Agent Modeling Language (AML): A Comprehensive Approach to Modeling MAS

Ivan Trencansky and Radovan Cervenka
Whitestein Technologies, Panenska 28, 811 03 Bratislava, Slovakia
Tel +421 (2) 5443-5502, Fax +421 (2) 5443-5512
E-mail: {itr,rce}@whitestein.com

Keywords: agent, multi-agent system, modeling language, agent-oriented software engineering

Received: May 6, 2005

The Agent Modeling Language (AML) is a semi-formal visual modeling language for specifying, modeling and documenting systems that incorporate features drawn from multi-agent systems theory. It is specified as an extension to UML 2.0 in accordance with major OMG modeling frameworks (MDA, MOF, UML, and OCL). The ultimate objective of AML is to provide software engineers with a ready-to-use, complete and highly expressive modeling language suitable for the development of commercial software solutions based on multi-agent technologies. This paper presents an overview of AML. The scope of the language, its structure and extensibility mechanisms are discussed, and the core AML modeling constructs and mechanisms are introduced and demonstrated by examples.

Povzetek: Opisana je vizualizacija agentnega jezika za modeliranje.

1 Introduction

The Agent Modeling Language (*AML*) [3, 5, 4] is a semi-formal[1] visual modeling language for specifying, modeling and documenting systems that incorporate concepts drawn from *Multi-Agent Systems* (*MAS*) theory.

The most significant motivation driving the development of AML was the extant need for a ready-to-use, comprehensive, versatile and highly expressive modeling language suitable for the development of commercial software solutions based on multi-agent technologies. To qualify this more precisely, AML was intended to be a language that: (1) is built on proved technical foundations, (2) integrates best practices from agent-oriented software engineering (AOSE) and object-oriented software engineering (OOSE) domains, (3) is well specified and documented, (4) is internally consistent from the conceptual, semantic and syntactic perspectives, (6) is versatile and easy to extend, (7) is independent of any particular theory, software development process or implementation environment, and (8) is supported by Computer-Aided Software Engineering (CASE) tools.

Given these requirements, AML is designed to address the most significant deficiencies with current state-of-the-art and practice in the area of MAS oriented modeling languages, which are often: (1) insufficiently documented and/or specified, or (2) using proprietary and/or non-intuitive modeling constructs, or (3) aimed at modeling only a limited set of MAS aspects, or (4) applicable only to a specific theory, application domain, MAS architecture, or technology, or (5) mutually incompatible, or (6) insufficiently supported by CASE tools.

The objective of this paper is to present the approach applied to specification of AML, and a brief overview of the various modeling constructs AML provides to model MASs. Due to limitations in paper length, a comprehensive description of AML abstract syntax, semantics, and notation is not provided.

The rest of the paper is structured as follows: Section 2 presents the approach applied to specification of AML and the available extensibility mechanisms. Section 3 explains the AML fundamental entities and their features, sections 4, 5, 6, 7 and 8 present an overview of AML approach to modeling different aspects of agents and MASs, like social aspects, different kinds of interactions, capabilities, mobility, and mental attitudes. In the end the conclusions are drawn.

2 The AML Approach

Toward achieving the stated goals and overcoming the deficiencies associated with many existing approaches, AML has been designed as a language, which:

- incorporates and unifies the most significant concepts from the broadest set of existing multi-agent theories and abstract models (e.g. DAI [24], BDI [17], SMART [9]), modeling and specification languages (e.g. AUML [1, 11, 12], TAO [18], OPM/MAS [20], AOR [23], UML [15], OCL [14], OWL [19], UML-based ontology modeling [7], methodologies (e.g. MESSAGE [10], Gaia [25], TROPOS [2], PASSI [6],

[1] The term "semi-formal" implies that the language offers the means to specify systems using a combination of natural language, graphical notation, and formal language specification.

Prometheus [16], MaSE [8]), agent platforms (e.g. Jade, FIPA-OS, Jack, Cougaar) and multi-agent driven applications,

- extends the above with new modeling concepts to account for aspects of multi-agent systems thus far covered insufficiently, inappropriately or not at all,
- assembles them into a consistent framework specified by the AML meta-model (covering abstract syntax and semantics of the language) and notation (covering the concrete syntax), and
- is specified as an extension to UML in accordance with the OMG modeling frameworks (MDA, MOF, UML, and OCL).

2.1 The Language Definition

AML is built upon the Unified Modeling Language (UML) 2.0 Superstructure [15], augmenting it with several new modeling concepts appropriate for capturing the typical features of multi-agent systems (see Fig. 1).

The main advantages of this approach are:

- Reuse of well-defined, well-founded, and commonly used concepts of UML.
- Use of existing mechanisms for specifying and extending UML-based languages (metamodel extensions and UML profiles).
- Ease of incorporation into existing UML-based CASE tools.

The abstract syntax, semantics and notation of the language are defined at the *AML Metamodel and Notation* level. The *AML Metamodel* is further structured into two main packages: *AML Kernel* and *UML Extension for AML*.

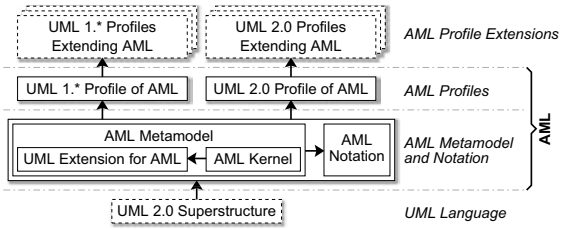

Figure 1: Levels of AML definition

The *AML Kernel* is a conservative[2] extension of UML 2.0, comprising specification of all the AML modeling elements. It is logically structured into several packages, each of which contains specification of modeling elements dedicated for modeling specific aspect of MAS.

The *UML Extension for AML* package adds some meta-properties and structural constraints to the standard UML elements. It is thus a non-conservative extension of UML, and therefore an optional part of the language. However, the extensions contained within are simple and can be easily implemented in most existing UML-based CASE tools.

Upon the AML Metamodel and Notation two UML profiles of AML are specified: *UML 1.* Profile for AML* (based on UML 1.*) and *UML 2.0 Profile for AML* (based on UML 2.0). The primary objective of these profiles is to enable implementation of AML into existing UML 1.* and UML 2.0 based CASE tools, respectively.

2.2 Extensibility of AML

AML is designed to encompass a broad set of relevant theories and modeling approaches, it being essentially impossible to cover all inclusively. In those cases where AML is insufficient, several mechanisms can be used to extend or customize it as required:

- *Metamodel extension* offers first-class extensibility (as defined by MOF [13]) of the AML metamodel and notation.
- *AML profile extension* offers the possibility to adapt AML for a given domain, platform or development method by means of UML Profiles, without the need to modify the underlying AML Metamodel and Notation.
- *Concrete model extension* allows to employ alternative MAS modeling approaches as complementary specifications to the AML model.

3 Modeling MAS Entities

In general, *entities* are objects that can exist independently of others. In order to maximize reuse and comprehensibility of the metamodel AML defines several auxiliary abstract metamodeling concepts called *semi-entities* and their types. Semi-entity types are specialized UML classes used to specify coherent set of features, logically grouped according to particular aspects of MASs. They are used to specify features of other types of modeling elements.

3.1 AML Semi-entities

AML defines the following semi-entities:

Behaviored semi-entities represent elements, which can own capabilities, observe and/or effect their environment by means of perceptors and effectors, provide and use services, and can be (de)composed into behavior fragments.

Socialized semi-entities represent elements, which can form societies, can participate in social relationships and can own social properties.

Mental semi-entities represent elements which can be characterized in terms of their mental attitudes, e.g. which information they believe in, what are their objectives,

[2] A conservative extension of UML is an extension of UML which retains the standard UML semantics in unaltered form [22].

needs, motivations, desires, what goal(s) they are committed to, when and how a particular goal is to be achieved, which plan to execute, etc.

3.2 AML Fundamental Entities

The fundamental entities that compose MASs are: agents, resources, and environments. AML therefore defines three modeling concepts, which can be used to model the above mentioned fundamental entities at both type and instance levels:

Agent type is used to specify the type of agents, i.e. self contained entities that are capable of interactions, observations and autonomous behavior within their environment.

Resource type is used to model the type of resources within the system, i.e. physical or informational entities with which the main concern is their availability (in terms of its quantity, access rights, conditions of usage/consumption, etc.).

Environment type is used to model the type of a system's inner environment[3], i.e. the logical or physical surroundings of entities which provide conditions under which the entities exist and function.

In AML, all the aforementioned entity types are specialized UML classes, and thus can utilize all the features defined for UML classes, i.e. can be instantiated, can own structural and behavioral features, behaviors, can be structured into parts and ports, participate in interactions, can participate in various kinds of relationships (e.g. associations, generalizations, dependencies), etc. The instances of the entity types (called entities) can be modeled by means of UML instance specifications classified according to the corresponding types.

Furthermore, all the AML fundamental entity types inherit features of behaviored semi-entities, and in addition to these, agent and environment types are also socialized and mental semi-entities.

Fig. 2 shows an example of a definition of an abstract class `3DObject` that represents spatial objects, characterized by shape and position, existing inside a containing space. An abstract environment type `3DSpace` represents a three dimensional space. This is a special `3DObject` and as such can contain other spatial objects. `3DSpace` provides a service `Motion` to the objects contained within (for details about services see Sect. 5.4). Three concrete `3DObject`s, an agent type `Person`, a resource type `Ball` and a class `Goal` are defined as specialized `3DObject`s. `3DSpace` is further specialized into a concrete environment type `Pitch` representing a soccer pitch containing two goals and a ball.

[3]*Inner environment* is that part of an entity's environment that is contained within the boundaries of the system.

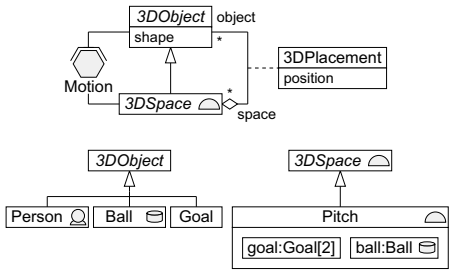

Figure 2: Example of entities, their relationships, service provision and usage

4 Modeling Social Aspects

MASs are commonly perceived as systems comprised of a number of autonomous agents, situated in a common environment, and interacting with each other in order that the desired functionality and properties of the systems could emerge. These properties of MAS are not always derivable or representable solely on the basis of properties and capabilities of individual agents, but are usually given also by their mutual relationships, interactions, coordination mechanisms, social attitudes, etc. Such aspects of MASs are commonly referred to as *social aspects*.

From the social perspective the following aspects of MAS are commonly considered in MAS models (for detaisl see [4]):

- *Social structure* concerning mainly with the identification of societies which can evolve within the system, specification of their properties, structure, identification of comprised roles, individual entities that can participate in such societies, what roles they can play, their mutual relationships, etc.

- *Social behavior* covering such phenomena as social dynamics (i.e. the ability of a society to react to internal and external events), norms (i.e. rules or standards of behavior shared by members of a society), social interactions (how individuals and/or societies interact with others in order to exchange information, coordinate their activities, etc.), and social activities of individual entities and societies (e.g. how they change their attitudes, roles they play, social relationships), etc.

- *Social attitudes* addressing the individual and/or common tendencies (usually expressed in terms of motivations, needs, wishes, intentions, goals, beliefs, commitments, etc.) to anything of a social value.

In this section the focus is on modeling social structure of multi-agent systems. AML modeling constructs which can be used to model social behavior and social attitudes are outlined in the subsequent sections, mainly 5, 6, and 8.

In order to accommodate special needs for modeling social aspects, AML utilizes concepts of: organization units, social relationships, entity roles, and role properties.

4.1 Organization Units

Organization unit type is a specialized environment type, and thus inherits features of behaviored, socialized and mental semi-entity types. They are used to specify the type of societies that can evolve within the system from both the external as well as internal perspectives.

From an *external perspective*, organization units represent coherent autonomous entities, which can be characterized in terms of their mental and social attitudes, can perform behavior, participate in different kinds of (social) relationships, can observe and interact with their environment, offer and use services, play roles, etc. Their properties and behavior are both (1) emergent properties and behavior of all their constituents, their mutual relationships, observations and interactions, and (2) the features and behavior of organization units themselves.

For modeling organization units from external perspectives, in addition to features defined for UML classes (structural and behavioral features, owned behaviors, relationships, etc.), also all the features of behaviored, socialized, and mental semi-entities can be utilized.

From an *internal perspective*, organization units are types of environment that specify the social arrangements of entities in terms of structures, interactions, roles, constraints, norms, etc.

For this purpose organization unit types usually utilize the possibilities inherited from UML structured classifier, and model their internal structure by contained parts and connectors, in combination with entity role types used as types of the parts.

For an example of an organization unit see Fig. 3 (b).

4.2 Social Relationships

Social relationship is a particular type of connection between social entities related to or having dealings with each other. For modeling such relationships, AML defines a special type of UML property, called social property. The social property can be used either in the form of an owned social attribute, or as the end of a social association, and can specify its social role kind[4].

For an example of modeling social relationships see Fig. 3.

4.3 Roles and Role Properties

Roles are used to define a normative behavioral repertoire of entities, and thus provide the basic building blocks of MAS societies. For modeling roles, AML provides *entity role type*, a specialized behaviored, socialized and mental semi-entity type. Entity role types are used to model abstractions of coherent set of features, capabilities, behaviors, observations, relationships, participation in interactions, and services offered or required by entities participating in a particular context. Each entity role type should be realized by a specific implementation possessed by an entity that can play that entity role type. An instance of an entity role type is called entity role and exists only while some behavioral entity plays it.

For modeling the ability of an entity to play an entity role type, AML provides *role properties*. Role property is a specialized UML property, used to specify that an instance of its owner (i.e. a behavioral entity) can play one or several roles of a particular entity role type. The role property can be used either in the form of a role attribute or as the end of a play association.

One entity can at each time play several entity roles. These entity roles can be of the same as well as of different types. The multiplicity defined for a role property constraints the number of entity roles of given type the particular entity can play concurrently. Additional constraints which govern playing of entity roles can be specified by UML constraints.

To allow explicit manipulation of entity roles in UML activities and state machines, AML defines a set of actions for entity role creation and disposal, particularly create role action and dispose role action.

Fig. 3 (a) contains the diagram depicting an agent of type `Person` which can play entity roles of type `Player`, `Captain`, `Coach`, and `Referee`. The possibility of playing entity roles of a particular type is modeled by play associations. Fig. 3 (b) depicts an organization unit `SoccerMatch`, which comprises three `referees` (of the `Referee` entity role type) and two `teams` (of the `SoccerTeam` organization unit type). The `SoccerTeam` itself consists of one to three `coaches`, and eleven to fifteen `players` of which one is the `captain`. The `players` are peers to each other (the `cooperate` connector), and subordinates to the `coaches` (the `manage` connector), and the `captain` (the `lead` connector). The `referees` are superordinate to the both `SoccerTeams` (the `control` connector).

Fig. 4 shows the instantiation of the previously defined types in a model of a system's snapshot, where the agent Lampard, of type Person, plays the entity role player, and the agent Terry, also of type Person, plays the entity role captain and leads Lampard. The agent `Mourinho`, playing the entity role `coach` manages both players Lampard and Terry.

5 Modeling Interactions

To support modeling of interactions in MAS, AML provides a number of UML extensions, which can be logically subdivided into: (1) generic extensions to UML interactions, (2) speech act based extensions to UML inter-

[4] AML predefines *peer*, *subordinate* and *superordinate* social role kinds, but this set can be extended as required.

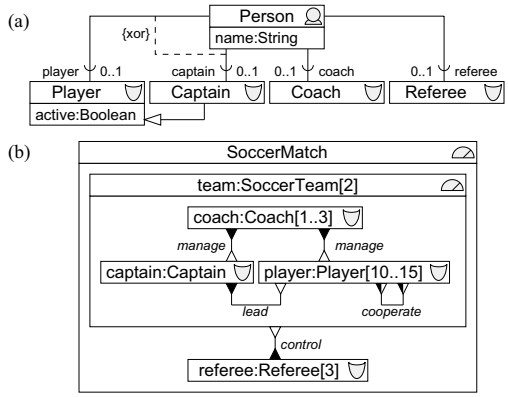

Figure 3: Example of social structure modeling

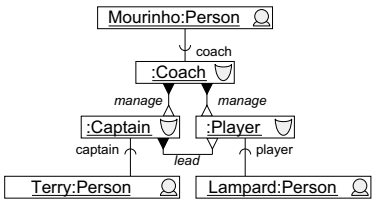

Figure 4: Example of the entity role instantiation and playing

actions, (3) observations and effecting interactions, and (4) services.

5.1 Generic Extensions to UML Interactions

Generic extensions to UML interactions provide means to model: (1) interactions between groups of entities (multi-message and multi-lifeline), (2) dynamic change of object's attributes to express changes in internal structure of organization units, social relationships, or played entity roles, etc., induced by interactions (attribute change), (3) modeling of messages and signals not explicitly associated with the invocation of corresponding methods and receptions (decoupled message), (4) mechanisms for modification of interaction roles of entities (not necessary entity roles) induced by interactions (subset and join dependencies), and (5) modeling the actions of dispatch and reception of decoupled messages in activities (send and decoupled message actions, and associated triggers).

Multi-message is a specialized UML message which is used to model a particular communication between (unlike UML message) multiple participants, i.e. multiple senders and/or multiple receivers.

Multi-lifeline is a specialized UML lifeline, used to represent (unlike UML lifeline) multiple participants in interactions.

Decoupled message is a specialized multi-message used to model the asynchronous dispatch and reception of a message payload without (unlike UML message) explicit specification of the behavior invoked on the side of the receiver. The decision of which behavior should be invoked when the decoupled message is received is up to the receiver what allows to preserve its autonomy in processing messages.

Attribute change is a specialized UML interaction fragment used to model the change of attribute values (state) of interacting entities induced by the interaction. Attribute change thus enables to express addition, removal, or modification of attribute values, and also to express the added attribute values by sub-lifelines. The most likely utilization of attribute change is in modeling of dynamic change of entity roles played by behavioral entities represented by lifelines in interactions, and the modeling of entity interactions with respect to the played entity roles (i.e. each sub-lifeline representing a played entity role can be used to model interaction of its player with respect to this entity role).

Subset is a specialized UML dependency between event occurrences owned by two distinct (superset and subset) lifelines used to specify that since the event occurrence on the superset lifeline, some of the instances it represents (specified by the corresponding selector) are also represented by another, the subset lifeline.

Similarly, *join* dependency is also a specialized UML dependency between two event occurrences on lifelines (subset and union ones), used to specify that a subset of instances, which have been until the subset event occurrence represented by the subset lifeline, is after the union event occurrence represented by the ŞunionŤ lifeline. The union lifeline, thus after the union event occurrence represents the union of the instances it has been representing before, and the instances specified by the join dependency.

Send decoupled message action is a specialized UML send object action used to model the action of dispatching a decoupled message, and *accept decoupled message action* is a specialized UML accept event action used to model reception of a decoupled message action that meets the conditions specified by the associated decoupled message trigger.

A simplified interaction between entities taking part in a player substitution is depicted in Fig. 5. Once the main `coach` decides which players are to be substituted (`p1` to be substituted and `p2` the substitute), he first notifies player `p2` to get ready and then asks the main `referee` for permission to make the substitution. The main `referee` in turn replies by an `answer`. If the `answer` is "yes", the substitution process waits until the game is interrupted. If so, the `coach` instructs player `p1` to exit and `p2` to enter. Player `p1` then leaves the pitch and joins the group of inactive players and `p2` joins the pitch and thereby the group of active players.

Fig. 6 shows an example of the communicative interaction in which the attribute change elements are used to model changes of entity roles played by agents. The diagram realizes the scenario of a captain change caused by the original captain (`player2`) substitution.

At the beginning of the scenario the agent

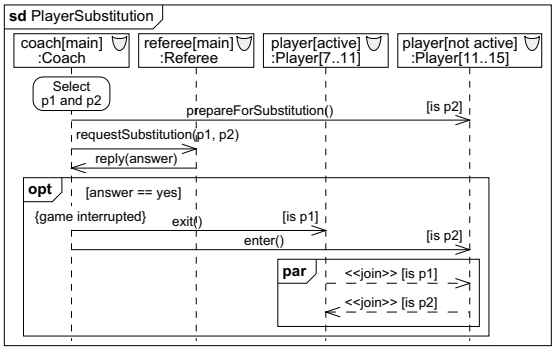

Figure 5: Example of a communicative interaction

`player2` is captain (modeled by its role property `captain`). During the substitution, the main `coach` gives the `player2` order to hand the captainship over (`handCaptainshipOver()` message) and the `player1` the order to become the captain (`becomeCaptain()` message). After receiving these messages, the `player2` stops playing the entity role `captain` (and starts playing the entity role of ordinary `player`) and the `player1` changes from ordinary `player` to `captain`.

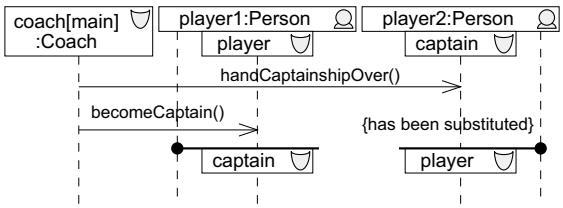

Figure 6: Example of a social interaction with entity role changes

5.2 Speech Act Specific Extensions to UML Interactions

Speech act specific extensions to UML interactions comprise modeling of speech-acts (communication message), speech act based interactions (communicative interactions), patterns of interactions (interaction protocols), and modeling the actions of dispatch and reception of speech-act based messages in activities (send and accept communicative message actions, and associated triggers).

Communication message is a specialized decoupled message used to model communicative acts of speech act based communication within *communicative interaction*s (a specialized UML interaction) with the possibility of explicit specification of the message performative and payload. Both the communication message and communicative interaction can also specify used agent communication and content languages, ontology and payload encoding.

Interaction protocol is a parametrized communicative interaction template used to model reusable templates of communicative interactions.

5.3 Observations and Effecting Interactions

AML provides several mechanisms for modeling observations and effecting interactions in order to (1) allow modeling of the ability of an entity to observe and/or to bring about an effect on others (perceptors and effectors), (2) specify what observation and effecting interactions the entity is capable of (perceptor and effector types and perceiving and effecting acts), (3) specify what entities can observe and/or effect others (perceives and effects dependencies), and (4) explicitly model the actions of observations and effecting interactions in activities (percept and effect actions).

Observations are in AML modeled as the ability of an entity to perceive the state of (or to receive a signal from) an observed object by means of *perceptors*, which are specialized UML ports. *Perceptor types* are used to specify (by means of owned *perceiving acts*) the observations an owner of a perceptor of that type can make.

Perceiving acts are specialized UML operations which can be owned by perceptor types and thus used to specify what perceptions their owners, or perceptors of given type, can perform.

The specification of which entities can observe others, is modeled by a *perceives* dependency. For modeling behavioral aspects of observations, AML provides a specialized *percept action*.

Different aspects of effecting interactions are modeled analogously, by means of *effectors*, *effector types*, *effecting acts*, *effects* dependencies, and *effect actions*.

An example is depicted in Fig. 8 (a) which shows an entity role type `Player` with two eyes–perceptors called `eye` of type `Eye`, and two legs–effectors called `leg` of type `Leg`. Eyes are used to see other players, the pitch and the ball, and to provide localization information to the internal parts of a player. Legs are used to change the player's position within the pitch (modeled by changing of internal state implying that no effects dependency need be placed in the diagram), and to manipulate the ball.

5.4 Services

The AML support for modeling services comprises (1) the means for the specification of the functionality of a service and the way a service can be accessed (service specification and service protocol), (2) the means for the specification of what entities provide/use services (service provision, service usage, and serviced property), and (if applicable) by what means (serviced port).

A *service* is a coherent block of functionality provided by a behaviored semi-entity, called service provider, that can be accessed by other behaviored semi-entities (which

can be either external or internal parts of the service provider), called service clients.

Service specification is used to specify a service by means of owned service protocols, i.e. specialized interaction protocols extended with the ability to specify two mandatory, disjoint and nonempty sets of (not bound) parameters, particularly: provider and client template parameters.

The *provider template parameters* of all contained service protocols specify the set of the template parameters that must be bound by the service providers, and the *client template parameters* of all contained service protocols specify the set of template parameters that must be bound by the service clients. Binding of these complementary template parameters specifies the features of the particular service provision/usage which are dependent on its providers and clients.

Service provision/usage are specialized dependencies used to model provision/use of a service by particular entities, together with the binding of template parameters that are declared to be bound by service providers/clients.

Fig. 7 shows a specification of the `Motion` service defined as a collection of three service protocols. The `CanMove` service protocol is based on the standard FIPA protocol `FIPA-Query-Protocol`[5] [21] and binds the `proposition` parameter (the content of a `query-if` message) to the capability `canMove(what, to)` of a service provider. The `participant` parameter of the `FIPA-Query-Protocol` is mapped to a service provider and the `initiator` parameter to a service client. The `CanMove` service protocol is used by the service client to ask if an object referred by the `what` parameter can be moved to the position referred by the `to` parameter. The remaining service protocols `Move` and `Turn` are based on the `FIPA-Request-Protocol` [21] and are used to change the position or direction of a spatial object.

Binding of the `Motion` service specification to the provider `3DSpace` and the client `3DObject` is depicted in Fig. 2.

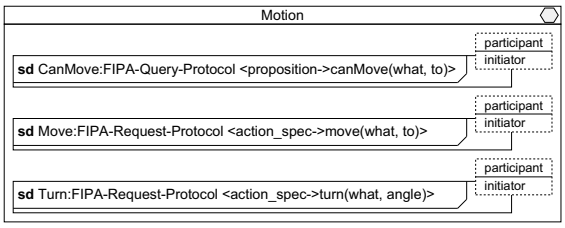

Figure 7: Example of service specification

[5]The AML specification of the interaction protocol can be found in [3].

6 Modeling Capabilities and Behavior

AML extends the capacity of UML to abstract and decompose behavior by another two modeling elements: capability and behavior fragment.

Capability is an abstract specification of a behavior which allows reasoning about and operations on that specification. Technically, a capability represents a unification of the common specification properties of UML's behavioral features and behaviors expressed in terms of their inputs, outputs, pre- and post-conditions.

Behavior fragment is a specialized behaviored semi-entity type used to model a coherent re-usable fragment of behavior and related structural and behavioral features. It enables the (possibly recursive) decomposition of a complex behavior into simpler and (possibly) concurrently executable fragments, as well as the dynamic modification of an entities behavior in run-time. The decomposition of a behavior of an entity is modeled by owned aggregate attributes of the corresponding behavior fragment type.

Fig. 8 (a) shows the decomposition of the `Player` entity role type's behavior into a structure of behavior fragments. In part (b) two fragments, `Mobility` and `BallHandling`, are described in terms of their owned capabilities (`turn`, `walk`, `catch`, etc.).

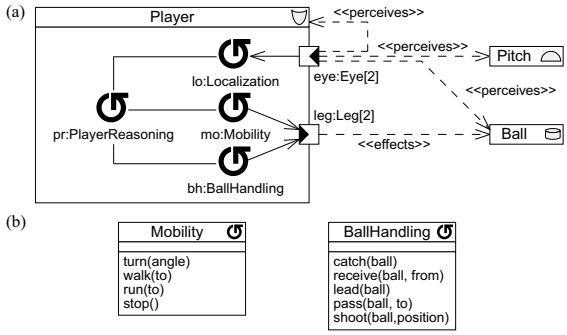

Figure 8: Example of behavior fragments, observations and effecting interactions

7 Modeling MAS Deployment and Mobility

The means provided by AML to support modeling of MAS deployment and agent mobility comprise: (1) the support for modeling the physical infrastructure onto which MAS entities are deployed (agent execution environment), (2) what entities can occur on which nodes of the physical infrastructure and what is the relationship of deployed entities to those nodes (hosting property), (3) how entities can get to a particular node of the physical infrastructure (move and clone dependencies), and (4) what can cause the en-

tity's movement or cloning throughout the physical infrastructure (move and clone actions).

Agent execution environment type is a specialized UML execution environment used to model types of execution environments within which MAS entities can run. While it is a behaviored semi-entity type, it can explicitly, for example, also specify a set of services that the deployed entities can use or should provide at run time.

Agent execution environment can also own *hosting properties*, which are used to classify the entities which can be hosted by the owning agent execution environment. The hosting property's *hosting kind* specifies the relation of the referred entity type to its owning agent execution environment (i.e. either *resident* of *visitor*).

Hosting association is a specialized UML association used to specify hosting property in the form of an association end.

Move is a specialized UML dependency between two hosting properties used to specify that the entities represented by the source hosting property can be moved to the instances of the agent execution environments owning the destination hosting property. Likewise the *clone* dependency is used.

Move and *clone actions* are specialized UML add structural feature actions used to model actions that cause movement or cloning of an entity from one agent execution environment to another one. Both the actions thus specify: (1) which entity is being moved or cloned, (2) the destination agent execution environment instance where the entity is being moved or cloned, and (3) the hosting property where the moved or cloned entity is being placed.

8 Modeling Mental Aspects

Mental semi-entities can be characterized in terms of their mental attitudes, i.e. motivations, needs, wishes, intentions, goals, beliefs, commitments, etc. To allow modeling all the above, AML provides: goals, beliefs, plans, contribution relationships, mental properties and associations, mental constraints, and commit/cancel goal actions.

Goal is a specialized UML class used to model goals, i.e. conditions or states of affairs with which the main concern is their achievement or maintenance. Goals can thus be used to represent objectives, needs, motivations, desires, etc.

Belief is a specialized UML class used to model a state of affairs, proposition or other information relevant to the system and its mental model.

The attitude of a mental semi-entity to a belief or commitment to a goal is modeled by the belief or the goal instance being held in a slot of the corresponding *mental property* (owned by the mental semi-entity, or a mental association relating the belief or the goal to the mental semi-entity).

Plan is a specialized UML activity used to model: predefined plans, or fragments of behavior from which the plans can be composed.

Mental constraint is a specialized UML constraint used to specify properties of owning beliefs, goals and plans which can be used within reasoning processes of mental semi-entities. Supported kinds of mental constraints are pre- and post-conditions, commit conditions, cancel conditions and invariants.

Contribution is a specialized UML relationship used to model logical relationships between goals, beliefs, plans and their mental constraints. The manner in which the specified mental constraint (e.g. post-condition) of the contributor influences the specified mental constraint kind of the beneficiary (e.g. pre-condition) as well as the degree of the contribution can also be specified.

Actions to model commitments to and de-commitments from goals within activities are also provided.

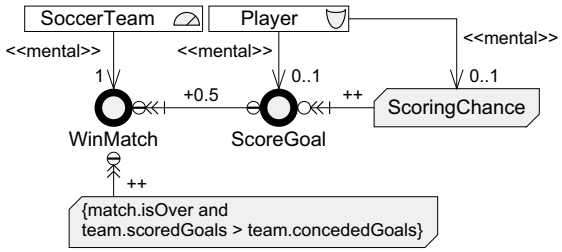

Figure 9: Example of a mental model

Fig. 9 shows an example of a snapshot of the mental model of a soccer team (represented by the `SoccerTeam` organization unit type) and its players (`Player` entity role type). The soccer team has the goal to win a match (modeled by the `WinMatch` goal). The goal `WinMatch` is accomplished, when the soccer match is over and the team has scored more goals than conceded. This is expressed by the sufficient contribution of the belief `{match.isOver and team.scoredGoals > team.concededGoals}` to the postcondition of the goal `WinMatch`. The soccer team players may have goals to score a goal (`ScoreGoal`) which it is feasible to commit to, when they are in a scoring chance. This is expressed by the necessary contribution of the belief `ScoringChance` to the precondition of the goal `ScoreGoal`.

9 Conclusion

The limitation in paper length has not allowed to present all the modeling elements and mechanisms AML provides (e.g. support for ontologies, contexts, etc.). Nevertheless, we believe that from what has been presented in this paper, it is evident that AML provides a rich set of modeling constructs for modeling applications that embody and/or exhibit characteristics of multi-agent systems. It integrates best modeling practices and concepts from existing agent oriented modeling and specification languages

into a unique framework built on foundations of UML 2.0 and OCL 2.0. The structure of the language definition together with the MDA/MOF/UML "metamodeling technology" (UML profiles, first-class metamodel extension, etc., gives AML the advantage of natural extensibility and customization. AML is also supported by CASE tools.

We feel confident that AML is sufficiently detailed, comprehensive and tangible to be a useful tool for software architects building systems based on, or exhibiting characteristics of, multi-agent technologies. In this respect we anticipate that AML may form a significant contribution to the effort of bringing about widespread adoption of intelligent agents across varied commercial marketplaces.

Acknowledgement

The authors are indebted to Stefan Brantschen, Monique Calisti, and Dominic Greenwood, for their support and fruitful comments which have inspired many ideas and thus substantially influenced the current version of AML.

References

[1] B. Bauer, J.P. Muller, and J. Odell. Agent UML: A Formalism for Specifying Multiagent Interaction. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*, pages 91–103. Springer-Verlag, Berlin, 2001.

[2] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 2(3):203–236, 2004.

[3] R. Cervenka and I. Trencansky. Agent Modeling Language: Language Specification. Version 0.9. Technical report, Whitestein Technologies, 2004.

[4] R. Cervenka, I. Trencansky, and Calisti. Modeling Social Aspects of Multiagent Systems: The AML Approach. In J.P. Muller and F. Zambonelli, editors, *The Fourth International Joint Conference on Autonomous Agents & Multi Agent Systems (AAMAS 05). Workshop 7: Agent-Oriented Software Engineering (AOSE)*, pages 85–96, Universiteit Utrecht, The Netherlands, 2005.

[5] R. Cervenka, I. Trencansky, M. Calisti, and D. Greenwood. AML: Agent Modeling Language. Toward Industry-Grade Agent-Based Modeling. In J. Odell, P. Giorgini, and J.P. Muller, editors, *Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004*, pages 31–46. Springer-Verlag, Berlin, 2005.

[6] M. Cossentino, L. Sabatucci, and A. Chella. A Possible Approach to the Development of Robotic Multi-Agent Systems. In *IEEE/WIC Conference on Intelligent Agent Technology (IAT'03)*, pages 539–544, Halifax, Canada, 2003.

[7] S. Cranefield, S. Haustein, and M. Purvis. UML-Based Ontology Modelling for Software Agents. In *IProceedings of the Workshop on Ontologies in Agent Systems, 5th International Conference on Autonomous Agents*, 2001.

[8] S.A. DeLoach. Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems. In *Agent-Oriented Information Systems '99 (AOIS'99)*, Seattle, WA, 1999.

[9] M. d'Inverno and M. Luck. *Understanding Agent Systems*. Springer-Verlag, Berlin, 2001.

[10] R. Evans, P. Kearny, J. Stark, G. Caire, F. Garijo, J.J. Gomez-Sanz, F. Leal, P. Chainho, and P. Massonet. MESSAGE: Methodology for Engineering Systems of Software Agents. Technical Report P907, EURESCOM, 2001.

[11] J. Odell, H.V.D. Parunak, and B. Bauer. Extending UML for Agents. In G. Wagner, Y. Lesperance, and E. Yu, editors, *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, pages 3–17, Austin, Texas, 2000.

[12] J. Odell, H.V.D. Parunak, M. Fleischer, and S. Brueckner. Modeling Agents and their Environment. In F. Giunchiglia, J. Odell, and G. Weiss, editors, *Agent-Oriented Software Engineering III: Third International Workshop, AOSE 2002*, pages 16–31. Springer-Verlag, Berlin, 2002.

[13] OMG. Meta Object Facility (MOF) Specification. Version 1.4, formal/2002-04-03, april 2002.

[14] OMG. UML 2.0 OCL Specification. ptc/03-10-14, October 2003.

[15] OMG. Unified Modeling Language: Superstructure version 2.0. ptc/03-08-02, 2003.

[16] L. Padgham and M. Winikoff. Prometheus: A Methodology for Developing Intelligent Agents. In F. Giunchiglia, J. Odell, and G. Weiss, editors, *Agent-Oriented Software Engineering III: Third International Workshop, AOSE 2002*, pages 174–185. Springer-Verlag, Berlin, 2002.

[17] A.S. Rao and M.P. Georgeff. Modeling Rational Agents within a BDI-Architecture. In J.F. Allen, R. Fikes, and E. Sandewall, editors, *Knowledge Representation and Reasonning (KR&R-91): Principles of Knowledge Representation and Reasoning*, pages 473–484. Morgan Kaufmann Publishers, San Mateo, California, 1991.

[18] V. Silva, A. Garcia, A. Brandao, C. Chavez, C. Lucena, and P. Alencar. Taming Agents and Objects in Software Engineering. In A. Garcia, C. Lucena, J. Castro, A. Omicini, and F. Zambonelli, editors, *Software Engineering for Large-Scale Multi-Agent Systems: Research Issues and Practical Applications*, volume LNCS 2603, pages 1–25. Springer-Verlag, Berlin, 2003.

[19] M.K. Smith, D. McGuinness, R. Volz, and C. Welty. Web Ontology Language (OWL), Guide Version 1.0, W3C Working Draft. URL: http://www.w3.org/TR/2002/WD-owl-guide-20021104, 2002.

[20] A. Sturm, D. Dori, and O. Shehory. Single-Model Method for Specifying Multi-Agent Systems. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 121–128. ACM Press, New York, NY, 2003.

[21] The Foundation for Intelligent Physical Agents. FIPA Specifications Repository. URL: http://www.fipa.org/repository/index.html, 2004.

[22] W.M. Turski and T.S.E. Maibaum. *The Specification of Computer Programs*. Addison-Wesley, London, 1987.

[23] G. Wagner. The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior. *Information Systems*, 28(5):475–504, 2003.

[24] G. Weiss. *Multiagent Systems–A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, 3^{rd} edition, 2001.

[25] F. Zambonelli, N.R. Jennings, and M. Wooldridge. Developing multiagent systems: the Gaia Methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, 2003.