Increasing the Intelligence of Low-Power Sensors with Autonomous Agents

Increasing the Intelligence of low-power Sensors with Autonomous Agents

Jannik William
ETH Zürich
Switzerland
williamj@student.ethz.ch

Matuzalém Muller dos Santos
Maiquel de Brito
Jomi Fred Hübner
Universidade Federal de Santa Catarina, Brazil
matuzalemifsc@gmail.com
{maiquel.b,jomi.hubner}@ufsc.br

Danai Vachtsevanou
Andres Gomez
University of St.Gallen
Switzerland
{firstname.lastname}@unisg.ch

ABSTRACT

Low-power sensors are becoming ever more powerful, increasing both their energy efficiency as well as their processing capabilities. Much work in recent years has focused on optimizing machine learning models to low-power systems, typically to locally process sensor data. Significantly less attention has been paid to other artificial intelligence fields such as knowledge representation and automated reasoning, which may contribute to building autonomous devices. In this work, we present a low-power sensor node with an autonomous *belief-desire-intention* agent. This kind of agent simplifies the implementation of both proactive and reactive behaviors, promoting autonomy in our target applications. It does so by locally perceiving and reasoning, and then wirelessly broadcasting an intention, which can be forwarded to an actuator. The capabilities of the autonomous agent are demonstrated with a light-control application. Experiments demonstrate the feasibility of running intelligent agents in low-power platforms with little overhead.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Computing methodologies** → **Intelligent agents**; • **Hardware** → *Sensor applications and deployments.*

KEYWORDS

autonomous agents, low-power systems, reasoning systems

ACM Reference Format:
Jannik William, Matuzalém Muller dos Santos, Maiquel de Brito, Jomi Fred Hübner, and Danai Vachtsevanou, Andres Gomez. 2022. Increasing the Intelligence of low-power Sensors with Autonomous Agents. In *Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (AIChallengeIoT) (SenSys '22), November 6–9, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3560905.3568301

1 INTRODUCTION

This paper investigates the use of autonomous agents technologies for programming low-power devices. In the context of IoT, we have low-power devices that need to be programmed for several kinds of applications that often require some level of autonomy. These devices should behave beyond a mere data source. In the context of AI, we have promising and validated approaches to conceive and implement autonomy, especially in the domain of Multi-Agent System (MAS) and BDI agents. By defining the agent's code using mental constructs like Beliefs, Desires, and Intentions, BDI agents assist us in implementing autonomy (along with proactiveness and reactiveness). However, the limited resources of low-power devices impose some challenges for adopting BDI agents.

The programming languages being used to develop BDI agents are conceived for desktop computers and our target devices lack the processing power required to run these agents. We are thus interested in finding out what is the minimal set of BDI properties an agent must have in order to function in low-power devices. This paper describes an initial proposal for this minimal set, applies the proposal in a practical scenario, and evaluates which benefits of BDI agents are kept given the reduced set of properties. The application scenario is illustrated in Fig. 1: an agent replaces a central server and is placed together with the sensor in order to control the illuminance of a room.

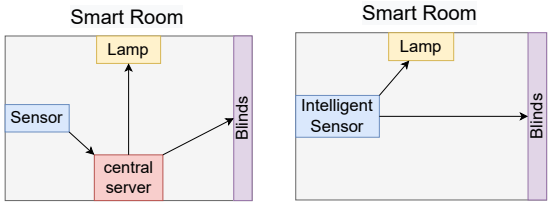
We start our investigation by looking for current approaches and initiatives to embed (BDI) agents into constrained devices (the results are briefly presented in Sect. 2). Taking these works as the starting point, we develop a new approach that intends to fulfill some drawbacks of previous work when considering the application domain we are focusing on (our general proposal is presented in Sect. 3). An initial application of this proposal was implemented in a (simple) light control system (presented in Sect. 4) and empirically evaluated (Sect. 5). The results, although still limited to one application, are promising and summarized in Sect. 6.

2 RELATED WORKS

An important challenge in this work is running BDI agents with communication capabilities in constrained low-power devices. Some proposals connect these devices, which manage sensors and actuators, to more powerful devices such as personal computers or single-board computers [1, 2, 6, 9, 10, 16]. The execution of the agents can be also split between constrained and unconstrained devices [15]. These powerful devices enable BDI agent execution and wireless communication. On the other hand, the power consumption of these systems exceeds the requirements of many IoT applications.

SenSys '22, November 6–9, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9886-2/22/11.
https://doi.org/10.1145/3560905.3568301



(a) Centralized approach. **(b) Intelligent agent approach.**

Figure 1: Low-power sensors are typically employed in a centralized approach, where decisions are made by a powerful centralized server. Autonomous agents can increase the intelligence of a low-power sensor and allow local decision-making to directly control actuators.

To run agents directly in constrained devices, some proposals use C++ [11], assembly-like languages [7], or on-chip synthesis of finite-state machine models [4]. These solutions are more energy-efficient than the previously mentioned ones since they use single, simpler devices that, in the case of [7] and [4], enable wireless communication, even with some limitations. However, they relinquish the high-level constructs of the BDI model, keeping to a limited extent features such as autonomy, proactivity, and reactivity.

Closer to the approach employed in this work, Bucheli et al. propose to translate BDI specifications written in AgentSpeak to C programs that run on constrained devices [5]. However, such translation meets the requirements of a particular application. This proposal is evaluated in simulated environments without explicit concerns about communication and power requirements. In comparison to these approaches, our work employs a general-purpose BDI agent programming framework to create agents that have all of the essential BDI features while also being able to run on constrained, low-power devices that provide communication infrastructure for agent interaction.

3 SYSTEM ARCHITECTURE

In [13], a lightweight single agent framework is suggested, which is based on a simplified variation of the Jason [3]. Jason is an interpreter for AgentSpeak, which is a high-level BDI agent programming language. The main objective of this work is to bring this embedded BDI framework and its simplified agents to an ultra-low power platform with tight processing, memory, and communication constraints. One of the components of the framework is a translation engine, which converts AgentSpeak programs into optimized C++ code. The other main components are a runtime library and the hardware-dependent code, which can be compiled in an executable binary (Fig. 2). The high-level workflow of the framework is to program the deliberation of the agent using the features of AgentSpeak, implementing perception and action functions together with other hardware-dependent code (like a real-time OS, sensor drivers and hardware interrupts) in C/C++.

3.1 Translation Engine

The translation engine is able to connect actions and perception functions available in the hardware with the naming convention used in the AgentSpeak program. It translates the AgentSpeak program to a special C++ structure representing the initial beliefs, goals, and plans of the agent. The resulting code structure can be compiled with a common C++ compiler and flashed to a microcontroller. For instance, if the agent code refers to a belief like `is_dark`, an instance of the C++ class `Belief` is included in the translated code. In this class, the belief is not identified by the string "is_dark", but a corresponding number (8-bit integer), e.g. 7, is used instead. This improves the memory and runtime performance of the embedded agent.

There exist further limitations to increasing performance. E.g. also the names of actions are converted to an integer. This limits these names to 256 distinct values. Furthermore, only propositions are allowed (instead of the predicates in Jason), thus no variables can be used in the agent program to be translated. These limitations however still allow us to develop reasonable complex agents (considering low-power embedded devices).

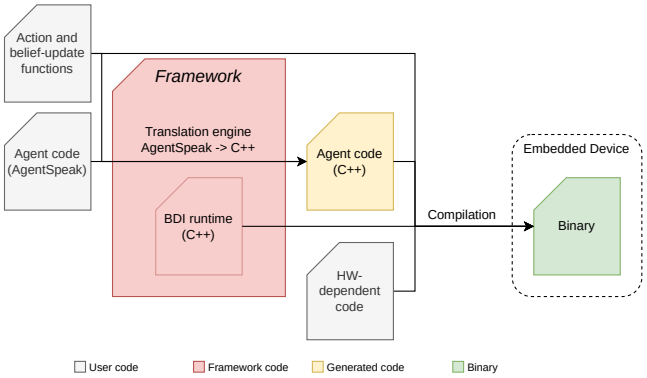


Figure 2: During the compilation process of the Embedded-BDI framework, an AgentSpeak program is translated to C++. Together with other platform-dependent code and the BDI runtime, the translated code is compiled into an executable.

3.2 BDI Runtime

The reasoning happens in the runtime part of the framework. It implements a simplified reasoning cycle from the Jason interpreter. The missing steps compared to a full Jason reasoning cycle are the message reception and selection steps. These are not implemented, as this framework only supports single-agent environments.

Fig. 3 summarizes the simplified reasoning cycle, which is called within an infinite loop by the microcontrollers, as shown in Alg. 1. The reasoning cycle consists of the following steps:

(1) *Update the belief base and generate events from new perceptions*
Percepts are stored as beliefs and this step updates them according to sensors values (as implemented in the "belief update functions" component coded in C++). For instance, if the light sensor is greater than a value, the belief number 7 (corresponding to `is_dark`) is assigned to true. If the value of a belief changes, an event is generated. Since the environment is limited to single agents, all possible beliefs are

already created in the translation phase in the belief base with either a true or false value.

(2) *Event selection*
An event is selected from the event queue to be processed.

(3) *Plan selection*
Plans relevant to the current event are selected. Additional filtering based on the context of the plan and the belief base is being made. Finally, the first applicable plan is selected and its instructions are added as an intention stack to the other intention stacks.

(4) *Selection of an intention for execution*
An active intention (not suspended by waiting for an event to be processed) is selected.

(5) *Execution of one step of an intention*
Executes a single instruction of an intention. Supported instructions are the addition and deletion of a belief, adding a subgoal, and adding new goals.

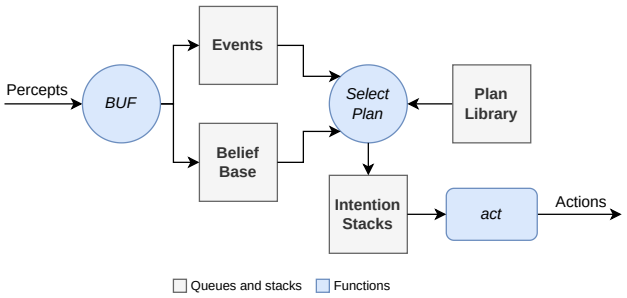


Figure 3: The practical reasoning cycle of the Embedded-BDI runtime is derived as a simplified version of the Jason interpreter. It is developed for low-power devices and therefore concentrates on the core functionality of a single agent runtime.

As the demo application requires communication with actuators, we need some means to communicate. The current version of the framework does not explicitly allow this, since single-agent environments do not need communication at the Jason level. Consequently, we encapsulate wireless communication into an action, since it is used exclusively to control an actuator. In our application, we use Bluetooth Low Energy (BLE) advertising packets to transmit these commands, which are then forwarded to the actuator device using a standard BLE receiver.

4 LIGHT CONTROL WITH A BDI AGENT

Our approach aims to enable the design and deployment of autonomous systems required for the more sophisticated, context-aware, and pervasive control of the increasingly available low-power sensors and actuators. The presented IoT application is composed of a single BDI agent that is situated in a room with a lamp and window blinds (Fig. 1). The agent has the design objective to preserve the optimal illuminance conditions for the rest of the occupants while managing to cope with the dynamicity of the environmental context (phase 1), and of the user's goals (phase 2). For example, the agent continuously perceives the illuminance level

Algorithm 1: Reasoning loop as executed by the microcontroller

```
Add initial beliefs to belief base;
Add initial intentions to intention stacks;
while true do
    Perceive the environment;
    Update belief base based on changes in environment;
    Generate events based on changes;
    if Event queue is not empty then
        Select event to be processed;
        Select relevant plans;
        Filter plans based on context;
        Select first plan;
        Add instructions of the plan into intentions;
    end
    if Intention stacks are not empty then
        Select single instruction of an intention;
        Execute this instruction;
    end
end
```

in its surroundings, both inside and outside the room. Every time the room becomes dark and the illuminance level should optimally be high, the agent strives to brighten up the room. The agent is able to exhibit such behavior by perceiving and acting on the room environment, i.e. it is able to:

- perceive indoor illuminance conditions (bright/dark),
- perceive outdoor illuminance conditions (cloudy/sunny/night),
- update the illuminance conditions by acting on a lamp (turn on/off), and
- update the illuminance conditions by acting on the blinds (raise/lower).

In the following, we present the evolution of our application across two phases. Listing 1 captures the final AgentSpeak program used by an agent to address the objectives of the application. By programming the agent decision strategy on top of abstractions that directly enable the implementation of reactive, goal-driven, and contingency behaviors (e.g., beliefs, goals, and plans in AgentSpeak), the autonomy of our light-control application successfully emerges from only 21 lines of code.

4.1 Adapting to the Environmental Context

In the first phase of the application scenario, the agent is launched and starts to continuously observe the indoor and outdoor illuminance conditions (`dark/bright_inside` and `sunny/cloudy/night_outside`), and strives to preserve the optimal conditions, in this case, to keep the room `bright_inside`.

Once the agent perceives that it is `dark_inside` the room, it reacts and adapts its behavior towards pursuing the goal of making the room `brighter`. For this, the agent can re-configure the system to increase the illuminance in a standard mode (`standard_mode_available`) by turning on a lamp (`turn_on_lights`). Yet, the

agent will select to re-configure the system differently if a more eco-friendly configuration is available (`eco_mode_available`) based on the current environmental context. As a result, when the agent perceives that it is `sunny_outside`, its reactiveness allows it to directly adapt and raise the blinds, instead of settling for a more energy-inefficient default option.

Finally, there is a possibility, that even if the agent proceeds with raising the blinds, the room is again perceived as dark later in time or in succession, e.g., because the outdoor illuminance level is no longer sufficient or the blinds did not operate appropriately. In this case, the agent's proactiveness keeps the agent committed to its initial goal of brightening the room, leading to the decision of additionally turning on the lamp.

Listing 1: The AgentSpeak program of an agent maintaining the illuminance conditions in a room. The agent strives to achieve different maintenance goals (l.3-7,22-26) by reacting to the corresponding user input at run time (l.1,20). While the agent pursues a specific maintenance goal, it remains reactive and dynamically adapts to the environmental conditions in the room (l.9-13).

```
1  +user_turn_on <- !!preserve_light.
2
3  +!preserve_light: bright_inside & user_turn_on
4      <- !!preserve_light .
5  +!preserve_light: dark_inside & user_turn_on
6      <- !brighten ; !!preserve_light .
7  -!preserve_light: user_turn_on <- !!preserve_light .
8
9  +sunny_outside: user_turn_on <- +eco_mode_available .
10 +cloudy_outside: user_turn_on
11     <- -eco_mode_available ; +standard_mode_available .
12 +night_outside: user_turn_on
13 <- -eco_mode_available ; +standard_mode_available .
14
15 +!brighten: eco_mode_available
16     <- turn_off_lights ; raise_blinds ; -eco_mode_available .
17 +!brighten: standard_mode_available
18     <- turn_on_lights ; -standard_mode_available .
19
20 +user_turn_off <- !!preserve_dark .
21
22 +!preserve_dark: night_outside & user_turn_off
23     <- lower_blinds ; !!preserve_dark .
24 +!preserve_dark: bright_inside & user_turn_off
25     <- turn_off_lights ; lower_blinds ; !!preserve_dark .
26 -!preserve_dark: user_turn_off <- !!preserve_dark .
```

4.2 Adapting to User Goals

In the second phase of the application scenario, the agent continues to be driven by its goal to preserve the optimal illuminance conditions, but this goal can be flexibly further configured at run time based on the preferences of a human occupant. For this, the application is extended with additional sensory input that enables the agent to better refine its goal-driven behavior. The agent, now, additionally perceives the user's preference to preserve high or low illuminance conditions (user turn on/off), and respectively, strives to keep the room `bright_inside` or `dark_inside`.

In case the agent perceives the occupant's `user_turn_on` preference, and that it is `dark_inside` the room, then its behavior is performed as in the first phase of the application scenario. However, if the agent perceives `user_turn_off` and `bright_inside`, it instead starts pursuing the goal of making the room darker. For achieving this goal, the system can be re-configured with the options of lowering the blinds and, optionally, turning off the lights – options that are followed or rejected by the agent based on the current environmental context (e.g., based on whether it is bright or dark inside the room).

It should be mentioned that the agent is capable of flexibly altering its behavior upon demand without staying committed to an outdated goal until fulfillment. For example, if the agent started pursuing to preserve high illuminance levels, but behaved currently without success because raising the blinds was insufficient, it will not directly proceed with turning on the light. Instead, it will deliberate again about its goals and if in the meantime, the human occupant expresses a preference for low illuminance levels, then the agent will directly adapt to the new goal and lower the blinds.

5 EXPERIMENTAL EVALUATION

In this section, we evaluate the proposed BDI framework using the previously described light control application. The experimental set-up is described first, along with the used hardware components and infrastructure. The light-control experiment is then presented in detail, along with its environmental traces. We then perform a low-level performance evaluation of the BDI framework and present our analysis.

5.1 Experimental Set-Up

The ultra-low power Edge platform from Sparkfun Electronics has been used for this application scenario. This module contains an Ambiq Apollo3blue cortex M4 microcontroller with 1 MB of Flash memory, 384 KB of SRAM, and a 48 MHz nominal clock frequency. It also contains a Bluetooth Low Energy (BLE) radio, which is used to broadcast the actions chosen by the BDI agent. The Apollo3blue is connected to two optical sensors, the VEML7700 ambient light sensor, and the AS7262 spectrometer. These low-power sensors are used to determine the outdoor (VEML sensor) and indoor (AS sensor) illuminance conditions of a laboratory at the University of St.Gallen. The Sparkfun Edge also contains a LIS2DH12 accelerometer, which is used as a simple user interface. When the Z-axis is positive, it indicates the user's preference for light. If the user turns the device upside down, he's indicating a dark preference. For simplicity of experimentation, we place the device close to a window, and using a 20 cm cable, the VEML sensor is placed outside, where it can determine the outdoor luminosity even when the blinds are shut. The indoor illuminance is obtained by averaging the AS7262's six channels. The final set-up is shown in Fig. 4. Though not visible in the picture, there are controllable blinds and lamps in this room. The embedded BDI agent can control these actuators via BLE advertisements, which are received by a Raspberry Pi and translated to actuator commands. For the experiments, we also attached a UART cable to be able to record the logs of the experiment for post-processing.

5.2 Light Control using BDI Agents

To deploy the embedded BDI agent for our light control application, we first automatically translate the AgentSpeak code presented in

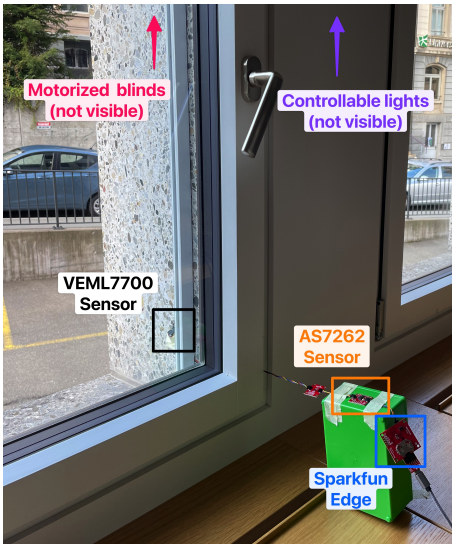


Figure 4: The light control experimental setup is composed of the sensing system and two actuators (blinds and lamp are not visible in the picture). The BDI agent ran on the Sparkfun Edge board and controlled the actuators via BLE.

Listing 1 to a C++ counterpart. This generated code is then integrated as a FreeRTOS library, together with the rest of the hardware-dependent code, and compiled for the Sparkun Edge board. Using the setup we have just described, the embedded BDI agent is deployed to control the room's light by performing a perception and reasoning cycle once every three seconds. The experiment lasted 1208 seconds, and the initial conditions were a dark room, blinds lowered, in the evening hours, with the device upside-down, indicating a dark room user preference.

The trace featuring the indoor/outdoor luminosities, as well as the agent goals and actions is shown in Fig. 5. At $t = 50$ s, the user flips the Edge board, changing the BDI agent's goal to *brighten* to the room. At this point, the external sensor indicated a luminosity of 400 Lux which surpasses the threshold for the *eco-mode* plan in Lines 15-16 from Listing 1. A few reasoning cycles later, at $t = 75$ s, the agent triggers the raising of blinds. Natural light then raises the indoor readings to around 40 lux for about 600 seconds. Since the experiment was performed during evening hours, the outdoor illuminance is decreasing. At $t = 678$ s, the indoor illuminance crosses the 30 Lux threshold, which results in the room being considered as *dark*. At this point, the BDI re-evaluates the conditions, and reasons that it now necessary to turn on the light. This action is broadcast wirelessly 3 reasoning cycles later at $t = 687$ s, and immediately raises the illuminance to almost 200 Lux. A few minutes later, at $t = 1085$ s, the user flips the board again and changes the agent's goal to *darken*. It takes several perception and reasoning cycles before the BDI agent achieves the new goal by taking the actions to turn off the light and lower the blinds, three seconds apart.

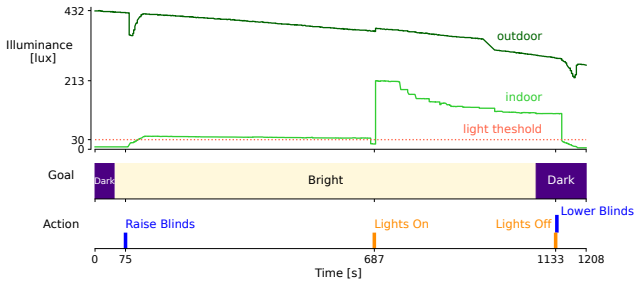


Figure 5: In this experiment, the user-controlled goal was achieved by an autonomous agent, using different plans depending on the environmental circumstances. When there is enough light outside, raising the blinds can brighten the room. Otherwise, artificial lighting must be used.

5.3 Low-Level Metrics

In the previous section, we evaluate the autonomous behavior of the BDI agent when controlling the room lights, which is related exclusively to the hardware-independent AgentSpeak program. We now turn our attention to the hardware-dependent code that runs on the Apollo3blue microcontroller.

Memory. As discussed in Sec. 3, the AgentSpeak program must first be translated to optimized C++ code. This step automatically converts the fourteen AgentSpeak lines of code to 371 lines of C++ code. Together with the embedded BDI runtime and the FreeRTOS code, the compiled binary occupies 341.9 KB of Flash and 393.2 KB of RAM memory. The memory sections text, data and BSS occupy 327888, 3440, and 389768 bytes, respectively.

Energy Characterization. To determine the energy consumption of our embedded BDI agents, we perform measurements with the RocketLogger [14]. This device measures the Sparkfun Edge's current, voltage, and GPIOS with high accuracy to perform the application's energy characterization. Using an external power 3.3 V power supply and separately measuring the sensing subsystem and the Sparkfun subsystem, we can determine the energy consumption of the perception and reasoning cycle. Each power trace included at least 100 iterations, for statistical significance. The mean and standard deviation of the energy E and execution times t are summarized in Table 1. The perception task, which reads data from two different sensors via I2C takes the longest time and largest energy to complete. The reasoning task is by far the shortest and least energy-consuming task. The communication task, which is implemented within the Cordio library has been configured to last 15 ms, during which three BLE advertisements are transmitted.

Table 1: Energy and performance evaluation, with mean and standard deviation, measured at 3.3 V using a RocketLogger.

Task	E_{mean}	E_{std}	t_{mean}	t_{std}
Perception	15.1 mJ	4.4 µJ	569.4 ms	63 µs
Reasoning	1.2 µJ	<0.1 µJ	334.3 µs	2.9 µs
Communication	47.8 µJ	<0.5 µJ	15.2 ms	31 µs

5.4 Analysis

In this section, we briefly discuss some results of our proposed low-power BDI agent. This proposal is used to deploy a BDI agent that controls the light level of a room. In the performed experiment, the agent autonomously deliberated between different action plans to achieve its given goals. The agent demonstrates reactive behavior with its actions soon after the user changes the agent's goal by flipping the board. Furthermore, its proactive behavior kept it committed to the *brighten* goal, turning on the lights as soon as the natural light was not bright enough for the room. This level of autonomy increases the intelligence of low-power sensors, allowing them to make decisions in a decentralized manner. Furthermore, these autonomous BDI agents have very quick reasoning cycles, typically lasting only hundreds of microseconds, thus incurring a relatively low energy overhead compared to just perceiving.

Limitations. Though low-power BDI agents have significant potential, there are still several limitations in our current implementation. The biggest one is that our framework supports only a single agent. For our light control application, this limits the quality of the sensing, since it is not scalable to deploy multiple sensors in larger areas. In our experiments, the indoor sensor readings are not entirely representative of the room, only the area near the bottom of the window where the sensor is located. To have more accurate estimates of the room, multiple sensors would be needed, preferably controlled by multiple agents.

Future work. We plan to alleviate current limitations by composing systems with several agents cooperating to solve problems in a decentralized approach. For that, further improvements in the communication infrastructure, such as bi-directional synchronous communication presented in [12], should be developed. We will also continue our investigation of the minimal set of BDI features we can embed in limited hardware, and explore the feasibility of running intelligent agents on batteryless platforms [8].

6 CONCLUSIONS

The preliminary findings of this work demonstrate that it is feasible and encouraging to develop autonomous agents along with low-power sensors using theories like BDI and tools like Jason. Of course, we have a scaled-down version of Jason that can be embedded in such limited hardware, but we retained the core capabilities of agents: perception, reasoning, and action (by communication). The hardware integration is still implemented in C++, which is a better language for this task, while the application strategy is implemented in a high-level language (AgentSpeak), which is better for specifying decision-making processes. In our application, the agent-based low-power sensor has the following main characteristics: prompt reaction to events in the environment (changing its behavior and goals); and proactivity towards the goal of maintaining the room in the required conditions. The cost of these features (i.e., the overhead of the BDI agent cycle) is acceptable in our setup.

ACKNOWLEDGMENTS

This work has been partially funded by the GFF-IPF Grant of the University of St.Gallen, and by the Swiss National Science Foundation, grant No. 189474 (Hypermedia Communities of People and Autonomous Agents). The authors would like to thank Lukas Meier, Ganesh Ramanathan, Naomi Stricker, and Kaveh Razavi for their contributions to this work.

REFERENCES

[1] Omar Bahri, Asmaa Mourhir, and Elpiniki I Papageorgiou. 2020. Integrating fuzzy cognitive maps and multi-agent systems for sustainable agriculture. *Euro-Mediterranean Journal for Environmental Integration* 5, 1 (2020), 1–10.

[2] Reydson Schuenck Barros, Victor Hugo Heringer, Carlos Eduardo Pantoja, Nilson Mori Lazarin, and Leonardo Machado de Moraes. 2014. An Agent-oriented Ground Vehicle's Automation using Jason Framework. In *Proceedings ICAART Conference*. SciTePress, 261–266. https://doi.org/10.5220/0004917102610266

[3] Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge. 2007. *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons.

[4] Stefan Bosse. 2014. Distributed Agent-Based Computing in Material-Embedded Sensor Network Systems With the Agent-on-Chip Architecture. *IEEE Sensors Journal* 14, 7 (2014), 2159–2170. https://doi.org/10.1109/JSEN.2014.2301938

[5] Samuel Bucheli, Daniel Kroening, Ruben Martins, and Ashutosh Natraj. 2015. From AgentSpeak to C for Safety Considerations in Unmanned Aerial Vehicles. In *Proceedings TAROS Conference*. Springer. https://doi.org/10.1007/978-3-319-22416-9_9

[6] A. Calce, P. Mojiri Forooshani, Andrew Speers, K. Watters, T. Young, and Michael R. M. Jenkin. 2013. Autonomous Aquatic Agents. In *Proceedings ICAART Conference*. SciTePress.

[7] Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu. 2009. Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 4, 3 (2009), 1–26.

[8] Andres Gomez, Lukas Sigrist, et al. 2016. Dynamic Energy Burst Scaling for Transiently Powered Systems. In *Proceedings DATE Conference*. EDA Consortium.

[9] Nilson Mori Lazarin and Carlos Eduardo Pantoja. 2015. A robotic-agent platform for embedding software agents using raspberry pi and arduino boards. *9th Software Agents, Environments and Applications School* (2015).

[10] Carlos Eduardo Pantoja, Vinicius Souza de Jesus, Fabian Manoel, and José Viterbo. 2018. A Heterogeneous Architecture for Integrating Multi-Agent Systems in AmI Systems (S). In *Proceedings SEKE Conference*. KSI Research Inc. https://doi.org/10.18293/SEKE2018-211

[11] S R R Dhiwaakar Purusothaman, Ramesh Rajesh, Karan K Bajaj, and Vineeth Vijayaraghavan. 2013. Implementation of Arduino-based multi-agent system for rural Indian microgrids. In *Proceedings ISGT Asia Conference*. 1–5.

[12] Luca Rufer, Naomi Stricker, et al. 2022. Demo abstract: Dpp3e: A harvesting-based dual processor platform for advanced indoor environmental sensing. In *Proceedings IPSN Conference)*. IEEE.

[13] Matuzalém Muller dos Santos. 2022. Programação orientada a agentes BDI em sistemas embarcados. (2022).

[14] Lukas Sigrist, Andres Gomez, Roman Lim, Stefan Lippuner, Matthias Leubin, and Lothar Thiele. 2016. RocketLogger: Mobile Power Logger for Prototyping IoT Devices: Demo Abstract. In *Proceedings SenSys Conference*. ACM.

[15] Gabriel Villarrubia, Juan F De Paz, Daniel H Iglesia, and Javier Bajo. 2017. Combining multi-agent systems and wireless sensor networks for monitoring crop irrigation. *Sensors* 17, 8 (2017), 1775.

[16] T Wanyama and B Far. 2017. Multi-Agent System for Irrigation Using Fuzzy Logic Algorithm and Open Platform Communication Data Access. *International Journal of Computer and Information Engineering* 11, 6 (2017), 690–695.