

AML: Agent Modeling Language

Toward Industry-Grade Agent-Based Modeling

Radovan Červenka, Ivan Trenčanský, Monique Calisti, and Dominic Greenwood

Whitestein Technologies, Panenská 28, 811 03 Bratislava, Slovakia
Tel +421 (2) 5443-5502, Fax +421 (2) 5443-5512
{rce,itr,mca,dgr}@whitestein.com
<http://www.whitestein.com>

Abstract. The *Agent Modeling Language (AML)* is a semi-formal visual modeling language, specified as an extension to UML 2.0. It is a consistent set of modeling constructs designed to capture the various aspects of multi-agent systems. The ultimate objective for AML is to provide a means for software engineers to incorporate aspects of multi-agent system engineering into their analysis and design processes. This paper presents an introductory overview of AML, discussing the motivations driving the development of the language, the scope and approach taken, the specific language structure and optional extensibility. The core AML modeling constructs are explained and demonstrated by example where possible. Extensions to OCL and CASE tool support are also discussed.

1 Introduction

The Agent Modeling Language (AML) is a semi-formal¹ visual modeling language for specifying, modeling and documenting systems that incorporate concepts drawn from Multi-Agent Systems (MAS) theory.

The primary application context of AML is to systems explicitly designed using software multi-agent system concepts. AML can however also be applied to other domains such as business systems, social systems, robotics, etc. In general, AML can be used whenever it is suitable or useful to build models that (1) consist of a number of autonomous, concurrent and/or asynchronous (possibly proactive) entities, (2) comprise entities that are able to observe and/or interact with their environment, (3) make use of complex interactions and aggregated services, (4) employ social structures, and (5) capture mental characteristics of systems and/or their parts.

Why Another Modeling Language? The most significant motivation driving the development of AML stems from the extant need for a ready-to-use, complete

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. It is not based on a strict formal (e.g. mathematical) theory.

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:

- is built on proven technical foundations,
- integrates best practices from agent-oriented software engineering (AOSE) and object-oriented software engineering (OOSE) domains,
- is well specified and documented,
- is internally consistent from the conceptual, semantic and syntactic perspectives,
- is versatile and easy to extend,
- is independent of any particular theory, software development process or implementation environment, and
- is supported by Computer-Aided Software Engineering (CASE) tools.

Given these requirements, AML is designed to address and satisfy the most significant deficiencies with current state-of-the-art and practice in the area of MAS oriented modeling languages, which are often:

- insufficiently documented and/or specified, or
- using proprietary and/or non-intuitive modeling constructs, or
- aimed at modeling just a restricted set of MAS aspects, or
- applicable only to a specific theory, application domain, MAS architecture, or technology, or
- mutually incompatible, or
- insufficiently supported by CASE tools.

Paper Layout. The purpose of this paper is to present an overview of AML. However, due to limitations in paper length it is not possible to provide a comprehensive description of AML including its abstract syntax, semantics, notation, and UML profiles. Therefore we have chosen to discuss the principles that guided the specification of AML, our specific approach and a brief overview of the various AML modeling constructs. Using concrete examples, we also illustrate some of the more essential AML concepts. The paper is structured as follows:

Section 2 presents our approach in terms of language definition and specification, scope of the language, foundations, and structure and extensibility. Sections 3, 4, 5, and 6 then present the overview of AML organized according to the packages defined in the AML metamodel: Architecture, Behavior, Mental Aspects, and Contexts. Section 7 describes an extension of the OCL Standard Library required for modeling modal, deontic, and temporal logic, and cognitive primitives. Section 8 provides an overview of the CASE tool support for AML and Section 9 draws conclusions and recommendations for further work.

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 [1], BDI [2], SMART [3]), modeling and specification languages (e.g. AUML [4–6], GRL [7], TAO [8], OPM/MAS [9], AOR [10], UML [11], OCL [12], OWL [13], UML-based ontology modeling [14], OWL-S [15]), methodologies (e.g. MESSAGE [16], Gaia [17], TROPOS [18], PASSI [19], Prometheus [20], MAS CommonKADS [21], MaSE [22]), 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 a *conservative extension of UML*² to the maximum possible extent.

2.1 Scope

AML is designed to support business modeling, requirements specification, analysis, and design of software systems that use software agent concepts and principles.

The current version of AML offers:

- Support for the human mental process of requirements specification and analysis of complex problems/systems, particularly (1) mental aspects, which can be used for modeling intentionality in use case models, goal-based requirements, problem decomposition, etc. (Sect. 5), and (2) contexts, which can be used for situation-based modeling (Sect. 6).
- Support for the abstraction of architectural and behavioral concepts associated with multi-agent systems, i.e. ontologies (Sect. 3.1), MAS entities (Sect. 3.2), social aspects (Sect. 3.3), behavior abstraction and decomposition (Sect. 4.1), communicative interactions (Sect. 4.2), services (Sect. 4.3), observations and effecting interactions (Sect. 4.4), mental aspects used for modeling mental attitudes of entities (Sect. 5), MAS deployment and agent mobility (Sect. 3.4).

2.2 Outside the Scope of AML

AML does not cover the details of operational semantics, which might be dependent on a specific execution model given by an applied theory or deployment environment (e.g. agent platforms, reasoning engines, other technologies used).

² A conservative extension of UML is a strict extension of UML which retains the standard UML semantics in unaltered form[23].

2.3 UML 2.0 as a Base

AML is based on the Unified Modeling Language (UML) 2.0 Superstructure [11], augmenting it with several new modeling concepts appropriate for capturing the typical features of multi-agent systems.

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.

2.4 Structure of AML

AML is defined at two distinct levels – *AML Metamodel and Notation* and *AML Profiles*. Fig. 1 depicts these two levels, their derivation from UML 2.0 and optional extensions based on UML 1.* and 2.0.

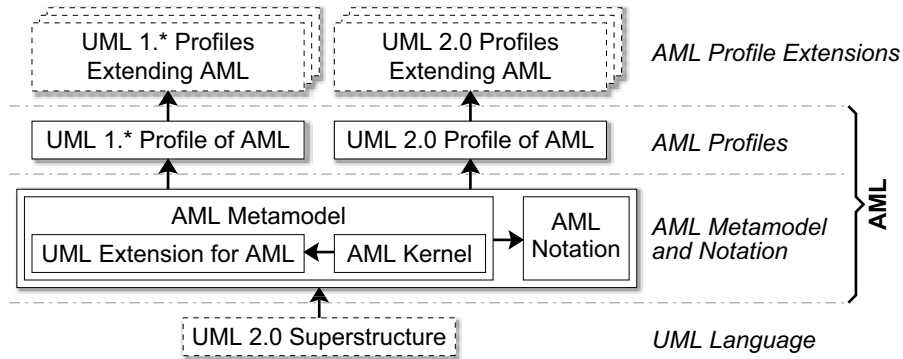


Fig. 1. Levels of AML definition

With reference to Fig. 1, the *UML Language* level contains the UML 2.0 Superstructure defining the abstract syntax, semantics and notation of UML. AML uses this level as the foundation upon which to define MAS-specific modeling constructs.

The *AML Metamodel and Notation* level defines the AML abstract syntax, semantics and notation, structured into two packages: *AML Kernel* and *UML Extension for AML*.

The *AML Kernel* package is the core of AML where the AML specific modeling elements are defined. It is logically structured into several packages, each of which covers a specific aspect of MAS. The most significant of these packages are described in sections 3, 4, 5, and 6. The AML Kernel is a conservative extension of UML 2.0.

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

At the level of *AML Profiles*, two UML profiles built upon the AML Meta-model and Notation are provided: *UML 1.* Profile for AML* (based on UML 1.*) and *UML 2.0 Profile for AML* (based on UML 2.0). These profiles, inter alia, enable implementation of AML within UML 1.* and UML 2.0 based CASE tools, respectively.

Based on AML Profiles, users are free to define their own language extensions to customize AML for specific modeling techniques, implementation environments, technologies, development processes, etc. The extensions can be defined as standard UML 1.* or 2.0 profiles. They are commonly referred to as *AML Profile Extensions*.

2.5 Extensibility of AML

AML is designed to encompass a broad set of relevant theories and modeling approaches, however we are aware that it is impossible to cover all of them. In those cases where AML is insufficient, several mechanisms can be used to extend or customize AML as required.

Each of the following extension methods (and their combinations) can be used:

- *Metamodel extension*. This offers first-class extensibility (as defined by MOF [24]) of the AML metamodel and notation.
- *AML profile extension*. This offers the possibility to adapt AML Profiles using constructs specific to a given domain, platform, or development method, without the need to modify the underlying AML metamodel.
- *Concrete model extension*. This offers the possibility to use alternative MAS modeling approaches as complementary specifications to the AML model.

3 Architecture

This section provides an overview of the AML constructs used to model architectural aspects of multi-agent systems.

3.1 Ontology

AML supports ontology modeling in terms of ontology classes and instances, their relationships, constraints, ontology utilities, and ontology packages.

Ontology package is a specialized UML package used to specify a single ontology. By utilizing the features inherited from UML package (package nesting, element import, package merge, etc.), ontologies can be logically structured.

Ontology class is a specialized UML class used to represent an ontology concept. Attributes and operations of the ontology class represent its slots. Ontology functions, actions, and predicates belonging to a concept modeled by the ontology class are modeled by its operations. Ontology classes can use all types of relationship allowed for UML classes (associations, generalizations, dependencies, etc.) with their standard UML semantics.

Ontology utility is a specialized UML Class used to cluster global ontology constants, variables, functions, actions, and predicates, which are modeled as its features. These features can be used by (referred to) other elements within the owning ontology. One ontology package can contain several ontology utilities allowing logical clustering of the features.

The diagram in Fig. 2 depicts a simplified version of a `SoccerMatch` ontology. Rectangles with the special icon (“C” placed in a rounded square) represent ontology classes that model concepts from the domain. Their relationships are modeled by standard UML relationships with standard semantics.

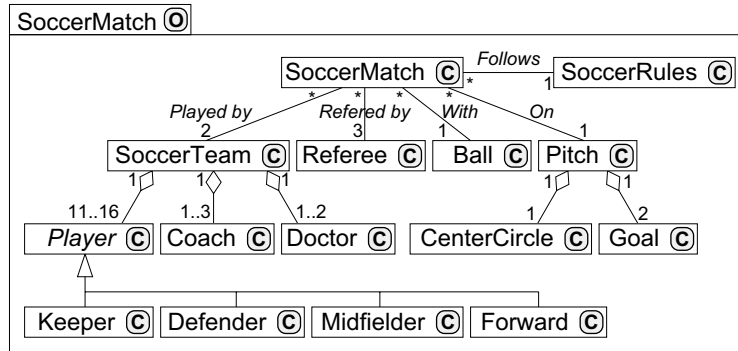


Fig. 2. Example of an ontology

3.2 Fundamental Entity Types

In general, *entities* represent objects that can exist in the system independently of other objects. AML defines three modeling constructs used to model types of MAS entities, namely: agents, environments, and resources. Entities can also be modeled at the instance level by UML instance specifications categorized according to the corresponding types:

Agent type is a specialized UML class used to model a type of *agents*, i.e. self contained entities that are capable of interactions, observations and autonomous behavior within their environment.

Resource type is a specialized UML class used to model a type of resources within the system³. A *resource* is a physical or informational entity with which

³ A resource positioned outside a system is modeled as a UML actor.

the main concern is availability (in terms of its quantity, access rights, conditions of usage/consumption, etc.).

Environment type is a specialized UML class used to model a type of system’s inner environment⁴, i.e. the logical or physical surroundings of entities which provide conditions under which the entities exist and function. As environments are usually complex entities, different environment types are usually used to model different aspects of the environment.

In AML, all the aforementioned entity types are collectively called *behavioral entities*. They can own capabilities, be decomposed into behavior fragments (see Sect. 4.1), provide and use services (see Sect. 4.3), and own perceptors and effectors (see Sect. 4.4). Additionally, agent and environment types are *autonomous entities* that can be characterized in terms of their mental attitudes (see Sect. 5). All entities can make use of modeling mechanisms inherited from UML class, i.e. they can own features, participate in varied relationship types, be internally structured into parts, own behaviors, etc.

Fig. 3 shows 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. 4.3).

Three concrete `3DObjects` are defined: an agent type `Person`, a resource type `Ball` and a class `Goal`. `3DSpace` is furthermore specialized into a concrete environment type `Pitch` representing a soccer pitch containing two goals and a ball.

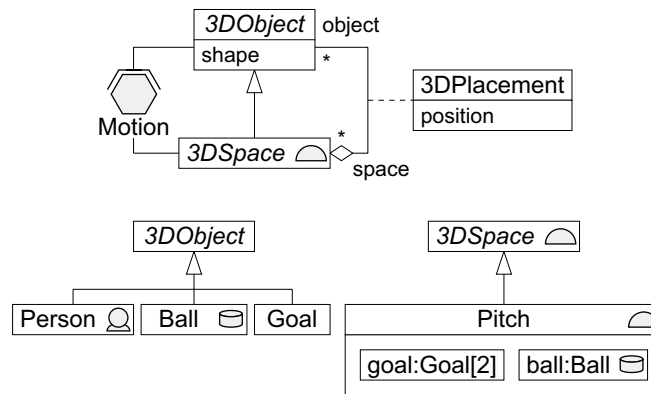


Fig. 3. Example of entities, their relationships, service provision and usage

⁴ *Inner environment* is that part of an entity’s environment that is contained within the boundaries of the system.

3.3 Social Aspects

AML defines several elements for modeling the social aspects of MAS, including structural characteristics of socialized entities and certain aspects of their social behavior.

Organization unit type is a specialized environment type used to model a type of *organization units*. From an external perspective, organization units represent coherent autonomous entities, the properties and behavior of which 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. 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.

Social Relationships are modeled by *social properties* and *social associations*. The current version of AML supports the modeling of superordinate-subordinate and peer-to-peer relationships, but this set can be extended as required (e.g. to model producer-consumer, competitive, or cooperative relationships).

Entity role type is a specialized UML class used to represent a coherent set of features, behaviors, participation in interactions, and connections to services offered or required by behavioral entities participating in a particular context. Each entity role type, being an abstraction of a set of capabilities, should be realized by a specific implementation possessed by a behavioral entity that can play that entity role type.

An instance of the entity role type is called an *entity role*⁵. It represents the execution of behaviors, usage of features and/or participation in interactions as defined by the particular entity role type. A given entity role exists only while a behavioral entity instance plays it.

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, and related triggers.

The possibility of playing an entity role by a behavioral entity is modeled by *role property* and *play association*. Mechanisms are also offered for modeling dynamic changes of roles, reasoning about played roles, expressing multiplicities and constraints on played entity roles and navigation through the structural features and capabilities of entity roles types from their playing entities (used for example in model navigation expressions).

Fig. 4 part (a) contains a diagram which express that an agent of type **Person** can play entity roles of type **Player**, **Doctor**, **Coach**, and **Referee**. The possibility of playing entity roles of a particular type is modeled by play associations. Fig. 4 part (b) depicts an organization structure containing the entities participating in a soccer match. An environment type **Pitch** contains one **referee** (of the **Referee** entity role type) and two **teams** (of the **SoccerTeam** organization unit type). **SoccerTeam** itself consists of one to three **coaches**, one or

⁵ AML uses the term “entity role” to differentiate agent-related roles from the roles defined by the UML 2.0, i.e. roles used for collaborations, parts, and associations.

two doctors, and seven to eleven players. The players are subordinate to the coaches (by the `lead` connector), and to referees (by the `refer` connector), but peers to doctors (by the `treat` connector).

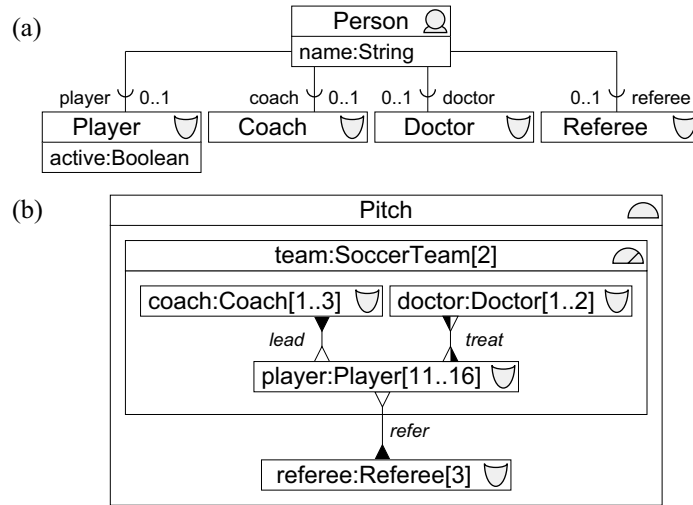


Fig. 4. Example of social structure modeling

3.4 MAS Deployment and Mobility

To model deployment of an MAS to a physical environment, AML extends the UML deployment model with *agent platform* as a specialized execution environment, and specialized types of artifacts representing deployed entities. Particularly, *agent artifact*, *environment artifact*, and *resource artifact*.

Modeling of structural and behavioral aspects of agent mobility is also provided. The structural aspects allow the specification of which agent artifacts are mobile, on what deployment targets they can appear, their relationships to deployment targets, and what actions (move or clone) can make them appear on a particular deployment target. The behavioral aspects allow the specification of the move and clone actions and the corresponding triggers used in activities and state machines to incorporate mobility into behavior specifications.

4 Behavior

This section contains an overview of the AML constructs used to model behavioral aspects of multi-agent systems.

4.1 Behavior Abstraction and Decomposition

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

Capability is used to model an abstract specification of a behavior that 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 inputs outputs, pre- and post-conditions.

Behavior fragment is a specialized class used to model a coherent re-usable fragment of behavior. It enables the decomposition of a complex behavior into simpler and (possibly) concurrently executable fragments. A behavior fragment can be shared by several entities and the behavior of an entity can, possibly recursively, be decomposed into several behavior fragments.

Fig. 5 part (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.).

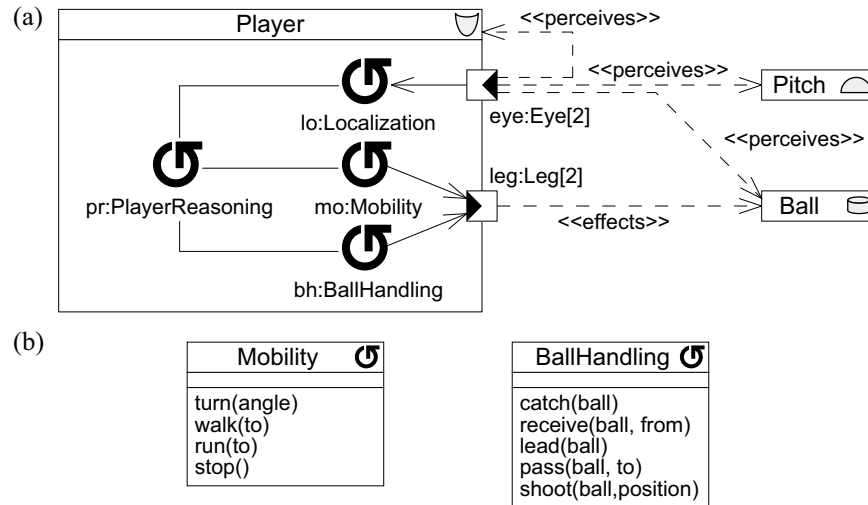


Fig. 5. Example of behavior fragments, observations and effecting interactions

4.2 Communicative Interactions

AML provides generic extensions to UML interactions in order to model interactions between groups of objects (using *multi-message* and *multi-lifeline*), dynamic change of an object’s attributes induced by interactions (using *role*

change), modeling of messages and signals not explicitly associated with an invocation of corresponding methods and receptions (using *agentified message* and *agentified signal*).

In addition to these generic concepts, AML also models communicative act based interactions commonly used in multi-agent systems, particularly: *communicative acts* (specialized agentified messages), *communicative interactions* (specialized UML interactions used to model communicative act based interactions), and *interaction protocols* (parametrized communicative interactions used to model patterns of interactions).

A simplified interaction between entities taking part in a player substitution is depicted in Fig. 6. 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.

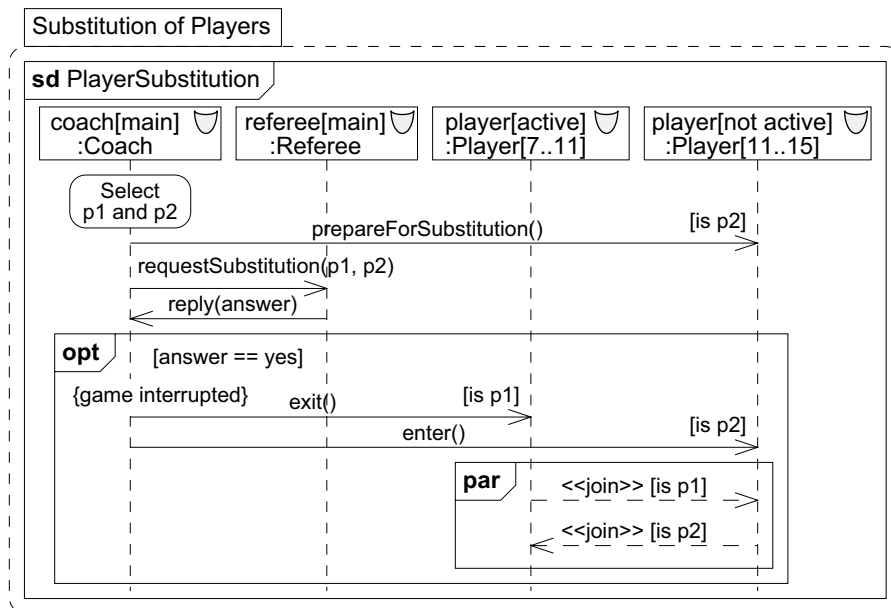


Fig. 6. Example of a communicative interaction

4.3 Services

Services are encapsulated blocks of functionality that entities can offer to perform upon request. They are modeled in AML in terms of service specifications, service provisionings and service usages.

Service specification is a modeling element for specifying the functionality and accessibility of a service. Technically it specifies a set of specialized interaction protocols (called *service protocols*) each of which determines two sets of template parameters that must be bound by the service's providers and clients.

Service provision is a specialized dependency used to model provision of a service by particular entities, together with the binding of template parameters that are declared to be bound by service providers.

Service usage is a specialized dependency used to model usage of a service by particular entities, together with the binding of template parameters that are declared to be bound by service 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** [25] 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** [25] 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. 3.

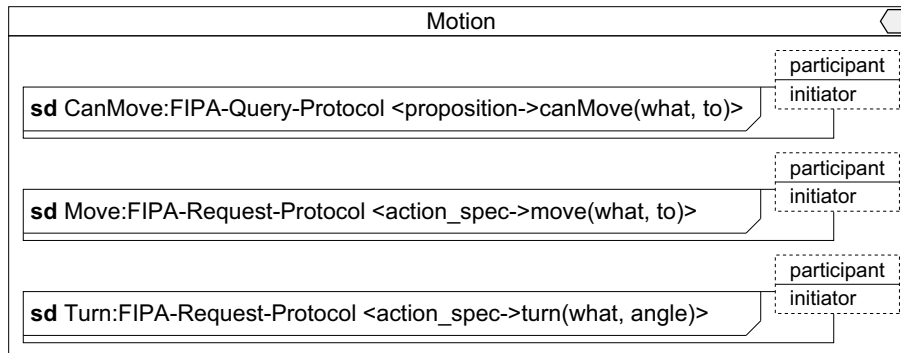


Fig. 7. Example of service specification

4.4 Observations and Effecting Interactions

AML defines several constructs for modeling observations (i.e. the ability of entities to observe features of other entities) and effecting interactions (i.e. the ability of entities to manipulate, or modify the state of, other entities).

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

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. 5 (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 Mental Aspects

This section provides an overview of the AML constructs used to model beliefs, goals, plans, and mental relationships. These can be used for:

1. Enriching use case modeling through the expression of intentionality, goal-based requirements modeling, problem decomposition, etc.
2. Modeling mental attitudes of autonomous entities, which represent their informational, motivational and deliberative states.

Belief is a specialized UML class used to model information which autonomous entities have about themselves or their environment with a certain degree of subjective confidence.

Goal is a specialized UML class used to model goals, i.e. conditions of states of affairs, the achievement or maintenance of which is controlled by an autonomous entity or to which a modeling element may contribute.

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

Mental relationships are specialized UML relationships used to model different types of relationships between mental states, e.g. means-ends, (de)composition, dependency, correlation, commitment, contribution, etc.

6 Contexts

AML offers the means to logically structure models according to situations that can occur during a system’s lifetime and to model elements involved in handling those situations. For this reason AML provides a modeling element called *context*, which is a specialized UML package used to contain a part of the model relevant for a particular situation. The situation is specified either as a constraint or an explicitly modeled state associated with the context.

Fig. 6 shows the interaction `PlayerSubstitution` placed within a context `Substitution of Players`. This context could also contain an activity diagram modeling the substitution algorithm, specification of necessary structural features, relationships, and capabilities of affected entity roles, etc.

7 Extension of OCL

AML defines a set of operators used to extend the OCL Standard Library [12] to include expressions belonging to modal logic (operators `possible` and `necessary`), deontic logic (operators `obliged` and `permitted`), temporal logic (operators `until`, `past`, `future`, `next`, etc.), and cognitive primitives (operators `believe`, `know`, `desire`, `intend`, `feasible`, etc.), see [1].

8 CASE Tools Support

A necessary condition of successful dissemination of a modeling language into the software engineering community is the provision of tools supporting that language. Therefore we provide an implementation of AML in two CASE tools: Rational Rose 2003 (modeling CASE tool supporting UML 1.4) and Enterprise Architect 4.0 (modeling CASE tool supporting UML 2.0). The AML implementation consists of support for UML 1.* and 2.0 profiles for AML, a set of modeling utilities (specialized element specification dialogs, model consistency checker, etc.), and forward-engineering tools for the agent platform TAP1⁶.

9 Conclusions and Further Work

AML represents a consistent framework 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. AML is also specified in accordance with OMG modeling frameworks (MDA, MOF, and UML), see Sect. 2.3. The structure of the language specification (see Sect. 2.4) together with the MDA/MOF/UML “metamodeling technology” (UML profiles, first-class metamodel extension, etc., see Sect. 2.5) gives AML

⁶ TAP1 is the commercial agent platform of Whitestein Technologies AG.

the advantage of natural extensibility and customization. In addition, AML is supported by CASE tools (see Sect. 8).

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 technology. 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.

Current status: AML is ready for use today. The AML Specification version 1.0 is available for public review and its suitability for large-scale software development projects is being validated in real customer software projects. Further evaluation and feedback is needed to identify the perceived and actual value of the work and establish contexts for future work.

Further work: In the immediate future we anticipate revising the AML specification according to feedback from the public review and ongoing commercial projects. Beyond this we intend to extend the scope of AML to incorporate additional aspects of MAS (e.g. security), and extend CASE tools support for other agent platforms (e.g. JADE).

References

1. Weiss, G.: Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence. 3rd edn. The MIT Press (2001)
2. Rao, A., Georgeff, M.: Modeling rational agents within a BDI-architecture. In Allen, J., Fikes, R., Sandewall, E., eds.: KR'91: Principles of Knowledge Representation and Reasoning. Morgan Kaufmann, San Mateo, California (1991) 473–484
3. d’Inverno, M., Luck, M.: Understanding Agent Systems. Springer-Verlag (2001)
4. Bauer, B., Muller, J., Odell, J.: Agent UML: A formalism for specifying multi-agent interaction. In Ciancarini, P., Wooldridge, M., eds.: Agent-Oriented Software Engineering. Springer-Verlag (2001) 91–103
5. Odell, J., Parunak, H., Bauer, B.: Extending UML for agents. In Wagner, G., Lesperance, Y., Yu, E., eds.: Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, Austin, Texas, ICue Publishing (2000) 3–17
6. Odell, J., Parunak, H., Fleischer, M., Brueckner, S.: Modeling agents and their environment. In: Proceedings of AOSE 2002, Bologna, Italy, Springer (2002) 16–31
7. Liu, L., Yu, E.: From requirements to architectural design using goals and scenarios. In: Software Requirements to Architectures Workshop (STRAW 2001), Toronto, Canada (2001)
8. Silva, V., Garcia, A., Brandao, A., Chavez, C., Lucena, C., Alencar, P.: Taming agents and objects in software engineering. In Garcia, A., Lucena, C., Castro, J., Omicini, A., Zambonelli, F., eds.: Software Engineering for Large-Scale Multi-Agent Systems: Research Issues and Practical Applications. Volume LNCS 2603. Springer-Verlag (2003) 1–25
9. Sturm, A., Dori, D., Shehory, O.: Single-model method for specifying multi-agent systems. In: 2nd International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2003), Melbourne, Australia (2003)

10. Wagner, G.: The agent-object-relationship meta-model: Towards a unified conceptual view of state and behavior. *Information Systems* **28** (2003)
11. OMG: Unified modeling language: Superstructure version 2.0. ptc/03-08-02 (2003)
12. OMG: UML 2.0 OCL specification. ptc/03-10-14 (2003)
13. Smith, M., McGuinness, D., Volz, R., Welty, C.: Web ontology language (OWL), guide version 1.0, W3C working draft. URL: <http://www.w3.org/TR/2002/WD-owl-guide-20021104> (2002)
14. Cranefield, S., Haustein, S., Purvis, M.: UML-based ontology modelling for software agents. In: *Proceedings of the Workshop on Ontologies in Agent, 2001*. (2001)
15. Martin, D.e.: OWL-S 1.0 release. URL: <http://www.daml.org/services/> (2003)
16. Evans, R., Kearny, P., Stark, J., Caire, G., Garijo, F., Sanz, G., Leal, F., Chainho, P., Massonet, P.: MESSAGE: Methodology for engineering systems of software agents. Technical Report P907, EURESCOM (2001)
17. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: the gaia methodology. *ACM Trans on Software Engineering and Methodology* **12** (2003) 317–370
18. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: TROPOS: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems* **2** (2004) 203–236
19. Cossentino, M., Sabatucci, L., Chella, A.: A possible approach to the development of robotic multi-agent systems. In: *IEEE/WIC Conference on Intelligent Agent Technology (IAT'03)*, Halifax, Canada (2003)
20. Padgham, L., Winikoff, M.: Prometheus: A methodology for developing intelligent agents. In: *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, Bologna, Italy (2002)
21. Iglesias, C., Garijo, M., Gonzalez, J., Velasco, J.: Analysis and design of multiagent systems using MAS-CommonKADS. In Singh, M., Rao, A., Wooldridge, M., eds.: *Intelligent Agents IV (LNAI Vol. 1365)*. Volume 1365. Springer-Verlag (1998) 313–326
22. DeLoach, S.: Multiagent systems engineering: A methodology and language for designing agent systems. In: *Agent-Oriented Information Systems '99 (AOIS'99)*, Seattle, WA (1999)
23. Turski, W., Maibaum, T.: *The Specification of Computer Programs*. Addison-Wesley (1987)
24. OMG: Meta object facility (MOF) specification. Version 1.4, formal/2002-04-03 (2002)
25. The Foundation for Intelligent Physical Agents: FIPA specifications repository. URL: <http://www.fipa.org/repository/index.html> (2004)
26. Jennings, N.R.: On agent-based software engineering. *Artificial Intelligence* **117** (2000) 277–296
27. Jennings, N., Wooldridge, M.: Software agents. *IEEE Review* **42** (1996) 17–21
28. Wooldridge, M., Ciancarini, P.: Agent-oriented software engineering: The state of the art. In: *Handbook of Software Engineering and Knowledge Engineering*. World Scientific Publishing Co. (2001)