Applied Ontology 12 (2017) 157–188
DOI 10.3233/AO-170182
IOS Press

Model-driven engineering of multi-agent systems based on ontologies

Artur Freitas *, Rafael H. Bordini and Renata Vieira
Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil
E-mails: artur.freitas@acad.pucrs.br, rafael.bordini@pucrs.br, renata.vieira@pucrs.br

Abstract. Model-driven engineering provides abstractions and notations to improve the understanding and to support modeling, coding, and verification of applications for specific domains. Ontologies, on the other hand, provide formal and explicit definitions of shared conceptualizations and enable the use of semantic reasoning. Although these areas have been developed by different communities, significant synergy can be achieved when both are combined. These advantages can be explored in the development of multi-agent systems, given their complexity and the need for integrating several components that are often addressed from different angles. This work investigates how to apply ontologies for agent-oriented software engineering. Initially, we present a new modeling approach where multi-agent systems are designed by instantiating our proposed ontology. An additional contribution is a tool that uses instantiated ontological designs to generate programming code for such systems. Several advantages can be obtained from the application of our ontology-based approach, in terms of specification, development, and verification of agent-oriented software, as indicated by the experiments we have carried out.

Keywords: Model-driven engineering, ontology, multi-agent system, agent-oriented software engineering

Accepted by: Roberta Ferrario

1. Introduction

Some of the key issues in developing Multi-Agent Systems (MAS) are: *(i)* techniques for integrating design and code; *(ii)* extension of agent-oriented programming languages to cover certain aspects that are currently weak or missing (e.g., social concepts, and modeling the environment); and *(iii)* development of debugging and verification techniques, with a particular focus on using model checking in testing and debugging, and applying model checking to design artifacts (Bordini et al., 2006). Our research is mainly concerned with two of these problems. First, it directly addresses the integration of design and code, which is demonstrated in our approach to generating code from models represented as ontology instantiations. Second, our approach enables features of semantic reasoning and verification techniques on top of such instantiated ontological models. Mechanisms for solving the aforementioned issues are claimed to be crucial for the practical adoption and deployment of agent technology (Bordini et al., 2006). Thus, the investigation of software development methodologies should provide interesting answers, solutions and improvements for problems in Agent-Oriented Software Engineering (AOSE).

Model-Driven Engineering (MDE) employs *models* as the cornerstone of software development processes (Gascueña et al., 2012) in order to improve productivity, portability, interoperability, maintenance,

*Corresponding author: Artur Freitas, Postal Code 90619-900, Av. Ipiranga, 6681 - Porto Alegre, Brazil. Tel.: +55-51-3320-3500; E-mail: artur.freitas@acad.pucrs.br.

and so on. MDE is a research area that provides abstractions and notations to improve the understanding and support the modeling of applications for specific domains. These advantages can be employed for developing MAS given their complexity and the need for integrating several components that are often addressed from different angles. For example, the JaCaMo (Boissier et al., 2013) framework for MAS programming combines three separate technologies: Jason (Bordini et al., 2007) for coding the dimension of autonomous agents in AgentSpeak, CArtAgO (Ricci et al., 2006) for programming the environment as artifacts in Java, and Moise (Hübner et al., 2010) for specifying MAS organizations. Currently, as the programmer has three distinct starting points to code the MAS, this makes it desirable to have a single, unified and comprehensive meta-model that combines these dimensions. Besides filling the gaps between design and development, this modeling framework can be the basis of features such as code generation, support during programming, and reasoning to analyse a given system implementation.

In this context, modeling approaches are present in most agent methodologies (Gascueña et al., 2012) and MDE techniques for AOSE emerges naturally. A typical example of MDE for MAS is Prometheus (Padgham and Winikoff, 2002); however, differently from our work, it does so without exploring any use of ontology as part of its approach. In fact, there are already modeling frameworks contemplating more than one dimension of MAS (Gascueña et al., 2012; Padgham and Winikoff, 2002; Uez and Hübner, 2014), but without using ontology, semantic reasoning or employing the model during the programming step. In the literature, when an ontology is used to model a MAS, only a part is modeled, such as the environment (Okuyama et al., 2006) or organization (Zarafin, 2012). Currently, the use of ontologies to model integrated frameworks that consider the co-specification of different MAS dimensions is still an open issue (Freitas et al., 2014). We observe two additional points: *(i)* MDE and ontologies share a number of principles and goals; and *(ii)* there is much work in combining ontologies and MAS. These synergies led us to propose and investigate the use of ontology for MDE of MAS, which resulted in a model-based approach that simultaneously covers all these issues. Moreover, we propose a tool to load an ontological instantiation in order to generate code for the different MAS platforms that are part of JaCaMo.[1] Our idea was derived when considering the research comparing ontologies with MDE (Atkinson et al., 2006; Kappel et al., 2006; Staab et al., 2010), and also observing that when ontologies are used in MAS it is usually with goals other than modeling MAS (Klapiscak and Bordini, 2008; Mascardi et al., 2014; Moreira et al., 2005).

We started our investigation by systematically surveying the literature for current approaches and advantages that ontologies can provide for MAS. Then, we explored uses and benefits of applying ontology in the global modeling of MAS where such framework has to offer support also for the programming phase. A particular feature of our approach is that we are proposing the use of ontology as the basis for modeling, development and verification of MAS. To the best of our knowledge, there is no previous work that has achieved all that. Therefore, our main contributions are threefold. First, we argue that the use of ontologies improves the modeling, programming and verification of MAS, which emerges from our analysis of the state of the art and research directions on the integration of MDE, ontology and MAS. Second, we propose and investigate an ontology that considers the global modeling of a MAS, as it incorporates the agent, environment and organization dimensions. This approach covers MAS design and development as a whole in an integrated formalism. Third, we present our developed MDE tool (which consists of an Eclipse plug-in) to support MAS programming in JaCaMo based on an instantiation of our proposed ontology. This tool is our prototype for providing features such as drag-and-drop and auto-complete to generate MAS code according to what was specified in a given ontology instantiation.

[1] The following analogy may improve the reader's understandability: the ontology proposed in this work acts as a meta-model, and its instantiation represents a model, which in our case is applied to specify a MAS implementation project for JaCaMo.

This paper is structured as follows. Section 2 presents the background knowledge for the rest of our work, which addresses the areas of MAS, MDE, and ontologies for MAS. Section 3 explains related work through the interconnection of these areas, emphasizing previous ontologies related to MAS aspects: agents, organizations and environments. Section 4 explains our ontology-based MDE approach for MAS modeling; Section 5 shows a tool we developed for MAS programming; and Section 6 presents an empirical evaluation of our proposed ontology and tool. Then, our final remarks are given in Section 7.

2. Background

The design of complex systems should consider models that are clear to communicate, provide support during programming, and allow reuse and reasoning over the specification (Freitas et al., 2015). Therefore, we investigate the use of ontologies for achieving such goals, as they can also offer code generation features and help in organizing the many concepts which are involved in MAS's modeling, development and verification. Since ontology will be playing the role of a meta-model for MAS, this Section briefly explains the main issues of MAS, MDE, and ontology that have led us to our proposal.

2.1. Multi-agent systems

Agents are reactive systems that can independently determine how to best achieve their goals and perform their tasks (Bordini et al., 2007) while demonstrating properties such as autonomy, reactivity, proactiveness and social ability. Agents are situated in an environment, where they can perceive and modify it, and they should be able to exchange information, cooperate and coordinate activities. MAS development integrates aspects from different dimensions (e.g., agent, environment and organization) that are addressed by different technologies. For example, MAS programming in JaCaMo (Boissier et al., 2013) requires the development of code in Jason, CArtAgO and Moise. These three distinct formalisms and starting points to develop the MAS make desirable a single model combining all MAS's dimensions which can also offer an abstraction to such programming platforms. However, current AOSE models and methodologies (e.g., Prometheus (Padgham and Winikoff, 2002)) are deficient in at least one of the following areas of MAS development (Tran and Low, 2008): agent internal design (design of agent mental constructs such as beliefs, goals, plans and actions), interaction design (design of interaction protocols and exchanged messages) or organization modeling (design of acquaintances and authority relationships amongst agents or agents' roles). One problem in this scenario is that such characteristics can appear only hard-coded in agent-oriented programming platforms without being fully and explicitly contemplated in high-level models of MAS.

The development of MAS, like developing any software system, encompasses activities traditionally classified into three broad areas (Bordini et al., 2006): software engineering (e.g., requirements elicitation, analysis, design), implementation (using some suitable programming language), and verification/validation. In current practice, the way in which a MAS is typically developed is that the developer designs the agent organization and the individual agents, then takes the detailed design and manually codes the agents in some programming language. The problem with developing the implementation completely manually from the design is that this creates the possibility for the design and implementation to diverge, which makes the design less useful for further work in maintenance and comprehension of the system. Thus, it is desirable to have code and design being seen as different views on what is really a single conceptual activity. The key thing that should be done with respect to this issue is developing techniques and tools that allow for designs and code to be strongly integrated with consistency checking

and change propagation (Bordini et al., 2006). Among the possible approaches for eliminating the "gap" between code and design there are: *(i)* the generation of code from design; and *(ii)* the extraction of changes in design/code for applying to the other (Bordini et al., 2006). To consider useful the generation of code from design, techniques are required to ensure the continued consistency of design and code as one or the other is changed. Thus, such techniques must be developed to link one particular design notation with one target programming language.

As design notation, UML (Unified Modeling Language), in its original form, provides insufficient support for modeling MAS (da Silva and de Lucena, 2003), such as the ones developed in the JaCaMo programming platform (Boissier et al., 2013). The coding of MAS in JaCaMo comprises three distinct dimensions, namely: agent, environment and organization. However, these dimensions are not uniformly integrated into a single formalism: agents are programmed in Jason (Bordini et al., 2007) using the AgentSpeak language; environments are coded as artifacts in Java using the CArtAgO API (Ricci et al., 2006); and organizations are specified in Moise (Hübner et al., 2010) using XML. Jason (Bordini et al., 2007) is an AgentSpeak language platform implementation that focuses on agent actions and mental concepts; it is an open source interpreter that offers features such as speech-act based agent communication, plans annotation, architecture customization, distributed execution and extensibility through internal actions. On the environment side of agent systems, CArtAgO (Ricci et al., 2006) is a platform to support the artifact notion in MAS. Artifacts are defined as function-oriented computational devices which provide services that agents can exploit to support their individual and social activities. Lastly, the specification of agents at the organization level can be achieved using an organization modeling language, such as Moise (Hübner et al., 2010). Moise explicitly decomposes the specification of an organization into its structural, functional and normative dimensions. Some problems of this current approach are that: *(i)* programmers have three distinct starting points to code the MAS; *(ii)* it is difficult to keep track of problems because errors in one level can affect the other levels; *(iii)* it becomes cumbersome to explore interconnections between the different layers; and *(iv)* it requires the programmer the knowledge about different paradigms (Freitas et al., 2014). Agent-based systems require adequate techniques that explore their benefits and their peculiar characteristics (da Silva and de Lucena, 2003). To address these issues, we are proposing a unified semantic model which covers these three MAS programming dimensions and integrates these formalisms. An integrated ontology model that represents these MAS dimensions also enables semantic reasoning and can be used as a common vocabulary in agent-oriented programming. These dimensions can interconnect with each other, and reuse relevant concepts from the other MAS dimensions. Each dimension details different aspects, and these interconnections result in an integrated knowledge model with a clear correspondence to an integrated programming platform, such as JaCaMo (Boissier et al., 2013).

Currently, we have separate approaches for addressing the modeling and programming of MAS, resulting in gaps and conceptual divergences in AOSE. In this research we are interested in the use of ontology for representing characteristics of multi-agent dimensions (agent, environment and organization). While JaCaMo (Boissier et al., 2013) is a programming platform that uses three different formalisms for coding MAS, Prometheus (Padgham and Winikoff, 2002) is an agent modeling approach but without applying or exploring any formal ontology as part of its technique.

2.2. *Model-driven engineering*

Models help us understand a complex problem and its potential solutions through abstraction (Selic, 2003). Therefore, it seems obvious that software systems, which are often among the most complex

engineering systems, can benefit greatly from using models and modeling techniques. MDE is related to the design and specification of modeling languages (Atkinson and Kühne, 2003; Staab et al., 2010). In short, MDE provides abstractions and notations for better understanding and easier modeling applications of a specific domain. It is usually applied to: *(i)* produce high-quality results quickly; *(ii)* reuse solutions effectively; *(iii)* specify complex structured information concisely; *(iv)* design rich textual and graphical notations; and *(v)* implement powerful runtime solutions. One of the basic principles of MDE is to consider models as first class entities and any software artifact as a model or as a model element (Bézivin, 2006). MDE employs *models* as the cornerstone of software development processes (Gascueña et al., 2012) in order to improve productivity, portability, interoperability, maintenance and so on. Also, it is possible to gradually evolve an abstract software model into the final product through a process of incremental refinement, without requiring a change in skills, methods, concepts, or tools (Selic, 2003). However, the models must be formally connected to the actual software to ensure that programmers are following the design decisions captured in a model during implementation (Selic, 2003).

Models are used to reason about a problem domain, design a solution in the solution domain, and they are considered effective if (Roebuck, 2012): *(i)* they can serve as a basis for implementing systems; and *(ii)* they make sense from the point of view of a user that is familiar with the domain. To be useful and effective, an engineering model must possess, to a sufficient degree, the following five key characteristics (Selic, 2003): abstraction, understandability, accuracy, predictiveness and inexpensiveness. *Abstraction* is almost the only available means of coping with the complexity of the demand for ever-more sophisticated functionality from our software systems. Since a model is always a reduced rendering of the system that it represents, the essence can be more easily understood when details considered irrelevant for a given viewpoint are removed or hidden. *Understandability* is a direct function of the expressiveness of the modeling form used (expressiveness is the capacity to convey a complex idea with little direct information). A model with *accuracy* must provide a true-to-life representation of the modeled system's features of interest. A *predictive* model should be able to correctly predict the modeled system's interesting but nonobvious properties, either through experimentation (such as by executing a model on a computer) or through some type of formal analysis. Finally, a model is considered *inexpensive* when it is significantly cheaper to construct and analyze than the modeled system.

One of the most relevant MDE concept is the idea of meta-models, which are models to describe models. Meta-models define general concepts of a given problem domain and their relationships (Gascueña et al., 2012), and their advantage in the development process is the higher abstraction level to work with. This role of meta-models is also played by ontologies, as highlights the next Section. Software models capture relevant characteristics of a software artifact to be developed, yet, most often these software models have no formal semantics or the underlying (often graphical) software language varies from case to case in a way that makes it hard if not impossible to fix its semantics (Staab et al., 2010). Also, it is not always clear how the concepts used to express the models mapped to the underlying implementation technologies such as programming language constructs, operating system functions, and so forth (Selic, 2003). This semantic gap is exacerbated if the modeling language is not precisely defined, leaving room for misinterpretation. Since ontology languages are described by meta-models and allow for describing structural and behavioural models, they provide the capability to combine them with software modeling languages (Staab et al., 2010).

Above, we provided several definitions for the most relevant terms backed up by different authors from the MDE community and from the ontology community. However, we believe that it is important to explain our own semantics about these terms. We consider both a meta-model and a modeling language to be means of describing how a domain could be instantiated. Thus, a model is an instantiation

or a specific case that follows an explicit meta-model or modeling language. The definitions of ontology from the literature are given in the next subsection, but our own definitions are given in sequence in order to advance and establish a comparison with the MDE terms. Ontologies are formalized in a language as well, and they can be created from scratch or include and extend some others as starting point. An ontology can be seen as a conceptualization, usually referred as terminological components (TBox). Then, an ontology can be populated/instantiated with facts or assertions, usually called assertion component (ABox), which is associated with a terminological vocabulary. Shortly, TBox statements describe a system in terms of controlled vocabularies, for example, a set of classes and properties; and ABox are TBox-compliant statements about that vocabulary. These details will become clear as we advance in the topic of ontology which brings background definitions from other authors that complement our viewpoint.

2.3. Ontologies and multi-agent systems

Ontology is defined as an *explicit specification of a conceptualization* (Gruber, 1993), where a *conceptualization* is an abstract, simplified view of the world that we wish to represent for some purpose. Some essential properties of ontologies are: *(i)* ontologies describe a specific domain;[2] *(ii)* ontology users agree to use the terms consistently; *(iii)* ontology concepts and relations are unambiguously defined in a formal language by axioms and definitions; *(iv)* relationships between ontology concepts determine the ontology structure, e.g., hierarchical or non-hierarchical (Hadzic et al., 2009). The generalization/specialization relationship (i.e., "is-a" relationship) between two concepts is an example of a hierarchical relationship between concepts; *(v)* ontologies can be understood by computers.

Ontology has different meanings within different contexts, e.g., in Philosophy and Metaphysics, ontology encompasses nature and existence, beings and relation between beings (Hadzic et al., 2009). The resulting knowledge is intended to be explicit, shared and understood by humans. In Computer Science and Artificial Intelligence, ontology is used to formally and explicitly represent shared domain knowledge through definitions and axioms of concepts and relationships between concepts (Hadzic et al., 2009). This work refers to ontology in the context of Computer Science and Artificial Intelligence, in which one of the main differences is that ontologies are designed to be machine-understandable.

Ontologies are knowledge representation structures, usually based on Description Logics (Baader et al., 2004), composed of concepts, properties, individuals, relationships and axioms. A concept (or class) is a collection of objects that share specific restrictions, similarities or common properties. A property expresses relationships between concepts. An individual (instance, object, or fact) is an element of a concept. A relationship instantiates a property to relate two individuals. Finally, an axiom (or rule) imposes constraints on values of concepts or individuals normally using logic languages (that can be used to check ontological consistency or infer new knowledge).

Ontology empowers the execution of semantic reasoners that provide functionalities such as *consistency checking*, *concept satisfiability*, *classification*, and *realization*. In other words, reasoners are able to automatically infer logical consequences from a set of axioms. Pellet (Sirin et al., 2007) is one example of semantic reasoner implementation over OWL ontologies. OWL (Web Ontology Language) is a language for processing web information and semantic web standard formalism to explicitly represent the meaning and relationships of terms (Bechhofer et al., 2004).

[2]This definition we cited claims that an ontology describes a specific domain, which means a part of knowledge in some degree. It does not claim that all ontologies are domain specific, because there are the so called upper level (or foundational) ontologies.

Ontologies can offer significant benefits for MAS (Tran and Low, 2008): interoperability, reusability, support for MAS development activities (such as system analysis and agent knowledge modeling) and new features for MAS operation (such as agent communication and reasoning). Also, ontologies may be used at different stages within MAS (Hadzic et al., 2009) so as to: enable decomposition of the overall problem; support the process of information retrieval and reuse; support the process of analysing and manipulating information; and enable communication between cooperatively working agents. We consider that these essential properties have a role to play in the process of modeling and development of complex systems, and models based on these underlying properties may be explored in many different ways. More specifically, this research is interested in investigating the use of ontology for designing MAS. We have investigated an ontology-based MDE approach as an integrated global model of MAS's main characteristics, and explored ways of using such model to support MAS programming. Although the advantages of ontologies for agents are clear (Tran and Low, 2008), few MAS platforms currently integrate ontology techniques. Limited ontological support is provided by the AOSE methodologies since they do not incorporate ontologies throughout the entire systems development life cycle nor consider ways in which ontologies can be used to account for interoperability and verification during design (Tran and Low, 2008).

The interest in using ontology with agent systems is not recent. For example, the InfoSleuth project in the late 90's used ontologies to model agents and included explicit ontology agents (Bayardo et al., 1997). Ontologies in InfoSleuth were used to specify both the infrastructure underlying the agent-based architecture and to characterize the information content in the underlying data repositories. According to its authors, their motivations for using ontologies were two-fold: *(i)* capturing and reasoning about information content (e.g., database schema, conceptual models); and *(ii)* specification of the agent infrastructure (e.g., agent configurations, and workflow specifications). The ontology agents managed the semantics of the domain/environment in which the agents operate, and the ontology models for agents described the agents' knowledge and attributes. One of our main differences is that the ontology models in that paper had been used for interoperability, but not to support implementation like the MDE models in our work. Also, that work used technologies from that time, for example, their agents were implemented using Java, which is not considered an agent-oriented programming language.

In this Section we explained the topics of MAS, MDE, and ontology. We already started to link areas among each other, and their interconnections are explored with more detail in the next Section.

3. Literature review

This Section shows the connections among MDE, MAS and ontologies from three viewpoints. First, we discuss the importance and advantages of models for MAS. Second, relations of MDE with ontologies are given. Third, combinations of ontologies with MAS are shown with emphasis on approaches that use ontology for modeling MAS.

3.1. Model-driven engineering and multi-agent systems

Several models and methodologies can be found in literature to formalize and define the processes of MAS design and implementation. For example, Prometheus (Padgham and Winikoff, 2002) is one of the most well known MAS modeling methodology for developing intelligent agent systems. It defines a development process with associated deliverables proven to be effective in assisting developers to design, document, and build agent systems based on concepts such as goals, beliefs, plans, and events.

Prometheus (Padgham and Winikoff, 2002) contains three phases: *system specification*, *architectural design* and *detailed design*. It starts with the *system specification* phase, which focuses on identifying the basic system functionalities, along with inputs (percepts), outputs (actions) and any important shared data sources. Then, the *architectural design* phase uses the outputs from the previous step to determine which agents the system will contain and how they will interact. Lastly, the *detailed design* phase looks at the internals of each agent and how it will accomplish its tasks within the overall system. Among future work for Prometheus (Padgham and Winikoff, 2002) there is the introduction of social concepts to improve its current models. Therefore, some aspects of MAS are not covered by models in Prometheus, which also does not explore the use of formal ontology as part of its approach.

Prometheus is usually used as reference for combining MDE with MAS. For example, the Prometheus AEOlus (Uez and Hübner, 2014) allows the integrated development of the three MAS dimensions (agent, environment and organization). It contributes with: *(i)* a new meta-model that combines the meta-models of Prometheus and JaCaMo; *(ii)* a new interactive incremental process based on the Prometheus process; and *(iii)* a code generation approach for JaCaMo based on this new meta-model. Prometheus AEOlus (Uez and Hübner, 2014) improves modeling, code generation and reduces the conceptual gap between the analysis and implementation phases. It extends Prometheus by including concepts to consider the environment and organization dimension of JaCaMo, and it applies concepts from JaCaMo to improve Prometheus development process to ensure that concepts used during the design and analysis stages will be used in the implementation stage. However, the meta-model is not integrated with semantic technologies, reasoners, ontologies and it is not used during MAS programming. The code generation in Prometheus AEOlus (Uez and Hübner, 2014) requires the refinement of entities in the model to generate code (for JaCaMo components, i.e., Jason, Moise and CArtAgO). Thus, the models must be refined to include platform-specific information, and once the first version of the MAS code is generated, the models are no longer used during the programming step to complete the MAS development.

Research in the direction of building tools for developing MAS through exploiting MDE techniques have led to a new proposal (Gascueña et al., 2012) of using Ecore with Prometheus. Ecore is used by the Eclipse Meta-modeling Framework to define meta-models, and it is applied to develop the meta-model concepts specific to Prometheus. More specifically, this work addressed the generation of MAS graphical editors based on the models and how agent code generators can be developed from such visual models. In the end, MAS programming code can be generated from the models, ranging from code skeletons to completely deployable products (Gascueña et al., 2012). To demonstrate this claim, templates have been created to automatically generate code in JACK language, which uses the BDI model to represent the internal structure of its agents. Once the model is converted to code, the developer must continue the MAS programming phase without using the model. Similarly as we see in other related work, this approach does not explore ontology as part of it and the MDE proposal is not used during MAS coding.

New aspects of MAS for programming platforms are also created and proposed as models by Zatelli and Hübner (2014). Their models specify the interaction as a first-class abstraction to define MAS with respect to agents, environment, interaction, and organization. The interaction allows the definition of the desired sequence of steps to achieve the organizational goals (while the organizational goals provide information about what the agents need to do, the interaction protocols provide a more detailed description about how to behave to achieve them). More specifically, this work presents a conceptual model for the interaction component, a programming language to specify the interaction, and how the proposed approach could be integrated in the JaCaMo MAS platform. Such contributions allow developers to model the interaction in a separate component. Thus, the interaction does not need to be hard-coded inside the code of agents or other components (Zatelli and Hübner, 2014).

MAS-ML is a Multi-Agent System Modeling Language (da Silva and de Lucena, 2003) that extends UML based on TAO (Taming Agents and Objects). The TAO meta-model defines the static and dynamic aspects of MAS. New diagrams – Organization and Role diagrams – have been created because of the set of different elements and relationships defined in the TAO meta-model that have been incorporated in the UML meta-model. Also, UML diagrams that already exist – Class and Sequence diagrams – have been adapted. Explanations about the diagrams can be found as follows (da Silva and de Lucena, 2003):

- The **Sequence diagram** represents the dynamic interaction between the elements that compose a MAS – i.e., between objects, agents, organizations and environments.
- The extended **Class diagram** also represents agents, organizations and the relationships between agents, organizations and classes as defined in TAO.
- The **Organization diagram** models the system organizations identifying their habitats, the roles that they define and the elements – objects, agents and sub-organizations – that play these roles.
- The **Role diagram** is responsible for clarifying the relationships between agent roles and object roles.

MAS-ML has as similarity with our proposal the fact of defining agent, organization and environment as first order abstractions. However, our approach proposes to include the use of ontology as meta-model for MAS. Also, MAS-ML claims that it would be easier to use a programming language that considers these elements as first order abstractions to implement MAS-ML models (da Silva and de Lucena, 2003). While MAS-ML states as contribution "the mapping of the design elements in the agent level of abstraction to a programming language" but without determining a specific programming language, our proposal focuses on JaCaMo.

Model-Driven Development with agent-based models can facilitate the implementation of methods and tools for the development of MAS (Pavón et al., 2006). This Section links MDE with MAS and it shows that models are commonly used to generate code automatically, but without making use of ontologies, or without offering any type of model-based support during the programming and verification steps. In most cases, agent-oriented methodologies focus on the agent concept at the analysis level or look for visual or formal representations of elements present in an already implemented agent framework. We believe that such models can play a role not only in model transformation approaches that generate a first or skeleton version of MAS code, but also being employed until the end of MAS development. However, moving from agent models to implementation is, currently, not fully addressed by most agent-oriented methodologies in a systematic way (Pavón et al., 2006), which leaves a gap between design and implementation. The approaches presented in this Section so far do not relate the models with ontologies in the areas of AOSE. In further Sections, we present the current research in this direction, but first next Section discusses relations of MDE with ontologies from a general viewpoint (without focusing on the specific context of MAS).

3.2. Model-driven engineering and ontologies

Usually, models are specified by instantiating meta-models, but none of the approaches in MDE we see in previous Section for MAS explores ontology for improving MAS modeling. The definitions we found (Atkinson et al., 2006) claim that "all ontologies are models, but not all models are ontologies", however "there is no widely accepted definition of what distinguishes models from ontologies". These two areas are, superficially, very similar, and in fact are sometimes visualized using the same language (e.g., UML). To better characterise their differences, we observe that models tend to use the close world

assumption and focus on realization issues, while ontologies usually rely on the open world assumption and focus on capturing abstract domain concepts and their relationships (Atkinson et al., 2006).

Also, ontologies and meta-models are usually designed with different goals in mind (Kappel et al., 2006). For example, meta-models prove to be more implementation-oriented as they often bear design decisions that allow producing sound, object-oriented implementations. Due to this, language concepts can be hidden in a meta-model, but they have to be made explicit in an ontology (Kappel et al., 2006). In fact, there are proposals to create ontologies from meta-models, such as the lifting procedure (Kappel et al., 2006), which was designed to achieve semantic integration in modeling languages.

Other research directions rely on exploring these areas in interconnected ways, which is, for example, the application of MDE with ontology technologies (Staab et al., 2010). Since OWL 2 has not been designed to act as a meta-model for defining modeling languages, Staab et al. (2010) show how to build such languages in an integrated manner by bridging pure language meta-models and OWL in order to benefit from both approaches. However, these areas differ in some other points (Atkinson et al., 2006) such as: *(i)* ontologies are generally used for run-time knowledge exploitation while models are not intended to contain instance data or be accessible at run-time; *(ii)* ontologies usually support "reasoning" while models cannot (or do not); and *(iii)* ontologies are expected to be represented with well-defined semantics in a language like OWL while models in a less precise language like UML. This reference does not claim that models do not contain instance data, but that models are not intended to contain it. Our goal in mentioning this is to expose several different authors' viewpoints in these areas since it is acknowledged that some definitions remain ambiguous and confusing.

Although the research areas of MDE and ontologies have been developed by two different communities, important synergies can be achieved by combining them. However, there are open research challenges for ontological approaches to model engineering, e.g., in which tasks ontologies and software models can be optimally used together and how ontologies should be integrated into MDE. Such investigations will lead to ways in which they can be made compatible and linked so as to benefit both communities.

This Section discussed relations of MDE with ontologies from a general viewpoint, i.e., without focusing on the specific context of MAS. Next Section shows approaches where ontologies are used in and for MAS. We highlight two different roles played by ontologies in MAS: situations where they are used without addressing modeling issues; and situations where ontologies are considered for AOSE.

3.3. Multi-agent systems and ontologies

One of the first approaches to consider ontologies to enhance agent programming is AgentSpeak-DL (Moreira et al., 2005). However, AgentSpeak-DL focuses on using ontology during agent reasoning, instead of modeling aspects of MAS in ontologies. AgentSpeak-DL extends agents' belief base with Description Logic, in which the belief base includes: (*i*) one immutable TBox (terminological box, or conceptualization) that characterises the domain concepts and properties; and (*ii*) one ABox (assertion box, or instantiation) with dynamic factual knowledge that changes according to the results of environment perception, plan execution and agent communication. AgentSpeak-DL approach enriches the agent belief base with the definition of complex concepts that can go beyond factual knowledge. The advantages of integrating agents and ontologies pointed out by Moreira et al. (2005) are: (*i*) more expressive queries in the belief base, since results can be inferred from the ontology and thus are not limited to explicit knowledge; (*ii*) refined belief update given that ontological consistency of a belief addition can be

checked; (*iii*) the search for a plan to deal with an event is more flexible because it is not limited to unification,[3] i.e., it is possible to consider subsumption relationships between concepts; and (*iv*) agents can share knowledge using ontology languages such as OWL. Although such advantages enable new reasoning mechanisms for MAS by means of ontologies, our work investigates a different research direction in which ontologies are used as part of AOSE methodologies to aid modeling and implementation of agent systems.

JASDL (Klapiscak and Bordini, 2008) also merges agent belief base and ontological reasoning since it implements AgentSpeak-DL to provide Jason agents with ontology manipulation capabilities using the OWL API. Agent programmers benefit from features such as plan trigger generalization based on ontological knowledge and the use of such knowledge in belief base querying (Klapiscak and Bordini, 2008). Some Jason modules were altered to implement JASDL such as: the belief base was extended to partly reside within an ontology ABox, which, combined with a DL reasoner, facilitates the reuse of available knowledge in ontologies (to increase inferences that an agent can make based on its beliefs and assure knowledge consistency); the plan library to enable enhanced plan searching; and the agent architecture to augment it with message processing to obtain semantically-enriched inter-agent communication.

CooL-AgentSpeak (Mascardi et al., 2014) is an extension of AgentSpeak-DL with plan exchange and ontology services. It implements a CArtAgO artifact functioning as an ontology repository tool which stores a possibly dynamic set of ontologies and offers related ontology matching and alignment features. It searches for ontologically relevant plans not only in the agent's local plan library, but in the other agents' libraries too, according to a cooperation strategy not based solely on unification and subsumption relations between concepts, but also on ontology matching. In short, CooL-AgentSpeak (Mascardi et al., 2014) performs cross ontological unification for agents that do not disclose their ontologies to each other (that cooperate while preserving their privacy).

So far, this Section presented approaches for incorporating ontological reasoning in agents. Although the advantages of using ontologies for agents are clear, few agent-oriented platforms are currently integrated with ontology techniques. Next we focus on examples of ontologies proposed for modeling MAS. Some aspects of MAS, such as the organizational properties found in Moise (Hübner et al., 2010), are already related to a programming framework, allowing to convert the ontology specification to a programming level (Zarafin, 2012), which provides more flexibility for modeling and developing agent organizations. Such OWL semantic description of MAS organizations also helps agents in becoming aware, querying, and reasoning about their social and organizational context in a uniform way (Zarafin, 2012). Another related work proposes an environment ontology (Okuyama et al., 2006) based on MAS environment aspects of agent programming technologies. This model can be used to specify environments and derive a project-level, complete, and executable definition of multi-agent environments. Semantic representations of MAS environments also improve the way agents reason about the objects with which they interact and the overall environment where they are situated (Okuyama et al., 2006). This is important because most agent-oriented programming languages are weak in allowing the developer to model the environment within which the agents will execute (Bordini et al., 2006). Such ontology models are desirable for all dimensions of MAS at the same time, but these levels have to be aligned so that they work as a common specification. This will make possible to model, reuse and extend a MAS in one dimension while maintaining the others, which enables the designer to work without going into specifics of the programming languages that define each dimension. In this context, a MAS would be

[3]The traditional plans' unification relies on pattern matching mechanisms based only on syntax and lexical approaches for comparing plans. Thus, semantics is not considered to infer that a plan could be attempted in a given situation.

more easily designed, expressed and communicated, and a specific modeled project might be verified and converted to code.

As far as we know, the use of ontology to support modeling, development and verification of MAS in the context of MDE is a new idea currently not fully explored in literature. We have found the use of ontologies to represent only partial aspects of MAS, such as an environment ontology (Okuyama et al., 2006) based on environment aspects of agent programming technologies. The use of an environment ontology adds three important features to existing multi-agent approaches (Okuyama et al., 2006): *(i)* ontologies provide a common vocabulary to enable environment specification by agent developers (since it explicitly represents the environment and agent essential properties, defining environments in ontologies facilitates and improves the development of multi-agent simulations); *(ii)* an environment ontology is useful for agents acting in the environment because it provides a common vocabulary for communication within and about the environment (it allows interoperability of heterogeneous systems); and *(iii)* environment ontologies can be defined in ontology editors with graphical user interfaces, making easier for those unfamiliar with programming to understand and design such ontologies. An environment description is a specification of its properties and behaviour, which includes concepts such as: *objects* (i.e., resources of the environment); *agents* (i.e., their "physical" representation in the environment that is visible to other agents); *actions* that each type of agent can perform in the environment; *reactions* of the environment and objects when an agent's actions affect them; *perception types* available to each type of agent; and *observable properties*, that is, the information about the simulation to which observers (e.g., the agents) have access.

In Okuyama et al. (2006), the relationship between the environment and other MAS dimensions was already foreseen, since they mention the intention of looking at higher-level aspects of environments, i.e., social environment aspects of agents, such as the specification of social norms and organizations in agent societies. In fact, on the MAS organization dimension, there is a semantic description of MAS organizations (Zarafin, 2012) using OWL to develop an ontology for organizational specifications of the Moise model (structural, functional, and normative levels). This approach helps agents in becoming aware, querying, and reasoning about their social and organizational context in a uniform way. Also, this work makes possible to convert between the ontology and the Moise specification, providing more flexibility for modeling and developing agent organizations. The semantic description of Moise (Zarafin, 2012) provides agent-side reasoning, querying features and benefits such as increased modularization, knowledge enriching with meta-data, reuse of specifications, and easier integration. With the semantic web effort aiming to represent the information in semantic formats, the MAS community can take advantage of these new technologies in MAS development tasks such as to integrate organizational models, to monitor organizations, and to analyze agent societies (Zarafin, 2012).

A comparison among related work and with the research in this paper is depicted in Table 1. It shows that there are already models and MDE approaches for more than one MAS dimension (Gascueña et al., 2012; Padgham and Winikoff, 2002; Uez and Hübner, 2014), but without using ontologies, semantic reasoning, or employing the model during the programming step. We also compare ontologies with MDE (Atkinson et al., 2006; Kappel et al., 2006; Staab et al., 2010), and we show that ontologies are used to extend MAS capabilities, but with other goals than modeling (Klapiscak and Bordini, 2008; Mascardi et al., 2014; Moreira et al., 2005). When an ontology is used for MAS modeling, only a part is modeled, such as the environment (Okuyama et al., 2006) or the organization (Zarafin, 2012). Next Section explains our work which consists of an ontology for MAS modeling, while the other following Section presents our tool for MAS development using the proposed ontology. Our research combines MDE with an ontology perspective for building MAS, and we integrate concepts from different

Table 1
Comparing related work in the areas of MDE, MAS and ontologies

Research	Overview of the work	Inclusion of ontologies	Dimensions of MAS modeled	MAS platforms used
		Model-Driven Engineering and Multi-Agent Systems		
Prometheus (Padgham and Winikoff, 2002)	Model and methodology for MAS development	No	Agent, environment and organization	JACK
Prometheus AEOlus (Uez and Hübner, 2014)	Approach for MAS modeling and programming	No	Agent, environment and organization	JaCaMo
MDE for MAS development (Gascueña et al., 2012)	Ecore meta-model of Prometheus for MAS development in Eclipse	No	Agent, environment and organization	JACK
MAS interaction component (Zatelli and Hübner, 2014)	A conceptual model and programming language for MAS interaction aspects	No	Interaction	JaCaMo
MAS-ML (da Silva and de Lucena, 2003)	MAS modeling language that extends UML based on TAO	No	Agent, environment and organization	Unspecified
		Ontologies and Multi-Agent Systems		
AgentSpeak-DL (Moreira et al., 2005)	An approach for using ontologies during agent reasoning	Agents get means to represent knowledge and interact with ontologies	Ontologies extend agents' belief base with DL	AgentSpeak
JASDL (Klapiscak and Bordini, 2008)	An implementation of AgentSpeak-DL in Jason	Jason agents can represent knowledge and interact with ontologies	Ontologies extend agents' belief base with DL	Jason
CooL-AgentSpeak (Mascardi et al., 2014)	AgentSpeak-DL's extension with plan exchange and ontology services	Each agent has its private ontologies	Ontologies extend agents' belief base with DL	Jason
		Ontologies for Modeling Multi-Agent Systems		
MAS env. ontology (Okuyama et al., 2006)	Ontology to specify and derive definitions of MAS environments	A new ontology for modeling MAS environments	Environment	Unspecified
MAS org. ontology (Zarafin, 2012)	A semantic description of Moise organizations in OWL	A new ontology for modeling Moise	Organization	Moise
This work	An ontology and a tool for model-driven development of MAS	A new ontology for AOSE (modeling, development and verification of MAS)	Agent, environment and organization	JaCaMo

agent dimensions in a single framework, so agent-oriented software engineers benefit from receiving new methodologies and tools to develop their systems as result of this comprehensive approach. In this context, our research pioneers in covering all these issues at the same time, since it consists of an ontology for MAS modeling and a model-based tool for MAS development and verification. Advantages derived from such approach are techniques for: *(i)* integrating design and code; *(ii)* supporting MAS programming with automatic code generation through model-based development; and *(iii)* performing verification with focus on the use of semantic reasoning and model checking.

4. An ontology modeling approach for engineering multi-agent systems

We claim that ontologies have an important role for all MAS dimensions and in the whole system development, rather than exclusively in the programming phase. This is in line with CArtAgO development directions (Ricci et al., 2006) in considering ontologies to represent the artifacts, with Moise's recent research which proposes a semantic description of multi-agent organizations (Zarafin, 2012), and with ontologies of MAS environments (Okuyama et al., 2006). These are however all separate initiatives, whereas in the development of MAS an ontology should interconnect the various specification levels. This allows for a unified view of systems engineering, and should co-exist with integrated agent platforms, such as JaCaMo (Boissier et al., 2013). As result, developers obtain a new paradigm for developing complex software systems with a semantic infrastructure applying the software and knowledge engineering principles. Unified MAS platforms, such as JaCaMo (Boissier et al., 2013), are being developed with the purpose of helping developers to build these complex projects, however, such unification must happen during the system design and at the modeling and knowledge level. Thus, we investigate the integration of agent programming platforms by applying ontology to streamline MAS development in JaCaMo.

We focus on a model-based approach using ontology to cover all required dimensions and abstractions of MAS. This differs from other approaches where: *(i)* ontologies are not used; *(ii)* only a part of such systems is modeled in ontologies; or *(iii)* the model is not integrated with coding or verification mechanisms. In our work, an ontology is used for modeling MAS and a model-based tool supports the steps of programming and verification. Thus, our motivation resides in improving MAS modeling and development by exploring ontology and MDE approaches. We performed several steps to achieve such contributions. We first investigated ontologies and model-based development approaches in MAS to address questions such as MAS modeling, knowledge representation and reasoning. Our results in this direction are reported in the previous Section, which contains a literature review in these areas, and a comparative analysis of existing solutions to identify their limitations. Then, we elaborated an ontology-based approach in the domain of MAS modeling to support developers of such systems. In this context, this Section presents our MAS ontology model; and the next Section explains the current developments of our MDE ontology-based tool. These Sections answer questions such as: *(i)* How to specify/model a MAS as ontology? *(ii)* How ontology-based elements can support MAS codification? and *(iii)* Which tools/benefits/reasoning may derive from such representation? We also point out: *(i)* the addition of ontology reasoning in our model of MAS; *(ii)* improvements to our MAS ontology-based tool; and *(iii)* current and future evaluations of these approaches in MAS modeling, development and verification.

In our work we have designed an ontology to define the main abstractions of MAS, namely the concepts from agents, environments and organizations. The underlying idea in our research is that the conception of any MAS project should start by modeling it in this ontology. This can be done by extending top-level concepts, and adding new instances and properties in order to specify the corresponding desired project to be implemented in terms of agent-oriented concepts. In such approach the MAS is first modeled based on our upper ontology of agent systems, which uses a single formalism to encompass the global characteristics of MAS. In these terms, our ontology can be seen as a language, a meta-model, a high-level conceptualization, or as a domain-independent model of every possible agent systems in which agent developers would use it to model/extend/instantiate their specific agent system project. The result from using the proposed ontology as a meta-model to define a specific MAS project is an instantiated ontology that corresponds, for example, to a project in JaCaMo.

Our investigation is also related with the exploration of approaches and tools that can apply such instantiated ontology model during the MAS coding step to offer support for programming and the use of ontology reasoning in our models to perform verifications in a specified project of MAS. We demonstrate in this paper how an instantiated model of our ontology can be applied to support the generation and development of MAS code. For example, elements in the MAS ontology model can be transformed into MAS code, such as by dragging content from model to code. Such modeling approach for MAS also enables the execution of inferences and verifications over the MAS instantiated ontology model, and reverse engineer can be applied for making transformations from MAS code to models.

Current frameworks for MAS development integrate different aspects concerning the agents, their environment and the organization in which they act. Following this, our MAS ontology model contains concepts to represent, in a single formalism the three MAS dimensions usually required (as, for instance, in JaCaMo are the agent dimension in Jason, the environment layer in CArtAgO, and the organization elements in Moise). A particular MAS begins to be modeled by *extending* the proposed ontology, which is done by creating new subclasses to its top-level concepts. Then, individuals are created in the process of *instantiating* the extended ontology. Also, the corresponding additional properties and relations among concepts and instances should be specified in this moment. Our ontology-based approach allows to model, reuse and extend a MAS in one dimension while maintaining the others, and designers can create models without going into specifics of the programming languages that define each dimension. In this context, a MAS design is better expressed and communicated, and the model can be more easily converted to code or a formal verification system.

The concepts from our ontology model were defined by analysing and combining the meta-models of Prometheus (Padgham and Winikoff, 2002) and JaCaMo (Boissier et al., 2013). These MAS frameworks were also used in a related work (Uez and Hübner, 2014), but without considering an ontology and without presenting a model-based development tool. Figure 1 shows the main concepts and properties in our ontology model, which are grouped according to the three MAS dimensions previously discussed (agent, environment and organization). The ontology represents different MAS concerns while allowing to relate them, therefore offering advantages such as increased maintainability, usability and extensibility for MAS modelers and developers. This approach also allows to gradually refine from high-level abstract views to elements directly available in concrete technical MAS programming platforms.

From the agent dimension, we are not interested in defining any possible and generic characteristics of any kind of agent, such as physical agents. Instead, we are interested in specifying only the concepts of virtual agents that make sense in the context of programming for this dimension, such as the characteristics of Jason implemented agents. The most important concepts from the agent dimension are the *Agents*

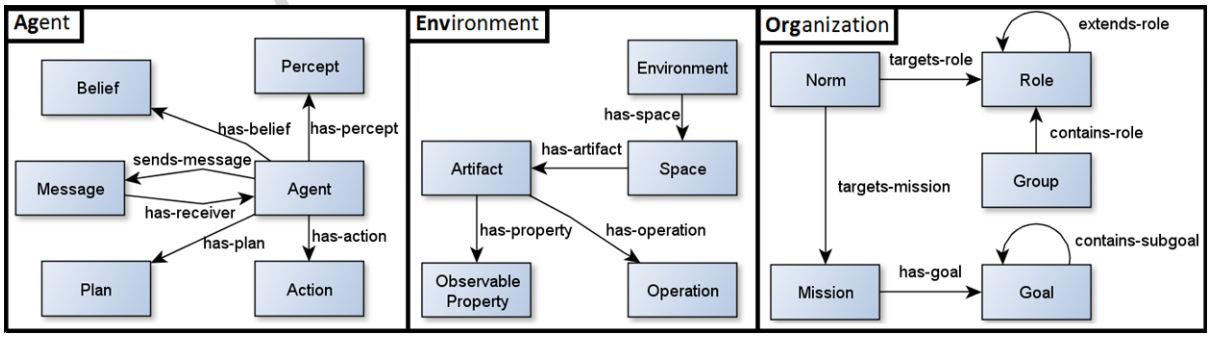

Fig. 1. Main concepts and properties in the MAS ontology model.

along with their *Actions*, *Percepts* and *Messages*. *Agent* is domain of properties such as *has-action*, *has-percept* and *sends-message*. The main concepts and properties regarding the agent dimension are depicted on the left side of Fig. 1.

From the environment dimension, we are also not interested in defining any possible and generic characteristics of any kind of environment, such as physical environments. Instead, we are interested in specifying only the concepts of virtual environments that make sense in the context of programming for this dimension, such as the characteristics of CArtAgO implemented environments. From the environment perspective, the main concepts are the *Artifacts*, *Spaces*, *Operations* and *Observable Properties*. *Artifacts* can be either the target (outcome) of agent activities, or the tools used by agents as means to support their activities (consequently, artifacts reduce the complexity of agents tasks' execution). This dimension contains properties such as *has-artifact*, *has-operation* and *has-observable-property*. The main classes and properties regarding the environment dimension are depicted in the center of Fig. 1.

From the organization dimension, similar to how the other dimensions were considered, we are not interested in defining any possible and generic characteristics of any kind of organization. Instead, the ontology is interested in specifying only the concepts of organizations that make sense in the context of programming for this dimension, such as the characteristics of Moise implemented organizations. Thus, the organization dimension specifies concepts such as *Role*, *Goal*, *Group*, *Norm* and *Mission*. A *Role* definition states that agents playing that role are willing to accept the behavioural constraints related to it. The organization functional dimension specifies how global collective *Goals* should be achieved, i.e., how they are decomposed in global plans, grouped in coherent sets (missions) to be individually distributed to agents. The normative dimension binds the structural level with the functional one to specify role's permissions, prohibitions and obligations for missions. Thus, exemplar properties in the organization dimension are that *Group contains-role Role*, *Mission has-goal Goal* and *Norm targets-role Role*. The main classes and properties regarding MAS organizations are depicted on the right side of Fig. 1. In the literature it is possible to find considerable research on ontology for representing generic organizational characteristics, however, it is inadequate to this ontology needs and goals, since it is interested in one special type of organization: how agents can be organized inside the structure offered by Moise. Thus, the definition of some concepts such as role, group, and so on, may differ between those works and ours, mainly because of such implementation-oriented approach for this specific viewpoint in the domain of MAS organizations.

The organization dimension can address ontological support for interactions among the agents, as first-class objects, by means of protocols and commitments. For some MAS, the structural characteristics of organizations are more important to be highlighted and demand more details to be modeled than functional or behavioral properties. In these cases, methodologies to support interactions explicitly are desirable. Existing ontologies for commitments in MAS provide a conceptual point of view for comprehending advanced organizational details in the form of interactions and commitments. One of these ontologies (Singh, 1999) defines a *commitment* as involving a proposition with three participating agents: a debtor, a creditor, and a context group. Also, different kinds of commitments are defined, such as obligation, taboo, convention, and pledge. It is important to be aware of these high-level conceptual viewpoints when aiming to integrate such ideas from theory to practice both at the modeling and at the programming levels.

Currently Moise implements a limited view of the many advanced theoretical ideas and possibilities regarding MAS organizations, and so does our proposed ontology since they are both aligned. However, future work could expand and enrich both the modeling and the programming capabilities of these techniques in order to better represent these rich and complex details regarding social aspects in the

development of agent systems. At the programming level, in JaCaMo, an approach to code commitment patterns could be proposed, as well as new native constructs directly at the development platform. At the modeling level, in our ontology, new concepts to represent commitments, debtors, creditors and contexts of commitments could be created. Currently, this version of the ontology addresses models of MAS that are aligned with JaCaMo. These models contain, for example, the initial and static representations of organizations for Moise which are applied to generate code for agent platforms. It is currently future work to extend the modeling approach to address runtime characteristics of MAS, such as to represent and monitor organizational properties for when the modeled system is in execution. This includes, for example, Moise's notion of an agent committing to achieve certain goals, a simpler notion than commitments as described above.

Our MAS modeling methodology consists in creating subclasses, instances and relationships based on the concepts and properties provided by the ontology, which can be done with any ontology editor. For example, suppose that someone wants to develop a MAS to simulate a soccer scenario. The concept of *Agent* would have subclasses to define its types, such as *Player* and *Coach*. Instances of *Player* would represent concrete individuals of this type, such as *player1*, *player2*, and so on. Specific types of *Actions* for this MAS may be modeled as subclasses with the names *Move*, *PassBall*, and *ChangeStrategy*. Section 4.1 provides a detailed example of how a MAS can be modeled in our ontology; it shows also how inferences can be obtained from it.

Our ontology-based model integrates the dimensions of MAS at the semantic level, since they are already being integrated in the programming level, for example, in JaCaMo (Boissier et al., 2013). Agent programmers benefit from an integration among these ontological levels with each programming dimension since the knowledge represented in one dimension can be reused in another, thus resulting in a greater interoperability of agent platforms. This enables to convert MAS defined in ontologies to code in specific agent platforms, and vice-versa. Also, a system designed with a higher degree of modularity is easier to maintain, given that it separates different concerns yet enables relations between them. For example, the characteristics of one dimension (e.g., environment) can be used to define properties on another (e.g., organizational). In fact, it is often the case that the concepts of one level are related to another but current MAS platforms do not allow for such relations to be explicitly represented. The classes and properties in our ontology for MAS are modeled in three sub-ontologies, one for each dimension: agent, environment and social organization. The connections among concepts in the ontology are encoded by means of the object properties, which determine how instances are allowed to relate among each other. Our tool uses instantiations of this ontology to support MAS programming in JaCaMo, as we show in Section 5.

Model verification refers to the processes and techniques that the model developer uses to assure that his or her model is correct and matches any agreed-upon specifications and assumptions (Carson, 2002). Moreover, the ontology in our proposal can be explored with its available reasoning mechanisms to implement model verification in the context of MAS. The literature reports that most practical approaches for verification of MAS are done on code, and most of the work done on model checking within the MAS research area is quite theoretical (Bordini et al., 2006). However, there are approaches that use existing model checkers, typically to check properties of particular aspects of a MAS. While this has the advantage of proving properties of the system that will be actually deployed, it is also often useful to check properties during the system design. In fact, all the work on model checking for MAS is claimed to be still in early stages so not really suitable for use on large and realistic systems (Bordini et al., 2006).

Considering this context, semantic reasoning may provide, for example, consistency checking and inferences about the MAS specified in the ontology. The possibility to reason about the model fosters

the implementation of model checking features in the context of MAS. For example, when considering only MAS organizations, it is possible to verify conflicts considering the existing norms, roles and missions. When an instantiation of MAS organization is combined with instantiated agents, it is possible to verify other kind of inconsistencies integrating information from more than one dimension, such as if the agents contain the required capabilities to achieve the existing organization goals. Similarly, other verifications are possible when focusing on the interactions of instantiations of agents and environments, i.e., verifying if agents' actions are valid in such environment.

As previously explained, when integrating and matching information from more than one dimension, it is possible to perform consistency checking through inferences obtained by reasoning with the ontology. One example is to identify if the actions from agent's dimension are available as operations provided by the environment dimension. If there is an action from an agent that does not exist in the environment, the invocation of such action in run time will result in an action failure. Thus, the verification of such instantiated model characteristics at design time may prevent future errors to happen at the execution time of the corresponding JaCaMo specified project. Similarly, it is possible to verify if the agents have the capability to achieve the goals specified in the organization. Organization goals are assigned to agents playing the organization roles, and an agent playing a specific role may not have plans to achieve goals that the organization may assign to it. Reasoning can be applied also to verify consistency among the norms in the organization. The combination of some norms can result in contradictions, for example, when a prohibition occurs simultaneously with an obligation or permission. These contradictions can appear when considering the missions of just one isolated role, or when combining the missions of two or more roles.

4.1. Example of MAS modeled as an ontology instantiation

Consider a soccer game scenario to be modeled in terms of MAS in our proposed ontology. In this scenario there are two different types of agents: players and coaches. Player agents can perform actions such as moving in the soccer field, or passing the ball. The coach can send messages to player agents in order to inform which roles they should adopt in each moment of the match. The environment is the soccer field, where all agents are situated. Agents in the soccer field environment can perceive things such as the ball position, and the match score. A team is an organization, composed of one coach and several players. The players can play different roles, such as defender, striker, and captain (or leader). For these roles, different missions may be assigned, such as defending, or attacking. This brief specification addresses concepts of each one of the three MAS dimensions. Next we illustrate how to formalize it by extending and instantiating our ontology.

Figures 2, 3, and 4 illustrate, respectively, examples from the agent, environment, and organization dimensions. These examples show how to make subclass extensions and instantiations considering the soccer scenario specification. In Fig. 2, we can see two subclasses of *Agent* created for this MAS: *Player* and *Coach*. We included two instances of *Players* to reference individual agents (named *player1* and *player2*). Each instance of an Agent's subclass represents a concrete agent in the system, whereas its type is specified by its class. Moreover, two types of *Actions* are defined as subclasses of the *Action* concept: *Move*, and *PassBall*.

The environmental characteristics of our MAS are depicted by the subclasses and instances illustrated in Fig. 3. For example, the subclass *SoccerField* represents a type of *Space* in which our agents are situated. A concrete individual of this type is specified by the instance *soccerField1*. One type of *Artifact* (i.e., resource) that exists in the environment for agents to interact with is defined by the subclass *Ball*,

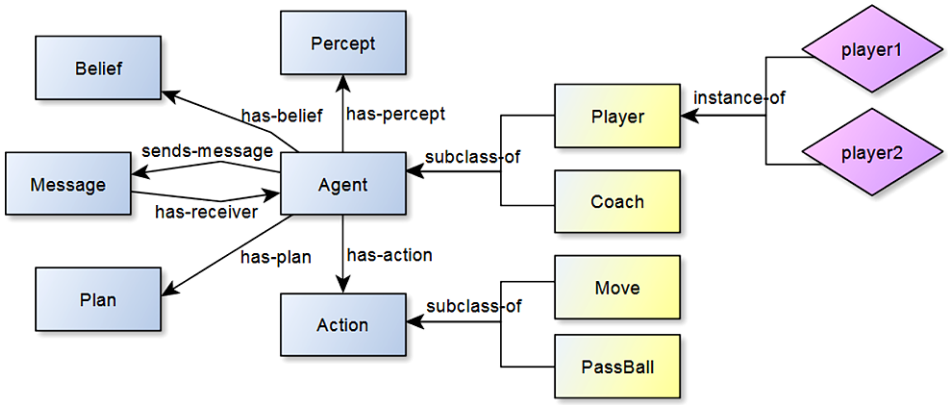

Fig. 2. Soccer scenario example of subclass extension and instantiation in the agent dimension of the ontology for MAS.

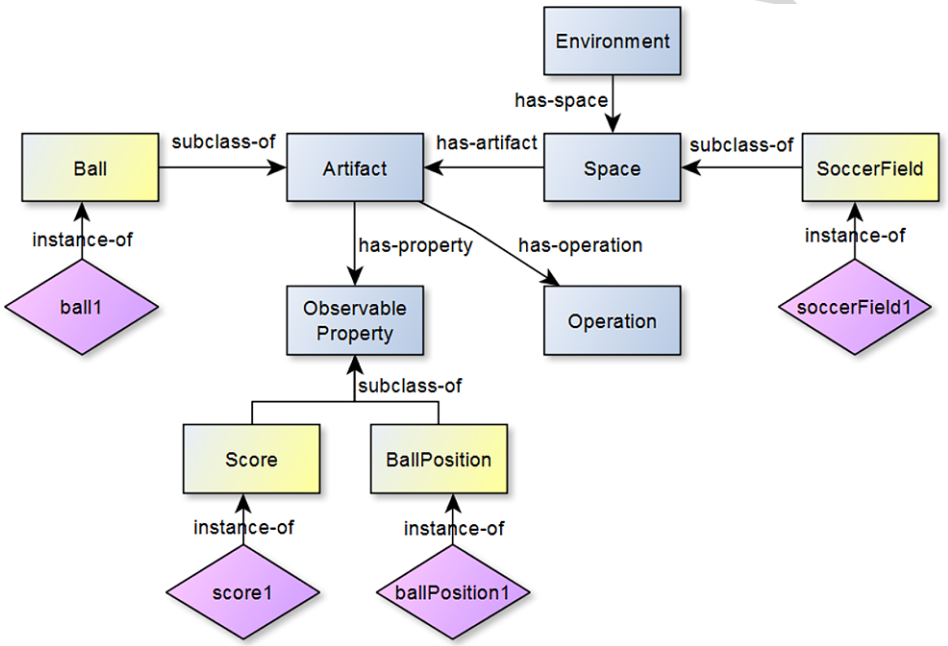

Fig. 3. Soccer scenario example of subclass extension and instantiation in the environment dimension of the ontology for MAS.

whereas *ball1* is assigned as a valid instance of it. The subclasses *Score* and *BallPosition* illustrate types of Observable Properties that resources may provide to agents, whereas an instantiation with values for these properties (for beginning the MAS simulation) is represented by *score1* and *ballPosition1*.

Figure 4 shows subclasses and instances to represent organization characteristics of this MAS. Two subclasses specify types of *Missions*: *Defending* and *Attacking*. Instances of these subclasses, such as *attacking1* and *defending1*, define an assignment of that Mission type to an agent. The types of *Roles* are given by the following three subclasses: *Defender*, *Striker*, and *Captain*. Instances of these subclasses of *Role* (e.g., *defender1*, *defender2*, *striker1*, *captain1*) define which agents are playing the corresponding roles. This example shows how to encode part of one possible strategy for modeling organizational

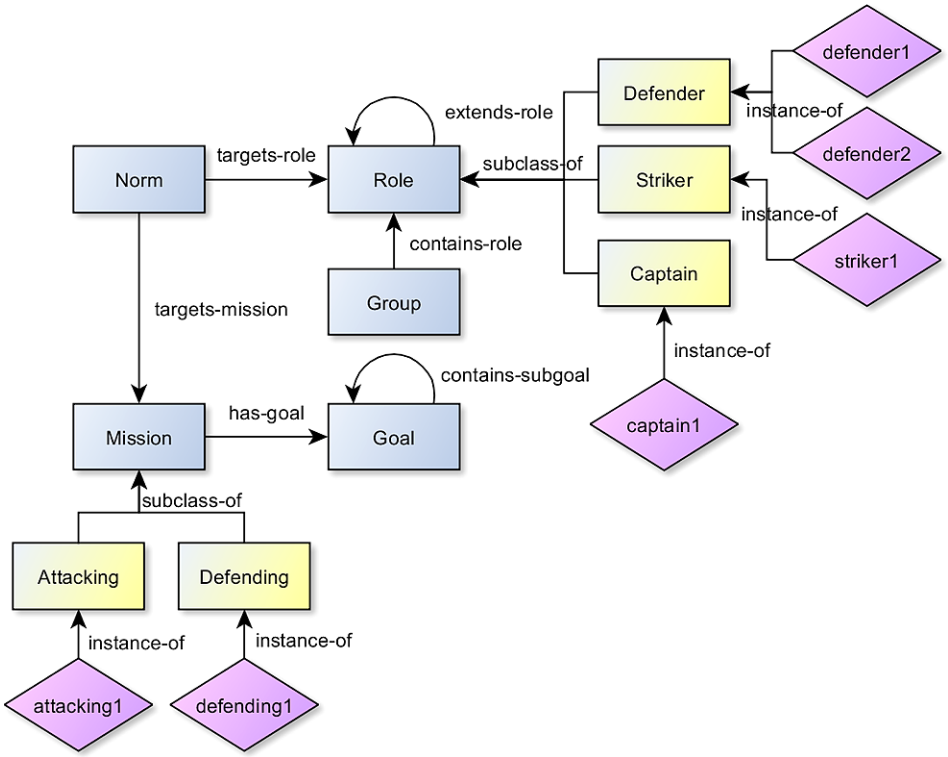

Fig. 4. Soccer scenario example of subclass extension and instantiation in the organization dimension of the ontology for MAS.

characteristics of agent systems. However, other strategies are possible and would result in different designs and implementations for this scenario. The modeling requires also the creation of relationships among the individuals, and the inclusion of some other axioms, which our example illustrates by using object properties and restrictions over the subclasses.

Figure 5 illustrates in more detail how such example can be specified using the visualization provided by the Protégé ontology editor.[4] The left side of this image illustrates the subclasses hierarchy, with emphasis on describing the *Player* concept. This part states the actions that players may execute, an environment where players are situated, and instances of this subclass. The right side of Fig. 5 presents information inferred by reasoners on this ontological instantiation. As we already explained, the desired MAS is specified and modeled on top of the elements provided by the proposed ontology. This is done by, for example, adding new classes, refining concepts, creating instances, asserting properties, and so on. After using any ontology editor for modeling the MAS, the resulting OWL file can be loaded into our tool that supports agent-oriented programming using a given ontology-based specification.

The classes *Coach* and *Player* represent roles that individual agents can play. Thus, as shown in Fig. 5, they were created as subclasses of *Ag_Agent*. This figure also illustrates some characteristics for the *Player* concept such as players being able to execute the actions of passing the ball and moving (this part is defined as a class restriction on the concept *Player*), and the role player has instances *player1* and *player2*. The environment where the agents are situated is specified using a relation that connects

[4] Available open source at http://protege.stanford.edu/.

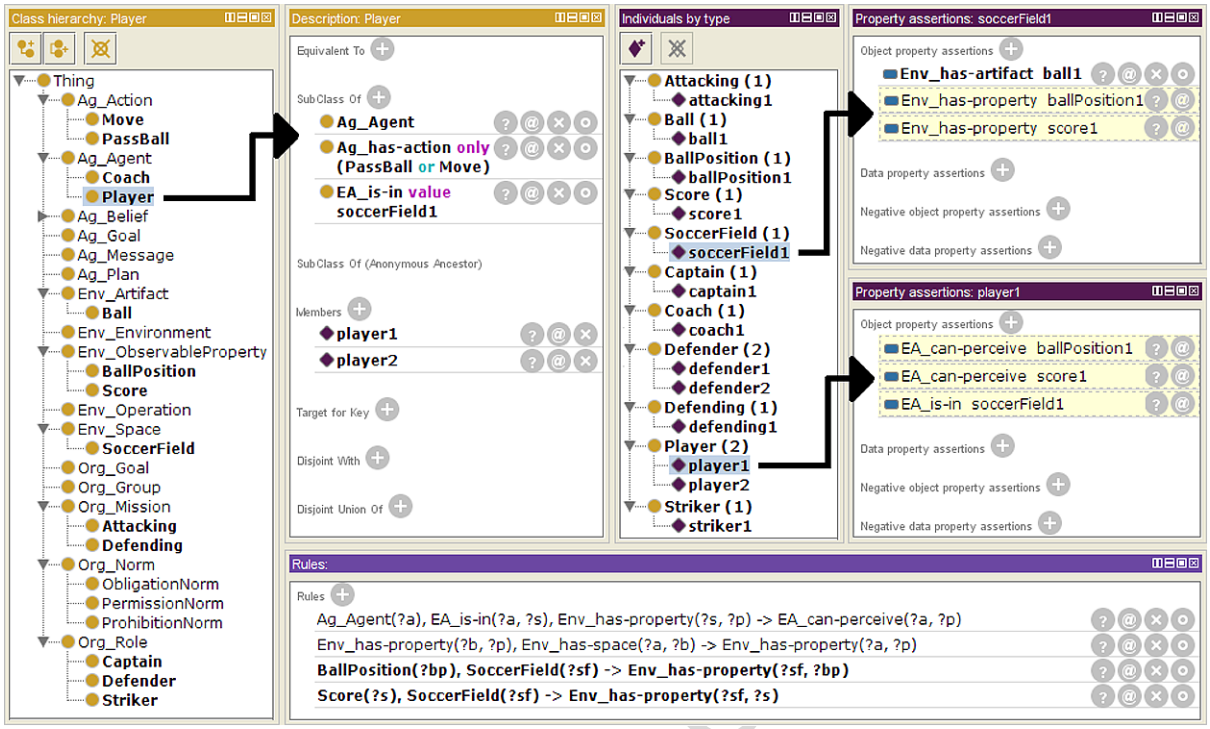

Fig. 5. Subclasses of agent with some conditions (e.g., "SubClass of" restrictions for Player), instances, and rules in the ontology together with properties that are asserted (in bold) and inferred (in dashed rectangles) by the semantic reasoning.

the *Environment* and *Agent* concepts: the property *EA_is-in* from *Ag_Agent* to *Env_Space*. Figure 5 also shows that *Rules*[5] can be inherited from the base ontology, and new *Rules* can be added particularly to a specific instantiation when defining a scenario (these are in bold). One general rule is that if an agent A is in a space S, and this space S has an observable property P, then it can be inferred that the agent A is able to perceive P if it chooses to do so. Rules created specifically for this scenario state that observations of ball position and score take place in spaces defined as soccer fields. As can be noted, it is possible to relate elements from any dimension (e.g., agent) with elements from another (e.g., environment).

The right-hand side of Fig. 5 shows details about individuals and property assertions regarding the modeled MAS soccer scenario. It illustrates the instantiation of concepts and properties that are asserted or inferred for some individuals. The inferences obtained by the execution of semantic reasoning over this example are shown in this figure inside dashed rectangles. It is asserted that the *soccerField1* instance has an artifact called *ball1*, and the previously defined rules allow the inference that this space contains the observable properties of *ball position* and *score*. Since *player1* is explicitly defined as an individual of *Player*, reasoners can use class restrictions of Players to imply its location (*soccerField1*). Also, rules support the inference of which observable properties this individual agent can perceive because of its location.

[5]The rules use an extension for OWL called SWRL (Semantic Web Rule Language).

5. An ontology-based multi-agent system development tool

This Section shows how ontology instantiated models such as the one just presented can be applied to generate MAS code for JaCaMo. We implemented a software tool that supports MAS programming by extending the features available in Eclipse. Eclipse (Budinsky, 2004) is an open source software development project that provides an Integrated Development Environment (IDE) in which a basic unit of function, or a component, is called a plug-in. In this context, we propose a plug-in for Eclipse that loads an instantiated ontology of MAS to provide code generation for JaCaMo.

The installation just requires the inclusion of the .jar file that corresponds to the plug-in in the directory named "plugins" in the folder where Eclipse is located. The plug-in can be activated to appear visually in the graphical interface of Eclipse by following these sequence: *Window* → *Show View* → *Other...* → *JaCaMo Ontology* → *Ok*. Figure 6 shows how to follow these steps in order to activate the plug-in. When it is enabled, the plug-in requests to be informed about the location of a file corresponding to an instantiated ontology so it can be loaded.

The plug-in was designed to be used in the "JaCaMo Perspective" of Eclipse (or related perspectives, such as Jason). The tool loads OWL ontologies and provides two model-based programming features to generate MAS code: drag-and-drop and auto-complete from instantiated ontologies. It was developed using the OWL API (Horridge and Bechhofer, 2011), which is an open source Java API (Application Programming Interface) for creating, manipulating and serializing ontologies in the OWL format.

The drag-and-drop functionality from ontology to agent code can be seen in Fig. 7, which depicts the Eclipse in Jason perspective. In the right side of the image, the developer can navigate in the ontology concepts, instances and properties (from the new Eclipse component developed in this work). These

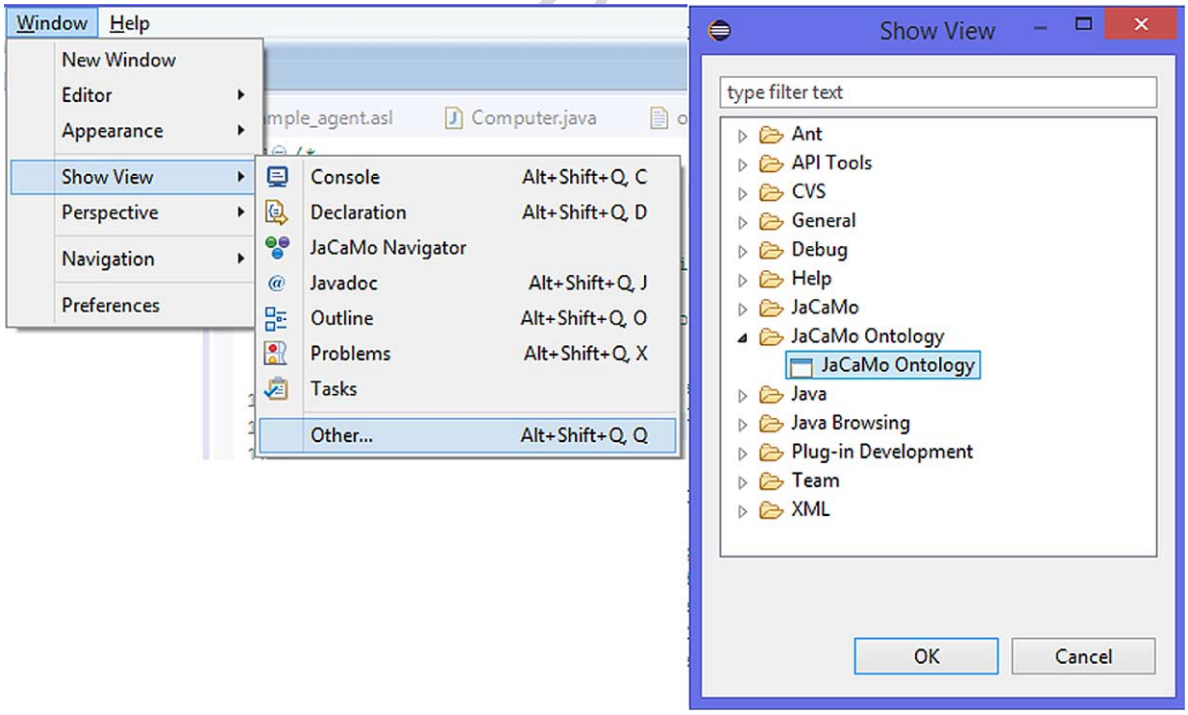

Fig. 6. How to activate our plug-in for ontology-based MDE programming of MAS in Eclipse for JaCaMo.

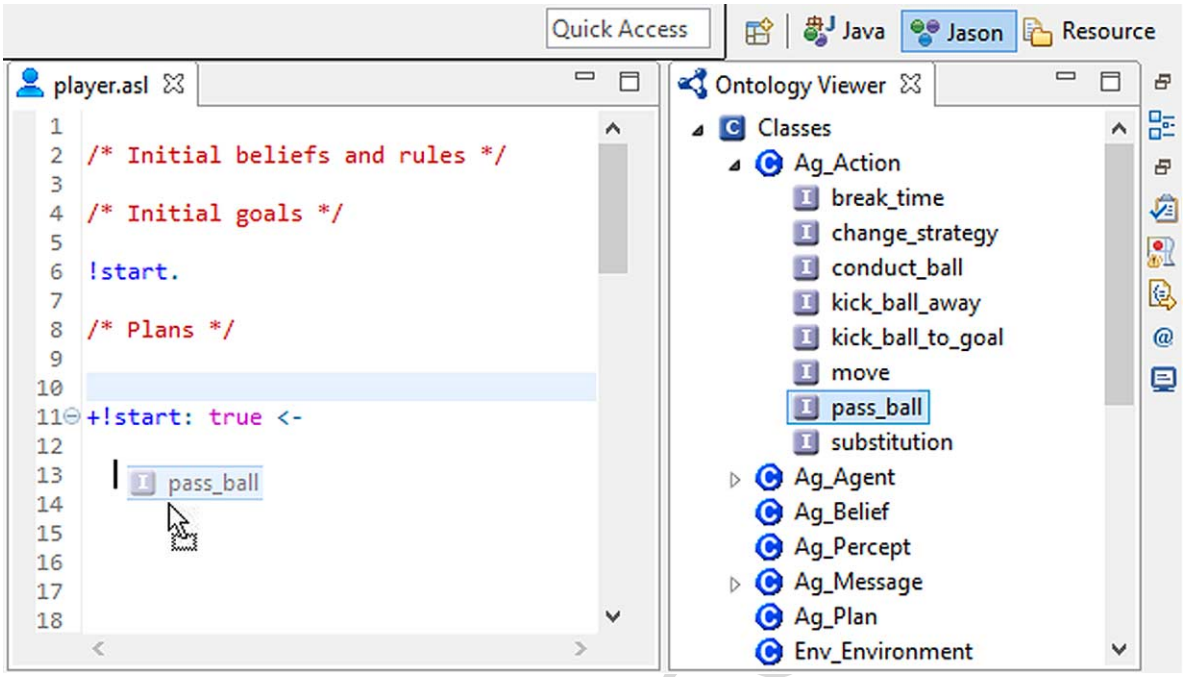

Fig. 7. Ontology-based drag-and-drop in Eclipse for MAS coding – first shown in our demo paper (Freitas et al., 2015).

elements from the model on the right side can be dragged to the left side that represents the AgentSpeak code of a Jason agent (in this case *player.asl*). As exemplified in Fig. 7, the programmer is dragging and dropping the action *pass_ball* to be inserted in a plan of this type of agent. Similarly, our tool provides the auto-complete feature from ontology to agent code, which is activated when the developer is typing MAS code (or press the shortcut "ctrl+space"). Then, the available options based on the ontology are presented to the programmer as suggestions. One example is when coding the plan's context, which may be composed of ontology-based queries (e.g., verifying if an individual belongs to a concept).

Our method to generate agent code from the ontology model can consider how each element can be inserted into the different parts of the MAS under development. Specifically, elements from the ontology can be dragged to generate code for each of the different parts of JaCaMo, such as Jason, CArtAgO, Moise, or the JCM file that corresponds to the initial specification of a JaCaMo project. In other words, the MAS code generation approach may consider the context, and the characteristics of what is being transformed into code (instance, concept, relationship, and so on). For example, when dropping *Org_ Roles* within a plan's body, the action to adopt these roles can be automatically created; when *Ag_Messages* are dropped, the action to send the corresponding messages can be generated; when dropping an *Org_ObligationNorm*, a plan could be created with a triggering event for the belief addition generated by the perception of the obligation to commit to that obligation; and one last example is when adding *Env_Artifacts* to an agent plan, then the programmer could be wishing to create, focus, or destroy a CArtAgO artifact, and these options may be suggested. Our strategies to convert elements from the proposed ontology model to MAS code exemplified above are depicted in Table 2. Moreover, while the MAS is being programmed, the most up-to-date information on its code can serve as feedback to update the instantiated ontology model to which it corresponds.

Table 2
Strategies to convert elements from our ontology model to MAS code

Ontology element	Can be dropped in	Generates MAS code regarding	When dropped changes ontology in
		Agent elements	
Ag_Action	plan body	agent action inside the plan	has-action
Ag_Goal	agent code	initial goal	has-goal
Ag_Agent	project folder	agent file	Ag_Agent
Ag_Message	plan body	send msg internal action	sends-message, Ag_Message
Ag_Belief	agent code	initial belief	has-belief
Ag_Plan	agent code	initial plan	has-plan
		Environment elements	
Env_Space	plan body	join workspace	has-space
Env_ObservableProperty	agent code	new plan triggered by that percept	has-plan
Env_Operation	plan body	agent action inside the plan using that operation	has-action
Env_Artifact	plan body	focus/create/destroy	–
		Organization elements	
Org_Goal	agent code	new plan to achieve that goal	has-plan
Org_Role	agent plan	adopt role action	has-action
Org_Group	agent plan	create/remove group	has-action
Org_Mission	agent plan	commit/remove mission	has-action
Org_ObligationNorm	agent code	plan to react to the percept of an obligation	has-plan
Org_PermissionNorm	agent code	plan to react to the percept of a permission	has-plan
Org_ProhibitionNorm	agent code	plan to react to the percept of a prohibition	has-plan

6. Empirical evaluation

In order to evaluate the practical consequences of our research, we conducted a proof of concept experiment with graduate computer science students enrolled in a course named "Multi-Agent Systems". Each participant evaluated the two parts of our proposal: *(i)* the use of the ontology for modeling different MAS; and *(ii)* the development of code for JaCaMo using the ontology-based software tool. Their experiences were surveyed by means of questionnaires covering each of these two tasks of MAS modeling and programming through instantiating the proposed ontology and using the plug-in.

The required background expertise and profile of the participants is as follows: they should already have prior knowledge on the topics of MAS and JaCaMo, they have to know how to use the Eclipse IDE to code JaCaMo projects, and they must not have any previous information about the ontology presented in this work until starting the experiment. The evaluation reported in this paper was conducted with 5 participants. Before starting the experiments, they received prior instructions about ontology and Protégé in order to perform the activities with the minimum required knowledge. The new part introduced to them was an overview of our MDE process: how to first model their projects in terms of our ontology, and then how to load such instantiated ontology in the plug-in for Eclipse that supports JaCaMo programming with the new code generation functionality. The results and analysis from evaluating the ontological model and tool are reported as follows.

6.1. Model evaluation

To evaluate if our ontology model correctly abstracts the desired MAS characteristics, the participants received the corresponding OWL file, and their first task was to instantiate the concepts and relationships to generate the populated OWL file to be used later in the programming step. This part evaluates if the MAS concepts defined in our ontology are correct, complete, and can be used for modeling and specifications of MAS. To interact with the ontology, the Protégé ontology editor was used.

After modeling their MAS in the ontology, the participants received questionnaires composed of: *(i)* affirmations to survey how much they agree based on a five-point Likert scale; and *(ii)* open questions to extract their opinions. The options given in the Likert scale are the following: strongly agree, agree, neither agree nor disagree, disagree and strongly disagree. The answers can be seen in the frequency diagram shown in Fig. 8, which corresponds to the following affirmations:

- **easy to specify:** it was easy (i.e., practical and efficient) to specify the MAS in the ontology;
- **easy to understand:** it was easy to understand and explain the MAS through the set of concepts of each dimension represented in the ontology;
- **correctness:** the ontology concepts in each dimension correctly specify any given MAS referring to JaCaMo;
- **completeness:** the MAS specification in the ontology is complete since the ontology concepts cover the major elements that can exist in MASs projected for JaCaMo;
- **usefulness in modeling:** the representation of MAS in an ontology was useful for modeling a project for JaCaMo; and
- **usefulness in programming:** the representation of MAS in an ontology can be useful for programming the a JaCaMo project.

Figure 8 shows that the affirmation most accepted is that the ontology is **complete**, followed by that it is **correct** and **useful in modeling**. The affirmations with more disagreement are that it is **easy to specify**, **easy to understand** and **useful in programming** the MAS. The following **advantages** of using the ontology were commented: *(i)* standardization of structures and concepts; *(ii)* more expressiveness to represent further MAS concepts than with just the diagrams of Prometheus and Moise; *(iii)* just one specification to model a complete MAS; *(iv)* avoid inconsistencies when naming the MAS elements (especially in projects with several people); and *(v)* the relation between MAS components can become clearer, such as among agents and actions. The **disadvantages** commented were that: *(i)* specify an ontology is not a trivial task (previous knowledge of the concepts is required); *(ii)* it can be more confusing

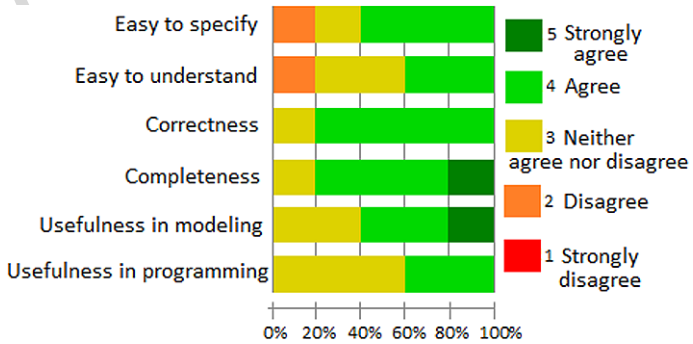

Fig. 8. Frequency diagram to evaluate benefits of the proposed ontological approach for modeling MAS.

to specify the MAS in an ontology; and *(iii)* the Protégé can be complex to edit the ontology (however other software could be used to improve productivity). The participants highlighted that the ontology is more flexible to embrace the needs of MAS, and it groups in one place what other approaches represent in several separate diagrams. Also, they found Prometheus and Moise diagrams more intuitive at first, but without covering as many details about the MAS environment as this ontology does. The features observed as improvements to this part of the work are to:

- *(i)* generate diagrams and code from the ontology, for example, a general system overview diagram containing agents, perceptions and so on;
- *(ii)* unify or integrate the ontology and diagrams of agents and organizations; and
- *(iii)* define a diagram or language to represent MAS environments.

6.2. Tool evaluation

The participants received our Eclipse plug-in, where they had to load the ontology previously instantiated by them and use the drag-and-drop and auto-complete features to develop agent code based on their models. Then, the participants were surveyed by means of questionnaires to extract their perceptions and opinions about the new tool. The answers can be seen in the frequency diagram depicted in Fig. 9, which investigates the following affirmations:

- **easy to understand:** it was easy to understand the functioning of the software that uses the ontology to help in programming;
- **easy to use:** it was easy to use the plug-in;
- **efficiency:** it was efficient (fast) to use the tool;
- **easy to visualize:** it was easy to visualize the ontology components in the way they appear in the software interface;
- **intuitive to use:** it was intuitive how the ontology elements generate MAS code (drag-and-drop and auto-complete);
- **model-program approximation:** the ontology and the software approximate and improve the transition between MAS specification/modeling and MAS programming;
- **help in development:** the ontology and the software help to develop code (MAS programming);
- **avoid programming errors:** programmers make less mistakes or inconsistencies when the code is generated from the ontology; and
- **usefulness in programming:** an ontology for MAS and the proposed plug-in can be useful when programming, as they bring developers a new functionality without impeding the use of other development tools (in this case, Eclipse).

Figure 9 shows the participants pointing out that the tool is **intuitive to use**, **easy to understand** and **easy to visualize**, followed by that it is **useful in programming**, **easy to use** and improves **model-program approximation**. The participants also commented that the plug-in helps in code consistency (e.g., it facilitates coding using the same terms), and it prevents developers from using terms outside the ontology. This approach also provides an overview about the MAS to be visualized inside the programming context, combined with features of dragging content from a model to MAS code.

More MAS code can be generated from this model, and the ontology can be more restrictive during MAS coding (i.e., indicate errors or mismatches between model and code). Also, the Eclipse plug-in could allow to edit the model (for example, to include new instances), which would discard the need of using an ontology editing tool to edit the MAS model. These are suggestions to improve the proposed

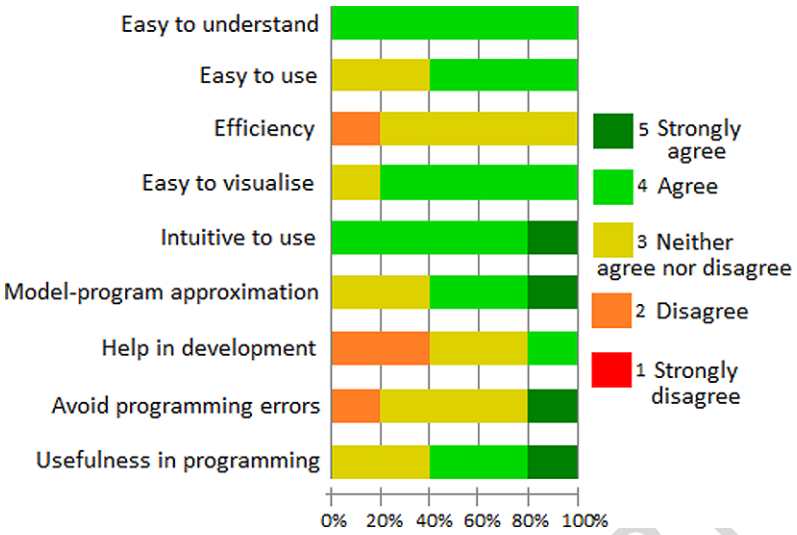

Fig. 9. Frequency diagram to evaluate benefits of the proposed tool.

such tool using the ontology model-based development in agent-oriented programming paradigms. Also, the automatic update of the ontology when the MAS code changes is not currently implemented (it can be cumbersome to manually synchronise model and code). Currently, the programmer has to manually change the ontology to reflect code changes and such ontology maintenance can be very laborious. This can be solved by implementing features to highlight mismatches between MAS code and its corresponding model, which is interesting to keep aligned (in other words, refactoring mechanisms to improve synchronizations between model and code).

6.3. *Comparison between Prometheus and the ontology-based approach*

Our evaluation of the use of the proposed model and tool so far indicates that they are advantageous for MAS development in various different ways. However, it is interesting to have results on comparative experiments in which the participants perform the same task using both a previous approach and the proposed one. While the previous experiments were more exploratory (to discover new things regarding the research), the later ones focus on establishing comparisons and confirming characteristics from predefined hypothesis about the proposal.

This comparative evaluation focuses on the comprehensive modeling of a same MAS example using two different approaches: Prometheus (Padgham et al., 2005) and our ontology. In the first experiment round that is reported in Sections 6.1 and 6.2, each participant modeled and developed a different MAS application using solely the ontological approach and we surveyed their opinions. In the second round, all participants have to develop the same MAS application model using two different approaches. In both cases, the target MAS programming platform was JaCaMo (Boissier et al., 2013). The population that joined these new rounds of experiments is composed of 5 participants. They were also enrolled in the course named "Multi-Agent Systems", but from a different semester than participants of the previous experiments. They have declared their expertise and profile as follows:

– **Multi-Agent Systems and JaCaMo:** 3 participants only had their first contact with MAS in that course, while 2 had already previously worked with MAS in practice. The same proportion applies to the question of working with JaCaMo in practice.

– **Ontology and Protégé:** 3 participants had previous knowledge in ontology and Protégé before this course, whereas 2 only had their first contact with ontologies during the experiment.
– **Prometheus:** their first contact with Prometheus was in the course, during the experiment.

Initially, they all received explanations and demonstrations of both approaches (ontology and Prometheus). The tool for working with Prometheus models was the Eclipse plug-in called Prometheus Design Tool, and the tool for the ontology approach was Protégé. They were guided through a learning and an experiment scenario. The learning scenario was an adapted specification of the JaCaMo Hello World,[6] and the experiment scenario was an adapted specification of the Gold Miners application.[7] For the experiment, all participants used both approaches for modeling a specific MAS, but they were divided into two groups. Each group started with a different methodology to avoid bias.

After modeling the same MAS using the two different approaches, the participants were given a 5-point Likert scale for assessing the following assertions (with "X" being replaced by Prometheus or Ontology in each affirmation):

– **A1.** To implement a system in JaCaMo, I find it easy to specify models in X.
– **A2.** I could understand the elements provided by the X approach.
– **A3.** The characteristics of a JaCaMo MAS are correctly represented in the X approach.
– **A4.** The X approach is complete since it covers all the essential elements of JaCaMo systems.
– **A5.** I believe that it is useful to represent my JaCaMo system in the X approach.
– **A6.** The X approach provides good support for MAS programming in JaCaMo.

Our comparison of the two approaches under those criteria is summarised in Fig. 10. In general, we observed better acceptance towards the ontology model, with the exception of assertion 2 (A2), regarding how easy it is to understand the elements, and this may also be due to the fact that the ontology was considered more complete (A4). It was considered more correctly aligned with JaCaMo (A3) and therefore more useful for MAS modeling in this context (A5). The participants' opinions were more positive for the ontology than Prometheus in their support for programming as well (A6).

In the last part of our questionnaires, we asked each participant to make free comments and suggestions about any of the two approaches. For instance, we wanted to know if they thought that the meaning of something was confusing or wrong for representing JaCaMo properties, and if something is missing

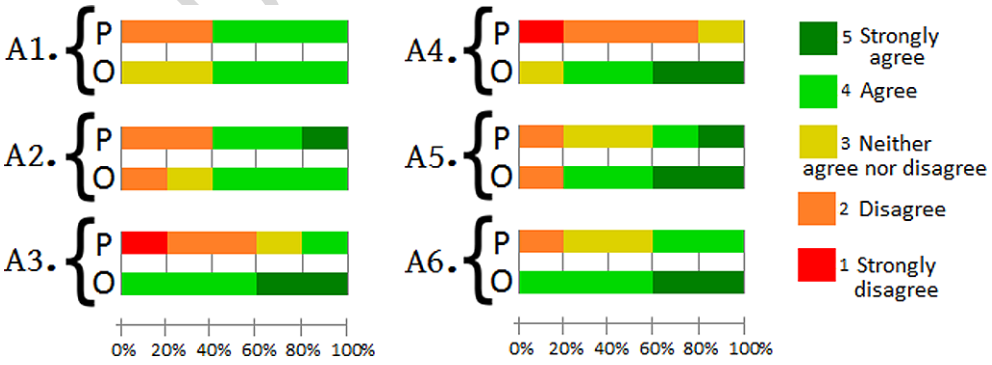

Fig. 10. Comparing Prometheus (P) with the ontological approach (O).

[6] Obtained from http://jacamo.sourceforge.net/tutorial/hello-world/.
[7] Obtained from https://multiagentcontest.org/2007/.

in order to fully specify MAS as JaCaMo projects. We obtained from one of the participants the opinion that Prometheus is more generic and does not cover some JaCaMo aspects, and that the ontology covers well the properties of JaCaMo, but, in some cases, its specification may be more laborious.

Another participant observed that the ontology serves best the project needs than Prometheus, since it is possible to better represent in the ontology what is intended to be implemented in JaCaMo.

A participant pointed out that the use of Prometheus is more natural because it is a visual tool, with easily defined flows, however it does not provide all the necessary functionality to define a complete MAS to be later developed in JaCaMo. The ontology approach was claimed to be more complete, but with a slower learning curve. According to another participant, the modeling in Prometheus is more intuitive because it has a graphical interface with drag-and-drop. However, the approach that uses ontology through Protégé provides more options, so they were able to do a much more complete modeling. It was suggested that, after finishing the ontological modeling, it would be interesting to export it as diagrams.

This subsection discussed the results for the experiments comparing our approach with Prometheus for modeling MAS. An extended comparative analysis between the models built in each of the approaches, and comparisons of each approach for other stages of AOSE is planned as future work. Although the results reported here are limited to the given population and applications that took place in the experiments, they help to indicate some interesting advantages regarding the approach proposed in this paper.

7. Final remarks

Although ontologies for MAS have been considered in many ways, few agent-based systems development platforms integrate ontology techniques. The use of ontologies for MAS modeling and development is emerging; however, current ontologies for MAS only cover parts of the whole picture, such as the environment or the organization. On the other hand, there are models and MDE approaches addressing the overall MAS development, but without using ontology, semantic reasoning or employing the models during the programming step. This context led to our proposal addressing the use of ontology in model-based design of MAS. Our previous research addressed MAS with ontologies, e.g. the inital idea of integrating ontologies in MAS platforms (Freitas et al., 2015); and, more specifically, the role of ontology within AOSE (Freitas et al., 2014). We also investigated the use of ontologies as semantic representation for agent plans (Freitas et al., 2014). Then, more aligned with the main topic of this paper, we presented a demonstration[8] of our ontology-based approach and tool for engineering MAS (Freitas et al., 2015). Next, one of our studies discusses the implications of coding in JaCaMo a complex MAS modeled in Prometheus (Freitas et al., 2016), thus assessing the combined used of Prometheus for modeling and JaCaMo for programming. We also made contributions to the complementary area of providing an infrastructure for MAS to interact with ontologies (Freitas et al., 2015) and we demonstrated its use in a scenario where agents argue about task reallocation based on ontologies (Panisson et al., 2015).

Based on that previous work, we presented in this paper a MAS modeling methodology based on ontologies and a development tool based on models instantiated through such a methodology. The motivations for this research reside in the difficulty to design, model, integrate and develop MAS, which normally require the inclusion of several system's components often approached from different angles and formalisms. For example, the JaCaMo (Boissier et al., 2013) framework for MAS programming combines three separate technologies: Jason (Bordini et al., 2007) for coding autonomous agents in an AgentSpeak dialect, CArtAgO (Ricci et al., 2006) for programming the environment as artifacts in

[8] A video of this demonstration is available at https://youtu.be/Lt5ZVG1cgBQ.

Java, and Moise (Hübner et al., 2010) for specifying MAS organizations in XML. These distinct starting points to code the MAS make it desirable a single model combining all MAS dimensions, that could be used to give initial support to code generation for all those platforms.

We claim that our MAS modeling and development approaches increase the flexibility and ease the engineering of such systems. First, the MAS is modeled from the start in a single integrated formalism, and the ontology allows the designer to connect and reuse knowledge of one dimension into others and also across different applications, improving both system development as well as MAS interoperability. For example, the characteristics of a MAS dimension (e.g., environment) could be used to define properties of another (e.g., organizational). Our approach also enables the conversion of MAS defined in ontologies to programming code in specific agent platforms while remaining flexible enough to accommodate the needs of MAS designers. Based on such modeling approach, we elaborated a tool as a plug-in for Eclipse that uses MDE to support MAS programming according to its ontology design.

This research opens possibilities of applying the proposed ontology in many other ways. In terms of MAS design, such ontology model provides a global conceptual view which in combination with MDE can result in tools, for example, to verify model consistency, perform inferences with semantic reasoners, query instantiated models, develop/visualize MAS specifications in ontologies, and support for programming. As result, developers obtain new features for developing complex software systems with an infrastructure that combines and applies modeling, software and knowledge engineering principles. For example, MDE can obtain unambiguous definitions from meta-models formally defined in ontology languages, and reasoners can validate meta-models automatically or generate MAS code from models, all of which contribute to more principled ways to develop MAS. As future work, we plan to investigate these other possibilities of applications that can be derived from this research.

As result of our work in the areas of MDE, ontology and MAS, the first practical evaluations of our research are indicating that the use of ontology facilitates the modeling of MAS, supports agent programming and provides a basis for reasoning about the modeled system. Our experiments help to highlight advantages as well as limitations and possibilities for improvements in the current state of the proposed technique. Thus, the initial evaluations reported here provide sufficient evidence (although in a simulated laboratory environment) to suggest that our approach is feasible for representing useful models in the desired domain of agent-based software systems. It is currently ongoing work in this research to make more comparisons about the processes of modeling, programming and verification with the standard approach versus our new one, as well as an analysis of the resulting models and codes.

One could also consider the possibility of applying UML to substitute the use of OWL to describe a given content or domain of knowledge. It would be also the case that one of these alternative choices brings different sets of tools to work with, with more advantages in some aspects, however there would also be drawbacks in other parts. This work points out our investigation considering the use of ontology as an alternative. Studies to compare these different paradigms would be interesting for the communities working on them, and specially when considering the application context of this work, which is to support AOSE. As we already mentioned in this paper, some authors argue that UML, in its original form, provides insufficient support for modeling MAS (da Silva and de Lucena, 2003). However, it might be the case that some new extension of UML may enable its suitable application for AOSE.

This work emphasizes an ontology for modeling agents, the environment in which they operate, and the organization to support the coordination of autonomous agents. Many researchers in MAS believe that interactions among the agents are crucial issues to be considered when developing such systems. Interaction includes communication, intentions, obligations, and commitments. Although this topic is being conceptually discussed for quite some time (Singh, 1999), only in recent work (Zatelli et al.,

2016) JaCaMo has been extended in order to provide such features at the programming level by means of a new *interaction dimension*. Thus, the modeling of interactions to extend our ontology would be an interesting future work to explore. This has not been fully addressed so far given how recent such developments are in the context of MAS projects in IDEs for JaCaMo.

References

Atkinson, C., Gutheil, M. & Kiko, K. (2006). On the relationship of ontologies and models. In *Proceedings of the 2nd Workshop on Meta-Modelling* (Vol. 96, pp. 47–60).

Atkinson, C. & Kühne, T. (2003). Model-driven development: A metamodeling foundation. *IEEE Softw.*, *20*(5), 36–41. doi:10.1109/MS.2003.1231149.

Baader, F., Horrocks, I. & Sattler, U. (2004). Description logics. In *Handbook on Ontologies* (pp. 3–28). Springer. doi:10.1007/978-3-540-92673-3_1.

Bayardo, R.J. Jr., Bohrer, W., Brice, R., Cichocki, A., Fowler, J., Helal, A., Kashyap, V., Ksiezyk, T., Martin, G., Nodine, M., Rashid, M., Rusinkiewicz, M., Shea, R., Unnikrishnan, C., Unruh, A. & Woelk, D. (1997). InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. *ACM SIGMOD International Conference on Management of Data*, *26*(2), 195–206. doi:10.1145/253262.253294.

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F. & Stein, L.A. (2004). OWL web ontology language reference. Technical report, W3C. http://www.w3.org/TR/owl-ref/.

Bézivin, J. (2006). Model driven engineering: An emerging technical space. In *Generative and Transformational Techniques in Software Engineering* (pp. 36–64). Springer. doi:10.1007/11877028_2.

Boissier, O., Bordini, R.H., Hübner, J., Ricci, A. & Santi, A. (2013). Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, *78*(6), 747–761. doi:10.1016/j.scico.2011.10.004.

Bordini, R.H., Dastani, M. & Winikoff, M. (2006). Current issues in multi-agent systems development. In G.M.P. O'Hare, A. Ricci, M.J. O'Grady and O. Dikenelli (Eds.), *Engineering Societies in the Agents World* (Vol. 4457, pp. 38–61). Springer. doi:10.1007/978-3-540-75524-1_3.

Bordini, R.H., Hübner, J.F. & Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons. doi:10.1002/9780470061848.

Budinsky, F. (2004). *Eclipse Modeling Framework: A Developers Guide*. Addison-Wesley.

Carson, J.S. (2002). Model verification and validation. In *Proceedings of the 2002 Winter Simulation Conference* (Vol. 1, pp. 52–58). doi:10.1109/WSC.2002.1172868.

da Silva, V.T. & de Lucena, C.J. (2003). MAS-ML: A multi-agent system modeling language. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (pp. 304–305). ACM. doi:10.1145/949344.949424.

Freitas, A., Bordini, R.H., Meneguzzi, F. & Vieira, R. (2015). Towards integrating ontologies in multi-agent programming platforms. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology* (Vol. 3, pp. 225–226). doi:10.1109/WI-IAT.2015.207.

Freitas, A., Cardoso, R.C., Vieira, R. & Bordini, R.H. (2016). Limitations and divergences in approaches for agent-oriented modelling and programming. In M. Baldoni, J.P. Müller, I. Nunes and R. Zalila-Wenkstern (Eds.), *Engineering Multi-Agent Systems* (pp. 88–103).

Freitas, A., Hilgert, L., Marczak, S., Meneguzzi, F., Bordini, R.H. & Vieira, R. (2015). A multi-agent systems engineering tool based on ontologies. In *34th International Conference on Conceptual Modeling*. Springer.

Freitas, A., Panisson, A.R., Hilgert, L., Meneguzzi, F., Vieira, R. & Bordini, R.H. (2015). Integrating ontologies with multi-agent systems through CArtAgO artifacts. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology* (pp. 143–150). doi:10.1109/WI-IAT.2015.116.

Freitas, A., Schmidt, D., Panisson, A., Meneguzzi, F., Vieira, R. & Bordini, R.H. (2014). Applying ontologies and agent technologies to generate ambient intelligence applications. In *Joint Proceedings Collaborative Agents – Research & Development, CARE for Intelligent Mobile Services & Agents, Virtual Societies and Analytics* (pp. 22–33). doi:10.1007/978-3-662-46241-6_3.

Freitas, A., Schmidt, D., Panisson, A., Meneguzzi, F., Vieira, R. & Bordini, R.H. (2014). Semantic representations of agent plans and planning problem domains. In F. Dalpiaz, J. Dix and M.B. van Riemsdijk (Eds.), *Engineering Multi-Agent Systems* (Vol. 8758, pp. 351–366). Springer. doi:10.1007/978-3-319-14484-9_18.

Gascueña, J.M., Navarro, E. & Fernández-Caballero, A. (2012). Model-driven engineering techniques for the development of multi-agent systems. *Engineering Applications of Artificial Intelligence*, *25*(1), 159–173. doi:10.1016/j.engappai.2011.08.008.

Gruber, T.R. (1993). A translation approach to portable ontology specifications. *Knowl. Acquis.*, *5*(2), 199–220. doi:10.1006/knac.1993.1008.
Hadzic, M., Wongthongtham, P., Dillon, T. & Chang, E. (2009). *Ontology-Based Multi-Agent Systems*. Springer. doi:10.1007/978-3-642-01904-3.
Horridge, M. & Bechhofer, S. (2011). The OWL API: A Java API for OWL ontologies. *Semant. Web*, *2*(1), 11–21. doi:10.3233/SW-2011-0025.
Hübner, J.F., Boissier, O., Kitio, R. & Ricci, A. (2010). Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, *20*(3), 369–400. doi:10.1007/s10458-009-9084-y.
Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W. & Wimmer, M. (2006). Lifting metamodels to ontologies: A step to the semantic integration of modeling languages. *Model Driven Engineering Languages and Systems*, *4199*, 528–542. doi:10.1007/11880240_37.
Klapiscak, T. & Bordini, R.H. (2008). JASDL: A practical programming approach combining agent and semantic web technologies. In *The 6th International Workshop on Declarative Agent Languages and Technologies* (Vol. 5397, pp. 91–110). Springer. doi:10.1007/978-3-540-93920-7_7.
Mascardi, V., Ancona, D., Barbieri, M., Bordini, R.H. & Ricci, A. (2014). CooL-AgentSpeak: Endowing AgentSpeak-DL agents with plan exchange and ontology services. *Web Intelligence and Agent Systems*, *12*(1), 83–107. doi:10.3233/WIA-140287.
Moreira, Á.F., Vieira, R., Bordini, R.H. & Hübner, J.F. (2005). Agent-oriented programming with underlying ontological reasoning. In *Proceedings of the 3rd International Workshop on Declarative Agent Languages and Technologies* (pp. 155–170). Berlin, Heidelberg: Springer. doi:10.1007/11691792_10.
Okuyama, F.Y., Vieira, R., Bordini, R.H. & da Rocha Costa, A.C. (2006). An ontology for defining environments within multi-agent simulations. In *Workshop on Ontologies and Metamodeling in Software and Data Engineering*.
Padgham, L., Thangarajah, J. & Winikoff, M. (2005). Tool support for agent development using the Prometheus methodology. In *Fifth International Conference on Quality Software* (pp. 383–388). doi:10.1109/QSIC.2005.66.
Padgham, L. & Winikoff, M. (2002). Prometheus: A methodology for developing intelligent agents. In F. Giunchiglia, J. Odell and G. WeiB (Eds.), *Agent-Oriented Software Engineering III* (Vol. 2585, pp. 174–185). Springer. doi:10.1007/3-540-36540-0_14.
Panisson, A.R., Freitas, A., Schmidt, D., Hilgert, L., Meneguzzi, F., Vieira, R. & Bordini, R.H. (2015). Arguing about task reallocation using ontological information in multi-agent systems. In *12th International Workshop on Argumentation in Multiagent Systems*.
Pavón, J., Gómez-Sanz, J. & Fuentes, R. (2006). Model driven development of multi-agent systems. In A. Rensink and J. Warmer (Eds.), *Model Driven Architecture – Foundations and Applications* (Vol. 4066, pp. 284–298). Berlin, Heidelberg: Springer. doi:10.1007/11787044_22.
Ricci, A., Viroli, M. & Omicini, A. (2006). CArtAgO: An infrastructure for engineering computational environments in MAS. In D. Weyns, H.V.D. Parunak and F. Michel (Eds.), *3rd International Workshop Environments for Multi-Agent Systems* (pp. 102–119).
Roebuck, K. (2012). *Model-Driven Architecture (MDA): High-Impact Strategies – What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors*. Emereo Publishing.
Selic, B. (2003). The pragmatics of model-driven development. *IEEE Software*, *20*(5), 19–25. doi:10.1109/MS.2003.1231146.
Singh, M.P. (1999). An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, *7*(1), 97–113. doi:10.1023/A:1008319631231.
Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A. & Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Web Semant.*, *5*(2), 51–53. doi:10.1016/j.websem.2007.03.004.
Staab, S., Walter, T., Groner, G. & Parreiras, F. (2010). Model driven engineering with ontology technologies. In U. Abmann, A. Bartho and C. Wende (Eds.), *Reasoning Web. Semantic Technologies for Software Engineering* (Vol. 6325, pp. 62–98). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-15543-7_3.
Tran, Q.-N.N. & Low, G. (2008). MOBMAS: A methodology for ontology-based multi-agent systems development. *Inf. Softw. Technol.*, *50*(7–8), 697–722. doi:10.1016/j.infsof.2007.07.005.
Uez, D.M. & Hübner, J.F. (2014). Environments and organizations in multi-agent systems: From modelling to code. In *2nd International Workshop on Engineering Multi-Agent Systems* (pp. 181–203). doi:10.1007/978-3-319-14484-9_10.
Zarafin, A.-M. (2012). Semantic description of multi-agent organizations. Master's thesis, Automatic Control and Computers Faculty, Computer Science and Engineering Department, University Politehnica of Bucharest.
Zatelli, M.R. & Hübner, J.F. (2014). The interaction as an integration component for the JaCaMo platform. In *2nd International Workshop on Engineering Multi-Agent Systems* (pp. 431–450). doi:10.1007/978-3-319-14484-9_22.
Zatelli, M.R., Ricci, A. & Hübner, J.F. (2016). Integrating interaction with agents, environment, and organisation in JaCaMo. *International Journal of Agent-Oriented Software Engineering*, *5*(2–3), 266–302. doi:10.1504/IJAOSE.2016.080889.