Intelligent Environments 2016
P. Novais and S. Konomi (Eds.)

doi:10.3233/978-1-61499-690-3-66

From Raw Data to Agent Perceptions for Simulation, Verification, and Monitoring

Michele BOTTONE [a], Giuseppe PRIMIERO [a], Franco RAIMONDI [a,1], Neha RUNGTA [b]

[a] *Department of Computer Science, Middlesex University, London, UK*
[b] *NASA Ames Research Center, Moffett Field, CA, USA*

Abstract. In this paper we present a practical solution to the problem of connecting "real world" data exchanged between sensors and actuators with the higher level of abstraction used in frameworks for multiagent systems. In particular, we show how to connect an industry-standard publish-subscribe communication protocol for embedded systems called MQTT with two Belief-Desire-Intention agent modelling and programming languages: Jason/AgentSpeak and Brahms. In the paper we describe the details of our Java implementation and we release all the code open source.

Keywords. Intelligent environments, multiagent systems, modelling, MQTT, publish-subscribe

1. Introduction

Software infrastructures for Intelligent Environments and, more in general, for the Internet of Things, have been investigated by a number of authors [1,2,3,4]. To address the limited resources of devices in Intelligent Environments (think of motion or temperature sensors), standard architectures for web services have been adapted, such as in the case of CoAP [5] and, more recently, with the ASIP programming model [6].

The underlying communication mechanism in these scenarios is typically the *topic-based publish-subscribe pattern* [7]. In this pattern, entities that create messages (the publishers) are not aware of the potential receivers. Instead, they "broadcast" messages, possibly through a broker. The receivers, on the other hand, subscribe to messages, possibly through a broker. In topic-based publish-subscribe architectures each message has a *topic* and subscribers only receive messages for the specific topic they are interested in. Accordingly, publishers need to provide a topic for each message that is generated. The use of publish-subscribe architectures in Intelligent Environments, coupled with appropriate service abstractions, has been investigated by bridging this communication pattern with REST services [8,9,10], allowing the development of large-case instances using a range of resource-limited devices.

[1] Corresponding Author. Email: f.raimondi@mdx.ac.uk

In parallel with the development of "low-level" communication infrastructures, several approaches have investigated mechanisms to simulate, verify and monitor Intelligent Environments at a much higher level of abstraction. In this area, *multiagent systems* [11] are used to reason about the beliefs, desires, intentions, knowledge and correct behaviour of (rational) agents. Examples in this direction include the verification of domotics/e-health examples [12,13] and of avionic scenarios involving the interaction of humans (pilots and air traffic controllers) with automated systems [14,15,16]. The analysis performed at this level of abstraction allows developers to reason about high-level interactions and requirements, such as "if the pilot believes that the auto-pilot is going to disengage, s/he should alert the co-pilot".

There is currently a gap between the "low-level" communication mechanisms and the "high-level", abstract activities performed in the formalism of multiagent systems. Some recent works [17,18] have addressed this issue by connecting the software that is being executed on actual devices with modelling languages for multiagent systems, thus enabling the verification of "high-level" models that have a clear semantics in "low-level" models.

To address the issue of connecting different levels of abstraction, in this paper we show how low-level messages in a publish-subscribe infrastructure can give rise to *perceptions* in a multiagent system and, correspondingly, how agents' actions can generate low-level messages. Our solution makes use of an intermediate connector between the publish-subscribe communication layer and the modelling framework for agents.

There are a number of open protocols for messages using the publish-subscribe communication model that are meant to provide a lightweight, asyncronous alternative to HTTP in the context of cloud computing and the Internet of Things for coordinating many separate environments with minimal configuration and overhead, among them MQTT, XMPP, and AMQP. In contrast to the latter two, which have support for extensions to more complex scenarios, APIs and domains, MQTT is very efficient and simple to implement, and was specifically designed for high-latency, resource-constrained devices with low bandwidth and power draw, features that make it ideal for use in embedded systems. For these reasons, it forms the communication backbone of our solution.

The rest of the paper is organised as follows: in Section 2 we provide an overview of MQTT; in Section 3 we describe two modelling frameworks for multiagent systems (Brahms and Jason/AgentSpeak); we present our solution and provide links to our open source software in Section 4.

2. An Overview of MQTT

The Message Queue Telemetry Transport (MQTT) is a lightweight protocol initially developed for wireless sensor networks [19] and running on top of TCP/IP connections.

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 per-

form 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 the on-line broker HiveMQ[2] and the associated on-line MQTT client[3]. The latter allows both subscription and publishing of messages. A local broker can also be implemented using the Python-based tool Mosquitto[4].

For the purposes of this work we will assume that a broker is available and we will employ the MQTT Java library Paho[5], 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)". In particular, we employ version 1.0.2 of the jar library[6]. We refer to the bridge code[7] for the details of how to implement the MQTT publish-subscribe client and listener. An overview of our bridging solutions is given in Figure 1.

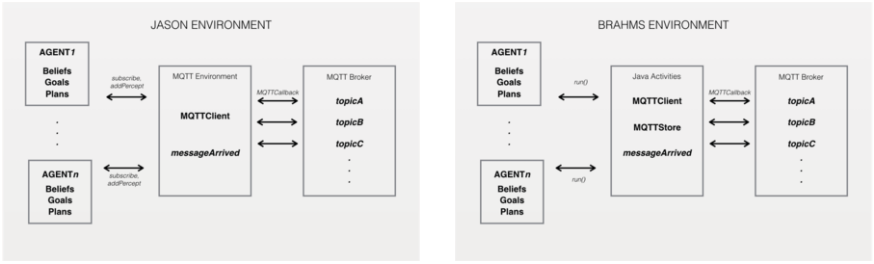

Figure 1. Left: Jason bridge. Right: Brahms bridge.

[2] http://broker.hivemq.com
[3] http://www.hivemq.com/demos/websocket-client/
[4] http://mosquitto.org/
[5] http://www.eclipse.org/paho/
[6] https://repo.eclipse.org/content/repositories/paho-releases/org/eclipse/paho/org.eclipse.paho.client.mqttv3/1.0.2/org.eclipse.paho.client.mqttv3-1.0.2.jar
[7] http://www.rmnd.net/MQTTBridges/

3. Modelling Multiagent Systems

An *agent* is usually considered to be an entity with one or more of the following properties: autonomy, social ability, reactivity, pro-activity [11]. As originally observed by McCarthy in his seminal paper [20], single and multiagent systems are used in a number of areas in Computer Science because humans have a natural attitude to ascribe *mental qualities* to complex systems to better understand and model their behaviour. Intelligent Environments are a prototypical domain for multiagent systems, as complex systems such as a robot in a domestic environment could be easily characterised by means of its (and of the surrounding humans) *beliefs*, *intentions* and *desires* (BDI). A number of BDI frameworks exist: in the following subsections we introduce Brahms [21] and AgentSpeak/Jason [22]. We have chosen these two examples because they are representative of two approaches to BDI architectures: Jason/AgentSpeak is essentially a *rule-based system* that makes use of the notion of planning; Brahms, on the other hand, is a Java-like modelling environment closer to the possible implementation logic of complex scenarios.

3.1. Jason/AgentSpeak

The underlying structure for AgentSpeak [22] is 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). Ideally, within the BDI architecture, agents should react to changes in their environment as soon as possible while keeping their proactive (i.e., desires-oriented) behaviour.

AgentSpeak(L) [22] 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 [23] programming revolves around plans, which are the closest thing there is to a function or method in a declarative language. 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 unachieve (sub)goals, or built-in or user-defined internal actions that change the environment or the agent's mental state over time. In Jason, ground literals are also extended by strong negation, annotations, and message passing.

Jason extends the AgentSpeak syntax into a flexible, extensible Java-based, open-source development environment and interpreter, which is easily customisable. In particular, Jason allows for the definition of bespoke *environments* extending a base Environment class in Java. We exploit this characteristic in the next section to interact with an MQTT infrastructure.

3.2. The Brahms Modelling Language

Brahms [21] is a Java-like, BDI-based modelling language for agents, with a specific target to model human-machine interactions. The Brahms language allows for the representation of situated activities of agents in a geographical model of the world. Situated activities are actions performed by the agent in some physical and social context for a specified period of time. The execution of actions is constrained (a) locally, by the reasoning capabilities of an agent and (b) globally, by the agents' beliefs of the external world, such as where the agent is located, the state of the world at that location and elsewhere, located artefacts, activities of other agents, and communication with other agents or artefacts. The objective of Brahms is to represent the interaction between people, off-task behaviours, multitasking, interrupted and resumed activities, informal interactions and knowledge, while being located in some environment representative of the real world.

Brahms models are described using a Java-like syntax that allows for inheritance. At each clock tick the Brahms simulation engine inspects the model to update the state of the world, which includes all of the agents and all of the objects in the simulated world. Agents and objects have states (factual properties) and may have capabilities to model the world (e.g., a display is modelled as beliefs, which are representations of the state of the environment). Agents and objects communicate with each other; the communications can represent verbal speech, reading, writing, etc. and may involve devices such as telephones, radios, displays, etc. Agents and objects may act to change their own state, beliefs, or other facts about the world. Brahms can be extended using *Java activities*, which are activities defined by the modeller using the Java programming language. We employ this feature in the following section to bridge Brahms with MQTT.

4. From MQTT to Multiagent Systems (and Back)

In Section 2 and Section 3 we have introduced, respectively, a low-level communication infrastructure and a high-level modelling, simulation and verification

```
1  public class MQTTEnvironment extends Environment implements
       MqttCallback {
2
3      public void init(String[] args) {
4          // [...]
5          try {
6              client = new MqttClient(MQTT_BROKER, "
                   JasonMQTTEnvironment");
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       }
21     }
```

Figure 2. Jason - MQTT bridging environment.

environment that can be applied to Intelligent Environments. In this section we present an approach to facilitate the interaction between these two levels.

4.1. Connecting MQTT with Jason

Jason is an example of multiagent framework in which modellers have access to the execution engine in such a way that the environment (and thus, the beliefs of the agents) can be modified by means of Java code. 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: this method is invoked when a message arrives from the broker. The message is parsed appropriately with the method `parse_message` (not reported here) and a new *percept* is created in Jason (line 18). This percept may result in a new belief in an agent and give rise to new intentions. The code for this example is available at `http://www.rmnd.net/MQTTBridges/`. This environment has been employed to monitor the trustworthiness of a monitoring infrastructure for an air conditioning system, as described in [24], scaling to several hundreds of sensors.

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. We refer to the code available online for additional details.

4.2. Connecting MQTT with Brahms

Brahms is a prototype of a framework in which developers are not allowed to modify the *state* of agents directly. This means that it is not possible to create new beliefs automatically whenever a new MQTT message is received. To address this issue we have created an intermediate *store* for MQTT messages, which should be run in parallel with Brahms. Even if beliefs cannot be created automatically, as described above Brahms can be extended with Java actions that can be invoked by agents. Our idea is to use these actions to query the external store. The latter, in turn, acts as a buffer between the MQTT broker and the (asynchronous) actions of Brahms' agents.

Figure 3 reports excerpts of our implementation (the full code is available at `http://www.rmnd.net/MQTTBridges`). Notice that, differently from the case of Jason described above, this code now runs independently from Brahms and it is not an extension of any of the Brahms classes. Similarly to the Jason Environment, the class `MQTTStore` implements the interface `MqttCallback` to be able to subscribe to messages (line 1). This new store starts a TCP socket (defined in line 3, standard constructor code omitted but available online). The constructor method starting at line 9 establishes a connection with the broker and subscribes to appropriate topics. Also, it creates an empty list of MQTT messages. Messages received from the broker and corresponding to the appropriate topic are added to this list using the method `messageArrived` (lines 25 to 28). The method `run` is the method invoked by the TCP server whenever a new connection is made to this class. Essentially, this is the method invoked by Brahms Java activities to send and receive messages. To send a message, a Brahms activity needs to provide a topic and an actual message. When a Brahms activity requests the stored messages, the content of the list `mQTTMessages` (line 6) is returned and the list is emptied. This approach makes the class `MQTTStore` a buffer between the broker and the Brahms running instance, given that it is not possible to implement `MqttCallback` directly in Brahms.

We are currently using this approach to simulate and verify an avionic scenario in which human pilots and air traffic controllers are modelled and verified in Brahms, while traffic originating from Unmanned Aerial Vehicles (UAVs) is implemented using standard sensors and actuators communicating with MQTT. Our extensions allows Brahms to interact *directly* with UAVs using their actual code. This allows us to perform detailed simulations and verification, using the approach described in [25].

5. Conclusions

The correctness and reliability of Intelligent Environments is paramount if they have to operate for and in cooperation with humans. As described above, a number of approaches have addressed the problem of verification for Intelligent Environments using multiagent systems. These approaches, however, operate at a level of abstraction that is difficult to relate automatically with the actual protocols used to control sensors and actuators.

```
public class MQTTStore implements MqttCallback {

  ServerSocket incomingSocket;
  private MqttClient mQTTClient;
  // Incoming MQTT messages are stored here
  LinkedBlockingQueue<String> mQTTMessages = null;
  // [...]

  public MQTTStore() {
    //[...]
    mQTTClient = new MqttClient(BROKER_ADDRESS, CLIENT_NAME);
    mQTTClient.connect();
    mQTTClient.setCallback(this);
    mQTTClient.subscribe(SUBSCRIBED_TOPIC+"/#");
    // [..]
    mQTTMessages = new LinkedBlockingQueue<>();
    // [...]
  }

  public void run() {
    // Here: wait for connections and return the
    // list of MQTT messages received
  }

  @Override
  public void messageArrived(String topic, MqttMessage message)  {
    mQTTMessages.add(topic+"␣"+new String(message.getPayload()));
  }
}
```

Figure 3. Brahms - MQTT bridging environment.

In this paper we have described an approach to create a connection between these two levels. In particular, we have shown how the MQTT protocol used as a standard in many applications for the Internet of Things can be connected to BDI-based frameworks for multiagent systems. We have shown how to embed MQTT in Jason/AgentSpeak, a Prolog-like modelling language and supporting environment. This solution has allowed us to monitor the correct behaviour of an air conditioning system using the high-level rules described in [24]. We have also developed a second mechanism based on a TCP-based message store that can be used for multiagent frameworks that do not allow a direct interaction with MQTT. This is the case for the Brahms framework discussed above, but we remark that our store is generic and can be used by other multiagent system infrastructures. The use of Brahms has allowed us to simulate and verify an avionic scenario involving high-level models of pilots and air traffic controllers, and also UAVs operating using MQTT. We have released the full source code of our bridges with the hope that it will be used by the community of researchers in the area of reliable Intelligent Environments.

We are currently working at adapting our approach to other multiagent modelling frameworks, in particular NetLogo and Repast Simphony, and at incorporating efficient abstraction techniques based on symbolic execution for the verification step.

References

[1] L. Atzori, A. Iera, and G. Morabito. The Internet Of Things: A Survey. *Computer Networks*, **54**(15):2787–2805, 2010.

[2] Z. Sheng, S. Yang, Y. Yu, A. Vasilakos, J. Mccann, and K. Leung. A Survey on the IETF Protocol Suite for the Internet of Things: Standards, Challenges, and Opportunities. *Wireless Communications, IEEE*, **20**(6):91–98, 2013.

[3] J. Kim, J. Lee, J. Kim, and J. Yun. M2M Service Platforms: Survey, Issues, and Enabling Technologies. *Communications Surveys & Tutorials, IEEE*, **16**(1):61–76, 2014.

[4] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio. Interacting with the SOA-based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services. *Services Computing, IEEE Transactions on*, **3**(3):223–235, 2010.

[5] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP), 2014.

[6] G. Barbon, M. Margolis, F. Palumbo, F. Raimondi, and N. Weldin. Taking Arduino to the Internet of Things: The ASIP Programming Model. *Computer Communications*, **89-90**:128–140, 2016.

[7] K. Birman and T. Joseph. Exploiting Virtual Synchrony in Distributed Systems. *SIGOPS Oper. Syst. Rev.*, **21**(5):123–138, November 1987.

[8] M. Collina, G. E. Corazza, and A. Vanelli-Coralli. Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST. In *Personal Indoor and Mobile Radio Communications (PIMRC), 2012 IEEE 23rd International Symposium on*, pages 36–41. IEEE, 2012.

[9] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, **35**(2):114–131, 2003.

[10] T. Sheltami, A. Al-Roubaiey, A. Mahmoud, and E. Shakshuki. A publish/subscribe middleware cost in Wireless Sensor Networks: a review and case study. In *Electrical and Computer Engineering (CCECE), 2015 IEEE 28th Canadian Conference on*, pages 1356–1363. IEEE, 2015.

[11] M. J. Wooldridge. *An Introduction to MultiAgent Systems (2nd Ed.)*. Wiley, 2009.

[12] R. Stocker, L. Dennis, C. Dixon, and M. Fisher. *Logics in Artificial Intelligence: 13th European Conference, JELIA 2012, Toulouse, France, September 26-28, 2012. Proceedings*, chapter Verifying Brahms Human-Robot Teamwork Models, pages 385–397. Springer, Berlin, Heidelberg, 2012.

[13] M Webster, C. Dixon, M. Fisher, M. Salem, J. Saunders, K. Koay, and K. Dautenhahn. Formal Verification of an Autonomous Personal Robotic Assistant. *Formal Verification and Modeling in Human-Machine Systems*, 2014.

[14] N. S. Rungta, G. Brat, W. J. Clancey, C. Linde, F. Raimondi, C. Seah, and M. Shafto. Aviation Safety: Modeling and Analyzing Complex Interactions between Humans and Automated Systems. In *Proceedings of the 3rd International Conference on Application and Theory of Automation in Command and Control Systems*, ATACCS '13, pages 27–37, New York, NY, USA, 2013. ACM.

[15] R. Stocker, N. S. Rungta, E. Mercer, F. Raimondi, J. Holbrook, C. Cardoza, and M. Goodrich. An Approach to Quantify Workload in a System of Agents. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1041–1050. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[16] T. Chen, G. Primiero, F. Raimondi, and N. S. Rungta. A Computationally Grounded, Weighted Doxastic Logic. *Studia Logica*, pages 1–25, 2015.

[17] M. Kamali, L. A. Dennis, O. McAree, M. Fisher, and S. M. Veres. Formal Verification of Autonomous Vehicle Platooning, 2016.

[18] S. Bucheli, D. Kroening, R. Martins, and A. Natraj. From AgentSpeak to C for Safety Considerations in Unmanned Aerial Vehicles. In C. Dixon and K. Tuyls, editors, *Towards Autonomous Robotic Systems - 16th Annual Conference, TAROS 2015, Liverpool, UK, September 8-10, 2015, Proceedings*, volume **9287** of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2015.

[19] 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*, pages 791–798. IEEE, 2008.

[20] J. McCarthy. Ascribing Mental Qualities to Machines. In M. Ringle, editor, *Philosophical Perspectives in Artificial Intelligence*, pages 161–195. Humanities Press, 1979.

[21] W. J. Clancey, P. Sachs, M. Sierhuis, and R. Van Hoof. Brahms: Simulating Practice for Work Systems Design. *International Journal of Human-Computer Studies*, **49**(6):831–865, 1998.

[22] R. H. Bordini, J. F. Hübner, and M. J. Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007.

[23] J.F. Hübner and R.H. Bordini. Jason. `http://jason.sourceforge.net`.

[24] M. Bottone, G. Primiero, F. Raimondi, and V. De Florio. A model for trustworthy orchestration in the Internet of Things. In *Proceedings of Intelligent Environments 2016*, 2016.

[25] J. Hunter, F. Raimondi, N. S. Rungta, and R. Stocker. A Synergistic and Extensible Framework for Multi-agent System Verification. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 869–876, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.