Directions for implementing BDI agents in embedded systems with limited hardware resources

Matuzalem Muller dos Santos[1], **Jomi Fred Hübner**[1], **Maiquel de Brito**[2]

[1]Dep. de Automação e Sistemas
Universidade Federal de Santa Catarina (UFSC) – Florianópolis – SC – Brazil

[2]Dep. de Engenharia de Controle, Automação e Computação
Universidade Federal de Santa Catarina (UFSC) – Blumenau – SC – Brazil

matuzalem.m@posgrad.ufsc.br, jomi.hubner@ufsc.br, maiquel.b@ufsc.br

Abstract. *The characteristics of autonomy, reactivity, and proactivity, commonly present in embedded systems used in cyber-physical systems, are often similar to the ones of agents. Although the usage of agents in these scenarios would be beneficial, the lack of tools to implement agents in hardware commonly used in low-cost embedded systems is one of the reasons that prevent embedded agents from becoming a reality. This paper discusses some implementation challenges and design considerations required to develop a framework that allows implementing BDI agents in embedded systems.*

1. Introduction

Over the past decade, research in the field of Cyber-Physical Systems (CPS) has made significant advancements in designing and implementing complex systems using low-cost hardware. However, while the cost of hardware commonly used in embedded systems has decreased and its processing capabilities have increased, it is still a challenge to implement complex systems that require features that are typical of agents, such as autonomy, proactivity, reactivity, and social ability [Aliyuda 2016].

Common development tools used for programming agents, such as JADE and Jason, are Java-based and require more resources than is available in the hardware commonly used for embedded systems. Given this scenario, we are motivated to explore the implementation of tools that promote agents in embedded systems. Due to its popularity, the BDI architecture is an interesting agent paradigm to be considered when conceiving this tool, as both reactive and proactive agents can be implemented using this architecture.

This work provides directions for implementing BDI agents in embedded systems with limited hardware resources. The following section provides background information on CPS, embedded systems, agents, and the BDI architecture, followed by a review of related work and our considerations on implementing BDI agents in embedded systems.

2. Background

This section presents the keys concepts introduced in this article. First, cyber-physical and embedded systems are introduced, which is later be associated with BDI agents, AgentSpeak and Jason.

2.1. Cyber-Physical and Embedded Systems

Cyber-physical systems refer to the confluence of cyber and physical systems, such as embedded systems, distributed sensor systems, and control systems [Rajkumar et al. 2010]. CPS focuses on the *interaction* between its composing systems, allowing the creation of adaptive and predictive systems for enhanced performance [Park et al. 2012]. The computational core of CPS is embedded systems, often distributed [Horvath and Gerritsen 2012].

The field of embedded systems is wide and varied, and it is thus difficult to provide a precise definition for the term. According to [Berger 2002], embedded systems are applied computer systems that run on custom hardware, unlike personal computers that can be used for general computing. Examples of applications that contain embedded systems are calculators, digital watches, heating systems, and televisions.

Since embedded systems are often designed for scenarios with resource-constrained characteristics, computational resources and energy are often limited. According to [Marwedel 2010], one can consider the following metrics to evaluate resource efficiency: energy consumed, usage of execution time, code size, weight, and cost. To build an efficient embedded system, it is expected that the energy consumed, cost, code size, and weight are minimized, while usage of execution time is maximized. Some techniques for efficient resource usage are static memory allocation, out-of-memory running avoidance, and real-time operating system (RTOS) usage to follow real-time restrictions.

2.2. BDI Agents

Due to the multiple applications and scenarios in which agents are used, multiple definitions of the term are available in the literature. [Wooldridge 2009] states that "an agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives". Although there are multiple definitions of the term agent, it is agreed that agents usually present the following common characteristics [Wooldridge and Jennings 1995]:

- Autonomy: agents control their actions and goals, without human intervention;
- Social Ability: agents can interact between themselves over an agent-communication language;
- Reactivity: agents are sensible to the environment and can change their actions based on these perceptions;
- Proactivity: agents exhibit goal-directed behavior, which means that they do not solely respond to changes in the environment.

The Belief-Desire-Intention architecture, based on Bratman's behavioral model [Bratman 1987], is a popular agent architecture. In the BDI architecture, actions taken by an agent result from the agent's beliefs, desires, and intentions [Bordini et al. 2009].

***B**eliefs* are information the agent has about the world. Ambient temperature and time are examples of information that can be represented as beliefs. Note that this information can be out of date or inaccurate [Bordini et al. 2007a]. ***D**esires* represent the state of affairs that the agent might like to accomplish. For example, an agent embedded in an air conditioning may desire the ambient temperature to be under a specific

value, but that does not necessarily mean that the agent is taking actions to fulfill that desire. Lastly, ***I**ntentions* are the state of affairs that the agent has decided to work towards [Bordini et al. 2007a]. In practical terms, these represent the course of actions under execution to achieve the agent's desires [Mascardi et al. 2005].

Beliefs, *Desires*, and *Intentions* are constructs used in the decision model known as *practical reasoning*. Practical reasoning consists of two activities: *deliberation* and *means-end reasoning*. During deliberation, an agent decides what states of affairs it commits to achieve (i.e., which desires are "promoted" to intentions). On means-end reasoning, the agent decides how to achieve these states of affairs [Bordini et al. 2007b]. Practical reasoning is represented in Figure 1.

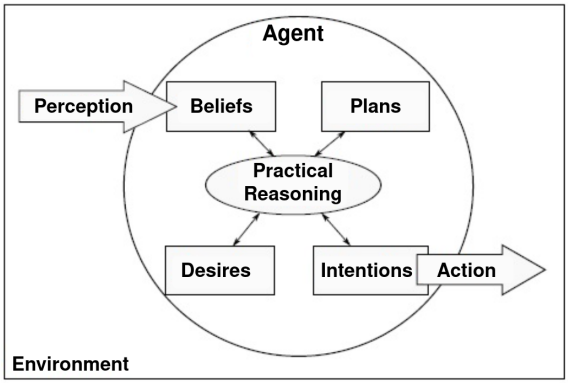

Figure 1. Representation of Practical Reasoning
[Santos 2015]

To summarize Figure 1: the agent perceives the environment and updates its beliefs. After updating its beliefs, the agent starts the deliberation process to determine which objectives it wishes to achieve, considering its current beliefs, desires, plans, and intentions. Once an objective is chosen, a plan of action is selected to fulfill that objective via the means-ends reasoning process. Finally, the agent acts in the environment and later repeats the cycle to re-evaluate its objectives and the state of the world.

2.3. AgentSpeak & Jason

Agent-Oriented Programming (AOP) refers to a paradigm of programming computer systems using the abstraction of agents. In particular, several languages for AOP are based on agent with "mental states", which in the BDI architecture are represented by beliefs, desires, and intentions. AgentSpeak is a popular programming language for AOP, but other languages can also be used, such as the Procedural Reasoning System (PRS), dMARS, and Jadex.

AgentSpeak has been conceived by [Rao 1996] and can be used to implement deliberative agents. The main constructors of the AgentSpeak language are beliefs, goals, and plans. Beliefs represent information the agent has about the world. Goals express the properties of the states of the world that the agent wishes to bring about. Lastly, plans define how an agent can act to reach goals [Bordini et al. 2007b].

Along with the language constructors, AgentSpeak also relies on other components and data structures for agent reasoning. These components are the belief base, plan library, event and intention queues [Santos 2015].

The belief base stores the agent beliefs, which are updated by perceiving the environment. By perceiving the environment and updating its beliefs, events are generated, which are queued for processing and can lead to the execution of a plan available in the plan library. An intention is represented by the execution of applicable plans for desires. Intentions are stored in a data structure: the intention queue.

One of the main differences between BDI programming and traditional programming methodologies is that BDI agents do not end their execution after completing a plan or an action. Instead, BDI agents are always running, perceiving the environment, and (re)evaluating their goals.

To run AgentSpeak, it is necessary an interpreter (or compiler) that supports the language syntax and reasoning cycle. Jason is a popular AgentSpeak interpreter that provides a platform for developing multi-agent systems with additional programming capabilities, extending the AgentSpeak syntax with annotations, internal actions, and others [Bordini et al. 2007a].

3. Related Work

Research in the field of embedded agents is rich and diverse. When searching for the term "embedded agents" in Google Scholar, and filtering the results for the past 5 years (from 2016 to 2021), about 314,000 results are found.

In [Barros et al. 2014] and [Lazarin and Pantoja 2015], the authors use Raspberry Pi and Arduino boards for agent reasoning and acting/perceiving the environment, respectively. Although the projects succeed in proposing a feasible way to embed agents, it falls short in offering a homogeneous system where the agent can reason, perceive, and act in the environment in a single computational platform.

Another attempt to propose using agents in embedded systems is the work of [Aliyuda 2016]. Despite the thorough analysis of how multi-agent architectures can tackle CPS design challenges, the author does not implement embedded agents to address the scenarios described. The work proposes to use the JADE framework to develop agents to address the design issues of CPS. However, the JADE framework is not compatible with most hardware used for embedded systems and would hardly suffice the requirements of running agents in common low-cost hardware.

Lastly, [Bucheli et al. 2015] implements an AgentSpeak translator to embed agents in Unmanned Aerial Vehicles (UAV). The AgentSpeak translator implemented aims to allow programming agents for UAVs with specific characteristics. Therefore, its usage was not tested and validated for common embedded hardware and other applications, making multi-platform and multi-application compatibility harder to achieve.

Due to the hardware constraints presented by common embedded systems, using the tool set usually applied to implement BDI agents becomes a challenge. Frameworks such as JADE and Jason are Java-based, which requires large memory and processing capabilities to run the Java Virtual Machine (JVM) and the agent code on top of it.

To address this challenge, some propose to run heterogeneous systems where agent reasoning and acting in the environment are split between two computational systems. However, this increases the cost of the system, which should be minimized.

4. Agent Design for Embedded Systems

Given the existing solutions, we propose directions for developing a tool that allows the implementation and execution of BDI agents in hardware commonly used in embedded systems, such as Arduino UNO and ESP32 boards. Based on our research, it is desirable that the tool is implemented in a programming language commonly used in embedded hardware, allowing efficient resource management and code optimization. In addition, it is desired that the programmer has control over the size of the internal data structures of the agent, allowing relative control over memory usage and agent characteristics.

This section describes the initial steps towards developing a framework to implement and run BDI agents in the constrained hardware used by embedded systems. These steps include the development methodology, the requirements of the framework, and the requirements of the implemented agents.

4.1. Development Methodology

It is desired that the reasoning cycle and basic features of the framework are based on existing programming languages and interpreters, as the abstractions provided by those can give a sound basis for designing the framework and deciding how features can be implemented. Due to its popularity, we indicate the AgentSpeak programming language and the Jason interpreter as the basis to design the software architecture and define the features of the framework.

In addition, we also recommended the development methodology of the framework to be incremental, where less-complex features are implemented first to validate the framework usage with common embedded hardware. Although the hardware resources required by the framework can only be validated during its implementation, we indicate that the range of hardware between an Arduino UNO and an ESP32 are good target platforms to embed agents due to their processing power, low cost, and popularity. Once confirmed that the target platforms support specific features, these can be implemented and improved, allowing more control over the development process, features implemented, and overall compatibility.

4.2. Requirements of the Framework

The framework should not target a specific hardware platform. Instead, it should be compatible with platforms that provide the minimum requirements of the framework; this way, agents can be implemented in heterogeneous hardware and used for various applications.

Although Jason provides a good starting point for designing the framework, it is desired that, unlike Jason, which is Java-based, the framework is implemented using a language commonly used in the development of embedded systems. This will ensure that low-level abstractions can be easily implemented and hardware is managed wisely. Good candidates are the C and C++ programming languages.

The main reasoning cycle of the agent, where beliefs are updated, events are processed, goals are reviewed, and intentions are adopted, must be kept, as this is an essential feature of BDI agents and what distinguishes them from other computational systems.

Jason employs some elements of logic programming on the agents' knowledge representation and reasoning. These elements include predicates and unification. Due

to the incremental development methodology, it is recommended that the framework first provides support for simpler instructions, such as propositions. Support for predicates can be added later – the memory management and processing load of predicates, which can assume a multiple ranges of types of values, is a big challenge for embedded platforms and can lead to potential performance issues or incompatibility of the framework with the target platforms.

Since the unification algorithm in plans and rules can take a large amount of processing [Junior 2015, Miller and Esfandiari 2021], it is recommended that the unification operators supported by the framework are simple, as complex algorithms will use too much processing time to evaluate plans and goals. Likewise, belief update and action functions should be simplified, so the agent does not spend too much processing updating beliefs or performing actions in the environment.

The internal data structures of the framework should be optimized to reduce memory usage and program size. For example, the framework can use integer variables to represent beliefs, even though the programmer provides these as text values. In addition, all internal data structures, such as the event and intention queues, must have limited size, and memory allocation should be static to minimize the risk of having the agent running out of memory due to dynamic allocation of memory.

Edge cases for full data structures must be taken into consideration and designed so that the risk of dropping a running intention is minimized. For example, if a belief update event results in adopting an intention and the intention queue is full, the agent should discard the new intention instead of replacing one of the existing (and running) intentions with it. Otherwise, the agent will stop executing an ongoing plan and may leave the state of the world in an unexpected state. For example, if the agent is embedded in a vehicle, the vehicle may stop moving abruptly due to the intention related to movement being dropped to process a new intention. However, it is important to note that this behavior can be adapted and improved with the addition of priorities when handling intentions, as some intentions can be more important than others.

4.3. Requirements of the Implemented Agents

Unlike the Jason interpreter, the framework should allow implementing fully-compiled agents for execution. Agent code in AgentSpeak format is parsed and converted into corresponding C/C++ code for compilation, avoiding iterative parsing of the agent code and allowing better performance.

Because each hardware platform has distinct I/O interfaces, the functions to update beliefs and act in the environment should be provided to the framework by the programmer, since adding internal actions to the framework will increase the size of the program and can raise incompatibility for multi-platform support.

It is also important to note that the agent reasoning cycle must be considered when implementing functions to update beliefs and act in the environment: hardware features such as interruptions must be avoided on these functions, as they can interfere with the reasoning cycle and break agent functionalities.

Lastly, programmers should also have the option to configure the size of the internal data structures of the agent, such as the event and intention queues. Because memory

is limited and recursion functions are allowed, events are constantly generated, and plans can be stacked during execution, the size of the internal data structures can vary depending on implementation and usage. Table 1 summarizes the requirements and proposals from this section.

Requirement	Proposal
Development Methodology	Incremental
Platform Support	Multi-Platform
Programming Language for Framework Development	C and/or C++
Programming Language for Agent Implementation	AgentSpeak
Main AgentSpeak Features Supported in Initial Versions	Propositions Simple Unification Algorithms
Memory Allocation	Static
Size of Internal Data Structures	Configurable in the Framework
Internal Methods	No Internal Methods
Belief Update and Action Functions	Provided by the Agent Programmer
Hardware Interruptions	Not Supported

Table 1. Summary of Framework Requirements

5. Conclusion

We discuss the implementation challenges and design considerations for creating a framework that allows implementing BDI agents in common low-cost hardware used in embedded systems.

Although the incremental development methodology suggested may result in a small set of features being added to the framework at times, the ability to be able to constantly review and evaluate which features should be added to the framework and how they can be implemented ensures that multi-platform support can be reached more easily.

In addition, our insights about the framework's requirements aim to tackle one of the main challenges for embedding agents; the limited hardware resources, combined with the advanced features of the BDI architecture, make it difficult to implement complex reasoning algorithms in simple hardware.

We believe that taking Jason as basis for software architecture and features, it is possible to implement the framework proposed, given that the strategies suggested in this paper are followed.

References

Aliyuda, A. (2016). Towards the design of cyber-physical system via multi-agent system technology. In *International Journal of Scientific & Engineering Research*, volume 7.

Barros, R., Heringer, V., Lazarin, N. M., and Moraes, L. (2014). An agent-oriented ground vehicle's automation using jason framework. pages 261–266.

Berger, A. S. (2002). *Embedded Systems Design: An Introduction to Processes, Tools, and Techniques*. CMP Books. Taylor & Francis.

Bordini, R. H., Dastani, M., Dix, J., and Seghrouchni, A. E. F. (2009). *Multi-Agent Programming: Languages, Tools and Applications*. Springer Publishing Company, Incorporated, 1st edition.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007a). *Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, Inc., Hoboken, NJ, USA.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007b). *Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, Inc., Hoboken, NJ, USA.

Bratman, M. (1987). *Intention, plans, and practical reason*. Harvard University Press, Cambridge, MA.

Bucheli, S., Kroening, D., Martins, R., and Natraj, A. (2015). From agentspeak to c for safety considerations in unmanned aerial vehicles. pages 69–81.

Horvath, I. and Gerritsen, B. (2012). Cyber-physical systems: Concepts, technologies and implementation principles.

Junior, M. F. S. (2015). Melhorando o desempenho de agentes bdi jason através de filtros de percepção.

Lazarin, N. M. and Pantoja, C. (2015). A robotic-agent platform for embedding software agents using raspberry pi and arduino boards.

Marwedel, P. (2010). *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems*. Springer Publishing Company, Incorporated, 2nd edition.

Mascardi, V., Demergasso, D., and Ancona, D. (2005). Languages for programming bdi-style agents: an overview. pages 9–15.

Miller, J. and Esfandiari, B. (2021). Analyzing the execution time of the jason bdi reasoning cycle. 9th International Workshop on Engineering Multi-Agent Systems.

Park, K.-J., Zheng, R., and Liu, X. (2012). Cyber-physical systems: Milestones and research challenges. *Comput. Commun.*, 36:1–7.

Rajkumar, R., Lee, I., Sha, L., and Stankovic, J. (2010). Cyber-physical systems: The next computing revolution. In *Design Automation Conference*, pages 731–736.

Rao, A. S. (1996). Agentspeak(l): Bdi agents speak out in a logical computable language. In Van de Velde, W. and Perram, J. W., editors, *Agents Breaking Away*, pages 42–55, Berlin, Heidelberg. Springer Berlin Heidelberg.

Santos, F. R. (2015). Avaliação do uso de agentes no desenvolvimento de aplicações com veículos aéreos não-tripulados.

Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition.

Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115–152.