Autonomous Agents on the Edge of Things

Demonstration Track

Timotheus Kampik
Umeå University
Umeå, Sweden
tkampik@cs.umu.se

Andres Gomez, Andrei Ciortea, Simon Mayer*
University of St. Gallen
St. Gallen, Switzerland
{andres.gomez,andrei.ciortea,simon.mayer}@unisg.ch

ABSTRACT

This paper describes a demonstration setup that integrates cognitive agents with the latest W3C standardization efforts for the Web of Things (WoT). The conceptual foundations of the implemented system are the integration of cognitive agent abstractions with W3C Web Things, which are generic abstractions of devices and virtual services that provide agents with various interaction affordances (e.g., actions, events). Together with the W3C WoT Scripting API, which is an ECMAScript-compatible API for W3C WoT environments, these standards allow JavaScript-based agents to be deployed and to operate in heterogeneous WoT environments. The agents can then be effectively distributed across the physical-virtual space in a *write once, run anywhere* manner: we deploy agents across a heterogeneous information system landscape that includes Web servers, browser-based front-ends, and constrained devices (microcontrollers). The deployment only requires minor platform-specific adjustments to consider resource and performance limitations on constrained devices. As a running example, we demonstrate a semi-autonomous assembly scenario with human-in-the-loop support.

KEYWORDS

Engineering Multi-Agent Systems, IoT, Constrained Devices

ACM Reference Format:

Timotheus Kampik and Andres Gomez, Andrei Ciortea, Simon Mayer. 2021. Autonomous Agents on the Edge of Things: Demonstration Track. In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), Online, May 3–7, 2021*, IFAAMAS, 3 pages.

1 INTRODUCTION

The vision of ubiquitous computing has become a reality: computing devices are omnipresent in a broad variety of manifestations, from washing machines and kitchen appliances we use in our private homes to self-checkouts in our grocery stores and production lines in our factories. At the same time, computing devices are becoming increasingly interconnected and autonomous, which enables them to take over duties that used to require human control; for example, modern cars can autonomously park themselves and communicate with their producer for faster troubleshooting (or even for predictive maintenance [2]). These developments can unlock new practical use cases for research on Engineering Multi-Agent Systems (EMAS), which among others deals with the development of design patterns, programming languages, and software frameworks for autonomous agents and multi-agent systems (MAS) [10]. Furthermore, the recent standardization of the Web of Things (WoT) at the W3C[1] and IETF[2] can facilitate the deployment of MAS across heterogeneous ubiquitous environments.

Nevertheless, according to a recent report of the EMAS community, more work is required to bring MAS technologies closer to mainstream software engineering [10]. Some recent initiatives are looking at the integration and deployment of goal-directed agents in WoT environments [3, 11, 13]. These works aim to narrow the gap between MAS technologies and Web/WoT standards, where the latter are often pragmatically oriented towards industry adoption.

This demonstration works towards a similar objective to showcase the integration of agent-oriented programming approaches and technologies with ongoing standardization efforts for the W3C WoT Scripting API [7]: we deploy cognitive agents across a heterogeneous information system landscape that includes Web servers, browser-based front-ends, and constrained devices (microcontrollers). The software used in our demonstrator is documented and available on GitHub.[3] Table 1 provides a non-exhaustive overview of environments that the implementation approach supports.

2 DEPLOYING AGENTS ACROSS A HETEROGENEOUS SYSTEM LANDSCAPE

The promise of *write once, run anywhere* was initially used to promote the Java programming language, but it is now more closely associated with JavaScript [9]: JavaScript applications are more conveniently deployable to many different platforms such as browsers, all mainstream mobile phone platforms, desktop computers, WoT gateways and devices, but also to servers and Function-as-a-Service (FaaS) environments[4]. Recently, several JavaScript libraries emerged that allow for the implementation of autonomous agents and multiagent systems in JavaScript: *Eve* [4] and the *JavaScript Agent Machine Platform* [1] are tools for implementing distributed systems, whereas the JS-son library [6] provides design patterns for implementing agent internals (*e.g.*, BDI reasoning loops). In the JavaScript ecosystem, distribution support is already provided by a range of mature, industry-scale tools and frameworks.

In the context of the WoT, a range of emerging standards embrace JavaScript. The W3C WoT Scripting API [7] is an ongoing effort to standardize an ECMAScript-compatible interface for discovering,

*Andrei Ciortea is also with Inria, Université Côte d'Azur, CNRS, Sophia Antipolis, France.

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, Online.

[1] https://www.w3.org/WoT/
[2] https://datatracker.ietf.org/wg/core/about/
[3] https://github.com/TimKam/JS-son/tree/master/examples/wot
[4] https://www.martinfowler.com/articles/serverless.html, accessed: Jan 6, 2021.

Deployment Target Type	Example Instance	API Peculiarities	Limitations
Server	WoT gateway	Node.js JavaScript runtime[4] APIs	-
Web browser	Web application, Google Chrome	Vendor-specific browser APIs	-
Device running mobile OS	WoT device running on Android OS	Vendor-specific APIs	Indirect access to native APIs
Desktop computer/laptop	Electron[5] control panel client	Framework-specific APIs	Indirect access to native APIs
Function-as-a-Service	Amazon Lamda function[6]	Vendor-specific subset of Node.js API	-
Microcontroller	Expruino *Pixl.js* device	Espruino runtime APIs	Sub-optimal performance

Table 1: Examples of potential deployment target types, the peculiarities of their APIs, and their limitations.

consuming, and exposing W3C Thing Descriptions [5], *i.e.* abstractions of devices and digital services in WoT environments. The WoT Scripting API targets primarily gateways and is part of the Abstract WoT Servient Architecture [8], a software stack that implements the W3C WoT standards and provides a runtime environment for applications — in our case, for software agents. The WoT Scripting API thus provides our agents with an abstract uniform interface for discovering and interacting with local WoT environments.

In our demonstration, we use the JS-son library [6] to program and deploy agents in any runtime environment that supports a JavaScript interpreter. We implemented a JavaScript module that can instantiate a JS-son agent based on a W3C Thing Description obtained via the WoT Scripting API. We therefore model W3C Web Things [8] as agents and extend them with autonomous behavior. In this context, we can consider a WoT *servient* as a a gateway that exposes a multi-agent system, i.e., a collection of passive *things* (comparable to *artifacts* in the Agents & Artifacts meta-model [12]), and pro-active, re-active and social *agents*. An agent can expose selected (and possibly pre-processed) beliefs as WoT *thing properties*, whereas WoT *actions* expose an agent's *services*. Agents can either run directly on gateways or devices. Agents running on devices may be mirrored on gateways by mock agents. Table 1 provides an overview of these deployment target types, the peculiarities of their APIs and their limitations (if there are any limitations that go beyond well-known limitations of the JavaScript programming language and interpreter).

A particular challenge is to deploy agents and MAS on constrained devices that do not have the resources to run fully-fledged MAS frameworks and platforms. As a starting point, we deploy JS-son agents on an Espruino Pixl.js device[5], which features an 64MHz ARM Cortex M4 microcontroller, 64kB RAM, 512kB Flash, and a JavaScript interpreter. Although JS-son is a lightweight library, with no dependencies and less than 1000 lines of code in its core, making JS-son agents run on an Espruino device requires adjustments to the code base. For this reason, we created an *Edge JS-son* variant of the library, which is (most notably) limited to belief-plan deliberation and manages plans in a global array such that agents merely need to maintain pointers to these plans.

3 DEMONSTRATION EXAMPLE

Our demonstrator[6] brings together several physical devices (a robotic arm; a low-power MCU-controlled display; various interactive triggers) and several simulated sensors. The robot's agent runs in a WoT servient and uses the WoT Scripting API to access and operate the robot, whereas the display's agent runs directly on the display's MCU. Figure 1 depicts the scenario's high-level architecture. At run time, this setup demonstrates the discovery of things via the WoT Scripting API and the deployment of an agent to an Espruino device using a custom, Web-based deployment interface. It then continues to show the sensor-based triggering of actions in agentified things which are processed according to the agent's own procedural reasoning. Finally, through simulated sensor measurements that trigger visual alerts and shut-down recommendations, we show how the interaction with the MAS can be facilitated even when parts of it are unavailable.

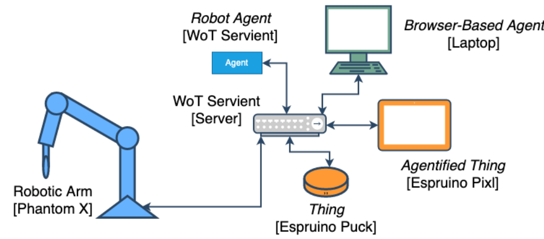

Figure 1: Hardware devices in the application scenario.

The demonstrator represents a setup that could potentially be found in factory where mass individualized products are assembled just-in-time according to customer requests. When an order is received, the robot's agent configures the manufacturing line to produce any custom parts that are not in stock and then assembles and packs the product(s). In this process, the robot's agent needs to interact with sensors to ensure its attainment of different goals (e.g., keeping the assembly line from overheating) while assembling the products as rapidly as possible. Humans receive guidance through browser-based applications on personal computers as well as through low-power displays that host user interface agents (and which, for example, still work in case of a power outage).

4 CONCLUSION

This demonstration showcases agent-oriented programming (AOP) for modern, standard-compliant ubiquitous computing ecosystems. We speculate that a broader adoption of AOP for ubiquitous computing is primarily a question of aligning with the latest developments in order to make agent-orientation abstractions available to industry practitioners and the broader software engineering community. This alignment would allow, in turn, to systematically evaluate the benefits, shortcomings, and potential extensions of AOP.

Acknowledgments. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, as well as by a GFF-IPF Grant of the Basic Research Fund of the University of St.Gallen.

[5]https://shop.espruino.com/pixljs, accessed: Jan 6, 2021.
[6]The demonstration video is available at https://youtu.be/MUhUuqd2jt0.

REFERENCES

[1] Stefan Bosse. 2016. Mobile Multi-agent systems for the Internet-of-Things and clouds using the JavaScript agent machine platform and machine learning as a service. In *2016 IEEE 4th international conference on future internet of things and cloud (FiCloud)*. IEEE, 244–253.

[2] Thyago P. Carvalho, Fabrízzio A. A. M. N. Soares, Roberto Vita, Roberto da P. Francisco, João P. Basto, and Symone G. S. Alcalá. 2019. A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering* 137 (2019), 106024. https://doi.org/10.1016/j.cie.2019.106024

[3] Andrei Ciortea, Simon Mayer, and Florian Michahelles. 2018. Repurposing Manufacturing Lines on the Fly with Multi-agent Systems for the Web of Things, In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018. *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2018)*, 813–822.

[4] J. d. Jong, L. Stellingwerff, and G. E. Pazienza. 2013. Eve: A Novel Open-Source Web-Based Agent Platform. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*. 1537–1541. https://doi.org/10.1109/SMC.2013.265

[5] Sebastian Kaebisch, Takuki Kamiya, Michael McCool, Victor Charpenay, and Matthias Kovatsch. 2020. *Web of Things (WoT) Thing Description, W3C Recommendation 9 April 2020*. W3C Recommendation. World Wide Web Consortium (W3C). https://www.w3.org/TR/2020/REC-wot-thing-description-20200409/

[6] Timotheus Kampik and Juan Carlos Nieves. 2020. JS-son - A Lean, Extensible JavaScript Agent Programming Library. In *Engineering Multi-Agent Systems*, Louise A. Dennis, Rafael H. Bordini, and Yves Lespérance (Eds.). Springer International Publishing, Cham, 215–234.

[7] Zoltan Kis, Daniel Peinter, Cristiano Aguzzi, Johannes Hund, and Kazuaki Nimura. 2020. *Web of Things (WoT) Scripting API, W3C Working Group Note 24 November 2020*. W3C Working Group Note. World Wide Web Consortium (W3C). https://www.w3.org/TR/2020/NOTE-wot-scripting-api-20201124/

[8] Matthias Kovatsch, Ryuichi Matsukura, Michael Lagally, Toru Kawaguchi, Kunihiko Toumura, and Kazuo Kajimoto. 2020. *Web of Things (WoT) Architecture, W3C Recommendation 9 April 2020*. W3C Recommendation. World Wide Web Consortium (W3C). https://www.w3.org/TR/2020/REC-wot-architecture-20200409/

[9] J. Lin and K. El Gebaly. 2016. The Future of Big Data Is ... JavaScript? *IEEE Internet Computing* 20, 5 (2016), 82–88. https://doi.org/10.1109/MIC.2016.109

[10] Viviana Mascardi, Danny Weyns, Alessandro Ricci, Clara Benac Earle, Arthur Casals, Moharram Challenger, Amit Chopra, Andrei Ciortea, Louise A. Dennis, Álvaro Fernández Díaz, Amal El Fallah-Seghrouchni, Angelo Ferrando, Lars-Åke Fredlund, Eleonora Giunchiglia, Zahia Guessoum, Akin Günay, Koen Hindriks, Carlos A. Iglesias, Brian Logan, Timotheus Kampik, Geylani Kardas, Vincent J. Koeman, John Bruntse Larsen, Simon Mayer, Tasio Méndez, Juan Carlos Nieves, Valeria Seidita, Baris Tekin Teze, László Z. Varga, and Michael Winikoff. 2019. Engineering Multi-Agent Systems: State of Affairs and the Road Ahead. *SIGSOFT Softw. Eng. Notes* 44, 1 (March 2019), 18–28. https://doi.org/10.1145/3310013.3322175

[11] Simon Mayer, Dominic Plangger, Florian Michahelles, and Simon Rothfuss. 2016. UberManufacturing: A Goal-Driven Collaborative Industrial Manufacturing Marketplace, In Proceedings of the 6th International Conference on the Internet of Things. *Proceedings of the 6th International Conference on the Internet of Things (IOT 2016)*, 111–119.

[12] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. 2007. Give Agents Their Artifacts: The A&A Approach for Engineering Working Environments in MAS. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems* (Honolulu, Hawaii) *(AAMAS '07)*. Association for Computing Machinery, New York, NY, USA, Article 150, 3 pages. https://doi.org/10.1145/1329125.1329308

[13] Ahmed Shafei, Jack Hodges, and Simon Mayer. 2018. Ensuring Workplace Safety in Goal-based Industrial Manufacturing Systems, In Proceedings of the 14th International Conference on Semantic Systems, SEMANTICS 2018, Vienna, Austria, September 10-13, 2018. *Proceedings of the 14th International Conference on Semantic Systems (SEMANTICS 2018)*, 90–101.