Notes from paper
Bernhard Bauer, James Odell
UML 2.0 and Agents: How to build agent-based systems with the new UML standard.
Journal of Engineering Applications of Artificial Intelligence Volume 18, Issue 2 , March 2005, Pages 141-157 (http://www.jamesodell.com)

UML 2.0 AND MDA: THEIR USAGE FOR AGENT-BASED SYSTEMS

UML - STRUCTURAL DIAGRAMS - STATIC ASPECTS

CLASS DIAGRAMS Are good for:

- ORGANIZATIONAL MODELS (static aspects of; its associations and part-of relationships)
- ONTOLOGIES (organizational model knowledge)
- DEFINITION OF SUB-TASKS and sub-goal hierarchies (using generalizations)
- DEFINE STRUCTURAL ASPECTS OF TASKS using aggregation and composition, constraints lik goals, control features, services (functions as interfaces) can be added via attributes, functions and associations.
-

(APPLIE TO AGENT MODELLING)

- AGENT MODEL can be defined using class names, inheritance (generalization) of classes and adding name, type, position/role, capabilities and constraints, either directly or via associations. Role Hierarchy can be defined using generalization. However roles cannot be modeled in the necessary detail with any UML 2.0 diagrams.
- SERVICE MODELS defining services with input/output parameters and pre-/post-conditions a classes with attributes and functions (the service interfaces)

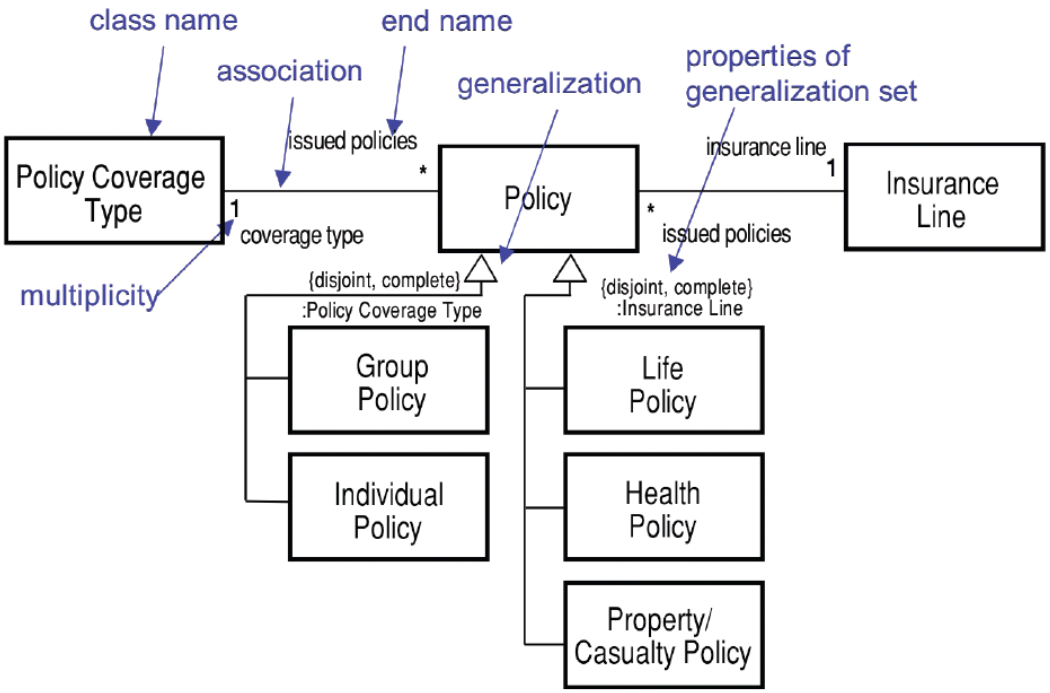


Figure 1 Class diagram

COMPOSITE STRUCTURE DIAGRAMS are good for:

- DESCRIBES THE INTERNAL STRUCTURE OF A CLASSIFIER, including the interaction points of

the classifier to other parts of the system.

- APPLIED DURING TOP-DOWN MODELING OF THE SYSTEM, to model the relationship between parts of the system through specific interfaces (ports) in a precise manner.
- DESCRIBE THE ARCHITECTURE OF THE SYSTEM (ARCHITECTURE DIAGRAM) and for specification and application of patterns.

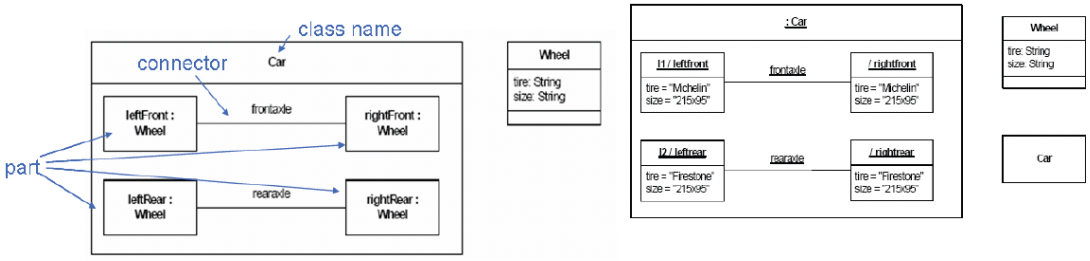


Figure 3 Composite Structure Diagram and its Instantiation

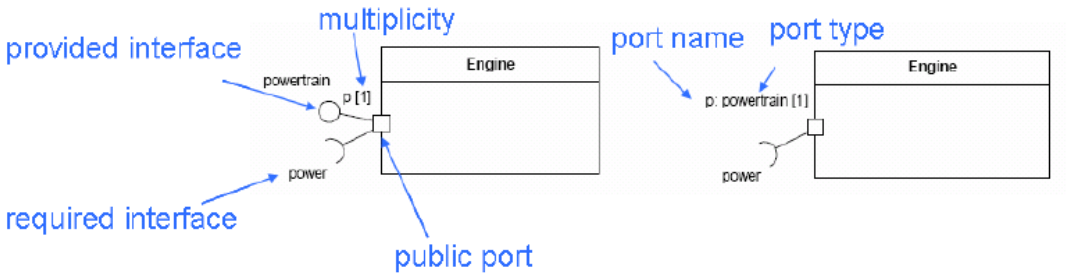
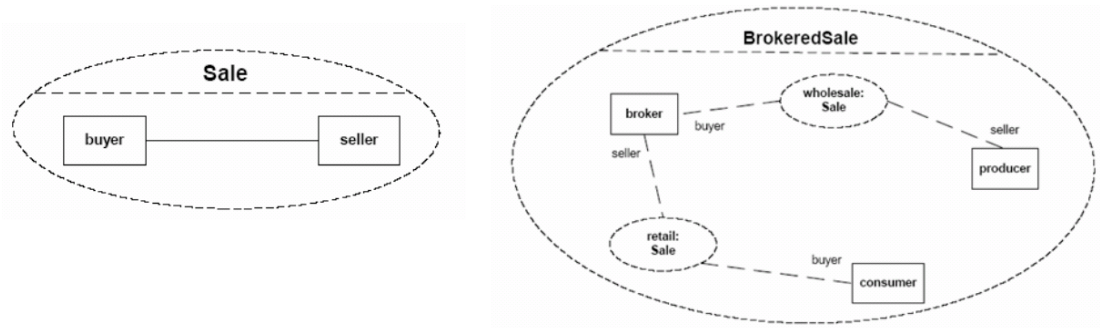


Figure 4 Composite Structure Diagram – Ports

(APPLIED TO AGENT MODELING)

- MODELLING AN ORGANIZATION, its dependencies and workflows between agents.
- REPRESENT ORGANIZATION'S EXTERNAL INTERFACES as well as the internal behavior and interfaces of an agent. The interfaces defines the speech acts understood by the agent as well as the actions performed by an agent.
- EXPRESS COLLABORATIONS the collaboration defines an interaction among roles, as iilustrated in Sale and BrokeredSale collaborations in Fig. 5.



- **Figure 5 Collaborations**

- FOR DEFINING THOSE AGENTS PATTERNS that cab be instantiated in different contexts, suc as typical agent broker architecture or negotiation pattern.
- DEFINE THE ARCHITECTURE OF AN AGENT-BASED SYSTEM and how a given architecture can be instantiated in different contexts.
- REPRESENT SOCIAL STRUCTURES such as groups and roles. A GROUP is a set of agents that are related via their roles, where these links must form a connected graph within the group. Another way to look at this is that a GROUP is a composite structure consisting of interrelated roles where each of the group's roles has any number of agent instances.
- AGENTIFIED GROUPS possesses all the features that any agent might possess. For example, it can send and receive messages directly and take on roles. Such a group is an agent in its own right, and therefore is a subclass not only of Group but also of Agent (such groups can also be referred to *as organizations*). NON-AGENTIFIED GROUPS are still first-class entities; however these entities do not possess agent properties. Thus, they are as objects rather than agents.

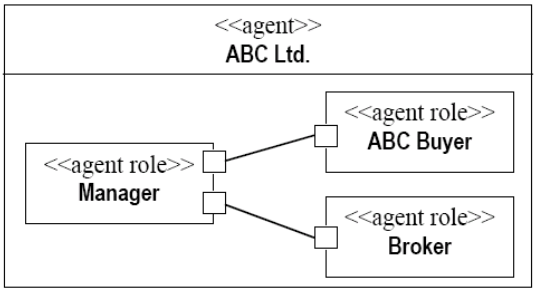


Figure 6 Example of the ABC Ltd. Agentified Group and its associated Roles.

- FIG 6 represents the GROUP "ABC Ltd" as a composite structure with three associated roles, "Manager", "Broker" and "ABC Buyer". The "Manager" interacts directly with the "ABC Buyer" and the "Broker". In this situation, it is possible to interact with the agent "ABC Ltd" without knowing directly about any specific "Manager", "Broker" or "ABC Buyer" within the department; thus, this group is AGENTIFIED. The stereotype "<<agent>>" indicates that the group is Agentified. Groups can also be formed simply to establish a set of agents for purposes such as intra-group synergies or conceptual organization.
- A NON-AGENTIFIED GROUP is a Group that is not a subclass of Agent. Fig. 7 shows a Non-Agentified version of "ABC Customer Sales Dept". It has the same associated Roles; howeve it does not have the "<<agent>>" stereotype. In order to interact with this Department, yo must interact directly with one of its members: a "Manager", an "ABC Buyer" or a "Broker".

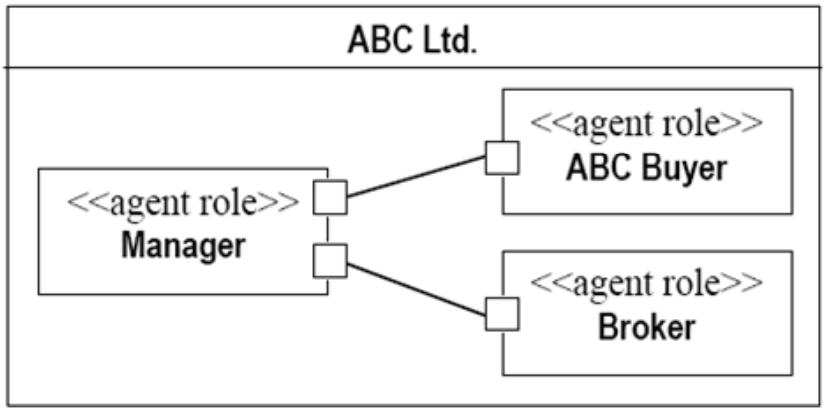


Figure 7 Example of the Non-Agentified ABC Ltd.

COMPONENT DIAGRAMS are good for:

- DESCRIBES THE ORGANIZATIONS AND DEPENDENCIES AMONG COMPONENTS. A componen a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. (...) It describes components, interfaces, ports as well as the realization, implementation and usage relationships with its classes and artifacts.

(APPLIED TO AGENT MODELING)

- DEFINE THE INPUT/OUTPUT BEHAVIOR OS TAKS and for the decomposition of the system architecture.
- THE BLACK-AND-WHITE-BOX NOTATION allows defining private and public interfaces of agents.
- THE MANIFEST-STEREOTYPE can be used to show how an agent component is deployed in distinguish systems.

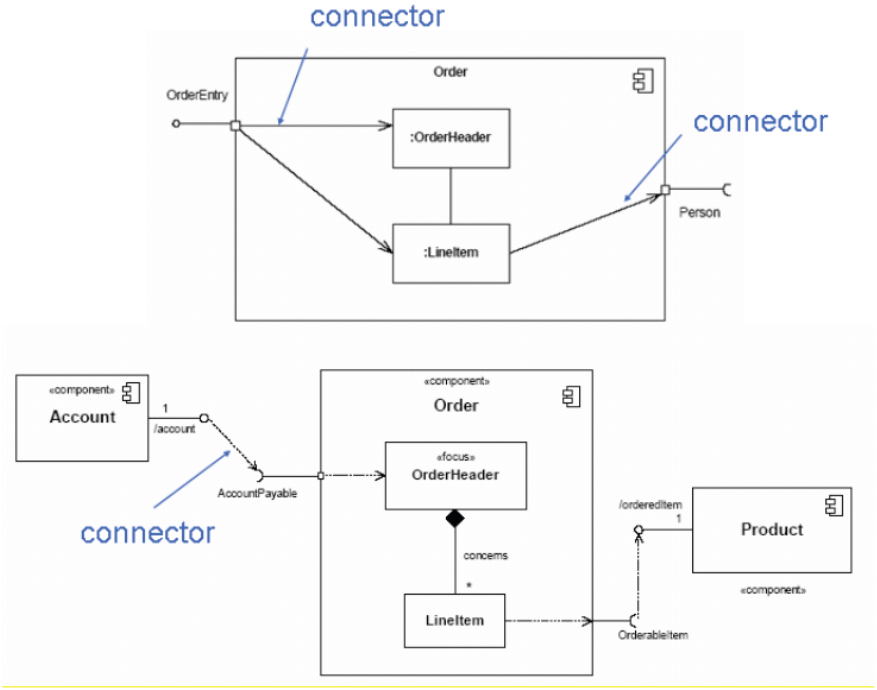


Figure 8 Component Diagram

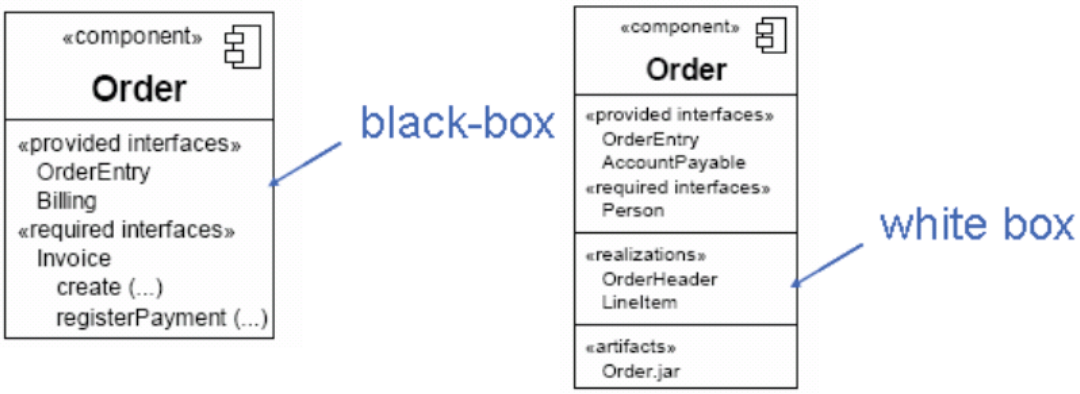


Figure 9 Component Diagram – Black and White Box

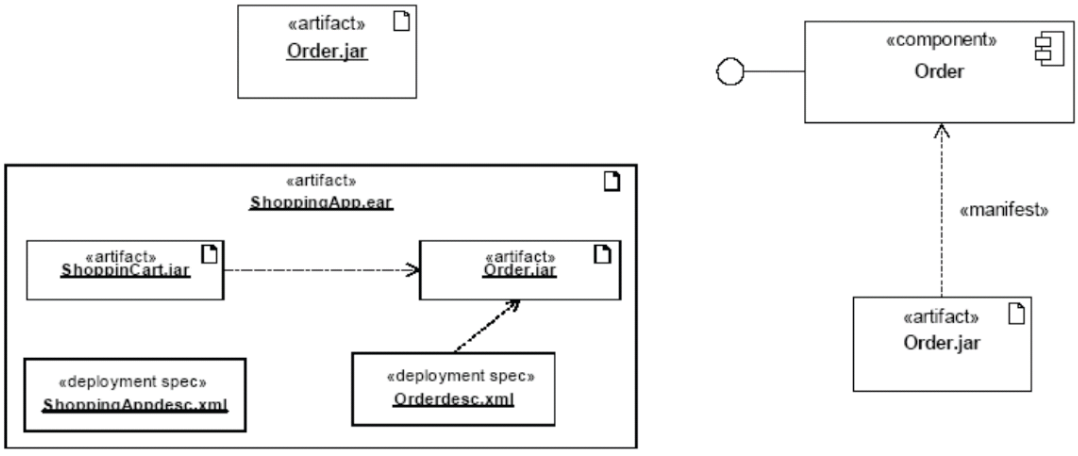


Figure 10 Component Diagram examples.

DEPLOYMENT DIAGRAMS are good for:

- DESCRIBES THE EXECUTION ARCHITECTURE OF SYSTEMS and the system architecture. System artifacts are represented as nodes, which are connected through communication paths to create network systems of arbitrary complexity. Nodes represent run-time computational resources, with memory and, sometimes processing capability. Run-time objects and components may reside on nodes. Nodes (...) and represent either hardware devices or software execution environments.

devices or software execution environments.
- APPLIED TO SHOW THE RUN-TIME ENVIRONMENT OF A SYSTEM and to represent "software server" as well as to describe the distribution of components.
- ARTIFACTS are implementations of any packageableElement (Fig 11).
- ARTIFACTS THAT REIFIES OR IMPLEMENTS deployment specification properties is a deployment descriptor (Fig. 12)

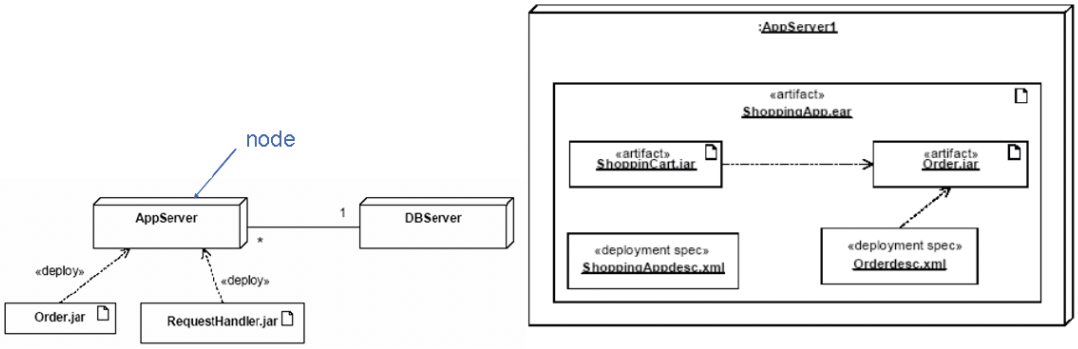


Figure 11 Deployment diagram and complex node

Figure 12 Composite Structure Diagram – Ports

(APPLIED TO AGENT MODELING)
- DESCRIBE THE PHYSICAL DISTRIBUTION OF AGENT INSTANCES, in particular they can be applied for defining the migration of agents.
- THIS SPECIFICATION DEFINES THE PLATFORM DESIGN of an agent-based system. Having e.g. Generic agents, the deployment specification can be applied to define the customization of agents in a specific context.

UML - BEHAVIORAL DIAGRAMS - DYNAMIC ASPECTS

USE CASE DIAGRAMS are good for:
- SPECIFY REQUIRED USAGES OF A SYSTEM. They are used to capture the requirements of a system, that is, what a system is supposed to do.
- DEFINE THE EXTERNAL VIEWPOINT ON THE SYSTEM and to support encapsulation. They define the "WHAT" instead of "HOW" a system is realized from the perspective of an external communication partner.
- DESCRIBES THE SYSTEM, THE USE CASES OF A SYSTEM, EXTERNAL ACTORS and their relationships between actors and use-cases, between actors and between use-cases.

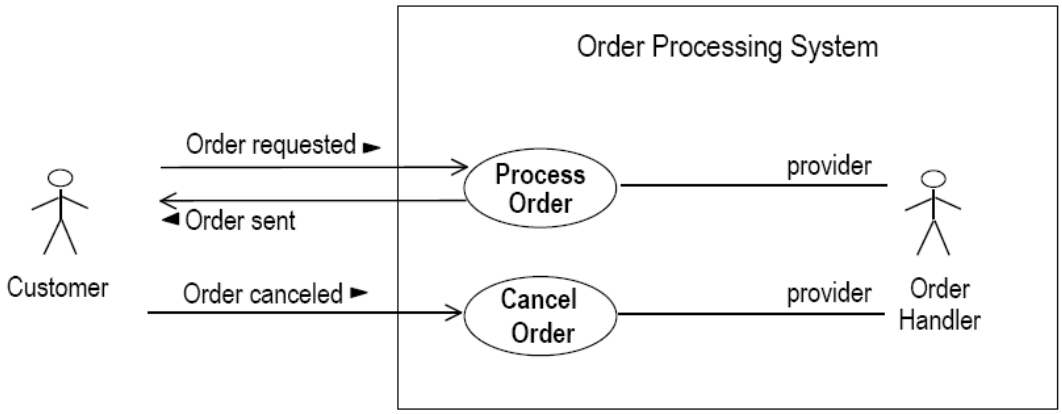


Figure 13 A Use Case Diagram for an Order Processing application.

(APPLIED TO AGENT MODELING)

- CAN ALSO BE USED THESE SAME NOTIONS OF ACTOR, use case, and subject, as illustrated in Fig. 13.
- SOME CHANGES HAVE BEEN DONE:
  1) Includes the events from the requesting actor to which the subject must respond (sometimes referred to as PERCEPTS) and the events that affect an actor in some way (sometimes referred to as ACTIONS). These are indicated as names on the associations between actors and use cases. The associations also indicate directionality for these events.
  2) Indicates the providing actor for the service defined by the use case. Both the requesting actor and providing actor are vital to the service-oriented approach. The metamodel for the W3C's Web Services Architecture (W3C, 2004) defines both of these notions in terms of agents. In Fig. 13 the Customer Actor is the REQUESTING AGENT for the Process Order Service and the Order Handler Actor is the PROVIDING AGENT. Fig. 14 illustrates some of these same ideas for Use Case Diagrams that represent direct relationships.

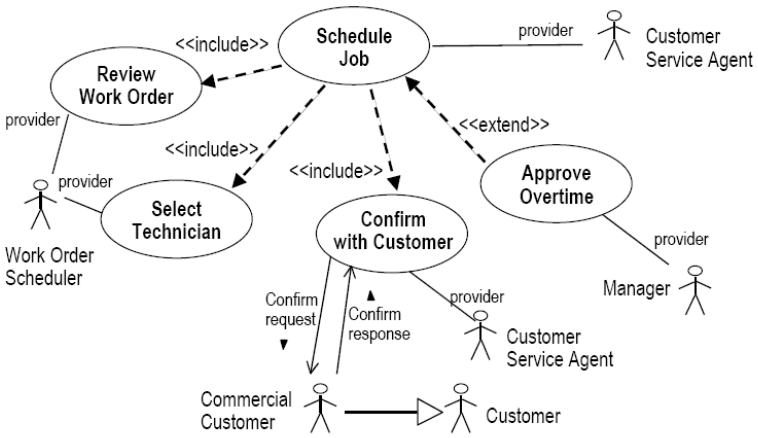


Figure 14 A job scheduling Use Case Diagram with include and extend relationships.

Such association, therefore, may include multiplicities, end names, association name, and so on.

- Fig. 15 depicts two internal actors for a Bus Transportation System: Bus Driver and Bus Payment Machine. The Bus Driver is the actor that provides the general Bus Service. However, an Obtain Payment use cases is included in the Bus Service which is provided by a different actor.

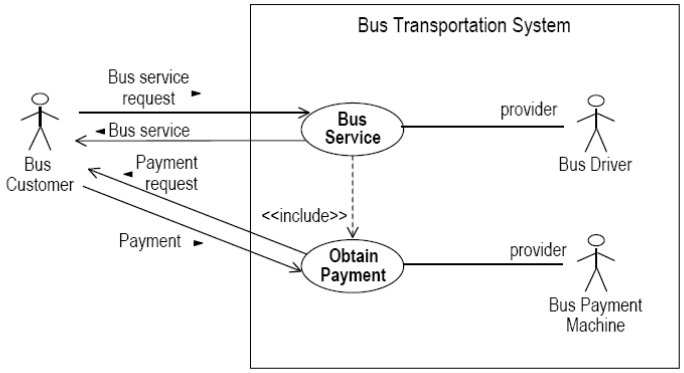


Figure 15 An example when internal actors can be providing and/or requesting actors.

- In UML 2.0, an actor "specifies a role played by a user or any other system that interacts with the subject." An actor, then, can represent a single role, such as the Customer or Manager actors in the Fig. 15.
- Internal actors might also become requestors for services from external actors. In Fig. 16, the Shipment Inquiry Interface actor requests a Shipment Search service from the Tracking Inquiry agent.

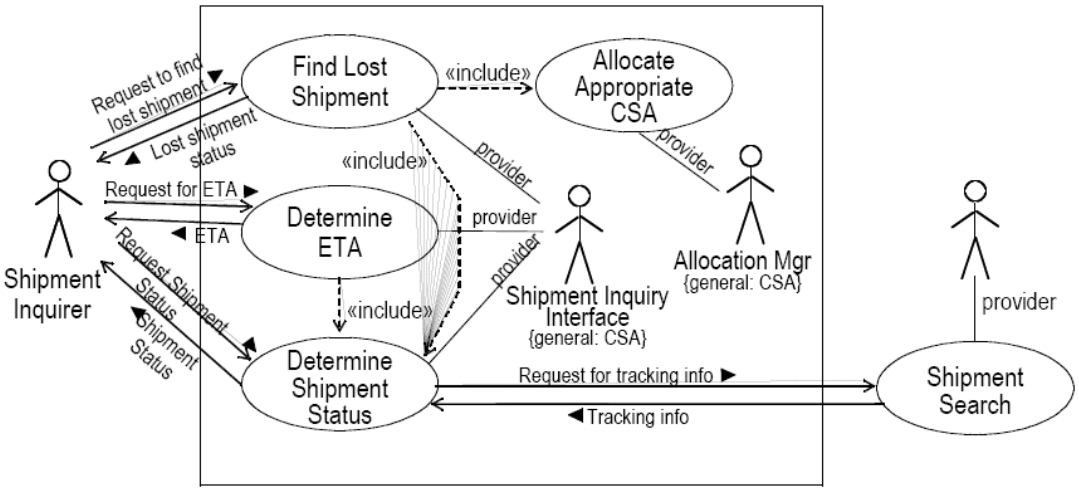


Figure 16 Indicating Actor roles and resources.

ACTIVITY DIAGRAMS are good for:

- ACTIVITY MODELING EMPHASIZES THE SEQUENCE AND CONDITIONS for coordinating lower-level behaviors. These are commonly called control flow and object flow models. The actions coordinated by activity models can be initiated because other actions finish executing, because objects and data become available, or because events occur external to the flow.

(APPLIED TO AGENT MODELING)

- FOR AGENTS all conditions of the previous item are useful. However, from the agent standpoint, additional UML 2.0 are also practical. For example, in Fig. 17, the Activity Diagram represents a business process that an agent system might support. By its title, it suggests that this is a plan (a sequence of actions/steps towards a objective) for the Process order Service. Each plan can be expressed as an Activity Diagram. However, UML 2.0, needs to be extended to define plan rule conditions. *The plan rule* specifies those conditions under which the associated activity may be invoked.

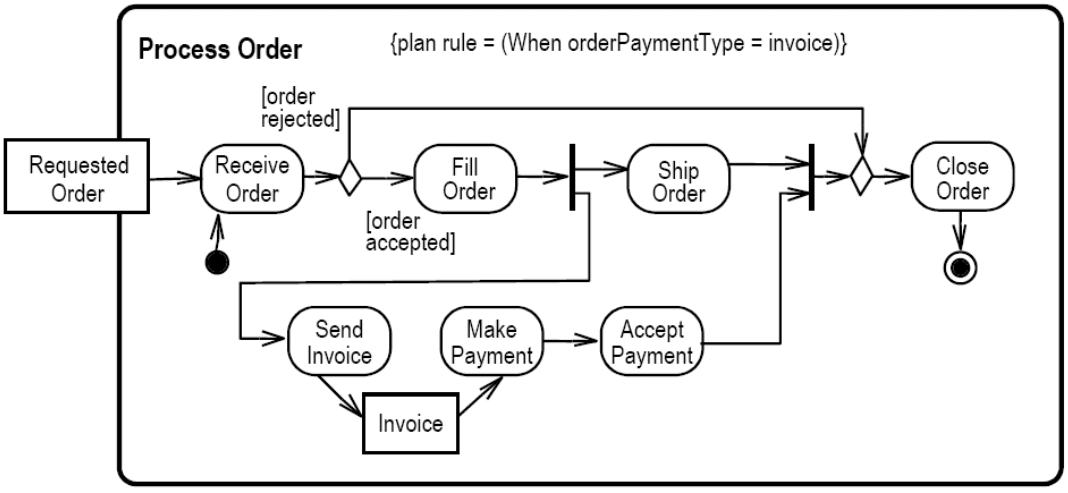


Figure 17 An Activity Diagram for a Process Order service.

- In the example above (Fig. 17), *the plan rule* condition indicates that when Process Order Service is requested for an order is to be invoiced, this particular Activity Diagram plan is executed. Different Activity Diagrams may specify alternate plans for Process Order based on, say, credit card or cash payment instead. Note it is necessary an extension to UML 2.0 to support the ability of a process (i.e., service or goal) to choose from multiple plans. Currently UML 2.0 can only invoke a single Activity Diagram for a given process.
- INDICATING THE ROLE FOR THE PROCESS WITHIN AN ACTIVITY DIAGRAM is also useful. Activity Diagrams can represent this in two ways: PARTITIONS and ANNOTATED PROCESSES In Fig. 18 PARTITIONS for Order Handler and Invoice Handler roles are represented as swimlanes.

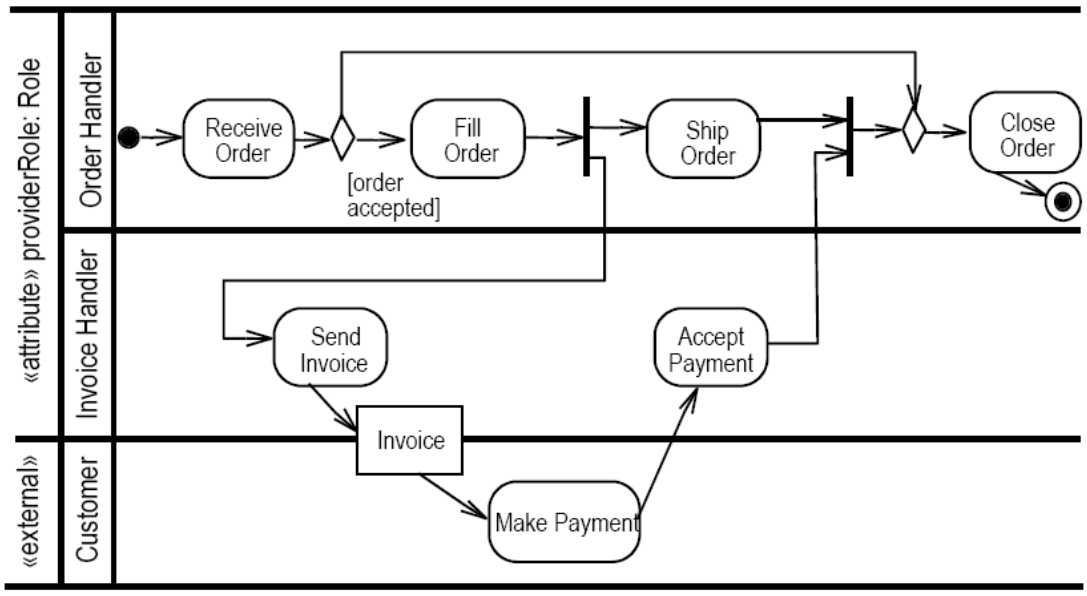


Figure 18 An Activity Diagram with role-based swim lanes.

- Graphical swimlanes are not always that clearest express partitioning. In UML 2.0, each process on an Activity Diagram can be notated individually with the appropriate designation. In Fig. 19, the Order Handler and Invoice Handler roles are placed within each processes' round-cornered rectangle.

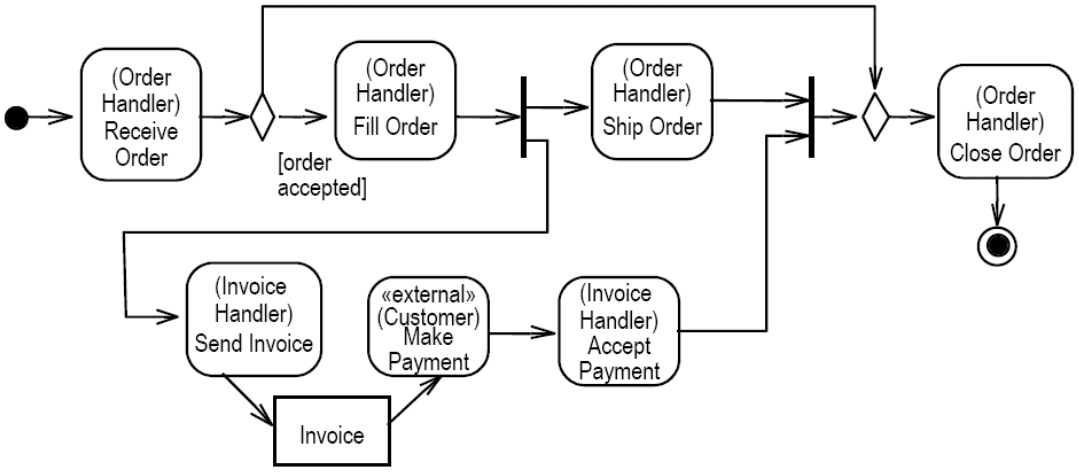


Figure 19 An Activity Diagram with role-based annotations.

- IN UML 2.0 NO NOTION OF GOAL, per se, EXISTS. However, two ways of think about goals for activity diagrams are supported.
  1) *The Activity Final Node (the bulls eye)* is considered the goal for the activity, because it is the end point for the process.
  2) At a more macro level, *The* Service can be thought of as supporting a goal.
  The only other issue that has been identified involves the control nodes in the flow (decision/merge nodes and fork/join nodes). While the semantics of the control node is understood, the responsibility for its underlying processing is not. If the Activity Diagram has a "control" agent that coordinates the process flow, each Activity Diagram can be associated with a providing role. Another option is that each of the control nodes could be associated with a role that provides the control node functionality with in the activity context. Both options can also be chosen, where a control agent is responsible for coordinating the overall process flow be delegating to more specialized roles to handle each control node - just as it could for the individual processes in the diagram.

STATE MACHINE DIAGRAMS are good for:

- DESCRIBES THE DISCRETE BEHAVIOR MODELED THROUGH FINITE STATE-TRANSITION SYSTEMS. The sequences of states that an object or an interaction goes through during its life in response to events, together with its responses and actions can be modeled.
- ARE APPLIED FOR THE STATE DESCRIPTION OF e.g., classifiers, for detailing use cases, for

behavior description of interfaces and ports, for detailed descriptions of event and signal handling.

- *BEHAVIOURAL STATE MACHINES* are state machines that can be used to specify behavior of various model elements. For example, they can be used to model the behavior of individual entities (e.g., class instances).
- *PROTOCOL STATE MACHINES* are used to express usage protocols. Theu express the legal transitions that a classifier can trigger. The state machine notation is a convenient way to define a lifecycle for objects, or an order of the invocation of its operation. Because protocol state machines do not preclude any specific behavioral implementation, and inforces legal usage scenarios of classifiers, interfaces and ports can be associated to this king of state machines.

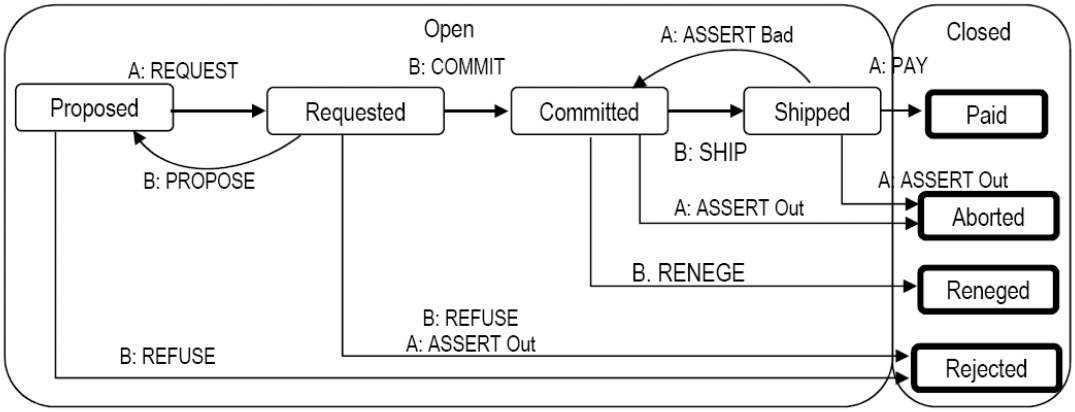


Figure 20 A State Machine Diagram for a Process Order service.

(APPLIED TO AGENT MODELING)

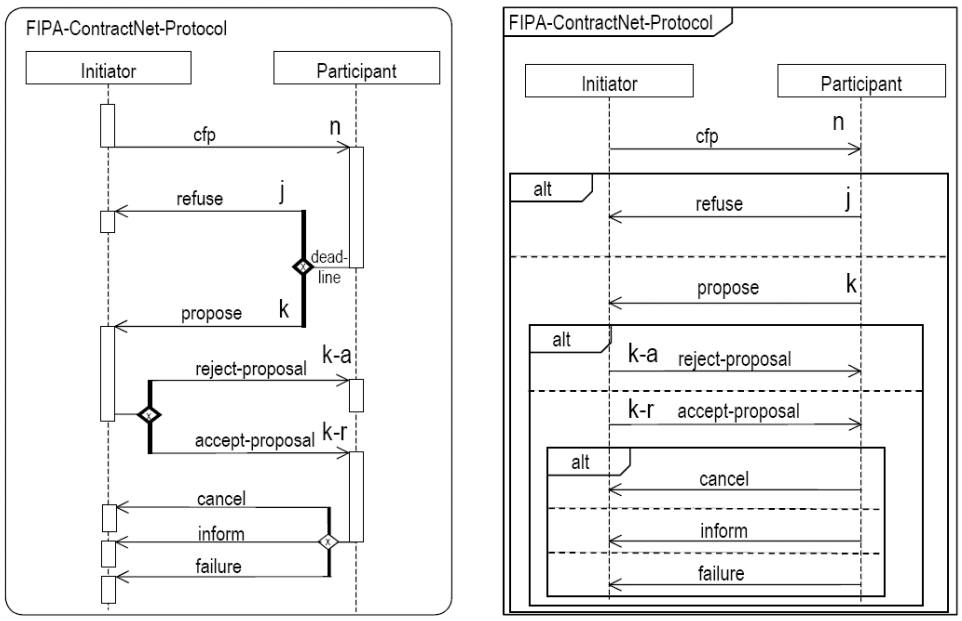
- STATE MACHINE CAN BE APPLIED FOR MODELING INTERACTION PROTOCOLS, similar to sequence diagrams and to model plans.

SEQUENCE DIAGRAMS are good for:

- ARE APPLIED TO MODEL INTERACTIONS AND IN VARIOUS PHASES OF THE SOFTWARE DEVELOPMENT PROCESS (e.g., use case refinement, modeling of test scenarios, communication model, detailed modeling of messages exchanges or specification of interfaces). Focuses on message interchange between a number of lifelines. In particular, describes an interaction by focusing on the sequence of messages that are exchanged, along with their corresponding event occurrences on the lifelines.

(APPLIED TO AGENT MODELING)

- Prior to UML 2.0, FIPA defined an agent-based that extended UML 1.x to include roles, decision points, concurrency, modularity, and multicasting (Fig. 21a).



(a) UML 1.x with extensions (b) UML 2.0

Figure 21 UML 1.x agent extensions and UML 2.0 Sequence Diagrams

- UML 2.0 includes representation of all these notions except role and multicasting support. UML 2.0 adds, e.g., loops, alternatives, parallelism, sequences and critical fragments.
- ROLE NOTATION CAN INITIALLY INCLUDE MAKING EACH LIFELINE A ROLE. The current UML 2.0 metamodel is not far from this general concept, but the agent-based notion of role is not defined by UML 2.0.
- TO SUPPORT THE NOTATION OF MULTICAST AND MULTIRESPONSE, a cardinality-based notation was added to the message lines (Fig. 21b). For example, *the cfp* message is annotated to indicate a message that would be multicast from an INITIATOR *to n* PARTICIPANTS. The response then involves a refusal *from j* PARTICIPANTS and proposal *for* PARTICIPANTS; and so on.

COMMUNICATION DIAGRAMS are good for:

- (Formerly known as Collaboration Diagrams in UML 1.x) DESCRIBES THE INTERACTION BETWEEN LIFELINES WHERE THE ARCHITECTURE OF THE INTERNAL STRUCTURE AND HOW THIS CORRESPONDES WITH THE MESSAGE PASSING IS CENTRAL. They correspond to simple Sequence Diagrams that use none of the structuring mechanisms such as interaction occurrences and combined fragments. It also assumes a strict ordering of messages.
- DUE TO THEIR LIMITED EXPRESSIVENESS, this diagrams can only be used to represent simp and straight-forward interactions.

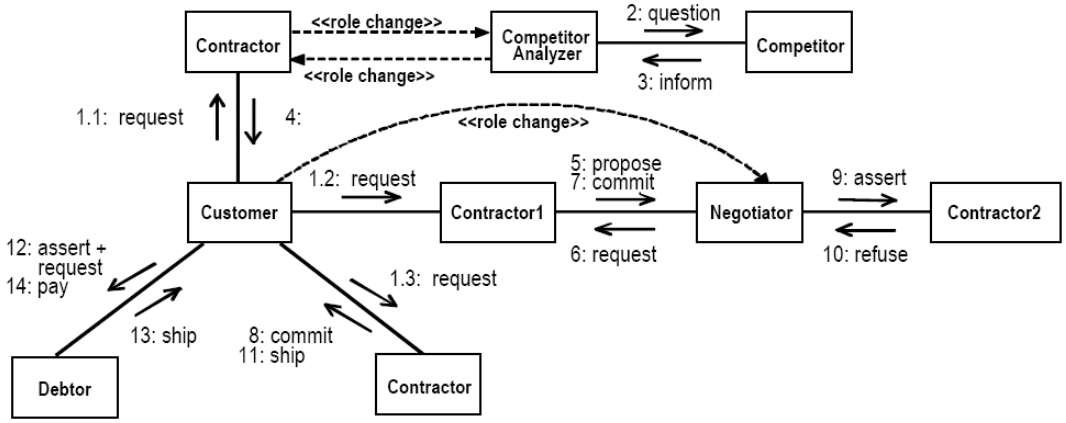
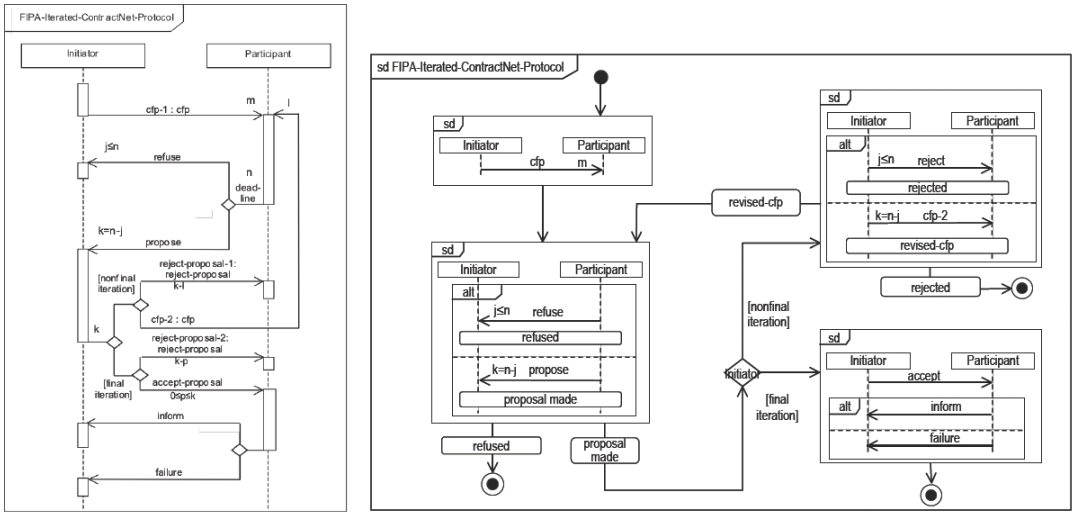


Figure 22 A Communication Diagram with role-based annotations.

INTERACTION OVERVIEW DIAGRAMS are good for:

- DEFINE INTERACTIONS THROUGH A VARIANT OF ACTIVITY DIAGRAMS in a way that promot overview of the control flow. Focus on the overview of the flow of control where the nodes are interactions. The lifelines and the messages do not appear at this overview level. For example, the UML 1.x diagram in Fig. 23a would be graphically cumbersome to express as a sequence diagram in UML 2.0, because of the resulting plethora of boxes within boxes. By

using the Interaction Overview Diagram, the flow can be more clearly delineated as depicted in Fig. 23b.



(a) UML 1.x agent extensions with looping (b) A UML 2.0 Interaction Overview Diagram representation for Figure 24.

Figure 23 UML 1.x agent extensions with looping.

- The Interaction Overview Diagram (IOD) contains sequence diagrams.

(APPLIED TO AGENT MODELING)

- IF THE INTERACTION OVERVIEW HAS A "control" agent that coordinates the process flow, each IOD can be associated with a providing role.
- ANOTHER OPTION is that each of the control nodes could be associated with a role that provides the control not functionality with in the IOD context.
- BOTH OPTIONS may also be chose, where a control agent is responsible for coordinating the overall process flow by delegating to more specialized roles to handle each control node.

MDA and CURRENT CHARACTERISTICS

- The artifacts in MDA are formal models, i.e., models that can be understood by computers and finally be transformed into a representation that lends itself to execution. It is composed of three core models: Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM).
- CIM - COMPUTATION INDEPENDENT MODEL: this is the most abstract model within MDA. Is independent of computational technology. It describes the BUSINESS LOGIC and therefore defines BUSINESS PROCESSES and WORKFLOWS in detail.
- PIM - PLATFORM INDEPENDENT MODEL: this model is defined at a high level of abstraction; is independent of any implementation technology. It describes a SOFTWARE SYSTEM that SUPPORTS SOME BUSINESS. Whether a system will be implemented on a mainframe with a relational database, on an EJB application server or on an agent-platform is irrelevant at the PIM level.
  PSM - PLATFORM SPECIFIC MODEL: in the next step, the PIM is transformed into one or mor PSMs. It is tailored to specify a system in terms of the implementation constructs available i one specific implementation technology. A PIM is transformed into one or more PSMs. For each specific technology platform a separate PSM is generated. Most systems today span several technologies; therefore it is common to have many PSMs with one PIM. The final ste in the development is the transformation of each PSM to code. Because a PSM fits its technology rather closely, this transformation is relatively straightforward.

MDA and AGENT-BASED SYSTEMS

Summarizing the different approaches of existing agent methodologies (see section 2 of paper) and the usage of UML 2.0 diagrams from section 3.1, the authors distill the following necessary aspects to be covered by a MDA covering major areas of agent-based systems (only focusing on CIM and PIM):

CIM - Computational Independent Model, has to deal with the following aspects:

- USE CASES: Taken from OO Software Development, use case scenarios are a suitable method to derive the functional requirements of a systems needed to be derived. UML 2.0 use cases diagrams are applied.

use cases diagrams are applied.

- ENVIRONMENT MODEL (see paper Modeling Agents and their Environment, by James Odell): Considering several aspects of environment modeling ranging from physical environments to agent communication and to how their considerations could be embedded into FIPA architecture.
- DOMAIN/ONTOLOGY MODEL: This model defines the ontologies of the domain and relates them to other existing ontologies, e.g., UML class diagrams and Semantic Web representatic languages.
- ROLE MODEL: This model describes the roles in a domain, on the one hand in the traditional object-oriented sense (actor-role relationship), but also defining roles characterizing social relationships within an agent-based system.
- GOAL/TASKS MODEL: This model defines the objectives of an agent in terms of soft and har goals, and should also support means-end analysis (as in Tropos). Moreover, the notion of tasks and plans should be provided to support the description of agent behavior at a high level of abstraction.
- INTERACTION MODEL: This model defines the regime of interaction and collaboration among entities and groups of entities, at a level which abstracts from specific interaction protocols.
- ORGANIZATION/SOCIETY MODEL: This model defines to a reasonable extent the real-world society and organization and hence the social context within agents in an agent-based system acts and interacts.
- BUSINESS PROCESS MODELS: The notion of business process is key for corporate business applications. Business processes describe the means and the ends of business interactions. For agents to support corporate applications, it is important to be able to access executable definitions of business processes, to reason about the semantics of goal-directed business processes (see http://www.agentissoftware.com), and to relate business process to the organizational model, the interaction model, and the task model.

PIM - Platform Independent Model, has to deal with the following aspects:

- INTERACTION PROTOCOL MODEL: This model defines the interaction between different agent class, agent instances and roles at the level of interaction protocols, such as the Contract Net.
- INTERNAL AGENT MODEL: This model deals in particular with goals, beliefs and plans of age classes, how they are defined and which underlying architecture is used. Defines the interna behavior of an agent(s).
- AGENT MODEL: This model describes the behavior of agents and agent groups, i.e., how different agent are collaborating together independent of their implementation. The interaction model defines the concrete interaction of the agents, whereas the internal agent model defines the internal behavior of an agent, e.g., in terms of BDI, and the agent model defines the behavior of an agent seen by other agents.
- SERVICE/CAPABILITY MODEL: Defines the services and capabilities of agents, mostly using service description languages and mechanisms such as UDDI or DAML-S.
- ACQUAINTANCE MODEL: This model provides agents with models of other agents' beliefs, capabilities and intentions. It can be used to determine suitable partners for collaboration or to predict others' behavior, e.g., in a coordination task.
- DEPLOYMENT/AGENT INSTANCE MODEL: This model describes which agent instances exist, the migration is considered as well as the dynamic creation of agents.