Abstract

This thesis deals with two important topics:

- The use of the best possible technology in the development of telecommunications systems in a world of rapid change and increasing competition. The word "best" implies a technology that satisfies the requirements of the application and a technology that enables high design efficiency in bringing projects to rapid and successful results.
- The introduction and exploitation of functional programming in industry. Functional programming is a long established discipline within academia where its advantages are well known.

The thesis is built around the actual case of the concurrent functional programming language Erlang which was developed at the Computer Science Laboratory at Ericsson and which is now available as open source. Erlang is now used in about 20 systems, notably the ATM switching system AXD 301 where Ericsson in two years moved from having no system to having a veritable flagship.

Starting as a purely technical project the progress of Erlang came to touch upon many other topics such as *Management of Technology* and *Open Source*.

Acknowledgements

Acknowledgements are due above all to the Erlang Design Team (*Joe Armstrong, Claes Wikström, Mike Williams,* and *Robert Virding*), members of Erlang Systems, the OTP Product Unit, the Erlang application projects and all the enthusiastic Erlang users around the world. I also thank professor Gerald Q. Maguire Jr. who got me started writing and supported me all through the process.

The following people have read and commented on the thesis: Joe Armstrong, Roy Bengtsson, Catrin Granbom, Torbjörn Keisu, Kenneth Lundin, Richard O'Keefe, Phil Wadler, Ulf Wiger, and Mike Williams.

Acknowledgements are not least due to Ericsson for supporting the Erlang development consistently for many years and enabling Erlang to be used for very large and important projects.

Bjarne Däcker, Stockholm, October 2000

Contents

1 Introduction

1.1 Overview

This thesis traces the progress of Erlang from applied research in an industrial research laboratory through productification, technology transfer to a product unit and further dissemination through open source in parallel with widening use in application systems for the market. My prime task in this process has been as manager of the *Computer Science Laboratory* (CSLab) encouraging the work and helping to establish contacts with external research and with prospective users.

1.2 Background

"Lisp" is an abbreviation of "list programming", but often "symbolic programming" is a more accurate term. The fact is that many programming problems in industry are symbolic in nature and thus lend themselves naturally to programming languages like Lisp [Wi81].

I learnt Lisp at a Summer school at Uppsala in 1974 and immediately saw its possibilities. I was then working with hardware *Computer Aided Design* (CAD) systems and had among other things spent one year once in designing a database in Fortran to represent relay set circuit diagrams.

Soon a natural application presented itself. Ericsson then used two software tools to aid in the design of circuit boards. One was a system for digital simulation and test data generation. The other was a program to create the circuit board layout. Thus one program dealt with logic entities like gates and flip-flops and the other dealt with mechanical entities like discrete components and integrated circuits. The programs had very different looking inputs and it was very difficult to ascertain that those were, indeed, equivalent. Given the usual technology at the time, Fortran [Ek79] and PL/I [Hu87], to create a program to convert between the two programs was almost impossible or would have led to a very large and complex project.

Lisp instead made it quite feasible to design a database that could represent both the electronic and the mechanical aspects of a circuit board. The next step was to create programs that could generate the database given the input to either of the CAD programs. The step after that was to write programs that instead could generate their respective inputs from the database.

The program needed some extra input when, for example, gates were designed using discrete components, but it solved a tricky problem for the designers.

CAD represents a typical area for symbolic computation and when numerical computations came in, it turned out that Lisp could handle them just as efficiently.

In 1982 another possible application presented itself. Ericsson's AXE 10 [He76] system was being sold around the world and increasing demands were made for higher capacity and throughput. This led to the design of the APZ 212 [Hj90] processor. Thus a compiler for the AXE programming language Plex [He76] had to be developed at short notice. Even more important was an interactive interpreter since many programs had to be developed and tested before the APZ 212 hardware would be available.

My suggestion was to use Lisp and I got the go-ahead. The first step was to write a parser to create an internal representation of a Plex program for the interpreter to work with. This could be stored using `(write ...)` and retrieved using `(read ...)` and since it was only a large S-expression, it could easily be inspected for mistakes.

The next step was to develop a full screen interactive interpreter using VT100 terminals attached to a VAX running UNIX [At84]. It turned out that the designers liked the interpreter very much and also quickly got used to seeing their programs pretty printed on the screen. The interpreter also contained an editor using the parser one statement at a time.

There had been some concern that Lisp would lead to very long execution times. However, nothing of this sort was noticed. Most execution times were dominated by the time required to print output to the screen.

This interpreter was a prototype that handled only a central subset of Plex. However, it was taken over by a programming team that completed and productified it and ported it to the IBM 370 computers which were used for program development for AXE 10.

By chance a much larger project had been running to develop a Plex interpreter using an IBM version of Pascal. For some time the two projects ran in parallel. The good thing was that this provided large volumes of data for comparison. It turned out that the interpreter in Lisp was smaller by a factor ranging from 7 to 10 for different parts of the system [Dä83, Al84].

For an industry concerned with the software "crisis" [Gi94] this should have been very interesting news. The experiences from this application were high design efficiency and adequate execution performance. Our hopes was that this would have opened the way for wide use of Lisp in the development of software development environments, but nothing happened. Everybody seemed to agree that using Lisp for this type of application is more efficient than using languages like C or Pascal, yet it did not catch on.

One of the key activities when starting a design project is the selection of the appropriate technologies to be used. This means that there ought to be a list of candidate technologies with descriptions of their respective properties. Then the characteristics of the application have to be defined and the selection process becomes a matter of matching. The characteristics must take into account many aspects such as cost, capacity, performance, documentation, standards compliance, design efficiency, support, training, etc.

The Lisp experience described above indicated that there should be other interesting technologies available for industrial exploitation. At that time *Artificial Intelligence* (AI) was a very hot topic and CSLab (see below) also built an operator support system [Sk86] exploiting some AI technologies.

1.3 Summary of Papers

The key paper is [Dä91] which presents the ideas of dealing with software technology according to the same principles as older established engineering sciences, especially with regard to *applied research* in the laboratory environment. The Erlang history is one case demonstrating that those principles also work in practice.

The papers [Dä83, Al84] describe the work of building the Plex interpreter using Lisp where I both took the initiative and built the first usable system. The paper [Dä86] describes experiments with building prototype telecom systems using different programming technologies where my contribution was the rule-based system.

The paper [Dä93a] summarises Erlang design work carried out at CSLab and the papers [Dä93b, Dä94a, Dä94b, Dä95] describe early experiences from the use of Erlang.

1.4 Layout of this Thesis

Chapter 2 provides a context for this thesis by describing the general principles behind *applied research* in the industrial environment. Chapter 3 defines the requirements put on a software technology for telecommunications applications and also early experiments at CSLab with different available programming technologies.

Chapter 4 introduces *Erlang* and its development and matches it against the above requirements. Chapter 5 describes the development of the *Open Telecom Platform* using Erlang and extending its usability and chapter 6 provides an overview of some important Erlang applications which gives more detailed material to validate Erlang against the telecommunications requirements.

Those chapters also describe the spread of Erlang within Ericsson and outside Ericsson primarily to academia and also attempts at marketing Erlang. Chapter 7 describes an internal set-back and chapter 8 the work of spreading Erlang through the new medium of *open source.*

Chapter 9 discusses the spread and use of functional programming and, finally, chapter 10 presents my conclusions on this matter based on the Erlang experience.

2 Management of Technology

Management of Technology [Gr82] is an area of study within Industrial Management and Economy that focuses on the interaction between technical and economic changes and on how this interaction can be influenced in desirable directions. Management of Technology in large divisionalized companies like ABB and Ericsson involves many complex aspects:

- The term *Research and Development* (R&D) is often very loosely used. In industry development is by far the largest component and most research is *applied* in the form of trying out new techniques on old (or emerging) applications. The term "research" is also often used to describe experiments prior to product development.
- The purpose of *basic research* is to find new knowledge for mankind whereas the purpose of *applied research* is to apply new knowledge to reality. Applied research can be *technology driven* or *market driven*. In the first case some new technology has been created, then work is done to find uses for it. In the latter case some need or opportunity has arisen which cannot be adequately handled with current technologies, see Figure 2.1.
- To qualify as *applied research* as different from regular product development, the work needs to be of scientific quality, for example by being presented at significant conferences or journals in the respective field.
- The organisation of research has to be a compromise between centralization and decentralization since there has to be close contacts both with

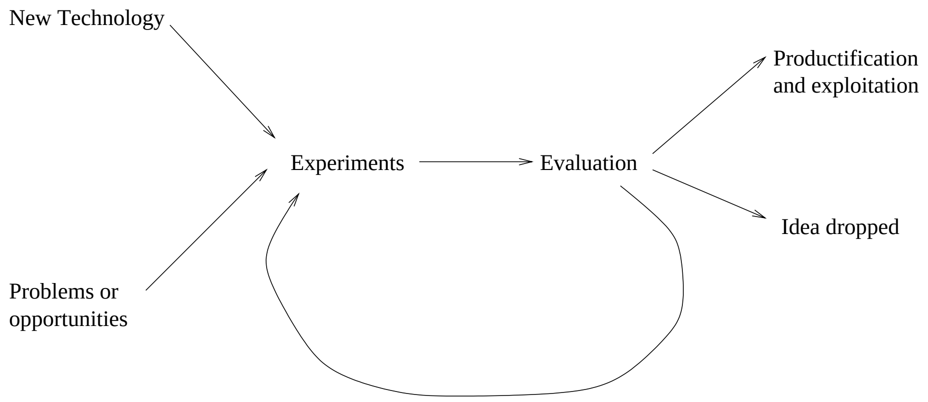

Figure 2.1: The process of Applied Research.

Knowledge standing	Patents Papers at international conferences References in internationally recognized journals Publications in internationally recognized journals
Competitor standing	Comparison with existing and potential competitors to identify strong/weak points
Knowledge transfer	Visits by customers Internal presentations/demonstrations Facts, based on prototyping etc., delivered Ideas accepted by System & Technology development process
Interworking	Guest researchers Lectures given at universities Working with recognized Centres of Excellence Working with potential producers/users of the result Participation in national/international research projects Participation in national/international committees for research funding

Table 2.1: Evaluation criteria for Applied Research in Ericsson [Er95].

strategic planning and with product development in the product divisions. If not handled well this can give rise to conflicts and misunderstandings.

- Research can be placed in some central unit (like Bell Laboratories) or be spread out as decentralized laboratories organised within the various divisions (or subsidiaries) through some matrix organisation. The present situation within Ericsson is a combination of both. Further reasons for locating research laboratories within subsidiaries in different countries are to facilitate interaction with university research in the different countries and access to qualified personnel. (The term *laboratory* is henceforth used to denote an organisational unit working with research.)

- Financing of research is often a combination of central financing with project assignments from design departments in some suitable ratio 50/50 (ABB Corporate Research) or 70/30 (Ericsson Research). That the design departments are interested in having researchers participate in projects or are placing projects in the laboratories is one test of the competence and co-operative spirit of the laboratories.

- Evaluation of the performance of industrial research is a difficult matter. University research is evaluated based on its publication record and product development on the profitability of its products. Table 2.1 lists the evaluation criteria that were defined in 1995 for applied research laboratories in Ericsson [Er95].

- For companies developing systems it is important to note that every system is dependent on many different technologies. This means that different laboratories typically are either focused on some technology (like fibre optics) used in many different systems or are focused on new applications where they start from many different technologies (sometimes originating in other laboratories).

- A systems company might both use its own technology in its systems and sell them as components, for example ABB both uses its own relays and sells them and Ericsson's power division sells both inside and outside the company. This implies an internal technology market. (In January 2000 Ericsson divested its energy business [Pr00a].)
- A laboratory can never take responsibility for a product either delivered to customers as an application or delivered to design departments as a technology. This requires quite a different organisation and a different mode of work, handling new releases, handling error reports and user complaints, production of professional quality documentation, organisation of courses, training, consulting, etc.
- This in turn requires transfer of the results to another type of unit or perhaps transforming the laboratory into a product design unit (department). In any case the researchers have to take part in this and perhaps transfer out of the laboratory.
- There are different ways of manning a laboratory. One case could be to recruit senior designers interested in research, another to take in young people from a university who can learn the latest technology in the laboratory and then move with their system out into the design organisation.
- Technology research has to be seen as part of the technology provisioning process. Most technology will be obtained from suppliers. Results from the laboratories are the exception and there are primarily two reasons for using them:
  - There are requirements which cannot be fulfilled by external technology.
  - There are technologies which can provide competitive advantage. (In this case patents are of the utmost importance.)
- Compliance with internationally accepted standards is essential to achieve interoperability and to enable systems to be upgraded by renewing just parts of them.
- Internally developed technology that gives advantages in the near term can cause problems in the long term – especially if it is overtaken by external developments.

The primary difference between a research project and a regular development project is that research has a greater uncertainty, but also offers hopes of greater gains if it is successful. This means that research projects have to be carried out in moderately long steps with careful monitoring of progress.

The important realization is that all the principles of *Management of Technology* that are the results of many years of working with materials engineering, chemical engineering, etc. actually can apply to software as well. This should give a new and perhaps sounder interpretation of the term “software engineering” [Dä91].

3 The Problem

3.1 Telecommunications Programming and Chill

Applied research needs to start by defining "the problem". Table 3.1 summarizes the requirements on a programming technology for telecommunication switching systems. These lead to both large systems and large projects. Thus a key question is that of design productivity, i.e. how to be able to design such systems with smaller design teams in a shorter time.

In the 1970's there was much talk of the "software crisis" [Gi94] as a term to describe the problems of programming projects running late, using too much manpower, and resulting in systems of unacceptably low quality. Most attempts at dealing with this have been to introduce more formalized work methods. This is often termed "software engineering" and the avowed aim has been that software should be produced in an engineering fashion.

The telecommunications industry was early to use computers in their systems and Ericsson installed its first *stored program controlled* (SPC) system AKE 12 [Ka68] in Tumba in the Autumn of 1966. Since telecommunications lead to very large complex systems and projects a special conference series *Software Engineering for Telecommunications Switching Systems* (SETSS) was set up by the British *Institution of Electrical Engineers* (IEE).

The strange word is "crisis". Projects in other engineering fields also experience problems. A case in point is the tunnel building project through Hallandsåsen, Sweden [Da96]. Such occurrences, however, are not classified as crises, only as faulty professionalism. The fact is that many programming projects run well even without the formal "methods". The keys, as in other engi-

1	*Handling of a very large number of concurrent activities*
2	*Actions to be performed at a certain point in time or within a certain time*
3	*Systems distributed over several computers*
4	*Interaction with hardware*
5	*Very large software systems*
6	*Complex functionality such as feature interaction*
7	*Continuous operation for many years*
8	*Software maintenance (reconfiguration, etc.) without stopping the system*
9	*Stringent quality and reliability requirements*
10	*Fault tolerance both to hardware failures and software errors*

Table 3.1: Requirements on a programming technology for telecommunication switching systems [Dä93a].

neering areas, are competent people, appropriate architecture, good technology, and clear goals.

Problems are one thing, “crises” another. The word implies a feeling of lack of control, i.e., that software is inherently something strange and unpredictable. Software is not strange to people working in the field, but probably was to an industry getting increasingly dependent upon it a couple of decades ago.

There are different ways of improving project performances. Methods represent the organisational approach, i.e., that the work is reasonably well structured and properly carried out. Another is qualified (and motivated) personnel. Another often overlooked factor is technology. This thesis examines the introduction of new software technology.

Software development is a matter of programming. Some systems aim to accept specifications and then generate code. Such systems approach the problem top down. A complementary approach is to raise the level of programming by using a language technology at a high abstraction level which provides built-in support required by the application domain. This thesis deals with the latter approach. (Edward Yourdon places “better programming languages” at the top of his personal list of “silver bullets” [Yo92].)

Fortran [Ek79] was developed around 1957, Pascal [Je75] around 1970 and C [Ke78] in the middle 1970’s. It is remarkable that today most programs are still developed in languages of these types while computer hardware has gone from discrete components to VLSI.

Pascal was originally developed as a language to teach students programming. Significant additions in Modula [Wi76] were *modules* and *processes*. Modules are necessary for building large systems and processes are necessary to describe concurrency. Each process instance functions like a sequential program on its own computer through a timesharing scheme [Bu90]. Other programming languages of this class are Chill [CC84b] and Ada [Ad83].

The C.C.I.T.T. standardised programming language Chill provided three mechanisms for communication and synchronization between processes:

- **Regions**,
- **Buffers** (message mailboxes),
- **Signals** (messages).

Quote “there are several reasons why Chill provides three different mechanisms for process communication:

- The ideas about what is the best method of communication between processes have not yet been stabilised in the world of programming language design. It would be too early to supply only one method of communication.
- Experience with communication between processes in a distributed system (without common memory between processors) is very limited. One communication mechanism may not be able to function optimally in both distributed and common memory architectures.” [Sm83]

The effect of this was that each company that used it chose some suitable subset.

In late 1999 C.C.I.T.T. stopped maintaining the Chill standard.

In 1979 I was responsible for the creation of the in-house programming language EriPascal [Dä79, Bå84] for a new processor APN 167 [Ma86]. EriPascal was similar to a subset of Chill but with a Pascal like syntax. Like Modula it contained modules and processes. In EriPascal only messages were used. There were also plans to create an EriChill by using a different compiler front-end which accepted a Chill like syntax, but there were no user requests for it.

Still the basic sequential programming paradigm remained at the level of Fortran and Pascal even though the whole idea of functional and logic programming had been around almost as long as Fortran. For example, Lisp [Mc65] was created already in 1958 [We81].

3.2 Programming Language Experiments

In 1984 the *Computer Science Laboratory* (CSLab) [Csw] at Ericsson was formally established by myself and three colleagues, Göran Båge, Seved Torstendahl, and Mike Williams. Further colleagues joined. CSLab has had a low personnel turnover and has averaged between 12 and 15 people.

The laboratory soon was equipped with a VAX 750 running one of the first UNIX systems at Ericsson and an MD 110 private exchange. Per Hedeland modified the MD 110 so that it could be controlled from the VAX. At that time telephony (or any real time) programming involved two computers, a *host computer* where editing, compilation and linking took place and a *target computer* embedded in the actual system. The VAX combined the functions of both which reduced the turn around time considerably and created an excellent environment for experimentation.

Based on this environment a series of experiments were carried out with different programming languages and paradigms which involved at least five persons in the laboratory. The following techniques were tried [Dä84a, Dä86]:

- **Imperative programming languages:** Concurrent Euclid [Ho83b] and Ada [Ad83].
- **Declarative programming languages:** PFL [Ho83a] and Prolog [Cl81].
- **Rule based programming:** OPS4 [Fo79].
- **Object oriented languages:** Frames [We83] and CLU [Li79].

My contribution was the experiment with rule based programming using OPS4. This could express many requirements very neatly. One problem in conventional telecommunications programming is that a subscriber may hang up at any time during a call. This means that all equipment involved has to be reset. In relay systems it sufficed to release the holding wire. With OPS4 this became equally simple. One resetting rule for each type of equipment was all that needed to be defined. This rule triggered whenever it applied, even in different states of a call.

Rules will trigger whenever they apply regardless of their order in the source code so that when sequencing is required this has to be specified using some state variable. One problem, of course, was how to handle really large systems of many rules. The good feature of rule based programming is that it encourages a very declarative style of programming. It also leads to highly parallel designs as only when rules are sequenced do they have to be applied in order. Thoughtful use of guards and pattern matching can achieve a programming style similar to rule based programming.

One conclusion was that telephony systems obviously could be programmed in any language and the different implementations displayed rather similar structure. However, certain desirable properties of a programming technology for telephony became apparent:

- **Massive fine grained concurrency:** Complex real time systems usually have to handle several concurrent activities which, in turn, normally are handled through some *process* (or thread) concept. However, typical for telecommunications are the large amount of equipment and the great number of simultaneous calls which means that the processes have to be very *light weight*.

- **Asynchronous message passing:** This seemed to be a very typical and normal requirement. Both Plex [He76] from Ericsson and SDL [CC84a] from C.C.I.T.T. are based on message passing. Also message passing naturally extends to distributed systems.

Unfortunately there was no Chill implementation available for the UNIX VAX. However, the system programmed in Concurrent Euclid [Ho83b] was structured along the same lines as would be a system written in Chill.

The report of these programming language experiments ended with the following conclusion:

- "It is becoming obvious that future telecommunication systems cannot be programmed with one language using one methodology. Future systems will probably be built using many of the techniques used in these experiments. For example expert system technology might be used for the maintenance functions and the man machine interface, logic programming might be suitable for programming the signal system interfaces and parts of traffic handling and the underlying operating system might be programmed in an advanced imperative language." [Dä86]

4 Development of Erlang

The primary aim of this exercise, however, was not merely to see if it was possible to program telephony systems in a variety of ways, but also to find which style of programming lead to the shortest and most *beautiful* programs closest to the level of formal specifications. These features have the greatest impact on quality and programmer efficiency since it has been shown that programmer productivity in number of error free lines of code per day is largely independent of the programming language

The language experiments had given many new insights [Dä86], but had not reached a conclusion that one specific language was "the best". Of all the languages tried only Prolog could handle updating of software in running systems. The Ericsson AXE 10 [He76] can handle update of running code through the use of special hardware and switch over between the duplicated processors such that one processor carries on the traffic processing while the other loads the new version of the code.

4.1 Prototype Developments

Joe Armstrong led the next round of experiments and started with Prolog, because of its terse and clear style, and added concurrency. However, the language began to change in the direction of a functional style [Ar92b]. For example, one of the key characteristics of Prolog is backtracking and this could not be used because it is not possible to backtrack over hardware. (A tone signal once sent out cannot be taken back [Ar86].)

The name given to this experimental language was **Erlang** after the Danish mathematician Agner Krarup Erlang, creator of the *Erlang loss formula* [Erl]. This followed the tradition of naming programming languages after dead mathematicians, other examples are Pascal, Euclid, and Occam [Jo88]. Erlang is aptly described as a *concurrent functional* programming language combining two main traditions, Figure 4.1:

- **Concurrent programming languages:** Modula, Chill, Ada, etc. from which Erlang inherits modules, processes, and process communication.
- **Functional and logic programming languages:** Haskell [Haw], ML [Wi87, Mlw], Miranda [Th95], Lisp [Wi81], etc. from which Erlang inherits atoms, lists, guards, pattern matching, *catch* and *throw*, etc.

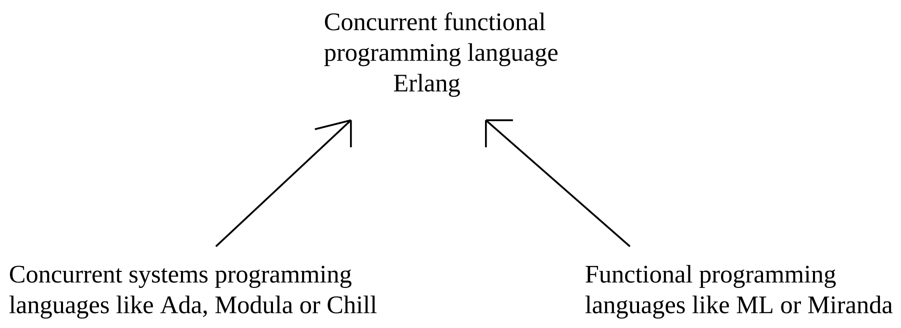
Concurrent functional programming language Erlang
Concurrent systems programming languages like Ada, Modula or Chill
Functional programming languages like ML or Miranda

Figure 4.1: The ancestry of Erlang.

Significant design decisions behind the development of Erlang:

- It should be built on a virtual machine which handles concurrency, memory management, etc., thus making the language independent of the operating system and ensuring program portability.
- It should be a *symbolic* language with garbage collection, dynamic typing, data types like atoms, lists, and tuples.
- It should support *tail recursion* [Ok90] so that all loops, even infinite like in drivers, can be handled by recursion.
- It should support *asynchronous message passing* and a selective message *receive* statement.
- It should enable default handling of errors, for example through *trap exits*. This also enables an aggressive style of programming.

Erlang is a simple language to learn in that it contains very few concepts. Pattern matching enables a very declarative (and self documenting) style of programming.

Programming languages form an intermediate level between the computer hardware and the application. It is necessary that the language provides powerful concepts that can facilitate the application; yet it is equally necessary that those concepts can be efficiently implemented. The first large Pascal program developed was the Pascal compiler. This enabled Niklaus Wirth to find the right compromise between concepts for a complex application and their implementability.

A significant occurrence at the end of 1987 was that CSLab came to cooperate with a telecommunications prototyping team lead by Kerstin Ödling at *Ericsson Business Communications AB* (EBC). Team members were Håkan Karlsson, Håkan Larsson and Åke Rosberg. Based on a study of several different existing *Private Branch Exchange* (PABX) systems they had defined an improved PABX architecture, *Audial Communication System* (ACS), and the next natural step was to build a prototype. For this they needed a suitable programming technology and for this they chose Erlang which itself was still a prototype.

This lead to a highly constructive collaboration during 1988 and 1989. The project, named ACS/Dunder, was reported in December 1989 [Pe89, Dä89, Öd93]. By this time a prototype system with a functionality corresponding to about 1/10 of the complete MD 110 [Mö82, Mdw] had been designed and verified. Based on the ACS architecture and the Erlang programming language the prototype showed an improvement in design efficiency by a factor of 20 over current technology.

The ACS architecture structured the system into a *Basic Operating System* (BOS), an *Applications Operating System* (AOS), and the applications. BOS had some very sophisticated structures for supervising the system, which were later used for the *Open Telecom Platform* (see below).

Erlang itself underwent many changes during this work – which the users patiently endured. Most notable was a major syntax revision from a Prolog style to a functional style.

Declarative programming and Erlang were mentioned in a presentation by Lars Ramqvist on Strategies and Technologies for the 1990's [Ra88].

In May-June 1990, the *XIII International Switching Symposium* (ISS'90) took place at Älvsjö Exhibition Centre in Stockholm with about 2000 participants. ISS is the major event in telecommunications and this is where Joe Armstrong and Robert Virding first officially presented Erlang to the world [Ar90]. Erlang was also demonstrated during a technical visit which was shown to eight different groups during a hectic day. I had written the demonstration telephony application that was used.

The original purpose of this demonstrator was to experiment with ways to structure a telephony system with features such as call back, short number, conference call, etc. Ideally each feature should be represented as a separate program module. However, since every feature comes in at many different points in the system a system usually becomes a complex weave when all of them are combined. In the demonstrator the features were structured as increments that were invoked from the outgoing call and incoming call state machines.

The presentation at ISS was noted by the *Royal Swedish Academy of the Engineering Sciences* (IVA) in their yearly summary of technical progress in Sweden [Iv90].

4.2 Steps towards a Product

ACS/Dunder was based on an Erlang interpreter written in Prolog. This was acceptable for a prototype, but not for a real product. (For one thing a real time system cannot be allowed to stop for a couple of seconds now and again for garbage collection. A proper implementation would have to include an incremental real time garbage collector.) The users at EBC also required a speed-up by at least a factor 40.

The Erlang design team consisting of Joe Armstrong, Mike Williams and Robert Virding started experimenting. One attempt was a cross compiler to Strand [Fo89]. Finally an abstract machine, *Joe's Abstract Machine* (JAM), was invented inspired by the Warren Abstract Machine [Ma88] with added primitives for concurrency and exception handling [Ar92a]. Joe wrote the compiler, Mike the emulator and Robert the support libraries. This turned out to be 70 times faster than the original Prolog interpreter. Although the EBC group by now

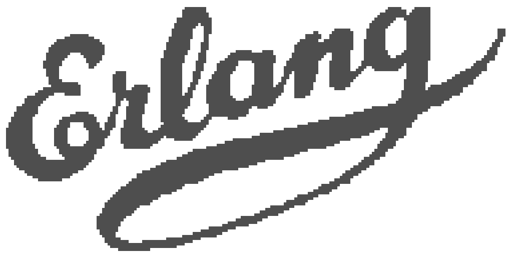

Figure 4.2: My original Erlang logotype.

had escalated their demands the JAM implementation proved that concurrent functional programming could be used for “soft real time” system products, i.e. response times measured in milliseconds.

In telecommunication systems “hard real time” is usually handled by dedicated (device or regional) processors close to the hardware, and thus feature response times of microseconds.

In 1990, Erlang was recommended for prototyping purposes within Ericsson and several different system were designed for very different purposes. A group at ERA (*Ericsson Radio AB*) built a demo system [Ah92] to control a cordless exchange. Göran Båge at CSLab built a control system for a photonic switch which was shown at Telecom’91 in Geneva. Sebastian Strollo (also at CSLab) built a demo system [Bu92] which was displayed at the *European Conference on Optic Communication* (ECOC) in Paris in 1992. Both these systems were set up for permanent display at the switching laboratories in Ericsson. Claes Wikström and Sebastian Strollo also built a demo system for TELI of their planned Ermes pan-European paging system [Erm]. (TELI was a subsidiary of Televerket later Telia, the Swedish national telecom operator.)

Erlang was used in *Research in Advanced Communication for Europe* (RACE) projects such as Biped [Mo93], but an interesting observation was that it was only used in hardware oriented projects. In such projects it was necessary to create a functioning prototype system to control some hardware in a short time using only a few people. The software oriented RACE projects were much more concerned with methodology than with technology.

In December 1989 the Erlang Design Team and myself were invited to Bellcore to present Erlang and to give an Erlang course. This lead to John Unger building a large subsystem of Cruiser, a system for multi-media communication, in Erlang. This was also the start of a regular course activity and the design team worked out a four day course which was given at CSLab and I found myself in the role of course organiser. In 1991 we gave three courses at *Ellemtel Utvecklings AB* (EUA), Stockholm, plus one at Ericsson in Rome and one at Ericsson in Melbourne. In 1992 there were further courses plus one at Telefónica in Madrid. The course was also held at the *Royal Institute of Technology* (KTH) in Stockholm and became part of their course on programming of parallel systems. At Uppsala a student project, Distorsion [Di91], based on a telephone exchange and Erlang, involving about 20 students, was carried out in 1991 which led to seven students doing their Master’s Theses at CSLab.

Figure 4.3: Erlang Systems logotype.

In 1992 the decision was taken to develop Erlang into a product for use in production projects. EBC also decided to build the Mobility Server based on the ACS/Dunder prototype (see above). The Erlang Design Team got permission to write a regular text book, which became *Concurrent Programming in Erlang* [Ar93] printed by Prentice Hall Ltd.

In October 1992, the *XIV International Switching Symposium* took place in Yokohama, Japan. Ericsson presented only four regular papers of which no less than three dealt with prototype systems developed using Erlang [Ah92, Bu92, Er92]. At the plenary summary at the end of the symposium the software expert, Peter Cashin from Bell Northern Research, stated that:

- "We should take our hats off to Ericsson for looking at software design in a whole new way."

Although Ericsson had been rather inconspicuous during the symposium it was the only company to be mentioned when the symposium was summed up.

4.3 Erlang Systems

In 1993 *Erlang Systems AB* was established as a subsidiary of *Ericsson Infocom AB.* The business idea was to sell Erlang as a tool in the external market and also to provide training and consulting. It turned out that the market was more difficult than expected and Erlang Systems was subsequently made into a regular department within *Ericsson Software Technology AB.* Roy Bengtsson was then appointed manager of Erlang Systems.

The immediate effect of Erlang Systems was the development of documentation and course material of professional quality. Also teaching and consulting were carried out in a way that was beyond what a research laboratory could handle. The customers were and still are primarily from various projects within Ericsson.

During 1998 and 1999 there was almost one course every week, see Figure 4.4. Most courses were given at their premises in Kista, north of Stockholm. However, often it has been more practical for the teacher to travel to the site of the application project.

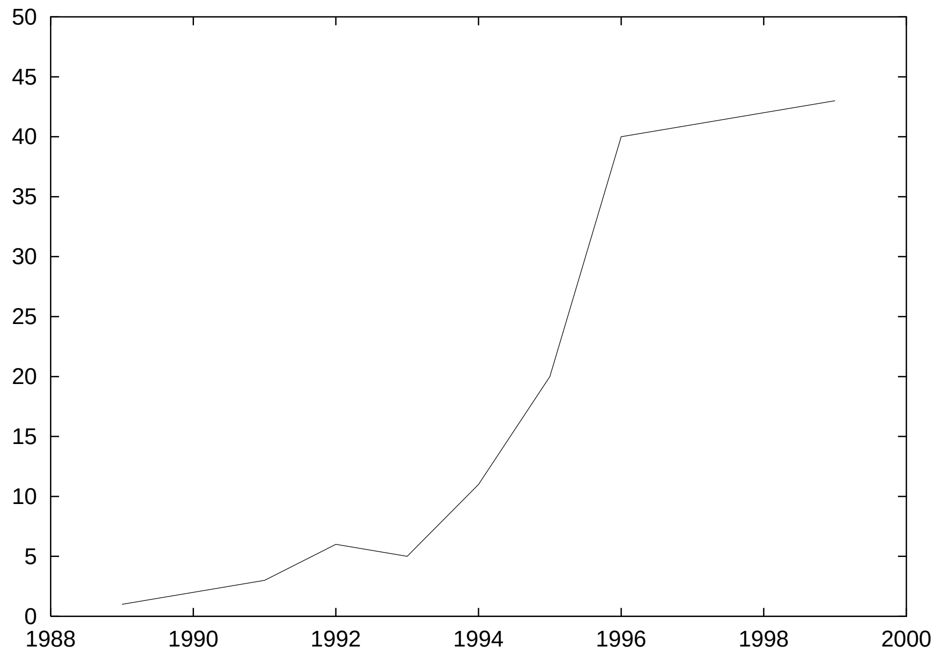

Figure 4.4: Total number of courses given per year. There are 10-12 pupils at each course.

The initial course offering consisted of the following course program:

- Basic Erlang, 4 days,
- Interoperability, 4 days,
- Tools and libraries, 4 days,
- Advanced Erlang, 4 days.

Table 10.2, page 51, details the course curriculae during 1998 and Table 10.3, page 52, the number of students and courses during 1997-1999.

In 1993 *Concurrent Programming with Erlang* [Ar93] was published. Erlang became known around the world primarily in academic circles and CSLab delivered systems around the world (by sending magnetic tapes through the regular mail).

4.4 Further Technical Developments

In 1992 Claes Wikström at CSLab developed an ASN.1 [St90] compiler [Wi92]. This was the first telecommunications oriented "tool" written in and for Erlang. Claes also joined the design team when he developed Distributed Erlang [Wi94] in 1993. Processes in different Erlang systems (termed nodes) can communicate through message passing as easily as between processes within one Erlang system. Erlang nodes can reside in different processors in a network. Claes also

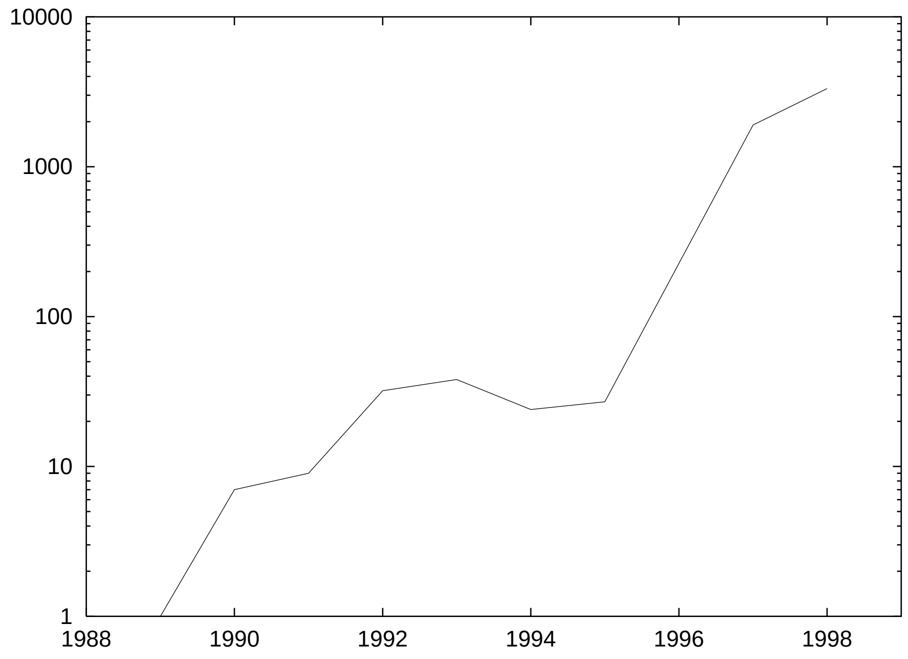

Figure 4.5: Total number of external deliveries of Erlang per year free of charge for academic or evaluation purposes prior to the *open source*. External marketing was stopped during 1995.

extended the Erlang error handling mechanisms to deal with errors in the communication or when nodes go down. This was yet a further break-through since distributed programming is notoriously very difficult. Claes became coauthor of the second edition of *Concurrent Programming with Erlang* which appeared in 1996 [Ar96a].

In 1994 Magnus Fröberg, also at CSLab, developed a translator from the *Specification and Description Language* (SDL) [CC84a, Ro85] to Erlang [Fr93]. SDL is a system description language created by C.C.I.T.T. which is widely used in telecommunications. For example, many communication protocols are described using SDL. SDL exists both in a graphical flow chart like form and in a textual form looking much like a program. It turned out that the generated Erlang code was about the same volume as the SDL textual form. Despite the interest in SDL and that many have asked about this tool there have been no users. Probably Erlang programmers do not see the need for SDL and vice versa. (A later Master's Thesis [Wi95] developing an application first using Erlang and then using SDL/SDT showed that the Erlang design took only half the time. However, the programmer had a functional programming background.)

Having Erlang as a distributed concurrent functional language enabled CSLab to attack further complex applications. In 1995 Hans Nilsson and Claes Wikström developed a distributed real time database with transactions and query processing called Amnesia [Ni96a, Ni96b]. This was later productified and renamed Mnesia [Ma99].

In October 1993 Erlang V4.1, the first commercial release, was delivered.

In 1995 Erlang Systems took over the academic distribution of Erlang. After some time this was made into a free distribution over Internet for research and education, see Figure 4.5.

Erlang Systems also issues academic licences which gives free access to all teaching material. Table 10.4, page 53, lists the academic licence holders in 1999.

On May 4-5, 1994, I organised the *First International Erlang User Conference* together with Erlang Systems. It was held at Ellemtel with about 100 participants. This has now become a yearly event (see Appendix 2 and 3, page 71 and 72). The conference in 2000 gathered 145 participants which filled the lecture hall and a waiting list was in operation from four weeks before the event.

Over the years a large number of Master's Theses (see Appendix 1, page 68) have been written on subjects related to Erlang, some of which have been quite advanced such as an implementation of Erlang based on OS threads [He98]. Another dealt with the design of an SNMP [St99a] agent [Bj95] which was later turned into a regular product as part of OTP (see below).

4.5 Early Marketing Efforts

Mike Williams, Roy Bengtsson and Per Erik Witasp, who was in charge of external marketing (but with a very limited budget), made a couple of lecture tours round Sweden and USA presenting Erlang but with little success. They had discussions with Integrated Systems, Inc. [Isw] about possible cooperation but this fell through when the key contact person left ISI.

4.6 Erlang Summary

The *open source* Erlang White Paper [Whp] provides an overview as well as 14 program examples.

4.6.1 Sequential Erlang

Erlang is a single assignment, functional programming language language with dynamic typing not unlike Scheme. Its syntax, however, is more like ML.

Erlang has data types like atoms, numbers, lists, and tuples and uses *pattern matching* to select between alternatives. The only loop construct is recursion.

An Erlang program is built up of modules and modules are separately compiled and loaded. Only explicitly exported functions can be called from another module.

4.6.2 Concurrency

Functions can be *spawned* to create a concurrent process (or thread of control). Concurrency is supported by the Erlang implementation without help from the operating system. Processes have no shared memory and communicate by sending and receiving messages asynchronously. Receiving processes select messages through pattern matching.

Processes in Erlang processes are extremely lightweight and their memory requirements can vary dynamically. Erlang implementations support applications with very large numbers of concurrent processes (typically in the region of 20,000-30,000).

Erlang supports programming "soft" real time systems, which require response times on the order of milliseconds. Long garbage collection delays in such systems are unacceptable, so Erlang is able to reclaim memory in small parts of the system every time the garbage collector is invoked.

4.6.3 Distribution

Erlang permits transparent distribution. An Erlang program running on a computer is termed an Erlang node. A distributed Erlang system consists of several Erlang nodes spread over many computers (perhaps running different operating systems) connected over a network.

One Erlang node can create concurrent processes running on other nodes and Erlang processes in different nodes can communicate through message passing in the same way as processes within one node with automatic marshalling.

4.6.4 Robustness

An Erlang process can crash (because of type error, division by zero etc.) but this will only bring down that process not the entire system (or node). Erlang processes, however, can monitor each other so that an error can be received as an error message. This enables the design of robust systems where supervisor processes can take action, reclaim resources, log errors, restart a transaction etc.

These mechanisms extend also over different nodes in a distributed system. For example, a distributed system can be configured to fail-over to other nodes in case of failures and automatically migrate back to recovered nodes.

This enables the design of soft-fail systems where in a telecommunications system an error in one call may bring down that call while the rest of the system is not affected.

4.6.5 Software Upgrading in Running Systems

Erlang allows program code to be changed in a running system ("hot" code loading). When a new version of a module is loaded, newly spawned processes will run the new version while on-going processes continue and finish undisturbed. It is thus possible to install bug fixes and upgrades in a running system without disturbing its (currently running) operation.

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, even when the system is running. If testing uncovers bugs, only the buggy code need be replaced.

4.6.6 External Interfaces

Erlang processes communicate with the outside world using the same message passing mechanism as is used between Erlang processes. This mechanism is

		Processor		
		SPARC	*Pentium*	*PowerPC*
Operating System	*Sun Solaris 2*	X	-	-
	Sun Solaris x86	-	X	-
	VxWorks	X	-	X
	Windows NT 4.0	-	X	-
	Windows 95	-	X	-
	Linux	-	X	-

Table 4.1: Supported platforms, March 1999. Erlang can be ported to any system running C. Open source Erlang has also been ported to FreeBSD by an external user and included in their latest release.

used for communication with the host operating system and for interaction with programs written in other languages. If required for reasons of efficiency, a special version of this concept allows C programs to be directly linked into the Erlang runtime system.

4.6.7 Portability

Since Erlang is implemented in C it is essentially available on all systems that run C. Erlang is at present supported for the following operating systems; Solaris, Windows NT, VxWorks, and Linux, see Table 4.1.

4.6.8 Program Development

Erlang allows the same rapid prototyping and interactive development as, for example, Lisp but extended into the world of concurrency and distribution. The error handling mechanisms and hot code loading allow the design of high availability, robust, non-stop systems.

4.7 Match against Requirements

Table 4.2 matches Erlang against the requirements defined in Section 3 and Table 3.1 above.

Programming is both a *top down* activity from requirements and overall structure of the system and a *bottom up* activity based on the *abstractions* provided by the programming language. Abstractions such as *modules*, *processes*, *higher order functions*, etc. are to the programmer like transistors, capacitors, resistors, etc. to the hardware designer. It is important that the abstractions be few, simple to understand and yet powerful and provide the needed functionality. For example, if the application is inherently concurrent it would be very difficult to program without some process concept. In that case the application program itself would have to include some form of scheduler. Distribution, error handling, and hot code loading are extremely complicated requirements and the support for them provided by Erlang enables the programmer to concentrate on the application rather than on the basic programming technology.

1	*Handling of a very large number of concurrent activities*	Concurrency is provided through a *light weight process* concept and a typical Erlang implementation can handle 20-30,000 concurrent processes in one node.
2	*Actions to be performed at a certain point in time or within a certain time*	Erlang handles *soft* real time.
3	*Systems distributed over several computers*	An Erlang system may contain nodes distributed over many computers running different operating systems over a network.
4	*Interaction with hardware*	Erlang can easily communicate with hardware drivers.
5	*Very large software systems*	Modularisation is supported by the *module* concept. W.r.t. actual size, see Section 6.
6	*Complex functionality such as feature interaction*	Depends on the applications, see Section 6.
7	*Continuous operation for many years*	Depends on the applications, see section 6.
8	*Software maintenance (reconfiguration, etc.) without stopping the system*	Erlang permits *hot* code loading.
9	*Stringent quality and reliability requirements*	Depends on the applications, see Section 6.
10	*Fault tolerance both to hardware failures and software errors*	Erlang contains mechanisms to catch and contain errors and to design supervision structures.

Table 4.2: Matching Erlang against the requirements on a programming technology for telecommunication switching systems.

5 Open Telecom Platform

In late 1995, a remote access system was being developed at *Ericsson Telecom AB* (ETX) using *Asymmetrical Digital Subscriber Line* (ADSL) technology which enables fast transmission over copper wires and a decision had to be made regarding the selection of an appropriate programming technology. Three different proposals were prepared; one of which came from CSLab, termed the *Open System* proposal.

Joe Armstrong, Mike Williams, and myself had long proposed an open system approach where different technologies, computers, languages, databases, management systems, etc. could cooperate. Many such system components would come from suppliers and some from Ericsson's own developments (where either no existing component was available or Ericsson had technology that provided a commercial advantage).

The proposal [An95] from CSLab was based upon:

- Commercial processors,
- Commercial operating systems,
- Erlang,
- The productified Amnesia DBMS, renamed Mnesia,
- The productified SNMP agent,
- The BOS from the Mobility Server, rewritten and better integrated with Erlang, renamed *System Architecture Support Libraries* (SASL[1]),
- Productified development tools (debugger, interpreter, etc.),
- Interworking with device processors (usually programmed in C),
- Interworking with other software (protocol stacks, routing software, etc. usually written in C).

Careful estimates of execution times were also made. As it turned out the Open System proposal was given the go ahead with a tight schedule to produce a prototype system in six months, i.e., to deliver by the end of May 1996. I was appointed project manager of the first phase of the project and Mike Williams was appointed systems designer of the access node project.

[1]The name SASL usually means *St Andrews Static Language* which was a precursor to Miranda.

Immediately after the start of this project an even more important application project started, the development of an ATM switch. The large AXE/N project (where Erlang was not used) which had been running during 1987-95 had been closed down, which also caught the notice of the press [Ol95, Wa95a, Wa95b] and it was a matter of great urgency to rapidly fill this gap in the product range. The new project involved about 200 people among them at least 60 Erlang programmers. Erlang Systems became deeply involved with teaching and consulting.

At this point external marketing of Erlang for product development was stopped since all efforts were to be concentrated on the Open System project. Erlang Systems had worked for a couple of years on external marketing which turned out to be very difficult since few dared to use a programming language from Ericsson which was not widely used by Ericsson itself. Now when that situation had changed Erlang Systems could have made a large marketing push, but instead had to close down the external marketing. However, it was still permitted to deliver Erlang for research, education and prototyping, primarily to universities.

For some time the access node and the ATM switch projects were the only projects permitted to use the the emerging open system and for the same reason, to focus the “resources”. Other projects had to apply to the steering group for permission.

The system was named *Open Telecom Platform* (OTP) and the project delivered the first prototype system on time at the end of May 1996. From the beginning it was clear that management of OTP [To97] would have to be handled by a specific product unit and this was being created in parallel. After the first prototype phase Catrin Granbom took over as project manager and Seved Torstendahl as product manager.

Two weeks later the ATM project had passed their first dead-line. After that both OTP and the ATM project have stuck to their time schedules and the ATM product was announced in March 1998.

OTP system components:

- Distributed application management,
- SASL - error logging, release handling,
- OS resource monitoring,
- EVA - protocol independent event/alarm handling,
- Mnesia - real time active data replication,
- SNMP - operations and maintenance interface,
- INETS - simple HTTP support.

A key subsystem in OTP is the *System Architecture Support Libraries* (SASL) which give a framework for writing applications. SASL provides:

- Start-up scripts,
- An application concept,
- Behaviours (design patterns),

- Error handling,
- Debugging,
- High-level software upgrades in runtime without shutdown.

The *behaviours* provide the programmer with yet higher level abstractions for efficient program design [Dpw]:

- Supervision,
- Servers,
- Event handling,
- Finite State Machines.

SASL raises the level of abstraction and gives the system designers a powerful framework for systems design with built-in support for distribution, incremental code loading, error handling (and logging), etc. Mnesia and other OTP components are handled as “applications” by SASL.

SASL consolidated ideas on programming patterns that came both from the Erlang group at CSLab and the BOS developed by the prototyping team at EBC.

A new unit was created for management, support, and further development of OTP. Early 1997 the *OTP product unit* was formally established with Torbjörn Johnson as manager. Upon his retirement he was replaced by Mike Williams and in 1998, when he in turn moved on within Ericsson, Kenneth Lundin took over.

Technology transfer from CSLab to the OTP product unit was handled as follows:

- Already in the first prototype phase, the product unit took over systems integration and release management.
- From the second development phase, the product unit took over project leadership and product management.
- Designers from the product unit joined the different design teams (for complier, SASL, etc.) and CSLab personnel were phased out over a longer period.
- CSLab and the OTP product unit are still colocated.

By the end of 1998 the OTP unit numbered about 20 people and had taken complete control over the system. With successive releases new functionality has been added such as an implementation of the *Common Object Request Broker Architecture* (CORBA) [Omg, Si96]. This has been available since OTP release R5B in February 1998.

Erlang Systems adjusted their courses and developed an *OTP Programming* course.

6 Selected Industrial Applications

6.1 AXD 301 ATM Switch

The AXD 301 [Axd, Bl98, Bl99] is a new generation high performance *Asynchronous Transfer Mode* (ATM) switching system. It is extremely compact and has linear scalability of switching and control capacity. The system is intended for public network operators and *Internet Service Providers* (ISP's).

The AXD 301 is a key building block in Ericsson's multi-service network solution. Applications include ATM connectivity networks, scalable Frame Relay / ATM networks and *Multiprotocol Label Switching* (MPLS) for efficient handling of IP traffic as well as multi-service business networks and residential broadband networks. AXD 301 can also be combined with Ericsson's AXE narrow band switching system to provide a full range of narrow band services over ATM. All of these applications can run simultaneously on the same switch.

Main features:

- **Scalability:** From 10 Gbit/s in one subrack up to 160 Gbit/s,
- **Carrier class:** Non-stop operation,
- **Modularity:** Extendible with new functions,
- **Functionality:** All ATM Forum and ITU signalling and service categories,
- **Manageability:** Embedded web based element manager with a standard interface to management systems (i.e., SNMP).

The internal computing resources of the 10 Gbit/s switching system consist of:

- Two general-purpose control processors which handle network-signalling termination, call control, and operation & maintenance,
- Simple device-control processors, one on each ATM termination board and switch-core board for low-level control of the switch hardware.

For inter-processor communication and network signalling, every processor is connected to the ATM switch core. The device processors on the ATM termination boards are connected through the local switch port, the control processors and device processors on the switch core boards are connected (via the subrack backplane) to the switch port of the nearest ATM termination board.

During normal operation, one control processor handles calls, while the other processor handles operation and maintenance. In addition, each processor acts as a standby for its counterpart. In the event that one of the processors should fail or be taken out of operation, the system automatically switches over to single-processor mode.

An internal, distributed, real time database management system (based on Mnesia) copies data to each control processor, ensuring that configuration data and all data relating to operator ordered connection setup are protected from processor failure.

AXD 301 was announced in March 1998. Quote "with Erlang/OTP as the core technology, an open architecture was built, enabling the use of Erlang, C, Java [Go96] and dynamically generated HTML/JavaScript in the areas appropriate for each language:

- C for drivers, low level protocols and integrating third party components (280,000 lines).
- Java, JavaScript and HTML for the management interface. (2,500 lines of Java and 110,000 lines of JavaScript and HTML (generated code).)
- Erlang for control and management functions (290,000 lines)." [Wi98a]

This gives a nice illustration of the assumption made above that telecommunication systems would likely be designed using a combination of technologies.

The sourced software are from previous Ericsson projects but most importantly 80,000 lines of C from Trillium Software for handling *Private Network-Network Interface* (PNNI) routing.

(These figures have grown considerably since then, see Table 6.1, page 35, row 5.)

Even more important was that the project has been able to work in successive incremental prototypes. One drawback with the sequential "waterfall" [Yo92] model is the difficulty to go back when errors in design such as capacity problems occur. Functional but limited AXD 301 systems were available at a very early stage, thus enabling measurements and allowing the designers to evaluate their design. Successive versions refined the system and added new functionality.

To date 250 AXD 301 systems have been delivered to 20 countries. The AXD 301 system is also an integral part of Ericsson's ENGINE concept [Pr99c] which has been ordered by operators such as British Telecom and Telefónica.

6.2 ANx Access Node

ANx-DSL [Ni98] is Ericsson's access solution for delivery of high speed IP/ATM based services to medium, small business and residential users over existing copper networks.

The ANx-DSL system uses *Asymmetric Digital Subscriber Line* (ADSL) technology to dramatically increase bandwidth in the existing copper access network. This means that people working from home can download large data files faster than ever before. An additional advantage is that Ericsson's ANx ADSL platform supports digital video services.

The ANx-DSL equipment is installed in the operator's central office, where a splitter and a multiplexer separate voice and data signals to and from the subscriber, and concentrate data channels for forwarding to the data network backbone. Even at low take-up rates, this is a flexible and scaleable system, easy to install and extremely cost-effective.

The control system is based on OTP and Erlang. The database and the hard disk for secure storage of permanent data are also located at the control processor. Each board in the system has a device processor or a board controller, which is a low-end microprocessor. The device processor communicates with the control processor through in-band ATM communication or via a separate Ethernet LAN. The board controllers communicate via ATM with one of the device processors, which acts as a shelf controller; in other words, the device processor contains specific software for maintaining the board controllers. In ANx-DSL, the ADSL boards and the network termination boards have board controllers, whereas the exchange termination boards have device processors.

In October 1998 Ericsson signed an agreement with Telia to supply ANx during a two year period [Pr98].

6.3 GPRS

General Packet Radio Service (GPRS) [Gprs, Gr99] is a standard from the *European Telecommunications Standards Institute* (ETSI) on packet data in GSM systems. By adding GPRS functionality to the public land mobile network, operators can give their subscribers resource-efficient access to external IP networks and enable them to stay always connected.

GPRS offers air-interface transfer rates up to 115 kbit/s subject to mobile terminal capabilities and carrier interference. Moreover, GPRS allows several users to share the same air-interface resources and enables operators to base charging on the amount of transferred data instead of on connection time.

The world's first demo of GPRS (which had been developed in Erlang) was shown at CeBit 1998 and the two nodes, SGSN and GGSN, are developed based on Erlang/OTP. By the beginning of year 2000 Ericsson holds more than 50% of the world market in GPRS and in February, 2000, Ericsson presented the first live GPRS phone in the first end-to-end live GPRS network demo [Pr00b].

6.4 A Comment on Software Engineering

Experiences from the use of Erlang in many sometimes very large projects indicate clearly the two different traditions within *software engineering*. The most successful projects are run by enthusiastic teams, working hands-on and producing rapid results. The prime examples is the AXD 301 which developed a small executable system very early and then continued by building successive increments, carefully adding new functionality and all the time monitoring system performance.

Less successful has been the top-down methodological waterfall approach where several teams (perhaps spread over several countries) specify and code the whole system and then send their parts for integration test. With this approach there is much poorer feed-back to the designers and the whole idea of interactive programming (one of the strong points of functional programming) is lost.

Design efficiency is the key issue. All the telecoms requirements listed in Section 3 above can be solved with various techniques, even assembler, but then with a huge effort. Internal studies [Wi98b] show an increased productivity by at least a factor four compared to conventional (C, C++, Java ...) technologies. This enabled the AXD 301 project to apply an incremental work method. It should perhaps be added that most programmers like Erlang as, like other interpretative languages (Lisp, Java, etc.), it adds to their work satisfaction to see quick results [He00b].

6.5 Experiences from the Field

The Mobility Server has been delivered in about 450 systems on 15 markets and the designers have considerable experience working with the system:

- "We have 250,000 lines of code today, distributed among 500 modules in the Mobility Server 1 and only two or three fault reports from the whole world come in each month. That gives very low sustaining costs. Erlang has got a high level of abstraction, which allows the designers to concentrate on what they do, not how. That contributes to there being fewer flaws in the code." [Fe98]

Not only are there fewer faults but they are easier to find. Conventional systems produce hexadecimal dumps which are very difficult to interpret but Erlang produces *symbolic* information, lists, tuples, etc. This is the power of functional programming. Erlang is sometimes described as an untyped programming language, but the truth is that it is dynamically typed. All data items carry type information with them.

6.6 User Testimonies

The following statements have been made by users and have been used in marketing by Erlang Systems [En98]:

- "We believe from project start, and still believe, that if we had tried to use any other technology, we would not even remotely have been able to, within the desired time frame of roughly 1.5 years, reach the desired level of functionality, system software maturity and stability." {AXD 301}

- "Also, the short time for writing and testing Erlang code has enabled a truly incremental approach to software development." {AXD 301}

- "I would say that without Erlang/OTP we could never have accomplished the required functionality in the short time frame ..." {GPRS}

- "It takes 2 years to make a Plex programmer productive, it took 2 months in Erlang." {*Ericsson AS,* Norway}

6.7 External Users and Consultants

In March 1999 the following external companies used Erlang on licence from Erlang Systems:

- Beijing Telestar Telecom Technology Institute, China, Service Creation Environment,
- Borsalino, France, Business Tools,
- Brainpool, Sweden, SMS services [St99b],
- Motivity, Canada, Protocol converter,
- one2one, UK, IN services [Hi00],
- Telia Promotor, Sweden, Telia Call Guide [Na99].

Since then Brainpool has dropped their Erlang based product. On the other hand Sendmail Inc [Smw] has joined as Erlang user [Fr00].

The following external consultancy companies offer Erlang consultancy services and have been certified by Erlang Systems:

- Certeam,
- Cesarini Consulting Ltd,
- Connecta Teknik AB,
- ENEA Data,
- Sjöland & Thyselius,
- UPEC.

Erlang Systems and the OTP product unit operate a commercial Erlang web site [Erw] which contains documentation, information about current courses etc.

6.8 CeBit 1998 and Marketing Efforts

In 1998 there were about 14 projects ongoing based on Erlang/OTP. In addition there were many projects using just Erlang. At the CeBit international trade fair in Hannover in April 1998 there were no less than nine Erlang based system products on display in the Ericsson stand [Wa98a, Wa98b]:

- Auto generated *Graphic User Interface* (GUI) for *Intelligent Network* (IN) services,
- ANx-DSL fast access system,
- AXD 301 ATM switch,
- Database Access Gateway,
- GPRS packet switch over GSM,

- Mobility Server,
- Network Intelligent Call Centre,
- Telia Internet Conference Set-Up,
- Professional Mobile Radio over GSM.

In April 1997 the ban on external marketing was lifted and Erlang Systems employed Jane Walerud as Sales Manager in October. The marketing goals were set high. Erlang/OTP was to have 10,000 users and be used in product development in 5 companies other than Ericsson by the end of the year 2001. The agreed marketing strategy was to find major partners to take over the technology and to concentrate on one niche market at a time time while stepping up the publicity efforts.

During 1998, most possible major partners, including SUN and MicroSoft, declined the Erlang/OTP technology. Erlang Systems then concentrated on two niches:

- **Embedded systems** in a partnership with Wind River Systems [Wrw] with an implementation of Erlang/OTP on the VxWorks operating system [Wre].
- **High availability telephony** in a partnership with Natural Micro Systems [Nmw] who have a leadership in the compact PCI technology.

These partnerships would not get all the way to the goal of 10,000 users in three years, so starting in June 1998, Jane Walerud instead concentrated on the *open source* initiative (see below).

6.9 Refined Match against Requirements

Table 6.1 makes a further match of Erlang against the requirements defined in Section 3 and Table 3.1 above.

1	*Handling of a very large number of concurrent activities*	There are about 200-4000 concurrent processes active in AXD 301.
		GPRS release 1 has about 500 concurrent processes. The next release is expected to have about 1000-2000 concurrent processes.
		Mobility Server has about 200 static processes and generates six dynamic processes for each call.
2	*Actions to be performed at a certain point in time or within a certain time*	In AXD 301 performance measurements, equipment supervision and output of charging (billing) information is carried out at regular intervals. (The same applies to GPRS.)
		Implementations of communication protocols contain many timers supported in Erlang by the time-out mechanism in the *receive* statement and built-in timers.
3	*Systems distributed over several computers*	The 40 Gbit/s version of AXD 301 consists of 8 control processors and 64 device processors.
4	*Interaction with hardware*	Interaction with switching hardware is often done through device processors which can handle the "hard" real time requirements at low level.
		There are standard methods for interaction with C and for communication with hardware drivers.
5	*Very large software systems*	AXD 301 release 3 consists of about 525 KLOC Erlang, 608 KLOC C and 8 KLOC Java (plus the OTP).
6	*Complex functionality such as feature interaction*	AXD 301 supports all ITU and ATM Forum protocols and interworking between them.
		GPRS consists of many cooperating protocols.
7	*Continuous operation for many years*	The Mobility Server was released as a product in 1994 and there are now more than 400 units in operation around the world. Very few errors in the field are reported and most problems now relate to hardware (like fan motors giving up).
8	*Software maintenance (reconfiguration, etc.) without stopping the system*	AXD 301 supports soft upgrades. The system itself figures out how it should be upgraded, processes suspended, code loaded into the system, etc.
9	*Stringent quality and reliability requirements*	AXD 301 requirements specify service stops of no more than 6 minutes/year. At the present time the system has about 60 execution years in the laboratory. (20 test channels in operation for 3 years.)
		GPRS is expected to have a system availability of 99.995 %. This corresponds to a system downtime of about 26 minutes/year.
10	*Fault tolerance both to hardware failures and software errors*	In AXD 301 either central processor can take over the other. The system is built using supervision hierarchies as supported by OTP/SASL. The design rules specify that functions either handle correct inputs or crash. In the latter case the supervisor at a higher level will identify and handle the error.
		GPRS is built around *n* processors and if one goes down during operation it will only cause a slight degrading of capacity.

Table 6.1: Further match of Erlang against the requirements on a programming technology for telecommunication switching systems. KLOC = 1,000 lines of non-commented code. OTP itself contains about 240 KLOC Erlang.

7 Backlash

Once upon a time C++ [St91] was the contender vs. Erlang. However, after some large project failures C++ fell into disrepute and instead Java [Go96] appeared. Java is like a tidied up version of C++ without pointers and with builtin garbage collection. Java also supports concurrency in the form of threads (which, however, are implementation dependent). The question is whether Java is a suitable programming language for large robust systems where Erlang is currently used. Many systems (such as the AXD 301 noted above) use Erlang for servers and the telecommunications application and Java for the graphics oriented management system, i.e., clients. (Present Java implementations have much longer context switching times than Erlang, but this might be changed in the future such that Java could also be used for large embedded applications like Erlang.)

In February 1998, Erlang was banned within *Ericsson Radio AB* (ERA) for new product projects aimed for external customers because:

- "The selection of an implementation language implies a more long-term commitment than selection of processors and OS, due to the longer life cycle of implemented products. Use of a proprietary language, implies a continued effort to maintain and further develop the support and the development environment. It further implies that we cannot easily benefit from, and find synergy with, the evolution following the large scale deployment of globally used languages." [Ri98]

Ongoing Erlang user projects were also required to make a plan for how to remove the Erlang *dependence*. This policy was issued without prior notice to the OTP Product Unit or to CSLab which were then part of *Ericsson Telecom AB* (ETX).

This was part of a scheme to outsource software technology at Ericsson to Rational Inc. [Raw], a provider of *Computer Aided Software Engineering* (CASE) tools. The *Joint Development Initiative* (JDI) agreement between Ericsson and Rational was concluded in 1997. Rational acquired the Swedish software company SoftLab AB [Sow] which had a long history of working with Ericsson and, indeed, was responsible for maintenance and further development of the Plex compilers, a core technology for Ericsson. Rational, however, wanted to turn SoftLab into a sales and consulting company which caused some turmoil [Pe98] and in late 1998 part of SoftLab was taken over by Ericsson [Cs99].

Although the Erlang policy only applies to ERA, no major Erlang based projects were started at Ericsson during 1998 or 1999. However, with the ongoing projects using Erlang, the volume of training and consulting remains at the same level as before.

8 Open Source and Continued Research

Software has been available as *freeware* for many years, but this had not gained respectability in industry. In 1997 a new term appeared, *open source*, to describe the remarkable success of the Linux [Re97] development and also to provide a legal framework for handling maintenance and user's improvements to the system. Open source [Opd, Jo00b] also implies a way of spreading software where users need not feel locked in by single vendors that might disappear. The *open source* phenomenon was described by Eric S. Raymond in a paper [Ra99] that influenced Netscape's decision to release the Communicator 5.0 source code.

During the Autumn of 1998 a discussion was raised about releasing Erlang as open source in order to facilitate its spread externally and hopefully attract even Ericsson competitors to use it. A small group visited Red Hat Inc. [Rew], the company that distributes Linux, and preparations started at CSLab to create an open source web site. On December 2 the OTP steering group gave its permission and one week later *open source Erlang* [Opw] was released.

The earlier marketing efforts had concentrated on making a business from marketing Erlang as a programming language with an implementation. The focus now changed to *spreading* the language and to eradicate the "proprietary" image. In addition, the open source distribution is a considerably more mature product in that it contains the full OTP implementation (SASL, Mnesia, libraries, etc.) as well.

During December 1998 there were 72,933 requests to the open source Erlang site although no press release had been issued. This dropped the following month and then kept a level of about 35,000 requests for several months. However, from October 1999 this figure has been rising steadily and there were 126,341 requests in September 2000, see Figure 8.1. Four mirror sites [Miw] were established, KTH, University of Uppsala, University of Vienna and *Software Engineering Research Centre* (SERC) in Melbourne.

A second release of open source Erlang was made in early December 1999 and a couple of articles on Erlang were published in Computer Sweden in February 2000 [He00a, He00b]. Open source Erlang is now maintained by the OTP product unit.

8.1 Bluetail AB

Eddie was an "innovation cell" using Erlang to "provide the tools which allow the construction of mission critical internet sites" [Edw]. In the middle of December

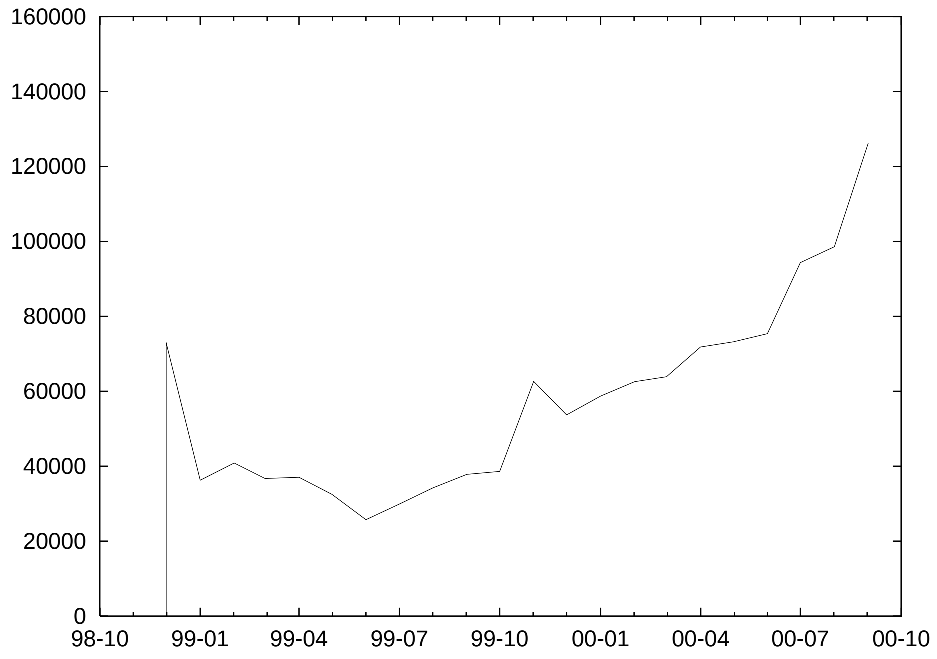

Figure 8.1: Total number of user requests per month to `www.erlang.org`.

1998 the Eddie team handed in their notices to leave Ericsson to set up their own company, *Bluetail AB* [Blw], based on external venture capital. (A new design team was found in Melbourne on my suggestion and have also set up a commercial company, *Lodbroker Pty* [Low].)

Bluetail "develops and sells software products to Internet organisations and thereby supplies reliability, scalability and managability to Internet services". Soon after several more people at CSLab, SARC (se below), the OTP product unit and Erlang Systems (in all about 15 people) left to join Bluetail. Among them were Joe Armstrong, Claes Wikström, and Robert Virding. Jane Walerud became Managing Director. I was invited to the board of Bluetail and participated with Ericsson's permission.

Bluetail received a lot of attention in the press [Ka00a, Sm00a]. Their first product was the *Mail Robustifier* and early in year 2000 they brought out the *Web Prioritizer* [Me00, Tå00]. Among Bluetail's customers were TeleNordia and SPRAY. At the beginning of year 2000, Bluetail signed a partnering agreement with Sendmail Inc [Smw].

In late August 2000 Alteon [Alw], a major US supplier of web switches, announced that they were in the process of acquiring Bluetail at a price of 1,4 billion SEK [Ka00b, Ol00, Pr00d, Sm00b]. Earlier in 2000 ADC Telecommunications bought Altitun for 7,9 billion SEK and Cisco bought Qeyton for 7,3 billion SEK. Altitun and Qyeton were, like Bluetail, Swedish startup companies founded largely by former Ericsson technicians [Au00].

By coincidence, Alteon is being bought by Nortel [Pr00c].

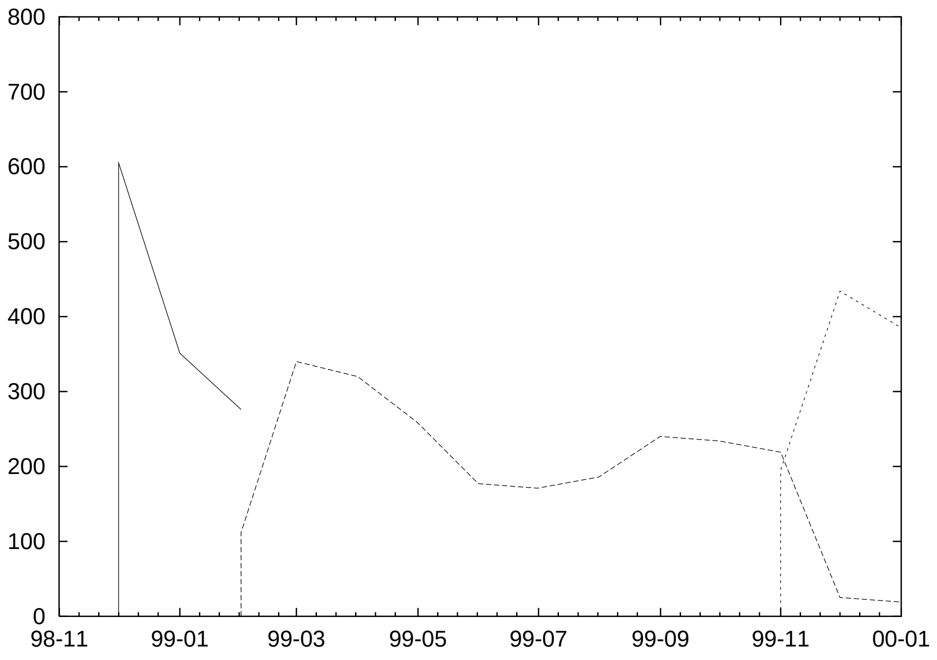

Figure 8.2: Total number of source code downloads per month from `www.erlang.org`. 1st release December 1998. Bug fix release February 1999. 2nd release November 1999.

8.2 Erlang in the Research World

Over the years a number of papers [Ar92b, Dä93b, Ha93, Vi93, Wi94, Ar95, Vi95, Wi96, Ar97a, Ar97b, Ar97c, Ar98, Da98, Ar99b, Ar99c, Ma99] have been presented on the development and use of Erlang in prestigious conferences and this has received much attention in the research world. As a consequence Joe Armstrong was invited as keynote speaker to the *ACM SIGPLAN International Conference on Functional Programming* on June 9-13, 1997, which took place in Amsterdam [Ar97b].

CSLab has worked with many distinguished researchers on Erlang and Erlang related topics, notably professors Marc Feeley (Montréal), Bengt Jonsson (Uppsala), Richard O'Keefe (Otago), and Philip Wadler (Glasgow, now at Lucent).

It has been a drawback, though, that the *Swedish Institute of Computer Science* (SICS) [Siw] has been so preoccupied with developing Oz [Moz]. However, distribution and error handling in Oz have been influenced by Erlang [Ha99].

Erlang has also helped to inspire work on distributed Haskell [Hu00, Po00].

At the *Twelfth International Workshop on Implementation of Functional Languages* [Mo00] in Aachen on September 4-7, 2000, there was a half-day session on Erlang thanks to Thomas Arts' initiative.

8.3 Research Continues

The wide exploitation of Erlang technology in the form of OTP did not mean the end of Erlang related research. Instead Erlang opened many constructive paths for continued research with a view to further industrial uses:

- Bogumil Hausman worked from 1992 on compiling Erlang to C with the aim of gaining execution speed. The compiled code obviously is considerably larger than the JAM byte code. *Bogdan's Erlang Abstract Machine* (BEAM) [Ha93, Ha94] uses the same run time library as JAM. In 1999 BEAM replaced JAM which is no longer supported [Ka99].

- Robert Virding developed an alternative implementation called *Virding's Erlang Engine* (VEE) using different techniques, notably generational garbage collection. In 1999 a new compiler was released which was based on VEE [Ka99].

- A Scheme [Dy96] implementation of Erlang. This is an ongoing project [Fe99] (partly funded by CSLab) at the University of Montréal lead by Marc Feeley, creator of Gambit Scheme [Gaw]. The purpose is to create an alternative Erlang implementation in the public domain.

- *High Performance Erlang* (HiPE) [Hiw, Jo99, Li99, Jo00a] is a project at *Computing Science Department* (CSD) [Csd], Uppsala University, to compile Erlang into Sun Microsystems Sparc native code using very aggressive optimization techniques. HiPE starts from the JAM byte code and concentrates on code generation. However, it is being redesigned to work from BEAM instead.

- Geoff Wong at *Software Engineering Research Centre* (SERC) [Sew], Melbourne, is performing research into *continuous system monitoring* [Wo98] whereby a separate computer monitors non-functional aspects like reliability, capacity, etc. during the actual operation.

- *List comprehensions, records* and *higher order functions* [Er44] have been added to Erlang. Most of this work has been done by Robert Virding.

- A *formal language specification* for Erlang is nearing completion. This work was started by Jonas Barklund of CSD [Csd] and is being finished by Robert Virding. It is available together with the open source release. The grammar used "is almost, but not quite, an LALR(1) grammar" [Ers].

- *Core Erlang* [Cew] is an intermediate representation of Erlang, intended to lie at a level between source code and the intermediate code typically found in compilers developed primarily by CSD [Csd].

- Richard O'Keefe from the University of Otago, New Zealand, has studied Erlang with tools and libraries very carefully and has proposed certain changes [Ok98] to the language and its libraries to widen the applicability of Erlang for very large applications.

- Further work on *real time garbage collection* [Vi95, Bo97].

1984-6	Language experiments
1987	Early Erlang
1988-9	ACS/Dunder application prototype
	JAM emulator
1990	Erlang presented at ISS'90
	Use for prototyping
1991	First fast implementation
	ASN.1 compiler to Erlang
1992	User product projects (Mobility Server) started
1993	Distributed Erlang
1994	User products launched
	SDL translator
1995	Several user projects started
	SNMP Master's Thesis
	Amnesia DBMS prototype
1996	OTP development started
	Type system and program verification
1997	General availability
1998	Open source distribution
	Type system and program verification
1999	BEAM replaces JAM

Table 8.1: Erlang Development.

- Simon Marlow and Phil Wadler of the University of Glasgow have developed a *type system* for Erlang [Ma97]. This was taken over by Thomas Arts at CSLab who has reworked the system [Tyw] and is beginning to find users for it. The vision is to be able to cover the entire spectrum from untyped fast prototyping to well controlled typed systems for production.

- A *program verification* system [Vew] for Erlang is being developed in co-operation with *Swedish Institute of Computer Science* (SICS) [Siw] and Thomas Arts of CSLab. This system is beginning to be used for finding bugs in protocol implementations [Ar98, Da98, Ar99a, Ar99b, Ar99c].

- Sven-Erik Nyström from CSD [Csd] has worked on *static analysis* of Erlang programs.

- Dan Sahlin at CSLab and Lawrie Brown from University College of New South Wales have worked on *Safe Erlang* [Br99], an extension of Erlang with capabilities to be able to handle imported software in a secure manner.

- A couple of prototype projects have implemented *intelligent agents* using Distributed Erlang notably a large student project at Uppsala [Jo97].

- *Department of Computer Systems* (DoCS) [Dow] at Uppsala University has investigated implementation of Erlang for very small operating systems and have made a preliminary implementation of Erlang on ExoKernel [Exw] which had been developed at M.I.T.

- Claes Wikström and Tony Rogvall proposed a further extension of Erlang with a *bit syntax* [Ro99] which significantly improves Erlang's capabilities for programming communication protocol stacks. This was implemented by Patrik Nyblom and available from the Erlang/OTP release in September 2000 [Ny00].

CSLab has close co-operation with several universities and from 1997 I was appointed chairman of the board of *Advanced Software Technology* (ASTEC) [Asw], a competence centre supported by NUTEK [Nuw] and primarily based at Uppsala university. The HiPE and Erlang verification projects are both run under ASTEC.

In March 1998, the *Software Architecture Laboratory* (SARC) with Håkan Millroth as manager was spawned off from CSLab. Håkan came from CSD [Csd] and was appointed adjunct professor at Uppsala. SARC works closely with CSLab and there is some overlap, but SARC will focus on higher levels of system architecture for example defining recommended standard solutions (*design patterns*) to typical subsystems recurring in telecommunications systems. When Håkan left for Bluetail he was replaced by Torbjörn Keisu.

9 Discussion

9.1 Development of Programming Technology

Once upon a time I imagined the development of programming language technology as a path towards successively higher levels of abstraction, see Figure 9.1.

- Machine code,
- Assembler programming,
- Higher Order Languages (Fortran, Pascal, etc.),
- Declarative (i.e. functional and logic) programming,
- Very high level, perhaps A.I. based ...
- ... and so on ...

Seen from a perspective of the 1970's (when the design of compilers and operating systems had just started to be done in higher order languages) we should now be well headed towards Very High Level Languages for yet the next generation. This has not happened and perhaps there will not be any development of this kind. Perhaps this view (inspired from developments in hardware from discrete components to VLSI) is a misunderstanding.

In fact, declarative languages are just about the same age as higher order languages since Lisp appeared only shortly after Fortran. This means that the different types of languages exist in parallel. Figure 9.2 is perhaps more appropriate.

Returning to the conclusion in the paper [Dä86] describing the experiments using different programming languages and techniques (see above) that a complete system likely would be built using a combination of techniques suitable

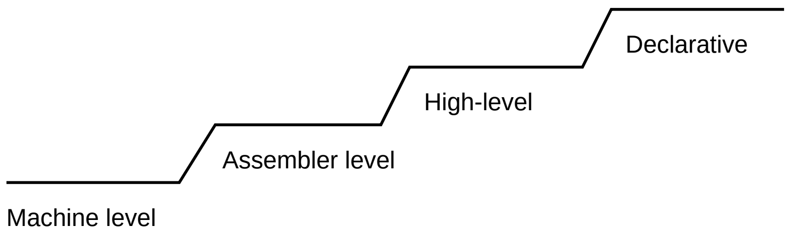

Figure 9.1: Successive programming language generations.

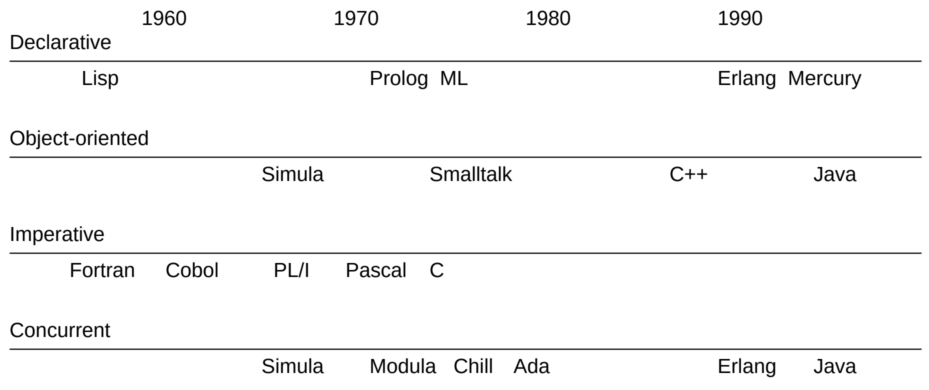

Figure 9.2: Concurrent programming language generations.

for different purposes, the question could be rephrased as why there is not a greater use of functional and logic programming? After all, they should be part of any programmer's "tool box".

9.2 On Applications of Functional Programming

Phil Wadler [Wa98d] presents the following list of possible reasons for the resistance to functional programming languages:

- Compatibility,
- Libraries,
- Portability,
- Availability,
- Packagability,
- Tools,
- Training,
- Popularity.

Most of these are self-defeating: because of the lack of X, no X will be created. All points except the last one are of a technical nature and can easily be remedied. The key point is the last which is a bit like a *Catch 22*. Lisp has been around a long time and proved itself many times without making the real break-through. There are also other functional programming languages of industrial quality, notably Clean [Clw], Mercury [Mew], and Oz [Moz].

It might well be that it is difficult to introduce functional programming into an old and established company culture. This, on the other hand, leaves the field wide open for exploitation by new companies free from tradition.

The *1st International Workshop on Practical Aspects of Declarative Languages* took place on January 18-19, 1999, at San Antonio, Texas. However, the only paper delivered by an industry representative was a paper on Mnesia [Ma99].

Stages	Actors	Cumulative % of Actors Adopting
Pioneer	Innovators	0-5 %
Early expansion	Early adopters	5-15 %
Takeoff	Popularizers	15-30 %
Bandwagon	Followers	30-80 %
Late	Conservatives	85-95 %
Terminal	Resistors	95-100 %

Table 9.1: Acceptance of New Technology.

9.3 Diffussion of Innovations

Everett Rogers in his book *Diffussion of Innovations* [Ro82] describes a series of "stages of adoption" for new technologies, see Table 9.1.

An important point in this model is that there is no smooth transition from one stage to the next. This applies especially when moving from the *takeoff* to the *bandwagon* since the first group are interested in the new technology and its possible uses whereas the second group is primarily interested in functionality which might, for example, be available to competitors.

With reference to Erlang's prototype projects, the early products and the projects following OTP development could be seen as the *early expansion.* Erlang/OTP was on the verge of *takeoff* in the form of projects that had seen these successes but were halted by the ERA policy.

9.4 The Magic of Words

This brings the question around to *marketing* where CSLab may have failed miserably. Technical merits may impress the technicians but something else is required to gain the acceptance of decision makers. It was mentioned above how "freeware" was treated with suspicion but when "open source" appeared (with an appropriate legal framework) it gained much higher respectability.

Recently we have seen another such case, "daily build". For many years the standard method for program system development has been the "water fall model" [Yo92] starting with specifications, ending with system integration. Against this has often been proposed "rapid prototyping" as a means to get an early check on implementability and performance. This essentially conservative and realistic approach is now gaining respectability as "daily build":

- "Focus on customer requirements and code - *Code is King*,
- Shorter distance between customer and programmer, who gets better understanding of the final product,
- The customer is able to see real progress in the form of executable code,
- Avoid a large integration problem at the end of the project." [Ol99]

Thus this apparently undisciplined *bottom-up* way of working gets accepted for its merits under a new term. The good point is that people who disliked fast prototyping can now embrace its advantages without having to concede that they could have been wrong earlier.

Also technicians working with applied research need to understand this magic of words. The term "functional programming" is old and worn and its marketing might need some new term.

9.5 Symbolic Programming strikes Back

One of the nice aspects of Lisp (mentioned above) is the use of S-expressions as an efficient way to save and read complex data structures, which also have the advantage that they can be easily inspected. This compares with hard coded binary structures and hexadecimal dumps.

The trend now is towards textual forms on a large scale, IETF protocols, postscript, HTML, and XML [Xmw]. Quote Phil Wadler:

- "In fact, XML is little more than a notation for trees and for tree grammars, a verbose variant of Lisp S-expressions coupled with a poor man's BNF (Backus-Naur form). [- - -] There is much for the language designer to contribute here. As all this is based on a sort of S-expression, is there a role for a sort of Lisp?" [Wa99]

10 Conclusions

For Erlang to be used inside Ericsson it was required that it was used outside and for Erlang to spread outside Ericsson there had to be wide use of it inside. The only way to get around this was a steady spread of the language in both spheres. In fact, in the dynamic world of telecommunications, the history of Erlang has proceeded in a see-saw fashion with focus alternating between internal and external use, see Table 10.1.

The development and use of Erlang shows that for a new programming language to be reasonably successful there are, at least, the following prerequisits:

- There has to be a sizeable and stable support organisation. The OTP product unit numbers 18-20 people financed by the in-house Ericsson projects.
- There must be training and consultants available.
- There has to be some niche that is sufficiently interesting and important for large sectors of industry. In Erlang's case high-availability, reliable, distributed systems and rapid design through high abstraction level and prototyping.
- The language must be reasonably simple to learn and to implement.

A most remarkable observation is that while hardware developments go ahead at great speed (note Moore's law) basic programming still remains at about the same level of technology as 30 years ago. Tremendous efforts are made on various methodolgies and the visions in industry seem to be in the direction of "software factories" where the work can be reduced to mere "coding".

Today when large numbers of people are available with university degrees in Computer Science there should be a greater emphasis on better technologies. Functional programming has been around just as long and difficult problems such as hot code loading and distributed programming can only reasonably be handled through better technology such as Erlang provides.

Was it worth the effort? Did the Erlang development produce the desired technical result and did it serve the needs for product development? The answer must be "yes" on both accounts. Erlang has also shown that:

- Functional programming can be used for very large applications involving large project teams.
- Functional programming can be used for industrial real time embedded applications.

	Internal usage	External usage	Comments
1984-6	-	-	Technology evaluations
1987-9	Use in prototypes	-	Experimental developments
1990-2		Academic distribution	Presented at ISS'90
			Experimental developments
			Noted at ISS'92
1993-5	Limited use in products	External marketing	*Erlang Systems* established
1996	Use for strategic product development	External marketing halted	OTP development
			OTP product unit established
1997		External marketing restarted	OTP development and deployment
1998	Nine products displayed at CeBit	3,323 evaluation systems delivered	
	Erlang stopped at ERA	Open source release	
1999	AXD 301 and GPRS win important orders	Growing use for product development	*Bluetail* started
2000			*Alteon* buys *Bluetail*

Table 10.1: The history of Erlang summarized.

- Functional programming gives significant commercial advantages in raising design productivity and enabling rapid developments through prototypes and successive increments.

Erlang provides many examples of the difficulty of technology introduction, notably:

- Erlang and functional programming in general both enable and require a new way of working with much more interactivity. The top-down waterfall methodologies were designed to handle conventional programming languages. Technology and methodology both have to be changed.

- Marketing a new programming language and a new way of working requires a huge effort and investment. Twice Erlang Systems has tried marketing Erlang with limited resources and with meager results. Sun has probably spent a fortune on Java but that has paid back in the form of increased demand for computer equipment. Ericsson is a telecommunications company and selling Erlang would not sell more switches.

- *Open source* combined with a good support organisation provided the real break-through. Many more programmers can try Erlang and companies know that support and education are available if needed.

When CSLab was established its aim was described thus

- "CSLab's responsibility in the long term is to create a basic software technology for future telecom systems and support systems and in the near term to contribute to the introduction of new software technology in current systems." [Dä84b]

With Erlang/OTP CSLab achieved this aim and has in the process shown that *applied research* in an industrial laboratory environment, indeed, works

Basic Erlang	1	Sequential Erlang
	2	Concurrent Erlang
	3	Error Handling
	4	POTS and Advanced Topics
Continued Erlang	1	Repetition, More about data types, Erlang 4.4 extensions
	2	ETS and TV, Code loading, Distributed Erlang
	3	Ports, Funs, List comprehensions
	4	Catch and throw, Robustness and efficiency, Cover, Graphics
OTP Programming	1	Overview, Behaviours, Behaviours: Servers
	2	Behaviours: Finite state machines, Supervisors
	3	Behaviours: Events, Applications, Special processes
	4	System configuration, Introduction to Mnesia
Advanced Erlang	1	Ports, The interface generator
	2	Linked-in drivers, Sockets
	3	erl-interface, C nodes
	4	Jive (interface to Java), Inets

Table 10.2: Course Curriculae (day by day), 1998.

as was the proposition in the key paper [Dä91]. While technical progress has continued steadily – marketing, dissemination, and other interaction with the external world have shown a much more uneven progress. However, it is hoped that these experiences can be useful for other software developers and hence they have formed the main theme of this thesis.

Year	Course	No of days	No of courses	No of students
1997	*Basic Erlang*	4	18	191
	Continued Erlang	4	7	91
	OTP Programming	4	6	71
	Advanced Erlang	4	2	11
	SNMP	2	1	5
	User adapted	1-5	7	87
	Total		41	456
1998	*Basic Erlang*	4	21	230
	Continued Erlang	4	6	51
	OTP Programming	4	13	153
	Advanced Erlang	4	2	14
	Total		42	448
1999	*Basic Erlang*	4	22	229
	Continued Erlang	4	5	48
	OTP Programming	4	9	80
	Erlang Literacy	5	1	14
	Seminars and Special Courses	1-5	6	72
	Total		43	443

Table 10.3: Courses 1997-1999. The *User Adapted* courses combined material from the other courses adapted for the needs of the particular project. The *Erlang Literacy* course is adapted for test and installation personnel. Most courses were given at Erlang Systems' premises in Kista north of Stockholm. Courses have also been given at the users' locations like Athlone (Ireland), Aachen and Hildersheim (Germany), Budapest (Hungary), Grimstad (Norway), Montréal (Canada), and Raleigh and Dallas (USA).

Australia	Adelaide Univ. CTIN
	Australian Defence Force Academy
	SERC, Melbourne
	University of Adelaide
Canada	Centre de réchérche informatique de Montréal
	Simon Fraser University, Vancover
	Université de Montréal
China	Beijing University of Posts & Telecom
	Shanghai JiaoTong University
Costa Rica	Inst. Technologica de Costa Rica
Croatia	University of Zagreb
Germany	Rheinisch-Westfalische Technische Hochschule
	Universität Kaiserslautern
Greece	National Technical University of Athens, CS dept
	National Technical University of Athens (NOC)
Hungary	Technical University Budapest, Math dept
India	Bengal Eng. College, Howrah
	Indian Inst. of Tech., Dehli
	Malaviya Regional Eng. College, Jaipur
Ireland	Trinity College Dublin
Italy	Coritel
Malaysia	University Teknologi Malaysi
The Netherlands	Katholike Universiteit Nijmegen
Russia	Tomsk State University
Spain	Jaume I University
	LFCIA
	Universidad Politecnica de Madrid
Sweden	AMU-Gruppen Syd
	Chalmers Tekniska Högskola
	Högskolan Ronneby/Karlskrona
	Ingenjörssskolan/KTH Haninge
	Ingenjörsskolan/KTH Kista
	Linköping University
	Mälardalens Högskola
	Royal Institute of Technology/NADA
	Royal Institute of Technology/Teleinformatics
	Stockholm University
	Swedish Institute of Computer Science
	Uppsala University
Thailand	Khon Kaen University
UK	University of Glasgow
	University of Hull
	University of York
USA	Central Michigan University
	UCSD
	University of California/LANL
	University of Minnesota-Morris
	University of Missouri
	University of Pennsylvania

Table 10.4: Academic licences, use of teaching materials, March 1999.

Bibliography

[Ad83] Reference Manual for the Ada Programming Language. ANSI/MIL-STD 1815, 1983.

[Ah92] Ingemar Ahlberg, John-Olof Bauner and Anders Danne. Prototyping Cordless using Declarative Programming. *XIV International Switching Symposium.* Yokohama, October 25-30, 1992.

[Ah93] Ingemar Ahlberg, John-Olof Bauner and Anders Danne. Prototyping Cordless using Declarative Programming. *Ericsson Review*, no 2, 1993.

[Al84] Magnus Alburg and Bjarne Däcker. Comparison between Lisp and Pascal for Use in Developing Programming Support Environments. *NT-P Symposium on Languages and Methods for Telecommunications Applications.* Åbo, March 6-8, 1984.

[Alw] Alteon WebSystems. Web site `http://www.alteonwebsystems.com`

[An95] Matz Andersson, Joe Armstrong, Lars Borg, Bjarne Däcker (chairman), Per Hedeland, Hans Heilborn, Tommy Johansson, Sebastian Strollo, Tony Rogvall, Claes Wikström and Mike Williams. ATM Control System, Proposal from the Open Platform Group. EUA/SU 95 038. 1995-10-10. Internal paper.

[Ar86] Joe Armstrong, Nabiel Elshiewy and Robert Virding. The Phoning Philosophers' Problem or Logic Programming for Telecommunications Applications. *Third IEEE Symposium on Logic Programming.* Salt Lake City, September 23-26, 1986.

[Ar90] Joe Armstrong and Robert Virding. Erlang - An Experimental Telephony Programming Language. *XIII International Switching Symposium.* Stockholm, May 27-June 1, 1990.

[Ar92a] Joe Armstrong, Bjarne Däcker, Robert Virding and Mike Williams. Implementing a Functional Language for Highly Parallel Real Time Applications. *Software Engineering for Telecommunication Systems and Services.* Florence, March 30-April 1, 1992.

[Ar92b] Joe Armstrong, Robert Virding and Mike Williams. Use of Prolog for Developing a new Programming Language. *The Practical Application of Prolog.* London, April 1-3, 1992.

[Ar93] Joe Armstrong, Robert Virding and Mike Williams. Concurrent Programming in Erlang. Prentice-Hall, 1993, ISBN 0-13-285792-8, 1st edition.

[Ar95] Joe Armstrong and Robert Virding. One Pass Real Time Generational Mark-sweep Garbage Collection. *International Workshop on Memory Management.* Kinross, Scotland, September 27-29, 1995.

[Ar96a] Joe Armstrong, Robert Virding, Claes Wikström and Mike Williams. Concurrent Programming in Erlang. Prentice-Hall, 1996, ISBN 0-13-285792-8, 2nd edition.

[Ar96b] Joe Armstrong. Erlang - A Survey of the Language and its Industrial Applications. *Ninth Exhibition and Symposium on Industrial Applications of Prolog.* Tokyo, October 16-18, 1996.

[Ar97a] Joe Armstrong. Design Patterns for Designing Switching Software. *High Level Concurrent Languages.* Schloss Dagstuhl, January 20-22, 1997.

[Ar97b] Joe Armstrong. The Development of Erlang. *ACM SIGPLAN International Conference on Functional Programming.* Invited paper. Amsterdam, June 9-13, 1997.

[Ar97c] Joe Armstrong and Thomas Arts. Erlang and its Applications. *Workshop on Constraint Programming for Time Critical Applications.* Invited paper. Schloss Hagenberg, Austria, October 27-28, 1997.

[Ar98] Thomas Arts, Mads Dam, Lars-Åke Fredlund and Dilian Gurov. System Description: Verification of Distributed Erlang Programs. *Fifteenth International Conference on Automated Deduction.* Lindau, July 5-10, 1998.

[Ar99a] Thomas Arts and Izak van Langevelde. How muCRL supported a Smart Redesign of a Real-life Protocol. *International workshop on Formal Methods in Industrial Critical Systems.* Trento, July, 1999.

[Ar99b] Thomas Arts and Jürgen Giesl. Applying Rewriting Techniques to the Verification of Erlang Processes. *Computer Science Logic.* Madrid, September, 1999.

[Ar99c] Thomas Arts and Mads Dam. Verifying a Distributed Database Lookup Manager written in Erlang. *World Congress on Formal Methods.* Toulouse, September, 1999.

[Asw] Advanced Software Technology. Competence center at Uppsala university. Web site `http://www.docs.uu.se/astec/`

[At84] The UNIX System. *AT&T Bell Laboratories Technical Journal*, vol 63, no 8, part 2, October, 1984.

[Au00] Tomas Augustsson. *Svenska Dagbladet*, August 30, 2000.

[Axd] AXD 301 High-Performance IP & ATM Switch. Web site `http://www.ericsson.se/datacom/products/wan_core/axd301`

[Ba86] Victor R. Basili, Richard W. Selby and David H. Hutchens. Experimentation in Software Engineering. *IEEE Transactions on Software Engineering*, no 7, July, 1986.

[Bj95] Martin Björklund and Klas Eriksson. A Framework for SNMPv2 in Erlang. KTH/NADA, Master's Thesis, 1995.

[Bl98] Staffan Blau and Jan Rooth. AXD 301 - A new Generation ATM Switching System. *Ericsson Review*, no 1, 1998.

[Bl99] Staffan Blau, Jan Rooth, Jörgen Axell, Fiffi Hellstrand, Magnus Buhrgard, Tommy Westin and Göran Wicklund. AXD 301: A new Generation ATM Switching System. *Computer Networks*, no 31, 1999, pp 559-582.

[Blw] Bluetail AB. Web site `http://www.bluetail.com`

[Bo97] Kent Boortz and Dan Sahlin. A Compacting Garbage Collector for Unidirectional Heaps. *Ninth International Workshop on Implementation of Functional Languages.* St Andrews, Scotland, September 1997. Selected Papers, Springer-Verlag LNCS Vol 1467.

[Br75] Fred P. Brooks. The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley Publishing Company, 1975.

[Br99] Lawrie Brown and Dan Sahlin. Extending Erlang for Safe Mobile Code Execution. *The Second International conference on Information and Communication Security.* Sydney, Australia, November, 1999.

[Bu90] Alan Burns and Andy Wellings. Real-time Systems and their Programming Languages. Addison-Wesley Publishing Company Inc, 1990, ISBN 0-201-17529-0.

[Bu92] M Buhgard, P Granestrand, M Lindblom and L Thylén. Photonic Switching in High Capacity Networks. *XIV International Switching Symposium.* Yokohama, 1992.

[Bå84] Göran Båge. The Programming Language EriPascal. LME/UE 83 018, 1984-05-04. Internal paper.

[Ca99] Maurice Castro. Erlang in Real Time. ISBN: 0864447434. 1999.
Web site `http://www.serc.rmit.edu.au/~maurice/erlbk/`

[CC84a] Specification and Description Language SDL. C.C.I.T.T. Recommendation Z.100, 1984.

[CC84b] CCITT High Level Language CHILL. C.C.I.T.T. Recommendation Z.200, 1984.

[Cew] Core Erlang Initiative.
Web site `http://www.csd.uu.se/projects/hipe/corerl/`

[Cl81] W. F. Clocksin and C. S. Mellish. Programming in Prolog. Springer-Verlag, 1981.

[Clw] Concurrent Clean Home Page.
Web site `http://www.cs.kun.nl/~clean/`

[Cs99] Ericsson köper del av SoftLab. *Computer Sweden*, January 18, 1999.

[Csd] Computing Science Department (CSD). Uppsala University.
Web site `http://www.csd.uu.se`

[Csw] Computer Science Laboratory. Ericsson Utvecklings AB.
Web site `http://www.ericsson.se/cslab`

[Da96] Hans Dahlquist. Tunnelbygget blir flera hundra miljoner dyrare. *Ny Teknik*, no 40, 1996.

[Da98] Mads Dam, Dilian Gurov and Lars-åke Fredlund. Compositional Verification of Erlang Programs. *Third International Workshop on Formal Methods for Industrial Critical Systems.* Amsterdam, May 25-26, 1998.

[Di91] Distorsion. Student project with 17 participants. UU/DoCS, 1991.

[Dow] Department of Computer Systems (DoCS). Uppsala University.
Web site `http://www.docs.uu.se`

[Dpw] Erlang Programming Rules and Conventions.
Web site `http://www.erlang.se/doc/programming_rules.shtml`

[Dy96] Kent Dybigg. The Scheme Programming Language: ANSI Scheme. Prentice-Hall. ISBN 0-13-454646-6.

[Dä79] Bjarne Däcker. EriChill/EriPascal Programmeringsspråk. Förslag. LME/X/Td 2419. 1979-06-15. Internal paper.

[Dä83] Bjarne Däcker. Using Lisp to Develop Programming Support Environments in the Industrial Environment. *International Workshop on Software Development Tools for Telecommunication Systems.* Anaheim, April 6-8, 1983.

[Dä84a] Bjarne Däcker, Nabiel Elshiewy, Per Hedeland, Carl Wilhelm Welin, and Mike Williams. Experiments with Programming Languages and Techniques for Telecommunications Applications. ETX/XT/DU 84 030. January, 1984. Internal paper.

[Dä84b] Bjarne Däcker. XT/DU Datalogi. Ansvarsbeskrivning. ETX/XT/DU 84 048. April, 1984. Internal paper.

[Dä86] Bjarne Däcker, Nabiel Elshiewy, Per Hedeland, Carl Wilhelm Welin, and Mike Williams. Experiments with Programming Languages and Techniques for Telecommunications Applications. *Software Engineering for Telecommunication Switching Systems.* Eindhoven, April 14-18, 1986.

[Dä89] Bjarne Däcker and Kerstin Ödling. ACS/DUNDER. Software Architecture and Technology. EBC/KX/DM 89 101. December, 1989. Internal paper.

[Dä91] Bjarne Däcker. Management of Technology with Regard to Software. *First Australian Conference on Telecommunications Software.* Invited paper. Melbourne, April 22-24, 1991.

[Dä93a] Bjarne Däcker. Erlang - A New Programming Language. *Ericsson Review*, no 2, 1993.

[Dä93b] Bjarne Däcker. Breakthrough in Software Design Productivity through the Use of Declarative Programming. *Eighth World Productivity Congress.* Stockholm, May 23-27, 1993.

[Dä94a] Bjarne Däcker. Introducing Concurrent Functional Programming into the Telecommunications Industry. *TELECOM'94.* Varna, September 20-22, 1994.

[Dä94b] Bjarne Däcker. Industrial Applications of Declarative Programming. *SOFT 13 - Improved Productivity of Quality Software.* Linköping, October 3-4, 1994.

[Dä95] Bjarne Däcker. The Development and Use of Erlang. Concurrent Functional Programming in Industry. *ConTel'95. Conference on Telecommunications.* Zagreb, June 7-9, 1995.

[Edw] The Eddie Open Source Project. Web site `http://www.eddieware.org`

[Ek79] T Ekman and G Eriksson. Programmering i Fortran 77. Studentlitteratur 1979. ISBN 91-44-16663-X.

[En98] *Erlang/OTP News*, CeBit Special, April, 1998.

[Er44] Erlang Extensions Since 4.4
Web site `http://www.erlang.org/doc/r7a/doc/extensions/part_frame.html`

[Er92] Dick Eriksson, Mats Persson and Kerstin Ödling. A Switching Software Architecture Prototype Using Real Time Declarative Language. *XIV International Switching Symposium.* Yokohama, 1992.

[Er95] Bernt Ericson. Applied Research at Ericsson. LME/DT-95:3003 Ue. January, 1995. Internal paper.

[Erl] Table of the Erlang Loss Formula. Telefon AB LM Ericsson. X/Yg 102 903 Ue. Stockholm, 1979.

[Erm] Enhanced Radio MEssaging System (ERMES).
Web site `http://www.ermes.org`

[Ers] Erlang Language Specification.
Web site `http://www.erlang.org/download/erl_spec47.ps.gz`

[Erw] Erlang Systems. Web site `http://www.erlang.se`

[Euc99] Fifth International Erlang/OTP User Conference, Stockholm, September 30, 1999. Web site `http://www.erlang.se/euc/99`

[Euc00] Sixth International Erlang/OTP User Conference, Stockholm, October 3, 2000. Web site `http://www.erlang.se/euc/00`

[Exw] ExoKernel Home Page.
Web site `http://www.pdos.lcs.mit.edu/exo.html`

[Fe98] Anna Fedoriw. Easy Design, Fewer Flaws and Low Sustaining Costs. *Erlang/OTP News*, February 1998.

[Fe99] Marc Feeley, Patrick Piché, Sylvain Beaulieu, Martin Larosse, and Mario Latendresse. Status Report on the ETOS Erlang to Scheme Compiler. *Fifth International Erlang/OTP User Conference.* Stockholm, September 30, 1999.

[Fo79] C. L. Forgy. OPS4 User's Manual. Technical Report CMU-CS-79-132. Department of Computer Science. Carnegie-Mellon University, 1979.

[Fo89] I. Foster and S. Taylor. STRAND. New Concepts in Parallel Processing. Prentice-Hall, 1989.

[Fr93] Magnus Fröberg. Automatic Code Generation from SDL to a Declarative Programming Language. *Sixth SDL Forum.* Darmstadt, October 11-15, 1993.

[Fr00] Scott Lystig Fritchie, Jim Larson, Nick Christenson, Debi Jones, Lennart Öhman. Sendmail Meets Erlang: Experiences Using Erlang for Email Applications. *Sixth International Erlang/OTP User Conference.* Stockholm, October 3, 2000.

[Gaw] Gambit Scheme Home Page.
Web site `http://www.iro.umontreal.ca/~gambit`

[Gi94] W. Wayt Gibbs. Software's Chronic Crisis. *Scientific American.* September, 1994.

[Go96] James Gosling, Bill Joy and Guy Steele. The Java Language Specification. Addison-Wesley, 1996.

[Gprs] Always "on-line" with GPRS. Web site
`http://www.ericsson.se/wireless/products/mobsys/gsm/subpages/wise/gprs.shtml`

[Gr82] Ove Granstrand. Technology, Management and Markets. Pinter. London, 1982.

[Gr99] Håkan Granbohm and Joakim Wiklund. GPRS - General Packet Radio Service. *Ericsson Review*, no 2, 1999.

[Ha93] Bogumil Hausman. Turbo Erlang. *International Logic Programming Symposium.* Vancouver, October 26-29, 1993.

[Ha94] Bogumil Hausman. Turbo Erlang: Approaching the Speed of C. In *Implementations of Logic Programming Systems*, pp. 119-135. Kluwer Academic Publishers, 1994.

[Ha99] Seif Haridi. Missförstånd om Mozart. Letter to the Editor. *Datateknik 3.0*, no 6, 1999.

[Haw] Haskell Home Page. Web site `http://haskell.cs.yale.edu`

[He76] Göran Hemdal. AXE 10 - Software Structure and Features. *Ericsson Review*, no 2, 1976.

[He98] Pekka Hedqvist. A Parallel and Multi-threaded Erlang Implementation. UU/CSD, Master's Thesis, 1998.

[He00a] Thomas Hedlund. Språket bäst i komplexa realtidssystem. *Computer Sweden*, no 11, 2000.

[He00b] Thomas Hedlund. Roligt att utveckla i Erlang. *Computer Sweden*, no 11, 2000.

[Hi00] Sean Hinde. Use of Erlang/OTP as a Service Creation Tool for IN Services. *Sixth International Erlang/OTP User Conference.* Stockholm, October 3, 2000.

[Hiw] High Performance Erlang.
Web site `http://www.csd.uu.se/projects/hipe/osh`

[Hj90] Thomas Hjalmarsson. AXE 10 Central Processors. *Ericsson Review*, no 1, 1990.

[Ho83a] Sören Holmström. PFL - A Functional Language for Parallel Programming. Report no 83.03-R, Programming Methodology Laboratory. Chalmers University of Technology, 1983.

[Ho83b] R. C. Holt. Concurrent Euclid, the UNIX System and TUNIS. Addison-Welsley, 1983.

[Hu87] Joan Kirkby Hughes. PL/I Structured Programming. John Wiley & Sons Inc, 1987.

[Hu89] John Hughes. Why Functional Programming Matters. *The Computer Journal*, vol 32, no 2, 1989.

[Hu00] Frank Huch and Ulrich Norbisrath. Distributed Programming in Haskell with Ports. *Twelfth International Workshop on Implementation of Functional Languages.* Aachen, September 4-7, 2000.

[Isw] Integrated Systems, Inc. Web site `http://www.isi.com`

[Iv90] Ny generation av programspråk på väg. *Framsteg inom forskning och teknik 1990.* IVAs årsbok 1990.

[Je75] Kathleen Jensen and Niklaus Wirth. Pascal - User Manual and Report. Springer-Verlag, 1975.

[Jo88] Geraint Jones and Michael Goldsmith. Programming in Occam2. Prentice-Hall, 1988.

[Jo97] Ing-Marie Jonsson, Dan Sahlin et al. A Platform for Secure Mobile Agents. *Practical Applications of Agents and Mobility.* London, April 21-23, 1997.

[Jo99] Erik Johansson, Sven-Olof Nyström, Mikael Pettersson, and Konstantinos Sagonas. HiPE: High Performance Erlang. ASTEC Technical Reports 1999. Web site `http://www.docs.uu.se/astec/Reports`

[Jo00a] Erik Johansson, Mikael Pettersson, and Konstantinos Sagonas. A High Performance Erlang System. *2nd International Conference on Principles and Practice of Declarative Programming.* Montréal, September 20-22, 2000.

[Jo00b] Torbjörn Johnson. Open Source Software - Industriell Användning. Sveriges Verkstadsindustrier, 2000 (to be published).

[Ka68] K. Katzeff and T. Andersson. The Tumba Stored Program Controlled Telephone Exchange. *Ericsson Review*, no 3, 1968.

[Ka99] Magnus Karlson. Coming Releases of Erlang/OTP. *Fifth International Erlang/OTP User Conference.* Stockholm. September 1999.

[Ka00a] Lars Anders Karlberg. Genombrott för forskarna som Ericsson inte ville ha. *Dagens IT*, no 7, February 16, 2000.

[Ka00b] Lars Anders Karlberg. Nortel köper avhoppade Ericsson-forskare. *Dagens IT*, August 28, 2000.

[Ke78] B. W. Kernighan and D. M. Ritchie. The C Programming Language. Prentice-Hall, 1978.

[Li79] Barbara Liskov et al. CLU Reference Manual. MIT/LCS/TR-225, 1979.

[Li99] Thomas Lindgren and Christer Jonsson. The Design and Implementation of a High-Performance Erlang Compiler. ASTEC Technical Reports 1999.

[Low] Lodbroker Pty. Web site `http://www.lodbroker.com/`

[Ma86] Peter Magnéli. Communications Computer APN 167 with ERIPASCAL. *Ericsson Review*, no 4, 1986.

[Ma88] D. Maier and D. S. Warren. Computing with Logic: Logic Programming with Prolog. Benjamin Cummings, 1988.

[Ma97] Simon Marlow and Philip Wadler. A Practical Subtyping System for Erlang. *ACM International Conference on Functional Programming.* 1997.

[Ma99] Håkan Mattsson, Hans Nilsson and Claes Wikström. Mnesia - A Distributed Robust DBMS for Telecommunications Applications. *First International Workshop on Practical Aspects of Declarative Languages.* San Antonio, Texas, January 18-19, 1999.

[Mc65] LISP 1.5 Programmer's Manual. J. McCarthy et al. M.I.T. Press, Cambridge, 1965.

[Mdw] MD110 BC10.
Web site `http://www.ericsson.se/enterprise/portfolio/system`

[Mew] The Mercury Project.
Web site `http://www.cs.mu.oz.au/research/mercury/`

[Me00] Jan Melin. Trogna besökare får gräddfil på Internet. *Ny Teknik*, no 8, 2000.

[Mi98] Håkan Millroth. Platform for High-Availability Applications: Erlang/OTP vs Java. Internal paper.

[Miw] Open Source Erlang. Mirror Sites.
Web site `http://www.erlang.org/mirrors.html`

[Mlw] The ML language.
Web site `http://burks.bton.ac.uk/burks/language/ml/`

[Mo93] Francisco Monfort. Control Switching Implementation of the BIPED Demonstrator. *Second Australian Conference on Telecommunications Software.* Sydney, 1993.

[Mo00] Markus Mohnen and Pieter Koopman, editors. Proceedings of the 12th International Workshop on Implementation of Functional Languages. Aachen, September 4-7, 2000. Aachener Informatik-Berichte, ISSN 0935-3232.

[Moz] Mozart Programming System. Web site `http://www.mozart-oz.org`

[Mö82] Rolf Mörlinger. MD 110 - a Digital SPC PABX. *Ericsson Review*, no 1, 1982.

[Na99] Hans Nahringbauer. Telia Call Guide. *Fifth International Erlang/OTP User Conference.* Stockholm. September 1999.

[Nmw] Natural Micro Systems.
Web site `http://www.naturalmicrosystems.com`

[Ni96a] Hans Nilsson. Amnesia - An Industrial Deductive DBMS with Transactions and Distribution. *Logic Databases and the Meaning of Change.* Dagstuhl, September 23-27, 1996.

[Ni96b] Hans Nilsson and Claes Wikström. Mnesia - An Industrial DBMS with Transactions, Distribution and a Logical Query Language. *International Symposium on Co-operative Database Systems for Advanced Applications.* Kyoto, 1996.

[Ni98] Patrik Nilsson and Michael Persson. ANx - High speed Internet Access. *Ericsson Review*, Special Issue on Internet Access, 1998.

[Nuw] NUTEK - Närings och teknikutvecklingsverket.
Web site `http://www.nutek.se`

[Ny00] Patrik Nyblom. The Bit Syntax - The Released Version. *Sixth International Erlang/OTP User Conference.* Stockholm, October 3, 2000.

[Ok90] Richard O'Keefe. The Craft of Prolog. The MIT Press, 1990.

[Ok98] Richard O'Keefe. Abstract Patterns for Erlang. *Fourth International Erlang/OTP User Conference.* Stockholm, September 22-23, 1998.

[Ol00] Nils-Olof Ollevik. Alteon köper Bluetail. Svenska Dagbladet. August 28, 2000.

[Ol95] Hans Olsson. Ericsson lägger ner utveckling. Dagens Nyheter. December 8, 1995.

[Ol99] Kent Olsson and Even-André Karlsson. Daily Build - Rapid Development and Control. *Sveriges Verkstadsindustrier*, 1999.

[Omg] Object Management Group. Web site `http://www.omg.org`

[Opd] Open Source Definition.
Web site `http://www.opensource.org/osd.html`

[Opw] Open Source Erlang.
Web site `http://www.erlang.org`
Current statistics `http://www.erlang.org/stats.html`

[Pe89] Mats Persson. ACS/Dunder Prototyping Report. Executive Summary/Management Report. EBC/KX/DC 89:069. December, 1989. Internal paper.

[Pe98] Eia Persson. 20 jobb är i farozonen. *Östgötacorrespondenten*, November 21, 1998.

[Po00] R F Pointon, P W Trinder, and H-W Loidl. The Design and Implementation of Glasgow distributed Haskell. *Twelfth International Workshop on Implementation of Functional Languages.* Aachen, September 4-7, 2000.

[Pr98] Ericsson signs two year contract for ADSL with Telia. *Ericsson Press Releases.* October 19, 1998.

[Pr99a] Ericsson wins groundbreaking GBP 270 million contract with BT. *Ericsson Press Releases.* January 21, 1999.

[Pr99b] Ericsson and T-Mobil in world's first GPRS contract. *Ericsson Press Releases.* January 26, 1999.

[Pr99c] Ericsson presents ENGINE. *Ericsson Press Releases.* November 16, 1999.

[Pr00a] Ericsson sells its Energy Systems business to Emerson Electric. *Ericsson Press Releases.* January 18, 2000.

[Pr00b] Ericsson shows first live GPRS phone in first end-to-end live GPRS network demo. *Ericsson Press Releases.* February 2, 2000.

[Pr00c] Nortel Networks to Acquire Alteon WebSystems for US$7.8 Billion. *Alteon Press Releases.* July 28, 2000.

[Pr00d] Alteon WebSystems to Acquire Bluetail for $152 Million. *Alteon Press Releases.* August 28, 2000.

[Ra88] Lars Ramqvist. Ericsson's Strategies and Technologies for the 1990's. *Ericsson Review,* no 3, 1988.

[Raw] Rational Inc. Web site `http://www.rational.com`

[Ra99] Eric S. Raymond. The Cathedral and the Bazaar. Web site
`http://www.tuxedo.org/~esr/writings/cathedral-bazaar`

[Re97] Red Hat Linux 5.0. The Official Red Hat Linux Installation Guide. Red Hat Software Inc., 1997.

[Rew] Red Hat Inc. Web site `http://www.redhat.com`

[Ri98] Tommy Ringqvist. BR Policy concerning Use of Erlang. ERA/BR/TV-98:007. March 12, 1998. Internal paper.

[Ro82] Everett Rogers. Diffussion of Innovations. Free Press, Chicago, 1982.

[Ro85] Anders Rockström. An Introduction to the C.C.I.T.T. SDL. Televerkets tryckeri. Stockholm. 1985

[Ro99] Tony Rogvall and Claes Wikström. Protocol Programming in Erlang using Binaries. *Fifth International Erlang/OTP User Conference.* Stockholm. September 30, 1999.

[Sa96] Dan Sahlin. The Concurrent Functional Programming Language Erlang - An Overview. *Joint International Conference and Symposium on Logic Programming.* Bonn, September 2-6, 1996.

[Sew] Software Engineering Research Centre.
Web site `http://www.serc.rmit.edu.au`

[Si96] Jon Siegel. CORBA. Fundamentals and Programming. John Wiley and Sons Inc. 1996.

[Siw] Swedish Institute of Computer Science.
Web site `http://www.sics.se`

[Sk86] Roger Skagerwall and Carl Wilhelm Welin. Design of an Expert System and Man-Machine Interface for Operation and Maintenance of AXE Telephone Exchanges. *International Seminar on Digital Communications.* Zürich, March 11-13, 1986.

[Sm83] C. H. Smedema, P. Medema and M. Boasson. The Programming Languages Pascal, Modula, CHILL and Ada. Prentice-Hall, 1983, ISBN 0-13-729756-4.

[Sm00a] Johan Smitt. USA nästa för uppstickare. *Dagens Nyheter*, February 20, 2000.

[Sm00b] Johan Smitt. Miljardklipp för Bluetail. *Dagens Nyheter*, August 29, 2000.

[Smw] Sendmail, Inc. Web site `http://www.sendmail.com`

[Sow] SoftLab AB. Web site `http://softlab.ericsson.se/`

[St90] D. Steedman. Abstract Syntax Notation One (ASN.1) Tutorial and Reference. Technology Appraisals, 1990.

[St91] Bjarne Stroustrup. The C++ Programming Language. Addison-Wesley, 1991.

[St99a] William Stallings. SNMP, SNMPv2, SNMPv3, and RMON 1 and 2. Addison-Wesley, 1999.

[St99b] Fredrik Ström. Use of Erlang/OTP in the Brainpool M/3 Communication System. *Fifth International Erlang/OTP User Conference.* Stockholm, September 30, 1999.

[Th95] Simon Thompson. Miranda. The Craft of Functional Programming. Addison-Wesley, 1995.

[To97] Seved Torstendahl. Open Telecom Platform. *Ericsson Review*, no 1, 1997.

[Tyw] Erlang Type System.
Web site `http://www.ericsson.se/cslab/~thomas/types.shtml`

[Tå00] Jan Tångring. Gå före i kön till webbplatsen. *Datateknik 3.0*, no 3, 2000.

[Vew] Verification of Erlang Programs.
Web site `http://www.sics.se/fdt/Erlang`

[Vi93] Robert Virding. Erlang. *FORTE - Sixth International Conference on Formal Description Techniques.* Boston, October 26-29, 1993.

[Vi95] Robert Virding. A Garbage Collector for the Concurrent Real-Time Language Erlang. *International Workshop on Memory Management.* Kinross, Scotland, September 27-29, 1995.

[Wa95a] Anders Wallerius. Ericsson ger upp framtidens telenät. *Ny Teknik/Teknisk Tidskrift*, no 50-52, 1995.

[Wa95b] Anders Wallerius. Programmeringen blev för svår. *Ny Teknik/Teknisk Tidskrift*, no 50-52, 1995.

[Wa98a] Jane Walerud. The Hidden Asset at CeBit. Erlang/OTP behind many Successes at the Hannover Trade Fair. *Erlang/OTP News*, CeBit Special, April, 1998.

[Wa98b] Jane Walerud. Professional Mobile Radio over GSM. The Ninth Erlang based System at CeBit. *Erlang/OTP News*, May, 1998.

[Wa98c] Jane Walerud. From Idea to Reality in Six Months. *Erlang/OTP News*, September, 1998.

[Wa98d] Philip Wadler. Why No One Uses Functional Languages. *SIGPLAN Notices - Functional Programming Column.* August 1998, pp 23-27.

[Wa99] Philip Wadler. The Next 700 Markup Languages. *Second Conference on Domain-Specific Languages.* Invited paper. Austin, October 3-5, 1999.

[We81] Richard L. Wexelblat, editor. History of Programming Languages. Academic Press Inc. 1981, ISBN 0-12-745040-8.

[We83] Carl Wilhelm Welin. The Frames System. LME/XT/DU 83 159, 1983, Internal paper.

[We95] Bruce F. Webster. Pitfalls of Object-Oriented Development. M&T Books, 1995.

[Whp] Open-source Erlang - White Paper.
Web site `http://www.erlang.org/white_paper.html`

[Wi76] Niklaus Wirth. Modula: A Language for Modular Multiprogramming. Eidgenössische Technische Hochschule. Zürich, 1976.

[Wi81] Patrick H. Winston and Berthold K. P. Horn. LISP. Addison-Wesley Publishing Company. 1981.

[Wi87] Åke Wikström. Functional Programming Using Standard ML. Prentice-Hall, 1987.

[Wi92] Claes Wikström. Processing ASN.1 Specifications in a Declarative Language. *Software Engineering for Telecommunication Systems and Services.* Florence, March 30-April 1, 1992.

[Wi94] Claes Wikström. Distributed Programming in Erlang. *First International Symposium on Parallel Symbolic Computation.* Linz, September 26-28, 1994.

[Wi95] Martin Wikborg. Comparing Erlang and SDL/SDT for Software Development. UU/DoCS, Master's Thesis, 1995.

[Wi96] Claes Wikström. Implementing Distributed Real-time Control Systems in a Functional Language. *IEEE Workshop on Parallel and Distributed Real-Time Systems.* Honolulu, April 15-16, 1996.

[Wi98a] Ulf Wiger. Ericsson ATM Switch AXD 301 - A New Way to Design Systems. *Erlang/OTP News*, April, 1998.

[Wi98b] Mike Williams. Erlang/OTP Economics. ETX/DN/S-98:353, May 25, 1989. Internal paper.

[Wo98] Geoff Wong. Continuous System Monitoring. Ph.D. Thesis under way. RMIT, November 1998.

[Wre] Wind River Systems Erlang Home Page.
Web site `http://www.wrs.com/products/html/erlang.html`

[Wrw] Wind River Systems. Web site `http://www.wrs.com`

[Xmw] The XML Industry Portal. Web site `http://www.xml.org`

[Yo92] Edward Yourdon. Decline and Fall of the American Programmer. Yourdon Press, PTR Prentice Hall, 1992.

[Öd93] Kerstin Ödling. New Technology for Prototyping New Services. *Ericsson Review*, no 2, 1993.

Appendix 1: Master's Theses and Students' Projects

The following is a list of theses and projects which have either contributed to the Erlang system or utilized it to implement applications and systems.

- P-A Eriksson and J Tjernlund. Erlang och realtidskontrollerad järnväg. Tekniska Högskolan, Luleå, 1990.
- Martin Sköld. Distributed Real Time Databases. LiTH/IDA, 1990.
- Adam Aquilon. Automatic Code Generation from Sequence Charts. KTH/EIT, 1991.
- Rudolf Hersén. Sequence Chart Editor. KTH/EIT, 1991.
- Jörgen Bergstedt and Thomas Persson. Intelligent Network. Tekniska Högskolan, Luleå, 1991.
- Distorsion. Student project with 17 participants. UU/DoCS, 1991. Also as [Di91].
- Lennart Öhman. Framprovocering av fel i programkod. UU/DoCS, 1992.
- Anders Dahlin and Peter Jansson. SUN Controlled Telephone. UU/DoCS, 1992.
- Klas Mikaelsson and Henrik Forsgren. Demonstrationssystem för telefoni implementerat i Erlang. UU/DoCS, 1992.
- Patric Jansson and Björn Axelsson. Direktledningssignalsystem för en MD 110 växel. UU/DoCS, 1992.
- Björn Bergqvist. Testmiljö för accessignalering. KTH/EIT, 1992.
- Joakim Grebenö and Niklas Hanberger. An Object Oriented Call Model (OOCM). UU/DoCS, 1993.
- Magnus Höglund. Distributed Telephony with Erlang. LiTH/IDA, 1993.
- Li Wei. Gateway between Packet and Switched Networks for Speech Communication. KTH/EIT, 1994.
- Lars Björup. The Connection Model Implemented in Erlang. LiTH/IDA, 1994.
- Johan Thureson. Q.93B Test Tool. KTH/EIT, 1994.
- Beshar Zudhy. Erlang Port to the Parsytec MIMD Parallel Platform. LiTH/IDA, 1994.
- Andreas Ermedahl. Discrete Event Simulation in Erlang. UU/CSD, 1994.
- Ali Imitiaz Shah. Design and Implementation of ET-155 Device Processor Software. KTH/EIT, 1994.
- Jan-Erik Thomasson. Hantering av telefonisystem med hjälp av Erlang. KTH/NADA, 1994.

- Tobias Lindgren. An Erlang Interface to SQL. LiTH/IDA, 1994.
- Anders Frank and Ola Samuelsson. A Graphical User Interface for Erlang. UU/CSD, 1994.
- Sim94 - A Concurrent Simulator for Plan-driven Troops. Student project with about 27 participants. UU/UPMAIL Technical Report 98, February 15, 1995. ISSN 0283-359X.
- Martin Björklund and Klas Eriksson. A Framework for SNMPv2 in Erlang. KTH/NADA, 1995. Also as [Bj95].
- Kent Engström. Parallel Erlang. LiTH/IDA, 1995.
- Samuel Tronje. Process-based Simulation of Interactive Agents in a Dynamic Terrain. UU/CSD, 1995.
- Greger Ottosson. An Extension of Erlang with Finite Domain Constraints. UU/CSD, 1995.
- Martin Wikborg. Comparing Erlang and SDL/SDT for Software Development. UU/DoCS, 1995. Also as [Wi95].
- Johan Agat and Lennart Dahlström. En undersökning av två deklarativa programspråk. CTH/IDV, 1995.
- Tomas Aronsson and Johan Grafström. A Comparison between Erlang and C++ for Implementation of Telecom Applications. LiTH/IDA, 1995.
- Plan95: A Distributed Planning System. UU/UPMAIL Technical Report 122, 1996.
- Niklas Kaltea. A Specification Language for Intelligent Agents. UU/CSD, 1996.
- Johan Carleson. Industriella Erfarenheter av Erlang. LiTH/IDA, 1996.
- Jian-Liang Cai. Implementation of an Object-Oriented DBMS Using the Erlang Programming Language. RMIT, 1996.
- Kristina Sirhuber. YERL - A Literate Documenting Tool and a Program Development Environment for Erlang. UU/DoCS, 1996.
- Peter Molin and Fredrik Ström. A GUI Builder for Erlang/GS. UU/CSD, 1996.
- Anders Lindgren. A Prototype of a Soft Type System for Erlang. UU/CSD, 1996.
- Erik Johansson and Christer Jonsson. Native Code Compilation for Erlang. UU/CSD, 1996.
- Babbis Xagorarakis. Java RMI Interface to Erlang, Implementation och Utvärdering. UU/CSD, 1997.
- Richard Carlsson. Towards a Deadlock Analysis for Erlang Programs. UU/CSD, 1997.

- Gustaf Naeser. Safe Erlang. UU/CSD, 1997.
- Hans Danielsson and Kent Olsson. How to Measure Reliability in an Erlang System. LTH/DCS, 1998.
- Pekka Hedqvist. A Parallel and Multi-threaded Erlang Implementation. UU/CSD, 1998. Also as [He98].
- Ronny Andersson. SQL Compiler For the Mnesia DBMS. CTH, 1998.
- Johanna Isaksson and Elinor Sturesson. Design Guidelines for Erlang. CTH and RMIT, 1999.
- Hans Danielsson and Kent Olsson. How to Measure Reliability in an Erlang System. LTH and RMIT, 1999.
- Clara Benac-Earle. Symbolic Program Execution using the Erlang Verification Tool. UU/CSD, 2000.
- Raimo Niskanen. Integration of Erlang and TelORB. KTH/IT, 2000.
- Rickard Green. Enhancing Security in Distributed Erlang by Access Control. KTH/IT, 2000.
- Bertil Karlsson. Secure Distributed Communication in SafeErlang. KTH/IT, 2000.
- Peter Andersson and Markus Kvisth. A General Protocol Stack Interface in Erlang. UU/CSD, 2000.

Appendix 2: User Conference, September 30, 1999

Papers presented at the *Fifth International Erlang/OTP User Conference.* Web site `http://www.erlang.se/euc/99`

- Hans Nahringbauer, Telia Promotor AB. Telia Call Guide. Also as [Na99].
- Fredrik Ström, Brainpool AB. Use of Erlang/OTP in the Brainpool M/3 Communication System. Also as [St99b].
- Marc Feeley, Patrick Piché, Sylvain Beaulieu, Martin Larosse, and Mario Latendresse, Université de Montréal. Status Report on the ETOS Erlang to Scheme Compiler. Also as [Fe99].
- Håkan Millroth, Bluetail AB. Mail Robustifier Product based on Erlang/OTP.
- Per Bergqvist, Ericsson Radio AB. Hatchet.
- Johan Blom, Ericsson Wireless Internet AB. A Modular WAP Reference Stack Protocol Implementation.
- Hans Nilsson, Ericsson Utvecklings AB. An Experimental SIP Implementation in Erlang.
- Magnus Karlson, Ericsson Utvecklings AB. Coming Releases of Erlang/OTP. Also as [Ka99].
- Maurice Castro, SERC. Towards an Event Modelling Language.
- Tony Rogvall and Claes Wikström, Bluetail AB. Protocol Programming in Erlang using Binaries. Also as [Ro99].

Appendix 3: User Conference, October 3, 2000

Papers and demos presented at the *Sixth International Erlang/OTP User Conference*. Web site `http://www.erlang.se/euc/00`

- Sean Hinde, one2one. Use of Erlang/OTP as a Service Creation Tool for IN Services. Also as [Hi00].
- Scott Lystig Fritchie, Jim Larson, Nick Christenson, Debi Jones, and Lennart Öhman. Sendmail Meets Erlang: Experiences Using Erlang for Email Applications. Also as [Fr00].
- Per Bergqvist, CellPoint. MPowered by Erlang.
- Bengt Tillman, Ericsson Radio Systems AB. NETSim - Six Years with Erlang.
- Mikael Pettersson, Uppsala University. A High Performance Erlang System.
- Robert Tjärnström and Peter Lundell, Ericsson Telecom. ECOMP - an Erlang Processor.
- Richard A. O'Keefe, Otago University. An Erlang DTD.
- Ulf Wiger, Ericsson Telecom. XMErl - Interfacing XML and Erlang.
- Mickaël Rémond, IDEALX. XML and Erlang: Building a Powerful Data Management Tool.
- Richard Carlsson, Uppsala University. Extending Erlang with structured Module Packages.
- Kenneth Lundin, OTP Product Unit. Highlights from Erlang 5.0 / OTP R7B.
- Jakob Cederlund, OTP Product Unit. COMET - An Erlang-to-COM Port.
- Patrik Nyblom, OTP Product Unit. The Bit Syntax - The Released Version. Also as [Ny00].
- Lars-Åke Fredlund, SICS. A Tool for Verifying Software Written in Erlang.
- Miguel Barreiro, Victor M. Gulias, and Juan J. Sanchez, Universidade da Coruña. A Monitoring and Instrumentation Tool Developed in Erlang.