An Introduction to Erlang

From behind the trenches...

GOTO Copenhagen
May 13th, 2011

Francesco Cesarini
Founder, Technical Director

@FrancescoC
francesco@erlang-solutions.com
So Here I Am....
The Computer Science Lab
Telecom Applications: Issues

Complex
No down time
Scalable
Maintainable
Distributed

\textbf{vs}

Time to Market
The Ancestors

Languages like SmallTalk, Ada, Modula or Chill

Functional languages like ML or Miranda

Logical languages like Prolog
Erlang Highlights

- Declarative
- Concurrent
- Robust
- Distributed
- Hot code loading
- Multicore Support
- OTP

Functional programming language
High abstraction level
Pattern matching
Concise readable programs
Erlang Highlights: Factorial

Factorial using Recursion

Definition

\[ n! = \begin{cases} 
1 & n = 0 \\
n \times (n-1)! & n \geq 1
\end{cases} \]

Implementation

```erlang
-module(ex1).
-export([factorial/1]).
factorial(0) ->
1;
factorial(N) when N >= 1 ->
N * factorial(N-1).
```

Eshell V5.0.1  (abort with ^G)
1> c(ex1).
{ok,ex1}
2> ex1:factorial(6).
720
QuickSort using List Comprehensions

-module(ex2).
-export([qsort/1]).

qsort([Head|Tail]) ->
    First = qsort([X || X <- Tail, X =< Head]),
    Last  = qsort([Y || Y <- Tail, Y > Head]),
    First ++ [Head] ++ Last;
qsort([]) ->
    [].

Eshell V5.0.1 (abort with ^G)
1> c(ex2).
{ok,ex2}
2> ex2:qsort([7,5,3,8,1]).
[1,3,5,7,8]

"all objects Y taken from the list Tail, where Y > Head"
Parsing a TCP packet using the Bit Syntax

<< SourcePort:16, DestinationPort:16, SequenceNumber:32,
   AckNumber:32, DataOffset:4, _Reserved:4, Flags:8,
   WindowSize:16, Checksum:16, UrgentPointer:16,
   Payload/binary>> = Segment,

OptSize = (DataOffset - 5)*32,
<< Options:OptSize, Message/binary >> = Payload,
<< CWR:1, ECE:1, URG:1, ACK:1, PSH:1,
   RST:1, SYN:1, FIN:1>> = <<Flags:8>>,

%% Can now process the Message according to the
%% Options (if any) and the flags CWR, ..., FIN

etc...
Erlang Highlights

Declarative

Concurrent

Robust

Distributed

Hot code loading

Multicore Support

OTP

Either transparent or explicit concurrency
Light-weight processes
Highly scalable
Creating a new process using spawn

-module(ex3).
-export([activity/3]).

activity(Name, Pos, Size) ->
    ................

Pid = spawn(ex3, activity, [Joe, 75, 1024])

activity(Joe, 75, 1024)
Erlang Highlights: Concurrency

Process creation time

Source: Joe Armstrong
SICS

© 2011 - Erlang Solutions Ltd.
Processes communicate by asynchronous message passing

Pid ! {data,12,13}

receive
  {start} -> ..........  
  {stop}  -> ..........  
  {data,X,Y} -> ..........  
end
Erlang Highlights: Concurrency

Message passing times

Source: Joe Armstrong, SICS

© 2011 - Erlang Solutions Ltd.
Erlang Highlights

- Declarative
- Concurrent
- Robust
- Distributed
- Hot code loading
- Multicore Support
- OTP

Simple and consistent error recovery
Supervision hierarchies
"Program for the correct case"
Cooperating processes may be linked together using

\[
\text{using}
\begin{align*}
\text{spawn\_link}(\ldots, \ldots, \ldots) \\
\text{or} \\
\text{link}(\text{Pid})
\end{align*}
\]
Erlang Highlights: Robustness

When a process terminates, an exit signal is sent to all linked processes.

... and the termination is propagated.
Exit signals can be trapped and received as messages

receive
  \{‘EXIT’,Pid,...\} -> ...
end
Robust systems can be built by layering

“Supervisors”

“Workers”
Erlang Highlights

Declarative
Concurrent
Robust
Distributed
Hot code loading
Multicore Support
OTP

Explicit or transparent distribution
Network-aware runtime system
Erlang Highlights: Distribution
Simple Remote Procedure Call

```
{rex, Node} ! {self(), {apply, M, F, A}},
  receive
    {rex, Node, What} -> What
  end

loop() ->
  receive
    {From, {apply, M, F, A}} ->
      Answer = apply(M, F, A),
      From ! {rex, node(), Answer},
      loop();
    _Other -> loop()
  end.
```
Erlang Highlights

Declarative
Concurrent
Robust
Distributed
Hot code loading
Multicore Support
OTP

Easily change code in a running system
Enables non-stop operation
Simplifies testing
Erlang Highlights

Declarative
Concurrent
Robust
Distributed
Hot code loading
Multicore Support
OTP

SMP support provides linear scalability out of the box thanks to its no shared memory approach to concurrency.
Ericsson’s strategy with SMP

Make it work -> measure -> optimize

Hide the problems and awareness of SMP from the programmer
Programmed in the normal style using processes for encapsulation and parallelisation
Multicore Erlang

Erlang VM

Scheduler #1
run queue

Scheduler #2
run queue

...}

Scheduler #N
run queue

migration logic
Tilera “Tile64”

Chatty

500 processes created

Each process randomly sends messages and receives a response from all other processes
Multicore Benchmark - Big Bang

Big_bang benchmark, 500 processes chatting
- Bound schedulers
- Default (unbound)

(No Tilera-specific optimizations!)

c. 0.4x @ 48 cores

Erlang/OTP R13B on Tilera Pro 64-core

© 2011 - Erlang Solutions Ltd.
Erlang Highlights

- Declarative
- Concurrent
- Robust
- Distributed
- Hot code loading
- Multicore Support

Provides the design patterns, libraries and tools to develop distributed fault tolerant systems.

OTP
Open Telecom Platform

Applications & Libraries

System Design Principles
OTP: System Design Principles

A set of abstract principles and design rules.
- They describe the software architecture of an Erlang System
- Needed so existing tools will be compatible with them
- Facilitate understanding of the system among teams

A set of generic behaviours.
- Each behaviour is a formalisation of a design pattern
- Contains frameworks with generic code
- Solve a common problem
- Have built in support for debugging and software upgrade
- Facilitate understanding of the sub blocks in the system
Erlang Highlights

Declarative
Concurrent
Robust
Distributed
Hot code loading
Multicore Support
OTP
I wrote my Erlang system in 4 weeks!
The Myths of Erlang....

Is it Documented?

Is the developer supporting it?

What visibility does support staff have into what is going on?

- SNMP
- Live Tracing
- Audit Trails
- Statistics
- CLI / HTTP Interface

How much new code was actually written?
Upgrades during runtime are Easy!
The Myths of Erlang....

Yes, it is easy for
- Simple patches
- Adding functionality without changing the state

Non backwards compatible changes need time time
- Database schema changes
- State changes in your processes
- Upgrades in distributed environments

Test, Test, Test
- A great feature when you have the manpower!
We achieved 99.99999999 availability!
The Myths of Erlang....

“As a matter of fact, the network performance has been so reliable that there is almost a risk that our field engineers do not learn maintenance skills”

Bert Nilsson, Director NGS-Programs Ericsson

Ericsson Contact, Issue 19 2002
99,999 (Five Nines) is a more like it!
- Achieved at a fraction of the effort of Java & C++

Upgrades are risky!

Non Software related issues
- Power Outages
- Networking
- Hardware Faults
The Myths of Erlang....

99,999 (Five Nines) is a more like it!
- Achieved at a fraction of the effort of Java & C++

Upgrades are risky!

Non Software related issues
- Power Outages
- Network Failures, Firewall Configurations
- Hardware Faults
Questions
More Information

Programming Erlang
- Software for a Concurrent World
- by Joe Armstrong

Erlang Programming
- A Concurrent Approach to Software Development
- by Francesco Cesarini & Simon Thompson

Erlang and OTP in Action
- Large-scale software design with OTP
- by Richard Carlsson, Martin Logan & Eric Merrit
Thank You!

@FrancescoC on Twitter or francesco@erlang-solutions.com