

The migration from Erlang to OTP: A case study of a heavy duty TCP/IP Client Server application.

Francesco Cesarini¹
Mickaël Rémond²

September 27, 2001

Proceedings for the Seventh International Erlang User Conference, Stockholm, Sweden.

Abstract

A team of software engineers at IDEALX set out to build a proxy for a heavy duty TCP-IP application. They had Erlang experience from previous in house projects, but had never come in contact with the Open Telecom Platform's design principles. When an external Erlang/OTP consultant was taken in to review the code, he discovered a system written in pure Erlang. This paper describes reasons why development on OTP design principles were developed in the first place, and explains why they were bypassed in the prototype phase of this project. It concludes by describing how the migration of the Erlang prototype to an OTP product was achieved, looking at the advantages gained through such a migration. The intended readers are companies and individuals considering using Erlang in product development, but do not have access to in house Erlang expertise.

Introduction

IDEALX's business idea is to run in-house software development projects on behalf of customers using open source tools and components. One of these projects consisted in developing an instant messaging server where different protocols were interfaced towards the Jabber protocol, used by the ISP who commissioned the product. Due to the concurrent nature of the system, time scales, fault tolerance, high availability, and scalability, Erlang was chosen to develop the core of the system. The team had previous Erlang knowledge from in-house projects, and unanimously agreed that it was the best route to take. Inspired by the Sendmail³ presentation at the Sixth Erlang User Conference, the project decided to take in an Erlang/OTP consultant. His task was to review the code, review the system architecture, and provide OTP training. When he started reviewing the code, he discovered a working prototype developed in pure Erlang. It did not use OTP design rules, principles and applications.

The team had been looking for something similar to OTP, but as they did not exactly know what they were looking for, they did not know where to look. Only when they took in external Erlang/OTP help were a lot of their questions answered. When the

¹ francesco@erlang-consulting.com, www.erlang-consulting.com

² mickael.remond@erlang-fr.org, www.erlang-fr.org

³ Christenson et. Al. Sendmail Meets Erlang: Experiences using Erlang for email applications. Proceedings to the Sixth International Erlang User Conference, October 3, 2000.

migration of the system from Erlang to OTP took place, the size of the code was drastically reduced and extra functionality included in OTP was gained. We conclude the paper by sharing some of our ideas and needs to encourage other projects without experienced consultants or mentors to start using at least a subset of the OTP design principles from the start.

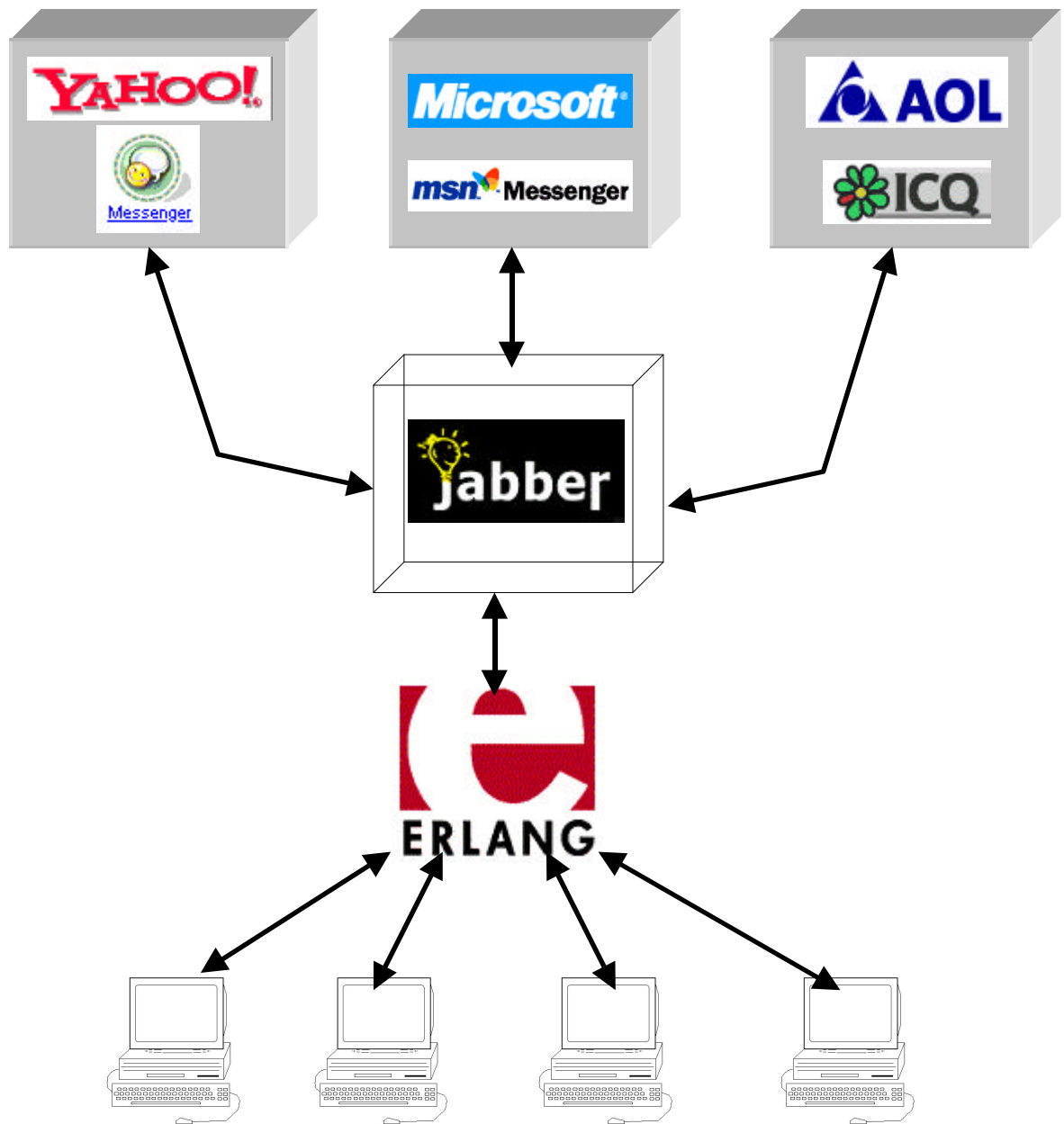


Diagram 1: A high level diagram of the product.

Product Requirements

A French internet service provider commissioned an Instant Messaging Proxy server based on the Jabber protocol⁴. Jabber is an XML-based open source initiative aiming at producing an open protocol alongside a gateway to existing competing instant messaging systems such as AOL, Microsoft, and Yahoo! At the time, Jabber could only handle fifty to one hundred simultaneous connections. The requirements from the ISP stated that the proxy needed to handle at least ten thousand simultaneous users. Scalability during run time was another requirement, as was the possibility of flexible software which at a later date could easily allow the addition of services such as advertisement messages to subsets of users.

Why OTP?

The first major product developed in Erlang was the Mobility Server. It was based on extensive prototypes started in early 1991. When working on the first prototype, the software engineers realised that similar problems had been solved differently in the various subsystems. Many models of client / server solutions had been implemented. Processes had different start and restart strategies, and error handling was not standardised.

A few people realised that a common approach, originally limited to start and recovery, was needed. In late 1993, alongside the Mobility Server project, development on BOS, the Basic Operating System, commenced. It introduced behaviours, restart strategies, release handling and code upgrade principles, alongside a set of design rules and guidelines that made processes exhibit a uniform behaviour towards the Erlang virtual machine. In 1995, the development of BOS and Erlang was combined, giving birth to the Open Telecom Platform. Prior to the first release of OTP, BOS was an obvious choice in all Erlang products developed within Ericsson.

Companies using Erlang/OTP have achieved many commercial successes. These successes are in part due to the in house knowledge and experience that has been built through many years of using the technology. Ericsson has been running Erlang/OTP projects since 1991, and has a department specialised in Erlang/OTP training and consulting. They have a direct communication and support channel to the Erlang/OTP maintainers. Nortel Networks employs many of the software engineers who originally invented Erlang, alongside members of the team that implemented the first release of OTP. Other companies, especially in Sweden, have people who have previous experience from Erlang/OTP based projects. They know that OTP should be used from the start. What happens when people come across Erlang through the web or through word of mouth and learn it from the documentation available on line and through the web, but are not made aware of the opportunities of OTP? What if they then go ahead and use it in a project without having the necessary mentorship? That is what happened at IDEALX.

IDEALX's prototype had many similarities to the early Erlang prototypes developed in the early 1990s. No coherent supervision structure existed. Some subsystems had restart strategies that were able to handle process crashes, others did not. Message

⁴ www.jabber.org and www.jabber.com

passing was used in an uncontrolled fashion, making debugging very hard. In addition, the team had made design decisions based on incorrect assumptions over how the virtual machine and the kernel functioned. These decisions resulted in bottlenecks, inefficient constructs, and bad use of concurrency.

Why OTP was bypassed?

In December 1998, when Erlang was released as an Open Source development environment, Mickaël Rémond, the project leader and co-author of this paper, discovered the language. He found it very appealing because it offered many interesting features that were new to him.

He had not developed many concurrent applications because it meant having to add a magnitude of complexity to the code, drastically changing the programming paradigm he was used to. The languages he had used prior to Erlang were not made to handle concurrency, making the implementation of multithreaded systems appear unnatural and awkward.

What he found interesting about Erlang was that using the language changed and improved his programming style. Concurrency became straightforward and natural. He further realized that mapping processes to truly concurrent activities in the real world meant simplifying and improving the application design. It became easier to think in terms of concurrent process than, for example, in terms of objects.

But, as he came from an object-oriented background, there was still something that he missed. Object orientation is not only used to design the system. It is also a way to organise and reuse large part of the source code. How was it possible to implement an efficient and well-structured program without object orientation?

OTP was certainly not the answer that came to mind. When looking at other functional languages, he discovered that some of them offer a mixed of orthogonal approaches. For example, Objective CAML is a functional object oriented language. Erlang had no such obvious approach.

This led Mickaël Rémond to ask the classical newbie question in response to a mail sent to the Erlang mailing list. *"How do you apply object oriented programming techniques in Erlang?"* It seemed the natural question to ask, and only later did he realise it was the wrong one. Fortunately, Joe Armstrong came to the rescue. He gave him a very important clue in explaining that you can apply a typical Object Oriented Approach in Erlang with a common pattern. An example can be found in Chris Rathman's Shapes OO example⁵. Joe Armstrong however added that object orientation was not needed because Erlang has something more powerful called behaviours.

"Something more powerful than object orientation?" It sounded too good to be true. It took some time before Mickaël Rémond really understood what Joe Armstrong meant. Object Orientation is not necessarily the best way to design and structure a piece of code. It might appear easier to learn and use, but when applied to larger

⁵ www.angelfire.com/tx4/cus/shapes/erlang.html

system, the theory does not work. Users are forced to add "technical" objects that often result in complicated hierarchies. This makes the system architecture complex and sometimes affects the performance of the application.

It was however difficult for him to understand how to use the behaviour alternatives. The design guidelines are not very well promoted, and you have to search deep into the documentation to find examples. Such an approach is obvious when working with proper Erlang mentorship, but it certainly was not obvious for someone who had downloaded Erlang/OTP soon after it was released as open source.

OTP Deterrents

Behaviours are part of the design principles in the Open Telecom Platform. Saying that, you are facing your first deterrent to dig deeper into behaviours. Unless its name implies it, the Open Telecom Platform was not only made to develop telecom applications. It is a much more general tool. In fact, OTP is useful to organise any big enough Erlang application.

The second problem is that behaviour documentation is not where you would expect it would be, and is more a reference document than a step-by-step tutorial. To find the behaviour documentation, you have to look into "system principles" and "design principles", and in the correct section of the STDLIB documentation. The new organisation of the documentation in R8 is clearer, but still is not sufficient.

We also need to emphasise how behaviours relate to each other and how they can be used to make your application more robust. Maybe the tutorial should illustrate the way to migrate a pure Erlang program into an Erlang/OTP application. A step-by-step explanation, applied to a real application design. A widely available advanced Erlang programming book, focused on practical OTP aspects would also be welcome.

In addition, more Open Source code for applications designed using behaviours are needed. When searching on the net, most examples are written in pure Erlang. It was hard to find good examples of applications designed in OTP. In some cases, not even OTP applications in the Open Source release follow their own design principles.

IDEALX's Erlang team did not really know how they could build a project using OTP's design principles, and thus, it was not the obvious choice. They decided to work on a prototype in pure Erlang, and only later realised their mistake and asked an Erlang/OTP consultant to review their work.

The Migration

Francesco Cesarini was brought in to provide technical consulting and OTP training. His time on location with the team was limited to one week, and that had to include all OTP training. That was a solution he tried to persuade IDEALX not to go ahead with. Often, when trying to cut training costs, the results are catastrophic. OTP design principles are complex, so people not experienced in Erlang often have a hard time grasping the subject. The group, however, consisted of a set of experienced programmers with a computer science background who had encountered many different programming paradigms. They had programmed extensively in Erlang, both

on in house projects and on the Instant Messaging proxy server. By stripping the course material to the bare essentials, they were able to cover design patterns, programming rules and the new bit syntax in two and a half days, exercises included⁶.

As soon as the team had a good understanding of how call-back functions worked and felt comfortable with the OTP design principles, it was possible to review their code in regards to style and efficiency. The review allowed the team to explain the architecture of their prototype, and their ideas and misconceptions on which they based their design decisions. A misconception on how the virtual machine worked resulted in a huge bottlenecks in the encoding and decoding processes. Believing that each process saved a private copy of the byte code, they wanted to reduce memory usage at the expense of message passing.

This was followed by putting together a new system architecture with emphasis on the process and application structures. This task only took a few hours because of the preparatory work during the review. The result was a simpler architecture, as the migration helped reduce the complexity while keeping the overall modularity of the system. Prior to the rewrite, the code was hard to follow. The use of the standard design patterns, namely generic servers, finite state machines and supervisors helped improve readability and structure.

The original prototype did not support distribution and load balancing. The OTP migration, however, facilitated designing those aspects in the system. The restart strategy came as a bonus, as it was achieved through the use of supervisors. Due to the time scale of the project, the team did not have time to introduce release handling and hot code upgrade. They did not feel confident with the upgrade, and for political reasons preferred stopping and restarting the system instead of risking a crash due to some error.

The overall performance⁷ of the system was greatly improved after the rewrite. The unnecessary concurrency had been removed resulting in a drastic reduction in the overall exchange of data among processes. The code review took into consideration efficient constructs, so the use of binaries and pattern matching on the bit syntax, also played a big role. It was an easy task for the team to migrate from a pure Erlang application to one using OTP design principles. The rewrite of the prototype into a real product took 20 days, and resulted in the reduction of the code size by approximately 50%. The original prototype had taken 25 days to develop.

Conclusion

The team that worked on the project would today never consider writing an Erlang application without using OTP from the start. It took them only a couple of days to become productive, and they immediately saw both the need and the advantages. But in order to understand behaviours well, they had to first grasp how to design good

⁶ It has to be pointed out that the training session was successful only because the participant's background. As the consultant had to be on location for a whole week, he agreed to the intensive course only because he knew there would have been more time for questions and exercises should the allocated time not have been enough.

⁷ No performance measurements were made prior to the rewrite.

Erlang applications. Attempting to learn OTP the week after having attended the Erlang course does not work.

Many advantages were achieved in the product after the migration. There were no more message management errors in that all message passing was handled through functional interfaces. Process supervision and restart strategies were standardised. Processes exhibited the same uniform behaviour and had basic event tracing and logging functionality. It was possible to generically control the whole system, distributing it across several nodes and processors. Even if not used in this release of the product, they had the existing tools for release handling, could easily integrate other OTP applications, and had a platform easily allowing the implementation of software upgrade during run time.

By creating BOS and later OTP, a gap that existed with pure Erlang was filled. It is a gap dealing with methodology, structuring of programs and re-usage of code. This was the gap that Mickaël Rémond was trying to fill when he came across Erlang in 1998 and asked about OO methodology.

Unfortunately, the Erlang book came out before OTP. The documentation is well hidden, and few examples of well-structured OTP applications are available. While waiting for new books, new documentation, and more open source examples, we hope that this paper will lead to a better understanding of the power of OTP to companies and people who do not have access to Erlang/OTP expertise but are considering Erlang.