The Journal of Systems and Software 108 (2015) 77–109

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

MAS-ML 2.0: Supporting the modelling of multi-agent systems with different agent architectures

Enyo José Tavares Gonçalves [a,*], Mariela I. Cortés [b], Gustavo Augusto Lima Campos [b], Yrleyjander S. Lopes [b], Emmanuel S.S. Freire [b], Viviane Torres da Silva [c], Kleinner Silva Farias de Oliveira [d], Marcos Antonio de Oliveira [e]

[a] Universidade Federal do Ceará, Av. José de Freitas Queiroz, 5003 – Cedro CEP 63900-000, Quixadá, CE, Brazil
[b] Universidade Estadual do Ceará, Fortaleza, CE, Brazil
[c] IBM Research Brazil, Rio de Janeiro, RJ, Brazil
[d] Universidade do Vale do Rio dos Sinos, São Leopoldo, RS, Brazil
[e] Universidade Federal do Ceará, Quixadá, CE, Brazil

ARTICLE INFO

Article history:
Received 26 September 2014
Revised 16 April 2015
Accepted 4 June 2015
Available online 12 June 2015

Keywords:
Proactive and reactive agents
Multi-agent system
Modelling language

ABSTRACT

Multi-agent systems (MAS) involve a wide variety of agents that interact with each other to achieve their goals. Usually each agent has a particular internal architecture defining its main structure that gives support to the interaction among the entities of MAS. Many modelling languages have been proposed in recent years to represent the internal architectures of such agents, for instance the UML profiles. In particular, we highlight MAS-ML, an MAS modelling language that performs a conservative extension of UML while incorporating agent-related concepts to represent proactive agents. However, such languages fail to support the modelling of the heterogeneous architectures that can be used to develop the agents of an MAS. Even worse, little has been done to provide tools to help the systematic design of agents. This paper, therefore, aims to extend the MAS-ML metamodel and evolve its tool to support the modelling of not only proactive agents but also several other architectures described in the literature.

1. Introduction

Nowadays, agent technology has been widely applied to solve a vast set of problems. Russell and Norvig (2003) define an agent as an entity that can perceive its environment through sensors and act in the environment through actuators. Unlike objects, agents are more complex entities with behavioural properties, such as (i) autonomy (i.e., they are able to execute without interacting with humans), and (ii) interaction (i.e., they are able to interact by sending and receiving messages and not by explicit task invocation (Wagner, 2003). Multi-agent system (MAS) is the subarea of artificial intelligence that investigates the behaviour of a set of autonomous agents, aiming to resolve a problem that is beyond the capacity of a single agent (Jennings, 1996).

A simple agent can act based on reactive or proactive behaviour and can be classified according to its internal architecture that determines distinct agency properties, attributes and mental components (Russell and Norvig, 2003). Russell and Norvig (2003) define four types of agents according to their internal architectures: Simple Reflex Agent, Model-Based Reflex Agent, Goal-based Agent and Utility-Based Agent. The internal architecture of an agent is selected according to the environment characteristics and to the subproblem that the agent will resolve. In Weiss (1999), environments are classified as (i) fully observable or partially observable, depending of the degree of awareness; (ii) deterministic (foreseeable) or stochastic (unforeseeable), depending on the predictability of the environment evolution; (iii) episodic (an action does not depend on previous actions) or sequential (the execution of an action depends on previous actions); (iv) static (the environment does not change while the agent is reasoning) or dynamic (it changes while the agent is reasoning); and (v) discrete (it has a finite quantity of states) or continuous (it has a continuous quantity of states). MAS may encompass multiple types of agents with different internal architectures (Weiss, 1999).

The agent-oriented development paradigm requires adequate techniques to explore its benefits and features in order to support the construction and maintenance of this type of software (Zambonelli et al., 2001). As it is the case with any new software engineering paradigm, the successful and widespread deployment of MASs

* Corresponding author. Tel.: +55 88 3412 0919; fax: +55 88 3412 3348.
E-mail addresses: enyo.goncalves@gmail.com, enyo@ufc.br (E.J.T. Gonçalves), mariela@larces.uece.br (M.I. Cortés), gustavo@larces.uece.br (G.A.L. Campos), yrley@larces.uece.br (Y.S. Lopes), savio@larces.uece.br (E.S.S. Freire), vivianet@br.ibm.com (V.T. da Silva), kleinnerfarias@unisinos.br (K.S.F. de Oliveira), deoliveira.ma@ufc.br (M.A. de Oliveira).

http://dx.doi.org/10.1016/j.jss.2015.06.008
0164-1212/

requires modelling languages that explore the use of agent-related abstractions and promote the traceability from the design models to code. To reduce the risk when adopting a new technology it is convenient to present it as an incremental extension of known and trusted methods, and to provide explicit engineering tools that support industry-accepted methods of technology deployment (Castro et al., 2006). A modelling language for a multi-agent system preferably should be an incremental extension of a known and trusted modelling language. Since agents and objects coexist in MASs, the UML modelling language (Castro et al., 2006) can be used as a basis for developing MAS modelling languages. The UML modelling language is a de facto standard for object-oriented modelling. UML is used both in industry and academia for modelling object-oriented systems. Nevertheless, in its original form UML provides insufficient support for modelling MASs.

There are several MAS modelling languages (Argente et al., 2009) that have extended UML to model agent-oriented characteristics. However, there is still a need for a *modelling language capable of modelling different internal agent architectures*. The main goal of this paper is to present a *UML-based modelling language able to model the internal agent characteristics presented in Russell and Norvig (2003)*. In order to accomplish our goal, we have extended MAS-ML (multi-agent system modelling language) (Silva et al., 2008a), a modelling language that performs a conservative extension of UML based on the agent-oriented concepts defined in the conceptual framework TAO (Taming Agents and Objects) (Silva and Lucena, 2004). MAS-ML and TAO were originally designed to support the modelling of only proactive agents that are goal-based entities guided by predefined plans. However, not all MASs require proactive agents, as in the case of simulations for an ant colony (Dorigo and Stützle, 2004). Therefore, it is important to *extend MAS-ML and also TAO* to be able to model not only proactive agents but also reactive ones. Together with the extension of MAS-ML and TAO we also felt the need to *extend the MAS-ML modelling environment*.

We have chosen MAS-ML since it gives support to the modelling of (i) the main MAS entities: agents, organization and environments; (ii) the static and dynamic properties of a MAS; (iii) agent roles, what is important while defining agent societies; and (iv) proactive agents. Languages such as AUML (Agent Unified Modelling Language) (Odell et al., 2000; Guedes and Vicari, 2009) and AORML (Agent-Oriented Relationship Modelling Language) (Wagner, 2003; Wagner and Taveter, 2005) do not define (i) the environment as an abstraction, so it is not possible for model agents able to move from one environment to another; and (ii) the properties of organizations are not described, so it is not possible to model agents from one organization to other. One of the factors that influences the choice of the internal architecture is the type of environment in which the agent is inserted, so the environmental representation in diagrams of languages is desirable because once this is modelled, it is possible to explain its kind by textual notes, for example. For the organization, although not related to the choice of internal architecture, its representation is important from the point of view of coordination between agents. The ANote Modelling language (Choren and Lucena, 2005), for instance, does not support conventional objects used to model non-autonomous entities. In addition, in AML (Agent Modelling Language) (Cervenka, 2012), the communication relationships are modelled as a specialization of UML elements, which is not suitable for modelling agent communication.

In order to validate the extensions being proposed for MAS-ML 2.0, TAO and the modelling tool, a case study was performed in order to use our extensions to model the TAC-SCM system (Trading Agent Competition – Supply Chain Management) (Sadeh et al., 2003). Such a system simulates simultaneous auctions with consumers and suppliers of computers. Five agents implemented with different internal architectures were modelled by using MAS-ML 2.0 and other three modelling languages: original MAS-ML, AUML and AML. These different models allowed us to demonstrate the expressiveness of the proposed extensions over other approaches. MAS-ML 2.0 is able to model all four internal agent architectures while original MAS-ML, AUML and ML are able to model only the two simpler architectures.

The paper is structured as follows. The main internal architectures for agents, TAO and MAS-ML, are described in Section 2. Section 3 presents related works involving conceptual frameworks and modelling languages. The extension of TAO is described in Section 4, MAS-ML extension is detailed in Section 5 and the MAS-ML tool is detailed in Section 6. In Section 7 the modelling of the TAC-SCM (Trading Agent Competition – Supply Chain Management) (Sadeh et al., 2003) application is presented in order to illustrate the contribution of this work. Finally, conclusions and future works are discussed in Section 8.

2. Background

This section describes the main concepts needed to understand this work, including the concepts related to agent architectures, TAO conceptual framework and MAS-ML modelling language.

2.1. Agent architectures

The agent internal architectures can be categorized based on proactive and reactive foundations. In this context, four types of internal agent architectures were defined by Russell and Norvig (2003). These architectures are detailed in the next sections.

2.1.1. Simple Reflex Agents

A Simple Reflex (or reactive) Agent (Russell and Norvig, 2003) is considered the simplest internal architecture. Condition-action rules are used to select the actions based on the current perception. These rules follow the form "*if condition then action*" and determine the action to be executed if the perception occurs. This architecture assumes that at any time the agent receives information from the environment through sensors. These perceptions consist of the representation of state aspects that are used by the agent for making decisions. A subsystem is responsible for the decision-making, that is, responsible for processing the perception sequence and selecting a sequence of actions from the set of possible actions for the agent. The agent performs the selected action upon an environment through actuators.

2.1.2. Model-Based Reflex Agents

The structure of this kind of agent is similar to the Simple Reflex Agent presented before since it deals with the information by using condition-action rules. In order to handle partially observable environments and to reach a more rational performance, the agent is also able to store its current state in an internal model.

According to Weiss (1999), Model-Based Reflex Agents select actions by using the information in their internal states. A function called *next function* is introduced to map the perceptions and the current internal state into a new internal state used to select the next action. Such a state describes aspects of the world (called model) that cannot be seen in the current moment but were perceived earlier or have come out by inferences (Russell and Norvig, 2003).

2.1.3. Goal-Based Agents

Sometimes the knowledge about the current state of the environment is not enough to determine the next action and additional information about desirable situations is required. Goal-Based Agents are Model-Based Agents that set a specific goal and select the actions that lead to that goal. This allows the agent to choose a goal state among multiple possibilities.

Planning activity is devoted to finding the sequence of actions that are able to achieve the agent's goals (Russell and Norvig, 2003).

The sequence of actions previously established that leads the agent to reach a goal is termed plan (Silva and Lucena, 2004; Silva et al., 2008a). Thus, the Goal-Based Agent with planning involves the *next function* component and also includes the following elements:

- *Formulate goal function*, which receives the state and returns the formulated goal;
- *Formulate problem function*, which receives the state and the goal and returns the problem;
- *Planning*, which receives the problem and uses search and/or logic approaches to find a sequence of actions to achieve a goal; and
- *Action*, which is represented with its pre-conditions and post-conditions.

2.1.4. Utility-Based Agents

Considering the existence of multiple goal states, it is possible to define a measure of how desirable a particular state is. In this case, aiming to optimize the agent performance, the utility function is responsible for mapping a possible state (or group of states) to that measure, according to the current goals (Russell and Norvig, 2003). Thus, the utility function is incorporated into the architecture.

In addition, the Utility-Based Agent preserves the same elements as those of a Goal-Based Agent: *next function, formulate goal function, formulate problem function, planning and action.*

2.1.5. Internal architectures choice

We chose the internal architectures of agents related by Russell and Norvig (2003) because (i) they are widely used in development of MAS, and (ii) it is not possible model them with the available syntax in MAS modelling languages. Russell and Norvig (2003) and Weiss (1999) have related the choice of the internal agent architecture to the characteristics of the environment.

The proactive behaviour is especially suited to environments with well-defined tasks, that is, deterministic, observable and static environments. Designing proactive agents in stochastic and partially observable environments is a very complex task. Moreover, in dynamic environments, where preconditions can change while the agent is executing, the purely proactive behaviour can produce unwanted effects. In such cases, the agent must be able to stop the process of deliberation, or the execution of a plan that was deliberated, and respond adequately to the new environment conditions. Thus, in these environments where the response time is crucial, it is important to implement a reactive agent combined with proactive characteristics.

A Simple Reflex Agent may be better suited to completely observable environments, as this type of agent must make decisions based only on the current perceptions; therefore, it does not maintain historical past perceptions. In this case, we usually say that the agent forgets his own past.

Model-Based Reflex Agents can do better on a wider range of environments than reactive agents. They are able to maintain internal representations of the environment that is not being currently observed. Therefore, in partially observable environments reflex agents usually have higher performance than simple reactive agents.

Reflex agents are more suitable to sub-problems that require quick responses. When acting together, reflex agents can achieve good results, as in the case of ant colonies (Dorigo and Stützle, 2004). On the other hand, Goal-Based Agents that select a plan based on the goal the agent wants to execute are more suitable for environments that do not modify during the execution of the plan. If the environment changes, the sequence of actions being executed may fail due to the unexpected environment. Therefore, a Goal-Based Agent is suitable for partially observable, static and deterministic environments.

A Utility-Based Agent is more effective in two situations: (i) when the result of the execution of actions is uncertain, and (ii) when there is more than one goal to be achieved. In the first case, a utility-oriented agent can select the action that is more likely to produce the desired state. In the second case, the agent can select a goal with the highest utility. Thus, a Utility-Based Agent is more suitable for the non-deterministic and partially observable environments.

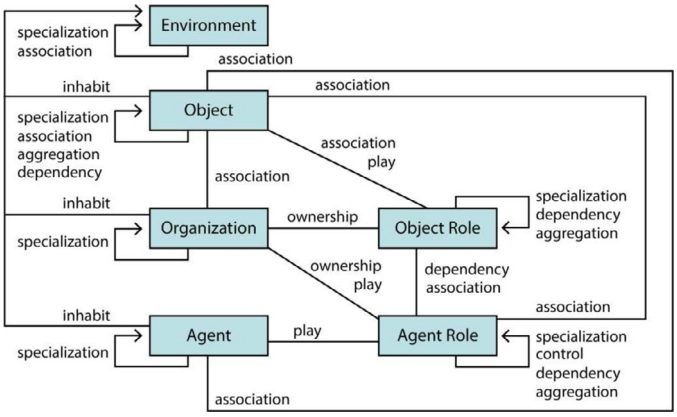

Fig. 1. Abstractions and relationships of TAO (Silva et al., 2003).

2.2. TAO: Taming Agents and Objects

The conceptual framework[1] TAO (Taming Agents and Objects) provides an ontology that covers the fundamentals of Software Engineering based on agents and objects and makes possible the development of MAS in large scale (Silva et al., 2003). This framework elicits an ontology that connects consolidated abstractions, such as objects and classes, and "emergent" abstractions, such as agents, roles and organizations, which are the foundations for agent and object-based software engineering. TAO presents the definition of each abstraction as a concept of its ontology and establishes the relationships between them. Fig. 1 shows the abstractions and relationships proposed in TAO. The abstractions of TAO are defined as follows:

- Object: It is a passive or reactive element that has state and behaviour and can be related to other elements.
- Agent: It is an autonomous, adaptive and interactive element that has a mental state. Its mental state has the following components: (i) beliefs (everything the agent knows), (ii) goals (future states that the agent wants to achieve), (iii) plans (sequences of actions that achieve a goal), and (iv) the actions themselves.
- Organization: It is an element that groups agents and sub-organizations, which play roles and have common goals. An organization hides intracharacteristics, properties and behaviours represented by agents inside it. It may restrict the behaviour of their agents and their sub-organizations through the concept of axiom, which characterizes the global constraints of the organization that agents and sub-organizations must obey.
- Object role: It is an element that guides and restricts the behaviour of an object in the organization. An object role can add information, behaviour and relationships that the object instance executes.
- Agent role: It is an element that guides and restricts the behaviour of the agent playing the role in the organization. An agent role defines (i) duties as actions that must be performed by the agent playing the role, (ii) rights as actions that can be performed by the agent playing the role, and (iii) protocol that defines an interaction between agent roles.
- Environment: It is an element that is the habitat for agents, objects and organizations. Environment has state and behaviour.

[1] A conceptual framework is a set of terms of a domain and its signify, which serves as a guide to standardizing the concepts in this domain, and it can be used as the basis for defining a modelling language, for example.

Additionally, Silva et al. (2003) define the following relationships in TAO: Inhabit, Ownership, Play, Specialization/Inheritance, Control, Dependency, Association and Aggregation/Composition. These relationships will not be described here since the extension proposed in this work does not involve relationships, but it is described in Silva et al. (2003).

The concepts are presented by Silva et al. (2003) in a semi-formal approach that uses templates to formalize the Objects, Agents, Organization, Object Role, Agent Role, Environment and its relationships. An agent template and agent role template are presented in the following:

Agent
Agent_Class Agent_Class_Name *Goals setOf{Goal_Name}* *Beliefs setOf{Belief_Name}* *Actions setOf{Action_Name}* *Plans setOf{Plan_Name}* *Events generated: setOf{Event_Name},* *perceived: setOf{Event_Name}* *Roles setOf{Role_Class_Name}* *Relationships setOf{R_Name}* *end Agent_Class*

The agent template presented above is composed of its name, goals, beliefs, actions, plans, events generated and perceived, beyond roles associated and relationships. The agent role presented below is composed of goals, beliefs, duties, rights, protocols, commitments and relationships.

Model_based_reflex_agent_role
Agent_Role_Class Agent_Role_Class_Name *Goals setOf{Goal_Name}* *Beliefs setOf{Belief_Name}* *Duties setOf{Action_Name}* *Rights setOf{Permission_Name}U setOf{Action_Name}* *Protocols setOf{Interaction_Class_Name}U setOf{Rule_Name}* *Commitments setOf{Action_Name}* *Relationships setOf{Relationship_Name}* *end Agent_Role_Class*

2.3. MAS-ML

MAS-ML is a modelling language that implements the TAO concepts, and it was originally designed to support the modelling of proactive agents that are goal-based guided by pre-established plans.

MAS-ML models all structural and dynamic aspects defined in the TAO metamodel by extending the UML metamodel. The structural diagrams defined by MAS-ML are Role Diagram, Class Diagram and Organization Diagram (Silva et al., 2004). The dynamic diagrams are Sequence Diagram and Activities Diagram (Silva et al., 2008b). Using all these diagrams, it is possible to model all structural and dynamic aspects of entities defined in TAO.

2.3.1. Static diagrams

The static diagrams proposed by MAS-ML are three extensions of the UML Class Diagram. The MAS-ML Class Diagram can represent class (entity of the original UML Class Diagram) and UML Class Diagram relationships (association, aggregation and specialization). Furthermore, the specific entities and relationships were included: Agent, Environment and Organization entities and inhabit relationship; MAS-ML Organization Diagram can represent Organizations, Sub-organizations, Classes, Agents, Agent Roles, Object Roles and Environment and ownership, play and inhabit relationships. An MAS-ML Role Diagram can represent Agent Role and Object Role and its relationships Control, Dependency, Association, Aggregation and Specialization.

The entities added to static diagrams are shown in Fig. 2. Fig. 2(A) shows the agent element used in MAS-ML static diagrams. It is a rectangle with rounded edges with three compartments: the upper has the agent name; the middle has goals and beliefs; and the lower has plans and actions. Fig. 2(B) shows the representation to agent role. It is a rectangle with edge curved down and with three compartments: the upper has the instance name; the centre has goals and beliefs; and the lower has duties, rights and details of communication protocol. Fig. 2(C) shows the representation of Organization. It is a union of a rectangle and an ellipse with three compartments: the upper has the instance name; the middle has goals, beliefs and axioms; and the lower has plans and actions. Fig. 2(D) shows the representation of Object Role. It is a rectangle with a cut on the top left and with three compartments similar to the UML class representation.

Fig. 2(E) shows the representation of environment. The upper compartment has the instance name, the middle compartment has properties and the lower compartment has methods (Operations). A fourth compartment was added to represent the elements that inhabit the environment.

MAS-ML represents four relationships in static diagrams: Inhabit, Ownership, Play, Control and Dependency. The inhabit relationship can be observed in Fig. 2(E): the compartment further down is the inhabit relationship.

Fig. 3(A) shows the ownership relationship. It is used to link an organization and the roles that can be played in the organization (Agent Role or Object Role) and is represented by a double line. Fig. 3(C) shows the Play relationship. It is a ternary relationship linking an agent, a role and an organization. It means that the agent can play such a role in the organization. This relationship is represented by a line connecting an agent to an ownership relationship. Fig. 3(B) shows the Control relationship. The relationship defines that an agent role controls another agent role. It models a social relationship between agent roles such as a master-slave. A single line with a circle represents this relationship; the circle indicates the controller.

2.3.2. Dynamic diagrams

The dynamic diagrams defined in MAS-ML are extended versions of the UML Sequence Diagram and Activities Diagram (Silva et al., 2008b). The entities modelled in a Sequence Diagram are shown in Fig. 4, where Object (Fig. 4(A)), Agent (Fig. 4(B)), Organization (Fig. 4(C)) and Environment (Fig. 4(D)) are represented. A Sequence Diagram identifies the entities (e.g., the agent or the organization), the roles that they are playing (if it is the case), the organizations in which the roles are being played – it is important since entities can play different roles in different organizations (Odell, Parunak and Bauer, 2003) – and also the environments in which they are immersed (D'Inverno and Luck, 2001). Representing the environment is particularly interesting since it can show the system distribution.

The relationships created by MAS-ML to the UML Sequence Diagram are shown in Fig. 5: the last two are related to agents and others are related to agents playing roles. The stereotypes related to role-activate, role-deactivate, role-commitment and role-cancel can be used to connect agents to agent roles.

The MAS-ML Activity Diagram is capable of representing the agent behaviour through UML Activity Diagram elements. Additionally, the stereotypes plan, belief, message, organization and environment can be used to represent its respective elements. Swim lanes represent agents, environments and roles. Fig. 6 shows an example of an Activity Diagram of MAS-ML.

3. Related work

This section involves works related to (i) conceptual frameworks and (ii) modelling languages, both in an MAS context and

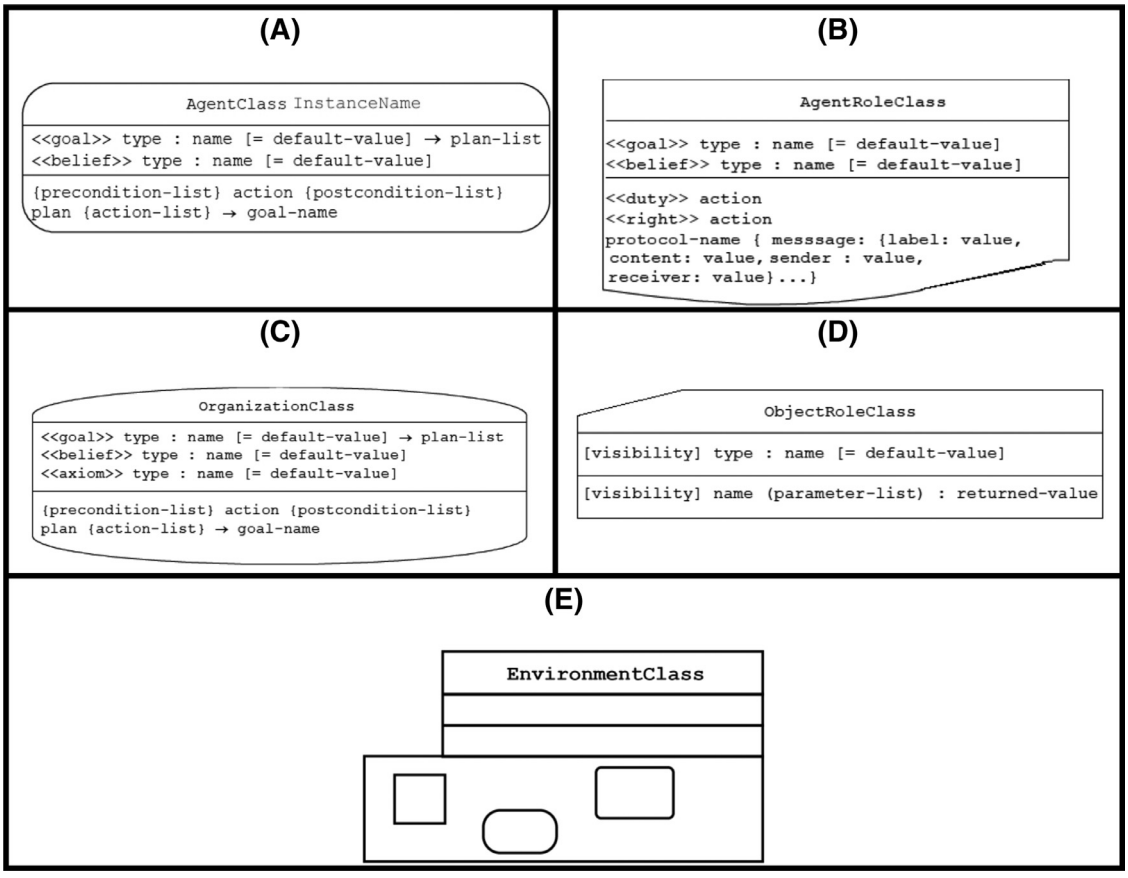

Fig. 2. MAS-ML entities representation in static diagrams (Silva, 2004).

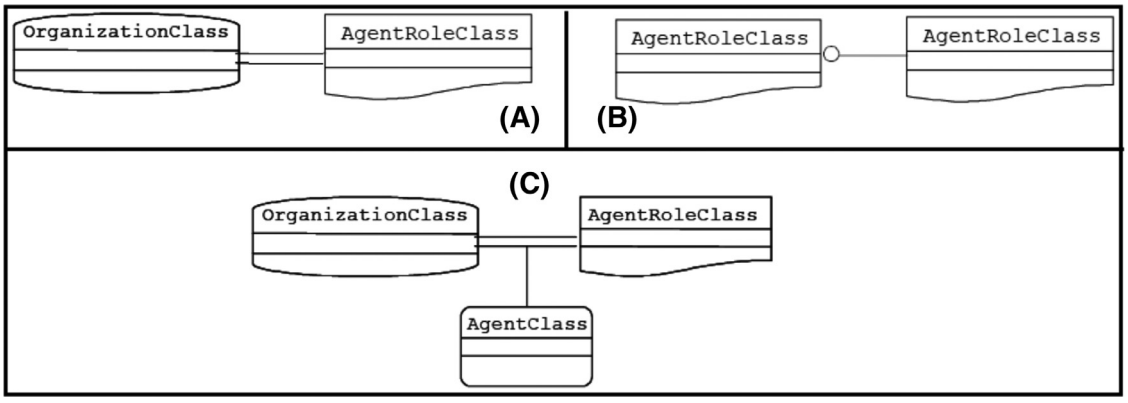

Fig. 3. MAS-ML static relationship (Silva, 2004).

considering systems that involve agents with different internal architecture presented in Section 2.1.

3.1. Conceptual frameworks

Some conceptual frameworks have been proposed for MAS. However, they provide limited support to internal architectures. Our aim is to analyse three conceptual frameworks considering the support provided to the modelling of the typical entities of the MAS along with their properties and their relationships. Also, the support given to the modelling of the internal agent architectures will be analysed.

The framework of D'Inverno and Luck (2001) defines a hierarchy composed of four layers having entities, objects, agents and autonomous agents. However, it presents the following limitations: (i) the environment entity has only structural features without transactions, (ii) no dynamic aspect associated with the proposed entities is defined, and (iii) it does not provide elements to define the different internal agent architectures.

Yu and Schmid (2001) propose a conceptual framework for the definition of role-based agent-oriented MAS. Agents are shown as an entity playing roles within any organization. As weak points we highlight the following aspects: (i) although agents are defined as an entity playing roles, this conceptual framework does not define the agent properties and the relationships between agents and roles; (ii) although the authors declare that roles are played in organizations, the proposal does not define the organization properties and the

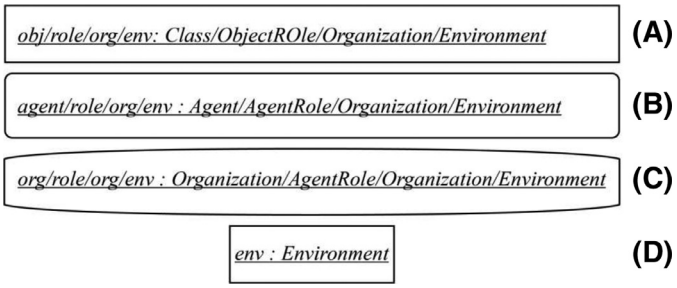

Fig. 4. Entities of Sequence Diagram (Silva, 2004).

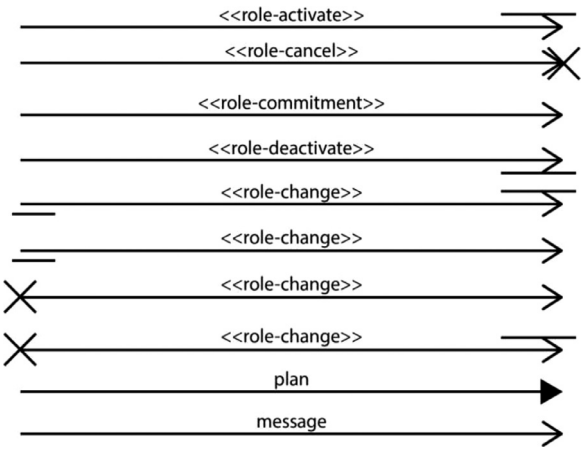

Fig. 5. Relationships of Sequence Diagram.

relationships among them and roles; (iii) neither does it define the environment entity that contains agents and organizations; and (iv) it only restricts the agent's behaviour in the context of a role.

Centeno et al. (2008) define a conceptual framework for organizational mechanisms in MASs. They use agents, environment and organization to define the organizational concepts related to its organization and operation. They argue that organizational mechanisms can be classified into two basic types: (i) those that provide additional information about the environment (its state, possible actions, their expected outcomes, etc.), and (ii) those that manipulate the environment proper. We argue that a key characteristic for an MAS to be "organization based" is to make use of a least one such mechanism.

Shirazi and Barfouroush (2008) define a conceptual framework for modelling and developing automated negotiation systems. This framework represents and specifies all the necessary concepts and entities for developing a negotiation system as well as the relationships among these concepts. This framework can also be used to model human negotiation scenarios for analysing these types of negotiations and simulating them with multi-agent systems.

The work of Lavinal et al. (2006) presents a generic agent-based framework as a first step towards the conception of self-managed systems. This conceptual framework consists of groups of management agents coupled with managed resources and endowed with specific management skills.

Beydoun et al. (2009) introduce a relatively generic agent-oriented metamodel whose suitability for supporting modelling language development is demonstrated by evaluating it with respect to several existing methodology-specific metamodels. First, the metamodel is constructed by a combination of bottom-up and top-down analysis and best practice. The concepts thus obtained and their relationships are then evaluated by mapping to two agent-oriented metamodels: TAO and Islander.

Moreover, the conceptual frameworks presented in this section do not present a description of internal architecture of agents, their internal elements and the respective roles.

3.2. Modelling languages

Several languages have been proposed for the modelling of MAS. However, they do not support the modelling of different internal architectures of agents available in Russell and Norvig (2003) and Weiss (1999). Besides, they have several drawbacks that have justified the choosing of MAS-ML to be extended in order to model different agent architectures.

The work of Odell et al. (2000) presents the AUML language. This modelling language aims to provide semiformal and intuitive semantics through a friendly graphical notation. AUML does not provide elements to represent the next-function, planning, formulate problem function, formulate goal function and utility function.

Wagner (2003) proposes the AORML modelling language, which is based on the AOR metamodel. This language does not give support to the modelling of the elements of the internal agent architectures. Therefore, it is not possible to differentiate agents with reactive and proactive architectures in AORML.

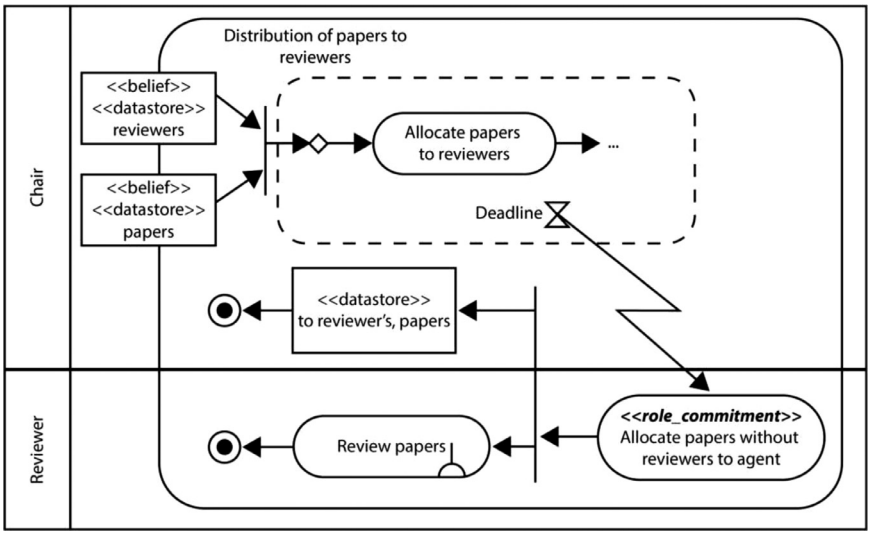

Fig. 6. An example of MAS-ML Activity Diagram (Silva et al., 2005).

Table 1
New TAO agent features.

Internal architecture	Structural features	Behavioural features
Simple Reflex Agent	–	Perception and action (oriented by condition-action rules)
Model-Based Reflex Agent	Belief	Perception, next function and action (oriented by condition-action rules)
MAS-ML Agent	Goal and belief	Plan and action (oriented by the plan chosen according to the goal)
Goal-Based Agent	Goal and belief	Perception, next function, goal-formulation function, problem-formulation function, planning and action
Utility-Based Agent	Goal and belief	Perception, next function, goal-formulation function, problem-formulation function, utility function planning and action

Moreover, the two languages mentioned above do not define the environment as an abstraction, so it is not possible to model the agent migration from one environment to another. This capability is inherent in mobile agents modelling (Silva and Mendes, 2003).

Choren and Lucena (2005) present the ANote modelling language that involves a set of models called views. In ANote, it is not possible to differentiate agents with reactive architectures from the proactive ones. In addition, ANote does not support conventional objects used to model non-autonomous entities. The language defines several concepts related to agents, but the concept of agent role is not specified. This concept is extremely important when modelling societies where agents can play different roles at the same time.

The Agent Modelling Language (AMOLA) (Spanoudakis and Moraitis, 2008) provides the syntax and semantics for creating models of multi-agent systems covering the analysis and design phases of the software development process. It supports a modular agent design approach and introduces the concepts of intra- and inter-agent control that is based on statecharts. AMOLA deals with both the individual and societal aspects of the agents. Since AMOLA does not model perceptions, planning, next-function, formulate-problem-function and formulate goal function elements, the internal architecture that uses this elements cannot be represented.

AML (Cervenka, 2012) is a modelling language based on a metamodel that enables the modelling of organizational units, social relations, roles and role properties. AML agents are composed of attribute list, operation list, parts and behaviours, and sensors and actuators can be modelled near the capabilities. Planning, next function, formulate problem function, formulate goal function and utility function elements are not semantically represented, therefore the internal architecture that uses this elements is not represented. It is worth mentioning that the semantic aspects of the communication are modelled as specializations of existing elements in UML, such as methods invocation, which is not adequate for modelling agent communication, for example.

4. Extending the TAO framework

According to Silva et al. (2003), TAO describes an agent as an autonomous, adaptive and interactive element that has as structural features its goals and beliefs, and as behavioural features its actions and plans. The structural features of an agent are the ones used to store information about other agents and the environment, about the states the agents want to achieve, and any other useful information. Behavioural features are the ones related to the agent execution such as the tasks the agent are able to execute and the mechanism used by the agent to select these tasks.

TAO extension was based on the inclusion of concepts used to define the internal architectures of agents (Section 2.1). Because of the internal architecture chosen when developing an agent, some of the structural and behavioural features previously defined in TAO to all agents cannot be modelled or explicitly defined. In this context, a new definition for *agent* was formulated as follows:

An agent is an autonomous, adaptive and interactive element. Its behavioural and structural features are predefined by its internal architecture as follows:

- Simple Reflex Agents have no structural features. Their behavioural features are characterized by their perceptions and actions. The actions to be executed are chosen by condition-action rules.
- Model-Based Reflex Agents have beliefs representing their structural features. Their behavioural features are composed of perceptions, actions (oriented by condition-action rules) and a next-function.
- MAS-ML agents have goals and beliefs as structural features and have plans and actions as their behavioural features. Plans are executed to achieve goals and are composed of actions.
- Goal-Based Agents have goals and beliefs as structural features and have perceptions, a next function, a goal-formulation function, a problem-formulation function, a planning function and actions as their behavioural features.
- Utility-Based Agents have goals and beliefs as structural features and have perceptions, a next function, a goal-formulation function, a problem-formulation function, a utility function planning and actions as their behavioural features. Table 1 lists the structural features and behavioural features related to each internal agent architecture.

It is still necessary to change the concept of the agent role with respect to the reactive agents. The structural features of the agent role in TAO are defined as being composed of beliefs and goals (Silva, 2004). We can add to this concept the fact that in the case of Simple Reflex Agents, beliefs and goals do not exist, and in the case of reactive agents based on knowledge, beliefs only exist as a structural feature.

Besides the conceptual approach, Silva et al. (2003) present the concepts in a semiformal approach that uses templates to help to describe the concepts. We use the same notation presented by Silva et al. (2003) to represent the agent concept for each, different agent architecture and the agent role for Simple Reflex Agents and Model-Based Reflex Agents. It is shown because the initial representation of MAS-ML to Agent and Agent Role, presented in Section 2.2, is not enough to represent all agent architectures.

The first template defines the Simple Reflex Agent class. The Simple Reflex agent class describes the perceptions, actions and relationships that are the same for all its agent instances. The agent template also lists the events that agents, instances of the SimpleReflexAgent class, can generate and perceive, and the roles that agents could play.

SimpleReflexAgent
Agent_Class Agent_Class_Name *Perceives setOf{Perceives_Name}* *Actions setOf{Action_Name}* *Events generated: setOf{Event_Name},* *perceived: setOf{Event_Name}* *Roles setOf{Role_Class_Name}* *Relationships setOf{R_Name}* *end Agent_Class*

The Model-Based Reflex Agent class describes beliefs, perceptions, next-function, actions and relationships that are the same for all its

agent instances. The agent template also lists the events that agents, instances of the ModelBasedReflexAgent class, can generate and perceive, and the roles that agents could play.

ModelBasedReflexAgent
Agent_Class Agent_Class_Name *Beliefs setOf{Belief_Name}* *Perceives setOf{Perceives_Name}* *NextFunction setOf{NF_Name}* *Actions setOf{Action_Name}* *Events generated: setOf{Event_Name},* *perceived: setOf{Event_Name}* *Roles setOf{Role_Class_Name}* *Relationships setOf{R_Name}* *end Agent_Class*

The Goal-Based Agent class describes goals, beliefs, perceptions, actions, next-function, formulate-goal-function, formulate-problem-function and relationships that are the same for all its agent instances. The agent template also lists the events that agents, instances of the GoalBasedAgent class, can generate and perceive, and the roles that agents could play.

GoalBasedAgent
Agent_Class Agent_Class_Name *Goals setOf{Goal_Name}* *Beliefs setOf{Belief_Name}* *Perceives setOf{Perceives_Name}* *Actions setOf{Action_Name}* *Planning setOf{Planning_Name}* *NextFunction setOf{Function_Name}* *FormulateGoalFunction setOf{GFName}* *FormulateProblemFunction setOf{PFName}* *Events generated: setOf{Event_Name},* *perceived: setOf{Event_Name}* *Roles setOf{Role_Class_Name}* *Relationships setOf{R_Name}* *end Agent_Class*

The Utility-Based Agent class describes goals, beliefs, perceptions, actions, next-function, formulate-goal-function, formulate-problem-function, utility-function and relationships that are the same for all its agent instances. The agent template also lists the events that agents, instances of the UtilityBasedAgent class, can generate and perceive, and the roles that agents could play.

UtilityBasedAgent
Agent_Class Agent_Class_Name *Goals setOf{Goal_Name}* *Beliefs setOf{Belief_Name}* *Perceives setOf{Perceives_Name}* *Actions setOf{Action_Name}* *Planning setof{Planning_Name}* *NextFunction setOf{Function_Name}* *FormulateGoalFunction setOf{GFName}* *FormulateProblemFunction setOf{PFName}* *UtilityFunction setOf{UF_Name}* *Events generated: setOf{Event_Name},* *perceived: setOf{Event_Name}* *Roles setOf{Role_Class_Name}* *Relationships setOf{R_Name}* *end Agent_Class*

The initial representation of the agent role class was not changed and this representation is used for all proactive agents. The templates used to represent the agent roles for new agents' representation are the following.

The Simple Reflex Agent role template presents the Simple Reflex Agent role class and the duties, rights, protocols and commitments that define the interactions. It also identifies the relationships of the agent roles, that is, its owner, the agents and organizations that may play the role, the objects associated with the role, and the associations with other roles. All role instances of the role class have the same properties and relationships.

Simple_Reflex_Agent_Role
Agent_Role_Class Agent_Role_Class_Name *Duties setOf{Action_Name}* *Rights setOf{Permission_Name}U setOf{Action_Name}* *Protocols setOf{Interaction_Class_Name}U setOf{Rule_Name}* *Commitments setOf{Action_Name}* *Relationships setOf{Relationship_Name}* *end Agent_Role_Class*

The Model-Based Reflex Agent Role template presents the Model-Based Agent Role class and the beliefs, duties, rights, protocols and commitments that define the interactions. It also identifies the relationships of the agent roles, that is, its owner, the agents and organizations that may play the role, the objects associated with the role, and the associations between the roles. All role instances of the role class have the same properties and relationships.

Model_based_reflex_agent_role
Agent_Role_Class Agent_Role_Class_Name *Beliefs setOf{Belief_Name}* *Duties setOf{Action_Name}* *Rights setOf{Permission_Name}U setOf{Action_Name}* *Protocols setOf{Interaction_Class_Name}U setOf{Rule_Name}* *Commitments setOf{Action_Name}* *Relationships setOf{Relationship_Name}* *end Agent_Role_Class*

The Goal-Based Agents and utility agents use the same notation presented initially by Silva et al. (2003) and presented in Section 2.2.

5. Extending the MAS-ML language

This section presents the extensions to MAS-ML in order to support the modelling of agents by using different internal architectures: Simple Reflex, Model-Based Reflex, Goal Based and Utility Based. The new version of MAS-ML is called MAS-ML 2.0.

According to UML (2009), tagged values, stereotypes and constraints are extension mechanisms. Additionally, adaptation of existing metaclasses and definition of new metaclasses can also be used. Stereotypes and definition of new metaclasses were used to represent Simple Reflex Agents, Model-Based Reflex Agents, Goal-Based Agents with planning and Utility-Based Agents. By following the architecture definitions presented in Section 2, we felt the need to define the following characteristics: perception, next-function, formulate-goal-function, formulate-problem-function, planning and utility-function.

Fig. 7 illustrates the MAS-ML 2.0 metamodel and highlights by using white double-line rectangles the extensions made to the MAS-ML metamodel. A subset of UML metaclasses is drawn as white single-line rectangles, MAS-ML metaclasses are represented as grey single-line rectangles, and MAS-ML stereotypes are represented with a grey rectangle with rounded edges.

The perceptions of an agent get information about the environment and/or other agents. Since there is not any metaclass in MAS-ML that can be used to represent such a concept, the *AgentPerceptionFunction* metaclass was created to represent the agent perception. *AgentPerceptionFunction* metaclass is related with the *Environment* metaclass because the agent perceives the environment, it is also related with *Constraint* metaclass to restrict the information that can be perceived through the agent sensors.

The planning task results in a sequence of actions in order to achieve a goal (Russell and Norvig, 2003). In addition, the following properties are observed: (i) unlike a plan (represented by *AgentPlan* in the original MAS-ML metamodel), the sequence of actions is created at runtime; and (ii) unlike a simple action (represented by *AgentAction* in the MAS-ML metamodel), the action of planning has a

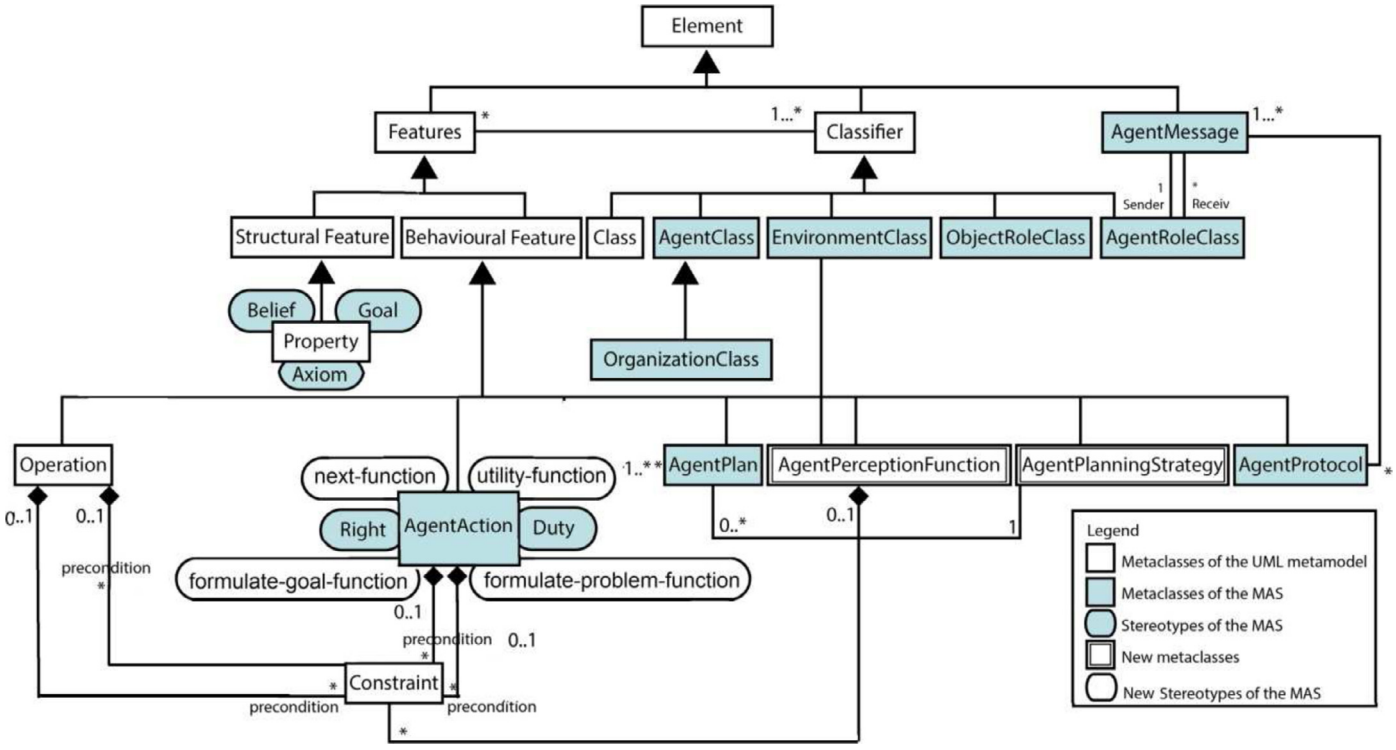

Fig. 7. MAS-ML metamodel extension.

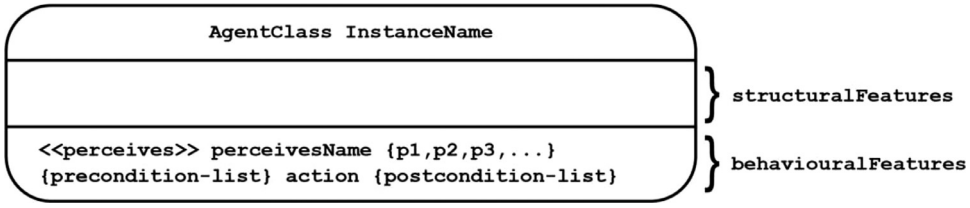

Fig. 8. Simple Reflex Agent representation in static diagrams.

goal associated with it. Thus, the new metaclass *AgentPlanningStrategy* was created to represent the planning functionality. An association relationship between *AgentPlanningStrategy* and *AgentPlan* was defined to represent the action of creating plans. The metaclasses *AgentPerceptionFunction* and *AgentPlanningStrategy* extend the *BehaviouralFeature* metaclass.

The next-function, formulate-goal-function, formulate-problem-function and utility-function are special agent actions that depend on the agent's internal architecture. The *<<next-function>>*, *<<formulate-goal-function>>*, *<<formulate-problem-function>>* and *<<utility-function>>* stereotypes were thus created and related to *AgentAction* metaclass. Finally, the condition action rules used by Simple Reflex Agents Model-Based Reflex Agents can be represented by using the agent's action representation (Silva et al., 2008a), therefore, is not explicitly represented in the metamodel.

5.1. Static representation of AgentClass

The new structural and behavioural features introduced in the MAS-ML metamodel are used to model the different types of agents and it impact the *AgentClass* metaclass representation in static diagrams. In the next sections, the new representation is shown. Each agent of this section was created by using the default name: *AgentClass InstanceName*. This name can be replaced when the agents of the application domain are being defined (Section 7.2.1 describes agents that use the generic notation presented in following sections).

5.1.1. Simple Reflex Agent structure

The representation for a Simple Reflex Agent (Fig. 8) does not include any structural element in middle compartment, but in the lower compartment the perceptions and actions, driven by condition-action rules and not by a specific plan, are represented.

5.1.2. Model-Based Reflex Agent structure

The Model-Based Reflex Agent represents an upgrade over the Simple Reflex Agent. Thus, the definition for the action element is kept the same. In addition, beliefs representing the state and the next function are included. Fig. 9 presents the graphical representation of *AgentClass* for a Model-Based Reflex Agent.

5.1.3. Goal-Based Agent with plan structure

The Goal-Based Agents with plan have the same structure initially proposed by Silva and Lucena (2004) including goals, beliefs, actions and plan. Fig. 2(A) shows the graphical representation of this agent.

5.1.4. Goal-Based Agent with planning structure

The Goal-Based Agents with planning incorporate additional complexity in the agent representation. Firstly, goals are considered in order to guide the agent behaviour. In order to consistently manipulate goals and states, the agent behaviour is enhanced with *<<perceives>>*, *<<formulate-goal-function>>* and *<<formulate-problem-function>>* elements. The already existent

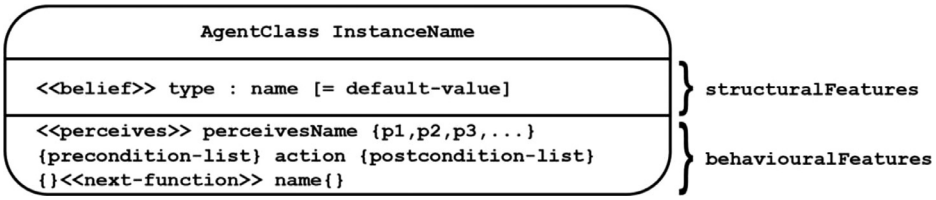

Fig. 9. Model Reflex Agent representation in static diagrams.

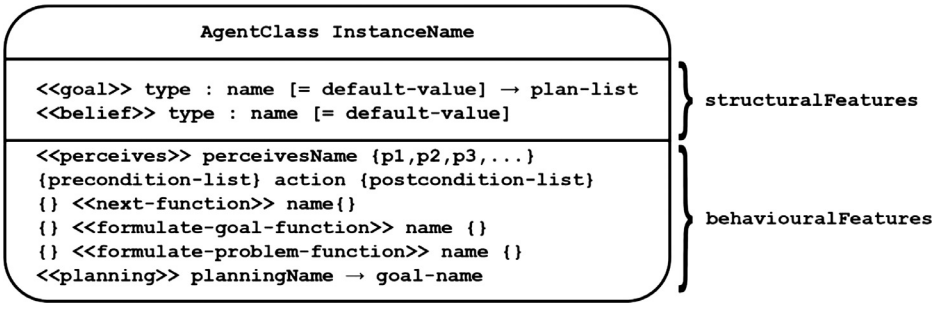

Fig. 10. Goal-Based Agent representation in static diagrams.

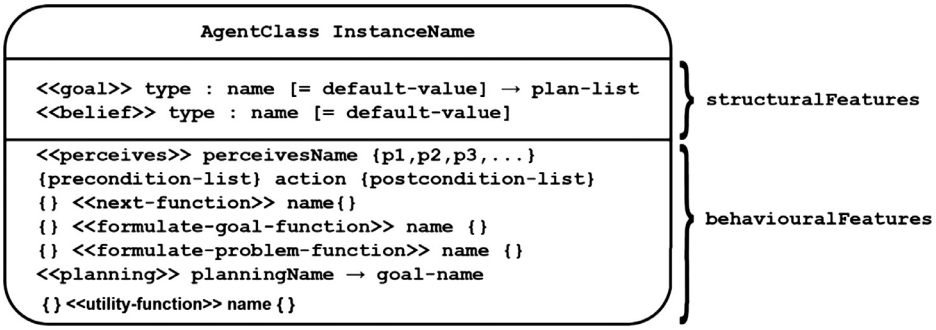

Fig. 11. Utility-Based Agent representation in static diagrams.

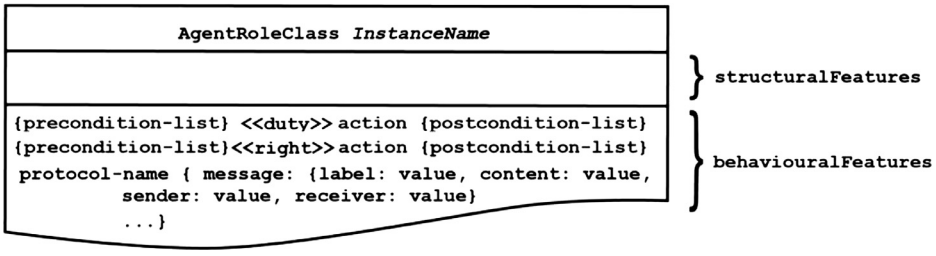

Fig. 12. Agent Role Class to Simple Reflex Agent Role representation in static diagrams.

<<*next-function*>> element is kept up. This function receives the current perception and the beliefs that must be updated (state).

In addition, instead of representing pre-established plans, the planning activity is incorporated. This activity involves a goal and uses the available actions to create a sequence of actions. Fig. 10 illustrates the *AgentClass* for a Goal-Based Agent using planning.

5.1.5. Utility-Based Agent structure

The representation for the Utility-Based Agent consists of a specialization of the Goal-Based Agent with planning. During the planning, the agents may be linked to reach more than one goal. In this case, the occurrence of conflicting goals or the existence of several states meeting the goals is possible. So, the utility function is incorporated into the agent structure in order to evaluate the usefulness degree of the associated goals. Thus, the <<*utility-function*>> element is added to represent the function responsible for the optimization of the agent performance. The graphical representation of *AgentClass* for a proactive agent based on utility is illustrated in Fig. 11.

5.2. AgentRoleClass static representation

An *AgentRoleClass* in MAS-ML is represented by a solid rectangle with a curve at the bottom. Similar to the class representation, it has three compartments separated by horizontal lines. The upper compartment contains the agent role name unique in its namespace. The intermediate compartment contains a list of goals and beliefs associated with the role, and below, a list of duties, rights and protocols.

Reactive agents do not have explicit goals and, more particularly, the Simple Reflex Agents do not have beliefs. Thus, their role representation must be adapted, and its representation is illustrated in Fig. 12.

In addition to the representation of the roles for Simple Reflex Agents, roles for Model-Based Reflex Agents include beliefs in order

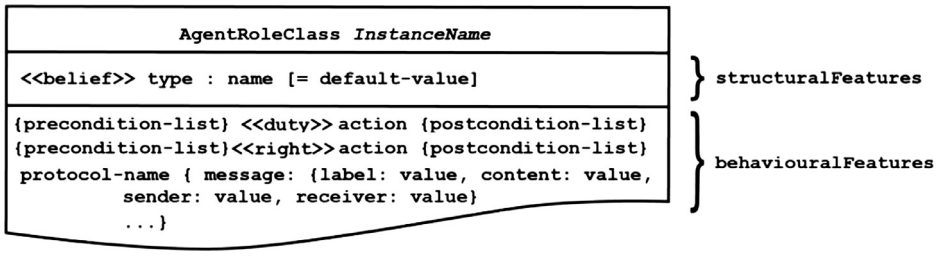

Fig. 13. Agent Role Class to Model-Based Reflex Agent Role representation in static diagrams.

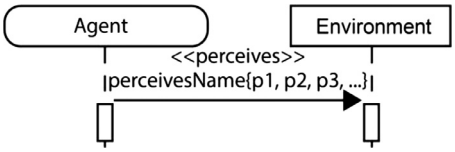

Fig. 14. Perception of the agent in Sequence Diagram.

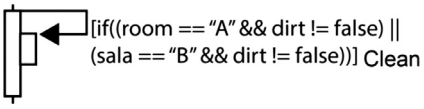

Fig. 15. Reactive agent action in the Sequence Diagram.

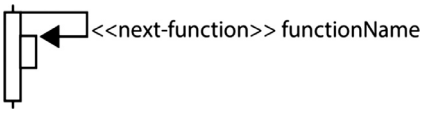

Fig. 16. Sequence Diagram of the next function.

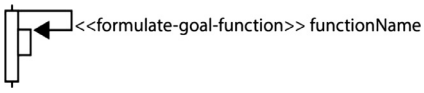

Fig. 17. Formulate goal function in Sequence Diagram.

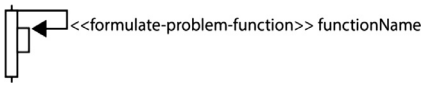

Fig. 18. Formulate-problem function in Sequence Diagram.

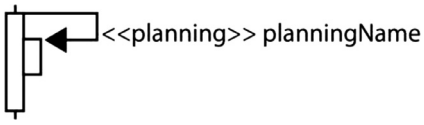

Fig. 19. Planning in the Sequence Diagram.

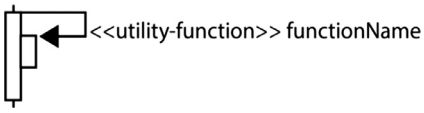

Fig. 20. Utility-function in the Sequence Diagram.

to partially handle observable environments. The agent role representation in this case is illustrated in Fig. 13.

The features of agent roles in other architectures are unchanged since both define beliefs and goals. The structural changes regarding the *AgentRoleClass* entity impact the Organization and Roles diagrams.

5.3. Representation of AgentClass in Sequence Diagram

Similar to the static diagrams, the new definitions of the *AgentClass* influence new representations of their behavioural features. In Section 5.3.1 we introduce the representations of the new elements that have been defined in order to be possible to model the execution of the different internal agent architectures, modelled in Sections 5.3.2–5.3.5.

5.3.1. Introducing the new elements in Sequence Diagram

The agent's perception is represented in the MAS-ML Sequence Diagram by an arrow with an open head leaving the agent to the environment, together with the *<<perceives>>* stereotype, the perception name and the elements that the agent perceives (Fig. 14).

In order to represent the actions executed by a reactive agent, it is necessary to represent its associated conditions. Fig. 15 illustrates an action that a reactive agent can execute.

The next function of reactive agents is represented in the Sequence Diagram of MAS-ML 2.0 by a closed arrow with full head, which starts at the agent and ends at the agent. The stereotype *<<next-function>>* is used followed by the name of the function. Fig. 16 illustrates the next function in the Sequence Diagram.

Therefore, if a Simple Reflex Agent is modelled, we have first its perception and then its actions guided by the condition-action rules. In the case of a Model-Based Reflex Agent, we have first the perception, then the next function and, finally, its actions guided by the condition-action rules.

The next function of proactive agents is executed before the formulate-goal function and is used by two types of proactive agents in this paper: formulate-goal function and problem function, as illustrated in Figs. 17 and 18.

In case of agents able to plan at runtime, the sequence of actions that the agent will execute cannot be modelled at design time. In this case, planning is represented by a closed arrowhead that begins and ends in the agent adorned with the stereotype *<<planning>>*. The actions that can be used for planning and achieve the objective(s) are represented as in Silva and Lucena (2004). Optionally, a textual note can specify the criterion or algorithm used to perform the planning. Fig. 19 illustrates the planning in the MAS-ML 2.0 Sequence Diagram.

The utility function element is represented in the Sequence Diagram by an arrow with full head that begins in the agent and ends in itself, together with the stereotype *<<utility-function>>*. Fig. 20 illustrates the representation of the utility function in the MAS-ML 2.0 Sequence Diagram.

In MAS-ML 2.0 the agent actions are modelled by using the iteration element and combined fragment, which are already defined in UML. This representation allows the modelling of any combination of actions. An example is shown in Fig. 21. Since the sequence of actions for the agents with planning is generated at runtime, the modelling of this sequence is not required.

The Goal-Based Agent with planning uses the representation proposed by Silva and Lucena (2004), as well as the plan defined during the design phase. In the case of Goal-Based Agent with planning, it

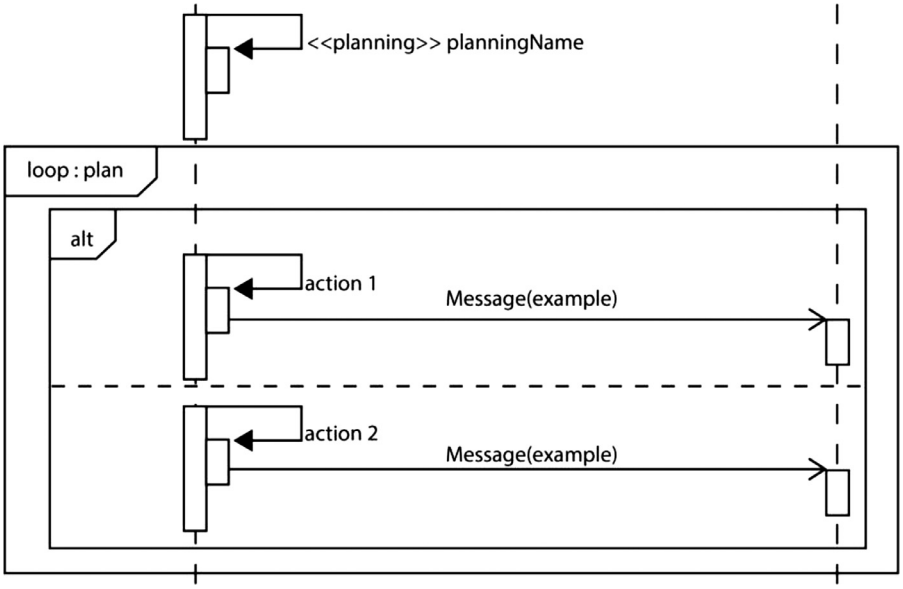

Fig. 21. Implementation of the actions of the agent with planning in the Sequence Diagram of MAS-ML 2.0.

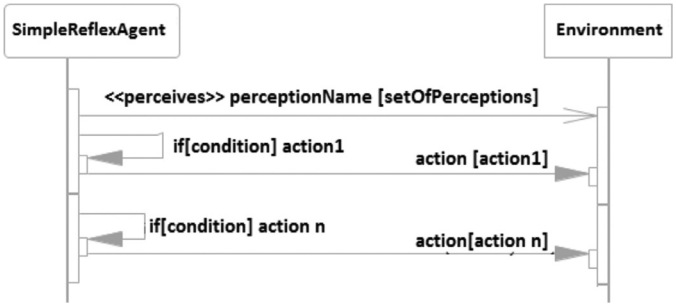

Fig. 22. Simple Reflex Agent in Sequence Diagram.

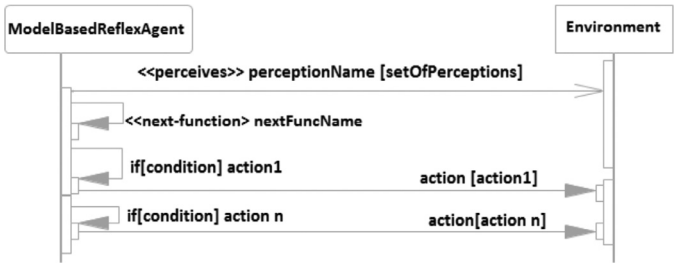

Fig. 23. Model-Based Reflex Agent in Sequence Diagram.

initially runs perception, then next function, formulate-goal function and formulate-problem function. The planning function is executed and its output is a sequence of possible actions.

Finally, the Utility-Based Agent needs the perception, next function, formulate-goal function, formulate-problem function, planning, utility function and results in actions that are performed in the predefined order.

5.3.2. Representation of Simple Reflex Agent in Sequence Diagram

The Simple Reflex Agent starts by executing its perception followed by the actions executed according to the condition-action rules. Fig. 22 shows the Simple Reflex Agent in a Sequence Diagram. The agent perceives the environment and acts by executing an action that changes the environment or that sends a message to another agent.

5.3.3. Representation of Model-Based Reflex Agent in Sequence Diagram

The Model-Based Reflex Agent starts by executing its perception. Then, the next-function is executed followed by the actions executed according to the condition-action rules. Fig. 23 shows the Model-Based Reflex Agent in a Sequence Diagram. The agent perceives the environment and acts by executing an action that changes the environment or that sends a message to another agent.

5.3.4. Representation of Goal-Based Agent in Sequence Diagram

The Goal-Based Agent starts its execution by perceiving the environment. Then, the next-function is executed followed by the formulate-goal function and the formulate-problem function. The actions are then executed according to the selected goal and plan. Fig. 24 shows the Goal-Based Agent in a Sequence Diagram. The agent perceives the environment and acts by executing an action that changes the environment or that sends a message to another agent.

5.3.5. Representation of Utility-Based Agent in Sequence Diagram

The Utility-Based Agent starts its execution by perceiving the environment, executing the next-function and performing the formulate-goal function and formulate-problem function. The actions are executed according to the chosen plan that considers the related utility-function. Fig. 25 shows the Utility-Based Agent in a Sequence Diagram. As in the previous agents, the agent perceives the environment and acts by changing the environment or sending a message.

5.4. Representation of AgentClass in Activity Diagram

The features proposed by Silva et al. (2005) and used in the Activity Diagrams were reused. Thus, each activity is represented by a rounded rectangle. The agent beliefs are represented by a square with the identification of the agent used by the beliefs and goals in the upper-right corner through a textual description denoted by the *<<Goal>>* stereotype.

5.4.1. Elements representation of reactive agents in Activity Diagram

The Activity Diagram of simple and Model-Based Reflex Agents represents the behaviour from perception to action. The behaviour of a Simple Reflex Agent is represented as follows: the initial activity is

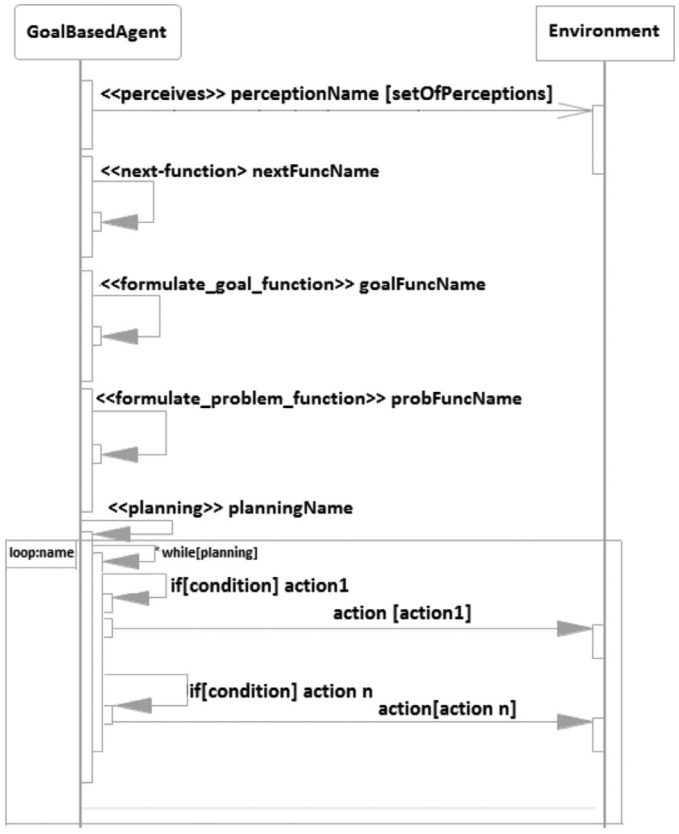

Fig. 24. Goal-Based Agent in Sequence Diagram.

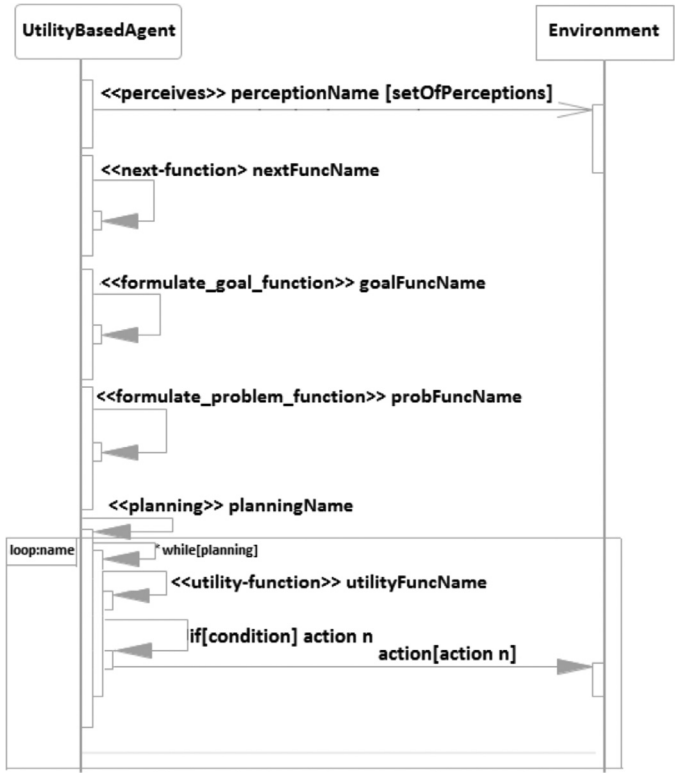

Fig. 25. Utility-Based Agent in Sequence Diagram.

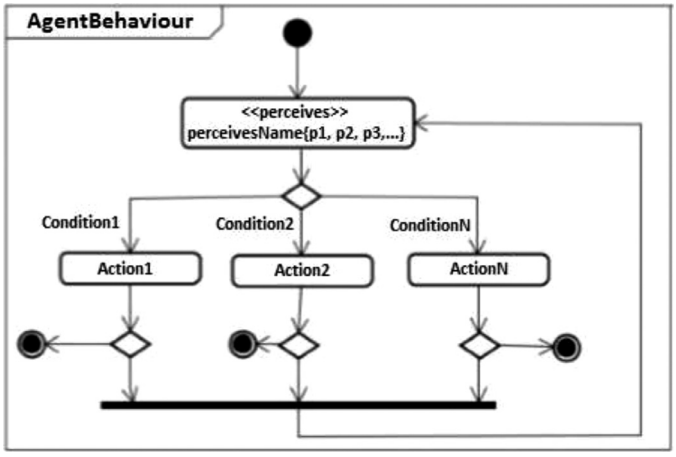

Fig. 26. Simple Reflex Agent in Activity Diagram.

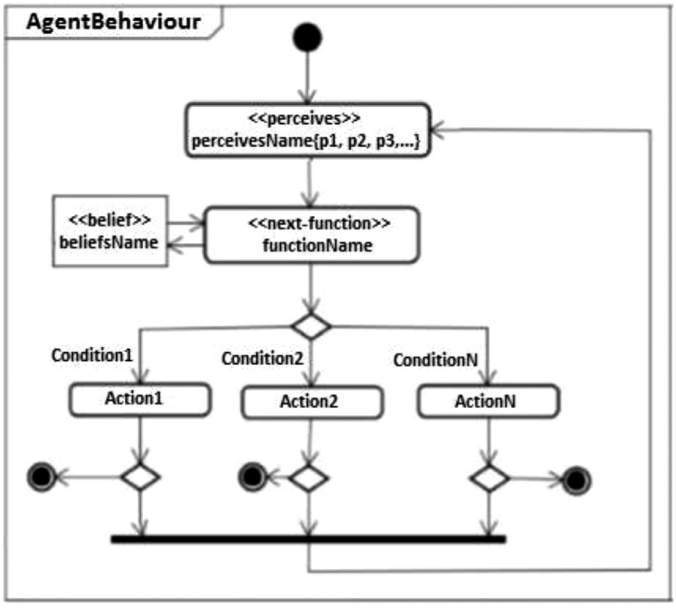

Fig. 27. Model-Based Reflex Agent in Activity Diagram.

the perception of the agent and, on the basis of the current perception, the condition action rules are used to select one of the possible actions. Finally, the selected action is performed. Fig. 26 shows the Simple Reflex Agent in Activity Diagram.

On the other hand, the behaviour of a Model-Based Reflex Agent is represented as follows: the initial activity is the perception of the agent that can be used by the next function to update its beliefs. After that, the condition-action rules are responsible for selecting one of the possible actions. Finally, the selected action is performed. Fig. 27 shows the Model-Based Reflex Agent in Activity Diagram.

5.4.2. *Elements representation of proactive agents in Activity Diagram*

The Activity Diagram of the Goal-Based Agent with planning represents the agent behaviour from perception to action. The behaviour of a Goal-Based Agent is represented on the Activity Diagram of MAS-ML 2.0 as follows: the initial activity is the perception of the agent and after that the next function updates the beliefs based on the current perception. The formulate goal and the formulate-problem functions are executed. The planning is performed to determine the action(s) that should be taken. Finally, the selected action(s) is performed. Fig. 28 shows the Goal-Based Agent in Activity Diagram.

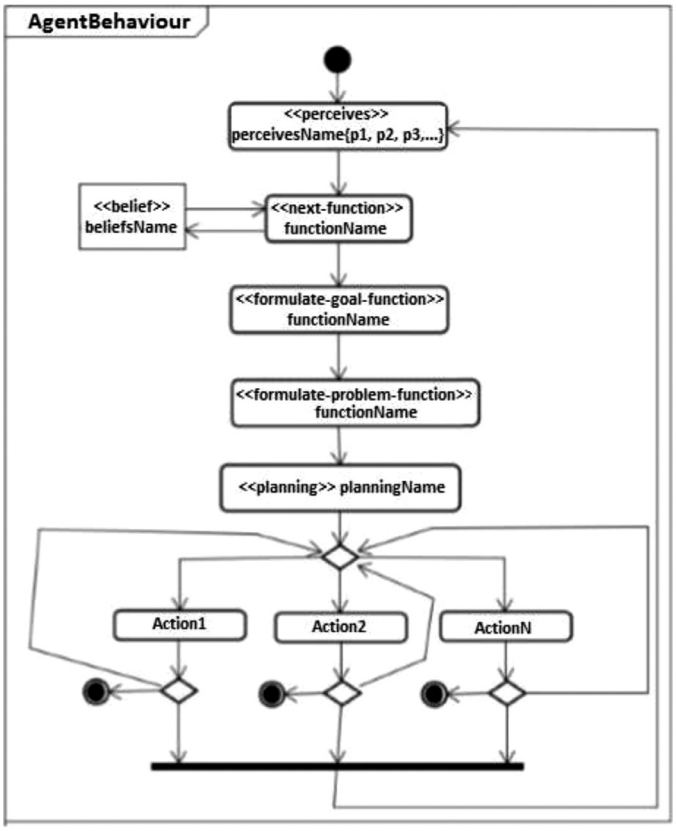

Fig. 28. Goal-Based Agent in Activity Diagram.

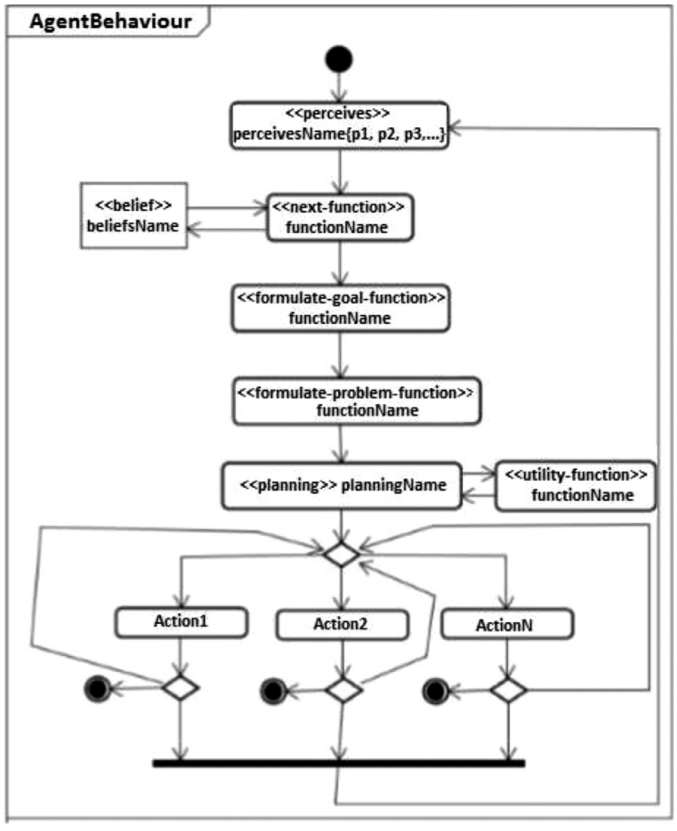

Fig. 29. Utility-Based Agent in Activity Diagram.

The behaviour of a Utility-Based Agent is represented on the Activity Diagram as follows: the initial activity is the perception, then the next function updates beliefs based on the current perception. The formulate-goal function and the formulate-problem function are executed. The planning is performed to determine what action(s) should be taken. The utility function helps in the choice of the action, and the selected actions are performed. Fig. 29 shows the Utility-Based Agent in Activity Diagram.

6. MAS-ML tool

This section presents the functions, technologies, existent modelling tools for MAS and details related to the development of the MAS modelling environment called MAS-ML tool. Initially, the process of tool generation is described. This is followed by a description of the development of each diagram with the respective elements represented. Finally, an overview of MAS-ML tool is shown.

6.1. Background of MDA

MDD (model-driven development) is characterized by a focus on modelling and not on implementation (France and Rumpe, 2007). This approach provides automation through the implementation of changes that may involve model-to-model or model-to-text.

Based on the MDD, the OMG (object management group) defines an architecture called MDA (Model-Driven Architecture) that aims to standardize and facilitate the integration of resources used. This architecture is based on a set of standards, such as Unified Modelling Language (UML) with MOF (Meta Object Facility), XMI (XML Metadata Interchange) and CWM (Common Warehouse Metamodel), that offers some core models or profiles for development (OMG, 2014a).

OMG also defines patterns for the code generation process, which utilizes a set of input instructions (these can be models) and gives as output the related source code. This standard establishes that the OMG concepts can be applied to different modelling languages, that is, they are not necessarily linked to a specific platform such as UML (OMG, 2014b).

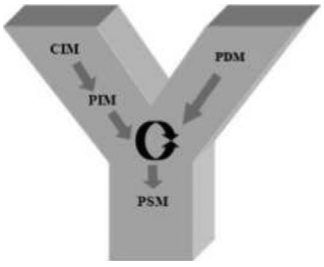

Fig. 30. MDA process. Available in http://research.petalslink.org/pages/viewpage.action?pageId=3639295.

The code generation process can be divided into stages. The concepts of each phase of code generation process are illustrated in Fig. 30 and hereafter described:

- CIM (Computation Independent Model): Focuses on the understanding of requirements to specify the application domain.
- PIM (Platform Independent Model): Defines system entities and their associated relationships.
- PSM (Platform Specific Model): The code generation will be held in a particular programming language and can use components, frameworks, middleware and libraries. Therefore, definitions need to be made in this regard.
- PDM (Platform Definition Model): Because the PSM is chosen, it is necessary to create relations between the elements present in the PIM and PSM.

Table 2
Validation rules description.

Rule	Purpose
Rule 1	If agent has plan then it has goal, belief and action.
Rule 2	If agent has plan then it does not have perception.
Rule 3	If agent has a goal, it has a plan or planning.
Rule 4	If agent has planning, it has belief, goal, perception and action.
Rule 5	If agent has plan, it has no planning.
Rule 6	If agent has planning, it has no plan.

6.2. Modelling tools for MAS

Based on the MDA approach, some modelling tools were proposed for modelling MAS. MAS-ML has a tool available; it was proposed by De Maria et al. (2005) for an MDA approach. However, this tool does not have available code; consequently, the application of extension proposed in this work and any changes cannot be applied in this tool. Cervenka (2012) describes profiles defined to IBM Rational Rose, Enterprise Architect and StarUML; all three tools provide the customizing mechanisms for defining and applying UML profiles in the application models; therefore, implementation of UML profiles for AML is straightforward. AUML cites a set of tools that can support AUML; however, a profile definition is necessary to introduce the AUML syntax in UML tools.

Moreover, the modelling tools presented in this section do not present a description of internal architecture of agents, their internal elements and the respective roles. MAS-ML tool can represent the internal architectures of Simple Reflex Agents, Model-Based Reflex Agents, Goal-Based Agents and utility agents.

6.3. Tool implementation

The MAS-ML tool is a modelling environment that supports the modelling of multi-agent systems based on the metamodel of MAS-ML. After defining the metamodel of the language, that is, the abstract syntax of the language, we felt the need for implementing a modelling tool to give support to the concrete syntax of the language. The tool was developed based on the model-driven approach proposed by the Graphical Modelling Framework (GMF, 2011). The development process proposed by the framework is divided into six steps (Domain Model, Graphical Definition Model, Tool Definition Model, Mapping Model and Domain Generation Model). These six steps were applied in the development of MAS-ML 2.0 tool, as follows:

Domain Model. First, the MAS-ML metamodel was specified using EMOF (*Essential Meta-Object Facility*), a metamodel definition language. The predefined stereotypes were added to *ActionClass* by ActionSemantics resource, as follows: 0 – without stereotype, 1 – next-function, 2 – utility-function, 3 – formulate-problem-function and 4 – formulate-goal-function.

Graphical Definition Model. In this step the entities, its properties, and relationships were defined by following the metamodel.

Tool Definition Model. The elements used in each palette are defined in this step. This step receives the domain model and definition model cited previously.

Mapping Model. In this step a mapping linking the domain models, graphical model and tool model is built. The generated mapping is used as input of the transformation process, which creates a specific modelling platform. The MAS-ML tool incorporates a model-checking mechanism in order to validate the construction of the diagrams. A set of six validation rules defined by using OCL (Object Constraint Language) (OCL, 2011) is used to check if the model was correctly formed (Table 2). Class, Organization, Role, Sequence and Activity Diagrams use these rules.

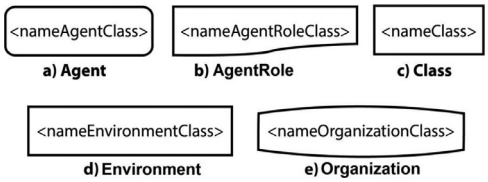

Fig. 31. The representation of MAS-ML entities in the Sequence Diagram of MAS-ML tool.

Tool Generation. The next step follows the generative approach proposed in Czarnecki and Eisenecker (2000) where the code is generated based on the model. Thus, GMF is used since it provides a generative component and a runtime infrastructure to develop graphical editors. The development process of each diagram is described below.

- *Class Diagram*. The Class Diagram of the MAS-ML tool was developed according to the steps listed in Section 6. Its Domain Model was created with the entities and relationships already defined in MAS-ML 2.0, and other steps were followed culminating in the tool generation. The Class Diagram contemplates the following elements: (i) Nodes: Class, AgentClass, OrganizationClass, EnvironmentClass, ActionClass, PlanClass, Property, Operation, Goal, Belief, Perception and Planning; (ii) Relationships: Association, Inhabit, Dependency, Generalization, Aggregation and Composite; and (iii) Notes.
- *Organization Diagram*. The Domain Model generated when creating the Class Diagram was used to create the Organization Diagram that contemplates the following: (i) the relationships ownership and play, (ii) agent role and object role, and (iii) agent, organization and environment. These new elements are covered in the domain model and graphical model that were used in the Organization Diagram. However, some adjustments in the domain model were required. The inhabit relationship has its semantics changed to allow agents and organization to inhabit the environment. The association, dependency, generalization, aggregation and composition were removed because they are not part of Organization Diagram.
- *Role Diagram.* The same Domain Model was used to create the Role Diagram. The elements that appear in both organization and Role Diagrams were preserved. The preserved entities are Agent Role and Object Role and the preserved relationships are Association, Control, Dependency, Generalization and Aggregation relationships. The graphical representation of elements, relationships and diagrams of MAS-ML tool are presented through a case study in the next sections.
- *Sequence Diagram.* The creation of the Sequence Diagram was started by mapping all the elements presented in the MAS-ML metamodel to Emfatic 2.0 language (Emfatic, 2014). In this process we associate the elements presented in the Sequence Diagram with their graphical representations. The elements presented in the diagram are as follows: Class, AgentClass, OrganizationClass, EnvironmentClass, ControlStructures, AgentRoleClass, PlanClass, ActionClass, Planning, and Perception. Graphical representations of Class, AgentClass, AgentRoleClass, OrganizationClass, and EnvironmentClass in the diagram are illustrated in Fig. 31.

The relationships present in the Sequence Diagram are as follows: Action, AgentMessage (Activate, Cancel, Commitment, Create, Deactivate, Default, Destroy), Change (DeactivateReactivate, DeactivateCreate, DestroyCreate, DestroyReactivate), For, If, Else, ObjectMessage, Perception, Plan, and Planning as depicted in Fig. 32.

- *Activity Diagram.* The Activity Diagram was created according to the same approach followed in the creation of the Sequence Diagram. The metamodel of the MAS-ML 2.0 was transcribed to

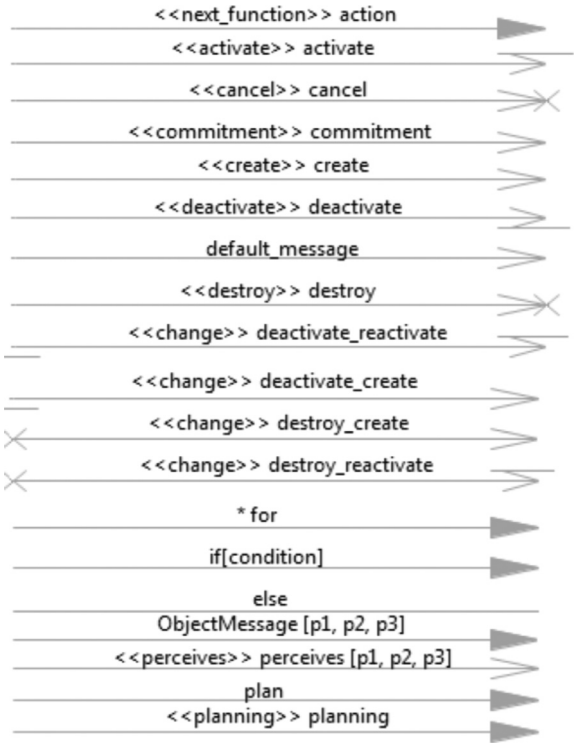

Fig. 32. The representation of MAS-ML relationships in the Sequence Diagram of MAS-ML tool.

Emfatic language, the elements were identified and their representations were defined. The elements are Environment, Organization, Role Agent, Initial State, Final State, Decision, Intercalation, Join, Fork, and Action. Environment, Organization, Role and Agent. Initial State and Final State mark the beginning and end of the execution of the activity respectively. Decision represents the conditional where the flow can follow a certain path depending on a predetermined value. Intercalation element determines the end of a decision, where paths that can be taken through a Decision converge. Fork represents the beginning of a run in parallel, and Join represents the point of synchronization between executions in parallel. The Action element represents the action performed by the agent, and the relationship Flow indicates the flow of actions that the agent performs. In Fig. 33 we show the main elements presented in the Activity Diagram using the same representation of the tool itself.

6.4. Overview of the MAS-ML tool

The deployed environment is a plug-in of Eclipse platform (Eclipse, 2013). Thus, the user can use the same environment to create models and code artefacts making use of the features offered by the platform. Given that many frameworks, including JADE, Jadex (Braubach et al., 2004), and Jason (Bordini et al., 2007), are based on the Java platform, the integration of frameworks from different sources and purposes helps developers to implement MAS and favours the code generation within the same development environment. Fig. 34 shows a Class Diagram view from the MAS-ML tool. The main components in the interface are highlighted by letters. Each component has a specific function in the tool, according to the following:

- ***Package Explorer (A)***. For each new modelling project, the files that are used during the modelling process are created and imported, for example, libraries, text documents, and images. The package explorer embodies the core functionalities allowed in a tree structure of files in order to improve their management and manipulation.
- ***Modelling View (B).*** The models that are created must necessarily be viewed in order to meet two basic requirements: understanding and communication. In this sense, the modelling view allows developers to view and edit the templates interactively.
- ***Palette Nodes (C)***. The builders that are part of the diagrams proposed by MAS-ML are the nodes and relationships palette. Thus, developers can create instances of these constructors that are displayed in the modelling view. It is possible to identify in Fig. 34 some of these elements such as ActionClass and AgentClass, for example.
- ***Relationship Palette (D)***. The relationships that can be established between manufacturers present in nodes are available in the palette relationship. In Fig. 34 it is possible to identify some of these relationships, for example, the association established between these two agents (i.e. AgentA and AgentB) shown in modelling view.
- ***Properties View (E)***. Ever-present concerns during the development of the MAS-ML metamodel are (i) the identification and representation of the characteristics of the properties of the MAS-ML metaclasses; and (ii) the organization and management of the properties that are inherited from metaclasses

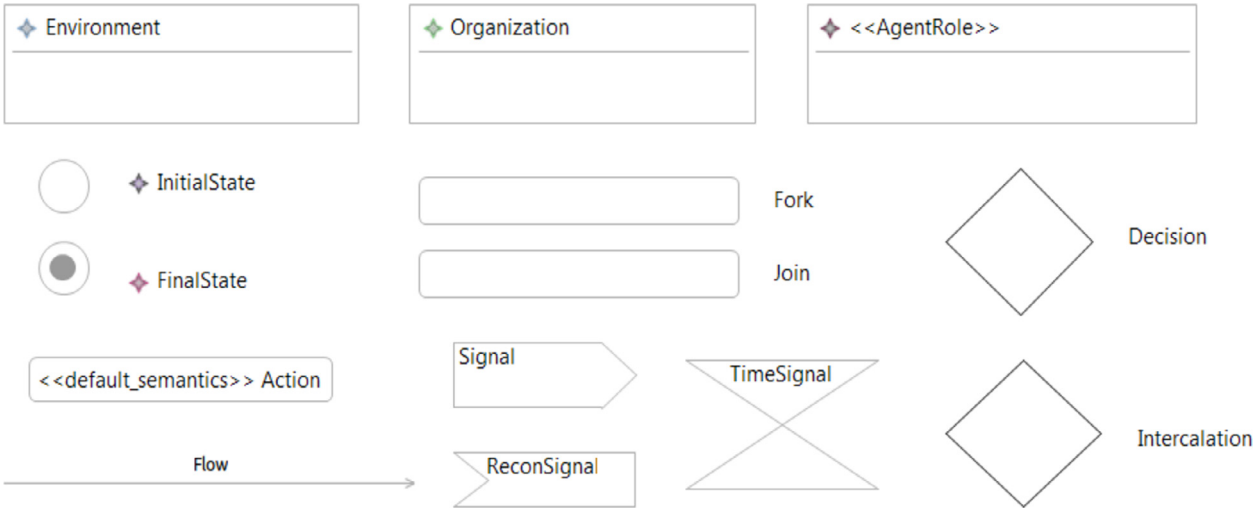

Fig. 33. The representation of MAS-ML elements in MAS-ML tool Activity Diagram.

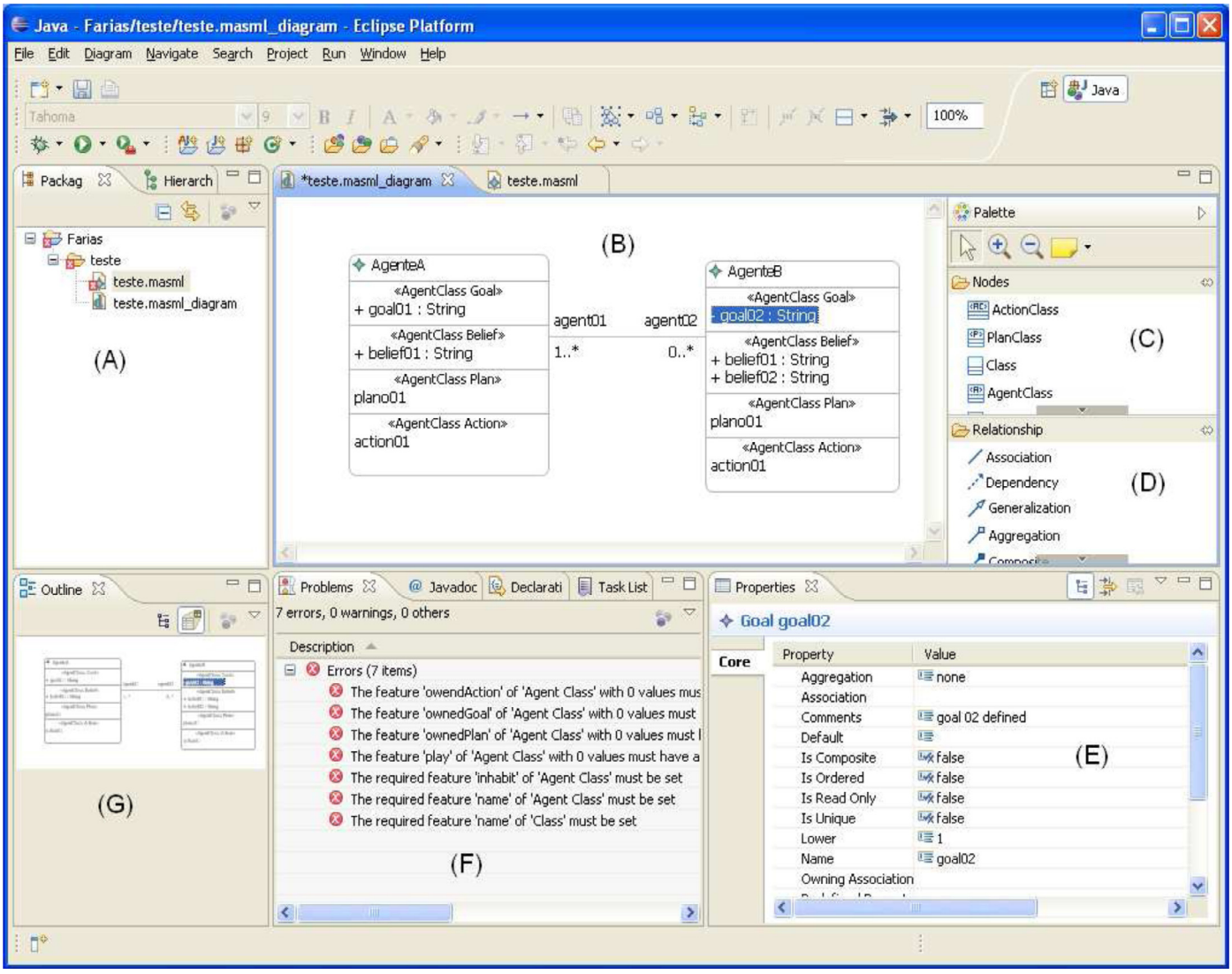

Fig. 34. An MAS-ML tool overview.

are extended from the UML to avoid creating conflicts or inconsistencies. These properties define exactly the models and are used, for example, to distinguish the models presented in the diagrams. Looking at Fig. 34, the difference between the two agents presented in the modelling view is the difference between the Strings "AgentA" and "AgentB" attributed to its name property. Thus, the importance of properties view is evident because it allows the manipulation of the model properties. This view displays the properties defined in the MAS-ML metamodel when the model is displayed and selected in the modelling view. In Fig. 34 the properties of AgenteB.goal02 can be seen.

Problems View (F). Given the need for validating the created models, the tool provides developers with a functionality of model validation. That is, the tool allows developers to check if the created model has (or not) inconsistencies considering the MAS-ML metamodel. If any inconsistency is detected, then they will be reported in the problem view. Fig. 26 shows some problems captured. In this case, the following well-formed rules are not respected: (i) every agent must have an action, (ii) every agent should have a goal, (iii) every agent must have a plan, and so on. These inconsistencies are compared with the model presented in (B). This feature is particularly important to enable the use of modelling MASs within the context of model-driven development, in which models are seen as first-order artefacts. Inconsistent models impair the transformation of models in model-centric software development.

Outline View (G). An overview of the distribution of the model elements can be seen in the outline view.

The MAS-ML tool code and generated plug-ins are available in https://sites.google.com/site/uecegessi/masmltool.

7. Case study

TAC-SCM application is used to illustrate the benefits of TAO and MAS-ML 2.0 where agents with different architectures are elicited to model different strategies to the problem solution. Section 7.1 describes the TAC-SCM environment, Section 7.2 shows the agents and agent roles by using the TAO templates (TAO templates were presented in Section 4) and Section 7.3 shows the MAS models done in the MAS-ML modelling tool that follows the specification of MAS-ML 2.0.

7.1. TAC-SCM

TAC (Trading Agent Competition) (Wellman et al., 2002) is an environment that enables the achievement of simultaneous auctions,

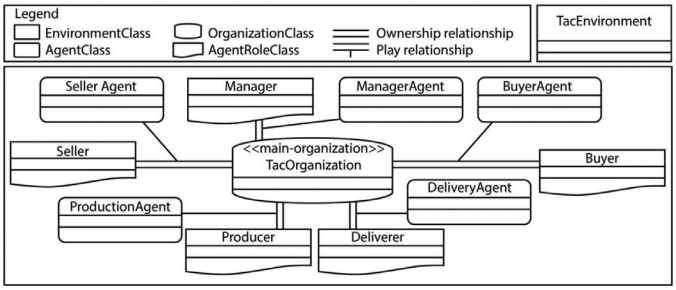

Fig. 35. Organization diagram proposed for TAC-SCM in MAS-ML 2.0.

test techniques, algorithms and heuristics to use in negotiation. There are two types of games in competition: TAC-Classic (Wellman et al., 2002) and TAC-SCM (Sadeh et al., 2003). TAC-SCM concerns the planning and management of the organization activities across a supply chain. The TAC-SCM scenario is designed to capture the challenges in an integrated environment for acquisition of raw materials, production and delivery of finished goods to customers. This environment is highly dynamic, stochastic and strategic (Arunachalam, 2004).

The game starts when one or more agents connect to a server game. The server simulates suppliers and customers, providing a bank, manufacturing and service of storage of goods to individual agents. The game is over a fixed number of simulated days, and in the end, the agent with largest sum of money in the bank is the winner (Collins et al., 2006).

7.2. Modelling TAC-SCM with TAO and MAS-ML 2.0

In a TAC-SCM environment it is possible to identify the main organization General Store and its two sub-organizations, Imported Bookstore and Secondhand Bookstore, performing the roles Market of Special Goods and Market of Used Goods respectively. The modelling of the TAC-SCM environment and the General Store organization uses TAO templates and are shown below. Moreover, in this system five types of agents are identified: DeliveryAgent, SellerAgent, BuyerAgent, SupplierAgent and ManagerAgent. All these agents, described in detail in the next Section, inhabit the environment TAC-SCM.

TACEnvironment

*Environment_Class TACEnvironment*1}
Behaviour setOf{Open, Heterogeneous}
Relationships setOf{Inhabit_TACEnvironment _GeneralStore,
Inhabit_TACEnvironment _Imported Bookstore,
Inhabit_ TACEnvironment _ DeliveryAgent,
Inhabit_ TACEnvironment _ SellerAgent,
Inhabit_ TACEnvironment _ BuyerAgent,
Inhabit_ TACEnvironment _ SupplierAgent
Inhabit_ TACEnvironment _ ManagerAgent, ...}
Events perceived: setOf{Group_Forming}
end Environment_Class

General_Store

Organization_Class General_Store
Relationships setOf{Agregation_GeneralStore_ImportedBookstore,
Agregation_GeneralStore_SecondhandBookstore, Ownership_buyer,
Ownership_seller, Ownership_delivery, Ownership_supplier,
Ownership_manager }
end Organization_Class

Fig. 35 depicts the Organization Diagram for TAC-SCM MAS. This diagram represents the TacOrganization and describes the agents and agent roles in the specific environment and their relationships of inhabit (all agents, all agent roles and the organization inhabit TacEnvironment), TacOrganization ownership each agent role and each agent plays its respective agent role in TacOrganization context.

The agents and their agent roles are presented in next subsections. The DeliveryAgent is presented in Section 7.2.1, the BuyerAgent is presented in Section 7.2.2, SellerAgent is presented in Section 7.2.3, Section 7.2.4 shows the ProductionAgent and Section 7.2.5 presents the ManagerAgent. This section presents the TAO template and the modelling of each agent in static diagrams, followed by the Sequence Diagram and finally in the Activity Diagram.

7.2.1. DeliveryAgent modelling in MAS-ML 2.0

The architecture of each agent, chosen according to the role played by the agent in the game, is discussed in the following. DeliveryAgent was modelled with the original MAS-ML agent representation and it must achieve the goal of delivering products to consumers, so it has a specific goal that can be achieved with a sequence of actions. Consequently, the DeliveryAgent class can be represented by the template of the MAS-ML agent, where the elements are identified to DeliveryAgent. Observe that it has not Events generated and perceived (the same is applied to other agents of this section). The DeliveryAgent template representation is shown below.

DeliveryAgent

Agent_Class DeliveryAgent
Beliefs {ordersToDelivery}
Goals {DeliveryProductsClient}
Actions{CheckDeliveryCurrentDate,CheckDeliveryAvailableProducts, DeliveryProduct}
Plans {deliveryProducts}
Events generated: *{ }*, perceived: *{ }*
Roles {Delivery}
Relationships{Inhabit_TACEnvironment, play_shipper}
end Agent_Class

Its representation in MAS-ML 2.0 static diagrams is shown in Fig. 36. It is defined according the MAS-ML architecture, and it has a goal named deliveryProductsClient that is related to deliveryProducts plan, the deliveryBeliefs is defined to the agent, the actions CheckDeliveryCurrentDate, CheckDeliveryAvailableProducts and DeliveryProduct are available and they are associated to deliveryProducts plan in this same sequence.

The DeliveryAgent has an associated agent role, the Deliverer. Its representation in TAO templates is shown below where its beliefs, actions and relationships are described. The communication protocol was defined with Fipa and commitments are defined to delivery.

Deliverer

Agent_Role_Class Deliverer
Beliefs {deliveryBeliefs}
Actions setOf { checkDeliveryCurrentDate, checkDeliveryAvailableProducts,
deliveryProducts }
Protocols {FIPA_Protocol}
Commitments {delivery}
Relationships {ownership_TacOrganization}
end Agent_Role_Class

Its representation in static diagrams of MAS-ML 2.0 is shown in Fig. 37. The beliefs of agent role are defined as deliveryBeliefs, and the Deliverer has three duties actions: checkDeliveryCurrentDate, checkDeliveryAvailableProducts and deliveryProducts. The protocol-to-message exchange has the label named request, and the content has the name of respective action, and the role that receives the messages is Producer.

Fig. 38 describes the Sequence Diagram of DeliveryAgent. Note that the behaviour starts with the deliveryProduct plan execution, and associated with this plan a sequence of actions is executed in the following order: checkDeliveryCurrentDate, checkAvailableProducts

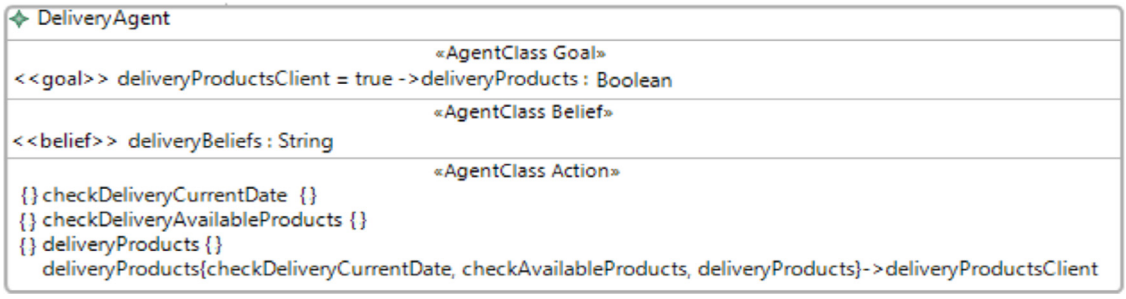

Fig. 36. A DeliveryAgent proposed for TAC-SCM in MAS-ML 2.0.

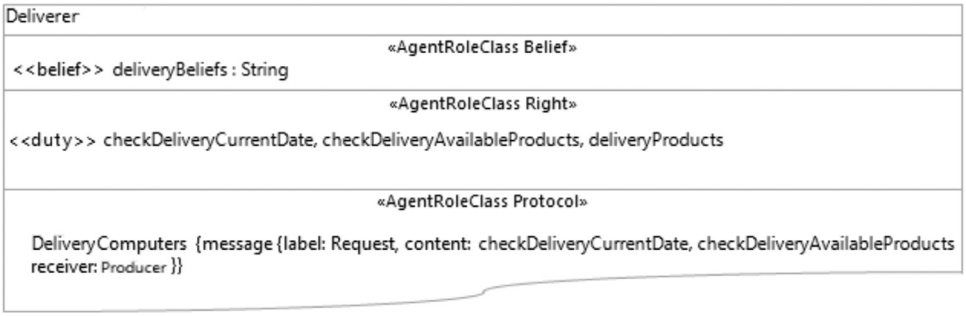

Fig. 37. Role of DeliveryAgent proposed for TAC-SCM in MAS-ML 2.0.

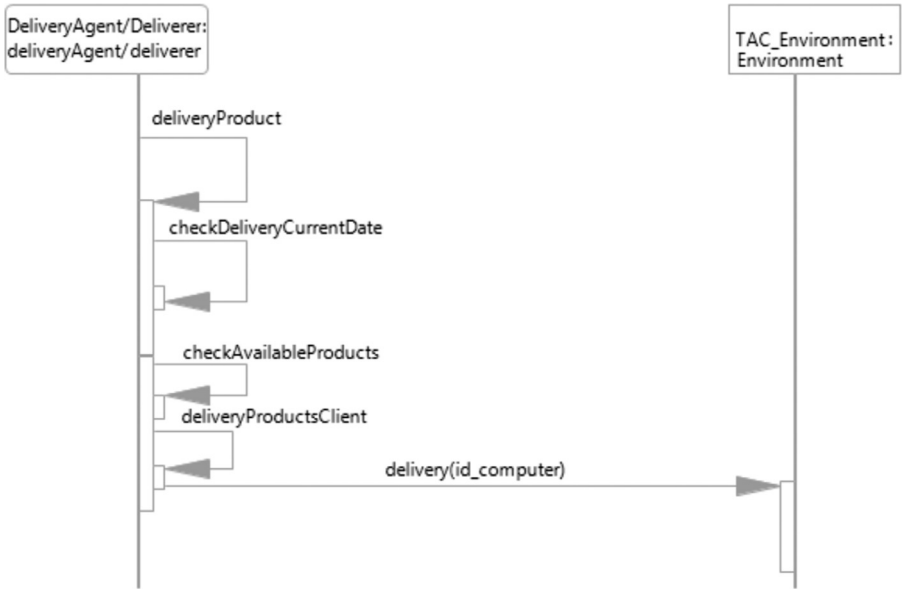

Fig. 38. Sequence Diagram of DeliveryAgent in MAS-ML 2.0.

and deliveryProductsClient. Lastly, DeliveryAgent sends a message to TAC_Environment to deliver a computer with a specific id. The DeliveryAgent is associated to Deliverer agent role.

The Activity Diagram of DeliveryAgent shows the deliveryProduct plan execution, and associated with this plan a sequence of actions is executed in the following order: checkDeliveryCurrentDate, checkAvailableProducts and deliveryProductsClient. This plan is associated to deliveryProductsToClient goal, and the agent should be associated to Deliverer agent role for executing it. Fig. 39 shows the Activity Diagram of DeliveryAgent.

7.2.2. SellerAgent modelling in MAS-ML 2.0

The SellerAgent agent offers PCs to consumers and receives the payment. As reflex agents respond quickly to perceptions (Weiss, 1999), the sellerAgent should be modelled as a reflex agent because of the necessity for quick answers. The TAO templates of the SellerAgent class (Simple Reflex Agent) are shown below:

SellerAgent

AgentClass
Perceptions {clientRequest, managerRequest, payment}
Actions {offerProduct, receivePayment, mountDeliveryProducts}
Events generated: { }, perceived: { }
Roles {Seller}
Relationships{Inhabit_environment_Tac,play_seller}
end Agent_Class

SellerAgent has the following actions: offerProduct, receivePayment and mountDeliveryProducts. It does not have a next function

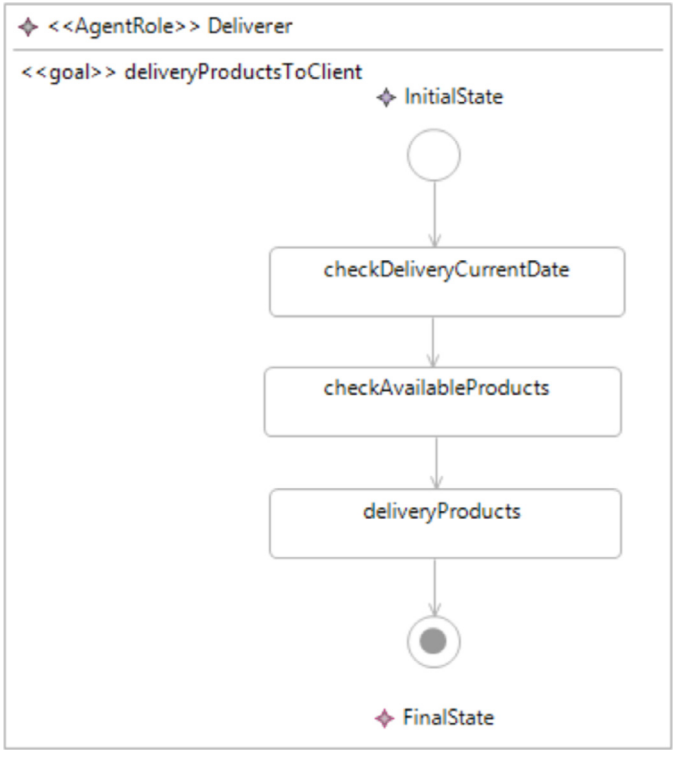

Fig. 39. Activity Diagram of DeliveryAgent in MAS-ML 2.0.

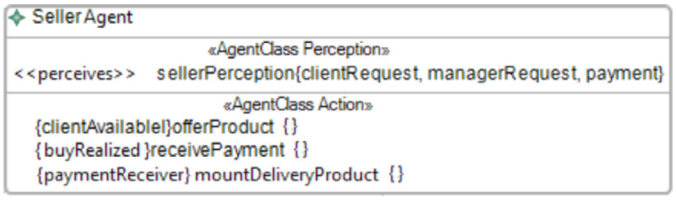

Fig. 40. A SellerAgent proposed to TAC-SCM in MAS-ML 2.0.

because it is a Simple Reflex Agent and it can perceive clientRequest, managerRequest, and payment. Fig. 40 shows the SellerAgent modelled in MAS-ML 2.0.

SellerAgent has an associated agent role, the Seller agent role. Its representation in TAO templates is shown below, where its actions and relationship with TacOrganization are shown to Seller. The communication protocol was defined with Fipa and commitments are defined to make_sales.

Seller
Agent_Role_Class Seller *Actions setOf {offer_product, receive_payment}* *Protocols {FIPA_Protocol}* *Commitments {make_sales}* *Relationships{ownership_TacOrganization}* *end Agent_Role_Class*

Its representation in static diagrams of MAS-ML 2.0 is shown in Fig. 41. The Seller does not have beliefs because it is a Simple Reflex Agent, but it has two rights: offerProduct and receivePayment. The protocol to message exchange SimpleNegotiation has the label named Request, the content has the name of respective action, and the role that receives the messages is Seller.

The Sequence Diagram of the SellerAgent (Fig. 42) shows their execution through its perceptions, where sellerRequest, managerRequest and payment are perceived by this agent, and one action associated with a condition-action rule is executed according to the perception of this agent. This agent is associated to Seller agent role.

The Activity Diagram of SellerAgent shows the perception of this agent and following it is executed one action according the condition-action rules. This agent does not have goals and a plan associated with a sequence of actions. Fig. 43 shows the Activity Diagram to SellerAgent.

7.2.3. BuyerAgent modelling

The BuyerAgent chooses when to make new requests for components and makes the payment. As reflex agents respond quickly to perceptions (Weiss, 1999), the BuyerAgent should be modelled as reflex agents because of the necessity for quick answers. The TAO templates of the BuyerAgent class (Model-Based Reflex Agent) are shown below:

BuyerAgent
AgentClass BuyerAgent *Perceptions { supplierQuotation, supplierConfirmation, manager_requirement}* *Beliefs {auction_price, stock_missing}* *Actions {RequestQuotation, bid, pay}* *Events generated: { }, perceived: { }* *Roles {Buyer}* *Relationships{Inhabit_TACEnvironment, play_buyer}* *end Agent_Class*

Fig. 44 shows the BuyerAgent (Model-Based Reflex Agent) in static diagrams of MAS-ML 2.0. Note that reflex agents do not have plan and goals, and a new component perceives is present in both. BuyerAgent has the following actions: requestQuotation; bid and pay; it has a next function nextFuncBuyer that actualizes its beliefs with the perceptions and realizes inference and it can perceive supplierQuotation, supplierConfirmation, manager_requirement. The beliefs of this agent are saved in a prolog file named beliefsBuyer.pl.

BuyerAgent has an associated agent role, the Buyer agent role. Its representation in TAO templates is shown below, where its beliefs, actions and relationship with TacOrganization are described. The communication protocol was defined with Fipa and commitments are defined to shop.

Buyer
Agent_Role_Class Buyer *Beliefs {auction_price, stock_missing}* *Actions setOf { requestQuotes, offer, pay }* *Protocols {FIPA_Protocol}* *Commitments {shop}* *Relationships{ownership_TacOrganization}* *end Agent_Role_Class*

Its representations in static diagrams of MAS-ML 2.0 are shown in Fig. 45. The beliefs of Buyer are defined as beliefsBuyer, and the Buyer has three right actions: requestQuotes, offer and pay. The protocol to message exchange SimpleNegotiation has the label named Request, the content has the name of respective action, and the role that receives the messages is Supplier.

The Sequence Diagram of the Fig. 46 illustrates the behaviour of BuyerAgent. Initially, the perception is executed and the information of supplierQuotation, supplierConfirmation and managerRequest are perceived, then the next function (nextFuncBuyer) is executed and beliefs updated with the perceptions and inferences made; finally, one of the possible actions is executed always that the condition-action rule is true. Note that BuyerAgent is related to Buyer agent role.

The Activity Diagram of BuyerAgent shows the perception of this agent and the next function performed in the following. Next step of this agent is to execute one action according to the condition-action rules. This agent does not have goals and a plan associated with a sequence of actions. Fig. 47 shows the Activity Diagram for BuyerAgent.

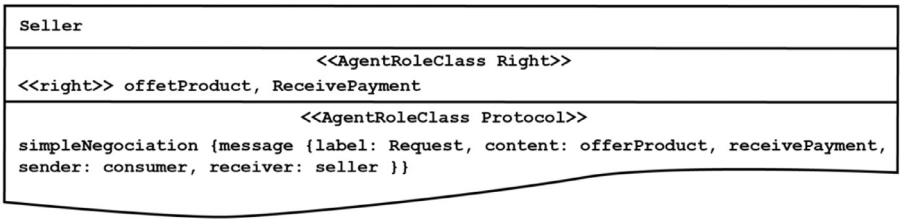

Fig. 41. AgentRole of SellerAgent in MAS-ML 2.0.

Fig. 42. Sequence Diagram of SellerAgent in MAS-ML 2.0.

7.2.4. ProductionAgent modelling in MAS-ML 2.0

ProductionAgent agent needs to attend to the current demand by assembling computers and managing inventory. To achieve this goal it cannot use a predefined plan, because this dynamic environment requires a different set of actions depending on current demand. Thus, SupplierAgent can be defined as a Goal-Based Agent.

The template of the ProductionAgent class is described as follows. Its goal, beliefs, perceptions, actions and relationship with TacOrganization are described. The communication protocol was defined by FIPA and planning and special functions (nextFunction, FormulateGoalFunction and formulateProblemFunction) are defined also.

ProductionAgent

Agent_Class ProductionAgent
Goals {satisfyDemand}
Beliefs{stock_parts,pc_for_production}
Perceptions {request_production}
Actions {check_available_parts,
request_necessary_parts, check_parts,
build_parts, test_computer, computer_available}
Planning {produce_computers}
NextFunction{Next_Function_production}
FormulateGoalFunction{GoalFuncProduction}
FormulateProblemFunction {ProbFuncProduction}
Events generated: {}, perceived: {}
Roles {producer}
Relationships {Inhabit_TACEnvironment, play_producer}
end Agent_Class

Fig. 48 shows the ProductionAgent and its internal components in static diagrams of MAS-ML 2.0. It is composed of the following actions: check_available_parts, request_necessary_parts, check_parts, build_parts, test_computer, computer_available, a next function nextFuncProduction that actualizes its beliefs with the perceptions and realizes inference, and perceives requestProduction. The beliefs of this agent are saved in a prolog file named beliefsProduction.pl and it has the goal of satisfyDemand. The ProductionAgent has a planning of produceComputers and it is related to satisfyDemand goal, but a sequence of actions was not defined because it is possible only in execution time. The formGoalManager and formProbManager are available also.

The ProductionAgent has an associated agent role, the Producer. Its representation in TAO templates is shown below, where its beliefs, actions and relationships are described. The communication protocol was defined with Fipa and commitments are defined to produce.

Producer

Agent_Role_Class Producer
Beliefs {beliefsProduction}
Actions setOf { checkAvailableParts, requestNecessaryParts, checkParts, buildParts,
testComputer, computerAvailable }
Protocols {FIPA_Protocol}
Commitments {produce}
Relationships {ownership_TacOrganization}
end Agent_Role_Class

Its representation in static diagrams of MAS-ML 2.0 is shown in Fig. 49. The beliefs of agent role are defined as beliefsProduction,

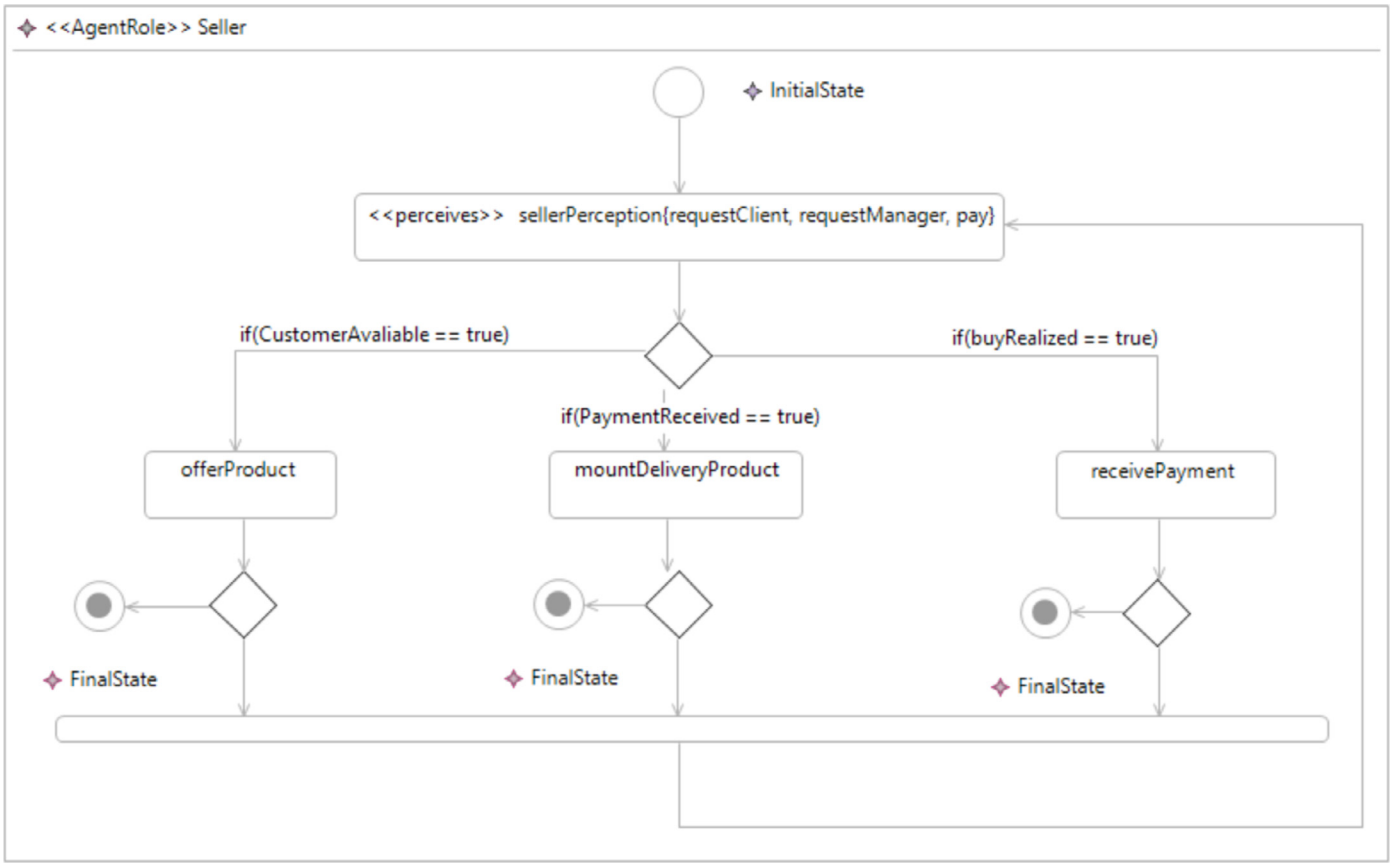

Fig. 43. Activity Diagram of SellerAgent in MAS-ML 2.0.

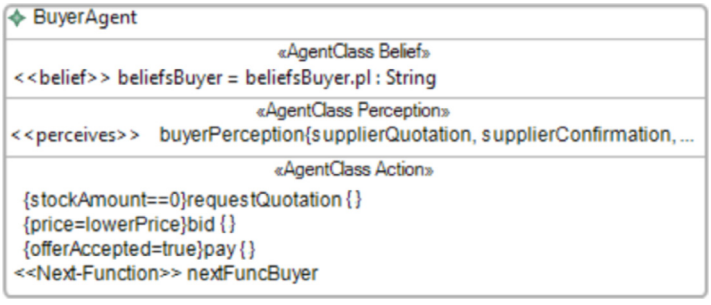

Fig. 44. A BuyerAgent proposed for TAC-SCM in MAS-ML 2.0.

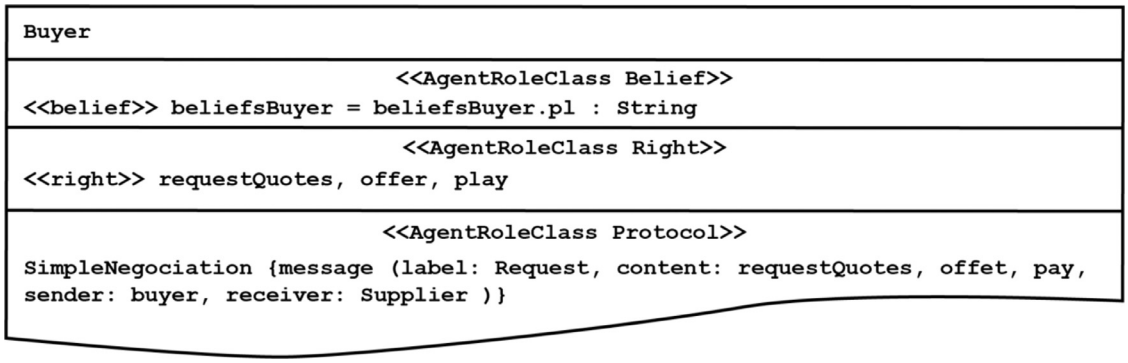

Fig. 45. AgentRole of BuyerAgent in MAS-ML 2.0.

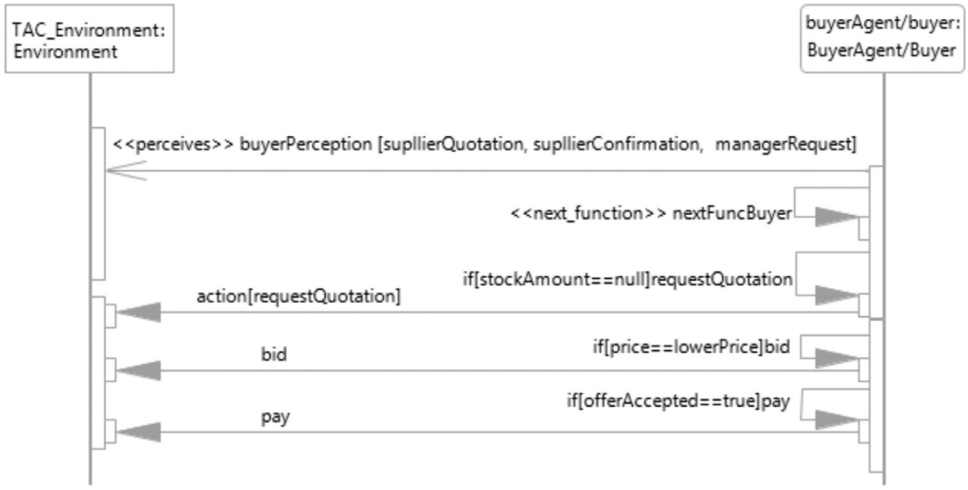

Fig. 46. Sequence Diagram of BuyerAgent in MAS-ML 2.0.

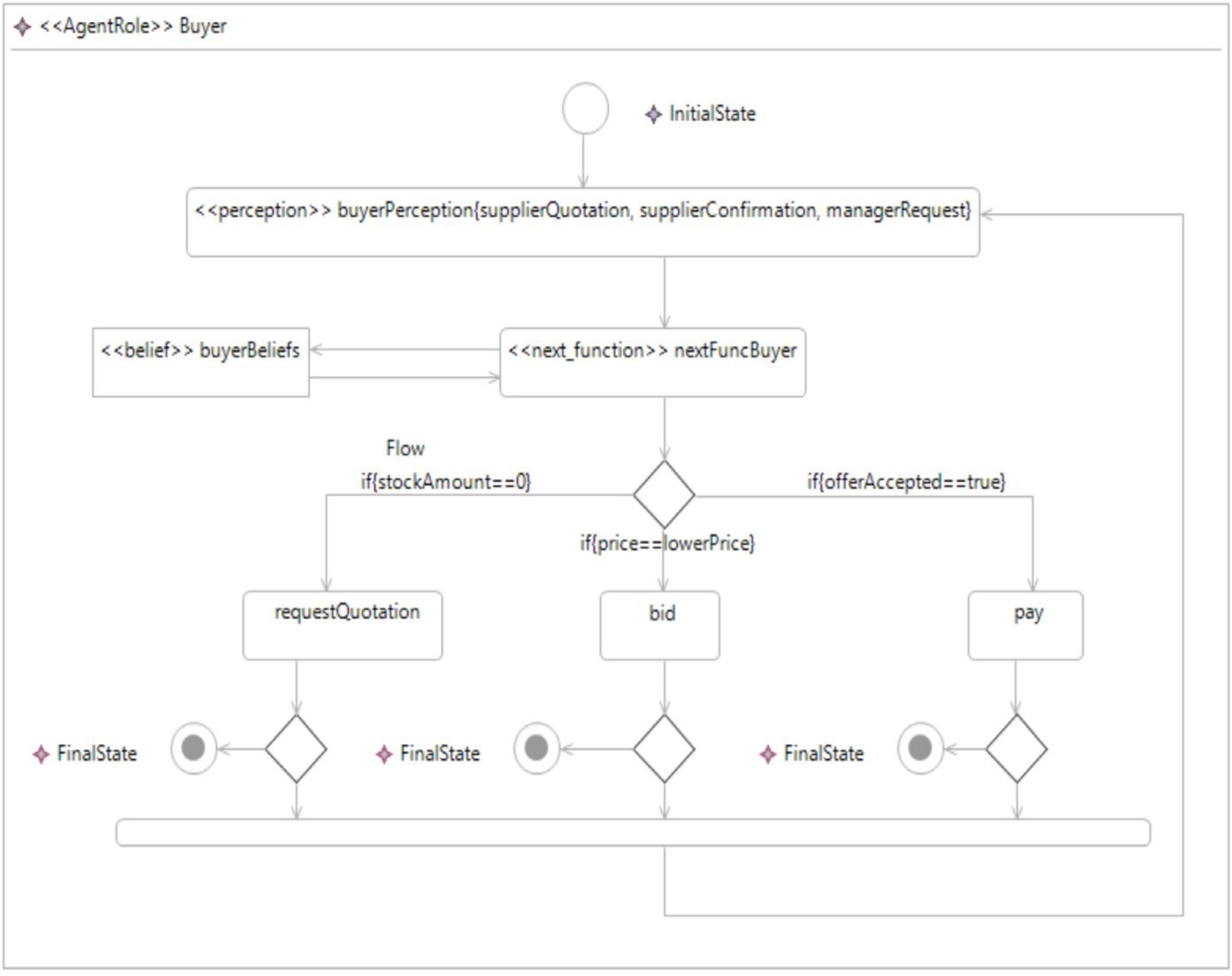

Fig. 47. Activity Diagram of BuyerAgent in MAS-ML 2.0.

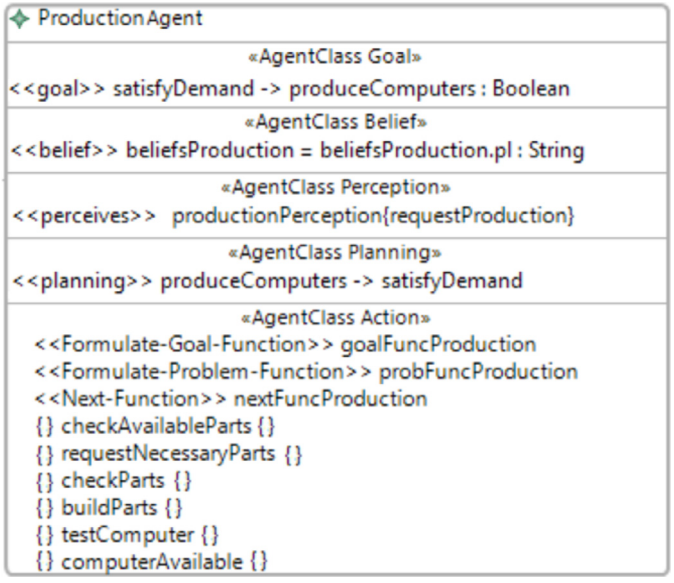

Fig. 48. The ProductionAgent proposed for TAC-SCM in MAS-ML 2.0.

and the Producer has six duties: checkAvailableParts, requestNecessaryParts, checkParts, buildParts, testComputer and computerAvailable. The protocol to message exchange has the label named request, the content was defined as requestNecessaryParts, and the role that receives the messages is buyer.

Fig. 50 shows the Sequence Diagram of the ProductionAgent. Note that the behaviour taken by the agent starts with its perception of productionRequest, after perceives the next function is executed objectify actualize the beliefs of this agent. goalFuncProduction and probFuncProduction are performed to give the goal and the problem formulated for the planning. The planning is executed and the sequence of actions resulting from planning are represented by a loop and actions associated to conditions stay into the loop and will execute according the planning.

The Activity Diagram of ProductionAgent shows the perception of this agent; following it is executes the nextFuncProduction to actualize its beliefs; goalFuncProduction and probFuncProduction are performed to give the goal and the problem formulated for the planning. The planning is executed and the sequence of actions resulting from planning is represented by a loop, and actions associated to conditions stay in the loop and will execute according to the planning. Fig. 51 shows the Activity Diagram for ProductionAgent.

7.2.5. ManagerAgent modelling in MAS-ML 2.0

Finally, the ManagerAgent agent is responsible for managing all staff and allocating the resources. This agent tries to maximize profit and sales: note that these goals can be in conflict. Then, the most appropriate architecture in this case is based on utility. The template of the ManagerAgent class is shown below: its goal, beliefs, perceptions, actions and relationship with TacOrganization are described. The communication protocol was defined with Fipa, and planning and special functions (next function, formulate goal function, formulate problem function and utility function) are also defined.

ManagerAgent

Agent_Class ManagerAgent
Goals {maximize_profit,maximize_sales}
Beliefs{Sales_information, shopping_information, productive_information }
Perceptions{buyer_requirements, seller_requirements, producer_requirements, shipper_requirements}
Actions{request_buying_price, request_selling_price, request_selling_computer, estimate_buying_price, estimate_selling_price}
Planning {manager_planning}
NextFunction {next_func_manager}
FormulateGoalFunction {form_goal_manager }
FormulateProblemFunction{ form_goal_manager }
UtilityFunction {utility_function_manager}
Events generated: {}, perceived: { }
Roles {manager}
Relationships {Inhabit_TACEnvironment, play_manager}
end Agent_Class

Fig. 52 shows the ManagerAgent and its internal components in static diagrams of MAS-ML 2.0. It is composed of the following actions: requestBuyingPrice, requestSellingPrice, requestSellingComputer, estimateBuyingPrice, estimateSellingPrice, a next function nextFuncManager that actualizes its beliefs with the perceptions and realizes inference; it can perceive BuyerRequest, SellerRequest, ProductionRequest and DeliveryRequest. The beliefs of this agent are saved in a prolog file named beliefsManager.pl and it has the goals of maximizeGain and maximizeSale. The ManagerAgent has a managerPlanning and it is related to maximizeGain and maximizeSale goals, but a sequence of actions was not defined because it is possible only in execution time. The formGoalManager, formProbManager and utilityFuncManager are also available.

The ManagerAgent has an associated agent role, the Manager. Its representation in TAO templates is shown below, where its beliefs, actions and relationships are described. The communication protocol was defined with Fipa and commitments are defined to produce.

Manager

Agent_Role_Class Manager
Beliefs {auction_price}
Actions setOf { requestBuyingPrice, requestSellingPrice, requestSellingComputer, estimateBuyingPrice, estimateSellingPrice }
Protocols {FIPA_Protocol}
Commitments {requestPrice, estimatePrice, sellingComputer}
Relationships {ownership_TacOrganization}
end Agent_Role_Class

Its representation in static diagrams of MAS-ML 2.0 is shown in Fig. 53. The beliefs of agent role are defined as beliefsManager; the Manager has four duties (requestBuyingPrice, requestSellingPrice, estimateBuyingPrice and estimateSellingPrice) and one right

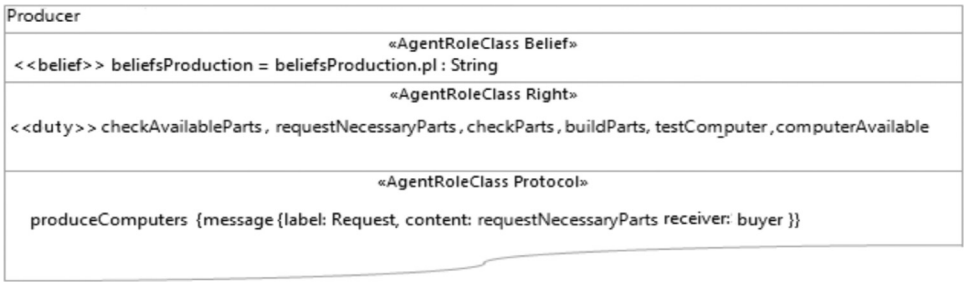

Fig. 49. Role of ProductionAgent proposed for TAC-SCM in MAS-ML 2.0.

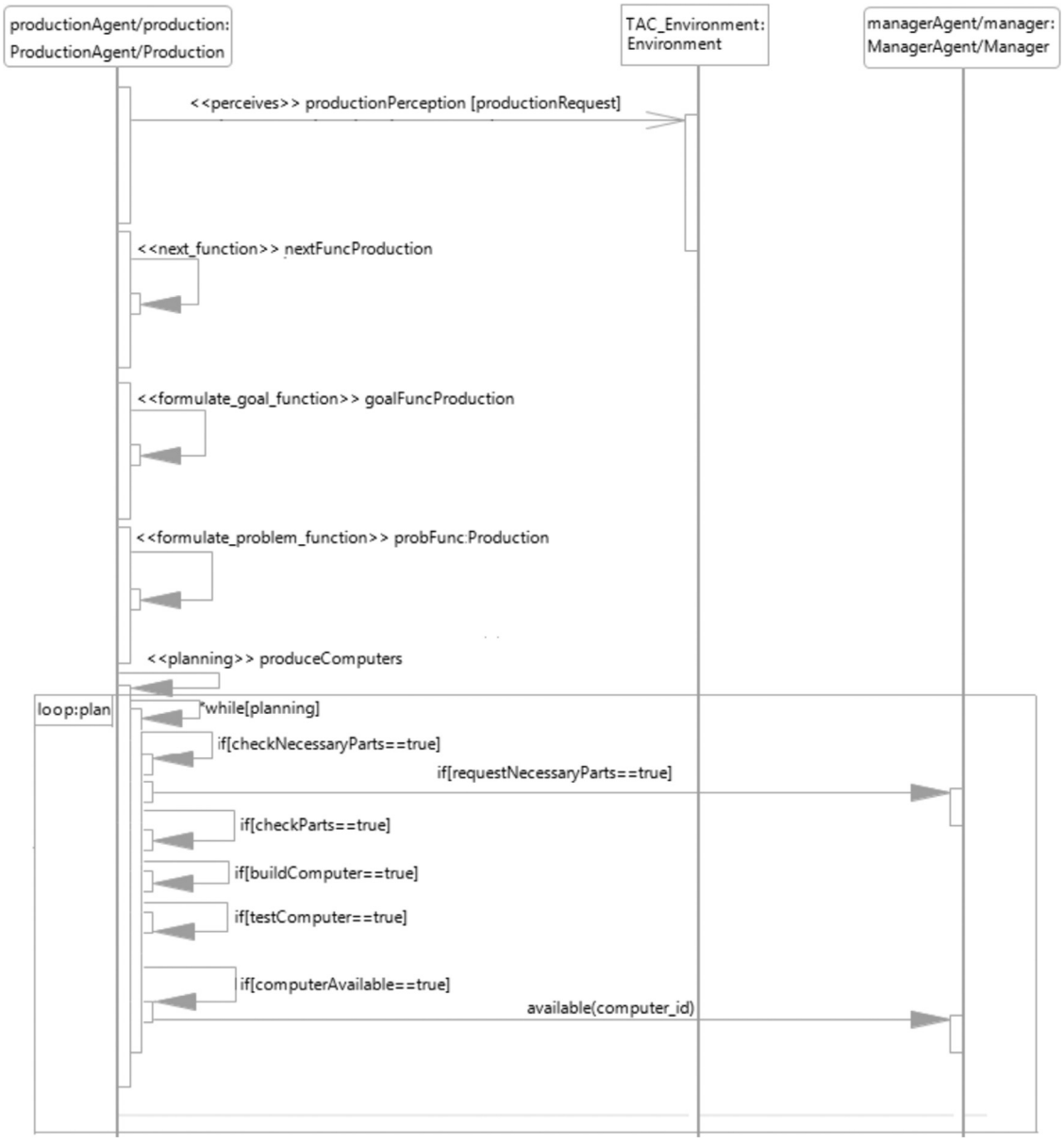

Fig. 50. Sequence Diagram of ProductionAgent in MAS-ML 2.0.

(requestSellingComputer). The protocol to message exchange (ManagerNegotiation) has the label named request and the content was defined as requestBuyingPrice or requestSellingPrice and the role that receives the messages is buyer or seller respectively.

Fig. 54 shows the Sequence Diagram of the ManagerAgent. Note that the behaviour taken by the agent starts with its perception of sellerRequest, buyerRequest, productionRequest and deliveryRequest, the next function is executed after perceives and it objectify actualize the agent beliefs. GoalFuncManager and probFuncManager are performed to give the goal and the problem formulated for the planning. The planning is executed and the sequence of actions resulting from planning are represented by a loop; the first step of this loop is execute the utility function to calculate the utility degree associated with each action, and actions associated with conditions stay in the loop and will execute after the utilityManager.

The Activity Diagram of ManagerAgent shows the perception of this agent; following it is executes the nextFuncManager to actualize its beliefs; goalFuncManager and probFuncManager are performed to give the goal and the problem formulated for the planning. The planning is executed, and the sequence of actions resulting from planning are represented by a loop, and actions associated with conditions stay in the loop and will execute according to the planning. Fig. 55 shows the Activity Diagram for ManagerAgent.

7.3. Modelling of TAC-SCM agents with other MAS modelling languages derived from UML

In this subsection will be described the modelling of agents presented in this case study of TAC-SCM using static diagrams of other languages that extend UML. The main objective is to demonstrate the increased expressiveness with the extension proposed by this paper in relation to other existing modelling languages. Of course, the existing gaps in the modelling of static diagrams have an impact on other diagrams of modelling languages.

The following modelling languages will be used in this comparison: MAS-ML (In its original form, as proposed by Silva, Choren and Lucena (2008a)), AML and AUML. The SellerAgent agents, BuyerAgent, DeliveryAgent, ProductionAgent and ManagerAgent will be presented so that they can be compared with the modelling of MAS-ML 2.0 in the case study presented.

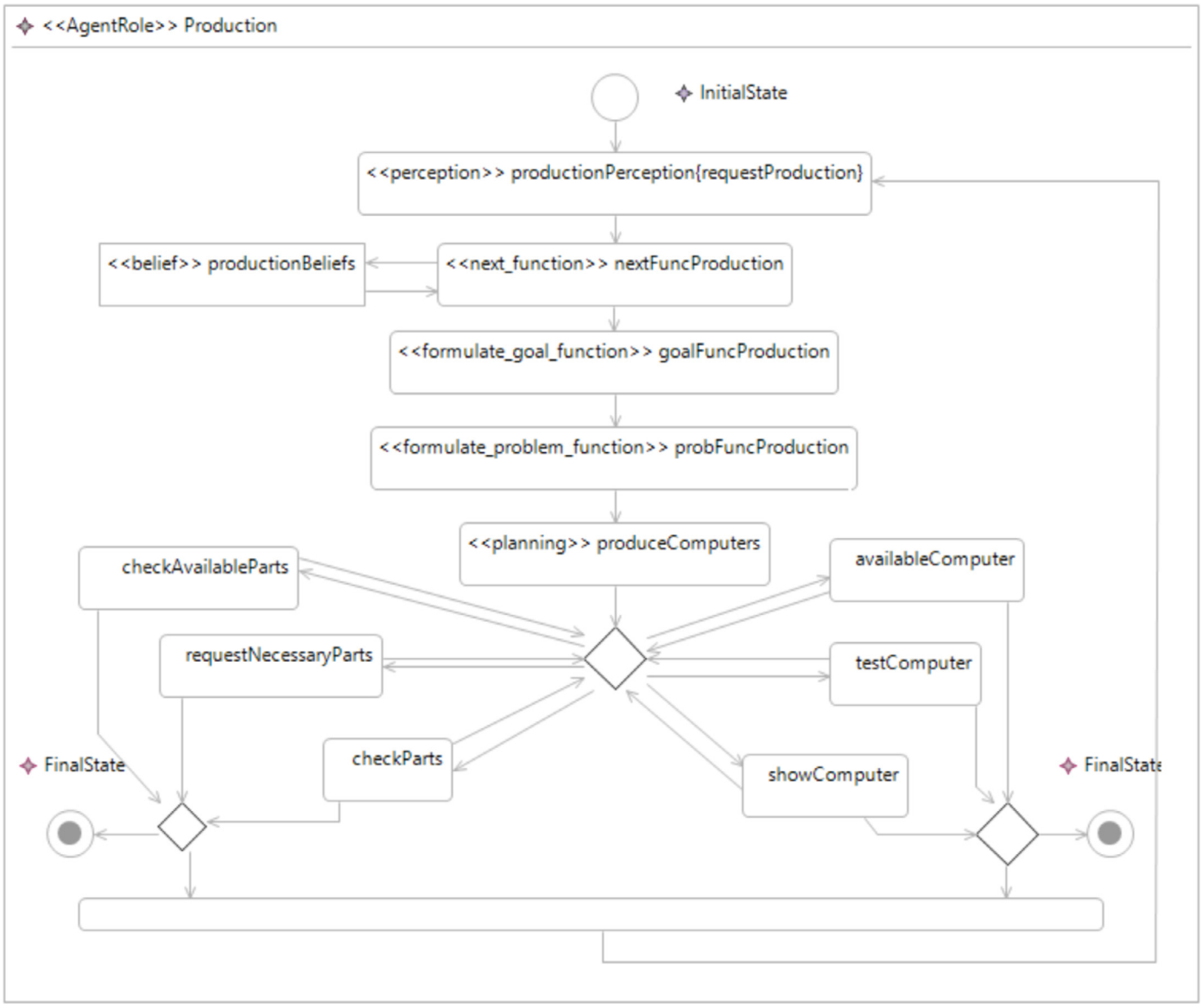

Fig. 51. Activity Diagram of ProductionAgent in MAS-ML 2.0.

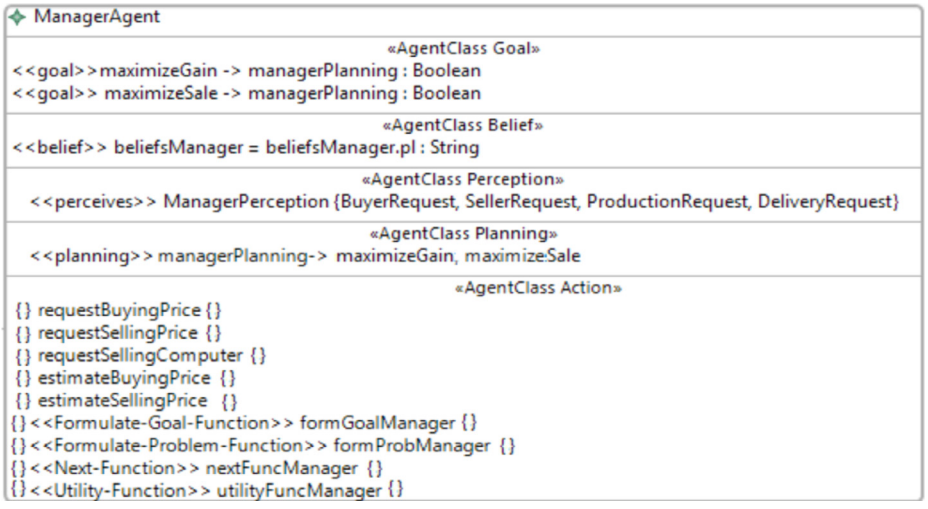

Fig. 52. A ManagerAgent proposed for TAC-SCM in MAS-ML 2.0.

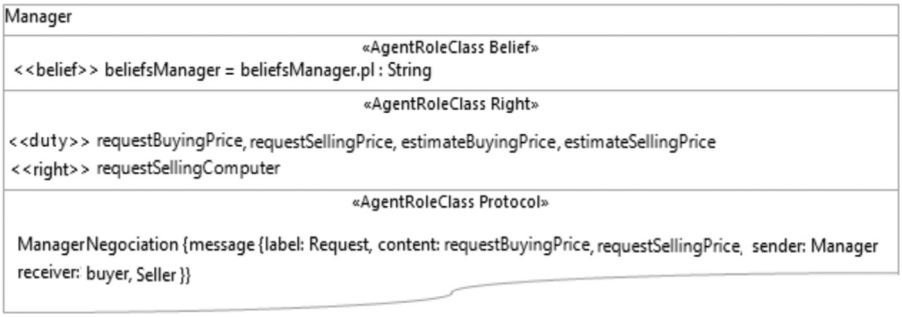

Fig. 53. Role of ManagerAgent proposed for TAC-SCM in MAS-ML 2.0.

7.3.1. Modelling of TAC-SCM agents with original version of MAS-ML

As proposed by Silva et al. (2008a), the agent must be represented in static diagrams of MAS-ML containing goals, beliefs, plans and actions. So, the agents here were modelled with these characteristics. The DeliveryAgent agent, shown in Fig. 36, was modelled with this architecture originally proposed by the language so will not be modelled again.

Fig. 56 shows the BuyerAgent with syntax available in the original version of MAS-ML; therefore, this agent has the goal of buyParts and beliefs, as well as having a predefined plan and the actions of requestQuotation, bid and pay. According to that presented in the case study, this agent should be modelled as a Model-Based Reflex Agent, so goals and predefined plans cannot be part of it. In addition, Model-Based Reflex Agents have perceptions and a next function, which are not possible to represent.

Fig. 57 shows the SellerAgent with syntax available in the original version of MAS-ML; therefore, this agent has the goal of selling customer products and associated beliefs, as well as having a predefined plan and the actions of offerProduct, ReceivePayment and mountDeliveryProduct. According to that presented in the case study, this agent should be modelled as a Simple Reflex Agent, so goals, beliefs and predefined plans cannot be part of it. In addition, reactive agents have perceptions, which are not possible to represent.

ProductionAgent with the syntax available in the original version of MAS-ML is shown in Fig. 58. This agent is composed of satisfyDemand goal, beliefsProduction, a predefined plan named produceComputers and the actions: check_available_parts, request_necessary_parts, check_parts, build_parts, test_computer and computer_available. According to that presented in the case study, this agent should be modelled as a Goal-Based Agent, so predefined plans cannot be part of it. In addition, Goal-Based Agents have perceptions, planning, next function, goal formulation function and problem formulation function, which are not possible to represent in the original version of MAS-ML.

ManagerAgent modelled with the syntax available in the original version of MAS-ML is shown in Fig. 59. This agent is composed of the goals of maximizeGain and maximizeSale, beliefs, a predefined plan called managerPlan and request_buying_price actions, request_selling_price, request_selling_computer, estimate_buying_price, and estimate_selling_price. According to that presented in the case study, this agent should be modelled as a Utility-Based Agent, so predefined plans cannot be part of it. In addition, Utility-Based Agents have perceptions, planning the next function, the formulate-goal function, formulation problem function and utility function, which are not possible to represent in the original MAS-ML.

7.3.2. Modelling of TAC-SCM agents with AUML

As proposed by Odell et al. (2000), the agent in Class Diagram of AUML is an entity that can consist of attributes, operations, capabilities, perceptions, protocols, related organizations and related roles. In addition, agents can have beliefs, classes, goals and strategies associated with it. Thus, agents modelled in this subsection will be composed of features proposed in AUML.

Fig. 60 shows the modelling of DeliveryAgent in Class Diagram of AUML language, and we can identify the Deliverer role of this agent in addition to the organization to which it belongs (TacOrganization). DeliveryAgent has beliefs, present in DeliveryBeliefs, and goal DeliveryProductsClient. It has also DeliveryProducts strategy, which functions as a plan associated with the agent. In DeliveryProducts the associated actions are present in the Actions compartment. The next function cannot be represented.

Fig. 61 brings the BuyerAgent modelled with Class Diagram of AUML, and we can identify the Buyer role of this agent in addition to the organization to which it belongs (TacOrganization). BuyerAgent are related to their beliefs present in BuyerBeliefs beyond the BuyerAct and operates as a plan associated with the agent. In BuyerAct the associated actions are present in the Actions compartment. According to that presented in the case study, this agent should be modelled as a reactive agent based on knowledge, so predefined plans cannot be part of it. Furthermore, the next function cannot be represented.

The SellerAgent modelled with Class Diagram of AUML is shown in Fig. 62, where we can identify the Seller role of this agent in addition to the organization to which it belongs (TacOrganization). The SellerAgent has the SellerAct and operates as a plane associated with the agent. In SellerAct the associated actions are present in the Actions compartment. According to that presented in the case study, this agent should be modelled as a simple reactive agent, so predefined plans could not be part of it.

The ProductionAgent modelled on Class Diagram AUML is shown in Fig. 63, where we can identify the Producer role of this agent in addition to the organization to which it belongs (TacOrganization). The beliefs of the agent are represented by ProductionBeliefs and its goal is represented by satisfyDemand. The ProductionAgent has the ProduceComputers and operates as a plane associated with the agent. In ProducerComputers the associated actions are present in the Actions compartment. According to that presented in the case study, this agent should be modelled as a Goal-Based Agent, so predefined plans could not be part of it. Furthermore, the next function, the formulate-goal function and formulate-problem function cannot be represented.

The ManagerAgent modelled with Class Diagram of AUML is shown in Fig. 64, where the Manager role for this agent can be identified in addition to the organization to which it belongs (TacOrganization). The beliefs of the agent are represented by ManagerBeliefs and goals are represented by maximizeSale and MaximizeGain. The ManagerAgent has the ManagerPlanning, which operates as a plan associated to the agent. In ManagerPlanning the associated actions are present in the Actions compartment. According to that presented in the case study, this agent should be modelled as a Utility-Based Agent, so predefined plans could not be part of it. Furthermore, the next function, the formulate-goal function, the formulation problem function and the utility function cannot be represented.

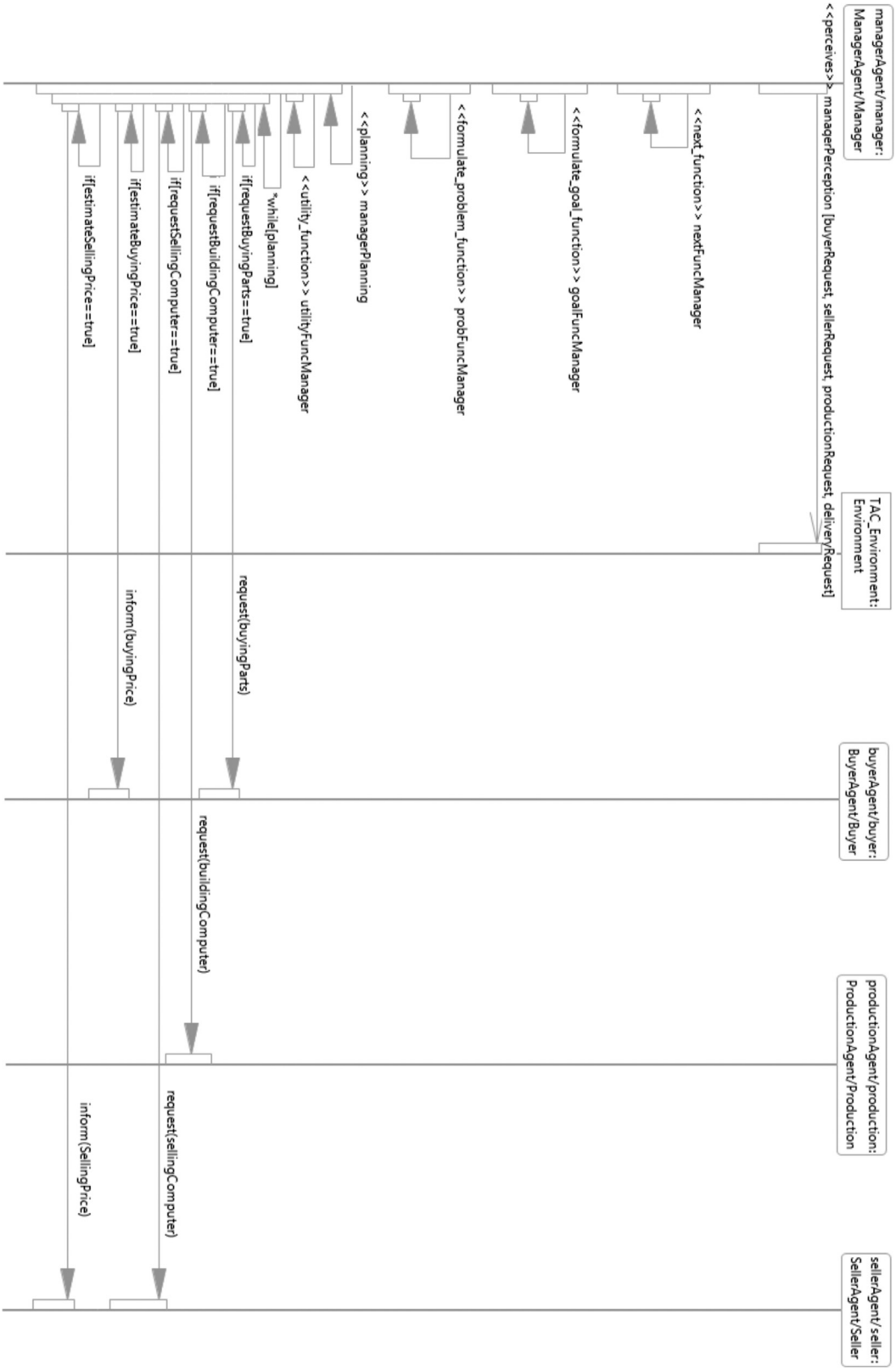

Fig. 54. Sequence Diagram of ManagerAgent in MAS-ML 2.0.

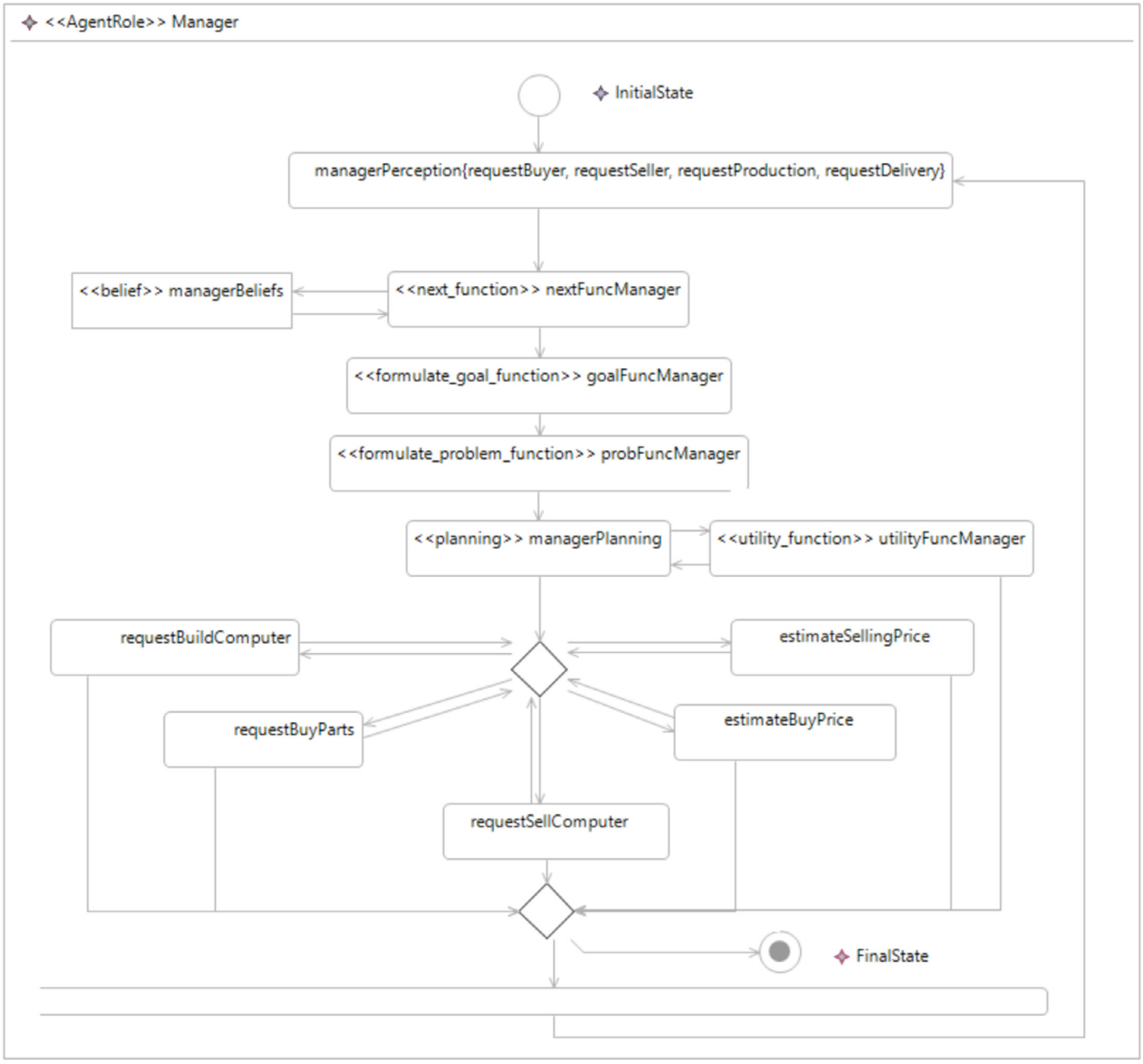

Fig. 55. Activity Diagram of ManagerAgent in MAS-ML 2.0.

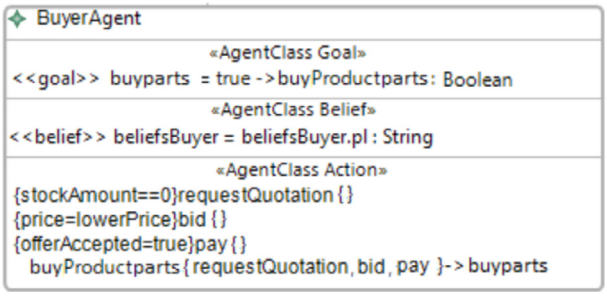

Fig. 56. The BuyerAgent modelled in MAS-ML.

In general, the reactive agents do not have strategy (plan), however, it was modelled with the actions in a class adorned with the strategy stereotype. In this modelling language, beliefs are represented as a class, therefore, may have related operations.

7.3.3. Modelling of TAC-SCM agents with AML

In AML, agents consist of plans, behaviour fragments, perceivers, actuators, beliefs and goals. The information that can be perceived is represented along with the perceivers of agents, and actions that agents can perform are represented along with their actuators. Thus, agents modelled in this subsection shall have the AML elements necessary for representation of their architecture.

Fig. 65 shows the modelling of DeliveryAgent in Class Diagram of AML modelling language, where it is possible to identify its goal (deliveryProductsClient) and beliefs (deliveryBeliefs), beyond plan (DeliveryProducts) related to behaviour fragments checkDeliveryAvailableProducts, deliveryProducts and checkDeliveryCurrentDate. The actuators legs are also available. The next function cannot be represented in AML, and the plan is not related to a specific goal. Additionally, goals and beliefs, which are structural characteristics of the agents, can be represented in specific classes that enable these classes to have operations.

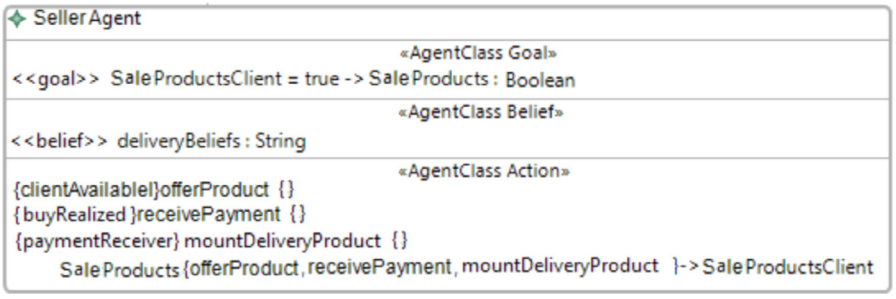

Fig. 57. The SellerAgent modelled in MAS-ML.

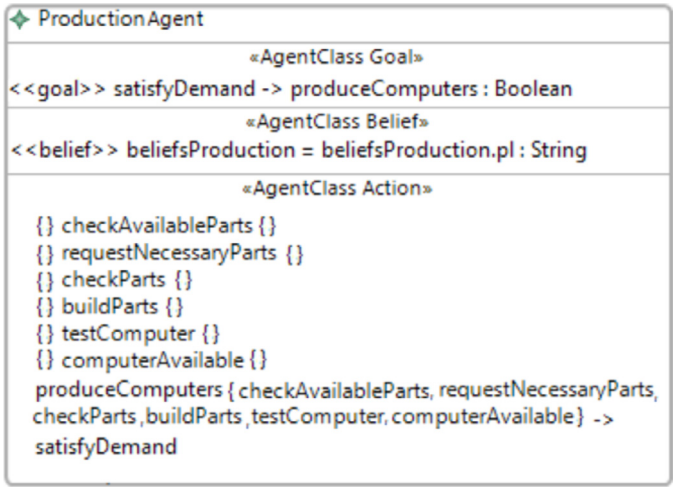

Fig. 58. The ProductionAgent modelled in MAS-ML.

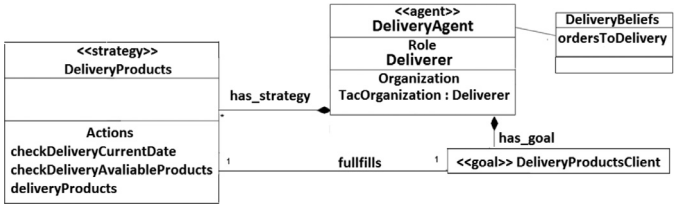

Fig. 60. The DeliveryAgent modelled in Class Diagram of AUML.

Fig. 66 shows the modelling of BuyerAgent in AML Class Diagram, where its beliefs (deliveryBeliefs), and behaviour fragments RequestQuotation, and Pay and Bid can be identified. The leg actuator and the perceiver eye are also available. However, the next function cannot be represented in AML and beliefs, which are structural features, are represented by particular classes, which enable have operations.

The SellerAgent modelled in AML Class Diagram is shown in Fig. 67, where the behaviour fragments offerProduct, ReceivePayment and mountDeliveryProduct can be identified. The leg actuator and the perceiver eye are also available . This modelling is appropriate to the Simple Reflex Agents.

The ProductionAgent modelled in AML Class Diagram is shown in Fig. 68. The beliefs of the agent are represented by BeliefsProduction and its goal is represented by satisfyDemand. This agent has eye as perceiver, legs as actuators and the following behaviour fragments: checkAvailableParts, requestNecessaryParts, checkParts, buildParts, computerAvailable, TestComputer, produceComputers, nextFunctionProduction, goalFuncProduction and probFuncProduction. According to that presented in the case study, this agent should be modelled as a Goal-Based Agent, so the next function, formulate-goal function and formulate-problem function should have been represented. These functions was modeller as behaviour fragments with the respective name, but there is no semantic available to differentiate them. It was also not possible to represent the planning of this agent.

The ManagerAgent modelled in AML Class Diagram is shown in Fig. 69. The beliefs of the agent are represented by BeliefsManager, and objectives are represented by maximizeSale and MaximizeGain. The ManagerAgent has the following behaviour fragments: requestBuyingPrice, requestSellingComputer, requestSellingPrice, estimateBuyingPrice, estimateSellingPrice, managerPlanning, nextFuncManager, formGoalManager, formProbManager and utilityFunctionManager. According to that presented in the case study, this agent should be modelled as a Utility-Based Agent, so the next function, formulate-goal function and formulate-problem function should be represented. Although some fragments have been defined with names of these functions, there is no semantic available in AML to differentiate them. It was also not possible to represent the planning of this agent.

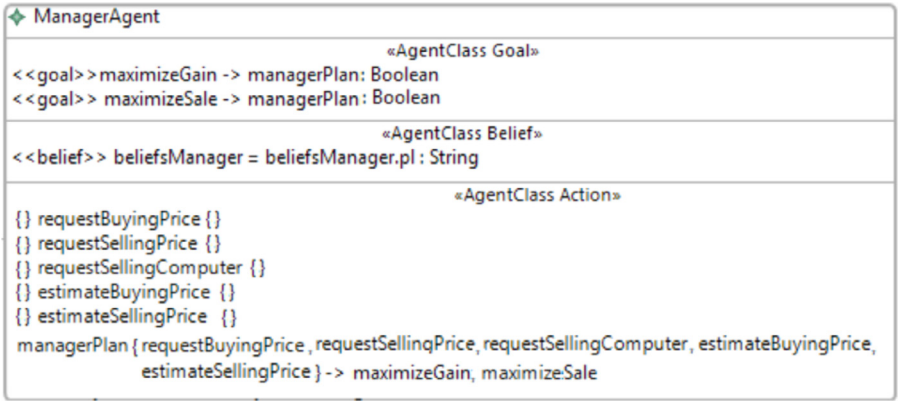

Fig. 59. The ManagerAgent modelled in MAS-ML.

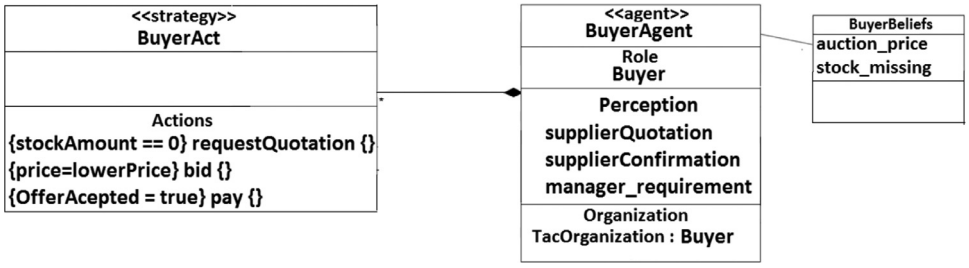

Fig. 61. The BuyerAgent modelled in Class Diagram of AUML.

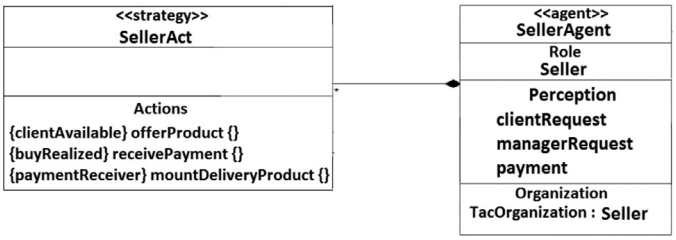

Fig. 62. The SellerAgent modelled in Class Diagram of AUML.

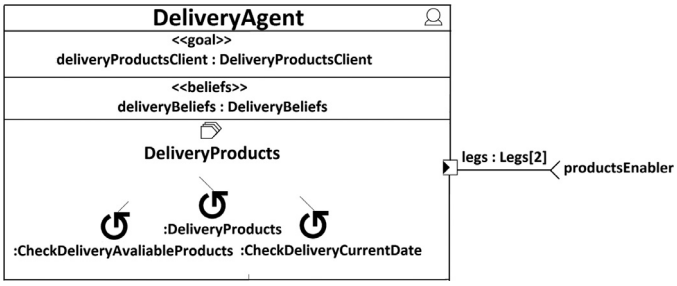

Fig. 65. The DeliveryAgent modelled in AML Class Diagram.

7.4. Discussion

In the above sections we have used four modelling languages to model TAC-SCM. In this section we highlight the structural and behavioural features that can and cannot be modelled in each modelling language. Our intention is to demonstrate that only MAS-ML 2.0 is able to model the internal agent architectures described in this paper, because it is the only language able to model all structural and behavioural features covered by these architectures. Table 3 shows

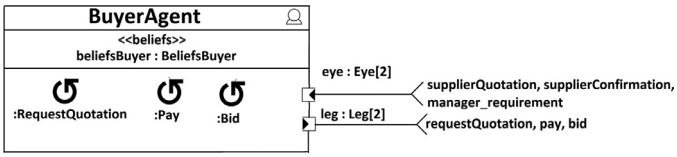

Fig. 66. The BuyerAgent modelled in AML Class Diagram.

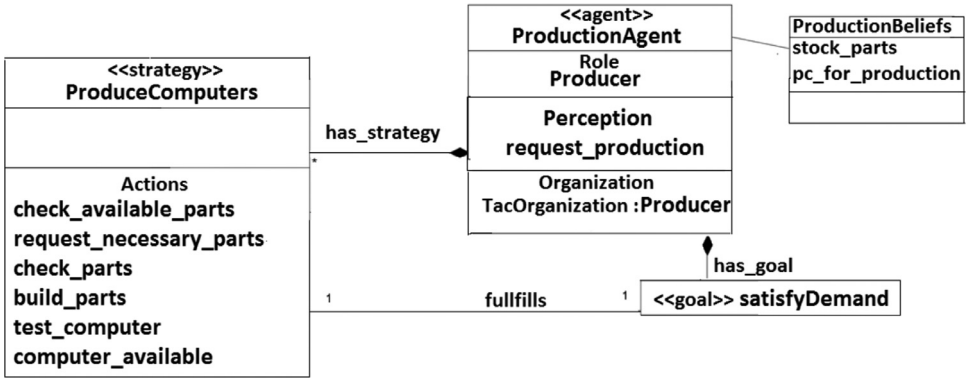

Fig. 63. The ProductionAgent modelled in Class Diagram of AUML.

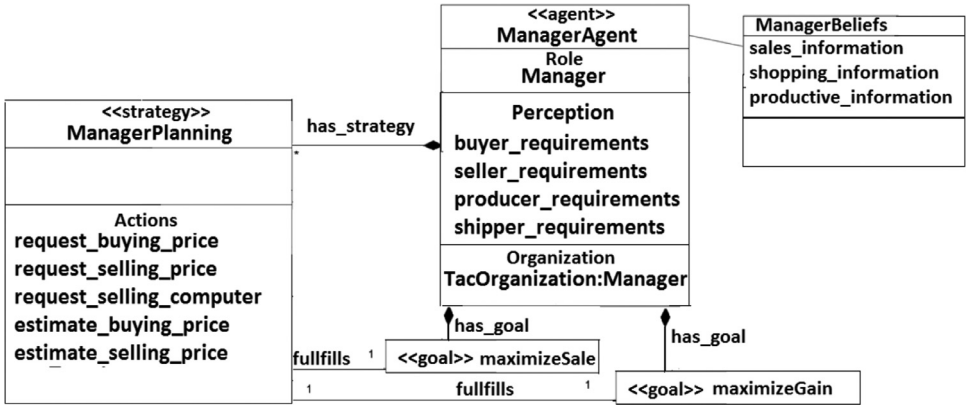

Fig. 64. The ManagerAgent modelled in Class Diagram of AUML.

Table 3
Features covered by the modelling languages.

	Structural features		Behavioural features							
	Belief	Goal	Action	Plan	Perception	Planning	Next function	Problem-formulation function	Goal-formulation function	Utility function
AUML	X	X	X	X	X	–	–	–	–	–
MAS-ML	X	X	X	X	–	–	–	–	–	–
AML	X	X	X	X	X	–	–	–	–	–
MAS-ML 2.0	X	X	X	X	X	X	X	X	X	X

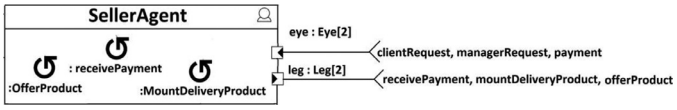

Fig. 67. The SellerAgent modelled in AML Class Diagram.

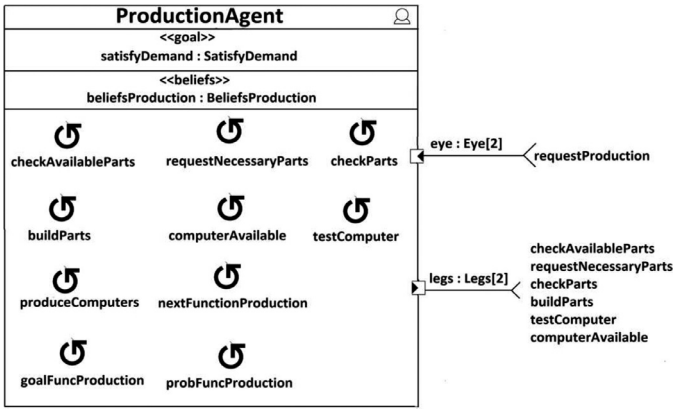

Fig. 68. The ProductionAgent modelled in AML Class Diagram.

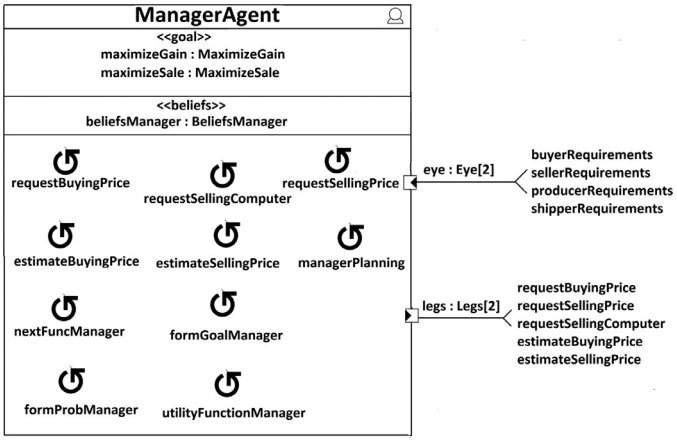

Fig. 69. The ManagerAgent modelled in AML Class Diagram.

that all structural features required by the architectures are modelled by the four modelling languages. The behavioural features of action, plan and perception are modelled by most of them. This indicates that the internal agent architecture *Simple Reflex Agents* and *MAS-ML agents*can only be modelled by using MAS-ML 2.0. can be modelled by AUML, AML and MAS-ML 2.0. These architectures cannot be modelled by MAS-ML because it is not able to model perception. However, only MAS-ML 2.0 is able to model planning, next function, problem-formulation function, goal-formulation function and utility function. Thus, the internal agent architectures Model-Based Reflex Agents, Goal-Based Agents and Utility-Based Agents

8. Conclusions and future work

Similar to object-oriented development, modelling is an essential activity for the development of MAS. Therefore, agents need to be modelled consistently with its implementation. According to Russell and Norvig (2003) the environment can be classified as deterministic or stochastic, fully observable or partially observable, static or dynamic, continuous or discrete, episodic or sequential, and a specific agent architecture can be chosen depending on the type of environment that the multi-agent system will perform. Russell and Norvig (2003) classify agents into four types of internal architectures: Simple Reflex Agents, Model-Based Reflex Agents, Goal-Based Agents and Utility-Based Agents.

Depending on the type of task that the agent will perform, an architecture may be more appropriate than another; for example, agents that have conflicting goals or goals that may not be fully achieved should be implemented as Utility-Based Agents, so they can use the utility function to assist in decision-making in relation to their goals' process. Reflex agents are suitable for situations that need quick responses to continuous changes in their environment. However, in a multi-agent system there is often a combination of agents each having a different architecture, such as in the statement of use given in this article.

There are several MAS modelling languages, however, none of them fully supports the modelling of different agent architectures proposed by Russell and Norvig (2003),. Therefore, a Simple Reflex Agent, Model-Based Reflex Agent, Goal-Based Agent and Utility-Based agent must be modelled with specific characteristics that it should possess internally. This paper presented an extension to the TAO metamodel and MAS-ML language in order to allow the modelling of diverse internal agent architectures published in the agent literature.

TAO and MAS-ML were originally designed to support the modelling of proactive Goal-Based Agents with plan. Thus, some issues were detected while trying to use the language to model reactive agents and other proactive architectures. In this sense, the TAO evolution incorporates new concepts related to agent, and the MAS-ML evolution proposed in this work involves the definition of two new metaclasses, *AgentPerceptionFunction* and *AgentPlanningStrategy*, in order to aggregate the representation of different agent behaviour. Also, new stereotypes to describe the behaviour of agents of specific architectures were defined and associated to *AgentAction* metaclass. The static structures of *AgentClass* and *AgentRoleClass* entities were also modified.

Then, the class, organization, role, sequence and Activity Diagrams were changed in consistency. The static diagrams were changed to model five types of agents, the Goal-Based Agents with plan were maintained and the Simple Reflex Agents, Model-Based Reflex Agents, Goal-Based Agents and Utility-Based Agents were created, each one with the respective elements. The initial representation of the agent roles was maintained, and two new representations were provided respectively associated with Simple Reflex Agents and Model-Based Reflex Agents. Dynamic diagrams were evolved with a new representation for each type of agent.

A modelling tool was created to support the new version of the language MAS-ML; it was named MAS-ML tool. It is an environment domain specific to modelling MAS, implemented as an Eclipse plug-in platform. MAS-ML tool supports the modelling of static and dynamic diagrams according to MAS-ML 2.0 and it allows perform modelling, validating the models created and persist models. OCL rules were established to enable model checking of diagrams created with this tool and to provide to designers the possibility of analysing the well formation of their diagrams.

Finally, a case study was presented with the modelling of an MAS for the TAC-SCM. Agents with different internal architectures were modelled by using four modelling languages: MAS-ML 2.0, original MAS-ML, AUML and AML. The architectures were chosen according to the activities to be performed by the agent. As it is demonstrated in Section 7.3 and resumed in Section 7.4, AUML and AML are able to model only two internal agent architectures and original MAS-ML are able to model only one agent architecture. Therefore, they are not adequate for modelling problems such as TAC-SCM where there is a need to model more complex agents that should, for instance, be able to plan, to formulate problems and goals and to evaluate their utilities. The only modelling language able to model such characteristics is MAS-ML 2.0.

As future works, other case studies are being conducted to provide further validation for this work, and new experimental studies will be performed to evaluate the technique proposed. Moreover, the possibility of TAO and MAS-ML extensions for other internal architectures is a possible work. Related to the MAS-ML tool, some improvements can be made on the graphical representation of builders to represent with more faithfulness the proposed representation in the MAS-ML metamodel. The code generation for a framework is an interesting possibility.

References

Argente, E., Beydoun, G., Fuentes-Fernandez, R., Henderson-Sellers, B., Low, G., 2009. Modelling with agents. In: 10th International Workshop on Agent-Oriented Software Engineering.

Arunachalam, R, 2004. The 2003 supply chain management trading agent competition. In: Third International Joint Conference on Autonomous Agents & Multi Agent Systems, pp. 113–120. July 2004 [S.l.: s.n.].

Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J.J., Pavon, J., Gonzalez-Perez, C., 2009. FAML: a generic metamodel for MAS development. IEEE Trans. Softw. Eng. 35 (6), 841–863.

Bordini, R.H., Hubner, J.F., Wooldridge, M., 2007. Programming Multi-Agent Systems in AgentSpeak Using Jason, vol. 1. John Wiley & Sons, p. 273.

Braubach, L., Pokahr, A., Lamersdorf, W., 2004. Jadex: a short overview. In: Net. ObjectDays 2004: AgentExpo.

Cervenka, R., 2012. Modeling multi-agent systems with AML. Software Agents, Agent Systems and Their Applications, pp. 9–27.

Castro, J., Alencar, F., Silva, C., 2006. Agent-oriented software engineering. In: Breitman, K., Anido, R. (Eds.), Updates in Computers. PUC-Rio, Rio de Janeiro, pp. 245–282 (Org.).

Centeno, R., Hermoso, R., Billhardt, H., Ossowski, S., 2008. Towards a Conceptual Framework for Organizational Mechanisms in Multiagent Systems. In: Workshop On Agreement Technologies in 8th Ibero-American Conference on Artificial Intelligence. Lisboa (Portugal). Lisboa (Portugal).

Choren, R., Lucena, C., 2005. The Anote Modeling Language for Agent-Oriented Specification, Software Engineering for Multi-Agent Systems III. Springer-Verlag, Berlin, Heidelberg, pp. 198–212.

Collins, J., Arunachalam, R., Sadeh, N.; Eriksson, J.; Finne, N.; Janson, S., 2006. The Supply Chain Management Game for the 2007 Trading Agent Competition. Available in http://www.sics.se/tac/tac07scmspec.pdf

Czarnecki, K., Eisenecker, U., 2000. Generative Programming – Methods, Tools, and Applications. Addison-Wesley June 2000.

De Maria, B.A., Silva, V.T., Lucena, C.J.P., 2005. Developing multi-agent systems based on MDA. In: 17th Conference on Advanced Information Systems Engineering. Porto. Porto.

Dorigo, M., Stützle, T., 2004. Ant Colony Optimization. The MIT Press, Cambridge, Massachusetts.

D'Inverno, M., Luck, M., 2001. Understanding Agent Systems. Springer, New York.

Eclipse Platform, 2013. Available in www.eclipse.org , Accessed in 2013 January 10.

Emfatic, 2014. Available in: http://eclipse.org/emfatic/, Accessed in July of 2014.

France, R., Rumpe, B., 2007. Model-driven development of complex software: a research roadmap, future of software engineering. In: ICSE'07. EUA. EUA.

GMF 2011. Available in www.eclipse.org/modelling/gmf/, Accessed 2013 January 10.

Guedes, G.T.A., Vicari, R.M., 2009. Applying AUML and UML 2 in the multi-agent systems project. Advances in Conceptual Modeling – Challenging Perspectives Lecture Notes in Computer Science 5833, 106–115.

Jennings, N.R., 1996. Coordination techniques for distributed artificial intelligence. Foundations of Distributed Artificial Intelligence. Wiley, pp. 187–210.

Lavinal, E., Desprats, T., Raynaud, Y., 2006. A generic multi-agent conceptual framework towards self-management. In: Network Operations and Management Symposium, NOMS 2006, 10th IEEE/IFIP.

OCL, 2011. available in: http://www.eclipse.org/modeling/mdt/?project=ocl, Accessed in 2011 June 20.

Odell, J., Parunak, H.V.D., Bauer, B., 2000. Extending UML for Agents. In: Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence (AIII'00), pp. 3–17.

OMG. 2014a. Object Management Group, 2014a Available in: http://www.omg.org. Last Accessed on: 06 November 2014.

OMG, 2014b MDA success stories. Available in: http://www.omg.org/mda/products_success.htm , Last Accessed on: 9 November 2014.

Russell, S., Norvig, P., 2003. Artificial Intelligence: A Modern Approach, 2nd ed. Prentice Hall, Upper Saddle River, NJ ISBN 0-13-790395-2.

Sadeh, N., Arunachalam, R., Erikson, J., Finne, N., Janson, S., 2003. A supply-chain trading competition. AI Mag. 24 (1), 92–94.

Shirazi, M.R.A., Barfouroush, A.A., 2008. A conceptual framework for modeling automated negotiations in multiagent systems. Negotiation J. 24 (1), 45–70.

Silva, P.S., Mendes, M.J., 2003. An approach to incorporate mechanisms of artificial intelligence in mobile agents. In: XXI Brazilian Symposium of Computers Network. Natal, Rio Grande do Norte,Brazil, pp. 837–852.

Silva, V.T.da 2004. From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling. PhD Thesis, PUC, Computer Science Department, Rio de Janeiro.

Silva, V., Garcia, A., Brandao, A., Chavez, C., Lucena, C., Alencar, P., 2003. Taming agents and objects in software engineering. In: Garcia, A., Lucena, C., Zamboneli, F., Omicini, A., Castro, J. (Eds.), Software Engineering for Large-Scale Multi-Agent Systems, LNCS, vol. 2603. Springer-Verlag, pp. 1–26 LNCS 2603ISBN 978-3-540-08772-4.

Silva, V., Lucena, C., 2004. From a conceptual framework for agents and objects to a multi-agent system modelling language. J. Autonomous Agents Multi-Agent Syst. 9 (1-2), 145–189 Kluwer Academic Publishers 2004.

Silva, V., Choren, R., Lucena, C., 2004. A UML based approach for modelling and implementing multi-agent systems. In: Proceeding of the third International Conference on Autonomous Agents and Multi-Agents Systems, 2. IEEE Computer Society, New York, USA, pp. 914–921.

Silva, V.T.da, Choren, R., Lucena, C.J.P.de, 2005. Using UML 2.0 activity diagram to model agent plans and actions. In: 4th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS). Netherlands, pp. 594–600.

Silva, V., Choren, R., Lucena, C., 2008a. MAS-ML: a multi-agent system modelling language. Int. J. Agent-Oriented Softw. Eng. 2 (4), 382–421.

Silva, V., Choren, R., Lucena, C., 2008b. Modelling MAS properties with MAS-ML dynamic diagrams. In: 8th International Bi-Conference Workshop, LNCS, vol. 4898. Springer-Verlag, pp. 1–18.

Spanoudakis, N., Moraitis, P., 2008. The agent modeling language (AMOLA). In: in 13th International Conference on Artificial Intelligence: Methodology, Systems, and Application. Springer-Verlag, Berlin, pp. 32–44.

UML: Unified Modelling Language Specification, version 2.2, OMG, 2009 available in: http://www.omg.org/technology/documents/modelling_spec_catalog.htm#UML.

Wagner, G., Taveter, K., 2005. Towards radical agent-oriented software engineering processes based on AOR modeling. In: Henderson-Sellers, B., Giorgini, P. (Eds.), Agent-Oriented Methodologies. Idea Group Publishing, pp. 277–316.

Wagner, G., 2003. The agent-object-relationship meta-model: towards a unified view of state and behaviour. Inf. Syst. 28 (5), 475–504.

Weiss, G., 1999. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, Massachusetts.

Wellman, M.P., Stone, P., Greenwald, A., Wurman, P.R., 2002. The 2001 trading agent competition. IEEE Internet Comput. 13, 935–941.

Yu, L., Schmid, B., 2001. A conceptual framework for agent-oriented and role-based work on modelling. In: Wagner, G., Yu, E. (Eds.), In: Proceedings of the 1st International Workshop on Agent-Oriented Information Systems.

Zambonelli, F., Jennings, N., Wooldridge, M., 2001. Organizational abstractions for the analysis and design of multi-agent systems. In: Agent-Oriented Software Engineering, LNCS 1957. Springer, Berlin, pp. 127–141.

Enyo was born in Aurora, Brazil and completed B.S. in Computer Science at the State University Acaraú Valley in 2007. He completed his Master in Computer Science at the State University of Ceará in 2009 with Prof. Mariela Cortés. Enyo is currently a Professor at Federal University of Ceará and doctoral student of Federal University of Pernambuco. Enyo's research is focused primarily on the software engineering methods and their application to a wide variety of problems in multi-agent systems, more particularly in the areas of software design of multi-agent systems. Enyo's current research projects include extensions of multi-agent systems modeling languages, developing methods for the model composition of multi-agent systems and development of multi-agent systems to specified domains, as MOODLE.