The concurrent functional programming language Erlang – An Overview

Dan Sahlin
Computer Science Laboratory
Ericsson Telecom
S-126 25 Stockholm, SWEDEN
e-mail: dan@erix.ericsson.se, phone: +46 8 719 8187
http://www.ericsson.se/cslab/~dan

July 25, 1996

Abstract

The concurrent functional programming language Erlang is now enjoying a more and more widespread use both within Ericsson Telecom, where it was developed, and also outside the company in industry and academia. We here briefly present why it was developed, what it looks like, where it has been used, a few words about the implementations and recent developments.

Keywords

functional programming, concurrent programming, telecom

1 The development of Erlang

Erlang is the result of a consistent search for a better programming tool for telecom applications at the Ericsson Computer Science Laboratory. At the beginning of the 1980:s a large number of programming languages were tested at the lab for controlling a small telephone exchange. From the experiments it became quite clear the symbolic programming languages, such as LISP and Prolog produced the shortest code.

As the programming effort and the maintenance is roughly proportional to the number of lines in a program, it was decided to concentrate on the symbolic programming languages. However, for a telecom application concurrency and error recovery are essential concepts, and the back-tracking of Prolog was of little use. Experiments were also made with Parlog, but the level of concurrency was considered to be a bit too fine-grained.

A decision was therefore made around 1986 to develop a new functional programming language, with emphasis on symbolic programming, concurrency, error recovery, and real-time.

In 1987, the first interpreter for the system, now named Erlang, was written in Prolog. Although the present Erlang system contains no Prolog code, this origin can still be seen in the syntax (e.g. variables are written starting with a capital letter).

Crucial for the development of Erlang was the interaction with the early users within the company. The language was kept as small as possible, only adding features that seemed to be useful. And at times features which were rarely used in actual applications were removed.

Erlang, as all other functional programming languages, features automatic memory management. For a real-time programming language this is a challenge as the system must not stop too long for garbage collection. In a typical Erlang application the parts of the program closest to the hard real time demands are written in C. The rest of the program communicates with the C code as if it had be written in Erlang.

To support the growing number of ERLANG users, the Ericsson company ERLANG SYSTEMS was founded in 1993. The development of the language however is still mainly performed at the Ericsson Computer Science Laboratory.

2 The ERLANG language

Here, a very brief overview of the main features of the language is given. The ERLANG book [1] presents the full language.

Just like Prolog, ERLANG is a dynamically typed language. The data types consist of the constant data types and the compound data types. The constant data types are: numbers (floats and (big) integers), atoms (written with an initial small letter to distinguish them from variables), process identifiers and references. The compound data types are: lists (written as in Prolog) and tuples (written with surrounding braces: e.g. `{123,a,X}`).

A program consists of a number of modules, each consisting of a number of function declarations, some of which are made visible outside the module by an `export` declaration, for example:

```
-module(mathematics).
-export([factorial/1]). % factorial with 1 argument is exported

factorial(0) ->
    1;
factorial(N) when N>0 ->
    N*factorial(N-1).
```

In the head of a clause pattern matching is used, and the definitions are searched sequentially until a match is found. Simple flat conditions (such as `when N>0` above) are also allowed. From another module the function could be called with `mathematics:factorial(X)`.

2.1 Concurrency

Concurrency in ERLANG is introduced through three primitives: `spawn` - for creating a process, send (written with infix '!') - for sending a message and `receive` for receiving messages. The following code, taken from [1], shows how they can be used.

```
-module(counter).
-export([loop/1]).

% The counter loop
loop(Val) ->
    receive
        increment ->
            loop(Val+1);
        {From,value} ->
            From!{self(),Val},
            loop(Val);
        stop ->
            true;
        Other ->
            loop(Val)
    end.
```

During execution last call optimization is performed so it is guaranteed that the stack does not grow.

A typical piece of code that does initialization and uses the counter process looks as follows

```
...
    Pid = spawn(counter,loop,[0]), % spawns counter:loop(0)

    Pid!increment,
    Pid!increment,

    Pid!{self(),value}, % self() returns the process id of this process.
    receive
        {Pid,Value} ->
            Value
    end,
...
```

Distributed computation is almost invisible in Erlang. When a process is spawned, an extra argument indicates on which node it is to be created. The process identifier could then be used as if the process had been local.

Error handling in Erlang is implemented using a catch/throw mechanism.

3 Applications at Ericsson

Quite a number of projects at Ericsson are using Erlang, but we will here focus on the *Mobility Server* which is being sold to customers. This system enables a private branch exchange to implement personal telephone numbers, i.e. the same number can be used for telephony or fax and is routed to right equipment, whether it is a desk phone, a cordless phone, a cellular phone, the home phone, a fax etc. Both the caller and the subscriber may control the connection.

The target system is quite large: 450 Erlang modules totally consisting of 200.000 lines of code. For the low-level and time critical parts 6700 lines of C code are used.

4 Implementations

Presently three core implementations of Erlang exist at the lab: JAM (Joe's Abstract Machine, by Joe Armstrong), VEE (Virding's ErlangEngine, by Robert Virding) and BEAM (Bogdan's Erlang Abstract Machine, by Bogdan Hausman). JAM and VEE are byte code interpreters, and presently it is JAM that is being distributed. BEAM [2], which combines threaded code and compiling to C, is expected to replace JAM as it offers better performance. All Erlang systems share a large part of the run time system and the libraries.

At the universities in Uppsala and Linköping projects are now starting aimed at implementing Erlang even more efficiently.

5 Recent developments

Although the latest edition of the Erlang book is from 1996 [1], some recent developments of the language are not mentioned there.

Lambda expressions have been added to the language. It is remarkable that many large applications have already been developed without lambda expressions, and the users so far seem to have been quite content without them.

The relational data base *Mnesia* has become more integrated into the language, with list comprehensions as the basic interface for the more complex queries. Mnesia features distribution and replication (for speed and robustness), fast key lookup access and optimization of advanced queries (including recursion). The data base resides in primary memory with an optional copy on disk.

An experimental *parallel implementation* of Erlang for multiprocessors with shared memory now exists. Without modification the load of the applications is automatically distributed on the available processors.

An experiment has also been carried out adding *finite domain constraints* to ERLANG [5]. The work showed that although constraint programming does not fit so well into the functional programming framework, it was quite feasible to do. As telecom applications seem to lack the need for constraint solving the system will not be included in current releases of the system.

In Glasgow, Simon Marlow and Philip Wadler are now finishing their ERLANG *type checker*. The user may optionally annotate the program with type information and the type checker will warn about inconsistencies.

Experiments are also been carried out at the lab with *mobile agents*. ERLANG already supports distributed computation well, but the security issues for an open system remain to be solved.

At the lab and ERLANG SYSTEMS many are presently involved into the "Open Telecom Platform" project, which aims to solidify ERLANG making it even easier to build telecom systems on ERLANG.

We are also working towards making the whole core ERLANG system, i.e. the compiler, run time system and essential libraries, more easily available to users in the world.

References

[1] Armstrong, J. L., Williams, M. C., Wikström, C. and Virding, S. R., *Concurrent Programming in* ERLANG*, 2:nd ed.* Prentice Hall (1996)

[2] Hausman, B. *Turbo* ERLANG*: Approaching the Speed of C* Implementations of Logic programming Systems, ed. Tick, E. and Succi, G., Kluwer 1994.

[3] Marlow, S. and Wadler, P., *Erltc: A Type Checker for* ERLANG http://www.dcs.gla.ac.uk/~simonm/erltc_toc.html

[4] Nilsson, H., Törnquist, T. and Wikström, C. *A Distributed Real-Time Primary Memory DBMS with a Deductive Query Language* 12th International Conference on Logic Programming, 1995.

[5] Ottosson, G. *An Extension of* ERLANG *with Finite Domain Constraints* Uppsala Master's Thesis in Computing Science, Uppsala University, 1995

[6] Wikström, C. *Distributed programming in* ERLANG First International Symposium on Parallel Symbolic Computation, Linz Austria, 1994.

[7] Wikström, C. *Implementing Distributed Real-Time Control Systems in a Functional Language* IEEE Workshop on Parallel and Distributed Real-Time Systems, April 15-16, 1996, Honolulu, Hawaii.