

A metrics for Evaluating Internet Applications written in Erlang

Lukas Larsson

IT University of Göteborg

Forskningsgången 6

417 56

Göteborg

Sweden

Abstract

Erlang was first used for developing web based applications soon after it was invented. What is missing however, are standardized components which handle common web related activities such as news bulletins and mailing lists. To guide development of generic web components an object oriented metrics was translated to a concurrency orientated one. The metric was used to improve three web components by making them less complex, both externally and internally. As a result, the metrics only showed a marginal decrease in complexity, proving that it is of little help when creating small generic web components. It might however, help when developing more complex components in larger systems, but for the components used in this case study, the metric failed to deliver. Further study has to be done to determine whether the metric is can be used to measure complexity of larger Erlang components.

Keywords: erlang, web development, components, complexity.

Introduction

Erlang was developed by Ericsson in the late 1980's to program the next generation of telecom applications. Telecom application are distributed, fault-tolerant, massively concurrent soft real-time systems with high availability requirements [Nyström05]. Internet based applications are often faced with the same challenges[Kant00], making Erlang a good candidate for development of web based services.

The computer science laboratory experimented with an Erlang HTTP proxy server as early as 1994¹, but to the general public, the results were first presented at the 1997 Erlang User Conference[EUC97] in the form of the

¹Verbal quote from Torbjorn Tornkvist, who wrote an Erlang proxy to study HTTP.

INETS server. Several different Internet based platforms and frameworks have been developed to take advantage of Erlang's characteristics [Wikstrom06,Eddie06].

Eddie[Eddie06], one of the early adopters of Erlang in a HTTP environment, dealt with load balancing and fault tolerance. The AXD301 web platform, used for operation and maintenance, dynamically developed web pages through SNMP instrumentation functions and generic Erlang data structures[Cesarini01]. These and other frameworks, however, are either not available to the open public or only deal with the transport layer in the ISO OSI model. They do not provide reusable components commonly needed for providing web services on the application level such as mailing lists, shopping carts and user management.

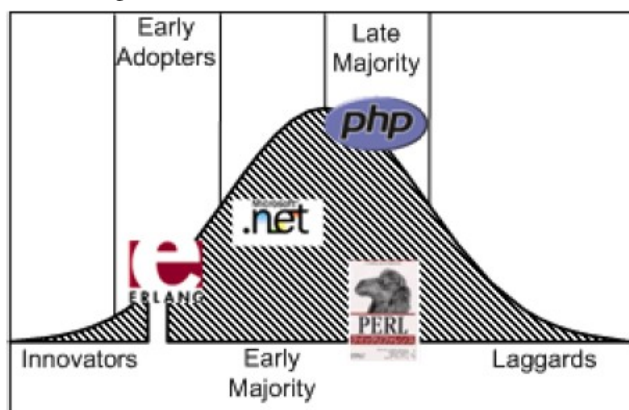


Illustration 1: Rogers' Diffusion of Innovation Curve in web service development.

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. The maturity level of a frameworks can also be measured by how wide spread it is. An innovation can be adopted by five different categories of users; Innovators, Early Adopters, Early Majority, Late Majority and the Laggards [Rogers03]. Widely accepted technologies would be placed in the Late Majority or even with the Laggards while new technologies would be classified as being in early stages of the curve.

Through the OTP Library, Erlang has reached a high level of maturity in the telecoms market. But when compared to “late majority” languages for web services such as php[php06] and perl[perl06], Erlang is still considered to be in the stage of the “early adopters”.

What makes Erlang special in developing web applications is its extremely powerful concurrency model, allowing a constant throughput of dynamic web pages under extremely heavy loads[Armstrong03]. Joe Armstrong showed in one of his presentations [Armstrong02] that in a denial of service attack, the amount 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.

A typical web application running an Apache web server with a MySQL database back end, glued together with perl and php needs all three applications to be independently managed and supported. Erlang will run all these applications within the same memory space², providing faster execution time as the needed data can be accessed directly. Erlang/OTP allows the different applications to be run independently, but using the same general frameworks which makes the system much easier to maintain. OTP's built in fault tolerance mechanisms implemented in Erlang gives us a system which has a very high availability, some systems even have an availability of up to and above five nines [Hinde00].

The web servers³ and databases[Hellman06] are mature, as is the use of Erlang for glue and service logic. What is lacking is a general framework for developing web based components, and the component themselves.

At Erlang Training & Consulting, an operation and maintenance system has been developed to ease development of web applications. Several components which use the operation and maintenance system have been created. The components handle all the bulletin, email lists and content management functionality of the websites developed using the operation and maintenance system.

This paper will introduce a metric for evaluating the internal and external complexity of web components, and therefore also their reusability [Cardino97]. The metrics was first translated from an object oriented mindset to concurrency oriented one. Then the metric was applied on the non-generic components developed for the first prototype of an Erlang based website. These components were later re factored and used in a wider number of web sites. The metrics were then re applied to determine any

2 Assuming a two tier architecture is not deployed. If so distributed Erlang socket connection are used to communicate between the front- and back-end.

3 Nortel's SSL Accelerator[Wikström01], T-Mobile's 3rd party gateway[Mullaparthi05] and many commercial websites.

effects on the complexity of components.

The case study indicates that developing more generic components does not decrease the complexity of the component, but rather it remains the same. Nothing can be said for sure with the limited amount of components used in this study. There is however nothing which indicates an increase in complexity when working with more generic components.

The Metrics

We use an adapted version of the metrics described by Clements, Kazman and Klein[Clements01]. The metric was originally designed to measure the complexity of components in telecommunication systems designed using an object oriented approach. The metric has to be translated from its object oriented approach into concurrency oriented programming approach.

Events and Calls are mapped to Function Calls. One might argue that a message is also an event, but two processes in two different components should only communicate through documented function calls, and not by sending messages directly in between them [ErlangDesignPrinciples].

Component Clusters are translated into Modules and include files, because in a web environment, components are relatively small and will not contain other components. Therefore, to give some meaning to this criterion, its granularity is increased from whole components to modules and include files.

Clements et al. Describes the concept of the Finite State Machine as a separate criterion in the metrics. FSMs are common in telecommunication systems, but when working with a more general metrics it is better to describe the FSM as an object oriented design pattern or, in Erlang terms, a OTP behavior. FSMs are therefore translated into OTP behaviors.

The natural translation of an Object is a process. Processes have states which change when messages from other processes are sent to it, just like objects.

The last and final concept to map to concurrency oriented programming is inheritance. There is no direct mapping for this concept, however, what is it really that inheritance allows? Sharing of functionality and structures between classes much in the same was as applications are used in concurrency oriented programming.

Now that we have established how to translate most of the key object oriented concepts into concurrency oriented programming, translating the original metrics is straight forward. The metrics consists of nine criteria which cover different aspects of component complexity:

1. **Interface Functions**

The number of interface functions which a

component has.

2. **Synchronous Calls**
The number of component external synchronous
3. **Asynchronous Calls**
The number of component external asynchronous calls.
4. **Modules**
How many modules and include files that the component consists of.
5. **Call Depth**
The maximum amount of non-recursive calls made as a result to an interface function.
6. **Behavior**
The number of behavior instances in the component.
7. **Process**
The number of static processes within the component.
8. **State Data**
The amount of non-associated loop data/state data within a component.
9. **Applications**
The number of application dependencies.

The mapping of Clements et al.'s object oriented metrics to the concurrency oriented metrics can be found in Appendix A.

The Components

We start by defining what a component is before describing what components were examined in this case study. The word component is used in many different contexts and therefore different sources describe the concept of a component differently. Clements Szyperski defines “a software component as a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties” [Szyperski96]. Szyperski's definition is good because it states requirements which we can test to see if something is a component or not.

In this paper, three different components were examined; a bulletin, a mailing list and a content management component. The three components were chosen because they provide generic functionality which many websites use.

The bulletin component allows websites to store different types of bulletin data. The publication date, the title, the detailed contents and whether to hide the bulletin from the website at the moment are some of the attributes which are available. The generic nature of the data stored allows

the component to be used in a many different situations, it can be used to show news bulletins, jobs bulletins and can also be used as an event calendar.

The mailing list component allows websites to provide a mailing list without having to worry about how to send emails, who to send them to and how to register new e-mail addresses to the mailing list. This component can easily be used in conjunction with the bulletin component described above to propagate all new bulletins to subscribers to the mailing list.

The content management component has very similar functionality to the bulletin component, but it can store more types of data, This allowing the user to index on the data. This allows the application to retrieve and display elements ordered or filtered by different fields.

The Method

A case study was conducted to determine if making web components more generic decreases their complexity. Three web applications were investigated and evaluated using the concurrency oriented metrics in Appendix A. The old components were then improved to be used with several websites. This time extra care was taken to try to reduce the complexity of the components by making them more generic, so that they would not have to be rewritten for each new website they were deployed upon. The new components were then evaluated using the same metrics.

Using the Metrics

Applying the metric to a component is achieved for most criteria. For criteria 1 to 3, the number of unique functions which fall within the criteria were counted. For criterion 4 the number of source files were counted. Criteria 7 and 9 were found by using the OTP process manager and the OTP application monitor respectively. The number of behaviors used was found by inspecting the source code.

Criteria 5 and 8 require extra attention. Criteria 5 assesses the maximum amount of non-recursive calls made as a result to an interface function. To measure this an Erlang program which uses an OTP syntax parser called `erl_scan` to analyze the source code and find the interface function with the most non-recursive calls. Criteria 8 is generally only present if an OTP behavior is part of the component. To get the value for criterion 8 we need to look at each of the OTP behaviors and see if they contain any state data. If they do we need to inspect that data and see if the data stored within the state is logically associated or not. The number of non-associative state data that is found while inspecting the component is what we use for criterion 8.

Result

The results from applying the metric to the three components are presented below.

Bulletin

When the the bulletin component was redeveloped to be used with several websites, the `gen_server`⁴, which in the first version was used to serialize data transmissions, was removed. Instead, the OTP database, Mnesia, was used to store the data which the `gen_server` previously handled. This caused a number of changes in the complexity of the component.

<i>Erlang Metrics</i>	<i>First - Bulletin</i>	<i>Second - Bulletin</i>
1. Interface Functions	14	12
2. Synchronous calls	12	9
3. Asynchronous calls	0	0
4. Modules	2	2
5. Call Depth	4	2
6. Behaviors	1	0
7. Processes	1	0
8. State data	2	0
9. Applications	2	2

Table 1: The metric evaluating the two Bulletin components.

Criteria 6 to 8 were brought down to zero and two interface functions (criteria 1) were removed from the component, as they solely were used for starting and stopping the static processes within the component. The call depth (criteria 5) and the number of synchronous calls (criteria 2) have also been reduced as the data processing aspects of the component was decreased. Both of the components depended on the Mnesia database application and the Erlang Standard Library, so criteria 9 remained unchanged.

Mailing list

No major changes had to be made to the mailing list component to make it work on several websites. Therefore the complexity metrics has remained unchanged for the

mailing list component, even after adopting it for usage on multiple websites. Some minor optimization were done within the component and the return values of the interface functions were modified to allow the component to be reused with greater ease.

<i>Erlang Metrics</i>	<i>First – Mailing list</i>	<i>Second – Mailing list</i>
1. Interface Functions	14	14
2. Synchronous calls	23	23
3. Asynchronous calls	0	0
4. Modules	4	4
5. Call Depth	9	9
6. Behaviors	0	0
7. Processes	0	0
8. State data	0	0
9. Applications	2	2

Table 2: The metric evaluating the two Mailing list components.

The large call depth (criteria 5) comes from the import feature, which allows the administrator to import email addresses from another mailing list directly into this one.

Content Manager

This component was developed from the beginning as a generic component for usage on multiple websites, therefore only one version of it exists. It was written for a single site, and with no changes, was deployed on several other sites.

⁴A standard behavior included in the Open Telecom Platform.

<i>Erlang Metrics</i>	<i>First – Content Manager</i>
1. Interface Functions	12
2. Synchronous calls	9
3. Asynchronous calls	0
4. Modules	2
5. Call Depth	4
6. Behaviors	0
7. Processes	0
8. State data	0
9. Applications	2

Table 3: The metric evaluating the Content Management components.

This component is almost identical in complexity compared to the second bulletin component. It is built on the same basic interface, but uses a more complex database and therefore has to use more functions to access the database, which is why criterion 2 is larger.

Discussion

Only the bulletin component changed when making it usable on several websites. Most of the changes probably occurred because the developer working with the component had gained a better insight in the problem at hand. One example of this is how criteria 6,7 and 8 moved from being 1,1 and 2 respectively to all being 0. This change most likely occurred because the developer gained a greater understanding of the problem at hand and could therefore improve part of the component.

The mailing list could be reused without changing anything that affected the complexity metrics. There were minor changes and it could be argued that those changes came from the developers new understanding of the problem. This can be because the component was with simplicity in mind from the beginning.

The content management component was already developed to be used with several websites and did not need redeveloping to be deployed. Of interest is that, just like the other components, criteria 6,7 and 8 are zero. This might be because the components are not complex enough to force the usage of those features, but it might also mean that web components should not use behaviors and permanent processes. It might also mean something completely different; further study of this area is

necessary to determine exactly what is going on with behaviors and processes in web components.

Conclusions

The case study indicates that developing more generic components does not decrease the complexity of the component, but rather, it remains the same. Nothing can be said for sure with the limited amount of components used in this study, there is however nothing which indicates an increase in complexity when working with more generic components.

The metric used to measure complexity might not have been suitable for evaluating the complexity of simple web components. It was originally designed to evaluate telecom components which, generally are more complex. If this means that it would be possible to efficiently use the metric on more complex web components or if it is not usable as all is not know at this time. One thing that is known though is that there is a lot of research has to be done in this area before being able to draw any solid conclusions.

Acknowledgement

Examinator: Thomas Arts, IT-Universitetet i Göteborg

Supervisor: Francesco Cesarini, Erlang Training & Consulting Ltd, London.

Bibliography

[EUC97] *Proceedings from the Third International Erlang User Conference*, 1997.

[Cesarini01] Francesco Cesarini, *Interfacing Erlang with Standardized Management Protocols*, Uppsala University, 2001.

[Rogers03] Everett M. Rogers, *Diffusion of Innovation*, 5th Ed, The Free Press, 2003.

[Armstrong03] Joe Armstrong, *Concurrency Oriented Programming in Erlang*, Swedish Institute of Computer Science, 2003.

[Clements01] Paul Clements, Rick Kazman, Mark Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley, 2001.

[Wikstrom06] Cleas Wikstrom, *Yaws*, <http://yaws.hyber.org>, 28 May 2006.

[Eddie06] Patrik Winroth et al., *The Eddie Mission*, <http://eddie.sourceforge.net/>, 28 May 2006.

[perl06] The Perl Foundation, *Perl*, <http://www.perl.org>, 29 May 2006.

[Php06] The Php Group, *Php*, <http://www.php.net>, 28

May 2006.

[ErlangDesignPrinciples] *Erlang.org*,
http://www.erlang.se/doc/doc-5.0.1/doc/design_principles/part_frame.html, 29 May 2006.

[Armstrong02] Joe Armstrong, Concurrency Oriented Programming in Erlang Presentation, *Lightweight Languages Workshop 2002*, Cambridge, MA, USA, November 9 2002.

[Component06] Tech Target, Component,
http://searchsmb.techtarget.com/gDefinition/0..sid44_gci211826.00.html, 15 June 2006.

[Szyperski96] Szyperski C. and Pfister C. Workshop on Component-Oriented Programming (WCOP'96), Summary. In Mühlhäuser M. (ed.) *Special Issues in Object-Oriented Programming—ECOOP'96 Workshop Reader*, pp. 127-130, dpunkt Verlag, Heidelberg, 1997.

[Nyström05] J.H. Nyström and P.W. Trinder, D.J., King, Proceedings of the 24th International Conference, SAFECOMP 2005, eds. R. Winther, B.A. Gran and G. Dahll, *Are High-level Languages suitable for Robust Telecoms Software?*, LNCS 3688, Computer Safety, Reliability, and Security, Springer-Verlag, 2005.

[Wikström01] Claes Wikström, Johan Bevemyr and Tony Rogvall, *The Best SSL Appliance in the World*, Proceedings from the Seventh International Erlang/OTP User Conference, 2001.

[Mullaparthi05] Chandrashekar Mullaparthi, *Third Party Gateway*, Proceedings from the Eleventh International Erlang/OTP User Conference, 2005. [Hellman06] Emil Hellman, *Evaluation of Database Management Systems for Erlang*, Proceedings of the Fifth ACM SIGPLAN Erlang Workshop, 2006.

[Kant00] Kant, K. and Mohapatra, P. 2000. Scalable internet servers: issues and challenges. *SIGMETRICS Perform Eval. Rev.* 28, 2 (Sep. 2000), 5-8.
DOI=<http://doi.acm.org.ezproxy.ub.gu.se/10.1145/362883.362891>.

[Hinde00] Sean Hinde, *Use of Erlang/OTP as a Service Creation Tool for IN services*, Proceedings of the Sixth Erlang/OTP User Conference, 2000.

[Cardino97] Cardino, G., Baruchelli, F., and Valerio, A. 1997. The evaluation of framework reusability. *SIGAPP Appl. Comput. Rev.* 5, 2 (Sep. 1997), 21-27.
DOI=<http://doi.acm.org.ezproxy.ub.gu.se/10.1145/297075.297085>.

Appendix A: The Metrics in detail

Erlang Metrics:	Description from: “Evaluating Software Architectures: Methods and Case Studies” [Clements01].	Translation with development of web applications in concurrence oriented programming.
1. Interface Functions	The number of synchronous and asynchronous calls to which an object reacts.	The number of interface functions which a component has.
2. Synchronous calls	The total number of synchronous calls made by an object to other objects, either to get or set some data/resource.	The number of synchronous calls which the component makes to modules not part of the component.
3. Asynchronous calls	The total number of asynchronous calls made by an object to other objects.	The number of asynchronous calls which the application makes to modules not part of the component.
4. Modules	The number of component clusters of which an object is composed. For example a car is made of wheels, steering mechanism, transmission, and so forth, and these objects are in turn made of other objects.	How many modules and include files that the component consists of.
5. Call Depth	The number of layers of encapsulation that define an object.	The maximum amount of non-recursive calls made as a result to an interface function.
6. Behaviors	If an object's behaviors is described by an FSM, the states which are also described by FSMs, this measures the depth of that indirection.	The number of behaviors in the component.
7. Processes	The total number of data classes used or referred by an object.	The number of static processes within the component.
8. State data	The number of variables needed by an object's FSM to deal with the machine's synchronization aspects.	The amount of non-associated loop data/state data within a component.
9. Applications	The total depth of an object (from the base class) in the system's inheritance tree.	The number of application dependencies.