

PHILADELPHIA UNIVERSITY



AN AGENT MODELING FORMALISM
(ENHANCING AUML CLASS DIAGRAM)

By

Mahmoud Adnan Ibrahim Sawalha

Supervisor

Dr. Said Ghoul

This Thesis was Submitted in Partial Fulfillment of the Requirements for the
Master's Degree in Computer Science

Deanship of Academic Research and Graduate Studies

Philadelphia University

February, 2008

AN AGENT MODELING FORMALISM
(ENHANCING AUML CLASS DIAGRAM)

By

Mahmoud Adnan Ibrahim Sawalha

Supervisor

Dr. Said Ghoul

This Thesis was Submitted in Partial Fulfillment of the Requirements for the
Master's Degree in Computer Science

Deanship of Academic Research and Graduate Studies

Philadelphia University

February, 2008

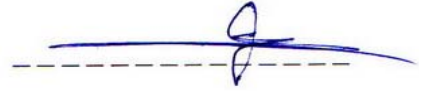
Successfully defended and approved on February, 2008

Examination Committee

Signature

Dr, Saad Ghoul _____, Chairman.

Academic Rank: Associate Prof



Dr, Saad Al-Mahdawy _____, Member.

Academic Rank: Associate Prof.



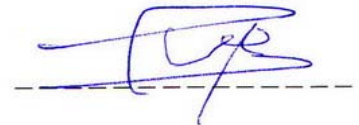
Dr, Ayman Issa _____, Member.

Academic Rank: Assistant Professor



Dr, Amjad Hudaib _____, External Member.

Academic Rank: Assistant Prof.



Dedication

This thesis is dedicated to my family. So many thanks to my parents, my brothers, and sisters for their unconditional support, I would never have reached this point without their support.

Also, I want to dedicate this thesis to my friends.

Acknowledgment

This research project would not have been possible without the support of many people. I would like to express my deepest sense of gratitude to my supervisor Dr. Said Ghoul for his patient guidance, encouragement and excellent advice throughout this study, his constant encouragement, support, and invaluable suggestions made this work successful, he has been everything that one could want in an advisor.

I am deeply indebted to my committee members for their time and effort in reviewing this work.

Finally, I wish to express my gratitude to all those who gave me the possibility to complete this thesis.

List of contents

Subject	Page
Title	ii
Committee Decision	iii
Dedication	iv
Acknowledgment	v
List of contents	vi
List of figures	viii
List of abbreviations	ix
Abstract	x
Chapter1: Introduction	1
1.1 Agent	2
1.2 AUML Class Diagram Problem	2
Planning	2
Roles	2
Knowledge Base	2
Configurations	3
Unpredictable Agent behavior	3
1.3. Importance of the problem	3
1.4. Insufficiencies of actual approaches dealing with the problem	3
1.5. The Contribution	3
Chapter 2: Cass study	4
2.1 RobocupRescue simulation	5
Overview of RobocupRescue simulation	5
Fire Brigade Agents	6
Police Force Agents	7
Ambulance Teams	8
2.2 Case study analysis	8
Fire Brigade Agents	8
Police Force Agents	10
Ambulance Teams	11
Chapter 3: Class Diagram in actual works	13
3.1. Class Diagram in Unified Modeling Language	14
Classes	14
Interfaces	15
Relationships	15
3.2. AUML Class Diagram	16
Agent Class	16
Agent Communication Language	19
Agent Service	19
Agent Class relationships	20

Applying the case study in AUML	20
3.3. Toward Agent-Oriented Conceptualization and Implementation	25
Applying the case study in Toward Agent-Oriented Conceptualization and Implementation	26
3.4. A Methodology for Ontology Based Multi-Agent Systems Development (MOBMAS)	28
Applying the case study in MOBMAS	29
3.5. Jadex	31
Applying the case study in Jadex	32
3.6. Developing Role-Based Open Multi-Agent Software Systems	35
Agent Class	35
Role Class	37
Applying the case study in Developing Role-Based Open Multi- Agent Software Systems	39
3.7. Conclusion	43
Chapter 4: An Agent Class Diagram Enhancement	44
4.1. Agent Structural Requirements	45
Autonomy	45
Communication	46
History	46
Social ability	47
Rationality	47
Unpredictable behavior	47
Learning ability	47
Mobility	47
Reasoning	48
Multi-agent planning	48
4.2. An AUML Class Diagram Enhancement	48
Agent Class	48
Role Class	55
Agent Communication Language	56
Agent Service	56
Agent Class Diagram Relationships	57
4.3. Applying the case study in AUML Class Diagram Enhancement	59
Chapter 5: The Evaluation	68
5.1. Introduction	69
5.2. Comparison with similar works	70
Multiagent System Structural Requirements	70
Model Coherence	75
Chapter 6: Conclusions and Future Work	76
6.1 Conclusions	77
6.2 Future Work	77
References	78

List of Figures

Figure Number	Figure Title	Page
Figure 2.1	A proposed workflow for a fire brigade agent	7
Figure 4.1	Automata-Reasoning Mechanism	54
Figure 4.2	Agent Class	55
Figure 4.3	Role Class	56
Figure 4.4	Agent Service	57
Figure 4.5	Inheritance relationship	57
Figure 4.6	Play relationship	57
Figure 4.7	Control relationship	57
Figure 4.8	Dependency relationship	58
Figure 4.9	Aggregation relationship	58
Figure 4.10	Inheritance relationship	58
Figure 4.11	Leading relationship	58

List of Abbreviations

ACL	Agent Communication Language
AIP	Agent Interaction Protocol
AOSE	Agent Oriented Software Engineering
AUML	Agent Unified Modeling Language
FIPA	Foundation for Intelligent Physical Agents
FIPA SL	FIPA Semantic Language
KQML	Knowledge Query and Manipulation Language
MAS	Multi-Agent Systems
OCL	Object Constraint Language
OMG	Object Management Group
OOSE	Object Oriented Software Engineering
UML	Unified Modeling Language

ABSTRACT

Multi-Agent Systems (MAS) has been used successfully for years with different purposes. It is used in systems that using some kind of intelligence and automation. Nowadays, there are a lot of modeling languages used to model MAS. One of the well-known MAS modeling languages is Agent Unified Modeling Language (AUML). AUML is an agent modeling language based on Unified Modeling Language (UML 2.0), it enhances some of UML diagrams and it doesn't use or enhance the remain of UML diagrams. Even, AUML is the closest agent modeling language to UML; it still has some serious weaknesses that have not been solved yet while dealing with agents.

This study enhanced the agent class diagram in the agent modeling language AUML and presents a new agent class diagram that solves some of the weaknesses of AUML by using strengthens of some other agent modeling languages.

CHAPTER 1
INTRODUCTION

The Unified Modeling Language UML [5, 10], was introduced for supporting Object Oriented Software Engineering (OOSE), it was developed by Object Management Group (OMG). OMG is an international, open membership, not-for-profit computer industry consortium. It is modeling standards to enable powerful visual design, execution, and maintenance of software and other processes. It defines and maintains the UML specifications which is published and promoted continuously in a set of versions. Agent Unified Modeling Language (AUML), was extended from UML for supporting Agent Oriented Software Engineering (AOSE), it was developed by Foundation for Intelligent Physical Agents (FIPA). FIPA is an international non-profit association of companies and organizations dedicated to promoting the industry of intelligent agents by openly developing specifications supporting interoperability among agents and agent-based applications [1, 2].

1.1 Agent

An agent is a computational entity such as a software program that can be viewed as perceiving and acting upon its environment and that is autonomous in that its behavior at least partially depends on its own experience [4, 8, 15].

1.2. AUML Class Diagram Problem

There are a set of problems in AUML models especially in Agent Class Diagram. The majority of these problems come from that AUML doesn't deals with knowledge; and there are no formal semantics in AUML diagrams at all [19, 20, 21]. We can classify AUML Agent Class Diagram problems in the following:

1.2.1. Planning

Any agent should have the ability to react based on plans [2, 19, 22]; in AUML there were no plans for agents, there is only an automata that defines a set of states for each agent communicative act "incoming message" that represented in state chart diagram, and what will be the reaction for that communicative act based on its internal state [2, 19, 22].

1.2.2. Roles

In AUML Role is defined as attribute without any corresponding behavior to that Role [2, 22].

1.2.3. Knowledge Base

AUML extended from Object-Oriented modeling language that means it doesn't support or contains a Knowledge Base, and the outcome will be a dummy agent that doesn't solve problems rationally [2, 22].

1.2.4. Configurations

In AUML, every time we instantiate an agent we should verify all its attributes, same thing when the agent want to die, it doesn't have any plan for dying, and these booth are very important for agents [1].

1.2.5. Unpredictable Agent behavior

Because Agent-Head Automata in AUML Agent Class Diagram is responsible for an agent behavior, AUML facing serious problems in unpredictable behavior; because all agent behaviors are implemented in a static and in a predictable way [2, 3].

1.3. Importance of the problem

AUML is the strongest agent modeling language as stated in the two surveys [29, 30], but it still doesn't have a complete solution for agent modeling; because since 2004, FIPA stopped developing and improving AUML diagrams [29].

The Agent Class Diagram is the kernel of these models and its improvement will lead to the improvements of all other models and consequentially to the enhancement of agent modeling in general.

1.4. Insufficiencies of actual approaches dealing with the problem

There are number of languages modeling Agent structure [12, 21, 25]. Some of these modeling languages based on object oriented concepts like AUML [19, 20, 21], PASSI [2], GAIA [2]. And the other modeling languages like CoMoMAS [2] and MASCommonKADS [12] use knowledge engineering to model multiagent systems. The insufficiencies of AUML (area of study) were enumerated in details above; the problem in other models is that they concentrate on a specific modeling field. Agent modeling languages are classified in two types, first type built upon the concept of knowledge engineering, and the other built upon pure object oriented concepts. Neither pure object oriented nor pure knowledge engineering can give us the optimal Agent Class Diagram Model.

1.5. The Contribution

In this research, we identified a more comprehensive set of agent structural requirements, by discovering the ones upon which are based the above works, and combining them in a complete useful in a justified way. We propose a set of new agent structures combining object oriented and knowledge engineering concepts.

CHAPTER 2
CASE STUDY

In this chapter, we will present a RobocupRescue Simulation system as a case study [34]. We will begin this chapter by taking an overview to RobocupRescue Simulation system, after that we will present all scenarios for each agent used in RobocupRescue system. After that, analyzing all these agents based on automated negotiation, knowledge, thinking, environment, events, services, goals, and resources.

2.1 RobocupRescue Simulation

Design and construction of multi-agent system infrastructures is a challenging but an interesting problem. Designing systems for soccer player robots, computer-aided design of a generic robot controller for a multi-robot system, design and implementation of automated highway systems, and the hot topic of trading agents are a few examples of the works in this field.

The engaged test bed is the rescue simulation environment. This test bed is basically designed for the goal of disaster mitigation of an earthquake. Three kinds of completely different agents are aimed to minimize the overall damage to the city. Such agents have various abilities and hence different responsibilities such as extinguishing burning buildings, rescuing injured civilians, etc. Also they are supposed to come across a mutual agreement so that their cooperation and coordination would enhance their collaborative efforts and this adds to their complexity. Two aspects of a multi-agent system with intelligence are eligible to note. The first one is the intelligence of each agent. The other one is considered with the system as a whole and it is the agents' coordination and cooperation to reach desired goals. In the implemented system both issues are considered and emphasized. It means that although each agent tries to perform his assigned tasks as perfect as possible, he tries to act so that the overall system benefits. In other words the agents are not selfish.

2.1.1 Overview of RobocupRescue Simulation

The main aim of RobocupRescue Simulation is simulating a disaster situation in a city. There is a kernel simulating the city and some simulators

simulating the disaster conditions. The parts that we have developed are the agents, and they are:

- Fire Brigades
- Ambulance Teams
- Police Forces

The main goal of the agents is to rescue more civilians. Although, ambulances are responsible for rescuing civilians, but polices will clear the roads so that ambulances and fire brigades can move in the city. Fire brigades have to extinguish fires to reduce the amount of damage (the less fire, the more alive civilians).

2.1.2 Fire Brigade Agents

In this environment, the fire brigades are responsible for controlling the spread of fire in the city, and extinguishing as many buildings as possible. For this purpose, each agent takes advantage of his visual perception and identifies the buildings on fire. For each burning building the agent autonomously tries to estimate how dangerous that building would be and how much it threatens its neighbors. After this phase, the fire brigades need to act upon the world's situation in a unified approach to increase their coordination. The most obvious approach in this phase is finding the most important buildings on fire and extinguishing them. So, the way agents calculate a building's priority plays an important role in this phase. A proposed workflow for a fire brigade is depicted in figure 2.1. As the figure suggests, the workflow contains four phases, namely: perception, analysis, decision making, and implementation. This means that the agent first receives raw information about the environment. Then by means of communication with other agents, his experiments and his experiences, the agent uses this information to gain some kind of knowledge that would be useful in the decision making phase. Then the agent's world model is investigated to find appropriate targets.

The most useful and the best estimated targets are selected in this decision making process. In the last phase the agent implements the desired actions according to the target he has chosen.

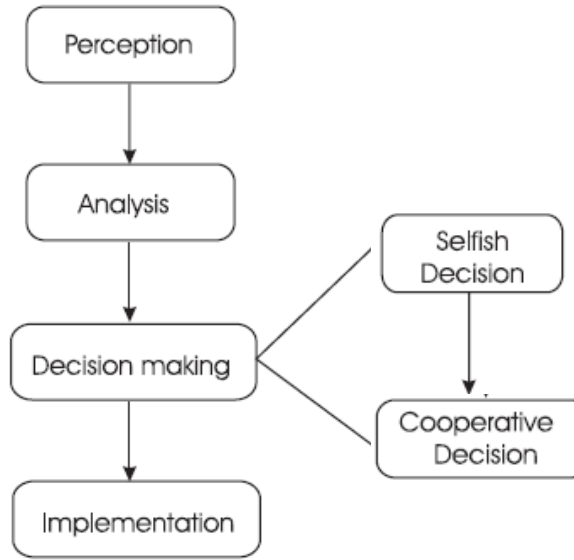


Figure 2.1: A proposed workflow for a fire brigade agent

The decision making section is the most important phase in the agent's workflow in each cycle. The fire brigade agents use a two layered architecture for this phase. In the first stage, the agents decide independently. They do not care the state of other agents and they selfishly choose some targets for their own. In the next stage, the agents try to both actively coordinate with other fire brigades and also communicate with other types of agents. In this way the overall rescue integrity is guaranteed and this collaboration enriches the result and the overall performance of the system. One of the advantages of this architecture is the independence of the two layers. This enables implementation and evaluation of different algorithm in each layer.

2.1.3 Police Force Agents

In Rescue Simulation, the police forces are supposed to clear roads. Trying to clear more roads is not the optimum action. Polices have to select the most important roads. Importance of a road is defined as how many times other agents will pass through this road in the following cycles.

In order to improve the police forces' in decision making, reinforcement learning has been used. In this method each agent has three actions as follows:

- Stay in his area: the police stay and walk around in the area his currently in.
- Help other agents: Selection of this action means the police will be leaving his current area so as to clear a specific road to help another agent achieve his goal
- Change the area of responsibility

2.1.4 Ambulance Teams

Ambulance team agent's rescue injured civilians. They obtain information of civilians by means of communication, and gathering auditory and visual information. In order to determine whether to go to rescue a civilian or move around to find an injured civilian.

2.2 Case study analysis

In Robocup Rescue system, there are three types of agents: Fire Brigade Agents, Police Force Agents, and Ambulance agents.

2.2.1 Fire Brigade Agents

Agent in Fire Brigade should contain the following structure:

- Automated Negotiation

Any agent in Fire Brigade Agents should have the ability to communicate, collaborate, cooperate, and negotiate with other agents in the same environment and with other agents in other environments. This operation should lead to reach, or to get closer, to the goal.

Automated negotiation happened in the second stage in decision making, the agents try to both actively coordinate with other fire brigades and also communicate with other types of agents.

- Knowledge

Agent must have some kind of knowledge about his experiments and experiences

- Thinking

Thinking happened in the Decision Making phase. It contains two stages: in the first stage, the agents decide independently; they do not care the state of

other agents and they selfishly choose some targets for their own. In the next stage, the agents try to both actively coordinate with other fire brigades and also communicate with other types of agents.

Each agent receives a visual perception and identifies the buildings on fire. For each burning building the agent autonomously tries to estimate how dangerous that building would be and how much it threatens its neighbors. After this phase, the fire brigades need to act upon the world's situation in a unified approach to increase their coordination. The most obvious approach in this phase is finding the most important buildings on fire and extinguishing them. So, the way agents calculate a building's priority plays an important role in this phase.

- Environment

Fire Brigade environment

- Events

All raw information about the environment received from its perception and some actions that agents do and that may affect other agents.

For example:

All visual information about burning buildings (received event)

All visual information about civilians (received event)

Fire Brigade whistle (sent event)

- Services

There are no services provided by this agent in this system

- Goals

Extinguishing burning buildings

Evacuation injured civilians

- Resources

Fire brigade vehicle

2.2.2 Police Force Agents

Agent in Police Force should contain the following structure:

- Automated Negotiation

Any agent in Police Force should have the ability to communicate, collaborate, cooperate, and negotiate with other agents in the same environment and with other agents in other environments. This operation should lead in the final stage to reach or to get closer to the goal.

- Knowledge

Agents must have knowledge about his experiments and experiences

- Thinking

Thinking in police forces needed in two areas:

Clearing roads: The police forces are supposed to clear roads. Trying to clear more roads is not the optimum action. Polices have to select the most important roads.

After selecting the most important road: Police forces have three actions to do as follows:

- Stay in his area: When this action is chosen, the police stay in his area and walks around the area he is currently in.
- Help other agents: Selection of this action means the police will be leaving his current area so to clear specific road to help another agent.
- Change the area of responsibility

- Environment

Police Force environment

- Events

All raw information about the environment received from its perception and some actions that agents do and that may affect other agents.

For example:

All visual information about roads to clear (received event)

All visual information about civilians (received event)

- Services

There are no services provided by this agent in this system

- Goals

Evacuation injured civilians

Clearing roads

- Resources

Police car and Police station

2.2.3 Ambulance Teams

Agent in Ambulance Teams should contain the following structure:

- Automated Negotiation

Any agent in Ambulance Teams should have the ability to communicate, collaborate, cooperate, and negotiate with other agents in the same environment and with other agents in other environments. This operation should lead in the final stage to reach or to get closer to the goal.

- Knowledge

All Agents must have knowledge about their old experiments and experiences

- Thinking

Ambulance team agent's rescue injured civilians. They obtain information of civilians by means of communication, and gathering auditory and visual information. The rescuer should know whether to go to rescue a civilian or move around to find an injured civilian.

- Environment

Ambulance Teams environment

- Events

All raw information about the environment received from its perception and some actions that agents do and that may affect other agents.

For example:

All visual information about civilians (received event)

- Services

There are no services provided by this agent in this system

- Goals

Evacuation injured civilians

Rescuing injured civilians

- Resources

Ambulance vehicle

Ambulance first aid

CHAPTER 3
CLASS DIAGRAM IN ACTUAL WORKS

In this chapter, we will present Class Diagrams in actual researches. We will begin this presentation by introducing Class Diagram in Unified Modeling Language, then introducing the Agent Class Diagram in Agent Unified Modeling Language that we will enhance, after that we will present a set of new Agent structures and their relationships, proposed by several actual recent research works.

3.1. Class Diagram in Unified Modeling Language UML

Class diagrams are one of the most fundamental diagrams in UML [5, 23]. They are used to capture the static relationships of software elements [5, 10, 23].

UML divides diagrams into two categories: structural diagrams and behavioral diagrams [10, 23]. Structural diagrams are used to capture the physical organization of the things in a system, while behavioral diagrams focus on the behavior of elements in a system. Class Diagram is one of UML structural diagrams that used to capture the static relationships of the software.

Class diagrams commonly contain the following:

3.1.1. Classes

Classes are the most important building block of any Object Oriented system. A class represents a group of things that have common state and behavior. In other words, a class can be seen as a set of objects that share the same attributes, operations, relationships, and semantics [5, 23]

- *Attributes*

An attribute is a named property of a class that describes a range of values that instances of the property may hold. A class may have any number of attributes or no attributes at all. An attribute represents some property of the thing you are modeling that is shared by all objects of that class. An attribute is therefore an abstraction of the kind of data or state an object of the class might encompass.

- *Operations*

An operation is the implementation of a service that can be requested from any object of the class to affect behavior. In other words, an operation is an abstraction of something you can do to an object that is shared by all objects of that class. A class may have any number of operations or no operations.

3.1.2. Interfaces

An interface is a collection of operations that are used to specify a service of a class or a component. A type is a stereotype of a class used to specify a domain of objects, together with the operations (but not the methods) applicable to the object.

3.1.3. Relationships

A relationship is a connection among entities. In Object Oriented modeling, the four most important relationships are generalizations, associations, and realizations.

- *Dependencies*

A dependency is a using relationship, specifying that a change in the specification of one entity may affect another entity that uses it, but not the reverse. Dependencies used when we want to show one thing using another.

- *Generalizations*

A generalization is a relationship between a general classifier (superclass) and a more specific classifier (subclass). With a generalization relationship from the child to the parent, the child will inherit all the structure and behavior of his parent. The child may even add new structure and behavior, or it may override the behavior of the parent.

- *Associations*

An association is a structural relationship, specifying that objects of one thing are connected to objects of another. It is used when the relationship between two elements is complex.

- *Realizations*

A realization is a semantic relationship between classifiers, where one classifier specifies a contract that another classifier guarantees to carry out. This relationship can be found in two places: between interfaces and the classes or components that realize them, and between use cases and the collaborations that realize them.

3.2. AUML Class Diagram

UML Class diagrams are modified deeply in order to encompass agent features such as mental state or interaction protocols. For pointing out the differences with class diagrams in UML, the Class Diagrams in AUML called Agent Class Diagram [2, 20, 21, 22].

AUML Agent Class Diagram is constituted by: Agent Class, Agent Communication Language (ACL), Agent Service, and relationships.

3.2.1. Agent Class

Agent Class in AUML consists of the following:

- *Name*

Three information may be supplied in agent name: instance, role, and class.

- Instance

Instances give the name of each agent involved.

- Role

A role is the behavior associated to an entity into a particular context.

- Class

In AUML, a Class is a set of agents that share the same set of agent characteristics.

- *State Description*

It defines the state of the agent. It has the same construction of attributes in UML:

[visibility] name [multiplicity] [:type] [= initial-value] [property-string]

- **Visibility** defines how an attribute can be seen and used by others.

Three cases are available: public, private and protected.

- **Name** is the name of the attribute. It is a textual string and name must be unique within the class.

- **Multiplicity** is used when it is necessary to represent several copies of the same attribute.

- **Type** represents the type of the attribute.

- **Initial-value** describes the initial value of this attribute.

- **Property-string** defines how attributes can be used: changeable is the default value and means that it is possible to update the value of this attribute, add-only is used for lists and means that only the insertion is possible, it is then not possible to update or to delete values in the list, frozen corresponds to constants, the value of the attribute cannot be modified.

- *Actions*

In AUML, two kinds of actions can be specified for an agent: proactive actions that are triggered by the agent itself and reactive actions that are triggered when receiving some message from another agent.

In its full form, the syntax of an action is:

[visibility] [pre-conditions] name [(parameter-list)] [post-conditions]

- **Visibility** defines how an attribute can be seen and used by others.
- **Parameter-list** contains both the name of the parameter and its type.
- **Pre-conditions** are constraints that must be true when an action is invoked.
- **Post-conditions** are constraints that must be true to complete an action.

- *Methods*

Methods like operations in UML. An operation is the implementation of a service that can be requested from any agent of the class to affect behavior; in other words, an operation is an abstraction of something you can do to an agent and that is shared by all agent of that class.

In its full form, the syntax of a method is:

[visibility] [pre-conditions] name [(parameter-list)]

[: return-type] [post-conditions] [property-string]

- **Visibility** defines how an attribute can be seen and used by others.
- **Parameter-list** contains a list of parameters.
- **Type** and **return-type** represent the type of the action and of the parameter respectively.
- **Pre-conditions** are constraints that must be true when an action invoked.
- **Post-conditions** are constraints that must be true to complete an action.

- **Property-string** defines how attributes can be used: changeable is the default value and means that it is possible to update the value of this attribute, add-only is used for lists and means that only the insertion is possible, it is then not possible to update or to delete values in the list, frozen corresponds to constants, the value of the attribute cannot be modified.
- *Service Description*
Service description can be seen as interface in UML. Service descriptions are represented with their operations as a class called Agent Service.
- *Supported Protocols*
Supported protocols are described as a list. Supported protocols are adorned with the roles played by the agent in these protocols.
- *Agent-Head-Automata*
The agent head automata define the behavior of an agent's head. Agents are composed of three parts: communicator, head, and body.
The agent communicator is responsible for the physical communication of the agent. The main functionality of the agent is implemented in the agent body. The agent's head behavior has to be specified with the agent head automata. Especially, this automata related to the incoming messages with the internal state, actions, methods and the outgoing messages, called the reactive behaviors of the agent.
Moreover, it defines the pro-active behaviors of an agent, i.e. it automatically triggers different actions, methods, and state-changes depending on the internal state of the agent. An example of pro-active behavior is to do some action at a specific time, e.g. an agent migrates at predefined times from one machine to another, or it is the result of some request-when communicative acts.

- *Group Representation “Organization”*

The compartment organization gives the different groups in which the agent evolves, which roles it plays and under which constraints, it can evolve in these groups. The syntax for this information written as the following:

[constraint] organization : role

- **Constraints** are written as a free-form text or as an OCL expression.
Constraints must be satisfied if agents want to belong to this group.

3.2.2. Agent Communication Language “ACL”

In order to communicate with other agents, agents use protocols and a specific agent communication language that describes the semantics associated to communicative acts.

Agent Communication Language in AUML consists of the following:

- *Agent Communication Language name*
Name of the communicative act.
- *Description*
Description of the content of this communicative act in natural language.
- *Message Content*
Description of the content associated to this communicative act. In FIPA ACL the message content is written in the “: content” compartment.
- *Semantics*
Contains the structure of this communicative act. It can be written by FIPA Semantic Language “FIPA SL”.

3.2.3. Agent Service

A service is an activity that an agent can perform and is provided to other agents.

Agent Service in AUML consists of the following:

- *Name*
Name of the service.
- *Description*
A description in natural language of this service.
- *Type*
The type of the service.

- *ACL*
List of agent communication languages used in this service.
- *Ontology*
A list of ontologies supported by the service.
- *Content Language*
A list of content languages supported by the service.
- *Properties*
A list of properties that discriminate the service.

3.2.4. Agent Class Relationships

There were many proposed relationships for AUML Class Diagram, but nothing has been accomplished yet because FIPA has some internal organizational problems [19, 20]. Agent main relationships are:

- *Generalization*
It is an organizational abstraction mechanism that creates an agent class from its constituent classes that satisfy a subclass-of to the generalized class. The inverse of generalization is known as specialization. The specialized agent class inherits the mental state of the generalized agent class.
- *Aggregation*
It is also an organizational abstraction mechanism by which an agent class is constructed from its constituent agent classes that satisfy a part-of relationship to the aggregated form.
- *Cooperation*
It is a behavioral abstraction mechanism that creates an organization or a society of cooperative agents from the constituent agent classes.

3.2.5. Applying the case study in AUML

In our case study we have three types of agents: Fire Brigades agents, Ambulance Teams, and Police Force Agents. For simplicity we will design two types of them Fire Brigades agents and Police Force Agents.

We can design an Agent Class Diagram in AUML as follows:

- Fire Brigades agents

Fire Brigades agent is an agent who is responsible for controlling the spread of fire in the city, and extinguishing as many buildings as possible. In AUML, we can model Fire Brigades agent as follows:

- Agent Class
 - *Name*
 - Instance: fireman-agent1
 - Role: fireman, rescuer, negotiator
 - Class: fire-extinguisher
 - *State Description*

```
public recover 1..10 {frozen}
public water-tank 1..1 :integer = 200 {changeable}
public multi 0..1 :integer = 40 {frozen}
public burn-building :integer {changeable}
```
 - *Actions*

```
public water-tank = 0 retreat water-tank = 200
```
 - *Methods*

```
public burn-building = 1 extinguish (burn-building, floors)
:string recover = recover+1 {changeable}
```
 - *Service Description*

Extinguishing burning buildings and rescuing civilians
 - *Supported Protocols*

Brokering interaction protocol
Query interaction protocol
Request interaction protocol
 - *Agent-Head-Automata*

It contains functions that manages all agents actions, methods, and messages
 - *Group representation*

Group representation used in systems that contains agents who have the ability to join more than one organization. In this case of Fire Brigades agent, the fireman couldn't move from one its organization to another one.

- Agent Communication Language
 - Name
 - Instance: Accept Proposal1
 - Description
 - Accept-proposal is a general-purpose acceptance of a proposal that was previously submitted. The agent sending the acceptance informs the receiver that it intends that the receiving agent will perform the action, once the given precondition is, or becomes, true.
 - Message Content
 - A tuple consisting of an action expression denoting the action to be done and a proposition giving the conditions of the agreement
It could be written like:
 - $\langle j, \text{INFORM } (i, p) \rangle \mid \langle j, \text{INFORM } (i, \neg p) \rangle$
 - Semantics
 - ```
(accept-proposal
:sender (agent-identifier :name i)
:receiver (set (agent-identifier :name j))
:in-reply-to fireman2
:content
((action (agent-identifier :name j)
(stream-content tank 19))
(B (agent-identifier :name j)
(ready fireman2)))
:language FIPA-SL)
```
  
- Agent Service
  - Name
    - Extinguishing burning buildings
  - Description
    - This service is responsible for extinguishing burning buildings, were these buildings specified by the agent itself.
  - Type
    - Public service
  - ACL
    - Accept Proposal, Agree, Cancel, and Call for Proposal
  - Ontology
    - Burning buildings ontology, Fire station ontology, and Rescuing ontology



- Content Language
  - A list of content languages like LOTA and FIPA SL
- Properties
  - None
- Message Content
  - A tuple consisting of an action expression denoting the action to be done and a proposition giving the conditions of the agreement
- Agent Class Relationships
  - Generalization
    - All a fireman agents inherits everything from its parent (Fire Brigade agent).
  - Aggregation
    - We can not define part-of relationship between agents in this environment.
  - Cooperation
    - We can make a cooperation relationship between policeman agent and fireman agent.
- Police Force Agents

Police Force Agent is an agent who is responsible, basically, for clearing roads and it may evacuate injured civilians. In AUML, we can model Police Force Agent as follows:

- Agent Class
  - Name
    - Instance: police agent1
    - Role: policeman, rescuer, negotiator
    - Class: fireman
  - State Description
    - public rank 1..1 :character = b {frozen}
    - public streets :string {frozen}
    - public street-priority :string = 0{changeable}
  - Actions
    - public street-priority < 20 change-street street-priority > 60

- Methods
  - public street-priority = 0 explore (streets) :string recover = recover+1 {changeable}
- Service Description
  - Evacuation injured civilians and clearing roads
- Supported Protocols
  - Brokering interaction protocol
  - Query interaction protocol
  - Request interaction protocol
- Agent-Head-Automata
  - It contains functions that manages all agents actions, methods, and messages
- Agent Communication Language
  - Name
    - Instance: Call for Proposal1
  - Description
    - It is a general-purpose action to initiate a negotiation process by making a call for proposals to perform the given action.
  - Message Content
    - A tuple containing an action expression denoting the action to be done, and a referential expression defining a single-parameter proposition which gives the preconditions on the action
  - Semantics
    - (call-for-proposal
    - :sender (agent-identifier :name j)
    - :receiver (set (agent-identifier :name i))
    - :content
    - ((action (agent-identifier :name i)
    - (move policeman))
    - :ontology Rescuing ontology)
- Agent Service
  - Name
    - Clearing roads
  - Description
    - This service is responsible for clearing roads.

- Type  
None
- ACL  
Accept Proposal, Agree, Cancel, and Call for Proposal
- Ontology  
Clearing roads ontology and rescuing ontology
- Content Language  
A list of content languages like LOTA and FIPA SL
- Properties  
None
- Agent Class Relationships
  - Generalization  
All a policeman agents inherits everything from its parent (Police Force agent).
  - Aggregation  
We can not define part-of relationship between agents in this environment.
  - Cooperation  
We can make a cooperation relationship between fireman agent and policeman agent.

After this presentation of a Class Diagram in UML and Agent Class Diagram in AUML, we will present a set of new Agent structures, proposed in several actual recent research works.

### **3.3. Toward Agent-Oriented Conceptualization and Implementation**

Agent Structure in Toward Agent-Oriented Conceptualization and Implementation [26] consists of Agent template that composed of the following:

- *Agent Name*
- *Location*  
Agent location in the Organization.

- *Communists "Goals"*  
All Agents' goals that should be achieved.
- *Knowledge Base*  
Knowledge Base is union of the set of domain classes for its attributes and the underlying knowledge base that is always accessible to it for its decision making.
- *Communication Languages*  
All Agent Communication Languages that an Agent can use.
- *Ontology*  
Contains the dictionary of the domain.
- *Interaction protocols*  
Indicating the types of interactions witnessed in the application, and they identify patterns of behavior.
- *Capabilities*  
A set of methods to communicate, migrate, and learn.
- *Accessory methods*  
All methods required to accomplish Agent specific responsibilities.

### **3.3.1 Applying the case study in Toward Agent-Oriented Conceptualization and Implementation**

In our case study we have three types of agents: Fire Brigades agents, Ambulance Teams, and Police Force Agents. For simplicity we will design two types of them Fire Brigades agents and Police Force Agents.

We can design an Agent Class Diagram in [26] as follows:

- Fire Brigades agents

Fire Brigades agent is an agent who is responsible for controlling the spread of fire in the city, and extinguishing as many buildings as possible. In [26], we can model Fire Brigades agent as follows:

- Agent Class
  - *Agent Name*  
fire-man1
  - *Location*  
Fire brigade organization
  - *Communities*  
Extinguishing burning buildings and rescuing civilians
  - *Knowledge Base*  
Contains all the required knowledge for the fire-man which is used in decision making
  - *Communication languages*  
FIPA ACL  
Knowledge Query and Manipulation Language (KQML)
  - *Ontology*  
Burning buildings ontology, Fire station ontology, and Rescuing ontology
  - *Interaction Protocols*  
Brokering interaction protocol  
Query interaction protocol  
Request interaction protocol
  - *Capabilities*  
It contains all functions that make the agent to communicate, migrate, and learn
  - *Accessory methods*  
Carrying civilian, carrying fireplug, move, and using the fireplug

- Police Force Agents

Police Force Agent is an agent who is responsible, basically, for clearing roads and it may evacuate injured civilians. In [26], we can model Police Force Agent as follows:

- Agent Class
  - *Agent Name*  
police-man1

- *Location*  
Police Force organization
- *Communists*  
Evacuation injured civilians and clearing roads
- *Knowledge Base*  
Contains all the required knowledge for the police-man which is used in decision making
- *Communication languages*  
FIPA ACL  
KQML
- *Ontology*  
Clearing roads ontology and Rescuing ontology
- *Interaction Protocols*  
Brokering interaction protocol  
Query interaction protocol  
Request interaction protocol
- *Capabilities*  
It contains all functions that make the agent to communicate, migrate, and learn
- *Accessory methods*  
Carrying civilian, move, and using police vehicle

### **3.4. A Methodology for Ontology Based Multi-Agent Systems Development “MOBMAS”**

Agent Structure in MOBMAS [27], consists of the following:

- *Agent class name*
- *Roles*  
Roles played by the agent. Each Role represents a set of functions, each one of them solves one agent goal.
- *Belief*  
There are two kinds of information:

- Belief State: corresponds to an agent's knowledge about a particular state of the world and captures the run-time facts about the state entities that exists in the agent's application and the Environment.
  - Belief Conceptualization: contains the Knowledge that an agent holds about the conceptualization of the world, particularly the conceptualization of the entities referred to in the Belief state.
- *Agent-Goal*  
It is the state of the world that an agent class would like to achieve.
  - *Events*  
It is defined as a significant occurrence in the environment that an agent may respond "react".
  - *Relationships*
    - Acquaintance  
Each acquaintance between agent classes is depicted as an undirected line connecting the agent classes. Inter-agent acquaintances can be derived from the acquaintances amongst roles.

### 3.4.1 Applying the case study in MOBMAS

In our case study we have three types of agents: Fire Brigades agents, Ambulance Teams, and Police Force Agents. For simplicity we will design two types of them Fire Brigades agents and Police Force Agents.

We can design an Agent structure in [27] as follows:

- Fire Brigades agents

Fire Brigades agent is an agent who is responsible for controlling the spread of fire in the city, and extinguishing as many buildings as possible. In [27], we can model Fire Brigades agent as follows:

- **Agent Class**
  - *Agent Name*  
fire-man1
  - *Roles*  
fireman, rescuer, negotiator
  - *Belief*  
Belief State: state of burning buildings, state of water-tank  
Belief Conceptualization: knowledge about dealing with burning buildings, knowledge about dealing with water-tank
  - *Agent Goal*  
Extinguishing burning buildings and rescuing civilians
  - *Events*  
State of all burning buildings, injured civilians, and other agents
- *Relationships*
  - Acquaintance  
Every agent has an acquaintance relationship with its neighbors.

- **Police Force Agents**

Police Force Agent is an agent who is responsible, basically, for clearing roads and it may evacuate injured civilians. In [27], we can model Police Force Agent as follows:

- **Agent Class**
  - *Agent Name*  
police-man1
  - *Roles*  
Police-man, rescuer, negotiator
  - *Belief*  
Belief State: state of burning buildings, roads, and civilians  
Belief Conceptualization: knowledge about dealing with burning buildings, clearing roads, and evacuation injured civilians
  - *Agent Goal*  
Evacuation all injured civilians, clearing all roads
  - *Events*  
State of all injured civilians, roads, and other agents
- *Relationships*
  - Acquaintance  
Every agent has an acquaintance relationship with its neighbors.



### 3.5. Jadex

Jadex is the famous platform for modeling Multiagent Systems [14]. It is an Agent Oriented reasoning engine for writing rational agents with XML and the Java programming language.

Agent Structure in Jadex called Agent Template, and it consists of the following:

- *Imports*

The imports tag is used to specify, which classes and packages can be used by Java expressions.

- *Capabilities*

Each agent has at least one capability which is given by the beliefs, goals, and plans, contained in an XML file.

A capability is basically the same as an agent, but without its own reasoning process. On the other hand, an agent can be seen as a collection of capabilities plus a separate reasoning process shared by all its capabilities.

In Jadex, Capabilities contains three types of information:

- *Beliefs*

Beliefs represent the agent's knowledge about its environment and itself. In Jadex the beliefs can be any Java objects. They are stored in a belief base, and can be accessed and modified from plans using the belief-base interface.

- *Goals*

Goals make up the agent's motivational stance and are the driving forces for its actions. Therefore, the representation and handling of goals is one of the main features of Jadex.

- *Plans*

Plans represent the agent's means to act in its environment. Therefore, the plans predefined by the developer compose the library of actions the agent can perform. Plans are selected in response to occurring events or goals. The selection of plans is done automatically by the system.

- *Events*

An important property of agents is the ability to react timely to different kinds of events.

Jadex supports two kinds of application-level events.

- Internal events can be used to denote an occurrence inside an agent,
- Message events represent a communication between two or more agents.

- *Properties*

Properties represented in static expressions. They can be defined in two different ways. First, you can use the properties section of the agent XML file and add an arbitrary number of properties. Secondly, the agent tag has an optional attribute "property-file" which refers to an XML file containing important definitions.

- *Configurations*

Configurations represent both the initial and/or end states of an agent type.

Initial instance elements can be declared that are created when the agent is started. This means that initial elements such as goals or plans are created immediately when an agent is born.

End elements can be used to declare instance elements such as goals or plans that will be created when an agent is going to be terminated.

- *Means-end Reasoning*

It includes a set of functions used to select and execute plans based on internal or external event.

- *Relationships*

Jadex uses all Java language relationships, without any additional features.

### **3.5.1 Applying the case study in Jadex**

In our case study we have three types of agents: Fire Brigades agents, Ambulance Teams, and Police Force Agents. For simplicity we will design two types of them Fire Brigades agents and Police Force Agents.

We can design an Agent Template in [14] as follows:

- Fire Brigades agents

Fire Brigades agent is an agent who is responsible for controlling the spread of fire in the city, and extinguishing as many buildings as possible. In [14], we can model Fire Brigades agent as follows:

- Agent Template

- *Imports*

```
<imports>
<import> fireman.* </import>
<import> search.civil.* </import>
</imports>
```

- *Capabilities*

- Beliefs

```
<beliefs>
<beliefset name="friend-Agent-names" class="String">
<fact>"fireman2"</fact>
<fact>"fireman3"</fact>
<fact>"fireman4"</fact>
</beliefset>
</beliefs>
```

- Goals

```
<goals>
<goal name="evacuation">
<parameter name="applicables" class="civilian"/>
<parameter name="result" class="civilian"
direction="in"/>
</goal>
</goals>
```

- Plans

```
<plans>
<plan name="fireman_move_plan">
<parameter name="move" class="Move">
<trigger>
<goal ref="makemove"/>
</trigger>
</plans>
```

- *Events*

```
<events>
<internalevent name="gui_update">
<parameter name="content" class="String"/>
</internalevent>
</events>
```

- *Properties*

```
<properties>
<property name="contentcodec.jade-management-s10">
new JadeContentCodec(new SLCodec(0),
JADEManagementOntology.getInstance())
</property>
</properties>
```

- *Configurations*

```
<configurations default="two">
 <configuration name="one">
 <capabilities>
 <initialcapability ref="mycap" configuration="a"/>
 </capabilities>
 </configuration>
</configurations>
```

- *Means-end Reasoning*

jadex\_rt.jar: The Jadex runtime jar includes the kernel of the Jadex reasoning engine.

- Police Force Agents

Police Force Agent is an agent who is responsible, basically, for clearing roads and it may evacuate injured civilians. In [14], we can model Police Force Agent as follows:

- Agent Template

- *Imports*

```
<imports>
 <import> policeman.* </import>
 <import> search.street.* </import>
 <import> search.civil.* </import>
</imports>
```

- *Capabilities*

- Beliefs

```
<beliefs>
 <beliefset name="friend-Agent-names" class="String">
 <fact>"policeman2"</fact>
 <fact>"policeman3"</fact>
 <fact>"policeman4"</fact>
 </beliefset>
</beliefs>
```

- Goals

```
<goals>
 <achievegoal name="moveto">
 <parameter name="location" class="Location"/>
 beliefbase.my_location.isNear(goal.location)
 </achievegoal>
</goals>
```

- Plans

```
<plans>
 <plan name="repair">
 <body> new RepairPlan() </body>
 <trigger>
 <condition> beliefbase.out_of_order </condition>
 </trigger>
 </plan>
</plans>
```

- *Events*

```
<events>
 <messageevent name="query" type="fipa"
 direction="receive">
 <value>Fipa.QUERY_REF</value>
 </parameter>
 <parameter name="content" class="String"
 direction="fixed">
 <value>"ping"</value>
 </parameter>
</messageevent>
</events>
```

- *Properties*

```
<properties>
 <property name="contentcodec.fipa-management-sl0">
 new JadeContentCodec(new SLCodec(0),
 FIPAManagementOntology.getInstance())
 </property>
</properties>
```

- *Configurations*

```
<configurations>
 <configuration>
 <capabilities>
 <initialcapability ref="mycap" configuration="b"/>
 </capabilities>
 </configuration>
</configurations>
```

- *Means-end Reasoning*

jadex\_rt.jar: The Jadex runtime jar includes the kernel of the Jadex reasoning engine.

## 3.6. Developing Role-Based Open Multi-Agent Software Systems

Agent Structure in Developing Role-Based Open Multi-Agent Software Systems [11], constituted by: Agent Class, Role Class, Relationships between Role Classes.

### 3.6.1. Agent Class

- *Attributes*

An agent is identified by its attributes such as the agent name, agent owner and agent identification.

- *Knowledge*

Knowledge about the agent and the Environment around that agent; it is represented as a special type of attributes.

- *Motivations*  
Motivations, which is defined as any desire or preference that can lead to the generation and adoption of goals, and also affect the outcome of the reasoning or behavioral task intended to satisfy those goals.
- *Sensor*  
The sensor of an agent perceives related environment changes and transforms the inputs into a set of sensor data.
- *Reasoning Mechanism*  
The reasoning-Mechanism is defined as a function that takes a set of sensor data and a set of motivations as arguments and maps them to a set of goals and sub-goals.
- *Role-Matching Mechanism*  
Based on the goals and sub-goals, the function role-Matching Mechanism further derives a set of needed roles with certain attributes. The agent then searches the role space for any available role instances that satisfies the role properties, and takes each needed available role instance from the role space to achieve its goals.
- *Committed Plan*  
To realize an agent's goal, a committed-plan is derived according to the role instances and the knowledge possessed by the agent, which includes the agent knowledge and the domain knowledge of each role instance taken by the agent.
- *Roles Taken*  
The state variable roles-Taken refers to a set of roles that are currently taken by the agent.

### 3.6.2. Role Class

- *Attributes*  
Represents a set of role attributes that describe the characteristic properties of a role, including role name and role identification.
- *Domain Knowledge*  
Specifies a set of domain knowledge that a role must possess to achieve its domain goals.
- *Domain Goals*  
Describes the current goal states and a set of domain goals that a role may achieve.
- *Domain Plans*  
Represents a set of plan trees that are used to achieve a goal or sub-goal by executing several actions in a specified order. Each plan tree is associated with a goal or a sub-goal; however, a goal or sub-goal may associate with more than one plan tree, and the most suitable one will be selected to achieve that goal or sub-goal.
- *Domain Actions*  
Refer to a set of actions that will be triggered to execute when an associated plan tree is selected to carry out.
- *Protocols*  
Defines the way how role instances may interact with each other.
- *Permissions*  
Describes the resources that are available to that role in order to achieve a goal or sub-goal.
- *Be Taken*  
It defines if a role instance has already been taken by an agent. An instantiated role is similar to the concept of object, which is an instantiated

entity of a class. It has certain goals, plan trees, and actions, it cannot start to execute until it is taken by an agent.

- *Relationships*

The relationships are only between two “role classes”. i.e. there are no relationship between Agents.

- Inheritance Relationship

An inheritance relationship between two role classes represents the generalization or specialization relationship between two role classes, where one class is a specialized version of another. Inheritance is a mechanism for incremental specification and design, whereby new classes may be derived from one or more existing classes.

- Leading Role Relationship

A leading role is responsible for hiring other roles in achieving its goal. For example, a company A is a leading role, which is responsible for hiring new employees. The leading role inherits all the data fields as well as all operations defined in the Role class. In addition, a leading role records the number of role instances that are required to achieve its goals.

- Composite Role Relationship

In the Composite-Role class, the state variable sub-Roles describe a set of role instances of type Role or its derivatives. Sub-roles can be added into or deleted from the sub-roles set.

- Aggregation Relationship

The aggregation relationship between role classes is most suitable for defining the hierarchy of a role organization. For instance, we can use a composite role to represent a team, a group or even a role organization.

- Association Relationship

The association relationship is one of the most common relationships between role classes. The association indicates an action that an instance of one role may perform on an instance of another role.



### 3.6.3. Applying the case study in Developing Role-Based Open Multi-Agent Software Systems

In our case study we have three types of agents: Fire Brigades agents, Ambulance Teams, and Police Force Agents. For simplicity we will design two types of them Fire Brigades agents and Police Force Agents.

We can design an Agent class in [11] as follows:

- Fire Brigades agents

Fire Brigades agent is an agent who is responsible for controlling the spread of fire in the city, and extinguishing as many buildings as possible. In [11], we can model Fire Brigades agent as follows:

- Agent Class
  - *Attributes*
    - Agent name: fireman1
    - Organization: Fire brigades agents
    - Type: none
  - *Knowledge*
    - Team leader: fireman2
    - Team-members: fireman1, fireman2, fireman3, fireman4
    - Agent state: working
  - *Motivations*
    - State of all burning buildings and injured civilians
  - *Sensor*
    - Environment sensor1,
    - Environment sensor2
  - *Reasoning Mechanism*
    - If environment sensor1 = injured civilian then => rescue
    - If environment sensor2 = burning building then => extinguish
  - *Role-Matching Mechanism*
    - When the agent wants to achieve extinguishing burning buildings goal, Role-Matching Mechanism derives a set of needed roles for extinguishing burning buildings goal, these roles may include: fireman role, rescuer role, policeman role. After that the agent searches the role space for the closer role properties with his derived roles. After finding the closer role, the agent makes an instance from that role to achieve extinguishing burning buildings goal.

- *Committed Plan*  
It contains all information about all domain knowledge gathered from the instances of agent roles related with knowledge possessed by the agent.
- *Roles Taken*  
Fireman role and rescuer role
- **Role Class**
  - *Attributes*  
Name: rescuer1
  - *Domain Knowledge*  
Knowledge about transporting injured civilians and first aid.
  - *Domain Goals*  
Evacuation injured civilians, give first aid to injured civilians.
  - *Domain Plans*  
Each goal in the “domain goals” may have one or more plan tree, and each plan tree contains a sequence of actions that could be done by the role to achieve its goal.
  - *Domain Actions*  
It is the set of that is used by domain plan, it could be: carrying injured civilian, transporting injured civilian, and giving first aid to injured civilian.
  - *Protocols*  
There is no specific type of agent communication protocols. We can use FIPA ACL or KQML protocols.
  - *Permissions*  
Ambulance first aid [only use]  
Oxygen cylinder [none]
  - *Be Taken*  
Rescuer
- **Relationships**
  - All relationships are between Roles*
  - Inheritance Relationship*  
All rescuer roles are inherited from Rescuer parent role.
  - *Leading Role Relationship*  
The rescuer role could be a leading role for negotiation role.

- *Composite Role Relationship*  
We can specify the relationship between fireman role and rescuer role as composite role relationship from fireman to rescuer; that means the rescuer role is a sub-role of fireman role.
  - *Aggregation Relationship*  
All fireman roles have an Aggregation Relationship with fireman Headquarters role.
  - *Association Relationship*  
The rescuer role could make association with policeman role; by this association the policeman may help the rescuer in some purposes like: carrying injured civilians.
- Police Force Agents

Police Force Agent is an agent who is responsible, basically, for clearing roads and it may evacuate injured civilians. In [11], we can model Police Force Agent as follows:

- Agent Class
  - *Attributes*  
Agent name: policeman1  
Organization: Police Force agents  
Type: none
  - *Knowledge*  
Team leader: policeman1  
Team-members: policeman1 and policeman2.  
Agent state: working
  - *Motivations*  
State of all streets and injured civilians
  - *Sensor*  
Environment sensor1,  
Environment sensor2
  - *Reasoning Mechanism*  
If environment sensor1 = injured civilian then => rescue  
If environment sensor2 = cars-in-road then => clearing roads
  - *Role-Matching Mechanism*  
When the agent wants to achieve clearing roads goal, Role-Matching Mechanism derives a set of needed roles for clearing roads goal, these roles may include: policeman role, rescuer role, negotiator role. After that the agent searches the role space for the closer role properties with his derived roles. After finding the closer role, the agent makes an instance from that role to achieve clearing roads goal.

- *Committed Plan*  
It contains all information about all domain knowledge gathered from the instances of agent roles related with knowledge possessed by the agent.
- *Roles Taken*  
Policeman role and rescuer role
- **Role Class**
  - *Attributes*  
Name: rescuer1
  - *Domain Knowledge*  
Knowledge about transporting injured civilians and first aid.
  - *Domain Goals*  
Evacuation injured civilians, give first aid to injured civilians.
  - *Domain Plans*  
Each goal in the “domain goals” may have one or more plan tree, and each plan tree contains a sequence of actions that could be done by the role to achieve its goal.
  - *Domain Actions*  
It is the set of that is used by domain plan, it could be: carrying injured civilian, transporting injured civilian, and giving first aid to injured civilian.
  - *Protocols*  
There is no specific type of agent communication protocols. We can use FIPA ACL or KQML protocols.
  - *Permissions*  
Ambulance first aid [only use]  
Oxygen cylinder [none]
  - *Be Taken*  
Rescuer
- **Relationships**

*All relationships are between Roles*

  - *Inheritance Relationship*  
All rescuer roles are inherited from Rescuer parent role.
  - *Leading Role Relationship*  
The rescuer role could be a leading role for negotiation role.
  - *Composite Role Relationship*  
We can specify the relationship between policeman role and rescuer role as composite role relationship from policeman to rescuer; that means the rescuer role is a sub-role of policeman role.

- *Aggregation Relationship*  
All policeman roles have an aggregation Relationship with their Headquarters role.
- *Association Relationship*  
The rescuer role could make association with policeman role; by this association the policeman may help the rescuer in some purposes like: carrying injured civilians.

### 3.7. Conclusion

In this chapter, we introduced a set of new agent structures. In [26], the agent structure can act rationally using its knowledge base and can make communications to other agents using communication protocols, communication languages, and ontology; but it still has a problem in planning, it doesn't have plans to reach goals, it reaches goals only by running the capabilities without any line of actions. In [27], agent structure has set of goals and can reach them by using its roles and beliefs, while it triggers the goal by events; but in this structure, the agent can't act depending on structured actions "plans". In [14], agent structure can reach its goals based on executing plans using Means-end Reasoning; this structure doesn't use a knowledge base, this states that its decisions are predefined decisions while it doesn't deals with roles. In [11], the agent can make decisions based on its knowledge, while it still can execute plans using the associated role; the problem appears when an agent receives an event which is always playing a role to achieve that goal, but the event may be a small request, and doesn't need to play a role, despite of having a knowledge by an agent that is separated from the domain knowledge; this states that when an agent plays a role then finishes playing that role it will eliminate all domain knowledge that is represented in the role knowledge. In [19, 20, 21, 22], agents knowledge is represented as attributes in State Description, and the agent does its thinking process in the Agent-head Automata, without using any inference from a knowledge base; this indicates that its decisions are built on a predefined behavior without any rationality.

From the previous review, we can collect the strengthens of these modeling languages and take in our consideration the coherence and consistency of all these components and integrate them together to make a good agent model having the maximum strengthens and the minimum problems.

**CHAPTER 4**  
**AN AUML CLASS DIAGRAM ENHANCEMENT**

In this chapter we will introduce a set of Agent Structural Requirements that gathered from several researches and trying to build a new Agent Class Diagram based on AUML Class Diagram and the strengthens of the other Agent modeling languages.

## 4.1. Agent Structural Requirements

To achieve a more complete and useful Agent structural requirements, we studied all Agents capabilities and characteristics, and upon what agents are based in actual multiagent systems researches [11, 14, 26, 27, 19, 20, 21, 22], then combine them in a complete and coherent set, as in the following:

### 4.1.1 Autonomy

When an agent has a certain independence from external control, it is considered autonomous [11]. *Without any autonomy, an agent would no longer be a dynamic entity, but rather a passive object* [18]. That means, Agents can operate and make their own decision on which action they should take, independent of humans or other agents [18], [28]. An agent is said to be an “autonomous agent” if its behavior and actions are not only based on the built-in knowledge, but also on its own experience. [18].

- *Reactive*

It is a property that allows agents to perceive and react to the changes in their environment [26]. *An agent should be capable of adapting itself for any changes taking place in its environment in order to carry out the functionalities upon which it has been designed* [28].

- *Proactive*

It might be possible to build agents that only act towards their goal or only react to their environment [33].

Agents can react not only to specific method invocations but to observable events within the environment, as well. *Proactive agents will actually poll the environment for events and other messages to determine what action they should take. In short, an agent can decide when to say "go"* [26], [28].

In reality, many agents are designed as hybrid agents, possessing both reactive and proactive characteristics [26], [28]. The challenge then is for the designer to balance these two very different behaviors in order to create an overall optimal behavior [26].

#### 4.1.2 Communication

An agent can communicate with other agents on a common topic of discourse by exchanging a sequence of messages in a speech-act-based language that others understand [26]. The domain of discourse is described by its Ontology [7].

Ontologies describe the concepts and their relationships with different levels of formality in a domain of discourse. For example, the ontology of a mobile device can specify its concepts using the following terms: manufacturer, memory, screen size. *It used mainly by Agents negotiation for sharing and reusing knowledge.*

- *Automated Negotiation*

Negotiation is one of the vaguest aspects pertaining to many different mechanisms of interaction to employ a set of existing conditions and constraints of a discrete-agents environment in order to optimize specific solutions and decisions [1]. *Negotiation is mainly based on the cooperation between agents, which have the desire to share their knowledge and conflicting interests [26]. That is, in a problem where each agent has different local knowledge negotiation can be an effective method for finding the one global course of action which maximizes utility without having to send all the local knowledge bases to a central location for consideration.*

- *Cooperation*

Cooperation means that the agent is able to coordinate with other agents to achieve a common purpose; *Cooperation involves communication and interaction between agents to achieve common goals [26].*

#### 4.1.3 History

Mechanisms are required to provide a historical recording of the agent's actions; *so that agent behavior can be audited and that agents can evaluate prior actions [26].*



#### **4.1.4 Social ability**

Interaction that is marked by friendliness or pleasant social relations, that is, where the agent is affable, companionable, or friendly.

*A software agent may have to possess social ability, to be capable of interacting with other agents to provide its service [17, 33].*

#### **4.1.5 Rationality**

It is the assumption that an agent will act in order to achieve its goals, and will not act in such a way as to prevent its goals being achieved; at least insofar as its beliefs permit [26], [33]. *An agent should act rationally, based on its mental state, toward achieve its internal pleasure [33].*

#### **4.1.6 Unpredictable behavior**

Agents may also employ some degree of unpredictable (or nondeterministic) behavior [33]. When observed from the environment, an agent can range from being totally predictable to completely unpredictable [18].

#### **4.1.7 Learning ability**

When designing an agent, the developer may furnish it with all the intelligence needed to carry out its assigned roles to achieve specific goals [26]. However, this is not the best approach for either the agent or the designer. An agent should be able to learn, in a dynamic manner, from its environment and from other agents, and employ the incorporated information from this cognition to build and update its knowledge base [33]. *A real-world agent should be able to learn from past experiences in order to improve on future solutions.*

#### **4.1.8 Mobility**

Mobility is the ability for a software agent, under certain circumstances, to migrate from one machine to another in a heterogeneous network environment to process its tasks locally on that machine [26]. When the immigration decision takes place, the agent is temporarily suspending its processing until it moves to the new destination to resume it [30].

#### 4.1.9 Reasoning

Reasoning is the decision-making mechanism, by which an agent decides to act on the basis of the information it receives, and in accordance with its own objectives to achieve its goals [33].

#### 4.1.10 Multi-agent planning

Multiagent planning is concerned with planning by multiple agents (globally), or inside the agent itself (locally). It can involve agents planning for a common goal, an agent coordinating the plans or planning of others, or agents refining their own plans while negotiating over tasks or resources.

- *Deliberative*

Deliberative agents can learn and/or evolve; that is, they can change their behavior based on their experience with other agents and the environment [18]. The key component of a deliberative agent is a central reasoning system. Deliberative agents generate plans to accomplish their goals [28]. *The deliberative agent increasing the agent's ability to generate a plan to successfully achieving its goals* [28]. The main problem with a purely deliberative agent occurs when dealing with real-time systems is reaction time [33].

## 4.2. An AUML Class Diagram Enhancement

In the following, we present a new Agent Class Diagram, overcoming the precedent AUML Agent Class Diagram insufficiencies and contributing to the achievement of the above identified structural requirements. It consists of Agent Class, Role Class, Agent Communication Language “ACL”, Agent Service, Agent Relationships, and Role Relationships.

### 4.2.1 Agent Class

Agent can be Static or Dynamic

- *Static Agent class* is one whose instances are required to play all of the assigned roles through out their lifetime.
- *Dynamic Agent class* is one whose instances may change their active roles from one time to another.

Agent Class in our approach consists of:

- *Identification “Name”*
- *Location*

Any Agent location should contain the name of the organization and the environment that it belongs to; because the organization may belong to more than one environment.

- **An Organization:** It is a group of Agents working together to achieve common purposes.
- **An Environment:** It involves determining all the entities and resources that the Multiagent System can exploit, control, or consume.

Written as: organization@environment

- *Middle Agents*

A set of all middle Agents that an agent can register itself to them.

Middle Agents compartments is related to Agent Service; when an agent register itself to middle agents it should register all its services “Agent Service” into the middle agents.

Written as: middle-gent@organization

- *Supported Protocols*

Supported protocols are described as a list. Supported protocols are adorned with the roles played by the agent in these protocols.

- *Destination*

A set of all Organizations that the Agent can go to (mobile), and a list of constraints for each one of them.

Each Organization may have a set of constraints that each agent should comply before it registers itself in that Organization.

Written as: organization@environment {set of constraints}

*Destination used to enhance the mobility requirement, and it is used by mobile agents to locate all organizations they can go to.*

- *Roles*

Every Agent must have at least one role, this role used to achieve some purposes by activating a plan or a set of plans.

MAS Environment define a set of System-Tasks that must be achieved, those System-Tasks are mapped into Role-Tasks, and the Role-Tasks are grouped into Roles [27]; that means each Role is a set of coherent tasks, each task solve one problem i.e. each task achieves one Goal.

An Agent can play a role in a static or a dynamic way in follows:

- Static Roles: A set of roles that an agent can play in asynchronous
- Dynamic Roles: A set of roles that an agent can play in synchronous

The default state of the role is to be synchronous; but we may have some role that couldn't be played in synchronous, because they may cause some conflict if they played together. These conflicts may happen from accessing resources and executing plans. *Role classes used to enhance the Multiagent Systems Planning, but it uses only the internal planning.*

Written as: (Static/Dynamic) role@organization@environment

- *Ontology*

It should contain all system Ontologies: Concepts, Interaction Protocols, Domain Ontology, and Application Ontology.

Ontology compartment is related to beliefs compartment; that means when an agent wants to build its ontology, one of the most important trees in that ontology is belief tree.

*Ontology used to enhance the Communication, Negotiation, and Cooperation between Agents.*

- *Environment resources*

A set of all resources that an Agent can access from the environment with its restrictions; these restrictions may be read, write, read-only, write-only, or combining between any two of them.

resource@environment {read/write/read-only/... }

- *Configurations*

Configurations represent both the initial and/or end states of an agent type.

- **Initial instance elements** can be declared that are created when the agent is started. This means that initial elements such as goals or plans could be created immediately when an agent is born.

- **End elements** can be used to declare instance elements such as goals or plans that will be created when an agent is going to be terminated.

*Configurations used to make the agent have basic characteristics like the human being.*

- *Belief “Knowledge Base”*

It is a small local Knowledge Base, that contains the state of the Agent and the state of its Environment.

Agent can extend its knowledge base from: its Environment, other Agents reactions, and from its own decisions.

The main purpose of this Knowledge Base is to let the agent have the ability to think by its own “mind” and take its own decisions. These decisions may always be under evaluation and set on a Knowledge Base as a good or a bad decision. i.e. learn from its experience.

*Agent Knowledge Base could be useful in Learning Ability, Deliberative, Communication, and Autonomy for an Agent.*

- *Intentions “Goals”*

It defines a set of goals “internal or external” that an agent can achieve.

Goals may be composed of sub-goals.

Goals in general are extracted from the Multiagent System as “system tasks” after that they are grouped into roles; each role contains a set of coherent goals and their plans.

*Mainly, Agent Goals used to support Agent Planning, and Agent Proactive behavior.*

- *Actions*

We can define a set of Actions that an Agent can perform for some small requests that don't need for playing a Role to achieve it "Re-Active Actions", the other type of Actions "Pro-Active Actions" are actions triggered by the agent itself, e.g. using timer. These two types of Actions define reactive and proactive behavior of the Agent.

Actions can trigger some events from events compartment.

*Agent Actions used to identify Agent Proactive and Reactive behaviors.*

- *Events*

There are two types of Events:

- Agent Generated Events

*Internal Events:* Agent generates this kind of events for its internal purposes and it is occurred with some agent's actions.

*External Events:* Agent generates this kind of events for outside purposes like send an event to an Environment resource.

- Agent Received Events

Agent may receive an event from outside the agent. i.e. receive events from the environment, or another Agent.

In general, events maybe generated by:

- Agents via execution of there Actions
- Environment Resources via the execution of their services
- Human users via their inputs to the system
- Outside "Environment" Sensors

*Events used to support Agent Proactive behavior.*

- *Methods "Accessory Methods"*

This compartment contains all methods that are required by an Agent to provide it while doing its work.

*Accessory Methods are all methods that support all Agent basic operations.*

- *Inference engine*

This compartment is responsible to derive answers from a Knowledge Base. The agent may use it for decision making.

*The main purpose of this compartment is to derive logical decisions to the Agent; these decisions used by some Agent Mental Reasoning methods to provide them with logical answers.*

- *Mental Reasoning methods*

This compartment contains a set of all agent intelligent methods like: Negotiation, Learning, Prediction, and Migration. Most of these methods use the agent Knowledge Base decisions by asking the Inference engine.

*By using this compartment, the Agent can do an automated negotiation, rational thinking, and automated mobility.*

- *Automata-Reasoning Mechanism*

Automata-Reasoning Mechanism is the mind of the Agent; it uses a technique to decide the appropriate action to trigger, or the appropriate role to play or the appropriate service to run, or to do nothing, based on the incoming event or the incoming communicative act and also based on the Agent internal state.

Agent in AUML doesn't have the ability to make decisions based on mental reasoning because it doesn't have a knowledge base, so its decisions based on a predefined algorithm designed on Agent-Head-Automata.

*Automata-Reasoning Mechanism mainly supporting the Rationality of Agents. And it causes the unpredictable behavior of the Agent.*

Figure 4.1 represents the Automata-Reasoning Mechanism in our approach.

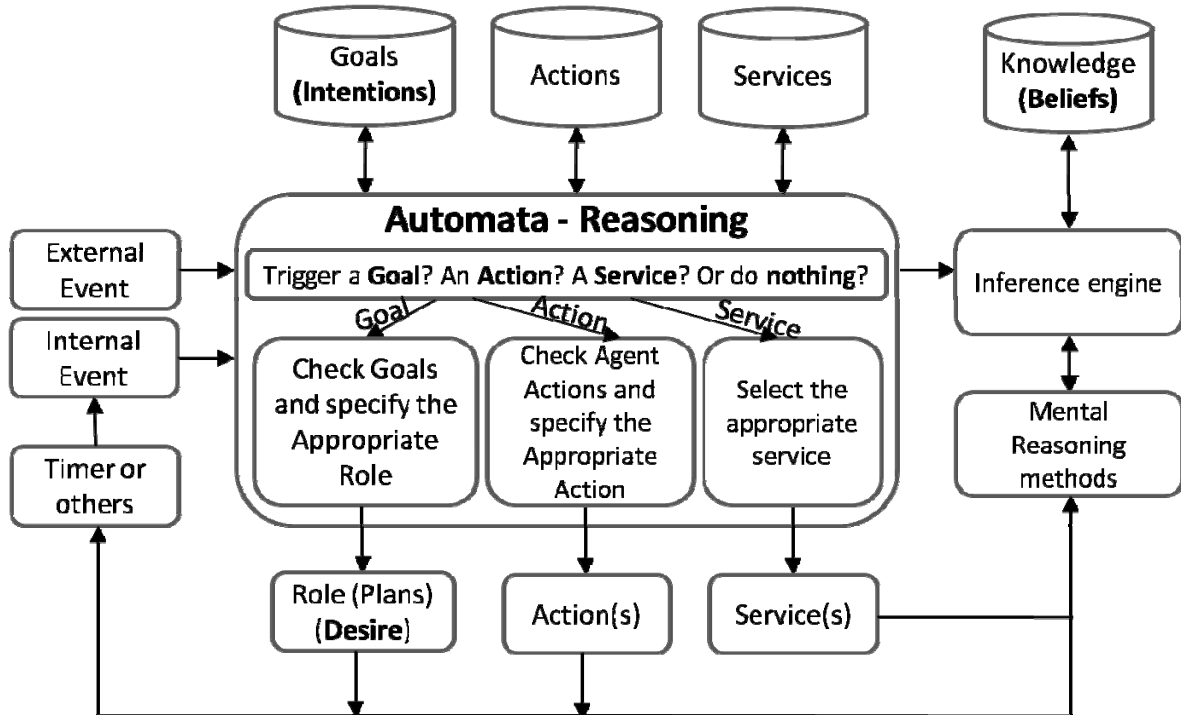


Figure 4.1: Automata-Reasoning Mechanism

The Automata-Reasoning runs when it receives an event which is made internally “normally by timers” or externally “from the outside”. After that, the Automata-Reasoning read this event “that contains Communicative Act” and make the decision that will be either triggering a goal, an action, a service, or to do nothing based on its previous experience and knowledge, if it triggers a goal the Automata will execute the appropriate role from the goals set, this role will run a plan to reach the goal, this plan could use some mental reasoning methods and could trigger a goal. If the Automata trigger an Action, it will directly execute an action or a sequence of actions based on the actions set, and it also could use the mental reasoning methods and could triggers an event. Finally if the Automata execute a service, it will directly execute a service from a set of services based on the requested service, and it also could use the mental reasoning methods and could triggers an event.

The Automata can use inference engine to reason about some information in the knowledge base for the ultimate purpose of formulating new conclusions.



<b>Agent Class</b>
Static / Dynamic
Agent Class Name
Location
Middle Agents
Supported Protocols
Destination
Roles
Ontology
Environment resources
Configurations
Belief
Intentions
Actions
Events
Methods
Inference engine
Mental Reasoning methods
Automata-Reasoning

Figure 4.2: Agent Class

#### 4.2.2 Role Class

We propose a Role Class consisting of the following:

- *Role Name*
- *Role Tasks*

Role tasks are derived from system tasks, and they contain a set of all tasks that a role can perform.

- *Desire “Plans”*

It should be described in “Role Framework” because one goal, for example “gaining money” can be done in two ways “Plans”; either by “legal job” or by “stealing money”, these two “Plans” needs two roles “Worker” and “Thief”.

Plans can be seen as a sequence of actions that used to achieve a goal or sub-goal. A goal or sub-goal may associate with more than one plan, and the most suitable one will be selected by using Plan Selector.

- *Plan Selector*

It is a function that is used to select the appropriate Plan based on the set of Role Tasks.

- *Actions*

A set of all Actions that all Plans need to accomplish their work.

In general we can see an action as a special type of functions.

- *Events*

Role can generate Events and send them to the Agent.

<b>Role Class</b>
<b>Role Name</b>
<b>Role Tasks</b>
<b>Desire</b>
<b>Plan Selector</b>
<b>Actions</b>
<b>Events</b>

Figure 4.3: Role Class

### 4.2.3 Agent Communication Language “ACL”

Agent uses Agent Communication Language to send messages to other agents. It has the same structure as it is in AUML.

*Agent Communication Language is the basic for any Agent to Agent general Communication; that means any Agent Communication, Negotiation, or Cooperation is done only by sending Communicative Acts.*

### 4.2.4 Agent Service

A service is an activity that an agent can perform and is provided to other agents.

Agent Service in AUML consists of the following:

- *Name*

Name of the service.

- *Description*

A description in natural language of this service.

- *Type*  
The type of the service.
- *Methods*  
A set of methods supporting the service.

<b>Agent Service</b>
<b>Service Name</b>
<b>Description</b>
<b>Type</b>
<b>Methods</b>

Figure 4.4: Agent Service

#### 4.2.5 Agent Class Diagram relationships

In our approach we propose the following relationships:

- *Agent Class relationships*
  - Inheritance (parent, child)  
Inheritance between two agents represents that the “child” Agent may takes his entire parent characteristics, or a set of his parent characteristics. Figure 4.5, shows the inheritance relationship.

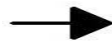


Figure 4.5: Inheritance relationship

- Play (Role, Agent)  
The play relationship specifies the roles that an Agent can play. When an *Agent class is related to a role class by the play relationship it means that the Agent instance can play one or more role instances*. Figure 4.6, shows the play relationship.



Figure 4.6: Play relationship

- Control (controller, controlled)  
The control relationship defines that the controlled entity must do anything that the controller entity requests. Figure 4.7, shows the control relationship. *By using control, we can specify the hierarchical chain management in Agents society.*



Figure 4.7: Control relationship

- Dependency (client, supplier)

In this relationship, the “client Agent” may be defined to be dependent on another one the “supplier Agent” to do its job. The dependency relationship specifies that the client agent cannot completely do its job unless it asks the supplier. Figure 4.8 shows the dependency relationship *This relationship is used between two agents, the first one “the client” didn’t have the service that the “supplier” has; in this case it sends a request to the “supplier” to do that service.*

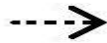


Figure 4.8: Dependency relationship

- Aggregation (aggregator, part)

The aggregator agent may use the functionalities available in its parts. The parts do not need to know that they are being aggregated to an aggregator, but the aggregator should know each of its parts. Figure 4.9, shows the aggregation relationship.



Figure 4.9: Aggregation relationship

- *Role Class relationships*

- Inheritance (parent, child)

Inheritance between roles means that a role may inherit some functionality from another role; this functionality maybe a plan, an action, or other. Figure 4.10, shows the inheritance relationship.

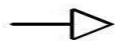


Figure 4.10: Inheritance relationship

- Leading (leader, subordinator)

The leader relationship between roles used to define the hierarchy of a role organization; that means when we define a leader and a subordinator, the subordinator should obey his leader. Figure 4.11, shows the leading relationship.



Figure 4.11: Leading relationship

### 4.3. Applying the case study in AUML Class Diagram Enhancement

In our case study we have three types of agents: Fire Brigades agents, Ambulance Teams, and Police Force Agents. For simplicity we will design two types of them Fire Brigades agents and Police Force Agents.

We can design an Agent Class Diagram in our approach as follows:

- Fire Brigades agents

Fire Brigades agent is an agent who is responsible for controlling the spread of fire in the city, and extinguishing as many buildings as possible. In our approach, we can model Fire Brigades agent as follows:

- Agent Class
  - *Identification*  
fireman-agent1
  - *Location*  
Fire Brigades agent organization@burning building environment
  - *Middle Agents*  
There is no need for middle agents
  - *Supported Protocols*  
Brokering interaction protocol  
Query interaction protocol  
Request interaction protocol
  - *Destination*  
In this case all agent organization will be the same, because the fireman agent couldn't be a policeman agent.  
Fire Brigades agent organization@ fire station environment  
Fire Brigades agent organization@ street environment
  - *Roles*  
(dynamic) fireman@ Fire Brigades agent@ burning building  
(dynamic) rescuer @ Fire Brigades agent@ burning building  
(static) negotiator @ Fire Brigades agent@ burning building
  - *Ontology*  
Burning buildings ontology, Fire station ontology, Rescuing ontology, and all interaction protocols ontology

- *Environment resources*
  - Water-tank @ burning building environment {read, write}
  - Water-tank @ street environment {read, write}
  - Water-tank @ fire station environment {read only}
  - Oxygen cylinder @ burning building environment {read, write}
  - Oxygen cylinder @ street environment {read only}
  - Oxygen cylinder @ fire station environment {read only}
  
- *Configurations*
  - There is no need for configuring initial instance elements or end elements
  
- *Belief*
  - It contains a small local Knowledge Base about the state of the Agent and the state of its Environment. And also, contains the experiences and expertise gathered in the agent lifetime.
  
- *Intentions*
  - Extinguishing burning buildings and rescuing civilians.
  
- *Actions*
  - Reactive actions
    - When a fireman agent detects that it has run out of water, all plans will be ignored except triggering “out of water” action, and going to fill out water-tank
  
  - Proactive action
    - There is no need to proactive behavior in this agent.
  
- *Events*
  - Agent generated events
    - Agent internal events
      - While a fireman agent extinguishing burning building he detects that it has run out of water, internal events will be triggered to the agent itself called “run out of water”, after that all plans will be ignored except triggering “out of water” action, and going to fill out water-tank then completing the plans.
    - Agent external events
      - Calling for help from any agents by broadcasting.
      - Whistling to all fireman team to regroup.
  - Agent received events
    - State of all burning buildings, injured civilians, and other agents.

- *Methods “Accessory Methods”*  
It contains all methods that support Automata-Reasoning Mechanism to do its jobs. Like:  
Methods for calling actions.  
Methods for dealing with inference engine.  
Methods for querying for the best goal.
- *Inference engine*  
This compartment is responsible to derive answers from a Knowledge Base. The agent may use it for decision making.
- *Mental Reasoning methods*  
It contains all Negotiation, Learning, Prediction, and Migration methods.
- *Automata-Reasoning mechanism*  
This compartment is responsible of receiving and sending all messages and events from itself to other agents and it is responsible for managing all the system functionalities including running Actions, Methods, Inference engine, and Mental reasoning methods to achieve its main goals.
- **Role Class**
  - *Role Name*  
Fireman\_transport1
  - *Role Tasks*  
By using this role, agent can do the following:  
Going to the burning building
  - *Desire*  
In general, desire could be represented as a set of trees which contains a set of related Role Actions. For example, a fireman could has a plan to go to the burning building named “going to the burning building” that contains the following sequence of actions: when the warning alarm fired, go to the extinguisher vehicle, and take the information for the destination, then drive the vehicle, after that when reaching the destination, get off the vehicle, grouping together, go to the burning building.
  - *Plan Selector*  
In this case we have only on possible plan. So we don’t have to select the most appropriate one.

- *Actions*
  - Going to the burning building plan contain the following actions:
  - Go to the extinguisher vehicle
  - Go to the burning building by vehicle
  - Get off the vehicle
  - Grouping together outside the vehicle
  - Go to the burning building by foot
  - Extinguish the fire in the building
  - Finding civilians
  
- *Events*
  - Calling for help from any agents by broadcasting.
  
- **Agent Communication Language**
  - *Name*
    - Accept Proposal1
  
  - *Description*
    - Accept-proposal is a general-purpose acceptance of a proposal that was previously submitted. The agent sending the acceptance informs the receiver that it intends that the receiving agent will perform the action, once the given precondition is, or becomes, true.
  
  - *Message Content*
    - A tuple consisting of an action expression denoting the action to be done and a proposition giving the conditions of the agreement
    - It could be written like:
    - $\langle j, \text{INFORM}(i, p) \rangle \mid \langle j, \text{INFORM}(i, \neg p) \rangle$
  
  - *Semantics*
    - (accept-proposal
    - :sender (agent-identifier :name i)
    - :receiver (set (agent-identifier :name j))
    - :in-reply-to fireman2
    - :content
    - ((action (agent-identifier :name j)
    - (stream-content tank 19))
    - (B (agent-identifier :name j)
    - (ready fireman2)))
    - :language FIPA-SL)



- Agent Service
  - We will not using agent service in this case because the agents are not doing a service because the agent doesn't have the ability to reject doing his work. We can replace the using of service by using the ACL from the agent itself. But in this case we must use one common type of content language.
  
- Agent Class Relationships
  - *Inheritance (parent, child)*  
Inheritance (Fire Brigade agent, fireman1)
  
  - *Play (Role, Agent)*  
Play (fireman, fireman1)  
Play (rescuer, fireman1)  
Play (negotiator, fireman1)
  
  - *Control (controller, controlled)*  
Control (fireman-agent1, fireman-agent2)  
Control (fireman-agent1, fireman-agent3)  
Control (fireman-agent1, fireman-agent4)
  
  - *Dependency (client, supplier)*  
There is no need for this relationship type in this case.
  
  - *Aggregation (aggregator, part)*  
There is no need for this relationship type in this case.
  
- Role Class relationships
  - *Inheritance (parent, child)*  
Inheritance (fire brigade agent, fireman)
  
  - *Leading (leader, subordinator)*  
Leading (fireman1, fireman2)  
Leading (fireman1, fireman3)  
Leading (fireman1, fireman4)

- Police Force Agents

Police Force Agent is an agent who is responsible, basically, for clearing roads and it may evacuate injured civilians. In our approach, we can model Police Force Agent as follows:

- Agent Class
  - *Identification*  
policeman-agent1

- *Location*  
Police Force Agent organization@ burning building environment
- *Middle Agents*  
There is no need for middle agents.
- *Supported Protocols*  
Brokering interaction protocol.  
Query interaction protocol.  
Request interaction protocol.
- *Destination*  
In this case all agent organization will be the same; the fireman agent couldn't change its organization to be a policeman agent.  
Police Force Agent organization@ Police station environment  
Police Force agent organization@ street environment
- *Roles*  
(dynamic) policeman@ Police Force agent@ burning building  
(dynamic) rescuer @ Police Force agent@ burning building  
(static) negotiator @ Police Force agent@ burning building  
(dynamic) policeman@ Police Force agent@ street  
(dynamic) rescuer @ Police Force agent@ street  
(static) negotiator @ Police Force agent@ street
- *Ontology*  
Clearing roads ontology and rescuing ontology, and all interaction protocols ontology.
- *Environment resources*  
Vehicle@ street environment {read, write}
- *Configurations*  
There is no need for configuring initial instance elements or end elements.
- *Belief*  
It contains a small local Knowledge Base about the state of the Agent and the state of its Environment. And also, contains the experiences and expertise gathered in the agent lifetime.
- *Intentions*  
Evacuation injured civilians and clearing roads.

- *Actions*
  - Reactive action
 

When a policeman agent saw an injured civilian who needs to be carried out of that place, all plans will be ignored except triggering “rescuing civilian” action.
  - Proactive action
 

There is no need to proactive behavior in this agent.
- *Events*
  - Agent generated events
    - Agent internal events
 

A policeman agent may have an internal timer that remembers him to check streets every hour.
    - Agent external events
 

Calling for help from any agents by broadcasting.
  - Agent received events
 

State of all streets, injured civilians, and other agents.
- *Methods “Accessory Methods”*

It contains all methods that support Automata-Reasoning Mechanism to do its jobs. Like:

  - Methods for calling actions
  - Methods for dealing with inference engine
  - Methods for querying for the best goal
- *Inference engine*

This compartment is responsible to derive answers from a Knowledge Base. The agent may use it for decision making.
- *Mental Reasoning methods*

It contains all Negotiation, Learning, Prediction, and Migration methods.
- *Automata-Reasoning mechanism*

This compartment is responsible of receiving and sending all messages and events from itself to other agents and it is responsible for managing all the system functionalities including running Actions, Methods, Inference engine, and Mental reasoning methods to achieve its main goals.

- **Role Class**
  - *Role Name*  
Policeman\_clearing\_roads1
  - *Role Tasks*  
By using this role, agent can have only one role task which is clearing all streets from civilians and cars.
  - *Desire*  
In general, desire could be represented as a set of trees which contains a set of related Role Actions. For example, a policeman may have a plan to clearing some roads named “clearing roads” that contains the following sequence of actions: when the policeman arrived to the specific place, he must find the most important street to clear, after specify the street, he beginning clearing that street from cars, then clearing it from civilians.
  - *Plan Selector*  
In this case we have only on possible plan. So we don’t have to select the most appropriate one.
  - *Actions*  
Clearing roads plan contains the following actions:  
Finding the most important street to clear  
Clearing that street from cars  
Clearing that street from civilians  
Search again for the most important street to clear
  - *Events*  
Calling for help from any agents by broadcasting.
- **Agent Communication Language**
  - *Name*  
Instance: Call for Proposal1
  - *Description*  
It is a general-purpose action to initiate a negotiation process by making a call for proposals to perform the given action.
  - *Message Content*  
A tuple containing an action expression denoting the action to be done, and a referential expression defining a single-parameter proposition which gives the preconditions on the action.

- *Semantics*

```
(call-for-proposal
:sender (agent-identifier :name j)
:receiver (set (agent-identifier :name i))
:content
((action (agent-identifier :name i)
(move policeman))
:ontology Rescuing ontology)
```
- **Agent Service**

We will not using agent service in this case because the agents are not doing a service because the agent doesn't have the ability to reject doing his work. We can replace the using of service by using the ACL from the agent itself. But in this case we must use one common type of content language.
- **Agent Class Relationships**
  - *Inheritance (parent, child)*

Inheritance (Police Force agent, policeman1)
  - *Play (Role, Agent)*

Play (policeman, policeman1)  
Play (rescuer, policeman1)  
Play (negotiator, policeman1)
  - *Control (controller, controlled)*

Control (policeman-agent1, policeman-agent2)
  - *Dependency (client, supplier)*

There is no need for this relationship type in this case.
  - *Aggregation (aggregator, part)*

There is no need for this relationship type in this case.
- **Role Class relationships**
  - *Inheritance (parent, child)*

Inheritance (Police Force Agent, policeman)
  - *Leading (leader, subordinator)*

Leading (policeman1, policeman2)

**CHAPTER 5**  
**THE EVALUATION**

## 5.1. Introduction

The practical evaluation of the proposed model may necessitate the implementation of a multiagent system application based on this approach and evaluating its effectiveness. However:

- This requires a team work over a large period of time, because it needs:
  - For each agent class we should build a knowledge base [6, 24],
  - For each agent class we should build an ontology for the entire System [11], [14], [19, 20, 21], [26], [27],
  - For each agent class we have to implement the Agent Interaction Protocols “AIP” for communication [19, 20],
  - For each role class we must design at least one plan tree with set of actions,
  - For each agent we should build at least one Mental Reasoning method.
  - We should build at least one organization in one environment.
  - At least we should build two agents to make a community.
- Research works in this area are mainly theoretical rather than practical [13, 16, 18, 19, 26],
- Not availability of data on real multiagent system applications, nor evaluation criteria [16, 18, 19].

This leads us to a comparative evaluation with other approaches based on some evaluation criteria.

However, a graphical environment supporting the development of Agent Class Diagram according to our proposal is developed by extending the Unified Modeling Language Class Diagram tool provided by Rational Rose.

The semantics research in Unified Modeling Language semantics have not been admitted yet, and the semantics of AUML is far to be established. So, this work is not concerned on any formal semantics definitions of the proposed concepts, but it is interested in their practical identifications and their informal definitions.

## 5.2. Comparison with similar works

We defined a set of evaluation criteria used to evaluate Agent Structure; these criteria are based on the completeness of Agent structure and the representation of its mental state. Also we enter the case study, from chapter 2, to make the evaluation more richness.

### 5.2.1 Multiagent System structural requirements

- *Agent Mental State*

In Toward Agent-Oriented Conceptualization and Implementation [26] as it is in MOBMAS [27], Agent Mental State is represented by Knowledge Base and belief compartments, this compartment doesn't contain agent previous experiences or expertise it contains only the basic underlying knowledge for the agent. In our case study, there are several problems arises when using this minimum level of knowledge representation. One of these problems is that agent couldn't learn from its previous experiences or expertise. The second problem is that the agent couldn't derive new solutions from its knowledge, because it is limited and couldn't be rise in the agent lifetime. For example when the fireman agent doing something wrong while rescuing a civilian from a burning building, in [26, 27], the agent couldn't have the ability to learn from his previous experiences, and when facing the same situation in the future he will fall in the same mistake again and again.

In Jadex [14], it is illustrated by Beliefs compartment; which is represented by objects, that means reasoning, in this agent, is done in objects state "attributes"; and that is insufficient for an agent to build a good decision. In our case study, agent must have the ability to learn from its previous experiences and deriving a new solutions based on its experiences and its underlying knowledge. For example the policeman agent, in our case study, should have an underlying knowledge about the environment "burning building environment" and he should have knowledge about his old experiences, which makes him qualified while facing problems in the environment. In [14], the agent has only the underlying knowledge about the environment, but doesn't have the ability to learn from his previous experiences, because he doesn't save his experiences in the knowledge base, that means he will fail each time to solve any simple reasoning problem.



In *Developing Role-Based Open Multi-Agent Software Systems* [11], it is represented by a Knowledge compartment, which contains knowledge about the agent itself and the environment around that agent. This is the best agent mental state modeling for two reasons, agent can make its decisions based on reasoning process which is done upon the knowledge, and second reason is that the agent can handle internal and external knowledge. As an example from our case study, when the fireman agent doing something wrong while rescuing a civilian from a burning building, the agent have the ability to learn from his previous experiences by save this experience into his knowledge, and when facing the same situation in the future he will try to solve the problem by using another solution.

In AUML [19, 20, 21], it is represented by State Description “attributes”, only about the agent internal state; which states that the agent doesn’t have a knowledge about its environment, and it doesn’t have the ability to make a rational decisions using some reasoning process. That means, and by using the previous example, the fireman agent will not be able to know that his solution is a good solution or a bad one; furthermore, when facing the same situation in the future he will fall in the same mistake.

Here, in our approach, we replace the AUML State Description compartment with Agent Belief which contains a knowledge base; that contains mainly of knowledge about agent environment, other agent’s reactions, and his previous decisions and reactions. That means, by using the previous example, the fireman agent will have a knowledge that contains all information about his organization as “Fire Brigade Agent”, environments “Burning Building, fire station, and street”, surrounding agents, resources and experience, in other words, the agent will have a complete set of knowledge about everything surrounding him. This complete set of knowledge, allows the agent to make decisions based on good underlying knowledge. In our previous example, the agent will be able to know that his solution is a good solution or a bad one; furthermore, when facing the same situation in the future he will not fall in the same mistakes.

- *Agent Mental Behavior*

In *Toward Agent-Oriented Conceptualization and Implementation* [26], agent mental behavior is indicated by Capabilities compartment; that represents all

agents intelligent functionalities; in this model, as it is in MOBMAS [27], an agent doesn't have the ability to act based on plans which destroy one of the main agent requirements, which is Multi-agent planning. Furthermore, in [27], agent doesn't have the capability to make a decision based on its opinion, while it doesn't have any intelligent functional behavior like reasoning, negotiation, and automated mobility. When applying [26, 27], in our case study, the fireman agent couldn't have plans for going to the burning building or to extinguishing them; which means the agent will not be able to perform his jobs unless there is a supervisor who controlling all agents moves.

In Jadex [14], it is represented by Means-end Reasoning, which is responsible for selecting the appropriate plan based on the occurrence of internal or external event; this reveals that the agent doesn't have the capability to make a decision based on its opinion, it is only select a plan based on event, and it couldn't reject to do any event. If we model our case study using [14], the fireman agent will not be able to execute a plan in the middle of executing another one; that means when the fireman agent executing the plan "extinguishing burning building" and after the fireman agent entering the burning building, he finds a civilian inside the building, in this case, the fireman agent should terminate the "extinguishing burning building" plan and starting the "rescuing injured civilians" plan, which is not allowed using this model language.

In Developing Role-Based Open Multi-Agent Software Systems [11], it is represented by Reasoning Mechanism, Role Matching Mechanism, in agent class, and Role Class; which is used to select the appropriate role class based on the occurred event; in this structure the Agent couldn't negotiate, predict, and reason, because it doesn't have a knowledge base, so it can't make decisions in rational way. In our case study, the policeman agent or fireman agent couldn't establishing a negotiation; that means all agents working alone, there is no cooperation between fireman agents to achieve a common goal which is "extinguishing burning building", and the same thing for policeman agents, they couldn't cooperate together to portion streets between them.

In AUML [19, 20, 21], it is indicated by Agent-Head-Automata; this automata is responsible of the reactive behaviors of the agent; that is related to the incoming messages with the internal state, and the result will be the outgoing messages, this automata is predefined in the agent body, so, the agent decisions

will be based on a predefined behavior that is opposite of basic agent behavior. In our case study, the fireman agent may facing an unpredictable and unimplemented problem, such as, when the fireman brigades entering a burning building, and after a while, the burning building collapsed slowly, in this case the agent will not be able to make an unpredictable behavior to survive.

In our approach, we changed the previous Agent-Head-Automata by Automata-Reasoning Mechanism, Mental Reasoning methods, Inference engine, and Agent Role; by using our approach the agent can act rationally based on its decisions (using a knowledge base), using its plans (by playing roles), and communicate or negotiation to reach its goals. By using the previous example, the fireman agent could make an unpredictable behavior to survive from the collapsed burning building; moreover, the fireman agent can make a negotiation with other fireman agents to rescuing him using the communication protocols, or the fireman agent may run some surviving plan to rescuing himself.

- *Dealing with Events*

In Toward Agent-Oriented Conceptualization and Implementation [26], agents do not deal with events “neither internal nor external”; in other words an agent will not be applicable to act based on what happened around it, in this case agent autonomy will be so weak. In our case study the fireman agent or the policeman agent will not be able to see any injured civilians or burning buildings; that means they are useless.

In MOBMAS [27], and AUML[19, 20, 21], events are classified only as environmental external events, same as in Role Design is in Developing Role-Based Open Multi-Agent Software Systems [11], which contain internal sensors for watching environment events; which means, agents couldn't send an announcement to any other agents or to the environment. In our case study, the fireman agent may need to ask for help from any surrounding agents; that means it should send a broadcasting event to all surrounding agents calling for help. In this modeling language the agent is capable only for sending messages; which mean it should send messages to all agents, one by one, calling for help, and this takes a long time.

In Jadex [14], events are classified as message events “comes from outside agents” and internal events, which is a good classification but it lacks the

dealing with events that might become from the environment itself, or from human users via their inputs to the system, or from outside sensors. In this case, the policeman agent or the fireman agent will not be able to see any injured civilians or burning buildings; that means they are useless.

In our approach, we extend the old AUML events by classifying events as agent internal generated events “internal and external”, agent received events, and role generated events; by using these kinds of events we can cover all kind of events that might be generated by agents or by roles, either internal or external. In our case study, the fireman agent may need to ask for help from any surrounding agents; that means it should send a broadcasting event to all surrounding agents calling for help; that means the agent should generate external event for that; which is resides in the agents events compartment. In the second case, the policeman agent or the fireman agent should be able to see injured civilians or burning buildings; for this case, there is an agent received events; which is responsible for receiving any external events, either from environment or agents.

- *Dealing with Roles*

In Toward Agent-Oriented Conceptualization and Implementation [26], there are no roles; which means we can define each agent to play one role. For example, each role in the fireman agent (fireman, rescuer, and negotiator) will be implemented as a separate agent, which is so weak.

In Jadex [14], MOBMAS [27], and AUML [19, 20, 21]; they use role on their agent structure as an attribute only; which means that all plans will be placed inside the agent itself; and that leads the agent to be more complex and its functionality will be conflicted by each other. For example, each role in the fireman agent (fireman, rescuer, and negotiator) will be implemented inside the agent himself, which means that the agent will not be able to use them in parallel in an efficient way.

The most complex Role Design is in Developing Role-Based Open Multi-Agent Software Systems [11], roles are composed of all the domain mental state, plans, actions, permissions, and goals; which states that all role knowledge, actions plans, and goals are not based on its decision neither than its plan nor its goals, this design will wok efficiently in our case study.

In, *our approach*, Role Class is changed from the previous AUML roles, which contain only role names, by adding a new role class that contains role tasks, plans, plan selector, events, and actions; which indicates that the agent can execute its plans based on the played role. For example, when the fireman agent wants to go to the burning building from his fire station, he should use a fireman-transport role; which contains all needed data for fireman transportation.

- *Dealing with Goals*

In all agent modeling languages, Toward Agent-Oriented Conceptualization and Implementation [26], Jadex [14], MOBMAS [27], and Developing Role-Based Open Multi-Agent Software Systems [11], agent has to define at least one goal; and this idea is not proposed in AUML [19, 20, 21]; which means that an agent in AUML doesn't act depending on its goals, but acting only on the basis of its incoming messages and on its internal state.

In *our approach*, agent has a set of goals; which indicates that an agent will act not only based on the incoming messages, but also on its goals.

- *Middle Agents*

Non of all agent modeling languages, Jadex [14], MOBMAS [27], Toward Agent-Oriented Conceptualization and Implementation [26], Developing Role-Based Open Multi-Agent Software Systems [11], and AUML [19, 20, 21], use middle Agents; which is used to register the agents' services, so when an external agent wants to request a service; it asks the middle agent about the appropriate agent, after that it connects to that agent and asks it for that service.

In *our approach* when an agent registers itself in an organization, it should register its services in the middle agents.

In our case study there is no need for middle-agents between agents.

### **5.3 Model Coherence**

By our case study, applied to our model, we stated the coherence and consistency of the all proposed compartments integrated together.

**CHAPTER 6**  
**CONCLUSIONS AND FUTURE WORK**

## 6.1. Conclusions

In this research, we build a complete and coherent agent structure that gathered from several research works, then enhancing the AUML Class Diagram by new techniques or by adapting its old model to reach the complete and coherent agent structure.

AUML Enhancements

### 6.1.1. Adaptation of

- State Description by represent it using a knowledge base,
- Events, by adding external events.

### 6.1.2. Introducing a set of new compartments

- Replacing Agent-Head-Automata that represent the agent mental behavior, with Automata Reasoning Mechanism, Inference engine, Mental Reasoning methods, and Role Class.
- Middle Agents,
- Configurations,
- Environment resources,
- Intentions,
- Destination.

## 6.2. Future Work

Till now, AUML has extended a small set of UML diagrams (Communication Diagram, Interaction Overview Diagram, Sequence Diagram, and Class Diagram); these diagrams were produced for first draft at the beginning of 2004 and till now there are no any extensions, and because the differences between Agent and Object, there may be some necessity to remodel these diagrams to comply with agent characteristics. In the case of AUML Class Diagram, there are a set of problems that still not solved yet by our approach, these problems are:

### 6.2.1. Semantics

The main problem in AUML that still exists in our approach, that it doesn't have a formal semantics.

### 6.2.2. Deliberative

The agent should have the ability to change its plans based on its experience. In our approach we have a set of predefined fixed plans.

## **REFERENCES**



1. 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, pp 141-157, 2005.
2. Wooldridge, M., Jennings, N. R., & Kinny, D. Methodology for agent-oriented analysis and design. *Proceedings of the 3rd International conference on Autonomous Agents*. Volume 9, Issue 3, 1999
3. Bernhard Bauer. UML Class Diagrams Revisited in the Context of Agent-Based Systems, in *Proceedings of the Second International Workshop on Agent Oriented Software Engineering*, Montreal Canada, pp 1–8, 2001.
4. Cossentino, M., and Potts, V. A CASE tool supported methodology for the design of multi-agent systems. *Proceedings of International Conference on Software Engineering Research and Practice*, Las Vegas, pp 315-321, 2002.
5. Dan Pilone, and Neil Pitman, *UML 2.0 in a Nutshell*, O'Reilly, 2005.
6. DeLoach, S. A., Wood, M. F., and Sparkman, C. H. Multi-agent systems engineering. *The International Journal of Software Engineering and Knowledge Engineering*, Volume 11, Issue 3, pp 303-328, 2001.
7. Devedzic, Vladan, Dragan Djuric, and Dragan Gasevic, *Model Driven Architecture and Ontology Development*. New York, Springer, 2006.
8. Garcia, A. Agents in Object-Oriented Software Engineering, in *Software, Practice and Experience*, Volume 34, Issue 5, pp 489-521, 2004.
9. Giunchiglia, F., Mylopoulos, J., and Perini, A. The Tropos software development methodology: Processes, models and diagrams. *Proceedings of AOSE Workshop*, Volume 2585, pp 162-173, 2003.
10. Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*, Addison Wesley Professional, 2005.
11. Haiping Xu, Xiaoqin Zhang, and Rinkesh J. Patel, *Developing Role-Based Open Multi-Agent Software Systems*. *International Journal of Computational Intelligence Theory and Practice* Volume 2, Issue 1, pp 39-56, 2007.
12. Iglesias, C., Garijo, M., González, J., Velasco, J. *Analysis and design of multiagent systems*, Verlag, 1997.
13. S. Kirn and O. Herzog and P. Lockemann and O. Spaniol. *Multiagent Engineering - Theory and Applications in Enterprises*, *International Handbooks on Information Systems*, Springer, 2006.

14. Jadex BDI Agent System - Online Documentation, [http://vsis-www.informatik.uni-hamburg.de/projects/jadex/jadex-0.96x/doc\\_overview.php](http://vsis-www.informatik.uni-hamburg.de/projects/jadex/jadex-0.96x/doc_overview.php), 2007.
15. James Odell, Objects and Agents Compared, *Journal of Object Technology*, Volume 1, Issue 1, pp 42-53, 2002.
16. James Odell, Agent Technology: What is it and why do we care?, *Enterprise Architecture*, Volume 10, Issue 3, pp 1-25, 2007.
17. James Odell, Van Parunak, Mitch Fleischer, and Sven Breuckner. Modeling Agents and their Environment, *Agent-Oriented Software Engineering* Volume 2585, pp 16-31, 2002.
18. Jin, Xiaolong, Jiming Liu, and Kwok Ching Tsui, *Autonomy Oriented Computing: From Problem Solving to Complex Systems Modeling (Multiagent Systems, Artificial Societies, and Simulated Organizations)*, Springer, New York, 2004.
19. Marc-Philippe Huget. Agent UML Class Diagrams Revisited, in *Proceedings of Agent Technology and Software Engineering*, 2002.
20. Marc-Philippe Huget, Agent UML Notation for Multiagent System Design. *IEEE Internet Computing*, IEEE, 2004.
21. Marc-Philippe Huget and James Odell and Bernhard Bauer. The AUML Approach. In *Methodologies and Software Engineering for Agent Systems*, 2004.
22. Marc-Philippe Huget, An Application of Agent UML to Supply Chain Management *Proceedings of Agent Oriented Information System*, Bologna, Italy, 2002.
23. Michael Jesse Chonoles and James. *UML 2 for Dummies*, Hungry Minds, 2003.
24. N. Glaser, Contribution to Knowledge Modeling in a Multi-agent Framework. PhD thesis, not published, University of Henry Poincare, France, 1996.
25. Padgham, L., & Winikoff, M, Prometheus: A methodology for developing intelligent agents. *Proceedings of the 3rd AOSE Workshop*, Bologna, Italy, 2002.
26. Patrik K. Biswas. *Architectural Design of Multi-Agent Systems: Technologies and Techniques*, IGI Global, 2007.
27. Quynh Nuh Tran, *MOBMAS - A methodology for Ontology Based Multi-Agent Systems Development*. PhD thesis, not Published, University of New South Wales, Australia, 2005.
28. Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni. *Multi-Agent Programming: Languages Platforms and Applications (Multiagent Systems, Artificial Societies, and Simulated Organizations)*. Springer. Volume 15, pp. 125–147, 2005.

29. Salaheddin J. Juneidi and George A. Vouros, Evaluation of Agent Oriented Software Engineering Main Approaches. Proceedings of the IASTED International Conference on, Software Engineering, Austria, 2004.
30. Silaghi Gheorghe Cosmin. Software Engineering Approaches for Design of Multi-agent Systems, Economy Informatics, Volume 5, 2005.
31. Silva Garcia, Anarosa Brandao, Christina Chavez, Carlos Lucena, Paulo Alencar. Taming Agents and Objects in Software Engineering, Software Engineering for Large-Scale Multi-Agent Systems, Springer, Volume LNCS 2603, pp 1-25, 2003.
32. Simon Kendal, Malcolm Creen. An Introduction to Knowledge Engineering, Springer, 2006.
33. Stefan Kirn, Otthein Herzog, Peter Lockemann, and Otto Spaniol. Multiagent Engineering: Theory and Applications in Enterprises (International Handbooks on Information Systems). New York, Springer, 2006.
34. Zafar Habibi, Mazda Ahmadi, Ali Nouri, Mayssam Sayyadian, Mayssam M. Nevisi, Implementing Heterogeneous Agents in Dynamic Environments, a Case Study in RoboCupRescue, Lecture Notes in Computer Science, Springer Berlin, Volume 2831, 2004

## الخلاصة

تعتبر الأنظمة متعددة الوكلاء (Multi-agent systems) من الأنظمة الحديثة في مجال علم الحاسوب, حيث أنها تستخدم في الأنظمة التي تستخدم نوعاً من الذكاء الاصطناعي والتشغيل الآلي. هناك الكثير من لغات النمذجة المستخدمة لنمذجة الأنظمة متعددة الوكلاء. و من أحد اللغات الشهيرة المستخدمة لذلك لغة هي (AUML), حيث تعتبر من لغات النمذجة المشهورة و التي تركز بالأساس على (UML 2.0). هذه الدراسة تتناول في طياتها أوجه الضعف الموجودة و التي لم يتم حلها حتى الآن في (AUML) وذلك من حيث قدرتها على التعاطي مع كل متطلبات (Agents). و من ثم محاولة تحسين أداءها وذلك عن طريق دمج بعض مكونات لغات نمذجة مختلفة و استخدام بعض أوجه القوة من هذه اللغات لمحاولة الإرتقاء بتصميم (AUML).

# تحسين و تطوير لغة النمذجة (AUML)

من قبل

محمود عدنان إبراهيم صوالحة

بإشراف

د. سعيد غول

قدمت هذه الرسالة استكمالاً لمتطلبات  
الحصول على درجة الماجستير في علم الحاسوب

عمادة البحث العلمي و الدراسات العليا

جامعة فيلادلفيا

شباط / ٢٠٠٨