

From HTTP to HTML – Erlang/OTP Experiences in Web Based Service Applications

Francesco Cesarini, Lukas Larsson, Michał Ślaski
Erlang Training and Consulting Ltd
401 London Fruit and Wool Exchange
Brushfield Street
London, E1 6EL
United Kingdom
{francesco, lukas, michal}@erlang-consulting.com

Abstract

This paper describes the lessons learnt when internally developing web applications in Erlang. On the basis of these experiences, a framework called the Web Platform has been implemented. The Web Platform follows a design pattern separating data processing and formatting, allowing the construction of flexible and maintainable software architectures. It also delivers mechanisms for building dynamic pages and components. On top of the platform and components, web interfaces to commercial Erlang systems have been built.

Categories and Subject Descriptors D.3.3 [Programming Languages]: Frameworks

General Terms Design, Reliability.

Keywords Erlang, HTTP, HTML, Web Frameworks

1. Introduction

Erlang was developed in the late 1980's by Ericsson to program the next generation of telecom applications. Telecom applications are distributed, fault-tolerant, massively concurrent soft real-time systems with high availability requirements. On a software level, these requirements are similar to those put upon Internet based applications, making Erlang a good candidate for development of web based services.

A typical web application will consist of an Apache[11] web server and a MySQL[12] database back-end, glued together with Perl[7] and PHP[4]. These components need to be independently managed and supported, and interfaces and data formats between them have to be defined. Erlang, on the other hand, will run all these components within the same platform and technology concept, allowing them to share the same memory space.

This facilitates support of these systems, which together with the built-in ability to upgrade software during runtime can provide systems with no downtime. Using built-in fault tolerance mechanisms ensures that errors are not only isolated, but also uniformly handled. Furthermore, what makes Erlang special in developing web applications is its extremely powerful concurrency

model, allowing a constant throughput of requests under extremely heavy loads[1]. Joe Armstrong showed in one of his presentations[8] that in a denial of service attack, the number of parallel connections needed to crash the Erlang web server was about 20 times as many as an Apache web server running on the same hardware.

Ericsson's computer science laboratory experimented with HTTP as early as 1994, writing an Erlang based proxy used to study the HTTP protocol[13]. This proxy evolved into a simple web server consisting of a couple hundred lines of code, and eventually became the first web server to be included in OTP. This server, often referred to as the INETS server, was presented at the third Erlang User Conference in 1997[3]. With 10k lines of Erlang, the development team had achieved 80% of the Apache server functionality, which at the time consisted of about 100k lines of code.

After presenting INETS to the general public, different platforms and frameworks have been developed to take advantage of Erlang's characteristics in web based contexts. One of the very early adopters, Eddie application[10], did what Erlang does best. By using distribution, a hybrid of machines and operating systems would join forces to create a server farm providing scalability, reliability and fault tolerance.

A web platform used for operation and maintenance of the AXD301 switch dynamically developed web pages through SNMP instrumentation functions and generic Erlang data structures[2]. It led the way to the first documented web platform based on INETS. Unfortunately, little information is available about this platform outside of Ericsson, and it has not been released as an open source.

The most powerful Erlang based web server around is Yaws[9]. It was released as an open source in 2002 and implements most of the features met in today's HTTP servers. What makes it really powerful, however, is its ability to execute dynamic pages with Erlang code.

Through OTP applications, Erlang has reached a high level of maturity in the telecoms market. But when compared to other "late majority"[6] languages for web services such as PHP and Perl, regardless of the above examples, Erlang is still considered to be in the stage of the "early adopters"[6] (see Figure 1). With the exception of the Erlang wiki¹, released in 1999, there have been few reusable components commonly needed for developing web services

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Erlang'06 September 16, 2006, Portland, Oregon, USA.
Copyright © 2006 ACM 1-59593-490-1/06/0009...\$5.00.

¹ Erlang wiki, <http://erlang.sics.se:5000/>

on an application level. Instead, they concentrate on what Erlang was originally intended for, namely the transport layer and protocols.

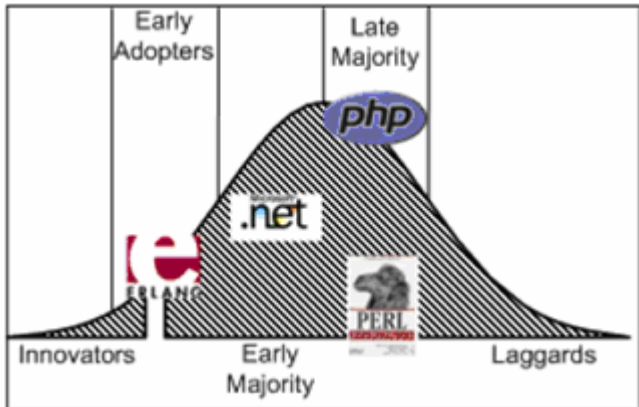


Figure 1. Rogers' Diffusion of Innovation Curve.

In this paper, we discuss the lessons learned when internally developing HTTP based Erlang services on the application layer at Erlang Training and Consulting. The results of these development and prototyping activities resulted in a generic web platform, tools, and components also described in the paper. This generic development had to be developed on top of an existing infrastructure to make Erlang a suitable language to develop the glue and logic and back-ends of web based applications, moving the language a little closer to the Early Majority group of users.

2. History of Erlang Training and Consulting

Until recently, Erlang Training and Consulting (ETC) did not have any in-house systems development organisation. This started to change gradually in 2004 through its summer internship program, where second-year students were put to work on prototypes and tools needed internally within the company. On a very busy schedule, the senior staff did not have the time to review their code and give them the necessary feedback on a system which had an Erlang web based interface. The result was a system with no clear separation of the data model and dynamic generation of HTML. Experienced developers will know that data, code and HTML generation have to be separated to avoid inflexible, unsupportable code. Indeed, the prototype worked well, but any modification or extension in the data model or change in the HTML format, no matter how small, required a tedious rewrite of the code.

From one extreme to another, the second prototyping attempt resulted in the adoption of XSLT. XSLT is an XML based language which describes XML transformations, in our case, from XML to XHTML. Initially, XSLT with its generic way of transforming and formatting XML files seemed to ideally fit our requirements. Several pages presenting dynamic data were developed, but with time, the transformation turned out to be too formal and too strict. Merging it with other parts of the web page was inconvenient, and further development became an uneasy task. The largest problem resulted in very complex data transformations with multiple data sources. This, in turn, resulted in slow execution speed and the need of strict HTML, something not even the most popular HTML authoring tools produced. Our graphics designer was developing HTML that had to be passed through a linter and manually corrected. And to

further convince us that this was not the way to go, the libxslt driver which we were using had memory leaks and no internal debug features. We quickly realised we had reached a dead end.

This is when we started developing our own web platform which would fit our requirements for a scalable architecture with dynamic content. The framework was started off as a prototype and tested internally on the erlang-consulting.com home page. Only after the successful migration of the site and extensive stress testing was the platform refactored and included in other commercial projects. Even though the commercial solutions had major time constraints and critical time to market requirements, Erlang, with its rapid development capability, allowed us to refactor the code and achieve the functionality and stability required by commercial systems.

3. The Web Platform

The Web Platform was thought as a framework for applications based on HTTP protocols, relieving the developer from implementing behaviours frequently used in dynamic pages. With its simple but extensible concept of including dynamic content in pages, libraries of reusable components can be built. It is based on the Yaws HTTP server giving high performance and throughput.

Yaws has a mechanism for generating dynamic pages that can be developed as a mix of Erlang and HTML code. The Erlang chunks should contain the *out/1* function, which is compiled during processing of a Yaws page and replaced with the generated output. The result of compilation is saved for the future, so it can be reused when a request to the same Yaws page occurs.

The Web Platform introduces a different approach to generating dynamic pages. It allows merging XHTML code with a dedicated XML tag called *wpart*. With the concept of *wparts*, it is possible to develop pieces of functionality that can be reused in different pages, providing solid framework for developing dynamic web services. A set of frequently used patterns has been implemented as a common library of *wparts*. They include:

- retrieving data
- formatting date, time and numbers
- iterating over a list of Erlang terms
- branching constructions

Having *wparts* defined as XML elements allows the Web Platform to validate a page with an XML parser ensuring that what is sent to the client browser is free of XHTML errors. A parsed file is converted to Erlang binary chunks kept in a Mnesia table. Since both the Web Platform and Mnesia applications are running within the same memory space, access to the structure with chunks of code is quick, as it also functions as a cache.

An important principle is the separation of data processing and formatting[5]. The *wpart* syntax forces a strict separation of the model and the view in applications. This allows web designers to focus on creating HTML markup, while the Erlang developers can focus on the back-end of the application. The web developers are able to use any WYSIWYG² HTML editor as the *wparts* will be interpreted as unknown elements. The *wparts* will also hide the Erlang code, which would be seen in Yaws pages, from the web

² What You See Is What You Get

developers. In the meantime Erlang developers will not need to edit any HTML markup.

To allow data to flow between wparts, the Web Platform provides a request dictionary. Yaws creates a process for each HTTP request and the Web Platform implements the request dictionary by creating a data store which is global for each HTTP request. In general, algorithms based on global variables are hard to debug and to follow, but at the same time it allows wparts to communicate with each other in a convenient manner. The data storage is released as soon as the request has been handled, and the process is terminated.

Wparts allow the usage of HTML templates much in the same way as Yaws uses Server Side Includes, but with wparts you do not have to separate related templates into different files. In this way templates used to format data of the same kind can be categorized.

4. Erlang Training and Consulting's home page

During the development of ETC's home page four requirements were identified. These are applicable to any web application.

- Clear separation of data, logic and presentation
- Modularity
- Common set of templates
- Common data storage

The first version of the ETC home page was not an Erlang solution. Unfortunately, web hosts do not provide Erlang installations. The first version was implemented using the "traditional" Apache web server, Perl and PHP scripting languages. Data used to generate dynamic pages was stored in a MySQL database. Every time a new feature had to be added, the whole application became increasingly more complex and the data flow became harder to understand. As the complexity of the home page grew, reconstruction of the system was necessary. At that time the Web Platform concept was formulated and a new version of the home page was designed. Using the Web Platform and following the requirements specified above resulted in a more consistent architecture than the original PHP solution, and easier to maintain.

The Web Platform deals with the complexity by dividing the web site into static and dynamic content. Static content can be updated and changed during runtime but does not need to be processed in any specific way by a server. This includes all PDF documents, images, some CSS and HTML files. Such content is sent to a client using the transport layer of the Yaws server. Dynamic content is generated as a result of executing a function over data kept in a Mnesia table. The function takes GET and POST data, which together with wpart element attributes determine what kind of transformation should be applied.

Sets of functions generating dynamic content are developed as components. A component delivers functions to transform data of the same kind. As an example we can describe Erlang events that are short announcements about speeches, meetings or conferences potentially interesting for the Erlang community (see Figure 2).

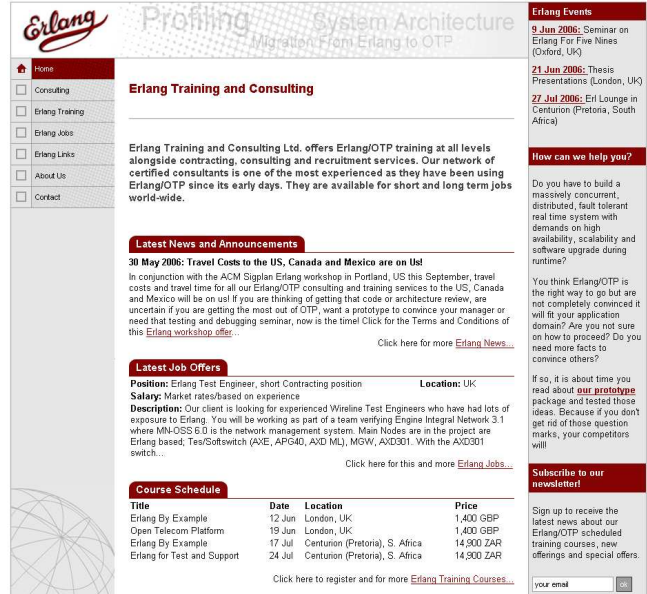


Figure 2. Home page with Erlang events component.

The event component consists of:

- A Mnesia table
- Two Erlang modules
- XHTML file with wparts

In the table, data like date, location and description is stored. The modules implement transformation functions and a call-back function which is used to hook transformations to a web page. To make a decision about which transformation to execute, URL of the client request or a GET/POST variable can be used. The XHTML file is split into parts. Every part is used to present a different aspect of Erlang events or to present the same aspect in different ways. For instance, we can list all events or only a short summary about future events. A form, possibly password protected, allows a user to create or edit events.

To ensure a consistent appearance in a web site, reused elements are stored as generic templates (see Figure 3). This avoids the maintenance issues raised by cut and paste programming. Templates are implemented as a set of dynamic pages that can be included by a dedicated wpart element. Any data that should be formatted in a template is passed on with a request dictionary.

5. Web Components

One of the signs of maturity within any engineering discipline is the standardization of tools and components used during construction and the availability of third party components, both open source and commercial. Two of the most well known web development languages today are PHP and Perl. One of the advantages they have over Erlang is the community behind them and the availability of reusable components.



Figure 3. A generic template reused for each estate.

Taking existing Perl and PHP based components off the shelf always results in compromises over functionality versus time to market. A component written in other programming languages might not have all of the desirable features listed in the requirement specification, and adding them has often resulted in a steep learning curve when having to understand the code. Owning the components will ensure that they do exactly what we required them to do, with the ability to adapt them to our needs and wants or the application.

Many of the components currently used were developed alongside the Web Platform, as it was constantly evolving with non backwards compatible interface changes. The components had to be made generic enough to cater for these changes, and were thus developed in two layers. One layer addresses the platform specific issues involved with the task that the components was designed for, and one layer which handles the generic features that could be reused throughout the development period. This ensured the components did not have to be rebuilt from scratch each time a new platform was released, future proofing our system for two-tier security architecture. The two software layer approach created reusable components, which if not requiring different format of the output, could be reused as they are. Two of the components created were the bulletin and the content manger. These components are currently being used in three websites, requiring very few, if any changes.

The first component developed was the bulletin. It was originally only used for news announcements, but has since been expanded to be used as both an event and a job bulletin. While the code implementing the first layer of the component has changed to cater for its different uses, the core remained the same.

As problems being solved required more complex data models, the bulletin component was not powerful enough. This resulted in a general content management component, which through indexing allows data to be accessed and sorted using different keys. The complexity of the component, however, comes at a price. At the time of writing, it is still cumbersome to work with it, as there is no general way of accessing its data. As the Web Platform matures, the ways in which data can be retrieved from this component will be addressed.

6. When Erlang Was Not Enough

The major hurdle we encountered when developing web based applications are in Mnesia's limitations. Our arguments on having a web server, glue and logic and database in the same memory space does not hold with complex user interfaces. When developing a graphical web front-end for a client, users were able to define their own entries in a generic table and given the possibility to sort HTML output by any of the fields, as well as executing queries with complex logic. To avoid complex code, the developers were forced to migrate from Mnesia and Query Lists Comprehensions to a relational database with SQL. Due to time constraints, investigating the open source RDBMS³ solution was not an option. RDBMS is a front-end to Mnesia developed by Ulf Wiger and Rudolph Van Graan which truly makes Mnesia relational. At the time of the project, it was in its early days of development, but results and stability recently achieved should make it a viable candidate.

A lack of dynamic records caused flexibility issues when front-end features were added or removed. The whole code had to be recompiled every time the record definition changed. Even if this is not a web-specific issue, dynamic record is a construct which would be much appreciated in Erlang, and welcomed by the community.

Other problems included the lack of strings as data types. The existing list implementation used to denote strings is highly inefficient and consumes too much memory, as it does not take advantage of the knowledge that it is storing an array of ASCII characters. Workarounds converting HTML to binaries for efficiency reasons, both memory and message passing wise, had to be made in the platform.

Lastly, the regular expression parser in Erlang is highly inefficient. If many regular expressions are used, an external parser has to be interfaced to the Erlang application, either through a port or a linked in driver. Having an efficient Erlang implementation would simplify the architecture and remove many of the issues associated with interfacing external programs.

7. Benchmarks

To evaluate the overall performance of the applications built on top of the Web Platform, we stress-tested the erlang-consulting.com web site. The web site is deployed on a server equipped with Intel(R) Pentium(R) 4 CPU 2.80GHz and 1 GB of RAM memory. The benchmark was executed using the Tsung⁴ load testing tool. We ran the benchmark on the same machine as the Yaws server in order to decrease the effect of network latency on the results. The results show the time needed to send a request to the server and to get the generated page (see Figure 4). The static pages tested

³ RDBMS, <http://ulf.wiger.net/rdbms/>

⁴ Tsung, <http://tsung.erlang-projects.org/>

included the index, consulting and about page. The dynamic pages included the news and jobs pages.

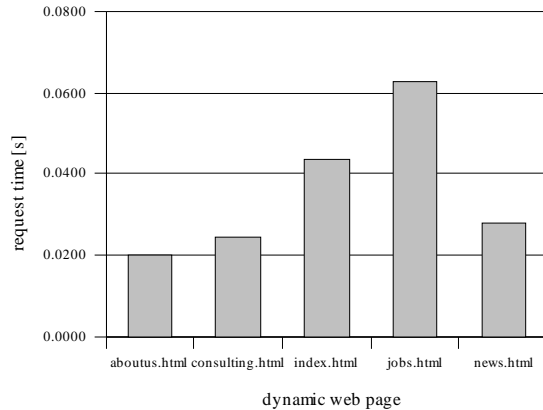


Figure 4. Request times for www.erlang-consulting.com

In reality, time needed by a web browser to contact the server and download a page will be different for every user based on network latency, browsers and CPU power, so presented figures should be viewed with caution. The fact that the application on which we were running the stress tests was running on the same machine will furthermore have affected the results. What is interesting and important in these results is the comparison between dynamic pages and static pages, and the comparison between dynamic pages that use different components. The latter allows us to evaluate the complexity and efficiency of developed modules. For instance, the index.html page is generated using three components, while the jobs.html page uses only one. The number of components used in a dynamic page does not affect the performance, but the way the components are built and used has. In the jobs page, for example, sorting and filtering of the jobs results in the higher request time than the news page, where a similar amount of data is just displayed in chronological order. Also of interest is the fact that for simple dynamic pages, the results are very close to the static ones.

8. Conclusions

The Erlang web servers and databases are mature, as is the use of Erlang for glue and logic. What is lacking while developing interfaces for operation and maintenance of network systems built on Erlang/OTP, is a general framework for developing web based components, and the components themselves. It means a solid web development framework is needed which would reduce time and efforts spent on implementing components that follow similar design patterns.

At ETC the Web Platform framework was developed. The Web Platform delivers a generic concept of wparts used for implementing reusable web components. The next step is to identify generic web components and implement them using the Web Platform framework.

The learning curve and entry costs to develop the tools and components have been high. Now that we have reached this level of maturity within our organisation, we have the competitive advantage Erlang/OTP provides over the classic web application languages in the early and late majority adopters. The same can however be argued over Telecom based Erlang projects in 1993, when many of

the existing predecessors of the OTP applications were being implemented in parallel, or did not exist at all.

9. Future Work

The Web Platform is still in its early days, and still has a long way to go. Planned future work includes creating more generic components such as billing, blog, user registration, shopping carts and credit card/paypal components.

In the platform itself, a type checker parsing incoming data, both in terms of data types and ranges, is next in line to be implemented. A syntax notation is being developed at the time of writing which can include type checks on data types, ranges, dates, and predefined lists. All type checking will be done based on strings, and conversion to atoms will only be allowed on sets of predefined attributes. If the element (in string format) is not found, the transformation will not take place⁵.

The infrastructure to allow a two-tier security architecture, where the front-end machine runs the web servers, parses the data and generates the HTML dynamic code, while the back-end machines handle the transactions and host the databases is in place, but still has not been implemented. The web components developed so far are ready for this migration thanks to the two-tier module approach. An RPC based API, which uses sockets to communicate with other nodes has already been implemented, and will be used as soon as there is a requirement for such a level of security.

Finally, a Master's Thesis is being planned to compare our web platform with Ruby On Rails. It will give us a better understanding of similarities and differences between the two solutions, and give us guidelines on what was done right, what could have been done better, and what has been missed. The results of the thesis will hopefully give assurances that Erlang, development of web-based applications, is slowly moving from the group of Early Adopters into the group of Early Majority users.

Acknowledgements

Special thanks go to Martin Carlson for giving us the first version of the Web Platform. Without him, this paper would not have seen the light. Jan-Henry Nyström also deserves a special mention, as his feedback in the early versions of this paper helped coin the final result.

References

- [1] Joe Armstrong, *Concurrency Oriented Programming in Erlang*, Swedish Institute of Computer Science, 2003.
- [2] Francesco Cesarini, *Interfacing Erlang with Standardized Management Protocols*, Uppsala University, 2001.
- [3] Ericsson, *Proceedings from the Third International Erlang User Conference*, 1997.
- [4] Converse, T. and Park, J. 2000 *PHP 4 Bible*. 1st. John Wiley & Sons, Inc.
- [5] Terence Parr, Enforcing strict model-view separation in template engines. In *Proceedings of the 13th international Conference on World Wide Web* (New York, NY, USA, May

⁵ This will avoid denial of service attacks, where atoms are generated but never garbage collected.

- 17 - 20, 2004). WWW '04. ACM Press, New York, NY, 224-233, 2004.
- [6] Everett M. Rogers, *Diffusion of Innovation*, 5th Ed, The Free Press, 2003.
 - [7] Larry Wall, Tom Christiansen and Jon Orwant, *Programming Perl 3rd Edition*, O'Reilly Media, July 14, 2000.
 - [8] Joe Armstrong, *Concurrency Oriented Programming in Erlang Presentation*, Lightweight Languages Workshop 2002, Cambridge, MA, USA, November 9 2002.
 - [9] Claes Wikström, Yaws, <http://yaws.hyber.org>
 - [10] Patrik Winroth et al., The Eddie Mission, <http://eddie.sourceforge.net>
 - [11] The Apache HTTP Server Project, <http://httpd.apache.org>
 - [12] MySQL, <http://www.mysql.com>
 - [13] Verbal quote from Torbjörn Törnkvist, former member of the Ericsson Computer Science Laboratory