Theoretical Foundations for Agents and Objects in Software Engineering

Viviane Torres da Silva[1] Alessandro Fabrício Garcia[1] Anarosa Alves Franco Brandão[1]

Christina von Flach Garcia Chavez[2] Carlos José Pereira de Lucena[1] Paulo Sergio Conceição 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

PUC-RioInf.MCC 30/02 Dezembro, 2002

Abstract. Agent-based software engineering has been proposed in addition to object-oriented software engineering for mastering the complexity associated with the development of large-scale distributed systems. However, there is still a poor understanding of the interplay between the agent and object notions 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 support consistent application of its core abstractions. In this context, this paper presents TAO, an evolving innovative conceptual framework based on the agent and object abstractions, which are the foundations for modeling large-scale software systems. The conceptual framework provides support for decomposing 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.

Keywords: agents, objects, conceptual framework, software engineering, large-scale systems

Resumo. Engenharia de software baseada em agente vem sendo proposta como o objetivo de abordar a complexidade associada ao desenvolvimento de sistemas distribuídos de larga escala adicionando novas abstrações e características a engenharia de software baseada em objetos. Todavia, na perspectiva da engenharia de software, o relacionamento entre as duas principais abordagens utilizadas, objetos e agentes, ainda não está suficientemente entendido. Sendo assim, as muitas facetas da engenharia de software baseada em agentes são raramente utilizadas em várias fases do desenvolvimento do ciclo de vida do software. Como conseqüência, este trabalho apresenta TAO, um framework conceitual baseado nas abstrações básicas para modelagem de sistemas de software para larga escala, agentes e objetos. O framework conceitual provê suporte a decomposição de sistemas de software em organizações de elementos passivos, objetos, e elementos autônomos, agentes. Cada um destes elementos desempenha papéis ao interagir entre si e coordenar suas ações com o objetivo de atingir os objetivos do sistema.

Palavras-chave: agentes, objetos, framework conceitual, engenharia de software, sistema de larga escala

1 Introduction

With the advances on the networking technologies [19, 45, 61], 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 [42, 63, 71]. Object-oriented software engineering [4, 5, 41, 54] 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 [14, 15, 31, 39, 50, 60, 69]. So, the limitations of the object paradigm has spurred research on agent-based software engineering [27, 28, 29] as an additional approach to the development of large-scale systems from its conceptual modeling [7, 62, 65] to its computational modeling [13, 16, 49].

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) [58] and their underlying theories bring with themselves more natural support for autonomy, coordination, mobility, organizations, openness, and intelligence. In this sense, the discipline of Software Engineering is trying to understand how the lessons learned on the application of these agent theories in Artificial Intelligence can be used to overcome the limitations of object-oriented software engineering and master the complexity of modern software. The successful and widespread deployment of large-scale MASs requires a unifying set of central abstractions to uniformly support modeling languages and respective methodologies for an agent-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 it is the case with any new software engineering paradigm, researchers are beginning to strive for the underlying methodologies that guide the process of constructing MASs [22, 34, 37, 51, 65]. Many, such as Agent UML [48] and MAS-CommonKADS [21], are extensions of previous object-oriented methodologies and languages, while others, such as AAII methodology [34], 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 software engineers understanding 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-based software engineering are still rarely used in the various phases of the software lifecycle because of the lack of a comprehensive framework to support consistent application of its core concepts [14, 66].

In this context, this paper presents TAO, an evolving conceptual framework that provides the theoretical foundations for an agent-based software engineering. The framework specifies an ontology that defines the essential concepts for developing MASs, where each concept is viewed as an abstraction to modeling languages and methodologies to be applied in different phases of the MAS development. The proposed ontology provides software engineers with support for easily decomposing a large-scale MAS in terms of agents, objects, and their common and distinguished abstractions, since it identifies abstractions for an agent-based software engineering in the light of classical abstractions of object-oriented software engineering. We classify the abstractions used to establish our theoretical foundations into three main categories:

Fundamental abstractions provide software engineers with basic abstractions to defining objects and agents and their internal elements;

Grouping abstractions support software engineers with the construction of organizations or groups of agents and objects, and their coordination elements;

Environment abstractions allow software engineers to think about the limitations and constraints of the environments in which passive and proactive components (i.e. agents and objects) operate and pursue their goals.

TAO can be tailored to different domains since its basic ontology can be extended to accommodate new abstractions for these domains. TAO enables different research teams to compare and discuss their formulations based on the unified terminology enabled by the proposed foundations enable. The remaining of this paper is organized as follows. Section 2 presents the conceptual framework and a brief view of its abstractions and their relationships. Section 2 also describes the case study used in the paper in order to exemplify our definitions. A complete definition of the fundamental, environment and group abstractions are presented in sections 3 and 4. Section 5 introduces and defines the relationships between those abstractions. Section 6 reviews some related work and section 7 discusses some important issues. Finally, section 8 presents the conclusions of our work.

2 The Conceptual Framework

2.1 The Role of the Theoretical Foundations

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. The purpose of conceptual models is to provide an understanding of the problem domain describing the problem [8]. In order to produce a solution, computational models may be generated based on conceptual models. Computational models describe what problem-solving software systems will be like [8]. 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 theoretical foundations for agent-based software engineering. TAO presents the definition of each abstraction as a concept of its ontology, and the provision of relationships between them.
The definition of the relationships allows software engineers to understand the interactions between abstractions from agent-based and object-oriented software engineering.
Fig. **1** shows a three-layer picture that illustrates the role of TAO from a software engineering viewpoint. It presents the three layers that underlie the process of problem and solution modeling. The abstractions and the relationships presented in TAO are described at the meta-level. The meta-level realizes the domain-independent abstractions. It presents meta-concepts such as agents and objects, meta-properties defined for each abstraction and meta-relationships that link those abstractions. The first layer in

Fig. **1** encompasses a partial meta-model of the TAO framework. This meta-model structures the relationships between the agent, role and organization abstractions.

The second layer, the domain level, depicts the concepts specific to the application domain. The concepts at the meta-level are instantiated into domain models using the domain information. These domain models describe how instances of meta-concepts are linked through instances of meta-relationships in the context of the domain. The example presented in

Fig. **1** shows a domain model where the meta-concepts *Agent*, *Role* and *Organization* are instantiated as *User Agent*, *Buyer* and *Marketplace*, respectively.

In order to create domain models, modeling languages are needed. Modeling languages are used to provide a common language that elicits the meaning of each concept described in the conceptual model. For each meta-concept that appears in the meta-level, the modeling language creates a symbol that has a meaning in the domain. The domain level uses those symbols to represent the instances of meta-concepts and meta-relationships defined at the meta-level.

The instance level presented in the third level characterizes the possible domain-level occurrences. This level describes the specific instances of the domain-level concepts. 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 equivalent to their desire. They can move to another marketplace in order to look for offers that they did not find in the original one. Alternatively, they can make 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.

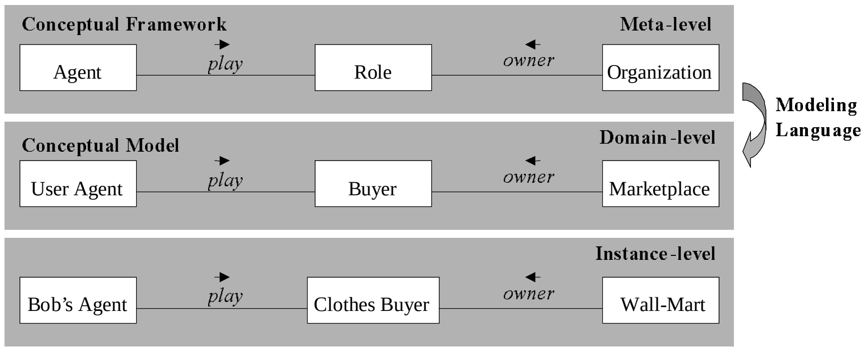

Fig. 1. From the meta-level to the instance-level

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 most important abstractions of TAO and their relationships. There is a total of 8 abstractions including entity, object, agent, organization, environment, event, object role and agent role. The entity abstraction provides a basic definition to describe the different entities.

An entity has properties and relationships with others entities.

Objects, agents, environments, organizations, object roles and agent roles are abstractions whose definitions are based on the definition of the entity abstraction. Their definitions extend the definition of an entity by identifying their specific properties and relationships. The *properties* of an entity describe its state and behavior characteristics. The *state* of an entity defines information about other entities of the system and the *behavior* of an entity defines the actions or operations that the entity can perform. An entity can change its state and interact with other entities. An entity must be related to another entity to interact with it, i.e., it must have a *relationship* between the two entities so that the two entities can interact.

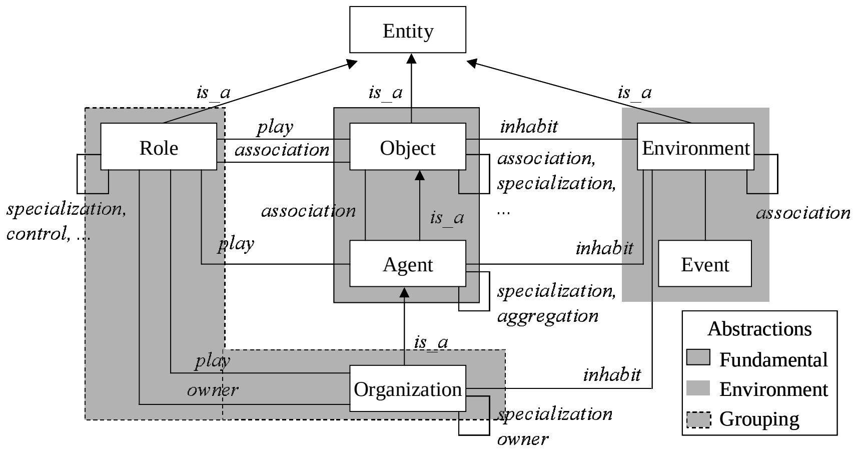

Fig. 2. The most important abstractions of the conceptual framework

The *relationships* link two entities and describe how these entities are related to each other. As described in Fig. 2, different entities are related in different ways, i.e. there are different types of relationships (Section 5). An entity *class* defines properties and relationships that are the same to all its *instances*. An instance is a concrete manifestation of an abstraction to which a set of properties and relationships are applied [5]. An entity instance of a class fulfills the description of their class.

Besides defining the abstractions, we introduce templates associated with each abstraction. The templates are used to define each abstraction in a rigorous way. They list the set of properties and relationships of each entity. As the templates are defined in the meta-level, they are instantiated in the domain-level. In order to exemplify the use of our templates, we have applied them to the marketplace domain (Section 2.1) through this paper.

2.3 The Abstractions and their Relationships: An Overview

A multi-agent system (MAS) comprises classes and instances of *agents*, *objects* and *organizations*. Organizations group together the agents of a MAS [40, 58]. Agents, organizations, and objects, inhabit (or are immersed in) *environments* [28, 37] that provide *resources* and make available *services*. Resource are non-autonomous entities such as databases or external programs used by agents or organizations. Objects are adequate for modeling resources [40]. While objects represent passive elements, such as resources, agents represent autonomous entities that manipulate objects. Agents, objects and organizations make available services to other agents and organizations [34].

Organizations can have *sub-organizations* and each organization is composed of a group of agents and their sub-organizations. An organization describes a set of *roles* [3] that limits the behavior of its agents and sub-organizations that play the defined roles [64]. In the perspective of an organization its sub-organizations can be viewed as agents that play roles and have relationships with other agents. An organization can also define roles for its objects.

Agents and objects can be members of different organizations and play different roles in each of them [52]. Every agent of the MAS plays at least one role in an organization. While agents play roles only in the organization context, objects additionally can play roles out of an organization. Agents may interact with each other and cooperate either to achieve a common goal, or to achieve their own goals [68]. 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 between their agents. Fig. 3 illustrates the elements of a MAS and their relationships. We enumerate the central concepts illustrated in the figure.

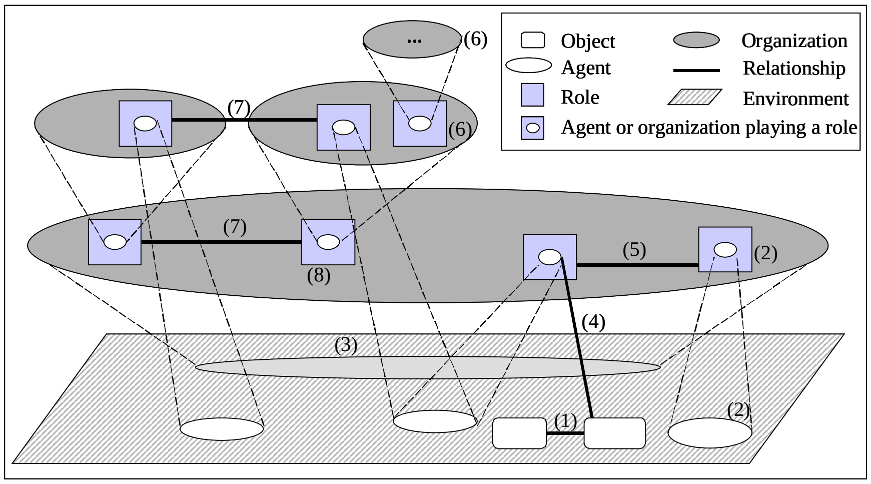

Fig. 3. The abstractions and their relationships.

1. Relationship between objects. Objects are resources that inhabit the environment. An object may be related to other objects of the environment.
2. Agents inhabit the environment and play roles in organizations.
3. MAS organizations inhabit the environment.
4. Relationship between objects and agents. When playing roles agents may access the resources of the environment.
5. Relationship between agents. Agents play roles to interact with other agents.
6. Organizations may have sub-organizations that play roles in the organization.
7. If agents of two different sub-organizations are related, the sub-organizations are related.
8. In the perspective of the organization, sub-organizations are viewed as agents.

3 Fundamental and Environment Abstractions

This section describes the fundamental abstractions, object and agent, and the environment abstractions. Each abstraction is described in terms of the following issues: (i) basic definition, (ii) state properties, (iii) behavior properties, (iv) its interplay with other abstractions, (v) its associated template, (vi) its application to model the markeplace example. This section also highlights the most important differences and commonalities 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 entity that has state and behavior and can be related to other entities.

An object is an entity that has a state and a number of operations (behavior) to either examine or change its state [24]. An object extends the definition of an entity since it defines state and behavior properties, and relationships with other objects or other entities. 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 a system's entities, such as other objects, agents, and roles.

An object has control of its state. It performs operation that can modify its state during its lifetime. On the other hand, an object cannot modify its behavior[1] 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 entities that do whatever anyone asks them to do and only when they are asked. A common object *class* defines the structure and behavior of similar objects [4]. The following template defines the state, behavior and relationships that an object class must have.

```
______________________Object______________________

 Object_Class Object_Class_Name
        State  setOf{Information}
 Behavior setOf{Operation}
        Relationships setOf{Relationship_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 represents 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 it 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}
End Object_Class
_________________________________________________________________________
```

[1] Computational reflection has been introduced in the object paradigm [Patti Maes] 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 entity that has a mental state.

Software agents are complex objects with an attitude [6]; that is, they extend objects with a structured state and agency behavioral properties. According to [20, 29, 47], agents are interactive, autonomous and adaptive entities, and those are the three fundamental characteristics that define agency. Characteristics such as learning ability, mobility and rationality 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 [29, 47, 58]. 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. Agents are goal-oriented; thus, an agent within an 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, sends and receives messages while executing plans. 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 pos-conditions are satisfied the correctness of the action is guaranteed.

The *behavior* of an agent is expressed through its 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 entities when playing roles in an organization. Agents interact with others since they have relationships with other system's entities. The *relationships* describe how an agent is linked to another entity. 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. The agent is capable of perceiving events, receiving messages, sending out new messages and generating events. Agents are adaptive entities since they can adapt 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 agent instances of it.

```
______________________Agent_________________________

  Agent_Class Agent_Class_Name
        Beliefs setOf{Belief_Name}
        Goals setOf{Goal_Name}
        Actions setOf{Action_Name}
        Plans setOf{Plan_Name}
        Relationships setOf{Relationship_Class_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}
     Relationships {Inhabit_Env_User_Agent, Play_Buyer,
                    Play_Seller, Play_Mediator}
End Agent_Class
_____________________________________________________________
```

3.3 Agent vs. Object

We define an agent as an extension of an object because it extends the definition of state and behavior associated with objects. The state of an agent has a “mentalistic” structure [62], 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. The mental state is consisted of beliefs that are equivalent to the object’s state and also of goals, plans and actions that define the agent’s behavior. Moreover, the behavior of an agent extends the behavior of objects because an agent has full control of its behavior, i.e., agents can say no to the requests of other agents, agents can change their behavior, adding new actions to be executed and agents do not require external stimuli to carry out their jobs. Thus, an agent is an active entity and an object is a passive entity.

Another difference between agents and objects is related to the agency characteristics. As we have already seen, an agent is an autonomous and interactive entity that sends and receives messages. As shown in Fig. 4 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. The more autonomous an agent is the less interactive it needs to be. An proactive agent does not need to cooperate with anyone to achieve its own goal. On the other hand, the less autonomous an agent is the more interactive it needs to be to achieve its goals.

Although, an object is an interactive entity it is not an autonomy one. It is a reactive and passive entity since it needs the solicitation of another entity to do its job and since it responds to any solicitation. Classically, an object interacts a lot with other objects in order to do their jobs. From another point of view, an agent may be a proactive entity that does not need to interact with other agents to do its job.

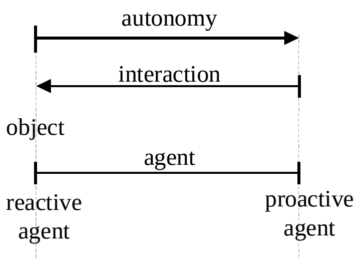

Fig. 4. Autonomy x Interaction

3.4 Environment and Events

An environment is an entity that is the habitat for agents, objects and organizations. An environment can be heterogeneous, dynamic, open, distributed and unpredictable [51].

An environment extends the definition of an entity 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 permission to them. Resources are objects that can be accessed by other objects and by agents or organizations when playing roles. The permissions associated with resources restricts the access of objects and agents to them.

Agents, objects and organizations make available services 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 [51]. An environment can be a passive entity such as an object or can be an active entity such as an agent having agency characteristics such as autonomy, adaptation and interaction. Thus, an environment may be seen as an agent when appropriate. When an environment is an active entity, it can generate events that can be perceived by agents [62]. Events also can be generated by an action carried out by agents or objects. An event generated by the environment, by an agent or by an object can trigger the execution of an action associated with the agent and objects that perceive it.

An environment is the habitat of MAS organization, agents and objects. Organizations may inhabit multiple environments but agents and objects must inhabit only one environment at any given moment. Different environments can be the habitat of different entities and can have different characteristics, resources and services. The *relationships* of an environment describe which entities 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 instance 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, Entity_Class_Name>}
   Services setOf{<Service, Permission, Entity_Class_Name>}
   Behavior setOf{Properties}
   Relationship setOf{Relationship_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 support dealing with products and 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, ...}
 end Environment_Class
_______________________________________________________________
```

4 Grouping Abstractions

MAS comprises a set of grouped agents immersed in one or more environments whose global behavior derives from the interaction among the constituent agents [68]. The group of agents comprising the MAS defines organizations whose goals are the same as the MAS. An MAS has at least one organization that represent 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.

4.1 Organization

An organization is an entity that groups agents that 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 their sub-organizations. Recursively, each sub-organization can have others sub-organizations defined within it.

From the perspective of entities 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. This 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 respects 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 it. 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 entity. 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 entity in fact occurs between an agent inside the organization and the entity.

The organization template presents an organization class that describes the rules and laws as well as the relationships associated to all instances.

```
______________________ 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* inhabits the environment *MarketPlaceEnv*, it owns the roles *Buyer*, *Seller* and *Mediator*, and the sub-organization *Buying_Group*. Furthermore, it has objects that provide support for negotiating products, such as *Offer*, as well as some rules and laws guiding the behavior of agents playing roles and agents within *MarketPlaceOrg*, respectively.

```
___________________________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
___________________________________________________________________
```

4.2 Roles

Defined in the context of an organization, a role is an entity 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 sub-organization. A role is an entity 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 because it can add behavior and relationships to the object that plays the role and can restrict the access to the object [36].

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 entity that was not defined in the object.

From the point of view of the entity that is related to the object that is playing a role, the role identifies the properties that the entity 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 entities 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 entity since it has state, behavior and relationship with other entities. 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 [64] of the organization's agents. The duties define actions, services and functions 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. Each agent has a partial notion of the whole system [30, 51] and none of the agents have sufficient competence, resources or information to solve the whole problem [25].

The protocols and commitments define the interactions between roles and other entities. Protocols may define a set of interactions and rules that the entities playing the role and participating in the protocol must obey. A commitment defines a set of actions that an entity 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 and duties, rights, and protocols and commitments that define the interactions. This 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*.

```
_________________________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
_______________________________________________________________
```

5 Relationships

This section presents the relationships between all entities of the conceptual framework. There are eight different relationships classified in two different ways that associate objects, agents, environments, organizations and roles.

5.1 Relationship Classification

A relationship is classified in two different ways. A relationship is mandatory or optional, and directed or undirected. The mandatory and optional classification is characterized if the relationship defined in the meta-level must or must not be associated with two entities' classes in the domain-level. The directed or undirected classification describes if the relationship between two entities is defined or not through their roles.

- Mandatory – Mandatory relationships are relationships that are partially independent of the domain-level. The conceptual framework defines a set of mandatory relationships that must be used whenever the entities' classes appear in the domain-level. It is independent of the domain-level since, if two entities are related by a mandatory relationship in the meta-level, their entities' classes must be associated by the same relationship in the domain-level. But it is partially independent because some entities defined in the meta-level may not appear in the domain-level. This relationship does not enumerate which entity must appear as the entities' class in the domain-level, but it imposes that if they appear the mandatory relationship must be used.

- Optional – Optional relationships are relationships that are completely dependent of the domain-level. The conceptual framework defines a set of optional relationships linking two entities that should be used depending on the problem domain. An optional relationship defined to link two entities in the meta-level may not be used in the domain-level to link its entities' classes.
- Undirected – Undirected relationships are all relationships that depend on the role that an entity plays. An undirected relationship occurs when an entity is playing a role and is related to another entity by the role. In this way, there is an undirected relationship that links the entities. This relationship is the same as the one defined by the roles played by the entities. Only entities that play roles can be related in an undirected way.
- Directed – Directed relationships are relationships that link two entities in a directed way, independently the role played by the entities.

5.2 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, $org \in Org$, org, $subOrg \in Org$ and subOrg always represents a sub-organizations. Let R be a set of roles, $R = RObj \cup RAg$ where RObj is a set of object roles and RAg is a set of agent roles, $r \in R$, $ro \in Robj$ and $ra \in RAg$. There is a list of eight relationships described below. For each one, we present its definition, its classification and the entities it links.

- Inhabit (I)

  Mandatory, Directed: I(a,e), I(o,e), I(org,e)

Some entities must inhabit environments and can dynamically change their habitat. The inhabit relationship specifies that the entity that inhabits - the citizen - may leave in and enter habitats, respecting the habitat rules. Normally, the habitat does not guide the executions of its citizens not impose when to enter or leave or what actions they must carry out. On the other hand, the habitat restricts which entities can enter, which resources and services they can access and which services they can provide. When a citizen changes habitat it no longer is subordinated to its old habitat.

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

- Ownership (Ow)

  Mandatory, Directed: Ow(org, r), Ow(org, subOrg)

  Mandatory, Undirected: Ow(org, a), Ow(org, o)

Some entities must be members of another entity. The ownership specifies that an entity - the member - is defined in the scope of other entity - 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 its 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 a role as defined by that organization. Indirectly, agents and objects that play roles in a organization are members of that organization.

- Play (P)

  Mandatory, Directed: P(a,ra), P(subOrg,ra)

  Optional, Directed: 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)
  Optional, Directed: S(o,o), S(a,a), S(org,org), S(ro,ro), S(ra,ra)

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

- Control (C)[2]
  Optional, Directed: C(ro,ro), C(ra,ra), C(ra,ro)
  Optional, Undirected: C(o,o), C(a,a), C(a,o)

The control relationship defines that the controlled entity that plays the role does anything that the controller entity 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)
  Optional, Directed: D(ro,ro), D(ra,ra), D(ra,ro)
  Optional, Undirected: D(o,o), D(a,a), D(a,o)

An entity - the client - may be defined to be dependent of 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 only can depend on another object role.

- Association (As)
  Optional, Directed: As(r,r), As(e,e), As(o,o), As(a,o), As(o,org)
  Optional, Undirected: As(o,o), As(a,o), As(a,a), As(subOrg,subOrg)

If an entity is associated with another entity, it knows that the other entity exists. The association relationship must define how an entity interacts with another one. Roles are directed associated to other roles as well as environments.

Objects may be directly or indirectly associated with other objects. The same happens between an agent and an object that may be directly or indirectly associated through its roles. But two agents and two sub-organizations cannot be directly associated. Agents and sub-organizations are associated when playing roles. An organization can be associated with objects that do not play roles in the organization. In the case an object plays a role in the organization, the organization and the object are indirectly linked by the owner relationship defined between the role and the organization.

- Aggregation/Composition (Agg)
  Optional, Directed: Agg(ro,ro), Agg(ra,ra)
  Optional, Undirected: Agg(o,o), Agg(a,a)

If an entity is aggregated to other entity, we say that it is part of an aggregator. The aggregator may use the functionalities available in its parts but the parts do not necessarily have any relationship. The parts do not need to know that is being aggregated to an aggregator, but the aggregator knows each of its parts. Depending on the strength of the aggregation, the part may not exist without the aggregator.

[2] We are extending the relationships control and dependency described in [Zambonelli].

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

```
________________________Relationship___________________________

 Relationship Relationship_Name
       INHABIT: habitat, citizen
     | OWNERSHIP: owner, member
     | INHERITANCE: super-entity, sub-entity,
     | PLAY: entity, role
     | CONTROL: controller, controlled, condition
     | DEPENDENCY: client, supplier, condition
     | ASSOCIATION: Entity_Class_Name1,Entity_Class_Name2
     | AGGREGATION: aggregator, part
 end Relationship
_______________________________________________________________
```

Only two relationships are exemplified in the case study. 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
____________________________________________________
```

6 Related Work

Wagner [62] 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. 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 [7]. 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 [23] defines a conceptual framework with four important limitations: (i) it does not define all possible relationships between its entities; (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 may be very difficult to be used by software engineers and methodology developers.

Finally, Yu and Schmid [67] 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.

7 Discussions and Ongoing work

Our research group [12, 59] has been conducting a set of empirical studies [10, 44, 56] 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 [11, 13]. 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 unified 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 eight 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 [9, 37, 40, 62, 65, 67], languages [35, 43, 57], and theories [53, 58, 62]. 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 means 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.

We are knowable to develop methodologies and methods for large-scale MAS based on a unified conceptual framework of agents and objects. 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 [33]. 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 and conceptual models to generate computational models. In this way, abstractions used in conceptual models may be mapped to other abstractions used in computational model. 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 [17, 32, 60] is well understood in the object-oriented context, but only a few preliminary works have been published that discusses it in terms of agent-based software engineering (such as [11, 15]).

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 [18].

8 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 [50]. 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 additionally 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 [50]. 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 help face 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 theoretical foundations that provide 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 into 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.

References

1. Basili, V. et al. Experimentation in Software Engineering. IEEE Transactions on Software Engineering, SE-12(7), July 1986.
2. Bäumer, D. et al. Role Object. In Proceedings of the 1997 Conference on Pattern Languages of Programs (PLoP '97), 1997.
3. Biddle, J., Thomas, E.: Role Theory: Concepts and Research. In: John Wiley and Sons. New York (1966)
4. Booch, G. "Object-oriented analysis and design with applications". The Benjamin/Cummings Publishing Company, Inc., 2nd edition, USA, 1994.
5. Booch, G., Rumbaugh, J., Jaconbson, I.: The Unified Modeling Language User Guide. In: Addison Wesley (1999)
6. Bradshaw J., 1997. An introduction to software agents. In: J. Bradshaw (Ed.), Software Agents, pp 3-46. Press, American Association for Artificial Intelligence/MIT, Cambridge.
7. Dardenne, A., Lamsweerde, A., Fickas, S.: Goal-directed Requirements Acquisition. In: Science of Computer Programming, (1993) 20:3-50.
8. 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)
9. Elammari, M., Lalonde, W.: An agent-oriented methodology: High-level and intermediate models. In: Wagner, G., Yu, E. (eds).:. Proc. of the 1st Int. Workshop on Agent-Oriented Information Systems (1999).
10. Garcia, A., Cortés, M., Lucena, C.: A Web Environment for the Development and Maintenance of E-Commerce Portals based on a Groupware Approach. In: Information Resources Management Association International Conference (IRMA) - Web Engineering for E-Commerce Applications, (2001) pp. 722-724.
11. Garcia, A., Silva, V., Lucena, C., Milidiú, R.: An Aspect-Based Approach for Developing Multi-Agent Object-Oriented Systems. XXI Brazilian Symp. on Software Engineering, Rio de Janeiro, Brazil, October 2001, pp. 177-192.
12. Garcia, A., Chavez, C., Silva, O., Silva, V., Lucena, C. "Promoting Advanced Separation of Concerns in Intra-Agent and Inter-Agent Software Engineering". Workshop on Advanced Separation of Concerns in Object-oriented Systems (ASoC) at OOPSLA'2001, Tampa Bay, USA, October 2001
13. Garcia, A., Silva, V., Chavez, C., Lucena, C. "Engineering Multi-Agent Systems with Aspects and Patterns". Journal of the Brazilian Computer Society, Special Issue on Software Engineering and Databases, August 2002.
14. Garcia, A., Lucena, C. Software Engineering for Large-Scale Multi-Agent Systems – SELMAS 2002. (Post-Workshop Report) ACM Software Engineering Notes, August 2002.
15. Garcia, A., Lucena, C., Cowan, D. Agents in Object-Oriented Software Engineering. Software: Practice and Experience, Elsevier, 2003. (Accepted to Appear)
16. Genesereth, N., Ketchpel, S. Software Agents. Communications of the ACM, v. 37, n. 7, July 1994.
17. Glaser, N., Morignot, P.: The Reorganization of Societies of Autonomous Agents. In: Boman, M., Velde, W. (eds.): Proc. of the 8th European Ws. on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW'97). Springer, Berlin, Germany (1997)
18. Harrison, W., Ossher, J. Subject-Oriented Programming: A Critique of Pure Objects. In Proceedings of OOPSLA'93, ACM, pages 411-428, 1993.
19. Helfin, J.; Hendler, J. Semantic Interoperability on the web. In: 17th National Conference on Artificial Intelligence (AAAI-2000), pp. 443-449, 2000.
20. 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.
21. Iglesias, C.; Garrijo, M.; Gonzalez, J, Velasco, J: Analysis and Design of Multiagent Systems using MAS-CommonKADS. Intelligent Agents IV: Agent Theories, Architectures and Languages, 1997, Singh, M. et al, eds., LNCS 1365.
22. Iglesias, C.; Garrijo, M.; Gonzalez, J. A Survey of Agent-Oriented Methodologies. In: 5th International Workshop on Intelligent Agents: Agent Theories, Architectures, and Languages (ATAL-98), Paris, France, July 1998, pp. 317-330.
23. d'Inverno, M., Luck, M.: Understanding Agent Systems. Springer Series on Agent Technology. Springer (2001)
24. Jacobson, I.: Object-oriented software engineering. Addison-Wesley (1992)
25. Jennings, N.: Commitments and conventions: The foundation of coordination in multiagent systems. Knowledge Engineering Review Vol.8(3) (1993) 223–250
26. Jennings, N.; Wooldridge, M. Applications of Intelligent Agents. In: Agent Technology: Foundations, Applications, and Markets, Springer-Verlag, Heidelberg, Germany, 1998, pp. 3-28.

27. 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, p. 7-38.
28. Jennings, N.: Agent-oriented software engineering. In: Proceedings of the Twelfth International Conference on Industrial and Engineering Applications of Artificial Intelligence (1999) 4-10.
29. Jennings, N.: On agent-based software engineering. Artificial Intelligence. Vol. 117(2), (2000) 277-296.
30. Jennings, N., Wooldridge, M.: Agent-oriented software engineering. In: Bradshaw, J. (ed.): Handbook of Agent Technology, AAAI/MIT Press. (2000)
31. Jennings, N. An Agent-based Approach for Building Complex Software Systems. Communications of the ACM, v. 44, n. 4, 2001, p. 35-41.
32. G. Kiczales et al. Aspect-Oriented Programming. *European Conference on Object-Oriented Programming (ECOOP), LNCS*, (1241), Springer-Verlag, Finland., June 1997.
33. 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.
34. 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.
35. Kinny, D.: The # Calculus: An Algebraic Agent Language. In: Intelligent Agents VIII. Springer-Verlag Vol.2333 (2002)32-50
36. Kristensen, B., Østerbye, K.: Roles: Conceptual Abstraction Theory and Practical Language Issues. Theory and Practice of Object Sytems Vol. 2(3) (1996)143–160.
37. Lind, J.: MASSIVE: SoftwareEngineering for Multiagent Systems. PhD thesis, University of the Saarland (2000).
38. P. Maes. “Concepts and Experiments in Computational Reflection”. *Proc. of OOPSLA’87, ACM SIGPLAN Notices*, 22(12):147-155, October 1987.
39. Mamei, M. et al. Engineering Mobility in Large Multi Agent Systems: a Case Study in Urban Traffic Management. In: A. Garcia, C. Lucena, J. Castro, A. Omicini, F. Zambonelli (Eds). Software Engineering for Large-Scale Multi-Agent Systems. Springer, LNCS, 2003.
40. MESSAGE: Methodology for Engineering Systems of Software Agents Initial Methodology. Technical report, EURESCOM, July (2000)
41. B. Meyer. “Object-Oriented Software Construction”. Prentice-Hall, 2nd edition, 1997.
42. Minsky, N.; Ungureanu, V. Law-Governed Interaction: A Coordination and Control Mechanism for Heterogeneous Distributed Systems. ACM Transactions on Software Engineering and Methodology, v. 9, n. 3, July 2000, pp. 273-305.
43. Moss, S., Gaylard, H., Wallis, S., Edmonds, B.: SDML: A Multi-Agent Language for Organizational Modelling. Computational Mathematical Organization Theory, 4(1) (1998) 43-69
44. Neto, A., Lucena, C.: CommercePipe: Consumer to Business Commerce Channels on the Internet: SEA (Software Engineering Applications) IASTED, LAS VEGAS, October (2000)
45. Newcomer, E; Hurley, O. Web Services Definition. In: Proceedings of the World Wide Web Consortium Workshop on web Services, April, 2001.
46. H. Nwana. Software Agents: An Overview. *Knowledge Engineering Review*, 11(3):1-40, 1996.
47. Object Management Group – Agent Platform Special Interest Group. Agent Technology – Green Paper. Version 1.0, September (2000).
48. J. Odell, H. Parunak, B. Bauer. Extending UML for Agents. In: Odell, J., Parunak, H. and Bauer, B. (Eds.), Proc. of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, 2000.
49. 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.
50. Omicini, A.: From Objects to Agent Socities: Abstractions and Methodologies for the Engineering of Open Distributed Systems. In: Corradi, A., Omicini, A., Poggi, A. (eds.): WOA (2000) 29—34
51. Omicini, A.: SODA: Societies and Infrastructure in the Analysis and Design of Agentbased Systems. In: Ciancarini, P., Wooldridge, M. (eds.) Agent-Oriented Software Engineering. Springer-Verlag (2001) 185-194.
52. Parunak, H., Odell, J.: Representing social structures in UML. In: Proceeding of Agent-oriented Software Engineering AOSE (2001) 1-16.
53. Petrie, C.: Agent-Based Software Engineering. In: Ciancarini, P., Wooldridge, M. (eds.). Agent-Oriented Software Engineering. Springer-Verlag (2001)
54. Rumbaugh et al. Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, New Jersey (1991).

55.Rasmus, D.: Rethinking Smart Objects – Building Artificial Intelligence with Objects. Cambridge University Press (1999)
56.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(1) (2000)
57.Shoham, Y.: Agent0: A simple agent language and its interpreter. In: Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91) (1991) 704–709
58.Shoham Y.: Agent-Oriented Programming. Artificial Intelligence No 60 (1993) 24–29
59.SoC+Agents Group. Separation of Concern and Multi-Agents Systems Group. URL: http://www.teccomm.les.inf.puc-rio.br/SoCagents/
60.Tarr, P. et al. N Degrees of Separation: Multi-Dimensional Separation of Concerns In Proc. of the 21st International Conference on Software Engineering (ICSE'99), May 1999.
61.The Semantic Web Portal. Available in: http://www.semanticweb.org.
62.Wagner, G.: Agent-Object-Relationship Modeling. In: Proc. of Second International Symposium - from Agent Theory to Agent Implementation together with EMCRS 2000, April (2000)
63.Willmott, S., Dale, J., Burg, B., Charlton, C., O'brien, P. Agentcities: A Worldwide Open Agent Network. Agentlink News, November, 2001, pp. 13-15.
64.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
65.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
66.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, Lecture Notes in AI, 2001.
67.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)
68.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)
69.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. Nova Iorque: Springer-Verlag, 2001.
70.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.
71.Zambonelli, F.; Parunak, H. Signs of a Revolution in Computer Science and Software Engineering. In: Third International Workshop Engineering Societies in the Agents World, Madrid, Spain, 2002.