WebAssembly Modules as Lightweight Containers for Liquid IoT Applications

Niko Mäkitalo[1], Tommi Mikkonen[1], Cesare Pautasso[2], Victor Bankowski[1], Paulius Daubaris[1], Risto Mikkola[1], and Oleg Beletski[3]

[1] University of Helsinki, Helsinki, Finland
first.last@helsinki.fi
[2] University of Lugano, Lugano, Switzerland
cesare.pautasso@usi.ch
[3] Huawei Technologies, Helsinki, Finland
oleg.beletski@huawei.com

Abstract. Going all the way to IoT with web technologies opens up the door to isomorphic IoT system architectures, which deliver flexible deployment and live migration of code between any device in the overall system. In this vision paper, we propose using WebAssembly to implement lightweight containers and deliver the required portability. Our long-term vision is to use the technology to support developers of liquid IoT applications offering seamless, hassle-free use of multiple devices.

Keywords: Light-weight containers · internet of things · IoT · liquid software · containers · WebAssemly · web of things · WoT.

1 Introduction

Today, in the context of Internet of Things (IoT), web APIs are commonly used, but actual devices and applications in them are often implemented with native technologies. However, going all the way to IoT with web technologies would open up the door to *isomorphic* IoT system architectures. In such architectures, devices, gateways, and the cloud can run the same software components and services, unaltered. This will allow flexible migration of code between any element in the overall system. Practical isomorphic application scenarios include virtual assistants and other ubiquitous applications for messaging, gaming, reading, writing, listening to music/podcasts/news, or watching video.

Unfortunately, today's container techniques are often too heavy-weight for that, especially when considering devices with limited resources or direct access to hardware [5, 21]. A recent taxonomy of IoT client architectures [25] distinguishes bare metal RTOS systems, systems with a language runtime, and systems with full OS. Besides, some propose containers as a solution for IoT systems, where requirements regarding resources are relaxed. However, the taxonomy overlooks other architecture options than that the containers are built on top of an OS.

In this paper, we propose using the language runtime approach – the simplest option to enable 3rd party application code [25] – as the basis for lightweight containers. As the concrete implementation environment, we use WebAssembly (WASM). WASM was initially conceived to enable near-native execution speed [10] inside the browser. Following the same path of JavaScript runtimes which left the browser many years ago, today there are WASM implementations that can be run outside the browser as well [3].

Our long-term vision is to use WASM to implement the concept of liquid software [20] – user-centric, hassle-free use of multiple computers with software which can dynamically flow between them – in the context of the IoT. In essence, building liquid web applications needs two facilities, (i) ability to relocate code freely across different computing environments; and (ii) ability to synchronize the state of the application across all devices running the code. In our previous work, we have used the DOM [26] and Web Components with Polymer [7] as the underlying technology. However, both technologies are closely tied to the browser, and target at the UI layer of web applications. In particular, the unit of deployment in both approaches has been a web page, which is not optimal for embedded device, especially those that have no screen. WASM's characteristics – small footprint, near-native performance, advanced security, support for modules, and in-built support for isomorphic use – make it an attractive candidate for considering to use the Web as a platform for IoT applications.

2 Background and Related Work

This work is based on and related to the following relatively distinct technologies:

Web of Things. Web of Things (WoT) describes a set of standards by the W3C for solving the interoperability issues of different Internet of Things (IoT) platforms and application domains[4]. In essence, WoT is about making each 'thing' part of the Web by giving it an URI that can be used for communicating with it. The communication with each thing should be supported with a common data model and a uniform interface that is recognized by every thing[5].

Assuming such Web API for things deployed widely, programming IoT could be simplified to a large degree. Then, every device would provide this API and its features for programs that want to address its properties. Given a powerful enough API – in the context of this work, powerful enough to allow offloading of software on the fly – the promise of the Programmable Web [19] could be extended to cover the programmable world concept [27], using the Web as the underlying standard, interoperable technology platform [18].

WebAssembly. WASM [29] is a fast, safe and portable binary instruction format which can be executed on a stack-based virtual machine that can leverage contemporary hardware [3, 12]. WASM code is validated and run in a sandboxed environment; there is no ambient access to the computing environment in which code is being run except through explicit permission. Actual programs have

[4] https://www.w3.org/TR/wot-architecture/Overview.html, accessed Oct. 21, 2020.
[5] https://iot.mozilla.org/wot/, accessed Oct. 21, 2020.

compact representation, so they are small to transmit, especially in comparison to text or native code. Programs can be written by a variety of programming languages and then compiled to WASM for execution.

WASM programs are organized into modules, which are the unit of deployment, loading, and compilation [10]. Each module can contain definitions for types, functions, tables, memory areas, and global variables. These definitions may be imported or exported. To support rapid startup and dynamic configurations, WASM offers facilities for execution time dynamic linking.

Each WASM module executes within a sandboxed environment separated from the host runtime using fault isolation techniques. Hence, applications execute independently, and only specific features can be accessed by providing explicit permissions to APIs. Moreover, the security policies of its embedding are applied to the module. Within a web browser, this means the same-origin policy. On a non-web platform, no uniform model exists yet. So far, domain-specific and capability-based security models have been proposed.

The original design goals of WASM were to make it compatible with the web browser [29]. To this end, WASM applications can call into and out of the JavaScript context and access browser functionality through the same Web APIs accessible from JavaScript. For web pages and browser applications, which have already become overly complex [4], embedding WASM to JavaScript is an option that does not add many memory or performance related constraints.

Despite the increasing computing capacity of chips, it is still expected that the future networks include memory and performance-related challenges as many devices have limited memory. Moreover, in the context of IoT systems, computers, in general, have diverging performance capabilities, ranging from almost bare metal in sensors to cloud systems where everything is virtualized [22].

Lightweight WASM Containers for IoT. There are numerous WASM virtual machines that can be run outside the browser. The exact features of these systems vary[6], with some targeted for smallest devices, with the simplest possible interpretation, and others supporting sophisticated features such as streaming and ahead-of-time compilation[7]. Thus, small memory footprint and near-native performance make it an attractive alternative for building IoT systems [28].

Some work on the performance of WASM has already been composed, but the results seem inconclusive. The reasons are many, and include the fact that there are multiple runtimes for WASM, with varying performance and resource consumption[8]. For instance, [13] claim that in many of their benchmark applications, WASM was slower than native by a factor of 1.5. The work was conducted inside the browser, not using a runtime only, which may have affected the results. At the same time, [28] claim that the Wasmachine runtime is up to 11% faster than Linux for common IoT and fog applications.

[6] https://github.com/appcypher/awesome-wasm-runtimes, accessed Jan. 6, 2020.
[7] https://github.com/wasm3/wasm3/blob/master/docs/Performance.md, accessed Jan. 5, 2020.
[8] https://medium.com/wasmer/benchmarking-webassembly-runtimes-18497ce0d76e, accessed Oct. 21, 2020.

With the above facilities, we seek to build lightweight IoT containers, using the WASM language runtime as the basis for the implementation. Such systems can support third-party application development and dynamic changes, and it is possible to update the device software (or parts thereof) dynamically without having to reflash the entire firmware. Basically, applications run in a sandbox that provides only limited access to the underlying platform features – something that WASM immediately provides us at the level of modules.

Despite the idea's attractiveness, it seems that the idea has not received much attention in research. A recent thesis that includes a literature review points out that little research has been invested in considering the use of WASM modules as lightweight containers [23]. The study also points out that there are issues with memory usage at runtime when comparing WASM to Docker containers. What the study overlooks, however, is the fact that WASM module images are smaller than corresponding Docker images (or even smaller than compiled C/C++ modules), where facilities related to the infrastructure are included. Furthermore, while some WASM virtual machines can be run in various micro-controllers[9] and play the role of an operating system [28] – even bare-metal implementation is proposed[10].

Finally, to complement the ability to use WASM runtimes and modules as lightweight containers for IoT devices, WASM has also been used for serverless computing [11, 24]. Hence, the same technology has been demonstrated to be feasible across all the elements needed to build IoT applications.

3 Our Vision

Our prime motivation of this work is to rely on Web technologies all the way to IoT. Figure 1 represents how our goal is to push the boundaries of the development up to a point we reach *isomorphic* computations, where no constraints regarding the underlying architecture or platform are placed on applications, but they can be run everywhere, taking the context and its computational resources into account. In this deployment, using the Web as the underlying platform liberates developers from the restrictions of mainstream containers that rely on virtualizing a full operating system. This, in turn, results in more fine-grained deployment. Moreover, it is possible to consider hardware-related aspects by discovering the features available in this particular computing unit. Hence, the deployment loads modules on-demand basis only when necessary and customizes which module gets loaded, depending on the device. Such dynamic self-configuration is difficult to achieve with static images, which are commonly used by mainstream containers. Finally, since the device configurations and availability can change over time, the running software's deployment configuration needs to adapt dynamically. In other words, we want to go past dynamic deployment only and reach the full liquid web software vision [20], where software can flow and adapt to multiple devices. In essence, the solution must be able to migrate

[9] https://github.com/bytecodealliance/wasm-micro-runtime, accessed Oct. 21, 2020.
[10] https://github.com/lastmjs/wasm-metal, accessed Oct. 21, 2020.

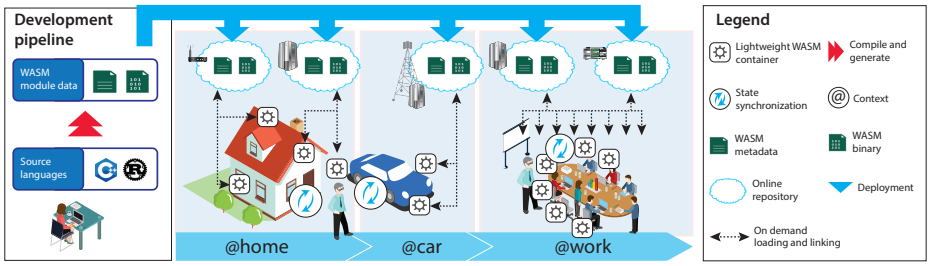

Fig. 1. Liquid IoT Application Lifecycle.

the execution's code and state so that the execution can continue one computer from the same program execution state it had on the previous computer [6].

WebAssembly's characteristics – small footprint, near-native performance, advanced security, support for modules, availability of language runtimes for different hardware devices – lead to potential support for isomorphic IoT applications. Using WebAssembly, one can relocate application code in a fine-grained fashion to different computing units commonly used in IoT systems. To this end, an approach similar to Apple's Handoff API [9] where applications can roam from device to device can be constructed, or one can rely on mobile agents for IoT like in [14], for instance.

In addition to simply deploying WebAssembly modules, it is also possible to support self-configuration by allowing the application to determine its environment, and dynamically load the necessary modules on the fly. Then, the initial deployment can be rapid – only a bootloader that is able to determine its functions at a particular location is needed. With the isomorphic nature of WebAssembly, actual application code can be the same despite its eventual location in the IoT architecture. This turns WebAssembly modules into lightweight containers that can easily be relocated.

Relocating and adapting code is only half of the liquid web application vision; also, application state and data should be transferred [20]. As WebAssembly relies on binary formats, techniques proposed in previous work, relying on browser facilities, cannot transform the application state. However, serialization techniques proposed in, e.g., [2] can be used to transfer the state of the applications when a WebAssembly module is relocated somewhere else in the IoT system. For the data part, techniques proposed in our previous work will be enough [7].

4 Proof of Concept Design

Currently, WebAssembly virtual machines outside of the browser do not support dynamic linking. Instead, all parts of an application must be present to run it. This essentially predefines task allocation at startup, and does not leverage full benefits of isomorphic architectures. To support more liberal configurations, we have implemented an execution time dynamic linking system, where modules

can be loaded on the need basis [17]. With this facility, the application can adapt to the role of the bigger context. The implementation uses execution time shared-everything linking approach, meaning that modules can use each other's functions and resources once they have been loaded. A video of these loading capabilities is available for demonstration purposes on YouTube[11].

Based on its context, the application can decide what modules to load. Modules can be loaded from the local disk or from an online repository, which in turn can contain parts of the code that can be freely allocated in the IoT network. At present, the implementation still lacks support for migrating live applications. Here, we plan to follow the approach of [14], where the developer defines the migration with a special API, at least initially.

5 Way Forward to the Vision

While the research done for this paper has been promising, there are numerous issues that still require practical solutions. Some key issues are listed below.

State synchronization. As already mentioned, our proof-of-concept implementation lacks support for application state migration. The main design decision documented in the design space [8] concerns whether developers need to explicitly annotate the state to be migrated and synchronized or whether the underlying runtime transparently takes care of it. In particular, reflection will be a topic of further investigation to help automate the migration.

Dynamic orchestration. Migrating applications from one computer to another cannot happen randomly, but it needs orchestration. This facility is to some degree a novel avenue to us, although it has received some attention in the context of stream processing [1]. In addition to an API that assumes full control, as in [1], we also plan to consider techniques used for self-organization [16].

Generalized API for hardware access. To truly enable isomorphic software architectures, also the environment where the software is run should be similar. In our present implementation, we have introduced adaptability mechanisms for taking the environment into account, but for large-scale use, such requirement can be a burden. Instead, a generalized API for hardware access would be a better solution. At the moment, WebAssembly offers WASI[12], a modular system interface for WebAssembly applications, but it is not generic enough for arbitrary IoT devices. However, it can act as a starting point for designing a uniform hardware access API across IoT architectures. Finally, even with a generalized hardware API, mechanisms are needed to discover what hardware modules are present at runtime, where the situation may change over time.

Fine-grained security model. While WebAssembly provides a sandboxing mechanism for applications at runtime level, something more comprehensive is needed at the scale of full liquid applications, their adaptive configurations, and migration. Here, our plan is to seek inspiration from mobile agents [15].

[11] https://youtu.be/gZj3M31ZfuI, accessed Dec. 28, 2020.
[12] https://wasi.dev/, accessed Oct. 21, 2020.

However, to truly address this aspect in detail, more specific use cases need to be considered, whereas here we have focused on technological factors only.

Benchmarking. As already mentioned, there is no conclusive data on the performance of WebAssembly applications in comparison to native ones. Performing systematic tests in the context of IoT and containers is therefore in our interests when our prototype implementation is more mature. Moreover, issues related to migration and liquid features also require benchmarking in the context of IoT to better understand the feasibility of the approach.

6 Conclusion

Going all the way with web in IoT development will help iron out numerous device and technology specific complications. In this paper, we propose using WebAssembly as a mechanism for building lightweight containers, which are capable of assuming different roles, depending on their location and roles in an IoT application. We demonstrated the use of the technology with a proof-of-concept implementation, and provided links to solutions that can be used to fill in the missing pieces needed for migrating full-fledged live applications.

References

1. Babazadeh, M., Pautasso, C.: A restful api for controlling dynamic streaming topologies. In: Proceedings of the 23rd International Conference on World Wide Web. pp. 965–970 (2014)
2. Bellucci, F., Ghiani, G., Paternò, F., Santoro, C.: Engineering javascript state persistence of web applications migrating across multiple devices. In: Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems. pp. 105–110 (2011)
3. Bryant, D.: Webassembly outside the browser: A new foundation for pervasive computing. In: Keynote at ICWE'20, June 9-12, 2020, Helsinki, Finland. (2020)
4. Butkiewicz, M., Madhyastha, H.V., Sekar, V.: Characterizing web page complexity and its impact. IEEE/ACM Transactions on Networking **22**(3), 943–956 (2013)
5. Celesti, A., Mulfari, D., Fazio, M., Villari, M., Puliafito, A.: Exploring container virtualization in iot clouds. In: 2016 IEEE International Conference on Smart Computing (SMARTCOMP). pp. 1–6. IEEE (2016)
6. Fuggetta, A., Picco, G.P., Vigna, G.: Understanding code mobility. IEEE Transactions on software engineering **24**(5), 342–361 (1998)
7. Gallidabino, A., Pautasso, C.: The liquid.js framework for migrating and cloning stateful web components across multiple devices. In: Proceedings of the 25th International Conference Companion on World Wide Web. pp. 183–186 (2016)
8. Gallidabino, A., Pautasso, C., Mikkonen, T., Systä, K., Voutilainen, J.P., Taivalsaari, A.: Architecting liquid software. J. Web Eng. **16**(5&6), 433–470 (2017)
9. Gruman, G.: Apple's handoff: What works, and what doesn't. InfoWorld (2014)
10. Haas, A., Rossberg, A., Schuff, D.L., Titzer, B.L., Holman, M., Gohman, D., Wagner, L., Zakai, A., Bastien, J.: Bringing the web up to speed with webassembly. In: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 185–200 (2017)

11. Hall, A., Ramachandran, U.: An execution model for serverless functions at the edge. In: Proceedings of the International Conference on Internet of Things Design and Implementation. pp. 225–236 (2019)
12. Jacobsson, M., Willén, J.: Virtual machine execution for wearables based on webassembly. In: EAI International Conference on Body Area Networks. pp. 381–389. Springer (2018)
13. Jangda, A., Powers, B., Berger, E.D., Guha, A.: Not so fast: analyzing the performance of webassembly vs. native code. In: 2019 USENIX Annual Technical Conference. pp. 107–120 (2019)
14. Järvenpää, L., Lintinen, M., Mattila, A.L., Mikkonen, T., Systä, K., Voutilainen, J.P.: Mobile agents for the internet of things. In: 2013 17th International Conference on System Theory, Control and Computing. pp. 763–767. IEEE (2013)
15. Kumar, S.A., et al.: Classification and review of security schemes in mobile computing. Wireless Sensor Network **2**(06), 419–440 (2010)
16. Kurzyniec, D., Wrzosek, T., Drzewiecki, D., Sunderam, V.: Towards self-organizing distributed computing frameworks: The H2O approach. Parallel Processing Letters **13**(02), 273–290 (2003)
17. Mäkitalo, N., Bankowski, V., Daubaris, P., Mikkola, R., Beletski, O., Mikkonen, T.: Bringing webassembly up to speed with dynamic linking. Accepted to SAC'21
18. Mäkitalo, N., Nocera, F., Mongiello, M., Bistarelli, S.: Architecting the web of things for the fog computing era. IET Software **12**(5), 381–389 (2018)
19. Maximilien, E.M., Ranabahu, A.: The programmable web: Agile, social, and grassroot computing. In: International Conference on Semantic Computing (ICSC 2007). pp. 477–481. IEEE (2007)
20. Mikkonen, T., Systä, K., Pautasso, C.: Towards liquid web applications. In: International Conference on Web Engineering. pp. 134–143. Springer (2015)
21. Morabito, R.: A performance evaluation of container technologies on internet of things devices. In: 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). pp. 999–1000. IEEE (2016)
22. Morabito, R., Cozzolino, V., Ding, A.Y., Beijar, N., Ott, J.: Consolidate iot edge computing with lightweight virtualization. IEEE Network **32**(1), 102–111 (2018)
23. Napieralla, J.: Considering webassembly containers for edge computing on hardware-constrained iot devices. Master's thesis, Blekinge Institute of Technology, Karlskrona, Sweden (2020)
24. Shillaker, S., Pietzuch, P.: Faasm: Lightweight isolation for efficient stateful serverless computing. arXiv preprint arXiv:2002.09344 (2020)
25. Taivalsaari, A., Mikkonen, T.: A taxonomy of iot client architectures. IEEE software **35**(3), 83–88 (2018)
26. Voutilainen, J.P., Mikkonen, T., Systä, K.: Synchronizing application state using virtual dom trees. In: International Conference on Web Engineering. pp. 142–154. Springer (2016)
27. Wasik, B.: In the programmable world, all our objects will act as one. Wired. Available online: http://www. wired. com/2013/05/internet-of-things-2/(accessed on Oct. 13, 2020) (2013)
28. Wen, E., Weber, G.: Wasmachine: Bring iot up to speed with a webassembly os. In: 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). pp. 1–4. IEEE (2020)
29. World Wide Web Consortium: WebAssembly Core Specification (2019), https://www.w3.org/TR/wasm-core-1/, https://webassembly.github.io/spec/core/_download/WebAssembly.pdf