Exposé zur Bachelor-Thesis

”Beyond .*Script - Implementing A Language For The Web”

Erstellt von: Veit Heller (s0539501)
Studiengang: Angewandte Informatik (Bachelor)
Hochschule für Technik und Wirtschaft (HTW) Berlin

Betreuer:
Prof. Hendrik Gärtner
Prof. Henrik Lochmann

Background

The modern web is comprised of an abundance of very different beasts. Technologies that powered the first versions of the World Wide Web, such as HTML, CSS and JavaScript, and relatively new conceptions like TypeScript, CoffeScript, PureScript, ClojureScript, Elm, LASS, SCSS, Jade and Emscripten - to name but a few - are shaping the internet as we know it. There is one flaw that many of the new technologies have in common, as different as they may look and feel - they are mere preprocessors. In the end, it all boils down to the classic technologies again and we are left with the same limited capabilities we have had for the last 20 years.

All of this is for a good reason: having fewer technologies means that browsers and clients have to care for only a handful of things instead of fighting a hydra. However, there is undoubtedly potential lost when we are focusing on what we already have instead of what could be. There is not a lot of research being done to develop embedded languages and technologies, which would be viable alternatives for the web.

Building a new programming language directly into the browser would not, however, be a good idea. Standard committees exist for a reason and building and maintaining a production size interpreter, making it work within the context of a widely used browser and integrating it seamlessly with existing components must not be underestimated. This all leads to the conclusion that the prototype of such a language has to be created on top of existing technologies (i.e. it has to be implemented in JavaScript). Soundness, security and stability have to be guaranteed.

Purpose of this work

To substantiate these ideas, a preexisting language shall be implemented in JavaScript. By removing the creation of the language from the problem space, the design process will be simplified. This also allows for reusing the existing standard library and any software that was created in the source language. To narrow it down further, the existing implementation of the language should be cross-compiled, so as to not build any components that are already there.

To allow the implementation to run in the environment of a webpage without requiring any change in the browser, the compilation target will be JavaScript. In order to make the experience resemble native browser technologies as closely as possible, a few additions to the source language will have to be made.

Zepto, a Scheme derivative with a fairly simple implementation, was chosen as a compilation source. It is based on the R5RS standard of the Scheme programming language.

All of this work seeks to develop a novel approach for implementing and using languages on the web, possibly even for those that were never designed to do that. Many industries that usually operated in the environment of desktop computing or even data centers, e.g. data science and gaming, now increasingly rush towards the web in an effort to mine its capabilities. They usually have different needs than the traditional web development industry, which is apparent in the way JavaScript was designed. Libraries are created to work around these deficiencies, although at the core, nothing changes. Fast computing with infinite precision numbers, effective, comfortable multithreading, and type safety are only some of the features those industries will typically need. Being able to develop Domain Specific Languages or even general purpose programming languages is a real advantage for these communities.

R5RS is a standard of the Scheme programming language, a Lisp derivative popular in universities, research and teaching. Zepto was chosen as a basis for its extensibility. Through its massive metaprogramming capabilities, it functions as a swiss knife for building domain-specific languages and toolkits. Many of the aforementioned needs are already baked into the language definition.

The focus of this work lies in the creation of a working interpreter for the web, adhering to all the requirements that were previously mentioned. The toolchain has to be easily reproducible and maintainable. The overloading of existing HTML elements - i.e. `script` tags - will be implemented to make the emulation of native programming as realistic as possible.

Structure of this work

In the preliminary the terminology should be explained and the basis of the subject matter is to be laid out.

Within the context of an analysis of *Related Work* frequently used approaches and standards (if they exist) should be evaluated. The approaches that are being used have to match the design goals of the project and its structure. Evaluation models will be chosen to make possible an unbiased analysis.

A *Concept Design* has to be created, based on preliminary goals and standard approaches. The system has to work on standard web technologies without the need of any third party libraries.

Following that, the *System Design* explains the architecture of the runtime and the language itself. Interfaces have to be created to integrate it seamlessly into existing code and runtimes.

The *Implementation* details the steps needed to make Zepto useful in a browser context. This includes cross compilation technology, any additions to the original source and an explanation of the resulting pipeline.

An *Evaluation of the Prototype* will be documented. It details the functionality and interface and how to work with it and applies aforementioned evaluation models to analyze and interpret the findings.

Lastly, the *Summary and Outlook* will present a look into the future of the language and related technologies.

Outline

Timetable

1 Research (1 week)
- 1.1 Cross Compilation . . . . . .
- 1.2 Hijacking of HTML Components . . . . . .

1 Concept (1 weeks)
- 2.1 Construction Design . . . . . .
- 2.2 Additional Features . . . . . .

3 Implementation (3 weeks)
- 3.1 Language Port . . . . . .
- 3.2 Integration into the Browser . . . . . .

4 Writing the Bachelor Thesis (3 weeks)

5 Corrections (2 weeks)
- 5.1 Test/Correction Prototype . . . . . .
- 5.2 Correction of Bachelor Thesis . . . . . .

Bibliography

Aho, A. V., M. S. Lam, R. Sethi, and J. D. Ullman (2006). *Compilers: Principles, Techniques, and Tools*. Pearson Eudcation, Inc.

Fogus, M. (2013). *Functional JavaScript*. O'Reilly Media.

Parr, T. (2010). *Language Implementation Patterns*. The Pragmatic Programmers, LLC.

Queinnec, C. (2003). *Lisp in Small Pieces*. Cambridge University Press.