Modelling The Interaction Between Objects: Roles as Affordances

Matteo Baldoni[1], Guido Boella[1] and Leendert van der Torre[2]

[1]Dipartimento di Informatica. Università di Torino - Italy.
E-mail: {baldoni,guido}@di.unito.it
[2]University of Luxembourg. E-mail: leendert@vandertorre.com

Abstract. In this paper we present a new vision of objects in ontologies where the objects' attributes and operations depend on who is interacting with them. This vision is based on a new definition of the notion of role, which is inspired by the concept of affordance as developed in cognitive science. The current vision of objects considers attributes and operations as being objective and independent from the interaction. In contrast, in our model interaction with an object always passes through a role played by another object manipulating it. The advantage is that roles allow to define operations whose behavior changes depending on the role and the requirements it imposes, and to define session aware interaction, where the role maintains the state of the interaction with an object. Finally, we provide a description of the model in UML and we discuss how roles as affordances have been introduced in Java.

1 Introduction

Object orientation is a leading paradigm in knowledge representation, modelling and programming languages and, more recently, also in databases. The basic idea is that the attributes and operations of an object should be associated with it. The interaction with the object is made via the public attributes of the class it is an instance of and via its public operations, for example, as specified by an interface. The implementation of an operation is specific of the class and can access the private state of it. This allows to fulfill the data abstraction principle: the public attributes and operations are the only possibility to manipulate an object and their implementation is not visible from the other objects manipulating it; thus, the implementation can be changed without changing the interaction capabilities of the object.

This view can be likened with the way we interact with objects in the world: the same operation of switching a device on is implemented in different manners inside different kinds of devices, depending on their functioning.

The philosophy behind object orientation, however, views reality in a naive way. It rests on the assumption that the attributes and operations of objects are objective, in the sense that they are the same whatever is the object interacting with it.

This view has two consequences which limit the usefulness of object orientation in modelling knowledge:

– Every object can access all the public attributes and invoke all the public operations of every other object. Hence, it is not possible to distinguish which attributes and operations are visible for which classes of interacting objects.
– The object invoking an operation (caller) of another object (callee) is not taken into account for the execution of the method associated with the operation. Hence, when an operation is invoked it has the same meaning whatever the caller's class is.
– The values of the private and public attributes of an object are the same for all other objects interacting with it. Hence, the object has always only one state.
– The interaction with an object is session-less since the invocation of an operation does not depend on the caller. Hence, the value of private and public attributes and, consequently, the meaning of operations cannot depend on the preceding interactions with the object.

The first three limitations hinder modularity, since it would be useful to keep distinct the core behavior of an object from the different interaction possibilities that it offers to different kinds of objects. Some programming languages offer ways to give multiple implementations of interfaces, but the dependance from the caller cannot be taken into account, unless the caller is explicitly passed as a parameter of each method.

The last limitation complicates the modelling of distributed scenarios where communication follows protocols.

Programming languages like Fickle [1] address the second and third problem by means of dynamic reclassification: an object can change class dynamically, and its operations change their meaning accordingly. However, Fickle does not represent the dependence of attributes and operations from the interaction.

Sessions are considered with more attention in the agent oriented paradigm, which is based on protocols ([2, 3]). A protocol is the specification of the possible sequences of messages exchanged between two agents. Since not all sequences of messages are legal, the state of the interaction between two agents must be maintained in a session. Moreover, not all agents can interact with other ones using whatever protocol. Rather the interaction is allowed only by agents playing certain roles.

However, the notion of role in multi-agents systems is rarely related with the notion of session of interaction ([4]). Moreover, it is often related with the notion of organization rather than with the notion of interaction ([5]).

In this paper, we address the four above problems in object oriented knowledge representation by introducing a new notion of role. This is inspired by research in cognitive science, where the naive vision of objects is overcome by the so called ecological view of interaction in the environment. In this view, the properties (attributes and operations) of an object are not independent from whom is interacting with it. An object "affords" different ways of interaction to different kinds of objects.

The structure of this paper is as follows. In Section 2 we discuss the cognitive foundations of our view of objects. In Section 3 we define roles in terms of affordances and in Section 4 we explain how to describe roles in UML. In Section 6 we summarize how our approach to roles leads to the design of a new object oriented programming language, powerJava. Related work and conclusion end the paper.

2 Roles as affordances

The naive view of objects sees them as having objective attributes and operations which are independent from the observer or from other objects interacting with them. Instead, recent developments in cognitive science show that attributes and operations emerge only at the moment of the interaction and change according to what kind of object is interacting with another one:

1. Objects are conceptualized on the basis of what they "afford" to the actions of the entities interacting with them. Thus, different entities conceptualize and interact with the same object in different ways.
2. The classification of entities in taxonomies of categories is not composed by uniform levels. Rather, some levels of categories have a privileged status. In the taxonomy of natural kinds this level is the level of the genus (i.e., dog, cat, pine, oak): the likely explanation is that this is the level where the characteristic ways of interacting with the entities classified by these categories are located. At the upper level (e.g., mammal, tree) no common way of interaction is possible with all the entities of the category; while at the lower level (e.g., terrier, white oak) there is less difference in the way entities of different categories are manipulated.

Interaction, thus, is the common denominator. Since we do not consider in this paper the problem of class hierarchies, we will focus on the first aspect: "affordances".

The notion of "affordance" has been made popular by Norman [6] (p. 9):

> "The term affordance refers to the perceived and actual properties of the thing, primarily those fundamental properties that determine just how the thing could possibly be used. A chair affords ('is for') support, and, therefore, affords sitting."

This is the view in which the notion of affordance has been adopted in another branch of computer science: human-computer interaction (e.g., [7]). Seeing affordances in this way, however, does not solve the problem of the subjectivity of attributes and operations, and, indeed, it is a partial reading of the original theory of affordances. We resort here to the original vision, instead.

The notion of affordance has been developed by a cognitive scientist, James Gibson, in a completely different context, the one of visual perception [8] (p. 127):

> "The affordances of the environment are what it offers the animal, what it provides or furnishes, either for good or ill. The verb to afford is found in the dictionary, but the noun affordance is not. I have made it up. I mean by it something that refers to both the environment and the animal in a way that no existing term does. It implies the complementarity of the animal and the environment...
>
> If a terrestrial surface is nearly horizontal (instead of slanted), nearly flat (instead of convex or concave), and sufficiently extended (relative to the size of the animal) and if its substance is rigid (relative to the weight of the animal), then the surface affords support...

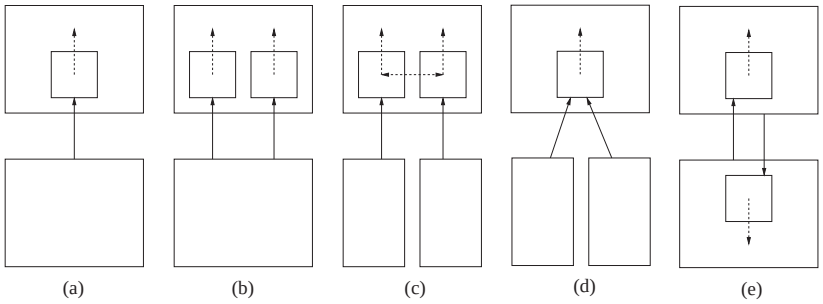

Fig. 1. The possible uses of roles as affordances.

> Note that the four properties listed - horizontal, flat, extended, and rigid - would be physical properties of a surface if they were measured with the scales and standard units used in physics. As an affordance of support for a species of animal, however, they have to be measured relative to the animal. They are unique for that animal. They are not just abstract physical properties. If so, to perceive them is to perceive what they afford. This is a radical hypothesis, for it implies that the 'values' and 'meanings' of things in the environment can be directly perceived.
>
> The activity of an observer that is afforded depends on the layout, that is, on the solid geometry of the arrangement. The same layout will have different affordances for different animals, of course, insofar as each animal has a different repertory of acts. Different animals will perceive different sets of affordances therefore. ... Animals, and children until they learn geometry, pay attention to the affordances of layout rather than the mathematics of layout.

Gibson refers to an ecological perspective, where animals and the environment are complementary. But the same vision can be transferred to objects. By "environment" we intend a set of objects and by animal of a given specie we intend another object of a given class which manipulates them. Besides physical objective properties objects have affordances when they are considered relative to an object managing them.

How can we use this vision to introduce new modelling concepts in object oriented knowledge representation? The affordances of an object are not isolated, but they are associated with a given specie. So we need to consider sets of affordances. We will call a *role type* the different sets of interaction possibilities, the affordances of an object, which depend on the class of the interactant manipulating the object: the *player* of the role. To manipulate an object it is necessary to specify the role in which the interaction is made.

But an ecological perspective cannot be satisfied by considering only occasional interactions between objects. Rather it should also be possible to consider the continuity of the interaction for each object, i.e., the state of the interaction. In terms of a distributed scenario, a session. Thus a given role type can be instantiated, depending on a certain player of a role (which must have the required properties), and the *role instance* represents the state of the interaction with that role player.

3 Roles and sessions

The idea behind affordances is that the interaction with an object does not happens directly with it by accessing its public attributes and invoking its public operations. Rather, the interaction with an object happens via a role: to invoke an operation, it is necessary first to be the player of a role offered by the object the operation belongs to. The roles which can be played depend on the properties of the player of the role (the *requirements*), since the roles represent the set of affordances offered by the object.

Thus an object can be seen as a cluster of classes gathered around a center class. The center class represents the core state and behavior of the object. The other classes, the role types, are the containers of the operations specific of the interaction with a given class, and of the attributes characterizing the state of the interaction. Not only the kind of attributes depend on the class of the interacting object, but also the values of these attributes may vary according to a specific interactant. A role instance, thus, models the session of the interaction between objects and can be used for defining protocols.

If a role represents the possibilities offered by an object to interact with it, the methods of a role must be able to affect the core state of the objects they are roles of and to access their operations; otherwise, no effect could be made by the player of the role on the object the role belongs to. So a role, even if it seems a usual object, is, instead different: it depends on the object the role belongs to and they access its state.

Many objects can play the same role as well as the same object can play different roles. In Figure 1 we depict the different possibilities. *Boxes* represent objects and role instances (included in external boxes). *Arrows* represent the relations between players and their roles, *dashed arrows* the access relation between objects.

- Drawing (a) illustrates the situation where an object interacts with another one by means of the role offered by it.
- Drawing (b) illustrates an object interacting in two different roles with another one. This situation is used when an object implements two different interfaces for interacting with it, which have methods with the same signature but with different meanings. In our model the methods of the interfaces are implemented in the roles offered by the object to interact with it. Moreover, the two role instances represent the two different states of the two interactions between the two objects.
- Drawing (c) illustrates the case of two objects which interact with each other by means of the roles of another object (which can be considered as the context of interaction). This achieves the separation of concerns between the core behavior of an object and the interaction possibilities in a given context. The meaning of this scenario for coordination has been discussed in [9].
- In drawing (d) a degenerated but still useful situation is depicted: a role does not represent the individual state of the interaction with an object, but the collective state of the interaction of two objects playing the same role instance. This scenario is useful when it is not necessary to have a session for each interaction.
- In drawing (e) two objects interact with each other, each one playing a role offered by the other. This is often the case of interaction protocols: e.g., an object can play the role of *initiator* in the Contract Net Protocol if and only if the other object plays the role of *participant* [10]. The symmetry of roles is closer to the traditional vision of roles as ends of a relation (like also in UML, see Section 7).

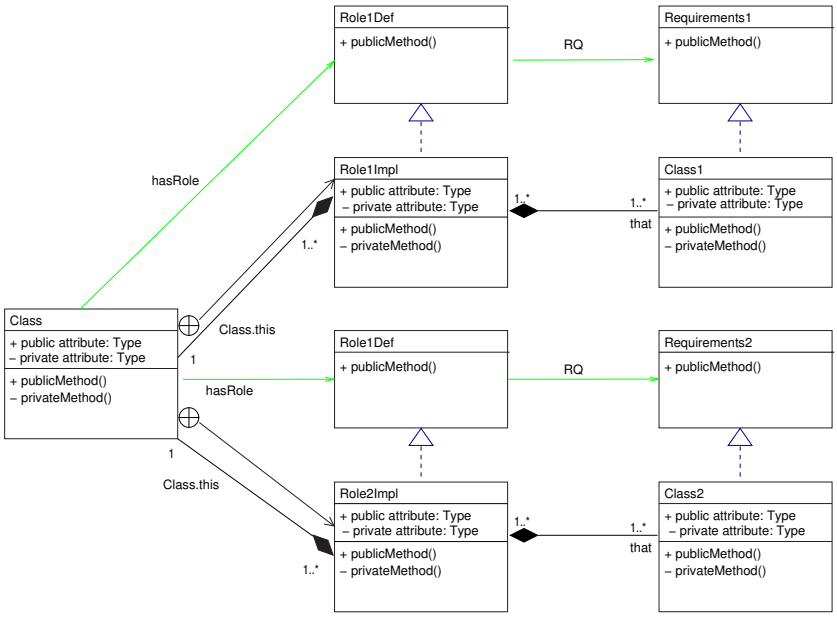

Fig. 2. Roles as affordances in UML.

4 Representing affordances in UML

Despite the conceptual difference between the traditional view of object orientation and the addition of roles as affordances, it is still possible to represent them in a object oriented modelling language like UML. So in this paper, rather than introducing new constructs in UML, we less ambitiously present how to model roles as affordances in the existing UML, to make our proposal more comprehensible.

The first problem is how to represent the roles as set of affordances of an object. Role types describe attributes and operations, so they can be modelled as classes in UML. Role instances maintain the specific values of the attributes in an interaction with the role player, so they are modelled as objects.

However, role instances are always associated with two other objects: the object of which they are roles and the object playing the role. We represent these relations by means of two composition arrows between the object and the role instance (denoted as `Class.this` in the role instance) and between the player and the role instance (denoted as `that` in the role instance). A role instance can be a role of one object only, but it can have more than one player. Instead, different role instances can have associated the same object they are role of.

Second, as discussed in Section 3, the role can access the attributes and operations of the object the role belongs to. This can be represented by saying that the namespace of the role belongs to the namespace of the class it is a role of. In UML the nested notation used in Figure 1 is not the correct way to show a class belonging to the namespace of another class. Instead, the anchor notation (a cross in a circle on the end of a line) is

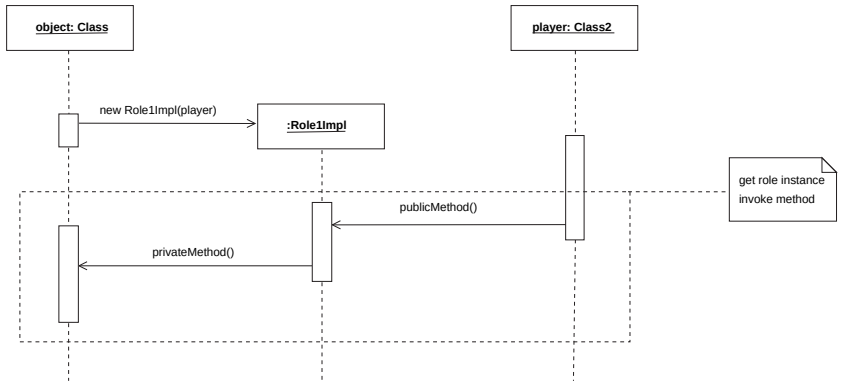

Fig. 3. The interaction with an object via a role.

used between two class boxes to show that the class with the anchor icon declares the class on the other end of the line. This is the way inner classes are denoted in UML. As we discuss in Section 6 the construct of inner classes can be used to introduce roles in object oriented programming languages.

Moreover, we have to represent the dependence of a role from the properties of the player object. As discussed in Section 3, the role represents the attributes and operations which depend on a specific kind of object playing the role: a role can be played (i.e., an object can be manipulated in a certain way) only by a specific kind of players. Thus, we need to specify the requirements for playing each role class. If we specify requirements by means of a class, we restrict the set of possible players too much. We only need a partial specification to describe what is needed to play a role. Thus requirements are specified by an interface: only the objects which are instance of a class implementing the requirements can play the role.

However, there is still one unresolved issue. The class with roles cannot be given a partial specification of its interaction possibilities by means of a single interface, since the roles associated with it may share some operations but not other ones. Thus, we associate with the class a set of role definitions, one for each role class associated with it. The role definitions specify the operations which the player of the role is endowed to invoke. A role definition differs from an interface since it has associated the requirements of the role.

In Figure 2 we represent our model. We have a class `class` with two role definitions (`hasRole` relates it to `Role1Def` and `Role2Def`) representing the set affordances offered by the object to players satisfying the requirements (the interface related by the `RQ` association). The role definitions are implemented by classes which are connected with the class `Class` by a composition relation and by a namespace association (anchor link).

In Figure 2 we consider the possibility to directly interact with the class `Class` by directly accessing its “objective” attributes and operations. However, nothing prevents that the object does not have any public attribute or operation, so that the interaction can be only made via one of its roles.

Finally, note that it is possible to have a single instance of a role implementation which is associated with multiple players: this can be used to represent the situation where no session is needed and it is sufficient to model multiple implementations of the same operation in a single class.

This means that we have three possibilities of interaction in our model:

- Traditional direct interaction with an object via its objective properties.
- Session-less interaction with an object via a role which presents to the object a state and operations different from the core object, but common with all the other objects playing that role (and which satisfy the role's requirements).
- Session aware interaction via a role instance representing the state of the interaction with a particular player of the role.

Thus, it is possible to select the option most suited for the situation to model, without necessarily having a role type or a role instance for each object interacting.

In summary, an object with affordances is represented by a core object associated with other objects of the classes representing the role implementations. Each role instance represents the state of the interaction with another object, and its class specifies which methods can be invoked by the players of that role if they satisfy the role's requirements.

What is still missing is how our model must be used. When another object wants to interact with it, it has to choose which role to play - assuming that it has the requirements to play it.

The sequence diagram in Figure 3 reports the interactions between the object that defines the role implementation of the role instance (`object:Class`) and a player of that role (`player:Class2`), via a role instance (`:Role1Impl`). The figure is relative to the class diagram described in Figure 2. The player and the object that defines the role exist independently from each other, while, the role instance is created in the context of the instance of the object that defines it (`object:Class`). A role instance, representing a set of affordances, depends both on the object that defines it, in the context of which it is created, and on its player, which is actually passed as a parameter during its creation. In other words, a role instance object represents an association relation with a independent state (the session of the interaction between the former two objects). The object player, in order to interact with the other object, should use an affordance of the last one, more precisely of the role instance that represents the interactions between them. First of all, it has to find the right role instance (`get role instance`) and then to invoke the method on the role instance. However, as a difference with a normal association relation, a role instance (a set of affordances) has access to the object that defines it. In this way, the role can effectively specify a way to interact with its defining object in terms of affordances, providing also a controlled access to its methods and state. In Figure 3, the role instance (`Role1Impl`) offers a way to access, in a controlled way, a private method through an affordance - i.e., a public method - used by the player. The player delegates the role instance for the access to the state of the other object, and, on the other hand, the role instance offers a power to access to the state of the other object.

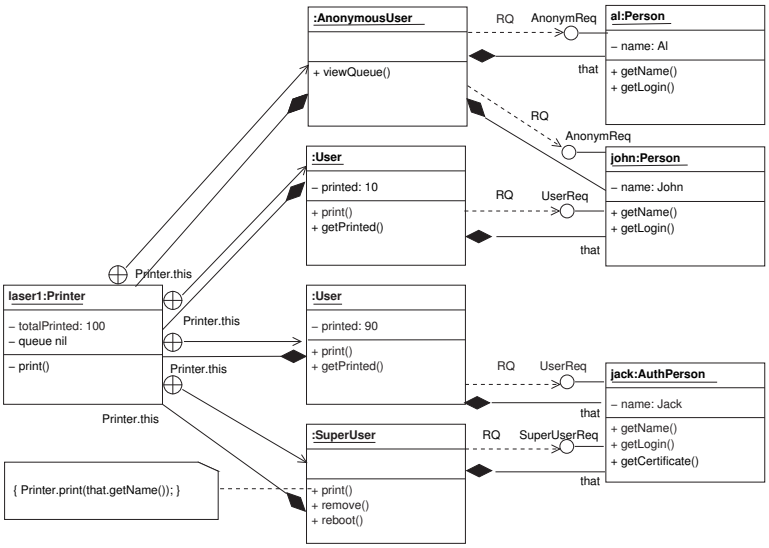

Fig. 4. Three ways of accessing a printer.

5 Example

Figure 4 represents a UML object diagram of a printer which can be used in different ways by playing different roles: `SuperUser`, `User` and `AnonymousUser`. Different requirements are needed to play these roles: `SuperUserReq`, e.g., requires the methods `getName()`, `getLogin()` and `getCertificate()`. Each role provides different operations (e.g., only a `SuperUser` can remove a job from a queue), or the same operation in different manners. E.g., the `print()` method of a `SuperUser` does not count the number of printed copies, the `User`'s updates the copy counter `printed`). The local information about the number of printed copies (`printed`) is stored in the `User` instance, since it depends on its player. The object `Printer` has no public properties. Its private operation `print()` is used by the `print()` operations of the roles `User` and `SuperUser`, which are different from it. The private attribute `queue` is accessed by the operation `viewQueue()` of the `AnonymousUser` operation. It can access the private attribute since the class `AnonymousUser` belongs to the same namespace as `Printer`.

There are four unnamed instances of the three role types. `jack`, a `AuthPerson`, plays two roles, so it is both part of an instance of `SuperUser` and of an instance of `User`. As a `User` it has different attributes than as a `SuperUser` and different operations avaliable. The role `AnonymousUser` has only one instance since it is not necessary to keep a session for each anonymous player. The same role instance is played by different objects implementing `AnonymousReq` (which requires only `getName()`).

The requirements can be used via the `that` reference, linking the role to its player. E.g., the method `print()` of a `SuperUser` calls the private `print()` operation of the `Printer`, passing as parameter the name of the player (`that.getName()`).

```
class Printer {
  private int printedTotal;

  definerole User {

    private int printed;

    public void print(){ ...
      printed = printed + pages;
      Printer.print(that.getName());
    }
  }
}
```

```
role User playedby UserReq
   { void print();
     int getPrinted(); }

interface UserReq
   { String getName();
     String getLogin();}

jack = new AuthPerson();
laser1 = new Printer();
laser1.new User(jack);
laser1.new SuperUser(jack);
((laser1.User)jack).print();
```

Fig. 5. A role `User` inside a `Printer`.

6 From modelling to programming languages

Baldoni *et al.* [11, 10] introduce roles as affordances in powerJava, an extension of the object oriented programming language Java. Java is extended with:

1. A construct defining the role with its name, the requirements and the operations.
2. The implementation of a role, inside an object and according to its definition.
3. How an object can play a role and invoke the operations of the role.

Figure 5 shows by means of the example of Section 4 the use of roles in powerJava. First of all, a role is specified as a sort of interface (`role` - right column) by indicating who can play the role (`playedby`) and which are the operations acquired by playing the role. Second (left column), a role is implemented inside an object as a sort of inner class which realizes the role specification (`definerole`). The inner class implements all the methods required by the role specification as it were an interface.

In the bottom part of the right column of Figure 5 the use of powerJava is depicted. First, the candidate player `jack` of the role is created. It implements the requirements of the roles (`AuthPerson` implements `UserReq` and `SuperUserReq`). Before the player can play the role, however, an instance of the object hosting the role must be created first (a `Printer laser1`). Once the `Printer` is created, the player `jack` can become a `User` too. Note that the `User` is created inside the `Printer laser1` (`laser1.new User(jack)`) and that the player `jack` is an argument of the constructor of role `User` of type `UserReq`. Moreover `jack` plays the role of `SuperUser`.

The player `jack` to act as a `User` must be first classified as a `User` by means of a so-called *role casting* (`((laser1.User) jack)`). Note that `jack` is not classified as a generic `User` but as a `User` of `Printer laser1`. Once `jack` is casted to its `User` role, it can exercise its powers, in this example, printing (`print()`). Such method is called a power since, in contrast with usual methods, it can access the state of other objects: namespace shares the one of the object defining the role. In the example, the method `print()` can access the private state of the `Printer` and invoke `Printer.print()`.

7 Related work

There is a huge amount of literature concerning roles in knowledge representation, programming languages, multiagent systems and databases. Thus we can compare our approach only with a limited number of other approaches.

First of all, our approach is consistent with the definition of roles in ontologies given by Masolo *et al.* [12]. They define a role as a social entity which is definitionally dependent on another entity and which is founded and antirigid. Definitionally dependent means that a concept is used in its definition. As discussed in [13], in our approach this corresponds to the stronger property that a role is defined "inside" the object it belong to (i.e., in its namespace or as an inner class). Foundation means that the existence of a role instance requires the existence of another entity. In our model a role instance requires both the existence of a player and the existence of the object the role belongs to. Antirigidity means that the role is not a permanent feature of an entity. In our model a role can cease to exist even if both its player and the object maintain their original class.

A leading approach to roles in programming languages is the one of Kristensen and Osterbye [14]. A role of an object is "a set of properties which are important for an object to be able to behave in a certain way expected by a set of other objects". Even if at first sight this definition seems related, it is the opposite of our approach. By "a role of an object" they mean the role played by an object. They say a role is an integral part of the object and at the same time other objects need to see the object in a certain restricted way by means of roles. A person can have the role of bank employee, and thus its properties are extended with the properties of employee. In our approach, instead, by a role of an object we mean the role offered by an object to interact with it by playing the role. We focus on the fact that to interact with a bank an object must play a role defined by the bank, e.g., employee, and to play a role some requirements must be satisfied. The properties of the player of the role are extended, but only in relation with the interaction with the bank.

Roles based on inner classes have been proposed also by [15, 16]. However, their aim is to model the interaction among different objects in a context, where the objects interact only via the roles they play. This was the original view of our approach [17], too. But in this paper and in [10] we extend our approach to the case of roles used to interact with a single object to express the fact that the interaction possibilities change according to the properties of the interactants.

The term of role in UML is already used and it is related to the notion of collaboration: "while a classifier is a complete description of instances, a classifier role is a description of the features required in a particular collaboration, i.e. a classifier role is a projection of, or a view of, a classifier." This notion has several problems, thus Steimann [18] proposes a revision of this concept merging it with the notion of interface. However, by role we mean something different from what is called role in UML. UML is inspired by the relation view of roles: roles come always within a relation. In this view, which is also shared by, e.g., [19, 20], roles come in pairs: buyer-seller, client-server, employer-employee, *etc.*. In contrast, we show, first, that the notion of role is more basic and involves the interaction of one object with another one using one single role, rather than an association. Second, we highlight that roles have a state and add properties to their players besides requiring the conformance to an interface.

8 Conclusion

In this paper we introduce the notion of affordance developed in cognitive science to extend the notion of object in the object orientation paradigm for knowledge modelling. In our model objects have attributes and operations which depend on the interaction with other objects, according to their properties. Sets of affordances form roles which are associated with players which satisfy the requirements associated with roles. Since roles have attributes they provide the state of the interaction with an object.

The notion of affordance has been used especially in human computer interaction. In this field the difference between Gibson's interpretation of the concept and the one proposed by Norman has been clarified for example by McGrenere and Ho [21]. In particular, they notice that a feature of Gibson's interpretation is "the offerings or action possibilities in the environment in relation to the action capabilities of an actor". However, to our knowledge the fact that affordances depend on the ability of the actor has not been exploited elsewhere.

Our model allows by means of affordances a more flexible interaction with objects, composed of the non-exclusive following alternatives:

- Traditional direct interaction with an object via its objective properties.
- Session-less interaction with an object via a role which presents to the object a state and operations different from the core object.
- Session aware interaction via a role instance representing the state of the interaction with a particular player of the role.

In this paper we describe this model in UML without extending the language. In Section 6 we summarize how this model has been used to extend Java with roles.

In [17] we present a different albeit related notion of role, with a different aim: representing the organizational structure of institutions which is composed of roles. The organization represents the context where objects interact only via the roles they play by means of the powers offered by their roles (what we call here affordances). E.g., a class representing a university offers the roles of student and professor. The role student offers the power of giving exams to players enrolled in the university.

In [11] we investigate the ontological foundations of roles, while in [9] we explain how roles can be used for coordination purposes.

In this paper, instead, we use roles to articulate the possibility of interaction provided by an object.

Future work concerns the symmetry of roles as part of a relation. In particular, the last diagram of Figure 1 deserves more attention. For example, the requirements to play a role must include the fact that the player must offer the symmetric role (e.g., initiator and participant in a negotiation). Moreover, in that diagram the two roles are independent, while they should be related. Finally, the fact that the two roles are part of a same process (e.g., a negotiation) should be represented, in the same way we represent that student and professor are part of the same institution.

References

1. Drossopoulou, S., Damiani, F., Dezani-Ciancaglini, M., Giannini, P.: More dynamic object re-classification: Fickle$_{II}$. ACM Transactions On Programming Languages and Systems **24** (2002) 153–191
2. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: an organizational view of multiagent systems. In: LNCS n. 2935: Procs. of AOSE'03, Springer Verlag (2003) 214–230
3. Juan, T., Sterling, L.: Achieving dynamic interfaces with agents concepts. In: Procs. of AAMAS'04. (2004)
4. Omicini, A., Ricci, A., Viroli, M.: An algebraic approach for modelling organisation, roles and contexts in MAS. Applicable Algebra in Engineering, Communication and Computing **16** (2005) 151–178
5. Zambonelli, F., Jennings, N., Wooldridge, M.: Developing multiagent systems: The Gaia methodology. IEEE Transactions of Software Engineering and Methodology **12(3)** (2003) 317–370
6. Norman, D.: The Design of Everyday Things. Basic Books, New York (2002)
7. Amant, R.: User interface affordances in a planning representation. Human Computer Interaction **14** (1999) 317–354
8. Gibson, J.: The Ecological Approach to Visual Perception. Lawrence Erlabum Associates, New Jersey (1979)
9. Baldoni, M., Boella, G., van der Torre, L.: Roles as a coordination construct: Introducing powerJava. Electronic Notes in Theoretical Computer Science **150** (2005)
10. Baldoni, M., Boella, G., van der Torre, L.: Bridging agent theory and object orientation: Interaction among objects. In: Procs. of PROMAS'06 workshop at AAMAS'06. (2006)
11. Baldoni, M., Boella, G., van der Torre, L.: Powerjava: ontologically founded roles in object oriented programming language. In: Procs. of OOOPS Track of SAC'06. (2006)
12. Masolo, C., Vieu, L., Bottazzi, E., Catenacci, C., Ferrario, R., Gangemi, A., Guarino, N.: Social roles and their descriptions. In: Procs. of KR'04, AAAI Press (2004) 267–277
13. Boella, G., van der Torre, L.: A foundational ontology of organizations and roles. In: Procs. of DALT'06 workshop at AAMAS'06. (2006)
14. Kristensen, B., Osterbye, K.: Roles: conceptual abstraction theory and practical language issues. Theor. Pract. Object Syst. **2** (1996) 143–160
15. Herrmann, S.: Roles in a context. In: Procs. of AAAI Fall Symposium Roles'05, AAAI Press (2005)
16. Tamai, T.: Evolvable programming based on collaboration-field and role model. In: Procs. of IWPSE'02. (2002)
17. Baldoni, M., Boella, G., van der Torre, L.: Bridging agent theory and object orientation: Importing social roles in object oriented languages. In: Procs. of PROMAS'05 workshop at AAMAS'05. (2005)
18. Steimann, F.: A radical revision of UML's role concept. In: Procs. of UML2000. (2000) 194–209
19. Masolo, C., Guizzardi, G., Vieu, L., Bottazzi, E., Ferrario, R.: Relational roles and qua-individuals. In: Procs. of AAAI Fall Symposium Roles'05, AAAI Press (2005)
20. Loebe, F.: Abstract vs. social roles - a refined top-level ontological analysis. In: Procs. of AAAI Fall Symposium Roles'05, AAAI Press (2005)
21. McGrenere, J., Ho, W.: Affordances: Clarifying and evolving a concept. In: Procs. of Graphics Interface Conference. (2000) 179–186