AARHUS UNIVERSITY

Coversheet

This is the accepted manuscript (post-print version) of the article.

Contentwise, the accepted manuscript version is identical to the final published version, but there may be differences in typography and layout.

How to cite this publication

Please cite the final published version:

Anders Møller. 2018. Technical perspective: WebAssembly: a quiet revolution of the web. Commun. ACM 61, 12 (November 2018), 106. DOI:<https://doi.org/10.1145/3282508>

Publication metadata

Title:	Technical perspective: WebAssembly: a quiet revolution of the web
Author(s):	Anders Møller
Journal:	<i>Communications of the ACM</i>
DOI/Link:	https://doi.org/10.1145/3282508
Document version:	Accepted manuscript (post-print)

General Rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognize and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

If the document is published under a Creative Commons license, this applies instead of the general rights.

Technical Perspective: WebAssembly: A Quiet Revolution of the Web

By Anders Møller

When JavaScript was introduced in 1995, it was intended as a small scripting language for interacting with the HTML DOM. A typical use was validating user form input or making simple animations. For many years, JavaScript programs were mostly small, and the majority of the program code in web applications was running on the servers, not in the browsers. This changed with the advent of fast JavaScript engines like V8, which enabled a new generation of web applications that were executed mostly in the browsers to provide a better user experience.

Within the last decade, commonplace JavaScript programs have grown to many thousands of lines of code, and JavaScript is used far beyond what anyone had anticipated in 1995. Despite the ongoing evolution of the language, it has been stretched to its limits. This has led to people developing compilers from other languages to JavaScript (although JavaScript is horrible as compilation target), and to language extensions and specialized runtime support (in particular, `asm.js`). Still, JavaScript has maintained a remarkable monopoly, being the only programming language supported by all main browsers. Until now.

The following paper gives an overview of the initial design of WebAssembly, a new low-level programming language for web-based software. The language has well-defined semantics that ensures program execution to be independent of the underlying hardware and operating system. Browsers routinely run code from untrusted

sources, so safety is obviously also of paramount importance. For this reason, WebAssembly is equipped with a type system that prevents certain safety violations at runtime.

The paper explains the rationale behind the main design decisions, gives an overview of the language, and describes preliminary experience from developing the implementations. No individual features are groundbreaking from a programming languages design point of view (for example, the authors refer to the type system as “embarrassingly simple”); the value is in the sum of the design choices.

Perhaps the biggest feat is that the WebAssembly team, initiated by Luke Wagner at Mozilla and Ben Titzer at Google Munich, has managed to unify the browser vendors and coordinate the design and implementation effort. (The reader may remember the “browser wars” where certain browser vendors deliberately undermined standardization efforts in attempt to gain market shares.)

WebAssembly provides a powerful platform for running non-JavaScript code on the Web.

The paper gives an overview of the formalization, including the operational semantics and the type system. Quite remarkably, the designers have chosen to formally specify the language, which has contributed to the clean design. This approach also demonstrates the power of formal techniques and mechanized lan-

guage semantics: the essential type safety properties now have machine-checked proofs, which guarantees a solid foundation for the running software.

WebAssembly is now supported by all the modern browsers, and it has been embraced by a wide range of software companies. The primary goal of WebAssembly is (currently) not to replace JavaScript, but to complement it by making it easier to develop computationally demanding web applications, such as games, software for audio/video processing, virtual reality systems, and CAD tools, and to port desktop applications to the web.

This is just the first step. The initial focus of the WebAssembly team has been on compilation from C/C++, and the first major milestone has been reached. The future work will likely concentrate on extending WebAssembly with support for parallel execution, and beyond that memory management with garbage collection, which will simplify compilation from many high-level programming languages, for example Java, C#, Swift, and OCaml. Although specifically designed with browser-based execution in mind, despite the name, there is actually not much “Web”-specific in WebAssembly. Perhaps the “write once, run anywhere” slogan once used by Java will be resurrected with WebAssembly?

Anders Møller (amoeller@cs.au.dk) is a professor in Department of Computer Science at Aarhus University, Denmark.