By VIVIANE TORRES DA SILVA *and* CARLOS J.P. DE LUCENA

Emerging from a paradigm different from objects, the modeling of agents requires its own conceptual frameworks, modeling languages, and methodologies.

MODELING MULTI-AGENT SYSTEMS

Multi-agent systems (MASs) are gaining wide acceptance in both industry and academia as a powerful new paradigm for designing and developing software systems [11]. Along with this growth, new methodologies, modeling languages, development platforms, tools, and programming languages are being proposed. Agent-based systems require techniques to help explore their benefits and special characteristics. However, the various MAS methodologies, languages, and platforms involve distinct and varied sets of abstractions. It is often quite difficult for software engineers to understand the definition of each abstraction and the relationship between any two of them. In this context, there is a need for a conceptual framework that defines the abstractions, along with their relationships and behavior.

With any new software engineering paradigm, successful MAS deployment requires modeling languages, along with other agent-based software technologies, to explore the use of agent-related abstractions and promote traceability, from design model to code. Modeling languages must represent the structural (or static) and dynamic aspects of MASs by expressing the characteristics of their essential entities (such as agent, role, organization, and environment). Structural aspects are the definitions of the entities, as well as their properties and relationships. The dynamic aspects are related to the behavior of the entities [9].

To reduce risk when adopting a new technology it is convenient to present it as an incremental extension of known and trusted methods and provide explicit engineering tools that support industry-accepted standard methods of technology deployment [4]. A modeling language for a MAS should be an incremental extension of a known and trusted standard modeling language.

Since agents and objects coexist in MASs, the UML modeling language [9] can be used as a basis for developing MAS modeling languages. The UML modeling language is a de facto standard for object-oriented modeling. UML is used in industry and academia for modeling object-oriented systems. Nevertheless, the original form of UML (version 2.0) provides insufficient support for modeling MASs. The UML metamodel lacks support for modeling agents, organizations, and agent roles.

After an exhaustive review of theories, methodologies, and methods for MASs, we saw the need for a conceptual framework to define the commonly used abstractions found in MASs. The few conceptual frameworks proposed in the literature for describing MAS concepts [2, 12] do not define a number of structural and dynamic aspects (such as role and the ability to play it) commonly described in MASs.

A variety of agent-based techniques describe MASs based on different types of entities. Each technique describes a different set of properties and associates different relationships with each entity. Thus, there is a need to define the structural aspects of MASs by describing the properties and behavior of the entities frequently found in these systems. When describing the entities, the properties associated with them and their relationships must be defined. The relationships between properties must also be described.

The dynamic aspects are characterized by the internal execution of the entities (intra-actions) and by the interactions between entities. Different entities may execute and interact in different ways. Since MASs are composed of different entities, their dynamic aspects must be described.

The intra-actions of an entity are related to the behavioral properties the entity defines. For instance, the intra-actions of objects are related to the execution of methods, and the intra-actions of agents are related to the execution of actions and plans. The interactions between any two entities are influenced by the relationships linking them. Although agents interact by sending and receiving messages, the sequence and content of messages sent and received by agents are influenced by their relationships. Therefore, there is a need for describing the interactions between the entities based on the relationships that link them.

MAS-ML EXTENDS UML BY **PRESERVING ALL OBJECT-RELATED CONCEPTS IN THE UML METAMODEL,** EVEN AS IT INCLUDES THE AGENT-RELATED CONCEPTS IN TAO.

MAS Modeling Languages

Several proposed modeling languages for MASs extend the UML metamodel [1, 4, 10]. However, a modeling language should still be able to do the following: describe agent-related concepts as first-class abstractions; be based on an explicit description of a MAS metamodel; model the structural and dynamic aspects frequently described in MASs; and provide traceability from design model to code.

MAS modeling languages should be able to define MAS entities as first-class abstractions. All proposed modeling languages describe agents as first-class abstractions. However, entities (such as role, organization, and environment) are not defined as such in many of them. Due to this limitation, these languages cannot be used to model a number of structural and dynamic aspects of MASs (such as agents playing roles in different organizations). It is also not possible to model the relationships and interactions between agents, objects, and other MAS entities.

A metamodel defines a language for specifying models by describing the semantics of a set of abstractions and defining how these abstractions are instantiated [9]. The metamodel describes the semantics of

each abstraction, the metarelationships with other abstractions, and the graphical representation of the abstraction in models.

Several proposed modeling languages that extend UML do not clearly describe the extensions applied to the UML metamodel. Although they describe extensions to UML diagrams, such languages usually do not describe how the UML metamodel was extended in order to model new abstractions. The modeling languages describe the graphical representation of the new abstractions but do not clearly describe their semantics or the relationships among them.

ENTITIES		*Object Class*	*Agent Class*	*Organization Class*	*Agent Role Class*	*Object Role Class*	*Environment Class*
RELATIONSHIPS	*Object Class*	specialization association aggregation dependency	association	association	association	association play	inhabit
	Agent Class	association	specialization aggregation dependency	---	play	---	inhabit
	Organization Class	association	---	specialization	ownership play	ownership	inhabit
	Agent Role Class	association	play	ownership play	specialization control association aggregation dependency	dependency association	---
	Object Role Class	association play	---	ownership	dependency association	specialization association aggregation dependency	---
	Environment Class	inhabit	inhabit	inhabit	---	---	specialization association
PROPERTIES		attribute method	goal belief action plan	goal belief action plan axiom	goal belief duty right protocol	attribute method	attribute method **or** goal belief action plan

Table 1. TAO entities, properties, and relationships.

The modeling languages [4, 10] that describe the extensions applied to the UML metamodel use stereotypes based on the metaclass Class (representing object classes) to define agents. Since agents and objects do not share properties and relationships, agents should not be described based on objects.

A MAS modeling language should describe structural diagrams to model the structural aspects of MASs. The set of structural diagrams must be capable of modeling: the entities usually defined in MASs; the properties of these entities by associating the properties with the entities; and the relationships between the entities. The modeling languages proposed in the literature do not model a number of MAS entities (such as role, organization, and environment) so do not define the relationships between agents and these entities.

In order to model MAS entities, properties, and relationships, UML structural diagrams must be extended. Different diagram elements[1] can be created to represent MAS entities, properties, and relationships. Different diagram elements facilitate the visualization and modeling of these abstractions. If the modeling language defines more than one structural diagram, each diagram must then describe the set of entities, properties, and relationships that can be modeled. It is also important to specify if the diagrams define different views of the same abstractions or model different sets of abstractions.

MAS modeling languages must define dynamic MAS diagrams to model the dynamic aspects of these systems. The dynamic diagrams must be capable of modeling the interactions between the entities defined in the structural diagrams, as well as the internal execution of the entities. MAS dynamic diagrams can be defined by extending UML dynamic diagrams while defining the interactions and intra-actions of MAS entity instances.

Different types of interactions must be modeled in MAS dynamic diagrams. The different entities in MASs interact in different ways. The MAS dynamic diagrams must also model the internal behavior of the MAS entities. Moreover, different diagram elements must be created to represent the MAS entity instances.

A number of proposed modeling languages do not represent the different types of interactions related to objects and agent-related abstractions. Moreover, many of them also do not model the internal execution of the agent-related abstractions.

Developing ways to implement agent-based systems is a key issue in getting agent technology into the software development mainstream. In order to implement MASs designed through a MAS modeling language, MAS design models must be transformed into code. MAS design models are high-level models consisting of agent-related abstractions. To transform MAS models into code, agent-related abstractions must be mapped into abstractions defined in the programming language.

TAO Conceptual Framework

The goal of the Taming Agents and Objects, or TAO, conceptual framework [8] is to define a core

[1] Diagram elements help to graphically represent abstractions in diagrams.

set of MAS abstractions developed based on our investigation of existing agent-based and object-oriented methodologies, languages, and theories. TAO combines the abstractions frequently described in the literature on MASs. The benefit of having such a framework is to provide support for developing new methodologies, methods, and languages based on the concepts defined and related in the framework. Each concept is viewed as a candidate abstraction for modeling languages, methodologies, and support environments to be applied in different phases of MAS development.

	Primitive Dynamic Processes	High-Level Dynamic Processes
Object	creation destruction	---
Agent	creation destruction	entering an organization leaving an organization moving from one environment to another
Organization	creation destruction	entering an organization leaving an organization moving from one environment to another
Agent Role	creation destruction	---
Object Role	creation destruction	---
Environment	creation destruction	---

Table 2. TAO primitive and high-level dynamic processes.

TAO defines the structural and dynamic aspects of MASs, as well as the entities that may be described in MASs, along with their properties and the relationships associated with them (see Table 1). The dynamic aspects described in TAO are classified in primitive dynamic processes and high-level dynamic processes (see Table 2). The primitive dynamic processes describe the creation and destruction of entities. High-level dynamic processes are more complex domain-independent behavior described based on primitive dynamic processes. Domain-independent high-level dynamic processes describe patterns of behavior derived from the characteristics of the inhabit (environment), ownership, and play relationships between MAS entities. These relationships are domain-independent in several ways: agents, organizations, and objects inhabit environments; agents and suborganizations play at least one role; and every role is owned by an organization.

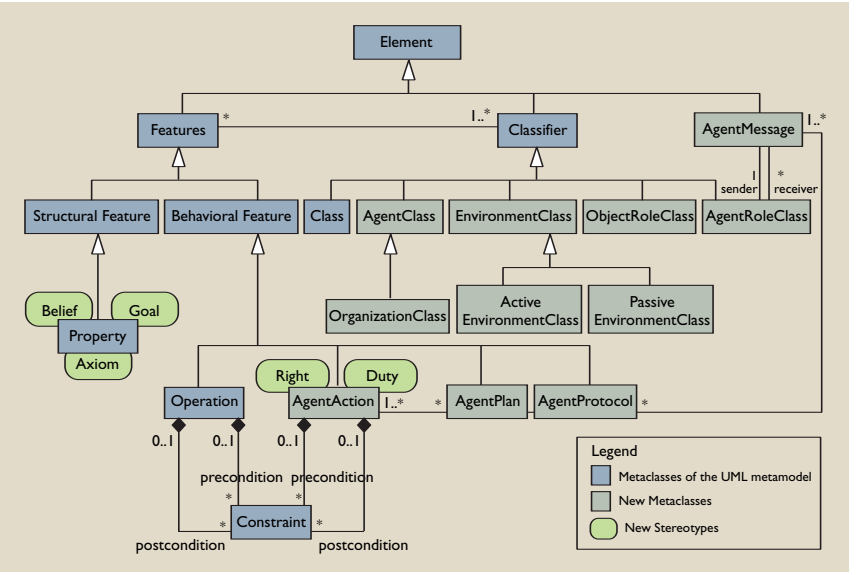

Figure 1. The extended UML metamodel incorporating MAS-ML entities and properties (some relationships omitted).

The MAS-ML Modeling Language

MAS-ML [5–7] aims to model all structural and dynamic aspects of the entities defined in TAO. The MAS-ML metamodel is defined by extending the UML metamodel according to the concepts defined in TAO. When extending UML according to TAO concepts, it is not possible to use only the tag, constraints, and stereotype-extension mechanisms provided by UML. New metaclasses and stereotypes associated with new entities, properties, and relationships defined in TAO but not presented in UML are incorporated into the UML metamodel. Since we aim to produce a conservative extension of UML, metaclasses defined in UML are not modified during the extension. Figure 1 outlines a subset of metaclasses of the UML metamodel and the extensions made by MAS-ML, including the new metaclasses and stereotypes proposed by MAS-ML related to the entities and properties described in TAO. The icons representing the stereotypes are associated with the metaclasses on which the stereotypes are based. Figure 2 outlines the new metaclasses related to the relationships described in TAO that have been proposed by MAS-ML to the UML metamodel.

Due to the set of entities and relationships defined in the TAO metamodel incorporated into the UML metamodel, we have proposed a new set of structural

diagrams to focus on the extension aspects to be covered by the resulting MAS-ML. The structural diagrams in MAS-ML are the extended UML class diagram and two new diagrams: organization and role. MAS-ML extends the UML class diagram to represent the structural relationships between agents, agents and classes, organizations, organizations and classes, environments, and environments and classes. The organization diagram models system organizations and the relationships between them and other system entities. The role diagram models the relationships between the roles defined in organizations. The three structural diagrams—class, organization, and role diagrams—model all entities and relationships defined in TAO.

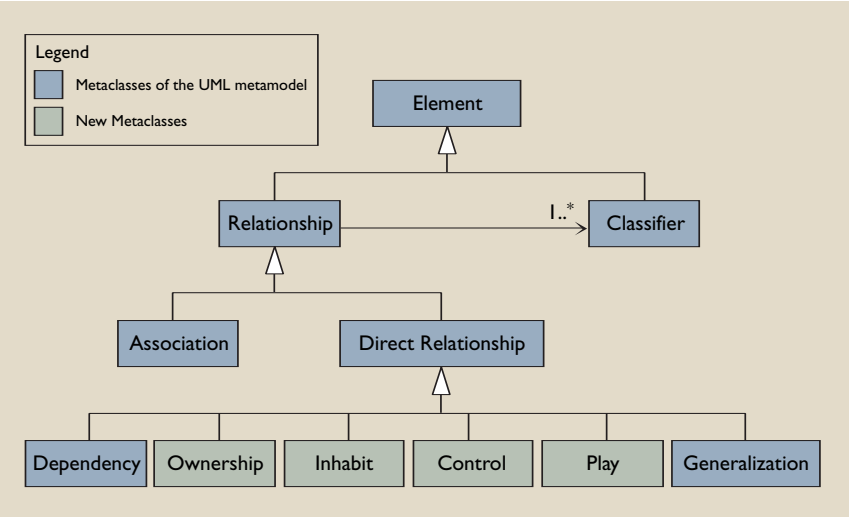

Figure 2. The extended UML metamodel incorporating MAS-ML relationships.

We further aim to extend the UML sequence and activity diagrams to represent the dynamic aspects of MASs. These diagrams make it possible to model interactions between agents, organizations, environments, and objects; execution of plans and actions associated with agents, organizations, and active environments; and protocols defined by roles. The entities modeled in the sequence diagrams are instances of the entity classes modeled in the structural diagrams. The methods, plans, and actions modeled in the sequence and activity diagrams are also defined in the structural diagrams associated with the respective entity classes.

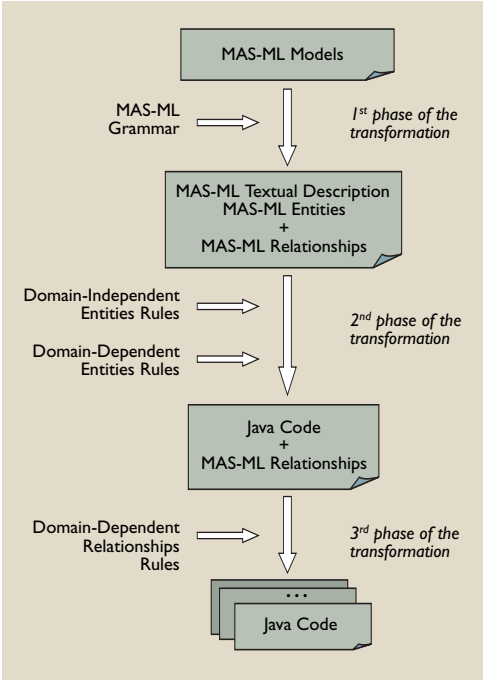

Figure 3. Transformation phases of the development process.

Since a sequence or activity diagram can model different entities executing different plans and actions at the same time, as well as the same entity playing more than one role, any of these diagrams can express concurrency and parallelism in the design models. Moreover, since a sequence or activity diagram can model different entities executing in different environments, as well as entities moving from one environment to another, any of these diagrams can express distribution in the design models. For representation of both the structural and dynamic diagrams and their use in modeling concrete MAS applications, see [5–7].

With the aim of implementing systems modeled through MAS-ML, a transformer was developed to generate code from the MAS-ML structural diagrams. The models described at the agent level of abstraction are automatically transformed into object-oriented code.

The transformation process consists of three phases (see Figure 3). In the first, the MAS-ML graphical models of the application are described textually through MAS-ML grammar. Using that grammar makes it possible to describe all information presented in MAS-ML structural diagrams, including entities and their properties and the relationships between them.

In the second, domain-independent-entity rules and domain-dependent-entity rules are applied to the textual description of the MAS-ML models to generate a partial transformation. The domain-independent-entity rules generate the set of domain-independent object-oriented modules and their relationships defined by an object-oriented abstract architecture. The set of classes in the architecture represents the entities that cannot be mapped directly from MAS-ML to Java classes. Entities (such as agents, organizations, and

agent roles) cannot be mapped directly into classes, because classes are defined based on attributes and methods. The entities are instead defined based on such properties as goals, actions, and protocols. The domain-dependent entities rules generate classes that represent the application entities; the classes extend the abstract classes defined in the architecture.

Layers	Models
Metamodel layer	UML metamodel, TAO metamodel, MAS-ML metamodel
Domain-model layer	MAS-ML models
Instance layer	Instances of the domain models

Table 3. MOF metadata architecture.

Finally, the third applies the domain-dependent-relationship rules to the output of the second phase. The classes in the previews phase are modified in order to represent the application relationships. The output of this phase is a set of object-oriented Java classes representing the application being modeled through MAS-ML.

Helping to explain the relationship among UML, TAO, and MAS-ML, we use a four-layer metadata architecture described in the Meta Object Facility (MOF) specification [3], including meta-metamodel, metamodel, domain model layer, and instance. Here, we concentrate on the three lowest layers: metamodel, domain model layer, and instance (see Table 3).

The metamodel layer includes the description of the structure and semantics of metadata. The Object Management Group (www.omg.org) defines the UML metamodel; we define the TAO metamodel (or conceptual framework). The MAS-ML metamodel extends the UML metamodel according to the concepts described in TAO. MAS-ML specifies a modeling language that incorporates both object- and agent-oriented concepts.

The domain model layer depicts data specific to the application domain. The concepts modeled using MAS-ML are instantiated according to the domain information used to create domain models. The instance (information or implementation) layer describes the specific instances of the domain model data.

Conclusion

Each of the MAS-related methodologies, languages, and platforms we've explored in the literature propose distinct and varied sets of abstractions. The TAO framework's main role is to help make clear the abstractions (and their relationships and interactions) when developing large-scale MASs. The proposed framework elicits an ontology connecting consolidated abstractions (such as objects and classes) and frequently used MAS abstractions (such as agents, roles, organizations, and environments) that are the foundations of agent-based and object-based software engineering.

Aiming to define a MAS modeling language that contemplates all the concepts described in TAO, we've proposed the MAS-ML language, extending UML by preserving all object-related concepts in the UML metamodel, even as it includes the agent-related concepts in TAO. Using MAS-ML, it is possible to describe agents, roles, organizations, and environments, as well as model the interactions among these entities and how they execute internally. C

References

1. Depke, R., Heckel, R., and Huster, J. Formal agent-oriented modeling with UML and graph transformation. *Science of Computer Programming 44,* 2 (Aug. 2002), 229–252.
2. D'Inverno, M. and Luck, M. *Understanding Agent Systems.* Springer, New York, 2001.
3. *Meta Object Facility Specification, version 1.4;* www.org/cwm.
4. Odell, J., Parunak, H., and Bauer, B. Extending UML for agents. In *Proceedings of the Agent-Oriented Information Systems Workshop* (Austin, TX, July 2000), 3–17.
5. Silva, V., Choren, R., and Lucena, C. Using the UML 2.0 activity diagram to model agent plans and actions. In *Proceedings of the Fourth International Conference on Autonomous Agents and Multi-Agent Systems* (Utrecht, The Netherlands, July 2005), 594–600.
6. Silva, V., Choren, R., and Lucena, C. A UML-based approach for modeling and implementing multi-agent systems. In *Proceedings of the Third International Conference on Autonomous Agents and Multi-Agent Systems* (New York, July 2004), 914–921.
7. Silva, V. and Lucena, C. From a conceptual framework for agents and objects to a multi-agent system modeling language. *Journal of Autonomous Agents and Multi-Agent Systems 9,* 1–2 (July 2004), 145–189.
8. Silva, V., Garcia, A., Brandao, A., Chavez, C., Lucena, C., and Alencar, P. Taming agents and objects in software engineering. In *Proceedings of Software Engineering for Large-Scale Multi-Agent Systems.* Springer, Berlin, 2003, 1–26.
9. *Unified Modeling Language Specification, version 2.0*; www.omg.org/uml/.
10. Wagner, G. The agent-object-relationship metamodel: Toward a unified view of state and behavior. *Information Systems 28,* 5 (July 2003), 475–504.
11. Wooldridge, M. and Ciancarini, P. Agent-oriented software engineering: The state of the art. In *Proceedings of Agent-Oriented Software Engineering.* Springer, Berlin, 2001, 1–28.
12. Yu, L. and Schmid, B. A conceptual framework for agent-oriented and role-based work on modeling. In *Proceedings of the First International Workshop on Agent-Oriented Information Systems* (Heidelberg, Germany, June 1999).

Viviane Torres da Silva (viviane@fdi.ucm.es) is an associate researcher in the Informatics Systems and Computation Department of Universidad Complutense de Madrid, Spain.

Carlos J.P. de Lucena (lucena@inf.puc-rio.br) is a professor in the Computer Science Department of Universidade Catolica do Rio de Janeiro, Brasil.