A Meta-model for Roles: Introducing Sessions

Valerio Genovese
Università di Torino, Dipartimento di Informatica
10149, Torino, Cso Svizzera 185, Italia
valerio.click@gmail.com

Abstract. Role is a widespread concept, it is used in many areas like MAS, DB, Programming Languages, Organizations, Security and OO modeling. 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 (and opposite) ways. Our aim is to build a formal framework through which we can describe different definitions appeared in the literature or implemented in computer systems. In particular we give a new 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 relationships.

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 (eg. [1]). In a certain way, we can view roles as a pivotal concept 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 behavior 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 catch 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 implementations.

The paper is organized as follows. In section 2 we introduce the model. In section 3 we analyze in more depth sessions in relationships, showing links among states of different entities engaged in a collaboration. Conclusions close the paper.

2 A Logical Model for Roles

We define the formalism of the framework in a way as much general as possible, this gives us an unconstrained model where special constraints are added later in order to describe different approaches.

2.1 Universal Level

At the universal level we describe the relationship between natural and role types\(^1\), in particular we define two relationship PL and RO through which we link roles to contexts and players (natural types) to roles.

**Definition 1** An universal model is a tuple

\[
< D, \text{Contexts}, \text{Players}, \text{Roles}, \text{Attr}, \text{Op}, \text{Constraints} \\
\text{PL, RO, AS, OS, RH, PH, CH} >
\]

where:

- \(D\) is a domain of classes
- \(\text{Contexts} \subseteq D\) is a set of institutions
- \(\text{Players} \subseteq D\) is a set of potential players or actors
- \(\text{Roles} \subset D\) is a finite set of roles \(\{R_1, ..., R_n\}\)
- \(\text{Attr}\) is a set of attributes
- \(\text{Op}\) is a set of operations
- \(\text{Constraints}\) is a set of Constraints

The static model has also a few relations:

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

\(^1\) Natural types refer to the essence of the entities whereas role types depend on an accidental relationship to some other entity (context).
– **PH ⊆ 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 ⊆ 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 $c'$.

Given the information contained in AS and OS relations, we use $\pi_{Attr}(o)$ and $\pi_{Op}(o)$ as shortcuts to refer about the set of attributes and operations defined for class $o$. At this point we can add into **Constraints** some logical rules in order to model different role notions. For example in powerJava each role type is linked with one and only one context type \[5\], so we can express this through the following constraint:

$$\forall x,y,z (x \in \text{Roles} \land y, z \in \text{Contexts} \land x \text{RO} y \land x \text{RO} z \rightarrow y = z)$$

### 2.2 Individual level

The *individual* part relies on the universal one and the elements of this level are individuals (or instances) of the types defined at the universal level.

**Definition 2** A **snapshot model** is a tuple

$$< O, I_{\text{contexts}}, I_{\text{players}}, I_{\text{roles}}, \text{Sessions}, \text{Val}, I_{\text{constraints}}, \text{I}_{\text{Roles}}, I_{\text{Attributes}}, I_{\text{Operations}}, I_{\text{Attr}} >$$

where:

– $O$ is a *domain* of objects, for each object $o$ is possible to refer to its attributes and operations through $\pi_{I_{\text{Attr}}(o)}$ and $\pi_{I_{\text{Op}}(o)}$, respectively.

– $I_{\text{contexts}} \subseteq O$ is a set of institutions which instantiate classes in Contexts.

– $I_{\text{players}} \subseteq O$ is a set of actors, which instantiate universals in Players.

– $I_{\text{roles}} \subset O$ is a set of *roles instances*.

– $I_{\text{Attributes}}$ is the set of objects’ attributes.

– $I_{\text{Operations}}$ is the set of objects’ operations.

– Sessions is a set of sessions, which keep the state of an interaction between actors and institutions (See Section 3).

– Val is a set of values.

– $I_{\text{constraints}}$ is a set of integrity rules that constraint elements in the snapshot.

In this section we call elements in $I_{\text{contexts}}, I_{\text{players}}$ and $I_{\text{roles}}$ respectively, *institutions, actors* and *roles instances*.

The snapshot model has also a few functions and relations:

– $I_{\text{Roles}}$ is a *role assignment function* that assigns to each role $R$ a relation on $I_{\text{context}} \times I_{\text{players}} \times \text{Sessions} \times I_{\text{roles}}$.

– $I_{\text{Attr}}$ is an *assignment function* which it takes arguments an object $d \in O$, and an attribute $p \in \pi_{I_{\text{Attr}}(d)}$, if $p$ has a value $v \in \text{Val}$ it returns it, $\emptyset$ otherwise.
When an object \( x \) is an individual of the universal \( y \), we say that \( x \) instantiates \( y \) and, in order to express this in a formal way, we write \( a :: b \) when \( a \) is an instance of \( b \). In general if \( x :: y \), attributes and operations defined for \( y \) at the universal level are assigned to \( x \). If \( a \in \pi_{\mathsf{Attr}}(B) \) we write \( x.a \in \mathbf{Attributes} \) as the attribute instance assigned to object \( x \), the same holds for elements in \( \mathbf{Operations} \).

The role assignment function \( I_{\mathsf{Roles}} \) gives us the notion of an actor who plays a role within a specific context: if \( i :: x \) is an institution, \( a :: y \) an actor, and \( o :: R \) a role, \( (i,a,o) \in I_{\mathsf{Roles}}(R) \) is to be read as: “the object \( o \) represents agent \( a \) playing the role \( R \) in institution \( i \)”. We will often write \( R(i,a,o) \) for this statement, and we call \( o \) the role instance.

Suppose we have a role instance \( \text{employee} \), and that the value of the attribute salary is \( 1000 \in \mathbb{R} \) usually, instead of writing \( I_{\mathsf{Attr}}(\text{employee}, \text{salary}) = 1000 \), we write \( \text{salary}(\text{employee}) = 1000 \).

The way we defined a snapshot leaves a lot of room for formulating further constraints in \( \mathbf{Constraints} \) 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. **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 \( \mathbf{contexts} \) contains only this trivial institution, as we do in section 3 when we model RBAC [2].

2. **Context coherence.** From an organizational point of view, there cannot be a student role’s player without a teacher one, also it would not 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:

   \[
   \forall y \left( y :: \text{Family} \rightarrow \exists x, o, z, p \left( \text{husband}(y, x, o) \land \text{wife}(y, z, p) \right) \right)
   \]

   Which means that in the snapshot exists \( y \in \mathbf{contexts} \) if and only if there exist two role instances \( p \) and \( o \) which represent respectively an husband and a wife played by actors \( x \) and \( z \) in \( y \).

3. **Complementary roles.** In general we can express the fact that playing a role \( R \) for an actor implies that there exists another actor playing a complementary role \( R' \) with the following constraint:

   \[
   R(i, a, o) \rightarrow R'(i, b, x)
   \]

### 2.3 The dynamic model

The dynamic model defines a structure to properly describe how the framework evolves as a consequence of executing an action on a snapshot. We well see in Section 4 and 5 how is up to this model to constraint agents’ dynamics.
Definition 3 A *dynamic model* is a tuple

\[ < S, TM, \text{Actions}, \text{Requirements}, D_{\text{constraints}}, I_{\text{Actions}}, I_{\text{Roles}}, \pi_{\text{Req}}, I_{\text{Requirements}}, > \]

where:

- \( S \) is a set of *snapshots*.
- \( TM \subseteq S \times \mathbb{N} \): it is a time assignment relationship, such that each snapshot has an associated unique time \( t \). For the sake of simplicity we define a discrete time through positive natural numbers.
- \( \text{Actions} \) is a set of actions.
- \( \text{Requirements} \) is a set of requirements for playing roles in the dynamic model.
- \( D_{\text{constraints}} \) is a set of integrity rules that constraints the dynamic model.
- \( I_{\text{Actions}} \) maps each action from \( \text{Actions} \) to a function on \( S \). \( I_{\text{Actions}}(s, a, t) \) tells us how the snapshot \( s \) changes as a result of executing action \( a \) at time \( t \). This function returns a couple in TM that binds the resulting snapshot with time \( t + 1 \). In general, to express that at time \( t \) is carried action \( a \) we write \( a_t \).
- About \( I_{\text{Roles}} \), we say that \( R_t(i, a, o) \) is true if there exists, at a time \( t \), the role instance \( R(i, a, o) \).
- \( \pi_{\text{Req}}(t, R) \) returns a subset of \( \text{Requirements} \) present at a given time \( t \) for the role of type \( R \), which are the requirements that must be fulfilled in order to play roles of type \( R \).

Intuitively, the snapshots in \( S \) represent the state of a system at a certain time. Looking at \( I_{\text{Actions}} \) is possible to identify the course of actions as an ordered sequence of actions such that:

\[ a_1; b_2; c_3 \]

represents a system that evolves due to the execution of \( a, b \) and \( c \) at consecutive times. We refer to a particular snapshot using the time \( t \) as a reference, so that for instance \( \pi_{\text{Attr}} \) refers to \( \pi_{\text{Attr}} \) in the snapshot associated with \( t \) in TM.

We suppose that, for every time \( t \), given an object \( p \) we can always say if it exist or not via the \( \exists_{\text{Attr}} \) operator, so that \( \exists_{\text{Attr}}(p) \) is true, if and only if \( p \) exists at time \( t \), false otherwise. We write \( \exists_{\text{Attr}}(p) \) when \( p \) exists in all the snapshots of the dynamic model.

A particular aspect of the dynamic model is role *addition* and *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 other 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, then we describe a small set of possible actions that can be applied in the dynamic model.
Role Assignment

Let $M$ be a snapshot.

$$M = < O, I_{\text{contexts}}, I_{\text{players}}, I_{\text{roles}}, \text{Sessions}, \text{Val}, I_{\text{Roles}}, \pi_{\text{Attr}}, \pi_{\text{Op}}, I_{\text{Attr}} >$$

Let $i \in I_{\text{contexts}}$, $a \in I_{\text{players}}$, and $R \in I_{\text{Roles}}$. 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:

$$M' = < O', I_{\text{contexts}}', I_{\text{players}}', I_{\text{roles}}', \text{Sessions}', \text{Val}', I_{\text{Roles}}', \pi_{\text{Attr}}', \pi_{\text{Op}}, I_{\text{Attr}} >$$

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 $O$.
- $I_{\text{contexts}}' = I_{\text{contexts}}$ or $I_{\text{contexts}}' = I_{\text{contexts}} \cup \{o\}$.
- $I_{\text{players}}' = I_{\text{players}}$ or $I_{\text{players}}' = I_{\text{players}} \cup \{o\}$.
- $I_{\text{roles}}' = I_{\text{roles}}$, $\text{Val}' = \text{Val}$. The sets of roles and possible values of attributes do not change.
- $I_{\text{Roles}}'(R) = I_{\text{Roles}} \cup \{(i, a, o)\}$
- $\pi_{\text{Attr}}'$ and $\pi_{\text{Op}}'$ can be different if attributes and operations of a role are inherited by its player.
- $I_{\text{Attr}}'$ is just like $I_{\text{Attr}}$ with respect to the properties of objects different from $i$, $a$, and $o$.

For role addition and deletion actions we use, respectively $\text{Req}_t(i, a, R), R, i \hookrightarrow_t a$, and $\text{Req}_t(i, a, R), R, i \hookleftarrow_t a$. Then using the notation of dynamic logic we write:

$$[\text{Req}_t(i, a, R)?; R, i \hookrightarrow_t a] \phi$$

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

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

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

$$(\neg \exists x R(i, a, x)) \rightarrow [R, i \hookrightarrow_t a] \exists! x R(i, a, x)$$

The dynamic level can be constrained in order to model inheritance of attributes and operations, here we discuss only attributes, for operations the discussion is similar.
In the model, both roles and objects have properties. A natural constraint is that role-instances at least get all the properties that are defined for that role:

\[ R_t(i, a, s, x) \rightarrow (\text{attr} \in \pi_{\text{Attr}}(R) \rightarrow \exists v : \text{attr}(x) = v) \]

With respect to the question if the role-instance should 'inherit' all the properties of the original player object there are different possible answers.

For example, in powerJava [5], no such inheritance is assumed at all - the properties of the role instance are precisely those of the role, and we have that:

\[ R_t(i, a, x) \rightarrow (\text{attr} \in \pi_{\text{Attr}}(R) \leftrightarrow \exists v : \text{attr}(x) = v) \]

But other options are possible as well. For example, one alternative approach is that roles can be best seen as 'views' on a certain object, providing only a subset of the properties of the original object, like in Fibonacci [7]. A constraint which reflects that view is that the role-player has only the properties that are defined for the original object as well as for the role:

\[ R_t(i, a, s) \rightarrow \pi_{\text{Attr}}(R) \subset \pi_{\text{Attr}}(a) \]

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:

\[ [R, i \leftarrow t a] (\pi_{\text{Attr}+1}(a) = \pi_{\text{Attr}}(a) \cup \pi_{\text{Attr}}(R)) \]

The same considerations hold for operations. In the above formula we introduced an ad-hoc union operator \( \cup_A \) that binds attributes of an object with attributes of a class instantiating them. For instance if we have an object \( o \) and a class \( T \), the union \( \pi_{\text{Attr}}(o) \cup_A \pi_{\text{Attr}}(T) \) add into \( \text{Attributes} \) the elements o.a, a \( \in \pi_{\text{Attr}}(T) \).

Methods

There are other actions through which it is possible to change the model as well, for instance objects may assign new values to their attributes [8]. 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).

3 Sessions and relationships

We explicitly introduce sessions because we argue that are strictly linked with the role's notion. As already said, we talk about sessions when it is necessary to keep the state of an interaction between entities. Sessions in our model are a couple \((ID, K)\) where \(ID\) is an identifier and \(K\) a set of attributes and operations. If an attribute is in \(K\) it means that its value maintains a particular information
on the state of the interaction between an actor playing a role and an institution offering it. Operations in $K$ are behavioral aspect of the interaction and they can change the value of the attributes that are in the same session, this means that operation in $K$ can change attributes in $K$ even if the are of different objects. For instance suppose to have $R(i,a,s,x)$, depending on what we want to model, we can look at sessions from three different points of view:

1. A session can collapse into one role instance $[1,5,9]$ ($ID = x$). This means that attributes and operations in $K$ are all a subset of $\pi_{Attr}(x) \cup \pi_{Op}(x)$ where $x \in I_{roles}$.
2. A session can collapse into the actor $[2,10]$ ($ID = a$). In that case peculiar attributes and operations for the interaction are linked with the object representing the actor.
3. A session can be an object with its own ID (like when we reify an association). It is important to underline that a session of this type can link different role instances embedding their attributes and operations in $K$, so that the state of a role instance $a$ can be influenced by the behavior of another role individual $b$.

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

$$R(i,a,s,x) \rightarrow \pi_K(s) \subseteq \pi_{Attr}(x) \cup \pi_{Op}(x)$$

Where $\pi_K(s)$ is a projection on the second element of couple $s$. The point is slightly different if roles are not instantiable, in this case we have:

$$R(i,a,s) \rightarrow \pi_K(s) \subseteq \pi_{Attr}(a) \cup \pi_{Op}(a)$$

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

![UML collaboration diagram](image)

**Fig. 1.** UML collaboration diagram

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) $[10]$. 
In Figure 1 the labels of the association ends correspond to our roles, a straight line between a Teacher and a Student identify an interaction between them, where tutor and student are the roles through which the interaction takes place.

Depending on what we have in mind, we can express the interaction between two instances of Person (one acting as a Teacher and the other one as Student) in two different ways, if \( x :: \text{Person}, \ y :: \text{Person}, \ \text{tutor} :: \text{Teacher} \) and \( \text{student} :: \text{Student} \):

1. \( \text{Tutor}(y, x, q, \text{tutor}) \land \text{Student}(x, y, q', \text{student}) \)
2. \( \text{Tutor}(y, x, q, \text{tutor}) \land \text{Student}(x, y, q, \text{student}) \)

Notice that \( x \) and \( y \) are both in contexts and players, because they offer and play roles at the same time. In the first view we have two separate sessions each one representing a specific direction of the association between \( x \) and \( y \), whereas in the second approach a common session \( q \) unifies the two-way association seeing it as a unique interaction with a unique state for both directions (\( x \rightarrow y \) and \( y \rightarrow x \)). It must be said that is not mandatory to model the interaction between \( x \) and \( y \) with role instances, if we do not want roles to be instantiated we simply let sessions refer to attributes and operations of \( x \) and \( y \).

The UML collaboration diagram (Figure 1) defines, at the specification level, how instances of different classes must behave in order to be engaged in the collaboration in a sort of relationship’s pattern. In Figure 2 we represent role instances inside the context that offers them, the relation of playing a role is translated through an arrow which goes from the actor to the role instance played.

The view of putting the role tutor inside the object studentD, together with having all objects being at the same time contexts and players of some roles, could seem counter intuitive, but is extremely powerful. Role instances are seen as set of affordances [11] that let the actor interact with another entity, in general an actor plays a role which is linked with a context, and the fact of playing that role gives him the power to modify the properties of the context. With this example in mind we can now translate the diagram in Figure 1 representing it through a set of constraints at the individual level:

\[
\begin{align*}
Tutor(&text{student}, text{teacher}, s_1, text{tutor}) \land \\
Faculty \text{Member}(text{faculty}, text{teacher}, s_2, text{faculty \_member}) \land \\
Lecturer(&text{course}, text{teacher}, s_3, text{lecturer}) \land &text{Student}(text{teacher}, text{student}, s_1, text{student}) \land \\
Participant(&text{course}, text{student}, s_4, text{participant}) \land \\
Faculty(text{teacher}, text{faculty}, s_2, text{faculty}) \land \\
Taken \text{Course}(text{student}, text{course}, s_4, taken \text{course}) \land \\
Given \text{Course}(text{teacher}, text{course}, s_3, given \text{course})
\end{align*}
\]

\(^2\) In this section we refer to classes with the first letter capitalized in order to distinguish them from instances which are in lower case.

\(^3\) In order to avoid confusion we refer to teacher, student, course and faculty as instances of the classes involved in the collaboration diagram.
This predicate represents a set of constraints that have to be applied to all entities that want to be engaged in the collaboration diagram in Figure 1. So it is impossible to play the role lecturer without offering the role student, and without being engaged in all others associations implied in the collaboration diagram.

We said that playing a role always translates into modeling an interaction, and that the state and behavior of this interaction is kept by a subset of attributes and operations of the entity engaged in the relationship. We introduce the term session to refer to this subset of elements because this abstraction let us model, in a formal and hopefully clear way, the links that relate the states of the elements playing roles in a relationships.

In general, when attributes’ values in a session $s_1$ are influenced by operations or actions carried out by other roles which elements are in another session $s_2$, we need to express an integrity rule that links the states of $s_1$ and $s_2$.

Referring to Figure 2, suppose that faculty_member and tutor have an attribute num_courses which value counts the number of courses held by the teacherA, if teacherA stops playing lecturer in courseC, 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 lecturer’s state (i.e the execution of a tutor’s operation can change one or more lecturer’s attributes).

Then we can define the following integrity rule in $D_{contraints}$ of the dynamic model:
\[\forall z, p, q :\]
\[
p :: \text{Faculty} \land q :: \text{Student} \land z :: \text{Teacher} \land
\]
\[\text{faculty}_\text{member}(p, z, s_1, x) \land \text{tutor}(q, z, s_2, y)\]
\[\rightarrow\]
\[\text{num\_courses}(x) = \text{num\_courses}(y) = \beta\]

Where \(\beta\) is the number of lecturer instances played by \(z\). Notice that in the dynamic model the value of \(\beta\) can always be deduced analyzing the set of snapshots in \(S\).

With the introduction of sessions we argue that to model properly a relationship is important to talk about states that are strictly linked with the role played, and that roles cannot be simple labels of association ends.

4 Conclusions

In this article we extend and improve the framework for modelling roles introduced in [8], moreover we redefine the session notion in order to analyze in more depth how roles are linked with relationships. In [8] it has been shown that the model is able to grasp different role notions, the aim is to find the basic elements which can describe what roles are. We think that relationships are the right place to investigate and to find a foundation of roles.

References