Implementing Virtual Pheromones in BDI Robots Using MQTT and Jason

Michele Bottone*, Filippo Palumbo†, Giuseppe Primiero*, Franco Raimondi* and Richard Stocker*
*Department of Computer Science, Middlesex University, London, United Kingdom
†Institute of Information Science and Technologies “Alessandro Faedo”, National Research Council, Pisa, Italy

***Abstract*—Robotic coordination is a crucial issue in the development of many applications in swarm robotics, ranging from mapping unknown and potentially dangerous areas to the synthesis of plans to achieve complex tasks such as moving goods between locations under resource constraints. In this context, stigmergy is a widely employed approach to robotic coordination based on the idea of interacting with the environment by means of markers called pheromones. Pheromones do not need to be “physical marks”, and a number of works have investigated the use of digital, virtual pheromones. In this paper, we show how the concept of virtual pheromones can be implemented in Jason, a Java-based interpreter for an extended version of AgentSpeak, providing a high-level modelling and execution environment for multi-agent systems. We also exploit MQTT, a messaging infrastructure for the Internet-of-Things. This allows the implementation of stigmergic algorithms in a high-level declarative language, building on top of low-level infrastructures typically used only for controlling sensors and actuators.**

***Index Terms*—stigmergy, cloud robotics, multi-agent systems**

I. Introduction

Multi-Agent Systems (MASs) are increasingly becoming a key tool for a variety of applications such as traffic control, manufacturing, e-commerce, patient monitoring, or games [1]. This is due to the possibilities offered by MAS to naturally model, design, and implement several aspects of multiple interacting intelligent agents within an environment. MAS can be used to solve problems that are difficult or impossible for individual agents or monolithic systems to solve. In the robotic field, MAS are often used to have multiple robots working together. In particular, they offer communication, coordination, task planning, and distributed agent frameworks. When there is the need to scale up to potentially thousands of robots, we can talk about Swarm Robotics (SR). Many old MAS algorithms could not support such large numbers and did not address robot physical formation. Swarm robotics uses the intelligence that emerges from the interactions among individual robots [2].

When implementing a SR system, an efficient method for sharing and utilising the information that is gathered by multiple robots is required. A key concept in the optimisation of SR is “stigmergy”: communication by way of the environment [3]. Stigmergy is used in biology to describe the influence of the persisting environmental effects of previous actions on the behaviour of an agent acting in the modified environment. It was originally proposed by Grassé [4] to explain his observations on termite building behaviour. Termites exchange among them the “quality” of a path by marking it with pheromones so that a positive feedback mechanism ends eventually in most insects following the “best” path. In this vision, pheromone robotics [5], [6] is a good approach to implement the interactions among multiple robots. Each individual robot gathers, simplifies, and shares information from the environment for interaction among multiple robots. In different works [7], [8], authors propose the concept of virtual pheromones, defined as engrams created by the agents not in the environment, but in a representation of it as a map.

A swarm robot system is a multi-robot ecosystem which consists of a large number of simple, lightweight, and interoperating robots. In this regard, the ability to connect to the Internet can be taken advantage in multiple ways. One is to exploit the resources cloud computing can offer in order to lower the single robot’s computational and memory capabilities. In this way, we can centralise the computational effort for the update of the virtual pheromone map in a remote back-end server, external to the swarm.

We present an approach to concretely realise coordination among robots, by addressing all layers from physical to high-level delivery mechanisms. In particular, we show how stigmergy can be achieved using a messaging mechanism for the Internet of Things (IoT), namely MQTT. At the same time, we show how to implement stigmergy in Jason, a Belief-Desire-Intention (BDI) framework for multi-agent systems. We describe an implementation of this framework that we have tested on a limited resource device (a Raspberry Pi 3).

The rest of the paper is organised as follows: Section II introduces the key concepts and technologies used related to the literature, Section III shows the implementation details and insights on the cooperation mechanism by means of MQTT topics and the pheromone construction by means of Jason BDI agents, while Section IV draws the conclusions.

II. Preliminaries and related work

A. *Stigmergy*

Stigmergy [9], [10] is a mechanism of spontaneous, indirect coordination between agents, where the trace left in the environment by an action stimulates the performance of a subsequent action, by the same or a different agent. The word Stigmergy is derived from the Greek words *stigma* (sign) and *ergon* (work/action), capturing the notion that an agent’s action leaves signs in the environment that he and other agents sense and that determine and incite their subsequent actions. It is a form of self-organisation that produces complex, apparently intelligent structures, without the need for any planning, control, or even communication among the agents. It was first observed in social insects: ants for example exchange information by laying down pheromones on their way back to the nest when they have found food. In this way, they collectively develop a complex network of trails, connecting the nest in the most efficient way to the different food sources.

This mechanism is widely used in pervasive computing domains, ranging from indoor localisation [11], to ambient assisted living [12] and general sensor network scenarios [13].

For its specific features, stigmergy can be easily integrated in a MAS as a different method of communication between agents, which exchange information by altering the environment releasing digital pheromones. These are "marks" on the virtual environment which can be perceived by agents for a limited time after their release. As a result, simple, local, and unplanned actions of the agents emerge in a complex and apparently intelligent behaviour of the system as a whole.

In the literature, several works exploited the stigmergy to build MASs. In [14] a mixture of formation, planning and coordination strategies are used to address the topic of self-assembly robots. They introduced the concept of extended stigmergy in order to enable robots to pass information to one another. In [15], the authors present a series of experiments where a group of mobile robots gather 81 randomly distributed objects and cluster them into one pile using stigmergy.

B. The Jason BDI framework

An *agent* is usually considered to be an entity with one or more of the following properties: autonomy, social ability, reactivity, pro-activity [16]. A MAS could be easily characterised by means of its *Beliefs*, *Intentions*, and *Desires* (BDI). Among the existing BDI frameworks [17], we have chosen Jason because it is representative of an important approach to BDI architectures: Jason/AgentSpeak is essentially a *rule-based system* that makes use of the notion of planning. The underlying structure for AgentSpeak [18] relies on the concept of a *reasoning cycle*: an agent has *beliefs*, based on what it perceives and communicates with other agents; beliefs can produce *desires*, intended as states of the world that the agent wants to achieve; the agent deliberates on its desires and decides to commit to some; desires to which the agent is committed become *intentions*, to satisfy which the agent executes plans that lead to action. The behaviour of the agent (i.e., its actions) is thus explained or caused by what it intends (i.e., the desires it decided to pursue).

AgentSpeak(L) is an abstract declarative programming language for implementing BDI agents with Prolog-like instructions, which can be extended to fit specific needs. Its syntax defines agent programs as a set of logical beliefs, rules and plans, and is formally defined in the following way.

For $\mathcal{S}$ a finite set of symbols including predicates, actions, and constants, and $\mathcal{V}$ a set of variables, one can define vectors of terms in first-order logic:

- If b is a predicate symbol and $\mathbf{t}$ a term, we define $b(\mathbf{t})$ to be a belief atom.
- if $b_A(\mathbf{t})$ and $b_B(\mathbf{t})$ are belief atoms, where A and B can be conjunctions, disjunctions or negations of belief literals, then the rule $b_A(\mathbf{t}) :- b_B(\mathbf{t})$ describes how the latter is inferred from the former.
- If $g(\mathbf{t})$ is a belief atom, then $!g(\mathbf{t})$ and $?g(\mathbf{t})$ are goals, $!g(\mathbf{t})$ denoting an achievement goal and $?g(\mathbf{t})$ a test goal.
- If $p(\mathbf{t})$ is a belief atom or goal, then $+p(\mathbf{t})$ and $-p(\mathbf{t})$ are triggering events with $+$ and $-$ denoting respectively the addition and deletion of a belief to be held or goal to be achieved.
- If a is an action symbol and $\mathbf{t}$ a term, then $a(\mathbf{t})$ is an action.
- If e is a triggering event, $c_1, \ldots, c_m$ are beliefs and $q_1, \ldots, q_n$ are goals or actions, the rule $e : c_1, \ldots, c_m \leftarrow q_1, \ldots, q_n$ defines a plan, with $c_1, \ldots, c_m$ its context and $q_1, \ldots, q_n$ its body.

Jason [19] programming revolves around plans, which are the closest thing there is to a function or method in declarative languages. Actions in the body of an expression are executed in sequence as a consequence of the triggering of the plan, which can consist of belief addition and removal, requests to achieve and un-achieve (sub)goals, or built-in or user-defined internal actions that change the environment or the agent's state over time. In Jason, ground literals are also extended by strong negation, annotations, and message passing. Jason extends the AgentSpeak syntax into an extensible and customisable Java-based, open-source development environment and interpreter. In particular, Jason allows for the definition of bespoke *environments* extending a base Environment class. The proposed approach is supported by the recent advances in literature regarding the use of Jason as language for the implementation of stigmergy in BDI agents. In particular, authors in [20] extend Jason with shared beliefs using stigmergy.

We exploit the easy customisation of Jason in the next section to implement an efficient interaction with an MQTT infrastructure and to realise the virtual pheromone approach.

C. MQTT

The Message Queue Telemetry Transport (MQTT) is a lightweight protocol initially developed for wireless sensor networks and running on top of TCP/IP connections [21]. MQTT messages are characterised by a *topic* and by a *Quality of Service* (QoS). A *broker* is required to dispatch messages from publishers to subscribers. When a client connects to the broker, it is identified by an ID and it can perform the following actions: connect, disconnect, subscribe (to a topic), unsubscribe (from a topic), and publish a message under a certain topic and at a given QoS. Topics are organised in a hierarchy, for instance the message t1/t2/t3/t4 has t1 has main topic, t2 as subtopic, t3 as subsubtopic, etc.. Subscribers can specify *patterns* using the symbols + (matching one occurrence of a topic) and # (matching any number of topics). For instance, t1/#/t4 will match t1/t2/t3/t4, but t1/+/t4 will not.

MQTT provides three levels of Quality of Service:

- Level 0: the message is sent *at most* once (either by the client or by the broker), with no guarantee of delivery.
- Level 1: the message is sent *at least* once, until a confirmation is received.
- Level 2: the message is sent *exactly* once.

Several open source implementations are available, both for brokers and for clients. In our experiments, we have employed an on-line broker and the associate MQTT client available at http://www.hivemq.com/. The latter allows both subscription and publishing of messages. A local broker can also be implemented using Mosquitto (http://mosquitto.org/).

For the purposes of this work, we will assume that a broker is available and we will employ the MQTT Java library Paho, which "provides open-source client implementations of MQTT messaging protocols aimed at new, existing, and emerging applications for Machine-to-Machine (M2M) and Internet of Things (IoT)" (http://www.eclipse.org/paho/). The Java code listed in Figure 1 shows how to create a client that connects to a public broker (line 14), subscribes to a topic (line 18), and sends a message (lines 21 to 23). When a message whose topic matches the one specified on line 8 is generated, a listener (line 17) is invoked. In previous work [22], we showed the potential of implementing MQTT on resource-constrained

```java
public class SimpleMQTTSubscriberAndPublisher implements
     MqttCallback {

  private MqttClient client;
  private static String BROKER_ADDR="tcp://broker.hivemq
       .com:1883";
  private static String CLIENT_NAME="TestStigmergyBDI";

  // The topic we are interested in.
  private static String SUBSCRIBED_TOPIC="MQTTTest";

  public SimpleMQTTSubscriberAndPublisher() {

    // Connect to the broker:
    client = new MqttClient(BROKER_ADDR, CLIENT_NAME);
    client.connect();

    // Set callback for subscriptions
    client.setCallback(this);
    client.subscribe(SUBSCRIBED_TOPIC+"/#");

    // Publish a message under a topic
    MqttMessage message = new MqttMessage();
    message.setPayload("Hello␣World".getBytes());
    client.publish("pubTopic/subtopic", message);
  }
}
```

Fig. 1. Java MQTT client (excerpts).

devices such as Arduino and Raspberry Pi. In this work, we exploit the features offered by MQTT to dispatch and distribute information about the virtual pheromone map shared by the robots embedding a Raspberry Pi platform.

III. Implementing virtual pheromones

A. Connecting MQTT with Jason

In Jason, modellers can modify the environment and thus, the beliefs of the agents. Specifically, a new Environment can be created by subclassing the Jason class `Environment.java`. Figure 2 reports excerpts from the implementation of a bridge between a MQTT infrastructure and Jason. The new class `MQTTEnvironment` subclasses the default Environment class and extends the initialisation method (line 3) with the connection to a MQTT broker and the subscription to appropriate topics (lines 9 and 10). The key method here is `messageArrived` on line 16 that is invoked when a message arrives from the broker. The message is appropriately parsed and a new *percept* is created (line 18). This percept may result in a new belief in an agent and gives rise to new intentions. Similarly, the Environment can be extended in a very simple way with new actions that can be invoked by agents when they want to publish a MQTT message.

```java
1  public class MQTTEnvironment extends Environment
        implements MqttCallback {
2
3    public void init(String[] args) {
4      // [...]
5      try {
6        client = new MqttClient(MQTT_BROKER, "JasonMQTT");
7        client.connect();
8        // [...]
9        client.setCallback(this);
10       client.subscribe("TOPIC/#");
11     } catch (Exception e) {
12       e.printStackTrace();
13     }
14   }
15
16   public void messageArrived(String topic, MqttMessage
          message) throws Exception {
17     // [...]
18     addPercept(Literal.parseLiteral(parse_message(topic,
            message)));
19   }
20 }
```

Fig. 2. Jason - MQTT bridging environment.

B. Implementing stigmergy in Jason

We start by extending the default environment `GridWorldModel`, following the approach presented in [20]. In particular, in the Java implementation of the Environment, we add a private field holding the intensity of pheromones in each cell of the grid. The intensity of pheromones in the grid can be updated in two ways:

- Directly by one of the agents: implementing a new internal action `mark` in the Java environment to send the appropriate MQTT message to all the agents.
- Via MQTT: the message is delivered to the Java environment by the callback method described in Figure 1.

Agents implemented in Jason can get a local map of the intensity of pheromones in their neighbourhood by invoking a new internal action `getMarks` implemented in the environment. The local map is an array of nine triples in the form `(x,y,i)`, where `i` is the pheromone intensity, centred around the current position of the agent.

The main difference between our approach and the one presented in [20] is that in our case we have *multiple instances* of Jason agents, one per robot, running in parallel. Moreover, the approach of [20] relies on a local server for messaging, while we employ MQTT as our communication mechanism. This has the additional benefit of allowing the incorporation of agents implemented in other frameworks, provided that they support MQTT messaging. Figure 3 shows an example of how two BDI robots interact with the MQTT Broker in order to send their positions to the back-end server and retrieve the local pheromone map (the `mark` structure) around their position. All the robots present in the environment, equipped with their own localisation system and wireless communication, subscribe to the relative topics, where the local pheromone map will be received: `/VPM/map/robotID`. At each time step, robots send their position to the back-end server publishing their coordinates to the MQTT topic: `/VPM/position/robotID`. Then, the back-end server updates the global virtual pheromone map and publishes the local pheromone map to each robot by means of the dedicated topics `/VPM/map/robotID`. Finally, the robots choose the new position among the nine possibilities as the coordinates of the point with the higher intensity (`i`). In absence of a local pheromone map, robots choose the new position randomly.

The update process of the global pheromone map is based on the potential field model [23]. At each time step, it computes the intensity at distance d_k from each pheromone k using equation 1:

$$p(d_k) = \begin{cases} p_k\left(1 - \dfrac{d_k}{\sigma}\right) & \text{if } 0 < d_k < \sigma \\ 0 & \text{if } d_k \geq \sigma \end{cases} \tag{1}$$

where $p(d_k)$ is the intensity of pheromone k at distance d_k due to diffusion, σ is the sensitivity range, and p_k is the actual intensity of pheromone k. Due to the stigmergic aggregation of all the N sources located within σ, the resulting pheromone intensity sensed in an arbitrary location is given by equation 2:

$$P = \sum_{k=1}^{N} p_k\left(1 - \frac{d_k}{\sigma}\right). \tag{2}$$

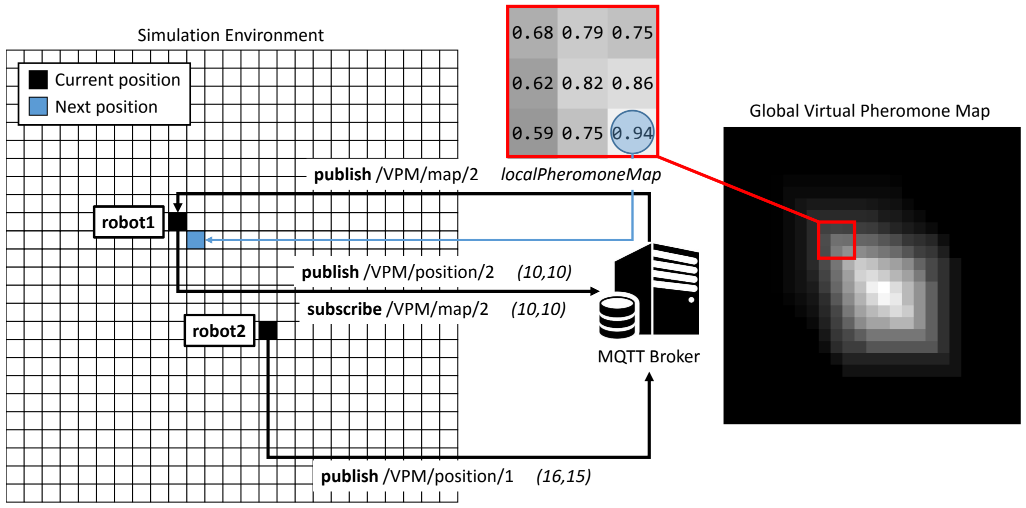

Fig. 3. An example of how two BDI robots interact with the MQTT Broker in order to send their positions to the back-end server and retrieve the local pheromone map (the `mark` structure) around their position.

Assuming that the evaporation effect decreases the pheromone intensity linearly, it is possible to update the resulting pheromone at time t as shown in equation 3:

$$P = \sum_{k=1}^{N} p_k \left(1 - \frac{d_k}{\sigma}\right)\left(1 - \frac{t - t_k}{\tau}\right) \quad (3)$$

where t_k is the time of creation of the pheromone k and τ is the evaporation parameter. The value of P around the current position of the robot is returned to it as a message (*localPheromoneMap* in Figure 3) published on the topic previously subscribed `/VPM/map/robotID`, and thereafter the robot acts as if it has its own pheromone sensing system [8], choosing the next position as the max of the local pheromone map (blue square in Figure 3).

IV. Conclusion

In this paper, we presented the implementation of a virtual pheromone system based on MQTT and Jason. The presented solution is suitable for modelling BDI agent behaviour in swarm robotics scenarios. The capabilities offered by the MQTT integration open a plethora of solutions in the cloud robotics field, ranging from the creation of complex and robust robots coordination across a variety of robotic platforms, to the exploitation of the rapid increase in data transfer rates to offload tasks without hard real time requirements. The presence of the stigmergic coordination system among robots allows the decentralisation of the computation outside the "swarm", with simpler entities with less requirements in terms of computational resources creating an emerging intelligence from the environment itself.

References

[1] M. Schumacher, "Multi-agent systems," *Objective Coordination in Multi-Agent System Engineering: Design and Implementation*, pp. 9–32, 2001.
[2] I. Navarro and F. Matía, "An introduction to swarm robotics," *ISRN Robotics*, vol. 2013, 2012.
[3] G. Beni, "From swarm intelligence to swarm robotics," in *International Workshop on Swarm Robotics*. Springer, 2004, pp. 1–9.
[4] P.-P. Grassé, "La reconstruction du nid et les coordinations interindividuelles chezbellicositermes natalensis etcubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs," *Insectes sociaux*, vol. 6, no. 1, pp. 41–80, 1959.
[5] D. Payton, R. Estkowski, and M. Howard, "Compound behaviors in pheromone robotics," *Robotics and Autonomous Systems*, vol. 44, no. 3, pp. 229–240, 2003.
[6] J. D. McLurkin, "Stupid robot tricks: A behavior-based distributed algorithm library for programming swarms of robots," Ph.D. dissertation, Massachusetts Institute of Technology, 2004.
[7] W.-S. Moon, J.-W. Jang, H.-S. Kim, and K.-R. Baek, "Virtual pheromone map building and a utilization method for a multi-purpose swarm robot system," *International Journal of Control, Automation and Systems*, vol. 13, no. 6, pp. 1446–1453, 2015.
[8] I. Susnea, G. Vasiliu, A. Filipescu, A. Serbencu, and A. Radaschin, "Virtual pheromones to control mobile robots. a neural network approach," in *2009 IEEE International Conference on Automation and Logistics*. IEEE, 2009, pp. 1962–1967.
[9] O. Holland and C. Melhuish, "Stigmergy, self-organization, and sorting in collective robotics," *Artificial life*, vol. 5, no. 2, pp. 173–202, 1999.
[10] E. Bonabeau, "Editor's introduction: stigmergy," *Artificial Life*, vol. 5, no. 2, pp. 95–96, 1999.
[11] F. Palumbo, P. Barsocchi, S. Chessa, and J. C. Augusto, "A stigmergic approach to indoor localization using bluetooth low energy beacons," in *Advanced Video and Signal Based Surveillance (AVSS), 2015 12th IEEE International Conference on*. IEEE, 2015, pp. 1–6.
[12] P. Barsocchi, M. G. Cimino, E. Ferro, A. Lazzeri, F. Palumbo, and G. Vaglini, "Monitoring elderly behavior via indoor position-based stigmergy," *Pervasive and Mobile Computing*, vol. 23, pp. 26–42, 2015.
[13] F. Zambonelli, A. Omicini, B. Anzengruber, G. Castelli, F. L. De Angelis, G. D. M. Serugendo, S. Dobson, J. L. Fernandez-Marquez, A. Ferscha, M. Mamei *et al.*, "Developing pervasive multi-agent systems with nature-inspired coordination," *Pervasive and Mobile Computing*, vol. 17, pp. 236–252, 2015.
[14] J. Werfel, Y. Bar-Yam, D. Rus, and R. Nagpal, "Distributed construction by mobile robots with enhanced building blocks," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 2787–2794.
[15] R. Beckers, O. Holland, and J.-L. Deneubourg, "From local actions to global tasks: Stigmergy and collective robotics," in *Artificial life IV*, vol. 181, 1994, p. 189.
[16] M. Wooldridge, *An introduction to multiagent systems*. John Wiley & Sons, 2009.
[17] R. H. Bordini, L. Braubach, M. Dastani, A. E. F. Seghrouchni, J. J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci, "A survey of programming languages and platforms for multi-agent systems," *Informatica*, vol. 30, no. 1, 2006.
[18] R. H. Bordini, J. F. Hübner, and M. Wooldridge, *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
[19] J. Hübner and R. Bordini, "Jason," http://jason.sourceforge.net.
[20] M. Barbieri and V. Mascardi, "Hive-BDI: Extending Jason with Shared Beliefs and Stigmergy," in *ICAART (2)*. Citeseer, 2011, pp. 479–482.
[21] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "MQTT - A publish/-subscribe protocol for Wireless Sensor Networks," in *Communication Systems Software and Middleware and Workshops (COMSware) 2008. 3rd international conference on*. IEEE, 2008, pp. 791–798.
[22] G. Barbon, M. Margolis, F. Palumbo, F. Raimondi, and N. Weldin, "Taking Arduino to the Internet of Things: the ASIP programming model," *Computer Communications*, 2016.
[23] I. Susnea, "Engineering human stigmergy," *International Journal of Computers Communications & Control*, vol. 10, no. 3, pp. 420–427, 2015.