DM8XX Proposal Talk
Subject 8: Erlang

Bjørn Madsen

IMADA, SDU

April 29, 2009

What is Erlang?

Erlang is...

What is Erlang?

Erlang is...

- a general purpose programming language and runtime environment

What is Erlang?

Erlang is...

- a general purpose programming language and runtime environment
- built for concurrency, distribution and fault tolerance

What is Erlang?

Erlang is...

- a general purpose programming language and runtime environment
- built for concurrency, distribution and fault tolerance
- a functional programming language with syntax reminiscent of Prolog

Functional programming

Functional programming

- Variables start with a capital letter: X, Item, People, Count

Functional programming

- Variables start with a capital letter: X, Item, People, Count
- Atoms are names, and start with a non-capital letter: apple, item, job

Functional programming

- Variables start with a capital letter: X, Item, People, Count
- Atoms are names, and start with a non-capital letter: apple, item, job
- Lists contain a sequence of other elements, of any type: [1,X,ok], [], [2,4,8,16]

Functional programming

- Variables start with a capital letter: X, Item, People, Count
- Atoms are names, and start with a non-capital letter: apple, item, job
- Lists contain a sequence of other elements, of any type: [1,X,ok], [], [2,4,8,16]
- Tuples group related elements into one unit: {name, "John", "Doe"}, {count, apples, 255}

Functional programming

- Variables start with a capital letter: X, Item, People, Count
- Atoms are names, and start with a non-capital letter: apple, item, job
- Lists contain a sequence of other elements, of any type: [1,X,ok], [], [2,4,8,16]
- Tuples group related elements into one unit: {name, "John", "Doe"}, {count, apples, 255}
- Variables can be bound to a value only once

Functional programming

- Variables start with a capital letter: X, Item, People, Count
- Atoms are names, and start with a non-capital letter: apple, item, job
- Lists contain a sequence of other elements, of any type: [1,X,ok], [], [2,4,8,16]
- Tuples group related elements into one unit: {name, "John", "Doe"}, {count, apples, 255}
- Variables can be bound to a value only once
- Strings are just lists of integers representing printable ascii-characters

Hello, world

"Hello, world" in Erlang.

Hello, world

"Hello, world" in Erlang.

First an erlang module, `hello.erl`:

```
-module(hello).
-export([hello_world/0]).

hello_world() -> io:format("Hello, world~n").
```

Hello, world

"Hello, world" in Erlang.

First an erlang module, `hello.erl`:

```
-module(hello).
-export([hello_world/0]).

hello_world() -> io:format("Hello, world~n").
```

Then we use it from the erlang shell:

```
1> c(hello). %% Compile the module hello
{ok,hello}
2> hello:hello_world(). %% Call the hello_world/0 function
Hello, world
ok
```

A larger example

Shopping list

A larger example

Shopping list

Suppose we have a shopping list, comprised of tuples of the item we need ot buy and the amount we need ot buy of it:

```
Buy = [{apples, 5},{newspaper, 1},{oranges, 42}].
```

A larger example

Shopping list

Suppose we have a shopping list, comprised of tuples of the item we need ot buy and the amount we need ot buy of it:

```
Buy = [{apples, 5},{newspaper, 1},{oranges, 42}].
```

We also have a mapping from an item to the price of a single unit of that item:

```
cost(apples) -> 3;
cost(newspaper) -> 15;
cost(oranges) -> 2.
```

A larger example

Shopping list

Suppose we have a shopping list, comprised of tuples of the item we need ot buy and the amount we need ot buy of it:

```
Buy = [{apples, 5},{newspaper, 1},{oranges, 42}].
```

We also have a mapping from an item to the price of a single unit of that item:

```
cost(apples) -> 3;
cost(newspaper) -> 15;
cost(oranges) -> 2.
```

We can use this to compute the total price of the items we have to buy:

```
Total = cost(apples) * 5 + cost(newspaper) * 1 + cost(oranges
```

A larger example

Computing the total

How can we do this smarter

A larger example

Computing the total

How can we do this smarter

```
total([{What, N}|T]) -> cost(What) * N + total(T);
total([]) -> 0.
```

A larger example

Computing the total

How can we do this smarter

```
total([{What, N}|T]) -> cost(What) * N + total(T);
total([]) -> 0.
```

But it can be done shorter with functional programming tools map, and sum:

A larger example

Computing the total

How can we do this smarter

```
total([{What, N}|T]) -> cost(What) * N + total(T);
total([]) -> 0.
```

But it can be done shorter with functional programming tools map, and sum:

```
total(L) ->
sum(map(fun({What, N}) -> cost(What) * N end, L)).
```

A larger example

Computing the total

How can we do this smarter

```
total([{What, N}|T]) -> cost(What) * N + total(T);
total([]) -> 0.
```

But it can be done shorter with functional programming tools map, and sum:

```
total(L) ->
sum(map(fun({What, N}) -> cost(What) * N end, L)).
```

If we use list comprehensions it can be done even shorter still:

```
total(L) ->
sum([cost(What) * N || {What, N} <- L]).
```

Distributed programming

Passing messages in Erlang is easy:

```
Receiver_PID ! Any_valid_erlang_term
```

Distributed programming

Passing messages in Erlang is easy:

```
Receiver_PID ! Any_valid_erlang_term
```

Receiving them is just as easy:

```
loop() ->
receive
{ok, _} -> loop();
{error, Msg} -> io:format("Error: ~w~n", [Msg])
end.
```

Distributed programming

Passing messages in Erlang is easy:

```
Receiver_PID ! Any_valid_erlang_term
```

Receiving them is just as easy:

```
loop() ->
receive
{ok, _} -> loop();
{error, Msg} -> io:format("Error: ~w~n", [Msg])
end.
```

Since the PID of an Erlang light-weight process includes information on the node at which it is running, a PID is enough to send a message to a process on another computing node.

My project

A distributed TSP-solver

As part of an earlier project a distributed TSP-solver in Java was developed.
In this project I will develop a distributed TSP-solver in Erlang and compare it to the Java-version.

My project

A distributed TSP-solver

As part of an earlier project a distributed TSP-solver in Java was developed.
In this project I will develop a distributed TSP-solver in Erlang and compare it to the Java-version.

- Compare speed on a single, multi-core computer

My project

A distributed TSP-solver

As part of an earlier project a distributed TSP-solver in Java was developed.
In this project I will develop a distributed TSP-solver in Erlang and compare it to the Java-version.

- Compare speed on a single, multi-core computer
- Compare speed distributed to several computers

My project

A distributed TSP-solver

As part of an earlier project a distributed TSP-solver in Java was developed.
In this project I will develop a distributed TSP-solver in Erlang and compare it to the Java-version.

- Compare speed on a single, multi-core computer
- Compare speed distributed to several computers
- Comparison of the code, which is easier to understand, which is easier to write?