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:

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 W3C1 and IETF2 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) environments4. Recently, several JavaScript libraries emerged that allow for the implementation of autonomous agents and multi-agent 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,

1https://www.w3.org/WoT/
2https://datatracker.ietf.org/wg/core/about/
3https://github.com/TimKam/JS-son/tree/master/examples/wot
Table 1: Examples of potential deployment target types, the peculiarities of their APIs, and their limitations.

<table>
<thead>
<tr>
<th>Deployment Target Type</th>
<th>Example Instance</th>
<th>API Peculiarities</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>Web application, Google Chrome</td>
<td>Node.js JavaScript runtime APIs</td>
<td>Indirect access to native APIs</td>
</tr>
<tr>
<td>Web browser</td>
<td>WoT device running on Android OS</td>
<td>Vendor-specific browser APIs</td>
<td>-</td>
</tr>
<tr>
<td>Device running mobile OS</td>
<td>Electron control panel client</td>
<td>Vendor-specific APIs</td>
<td>Indirect access to native APIs</td>
</tr>
<tr>
<td>Desktop computer/laptop</td>
<td>Amazon Lambda function</td>
<td>Framework-specific APIs</td>
<td>Sub-optimal performance</td>
</tr>
<tr>
<td>Function-as-a-Service</td>
<td>Espruino Ptx.js device</td>
<td>Vendor-specific subset of Node.js API</td>
<td>-</td>
</tr>
<tr>
<td>Microcontroller</td>
<td></td>
<td>Espruino runtime APIs</td>
<td></td>
</tr>
</tbody>
</table>

3 DEMONSTRATION EXAMPLE

Our demonstrator 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.

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 the products as rapidly as possible. Humans receive guidance 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.
REFERENCES


