Extending Object-Z for Multi-Agent Systems Specification

Anarosa A. F. Brandão[1], Paulo Alencar[2] e Carlos J. P. de Lucena[1]

[1] PUC-Rio, Computer Science Department, SoC+Agent Group,
Rua Marques de São Vicente, 225 - 22453-900, Rio de Janeiro, RJ, Brazil
{anarosa, lucena}@inf.puc-rio.br

[2] University of Waterloo, Computer Science Department, Computer Systems Group
Waterloo, Ontario, N2L 3G1 Canada
palencar@csg.uwaterloo.ca

PUC-RioInf.MCC 21/04 Junho, 2004

Resumo. O crescimento da World Wide Web nos últimos anos alavancou a pesquisa científica e tecnológica em várias áreas do conhecimento, dentre elas a engenharia de software. Neste sentido, considerando-se o crescimento do interesse e uso de agentes de software no desenvolvimento de sistemas, foram e estão sendo desenvolvidas novas técnicas de engenharia de software para dar suporte ao desenvolvimento deste tipo de sistemas. Neste artigo, apresentamos AgentZ, uma notação formal que combina os conceitos e relacionamentos propostos no framework conceitual TAO (Taming Agents and Objects) com as linguagens de representação formal Z e Object-Z. AgentZ foi construída para ser uma notação formal que permita a verificação de modelos de design, um assunto chave dentro da pesquisa em engenharia de software de sistemas multi-agentes, e, portanto, que pode ajudar a melhorar a qualidade de sistemas multi-agentes.

Abstract. Agent-orientation has gained increased importance in recent years with the emergence and growth of the World Wide Web, both as an area of study in itself, and as a component of other disciplines such as software engineering. As a result, this has led to an increased amount of research developing new informal and formal software engineering techniques to support agent-oriented system specification, design, validation and development. In this paper, we present a formal notation called AgentZ that combines the model concepts and structure proposed by TAO (Taming Agents and Objects), a conceptual framework that provides conceptual foundations for agents and objects, with the well known Z and Object-Z formal representation languages. AgentZ was built to provide a formal notation that allows the verification of design models, a key issue within the emerging agent-oriented software engineering research and, as a result, it can help to improve the quality of MAS.

This work is partially supported by CNPq-Brazil under the project “Engenharia de Software de Sistemas Multi-Agentes”, number 552068/2002-0 and by individual grants from CNPq-Brazil.

1. Introduction

Nowadays the use of software systems in business organizations is rapidly increasing and globalization is one of the trends behind the transformation of many of those systems into distributed information systems (DIS). Agent-orientation is emerging as a new paradigm in software engineering that seems to be well-suited for developing DIS using a multi-agent system (MAS) approach. In addition, the distribution of information and associated technologies indicate that open and distributed architectures are becoming essential for the development of software systems [13]. The complexity associated with these systems is growing fast and, in order to deal with this problem, the research community is developing new methodologies based on agent concepts. Several research results address the analysis and design development phases, and some modeling languages and methodologies such as MAS-ML[12], AUML [1], Gaia[18], MaSE [17], and AORML [16] have been proposed in the literature.

Software engineering of MAS is at its early stage of development and many related concepts and abstractions are still under development and formalization. Our research group[1] is working to provide a better understanding of the interplay between the notions of agents and objects in the development of MAS from a software engineering perspective. Following this path, we have first developed TAO [13], a conceptual framework that provides an approach to agent and object-based software engineering, while defining an ontology that establishes the essential concepts or abstractions that can be used to develop MAS. Thereafter, one of our colleagues developed MAS-ML. MAS-ML is a multi-agent system modeling language that extends UML (Unified Modeling Language) [15], based on the structural and dynamic properties presented in TAO. In this work we present a first version of a formal notation called AgentZ that combines the structure proposed in TAO for agents and objects with the well-known formal notation Z [14] and Object-Z[3,4].

The combination of agent and object-orientation structure with Z took advantage of the idea adopted by Object-Z of encapsulating state and operations in a single structure. AgentZ extends Object-Z with new constructs to enhance structuring and to accommodate new agent-orientated entities such as agents, organizations, roles and environments.

AgentZ is a formal notation that allows the verification of design models, a key issue within the emerging research area of agent-oriented software engineering. We believe that AgentZ can help produce better system design models and, as a result, will help pave the way for the development of MASs using a MDA approach [10].

The structure of this work is as follows. In Section 2 we describe the TAO conceptual framework, the MAS-ML modeling language and the Object-Z formal notation. In Section 3 we describe the abstract syntax of Agent-Z and some of its semantics. In Section 4 we illustrate our approach by an example and in Section 5 we describe some related work. Finally, in Section 6 we present our conclusions and future work.

[1] www.teccomm.les.inf.puc-rio.br/socagents

2. Background

The main reason for developing AgentZ is that agents and objects are conceptually different in essence. Actually, the state and behavior of agents and objects differ in a way that prevents the general use of object-orientation extension mechanisms. The state of an object is composed of stored information about itself, about the environment and about other objects, and does not have any predefined structure as well. On the other hand, the agent state is composed of its goals, beliefs, plans and actions, and does have some predefined structure. Object behavior is defined by the operations an object can perform, and agent behavior is guided by the agency properties such as autonomy, interaction and adaptation.

Our research group has developed TAO, and, as a spin-off from this investigation, one of our colleagues developed MAS-ML by augmenting the UML metamodel with some new metaclasses that represent agent abstractions. Based on the idea that MAS-ML extends the UML metamodel, we have decided to extend the Object-Z metamodel in a similar way to define AgentZ. Therefore, it will be possible to define a formal mapping between MAS-ML models and AgentZ specifications, since such a mapping can be defined between UML models and ObjectZ specifications [8]. In the following we introduce the TAO conceptual framework and briefly describe MAS-ML and Object-Z.

The TAO conceptual framework

TAO (Taming Agents and Objects) is a conceptual framework developed by our research group for two main purposes. The first was to better understand the interplay between the notions of agents and objects, and the second was to provide a systematic approach to agent and object-based software engineering. This framework defines an ontology with the essential abstractions that can be used to develop MAS.

In TAO, a MAS comprises classes and instances of agents, objects and organizations. TAO entities are agents, objects, organizations, roles (agent and object roles), environments and events. Agents, organizations, and objects inhabit environments [7, 9]. While objects represent passive elements, such as resources, agents represent autonomous elements that manipulate objects. Agents have beliefs and goals, they know how to execute some actions and plans, and they are always playing a role in an organization. An organization describes a set of roles [2] that may limit the behavior of its agents, objects and sub-organizations [19]. Furthermore, organizations have axioms that guide the behavior of their agents based on the roles they play. Agents and objects can be members of different organizations and play different roles in each of them [11]. Agents may interact with each other and cooperate either to achieve a common goal, or to achieve their own goals [21]. Agent interactions with elements that are not agents are based on relationships. Interactions between agents occur when messages described in a specific communication language are exchanged. An agent can interact with agents from the same organization or with agents from a different one. The relationships defined on TAO are *Inhabit, Play, Ownership, Control, Dependency, Association*, and *Aggregation*.

The MAS-ML modeling language

MAS-ML is a MAS modeling language that extends UML in a conservative way and is based on TAO metamodel [12]. MAS-ML adds new metaclasses to the UML metamodel in order to include TAO concepts that are not object-oriented. In the following, we describe the MAS-ML metamodel.

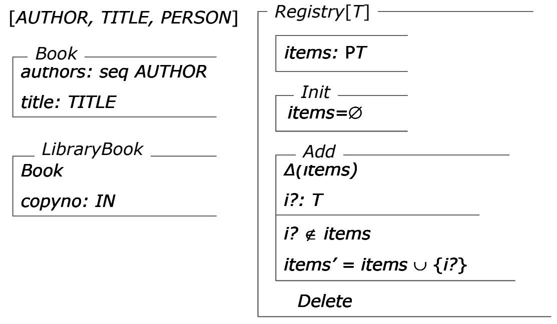
The MAS-ML metamodel extends (part of) the UML metamodel by adding new metaclasses to the metamodel and by creating new stereotypes to support agent-orientation. The new metaclasses *AgentClass*, *OrganizationClass*, *ObjectRoleClass*, and *AgentRoleClass* extend the UML metaclass *Classifier,* and they refer to agent, organization, object role, and agent role TAO abstractions, respectively. In addition, the new metaclasses *PlanClass*, *ActionAgent* and *ProtocolClass* extend the UML metaclass *Behavioral Feature* and they refer to plans, actions and protocols that an agent can perform.

The metaclass *AgentClass* has the structural features *Belief* and *Goal*. These features are defined by using stereotypes based on the *Attribute* metaclass, which is a specialization of the *StructuralFeature* UML metaclass. Moreover, an *AgentClass* is also associated with the new metaclasses *ActionAgent, PlanClass*, and *ProtocolClass*.

In our formal notation, we describe these new metaclasses as new constructs using a Z-like style following the Object-Z [3] idea. Furthermore, the new stereotypes define new or given sets. In this work we will focus on the *AgentClass*, the *AgentRoleClass*, and the *OrganizationClass* constructs.

Object-Z

Object-Z is an extension of the formal specification language Z to accommodate object-oriented concepts. This extension introduces a *class* structure to Z structures that encapsulates a single state schema with the operations that may affect that state [4]. Instances of *class* structures are called *objects*. In addition, Object-Z supports (multiple) inheritance, which means that complex classes can be specified in terms of simpler ones. One of the main benefits of Object-Z is to improve the clarity of large specifications through enhanced structuring [4]. Fig. 1 shows an example of an Object-Z specification for the library problem.



```
[AUTHOR, TITLE, PERSON]

Book
  authors: seq AUTHOR
  title: TITLE

LibraryBook
  Book
  copyno: IN

Registry[T]
  items: PT

  Init
    items=∅

  Add
    Δ(items)
    i?: T
    ----------
    i? ∉ items
    items' = items ∪ {i?}

  Delete
```

Fig. 1 Example of an Object-Z specification

3. AgentZ

Our formal notation is called AgentZ, which is obtained by adding some new constructs to Object-Z. As in the case of the Object-Z definition, we are also using the Z notation from [14]. In the following, we present the formal notation that will be illustrated in Section 4 through an example related to the market domain.

Basic Concepts

The metamodel of AgentZ, which is similar to a fragment of the MAS-ML metamodel, is shown in **Fig. 2**. As in MAS-ML the Class metaclass is borrowed from the UML metamodel, the (object) class in AgentZ is the *Class* schema borrowed from Object-Z. According to TAO, agents, objects, organizations, roles, and environments are elements, meaning that they have properties and relationships with other elements. They are represented in the AgentZ metamodel by defining Agent, Organization, AgentRole, Environment, and Object as extensions of Element. As agents are always playing at least one role in an organization, they depend on agent roles. Organizations extend agents in the sense that organizations can be seen as agents in the context of other organizations.

According to TAO, elements are entities that have properties and relationships, and agents are elements that extend objects by redefining their state and behavioral properties. In this sense, both agents and objects are element extensions that redefine the element state and behavioral properties. In this work we have followed ideas related to the definition of Object-Z to create a formal notation that accommodates both agent and object-orientation. The structural properties of an agent are expressed by its beliefs and goals. The agent behavioral properties are expressed by its plans and actions and the roles it plays. The state of an agent is a mental state that , in contrast with the state of an object, includes structural (goals and beliefs) and behavioral (actions and plans) properties. This is one of the main reasons why we have chosen to extend Object-Z by augmenting it with new structures instead of simply extending the *Class* schema of Object-Z.

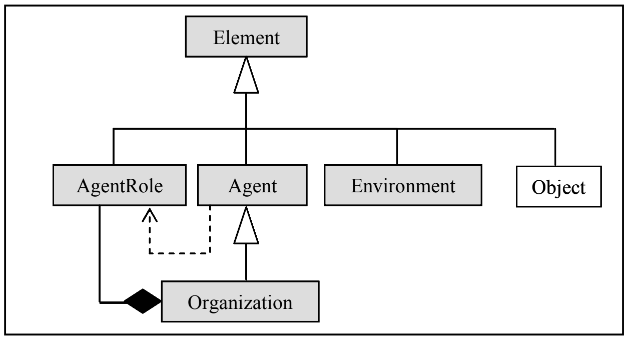


Fig. 2. The AgentZ metamodel

We adopt d'Inverno and Luck's [6] definition for Attribute: an attribute is every *perceivable feature*, and the set of all attributes is defined as [*Attribute*]. Beliefs and goals are perceivable features that can be defined as subsets of *Attribute*. The set of relationships defined in TAO are shown in **Fig. 3**. The main Agent Z extension is the new *AgentClass* construct. We note that in principle agents cannot be simply defined as a stereotype of objects since they are object extensions that redefine the object state and behavior, and we cannot use the *Class* construct of Object-Z to represent them.

Relationship := *Inhabit* | *Play* | *Specialization* | *Control* | *Dependency* | *Association* | *Aggregation* | *Ownership*

Belief == P*Attribute* and *Goal* == P*Attribute*

Fig. 3. Initial sets

In order to define some relationships described in **Fig. 3**, we define sets of names for each new construct of our formal notation. Each name is of type *String*, and this makes it possible to perform operations involving names. All the names are elements of the given set [*Names*]. The relationships are binary relations whose signatures are specified in **Fig. 4**.

An *AgentClass* is the structure for the agent abstraction. Each *AgentClass* instance is an agent and it has a name ending with the keyword *_Agent*. Agents are related to agent roles, to organizations, to objects, and to environments. An *AgentRoleClass* is the structure for the agent role abstraction. Each *AgentRoleClass* instance is an agent role and it has a name ending with the keyword *_AgRole*. Agent roles are related to agents, objects, and organizations. An *OrganizationClass* is the structure for the organization abstraction. Each *OrganizationClass* instance is an organization that has a name ending with the keyword *_Org* and that is related to all TAO elements. *Elements* are agents, objects, organizations, agent and object roles, and environments. An *Environment* is the structure for the environment abstraction. Its instance is an environment and it has a name ending with the keyword *_Env*. Moreover, an environment is related to citizens. *Citizens* are agents, objects, and organizations. Agents are always playing roles, and these roles define the protocols the agent must follow to communicate with other agents. Protocols are specified via the *ProtocolClass* schemas and their names end with the keyword *_Protocol*. Agent communication is defined through messages. Messages are specified via the *MessageAgent* schemas and their names end with the keyword *_Msg*. An agent achieves its goals through the execution of plans, and a plan consists of agent actions. A *PlanClass* is the structure that defines a plan an agent can execute. It is always related to a goal. In our formal notation, a *PlanClass* schema name ends with the keyword *_Plan*. Agent actions are specified by *ActAgentClass* schemas and their names end with the keyword *_ActAgent*. Agents' beliefs and goals can be described as logical expressions. Organizations have axioms that describe the laws that guide the behavior of their agents.

Table 1. List of the sets of names

Set_Name	description
Agent_Name	set of all AgentClass schema names, all of them ending with the keyword **_Agent**
AgRole_Name	set of all AgentRoleClass schema names, all of them ending with the keyword **_AgRole**
Obj_Name	set of all Class schema names, all of them ending with the keyword **_Obj**
ObjRole_Name	set of all ObjectRoleClass schema names, all of them ending with the keyword **_ObjRole**
Org_Name	set of all OrganizationClass schema names, all of them ending with the keyword **_Org**
Env_Name	set of all Environment schema names, all of them ending with the keyword **_Env**
ActAgent_Name	set of all ActAgent schema names, all of them ending with the keyword **_ActAgent**
Plan_Name	set of all PlanClass schema names, all of them ending with the keyword **_Plan**
Protocol_Name	set of all ProtocolClass schema names, all of them ending with the keyword **_Protocol**
Msg_Name	set of all MessageAgent schema names, all of them ending with the keyword **_Msg**
Citizen_Name	Agent_Name **U** Obj_Name **U** Org_Name
Roles_Name	AgRole_Name **U** ObjRole_Name
Element_Name	Citizen_Name **U** Roles_Name **U** Env_Name
Abstraction_Name	Element_Name – Env_Name
Aggregated_Name	Abstraction_Name – Agent_Name

The relationships defined in TAO are *Inhabit*, *Play*, *Dependency*, *Association*, *Control*, *Aggregation*, *Specialization/Inheritance*, and *Ownership*. They are binary relations between elements. Elements encompass agents, objects, organizations, agent roles, object roles, and environments. The *Inhabit* relationship relates each citizen to the environment in which it is registered. The *Play* relationship relates each citizen to the role it plays. *Dependency* is a relationship between object roles and between agent roles. It fixes that a change in a role that supplies another role affects the supplied one. *Association* is a relationship between elements. *Control* is a relationship between agent roles, meaning that an agent which plays a role that is controlled by other agent role must do everything the controller asks it to do. *Aggregation* is a relationship between objects, between object roles, between agent roles, and between organizations. It has the same meaning in object orientation, *e.g.*. the aggregated element is part of the aggregator element. *Specialization* is a relationship that relates a sub-element to a super-element in a sense that the sub-element can redefine the properties and relationships inherited from the super-element. *Ownership* is a relationship that relates an organization to the agent roles and object roles that are defined in it.

$$
\begin{aligned}
&Inhabit : Citizen\_Name \times Env\_Name\\
&Play : Citizen\_Name \times Role\_Name\\
&Dependency : Roles\_Name \times Roles\_Name\\
&Association : Element\_Name \times Element\_Name\\
&Control : AgRole\_Name \times AgRole\_Name\\
&Aggregation : Aggregated\_Name \times Aggregated\_Name\\
&Specialization : Abstraction\_Name \times Abstraction\_Name\\
&Ownership : Org\_Name \times Roles\_Name
\end{aligned}
$$

Fig. 4. Signature of the relationships

Our formal notation starts with the definition of an *Element* schema in the same way it is defined in TAO. An element is an entity that has properties and relationships but we have omitted its definition for brevity.

According to TAO, an *Environment* is an element that is the habitat for agents, objects, and organizations, which define the set of citizens. The main characteristic of a citizen has to be registered in a specific environment.

Agent and Agent Role structures

Syntactically, an *AgentClass* is a named box (**Fig. 5**) that extends an *Element* and includes a list of inherited *AgentClass* schema names, a list of included *ActAgentClass* schemas, a list of included *PlanClass* schemas and two sets of *AgentRoleClass* names. The inherited *AgentClass* schemas provide support to multiple inheritance. The included *ActAgentClass* and *PlanClass* schemas represent the actions and plans that can be performed by the agent, independently of the role it is playing. The sets of roles indicate the roles the agent can play during its lifecycle (*roles*) and the roles that it must play when the *AgentClass* is instantiated (*init_roles*). Following the way Object-Z was defined, there is an *Init* box inside the *AgentClass* structure, which enforces that when an *AgentClass* is instantiated the agent must be registered in an *Environment* and it must be associated with an initial role. This role must be one of the roles in the set *init_roles*, which means that the set *roles* contains *init_roles*.

An *AgentClass* also has an "axiom part". Separated from the descriptions previously described by a horizontal line, there is a specification of the *Element* extension and a restriction related to the sets of *AgentRoleClass* names. The *Element* extension is specified by the set of properties description as the union of the sets *Belief* and *Goal*, which represent the structural agent properties. In addition, the description of the relationships set is composed of the relationships *Inhabit, Play, Association,* and *Specialization*. The restriction about the sets of roles specifies that the initial roles must be in the set of roles the agent can play during its lifecycle.

As it can be seen, the *AgentClass* structure is quite complex, including in its description other new structures such as *ActAgentClass*, *PlanClass*, and *AgentRoleClass*. We will describe these new structures in the following paragraphs and then illustrate them using an example from the market domain.

AgentClass

Element

list of inherited *AgentClass* schemas

included *ActAgentClass* schemas

included *PlanClass* schemas

$roles, init\_roles : P_1 AgRole\_Name$

Init

Register

$InitRole = OneOf(init\_roles)$

$InitRole \in init\_roles$

$properties : Belief \cup Goal$

$relationships : \left\{ \begin{array}{l} Inhabit, Play, Association, \\ Specialization \end{array} \right\}$

$init\_roles \subseteq roles$

Fig. 5. AgentClass structure in AgentZ

In order to describe the AgentClass construct in more detail we define constructs used in its definition. The *PlanClass* schema is a named box whose name finishes with the keyword *_Plan*, and it includes the set of goals that the plan can achieve and the associated actions. Separated from them by a horizontal line, it includes an axiom part consisting of the sequence of actions that need to be executed in order to achieve the goal(s). Plans are not necessarily defined as ordered sequences of actions. An agent must have at least one plan, and in the case of planner agents, a plan can consist of building a plan to achieve its goals. An example of a *PlanClass* can be found in Section 4.

The *ActAgentClass* schema differs from the *Operation* schema of Object-Z in a significant way: it does not contain a list of affected states, but includes a list of pre-conditions and the result the action must produce. The action result can be a goal achievement, the satisfaction of another action pre-condition or even the maintenance of the initial pre-condition (e.g., in this case the action is not executed successfully). An example of this schema can be found in Section 4.

According to TAO, an agent is always playing a role which affects the agent behavior by defining the protocols the agent must follow in order to interact with other agents, the actions it can execute and the actions it must execute to achieve its goals. We define an *AgentRoleClass* schema as a named box (**Fig. 6**) and its name ends with the keyword *_AgRole*. Following this idea and the MAS-ML metamodel, an *AgentRoleClass* extends an *Element* and includes a list of *ProtocolClass* schemas, a set of *PlanClass* schema names (*plans*), and sets of action names (*duties* and *rights*). The set *duties* contains the actions the agent that play this role must perform and the set *rights* contains the actions the agent can perform. Following the same pattern used in the definition of the *AgentClass* schema, the extension of *Element* is specified by describing the *properties* as the union of the sets *Belief* and *Goal*, and by describing the relationships set as composed of *Control, Dependency, Association, Aggregation*, and *Specialization*. The restriction about the sets *duties* and *rights* is that the former set is contained in the latter.

AgentRoleClass

Element

included *ProtocolClass* schemas

$duties, rights : \mathbb{P}_1\ ActAgent\_Name$

$plans : \mathbb{P}_1 Plan\_Name$

$properties : Belief \cup Goal$

$$relationships : \begin{Bmatrix} Control, Dependency, Association, \\ Aggregation, Specialization \end{Bmatrix}$$

$duties \subseteq rights$

Fig. 6. AgentRoleClass structure in AgentZ

We note that in the *AgentRoleClass* schema there are some protocols schemas. In MAS-ML, protocols define the set of interactions that an agent must perform in order to communicate with other agents. Actually, these interactions are sequences of messages exchanged by agents while playing roles that can be defined as a relation between two sets *Msg_Name*. The definition of the structure of *ProtocolClass* includes a set of *Msg_Name* and a set of interactions.

Agent Organizations

The *OrganizationClass* schema is a named box (**Fig. 7**) whose name ends with the keyword *_Org*. As an organization extends the properties an agent has, its schema includes agent properties and relationships. The extension is obtained via the specification of the organization relationships, a declaration stating that the set of initial roles to be played by an organization is empty, as well as a declaration stating that the content of the set *roles* is composed of the roles that can be played by this organization within the context of another one. In addition, the OrganizationClass schema includes a list of *AgentClass* names. This list specifies the agents that are related to the organizations created from this schema. The *Ownership* relationship and the projection function *second* [14] define the set *roles*. The initial state of an organization is defined by its register in an environment. Moreover, an *OrganizationClass* schema has a set of axioms, which contains the laws that guide the behavior of the agents in the organization.

OrganizationClass

AgentClass
list of *AgentClass* names
$axioms : \mathbb{P}\ Axiom$

Init
Register

$$relationships : \left\{ \begin{array}{l} Inhabit, Ownership, Play, \\ Specialization, Aggregation \end{array} \right\}$$

$$roles = \{ second(org, role) \bullet (org, role) \in Ownership \}$$

$$init\_roles = \{\ \}$$

Fig. 7. OrganizationClass structure in AgentZ

4. Working example: a market place

The example we are considering, which involves a market place, is the same example used in [12]. We are considering a market place where buyers and sellers negotiate the exchanging of products. Sellers advertise their desire to sell products, publishing offers in the market. Buyers access the market in order to buy products. They look for offers that fulfill their needs. Buyers can buy wholesale or retail items. Usually, wholesale items have a lower price per unit. However, sometimes the buyer does not need all the units packaged as one item. Therefore, buyers can form groups to find other buyers interested in the same item. The group of buyers buys the item and distributes the units among the buyers.

Fig. 8 shows an example of the *AgentClass* schema. It is part of the system model that represents the user agent. The user agent of the example can be initialized as a buyer or a seller. The user agent beliefs are *Item*, *RetailOffer*, *WholeSaleOffer*, *Proposal* and *CounterProposal*. The goal of this agent is to deal with items.

An example of the *AgentRoleClass* schema can be seen in **Fig. 9**, where the role buyer, which can be played by the *User_Agent*, is described. The goal of this role is to buy an item and his duty is to look for items. The *rights* that the *User_Agent* has while playing the *Buyer_AgRole* include that one from *duties* added to the rights of accepting or rejecting an offer, receiving the item, and of joining a group to participate in a wholesale. It uses the FIPA Propose protocol and the Deal protocol to interact with the other *User_Agent* playing the roles *Seller_AgRole* or *Mediator_AgRole*, in order to achieve its goal. The definition of which agent role it will interact with is given by the defined relationships. The roles *Mediator_AgRole* and *Member_AgRole* are the ones the agent must choose to participate in a wholesale.

User_Agent

$beliefs : Belief$
$goals : Goal$
$relationships : Relationship$
$roles, init\_roles : \mathbb{P}\ AgRoles\_Name$
$lookForItem\_ActAgent; accept\_ActAgent;$
$reject\_ActAgent; receiveItem\_ActAgent;$
$joinGroup\_ActAgent$
$buyRetail\_Plan; buyWholesale\_Plan$

Init
$Register(self, Supermarket\_Env)$
$InitRole \in init\_roles \bullet (InitRole = Buyer\_AgRole) \vee (InitRole = Seller\_AgRole)$

$beliefs = \{Item, RetailOffer, WholeSaleOffer, Proposal, CounterProposal\}$
$goals = \{deal\_With\_Items\}$
$init\_roles = \{Buyer\_AgRole, Seller\_AgRole\}$
$roles = init\_roles \cup \{Mediator\_AgRole, Member\_AgRole\}$
$relationships = \{Inhabit, Play, Association\}$
$(self, Supermarket\_Env) \in Inhabit$
$\{(self, Buyer\_AgRole)(self, Seller\_AgRole)(self, Mediator\_AgRole)(self, Member\_AgRole)\} \subset Play$

Fig. 8. AgentClass structure example

The agent role *Buyer_AgRole* is owned by an organization called *Supermarket_Org*. This organization can be modeled as described in **Fig. 10**. There are two agents that may play roles inside it (*User_Agent* and *System_Agent*). It is registered in the *Supermarket_Env*, the environment where the organization inhabits. *Supermarket_Org* owns the roles *Seller_AgRole, Buyer_AgRole, Member_AgRole, Mediator_AgRole*, and *Verifier_AgRole*. Moreover, the organization is associated with some objects such as *Item, Offer* and *Proposal.*

We note that in the *User_Agent* class schema, the user agent has some plans and associated actions. In the following we describe the *buyRetail_Plan* (**Fig. 11**), a plan that agents can use to achieve the goal of buying an item being sold through a retail sale. This plan is composed of a sequence of actions, and has *lookForItem_ActAgent* as its initial action **(Fig. 12**).

As our focus is not on how the actions are implemented but on its pre-conditions and results, the *lookForItem_ActAgent* (**Fig. 12**) just specifies that in order to find an item, this item must not have been already found. After the action is executed there are two possible results: either the item was found or the *tryagain* expression was obtained, which means that the pre-condition can still be true after the execution of the action.

Buyer_AgRole

$beliefs : Belief$

$goals : Goal$

$duties, rights : \mathbb{P}\ ActAgent\_Name$

$plans : \mathbb{P}\ Plan\_Name$

$ProposeFIPA\_Protocol; DealWith\_Protocol$

$beliefs = \{Item, RetailOffer, WholeSaleOffer, Proposal, CounterProposal\}$

$goals = \{buyItem\}$

$duties = \{lookForItem\_ActAgent\}$

$$rights = duties \cup \left\{\begin{matrix} accept\_ActAgent, reject\_ActAgent, \\ receiveItem\_ActAgent, joinGroup\_ActAgent \end{matrix}\right\}$$

$plans = \{buyRetail\_Plan; buyWholesale\_Plan\}$

$(self, Seller\_AgRole) \in Association$

$(self, Mediator\_AgRole), (self, Member\_AgRole) \in Aggregation$

Fig. 9. AgentRoleClass structure example

Supermarket_Org

$User\_Agent$

$System\_Agent$

Init

$Register(self, Supermarket\_Env)$

$relationships = \{Inhabit, Ownership, Association\}$

$$roles = \left\{\begin{matrix} Seller\_AgRole, Buyer\_AgRole, Member\_AgRole, \\ Mediator\_AgRole, Verifier\_AgRole \end{matrix}\right\}$$

$(self, Supermarket\_Env) \in Inhabit;$

$$\left\{\begin{matrix} (self, Seller\_AgRole)(self, Buyer\_AgRole)(self, Member\_AgRole) \\ (self, Mediator\_AgRole)(self, Verifier\_AgRole) \end{matrix}\right\} \subset Ownership$$

$$\left\{\begin{matrix} (self, Item)(self, RetailOffer)(self, WholesaleOffer) \\ (self, Proposal)(self, CounterProposal) \end{matrix}\right\} \subset Association$$

Fig. 10. OrganizationClass structure example

$$\begin{array}{|l}
\textit{buyRetail\_Plan} \\
\hline
\textit{goals} : \mathbb{P}\textit{Goal} \\
\textit{actions} : \mathbb{P}\,\textit{ActAgent} \\
\hline
\textit{goals} = \{\textit{buyItem}\} \\
\operatorname{seq} \textit{actions} = \left\langle \begin{array}{l} \textit{lookForItem\_ActAgent}, \textit{accept\_ActAgent}, \\ \textit{payFor\_ActAgent}, \textit{receive\_ActAgent} \end{array} \right\rangle \\
\hline
\end{array}$$

Fig. 11. Example of PlanClass structure

$$\begin{array}{|l}
\textit{lookForItem\_ActAgent} \\
\hline
\textit{item}\,? : \textit{Item} \\
\hline
\textit{pre} \equiv \neg \textit{find}(\textit{item}\,?) \\
\textit{result} \equiv \textit{find}(\textit{item}\,?) \vee \textit{tryagain} \\
\hline
\end{array}$$

Fig. 12. Example of ActAgent Class structure

5. Related Work

There are several research results related to the formal specification of MASs and most of them target specific system features such as agent communication and agent behavior.

Hilaire *et al.* [5] combine Object-Z and statecharts to specify MAS since they understand that each of them, when considered in isolation, lack the expressiveness to specify the complex features associated with MASs. In this sense, we agree that Object-Z does not have enough expressiveness to specify MAS. For instance, instead of combining Object-Z with another existing formalism we have decided to extend it by augmenting it with new structures in order to support the specification of agent-related abstractions.

d´Inverno and Luck [6] defined a formal framework for MAS specification using Z. Their work is general and the formal specification that uses their framework is *ad hoc.* In contrast, our work provides a basis for the formalization of MAS-ML models.

AgentZ is a formal notation that addresses the systematic design of MAS using the specific set of modeling constructs defined in MAS-ML and, for this reason, it can be also used as a rigorous starting point for validation and implementation efforts.

6. Conclusions and Future Work

In this work we have presented the a first version of AgentZ, a formal notation that combines the agent and object-oriented structures proposed in TAO with the formal notations Z and Object-Z, in order to increase their expressiveness by allowing the encapsulation of the complexity associated with both agent and object abstractions. Therefore, by using a notation such as AgentZ, specifications may be shorter and more understandable, and characterize formal design models.

AgentZ was developed to provide a formal notation that allows the verification of MASs design models. In principle, it can be used to validate design properties such as the ones related to the structure, the relationships (e.g., roles, organizations) and the types involved in a specific MAS. In this sense, we believe it should help to improve the quality of the multi-agent system designs.

While a first version of AgentZ was described in this paper, there are many areas that need to be explored to improve this initial version. The semantics of AgentZ must be examined, which includes the definition of the new introduced types. The definition of a formal mapping between AgentZ models and MAS-ML models, which was one of the reasons that motivated us to begin developing AgentZ, will also be part of our future activities. Finally, there is a need of tools for AgentZ support.

7. References

1. Bauer, B. Müller, J.P. and Odell, J. *Agent UML: A Formalism for Specifying Multiagent Software Systems* In: Ciancarini and Wooldridge (Eds) Agent-Oriented Software Engineering, Springer-Verlag, LNCS vol 1957, 2001.
2. Biddle, J; Thomas, E. Role Theory: Concepts and Research. John Wiley and Sons, New York, 1966
3. Carrington, D. and Smith, G. *Extending Z for Object-Oriented Specifications*, 5th Australian Software Engineering Conference, Sydney, May 1990.
4. Duke, R., King, P., Rose, G., Smith, G. *The Object-Z Specification Language: version 1*, Software Verification Research Centre, The University of Queensland, Technical Report 91-01, April 1991.
5. Hilaire, v., Koukam, A., Gruer, P and Müller, J-P. Formal Specification and Prototyping of MAS, In: Omicini, A et al (Eds) ESAW 2000, LNAI 1972, Springer-Verlag, pp 114-127, 2000.
6. d'Inverno, M. and Luck, M.: *Understanding Agent Systems*, Springer Verlag, 2001.
7. Jennings, N. *Agent-Oriented Software Engineering.* In: Proceedings of the 20th Intl. Conf. on Industrial and Engineering Applications of Artificial Intelligence, pp 4-10, 1999.
8. Kim, S-K. and Carrington, D. *A Formal Mapping Between UML Models and Object-Z Specifications*, In. Bowen,J.P. et al (Eds): ZB 2000, LNCS 1878, pp 2-21, Springer Verlag, 2000
9. Lind, J. MASSIVE: Software Engineering for Multiagent Systems, PhD Thesis, university of Saarland, 2000.
10. *MDA – Model Driven Architecture*, http://www.omg.org/mda/
11. Parunak, H. and Odell, J. Representing Social Structures in UML. In: Proceedings of Agent Oriented Software Engineering, pp 1-16, 2001.
12. Silva, V. and Lucena, C. *From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language*, In: Sycara, K., Wooldridge, M. (Eds.), Journal of Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers, 2004. (to be published in March)
13. Silva, V. , Garcia, A., Brandão, A., Chavez, C., Lucena, C., Alencar, P. *Taming Agents and Objects in Software Engineering*, Lecture Notes in Computer Science, vol 2603, 2003.
14. Spivey, J.M. *The Z Notation: a Reference Manual*, Prentice Hall, 2nd edition, 1992. (on-line version at http://spivey.oriel.ox.ac.uk/~mike/zrm/ - 14/05/2003)
15. *UML – The Unified Modeling Language*, http://www.omg.org/uml/
16. Wagner, G. *The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior*, Information Systems, Vol 28, 5, 475 – 504, 2003

17. Wood, M.F. and DeLoach, S.A. *An Overview of the Multiagent Systems Engineering Methodology*, In: Ciancarini and Wooldridge (Eds) Agent-Oriented Software Engineering, Springer-Verlag, LNCS vol 1957, 2001.
18. Wooldridge, M., Jennings, N. and Kinny, David *The Gaia methodology for Agent-Oriented Analysis and Design*, Journal of Autonomous Agents and Multi-Agent Systems, vol 3, pp 285-312, 2000.
19. Wooldridge, M. and Ciancarini, P. *Agent-Oriented Software Engineering: The State of the Art*, In: Ciancarini and Wooldridge (Eds) Agent-Oriented Software Engineering, Springer-Verlag, LNCS vol 1957, 2001.
20. Wooldridge, M. and Ciancarini, P. *Agent-Oriented Software Engineering*, Handbook of Software Engineering & Knowledge Engineering Fundamentals, Chang, S. K. (ed), vol. 1, 2001.
21. Zambonelli, F., Jennings, N. and Wooldridge, M. *Organizational Abstractions for the Analysis and Design of Multi-Agent Systems*, In: Ciancarini and Wooldridge (Eds) Agent-Oriented Software Engineering, Springer-Verlag, LNCS vol 1957, 2001.