lessons learnt

Me

- Joe Armstrong **joe@cslab.ericsson.se**
- B.Sc Physics UCL 1972
- Ph.D (Failed!) High Energy Physics
- Researcher at MIRU Edinburgh 1977
- Chief Programmer EISCAT 1978-1982
- Swedish Space Corporation 1982-1986 (Wrote 90% of software for Viking Satellite)
- Ericsson Computer Science Lab. 1986 - today - invented Erlang

So what is Erlang/OTP?

- it's a new programming language (Erlang)
- it's a new way of developing telecoms applications (OTP)

and:

- it's battle tested in 9 Ericsson products. Mobility server, AXD301 ...
- 10 More products are one the way ...
- it's the biggest FPL used to earn money in the world ...
- it was developed and is maintained by a very small team ...
- it's good stuff ...

The Dark Ages

Late 1960's – AKE Tumba

- One Small CPU for everything
- Small amount of "core" memory
- Spaghetti

Early 1970's – AKE-13 Many installations, ARE-11 Many installations

- Multi processor (home made)
- Small amounts of "core" memory
- Spaghetti
- Addressing errors
- Must stop system to upgrade
- inflexible

Late 1970's – AXE-10 Thousands of installations

- Single/duplicated central processor
- Many duplicated regional processors
- High level (??) language PLEX
- Robust SW
- Simple concept (Blocks/signals)
- Great effort needed to write/modify SW

AXE-10 features

- Can Add/change/delete SW without stopping the system
- Fast:
  - Context switching
  - Message passing
  - PLEX machine
  - CPU off-loaded by DP's
- Size changes possible
- Data encapsulation with MMU support
- Very low level programming:
  - Goto's
  - No proper subroutines/functions
  - No libraries
  - Cut and paste programming
- Not main stream, must fix own:
  - Education
  - Processors
  - Compilers
  - Debuggers
- SW support on large main-frames (IBM, UNIVAC).

Essential Characteristics

- 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

Don't care about expressiveness, lazyness ...

Erlang

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

- 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
- + Easy to learn
- + Easy to port to new OS
- + Small SW volume = reduced design time
- - Needs more memory/processor power
- - Non standard

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 Tree

```
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
```

Concurrent and distributed Erlang

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)
```

Wars

- < 1970 AXE10 wars. PLEX wins - it's the only thing that works let's use it
- < 1988 Ignored
- 1990 – War starts. C++. Irritation. "It'll never work"
- 1994 – Threat. Heavy fighting. "It might work"
- 1995 – C++ project collapses. We won. "It does work"
- 1996 – Peace. Expansion. New division
- 1997 – Java wars start. "Can you write a paper comparing Java and Erlang?"
- 1998 – Java wars continue

You need a protector

Other Attempts

- Chill - abandoned
- ISO 9000 - useless
- TOS - abandoned
- AP's - started to be used
- APS tools - support system on SUN - OK
- High level PLEX - some improvements

But – basic technology (Blocks/signals/PLEX) unchanged.

The Solution

C++ with object orientation and standard processors

Variants:

Own Operating System

- Light-weight threads
- heavy-weight processes
- Code change while running
- Own distribution protocols
- No GC

Standard OS

- Unix
- Threads
- Sockets (TCP/IP, UDP)

Standard OS

Everything in one unix process (Erlang)

TOTAL FAILURE

- Memory leaks
- Memory fragmentation
- Huge amounts of work
- Hundreds of programmers
- Unfinished/Abandoned projects
- Ravioli Programming
- C++ and OO ok for some projects, but impossible for large real-time systems

Failed because languages did not solve essential problems (concurrency, exceptions ...)

Techniques

- 1986 - 1989 Prolog interpretor
- 1988 - JAM
- 1989 - Vee
- 1992 - Beam
- 1995 - Types
- 1996 - Hype
- 1998 - Erlang 4.6 standard
- 1998 - Erlang 5.0 standard
- 1998? - FPGA

Users

- 1986 - 1988 ACS/Dunder. Bollmora
- 1988 - 1993 Many small projects
- 1992 - 1995 MOB
- 1992 - 1994 A few medium projects (NetSim, Teletrain, ..)
- 1996 - ATM, Elvira (MOB2) (Now selling 40/month)
- 1997 - AXD301 (scalable 10-160 GBit/sec ATM switch)

What happens between the different projects?
There is a period of "re-charging" between projects.

The Business cycle

This doesn't happen:

- Define Problem
- Study alternative solutions
- Choose Best alternative
- Make product

This does happen:

- Don't define the problem
- Choose Hyped Technology (C++, Java) after pseudo discussion
- Do no experiments
- Go ahead with main product without building a prototype
- Project fails
- *Help – save us*
- New technology saves the day
- (1 year later) old technology starts fighting back

Openness

- 1800's. Ericsson made **everything** itself. There was even a factory for making nuts and bolts. Now it's outsourcing.
- 1974. PLEX (Proprietary = good thing).
- 1988. Erlang is "secret". (Propriety = good thing). "Don't give away our secrets to the opposition".
- 19?? - WWW - Netscape - Java - ...
- 1996 - Erlang is proprietary. (Proprietary = Bad thing = isolation). Give it away. Free Erlang.
- 1998 - Open source Erlang. Mozilla public license.
- 1998 - Eddie.

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
- 1998 – 40 developers + support staff. 400 Erlang programmers. 12 products. 250-300 downloads/day.

Needed ES to expand. Courses/consulting vital for first phase of expansion.
Needed OTP to get into mainstream. Good Documentation. PRIM/GASK. Clearcase. Revision control.
If it hasn't got a part number it doesn't exist.
Magnificent support from the Organisation.
Tuula.
We still do everything ourselves but we get more help.

Marketing

1988 – Propeller heads

- Declarative
- FP
- Short programs

1992 – Mid life crisis

- Not a language
- Erlang “tools”
- Integrated environment
- Platform

1997 – Suit tie - smile

- Time to market
- Quality
- Interoperability
- Careware - biggness

2000 – It’s a programming language.
We had a salesman - he quit!

Thoughts

- The “gap” – the best that research has to offer and the minimum acceptable by industry is too large
- 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)

How to market Erlang to managers

Don'ts

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

Do tell them about

- Time to market (shorter)
- Total life cycle costs (reduced)
- The IPSE, or IDE (don't use the word "emacs")
- The re-usable components, or API's (don't call them libraries)

Already Erlang is being chosen instead of C++, Java even C because of "time to market" – we have examples of pre-study to product of nine months. We are now aiming at development times of less than one year. Here we have a significant advantage over conventional imperative languages.

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"

Lessons learnt

- Concentrate on *essential features*
- You need a protector
- 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 for real

Joe Armstrong
Ericsson Computer Science Laboratory