Embedding Multi-Agent System Frameworks: A Benchmarking

Cleber Jorge Amaral[1]

[1]Instituto Federal de Santa Catarina - IFSC
Campus Sao Jose - SC, Brazil

{cleber.amaral}@ifsc.edu.br

Abstract. *The increase use of Multi-Agent Systems (MAS) is mobilizing efforts to develop techniques to embed agents. The development of new approaches may be motivated because of fact that the most popular MAS developing frameworks use high level languages. It brings challenges, in terms of capability, to run these applications and to guarantee good performance. This research aims to test these popular frameworks in also popular embedded platforms. The objective is to check whether it is feasible and how these frameworks perform common embedded tasks like sensing and actuation as well as in communication.*

1. Introduction

Many programming approaches were proposed in Multi-Agents Systems (MAS) field. Recent researches are focusing on BDI (*Beliefs - Desires - Intentions*) architecture, which fairly represents the agents knowledge about the world, the objectives to achieve and ongoing plans. This architecture can provide good resources to develop the logical theory as well as practical issues [Mascardi et al. 2005]. To develop agents, most of the approaches use high level programming languages, specially Java, which is the base language of 4 of top 5 most popular Multi-Agent Systems platforms [Kravari and Bassiliades 2015].

In the other hand, embedded platforms are limited in computation and storage resources comparing to computers [Semwal and Nair 2016]. It is mobilizing the development of new approaches suitable for these constrained applications. The strong abstraction of high level languages combined to interpretation or intermediate representation brings challenges related to performance, specially for embedded systems [Semwal et al. 2015].

In this research it is intended to test BDI approaches in different embedded platforms. It aims to check whether and which common embedded platforms may support these systems. In this sense, an application based on *Gold Miners* challenge will be developed for each MAS framework. These programs will run in different embedded platforms to check CPU and memory occupancy, as well as response time of the agent.

2. Methodology and Partial Results

The test scenario is being a combination of physical and simulated robots which should use sensors, actuators and communication. The proposed challenge is a known application of MAS called *Gold Miners*. In this problem there are multiple agents with the *miner* role, which means that they should go around the area and search for gold. When some miner

finds gold it should bring to an station. As represented in Figura 1, a *miner* agent in a MAS approach is going to run in the board. In this distributed system, this miner application connects to a server, joining in a central MAS.

In this research a MAS *Gold Miners* implementation will be developed for each BDI approach to be tested which are: Jack, JADE and Jason. The approaches should provide functions to distribute the system in a central and a *miner*. The central run in a computer server and the *miner* embedded in a board. The embedded miner should interact with physical world reading and writing pins. The embedded platforms to run the miner code are: Raspberry Pi 3 (Running Raspbian and ChibiOS) and Cubieboard 3 (running Cubieez operating system).

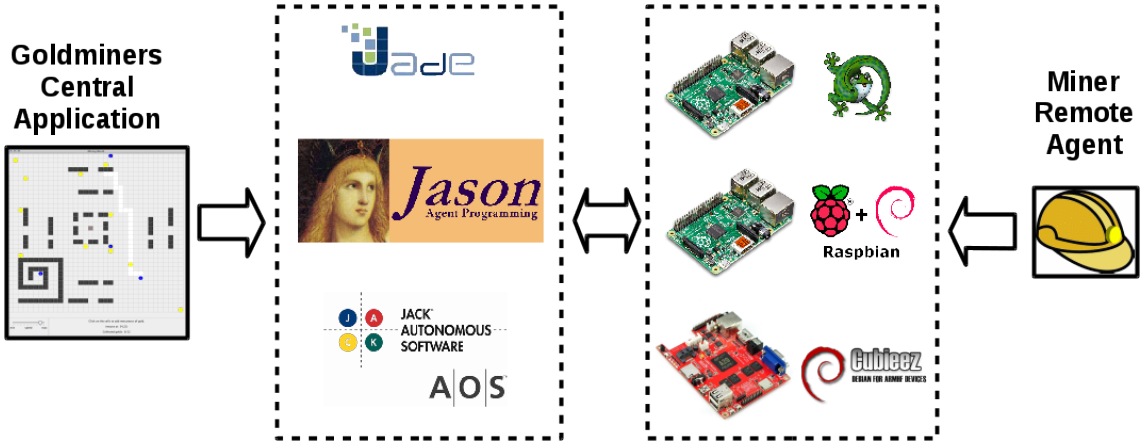

Figura 1. Test combinations of MAS frameworks and embedded platforms

At current time the *Gold Miners* challenge was adapted for JaCaMo (Jason + CartAgO + Moise) framework. A central application was developed to run in a computer server and provide a common environment for remote agents connect using CArtAgO infrastructure. The miner agent was deployed and tested using a Raspberry Pi 3 Model B running Raspbian 4.9.41-v7+. The movements of the miner were decided by the remote board which sent to the central, by CArtAgO artifacts, its intentions. The central received its attempts and checked if the movement was allowed and if the agent reached gold. When the agent found gold, the board turned a LED to ON state, as an output function. The LED was turned OFF when the agent put the gold in the depot.

The results showed that JaCaMo could be embedded, the agent communicated properly with the central which could update the physical state of the miner. The Raspberry board showed only 0,4% of its CPU usage when just the operating system was running. After the joining miner was launched the CPU usage[1] rose to 26,5%. The memory occupancy in same condition rose from 10% to 17%. For response time, the LED pin was connected to a configured input pin. After each state change of the LED an internal timer was triggered and stopped after the agent perceived its change by the input pin. This situation was tested 30 times. The average response time of this agent was 292 microseconds with a standard deviation of 443 microseconds[2].

[1]For CPU usage and memory occupancy the linux command 'top' was used. About memory, it is considering physical and swap

[2]For response time the Java function System.nanoTime() was used.

3. Conclusion

The average response time was about 300 microseconds what could be enough in common embedded applications. About standard deviation, it could be considered high. In one of those 30 samples the response time exceed 2 milliseconds, showing it as an inaccurate parameter. This result was indeed foreseen since Raspbian is a non real-time system. This behavior would be apart of any agent technology or even optimized applications. It is important to consider that it is not usual such non real-time operating systems in critical tasks of embedded systems, like a motor encoder reading, which requires short and precise processing time. In these cases it is common to have an specific processor, usually with a small application running in a super loop.

On the whole, current tests are showing that Jason agents can run properly on one of the most popular embedded platforms, Raspberry Pi 3. The processor usage and footprint are not being heavily affected and it is showing that bigger applications can be embed. The next steps of this research is to compare this results with other embedded platforms and operating systems, specially a real-time operating system where the response time should be more regular. Finally, the same application should be developed for the other MAS frameworks to test them in similar situation.

Referências

Kravari, K. and Bassiliades, N. (2015). A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11.

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

Semwal, T., Bode, M., Singh, V., Jha, S. S., and Nair, S. B. (2015). Tartarus: A multi-agent platform for integrating cyber-physical systems and robots. In *Conference on Advances In Robotics*, pages 20:1–20:6, New York, NY, USA. ACM.

Semwal, T. and Nair, S. B. (2016). Agpi: Agents on raspberry pi. *Electronics*, 5.