A Multi-agent Framework that Facilitates Decoupled Agent Functioning

Dave J. Russell, Elizabeth M. Ehlers

A multi-agent framework that facilitates decoupled agent functioning.

D.J. Russell, E.M. Ehlers

University of Johannesburg, Corner of University and Kingsway, Auckland Park
Johannesburg, 2006, South Africa
emehlers@uj.ac.za

Abstract: Society is becoming more aware of the impact of Artificial Intelligence and its relevance in everyday scenarios, from search engines to mobile phone assistants. Intelligent agents focus on agents as entities that can act on environments. These agents demonstrate qualities such as autonomy, situational awareness, embodiment, and flexibility. A challenge faced is creating a mechanism for artificial intelligent agents to cross-communicate in a world where technology is disparate and always changing. This constant change leads to numerous problems, from changing protocols and deprecated communication endpoints to information or functionality loss. The research investigates existing frameworks leveraged in agent programming and the standards in place such as the Foundation for Intelligent Physical Agents standard, and the Mobile Agent System Interoperability Facility. The standards are detailed and expanded to leverage modern technologies. The investigation provides the Decoupled Environment Agent Model that abstracts agent functioning from environment execution that is based on the existing standards. In this model, components function as if the they are agents. The model focuses on common interfaces and mechanisms for communication and discovery. The Decoupled Environment Agent System is an implementation of the model that is used to test and provide a mechanism where the various components of the model behave in a consistent simple manner. The information gathered in the research, the model and the results collected identify whether an agent can effectively be loosely coupled from the environment. The implementation shows that it is possible to construct an agent that is decoupled from the functioning environment.

Keywords: Multi-Agent Systems, Agent Environments, Foundation for Intelligent Physical Agents, Mobile Agent System Interoperability Facility, Knowledge Query and Manipulation Language

1 Introduction

The execution of multi-agent systems faces a challenge that is caused by the fragmentation of the environments that they execute on. Agents have been utilized to address problems in various industries from traffic control, medical analysis, industrial control, web-based sentiment, to simulation of human-like intelligence [1].

An intelligent agent is an executable that functions continuously and autonomously in environments [2]. The belief or knowledge of the agent is the current understanding of the environment based on messages that either originate from other agents or direct from the environment [1]. An environment is the domain in which an agent executes [3,4,5]. Agents working in a loosely coupled network that work together to find solutions to problems that are beyond individual agent capabilities form a multi-agent system [6]. This network of environments is where the fragmentation takes place. The fragmentations can take many forms from different operating systems, development platforms, runtimes and chipsets.

The importance of this work is that it aims to investigate the state of the art with regards to the current frameworks and solutions which try to address the problem disparate environments by abstracting the environment away from the agent functioning. This investigation assists in understanding the landscape of multi-agent systems and the various interactions required of an agent.

Secondly, this work is aimed at providing a possible solution for this problem by producing a model and a prototype. The idea behind the model and prototype is to leverage what has been produced prior and to enhance and add on top of the work through using modern technologies and mechanisms such as Hypertext Transfer Protocol (HTTP) and Representational State Transfer (REST).

This rest of this paper is organized as follows. Section 2 summarizes the findings of the literature review, this is followed by the results of the research and a discussion into the findings.

2 Literature Review

The literature review is aimed at solving three problems. The first is to define the various components of multi-agent systems, the second is to identify any frameworks or standards that exist and the last is to consider any other utilitarian tools that can assist in decoupling agents from their environments.

2.1 Definitions

Defining an agent is the first important in the scope within the research. An agent is defined as an executable that functions autonomously in its environment. Due to how agents interact a useful concept to understand interaction among individuals via the environment is stigmergy [7]. Stigmergy is a mechanism that uses social network consensus to coordinate between agents or actions indirectly. The understanding of stigmergy provides a starting point of understanding how agents interact.

Environments provide agents with domains in which they can execute and move between to perform actions on physical or logical resources [7]. When designing an agent environment management system, one must consider whether to allow for cross-environment communications or to require that agents only leverage resources on their existing environment. The different environment agent interaction models are summarized in Fig. 1.

Understanding how agents interact with their environments leads to a requirement to understand multi-agent systems. Multi-agent systems deal with the behavior management in collections of multiple independent agents [8]. Multi-agent systems can be leveraged in scenarios where agents, which are a part of problem-solving, may require having separate goals and beliefs; for example, manufacturing and hospital scheduling. The ability of multi-agent systems to provide an open, evolving architecture that can change at runtime to leverage new services or replace existing ones is very compelling for various scenarios [9].

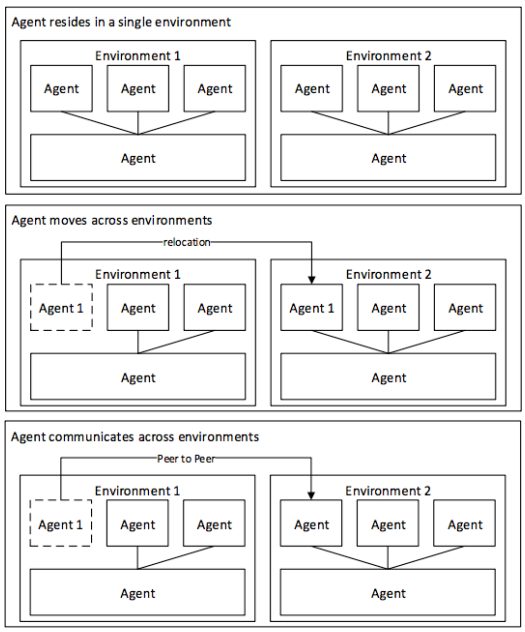

Fig. 1. Summary of the different environment agent interaction models

2.2 Standards

Throughout research three main standards emerged Foundation for Intelligent Physical Agents (FIPA), Mobile Agent System Interoperability Facility (MASIF), and Knowledge Query and Manipulation Language (KQML).

FIPA.

The Foundation for Intelligent Physical Agents, or FIPA, is an international body that focuses on the promotion of intelligent agent interoperability [10]. FIPA joined the IEEE (Institute of Electrical and Electronics Engineers) Computer Society in 2005 [11] and runs under the IEEE standards body. FIPA provides a list of specifications which specify architectural recommendations focusing on Intelligent Agents. It is a collection of standards which are intended to support the interoperation of different agents and the services they represent [12]. The standard aims to be holistic in nature and to provide an architecture that addresses a broad range of used mechanisms.

At the core of the FIPA standard defines how agents can discover and interact with each other and exchange messages [13]. The basic set of FIPA standards dictates defining an AMS (Agent Management System), a DF (Directory Facilitator) and an ACL (Agent Communication Language) for agent-based systems [14].

FIPA addresses some of the main challenges faced with abstracting out the implementation of multi-agent systems but does not tackle environments that are dynamic in nature or agent migration between environments [15]. No updates to this standard have taken place since 2002.

MASIF.

The Mobile Agent System Interoperability Facility (MASIF) is a standard specifically aimed at interfaces between mobile agent systems. The motivation behind MASIF is the integration of mobile technologies to produce a distributed computing model [15]. It presents a set of definitions and interfaces that provide an interoperable interface for mobile agent systems.

The MASIF specification defines a common model that encompasses all the major abstractions found in every mobile agent platform [16]. MASIF is similar to FIPA, in that FIPA starts with an abstract architecture description. MASIF relies heavily on the use of Common Object Request Broker Architecture (CORBA), which allows for access to CORBA- enabled objects. MASIF standardises agent management, agent transfer, agent and agent system names, agent system type, and location syntax [17].

MASIF addresses the interfaces between disparate systems that exist in a multi-agent system but does not define how agents migrated between environments and is dependent on the specific implementation details such as CORBA [12]. MASIF does not address how agents communicate [15], as it is assumed that CORBA addresses this issue, and does not deal with the dynamic environments.

KQML.

KQML is a message handling protocol and message format that can be used by agents to communicate [18]. KQML is based on speech acts and is a message format and handling protocol based on Lisp [19,20]. KQML supports direct and mediated communication, which is in line with the abstraction process that is the focus of this

research [18]. The KQML standard provides guidance on how the performatives are required to handle the various scenarios required by multi-agent system and contains all the required communication formats for decoupling an agent from its environment. This entails publication and subscribe models, to brokered messaging and peer-to-peer communications.

KQML is human-readable and straightforward for programs to interpret. KQML leverages performatives as message types and relate to basic speech acts in human speech [19], which is similar to the FIPA ACL [15].

2.3 Existing Solutions

There are a number of implementations, for example SMART (Scalable Mobile and Reliable Technology), D'Agents, Grasshopper, Aglets, SOMA, S-Aframe, and MiLog [16] [21] [22] [23] [24].

The literature study researched the various challenges and identifying existing solutions for dynamic environments. An issue identified was that agent frameworks have stagnated and are not updated as new knowledge emerges, or that they are based on and limited to specific technologies. Separation from these technologies and introduction of a core messaging system structure goes a long way to decouple agent functioning from the specifics of the environment. This use of web standards allows for an agent to execute in any format that it requires, from an executable to a JavaScript, and within an application framework.

The creation of a message handling mechanism allows for higher levels of interoperability between the various components in an agent system through providing a standardized mechanism for the components to interact.

The heterogeneous ontologies that exist for agents and the mechanisms for control and information retrieval present an additional problem. The introduction of an ontology framework for the messaging system to handle the various types of ontologies is important to ensure that agent queries result in meaningful and appropriate results. Section 4 details the Decoupled Environment Agent Model and Systems.

3 Materials and Methods

The model aims to provide a solution that decouples agents from the environments they execute in. This is done by leveraging interactions with the environment that are based on a similar concept to modern web services. The model caters for the dynamic changing components of the environment and is based on the FIPA standard. It includes changes aimed at simplifying the model.

The FIPA standard is well-established and was identified throughout the literature study. FIPA was used as a guideline for the architecture, and expanded upon to support a model based on web standards such as HTTP and REST-based communications.

The architecture is based on the FIPA Abstract Architecture, and further abstractions were set up to ensure the flexibility of the system. At the heart of the architecture is the messaging system, which is a standards-based cross-platform messaging bus. This

messaging system is a key mechanism for abstracting agent functioning and the environment it is executing in.

Fig. 2 gives a basic overview of the model architecture and how it is related to the various components of the multi-agent system.

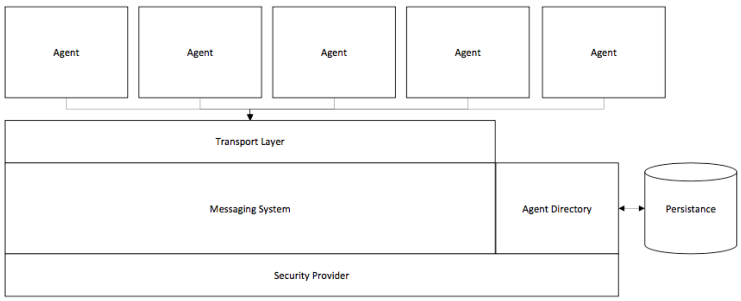

Fig. 2. Decoupled Environment Agent Model Architecture

The loosely coupled architecture of each of the components is based on existing standards. This model differs in a number of different ways, namely: The agent and service directory functionalities are merged into a single component. The messaging system follows more of a service bus mechanism that specifically handles synchronous and asynchronous communications between the various components. each agent/service within the ecosystem handles security internally and independently. Communications take place over HTTP using REST-based URI (Uniform Resource Identifier) and the JSON message format as opposed to ACL.

Throughout the investigation of intelligent agents, the need to differentiate between agents and services became trivial in nature, as agents can provide the functionality of a service. In the Decoupled Environment Agent Model (DEAM), an agent only needs information and the ability to perform control tasks. This information can come from any of the many diverse sources available in the environment. To address the problem of separating agent functioning from the environment, one needs to abstract the source of information from the agent. An agent is considered any component in an environment that either provides or consumes information with the aim of performing a task. For example, a web service which provides messages from a third-party service is considered an agent, as well as the agent calling the service.

The messaging system is the core of DEAM which enables the loose coupling of the agents from the environment. The messaging system focuses on coordinating messages between the various components of the environment from agent communication, security, agent directory, and message relays.

For the purposes of DEAM, the agent communication language is based on the KQML keyword sets but applied in a JSON format, as this is a simple mapping for KQML.

The model consists of three core components: the messaging model, the agent directory and the security provider. The messaging system is responsible for

coordinating all communication in the Decoupled Environment Agent System. The agent directory manages the agents that are resident within DEAM. The security provider a universal mechanism that handles authentication and authorization within DEAM.

The building of a simplified version of a system to act as a proving ground for a solution is the prototype, which is not a complete solution and serves as a demonstration and proof of concept of the model. The features excluded from the Decoupled Environment Agent System (DEAS) were: Agent Query Ontology, and security.

DEAS is a proof of concept to assist in validating the proposed DEAM, and to demonstrate that the technologies selected in this dissertation are suitable in demonstrating the model. The test scenario provides a simple world example of an agent and agent environment that the implementation can be tested against. The test scenario covers multiple agents attempting to perform different tasks. Table 1 describes the scenario in detail, and a visual representation is detailed in Fig. 3.

The DEAS applications focus on the core functionality of the platform and encapsulates the logic and interaction between the various components within the agent directory and the messaging systems.

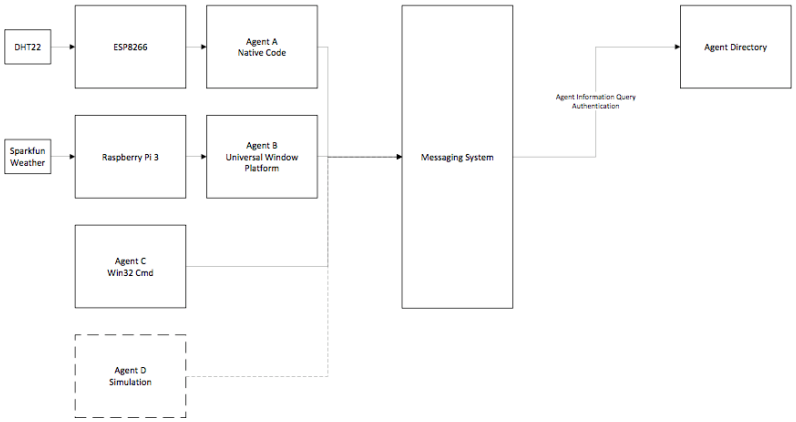

Fig. 3. Diagram of the prototype components

The first technology is Node.js server. Node.js is an application server which leverages JavaScript as its core language to build modular, single threaded and scalable websites. Node.js is a cross-platform framework that runs on Linux, Mac OS X, and Windows. Node.js is the core application and establishes connections to the data store, initialises the object-relational mapping, controls all URL routing, passes on the web rendering to the browser, initialises an HTTPS web server, and manages configuration for the sub-components.

The framework used for HTTP handling is Express.js which is a Node.js web application framework and leverages the Model View Controller (MVC) design pattern for the structuring of its various components.

JavaScript Object Notation (JSON) is an information interchange format that is lightweight and human-readable. JSON is built using two structures, a collection of name-value pairs, and ordered lists of values. The JSON format is described in du Rhone (2013). JSON is text-based and leverages Unicode code points.

The nature of the platform allows for agents to connect from any operating system or platform if it supports HTTP, REST and JSON. This prototype implements a number of agents, one being a Node.js Agent which can run on any operating system; another is a Windows Universal Application (UWP) which runs on any Windows 10 device; the last is an Arduino Agent which runs on the Huzzah Feather ESP8266 development board.

Scenario:

The scenario represents a decoupled visualisation of environment "health". The definition of "health" is whether the environment is inside specific parameters. The "health" of the environment is determined by humidity and temperature, which in turn is determined by a standalone agent with no direct access to either sensor or actuator.

There are three agents within the environment:

- Agent A is a physical device that produces humidity and temperature values when they change as well and supports requesting the information via an ask request.
- Agent B is a physical device which also produces humidity and temperature values from a different type of sensor. Agent B can show a value of Bad, Medium, and Good through a Red, Blue and Green LED respectively.
- Agent C is a processing agent which reads temperature and humidity values and tries to indicate whether the conditions are Good, Bad, or Medium.

The rules for the "health" of the environment are arbitrarily determined by the following:

$$\left\{\begin{array}{c} 19 < temperature < 21 \text{ and } 25 < Humidity < 35 : Good \\ temperature < 0 \; or \; temperature > 30 \text{ or } 10 > Humidity > 80 : Bad \\ all\ else : Medium \end{array}\right\}$$

Agent D is a simulation application that tests the system and simulates Agent A and Agent B.

Task:

Agent C needs to display the status of the environment by whichever means necessary by finding out what the current Temperature and Humidity values are and displaying health values through whatever means possible. Agent C will discover the sources of information as well as actuators and attempt to perform its task.

Table 1. Prototype Scenario

The nature of the platform allows for agents to connect from any operating system or platform if it supports HTTP, REST and JSON. This prototype implements a number of agents, one being a Node.js Agent which can run on any operating system; another is a Windows Universal Application (UWP) which runs on any Windows 10 device; the last is an Arduino Agent which runs on the Huzzah Feather ESP8266 development board.

The agents will have the following roles in the scenario:

- Coordination agent – coordinates information and sets the status;
- Sensor agent – primary agent that provides temperature and humidity information;
- Status and Sensor agent – provides a mechanism to display the status as well as secondary temperature and humidity information; and
- Simulation agent – an agent for testing.

The coordination agent is the agent that will control system health indicators by leveraging the various sources of information in the system to determine the health and then set the health actuator appropriately. The coordination agent will also play a role in ensuring that the various scenarios are exercised. The application implemented as a simple .Net Command Line application is written in C#.

The sensor agent provides temperature and humidity information. The sensor agent is implemented using a physical device, specifically the Huzzah Feather ESP 8266. Demonstrates the flexibility of the framework for platforms which leverage native development such as C++ and are not based on x86 technologies.

The status and sensor agent provide the is the actuator for the scenario and is a secondary source of temperature and humidity information. The status and sensor agent are implemented using a RaspBerry Pi 3 kit with a GrovePI+ shield. This agent provides temperature and humidity with a similar sensor as the ESP 8266 module, but also has the additional facility of visualising health via 3 LEDs and an RGB Display.

The simulation agent replicates the sensor agent and status and sensor agent by producing both temperature and humidity values for each, and the actuator representation for the status and sensor agent. The simulation agent initialises the messaging system and begins broadcasting temperature and humidity information. The simulator listens for status updates in parallel with the broadcast functionality that sends temperature, humidity and status information.

The next section will discuss the results of testing the prototype.

4 Results

To determine the effectiveness and success of a system, the system needs to be tested and checked against requirements and the design goals. Section 4 details the aspects that were tested for in the DEASP, and the test cases that were undertaken to determine whether the DEAM fulfils the requirements of a loosely coupling agent functioning from an environment.

As part of the testing of the DEAS, a full set of logs were produced which recorded each of the requests passed through the messaging system as well as the time it took to process the requests. DEAS successfully completed the test cases and demonstrated the DEAS can provide the functionality of a FIPA, KQML-based agent system while using REST and JSON. The DEAS was an effective solution to solving the specific scenario although has numerous gaps that would need to be addressed.

An issue identified during execution was the dependency on the successful running of the agent directory to perform security authentication. This dependency on a service for authentication allows for the system to scale in a more granular fashion, but creates more points of failure.

Language and ontology terms remain a challenge, as a dictionary that covers the lexicon does not exist. `s new ontology mechanism which automatically learns about the mappings, similar to the one described by Tao et al. (2017) in their paper Ontology-based data semantic management and application in IoT- and cloud-enabled smart homes would help resolve this problem.

The current framework requires agents to be configured with endpoint information which reduces flexibility of the system. A solution would be to leverage UDP protocol or similar mechanism to broadcast endpoint information.

The prototype proves that it is possible to create a framework that can fulfil the requirements of an agent framework, although there are still several challenges to be addressed to ensure the provision of a more real-world workable solution. The model was effective in addressing the scenario in that all test cases passed and performance was acceptable for a near real time system.

5 Conclusion

The research focused on the problem of loosely coupled agent frameworks. The dissertation has presented a literature review, a model (DEAM) and a prototype (DEAS). The research covered the current state of the art of agent theory and multi-agent systems as well as the various standards that exist today. The model provided a structure to build the prototype, and the prototype detailed the implementation of a loosely coupled framework.

The model defined a structure based on the FIPA standard, and leveraged modern development technologies such as HTTP / REST and JSON protocols. The FIPA standard was found to provide a sound basis, as well as leveraging KQML as the foundation of a communication language. KQML was easily translated from a LISP-based language to JSON, although there were some areas in the prototype that were not catered for by KQML such as the management of agents in the agent directory.

The DEAM and DEAS have proved that it is possible to build a completely decoupled multi-agent framework. This framework can support any operating system, environment, or agent execution environment. At this point in time, there is no widely adopted agent standards body such as the IEEE or the W3C. There is no standardisation on transport mechanism, and many of the proposed systems rely on different transport level protocols, which creates an additional barrier to integration.

DEAS used HTTP / REST and JSON and the performance demonstrated a reliable and suitable standard-based technology can be used in the future. The addition of SWAGGER on top of JSON could be considered for the more machine-based discovery of services. This would add to the level of automation and enable the messaging service to crawl for the services within the environment dynamically.

A loosely-coupled agent framework is completely possible, although there would be various challenges to face, specifically around the adoption of the framework in the industry, leveraging or establishing standards which are consistent, and ensuring that the framework is a living and growing artefact that will accept change as new technologies emerge. If a universal framework is the goal, more collaborative work would have to be done to further extend and improve on the existing standards to unify the separate frameworks into a single agreed-upon framework. The framework proposed demonstrates that a universal decoupled platform is possible that can handle the performance requirements of a real-time multi-agent system where the agents are loosely coupled to the environment.

6 Bibliography

1. Cohen, W., Koedinger, K. R., Li, N., Matusda, N.: Integrating representation learning and skill learning in a human-like intelligent agent. Artificial Intelligence 219(1), 67-91 (2015)
2. Bellavista, P., Corradi, A., Stefanelli, C.: Middleware services for interoperability in open mobile agent systems. Microprocessors and Microsystems 25(2), 75-83 (2001)
3. Catterson, S., McArthur, D., Davidson, E., M., V., Dimeas, A., Hatziargyriou, N., Ponci, F., Funabashi, T.: Multi-Agent Systems for Power Engineering Applications Part II: Technologies, Standards, and Tools for Building Multi-agent Systems. IEEE Transactions on Power Systems 22(4), 1753-1759 (2007)
4. Corradi, A., Cremonini, M., Stefanelli, C.: Locality Abstraction and Security Models in a Mobile Agent Environment. In : Proceedings of the 7th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Washington, DC, USA, pp.230-235 (1998)
5. Cucurull, J., Marti, R., Navarro-Arribas, G., Robles, S., Borrell, J.: Full mobile agent interoperability in an IEEE-FIPA context. The Journal of Systems and Software 82(12), 1927-1940 (2009)
6. Farahvash, P., Boucher, T.: A multi-agent architecture for control of AGV systems. Robotics and Computer-Integrated Manufacturing 20(6), 473-483 (2004)
7. Helleboogh, A., Vizzari, G., Uhrmacher, A., Michel, F.: Modeling Dynamic Environments in Multi-agent Simulation. Autonomous Agents and Multi-Agent Systems 14(1), 87-116 (2007)
8. Higashino, M., Takahashi, K., Kawamura, T., Sugahara, K.: Mobile Agent Migration Based on Code Caching. In : 2012 26th International Conference on Advanced Information Networking and Applications Workshops, Fukuoka, Japan, pp.651-656 (2012)
9. Higashino, M., Osaki, S., Otagaki, S., Takahashi, K., Kawamura, T., Sugahara, K.: Debugging Mobile Agent Systems., Vienna, Austria, pp.667-670 (2013)
10. Islam, N., Mallah, G., Shaikh, Z.: FIPA and MASIF Standards: A Comparative Study and Strategies for Integration. Proceedings of the 2010 National Software Engineering Conference 7, 1-6 (2010)
11. Kone, M., Shimazu, A., Nakajima, T.: The State of the Art in Agent Communication. Knowledge and Information Systems 2(3), 259-284 (2000)

12. Milojicic, S., Breugst, M., Busse, I., Campbell, J., Covaci, S., Friedman, B., Kosaka, K., Lange, D., Ono, K., Oshima, M., Tham, C., Virdhagriswaran, S.: MASIF: The OMG Mobile Agent System Interoperability Facility. Personal and Ubiquitous Computing - PUC 2(2), 50-67 (1998)

13. O'Reilly, G., Ehlers, E.: Synthesizing Stigmergy for Multi Agent Systems. In : Agent Computing and Multi-Agent Systems: 9th Pacific Rim International Workshop on Multi-Agents, PRIMA 2006, Guilin, China, August 7-8, 2006. Proceedings, Guilin, China, pp.34-45 (2006)

14. O'Reilly, G.: Utilizing Multi-agent technology and swarm intelligence for automatic frequency planning. (Accessed 2007) Available at: http://hdl.handle.net/10210/5670

15. Perdikeas, M., Chatzipapadopoulos, F., Venieris, I., Marino, G.: Mobile agent standards and available platforms. Computer Networks 31(1), 1999-2016 (1999)

16. Schoeman, M., Cloete, E.: Architectural Components for the Efficient Design of Mobile Agent Systems. In : Proceedings of the 2003 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement Through Technology, Johannesburg, pp.48-58 (2003)

17. Stone, P., Veloso, M.: Multiagent Systems: A Survey from a Machine Learning Perspective. Autonomous Robots 8(3), 345-383 (2000)

18. Tao, M., Ota, K., Dong, M.: Ontology-based data semantic management and application in IoT- and cloud-enabled smart homes. Future Generation Computer Systems 76(1), 528-539 (2017)

19. Tung Do, T., Faulkner, S., Kolp, M.: Organizational Multi-agent architectures for Information Systems. In : International Conference on Enterprise Information Systems - ICEIS , Angers, France , pp.89-96 (2003)

20. Urra, O., Ilarri, S., Trillo-Lado, R.: An approach driven by mobile agents for data management in vehicular networks. Information Sciences 381(1), 55-77 (2017)

21. Valckenaers, P., Sauter, J., Sierra, C., Rodrigues-Aguilar, J.: Applications and environments for multi-agent systems. Autonomous Agents and Multi-Agent Systems 14(1), 61-85 (2007)

22. Weyns, D., Omicini, A., Odell, J.: Environment as a first class abstraction in multiagent systems. Autonomous Agents and Multi-Agent Systems 14(1), 5-30 (2007)

23. Weyns, D., Helleboogh, A., Holvoet, T., Schumacher, M.: The agent environment in multi-agent systems: A middleware perspective. Multiagent and Grid Systems 5(1), 93-108 (2009)

24. Yang, J., Zhu, F., Yu, K., Bu, X.: Observer-based state estimation and unknown input reconstruction for nonlinear complex dynamical systems. Communications in Nonlinear Science and Numerical Simulation 20(3), 927-939 (2015)

25. Yang, Y., Sung, T.-W., Wu, C., Chen, H.-Y.: An agent-based workflow system for enterprise based on FIPA-OS framework. Expert Systems with Applications 37(1), 393-400 (2010)