Towards a General Framework for Modelling Roles

Valerio Genovese

Università di Torino

Abstract. Role is a widespread concept, it is used in many areas like MAS, Programming Languages, Organizations, Security and OO modelling. Unfortunately, it seems that the literature is not actually able to give a uniform definition of roles, there exist several approaches that model roles in many different (or even opposite) ways. In this draft we start to define a meta-model for roles. Our aim is to build a formal framework through which we can describe different roles appeared in the literature or implemented in up and running computer systems. In particular we give a new definition of role's foundation introducing *sessions*, which are a formal instrument to talk about role's states and we show how sessions may be useful to model many different role's accounts.

keywords: Roles, Organizations, Object Oriented Modelling, Multi-Agent Systems, Security.

1 Introduction

The notion of role is a modelling concept strictly linked with interaction between entities. In natural language, we notice that terms like "student", "employee" or "president" are linked with a person who plays them and a *context* in which the player interact, the term "student" refers to a person that is a student in a specific university [1]. In a certain way, we can view roles as a way to model an interaction, but problems arise because it is not completely clear how many different types of interactions exist and is possible to represent in the OO paradigm.

There are many definitions of roles, each one with a plausible approach based on intuition, practical needs and, sometimes, on a formal account. In security, roles are seen as a way to distribute permissions [2], in organizational models roles gives *powers* to their players in order to access an institution, in MAS roles could be seen as descriptions of the behaviour which is expected by agents who play them [3], in ontology research roles are an anti-rigid notion founded on a player and a context [4], and many more. Even in the same field of research, there exist in the literature completely different notions of role which are in contrast with each other. Roles are not so easy to grasp, it seems that each different approach underlines a particular part of a common phenomenon not definable in a unique way.

The main goal of this work is to provide a flexible formal model for roles, which is able to grasp the basic primitives behind the different role's accounts in the literature, rather than a definition. If it is possible to define such a model,then we can study the key properties of roles in different practical implementations.

The paper is organized as follows. In section 2 we introduce the model. In section 3 we describe different roles approaches with the new formalism to show its generality, in particular we analyze social roles [4], roles as non instantiable types [5].

2 A Logical Model for Roles

Examining the literature, from an ontological point of view, we find that roles come in an universal and individual flavour, so also in our model we decide to stick to this approach in order to be able to cover a wider range of roles' definitions. To describe the model we use an OO vocabulary, because is would be easier to eventually extend our role's definition into an implementative solution. We prefer to define the formalism as much general as possible, this gives us an unconstrainted model where special constraints are added later in order to describe different approaches.

2.1 Universal Level

Definition 1 An *universal model* is a tuple

$$< \mathsf{D}, \mathsf{Contexts}, \mathsf{Players}, \mathsf{Roles}, \mathsf{Attr}, \mathsf{Op}, \mathsf{Constraints} >$$

where:

- D is a domain of classes
- $\mathsf{Contexts} \subseteq$ D is a set of institutions
- $\mathsf{Player} \subseteq$ D is a set of potential players or *actors*
- $\mathsf{Roles} \subset$ D is a finite set of *roles* $\{R_1, ..., R_n\}$
- Attr is a set of *attributes*
- Op is a set of *operations*
- $\mathsf{Constraints}$ is a set of *Constraints*

The static model has also a few functions and relations:

- PL $\subseteq \mathsf{Players}$ x Roles: this relation states, at the universal level, which are the players that can play a certain role.
- RO $\subseteq \mathsf{Roles}$ x $\mathsf{Contexts}$: each role is linked with one or more contexts.
- AS $\subseteq \mathsf{D}$ x Attr: is an attribute assignment relationship, through which we can assign to each class its attributes.
- OS $\subseteq \mathsf{D}$ x Op: is an operation assignment relationship, through which we can assign to each class its operations.
- RH $\subseteq \mathsf{Roles}$ x Roles is a partial order relationship called role hierarchy, also written as $\geq_{RH}$. If $r <_{RH} r^{'}$, we say that r inherits all Attr and Op which belong to $r^{'}$.

- PH $\subseteq$ Players x Players is a partial order relationship called player hierarchy, also written as $\geq_{PH}$. If $p <_{PH} p^{'}$, we say that p inherits all Attr and Op which belong to $p^{'}$.
- CH $\subseteq$ Contexts x Contexts is a partial order relationship called context hierarchy, also written as $\geq_{CH}$. Is $c <_{CH} c^{'}$, we say that c inherits from $c^{'}$.

At this point we can add into Constraints some logical rules in order to model different role notion. For example in powerJava each role is linked with one and only one context [6], so we can express this through the following constraint:

$$\forall x, y, z(x \in \mathsf{Roles}\ y, z \in \mathsf{Contexts}\ x\mathsf{RO}y \wedge x\mathsf{RO}z \rightarrow y = z) \tag{1}$$

2.2 Individual level

Definition 1 A *snapshot model* is a tuple

$$< \mathsf{O}, \mathsf{Institutions}, \mathsf{Actors}, \mathsf{R\_Instances}, \mathsf{Sessions}, \mathsf{Attr}, \mathsf{Op}, \mathsf{Val} >$$

where:

- O is a *domain* of objects which instantiate classes in D.
- Institutions $\subseteq$ O is a set of institution which instantiate classes in Contexts.
- Actors $\subseteq$ O is a set of (potential) actors, which instantiate universals in Players.
- R_Instances $\subset$ O is a set of *roles instances*.
- Sessions is a set of *sessions*, which keep the state of an interaction between actors and institutions.
- Attr is a set of *attributes*
- Op is a set of *operations*
- Val is a set of *values*

The snapshot model has also a few functions:

- I_{Roles} is a *role assignment* that assigns to each role R a relation on Institutions x Actors x Sessions x R_Instances.
- π_{Attr} is a *projection function* that assigns to each role R a subset from Attr, which are the attributes of role R assigned by the relationship AS at the universal level.
- π_{Op} is a *projection function* that assigns to each role R a subset from Op, which are the operations of role R assigned by the relationship OS at the universal level.
- I_{Attr} is an *assignment function* which it takes as arguments an object P $\in$ O, and an attribute p $\in \pi_{Attr}(R)$, if p has a value v $\in$ Val it returns it, $\emptyset$ otherwise.

When an object x is the individual of the universal y, we say that x instantiates y and, in order to express this in a formal way, we write $x :: y$. Intuitively, a snapshot represents the state of a system at a certain particular point in time.

The role assignment function I_{Roles} gives us the *role instances*: if *i::x* is an institution, *a::y* an actor, s a session, and o*::R* a role, such that $(R, a) \in \mathsf{RO}$ and $(y, R) \in \mathsf{PL}$, the fact (*i,a,s,o*)$\in I_{Roles}(R)$ is to be read as: "the object o represents agent a playing the role R in institution i in session s". We will often write R(i,a,s,o) for this statement, and we call o the role instance. When it is not interesting to talk about sessions we can write R(i,a,o).

Suppose we have a role employee, and that the value of the attribute salary is 1000 € usually, instead of writing $I_{Attr}(\mathsf{employee}, \mathsf{salary}) = 1000$, we write

$$\mathsf{salary}(\mathsf{employee}) = 1000$$

We have used terms like institutions and actors from the literature on roles in organizations, but these terms must be taken rather broadly.Institutions suggests organizations like governments and banks, and we indeed have such applications in mind, with actors being people holding certain positions within such institutions. But the model is intended to capture a much wider range of phenomenons: institution may be folders in a file system or any object structured in roles , and actors its users, operations or attributes their permissions, and roles a way of organizing these permissions. Or even further away from the metaphor, an institution may be a relation (such as 'love') in an ER model, with roles of *lover* and *lovee* filled by actors.

We have tried to formulate the present definition in a way that is a compromise between simplicity and generality, which allows us to focus on facets of the model that are specific of roles without being hindered too much by formal details. The way we defined a snapshot leaves a lot of room for formulating further constraints that may or may not be reasonable to assume, depending on the particular role's definition we have in mind. Here are a number:

1. *Identity of role instances.* Should a role played by an actor be seen as an object per se, i.e. as a "qua-individual", or the fact that an actor plays a role simply extend or change the properties of the actor itself? The choice translates in a constraint on the roles in Roles. If we see qua-individuals as objects per se, this corresponds to the constraint that:

   $$R(i, a, s, x) \rightarrow x \neq a$$

   which is valid for powerJava [6], but also for social roles [4]. The opposite of this constraint is that roles simply change the objects themselves - qua individuals as such do not exist:

   $$R(i, a, s, x) \rightarrow x = a$$

   which is the natural option in an RBAC model, for example, in such a case we can write simply R(i,a,s) because,as already said, I_{Roles} maps R to Institutions x Actors x Sessions.
2. *Combinations of Roles.* In powerJava, each actor can play a role at most one time within a single institution, i.e.

   $$R(i, a, x) \wedge R(i, a, y) \rightarrow x = y$$

In this assumption that allows for the use of 'role casting' in powerJava to refer to role instances: an expression of the form *(i.R)a* can be used to denote the unique object x such that *R(i,a,x)*.
Variants on these constraint can be formulated as well.
If an actor can play at most one role within an institution translates to the fact that for each R $\neq R^{'}$:

$$R(i,a,x) \rightarrow \neg R^{'}(i,a,x)$$

3. *Inheritance of attributes*. In the model, both roles and objects have properties. A natural constraint is that role-instances all the properties that are defined for that role:

$$R(i,a,x) \rightarrow (\mathsf{attr} \in \pi_{Attr}(R) \rightarrow \exists v : \mathsf{attr}(\mathsf{x}) = v)$$

With respect to the question if the role-player should 'inherit' all the properties of the original player object there are different possible answers.
For example, in powerJava, no such inheritance is assumed at all - the properties of the role instance are precisely those of the role, and we have that:

$$R(i,a,x) \rightarrow (\mathsf{attr} \in \pi_{Attr}(R) \leftrightarrow \exists v : \mathsf{attr}(\mathsf{x}) = v)$$

But other options are possible as well. For example, one alternative view is that roles can be best seen as 'views' on a certain object, providing only a *subset* of the properties of the original object. A constraint which reflects that view has it that the role-player only has the properties that are defined for the original object as well as for the role:

$$R(i,a,x) \rightarrow (\mathsf{attr}(x) = v \leftrightarrow (\mathsf{attr}(R) \wedge \mathsf{attr}(a) = v))$$

The opposite view is that roles *add* properties to the players. For example, in the Zope security model (like also in RBAC) we have the following:

$$R(i,a,x) \rightarrow (\mathsf{attr}(x) = v \leftrightarrow (\mathsf{attr}(R) \vee \mathsf{attr}(a) = v))$$

4. *Dependence of roles on institutions*. In our model it is presupposed that the identity of a role instance depends not only on the role and the actor involved, but on an 'institution' as well. This is often, but not always, appropriate. We can mimic the case were the introduction on institutions is unnecessary with the introduction of a 'trivial' institution, and let Institutions contains only this trivial institution, as we do in section 3 when we model RBAC [2].
5. *Context coherence*. From an organizational point of view, there cannot be a student role's player without a teacher one, also it wouldn't be sensible to talk about the context family without someone who plays the role of husband and another one being the wife. To express this constraint we can state, for example, the following integrity rule:

$$\exists y :: Family \leftrightarrow husband(y,x,o) \wedge wife(y,z,p)$$

Which means that in the snapshot exists $y \in Institutions$ if and only if there exist two role instances o_1 and o_2 which represent respectively an *husband* and a *wife* played by actors x and z in y.

2.3 The dynamic model

There are two kinds of ways in which our model can change. In the next section, we turn to the question of changing the properties of objects in the model. But first we look at the ways that roles can be added and deleted. For simplicity, in this section we always write the role instance as *R(i,a,o)*, without directly talk about sessions.

Definition 1 A *dynamic model* is a tuple

$$< \mathsf{S}, \mathsf{Actions}, \mathsf{Requirements}, \mathsf{I}_{\mathsf{Actions}}, \mathsf{I}_{\mathsf{Roles_t}}, \pi_{\mathsf{Req}}(\mathsf{t}), \mathsf{I}_{\mathsf{Requirements_t}}, \mathsf{I}_{\mathsf{RClosure}} >$$

where:

- S is a set of *snapshots*.
- $\mathsf{Actions}$ is a set of actions.
- $\mathsf{Requirements}$ is a set of requirements.
- $\mathsf{I}_{\mathsf{Actions}}$ maps each action from $\mathsf{Actions}$ to a function on S. $I_{Actions}(a)$ tells us how a snapshot changes as a result of executing action a. If $\mathsf{Sessions} \neq \emptyset$ the change takes place if and only if all role instances in the resulting snapshot are coherent with their corresponding sessions (for a complete discussion about sessions see section 2.4).
- About I_{Roles_t} we say that $R_t(i,a,o)$ is true if there exists, at a time t, the *role instance* $R(i,a,o)$.
- $\pi_{Req}(t,R)$ returns a subset of $\mathsf{Requirements}$ present at a given time t for the role R, which are the requirements that must be fulfilled in order to create the R's role instance.
- $I_{Requirements_t}$ is a function that, given (i,a,R,t) returns True if the actor a fills the requirement in $\pi_{Req}(t,R)$ to play the role R in the institution i, False otherwise. We often write $Req_t(i,a,R)$.
- $I_{RClosure}(a,t)$ given an actor a it returns all its roles played by a at time t.
- $I_{RPlayers}(R,t)$ given a role R it returns all its players at time t.

Intuitively, the snapshots in S represent the state of a system at a certain time. We suppose that, for every time t, given an object p we can always say if it exist or not via the $?_t$ operator, so that $?_t(p)$ is true, iff p exists at time t, false otherwise.

A particular case of a dynamic model is something that we can call somewhat unelegantly a *role addition-deletion model.* It has actions corresponding to role assignment for each R, i and a, which are supposed to capture the effect of adding the role R within institution i to actor a, and actions that represent the taking away from a the role R in institution i.

Of course, these actions will not be arbitrary. We first identify a number of minimal properties that the action of role assignment need to satisfy.

Definition 1 ***(role assignment)*** *let M be a snapshot.*

$$< \mathsf{O}, \mathsf{Institutions}, \mathsf{Actors}, \mathsf{Roles}, \mathsf{Attr}, \mathsf{Op}, \mathsf{Val} >$$

Let $i \in \mathsf{Institutions}$, $a \in \mathsf{Actors}$, and $R \in \mathsf{Roles}$. There are two possibilities, if we want to assign role R to actor a: either if fails, or it succeeds. In the latter case, the resulting snapshot:

$$\mathsf{M}' = < \mathsf{O}', \mathsf{Institutions}', \mathsf{Actors}', \mathsf{Roles}', \mathsf{Attr}', \mathsf{Op}', \mathsf{Val}' >$$

should satisfy the following properties:

- A role assignment may add at most one new object to the domain (namely the newly introduced qua-individual). $O' = O \cup \{o\}$, where o may or may not already be in O.
- $\mathsf{Institutions}' = \mathsf{Institution}$ or $\mathsf{Institutions}' = \mathsf{Institution} \cup \{\mathsf{o}\}$.
- $\mathsf{Actors}' = \mathsf{Actors}$ or $\mathsf{Actors}' = \mathsf{Actors} \cup \{\mathsf{o}\}$.
- $\mathsf{Roles}' = \mathsf{Roles}$, $\mathsf{Attr}' = \mathsf{Attr}$, $\mathsf{Val}' = \mathsf{Val}$, $Op' = Op$. The sets of roles, attributes, operations and their possible values do not change.
- I'_{Roles} is just like I_{Roles}, except that $I'_{Roles}(R) = I_{Roles} \cup \{(i, a, o)\}$
- I'_{Attr} is just like I'_{Attr} with respect to the properties of objects different from i, a, and o.

For role addition and deletion we use, respectively $Req_t(i, a, R), R, i \hookrightarrow a$, and $Req_t(i, a, R), R, i \hookleftarrow a$. Then using the notation of dynamic logic we write:

$$[Req_t(i, a, R)?; R, i \hookrightarrow a]\phi$$

to express that, if actor a fills the requirements at time t ($Req_t(i, a, R)$ is True), after assigning role R within institution i, ϕ is true. If there are no particular Requirements (i.e. $\pi_{Req}(t, R) \in \emptyset$) we can omit Req_t. The above definition gives us the possibility to model that a role assignment introduces a role instance:

$$[R, i \hookrightarrow a]\exists x R(i, a, x)$$

or the fact that if a does not play the role R within institution i, then the role assignment introduces exactly one role instance:

$$(\neg \exists x R(i, a, x)) \to [R, i \hookrightarrow a]\exists ! x R(i, a, x)$$

And many more.

Definition 1 ***(object deletion)*** An object does not exist after deleting it*:*

$$[\mathsf{delete}(\mathsf{o})]\neg \mathsf{exists}(\mathsf{o})$$

If we delete a role-instance, then we also delete the role from the actor, and similarly for institutions and actors:

$$[\mathsf{delete}(\mathsf{i})]\neg \mathsf{R}(\mathsf{i}, \mathsf{a}, \mathsf{x})$$

$$[\mathsf{delete}(\mathsf{a})]\neg \mathsf{R}(\mathsf{i}, \mathsf{a}, \mathsf{x})$$

$$[\mathsf{delete}(\mathsf{x})]\neg \mathsf{R}(\mathsf{i}, \mathsf{a}, \mathsf{x})$$

For example, Depke *et al.* [7], "A role (instance) will be deleted when the agent is destroyed, i.e., its lifetime is dependent on that of the base agent.":

$$R(i, a, x) \rightarrow [delete(a)]\neg \mathsf{exists}(\mathsf{x})$$

We might also want to say something about: if an agent has certain properties *because he plays this role,* then upon object deletion, also all properties associated with that role must be removed from the actor.

Definition 1 ***(role deletion)***

$$< \mathsf{O}, \mathsf{Institutions}, \mathsf{Actors}, \mathsf{Roles}, \mathsf{Attr}, \mathsf{Op}, \mathsf{Val} >$$

Let $i \in \mathsf{Institutions}$, $a \in \mathsf{Actors}$, and $R \in \mathsf{Roles}$. Role deletion has different consequences depending on if the role instances have their own identity or not. In the latter case role deletion could be defined in the following way:

$$[R, i \hookleftarrow a] \equiv [delete(x)]$$

where x is the unique role instance linked with the institution i and played by a.

The second, and more subtle case needs to be taken into account when:

$$R(i, a, x) \rightarrow a = x$$

In such a case, we cannot simply remove the role instance x because this would mean to delete the actor once he stops playing role R. We know that when an object plays a role that has no identity it directly acquires new properties, properties that in our model are expressed through attr and Op. The constraint that represent such type of inheritance is, (as already mentioned in section 2.2):

$$R(i, a, x) \rightarrow (\mathsf{attr}(x) = v \leftrightarrow (\mathsf{attr}(R) \vee \mathsf{attr}(a) = v))$$

the same holds for Op. A way to formalize the fact that an actor relinquishes a role without an identity is:

$$[R, i \hookleftarrow a](\pi_{Attr}(a) \cap \pi_{Attr}(R) = \emptyset \wedge \pi_{Op}(a) \cap \pi_{Op}(R) = \emptyset$$

The above formula expresses that an actor who stops playing a role loses all the Attr and Op acquired by the role R.

Methods There are other ways to change the model as well - objects may assign new values to their attributes. Again, the effects of such changes may depend on choices made earlier (e.g. in the case of delegation, changing the attribute value of an object may change the value of that attribute also in some roles he plays)

Here, we will focus on the case in which the attribute-values can be changed by the *objects themselves*. What we will do is to define *methods* of objects with which they can change attributes of their own or those of others. Actually, to simplify the model, we define one single primitive: $\mathsf{set}(\mathsf{o}_1, \mathsf{o}_2, \mathsf{attr}, \mathsf{v})$, which means that o_1 sets the value of attr on o_2 to v.

Now, we will of course have that:

$$[\mathsf{set}(\mathsf{o}_1, \mathsf{o}_2, \mathsf{attr}, \mathsf{v})]\mathsf{attr}(\mathsf{o}_2) = \mathsf{v}$$

which means that if the action of setting this attribute succeeds, then the relevant object will indeed have this value for that attribute.

Now, one interesting question is how to define constraints on attributes access.

Power One observation of the work of Boella and Torre [8] is that certain aspects of the notion of *power* can be captured by how features of one agent can be changed be the actions of another, this approach promote what in software engineering is called modularity. In the present terminology, an object has *power* over another object if that object can change the values of attributes of other object. Or, formally, o_1 has power over o_2 if and only if:

$$\langle \mathsf{set}(\mathsf{o}_1, \mathsf{o}_2, \mathsf{attr}, \mathsf{v}) \rangle \top$$

It is important to underline that o_1 can have power over o_2 in three situations:

$$R(o_1, x, o_2) \vee (R(i, x, o_2) \wedge R(i, x, o_1)) \vee R(o_2, x, o_1) \vee o_1 = o_2$$

so o_1 and o_2 can be role instances or institutions.

In the work of Boella at al. [8], roles are seen as a way of organizing and assigning such powers. This idea is in particular realized in powerJava, in which the powers of players and role-instances are formally restricted by both the Java compiler as well as by the way that roles are represented in powerJava. Clearly objects can call the public methods of other objects, and thereby, possibly, change some of the attributes of an object. Roles add one extra dimension to that: linking a role to a player within an institution may give to the role instance access to methods that can change features of the institution over and above those that we given by the original model. In other words, role instances have powers over the institution within which the role is played.

2.4 Sessions

We explicitly introduce the concept of session because we argue that is strictly linked with the role's notion. As already said, we talk about sessions when is possible to keep the state of an interaction between entities. Sessions in our model are a set of objects Depending on what we want to model, we can look at sessions from at least three different points of view:

1. A session can collapse into one role instance.
2. A session can collapse into the actor (as we will see in Section 3.3).
3. A session can have its own identity and can link different role instances.

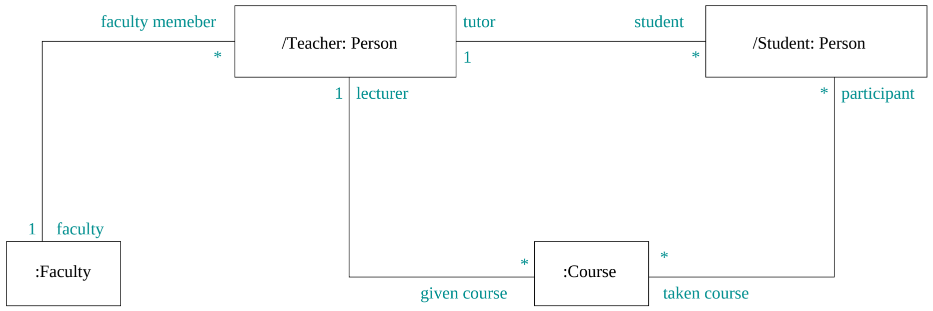
In powerJava the state of the interaction between a player and an institution is kept by the role instance:

$$R(i, a, s, x) \rightarrow s = x$$

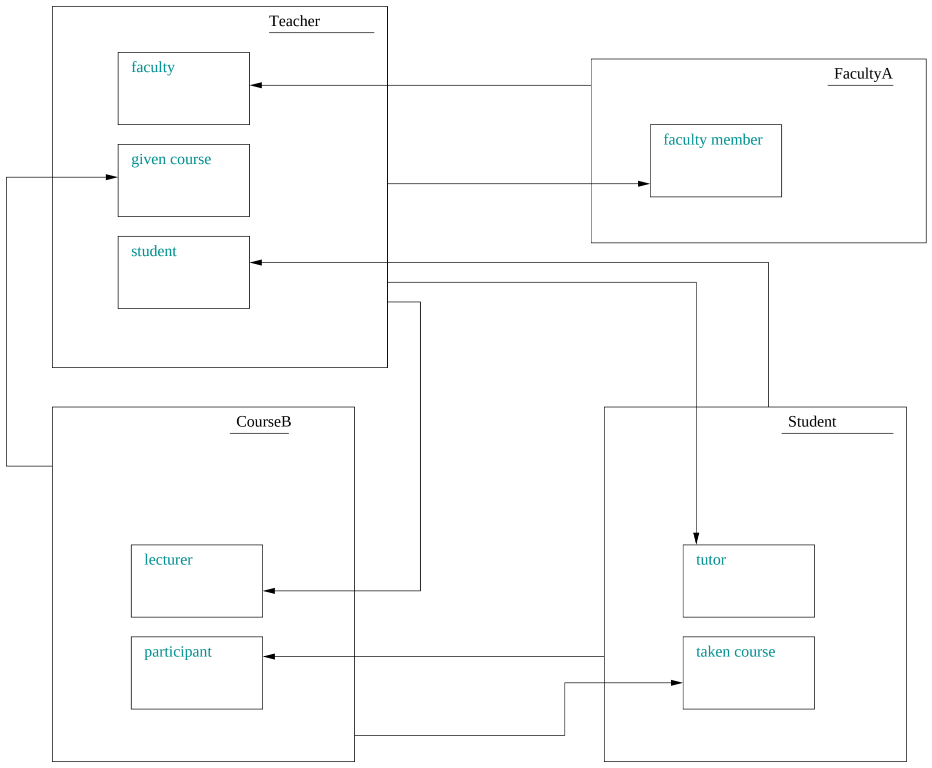
In such a case we represent the role instance as R(i,a,x) because the session collapses into the role instance. The second approach is taken into account in Section 3.3 where we try to model RBAC.

The third definition gives the possibility to unify the state of the interaction between different roles instances which participate in the same relationship or which are part of the same organizational model.

We start with an example to show how roles can be linked in a relationship:



In UML, roles serve two purposes: they label association ends, and they act as type specifiers in the scope of a collaborations (so-called classifier roles) [5]. We can translate this UML diagram into our model modelling the labels of association ends as roles and the classifier role as the object which plays the role instances to engage the relationship :



Where, for example, Teacher, which instantiate the class /Teacher:Person, is an Institution because it offers roles, but is also an actor because it plays roles which belong to other external institution.

Quoting Steimann [5] we can state that, for example:"…the role of a teacher unifies the roles of a faculty member, a lecturer and a tutor…". The problem here is that the three role are scattered between different objects (because they label associations ends) and this make difficult to link one role instance with another played by the same actor. For instance, suppose that *faculty member* and *tutor* have an attribute num_courses which counts the number of courses held by the *Teacher*, if *Teacher* stops playing *lecturer* in CourseB, num_courses in both *faculty member* and *tutor* should be decreased by one. There could also be a case where an action carried out as *tutor* can modify a *lecturer's* attribute.

In general, when the state of a role instances x does not depend only on the player and the institution, but also on other roles y and z, we say that x, y and z share the same session s, in other words:

$$R(i, a, s, x) \wedge R(i, a, s', y) \wedge R(i, a, s'', z) \rightarrow s = s' = s''$$

Giving a session s is possible to define a set of integrity rules that each role instance, participating in the session, have to respect. Referring to the above example, we can state that all role instances linked with Teacher have to share the same session:

$$\forall x, y, z\ \exists! s : faculty\_member(FacultyA, Teacher, s, x) \wedge$$

$$tutor(Student, Teacher, s, y) \wedge$$

$$lecturer(CourseB, Teacher, s, z)$$

Then we can define the following integrity rule associate with s:

$$\forall z, p, q\ \ :$$

$$p :: Faculty\ \wedge\ q :: /Student : Person\ \wedge z :: /Teacher : Person \wedge$$

$$faculty\_member(p, z, s, x) \wedge$$

$$tutor(q, z, s, y)$$

$$\rightarrow$$

$$\mathsf{num\_courses(x)} = \mathsf{num\_courses(y)} = \alpha$$

Where α is the number of Teacher instances played by z.

3 Different role's accounts

3.1 Social roles

This model is able to describe portion of the Social role's properties identified by Masolo *et al.* [4]

The key features of social roles

1. **Roles are 'properties'**: Quoting the referred article: "... *different* entities can *play* the *same* role". In order to link this sentence with our model we need to specify at which level we are reasoning, the sentence should be interpreted as [1]: "Different player *Universal* can play the same role *Universal*". In our model this is represented as:

$$\{Human, Frog\} \subseteq \mathsf{Players}$$

$$Fantasy\_Village \in \mathsf{Contexts}$$

$$Prince \in \mathsf{Roles}$$

$$\mathsf{RO} = (< Prince, Fantasy\_Village >)$$

$$\mathsf{PL} = (< Human, prince >, < Frog, Prince >)$$

2. **Roles are anti-rigid and they have 'dynamic' properties**:

[1] for an analysis at the Individual level see *"A role can be played by different entities, simultaneously"*

- *An entity can change role*: At individual level an actor can delete the role instance R(i,a,o), and play another role in place of it.
- *An entity can play the same role several times, simultaneously*: In our model an actor a who plays a role r in an institution i can have assigned only one *role instance*. However it is not clear what does it means to play the same role several times simultaneously, Masolo *et al.* conjecture that an actor can play simultaneously two different specific roles which are all specializations of a more general one. This point can be modelled in our formalism using role hierarchies, with sessions is also possible have a player which plays two role instances of the same role class R simultaneously.
- *A role can be played by different entities, simultaneously*: This sentence can be considered from two different angles: "Two player individual (Mario,Tom) can play the same role (employee) in an institution (bank)" in such a case we have two role instances $employee(bank, Mario, x)$ and $employee(bank, Tom, y)$ where $x \neq y$.
- *A role can be played by different entities, at different times*: The same role instance cannot be played by different entities, but we can have two different times $t^{'} \leq t$ in which:

$$(R_t(i, a, o) \wedge \neg R_t(i, b, c)) = true$$

$$(\neg R_{t'}(i, a, o) \wedge R_{t'}(i, b, c)) = true$$

3. **Roles have a relational nature**: "In other words we define the term role as a *founded* concept. In general, we say that α is founded on a property β if, necessarily, any *definition* of α ineliminably involves β, which is external to α". In our model, the role class R is definitionally dependent on another entity C if RO relation has a couple $< R, C >$ where C is a context. If we want to represent that *all* roles are founded on a context:

$$R \in \mathsf{Roles} \leftrightarrow \exists! C \in \mathsf{Contexts} : < R, C > \in \mathsf{RO}$$

4. **Roles are linked to contexts**: As already said above, the same happens in our model.

The key-properties for an entity to be a *role* are *anti-rigidity* and *foundation*. Foundation, as already mentioned, is an intrinsic property of our role model (think about *role instance*), the same holds for anti-rigidity, hence an object a playing a role R maintains its identity even after the role instance $R(i, a, o)$ ceases to exists. In other words we can represent the following *integrity rule*:

$$AR(R) = \forall a, o, t(R_t(i, a, o) \rightarrow \exists t^{'}(?_{t'}(a) \wedge \neg R_{t'}(i, a, o)))$$

Where AR stands for anti-rigidity.

3.2 Steimann's approach

In *object-oriented* and *conceptual modelling*, the representation of roles needs to take into account various modelling issues: multiple and dynamic classification, multiple inheritance, objects changing their attributes and behaviours, etc. Steimann introduces roles as 'first-class citizens' underlining the weaknesses which arise from others modelling approaches. To formalise his approach he defines a role-oriented modelling language called LODWICK [5].

In LODWICK roles are a kind of relationship's placeholder and playing a role for an actor means to *fill* that role in a relationship (i.e., to join the relationship taking the place held by the role filled). We already showed in Section 2.2.1 how we can simulate the idea of roles as placeholders in relationships, thanks to the fact that a role is strictly linked with a context and a player.

Here we would like to analyse how Steimann evaluates the adequacy of Lodwick's role concept, and then show how his approach could be modelled in our logical role's account. To do this several role's features are taken from different works in literature by Steimann and then discussed from the LODWICK point of view, it is interesting to notice that our model is able to describe all of them, even when they are in contradiction. We list all the features and to quote the replies that Steimann gives comparing LODWICK with them.

1. *A role comes with its own properties and behaviour*: "Yes. Roles are types, only that they cannot be instantiated. However, since the absolute properties of a role are inherited to the types filling them, they influence the properties of the instances playing them."
   This sentence can be translated in the following way:
   $$R(i,a,x) \rightarrow a = x$$
   $$R(i,a,x) \rightarrow (\forall \mathsf{attr} \in \pi_{Attr}(R) \rightarrow \exists v : \mathsf{attr}(\mathsf{a}) = v)$$
   Where the first predicate state that roles have no identity, and the second one express the fact that the properties of R influence a. In our formalism is also possible to model the case where roles are types but they can be instantiated:
   $$R(i,a,o) \rightarrow a \neq o$$
   in that case a interacts through o with i, and the property of the role instance are [2]:
   $$R(i,a,x) \rightarrow (\mathsf{attr}(x) = v \leftrightarrow (\mathsf{attr}(R) \vee \mathsf{attr}(a) = v))$$
2. *Roles depend on relationships:* "Yes. Roles occupy the places of relationships, and the relative part of a role's intension captures which relationships an object must participate in to be considered playing the role."
   Also in our model roles can be strictly linked with relationships, the fact that playing a certain role causes the player to be engaged in a relationship is implicit in our account, because the role is a link between two entities which

[2] The same holds for Op.

let the actor interact with the institution. Informally, we can say that the role instance implicitly defines a one way association ($\mathsf{actor} \rightarrow \mathsf{institution}$). It is also possible to model a situation where playing a role means to engage in a two way relationship, for example in the following situation:

$$Man, Woman \subseteq \mathsf{Players}$$

$$Man, Woman \subseteq \mathsf{Contexts}$$

$$husband, wife \in \mathsf{Roles}$$

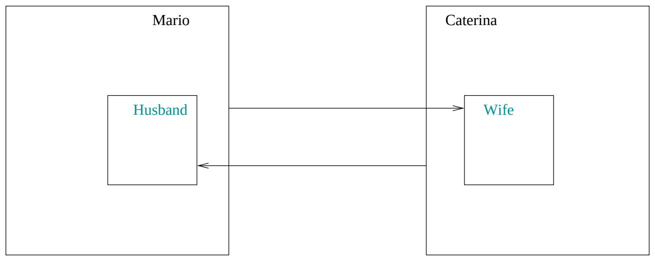
$$RO = (< husband, Woman >, < wife, Man >)$$

$$PL = (< Man, husband >, < Woman, wife >)$$

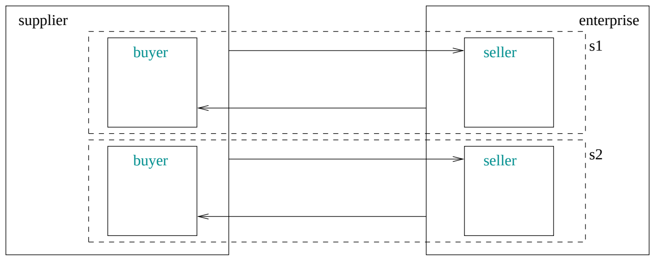
It would be sensible to impose that if *Mario::Man* plays the role husband, also *Caterina::Woman* plays the role wife with *Mario*, in other words:

$$husband(Caterina, Mario, x) \leftrightarrow wife(Mario, Caterina, y)$$

This relationship could be depicted in the following way:



Where we can see that *Mario* interacts with *Caterina* through the role instance *Husband* and complementary, *Caterina* interacts with *Mario* being his *Wife*. Another way to force the engagement in a two way relationship is through the context coherence, as already mentioned in Section 2.2.
With sessions we can explicitly link two role instances, in this way is also possible to model the following representation:



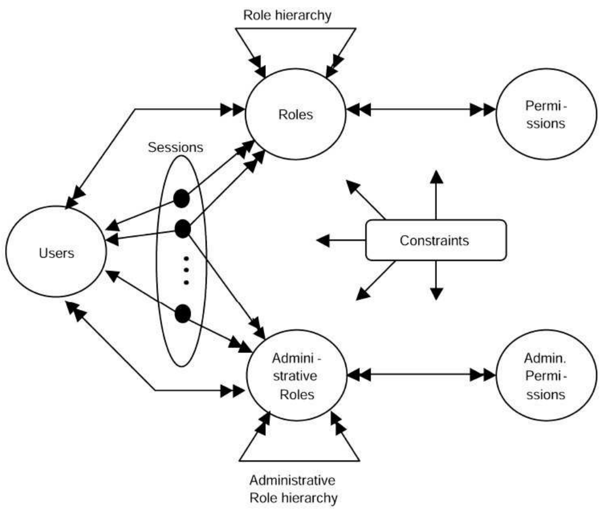
where a customer sells products to an enterprise, in one interaction the enterprise buys products for the IT department in *s1*, in the other for the HR division in *s2*. The customer has different accounts with the two departments, with the HR it sells discounted products, with the IT it sells at standard price. It is fundamental to notice that buyer in *s1* and buyer in *s2* are both instances of a common role class *Buyer*, the same happens between *seller* and a role class *Seller*. Thanks to sessions *s1* and *s2*, each one linking two role instances, it is possible to model this complex interaction.

3. *An object may play different roles simultaneously*: "Yes. An object may occur in as many different roles within the same or different associations as allowed by the relationships' specifications."
   In our model this situation could be easily expressed in the following:
   $$R(i,a,x) \wedge R'(j,a,y) \wedge x \neq y \wedge i \neq j$$
4. *An object may play the same role several time, simultaneously:* "Yes. An object may occur in the same role within different associations of the same or different relationships, as allowed by the relationship specifications."
   In our model the same role can be played several time in different institution so that:
   $$R(i,a,x) \wedge R(j,a,y) \wedge i \neq j$$
5. *An object may acquire and abandon roles dynamically*: " Yes. Roles are assumed by an object as associations with that object are added, and relinquised as associations are removed from the dynamic extensions of relationships."
   This is the same as in our model, for a complete discussion we refer to Section 2.3 where we define the role deletion.
6. *The sequence in which roles may be acquired and relinquished can be subject to restrictions*: "Possible. The specification of sequences lies in the responsability of the dynamic model."
   This is quite a subtle subject, but we can handle it exploiting the Requirements set. Suppose the we are in a Office and that the actor Leonard wants to become Director, one requirement could be that, in order to become Director you first need to be an employee, in our model suppose that $\pi_{Req}(t, Director)$ contains the following logical constraint:
   $$[director, bank \hookrightarrow leonard]\, \exists x\, director(bank, leonard, x)$$
   $$\rightarrow$$
   $$employee_t(bank, leonard, o)$$
7. *Objects of unrelated type can play the same role*: "Yes. This is one of the cornerstones of Lodwick's role formalizations; it follows from the definition of the role-filler relations linking the type and the role hierarchy."
   This point can also be easly expressed through the PL relation where we can put different universals in relation with the same role.
8. *Roles can play roles*: "No. This is not possible, since roles have no instances of their own."
   Albeit in our model we can express such a possibility, we can let $\mathsf{Players} \cap \mathsf{Roles} = \emptyset$ in order to be consistent with Lodwick model.

9. *A role can be transferred from one object to another*: "Possible. This however would require the introduction of variables, which would be an extension to Lodwick."
   Our model has its roots in roles' foundation, in fact a (instance of) role must always be associated with an instance of the institution it belongs to, besides being associated with an instance of its player. So it is impossible to transfer a role from one object to another, what we can do is to let a different role instance x played by actor a in session s have the same state of another one z played by b in the same session, such as the state of x is copied into z , this could be interpreted as a dummy role transfer.
10. *The state of an object can be role-specific*: "Partly. The associations an object participates in contribute to its state. These associations can be extended to capture the state that is associated with the object as playing the role. For example, the different salaries of a person in different employee roles may be included in the *employ* relationship."
    Our approach can model two substantially different situations, in the case that roles instances have not their own identity it is clear that the state of the actor is directly changed by the fact of playing a role R, because it acquires new operations and attributes. On the other side a role instance can come with its own identity, in this approach we can say that the state of the object in the interaction with other entities, is also composed by all the role instances it plays simultaneously (all roles instance share the same session). From this point of view, also in this case the state of an object can be role specific.
11. *Features of an object can be role-specific*: "Possible. Role are types and as such come with their own features. Role features are inherited to the types filling the roles, but a role-sensitive resolution mechanism (qualification) is needed if the same features are inherited from more than one role."
    As we already said, is it possible to model that if an actor plays a role a it acquires attr or/and op of the role instance played.
12. *Roles restrict access*: "Not applicable. Lodwick does not have notions of accessibility or visibility."
    If the role instance has its own identity it restrict access because it gives certain powers to the player playing it. These powers let the player access the private state of the Institution to which the role instance is linked. If we constraint the interaction with an object only through the roles it offers, we can model the situation in which roles restrict access.
13. *Different roles may share structure an behaviour*: "Partly. As noted under item 11, the features of role specifications are inherited down the role hierarchy to the types filling the roles. Vice versa, properties of the types filling roles are not inherited to these roles. For instance, if the type *Person* has a *placeOfBirth* attribute, this attribute is not shared by its role *Customer*. This however makes sense since not all customers are persons."
    Exploiting role hierarchies we can model inheritance of role's specifications, and through sessions we can let the behaviour of a role instance influenced by other roles.

14. *An object and its roles share identity*: “Yes. An object in a role is the object itself.”
    In our logical formalism: $R(i, a, o) \rightarrow a = o$
15. *An object and its roles have different identities*: “No. This follows from item 14.”
    In our logical formalism: $R(i, a, o) \rightarrow a \neq o$

3.3 RBAC model



—sets: U, R, AR, P, AP, S for sets of users, (regular) roles, administrative roles, (regular) permissions, administrative permissions, and sessions, respectively.
—$UA \subseteq U \times A$: user-role assignments
$AUA \subseteq U \times AR$: user-administrative role assignments
—$PA \subseteq P \times R$: permission-role assignments
$APA \subseteq AP \times AR$: permission-administrative role assignments
—$RH \subseteq R \times R$: role hierarchy
—$ARH \subseteq AR \times AR$: administrative role hierarchy
—$user : S \rightarrow U$, a function mapping a session to a single user
—$roles : S \rightarrow 2^{R \cup AR}$, a function mapping a session to a set of roles:
$roles(s) \subseteq \{r : R \mid (\exists r' \geq r) \cdot [(user(s), r') \in UA \cup AUA]\}$
—$permissions : R \rightarrow 2^{P \cup AP}$, a function mapping a role to a set of permissions:
$permissions(r) = \{p : P \mid (\exists r'' \leq r) \cdot [(p, r'') \in PA \cup APA]\}$
—collection of constraints

Fig. 1. Summary of RBAC96 model.

There are a few element which needs a deeper analysis to fit them in our role account.

– *Absence of an explicit context:* RBAC is a model which let a highly decentralized security administration thanks to a subtle role account, the model doesn't cope with contexts. In order to fit it with our approach we say that there is one dummy context which contains all system's roles.

– *Permissions:* In RBAC permissions are assigned to roles [2], a permission is an approval of a particular mode of access to one or more objects in the system. The terms *authorization*, *access right*, and *privilege* are also used in the literature to denote a permission. Permissions are always positive and confer on their holder the ability to perform an action in the system. A user who plays a role acquires all the system's permissions linked with the role played. One issue is how to fit the notion of permission in our model. In the literature in order to be able to define RBAC in a general and formal way, permissions are treated as uninterpreted symbols because permissions are implementation and system dependent. In fact each system has its own way to describe a permission and different accounts could dramatically differ one from another; from a formal point of view we are much more interested on *where* permissions are and not what they are. In RBAC permissions are assigned to role, so to fit with our model we decide to let permission be attributes so that $\mathsf{permissions} \subset \mathsf{Attr}$.
– *Sessions:* Users establish *sessions* during which they may activate a subset of the roles they belong to. Each session maps one user to possibly many roles. The double-headed arrow from session to R in Figure 1 indicates that multiple roles are simultaneously activated. The permissions available to the user are the union of permissions from all roles activated in that session. Each session is associated with a single user, as indicated by the single-headed arrow from the session to U in Figure 1. This association remains constant for a session's duration. A user might have multiple sessions open simultaneously,for example each in a different window on a workstation screen. Hence, each session is linked with a user and is always different from all other sessions, so we can say that a session is an instance of the user, if user x enters the system an instance y of x ($y :: x$) is created, and this instance (session) can, for example play roles (activate roles), there can exists many instances of x which are all linked with it but everyone is different from each other, in other words:
$$R(i, a, s, o) \rightarrow s = a$$
With this in mind we can state that the instantiation of a player individual x::y in our model corresponds to a session's activation. And the creation of the role instance R(i,x,o) correspond to the activation of the role R by the user y in the session x (where i is a dummy context). Playing a role gives to the user in that session all the permissions the are linked with R:
$$R(i, a, s, x) \rightarrow (\exists v : \mathsf{attr}(x) = v \leftrightarrow (\mathsf{attr}(R) \vee \mathsf{attr}(a) = v))$$
– *Administrative authority:* One of the most interesting points of RBAC is the possibility to use RBAC to manage itself. For this purpose the model introduces *administrative roles* AR and *administrative permissions* AP, the intent is for AR and AP to be respectively disjoint from regular role R and permissions P. The model shows that permissions can be assigned only to

roles and that administrative permissions can be assigned only to administrative roles; this is a built-in constraint. Usually, each administrative role is mapped to the subset of the role hierarchy it manages. With the introduction of AR and AP, in RBAC is defined a structured way to change what in our model is the *Universal Level*, in the literature there are many ways to administrate RBAC but each one could be easily merged with our model simply introducing an *administrative* meta-level which discriminate who and how can change the universal level.

4 Conclusions

In this draft we try to give a general, and relatively simple, formalism through which we grasp different notions of role. The idea is to constrain the model to meet others approaches in order to cover a wider literature on the subject. The more we can describe with this framework, the more we are sure that terms like *player, session, context* and *role instance* are pivotal elements which can give a possible reply to the challenging research question: what are roles?.

References

1. Loebe, F.: Abstract vs. social roles - a refined top-level ontological analysis. In Procs. of AAAI Fall Symposium on Roles, an interdisciplinary perspective Series Technical Reports FS-05-08. pages 93-100 (2005)
2. S.Sandhu, R., J.Coyne, E.: Role-Based Access Control Models. IEEE Computer, Volume 29, Number 2 38-47 (1996)
3. van der Torre, L., Boella, G.: Oranizations as socially constructed agents in the agent oriented paradigm. In Procs. of ESAW'04 (2004)
4. Masolo, C., Vieu, L., Bottazzi, E., Catenacci, C., Ferrario, R., Gangemi, A., Guarino, N.: Social roles and their descriptions. In Procs. KR2004, Whistler, Canada, June 2-5, 2004, pp.267-277 (2004)
5. Steimann, F.: On the representation of roles in object-oriented and conceptual modelling. Data and Knowledge Engineering, 35:83-848 (2000)
6. Baldoni, M., van der Torre, L., Boella, G.: Interaction between objects in powerJava. Journal of Object Technology(JOT) (2006)
7. R., D., R., H., M.Kuster: Roles in agent-oriented modelling. International Journal of Software Engineering and Knowledge Engineering (2001) 281–302
8. Boella, G., van der Torre, L.: The ontological properties of social roles in multiagent systems: Definitional dependence, powers and roles playing roles. Artificial Intelligence and Law Journal (AILaw) (2007)
9. Wieringa, R., de Jonge, W., Spruit, P.: Roles and dynamic subclasses: a model logic approach. In Procs. of 8th European Conference on Object-Oriented Programming (1994)
10. Wieringa, R., de Jonge, W.: The identification of objects and roles. (1991)
11. Herrmann, S.: Programming with Roles in ObjectTeams/Java. In proc. AAAI Fall Symposium 2005: Roles, an iterdisciplinary perspective (2005)
12. Oh, S., Sandhu, R., Zhang, X.: An effective role administration model using organization structure. ACM Transactions on Information and System Security Vol.9,No.2,May 2006, Pages 113-137 (2006)

13. Masolo, C., Guizzardi, G., Vieu, L., Bottazzi, E., Ferrerio, R.: Relational roles and qua-individuals. In Procs. of AAAI Fall Symposium on Roles, an interdisciplinary perspective (2005)
14. Baldoni, M., Boella, G., van der Torre, L.: Modelling the interaction between objects: Roles as affordances. In Procs. Knowledhe Science, Engineering and Management, First International Conference, KSEM 2006, Guilin, China, August 5-8, volume 4092 of Lecture Notes in Computer Science, pages 42-54. Springer, 2006 (2006)