Embedding Autonomous Agents into Low-Power Wireless Sensor Networks

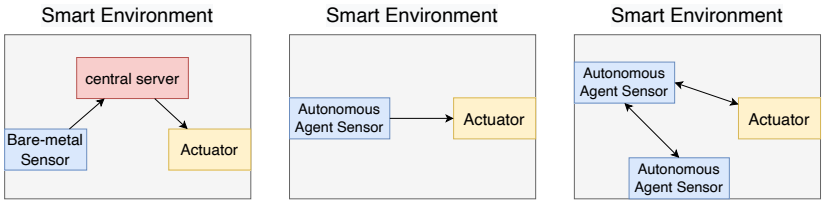
Danai Vachtsevanou[1], Jannik William[2], Matuzalém M. dos Santos[3], Maiquel de Brito[3], Jomi Fred Hübner[3], Simon Mayer[1], and Andres Gomez[4]

[1] Universität St.Gallen, Switzerland {firstname.lastname}@unisg.ch
[2] ETH Zürich, Switzerland jwilliam@student.ethz.ch
[3] Federal University of Santa Catarina, Brazil
matuzalemsantos@gmail.com, {maiquel.b,jomi.hubner}@ufsc.br
[4] TU Braunschweig, Germany gomez@ida.ing.tu-braunschweig.de

Abstract. Low-power sensors are increasingly becoming available, equipped with more energy-efficient processing and networking capabilities. Still, in order to accommodate the independent deployment and intermittent availability of such constrained devices, engineers often manually reconfigure system behavior for integrating sensors and actuators into complex and context-aware systems. The Multi-Agent Systems paradigm enables engineering systems where components can be deployed more independently and operate towards achieving their design objectives. In this process, they act autonomously and interact with others to perform context-aware decision-making without human intervention at run time. In this paper, we present autonomous agents implemented as low-power nodes that perceive and act in a shared environment through sensors and actuators. The autonomous agents on these constrained devices locally reason and act on the environment, and wirelessly interact with each other to share knowledge and enable more context-aware system behavior. The capabilities of our low-power autonomous nodes are demonstrated in a light-control scenario with two Belief-Desire-Intention agents. Our experiments demonstrate that running autonomous and social agents in low-power platforms incurs little overhead, indicating their feasibility.

1 Introduction

Research on autonomous agents and Multi-Agent Systems (MAS) provides solutions to engineering complex systems, where components – modeled as autonomous agents – can flexibly perceive and act in distributed, open, and heterogeneous environments. Such environments are prominent in the Internet of Things (IoT), where engineers integrate low-power devices into complex systems while catering to the independent deployment, intermittent availability, and heterogeneity of components. IoT engineers typically perform integration and maintenance manually. On the other hand, exploiting MAS methods for programming IoT applications offers the flexibility to achieve new user goals



(a) Centralized approach (b) Single-agent approach (c) Multi-agent approach

Fig. 1: Low-power sensors are typically employed in a centralized approach, where decisions are made by a powerful centralized server. Individual autonomous agents have been used to increase the intelligence of low-power sensors and allow local decision-making to directly control actuators. Scaling such single-agent systems to cooperative multi-agent systems introduces new challenges that we address in this paper.

and handle unexpected events. It also opens up possibilities for examining hybrid goal-driven and event-driven use cases, for example, where goal delegation allows the system to remain robust in the event that autonomous components become unavailable, possibly due to resource constraints.

The Belief-Desire-Intention (BDI) model [5] has been proposed to enable the engineering of agents whose procedural knowledge is defined in terms of beliefs, desires, and intentions, and where agents can flexibly balance between proactive and reactive behavior [13]. Additionally, agent procedural knowledge may include communicative actions, influenced by speech-act theory [16]. These actions enable agents to interact with each other, for example, to change another agent's beliefs. However, our ability to deploy BDI agents in IoT applications for leveraging their desirable properties is hindered by the limited resources of low-power devices, which constrain the system's reasoning cycle and therefore reduce the agents' proactivity, reactivity, and social ability [2]. At the same time, the programming tools (language and run-time environments) that are typically used to develop BDI agents for desktop computers or servers are not well-suited for low-power devices [10]. As a result, low-power sensors are typically used only in conjunction with a powerful centralized server, as depicted in Fig. 1a. Moving towards a multi-agent approach (Fig. 1c), where these sensors more flexibly interact and coordinate with each other – that is, where they are treated more like autonomous entities than dependent nodes – presents new challenges in providing tooling and a system architecture that meet the limitations of highly resource-constrained devices.

This paper explores the potential of using autonomous agent technologies for programming low-power devices in the context of the IoT, with the overarching research aim to isolate the minimal set of BDI properties an agent must have in order to function in low-power devices. This minimal set has been initially proposed in [18] (Fig. 1b). We extend that system to a network of BDI agents and propose a fitting system architecture. Our results show that this architec-

ture enables programming and running social BDI agents on low-power wireless networks while retaining small overhead with respect to memory and energy consumption.

Our proposed system architecture is implemented in the context of a (simple) light-control application where BDI agents that are deployed on low-power devices interact with each other to sense and control the environment and manage illuminance in a room. Our agents is deployed on the DPP3e [14], an ultra-low-power platform that has been custom-designed for next-generation IoT networks. DPP3e includes not only advanced sensing and processing capabilities but also a specialized communication protocol called *Low-power Wireless Bus* (LWB) [7], which uses concurrent transmissions to enable energy-efficient all-to-all communication. This allows every agent in the network to efficiently send *and* receive information, enabling their social behavior.

2 System Architecture

In [15], a lightweight single-agent framework is suggested, which is based on a simplified variant of Jason [3] (an interpreter for an extended version of AgentSpeak). The high-level workflow of the framework is to program the deliberation of the agent using the features of AgentSpeak, and implement perception and action functions together with other hardware-dependent code (like a real-time OS, sensor drivers, and hardware interrupts) in C/C++. The main objective of our 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; in the following, we discuss our proposed modifications to that system's Translation Engine (Section Section 2.1) and the BDI Runtime (Section Section 2.2). On the basis of these changes, we then introduce the concept of *Decision Cycles* to better balance agent responsiveness and resource requirements (Section 2.3).

2.1 Translation Engine

The Translation Engine presented in [15] converts AgentSpeak programs into optimized C++ code that represents the initial beliefs, goals, and plans of the agent. It is able to connect actions and perception functions available in the hardware with the naming convention used in the AgentSpeak program. The resulting code can be compiled and deployed to a microcontroller. In the process, the translation engine takes special care to optimize the memory and run-time performance of the embedded agent. For instance, while creating C++ classes to represent AgentSpeak beliefs and actions, the engine converts belief and action labels (e.g., `is_dark`) to corresponding identifiers that are 8-bit integer numbers, limiting the agents to 256 distinct beliefs and actions, respectively. 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 were

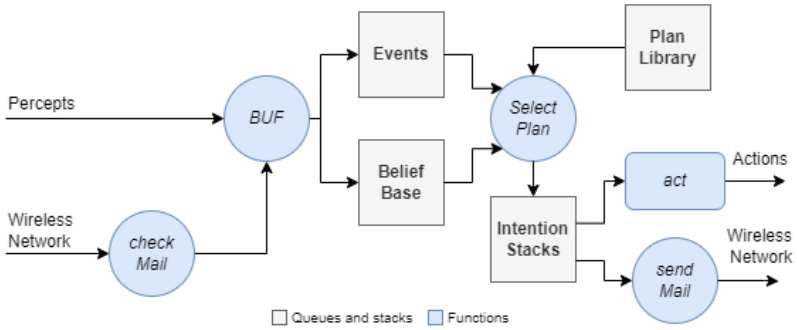


Fig. 2: The practical reasoning cycle of the Embedded-BDI runtime is derived from a simplified version of the Jason interpreter. For multi-agent programs, the runtime must be able to receive messages from the wireless networks and combine any external beliefs with its own perceptions.

carefully selected to still allow us to develop reasonably complex agents (considering low-power embedded devices).

In this work, we make use of the translation engine that has undergone a set of modifications. The original engine translates propositions based on their first appearance in the AgentSpeak program of an agent. This approach is effective with single agents, as these use the propositions only for internal operations. However, if multiple agents communicate with each other using their internal propositions, it can lead to misunderstandings, as the same proposition might be mapped to different identifiers in different agents (or vice versa).

The adopted solution is the use of hashed proposition names. As hashes produce reproducible output given a fixed input, the same proposition names get translated equally in every agent program. This solution does not depend on a global state or require all agents to be compiled at the same time. However, special considerations on hash size and function have to be made to minimize the probability of hash collisions. If two propositions do produce the same output, this would happen at compile time, and would be easily detected.

2.2 BDI Runtime

The second core component of the framework is a BDI Runtime that implements a simplified reasoning cycle from the Jason interpreter. Fig. 2 summarizes our simplified reasoning cycle, which is run in an infinite loop by the microcontrollers: At the beginning of each reasoning cycle, the agent updates its belief base with percepts from the environment. These percepts are generated using sensor values and the belief update function (BUF) component, coded in C++. We consider that all possible beliefs are known in the translation phase and thus the belief base is a fixed size mapping between beliefs and boolean values. For instance,

BUF verifies whether a light sensor value is smaller than a threshold and, if so, the belief `is_dark` is mapped to true, otherwise it is mapped to false.

The belief base may also be updated due to messages received through the wireless sensor network. Messages encode speech acts [16] that convey a *content* that acts on the mental states of the receiver based on an *illocutionary force*, i.e. the intention of the sender. In our previous work, we did not support the use of illocutionary forces. Now, we employ the illocutionary forces `tell` and `untell` – used with the intention of changing the receiver's beliefs. For example, an agent can send the content `sunny` in a `tell` or `untell` message so that the receiver believes that `sunny` is true or false, respectively.

Any change to the belief base due to percepts or received messages generates an event that is added to the event queue via the BUF component. Then, an event is selected from the event queue, and all applicable plans are selected, i.e. plans from the agent's plan library that are relevant to the event, and most probable to succeed based on the agent's current beliefs. The agent then selects an active intention and executes a single instruction. The instruction could include adding or deleting a belief or achievement-goal, or executing an action (including broadcasting actions).

2.3 From Reasoning Cycle to Decision Cycle

When agents are deployed on unconstrained devices, the reasoning cycle is usually executed at a high frequency. In case the agent has long term goals, it usually keeps executing reasoning cycles, even after an action was decided. We argue that this becomes a problem with respect to computational overhead in resource-constrained devices. Rather, it would be beneficial to execute the reasoning cycle until it has decided on an action and then suspend execution for a certain time. This lets the agent be responsive, once there is information available to support a decision. At the same time, this mechanism saves resources because empty reasoning cycles are avoided.

A possible solution is to suspend the execution of reasoning cycles if there are no events pending and if all intentions and all messages have been processed. There are however some programming patterns (e.g., maintenance goals [3]) that implies that the set of intentions is never empty. In AgentSpeak, this is implemented with the following pattern:

```
+!goal : ctxt1 <- act1 ; !!goal.
+!goal : ctxt2 <- act2 ; !!goal.
```

The chosen plan handles the goal by performing an action (here, `act1`), and then the goal remains relevant through the goal addition (here, `!!goal`) at the end of the plan. Thus the event and intention bases are never empty at the same time. This implies that the condition for the agent to suspend reasoning never happens. The proposed solution is based on *decision cycles*, in which the agent decides the actions to perform and eventually executes them. We typically want to execute actions caused by maintenance goals at most once in a decision cycle. We propose to only check the event base for *external* events, i.e., events not related to achievement goals. A decision cycle is thus composed of sensing, several

Listing 1: The AgentSpeak program of an agent that maintains the illuminance conditions in a room (cf. [18]).

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

reasoning cycles until there are no more external events, and (communicative) action. There needs to be more research done on this topic, as it is only tested with our use case scenario.

3 Light Control in a Building Automation MAS

To ground our approach, we present an application scenario where a MAS features two BDI agents (Fig. 1c): An `indoor_agent` perceives the illuminance conditions within a building (percepts: `{bright,dark}_inside`), and adjusts the conditions by acting on a lamp (actions: `{turn_on,turn_off}_lights`) or on a set of blinds (actions: `{raise,lower}_blinds`). The agent also perceives the illuminance preferences of room occupants (percepts: `user_turn_{on,off}`). Additionally, an `outdoor_agent` perceives the outdoor illuminance conditions (percepts: `{sunny,cloudy,night}_outside`).

Lst. 1 shows the program of the indoor agent – formerly presented in [18] as part of a single-agent application – which acts on the environment based on users' preferences. In contrast to previous work, the agent now only perceives the building's interior, with external weather information shared by the outdoor agent through agent-to-agent interaction.

The scenario begins with both agents launched. The indoor agent observes the indoor illuminance conditions, while the outdoor agent is initially in sleep mode. Upon perceiving the occupant's preference for high illuminance conditions, the indoor agent continuously strives to maintain a high illuminance level (lines 1-5) and will, by default, turn on the lights to increase brightness if the room is dark (lines 12 and 13). Accordingly, if the indoor agent perceives that the occupant now prefers low illuminance conditions in the room,

it will adopt the goal of maintaining such conditions (line 14). Therefore, if it is bright inside the room, the agent will turn off the lights, and additionally lower the blinds to eliminate any possible source of illumination (lines 16-18).

In the second scenario phase, the outdoor agent starts observing the outdoor environment and broadcasts messages to synchronize other agents' beliefs with the current weather conditions. The behavior of the agent is based on the AgentSpeak program shown in Lst. 2. The indoor agent is still committed to maintaining optimal illuminance conditions, however, now the agent's decision-making is more contextual since it takes into consideration the communicated weather conditions. For instance, when brightening the room, the agent reasons that a more eco-friendly option is available when the weather is sunny (Lst. 1, line 7), and decides to raise the blinds (Lst. 1, line 11) instead of turning on the lights.

```
1  !communicate. // initial goal
2  +!communicate : sunny_outside <-
3      .broadcast(tell, sunny_outside);
4      .broadcast(untell, cloudy_outside);
5      .broadcast(untell, night_outside);
6      !!communicate.
7  +!communicate : cloudy_outside <-
8      .broadcast(untell, sunny_outside);
9      .broadcast(tell, cloudy_outside);
10     .broadcast(untell, night_outside);
11     !!communicate.
12 +!communicate : night_outside <-
13     .broadcast(untell, sunny_outside);
14     .broadcast(untell, cloudy_outside);
15     .broadcast(tell, night_outside);
16     !!communicate.
```

Listing 2: The AgentSpeak program of an agent that perceives and broadcasts information about the weather conditions.

4 Experimental Evaluation

In this section, we evaluate the proposed BDI framework (labeled "Multi-Agent Version" hereafter) using the previously described light control application from [18] (labeled "Single-Agent Version"). The experimental setup 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.

4.1 Experimental Set-Up

In the Single-Agent Version of the experiment, we have used the commercial Edge platform by Sparkfun, which is based on the Ambiq Apollo3 Blue microcontroller. We integrated the embedded-BDI framework [15] with FreeRTOS and utilized ARM's Cordio BLE stack for asynchronous communication. Whenever an agent wanted to perform an action, it would transmit a Bluetooth advertisement and trigger different actuators. In this work, we use the DPP3e platform [14], which features a separation of application and communication processors. This allows for the independent development of the different domains, and can also save valuable memory in the application domain since it no longer needs to maintain a communication stack.

The DPP3e's application domain features an Ambiq Apollo 3 Blue Plus. Instead of using BLE advertisements, the DPP3e's communication domain uses the LWB protocol, enabling efficient all-to-all communication. Two DPP3e's are used to form a two-agent system, as described in Section 3.

One agent is located indoors, the other outdoors; each is equipped with a light sensor. The outdoor agent wirelessly transmits the values of the `{sunny,cloudy,night}_outside` beliefs to the indoor agent. A third DPP3e board is connected to a Raspberry Pi and acts as a gateway, limited to forwarding commands to the lights and blinds via the wired network. The UART output of all boards is logged for debugging and post-processing. Fig. 4 shows the set-up for the multi-agent experiment.

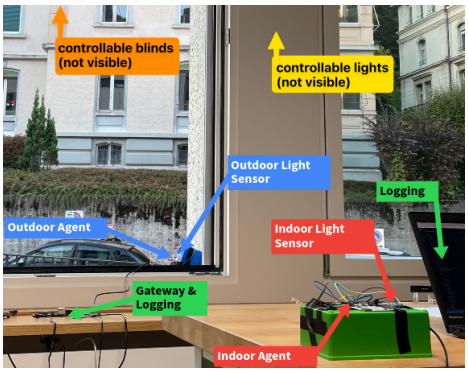


Fig 4: A user can set the indoor agent's goal either to brighten or darken a room. Depending on the information broadcast by the outdoor agent, the indoor agent can select among different actuators to achieve its goal.

The experiment is conducted in the evening when the outdoor illuminance is decreasing. The following light thresholds have been set: sunny/cloudy threshold is 4000 lux, cloudy/night threshold is 500 lux, and the indoor threshold: 19 lux. The decision cycle period was set to 15 s, which aligned with the communication period. These were however not synchronized (worst-case, it could be, that communication happens right before the decision cycle, leading to almost 15 s delay, until an action is sent).

4.2 Experimental Results

The measurements of the energy consumption were performed by an Otii Arc and are summarized in Table 1. The reasoning part corresponds to a full decision cycle including message fetching from the wireless communication domain. The energy and execution time values thus are significantly higher. The task that consumed the most energy is sensing. This is also significantly higher than in the single-agent experiment since longer integration times are chosen. Another significant energy consumer is the wireless communication, which now requires almost three orders of magnitude more energy, although they cannot be directly compared, as here, the communication is bidirectional.

The original embedded BDI framework [15] was explicitly developed for resource-constrained devices. The AgentSpeak functionality is fundamentally the same in both versions, so the translation yields the same C++ code size. The single-agent version implements the entire sense-reason-transmit functionally on a single chip (Apollo3 Blue), using 748 KB of read-only memory. The multi-agent

version is run on the DPP3e's application domain (Apollo3 BluePlus), implementing only the sensing-reason functionality on 311 KB of read-only memory.

Smart Light Control. The experiment lasted for 765 s. The initial conditions were, that the lights were off and the blinds down and the user preference was to brighten the room. This initially triggered the action of raising the blinds. The indoor illuminance is actually below the threshold, but the sunny condition prevents the agent to turn on the light. After 169 s, the outdoor conditions indicate, that it is cloudy. Since the outdoor agent has a near worst-case alignment of the decision cycle and communication cycle, it needs 14 s, until the beliefs are sent over the LWB. The indoor agent requires two full decision cycles to come to the conclusion to turn on the lights, meaning, the breaking condition got triggered to our disadvantage. This results in a total delay of 44 s from acquiring the belief, until the action gets triggered. There were two cycles required, because after acquiring the belief `cloudy_outside`, `+!brighten` was already processed, before `standard_mode_available` got added to the belief base (so there was no context matching the belief base for a plan for `+!brighten`). Therefore, the action gets triggered only in the next decision cycle. At 513 s, the user preference changed to darken. As `+user_turn_off` executes `!!preserve_dark`, the breaking condition gets triggered before a plan for `+!preserve_dark` can be executed. This results in a delay of one decision cycle, until the actions get triggered, resulting in a 15 s delay from the LWB communication period.

4.3 Analysis

The experiments showed, that it is possible to implement a low-power BDI agent on an embedded device, which can perform a basic home automation task. The agents show reactive and proactive behavior, where it reacts to user input and

Single-Agent Version					
Task	E_{mean}	E_{std_dev}	t_{mean}	t_{std_dev}	# samples
Sensing	15.1 mJ	4.4 µJ	569.4 ms	63 µs	114
Reasoning Cycle	1.2 µJ	<0.1 µJ	334.3 µs	2.9 µs	114
Communication	47.8 µJ	<0.1 µJ	15.2 ms	31 µs	147
Multi-Agent Version					
Task	E_{mean}	E_{std_dev}	t_{mean}	t_{std_dev}	# samples
Sensing	39.7 mJ	160.7 µJ	1113.4 ms	4.5 ms	190
Decision Cycle	54.8 µJ	17.5 µJ	1.609 ms	490.2 µs	115
Communication	25.9 mJ	4.0 mJ	419.9 ms	64.6 ms	115

Table 1: The multi-agents version requires longer mean execution times (t_{mean}) and higher mean energies (E_{mean}) because it tasks are more complex, e.g. sensing with dynamic integration times, a variable number of reasoning cycles, and full bidirectional wireless communication.

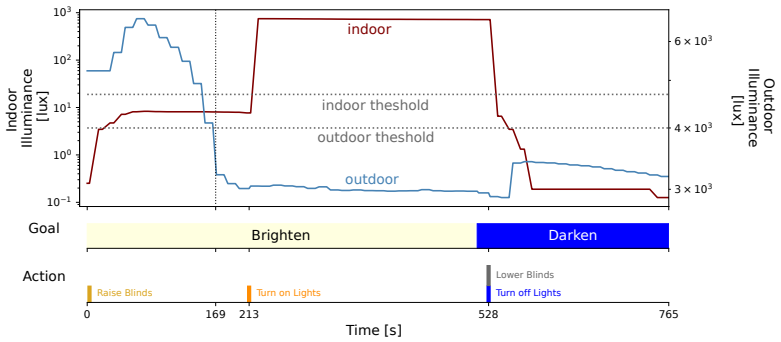


Fig 5: Upon adopting the `brighten` goal, the indoor agent first raises the blinds to increase the illuminance. Once the low threshold for natural light is crosses, the agent decides to turn on the artificial lights to maintain the `brighten` goal. After the user chooses to darken the room, both actuators are turned off.

proactively adapt to the environment. Compared to the traditional tasks of sensing and communication, reasoning takes up very little time and energy, so the cost of adding intelligence and autonomy to a simple system is very minimal. The results also showed the differences between the single and multi-agent systems. The single agent had practically no communication delay, whereas there was a worst-case delay of 15 s in the multi-agent system. There is however the advantage of the meshed communication in LWB, which BLE does not allow efficiently (BLE is point-to-point, which is not directly compatible with MAS). The computational delay could however be greatly improved, because of the introduction of the *full decision cycle*. This allows to do a "burst" of reasoning cycles directly one after the other and thus directly decide on an action.

Though low-power BDI agents have significant potential, there are still several limitations in our current implementation. Developing applications for low-power multi-agent systems still requires significant manual effort to synchronize BDI functions with hardware-dependent code and communication stacks. Furthermore, the specific characteristics of the communication protocol can limit the performance of the overall system, as shown by the latencies stemming from periodic communication.

5 Related Research

Developing efficient autonomous agents for low-power devices is crucial for applying BDI-based agent technology in the IoT domain. Previous research bridges the resource gap between low-power platforms and the resources required by agents by connecting low-power devices to more powerful ones [9, 11, 1, 17], but this approach may not be energy-efficient for many IoT applications. Our approach focuses on deploying agents exclusively on constrained devices.

Other approaches deploy the agents solely on constrained platforms. Agents are programmed in C++ [12], assembly-like languages [8], or using on-chip synthesis of finite-state machine models [4]. These agents have social abilities through wireless communication (e.g. [8, 4]), but are limited with respect to their autonomy, proactivity, and reactivity. Our aim is to retain the expressiveness of an agent programming language and the high-level constructs of the BDI model to preserve agent properties in IoT applications.

Towards combining the expressiveness of an agent programming language and the energy efficiency of a more low-level language, Bucheli et al. propose translating BDI specifications programmed in AgentSpeak [3] to C programs for low-power devices [6]. However, their translation is application-specific, while our work employs a general-purpose framework. Also, while their work is evaluated in simulated environments [6], we propose evaluating our framework in real applications with explicit energy and communication concerns.

Our previous work [18] focused on running AgentSpeak programs for single-agent systems with isolated domain vocabularies, which limits any social interaction between agents. While the feasibility of autonomous agents in wireless sensors was demonstrated, agents can only broadcast asynchronous messages to actuators. Due to its asynchronous nature, an agent can broadcast a message to an actuator as soon as it decides on an action. While this minimizes the latency between decision and actuation, unidirectional communication is not feasible for multi-agent systems. In this work, we address some challenges to enable multiple agents to run on a *network* of low-power wireless sensors. We extend the simplified embedded BDI reasoning cycle to include belief updates from/to the wireless network, enabling agents to disseminate their beliefs using a shared domain vocabulary.

6 Conclusions

In this paper, we presented a general-purpose BDI multi-agent programming framework that runs on low-power devices for wireless sensor networks. By integrating an energy-efficient all-to-all wireless protocol, we can develop social BDI agents capable of sharing beliefs in a scalable and efficient manner. We have implemented a real-world light control application on a state-of-the-art hardware platform and shown the feasibility of low-power multi-agent systems. Our experimental findings prove the low overhead of a constrained set of BDI features in Jason for low-power sensors while preserving the core properties of autonomous agents: Agents are capable of perceiving and acting on their environment, deliberating about goals and making decisions while balancing between their proactive and reactive behavior, and, finally, interacting and collaborating with other agents by exchanging messages.

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

References

1. Bahri, O., Mourhir, A., Papageorgiou, E.I.: Integrating fuzzy cognitive maps and multi-agent systems for sustainable agriculture. Euro-Mediterranean Journal for Environmental Integration **5**(1), 1–10 (2020)
2. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with jacamo. Science of Computer Programming **78**(6), 747–761 (2013)
3. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason. John Wiley & Sons (2007)
4. Bosse, S.: Distributed agent-based computing in material-embedded sensor network systems with the agent-on-chip architecture. IEEE Sensors Journal **14**(7), 2159–2170 (2014)
5. Bratman, M.E., Israel, D.J., Pollack, M.E.: Plans and resource-bounded practical reasoning. Computational Intelligence **4**, 349–355 (1988)
6. Bucheli, S., Kroening, D., Martins, R., Natraj, A.: From agentspeak to C for safety considerations in unmanned aerial vehicles. In: Proceedings TAROS Conference. Springer (2015)
7. Ferrari, F., Zimmerling, M., Mottola, L., Thiele, L.: Low-power wireless bus. In: Proceedings of ACM SenSys Conf. (2012)
8. Fok, C.L., Roman, G.C., Lu, C.: Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. ACM Transactions on Autonomous and Adaptive Systems (TAAS) **4**(3), 1–26 (2009)
9. Lazarin, N.M., Pantoja, C.E.: A robotic-agent platform for embedding software agents using raspberry pi and arduino boards. 9th Software Agents, Environments and Applications School (2015)
10. O'Hare, G.M.P., et al.: Embedding agents within ambient intelligent applications. In: Agents and Ambient Intelligence - Achievements and Challenges in the Intersection of Agent Technology and Ambient Intelligence. IOS Press (2012)
11. Pantoja, C.E., de Jesus, V.S., Manoel, F., Viterbo, J.: A heterogeneous architecture for integrating multi-agent systems in ami systems (S). In: Proceedings SEKE Conference. KSI Research Inc. (2018)
12. Purusothaman, S.R.R.D., Rajesh, R., Bajaj, K.K., Vijayaraghavan, V.: Implementation of arduino-based multi-agent system for rural indian microgrids. In: Proceedings ISGT Asia Conference. pp. 1–5 (2013)
13. Rao, A.S., Georgeff, M.P.: BDI agents: From theory to practice. In: Proceedings of the First International Conference on Multiagent Systems (1995)
14. Rufer, L., Stricker, N., et al.: Demo abstract: Dpp3e: A harvesting-based dual processor platform for advanced indoor environmental sensing. In: Proceedings IPSN Conference. IEEE (2022)
15. Santos, M.M.d.: Programação orientada a agentes BDI em sistemas embarcados. Master's thesis, Universidade Federal de Santa Catarina (2022)
16. Searle, J.R., Searle, J.R.: Speech acts: An essay in the philosophy of language, vol. 626. Cambridge university press (1969)
17. Wanyama, T., Far, B.: Multi-agent system for irrigation using fuzzy logic algorithm and open platform communication data access. International Journal of Computer and Information Engineering **11**(6), 690–695 (2017)
18. William, J., et al.: Increasing the intelligence of low-power sensors with autonomous agents. In: Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems. pp. 994–999 (2022)