Erlang and its applications

Joe Armstrong and Thomas Arts
Computer Science Laboratory
Ericsson Telecom AB
Box 1505, S - 125 25 Älvsjö
Sweden
{joe,thomas}@cslab.ericsson.se

Erlang is a functional programming language for programming concurrent and distributed systems initially developed by Joe Armstrong, Robert Virding, Claes Wikström and Mike Williams [2]. The development of Erlang started as an investigation into whether modern declarative programming paradigms could be used for programming large industrial telecommunication switching systems. The resulting language, Erlang, turned out to be suitable as well for programming telecommunication systems as for a wide range of industrial embedded real-time control problems.

Technical requirements

The technical problems that had to be focussed when developing the language originated from the typical industrial requirements (cf. [3]):

- **Real-time** - The typical applications require real-time response times in the order of milliseconds. Erlang is designed for programming "soft" real-time systems where not all timing deadlines have to be met. The language also provides mechanisms for allowing processes to timeout while waiting for events and to read a real-time clock.
- **Very large programs** - Control systems can have millions of lines of code, and are programmed by large teams of programmers.
- **Non-stop systems** - Control systems cannot be stopped for software maintenance. The Erlang abstract machine allows program code to be changed in a running system. Old code can be phased out and replaced by new code. During the transition, both old code and new code can be run at the same time. This enables faults to be corrected and software to be upgraded in systems without disturbing their operation.
- **Portability** - Erlang compiles to abstract machine code which can be run on any of a large number of different operating systems. This approach makes the system source and object code compatible.

- **Concurrency** - Our applications are best modeled by a very large number of concurrent processes. At any instant in time most of these processes are idle. The number of processes and their memory requirements vary with time and are extremely difficult to predict in advance. Erlang has lightweight processes whose memory requirements vary dynamically. No requirements for concurrency are placed on the host operating system.

- **Distribution** - Erlang is designed to be run in a distributed multi-node environment. Every computation in Erlang is performed within a process. Processes have no shared memory and communicate by asynchronous message passing. An Erlang system running on one processor can create a parallel process running on another system (which need not even be the same processor or operating system) and thereafter communicate with this process.

- **Garbage collection** - Erlang is used to program real-time systems. Long garbage collection delays in such systems are unacceptable. Erlang implementations are written using bounded-time garbage techniques, some of these techniques are described in [1, 7].

- **Incremental code loading** - Users can control in detail how code is loaded. In embedded systems, all code is usually loaded at boot time. In development systems, code is loaded when it is needed. If testing uncovers bugs, only the faulty code need be replaced.

- **Robustness** - The Erlang abstract machine has three independent error detection primitives which can be used to structure fault-tolerant systems. One of these mechanisms allows processes to monitor the activities of other processes, even if these processes are executing on other processors. We can group processes together in distributed systems and use these as building blocks in designing distributed transaction oriented systems.

- **External Interfaces** - Erlang has a "port" mechanism which allows processes to communicate with the outside world in a manner which is semantically equivalent to internal message passing between Erlang processes. This mechanism is used for communication with the host operating system and for interaction with other processes (perhaps written in other languages), which run on the host operating system. If required for reasons of efficiency, a special version of the "port" concept allows other software to be directly linked into the abstract machine. Examples of the use of the port mechanism are interacting with the host file system, interfacing to a graphical interface and a low level socket interface.

Organizational requirements

Except for the technical problems that have to be addressed, the success of Erlang also depends on overcoming organizational problems. One can

only speculate about the reasons why Erlang spread so successfully from the computer science laboratory to a number of commercial products.

- **Real problems** - We work on real problems. We tend to make progress when we cannot solve a particular problem with the existing technology. Progress has often come when a user came with a problem which could not be solved in Erlang.
- **Working within the organization** - We work *within* the Ericsson organization. It is far easier to "sell" an idea internally than to come to the organization from outside.
- **Organizational support** - There is a gap between the best that a laboratory with limited resources can produce and what is minimally acceptable for a commercial product. Ericsson has provided financial support and created new jobs as necessary to help fill this gap.
- **Providing good support** - Good documentation, courses, e-mail, hot-line telephone support etc. are essential in passing from the "enthusiast" to the "main-stream" phase of development.
- **Foreign language interfaces** - Typical systems are written in several different languages. Erlang is not good at everything. Large parts of a system might use purchased software packages written in C. Efficient integration with C is essential.
- **Lots of tools** - Project managers are not interested in programming languages. They are not interested in formal anything.

  They are however, interested in short "time to market" and in bug-free written software. The provision of large numbers of software tools can greatly reduce software development times and improve the quality of the software.

  These tools are specific to our problem domain. Thus we have tools for making SNMP MIBs, for manipulating ASN.1 data types, for building fault-tolerant duplicated data-bases with hot-standby [6] etc.

Erlang Applications

Erlang has started to be used by other people than the developers in 1987. The first projects, which were within Ericsson, were simple prototypes written using a very slow version of Erlang which was implemented in Prolog [8].

Work with the interpretor led to the development of a much faster Erlang machine [4] which was loosely based on the WAM with extensions to handle process creation and message passing.

The availability of a faster Erlang implementation encouraged the spread of the language within Ericsson and a number of experimental projects were started. At the time of writing some of these have developed into full-scale Ericsson products.

In 1994 the first International Erlang Conference was held in Stockholm. This conference, which publishes no proceedings has been held every year since 1994. The 1995 conference attracted 160 delegates from 10 different countries. The Erlang conference is the principle forum within Ericsson for reporting work done with Erlang.

By 1995 three projects had matured into stable commercial products, namely:

- **NETSim** - Network Element Test Simulator.
- **Mobility Server** - The Mobility Server is a fully featured PBX.
- **Distributed Resource Controller** - the distributed resource controller (DRC) is a scalable, robust resource controller written in distributed Erlang and running on Windows-NT.

Conclusion

After Erlang had shown to be useful in several projects, on January 1st, 1996 a new Ericsson division was created to support applications written in Erlang (the Open Telecom Platform (OTP)). The OTP division can provide Ericsson users with anything from a simple Erlang system which runs on a PC to an embedded system complete with hardware. The division is now taking care of the maintenance and support of the Erlang system.

A free version of Erlang, with extensive libraries that solve common application problems, is available on the web `http://www.erlang.se`.

References

[1] J. L. Armstrong, S. R. Virding. One pass real-time generational mark-sweep garbage collection. *International Workshop on Memory Management (IWMM'95)*, September 27-29, 1995, Kinross, Scotland

[2] J. L. Armstrong, M. C. Williams, C. Wikström and S. R. Virding. *Concurrent Programming in Erlang*, 2nd ed. Prentice Hall (1995)

[3] J. L. Armstrong. Erlang - A survey of the language and its industrial applications. In *Proceedings of the symposium on industrial applications of Prolog (INAP96)*. 16 – 18 October 1996. Hino, Tokyo Japan.

[4] J. L. Armstrong, B. Däcker, S. R. Virding, and M. C. Williams, *Implementing a functional language for highly parallel real time applications*. 8th Int Conf. on Software Engineering for Telecommunication Switching Systems, Florence 30 March – 1 April 1992.

[5] B. Hausman. Turbo Erlang: Approaching the speed of C. In *Implementations of Logic Programming Systems*, pp. 119-135, ed. Evan Tick and Giancarlo Succi, Kluwer Academic Publishers (1994).

[6] C. Wikström and H. Nilsson. Mnesia - An Industrial DBMS with Transactions, Distribution and a Logical Query Language. *International Symposium on Cooperative Database Systems for Advanced Applications*. Kyoto Japan 1996

[7] S. R.Virding. Collector for the concurrent real-time language Erlang. *International Workshop on Memory Management (IWMM'95)*, September 27-29, 1995, Kinross, Scotland

[8] K. Ödling. New technology for prototyping new services. In *Ericsson Review* No. 2 1993.