Representing and Developing Knowledge using Jason, CArtAgO and OWL

Antonio Chella*, Francesco Lanza*, Valeria Seidita*
*Dipartimento dell'Innovazione Industriale e Digitale
Universitá degli Studi di Palermo
Email: name.surname@unipa.it

***Abstract*—Contexts where agents and humans are required to collaborate and cooperate in a human-like fashion are complex systems where a high degree of self-adaptability of every component is demanding. A fundamental ingredient when developing and implementing this kind of systems is the knowledge representation. Knowledge of the goals, the environment, other agents' capabilities and task and of itself, is crucial in deciding which action to perform to reach an objective and to behave in a self-adaptive way. The problem of knowledge modeling and representation becomes more and more urgent if the agents' operation domain changes at runtime. Knowledge has to be updated and handled while the system is in execution. In this paper, we present a way for implementing a controlled semantic system to manage the belief base of a multi-agent system at runtime. Our work is based on the development of a specific approach for interfacing Jason, CArtAgO and Jena; the knowledge base representation employs OWL Ontology.**

I. Introduction

Today software systems are required to perform tasks not often known at design time, they should be composed of increasingly autonomous and self-adaptive entities acting in a continually changing environment. This is the case when human and agents interact as they were in a team to collaborate and cooperate towards a common and shared objective. Cooperation and collaboration should imply sharing knowledge about elements in the environment, actions, tasks, capabilities; deciding whether to perform an action by itself, delegating to the other or not performing an action as well; communicating basing on the same elements of the world; monitoring and anticipating how, and whether, each single action may change the state of the world. Also considering, in so doing, that the acting agent itself is an element of the environment. The latest consideration includes some inclusive point of view on the environment in contrast to the egocentric one, more used in the systems of a few years ago. The agent is part of the environment and it perceives itself in the same way it perceives all the other objects or elements. This assumption affects developing and implementing the part of the system devoted to the decision process.

Purposeful interactions in human-agent interaction entail concepts like autonomy, self-organization and self-adaptation. Agents possess features allowing to manage the before said concepts. For instance, the BDI agent paradigm implements a decision-making process based on *practical reasoning* that is very similar to the human reasoning process. Indeed, through deliberation and means-ends reasoning, agents may act as a human. A human decides how to act to realize his intentions based on the state of affairs around him. The state of affairs may concern the objects of the world, the internal state of the human in self and what the human infers from the observation of other human beings' behavior.

Having a way of handling knowledge is of fundamental importance in the decision process. Currently, the known theoretical and technological means allow us easily and efficiently managing systems that self-adapt to changing situations. The BDI theory and the related programming language such as AgentSpeak and Jason along with its reasoning cycle (Fig. 1) allow a runtime mapping of intentions and events. Also, plans selection from the plan library[1] to reach a goal even in a situation not entirely known at design time.

However, situations or domains where new stimuli from the outside cause changes in the state of the environment and new elements emerge during agents mutual interactions with the environment imply several problems in the implementation phase. Moreover, these elements have been taken into account at runtime to convey the decision-making process of agents operating in that environment. Normally, a common goal, as well as a set of plans to achieve it, are identified at design time, but the interaction of the agent with the environment and with the human being makes sure that the operating conditions change unpredictably. This fact happens when the interaction with the environment itself brings out new terms of operability that must be worked out to decide what action to take. Generally, when a team is made up of only humans, they choose actions from their experience, the knowledge they have of the other team members, the trust they place in the other team, their emotional state and the anticipation of the actions of others.

Reporting these behaviors on agents is a challenging task and the case of human-agent interactions is an excellent scenario to investigate and analyze problems related to knowledge management. We need a way to implement a reasoning cycle on a knowledge base changing and updating at runtime.

Contribution and outline of the paper. In this paper, we propose a means for representing and handling agents' knowledge by employing BDI agents theory and OWL. In particular, we create a Jason agent that acts as an interface towards OWL

[1]For a deep review on these concepts refer to BDI theory and Jason [4, 3, 2]

using Jena through CArtAgO artifacts and Jason's internal actions. The rest of the paper is organized as follows: in section II we better explain the motivation of our work; in section III we illustrate the proposed approach and, finally, in section IV we draw some conclusions.

II. Need for knowledge representation

Storing data, information and relevant acquisitions let humans learn, reason on facts and decide the right action to pursue an objective. In the same way, to build agents able to work in a specific environment, it is useful to give them the required know-how for doing actions and getting the jobs done. Implementing this feature on agents involves the definition of a suitable data structure that can emulate a memory architecture. For instance, developing multi-agent systems requires the development of a memory system for agents that allows them understanding and reasoning about the surrounding world.

A promising approach might be to implement agents using the BDI model. This approach descends by the philosopher Bratman [4]; he applied theories that govern human practical reasoning. Several programming languages implement the BDI model; in particular, AgentSpeak(L) is the most famous language that lets us implement a cognitive cycle that emulates the practical reasoning.

The Jason language extends AgentSpeak(L). A BDI agent can sense the environment and at the same time update the internal belief base accordingly. Jason has internal methods to represent belief base, goals and plans to reach it. Other information about Jason can be retrieved in [2][3]. Modeling the environment in a way useful for Jason agents may be done by the CArtAgO [14] framework. CArtAgO is a general purpose framework based on Agents & Artifacts metamodel. CArtAgO lets us develop a virtual environment based on artifacts. Thus, BDI Agent-Oriented Programming claims its place as the most useful paradigm to develop humanized agents. This model lets us talk about computer programs as cognitive agents owning a *mental state*. Therefore, beliefs, desires and intentions characterize an agent.

In this paper, we focus on a specific characteristic of agents, the *belief*. Beliefs are information that agents own about the surrounding environment. In general, a belief could be out of date or inaccurate. To define a good data structure with the aim to emulate a belief base representation system, we need to understand how a Jason belief base is organized. Looking at the Fig. 1, we can observe the role of the belief base in the reasoning cycle. All percepts are computed using two important functions *buf* and *brf*. Each one contributes to adding, removing or updating knowledge stored in the *Knowledge Base*; Fig. 2 shows a UML representation of the Belief Base interface and classes. This class presents several predefined methods that are devoted to managing the belief base; for instance, *add* and *remove* let add or remove literals hence update the belief base.

In the following section, we detail how we developed a multi-agent system able to create, add, remove, or better manage knowledge, also including semantic and additional properties, by a qualified agent realizing our approach.

What we claim in this paper is that representing knowledge and implementing methods for managing the knowledge base help developers to give life to entirely autonomous and self-adaptive agents.

In order to understand this work, it is worth to note the following definitions of the four different levels of adaptivity by Qureshi et al. [12]:

- Type I - self-adaptive systems of the first type are able to react to changes following the behavioral model given by designers, well specific decision points trigger actions on the basis of perceptions and available knowledge on the environment;
- Type II - systems of this type own many strategies to face changes, the best strategy is chosen at runtime and may impact on non-functional requirements and is the result of a compromise that considers perceptions and available knowledge also on special requirements;
- Type III - is applied to systems operating in not totally known environment, strategies are not established at design time but assembled at runtime according to the actual execution context;
- Type IV - is the most similar to biological systems, these kind of systems are able of self-modifying their specification and generating strategies from scratch to respond to changes.

Each level implies how much adaptivity agents have to acquire and learn from the surrounding world to operate their intentions into the environment.

In [15] authors describe a meta-model containing the main abstractions for designing these type of intelligent systems. In [10][8][5] other approaches dealing with theoretical aspects in developing self-adaptive systems are discussed. However, to the best of our knowledge, this research context still lacks proposals and solutions about implementation issues above all with regards to knowledge representation.

As Tulving states in [17] about human memory, a complete memory system equips two different type of memory system, *episodic memory* and *semantic memory*.

The semantic memory allows retrieving information that has not been directly stored in the global knowledge base system. Semantic memory does not save percepts but instead cognitive links to input signals. The semantic memory system is much less susceptible to losing information concerning the episodic one. Episodic memory stores temporal information about occurring episodes or specific events and merges them, when possible, to the temporal-spatial relation. The episodic memory system may store perceptible event in terms of properties, characteristics or attributes. Hence, to handle efficiently system knowledge we need an architecture letting the agents storing all the information about perceived data and at the same time, create and/or maintain semantic mapping among them.

In this paper, we try to solve this problem with a memory architecture realized by using an OWL Ontology. The OWL Ontology is a semantic ontology based on a markup language

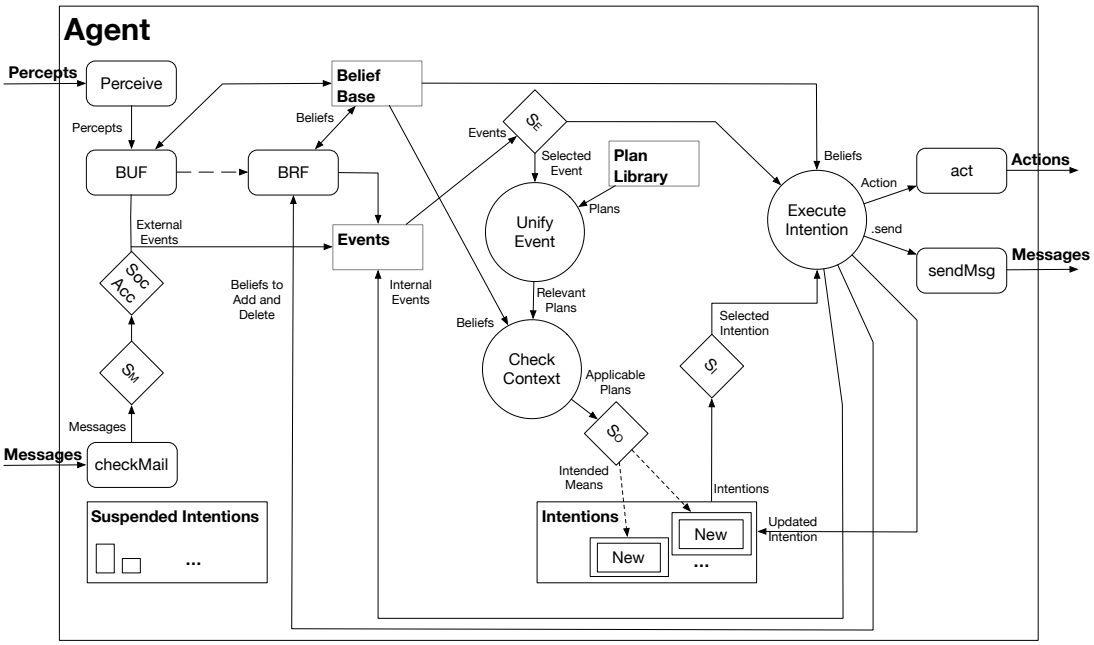

Fig. 1. Jason reasoning cycle redrawn from [3].

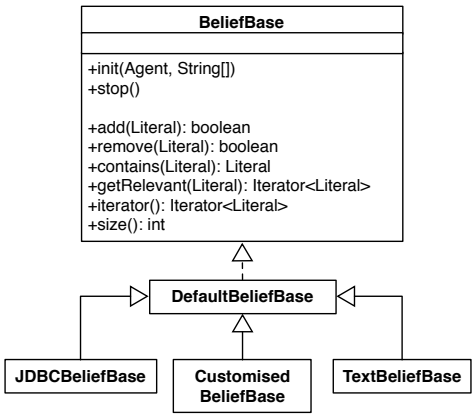

Fig. 2. Jason Belief Base UML redrawn from [3].

generally used to publish and share ontologies on the web [1]. Using OWL allows representing means and semantic of terms using vocabulary and relations between them. Apache Jena [9] is used to query the OWL ontology.

In order to implement the memory system, therefore, we propose a non-invasive method where, studying the mechanisms that govern a belief base, we suggest an agent approach to represent and develop knowledge with all its semantic mechanism without modifying any internal classes.

III. Making Runtime Knowledge

In this section, we explain how to implement a semantic and episodic memory architecture in a multi-agent system based on the Jason interpreter [2] and the CArtAgO framework [13]. We implement in CArtAgO a specific scheme where the memory structure is based on OWL Ontology [1] to made runtime knowledge. Fig. 3 shows a Jason agent system using CArtAgO for handling knowledge. From an ***— implementation —*** point of view, fundamental elements are the agent-types and the workspaces. An agent-type, as the name suggests, is a user-defined type, written in AgentSpeak Language; it contains the agent's definitions and a series of initial conditions. In Jason, the agent-type corresponds to a file with the *.asl* extension. An agent-type file presents a specific structure for the definition of beliefs, desires/goals the agent has to achieve and the list of plans/intentions the agent can use to reach all its desired goals. The agent-type contains all methods that could be used from the agent written by the developer; each plan is composed of several parts, the most important are: *(i)* event-trigger, *(ii)* context and *(iii)* the body of the plan. The event-trigger may be considered as the name of a plan or well, the event that triggers the plan after the context has been evaluated.

The process in charge of assessing the context is a unifying process; it considers all the beliefs considered true to activate the task. The body of the plan may contain internal actions, other kinds of action (such as CArtAgO's artifact) or other plans; another essential factor to consider is the belief source. During the reasoning cycle (Fig. 1), each belief has to be evaluated within a specific context to activate a plan, hence it is essential to know the information source (the agent itself, any other agent or the environment through artifacts); this fact becomes hard to handle at runtime in a changing environment.

The CArtAgO Framework realizes artifact-based working environments in a multi-agent system (MAS)[14]. The entire framework makes extensive use of *artifacts* as abstractions to handle tools, objects and resources belonging to the surrounding environment. Tools, objects and resources may be used and manipulated by agents to solve several tasks as cooperations,

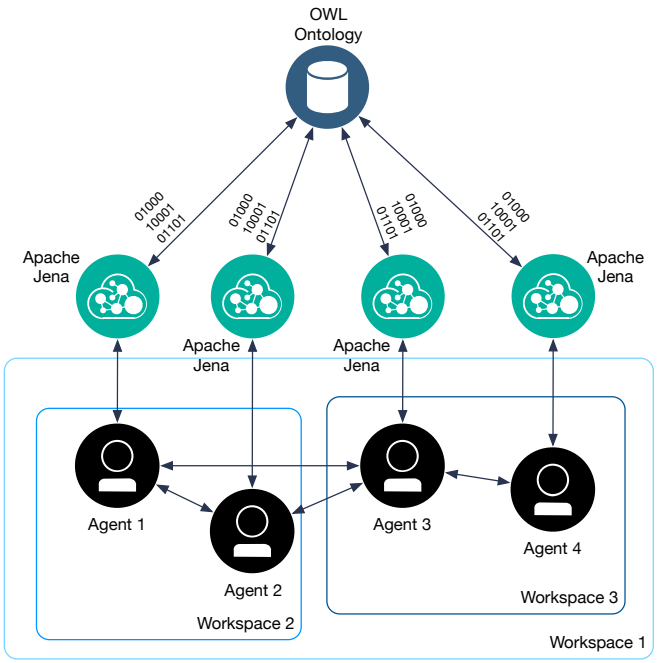

Fig. 3. Using an OWL Ontology inside a multi-agent system

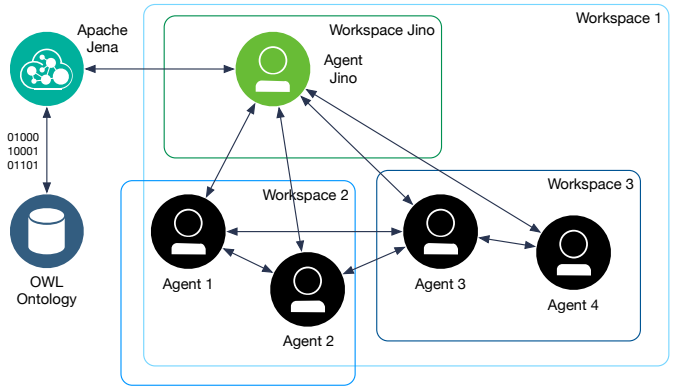

Fig. 4. Proposed approach to implement semantic and episodic memory system using Jason-CArtAgO e Jena-OWL

actions or simply perceptions. Thus, during the definition of a JaCa system, the developer has to write a configuration module that contains all information about agents and workspaces. He has to do that for each agent of the system thus losing the possibility to coordinate data flow among agents at runtime. This situation is illustrated in Fig. 3, here all the agents can interface with OWL ontology using the Jena Framework and to manage knowledge acquisition but, during the system execution, each agent would activate the Jason reasoning cycle with the consequence to not be able to implement self-adaptive cognitive agents.

Fig. 3 shows a general approach; each agent takes part in the system and has to implement and use the Jena framework within itself, for example using Internal Actions. Agents use Jena without centralized architecture and this affects the efficiency of the system. For instance, in [16], the argonaut prototype lets agents consult ontologies using Jena Framework implementing for each some defined internal actions. This specific distributed approach has been proven useful in a scenario where fundamental characteristics of cognitive systems working in a runtime application are not evaluated. Our intention is, instead, to realize a memory system for cognitive architectures working in a dynamic and unknown domain and where the real time is the most important characteristic.

We developed a knowledge system that can perfectly emulate the mechanism of a generic belief base such as the Jason one; it works well at runtime and at the same time guarantees inferring logical consequences from a set of axioms. Generally, a multi-agent system organizes memory in specific structures with the purpose to let agents gain information. A Jason multi-agent system implements a cognitive agent that uses the BDI model proposed in [4]. Here, designer handles the logic of programming based on the goal, beliefs, desires, and intentions of the agent. As already said, Jason beliefs are the core of our approach; we structured the multi-agent system with a centralized agent with the aim to connect the OWL ontology with the rest of the system (Fig. 4).

In so doing we define, or declare, multiple workspaces containing one on more agents; in each workspace, agents use one or more artifact to realize itself. The artifact of a workspace could be used only by the agents joined to the considered workspace. Thus all retrieved information perceived by the related artifact, are seen only by the joined agent. Employing workspaces allows developing security policies on information in the multi-agent system. Furthermore, every artifact can be used only by the agents belonging to the workspace that contains it. The default implementation of workspace contains predefined artifacts that provide functions at a low level to the agents. Artifact represents tools and functionalities letting the agent handle objects or in general resources in the environment. An artifact is a .java file with the aim to implements methods that could be used in AgentSpeak plans by the Jason agent.

The system represented in Fig. 3 contains three different workspaces. The workspace 1 contains tools and methods that could be used from every agent in the system; this means that the knowledge acquired using these tools could be accessible by all or could be acquired from every agent that uses the artifact in the plans defined in .asl file.
The proposed scheme implements, in each agent, the functions interfacing with apache Jena; concerning its complexity, this means that a connection through the ontology for each agent will exist. The only form of communication is related to the transmission and the execution of the regular plan.
In the workspace 2, such as in the workspace 3, agents communicate with each other with communicative internal action as
`.broadcast` and `.send` to catch or retrieve information from and to the agent society. Sending or forwarding perceptions, or in general beliefs, also increases the complexity

of the plans an agent has to execute. Indeed, the developer has to write plans handling messages from the senders (that contains the knowledge) to the receivers. The work proposed with the scheme in Fig. 4 uses a different method to implement semantic technologies in multi-agent systems. We realized this approach after having used the common model in Fig. 3 with a toy ontology and having determined how much slow is.

A. Proof of Concepts

Let us suppose agents with the goal of understanding objects not present in their knowledge base. As humans do, agents start from whatever they know and through interactions with humans. The agent system developed using the first approach (Fig. 3) revealed to be slow and hard to implement cause the need to handle all possible occurring cases. Also, agents' connection latency and the delay on possible plans actuation, entail a series of problems that could create inconsistency in the decision process. Some errors propagated in some functionalities of the semantic structure because of the high latency that severely decreased performances and did not let agents adapt to changing situations. Indeed, each agent that got a deprecated belief could not know which version of belief was correct. Moreover, using CArtAgO produced object properties that Jason converted in beliefs erroneously affecting the decision process.

The crucial difference between the two approaches is in the memory management system. We introduced a specialized agent called Jino. Jino has a specific agent-type that considers a series of plans composed to handle the memory system. Jino lets us manage the episodic and the semantic memory but also all the information acquired by beliefs. Through the communication system, Jino maintains the system synced with all the members. The value of beliefs is updated at runtime and when an external agent asks for a specific belief, Jino knows whether this is reliable or it is better to check into the ontology to retrieve a belief newer version.

Within the Jino's workspace, the agent is equipped with all tools necessary to manage the perceptions coming from agents with the specific role of perceiving the environment. The communication system always guarantees the direct connection between Jino and each agent. In order to acquire runtime knowledge, Jino gets started each time that a belief changes or an event occurs; this fact lets Jino add, remove or update some beliefs and manage others, at the same time. Thus, Jino manages changing knowledge without employing the ontology reasoner but only its working memory; this fact lets the system use resources at runtime and having beliefs updated and synchronized in the working memory without waiting for the reasoner (Fig. 5).
Jino, across the communication system, informs all agent-society about changes using the broadcast method. Whenever Jino believes that a certain belief is necessary only for a specific agent or a group of agents, it takes over to communicate only to the interested agent/group using the `.send` method. Moreover, when new knowledge is acquired, Jino firstly saves the result on the knowledge base with the aim to construct the

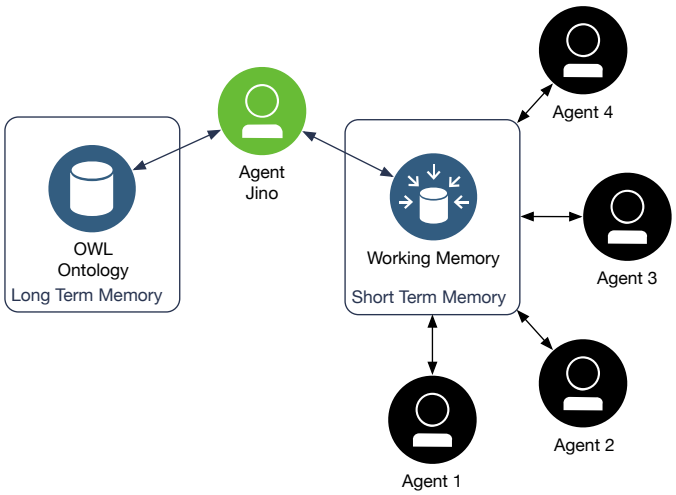

Fig. 5. How Jino works using its working memory.

long-term memory and then lets OWL's reasoner infer new concepts.

Only one connection with the ontology across Jena Framework is present and the reasoner is launched only when it is essential and not when a single agent desires a specific belief. Jino takes advantage of an internal memory representation that makes belief usable when someone asks.

IV. Discussions and Conclusions

Developing applications that make effective use of machine-readable knowledge sources is an important research area. In [11] several aspects regarding agent-oriented programming with underlying ontological reasoning are illustrated.

In this paper, we discussed how to develop and represent knowledge using Jason, CArtAgO, and OWL. The memory architecture, we presented, is based on a semantic representation system commonly used to develop a semantic web application. Semantic web makes extensive use of ontologies, cause they are explanatory about the logical connection between concepts, relations, and properties. OWL could be a good choice as information handler or rather as concept descriptor, relationships between concepts but also specific properties or constraints. We proposed a practical approach to integrating a Jason-CArtAgO multi-agent system with the Jena framework. In Fig. 4, we show a centralized system for handling the knowledge base using a specialized agent; this agent is built with the purpose to manage the system beliefs, in so doing it produces beliefs using the semantic connections retrieved from the Jena framework. To reduce the system complexity and the time for perception processing, we decided to adopt a centralized system. As it can be seen in Fig. 5, we can handle the entire system subdividing it into two parts, a long-term memory and a short-term memory, commonly known as working memory. The long-term memory has been endowed with all the information related to objects and resources perceived using CArtAgO artifacts. Moreover, it is possible to store relations and properties that we can

associate for each prediction; this fact is possible thanks to the logical axioms or better object properties that Jena lets define and retrieve. In the working memory, we managed the current situation model relates to the context in which the agents work. With the use of CArtAgO, we handled the knowledge base using artifacts. When a perception is perceived, the system converts it into Jason's belief and, once it is available, the specialized agent manages and organizes it into a specific form that let other agents ask resources without invoking the reasoner or others complex algorithm to evaluate information. This is the main contribution given by this work.

This specific approach has been proven useful in a scenario where fundamental characteristics of cognitive systems working in a run-time application are strictly evaluated. In this method, we use the useful semantic memory system inside a cognitive architecture where all relative constraints are respected.

In future works, we are planning to extend this module integrating it directly with a set of application that may help the knowledge acquisition module to understand context and catalog perceptions. Moreover, we will map the belief control system of this paper with the Jason reasoning cycle extension [7] for better handling the agent's decision process and we will fill the gap between the cognitive architecture for human-robot interaction developed in [6] and the agents' implementation framework. In conclusion, this work is a part of a a more significant project that focuses on the human-robot teaming research. The approach could be used to do runtime planning using several learning methods. We are going to develop the complete model that also implies the ability of one entity to understand what the other one is going to do. In this way, we aim at implementing human-robot interactions where each involved entity delegates or commits an action. The advantages of combining a multi-agent language as Jason with description logics lie in the reasoning cycle of the agent's expressiveness and consciousness indeed, queries to the belief base are more expressive since this was inferred by ontology and not explicitly written.

Acknowledgment

This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0232.

References

[1] Sean Bechhofer. "OWL: Web ontology language". In: *Encyclopedia of database systems*. Springer, 2009, pp. 2008–2009.

[2] Rafael H Bordini and Jomi F Hübner. "BDI agent programming in AgentSpeak using Jason". In: *International Workshop on Computational Logic in Multi-Agent Systems*. Springer. 2005, pp. 143–164.

[3] Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Vol. 8. John Wiley & Sons, 2007.

[4] Michael Bratman. "Intention, plans, and practical reason". In: (1987).

[5] Yuriy Brun et al. "Engineering self-adaptive systems through feedback loops". In: *Software engineering for self-adaptive systems*. Springer, 2009, pp. 48–70.

[6] Antonio Chella, Francesco Lanza, and Valeria Seidita. "A Cognitive Architecture for Human-Robot Teaming Interaction". In: *Proceedings of the 6th International Workshop on Artificial Intelligence and Cognition*. Palermo, 2018.

[7] Antonio Chella, Francesco Lanza, and Valeria Seidita. "Human-Agent Interaction, the System Level Using JASON". In: *Proceedings of the 6th International Workshop on Engineering Multi-Agent Systems (EMAS 2018)*. Stockholm, 2018.

[8] Rogério De Lemos et al. "Software engineering for self-adaptive systems: A second research roadmap". In: *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 1–32.

[9] Apache Jena. "Apache jena". In: *jena. apache. org [Online]. Available: http://jena. apache. org [Accessed: Mar. 20, 2014]* (2013), p. 14.

[10] Ivan J Jureta et al. "The requirements problem for adaptive systems". In: *ACM Transactions on Management Information Systems (TMIS)* 5.3 (2015), p. 17.

[11] Alvaro F Moreira et al. "Agent-oriented programming with underlying ontological reasoning". In: *International Workshop on Declarative Agent Languages and Technologies*. Springer. 2005, pp. 155–170.

[12] Nauman A Qureshi et al. "Towards a continuous requirements engineering framework for self-adaptive systems". In: *Requirements@ Run. Time (RE@ Run-Time), 2010 First International Workshop on*. IEEE. 2010, pp. 9–16.

[13] Alessandro Ricci and Andrea Santi. *Cartago by example*.

[14] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. "CArtAgO: A framework for prototyping artifact-based environments in MAS". In: *International Workshop on Environments for Multi-Agent Systems*. Springer. 2006, pp. 67–86.

[15] Luca Sabatucci, Valeria Seidita, and Massimo Cossentino. "The Four Types of Self-adaptive Systems: A Metamodel". In: *International Conference on Intelligent Interactive Multimedia Systems and Services*. Springer. 2017, pp. 440–450.

[16] Douglas Michaelsen da Silva and Renata Vieira. "Argonaut: Integrating jason and jena for context aware computing based on owl ontologies". In: *Agent, Web Services, and Ontologies Integrated Methodologies* (2007), p. 19.

[17] Endel Tulving et al. "Episodic and semantic memory". In: *Organization of memory* 1 (1972), pp. 381–403.