Taming Agents and Objects in Software Engineering[1]

Viviane Silva[1] Alessandro Garcia[1] Anarosa Brandão[1]
Christina Chavez[2] Carlos Lucena[1] Paulo Alencar[3]

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

[2] UFBA, Computer Science Department
Av. Ademar de Barros, s/n – 40170-110, Salvador, BA, Brazil
flach@ufba.br

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

Abstract. Agent-based software engineering has been proposed in addition to object-oriented software engineering as a means of mastering the complexity associated with the development of large-scale distributed systems. However, there is still a poor understanding of the interplay between the notions of agents and objects from a software engineering perspective. Moreover, the many facets of agent-based software engineering are rarely used in the various phases of the software development lifecycle because of the lack of a comprehensive framework to provide the software designers with a clear understanding of the use of these two key abstractions. In this context, this paper presents TAO, an evolving innovative conceptual framework based on agent and object abstractions, which are the foundations for modeling large-scale software systems. The conceptual framework allows for the characterization of large-scale software systems as organizations of passive components, the objects, and autonomous components, the agents, with each of these elements playing roles to interact with each other and to coordinate their actions in order to fulfill system goals.

1 Introduction

With the advances in Internet technologies [24, 50, 67], software systems are undergoing a transition from monolithic architectures based on passive components into open and distributed architectures composed of organizations of autonomous components, that operate and move across different environments in order to achieve their goals in a coordinated way [47, 69, 77]. Object-oriented software engineering [5,

[1] Lecture Notes in Computer Science, Springer, Volume 2603, "Software Engineering for Large-Scale Multi-Agent Systems", April 2003.

6, 46, 60] has succeeded to support the development of high-quality software systems, but the complexity raised in this architectural transition is no longer affordable in terms of its abstractions, modeling languages, and methodologies [19, 20, 36, 44, 56, 66, 75]. The limitations of the object paradigm has spurred research on agent-based software engineering [32, 33, 34] as an additional approach to the development of large-scale systems from their conceptual modeling [10, 68, 71] to their computational modeling [18, 21, 55].

While the object abstraction is fundamentally applied to model resources or passive components, the agent abstraction is naturally tailored to represent autonomous components in the software system. The notion of multi-agent systems (MASs) [64] and the underlying theories associated with their many properties bring with them more natural support for autonomy, coordination, mobility, organization, openness, and intelligence. In this context, the discipline of Software Engineering is trying to understand how the lessons learned from the application of these agent theories in Artificial Intelligence can be used to overcome the limitations of object-oriented software engineering and lead to a mastery of the complexity of modern software. The successful and widespread deployment of large-scale MASs requires a unifying set of central abstractions to support modeling languages and respective methodologies for an agent and object-centric software engineering. However, there is still a poor understanding of the interplay between the agent and object notions from a software engineering perspective.

As is the case with any new software engineering paradigm, researchers are beginning to strive to formulate the methodologies that guide the process of constructing MASs [27, 39, 42, 57, 71]. Many, such as Agent UML [54] and MAS-CommonKADS [26], are extensions of previous object-oriented methodologies and languages, while others, such as the AAII methodology [39], are extensions of knowledge engineering methodologies. Existing methodologies propose very distinct and varying sets of abstractions suitable for different domains. Each methodology has incorporated its own abstractions for conceptual and computational modeling, and there is no agreement about a common group of abstractions that can be used across different methodologies. As a consequence, it is very difficult to understand the interplay between agents and objects from a software engineering perspective, and the real contributions of agents in the construction of large-scale systems. The many facets of agent and object-based software engineering are still rarely used in the various phases of the software lifecycle because of the lack of a comprehensive framework [19, 72].

In this context, this paper presents TAO (Taming Agents and Objects conceptual framework), whose goal is to provide the foundations for agent and object-based software engineering. This paper is not a survey of existing concepts used by methods, languages and methodologies for MAS but a definition of an ontology that defines the essential concepts, or abstractions, for developing MASs. The benefit of having a conceptual framework is to provide support for developing new methodologies, methods and languages based on the essential concepts defined and related in the framework. Each concept is viewed as candidate abstraction for modeling languages, methodologies and support environments to be applied in different phases of the MAS development. We classify the abstractions used to establish our ontology into three categories: (i) fundamental abstractions, (ii) grouping abstractions, and (iii) environment abstractions.

TAO can be tailored to different domains since its basic ontology can be extended to accommodate new abstractions for new domains. TAO enables different research teams to compare and discuss their formulations based on the unified terminology of the proposed foundations. The remainder of this paper is organized as follows. Section 2 presents the conceptual framework and a brief view of the abstractions and their relationships. Section 2 also describes example used in the paper in order to illustrate our definitions. A definition of the fundamental, environment and grouping abstractions are presented in Sections 3, 4 and 5 and the relationships between those abstractions are introduced and defined in Section 6. Section 7 provides an overview on how the templates used throughout the paper should be formalized. Section 8 reviews some related work and Section 9 discusses some future directions for research in this area. Finally, Section 10 presents the conclusions of our work.

2 The Conceptual Framework

2.1 The Role of the Conceptual Framework

A conceptual framework is critical for both the problem understanding (conceptual modeling) and the solution proposal (computational modeling) of any development project as software systems become more complex [11]. The purpose of conceptual models is to provide an understanding of the domain describing the problem. Conceptual models describe the problem raised by the user and to be solved by the software system [11]. In order to produce a solution, computational models may be generated based on conceptual models. Computational models are used to describe the form of the software system that solves the problem. A computational model is the definition of a software product, and is the output of requirement specification and design activities [11]. Fig. 1 shows a four-layer picture that illustrates the role of the conceptual framework using the metadata architecture MOF [53] for a simple example. Although computational models do not appear in Fig. 1 these models are generated at the domain model layer.

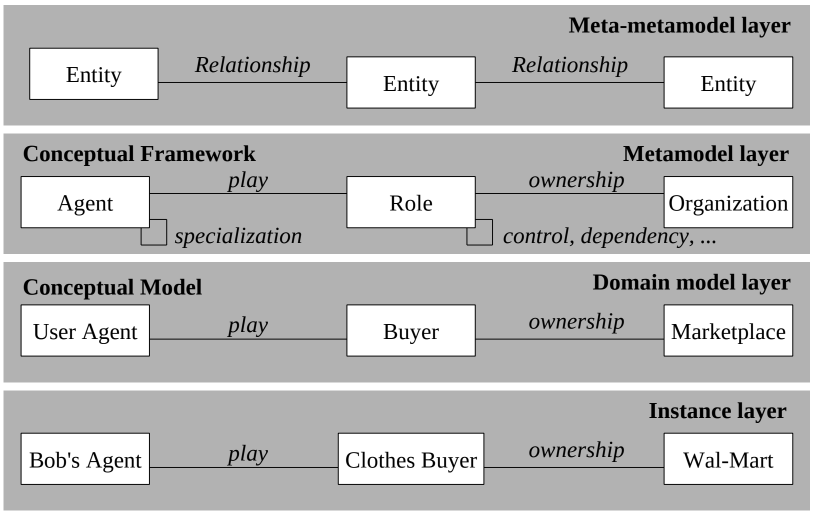

Fig. 1 The role of the conceptual framework using OMG-MOF [53]

The four MOF layers are: meta-metamodel layer, metamodel layer, domain model layer and instance layer. The meta-metamodel layer is comprised of the description of the structure and semantics of meta-metadata. In this paper, we use the ER model (Entity-Relationship model) [13] to describe the entity and relationship meta-metadata that appear in this layer. The entity and relationship meta-metadata provide the basic definition that describes the different entities and relationships instances that appear in the metamodel layer.

Our work is concentrated in the metamodel layer (conceptual framework) which we use to define the conceptual framework TAO developed for the MAS domain. TAO defines the set of metadata, i.e. entity instances and relationship instances that must appear on the metamodel layer level. The framework defines a total of 6 entity instances such as agents, roles and organizations and a total of 8 relationship instances such as play, control and owner (Fig. 1).

The main role of TAO is to provide a unified conceptual framework to understand distinct abstractions and their relationships in order to support the development of large-scale MASs. The proposed framework elicits an ontology that connects consolidated abstractions, such as objects and classes, and “emergent” abstractions, such as agents, roles and organizations, which are the foundations for agent and object-based software engineering. TAO presents the definition of each abstraction as a concept of its ontology, and makes provision for relationships between them.

In contrast with the metamodel layer, which is domain independent, the domain model layer (conceptual model) depicts the data specific to the application domain. The metadata at the metamodel layer are instantiated into data through domain models using the domain information. Entities and relationship instances defined in the metamodel layer are used in the domain model layer according to the domain information defining the conceptual model. The conceptual model is an instance of the conceptual framework based on domain information. The example presented in Fig. 1 shows a partial domain model where the metadata *Agent*, *Role* and *Organization* are instantiated as *User Agent*, *Buyer* and *Marketplace*, respectively and the relationships *play* and *owner* are selected from the set of relationship instances defined in the metamodel layer by the conceptual framework. Recall that those entities and relationships are used according to the domain information.

In order to create domain models, modeling languages, methodologies and methods such as [27, 39, 42, 54, 57, 71] are needed. Modeling languages are used to provide a common language that elicits the meaning of each data element described in the conceptual model. Methods and methodologies are used to guide and help the designer in the creation of domain models.

The instance (information) layer characterizes the possible domain model occurrences. This layer describes the specific instances of the domain model data that may occur during the lifetime of the modeled application. For instance, consider a marketplace domain where buyers and sellers negotiate products. Sellers advertise their desire to sell products, submitting offers to the marketplace. Buyers access the marketplace in order to buy products. They look for offers that match their needs. They can move to another marketplace in order to look for offers that they did not find in the original one. Alternatively, they can form groups to find offers with a lower price per unit. Fig. 1 shows some instances for the marketplace domain: *Bob's Agent* is an instance of a *User Agent*, *Clothes Buyer* is an instance of the *Buyer* role, and *Wal-Mart* is an instance of the *Marketplace* organization.

2.2 The Abstractions and their Categories

TAO classifies the set of abstractions it defines into three distinct categories: (i) *fundamental abstractions* – include the object and agent abstractions, which are the basis for building MASs (Section 3); (ii) *environment abstractions* – include definition of environments and events that are used to represent the environmental constraints and characteristics that influence instances of fundamental and grouping abstractions (Section 3.4); (iii) *grouping abstractions* – encompass abstractions for dealing with more complex situations in large-scale systems; it includes organizations and roles to model complex collaborations (Section 4). Fig. 2 presents the TAO abstractions and their relationships: our proposed conceptual framework (metamodel layer).

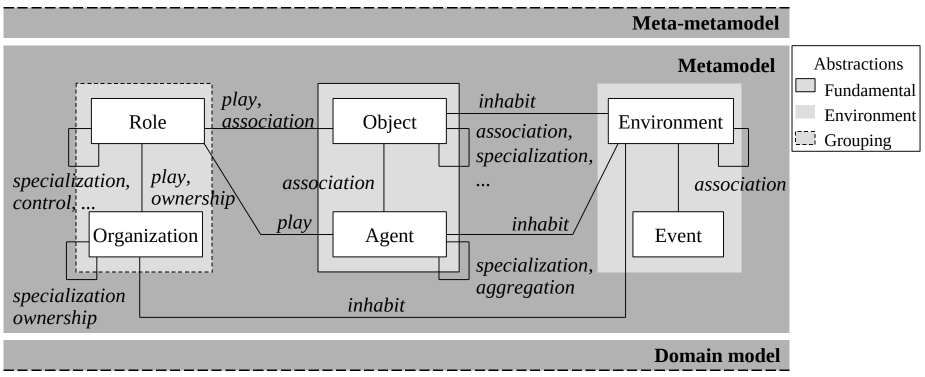

Fig. 2 The abstractions and relationships of the conceptual framework

As shown in Fig. 2, the entity instances defined in the conceptual framework TAO are object, agent, organization, role (agent role and object role), environment and event. Because of similarities among some entities, we have defined a new abstraction called element that is the basis for the definition of most entities.

Element definition:

An element is an entity instance that has properties and relationships with others elements.

Objects, agents, environments, organizations, object roles and agent roles are entities whose definitions are based on the element abstraction. Their definitions extend an element by identifying specific properties and relationships. Event is the only entity in TAO that is not based on the definition of element. It is because events do not have state, behavior or relationships.

The *properties* of an element describe its state and behavior characteristics. The *state* of an element defines information about other elements of the system and the *behavior* of an element defines the actions or operations that the element can perform. An element can change its state and interact with other elements. An element must be related to another element, i.e., a *relationship* must exist between two elements, so that they can interact. The *relationships* link two elements and describe how these elements are related to each other. As illustrated in Fig. 2, different elements are related in different ways, i.e. there are different types of relationships (Section 5). An element *class* defines properties and relationships that are common to all its *instances*. An element instance is a concrete manifestation of an abstraction to which a set of properties and relationships are applied [6]. An element instance of a class fulfills the description of its class.

Besides defining the abstractions, we introduce templates associated with each abstraction in the text. The templates are used to define each abstraction in a schematic way. They list the set of properties and relationships of each element. The templates correspond to the metamodel layer and their instances are related to the domain model layer. In order to exemplify the use of our templates, we have applied them to the marketplace example (Section 2.1) throughout this paper.

2.3 TAO: An Overview

A multi-agent system (MAS) comprises classes and instances of *agents*, *objects* and *organizations*. Organizations group together the agents of a MAS [45, 64]. Agents, organizations, and objects, inhabit (or are immersed in) *environments* [33, 42] that provide *resources* and make available *services*. Resources are non-autonomous entities such as databases or external programs used by agents or organizations. We use objects as an abstraction for modeling resources [45]. While objects represent passive elements, such as resources, agents represent autonomous entities that manipulate objects. Services are the public facilities, or public functions, provided and used by the entities of the system [39]. Entities such as agents, objects, organizations and the environment can make public some of its functions (services) to be used by others entities.

An organization describes a set of *roles* [4] that limits the behavior of its agents, objects and sub-organizations [70]. Agents and objects can be members of different organizations and play different roles in each of them [58]. Every agent of the MAS plays at least one role in an organization. Agents may interact with each other and cooperate either to achieve a common goal, or to achieve their own goals [74]. The agent interactions are based on *relationships* defined between organization roles. An agent may interact with agents from the same organization or from a different one. Two distinct organizations are also related when there are interactions and relationships among their agents.

3 Fundamental Abstractions

This Section describes the fundamental abstractions, object and agent, and the environment abstractions. Each abstraction is described in terms of the following issues: basic definition, state properties, behavior properties, its interplay with other abstractions, its associated template, its application to model the marketplace example. This Section also highlights the most important differences and common features between objects and agents. The types of relationships between abstractions are presented and discussed in Section 5.

3.1 Object

An object is a passive or reactive element that has state and behavior and can be related to other elements.

An object is an element that has a state and a number of operations (behavior) to either examine or change its state [29]. An object extends the definition of an element since it defines state and behavior properties, and relationships with other objects or other elements. The *state* of an object does not have any predefined structure. It stores information about itself, about the environment and about other objects. The *behavior* of an object defines the operations that it is capable of performing. The object *relationships* describe how objects are linked to system's elements, such as other objects, agents, and roles.

An object has control of its state. It performs operations that can modify its state during its lifetime. On the other hand, an object cannot modify its behavior[2] and has no control of it, i.e., an object is not autonomous in the sense it does everything that another one asks it to do. In this way, objects are passive elements that do whatever anyone asks them to do and only when they are asked.

Objects could play roles defined by the organization that uses the object. Because of the importance of the entity role in our conceptual framework, the object role is described in a specific template that is explained in Section 4.2. An object *class* defines the structure and behavior of similar objects [5]. The following template defines the state, behavior and relationships that an object class must have. The object template also lists the events that objects, instance of the object class, can generate and perceive and the roles that objects could play.

```
_________________________Object_________________________

Object_Class Object_Class_Name
       State  setOf{Information}
       Behavior setOf{Operation}
       Relationships setOf{Relationship_Name}
       Events generated: setOf{Event_Name},
              perceived: setOf{Event_Name}
       Roles setOf{Role_Class_Name}
end Object_Class
________________________________________________________
```

The application of the object template to our case study defines the object class `Offer` that characterizes the seller announcements. This class presents the product, the announcer identifier, the basic operations to change the offer price, and the relationships with objects, environment, roles, and the marketplace organization. The object inhabits the *`MarketPlaceEnv`* environment, has a `Product` object associated, can be accessed by *`Buyers`* and *`Sellers`*, and is defined in the context of the *`MarketPlaceOrg`* organization.

```
______________________ Offer ___________________________

Object_Class Offer
       State {price, Product, announcerId, count}
       Behavior {get_price, set_price}
       Relationships {Association_Offer_Product,
              Inhabit_Env_Offer, Association_Offer_Buyer,
              Association_Offer_Seller,
              Association_Offer_MarketPlaceOrg}
       Events generated: Announcement_Published
       Roles Announcement
End Object_Class
________________________________________________________
```

[2] Computational reflection has been introduced in the object paradigm [43] to support the dynamic adaptation of the behavior of object-oriented systems at run-time. However, it is not a property of the objects themselves; it is an extension of the object paradigm.

3.2 Agent

An agent is an autonomous, adaptive and interactive element that has a mental state.

Software agents are complex objects with an attitude [7]; that is, they are elements that extend objects with structured state and agency behavioral properties. According to [25, 34, 52], agents are interactive, autonomous and adaptive elements, and those are the three fundamental characteristics that define the agent property. Characteristics such as rationality, learning ability and mobility are additional characteristics that are neither necessary nor sufficient for the characterization of a software agent.

The *state* of an agent is expressed through mental components such as beliefs, goals, plans and actions [34, 52, 64]. The set of beliefs, goals, plans and actions is called the mental state of the agent. During the lifetime of an agent, its mental state can change, i.e., the agent can change its beliefs, goals, plans and actions.

An agent has beliefs or knowledge about the world, about itself and about other agents. The beliefs include what the agent knows, what the agent views, its memories and its perceptions about everything that happens in the MAS. The agent's goals consist of future states, or desires, which agents would like to reach, or satisfy. As agents are goal-oriented an agent within a MAS must have at least one goal to be achieved. An agent achieves a goal by executing a plan, which can be selected from a list of plans.

Plans define a sequence of actions that is executed by an agent to achieve goals. An agent updates its mental state, changes its roles, perceives and generates *events*, and sends and receives messages while executing actions. Actions have a set of pre and post-conditions. An action is executed if its pre-conditions are satisfied according to the beliefs of the agent. After the execution of the action, the agent checks the post-conditions according to its beliefs. If the post-conditions are satisfied the correctness of the action is guaranteed.

The *behavior* of an agent is expressed through its plans and actions that are based on its agency characteristics, e.g., interaction, autonomy and adaptation. Agents are interactive because they have the ability to interact with other elements when playing roles in an organization. Agents interact with others since they have relationships with other system elements. The *relationships* describe how an agent is linked to another *element*. For example, a relationship describes the roles that an agent plays and the environment that it inhabits. The types of relationships that an agent may have are defined in Section 5.

The autonomy characteristic refers to the proactive capacity of an agent — the agent does not need external stimulus (e.g., user events) in order to carry out a given task. Agents are adaptive elements since they can adapt their state and their behavior by responding to messages sent by the environment or other agents. By assuming a given situation the agent may simply react, or it may reflect upon what should be done.

The agent template defines an agent *class*. An agent *class* describes beliefs, goals, actions, plans and relationships that are the same for all its agent instances. The agent template also lists the events that agents, instances of the agent class, can generate and perceive, and the roles that agents could play. A specific template for agent roles was created and is explained in Section 4.2.

```
______________________Agent_________________________

Agent_Class Agent_Class_Name
      Beliefs setOf{Belief_Name}
      Goals setOf{Goal_Name}
      Actions setOf{Action_Name}
      Plans setOf{Plan_Name}
      Events generated: setOf{Event_Name},
            perceived: setOf{Event_Name}
      Roles setOf{Role_Class_Name}
      Relationships setOf{Relationship_Name}
end Agent_Class
____________________________________________________
```

We used the agent template in our case study to define an agent called `User_Agent` that represents the users in the system. The agent `User_Agent` knows what is a simple offer, a group offer and the environment it inhabits. Its goal is to deal with products and it can perform actions to deal with sellers and buyers and to request authorization to enter another marketplace. Besides that, it also has strategies for buying and selling products. It plays the roles of `Buyer` and `Seller`. It also plays the role of `Mediator` if it is coordinating a group in order to buy products.

```
_______________________User_Agent___________________________

Agent_Class User_Agent
    Beliefs {Offer, Offer_for_Group, MarketPlaceEnv}
    Goals {Dealing_products}
    Actions {ask_for_Entering_Authorization,
              deal_with_Seller, deal_with_Buyer}
    Plans {buying_Strategy1, buying_Strategy2,
            selling_Strategy}
    Events generated: Group_Forming
            perceived: Announcement_Published
    Roles Buyer, Seller, Mediator
    Relationships {Inhabit_Env_User_Agent, Play_Buyer,
    Play_Seller, Play_Mediator}
end Agent_Class
____________________________________________________________
```

3.3 Agent versus Object

We define an agent as an extension of an object, i.e. an agent is an object with additional features, because it extends the definition of state and behavior associated with objects. The state of an agent has a “mentalistic” structure [68], as we have already seen. The agent’s mental state extends the definition of state defined for objects because it adds to its state the definition of its behavior. So the state of an agent, its mental state, consists of also its state and its behavior. The mental state consists of: beliefs, that are equivalent to the object’s state, and also goals, plans and actions, that define the agent’s behavior. Moreover, the behavior of an agent extends the behavior of objects because: (i) an agent has full control of its behavior, i.e., agents can say no to the requests of other agents, (ii) agents can change their behavior, adding new actions to be executed [64], and (iii) agents do not require external stimuli to carry out their jobs. These three extensions make an agent an active element and an object a passive element.

Another difference between agents and objects is related to the agency characteristics. As we have already seen, an agent is an autonomous and interactive element that sends and receives messages. The autonomy and interactivity of an agent can vary from a completely reactive agent that interacts frequently with other agents to a completely proactive agent that may not need to interact with anyone to achieve its goals. In the case where the agent is a completely reactive element it may be considered an object. The more autonomous an agent is the less interactive it needs to be. A proactive agent does not need to cooperate with anyone to achieve its goal. On the other hand, the less autonomous an agent is the more interactive it needs to be to achieve its goals.

An object is an interactive element but it is not an autonomous one. Objects are reactive and passive elements since they need the assistance of another element to do their job and since they respond to any request for assistance. Classically, an object interacts a lot with other objects in order to complete their jobs. From the point of view of the autonomy and interaction characteristic, there is no difference between a reactive agent and an object. They both need a request from another element to do their jobs and they both need to interact with other elements in order to do their jobs.

4 Environment Abstractions

An environment is an element that is the habitat for agents, objects and organizations. An environment can be heterogeneous, dynamic, open, distributed and unpredictable [57].

An environment extends the definition of an element since it defines its properties, i.e., its state and its behavior, and its relationships. The *state* of an environment stores the lists of resources and services and associated access permissions. Resources are non-autonomous entities such as databases or external programs used by agents or organizations. We use objects as an abstraction for modeling resources [45]. The resources can be used by objects, by agents or by organizations when playing roles.

The permissions associated with resources restrict the access of objects and agents to them. Agents, objects and organizations make services available to other agents and organizations. The permissions associated with the services restrict the access of the agents and organizations.

The *behavior* of an environment is defined based on its characteristics. An environment can be heterogeneous, dynamic, open, distributed and unpredictable [57]. An environment can be a passive element such as an object or can be an active element such as an agent having agency characteristics such as autonomy, adaptation and interaction. An environment can also have other agency characteristics such as mobility. A palmtop may be considered a mobile environment. Thus, an environment may be seen as an agent when appropriate.

An environment is the habitat of MAS organizations, agents and objects. An organization may inhabit multiple environments but an agent and an object must inhabit only one environment at a given moment. An agent or an object must not be in two or more environments at any given moment. Different environments can be the habitat of different elements and can have different characteristics, resources and services. The *relationships* of an environment describe which elements inhabit it and which other environments are associated with it. These relationships are extensively described in Section 5.

The environment template presents an environment *class*. An environment class defines its state as a set of resources and a set of services, the behavior of its instances as a set of its properties and a set of relationships that are common to all environment instances.

```
___________________________Environment___________________________

Environment_Class Environment_Class_Name
  Resources setOf{<Resource, Permision, Element_Class_Name>}
  Services setOf{<Service, Permission, Element_Class_Name>}
  Behavior setOf{Properties}
  Relationships setOf{Relationship_Name}
  Events generated: setOf{Event_Name},
         perceived: setOf{Event_Name}
end Environment_Class
_________________________________________________________________
```

The environment template was used to define our case study environment. The `MarketPlaceEnv` is the habitat of agents, objects, and organizations. It provides services and resources to deal with products and it is open and heterogeneous.

```
______________________MarketPlaceEnv___________________________

Environment_Class MarketPlaceEnv
   Resources {<Offer, read, Seller>, <Offer, read, Buyer>,
            <Offer, write, Seller>, <Offer, write, Buyer>}
  Services {buy_Service, sell_Service, submit_Service}
  Behavior {Open, Heterogeneous}
  Relationships {Inhabit_Env_User_Agent, Inhabit_Env_Offer
              Inhabit_Env_MarketPlaceOrg, ...}
  Events perceived: Group_Forming
```

```
end Environment_Class
```

Events are generated in different ways. They can be generated by objects through the execution of their operations, by agents through the execution of their actions and by environment when the environment is an active element [68]. An event generated by the environment, by an agent or by an object can trigger the execution of actions associated with agents or operations associated with objects that perceive the event. As a consequence, events are related to actions and operations that generate them and to actions and operations that perceive them.

5 Grouping Abstractions

MAS comprises a set of grouped agents immersed in one or more environments whose global behavior is derived from the interaction among the constituent agents [74]. The group of agents comprising the MAS defines organizations whose goals are the same as the MAS. A MAS has at least one organization that represents the system and that groups all agents. The organization's agents exist to achieve the goals of the MAS. The agents have individual goals that, when they are grouped together, characterize the goals of the MAS.

5.1 Organization

An organization is an element that groups agents, which play roles and have common goals. An organization hides intra-characteristics, properties and behaviors represented by agents inside it.

Besides the organizations defined by the MAS, the MAS can have other organizations that are sub-organizations. Recursively, each sub-organization can have others sub-organizations defined within it.

From the perspective of elements outside of an organization, the organization can be viewed as an agent. An organization hides intra-characteristics, properties and behavior represented by agents inside it. However, an organization extends the properties and relationships defined by agents. An organization defines a set of rules and laws that agents and sub-organizations must obey. The rules and laws characterize the global constraints of the organization. An organization also defines roles that must be played by the agents and sub-organizations within it. Since all organizations define roles, rules and laws, any agent and any sub-organization is always playing at least one role and respecting the rules and the laws defined by the MAS organizations.

The *state* of an organization is represented by the state of the agents that play roles in it and by the rules and laws defined in the organization. An organization's *behavior* is based on the behavior of the agents that play roles in this organization. The behavior of an organization typically is more complex than the sum of the behaviors of the agents playing roles. The *relationships* describe how an organization is linked to another *element*. For example, the roles defined by an organization are linked to the organization through the relationship owner, describing that the organization is the owner of the roles. Another example is the association between two organizations characterizing that they will exchange messages. One may observe that interactions between an organization and another element in fact occur between an agent inside the organization and the element.

The organization template presents an organization class that describes the rules and laws as well as the relationships associated with all instances. In relationships we can find the roles that an organization owns, the environment that organization inhabits, and the resources and services that it provides.

```
______________________Organization_____________________

Organization_Class Organization_Class_Name
      Rules setOf{Rule}
      Laws setOf{Law}
      Relationships setOf{Relationship_Name}
end Organization_Class
________________________________________________________
```

We have defined the organization for our case study by using the organization template. The organization *MarketPlaceOrg* has rules and laws guiding the behavior of agents playing roles and agents within it, like *Mediator_creates_Buying_Groups_and_just_Him* and *Counter_Proposal _has_NMAX*, and it has relationships like *Inhabit_Env_MarketPlaceOrg*, which fix that it inhabits some environment, others like *Owner_Mktp_Role_Buyer* and *Owner_Mktp_Buying_Group_Org*, which fix that it owns some role and some sub-organization.

```
_________________________MarketPlaceOrg____________________________

  Organization_Class MarketPlaceOrg
     Rules {Counter_Proposal_has_NMAX,
            Buiyng_Group_has_MAX_Buyers,
            Verifier_authorizes_only_Buyers_to_Enter}
     Laws {Mediator_creates_Buying_Groups_and_just_Him,
          Everybody_uses_FIPA_ACL_Protocol_to_comunicate}
     Relationships {Owner_Mktp_Buying_Group_Org,
          Owner_Mktp_Role_Buyer, Owner_Mktp_Role_Seller,
          Owner_Mktp_Role_Mediator, Owner_Mktp_Role_Verifier,
          Inhabit_Env_MarketPlaceOrg,
          Association_Offer_MarketPlaceOrg, ...}
  end Organization_Class
____________________________________________________________________
```

5.2 Roles

Defined in the context of an organization, a role is an element that guides and restricts the behavior of an agent or an object in the organization. The social behavior of an agent is represented by its role in an organization.

The two most important properties of roles are (i) a role is always defined in the context of an organization and (ii) a role must be played by an agent, by an object or by a sub-organization. A role is an element since it defines a set of properties and relationships.

The *state* and *behavior* of an object role, similar to what is defined for objects, keep information and operations, respectively. An object role may add information to the state of the object and may restrict access to the object state. An object role also guides and restricts the behavior of an object. On one hand, the object role can add behavior and relationships to the object that plays the role and, on the other hand, can restrict the access to the object [41].

The *relationships* of an object role describe additional relationships and types of relationships that were not previously available to objects. For example, an object role may add an association to another element that was not defined in the object.

From the point of view of the element that is related to the object that is playing a role, the role identifies the properties that the element can see and identifies the available relationships. The object role template defines the states, behaviors and relationships available to the object that plays the roles and to other elements related to it. The object role template presents the role class, and all role instances of the role *class* have the same states, behaviors and relationships.

```
____________________Object_Role________________________

Object_Role_Class Object_Role_Class_Name
       State  setOf{Information}
       Behavior setOf{Operation}
       Relationships setOf{Relationship_Name}
end Object_Role_Class
_______________________________________________________
```

An agent role guides and restricts the behavior of an agent because associated with the role are goals, beliefs, duties, rights, protocols and commitments that an agent has while playing the role. An agent role is an element since it has state, behavior and relationships with other elements. The *state* of an agent role is defined by its beliefs and goals. The beliefs of the roles are related to the organization's facts, e.g., information about the other roles and information about the objects available in the organization. The goals of the roles characterize the goals that the agent must achieve while playing the role. The goals of the roles grouped together form the organization's goals.

The duties, rights, protocols and commitments define the *behavior* of an agent role. The duties of the roles describe the responsibilities [70] of the organization's agents. The duties define actions assigned to the agent playing the role. We will generalize and describe a duty as a set of actions. Besides the rules and laws described in the organization, the rights associated with each role describe the permissions on the resources and services available in the environment and about the behavior of the agents. The portion of the environment that an agent can sense and effect is determined by the agent's specific role. Normally, each agent has a partial notion of the whole system [35, 57] and none of the agents have sufficient competence, resources or information to solve the whole problem [30].

The protocols and commitments define the interactions between roles and other elements. Protocols define a set of interactions and rules that the elements playing the role and following the protocol must obey. A commitment defines a set of actions that an element playing a role must carry out in relation to other roles.

The definition of the *relationships* of an agent role is based on the protocols and commitments associated with the role. In this way, the agent role adds a set of relations to the agent that plays the role.

The agent role template presents the agent role *class* and the goals, beliefs, duties, rights, protocols and commitments that define the interactions. It also identifies the relationships of the agent roles, i.e., its owner, the agents and organizations that may play the role, the objects associated with the role, and the associations between the roles. All role instances of the role class have the same properties and relationships.

```
____________________________Agent_Role_______________________________

  Agent_Role_Class Agent_Role_Class_Name
       Goals setOf{Goal_Name}
       Beliefs setOf{Belief_Name}
       Duties setOf{Action_Name}
       Rights setOf{Permission_Name}U setOf{Action_Name}
       Protocols setOf{Interaction_Class_Name}U setOf{Rule_Name}
       Commitments setOf{Action_Name}
       Relationships setOf{Relationship_Name}
  end Agent_Role_Class
  ____________________________________________________________________
```

The agent role template was used to define the role buyer. The role *Buyer* is defined by the organization *MarketPlaceOrg*. Agents playing this role can deal with agents that play roles of *Seller*, *Mediator* and *Verifier*, when they are buying products from *Seller*, or asking to participate in a *Buying_Group*, or asking permission to enter another marketplace, respectively. They also have some duties, such as *buy_Product*, that can generate a commitment, like *pay_for_Product*, and some rights, like *accepting_Offer*. Moreover, their interactions must follow some protocols, such as the *FIPA_Protocol* [14].

```
____________________________Buyer____________________________________

  Agent_Role_Class Buyer
     Goals {buy_products}
     Beliefs {Offer, Product}
     Duties {submit_Offer, analyse_Offer, buy_Product,
              pay_for_Product}
     Rights {making_Counter_Proposal, rejecting_Offer,
              accepting_Offer, receiving_Product}
```

```
    Protocols {FIPA_Protocol}
    Commitments {pay_for_Product}
    Relationships {Association_Buyer_Seller,
        Association_Buyer_Mediator,Association_Buyer_Verifier,
        Association_Offer_Buyer, Owner_Mktp_Role_Buyer}
  end Agent_Role_Class
```

6 Relationships

This Section presents the relationships between all elements of the conceptual framework. There are eight different relationships classified in two different ways that associate objects, agents, environments, organizations and roles where some are basic and domain-independent relationships and other are domain-dependent relationships. The basic relationships are *play*, *owner* and *inhabit*. These relationships will always appear in any conceptual model since they do not depend on the problem domain.

6.1 Relationship Types

Let A be a set of agents, $a \in A$, E be a set of environments, $e \in E$ and O be a set of objects, $o \in O$. Let Org be a set of organizations, org, subOrg $\in$ Org and subOrg always represents a sub-organization. Let R be a set of roles, R = RObj ? RAg where RObj is a set of object roles and RAg is a set of agent roles, $r \in R$, ro $\in$ Robj and ra $\in$ RAg. For each one relationship presented below, we present its definition, its classification and the elements it links.

- Inhabit (I) : I(habitat, citizen) : I(e,a), I(e,o), I(e,org)

Some elements must inhabit environments and can dynamically change from a habitat, i.e. an environment, to another. The inhabit relationship specifies that the element that inhabits - the citizen - may leave and enter habitats, respecting the habitat permissions. Normally, the habitat does not guide the actions of its citizens and does not impose constraints on when to enter or leave the habitat or what specific actions they must carry out. On the other hand, the habitat restricts which elements can enter, which resources and services they can access and which services they can provide. When a citizen changes its habitat it is no longer subordinated to its old habitat.

When inhabiting environments, agents, objects and MAS organizations must respect the permissions that have been defined by them. Agents and objects inhabit only one environment at a given time, as opposed to organizations, which can inhabit more than one environment at the same time.

- Ownership (Ow) : Ow(owner, member) : Ow(org, r), Ow(org, subOrg)

Some elements must be members of another element. The ownership specifies that an element - the member - is defined in the scope of another element - the owner - and that a member must obey a set of global constraints defined by its owner. Members may be dynamically created or destroyed by their owner.

Organizations are owners of roles and sub-organizations. Each role and sub-organization has one owner organization. Agents or sub-organizations in an organization play roles as defined by their enclosing organization.

- Play (P): P(element, role) : P(a,ra), P(subOrg,ra), P(o,ro)

Objects, agents and sub-organizations must play roles. The play relationship defines that the object, agent or sub-organization that plays the role assumes properties and relationships defined by the role. The behavior of the object, agent or sub-organization is guided by and restricted to the scope of the role.

- Specialization/Inheritance (S) : S(super-element, sub-element) : S(o,o), S(a,a), S(org,org), S(ro,ro), S(ra,ra)

The specialization relationship defines that the sub-element that specializes the super-element may add and redefine the properties and behavior associated with the super-element.

- Control (C)[3] : C(controller, controlled) : C(ro,ro), C(ra,ra), C(ra,ro)

The control relationship defines that the controlled element that plays the role must do anything that the controller element asks it to do. An agent role may control another agent role or an object role. Object roles only can control another object role. An agent playing a role that controls another role played by another agent is related to the other agent by an undirected relationship of control.

- Dependency (D) : D(client, supplier) : D(ro,ro), D(ra,ra), D(ra,ro)

An element - the client - may be defined to be dependent on another one - the supplier - to do its job. The dependency relationship specifies that the client cannot completely do its job unless it asks the supplier. The client changes its behavior according to the supplier but the opposite is not true. The client does not influence its supplier. An agent role may depend on another agent role or on an object role. Object roles can only depend on another object role.

- Association (As) : As(associate1, associate2) : As(r,r), As(e,e), As(o,o), As(a,o), As(o,org)

If an element is associated with another element, it knows that the other element exists. The association relationship must define how an element interacts with another one. Roles may be directly associated with other roles as well as environments.

- Aggregation/Composition (Agg) : Agg(aggregator, part) : Agg(ro,ro), Agg(ra,ra), Agg(o,o), Agg(a,a)

If an element is aggregated with other element, we say that it is part of an aggregator. The aggregator 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 knows

[3] We are extending the relationships *control* and *dependency* described in [74].

each of its parts. Depending on the strength of the aggregation, the part may not exist without the aggregator.

The relationship template is used to define the links between the elements. For each relationship type, the template identifies the elements and its roles in the relationship.

```
___________________________Relationship______________________________

Relationship Relationship_Name
      INHABIT: habitat, citizen
    | OWNERSHIP: owner, member
    | SPECIALIZATION : super-element, sub-element,
    | PLAY: element, role
    | CONTROL: controller, controlled
    | DEPENDENCY: client, supplier
    | ASSOCIATION: associate1, associate2
    | AGGREGATION: aggregator, part
end Relationship
_____________________________________________________________________
```

As an example, we describe in this paper two kinds of relationships. Below we have an instance of the relationship template ownership that links the organization *MarketPlaceOrg* and the agent *User_Agent* and, net, an instance of the relationship template play linking the *User_agent* and the role *Buyer*.

```
______________ Owner_Mktp_Role_Buyer _________________

 Relationship Owner_Mktp_Role_Buyer
       OWNERSHIP: MarketPlaceOrg, Buyer
 end Relationship
_____________________________________________________

___________________Play_Buyer ________________________

 Relationship Play_Buyer
       PLAY: User_Agent, Buyer
 end Relationship
_____________________________________________________
```

7 Semantics of the Template-Based Representations

As we have stated previously, the main goal of this paper is to characterize a key set of abstractions that can be used to define a conceptual framework using agents and objects, and clarify the interplay between the agent and object abstractions. For this reason, we have focused on the choice and presentation of these concepts in an informal way rather than on formally defining the semantics of each of the templates. In addition, presenting these concepts using a formal notation would certainly lead us to representations that would be less legible because of their mathematical nature than the light template-based representation that we have adopted.

However, the formal semantics of the schemas and its parts is certainly a necessary result that will be provided as soon as we finish the process of refining our templates to our satisfaction. Even though we do not aim at giving a completely formal semantics for our templates in this paper, in this section we will provide an overview on how they should be formalized to indicate possible formalization choices.

Objects can be formalized in (temporal) logic following the theory we have presented in [1]. Each element class (e.g., an object or agent class) has a specific name and can be defined as a set of possible instances. For example, the names of object and agent element classes in the templates are Object_Class_Name and Agent_Class_Name, respectively. If the instances are objects, the class is called object class; if the instances are agents, the class is called an agent class, and so on. Given an element class C, ci e C (i =1,…,n) denotes an element of this class. Every instance of an element class has an associated single object identity (or identifier). @C represents the set of possible (element) identifiers of class C.

The formal specification of the object class template can be given as a tuple <DT,AT,AC> in the abstract data type style and a set AX of axioms (or properties), where DT is a data signature (i.e., it defines sorts such as Boolean and function symbols f(x1,…,xn) related to these sorts), AT is a set of attribute symbols a(x1,…,xn) that we call the state of an object, and AC is a set of action symbols g(x1,…,xn). The behavior is defined in our template as a set of properties AX that we call operations in our template. The relationships can be defined as a set of relations Rel: C1 X C2 X … X Cn, where C1, C2, …, Cn are element classes.

In order to formalize the agent class template, we need to formalize beliefs, goals, actions, plans, events, roles, and relationships. Relationships are also relations as in the case of objects. Beliefs are facts and rules that may be expressed in first-order logic (FOL) or Prolog. The goals and actions may be expressed in temporal logic with processes (e.g., mu-calculus) used, for example, in [12]. Events can be seen as atomic processes. Roles can be seen as a special relationship. A plan may be described as a sequence <ac1, ac2, …, can> of actions that should be executed to achieve a certain goal.

Environments were defined in terms of the set of resources (including the permission and the name related to each resource), a set of services (including the permission and name related to each service), a set of properties that describe its behavior, a set of events, and a set of relationships of this environment with other entities. Each resource and service has an identity. Permissions can be characterized as Prolog or temporal logic constraints. Each event can be seen as an atomic process. Services can also be defined using a process style following [12].

Organizations are defined in terms of a set of rules to which they conform, a set of laws they obey, and a set of relationships with other organizations. The rules can be defined in first-order logic or Prolog. The laws can be defined as constraints in FOL or temporal logic, and the relationships can be defined as mathematical relations. Agent and object roles were also defined. Roles for objects can be formally defined by a specific relation in a way similar to the way views were defined in [1]. In order to define roles for agents, we have to introduce extra attributes in the agent descriptions. However, the formalization may also be based on logic and process.

8 Related Work

A lot of work has been done in the area of developing models, methodologies, methods and languages [8, 27, 39, 42, 54, 57, 71] to help and guide software engineers to create conceptual models of MASs. However, not much has yet been done in the area of conceptual frameworks for MASs.

TAO as well as the work discussed later in this section [10, 28, 68, 73] is focused on the development of a new conceptual framework to explain the interplay among MAS abstractions. As indicated in Fig. 3, TAO and its related research work discussed next addresses the metamodel layer illustrated in the 4 layer metadata architecture proposed by MOF. We did not compare our work to well known methods and methodologies since they belong to the domain model layer of MOF as shown in Fig. 3. In other words conceptual frameworks and methodologies belong to different levels of abstraction.

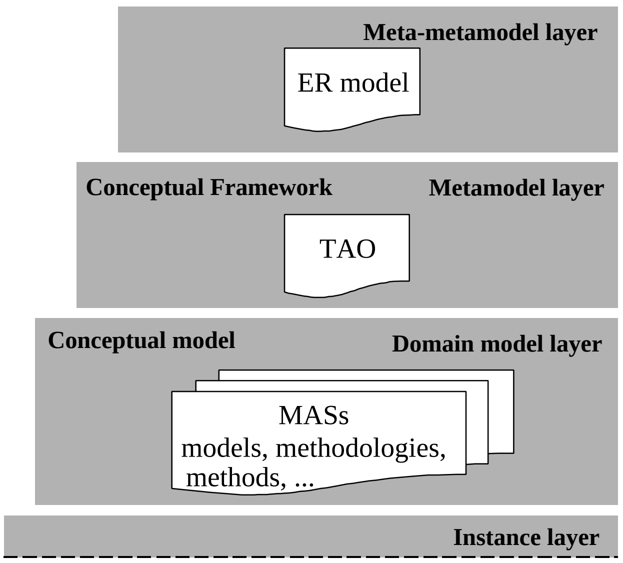

Fig. 3 Locating TAO in the MOF Metadata architecture.

Wagner [68] presents a conceptual framework of agent-oriented modeling restricted to model organizational information systems. Thus, it is not generally applicable to MAS and, consequently, it must be carefully used in the case of other types of systems. Although the proposed framework integrates agents and objects, it does not include important concepts such as actions, goals, organizations and, thus, roles as first class elements. Organizations are defined as institutional agents that are composed of rights and rules. The relationships between the institutional agent and simple agents are not defined. Moreover, the framework does not deal with pro-active agents.

KAOS is a conceptual framework that defines abstractions, such as entity, relationship and agent, as extensions of object [10]. An entity is an autonomous object that is independent of other objects; a relationship is a subordinate object; and an agent is an object that has choice and behavior. In this way, KAOS does not satisfactorily explain the distinction between an entity and an agent and why a relationship should be an object. It does not describe the characteristics of an object or explain how other abstractions extend it. Two other weaknesses of KAOS are: (i) it does not consider organizations and roles as important abstractions, and (ii) it does not describe the relationships between the defined abstractions.

d'Inverno and Luck [28] define a conceptual framework with four important limitations: (i) it does not define all possible relationships between its elements; (ii) it does not define organization and role; (iii) it defines new concepts like server agents, autonomous agents and neutral objects increasing the complexity of understanding the relationship between agents and objects; and (iv) their approach is so generic that it may be very difficult to be used by software engineers and methodology developers.

Finally, Yu and Schmid [73] define a conceptual framework for agent-oriented and role-based modeling. However, it does not define abstractions such as objects, object roles and organizations and, therefore, it does not connect these abstractions with the definitions of agent and role.

9 Discussions and Ongoing Work

Our research group [17, 65] has been conducting a set of empirical studies [15, 49, 62] for a number of years. These studies have generated a set of questions about the use of objects and agents in modeling and implementing systems [16, 18]. After exhaustive review of theories, methodologies and methods for multi-agent systems, we found that our questions have not been addressed yet. We felt the need for a conceptual framework that must completely define the abstractions and their relationships. TAO has three important goals: (i) to explain the relationships between objects and agents; (ii) to unify six main abstractions commonly used to model MASs; and (iii) to define the relationships between those abstractions.

The core set of abstractions used in TAO has been developed based upon our investigation of existing agent-based and object-oriented methodologies [13, 42, 45, 68, 71, 73], languages [40, 48, 63], and theories [59, 64, 68]. Our conceptual framework intends to explain how to use this set of abstractions, defining it and introducing a comparison between objects and agents and how they are related. For that we present a list of well-defined relationships. Furthermore, our conceptual framework is intended to be extensible so that new abstractions and relationships can be grouped together with the existing ones. For instance, software components are natural candidates to be included in the framework.

The benefit of having a conceptual framework for a family of problems is to provide support for developing new methodologies, methods and languages based on the essential concepts defined and related in the framework. Although some methodologies are agent-centered methodologies and do not consider objects as an abstraction, these methodologies also can be based on our conceptual framework. Our framework defines an object as an abstraction but it does not insist that this definition must be used.

Nonetheless, it should be noted that although our agents' template defines the internal architecture of an agent in terms of plans, goals, beliefs and actions, the users of our framework can map these concepts to other internal architecture styles, such as the BDI architecture [38]. Different architectures can utilize our framework, changing the templates and internal definitions of our set of abstractions. Another example occurs with the roles template, which may be completely different than the one presented here.

The conceptual framework was defined to be used to generate conceptual models. Abstractions defined in the conceptual framework are instantiated in the conceptual models. And thus, conceptual models are used to generate computational models. In this way, abstractions used in conceptual models may be mapped to other abstractions used in computational models. We are working on the creation of transformations for the set of abstractions defined in our conceptual framework to computational models. We are also concerned with possible adequate representations of both models.

Another work under way is related to the non-functional requirements. We believe that some non-functional requirements (such as reliability, security, ...) will be common to several abstractions within an application. In this sense, we are investigating how to allow abstractions to support an explicit separation of such crosscutting behavior. The notion of aspect [22, 37, 66] is well understood in the object-oriented context, but only a few preliminary works have been published that discuss it in terms of agent-based software engineering (such as [16, 20]).

Related to the MAS dynamic, we intend to study the dynamic of organizations. We will seek to improve reporting about the definition of commitments, protocols, rights, laws and actions. Some questions remain to be answered: How do agents enter and exit organizations? Why do they enter and why do they exit? How do organizations or agents define an organization's set of rules and laws? How do agents that enter an organization learn about and start to obey its conditions? Little work in this direction has been carried out [23].

10 Conclusion

Object-oriented software engineering and its associated theories have already proven to be effective for the development of software systems. Object-oriented theories and respective languages and methodologies have shown how suitably powerful abstractions, like the notions of object and class, can be fully exploited not only to define modeling languages, but also to support methodologies that drive all the phases of the engineering of software systems [56]. However, the advances in networking technologies and the coming of the Internet era are leading towards issues that traditional object-oriented software engineering is not ready to address. Large-scale software systems are now entrusted with typically complex tasks, which can involve massive amounts of passive components as well as autonomous components. These components affect numerous kinds of connected environments, and are subject to the uncertainties of open environments such as the Internet [56]. The inadequacy of object-oriented approaches does not derive from the methodologies, but rather from limitations of the object theories and their abstractions themselves, which are not powerful enough to examine these new issues. To cope with this situation, companies and researchers are investigating how agents can contribute to the mastering of the complexity of modern large-scale systems.

This paper presented a conceptual framework that provides a conceptual setting for engineering large-scale MASs based on agent and object abstractions. The identified set of abstractions is organized in terms of a unifying framework, providing software engineers with a deeper understanding of the fundamental concepts underpinning agent and object notions and their relationships. Objects are viewed as abstractions to represent passive elements, while agents provide a means of representing active elements in the software system. In addition, a set of additional abstractions is provided to model situations where organizations of cooperating agents and objects perform and coordinate their actions in dynamic environments to accomplish the organizations' goals. The core set of abstractions was developed based on our extensive work in investigating existing agent-based and object-oriented methodologies, languages and theories, and our extended experimental work on developing many large scale MASs. As a result, it can be tailored to different domains. Since its basic ontology can be extended to accommodate new abstractions for these domains, it enables different research teams to compare and discuss their formulations based on the unified terminology enabled by the proposed foundations.

Acknowledgments. This work has been partially supported by CNPq under grant No. 140646/2000-0 for Viviane Silva, grant No. 140179/95-0 for Anarosa Brandão, grant No. for Christina Chavez, grant No. 141457/2000-7 for Alessandro Garcia, and by FAPERJ under grant No. E-26/150.699/2002 for Alessandro Garcia. Viviane Silva, Alessandro Garcia, Anarosa Brandão, Christina Chavez and Carlos Lucena also are supported by the PRONEX Project under grant 7697102900. Paulo Alencar research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), Human Resources Development Canada (HRDC) and IBM Canada. The authors would also like to acknowledge the contributions of Don Cowan, Toacy Oliveira and Gustavo Robichez.

References

1. Alencar, P., Cowan, D., Lucena, C.: A Logical Theory of Interfaces and Objects. IEEE Transactions on Software Engineering, Vol. 28, n. 6, June, (2002): 548-575.
2. Basili, V. et al.: Experimentation in Software Engineering. IEEE Transactions on Software Engineering, SE-12(7), July (1986).
3. Bäumer, D. et al.: Role Object. In: Proceedings of the Conference on Pattern Languages of Programs, (1997).
4. Biddle, J., Thomas, E.: Role Theory: Concepts and Research. John Wiley and Sons, New York, (1966).
5. Booch, G.: Object-oriented Analysis and Design with Applications". The Benjamin/Cummings Publishing Company, Inc., 2nd edition, USA, (1994).
6. Booch, G., Rumbaugh, J., Jaconbson, I.: The Unified Modeling Language User Guide. Addison Wesley, (1999).
7. Bradshaw, J.: An Introduction to Software Agents. In: J. Bradshaw (Ed.), Software Agents, American Association for Artificial Intelligence/MIT, Cambridge, (1997): 3-46.

8. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Modeling Early Requirements in Tropos: a Transformation Based Approach. In: Proceedings of the 2nd Workshop on Agent-Oriented Software Engineering, Montreal, Canada, May, (2001).
9. Chen, P.: The Entity Relationship Model – Towards a Unified View of Data. ACM Transactions on Database Systems, vol. 1, no1. March 1976, 9-36.
10. Dardenne, A., Lamsweerde, A., Fickas, S.: Goal-directed Requirements Acquisition. Science of Computer Programming, (1993) 20:3-50.
11. Dieste, O., Juristo, N., Moreno, A., Pazos, J.: Conceptual Modeling in Software Engineering and Knowledge Engineering: Concepts, Techniques and Trends. In: Chang, S.K. (eds.): Handbook of Software Engineering and Knowledge Engineering Fundamentals. World Scientific Publishing Co., Vol.1 (2001)
12. Dong, J., Alencar, P., Cowan, D.: A Behavioral Analysis Approach to Pattern-Based Composition. Journal of Systems and Software, (2003). (To Appear)
13. Elammari, M., Lalonde, W.: An Agent-Oriented Methodology: High-level and Intermediate Models. In: Wagner, G., Yu, E. (eds).: Proceedings of the 1st International Workshop on Agent-Oriented Information Systems (1999).
14. Foundation of Intelligent Physical Agent: FIPA Interaction Protocols Specification, (2002). Available at URL http://www.fipa.org/repository/ips.html
15. Garcia, A., Cortés, M., Lucena, C.: A Web Environment for the Development and Maintenance of E-Commerce Portals based on a Groupware Approach. In: Proceedings of the Information Resources Management Association International Conference (IRMA'01), (2001): 722-724.
16. Garcia, A., Silva, V., Lucena, C., Milidiú, R.: An Aspect-Based Approach for Developing Multi-Agent Object-Oriented Systems. In: Proceedings of the 21st Brazilian Symposium on Software Engineering, Rio de Janeiro, Brazil, October, (2001): 177-192.
17. Garcia, A., Chavez, C., Silva, O., Silva, V., Lucena, C.: Promoting Advanced Separation of Concerns in Intra-Agent and Inter-Agent Software Engineering. In: Proceedings of the Workshop on Advanced Separation of Concerns in Object-Oriented Systems (ASoC) at OOPSLA'2001, Tampa Bay, USA, October, (2001).
18. Garcia, A., Silva, V., Chavez, C., Lucena, C.: Engineering Multi-Agent Systems with Aspects and Patterns. Journal of the Brazilian Computer Society, November, (2002).
19. Garcia, A., Lucena, C. Software Engineering for Large-Scale Multi-Agent Systems – SELMAS 2002. Post-Workshop Report, ACM Software Engineering Notes, August, (2002).
20. Garcia, A., Lucena, C., Cowan, D.: Agents in Object-Oriented Software Engineering. Software: Practice and Experience, Elsevier, (2003). (Accepted to Appear)
21. Genesereth, N., Ketchpel, S.: Software Agents. Communications of the ACM, v. 37, n. 7, July, (1994).
22. Glaser, N., Morignot, P.: The Reorganization of Societies of Autonomous Agents. In: Boman, M., Velde, W. (Eds.): Proceedings of the 8th European Ws. on Modeling Autonomous Agents in a Multi-Agent World, Springer, Berlin, Germany (1997).
23. Harrison, W., Ossher, J.: Subject-Oriented Programming: A Critique of Pure Objects. In: Proceedings of OOPSLA'93, (1993): 411-428.
24. Helfin, J., Hendler, J.: Semantic Interoperability on the Web. In: Proceedings of the 17th National Conference on Artificial Intelligence, (2000): 443-449.
25. Huhns, M., Singh, M.: Agents and Multiagent Systems: Themes, Approaches and Challenges. In: Huhns, M. Singh, M. (Eds.): Readings in Agents. Morgan Kaufmann (1998): 1–23.
26. Iglesias, C., Garrijo, M., Gonzalez, J, Velasco, J.: Analysis and Design of Multiagent Systems using MAS-CommonKADS. In: Singh, M. et al (Eds.), Intelligent Agents IV: Agent Theories, Architectures and Languages, LNCS 1365, (1997).

27.Iglesias, C., Garrijo, M., Gonzalez, J.: A Survey of Agent-Oriented Methodologies. In: Proceedings of the 5th International Workshop on Intelligent Agents: Agent Theories, Architectures, and Languages (ATAL-98), Paris, France, July (1998): 317-330.
28.d'Inverno, M., Luck, M.: Understanding Agent Systems. Springer Series on Agent Technology, Springer (2001).
29.Jacobson, I.: Object-Oriented Software Engineering. Addison-Wesley, (1992).
30.Jennings, N.: Commitments and Conventions: The Foundation of Coordination in Multiagent Systems. Knowledge Engineering Review, Vol.8, n. 3, (1993): 223–250.
31.Jennings, N., Wooldridge, M. Applications of Intelligent Agents. In: Agent Technology: Foundations, Applications, and Markets, Springer, Heidelberg, Germany, (1998): 3-28.
32.Jennings, N., Sycara, K., Wooldridge, M.: A Roadmap of Agent Research and Development. Journal of Autonomous Agents and Multi-Agent Systems, v. 1, n. 1, (1998): 7-38.
33.Jennings, N.: Agent-Oriented Software Engineering. In: Proceedings of the Twelfth International Conference on Industrial and Engineering Applications of Artificial Intelligence, (1999):4-10.
34.Jennings, N.: On Agent-based Software Engineering. Artificial Intelligence, Vol. 117, n. 2, (2000): 277-296.
35.Jennings, N., Wooldridge, M.: Agent-Oriented Software Engineering. In: Bradshaw, J. (ed.): Handbook of Agent Technology, AAAI/MIT Press, (2000).
36.Jennings, N.: An Agent-based Approach for Building Complex Software Systems. Communications of the ACM, v. 44, n. 4, (2001): 35-41.
37.Kiczales, G. et al.: Aspect-Oriented Programming. In: Proceedings of European Conference on Object-Oriented Programming (ECOOP), LNCS, (1241), Springer-Verlag, Finland, June (1997).
38.Kinny, D., Georgeff, M., and Rao, A.: A methodology and modeling technique for systems of BDI agents. In: Van de Velde, W., Perram, J. (eds.): Agents Breaking Away: Proceedings of the Seventh European Workshop on Modeling Autonomous Agents in a Multi-Agent World, Springer-Verlag, Vol.1038 (1996) 56-71.
39.Kinny, D., Georgeff, M.: Modelling and design of multi-agent systems. In: Müller, J., Wooldridge, M., Jennings, N. (eds.), Intelligent Agents III, Springer-Verlag, Vol.1193 (1997) 1-20.
40.Kinny, D.: The # Calculus: An Algebraic Agent Language. In: Intelligent Agents VIII, Springer-Verlag, Vol. 2333, (2002): 32-50.
41.Kristensen, B., Østerbye, K.: Roles: Conceptual Abstraction Theory and Practical Language Issues. Theory and Practice of Object Sytems Vol. 2, n. 3, (1996): 143–160.
42.Lind, J.: MASSIVE: Software Engineering for Multiagent Systems. PhD thesis, University of the Saarland (2000).
43.Maes, P.: Concepts and Experiments in Computational Reflection. In: Proceedings of OOPSLA'87, ACM SIGPLAN Notices, October (1987): 147-155.
44.Mamei, M. et al. Engineering Mobility in Large Multi Agent Systems: a Case Study in Urban Traffic Management. In: A. Garcia et al. (eds), Software Engineering for Large-Scale Multi-Agent Systems, Springer, LNCS, 2003.
45.MESSAGE website, Available at URL http://www.eurescom.de/Public/Projects/p900-series/P907/P907.htm.
46.B. Meyer. Object-Oriented Software Construction. Prentice-Hall, 2nd edition, (1997).
47.Minsky, N., Ungureanu, V.: Law-Governed Interaction: A Coordination and Control Mechanism for Heterogeneous Distributed Systems. ACM Transactions on Software Engineering and Methodology, Vol. 9, n. 3, July, (2000): 273-305.
48.Moss, S., Gaylard, H., Wallis, S., Edmonds, B.: SDML: A Multi-Agent Language for Organizational Modelling. Computational Mathematical Organization Theory, Vol. 4, n. 1, (1998): 43-69.

49. Neto, A., Lucena, C.: CommercePipe: Consumer to Business Commerce Channels on the Internet. SEA (Software Engineering Applications) IASTED, Las Vegas, October (2000)
50. Newcomer, E., Hurley, O.: Web Services Definition. In: Proceedings of the World Wide Web Consortium Workshop on Web Services, April, (2001).
51. Nwana, H.: Software Agents: An Overview. *Knowledge Engineering Review*, 11(3):1-40, 1996.
52. Object Management Group – Agent Platform Special Interest Group: Agent Technology – Green Paper. Version 1.0, September (2000).
53. Object Management Group: OMG MOF – Meta Object Facility (MOF) Specification. Version 1.4, April, (2002). Available at URL http://www.pmg.org/cwm.
54. Odell, J., Parunak, H., Bauer., B.: Extending UML for Agents. In: Odell, J., Parunak, H. and Bauer, B. (Eds.), Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence, 2000.
55. Omicini, A., Petta, P., Tolksdorf, R.: Engineering Societies in the Agents World II. Proceedings of the 2nd International Workshop on Engineering Societies in the Agents World, Praga, (2001).
56. Omicini, A.: From Objects to Agent Societies: Abstractions and Methodologies for the Engineering of Open Distributed Systems. In: Corradi, A., Omicini, A., Poggi, A. (eds.): WOA (2000): 29-34.
57. Omicini, A.: SODA: Societies and Infrastructure in the Analysis and Design of Agent-based Systems. In: Ciancarini, P., Wooldridge, M. (eds.), Agent-Oriented Software Engineering, Springer-Verlag, (2001): 185-194.
58. Parunak, H., Odell, J.: Representing Social Structures in UML. In: Proceedings of Agent-oriented Software Engineering, (2001): 1-16.
59. Petrie, C.: Agent-Based Software Engineering. In: Ciancarini, P., Wooldridge, M. (eds.), Agent-Oriented Software Engineering, Springer-Verlag, (2001).
60. Rumbaugh et al. Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, New Jersey (1991).
61. Rasmus, D.: Rethinking Smart Objects – Building Artificial Intelligence with Objects. Cambridge University Press (1999).
62. Ripper, P., Fontoura, M, Neto, A., Lucena, C.: V-Market: A Framework for e-Commerce Agent Systems. World Wide Web, Baltzer Science Publishers Vol. 3, n. 1, (2000).
63. Shoham, Y.: Agent0: A Simple Agent Language and its Interpreter. In: Proceedings of the Ninth National Conference on Artificial Intelligence, (1991): 704–709.
64. Shoham Y.: Agent-Oriented Programming. Artificial Intelligence, n. 60, (1993): 24–29.
65. SoC+Agents Group. Separation of Concern and Multi-Agents Systems Group. Available at UR: http://www.teccomm.les.inf.puc-rio.br/SoCagents/.
66. Tarr, P. et al.: N Degrees of Separation: Multi-Dimensional Separation of Concerns. In: Proceedings of the 21st International Conference on Software Engineering, May, (1999).
67. The Semantic Web Portal. Available at URL: http://www.semanticweb.org.
68. Wagner, G.: Agent-Object-Relationship Modeling. In: Proceedings of the 2nd International Symposium: From Agent Theory to Agent Implementation, April (2000).
69. Willmott, S., Dale, J., Burg, B., Charlton, C., O'Brien, P. Agentcities: A Worldwide Open Agent Network. Agentlink News, November, (2001): 13-15.
70. Wooldridge M., Jennings N., Kinny, D.: A Methodology for Agent-Oriented Analysis and Design. In: Proceedings of the Third International Conference on Autonomous Agents (Agents'99), ACM Press (1999): 69–76
71. Wooldridge, M., Jennings, N., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. In: Journal of Autonomous Agents and Multi-Agent Systems, Vol. 3, (2000): 285–312.

72. Wooldridge, M., Ciancarini, P.: Agent-Oriented Software Engineering: The State of the Art. In: P. Ciancarini And M. Wooldridge (Eds.), Agent-Oriented Software Engineering, Springer-Verlag, LNAI, 2001.
73. Yu, L., Schmid, B.: A Conceptual Framework for Agent-Oriented and Role-Based Work on Modeling. In: Wagner, G., Yu, E. (eds.): Proceedings of the 1st Int. Workshop. on Agent-Oriented Information Systems (1999).
74. Zambonelli, F., Jennings, N., Wooldridge, M.: Organizational Abstractions for the Analysis and Design of Multi-agent Systems. In: Ciancarini, P., Wooldridge, M. (eds.): Agent-Oriented Software Engineering, Springer-Verlag (2001).
75. Zambonelli, F. et al.: Agent-Oriented Software Engineering for Internet Applications. In: Omicini, A., Zambonelli, F., Klusch, M. (Eds.), Coordination of Internet Agents: Models, Technologies, and Applications, Springer-Verlag, New York, (2001).
76. Zambonelli, F., Jennings, N., Wooldridge, M.: Organizational Rules as an Abstraction for the Analysis and Design of Multi-agent Systems. Journal of Knowledge and Software Engineering, v.11, n.3, (2001).
77. Zambonelli, F., Parunak, H.: Signs of a Revolution in Computer Science and Software Engineering. In: Proceedings of the Third International Workshop Engineering Societies in the Agents World, Madrid, Spain, (2002).