Erlang

CIS-3030 Programming Languages
Vermont Technical College
Presented By
Peter C. Chapin

Outline

Origins and History

Overview of Features

Sequential Programming

Concurrent Programming

Conclusions

Origins

Research project of the Ericsson telecommunications company.

“How can one build reliable software systems from components that might contain errors?”

No existing language found to be suitable.

Ericsson develops a new functional language: Erlang.

1998: Ericsson stops in-house software development. Erlang goes open source http://www.erlang.org.

Success Story

In 1998 Ericsson delivers the AXD301 switch.

Probably the largest system ever written in a functional language.

1.7 million lines of Erlang.

Considered the most reliable product Ericsson ever produced.

Feature Overview

Functional with a high degree of purity.

Dynamic Typing

Language level support for concurrency: encourages the use of a large number of simultaneous threads.

Language level support for distributed processing and “hot swapping” of code.

Originally a concurrent dialect of Prolog.

Sequential Erlang

```
-module(geometry).
-export([areas/1]).
-import(lists, [map/2]).

areas(L) ->
  lists:sum(
     map(fun(I) -> area(I) end, L)).

area({square, X})        -> X*X;
area({rectangle, X, Y}) -> X*Y.
```

Interesting Tidbits

Has a “binary” type for holding raw binary data (such as packet headers, etc).

Variables only bound once:

X = 5; X1 = X + 6

X = X + 6 not allowed; X can not be rebound.

“Atoms” are raw names meaningful to the programmer.

Tuples { tcp, “lemuria.cis.vtc.edu”, 9000 } and lists [1, 3, 4, 5] are fundamental.

Strings and records are syntactic sugar.

Quicksort

```
qsort([]) -> [];

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

Philosophy of Erlang

- Each thread in an isolated address space.
  - No shared variables, no complicated synchronization.
  - If one thread malfunctions, it can't affect any other thread. Failures contained!
  - Threads communicate by message passing.
- Thread creation extremely lightweight.
  - Very large number of threads feasible (thousands).
  - No OS support needed (or desired). Erlang runtime is like an OS.

Generic Erlang Server

```
-module(server1).
-export([start/3, stop/1, rpc/2]).

start(Name, F, State) ->
  register(Name,
    spawn(fun() -> loop(Name, F, State) end)).

stop(Name) -> Name ! stop.

rpc(Name, Query) ->
  Name ! {self(), Query},
  receive
    {Name, Reply} -> Reply
  end.

loop(Name, F, State) ->
  receive
    stop -> void;
    {Pid, Query} ->
      {Reply, State1} = F(Query, State),
      Pid ! {Name, Reply},
      loop(Name, F, State1)
  end.
```

Conclusions

Commercially viable functional language. Proven in the real world.

High level abstractions as well as low level access.

Makes concurrent programming straight forward by providing language support for message passing and multiple threads.