Agent Modeling Language

Toward Industry-Grade Agent-Based Modeling

AL3-TF2, AOSE TFG, Ljubljana
February 28th - March 1st, 2005
Overview

- Goals and scope of AML
- Structure of AML
- Modeling examples
- CASE tool support
- Conclusions
What is AML?

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

Goals:

- Built on proven technical foundations.
- Integrates best practices from agent-oriented software engineering (AOSE) and object-oriented software engineering (OOSE) domains.
- Well specified and documented.
- Internally consistent from the conceptual, semantic and syntactic perspectives.
- Versatile and easy to extend.
- Independent of any particular theory, development process or implementation environment.
- Supported by Computer-Aided Software Engineering (CASE) tools.
Scope of AML

Two dimensions:

- Support for the human mental process of requirements specification and analysis of complex problems/systems.
  - Mental aspects, which can be used for modeling intentionality in use case models, goal-based requirements, problem decomposition, etc.
  - Contexts, which can be used for situation-based modeling.

- Support for the abstraction of architectural and behavioral concepts associated with multi-agent systems.
  - MAS entities
  - social aspects
  - ontologies
  - MAS deployment
  - agent mobility
  - behavior abstraction and decomposition
  - observations and effecting interactions
  - communicative interactions
  - services
  - mental aspects
Applicability of AML

- 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 to systems that:
  - Consist of a number of autonomous, concurrent and/or asynchronous (possibly proactive) entities.
  - Comprise entities that are able to observe and/or interact with their environment.
  - Make use of complex interactions and aggregated services.
  - Employ social structures.
  - Capture mental characteristics of systems and/or their parts.
Structure of AML

UML 2.0 as a base

- 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.
Extensibility of AML

Extension mechanisms

- **Metamodel extension.** This offers first-class extensibility (as defined by MOF) 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 means to employ alternative MAS modeling approaches as complementary specifications to the AML model.
Fundamental Entity Types

- **Agent type**

- **Resource type**
Environment type

- **3DObject**
  - shape
  - changeShape()
- **3DObject**
  - object
- **3DObject**
  - space
- **3DObject**
  - *
- **3DObject**
  - *

**Environment types**

- **3DPlacement**
  - position
- **3DPlacement**
  - space

**Pitch**

- **Pitch**
  - person: Person[*] {subsets object}
  - goal: Goal[2] {subsets object}
  - ball: Ball {subsets object}
Organization unit type and social relationships

- **ProjectBoard** — Supervises  
- **ProjectManager**  
  - Manages  
  - Assists  
- **ProjectAssuranceTeam**  
- **TechnicalTeam**  
- **AnalysisTeam**  
- **ImplementationTeam**  
- **TestingTeam**  
- **DeploymentTeam**

- **Social Aspects**
  - Superordinate
  - Subordinate
  - Peer
Social Aspects (cont.)

Entity role type and play association

- CooksFor
- DrivesFor
- Serves

Master

Valet

Driver

Cook

Person

Oliver:Person

John:Person

Mary:Person

Cook:Cook

Valet:Valet

Driver:Driver

Master:Master

Entity role type

Class level

Instance level

player type

play association

entity role

player
Role manipulation actions

- Sell security
- Register seller (marketMember)
- Try to match sell order
- Accept matching
- Settle

create role action

dispose role action
Ontologies
Ontologies (cont.)

 Ontology class

```
ontology Medical Staff Ontology

Person
  name

Patient
  insuranceType

Doctor
  skills

Nurse

Surgeon
  surgeon *
  department 1
  clinic

Anesthetist
  anesthetist *
  department 1
  anesthesia

Anesthesia Nurse
  * nurse
  department

Ward Nurse
```


MAS Deployment and Mobility

Agent execution environment, hosting attributes, and mobility relationships

- StockExchangeServer
  - server: TradingServer
  - orderPool: OrderPool
  - account: Account[*]
  - : LoadBalanceManager
  - broker: Broker[*]
    - {visitor}

- BackupStockExchangeServer
  - server: TradingServer
  - orderPool: OrderPool
  - account: Account[*]
  - : LoadBalanceManager
  - broker: Broker[*]
    - {visitor}

- ClientPC
  - : TradingClient
  - broker: Broker[*]
    - {resident}
Behavior Abstraction and Decomposition

- Capability = BehavioralFeature or Behavior

- Behavior fragment

- Behavior Abstraction and Decomposition
Communicative Interactions

- Multi-lifeline, multi-message, and subset

- initiator
- participants [m]

- multi-lifelines

- cfp
- refusers [n-j]
- referees [n-j]

- subset relationship

- propose
- refuse

- multi-messages

- reject-proposal
- accept-proposal

- alt
- inform-done [single]

- inform-result [single]

- failure [single]
Join

Attribute change

owner’s lifeline

entity role’s lifeline

attribute change

created entity role’s lifeline

entity role’s lifeline

created entity role’s lifeline

entity role’s lifeline
Communicative Interactions (cont.)

- Communicaton message, communicative interaction, and interaction protocol

```
sd FIPA-Request-Protocol
  acl=FIPA ACL
  cl=FIPA SL
  encoding=XML

:Initiator
  request(action)
  decideAbout(action)

  alt
    [refuse decision]
      refuse
    [else]
      opt
        [notification is necessary]
          agree
      failure
      inform-done

:Participant

Initiator : AutonomousEntityType
Participant : AutonomousEntityType
action : String
decideAbout : Capability
perform : Capability
```

Formal parameters:

- Initiator : AutonomousEntityType
- Participant : AutonomousEntityType
- action : String
- decideAbout : Capability
- perform : Capability

IP binding:

```
<bind>
  <Initiator -> PersonalAssistant
  Participant -> PortfolioManager
  action -> "buy(what, amount, price)
  decideAbout -> buyingAssessment
  perform -> do>
```

```
sd FIPA-Request-Protocol

sd BuyRequest
```
Services

- Service specification

<table>
<thead>
<tr>
<th>Service</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectoryFacilitator</td>
<td>{acl=fipa-acl, cl=fipa-sl0, ontology=fipa-agent-management}</td>
</tr>
<tr>
<td>sd Register:Fipa-Request-Protocol</td>
<td>&lt;action-&gt;register&gt;</td>
</tr>
<tr>
<td>sd Deregister:Fipa-Request-Protocol</td>
<td>&lt;action-&gt;deregister&gt;</td>
</tr>
<tr>
<td>sd Modify:Fipa-Request-Protocol</td>
<td>&lt;action-&gt;modify&gt;</td>
</tr>
<tr>
<td>sd Search:Fipa-Request-Protocol</td>
<td>&lt;action-&gt;search&gt;</td>
</tr>
</tbody>
</table>

Service protocols:
- Register
- Deregister
- Modify
- Search
Service provision and usage

- AgentManagement[1]
- DF
- AMS
- MTS
- DFProxy
- AMSProxy
- MTS
- :Agent[*]
- ap2apIn [0..1]
- ap2apOut [0..1]

- Service provision
- Service usage
Observations and Effecting Interactions

- Perceiving act, perceptor type, perceptor, and percepts
- Effecting act, effector type, effector, and effects
Percept action and effect action

- isOnFloor(blue) <<percept>> [yes] [no]
- putOnFloor(blue) <<effect>> [yes] [no]
- putOn(blue, red) <<effect>> [yes] [no]
Beliefs, goals, plans, and mental relationships

<table>
<thead>
<tr>
<th>TeammateHasBall</th>
<th>{degree=1}</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>commit</strong></td>
<td>{free(teamMate) AND notOffSide(teamMate)}</td>
</tr>
<tr>
<td><strong>cancel</strong></td>
<td>{NOT(free(teamMate)) OR NOT(notOffSide(teamMate))}</td>
</tr>
<tr>
<td><strong>pre</strong></td>
<td>{kickable(ball)}</td>
</tr>
<tr>
<td><strong>post</strong></td>
<td>{hasBall(teamMate)}</td>
</tr>
</tbody>
</table>

Mental States

- Beliefs
- Goals
- Plans
- Contributions
- Relationships
Ownership of and responsibility for mental states

- Mental attribute
- Responsible relationship
- Mental association

<table>
<thead>
<tr>
<th>OffenderRole</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>&lt;&lt;mental&gt;&gt; scoreGoal:ScoreGoal {degree=1.0}</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

- Person
- Cup
- Bottle
- Glass

- StoreFluid
- mental attribute

- ScoreGoal
Contexts

- Context

Substitution

SubstitutionAlgorithm

Coach

Player

Referee

Context's content

prepareSubstitution()
selectSubstitutedPlayers()
controlSubstitution()

prepareForSubstitution()
exit()
enter()

registerSubstitution()
superviseSubstitution()
CASE Tools Support

- AML add-in implementation in Enterprise Architect and IBM Rational Rose.
- Code generator for Living Systems® Technology Suite.
Conclusions and Further Work

Current Status

- Ready for use.
- Supported by CASE tools - Enterprise Architect and IBM Rational Rose.
- Used in commercial software projects.
- Specification version 1.0 will be soon available for public review.
- Further evaluation and feedback is needed.

Further Work

- Revision according to feedback from the public review and ongoing commercial projects.
- Extension of the scope of AML to incorporate additional aspects of MAS.
- Extension of the CASE tools support for other agent platforms (e.g. JADE).
- Specification of an AML-based software development methodology.