Seminar Report

OPA-Lang

Martin Karrer
martin.karrer@stundent.uibk.ac.at

30 July 2015

Supervisor: Priv.-Doz. Dr. René Thiemann

Abstract

Opa is a very young programming language targeting web development. Opa tries to combine the three mein goals of webdevelopment security, scalability and reliability. Opalang is developed by MLstate and was releast as stable on 13th February 2013. This articel will look at the benefits and features of this programming language. Also we will compare Opa to two other popular programming languages Ocaml and Ruby. For a better understanding some code examples and pictures will be used.

Contents

1 Introduction

Opa is a a very young programming language. The first language design and prototype was officially presented at the OWASP conference in 2010 and the source code of OPALang was published under the GNU Affero General Public License in June 2011. The language is mainly developed by MLState, a well known company in the world wide web for messaging and mail exchange. MLstate created this language to make web developing more fun again. The design of the language solves most problems for the developer, so the code writer does not have to waste time for checking browser compatibilities or maintaining the full stack, application from to database and client. [13]

OpaLang can be used for both client-side and server-side scripting, where complete programs are written in OpaLang and subsequently compiled to Nodejs on the server-side, MongoDB calls for the database, and JavaScript on the client-side. The compiler automatically handles all communication between the database, server and client components. "OpaLang implements strong, static typing, which can be helpful in protecting against security issues such as SQL injections and cross-site scripting attacks." [13]

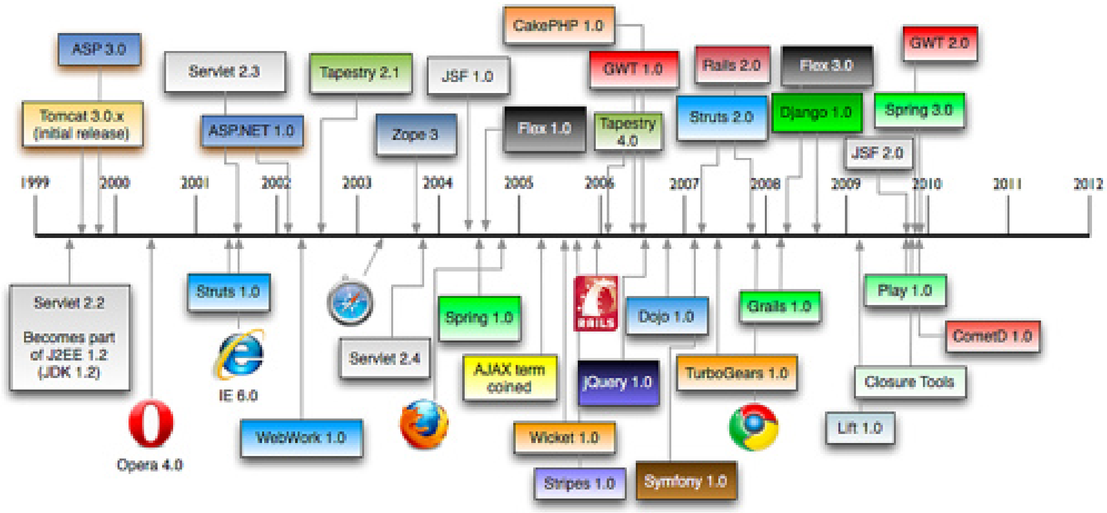
Figure 1: OpaLang Logo

OpaLang Code looks like JavaScript code on the first look, in development OpaLang shows a full functional feature set and behaves like a well known functional language like OCaml or Haskell.

2 Brief History of Webdevelopment

In 1989, developed Roy Fielding, Tim Berners-Lee and others at CERN, the European nuclear research center in Switzerland, the Hypertext Transfer Protocol, together with the concepts URL and HTML. With this base the foundations of the World Wide Web were created. The first results of these efforts was 1991, the HTTP 0.9 Version. [2] [3] At the same time at CERN, HTML (HyperText Markup Language) has been designed and released 1992. Two years later 1994 Hakon Wium Lie suggested Cascading Style Sheets in short CSS for describing the look and formatting of a document written in the hypertext markup language. [11] On April the 1st 1993 Mosaik, the first web browser with an general user interface was released in Version 1.0 which helped the world wide web to grow. The concept of cookies was originally developed by Netscape Communications Group and in 1994 published Netscape Navigator implemented cookies to keep a state with the server over the stateless communication with the HTTP protocol. [3] 1995 the first Version of PHP was released, the first server side scripting language for web servers. On September

18, 1995 Netscape released with the previous version of Navigator 2.0 a browser with an embedded scripting language called at this time Live scripts and had been developed by Brendan Eich, in the next Version the scripting language was renamed to JavaScript. Originally, Ruby on Rails was developed for a web application called Basecamp, but then extracted therefrom and presented in July 2004 to the public for the first time. The Version 1.0 was completed on 13 December 2005 and nowadays Ruby on Rails has a huge community. [5] From 2002 to the present the so called browser wars took place, fighting for the most users and including more and more new web standards and new versions of HTTP, CSS and HTML. Smartphones and other mobile devices for browsing the web, have created new browsers verisons and ensured that browser battle continues. [16]



3 Ideas and Goals

OpaLang was designed with security and scalability in mind. But it is not the only thing to make an application scaling and secure, it also should be maintainable and easy to write. OpaLang used all modern programming paradigms tested them and descidet to put them in to solve other problems. It is clear that Opalang was designed by web-developer who understood the problems and weak parts in the application chains and solved the problems all in a creative way. The main idea was to create one language for everything in a webapplication life cycle. MLState saw the problem in the myth of an full-stack developer. [15] It is too much know-how for one Developer. For example a full-stack PHP developer has to have knowhow about the hardware and operatingsystem for setting up an develop or production enviroment. He/She should have good programming skills in PHP and a lot of hardware to test on. But this is not everything a full-stack developer should know. Also security and scalabilty is a hot topic today. MLState improves this by the fact you only have to know how OpaLang works, everything else does the compiler for you. Every application is per design secure (in a limited way) and scalable, because every single key component in OpaLang is also built to be scalable (MongoDB, Node.js)

[7]

4 Usecases

Opalang is used for rapid web development, because it is easy to code and fast written. Opalang shaked up some web developer but did not make the breakthru at the moment. So OPALang is more likely used on Hackathons where someone tries to start a project with a new language. At the other hand OPAlang is used for rapid prototyping or on universities for some teaching purposes, which are not only webdevelopment, also source to souce compiling where the opalang compiler, which is written in ocaml, plays the leading role. [9]

5 Community and Surrounding

Having a solid community of people who uses the same language is very important for developing and gaining features. With a large community it is easier to get into a new language, because some platforms like stackoverflow.com offers a rich set of questions and good answers. Large communities help to spread news and the existence of this cool new language.

It is hard as an OPAlang Developer to find the community (if it is even there) the forum hosted by the MLState is down for about two years. In the IRC-Channel "opalang" on chat.freenode.net are only 5 People online, 3 of them are channelbots. The only two ways to connect to opalang developers is stackoverflow and the blog posts from MLState and the comment section of each blogpost.

The other point of view can be from the communities itself. PHP developers are a sworn in community offering frameworks and is a matured language, because the first version was released 1995. PHP has a large fanbase and good community on forums and IRC-Channels. If you ask people for OpaLang in the IRC-Channel you will get the answers with prejudices like: "Server in Node.js is Javascript and Javascript is, we all know, super slow". Same thing with *Ruby on Rails* developer. Those don't want to learn a new language and a different application stack, they are happy with Ruby and the Rails web framework, they do not see the point to switch to a fresh an young untested language.

If you want to start a new Project with OpaLang you have to be brave, feeling like going in an area no human ever has been before. This is maybe the reason why OpaLang is not commonly in use, because the features are great and the language itself offers everything a web developer expects and want to have as a web developer, as you will see in the next section.

5.1 The License Problem

At the beginning OpaLang got a lot of feedback, most of them were not critics about the language design or used technologies but about licensing:

> We chose initially to release Opa under the Affero GPL (AGPL) license. We made this choice to ensure that all improvements to Opa benefit the whole community – and also provide more programs written in Opa.

[12]

The two components in OpaLang were both released under AGPL:

- the compiler
- the run time environment

Because of the latter, every program written in OpaLang, that links to the runtime, must itself be released under the AGPL or the GPL. [12]

On 2012 MLState came with an acceptable solution to change standard library and the native backend license to the GPL license with the so-called ClassPath exception, like Java. The exception means you can link the GPL code with any code, opening the door to license your application under any license.

6 The Language Itself

OpaLang is a statically and strongly typechecked language. This means that the type of an expression is computed once and for all at compile-time, an expression has one unique type and this type can't be used as another one. The opalang compiler is written in ocaml and is a so calles source to source compiler, compiler source code in this case from OpaLang to Node.js and MongoDB executable code. [13] [8]

In the further reading of this sections i will point out the basics of Opalang and also some special features which are not so common in serverside web development.

6.1 Lexical Conventions

It is clear to see that OpaLang want to be easy to learn, because lexical conventions are similar to well known languages. For example Comments can be created in severals styles, like Ruby, Ocaml, C or C++ Style comments are interpreted as valid OpaLang comments.

```
// one line comment
/*
  multi line comment
*/
/*
  nested
  /* multi line */
  comment
*/
```

A comment is treated as whitespace for all the rules in the syntax that depend on the presence of whitespace.

It is a good idea to document code. Documentation can later be collected and compiled to HTML by the opadoc tool and collected into a cross-referenced searchable HTML document. Documentation takes the place for special comments, starting with /**, as we know from Java.

```
/**
 * I assure you, this function does lots of useful things!
 * @return 0
**/
function zero(){ 0 }
```

6.2 Basic Datatypes

OpaLang has 3 basic datatypes: strings, integers and floating point numbers.

6.3 Integers

Integers literals can be written in a number of ways:

```
x = 10                    // 10 in base 10
x = 0xA          // 10 in base 16, any case works (0Xa, 0XA, Oxa)
x = 0o12                  // 10 in base 8
x = 0b1010 // 10 in base 2
```

6.3.1 Floats

Floating point literal can be written in two ways:

```
x = 42.42
x = .42 // one can omit the integer part when the decimal part is given
x = 42. // and vice versa
x = 42.5e10 // scientific notation
```

6.3.2 Strings

Text is represented by utf8-encoded character strings which is immutable. String literals can aslo be written in the common C/Java/JavaScript and Ruby syntax:

```
x = "hello!"
x = "\"" // special characters can be escaped with backslashes
```

OpaLang features string insertions, which is the ability to put arbitrary expressions in a string as we know from Ruby. This feature is comparable to string concatenations or manipulation of format strings, but is generally both faster, safer and less error-prone: [13]

```
x = "1 + 2 is {1+2}"
// evaluates to "1 + 2 is 3"

function email(first_name ,last_name ,company){
  "{String.lowercase(first_name)}.
  {String.lowercase(last_name)}@{company}.com"
}

my_email = email("Darth","Vader","deathstar")
// evaluates to "darth.vader@deathstar.com"
```

Expressions have to be embedded into strings between curly braces.

6.3.3 Native Datastructures

Opalang also supports some native datastructs, which are commonly known in functional programming. Records, tuples and lists are supported by default, but any other datastruct known from Ocaml or Ruby can be built with ease.

6.4 Type System

One of the most important features of OpaLang is its typing system. Although OpaLang may look like a dynamic language and has many advantages of dynamic programming languages, it is a compiled language which relies on a state-of-the-art type system. OpaLang is a statically and strongly type checked language. This means that the type of an expression is computed once and for all at compile-time, an expression has one unique type and this type can't be used as another one. In OpaLang, any expression evaluates into a value and functions are first class values. This means that, like integers, booleans, strings in other languages, are functions in OpaLang and can be passed as arguments, returned as results, stored in data-types, and so on. [8]

6.5 Type Inference

OpaLang compiler offers type inference, i.e. it automatically determines the type of expressions from the context, without the user needing to write these types. Hence, a definition like

```
x = 42
```

will lead x to have the type int because the *integer* literal 42 is known to be of type *int*, this type representing the type of integer values.

6.6 Polymorphism

The type system of OpaLang features polymorphism, so some types in which some parts are not constrained and can be of any type. These feature is also offered in OCaml and

Java calls this feature *Generic Datatypes* which are internaly handeled in a different way as in OpaLang or OCaml. [14]

6.7 Pattern Matching

PatternMatching is a so called dispatch mechanism: choosing which variant of a function is the correct one to call. Opalang allows pattern matching according to this syntax:

```
match (expr) {
  case case_1: expr_1
  case case_2: expr_2
  ...
  case case_n: expr_n
  default: expr_def
}
```

the expression *expr* is matched against each pattern in sequence, stopping on the first one that matches. This fitting between the value and the pattern may induce bindings of pattern variables. The environment is then extended by these bindings and the right-side part expression of the matching case is then evaluated as the result of the whole expression. [6] [14]

As a consequence, the matched value and all the patterns must have a same type. Because any right-side part expression of the cases can be returned, they must also have a same type. [14]

6.8 Partial Application

It is possible to fix one of N arguments of a function, to get an new function with the remaining arguments. Some, who know OCaml or Haskell will do this their inutitive way.

From a function add with 2 arguments, we derive a new function add1 with less arguments (only 1) by partial application:

```
function add(x,y){ x+y }
add1 = add(1,_) // which means function add1(y){ add(1,y) }
x = add1(2) // x is 3
```

The trapdoor of this functionality is, that the side effect of the arguments are once computed and cached and not (as we may think) each time the function is called.

7 OPA vs. Ocaml

To compare Opalang and OCaml would be the same as comparing Haskell and OCaml. Opalang will be compiled to other sourcecodes by the OpaCompiler, this compiler is written in OCaml. This suggests us that OpaLang Core Developers love function programming and may also OCaml. [7] The whole featureset of OCaml can be found again in OPALang. The syntax has been used from other better known mostly imperative languages like Ruby and Javascript. Typechecking and Typeinferance is excatly the same as in OCaml and Patternmatching differs only by syntax. [4] Ocaml developer will feel like at home when they use partial application in exactly the same way as in OCaml. The only thing OpaLang does not have is the *AutoGarbageCollection* Feature and honestly it does not need it, because the end code in production is Node.js and Javascript code.

Feature	OpaLang	OCaml
Functional	YES	YES
Pattern-Matching	YES	YES
Typeinference	YES	YES
Polymorphism	YES	YES
Modules	YES	YES
Tail-Recursion	YES	YES
AutoGarbageCollection	NO	YES

8 OPA vs. Ruby

Ruby is a scripting language developed by Yukihiro ”Matz” Matsumoto in Japan. Ruby is object-oriented general-purpose programming language and was influenced by Smallralk, Ada, Perl and Lisp. It supports multiple programming paradigms, including functional, object-oriented, and imperative. It’s type system is dynamic and also has automatic memory management. Version 1.0 of Ruby was release on December 25, 1996 under the BSD License. Matsumoto has said that Ruby is designed for programmers productivity and fun, following the principles of good user interface design. [1] [13]

Both Ruby’s interpreter is cross platform same as OpaLang compiler, instead of generating code the ruby interpreter is generating machine executable instructions. [1] As we learned before, Opalang is a strong typed language, which ruby is not. Ruby is dynamic typed and distinguishes from Opalang and OCaml. Both languages are active in development, but Ruby has definitely more community driven development and also a higher community background. Ruby and OpaLang are supporting lambda calculus and Ruby the base functionality for functional programming. Ruby is know for its easy to learn aspect, which Opalang and Ocaml (basically all functional programming languages) not have.

Feature	OpaLang	OCaml
Functional	YES	YES
Pattern-Matching	YES	YES
Typeinference	YES	NO
Dynamic Typing	NO	YES
Polymorphism	YES	YES
Modules	YES	YES
Tail-Recursion	YES	YES [1]
AutoGarbageCollection	NO	NO [2]

9 Reason for Development

Reasons for development was definitely the lack of support of the common web languages for the developer. Also the fact that it is not easy with standard languages and databases to provide scalability and security by design. As web developer you have to have, or should have pretty good knowledge of the full stack and the minimum number of 3 languages (Javascript, PHP, SQL). With OpaLang you can handle database, client and server code in one language, so the developer only needs to know the markup language HTML and is able to build secure, scalable and reliable application code.

10 Alternatives

There are lot of alternatives around in this time, because OpaLang is use-case is developing web applications you can achieve the same with the language Ruby and the Ruby on Rails framework. Also PHP and Python can be used with Laravel or Django web framework. It is also possible to write in C the same application with CGI web hooks. [10] But these technologies are not hitting the same goal, because you have to know at least three programming languages, two markup languages and the technology of the full-stack of you application. OpaLangs purpose is to make web development easy and fun. Also producing a scalable, secure and reliable productive environment by design is really awesome. So yes there are alternative but non of them have security and scalabilty per design in mind and you can not write in one single language the code for server and client side.

11 Conclusion

OpaLang is a interesting functional approach for web developing. The Language itself has a rich feature set for imperative and even more in functional programming paradigms. The main goal, making web developing more easy with the support for scaleable and secure web applications and a robust runtime could shake up web development completely. Since June 2012 the new License, the GPL License with the ClassPath exception, you are not anymore forced to open your applications code to the public and maybe OpaLang will get more interesting for companies. Also the approach to write simultaneously the frontend and backend code, in the same language, within the same module completely new in the web developer scene. Most of programmer errors will be notices or most critical parts will be autoatically done by the compile, e.g. the communication between database server and clients, so no more dead AJAX calls. Until now this new web language does not have a large community base and it's hard to get into this language. OpaLang is still in active development by MLState and some features for other database types will be added in the future.

References

[1] R. Author. Ruby language. https://www.ruby-lang.org/. Accessed: 2015-08-14, Added: 2015-07-30 10:02:02.

[2] T. Berners-Lee. The original http as defined in 1991. http://www.w3.org/Protocols/HTTP/AsImplemented.html. Accessed: 2015-08-14, Added: 2015-07-30 10:02:02.

[3] T. Berners-Lee. Rfc: Hypertext transfer protocol – http/1.0. https://tools.ietf.org/html/rfc1945. Accessed: 2015-08-14, Added: 2015-07-30 10:02:02.

[4] O. T. Contributors. Some great news on opa. http://blog.opalang.org/2013/02/some-great-news-on-opa.html.

[5] David. Rails 1.1: Rjs, active record, respond_to, integration tests, and 500 other things! http://weblog.rubyonrails.org/2006/3/28/rails-1-1-rjs-active-record-respond_to-integration-tests-and-500-other-things/. Accessed: 2015-08-14, Added: 2015-07-30 10:02:02.

[6] Etangreal and MLState. Static enforcement of web application integrity through strong typing. https://www.mendeley.com/research/static-enforcement-of-web-application-integrity-through-strong-typing/. Accessed: 2015-02-30, Added: 2015-07-30 10:02:02.

[7] InfoQ and MLState. Questions and Answers by opa developers. http://www.infoq.com/articles/Opa. Accessed: 2015-02-30, Added: 2015-07-30 10:02:02.

[8] S. P. Jones. Haskell 98 language and libraries: The revised report. http://www.haskell.org/onlinereport/exps.html#pattern-matching. Accessed: 2015-02-30, Added: 2015-07-30 10:02:02.

[9] A. Koprowski. Leaving MLStat - discussing opa. http://www.a-koprowski.com/news/Opa. Accessed: 2015-02-30, Added: 2015-07-30 10:02:02.

[10] J. Korpela. Getting started with cgi programming in c. https://www.cs.tut.fi/~jkorpela/forms/cgic.html. Accessed: 2015-08-30, Added: 2015-07-30 10:02:02.

[11] C. Lilley. Rfc: The text/css media type. https://tools.ietf.org/html/rfc2318. Accessed: 2015-08-14, Added: 2015-07-30 10:02:02.

[12] MLState. Opa license change: Not just agpl anymore. http://blog.opalang.org/2012/05/opa-license-change-not-just-agpl.html. Accessed: 2015-02-30, Added: 2015-07-30 10:02:02.

[13] MLState. OpaLang rapid and secure web development. https://ocaml.org. Accessed: 2015-03-30, Added: 2015-07-30 10:02:02.

[14] MLState. Opalang wiki. *Github Wiki*, 2014.

[15] A. Shora. The myth of the full-stack developer. `http://andyshora.com/full-stack-developers.html`. Accessed: 2015-02-30, Added: 2015-07-30 10:02:02.

[16] G. Wolfe. The (second phase of the) revolution has begun. `http://archive.wired.com/wired/archive/2.10/mosaic.html`. Accessed: 2015-08-14, Added: 2015-07-30 10:02:02.