WHITESTEIN
Technologies

Agent Modeling Language

Toward Industry-Grade Agent-Based Modeling

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

wt.aml.AL3TF2 | v1.0 | 2005-02-23 | RCE, ITR

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.

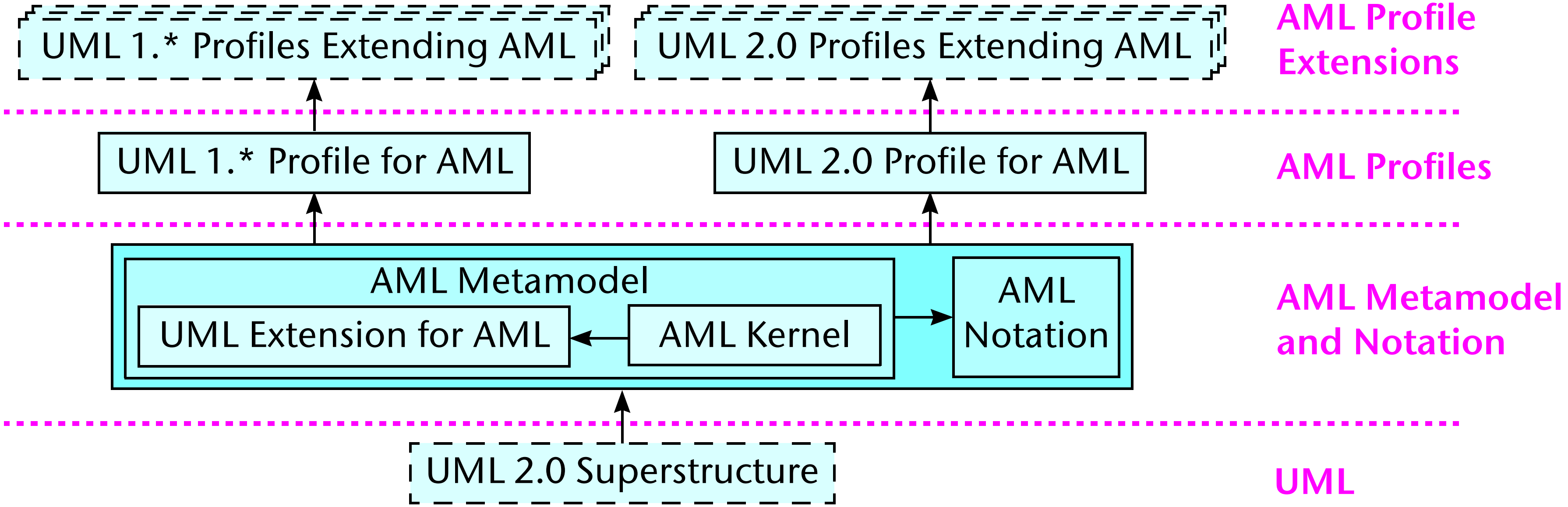
❒ AML can however also be applied to other domains such as business systems, social systems, robotics, etc., in general to systems that:

- 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

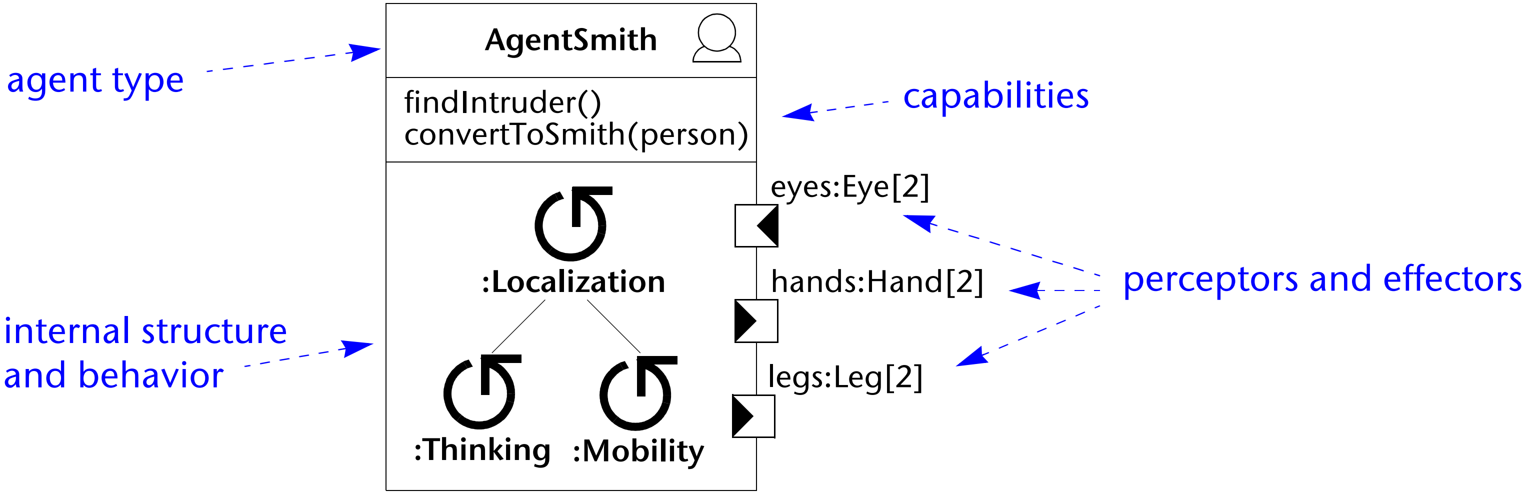
❐ ***Metamodel extension.*** This offers first-class extensibility (as defined by MOF) of the AML metamodel and notation.

❐ ***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.

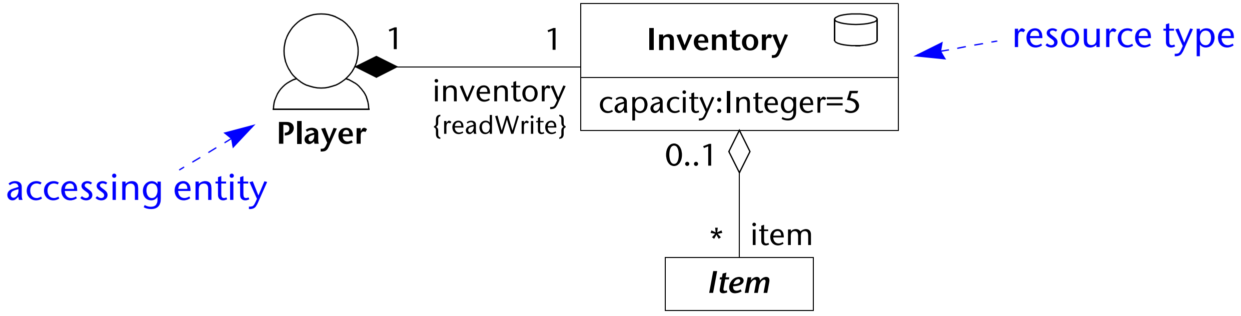
❐ ***Concrete model extension.*** This offers the means to employ alternative MAS modeling approaches as complementary specifications to the AML model.

Fundamental Entity Types

- Agent type

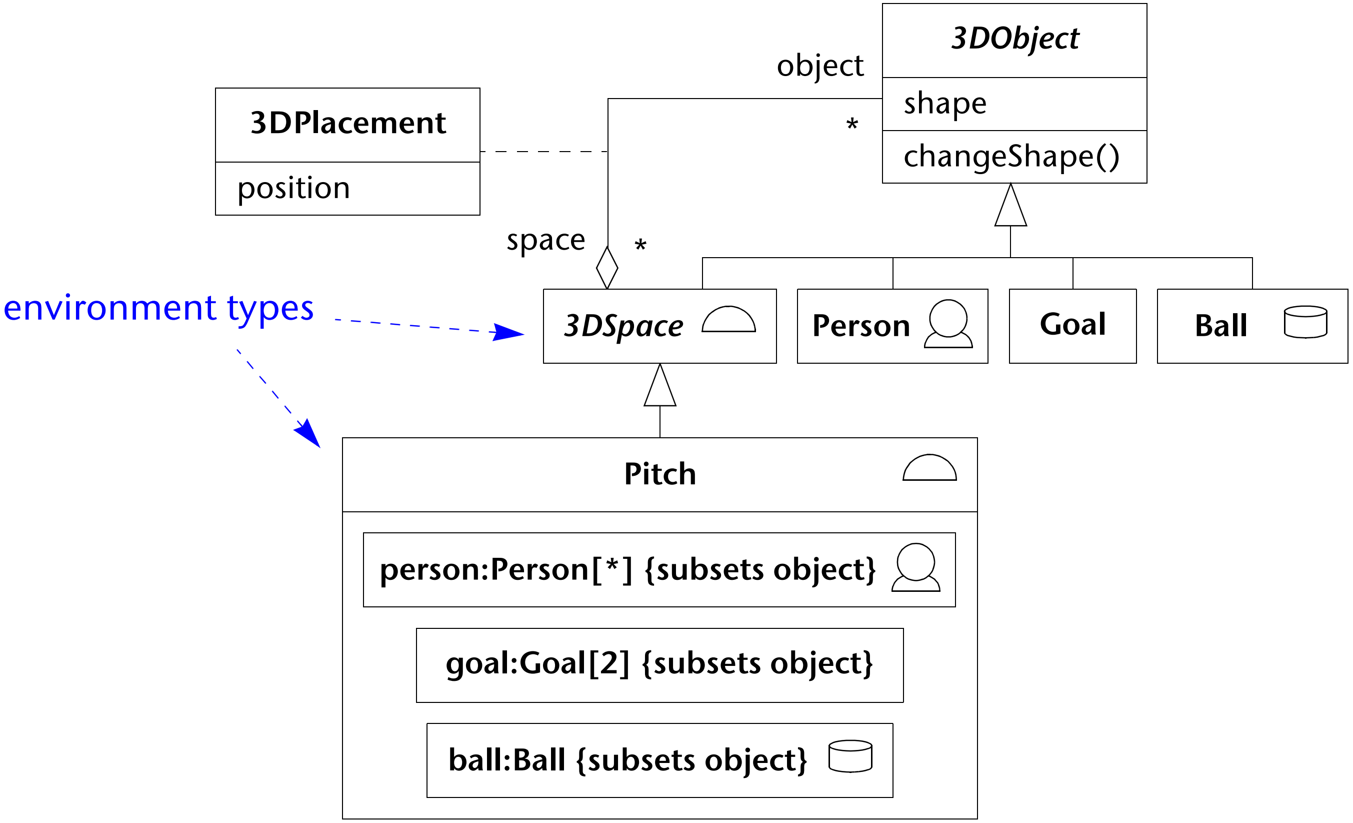


- Resource type



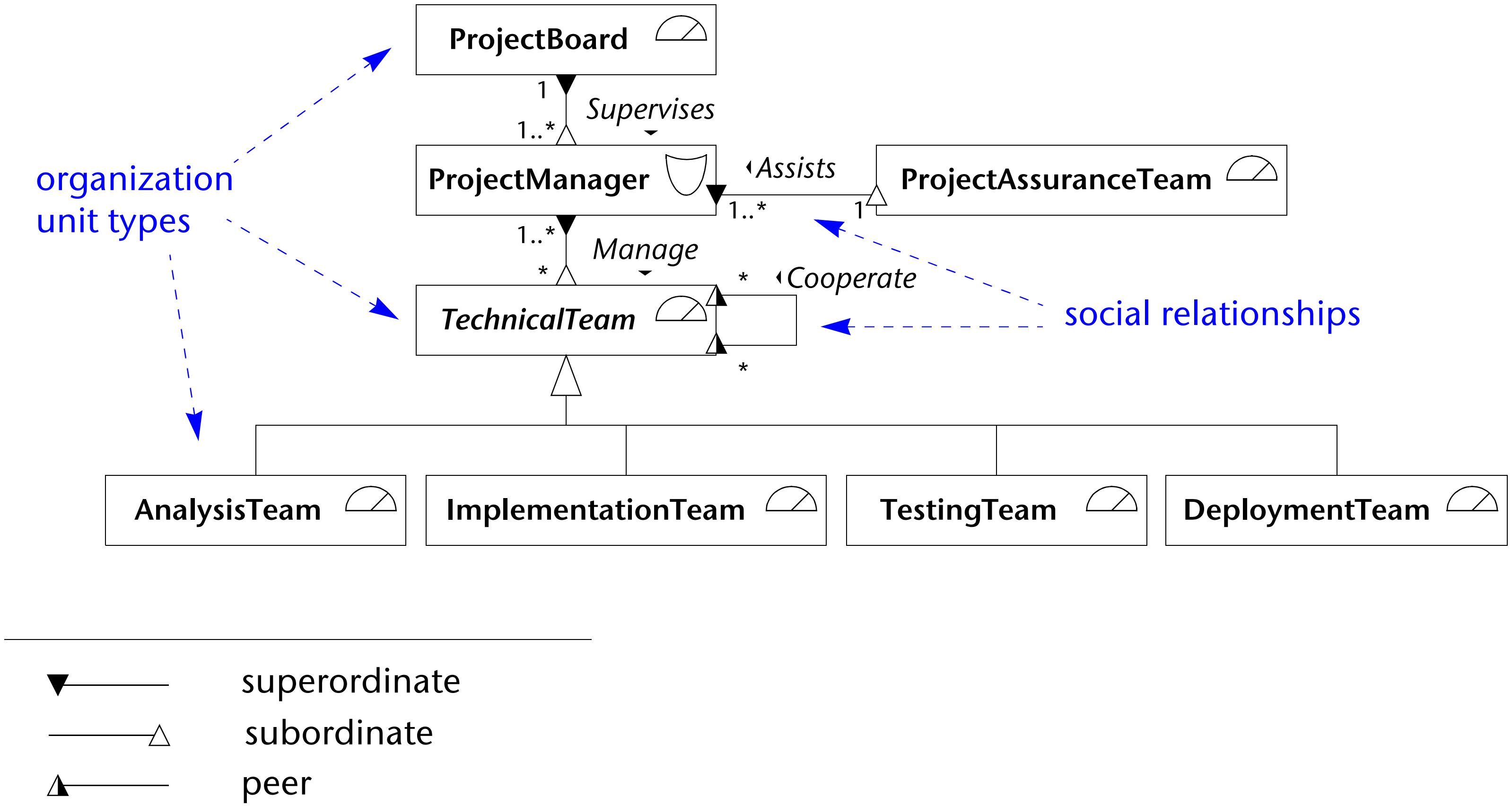
Fundamental Entity Types (cont.)

- Environment type



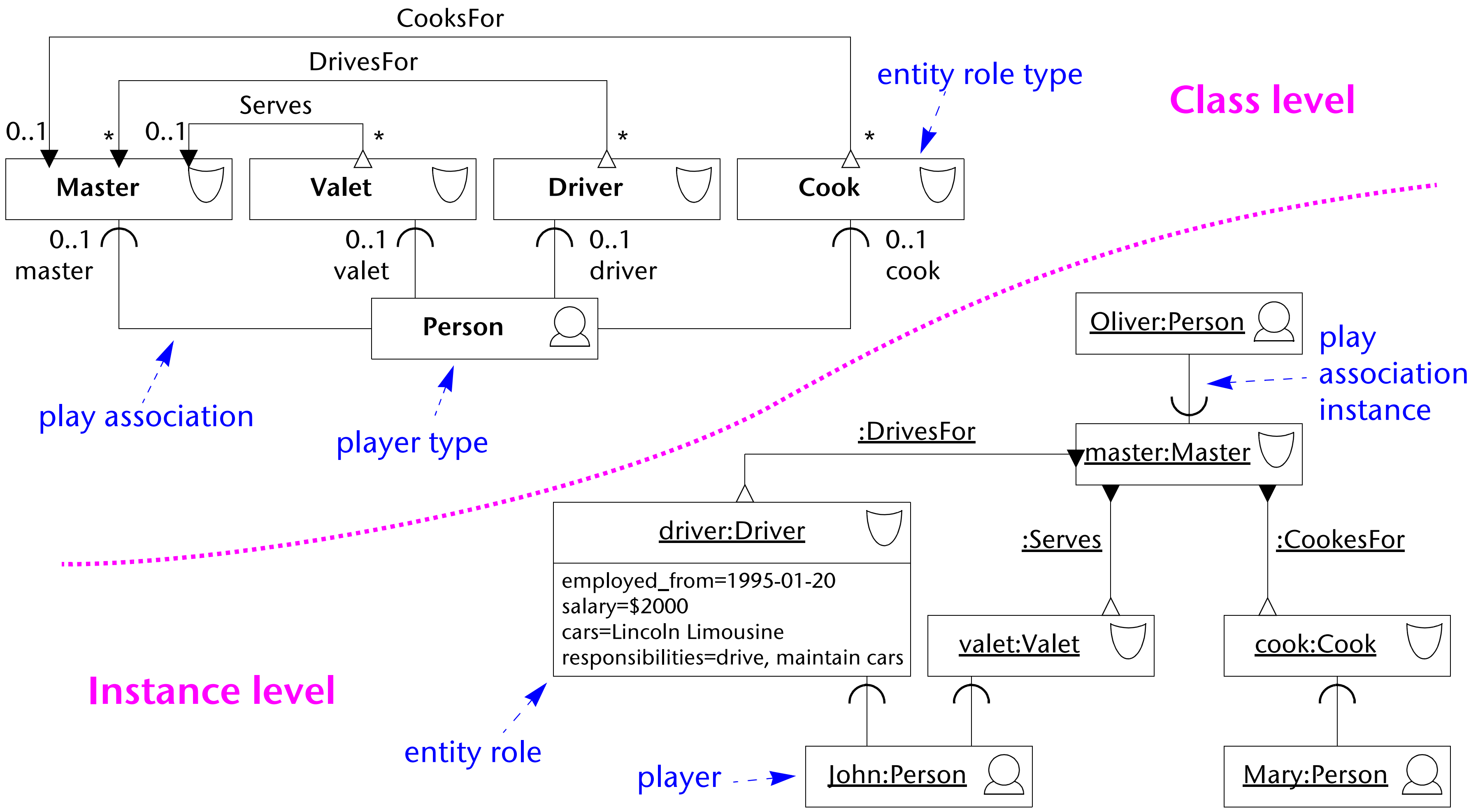
Social Aspects

- Organization unit type and social relationships



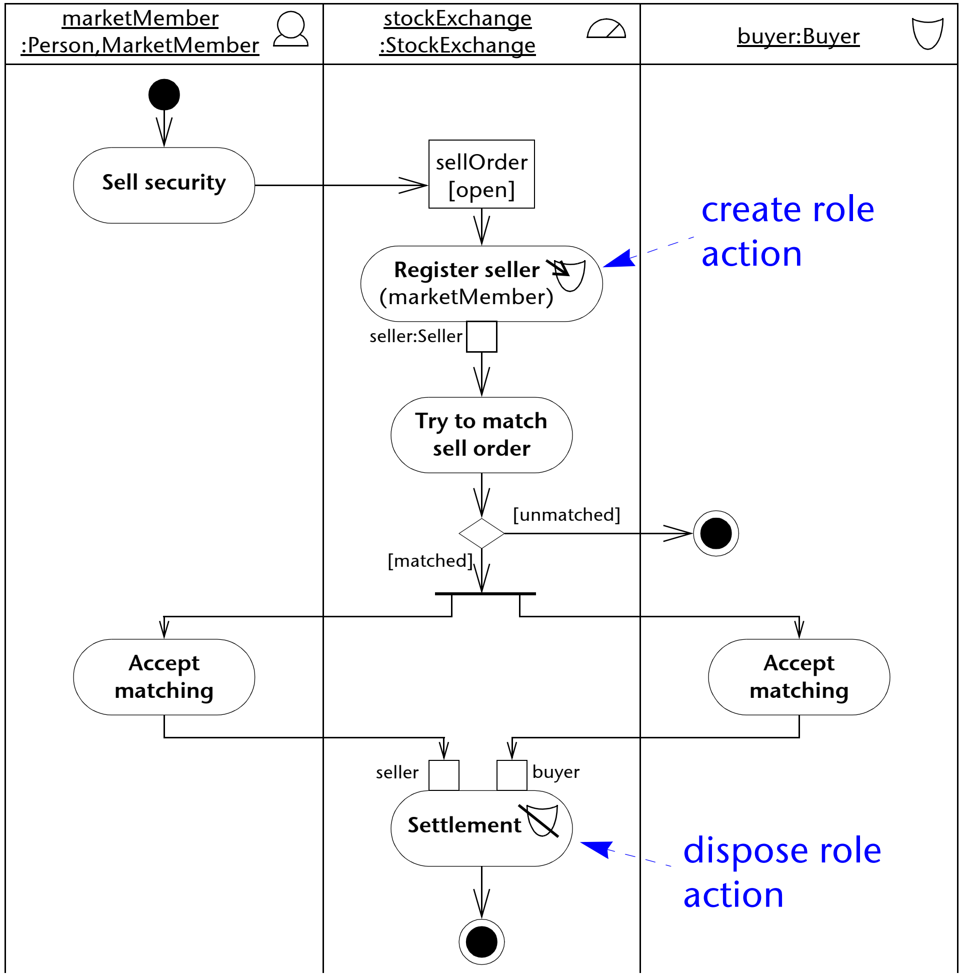
Social Aspects (cont.)

❒ Entity role type and play association



Social Aspects (cont.)

- Role manipulation actions

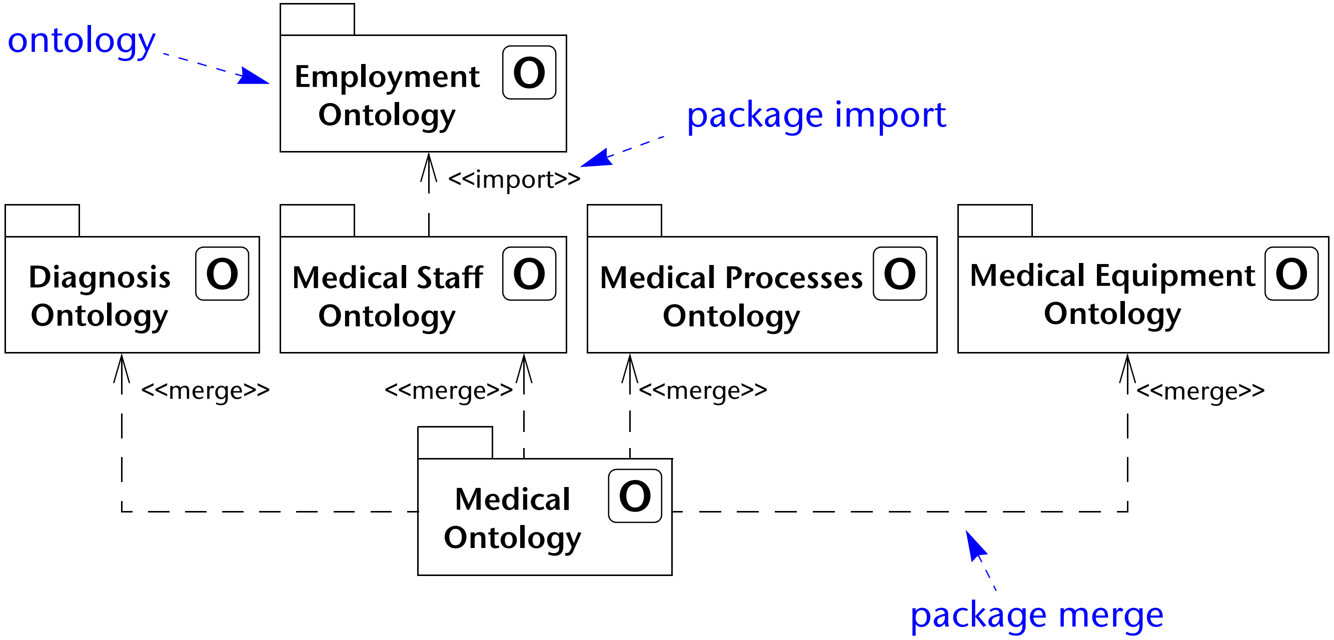


create role action

dispose role action

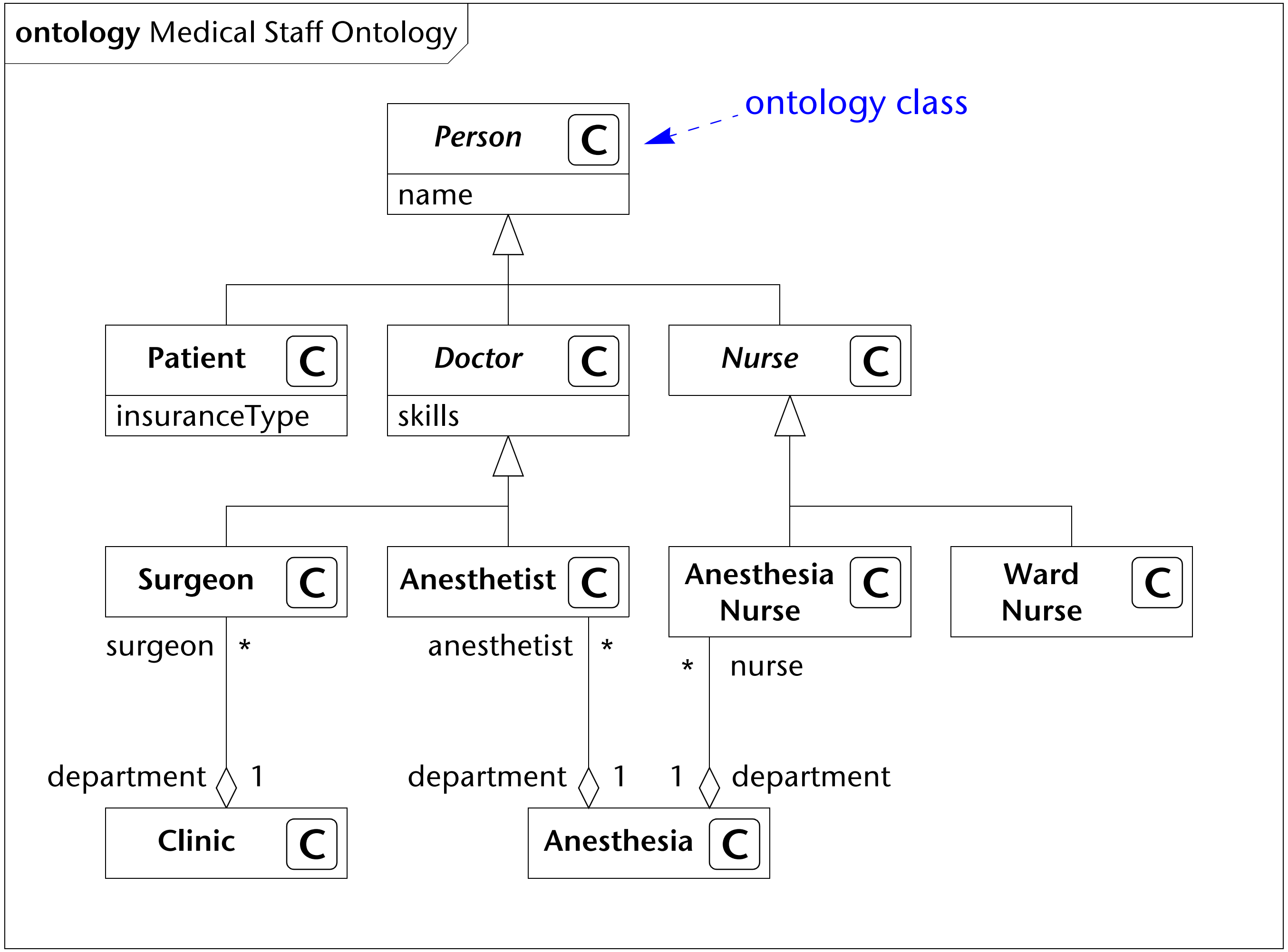
Ontologies

- Ontology



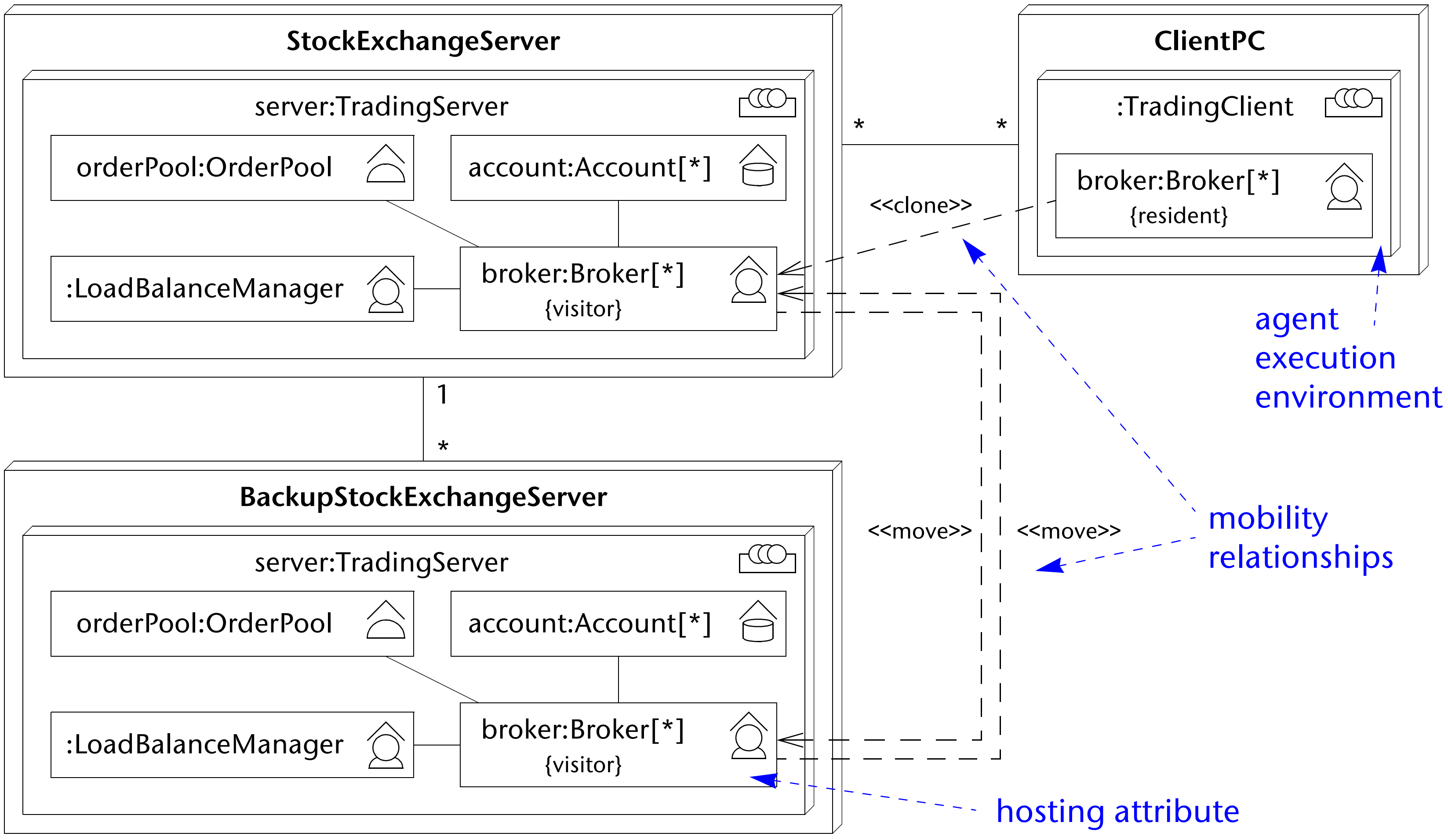
Ontologies (cont.)

- ❒ Ontology class



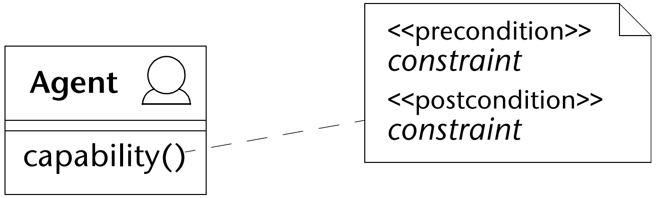
MAS Deployment and Mobility

❒ Agent execution environment, hosting attributes, and mobility relationships



Behavior Abstraction and Decomposition

- Capability = BehavioralFeature or Behavior



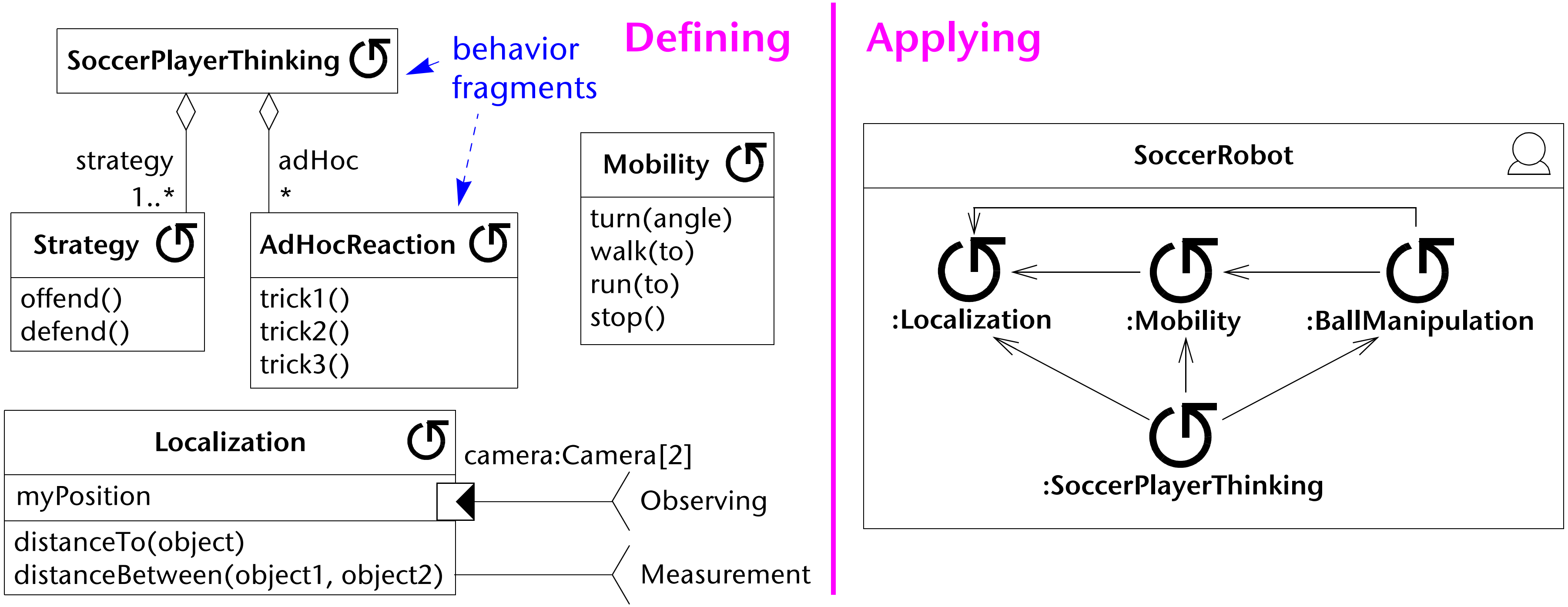
Agent

capability()

<<precondition>>
constraint

<<postcondition>>
constraint

- Behavior fragment



behavior fragments

Defining

Applying

SoccerPlayerThinking

strategy 1..*

adHoc *

Strategy

offend()
defend()

AdHocReaction

trick1()
trick2()
trick3()

Mobility

turn(angle)
walk(to)
run(to)
stop()

Localization

myPosition

distanceTo(object)
distanceBetween(object1, object2)

camera:Camera[2]

Observing

Measurement

SoccerRobot

:Localization

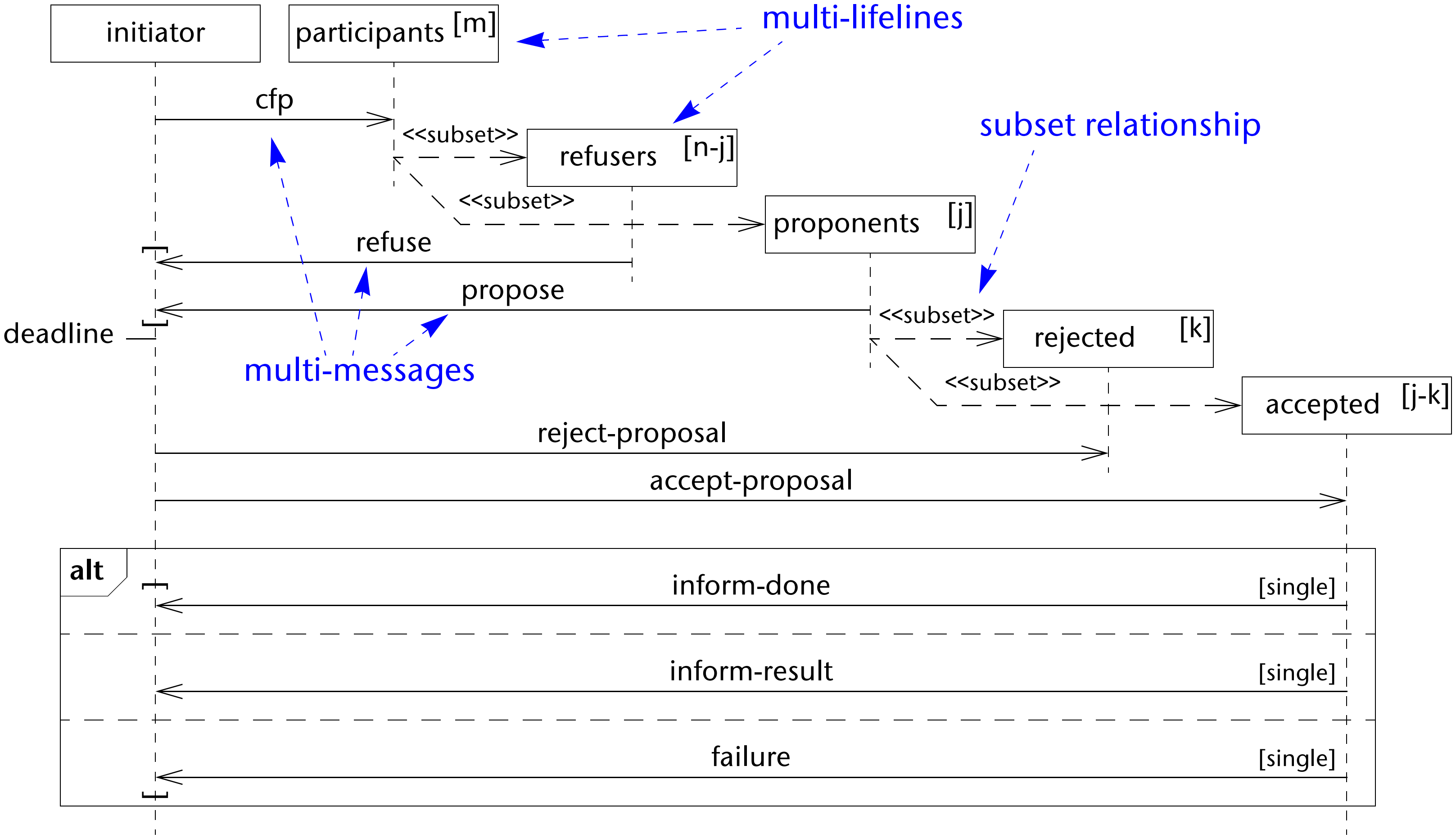
:Mobility

:BallManipulation

:SoccerPlayerThinking

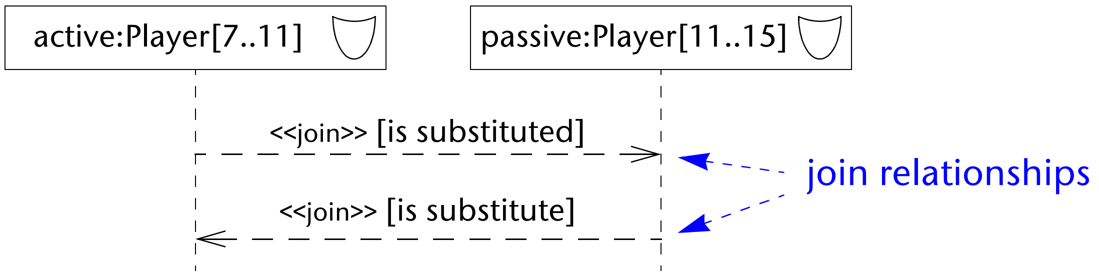
Communicative Interactions

- Multi-lifeline, multi-message, and subset

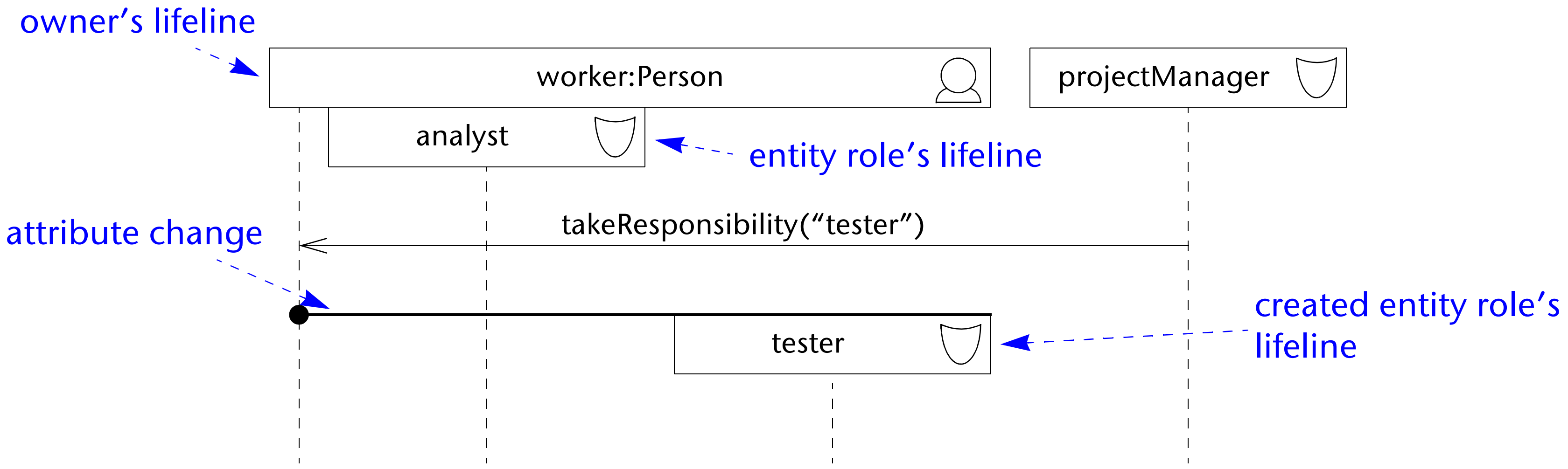


Communicative Interactions (cont.)

- ❐ Join

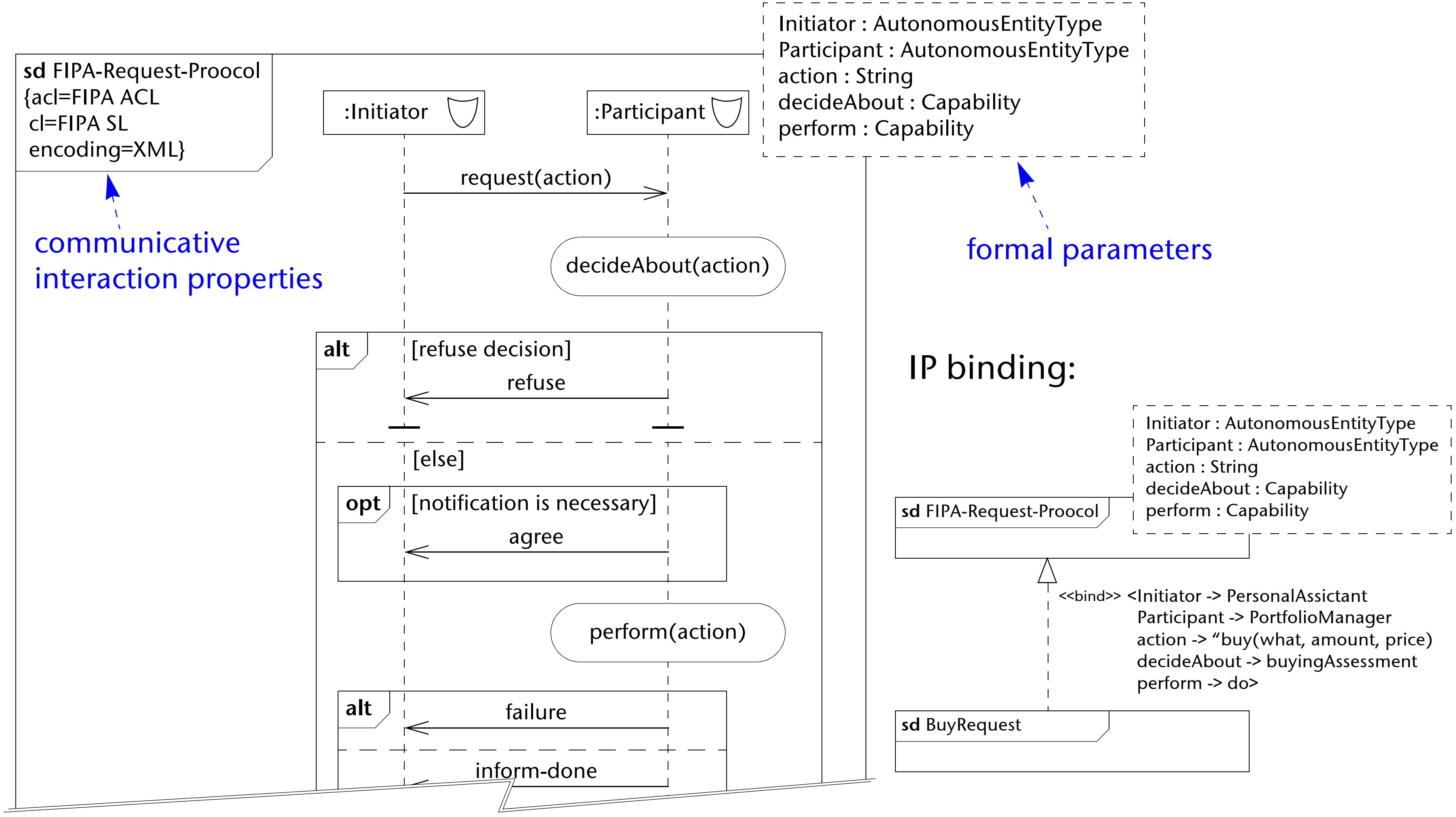


- ❐ Attribute change



Communicative Interactions (cont.)

- Communicaton message, communicative interaction, and interaction protocol



sd FIPA-Request-Proocol
{acl=FIPA ACL
cl=FIPA SL
encoding=XML}

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

:Initiator

:Participant

request(action)

decideAbout(action)

alt [refuse decision]

refuse

[else]

opt [notification is necessary]

agree

perform(action)

alt

failure

inform-done

communicative interaction properties

formal parameters

IP binding:

sd FIPA-Request-Proocol

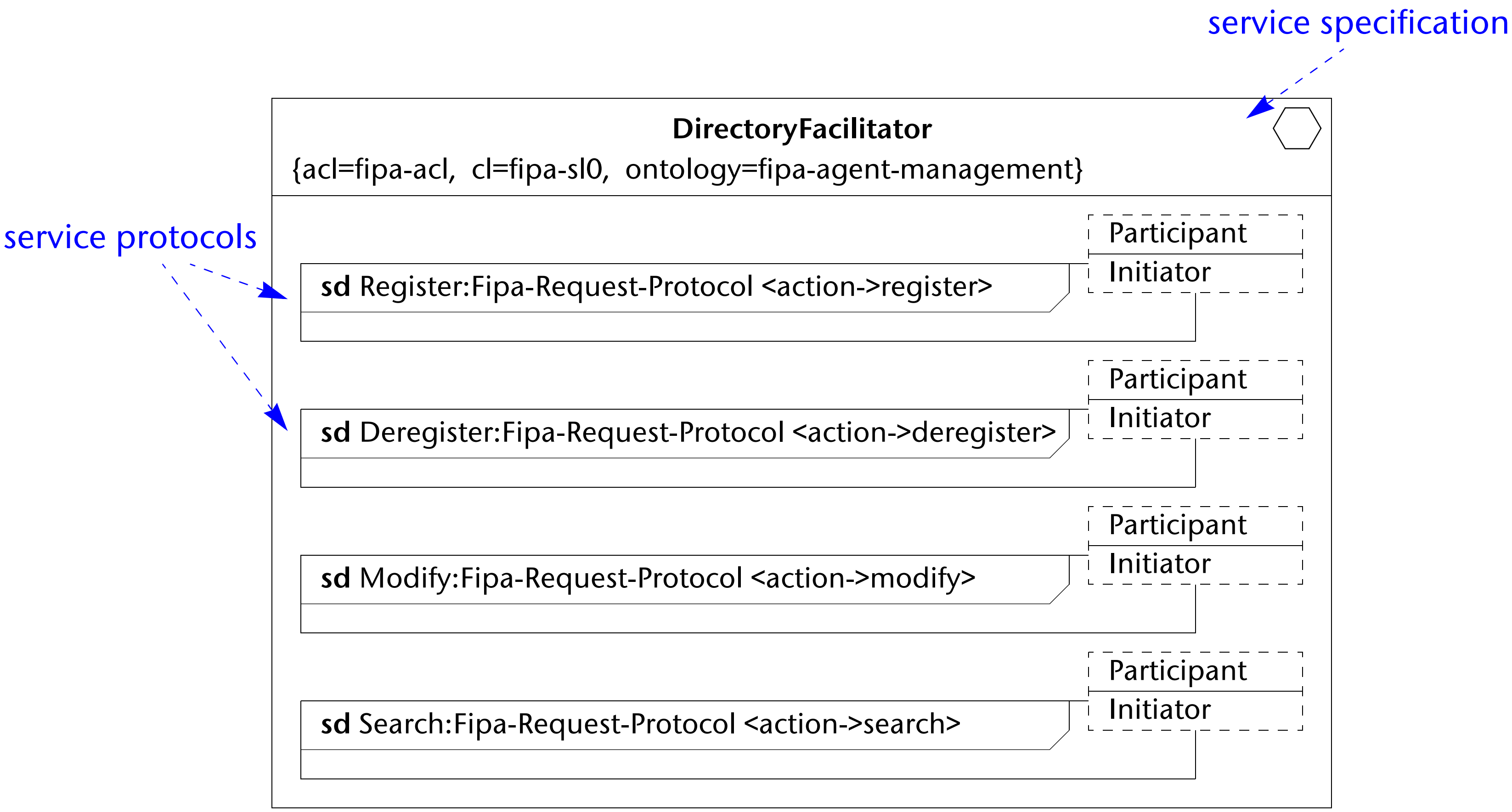
Initiator : AutonomousEntityType
Participant : AutonomousEntityType
action : String
decideAbout : Capability
perform : Capability

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

sd BuyRequest

Services

❒ Service specification



service specification

service protocols

DirectoryFacilitator

{acl=fipa-acl, cl=fipa-sl0, ontology=fipa-agent-management}

sd Register:Fipa-Request-Protocol <action->register>

Participant
Initiator

sd Deregister:Fipa-Request-Protocol <action->deregister>

Participant
Initiator

sd Modify:Fipa-Request-Protocol <action->modify>

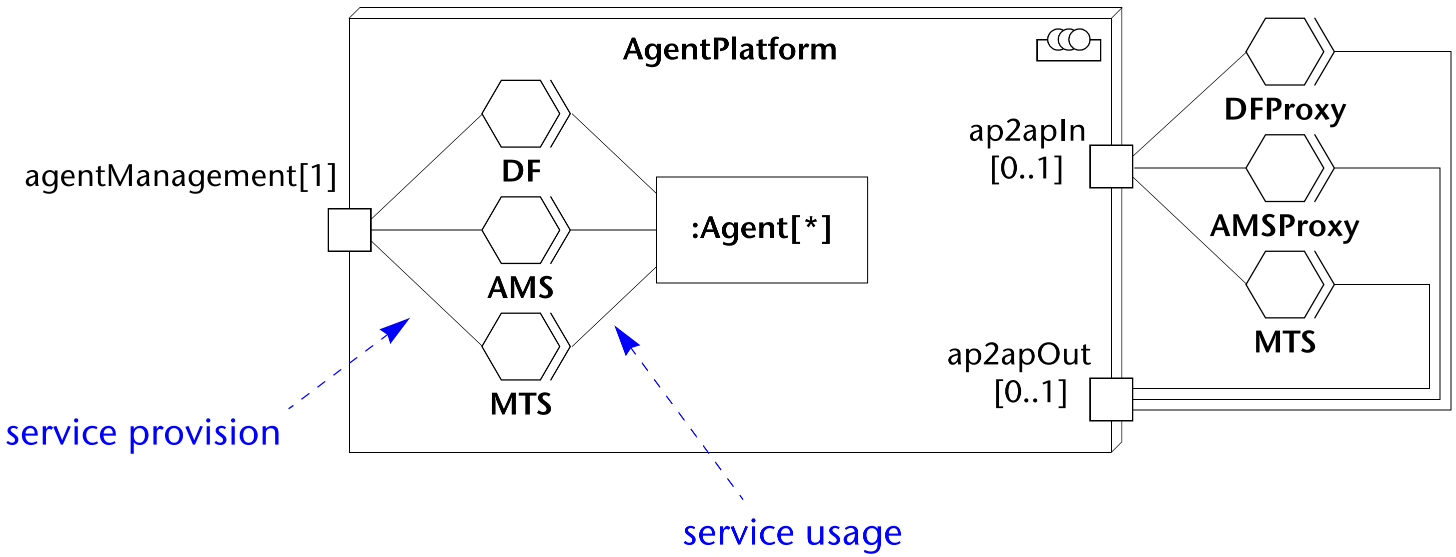
Participant
Initiator

sd Search:Fipa-Request-Protocol <action->search>

Participant
Initiator

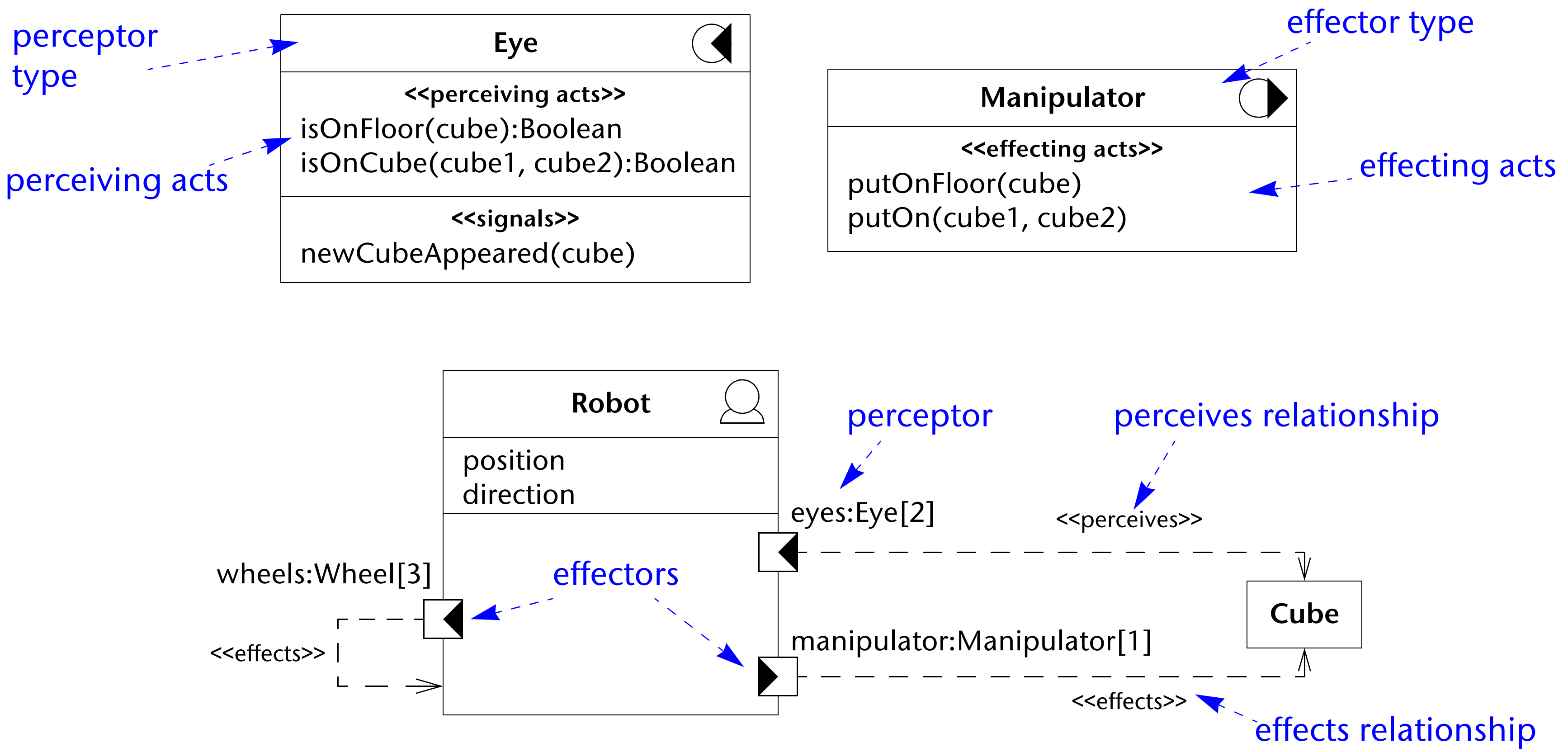
Services (cont.)

- ❐ Service provision and usage



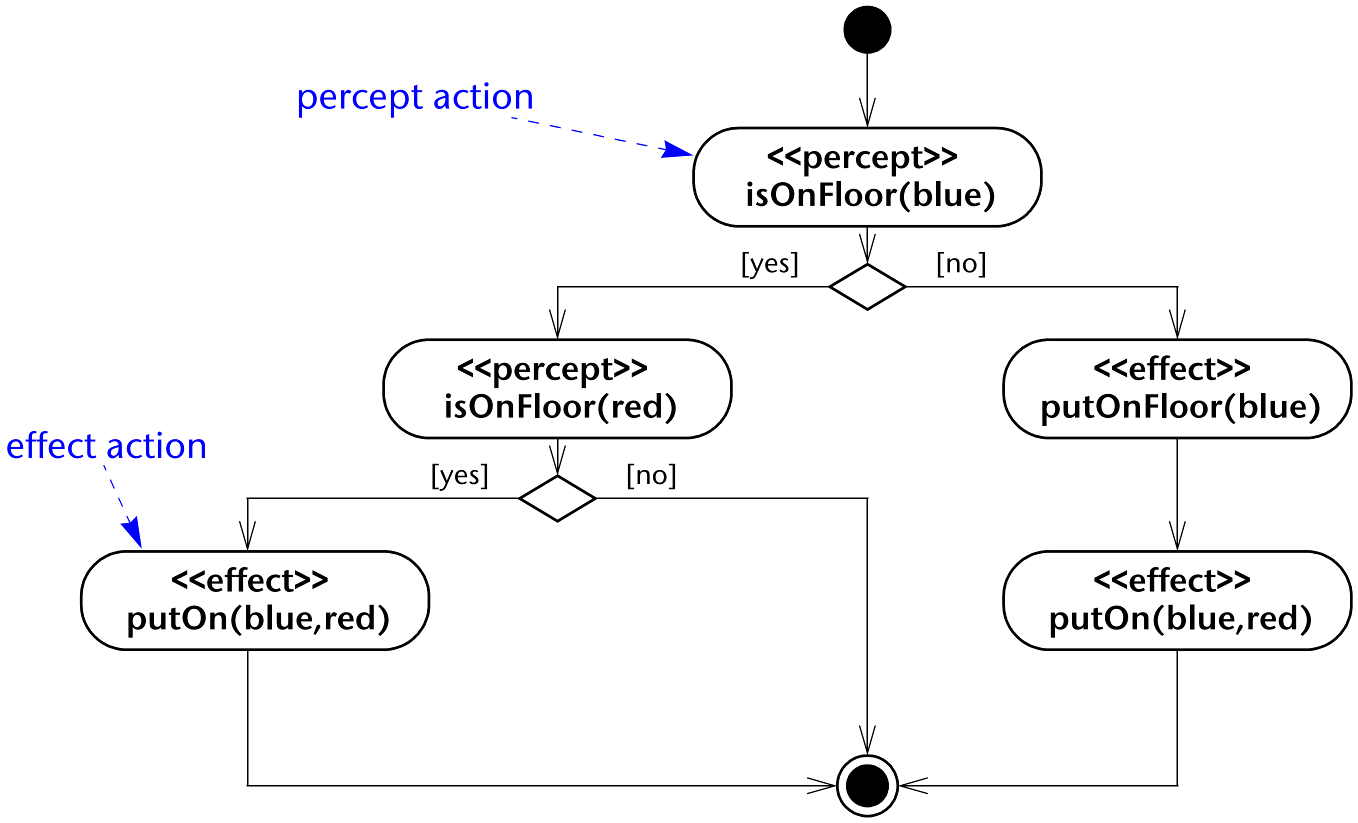
Observations and Effecting Interactions

- ❒ Perceiving act, perceptor type, perceptor, and percepts
- ❒ Effecting act, effector type, effector, and effects



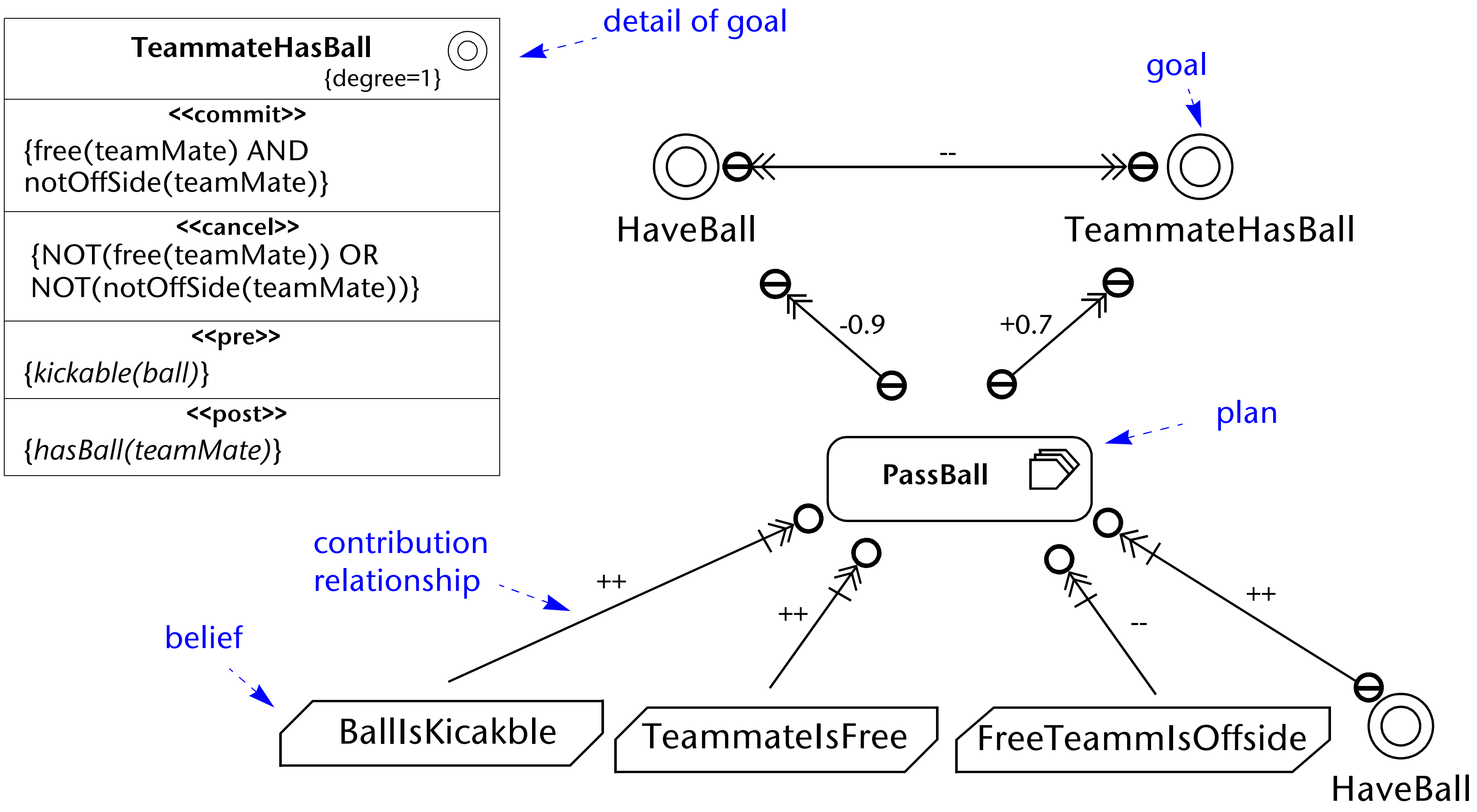
Observations and Effecting Interactions (cont.)

- ❒ Percept action and effect action


percept action
effect action
<<percept>> isOnFloor(blue)
[yes]
[no]
<<percept>> isOnFloor(red)
<<effect>> putOnFloor(blue)
[yes]
[no]
<<effect>> putOn(blue,red)
<<effect>> putOn(blue,red)

Mental States

❒ Beliefs, goals, plans, and mental relationships



TeammateHasBall
{degree=1}

<<commit>>
{free(teamMate) AND notOffSide(teamMate)}

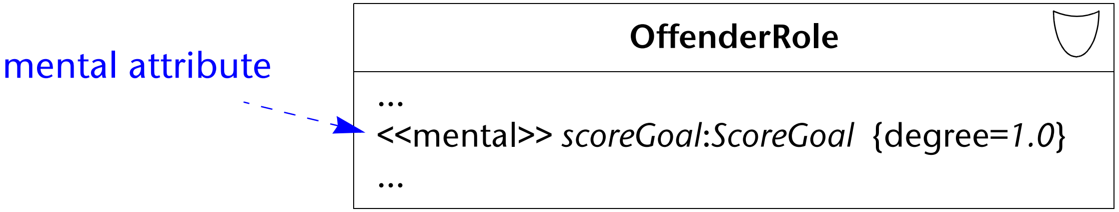
<<cancel>>
{NOT(free(teamMate)) OR NOT(notOffSide(teamMate))}

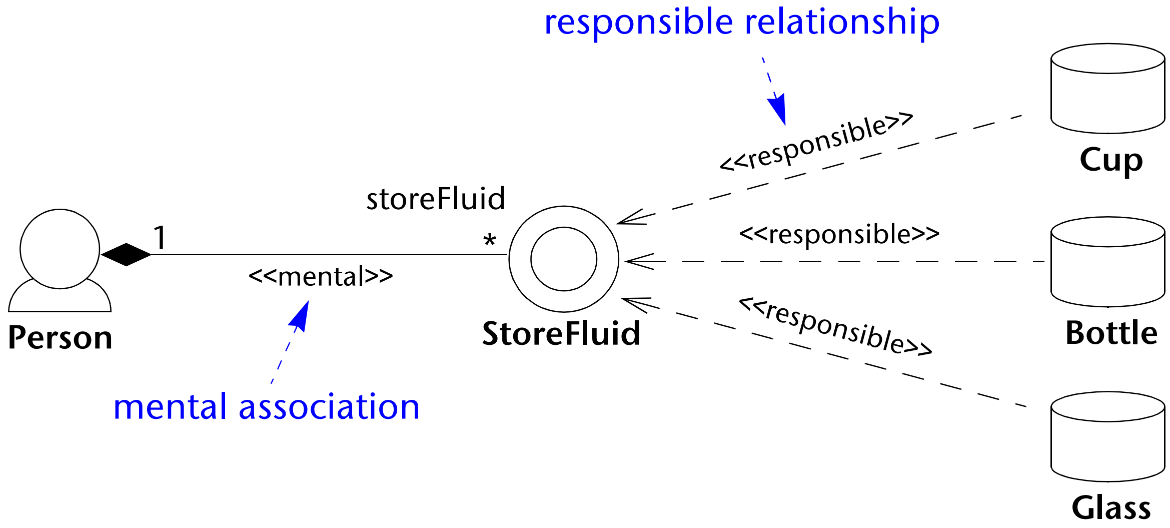
<<pre>>
{kickable(ball)}

<<post>>
{hasBall(teamMate)}

Mental States (cont.)

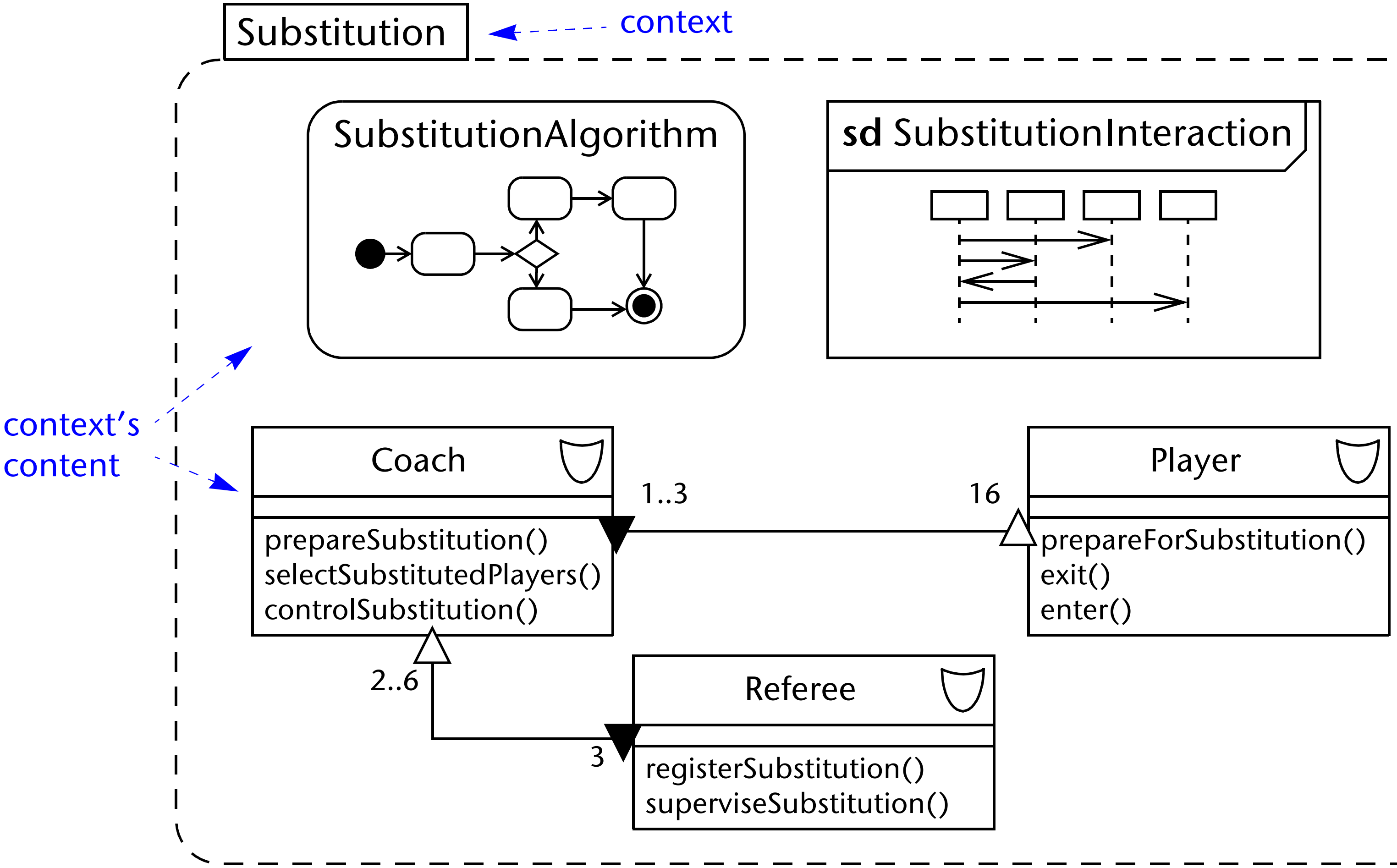
- Ownership of and responsibility for mental states





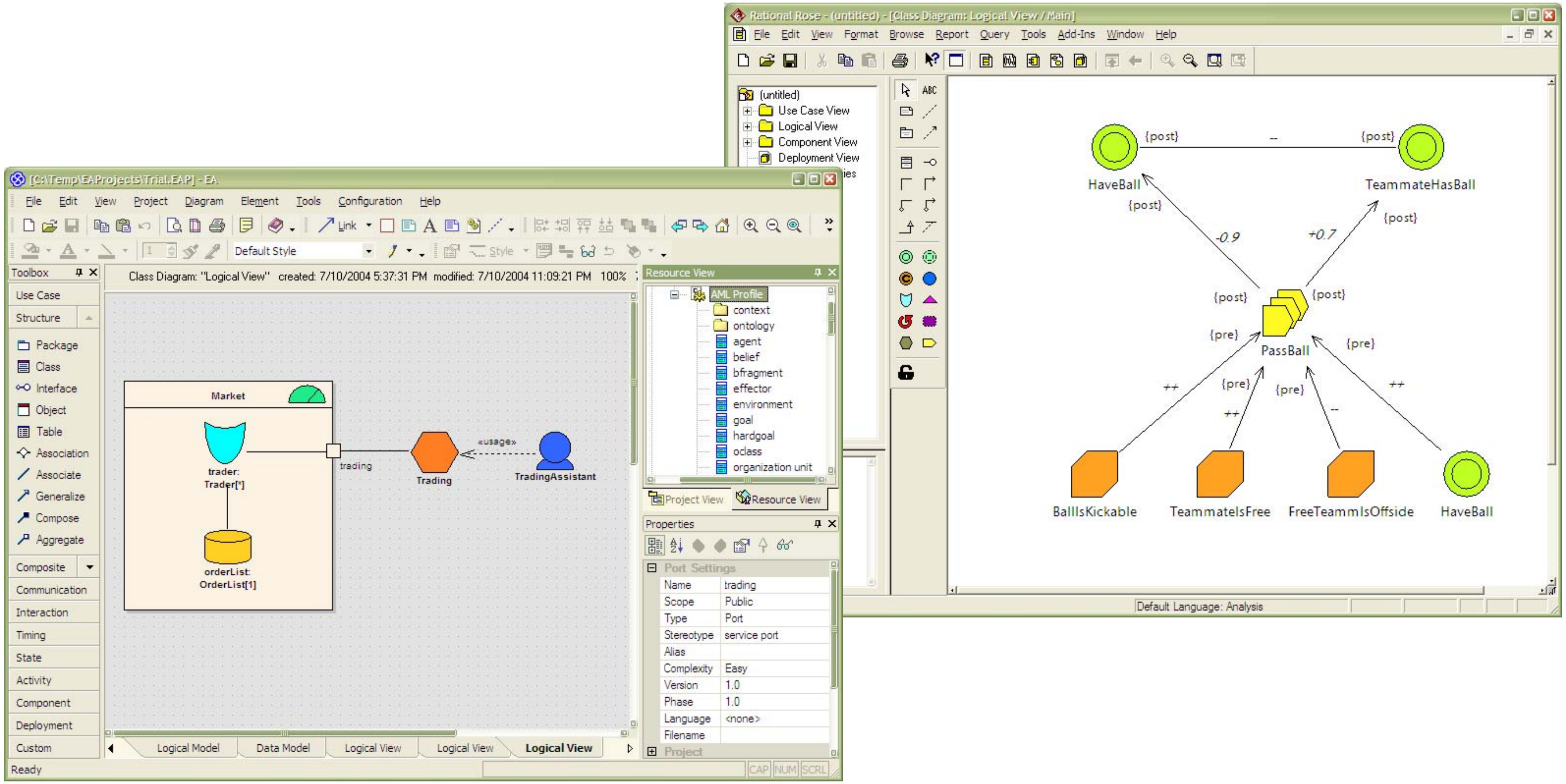
Contexts

- Context



CASE Tools Support

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



Name	trading
Scope	Public
Type	Port
Stereotype	service port
Alias	
Complexity	Easy
Version	1.0
Phase	1.0
Language	<none>
Filename	

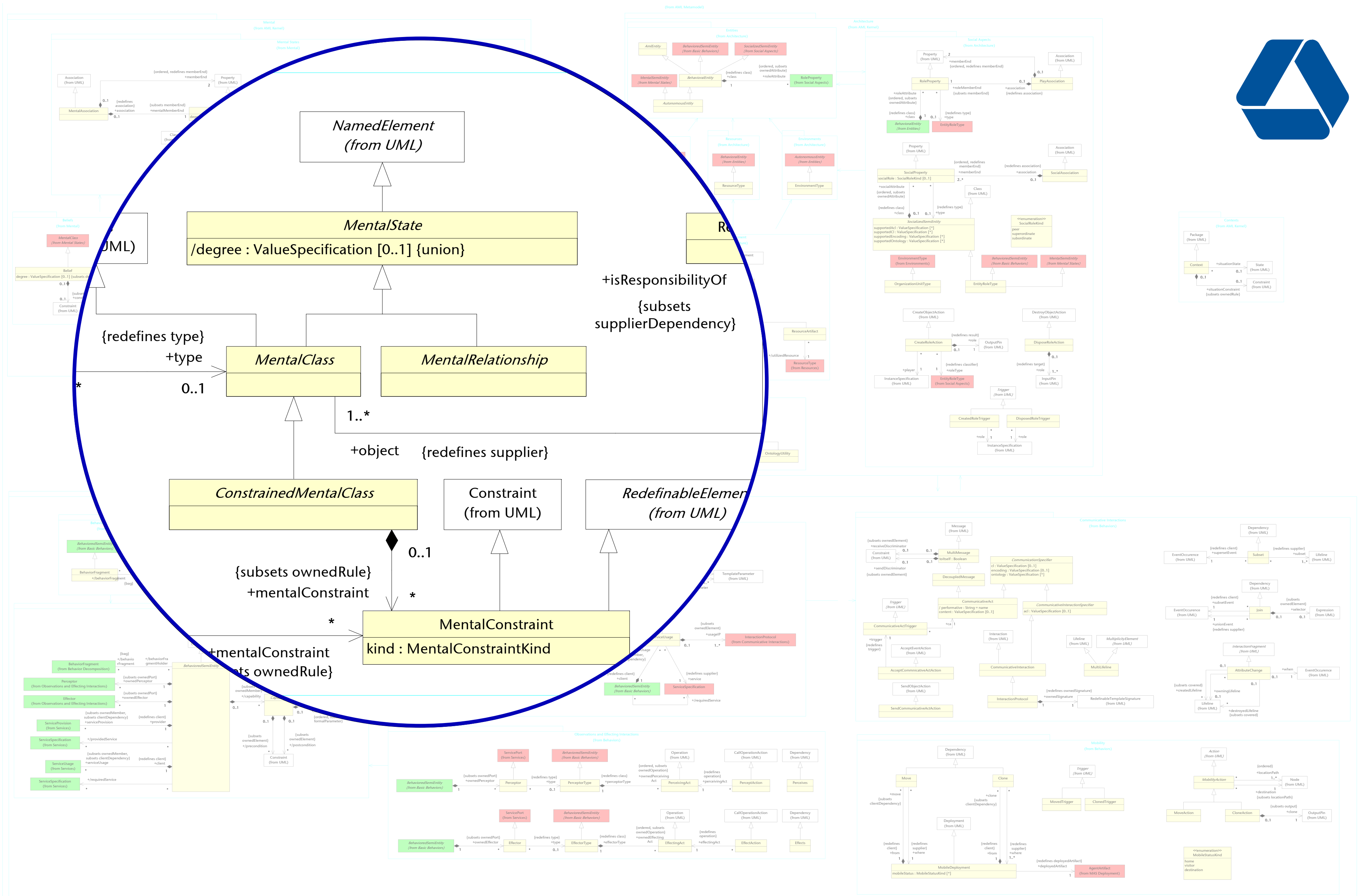
Conclusions and Further Work

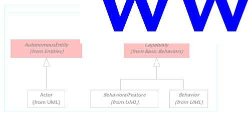
Current Status

- Ready for use.
- Supported by CASE tools - Enterprise Architect and IBM Rational Rose.
- Used in commertial 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.





www.whitestein.com