UNIVERSITÀ DEGLI STUDI DI TORINO
FACOLTA’ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
CORSO DI LAUREA IN INFORMATICA

RELAZIONE DI TIROCINIO

Un Metamodello per i Ruoli in Sistemi Multi-agente e Programmazione a Oggetti

Relatore: Prof. Guido Boella
Candidato: Valerio Genovese

Anno Accademico 2006/2007
Ringraziamenti

Desidero ringraziare Francesca per la sua comprensione e il suo entusiasmo.

La mia famiglia per avermi sempre sostenuto nell’inseguire le mie passioni.

Federico Bellan, Filippo Racca e gli altri miei compagni di corso per tutte le ore spese insieme a lezione e a studiare sui libri.

Il Dott. Roberto Grenna per la sua sincera disponibilità e simpatia.

Il Prof. Matteo Baldoni per le interessanti discussioni e idee.

Il Dott. Jelle Gerbrandy per essersi letto le mie bozze, per le correzioni e per i suoi preziosi suggerimenti.

Infine, questa tesi la devo al Prof. Guido Boella, desidero profondamente ringraziarlo per avermi proposto questo lavoro, per la sua fiducia e per essere sempre stato disponibile durante tutto quest’anno.
## Contents

List of Figures v

1 Introduction 1

2 Roles in OO and MAS 3
   2.1 Steimann’s Approach to Roles ................................. 3
      2.1.1 Roles as First Class Citizens ............................ 3
      2.1.1.1 Lodwick: a Modelling Language for Roles ......... 8
      2.1.2 Roles as Interfaces ........................................ 8
      2.1.2.1 Roles and Interfaces in UML ......................... 10
   2.2 Loebe: Abstract vs Social Roles .............................. 15
      2.2.1 A general approach to Roles .............................. 15
      2.2.2 Role Types ................................................ 16
      2.2.3 Abstract vs Social Roles .................................. 18
   2.3 Roles in Agent Deliberation .................................... 19
      2.3.1 Role Enactment and Role Activation ..................... 20
      2.3.2 Agent Roles and Agent Types ............................. 21
      2.3.3 Role Enacting and Role Deacting Agents ................. 22
      2.3.4 Activating and Deactivating Roles ...................... 24

3 A Meta-model for Roles ................................................. 26
   3.1 Introduction ..................................................... 26
   3.2 A Logical Model for Roles ...................................... 26
      3.2.1 Universal Level ............................................ 27
      3.2.2 Individual Level .......................................... 28
3.2.3 The Dynamic Model .................................. 32
  3.2.3.1 Role Addition .................................. 35
  3.2.3.2 Role Assignment ................................ 36
  3.2.3.3 Specializing the Dynamic Model ............... 37
  3.2.3.4 Object Deletion ................................ 37
  3.2.3.5 Role Deletion .................................. 38
  3.2.3.6 Methods ....................................... 39
  3.2.3.7 Powers ....................................... 39
  3.2.3.8 Operations .................................... 40

3.3 Sessions and Relationships .......................... 40
3.4 An Explanatory Example ............................... 45
  3.4.1 Universal Level .................................. 45
  3.4.2 Individual Level ................................ 46
  3.4.3 Dynamic Model .................................. 47

4 Comparing Different Role’s Accounts............... 48
  4.1 Social Roles ....................................... 48
    4.1.1 The Key Features of Social Roles .............. 48
  4.2 Steimann’s Approach ................................ 50
  4.3 RBAC Model ....................................... 57
  4.4 Zope Model ....................................... 60

5 Modeling Roles in Agent Organizations .......... 64
  5.1 Introduction ....................................... 64
  5.2 Agents and Roles ................................... 66
  5.3 Roles in Multiagent Systems ........................ 69
  5.4 Enact and Deact Roles .............................. 70
  5.5 The Public Dimension of Roles .................... 72
  5.6 The Combined Model ................................ 74

A Can roles play roles? ................................. 80
  A.1 Role as a Founded Concept ....................... 80
  A.2 The Counting Problem ............................. 81
  A.3 Conclusions ...................................... 82
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Role and natural types within an heterogeneous hierarchy</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Roles statically and dynamically: (a) The player implements the properties of the role. (b) The behaviour of a role played may be more specific than that of the player.</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>UML metamodel</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>new UML metamodel</td>
<td>12</td>
</tr>
<tr>
<td>2.5</td>
<td>(a) A class diagram (b) and its transcription in the new metamodel</td>
<td>13</td>
</tr>
<tr>
<td>2.6</td>
<td>(a) A collaboration diagram (b) and its transcription in the new metamodel</td>
<td>14</td>
</tr>
<tr>
<td>2.7</td>
<td>Simple Role Model</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>UML collaboration diagram</td>
<td>42</td>
</tr>
<tr>
<td>3.2</td>
<td>Example of object diagram with roles</td>
<td>43</td>
</tr>
<tr>
<td>5.1</td>
<td>Roles in MAS</td>
<td>77</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The notion of role is a modelling concept strictly linked with the interaction among 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 interacts. The term “student” refers to a person that is a student in a specific university (e.g. Loebe [2005]). 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 how they can be represented in the Object Oriented 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 [S.Sandhu & J.Coyne, 1996], in organizational models roles give powers to their players in order to access an institution, in Multiagent Systems roles can be seen as descriptions of the behaviour which is expected by agents who play them [Boella & van der Torre, 2004], in ontology research roles are an anti-rigid notion founded on a player and a context [Masolo et al., 2004], 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 accounts of the notion
of role 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 thesis is organized as follows. In Chapter 2 we describe three different accounts appeared in the literature: two from the Object Oriented point of view, and one from the Multiagent Systems paradigm. In Chapter 3 we introduce our meta-model for roles together with its three levels: universal, individual and dynamic. In Chapter 4 we model four different accounts of roles with the metamodel introduced in Chapter 3. In Chapter 5 we exploit the framework as a merger of concepts between role driven coordination and agent deliberation process.
Chapter 2

Roles in OO and MAS

2.1 Steimann’s Approach to Roles

This section is mainly based on the two articles Steimann [2000] and Steimann [2001]. Its aim is to underline some of the basic intuitions that Steimann has about roles in conceptual modeling and in OO programming.

The two articles cited above are extremely useful to approach roles, mainly because they give a comprehensive view on how roles are represented in the literature and what are the main problems in trying to catch this complex notion. Is interesting also to look at Steimann’s definition of roles, which has a strong basis in OO paradigm and has the benefit of being clear and unambiguous.

The first subsection introduces roles as first-class citizens in OO while the second one is about roles in UML and propose to merge the concepts of interface and role.

2.1.1 Roles as First Class Citizens

In Steimann [2000], Steimann advocates the introduction of roles as first-class citizen in object-oriented and conceptual modelling.

His approach is mainly based on the intuition that the notions of object and relationship are naturally complemented by the one of role. Although definitions of role concept abound in the literature, it seems that there is much confusion of what are the basic elements that define precisely what a role is.
2.1 Steimann’s Approach to Roles

Steimann tries to group a wide range of definitions of the role concept identifying a list of features appeared in the literature:

1. A role comes with its own properties and behaviour.
2. Roles depend on relationships.
3. An object may play different roles simultaneously.
4. An object may play the same role several times, simultaneously.
5. An object may acquire and abandon roles dynamically.
6. The sequence in which roles may be acquired and relinquished can be subject to restrictions.
7. Objects of unrelated types can play the same role.
8. Roles can play roles¹.
9. A role can be transferred from one object to another.
10. The state of an object can be role-specific.
11. Features of an object can be role-specific.
12. Roles restrict access.
13. Different roles may share structure and behaviour.
15. An object and its roles have different identities.

The fact that there are features that conflict with others is a clear evidence that, actually, there is no single definition of role integrating all the different accounts.

Is interesting to note that although there are many different properties associated with roles, the number of substantially differing definitions of the concept

¹We refer to Appendix A for a deeper analysis of this feature from the author’s point of view.
is quite small. To give an evidence of this intuition, in Steimann [2000] three possible views on roles are identified: roles as named places of a relationship, roles as a form of generalization and/or specialization, and roles as adjunct instances. Each view fails to cover some of the role’s features listed above.

Concerning named places of a relationship we can notice that this view is unable to model the feature that roles come with their own properties and behaviour, this is mainly due to the fact that this approach looks at roles as mere labels of types (the same as UML 1.4 roles names). Instead, it is possible to overcome this deficiency by regarding roles as types in their own right.

The problems linked with roles as specializations and/or generalizations are more subtle; to show this, suppose that Person and Government are two natural types ¹, suppose also to have two roles Killer and Swindler that can be both played by Person or Government. Given this situation it could be sensible to state Killer and Swindler as subtypes of both Person and Government (Fig. 2.1.a), the main problem is that the extent of Killer and Swindler should be the intersection of Person and Government which is counterintuitive, mainly because it is improbable that an instance of Person is also an instance of Government, and even if this is the case, it is impossible to instantiate a Killer as a Person that is not at the same time an instance of Government.

Another approach could be the one depicted in Fig. 2.1.b where Party² is super-type of Person, Government, Killer and Swindler, but with this view we cannot distinguish which elements of the hierarchy are roles and which are natural types.

¹For natural types we indend something that relate to the essence of the entities, in contrast role types depend on an accidental relationship to some other entity.
²Party is a common catch-all type.
2.1 Steimann’s Approach to Roles

Figure 2.2: Roles statically and dynamically: (a) The player implements the properties of the role. (b) The behaviour of a role played may be more specific than that of the player.

If seeing roles as mere specializations (subtypes) in a heterogeneous type hierarchy\(^1\) seems to be wrong, we can try to model roles as supertypes (Fig. 2.1.c). This approach is possibly even worst than the other one; in fact, a government for example has many properties not required by a killer or a swindler.

So, what are role types and how is possible to mix them with natural types? These are not trivial questions, the main problem is that a type hierarchy is static but the role notion seems to have both a static and a dynamic character. The approach depicted in Fig. 2.1.c in which roles are supertypes, models the fact that all persons and government can (potentially) appear in these roles. In contrast defining roles as subtypes underlines the dynamic viewpoint, namely that at any point in time only some of all persons and governments existing at that time are killers and/or swindlers.

In object oriented modelling, types are classes and objects elements that instantiate one or more classes, depending on the type hierarchy. We said that roles are supertypes statically, but subtypes dynamically, where “statically” refers to the state of an object, “dynamically” to the behavior of it [Loebe, 2005].

\(^{1}\)The type hierarchy is heterogeneous because it contains both role and natural types.
conflicts with the standard notions of inheritance in class-based modelling, where state as well as behavior is inherited from a superclass to its subclasses. With this in mind it is awkward to describe the relationship even between a single role type \textit{Student} and a natural type \textit{Person}. In fact a person in a student role should inherit the properties of \textit{Student} (which requires \textit{Student} to be a superclass of \textit{Person}), however, student’s behavior may be more specific than that of \textit{Person} (thus Students should be subclasses of \textit{Person}) (see Fig. 2.2).

The other approach criticized by Steimann considers \textit{roles as adjunct instances}. In this view roles are modelled as independent types the instances of which are carriers of role-specific state and behaviour, but not identity. An object and its roles are then related by a special \textit{played-by} relation, and an object and its role form an aggregate that appears indivisible from the outside. The dynamic picking up of a role corresponds to the creation of a new instance of the corresponding role type and its integration into the compound, and dropping a role means releasing the role instance from the unit and destroying it [Steimann, 2000]. There are many different practical implementations of this role notion, but the biggest problem with viewing roles as adjunct instances is that it requires an unusual notion of instance, where the classic one states that every object has its own identity, immutable and persistent. Even if it has been recognized that modelling roles as adjuncts instances is the only legitimate object-oriented implementation of roles [Couad & Mayfield, 1999] there are a few aspects that need to be analyzed.

The main problem with this view is that role instances have not an autonomous identity but they depend on other instances, i.e. the role players. Generally, when two different objects share the same identity, we talk about “object schizophrenia” as a violation of the assumption that every object has its own identity, immutable and persistent, making it distinct from all others\textsuperscript{1}. Steimann observes that in reality the role played by an object is not a different object, but merely its appearance in a given context, so he advocates that role instances should not have distinct identity\textsuperscript{2}.

\textsuperscript{1}Today it must be said that the number of researchers that consider role instances as a special case of “object schizophrenia” is rapidly lowering.

\textsuperscript{2}In our view, this is a very strong position. If this should be taken as a law, also the
2.1 Steimann’s Approach to Roles

2.1.1 Lodwick: a Modelling Language for Roles

After a deep analysis of the different role accounts appeared in the literature, Steimann introduces a role-oriented modelling language called LODWICK [Steimann, 2000] in which roles are first-class citizens that act as placeholders in relationships. In LODWICK roles and natural types have different type hierarchies that are linked through a role-filler relationship \( \leq_{NR} \). The idea is to define an association as an instance of relationship in which actors play roles occupying specific positions in the pattern. For example we write:

\[
\text{[Customer} \rightarrow a : \text{Person, Supplier} \rightarrow b : \text{Organization}] : \text{deliver} \quad (2.1)
\]

to express that \( a \) and \( b \) plays respectively roles \( \text{Customer} \) and \( \text{Supplier} \) in an instance of the relationship deliver.

LODWICK has a static and a dynamic model, through the description of the extensions and intensions of the elements that specifies the two models, Steimann gives a definition in which statically roles are supertypes of the type filling them whereas dynamically their extensions are subsets\(^1\).

With LODWICK Steimann manages to introduce roles as first-class citizens, resolving the conceptual problem of seeing them as specifications and/or generalizations in a type hierarchy with the introduction of two different hierarchies linked with a second order relationship \( (<_{NR}) \) and of two models that talk about roles from two different point of view (i.e., static and dynamic).

2.1.2 Roles as Interfaces

In Steimann [2001], Steimann proposes a modification of the UML meta model that consists in merging roles and interfaces in order to give a proven OO programming and design construct and a meaningful conceptual representation. Interfaces are an extremely important OO programming concept since they allow the decoupling of specification and implementation, on the other side, roles are a popular OO modelling concept. The two concepts are both present in the UML reification of a relationship should be considered bad OO design, in fact in the real world relationship are not objects with a proper identity.

\(^1\) For a complete description of the model signature we refer to Steimann [2000].
metamodel, and at a first glance, it seems that they have not much in common. In contrast, Steimann argues that merging Roles and Interfaces empowers programmers to model interfaces using the role’s conceptual tool, and also simplifies UML diagrams.

In Steimann [2001] it is underlined an interesting point of view which gives an alternative definition of classification in OO programming languages. The classical definition states that classification refers to an object being an instance of a class, in this view languages like Java or C++ fail to offer dynamic classification, because they are statically typed and multiple classification is limited to the object’s being an instance of a certain class and all its superclasses.

Steimann proposes an interesting view in which “classification of an object is defined as that object belonging to the dynamic extent of a certain type, i.e., as being assigned to a variable of that type”, with this approach classification is both multiple and dynamic. For instance, in Java instances of a class can be assigned to variables declared of this class and its superclasses but also to variables declared of interfaces implemented by the class. With this view in mind classification is multiple and also dynamic “because the object joins and leaves the dynamic extents of types”. It must by said that dynamicity has some constraints, in fact the number of types through which the object can enter and leave their extents is statically limited by the type (class and interface) declaration of a program.

In the context of UML collaborations, we can see how the classifier role can be compared with the interface concept through the notion of classification. In UML a classifier role is a classifier like a class or interface, but “since the only requirement on conforming instances is that they must offer operations according to the classifier role, as well as support attribute links corresponding to the attributes specified by the classifier role, and links corresponding to the association roles connected to the classifier role, there may be instances of any classifier meeting this requirement.” [omg, 1999], or put in another way, a classifier role represents a set of requirements that objects, that want to fill its place, have to fulfill. With this in mind we see how classification by a classifier role is multiple since it does not depend on the class instance classified, and dynamic because it takes place only when an instance assumes a role in a collaboration.

At this point we can list some interesting properties of roles:
2.1 Steimann’s Approach to Roles

- Roles can classify objects and role classification is multiple and dynamic.

- Roles are only partial specification of the objects playing them.

- Objects can play several roles and accordingly have many partial specifications that all add to the total specification of the classes they are an instance of.

- A role can be played by instances of different classes that are not related by inheritance and, in particular, role playing is independent from implementation.

Now if we see interfaces as types, like in UML or Java, we can appreciate how the properties of roles are the very properties of interfaces. Steimann, on the basis of this strong analysis, argues that roles and interfaces are largely the same concept.

The use of interfaces in OO programming is strongly promoted in order to restrict access between classes to the features actually required, and this restriction is best realized by declaring variables and formal parameters as interfaces rather than classes. This approach is sometimes summarized in the view “program to an interface, not an implementation”: the idea is that interfaces might be only partial specifications of classes, specifications that highlight one particular aspect or usage of a class, and that roles are the appropriate conceptual abstraction for this. Dividing the total interface of a class into several (possibly overlapping) facets promote substitutability between classes but this division should be done carefully identifying the correct usages of a class. As a rule of thumb, designers may be guided by testing whether the interface they are about to introduce conceptually is a proper role of the problem domain [Steimann, 2001].

2.1.2.1 Roles and Interfaces in UML

Steimann advocates that in order to ensure that interfaces are properly designed into OO programs from the beginning, a “suitable conceptual abstraction that blends well with other OO modeling concepts is needed” [Steimann, 2001]. Roles, it seems, are such an abstraction, and although roles and interfaces each have their
2.1 Steimann’s Approach to Roles

Figure 2.3: UML metamodel

place in OO modelling, one may argue that the two concepts should really be only one. In trying to merge these two concepts, Steimann proposes some changes to the UML metamodel (Fig.2.3). Quite obviously, numerous constraints are necessary to restrict the possible instantiations of this metamodel to those that make sense. In particular it must ensure that certain types of classifiers do not appear in certain contexts; for instances, it must be excluded that a classifier role specifies another classifier role as its base, or that an association mixes classes and classifier roles\(^1\). After an exhaustive discussion of pro and cons of the model in Fig.2.3, Steimann underlines that even if it seems that roles add much complexity to UML, they are necessary because otherwise the model is not expressive enough to address certain modeling problems.

Steimann tries to simplify the meta model (Fig.2.4), we next report the changes as described in Steimann [2001]:

1. The metaclasses **Interface** and **ClassifierRole** are merged into a new metaclass **Role**. The restrictions regarding interfaces in UML (that they cannot have

\(^1\)For a deeper analysis of the UML metamodel see Steimann [2001]
2.1 Steimann’s Approach to Roles

attributes or occur in other places than the target ends of directed associations) are dropped. **Class** and **Role** are strictly separated, while classes can be instantiated (unless of course they are abstract), roles can not. Also, classes and roles are generalized separately.

2. The association between classifier roles and their base classifiers is replaced by a new relationship, named **populates**, that relates classes with the roles their instances can play (it is convenient to speak of a class as *populating* a role and of an instance as *playing* a role). It is important that these are distinguished: populating corresponds to the subclass relationship among classes, while playing corresponds to the instance-of relationship of an instance to its class.

3. Association ends are required to connect to roles exclusively. Because roles are now interfaces and subroles can combine several interfaces, both pseudo-attributes **type** and **specification** are replaced by one new relationship, **fills**, associating each association end with one role. The classes whose instances actually participate in an association are specified only indirectly, via the **populates** relationship between classes and roles. Association ends need not be given a rolename; if they are, this name must equal to the connected roles. Every role must be unique within an association, i.e., no two association ends of one association must specify the same role.

4. **AssociationEndRole**, **AssociationRole**, and the generalization of associations are replaced by *association overloading*. For this purpose, a new metaclass, **Signature**, is introduced whose instances stand between an association and its (overloaded) association ends. Thus, rather that giving rise to the concept of association roles, an association restricted in the context of a collaboration

![Figure 2.4: new UML metamodel](image-url)
2.1 Steimann’s Approach to Roles

Figure 2.5: (a) A class diagram (b) and its transcription in the new metamodel

entails a new instance of Signature, comprising new association ends, each connected to a role defined by the collaboration.

The new UML metamodel reflects the suggested conceptual equivalence of interfaces and roles. In particular, interface specifiers and classifier roles are no longer treated as different concepts, and UML’s indifference with regard to classifier’s being a class or an interface is lifted. The association roles and association end roles of UML are not roles in the usual sense, calling them roles is a peculiarity of UML, and not calling them so (due to their abolition) is not likely to be considered as a loss.

As Steimann underlines, the merger of role and interface results not only in a reduction of concepts, it also gives interfaces a more prominent status in OO modeling, a status they have long earned in OO design and programming. However, while the changed metamodel may indeed be considered a simplification, it is the modeling language’s notation rather than its abstract syntax that must stand the test of practicability. More specifically: not the metamodel, but the diagrams drawn by the modeler must be compact, intelligible, and unambiguous.

For this reason we compare the new metamodel with “classic” UML through a class diagram (Fig. 2.5) and a collaboration diagram as examples.

In Fig. 2.5 we have a class diagram with classes representing roles, there are two associations, one ternary (buys) while the other is binary (produce). In the
2.1 Steimann’s Approach to Roles

Translation with the new metamodel, Steimann proposed to draw roles as circles, and the *populates* relation as dashed arrows. In this view the interaction between *Person* and *Factory* is constrained through the two interfaces/roles *Retailer* and *Producer*. Notice that the translation in Fig. 2.5(b) gives a much more flexible description of the diagram; in fact, every class which *populates* a role can substitute the actual one, so for instance if the class *Carpenter* populates the role *producer* it can be switched with *Factory* without any change in the other association ends.

The second example is depicted in Fig. 2.6 and represents the translation of a collaboration diagram. In “classic” UML collaboration diagrams, roles serve two purposes: they label association ends (the UML term for places of relationships), and they act as type specifiers in the scope of a collaboration. So for instance *Teacher* is a collaboration role and *student* is a label of an association end. In Fig. 2.6(b) collaboration roles are removed while the labels of association ends are translated into interfaces; as in Fig. 2.5 classes are linked with roles through the *populates* relation. It must be underlined that the loss of collaboration roles in the new diagram is not a problem, in fact what is important for the expressive power of the diagram are the roles which label association ends, because they constraint the interaction between entities. The view that roles model the inter-

![Collaboration Diagram](image-url)
action between entities is clear if we look at object diagrams where all instances are accessed via roles.

The proposed merger has also downsides; for instance defining roles as interfaces does not cover everything one might expect from the role concept. In fact, in certain situations it might be desirable that an object has a separate state for each role it plays, even for different occurrences in the same role. Another problem is that the implementation of classes populating many roles, such as Person, will become very large [Steimann, 2001].

To sum up we can say that equating interfaces with roles gives a proven OO programming and design construct. With a few additional commitments, the number of elementary modelling concepts can be considerably reduced, resulting in a simpler metamodel structure on one side and in a clearer separation between structure and interaction diagrams on the other [Steimann, 2001].

2.2 Loebe: Abstract vs Social Roles

2.2.1 A general approach to Roles

Loebe’s work on roles [Loebe, 2003, 2005] results in an account of roles which in its most abstract form involves three recurrent, interrelated notions as depicted in Fig. 2.7.

Roles form the central and mediating element of this model, that does not comprise a direct relationship between players and contexts. Each role \( q \) requires a player \( p \) and a context \( c \). The main connection among these notions can be termed as a determination relationship, such that players are determined by roles, whereas roles are determined by contexts. Concerning the relationships between players and roles and between roles and contexts we can state:

\[
\forall x (\text{Role}(x) \leftrightarrow \exists y z (\text{plays}(y, x) \land \text{roleOf}(x, z)))
\]
We next report some examples taken from Loebe [2005] because they will be useful in introducing Loebe’s different role types. First, assume that John is a student. The term student refers to a role played by the human John within his university, which in this case provides the context for that role. The second example refers to the fact that 2 is a factor of 4. In a coherent view with Fig. 2.7 we can see factor as a role term whose context is provided by the relationship being a factor of. The number 2 plays this role in relation to 4, whereas 4 plays the role of a multiple in relation to 2\(^1\). Similarly, when John moves a pen, he plays the role of a mover in the context of the overall movement process.

Looking at these examples, the question arises whether for instance human being is supposed to be understood as the player universal of student. More generally speaking, the player in Fig. 2.7 appears to reflect what is often called a type, natural type or natural kind and which we refer to as a natural universal: a universal whose instances have a more independent character than roles and which can play roles. As Loebe underlines, it is one of the most salient features of role accounts in computer science to contrast roles with natural universals. Instances of the latter are frequently used to model admissible universals for particular roles, i.e., to express restrictions on the players of certain roles. Put differently, natural universals are a means to refer to potential players of a role by their internal structure. In order to extend the base model with the notion of natural universals, we need to make a commitment to the view that, regarding instantiation, all notions introduced above come in an individual and a universal flavour. That means, for example, there are player individuals and player universals.

### 2.2.2 Role Types

Loebe in his analysis defines three different role types:

- **relational role**: corresponds to the way in which an argument participates in some relation.

- **processual role**: corresponds to the manner in which a single participant behaves in some process.

\(^1\)The author is not so confident in seeing a relationship modelled by roles between 2 and 4.
2.2 Loebe: Abstract vs Social Roles

- **social role**: corresponds to the involvement of a social object within some society.

It should be useful now to sort the examples cited above with respect to these types. 2 as a factor of 4 refers to a relationship, hence factor is considered a relational role universal, whereas John’s moving some pen is categorized as a process, hence a mover turns out to be a processual role universal, equals to the moved. Finally, at first glance, student should be classified as a social role universal because the context is provided by some university society.

Now the nature of the plays relation can be studied with respect to each of these role types, and it is informative to reconsider the role-of relationship as well.

As explained in Loebe [2005], relational roles (RR) could be seen as special qualities. This assumption implies a dependence of relational roles on their players. Indeed, the non-migration principle which applies to qualities therefore applies to relational roles as well:

\[ \forall xyz (RR(y) \land \text{plays}(x,y) \land \text{plays}(z,y) \rightarrow x = z) \]

Their distinctive feature compared to “usual” qualities like weight or age is an additional dependence on “complementary” relational roles. Assume, for example, that John is medically treated by Sue, i.e., there is a relator (a relation instance) connecting John and Sue such that John plays the role of the patient and Sue that of the attending physician. Here, the particular patient role of John and the physician role of Sue are interdependent, and both are dependent on their players. By means of role-of these two roles form a relator which connects John and Sue.

Processual roles (PR) rely on a different basis in dismantling a process. They “slice” processes with respect to the dimension of participants. When John moves his pen, he and the pen form participants of that process and the processual role which John plays captures what John does in that participation. Since their contexts are processes, processual roles are parts of processes and therefore processes themselves. Note that there is a mutual interdependence among all processual roles of a process (e.g., the mover and the moved from above), in best
analogy to relational roles. This in turn yields a distinction among processes such
that an independent process $p$ can be split into dependent processes $q_1, \ldots, q_n$ -
its processual roles - based on the participants of $p$.

From such an understanding of processual roles it becomes clear that par-
ticipation of $x$ in a process $y$, $\text{par}(x, y)$, can be defined in terms of role-of and
plays:

$$\forall xy(\text{par}(x, y) \leftrightarrow \exists z(\text{PR}(z) \land \text{plays}(x, z) \land \text{roleOf}(z, y)))$$

Social roles (SR) are defined as a sort of “complex properties” acquired by
the player; it must be said that in Loebe [2003, 2005] there is not a strong and
clear definition of what social roles are. This is mainly due to the ambiguity that
social roles have in various role accounts appeared in the literature. Social role is a
notion that tries to capture certain individual objects on a social ontological level,
hence possessing a dependence on other objects (like their players). Relations and
processes seem to be of primary relevance for social roles. Viewing patient as a
social role, with a partient ID and possibly some assignments in the form of rights,
norms or duties makes it hard to determine clear complements as compared to
relational roles in a patient-physician relation. Accordingly, for an understanding
of social roles, the context becomes rather vague and implicit, and the focus shifts
to the internals of social roles as well as to their relations to players, as discussed
above.

2.2.3 Abstract vs Social Roles

Loebe, after a deep analysis based on role types introduced in the previous section,
proposes an ontological division between abstract and social roles.

Due to their similarity, relational and processual roles are subsumed by a new
role type called abstract roles which is contrasted with social roles. Abstract roles
can be functionally characterized in a uniform manner, namely as a mechanism
of viewing some entity (i.e. the player) in a defined context. Put it differently,
players of abstract roles are looked at in an external manner in contrast to viewing
them as self-contained entities focusing on internals like their qualities or parts.
2.3 Roles in Agent Deliberation

This general reading of abstract roles is contrasted with social roles, which captures certain individual objects on a social ontological level, hence possessing a dependence on other objects. Due to being objects, social roles have their own qualities, relations, and processes in which they participate. Relations and processes seem to be of prior relevance for social roles. Viewing a patient as a social role, with a patient ID and possibly some assignments in the form of rights, norms or duties makes it hard to determine clear complements as compared to relational roles in a patient-physician relation. Indeed, social roles rather aggregate various relational and processual roles. Accordingly, for an understanding of social roles, the context becomes rather vague and implicit, and the focus shifts to the internals of social roles as well as to their relations to players, as discussed above.

Generalizing, we can say that abstract roles refers to a subset of players’ properties, whereas social roles are entities external to players with their own attributes and identity.

2.3 Roles in Agent Deliberation

In this section we introduce a widely acknowledged approach in defining roles’ dynamics, in particular we refer to Dastani et al. [2004] as the main source of the material exposed below. Most of the paragraphs below are a mere reformulation of Dastani et al. [2004]; thus the main contribution of this section is the introduction of four primitives: enact, deact, activate and deactivate, which are the actions that an agent can do in an open MAS in order to deal with roles.

The most important operations are enact and deact which mean that an agent starts and finishes to occupy a role, and activate and deactivate, which mean that an agent starts executing actions belonging to the role and suspends the execution of the actions. In Dastani et al. [2004], enacting role means internalizing the specification of the role, while activating a role means reasoning with the (internalized) specification of the role.

A role can be specified in terms of the information that becomes available to agents when they enact the role, the objectives or responsibilities that the enacting
agent should achieve or satisfy, and normative rules which can be used to handle these objectives.

### 2.3.1 Role Enactment and Role Activation

In the following, we assume a first order language $L$ and a set of basic actions $A$ based on which we define the belief language $L_B$, the goal language $L_G$, and the plan language $L_P$.

- $L_B = \beta ::= B\phi | \neg \beta | \beta \land \beta'$ for $\phi \in L$.
- $L_G = \kappa ::= G\phi | \neg \kappa | \kappa \land \kappa'$ for $\phi \in L$.
- $L_P = \pi ::= \alpha | \beta?; \pi; \pi'|\pi + \pi'|\pi || \pi'|\pi* \text{ for } \alpha \in A, \beta \in L_B$

Intuitively, $B\phi$ should be read as “believes $\phi$”, $G\phi$ as “has objective $\phi$”, $\beta?$ as “test if $\beta$”, $\pi; \pi'$ as “first do $\pi$ then do $\pi'$”, $\pi + \pi'$ as “choose either $\pi$ or $\pi'$”, $\pi || \pi'$ as “do $\pi$ and $\pi'$ simultaneously”, and $\pi*$ as “repeat doing $\pi$”. We don’t give a formal semantics of these languages since is not relevant for the purpose of this section.

Moreover, we assume various types of rules which can be used for various purposes. For example, as we will see in the context of role specifications, these rules can be used to specify different types of norms and obligations, and in the context of the agent specifications, they can be used to specify the dynamics of mental attitudes of agents such as modification or planning of objectives. For the purpose of this paper, we assume three different types of rules as specified below. The interpretation of these rules will be given when we define agent role and agent specification. Moreover, we do not claim that these types of rules are exhaustive, but we believe that they make sense for the purpose of enacting and deactivating roles by agents. The three types of rules are represented by the following three sets $PS$ (called plan selection rules), $GR$ (called goal revision rules), and $PR$ (called plan revision rules):

- $PS = \{\kappa \land \beta \Rightarrow \pi \mid \kappa \in L_G, \beta \in L_B, \pi \in L_P\}$
- $GR = \{\kappa \land \beta \Rightarrow \kappa' \mid \kappa, \kappa' \in L_G, \beta \in L_B\}$
• \( PR = \{ \pi \land \beta \Rightarrow \pi' \mid \pi, \pi' \in L_P, \beta \in L_B \} \)

In the following, we assume that roles are abstract entities which can be instantiated whenever they are enacted. Therefore, we use \( R_{name} \) to denote the set of names for role instantiations including a special name \( e \) for the passive role. We also use \( Rules \) to indicate the set of all triple of subsets of \( PS \), \( GR \), and \( PR \), i.e., \( Rules = 2^{PS} \times 2^{GR} \times 2^{PR} \).

### 2.3.2 Agent Roles and Agent Types

In this approach, we assume that a role determines the information that the enacting agent should have, the objectives that it should achieve, and the norms and obligations it has to fulfill. Here, we consider the objectives as the states that the agent wants to achieve.

**Definition 1** (Role) A role is a tuple \( \langle \sigma_i, \gamma_i, \omega_i \rangle \), typically denoted by \( r \), where \( \sigma_i \in L_B \) specifies the information that an agent receives when enacting this role, \( \gamma_i \in L_G \) specifies the objectives to be achieved by the agent that enacts this role, and \( \omega_i \in Rules \) be a triple consisting of rules representing conditional norms and obligations.

We assume that the objectives \( \gamma_i \) in the above definitions are achievement goals. Maintenance goals can be defined in terms of normative rules of the form \( \neg \kappa \land \top \Rightarrow \kappa \) which means that goals \( \kappa \) should be adopted whenever \( \kappa \) is not the case. A role can be incoherent in the sense that it may be specified in terms of inconsistent beliefs and goals. Also, normative rules that are ascribed to a role may suggest the adoption of inconsistent objectives. One may therefore introduce coherence conditions to exclude these cases.

**Definition 2** (Role coherency) Let \( \omega_i = (\omega^{PS}, \omega^{GR}, \omega^{PR}) \in Rules \). A role \( r = \langle \sigma_i, \gamma_i, \omega_i \rangle \) is coherent, denoted as coherent\((r)\), iff:

1. \( \sigma_i \nmodels \bot \): (consistent beliefs)
2. \( \gamma_i \nmodels \bot \): (consistent objectives)
3. \( \sigma_i \nmodels \gamma_i \) if \( \top \nmodels \gamma_i \): (non-trivial objectives are not achieved)
4. \( \bigwedge (\kappa \land \beta \Rightarrow \kappa') \notin \bot: \text{potential objectives are mutually consistent} \)

5. \( \forall (\kappa \land \beta \Rightarrow \kappa') \in \omega^{GR}: \kappa' \land \gamma_i \notin \bot: \text{potential objectives are consistent with role’s objectives} \)

An agent can enact different roles during its execution (one actively at a time) and enacting a role influences its mental attitudes. The type of an agent determines the roles that the agent can enact. Therefore, we informally require that the roles that an agent can enact should be mutually consistent (their mental attitudes, plans and rules are not in contradiction) since these roles influence the agent’s mental attitudes. We usually refer \( t \) as a generic agent type.

### 2.3.3 Role Enacting and Role Deacting Agents

In this section, we assume that role enacting agents have their own mental attitudes consisting of beliefs, goals plans, and rules that may specify their conditional mental attitudes as well as how to modify their mental attitudes. In addition, a role enacting agent is assumed to enact a set of roles among which only one of them is active at each moment in time; all other enacted roles are inactive. In Dastani et al. [2004] only one role can be active at each moment in time; all other enacted roles are deactivated. This is because (cognitive) agent has one single reasoning process, also called the agent’s deliberation, that determines the behavior of the agent based on the enacted (internalized) roles. One single reasoning process cannot be based on two or more enacted roles at the same time. Which role should be reasoned with at each moment in time is thought to be the agent’s decision.

Role enacting agents have distinct objectives and rules associated to the active role it is enacting, and sets of distinct objectives and rules adopted from enacted but inactive roles. The roles enacted by an agent are instantiations of the roles specified in the agent type \( t \). This can be compared to objects which are instantiations of classes. It is therefore possible that one role from \( t \) is enacted and instantiated several times. We call an agent with its own mental attitudes, an active role instantiation, a set of inactive role instantiations, and a type, a role enacting agent.
2.3 Roles in Agent Deliberation

Definition 3 (role enacting agent: rea) Let $\gamma_a \in L_G$, $\gamma_r \in L_G \times Rname$, and $\gamma \subseteq L_G \times Rname$. Let $\Pi_a \subseteq L_P \times L_G$ and $\Pi_r \in 2^{L_P \times L_G} \times Rname$, $\Pi_s \subseteq 2^{L_P \times L_G \times Rname}$. Let $\omega_a \in \text{Rules} \times Rname$, $\omega \subseteq \text{Rules} \times Rname$, and $e \in Rname$ be a special role instantiation name for passive role (not active). Then, a role enacting agent is a tuple $\langle \sigma, \Gamma, \Pi, \Omega, t \rangle$, where:

- $\sigma \in L_B$ specifies rea’s beliefs.
- $\Gamma = (\gamma_a, \gamma_r, \gamma)$ specifies rea’s objectives.
- $\Pi = (\Pi_a, \Pi_r, \Pi_s)$ specifies rea’s plans.
- $\Omega = (\omega_a, \omega_r, \omega)$ specifies rea’s rules.
- $t$ is the agent type.

A passive-role enacting agent (p-rea) is defined as a rea where $\Gamma = (\gamma_a, (\top, e), \gamma)$, $\Pi = (\Pi_a, (\emptyset, e), \Pi_s)$, and $\Omega = (\omega_a, ((\emptyset, \emptyset, \emptyset), e), \omega)$.

In the above definition, $\gamma_a$ and $\omega_a$ specify the agent’s own objective and rules, respectively. Moreover, $\gamma_r$ and $\omega_r$ specify respectively the objectives and rules associated to the active role that the agent enacts, and $\gamma$ and $\omega$ are sets of objectives and sets of rules of the enacted roles which are not active, respectively. Finally, $\Pi_a$ specifies agent’s own plans, $\Pi_r$ specifies the plans that are generated by the active role, and $\Pi_s$ specifies the plans of enacted but inactive roles. Note that an objective is associated with each plan to indicate the (initial) purpose of that plan. Also, a role instantiation name is associated with the objectives in $\gamma_r$ and $\gamma$, to the plans in $\Pi_r$ and $\Pi_s$, and with the sets of rules in $\omega_r$ and $\omega$. Finally, note that the last clause ensures that agent roles are consistent with the mental attitudes of the agent.

In Dastani et al. [2004], enacting a role by an agent means that the agent adopts the role (i.e., it adopts the information, objectives, and rules that are associated with the role) and uses a name to refer to the instantiation of this role. Enacting a role can be specified by a function that maps rea’, roles, and role instantiation names to rea’s. This function is defined on rea’s in general, rather than on coherent rea’s. In proposition below, we relate this function and the notion of coherent rea’s.
2.3 Roles in Agent Deliberation

**Definition 4** (Role enacting function) Let $S$ be the set of rea’s, $\langle \sigma, \Gamma, \Pi, \Omega, t \rangle \in S$, $R$ be the set of roles, $\langle \sigma_i, \gamma_i, \omega_i \rangle \in R$, and $r_i \in \text{Rname}$ be a role instantiation name. The role enacting function $F_{\text{enact}} : S \times R \times \text{Rname} \rightarrow S$ is defined as follows:

$$F_{\text{enact}}(\langle \sigma, \Gamma, \Pi, \Omega, t \rangle, \langle \sigma_i, \gamma_i, \omega_i \rangle, r_i) = \langle \sigma \land \sigma_i, \Gamma', \Pi', \Omega', t \rangle$$

where

$\Gamma = (\gamma_a, \gamma_r, \gamma)$ and $\Gamma' = (\gamma_a, \gamma_r, \gamma \cup \{ (\gamma_i, r_i) \})$

$\Omega = (\omega_a, \omega_r, \omega)$ and $\Omega' = (\omega_a, \omega_r, \omega \cup \{ (\omega_i, r_i) \})$.

An agent may decide to deactivate a role which means that the agent stops enacting the role. In Dastani et al. [2004], the agent that deacts a role will remove the objective and plans adopted by enacting the role.

**Definition 5** (Role deactivating function) Let $S$ be the set of rea’s, $\langle \sigma, \Gamma, \Pi, \Omega, t \rangle \in S$, and $r_i \in \text{Rname}$ be a role instantiation name. The role deactivating function $F_{\text{deact}} : S \times \text{Rname} \rightarrow S$ is defined as follows:

$$F_{\text{deact}}(\langle \sigma, \Gamma, \Pi, \Omega, t \rangle) = \langle \sigma, \Gamma', \Pi', \Omega', t \rangle$$

where

$\Gamma = (\gamma_a, \gamma_r, \gamma)$ and $\Gamma' = (\gamma_a, \gamma_r, \gamma \setminus \{ (\gamma_i, r_i) \mid \gamma_i \in L_G \})$

$\Pi = (\Pi_a, \Pi_r, \Pi_s)$ and $\Pi' = (\Pi_a, \Pi_r, \Pi_s \setminus \{ (X, r_i) \mid X \in 2^{L_P \times L_G} \})$

$\Omega = (\omega_a, \omega_r, \omega)$ and $\Omega' = (\omega_a, \omega_r, \omega \setminus \{ (\omega_i, r_i) \mid \omega_i \in \text{Rules} \})$.

### 2.3.4 Activating and Deactivating Roles

In Dastani et al. [2004], enacting a role does not imply activating the role. However, since enacting a role updates the belief base of the rea, the enacted role will indirectly influence the behavior of the role enacting agent. In order to direct the role enacting agent to achieve the role’s objectives, the enacted role should be activated. In fact activating a role is selecting and processing it. For this reason, we introduce two new functions for activating and deactivating agent roles. The role activating function maps passive-role enacting agents to role enacting agents. The objectives, plans, and rules of the enacted role become active entities and will affect the behavior of the role enacting agent.
2.3 Roles in Agent Deliberation

**Definition 6** (Role activating function) Let $S$ be the set of rea’s, $S^e$ be the set of passive-role enacting agents, $(\sigma, \Gamma, \Pi, \Omega, t) \in S^e$, $R$ be the set of roles, $(\sigma_j, \gamma_j, \omega_j) \in R$, and $r_i \in Rname$. The role activating function $F_{activate} : S^e \times R \times RName \rightarrow S$ is defined as follows:

$$F_{activate}(\langle \sigma, \Gamma, \Pi, \Omega, t \rangle, (\sigma_j, \gamma_j, \omega_j), r_i) = \langle \sigma \wedge \sigma_j, \Gamma', \Omega', t \rangle$$

where

- $\Gamma = (\gamma_a, (\top, e), \gamma)$, where $(\gamma_i, r_i) \in \gamma$
- $\Gamma' = (\gamma_a, (\gamma_i, r_i), \gamma \cup \{(\gamma_i, r_i) | \gamma_i \in L_G\})$
- $\Pi = (\Pi_a, (\emptyset, e), \Pi_a)$
- $\Pi' = (\Pi, (X, r_i), \Pi_s \cup \{(X, r_i) | X \in 2^{L_P \times L_G}\})$
- $\Omega = (\omega_a, (\emptyset, \emptyset, \emptyset, e), \omega)$, where $(X, r_i) \in \omega$
- $\Omega' = (\omega_a, (X, r_i), \omega' \cup \{(X, r_i) | \omega \in Rules\})$

The role deactivating function on the contrary maps role enacting agents to passive-role enacting agents. In fact, the activated enacting role may consist of objectives that are not achieved and plans that are not executed. These entities are saved and can be activated once again.

**Definition 7** (Role deactivating function) Let $S$ be the set of rea’s, $(\sigma, \Gamma, \Pi, \Omega, t) \in S$, $S^e$ be the set of passive-role enacting agents and $r_i \in Rname$. The role deactivating function $F_{deactivate} : S \times Rname \rightarrow S^e$ is defined as follows:

$$F_{deactivate}(\langle \sigma, \Gamma, \Pi, \Omega, t \rangle, r_i) = \langle \sigma, \Gamma', \Pi', \Omega', t \rangle$$

where

- $\Gamma = (\gamma_a, (\gamma_i, r_i), \gamma)$ and $\Gamma' = (\gamma_a, (\top, e), \gamma \cup \{(\gamma_i, r_i)\})$
- $\Pi = (\Pi_a, (X, r_i), \Pi_s)$ and $\Pi' = (\Pi_a, (\emptyset, e), \Pi_s \cup \{(X, r_i)\})$
- $\omega = (\omega_a, (X, r_i), \omega)$ and $\Omega' = (\omega_a, (\emptyset, \emptyset, \emptyset, e), \omega \cup \{(X, r_i)\})$

The functions defined above represent the formal semantics of $encat$, $deact$, $activate$ and $deactivate$ actions. In Chapter 4 we define four operations in a running example based on such actions.
Chapter 3

A Meta-model for Roles

3.1 Introduction

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.

3.2 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. The model is structured in three levels: universal, individual and dynamic.
3.2 A Logical Model for Roles

3.2.1 Universal Level

At the *universal* level we describe the relationship between natural and role types\(^1\).

**Definition 8** An *universal model* is a tuple

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

where:

- \(D\) is a domain of classes\(^2\). For each class \(C\) is possible to refer to its attributes and operations through \(\pi_{\text{Attr}}(C)\) and \(\pi_{\text{Op}}(C)\) respectively.

- \(\text{Contexts} \subseteq D\) 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 functions and 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.

\(^1\)Natural types refer to the essence of the entities whereas role types depend on an accidental relationship to some other entity (context).

\(^2\)Here the term *class* should be taken rather broadly, generally we refer to classes as elements that compose the universal level, words like *categories of types* are interchangeable at this level.
3.2 A Logical Model for Roles

- **OS ⊆ D x Op**: it is an operation assignment relationship, through which we can assign to each class its operations.

- **RH ⊆ 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 ⊆ 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 \( \geq_{CH} \). If \( c <_{CH} c' \), we say that \( c \) inherits from \( c' \).

At this point we can add into **Constraints** some logical rules in order to model different role notions. For example in powerJava each role type is linked with one and only one context type [Baldoni et al., 2006a], so we can express this through the following constraint:

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

3.2.2 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 9** A **snapshot model** is a tuple

\[
< O, l_{\text{contexts}}, l_{\text{players}}, l_{\text{roles}}, \text{Sessions}, \text{Val}, l_{\text{constraints}} \\
|_{\text{Roles}}, l_{\text{attributes}}, l_{\text{operations}}, l_{\text{Attr}} >
\]

where:

- **O** is a *domain* of objects which instantiate classes in D, for each object \( o \) is possible to refer to its attributes and operations through \( \pi_{l_{\text{Attr}}}(o) \) and \( \pi_{l_{\text{Op}}}(o) \), respectively.

- **l_{\text{contexts}} ⊆ O** is a set of institutions which instantiate classes in **Contexts**.
3.2 A Logical Model for Roles

- \( I_{\text{players}} \subseteq O \) is a set of (potential) actors, which instantiate universals in Players.

- \( I_{\text{roles}} \subseteq O \) is a set of roles instances. This set is empty if the model is contrained defining roles as elements that cannot be instantiated.

- Sessions is a set of sessions, which keeps the state of an interaction between \( I_{\text{players}} \) and \( I_{\text{contexts}} \) (See Section 3.3).

- Val is a set of values.

- \( I_{\text{constraints}} \) is a set of integrity rules that constraint elements in the snapshot.

We often 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 as argument an attribute instance \( d \in I_{\text{attributes}} \) and , if it exisits, it returns its associated value \( v \in \text{Val} \); \( \emptyset \) 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 \( x :: Y \) when \( x \) is an instance of \( Y \). In general if \( x :: Y \) attributes and operations defined for \( Y \) at the universal level are assigned to \( x \). If \( a \in \pi_{\text{Attr}}(Y) \) we write \( x.a \in I_{\text{attributes}} \) as the attribute instance assignet to object \( x \), the same holds for elements in \( I_{\text{Operations}} \). Intuitively, a snapshot represents the state of a system at a certain particular point in time.

The role assignment function \( I_{\text{Roles}} \) 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, \( s \) a session, and \( o :: R \) a role, such that \((R,X) \in \text{RO} \) and \((Y,R) \in \text{PL} \), \( (i,a,s,o) \in I_{\text{Roles}}(R) \) 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 \( o \)
3.2 A Logical Model for Roles

the role instance. When it is not interesting to talk about sessions we can write \( R(i,a,o) \). It is important to underline that if we want roles not to be instatiable the \( I_{Roles} \) must be seen in a slightly different way, therefore we write \( R(i,a,s) \) to express that \( a \) plays \( R \) in \( i \).

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

\[
salary(employee) = 1000
\]

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 players being people holding certain positions within such insitution. But the model is intended to capture a much wider range of phenomenons: institutions may be folders in a file system or any object structured in roles , and actors its users, operations or attributes their permissions, and roles as 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 \texttt{lover} and \texttt{lovee} filled by players.

But in many applications, the relationship between roles, institutions, players, and the resulting qua-individuals will be more subtle, and the properties of the qua-individuals will depend on combinations of the properties of the different ingredients that make up a role. A simple example of such an interaction may be the policy of assigning email addresses to users or a certain website by combining their name with the name of a domain, to obtain adressses such as ‘\texttt{boella@di.unito.it}’. This cannot really be expressed in a general way in our present system (as a consequence of our choice to keep the model simple), but for this specific case, the constraint would look like this:

\[
\text{User(di.unito.it, guido_boella, x) → email(x) = boella@di.unito.it}
\]

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 in $I_{\text{constraints}}$ 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 $\text{Roles}$. 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 [Baldoni et al., 2006a], but also for social roles [Masolo et al., 2004]. The opposite 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, $I_{\text{Roles}}$ maps $R$ to $I_{\text{contexts}} \times I_{\text{players}} \times X$.

2. *Combinations of Roles.* In powerJava, each actor can play a role at most one time within a single institution, i.e.

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

It is this assumption that allows for the use of ‘role casting’ in powerJava to refer to role instances: an expression of the form $(i,R.s)a$ can be used to denote the unique object $x$ such that $R(i,a,s,x)$.

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 $R \neq R'$:

$$R(i,a,s,x) \rightarrow \neg R'(i,a,s,x)$$
3.2 A Logical Model for Roles

3. 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 $I_{\text{contexts}}$ contains only this trivial institution, as we do in Chapter 5 when we model RBAC [S. Sandhu & J. Coyne, 1996].

4. 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:

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

Which means that in the snapshot exists $y \in I_{\text{contexts}}$ 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 $y$.

5. 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:

$$R(i, a, o) \leftrightarrow R'(i, b, x)$$

3.2.3 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.

Definition 10 A dynamic model is a tuple

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

where:
3.2 A Logical Model for Roles

- $S$ is a set of snapshots.

- $TM \subseteq S \times \mathbb{N}$: it is a time assignment relationship, such that each snapshot has an associated time $t$. For the sake of simplicity we define a discrete time through positive natural numbers.

- **Actions** is a set of actions.

- **Requirements** is a set of requirements for playing roles in the dynamic model $^1$.

- **D.contstraints** is a set of integrity rules that contraints the dynamic model.

- **IActions** maps each action from **Actions** to a relation on a set of snapshots $P \subset S$. $I_{Actions}(s, a, t)$ tells us which snapshots are the result of executing action $a$ at time $t$ from a certain snapshot $^2$. 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 $I_{Roles_t}$ we say that $R_t(i, a, o)$ is true if there exists, at a time $t$, the role instance $R(i, a, o)$.

- $\pi_{Req}(t, R, x)$ returns a subset of **Requirements** present at a given time $t$ for the role $R$, which are the requirements that must be fulfilled in order to play $R$ for actor $x$ $^3$.

- $I_{Requirements_t}$ 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 $Req_t(i, a, R)$.

---

$^1$Elements in **Requirements** could be seen as predicates that must be true in order to be fulfilled.

$^2$Notice that given an action, we can have several snapshots because we model actions with modal logic in which, from a world it is possible to go to more than one other possible world. This property is often formalized through the accessibility relationship. Thus, each snapshot can be seen as a possible world in modal logic.

$^3$So in this view, $\pi_{Req}(t, R, x)$ gives the set of predicates that must be true in order to respect the requirements.
Intuitively, the snapshots in S represent the state of a system at a certain time. Looking at $l_{\text{Actions}}$ 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.

Given a time $t$ we can, analyzing the correspondent snapshot, deduce many properties of the system, like all the roles played by an actor $a$ at that time, or all the player of a certain role $R$ or even the number of role instances associated with a specific actor.

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 snapshots 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 $\exists t$ operator, such that $\exists t(p)$ is true, if and only if $p$ exists in at least one snapshot associated with time $t$, false otherwise. We write $\exists(p)$ when $p$ exists in all the snapshots of the dynamic model.

Actions are described using dynamic modal logic [Baldoni et al., 2007], in particular they are modelled through precondition laws and action laws of the following form:

$$\square(A \land B \land C \supset \langle d \rangle \top)$$  \hspace{1cm} (3.1)

$$\square(A' \land B' \land C' \supset [d]E)$$  \hspace{1cm} (3.2)

Where the $\square$ operator express that the quantified formulas hold in all the possible words. Precondition law (3.1) specifies the conditions $A,B$ and $C$ that make an atomic action $d$ executable in a state. (3.2) is an action law\footnote{Sometimes action laws are called effect rules because $E$ can be considered the effect of the execution of $d$.} which states that if preconditions $A',B'$ and $C'$ to action $d$ holds, after the execution of $d$ also $E$ holds.

In addition we introduce complex actions which specify complex behaviors by means of procedure definitions, built upon other actions. Formally a complex
3.2 A Logical Model for Roles

action has the following form:

\[ \langle p_0 \rangle \varphi \subset \langle p_1; p_2; \ldots; p_m \rangle \varphi \]

\( p_0 \) is a procedure name, “;” is the sequencing operator of dynamic logic, and the \( p_i \)’s, \( i \in [1, m] \), are procedure names, atomic actions, or test actions\(^1\).

Now we show some examples of actions that can be introduced in the dynamic model in order to specialize the framework.

3.2.3.1 Role Addition

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 \( a \) of the role \( R \) in institution \( i \).

For role addition and deletion actions we use, respectively \( R, i \leftarrow_t a \), and \( R, i \leftarrow_t a \). Then using the notation of dynamic logic introduced above, we write:

\[ \Box (\text{Req}_t (i, a, R) \supset (R, i \leftarrow_t a) \top) \]

to express that, if actor \( a \) fills the requirements at time \( t \) (\( \text{Req}_t (i, a, R) \) is True), \( a \) can execute the role addition action that let him play role of type \( R \).

The above definition gives us the possibility to model that a role assignment introduces a role instance:

\[ \Box (\top \supset [R, i \leftarrow_t a] \exists x R_{t+1} (i, a, x)) \]

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

\[ \Box (\neg \exists x R (i, a, x) \supset [R, i \leftarrow_t a] \exists! x R_{t+1}(i, a, x)) \]

\(^1\)Test actions are of the form \( \langle \psi? \rangle \varphi \equiv \psi \land \varphi \).
3.2.3.2 Role Assignment

In this subsection we give an example of how a snapshot $M$ can change after the execution of a role addition action, we suppose this action to be deterministic\(^1\) and also that roles are instantiated.

Let $M$ be a snapshot.

$$M = \langle O, I_{\text{contexts}}, I_{\text{players}}, I_{\text{roles}}, \text{Sessions}, \text{Val}, I_{\text{Roles}}, I_{\text{attributes}}, I_{\text{operations}}, I_{\text{Attr}} \rangle$$

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 it fails, or it succeeds. In the latter case, the resulting snapshot:

$$M' = \langle O', I_{\text{contexts}}', I_{\text{players}}', I_{\text{roles}}', \text{Sessions}', \text{Val}', I_{\text{Roles}}', I_{\text{attributes}}, I_{\text{operations}} \rangle$$

should satisfy the following properties:

- A role assignment adds to the domain of object the new role instance $r :: R$.
  $$O' = O \cup \{r\}, \text{where } r.$$

- $I_{\text{contexts}}' = I_{\text{contexts}}$.

- $I_{\text{players}}' = I_{\text{players}}$ or $I_{\text{players}}' = I_{\text{players}} \cup \{a\}$. This depends on if $a$ was playing or not other roles before $r :: R$.

- $I_{\text{roles}}' = I_{\text{roles}} \cup \{r\}$, $\text{Val}' = \text{Val}$. The sets of roles instances acquire the newly introduced $r$, otherwise possible values of attributes do not change.

- $I_{\text{Roles}}'(R) = I_{\text{Roles}} \cup \{(R, (i, a, o))\}$. The domain of the function $I_{\text{Roles}}$ is extended respecting $R(i, a, o)$.

- $I_{\text{attributes}}'$ and $I_{\text{operations}}'$ extends $I_{\text{attributes}}$ and $I_{\text{operations}}$ with attributes and operations of $r :: R$.

- $I_{\text{Attr}}'$ is just like $I_{\text{Attr}}$.

- $\text{Sessions}'$ has all $\text{Sessions}$ elements plus the new attributes and operations of the introduced role instance, the session ID can be new or not, depending on what type of interaction we want to model (Section 3.3).

---

\(^1\)Only one snapshot stems from this execution
3.2.3.3 Specializing the Dynamic Model

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:

\[ R_t(i, a, s, x) \rightarrow \pi_{\text{I-Attr}}(x) \supseteq \{ x.b \mid b \in \pi_{\text{Attr}}(R) \} \]

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 [Baldoni et al., 2006a], no such inheritance is assumed at all - the properties of the role instance are precisely those of the role, and we have that:

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

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 [Albano et al., 1993]. 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:

\[ R_t(i, a, x) \rightarrow \{ x.b \mid b \in \pi_{\text{Attr}}(R) \} \subset \pi_{\text{I-Attr}}(a) \]

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:

\[ \square (\top \supset [R, i \leftarrow a](\text{l-attributes}_{t+1} = \text{l-attributes}_t \cup \{ a.b \mid b \in \pi_{\text{Attr}}(R) \})) \]

The same considerations hold for operations.

3.2.3.4 Object Deletion

An object does not exist after deleting it:

\[ \square (\top \supset [\text{delete}_t(o)] \neg \text{exists}(o)) \]
3.2 A Logical Model for Roles

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

\[ \Box (\top \supset \text{[delete}_t(i)] \neg R_{t+1}(i, a, s, x)) \]
\[ \Box (\top \supset \text{[delete}_t(a)] \neg R_{t+1}(i, a, s, x)) \]
\[ \Box (\top \supset \text{[delete}_t(x)] \neg R_{t+1}(i, a, s, x)) \]

For example in Depke et al. [2001] we have that “A role (instance) will be deleted when the agent is destroyed, i.e., its lifetime is dependent on that of the base agent.”:

\[ \Box (R_t(i, a, s, x) \supset \text{[delete}_t(a)] \neg \exists x s_{t+1}(x)) \]

3.2.3.5 Role Deletion

Let $i \in l\_contexts$, $a \in l\_players$, and $R \in l\_roles$. Role deletion has different consequences depending on whether the role instances have their own identity or not. In the first case role deletion could be defined in the following way:

\[ [R, i \leftarrow_t a] \varphi \equiv \text{[delete}_t(x)] \varphi \]

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:

\[ R(i, a, s, x) \rightarrow a = x \]

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, that in our model are expressed through attributes and operations. A possible constraint that represents such type of inheritance is, (section 2.2):

A way to formalize the fact that an actor relinquishes a role without an identity is:

\[ \Box (\top \supset [R, i \leftarrow_t a]| \text{l\_attributes}_{t+1} = \text{l\_attributes}_i \setminus \{a.b \mid b \in \pi_{\text{Attr}}(R)\} \land \text{l\_operations}_{t+1} = \text{l\_operations}_i \setminus \{a.d \mid d \in \pi_{\text{Attr}}(R)\})) \]

The above formula expresses that an actor who stops playing a role loses all the Attr and Op acquired by the role R.
3.2 A Logical Model for Roles

3.2.3.6 Methods

There are other actions through which it is possible to change the model as well, for instance agents may assign new values to their attributes [Genovese, 2007]. 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’s values can be changed by the objects themselves. What we will do is to define methods as special actions that represent objects that change their own attributes or those of others. Actually, to simplify the model, we define one single primitive action: \texttt{set}_t(o_1, o_2, \texttt{attr}, v), which means that object \textit{o}_1 sets the value of \texttt{attr} on object \textit{o}_2 to \textit{v} at time \textit{t}.

Now, we will of course have that:

\[ \Box(\top \supset [\texttt{set}_t(o_1, o_2, \texttt{attr}, v)] \texttt{attr}_{t+1}(o_2) = v) \]

which means that in any state, after the execution of \texttt{set}, if the action of setting this attribute succeeds, then the relevant object will indeed have this value for that attribute.

3.2.3.7 Powers

One observation of the work [Boella & van der Torre, 2007] is that certain aspects of the notion of power can be captured by how features of one agent can be changed by 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 objects. Or, formally, \textit{o}_1 has power over \textit{o}_2 if and only if:

\[ \langle \texttt{set}_t(o_1, o_2, \texttt{attr}, v) \rangle \top \]

It is important to underline that \textit{o}_1 can have power over \textit{o}_2 in three situations:

\[ R(o_1, x, o_2) \lor (R(i, x, o_2) \land R(i, x, o_1)) \lor R(o_2, x, o_1) \lor o_1 = o_2 \]

so \textit{o}_1 and \textit{o}_2 can be role instances or institutions.

Putting everything in the same formula we have:
3.3 Sessions and Relationships

\( \square (R(o_1, x, o_2) \lor (R(i, x, o_2) \land R(i, x, o_1)) \lor R(o_2, x, o_1) \land o_1 = o_2 \supset \langle \text{set}_t(o_1, o_2, \text{attr}, v) \rangle \top) \)

In Boella & van der Torre [2007], 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.

3.2.3.8 Operations

Elements of our framework come with operations that can be executed at the individual level in order to change the model dynamically, the semantics of each operations can be given exploiting the actions defined for the dynamic model. Suppose, for instance, to have an object individual \( x :: \text{Person} \) with \( x.\text{mail_address} \) attribute, and an operation \( x.\text{change\_mail} \) that changes the value of \( x.\text{mail\_address} \) to its argument. Using the \text{set} primitive is possible to define how the model evolves after the execution of \( x.\text{change\_mail} \) operation trough the following axiom:

\[ [x.\text{change\_mail}_t(s)]\varphi \equiv [\text{set}_t(x, x, \text{mail\_adress}, s)]\varphi \]

3.3 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 is necessary to keep the state of an interaction between entities. Sessions in our model are represented by mean of 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 mantains a particular information on the state of the interaction between an actor playing a role and an
3.3 Sessions and Relationships

institutions offering them. Operations in $K$ are behavioural aspects 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 they are of different objects. For instance, suppose we 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 [Baldoni et al., 2006a; Herrmann, 2005; Loebe, 2005] ($ID = x$). This means that attributes and operations in $K$ are all a subset of $\pi_{Attr}(x) \cup \pi_{Op}(x)$ where $x \in I_{\text{roles}}$.

2. A session can collapse into the actor [S.Sandhu & J.Coyne, 1996; Steimann, 2000] ($ID = a$). 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 behaviour of another role individual $b$.

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

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

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:

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

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

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) [Steimann, 2000].
In Figure 3.1 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 \( x :: Person, y :: Person, \text{tutor} :: Tutor \) and \( \text{student} :: Student \)

\begin{enumerate}
  \item \( \text{Tutor}(y, x, q, \text{tutor}) \land \text{Student}(x, y, q', \text{student}) \)
  \item \( \text{Tutor}(y, x, q, \text{tutor}) \land \text{Student}(x, y, q, \text{student}) \)
\end{enumerate}

Notice that \( x \) and \( y \) are both in I\text{\_contexts} and I\text{\_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 \( y \rightarrow x \)).

\[ \text{It must be said that is not mandatory to model the interaction between } x \text{ and } y. \]

\[ ^1\text{In this section we refer to classes with the first letter capitalized in order to distinguish them from instances which are in lower case.} \]

\[ ^2\text{In this case is sensible to state that } \pi_{1D}(q) = \text{tutor} \land \pi_{1D}(q') = \text{student}. \]

\[ ^3\text{In this view } q \text{ is an object with its own ID that embodies elements of both tutor and student.} \]
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 3.1) 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 3.2 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 approach 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
3.3 Sessions and Relationships

seen as set of affordances [Baldoni et al., 2006b] 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 3.1 representing it through a set of contraints at the individual level:

\[ Tutor(students, teacher, s_1, tutor) \land \]
\[ Faculty\_Member(faculty, teacher, s_2, faculty\_member) \land \]
\[ Lecturer(course, teacher, s_3, lecturer) \land Student(teacher, student, s_1, student) \land \]
\[ Participant(course, student, s_4, participant) \land \]
\[ Faculty(teacher, faculty, s_2, faculty) \land \]
\[ Taken\_Course(student, course, s_4, taken\_course) \land \]
\[ Given\_Course(teacher, course, s_3, given\_course) \]

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 3.1. 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 modelling an interaction, and that the state and behaviour 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 3.2, 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

\(^{1}\)In order to avoid confusion we refer to \( teacher, student, course, and faculty \) as instances of the classes involved in the collaboration diagram.
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 \texttt{D._constraints} of the dynamic model:

\[
\forall z, p, q : \\
p :: Faculty \land q :: Student \land z :: Teacher \land \\
faculty\_member_t(p, z, s_1, x) \land tutor_t(q, z, s_2, y) \\
\rightarrow \\
num\_courses(x) = num\_courses(y) = \beta
\]

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 can not be simple labels of association ends.

\section*{3.4 An Explanatory Example}

In this section we would like to tackle the translation of a simple example into our metamodel. During the analyses of different role accounts we already mentioned the pivotal elements of the framework, by the way we think that reasoning with a single example could be of great utility to get confident with the meta-model. The idea is to translate the following UML object diagram:

\begin{center}
\begin{tikzpicture}
  \node [draw, thick, shape=rectangle] (Mario) at (0,0) {Mario:Person};
  \node [draw, thick, shape=rectangle] (Tom) at (2,0) {Tom:Person};
  \node [draw, thick, shape=rectangle] (trainer) at (1,1) {trainer};
  \node [draw, thick, shape=rectangle] (player) at (1,-1) {player};
  \draw (Mario) -- (trainer) node [midway, above] {};
  \draw (trainer) -- (player) node [midway, above] {};
  \draw (player) -- (Tom) node [midway, above] {};
\end{tikzpicture}
\end{center}

\subsection*{3.4.1 Universal Level}

We now verbosely define all the elements that compose the universal level:

\textit{Sets:}
3.4 An Explanatory Example

\[D = \{\text{Person, Trainer, Player}\}\]
\[\text{Contexts} = \{\text{Person}\}\]
\[\text{Players} = \{\text{Person}\}\]
\[\text{Roles} = \{\text{Trainer, Player}\}\]
\[\text{Attr} = \{\text{age, weight, hours_of_training}\}\]
\[\text{Op} = \{\text{train, ask_advices}\}\]
\[\forall x, y, z (x \in \text{Roles } y, z \in \text{Contexts } x\text{RO} y \land x\text{RO} z \rightarrow y = z) \in \text{Constraints}\]

\text{Relations:}
\[\text{PL} = \{\text{(Person, Trainer), (Person, Player)}\}\]
\[\text{RO} = \{\text{(Trainer, Person), (Player, Person)}\}\]
\[\text{AS} = \{\text{(Person, weight), (Person, age), (Person, hours_of_training)}\}\]
\[\text{OS} = \{\text{(Trainer, train), (Player, ask_advices)}\}\]
\[\text{RH} = \{\text{(Trainer), (Player)}\}\]

Notice that \text{Person} is at the same time a context and a player because it can play both \text{Player}, \text{Trainer} but it can also offer them to interact with another instance of \text{Person}.

### 3.4.2 Individual Level

Now we fill the individual level:

\text{Sets:}
\[O = \{\text{Mario, Tom, trainer, player}\}\]
\[\text{l_contexts} = \{\text{Mario, Tom}\}\]
\[\text{l_players} = \{\text{Mario, Tom}\}\]
\[\text{l_roles} = \{\text{trainer, player}\}\]
\[\text{Sessions} = \{\text{Training_session}\}\]
\[\text{l_attributes} = \{\text{Mario.age, Mario.weight, Mario.hours_of_training,}\]
\[\text{Tom.age, Tom.weight, Tom.hours_of_training}\}\]
\[\text{l_operations} = \{\text{trainer.train, player.ask_advices}\}\]
\[\text{Val} = \{23, 43, 72, 78, 0\}\]
3.4 An Explanatory Example

Functions:

\[ \text{I}_{\text{Roles}} = \{(\text{Player}, (\text{Mario, Tom, training}\_\text{session, player})), \\
\quad (\text{Trainer}, (\text{Tom, Mario, training}\_\text{session, trainer}))\} \]

\[ \text{I}_{\text{Attr}} = \{(\text{Tom}.\text{age}, 23), (\text{Mario}.\text{age}, 43), (\text{Tom}.\text{weight}, 78), \\
\quad (\text{Mario}.\text{weight}, 72), (\text{Tom}.\text{hours}\_\text{of}\_\text{training}, 0)(\text{Mario}.\text{hours}\_\text{of}\_\text{training}, \emptyset)\} \]

3.4.3 Dynamic Model

If we want to define the dynamic model extensively we should list all possible snapshots resulting from the execution of all possible actions. In general it would be sensible to represent only the actions which are relevant to understand the key element of the example being described. In this case we can constraint the model stating that all trainers must have been players, so that for every time \( t \) and \( x :: \text{Person} \), \( \pi_{\text{Req}}(t, \text{Trainer}, x) \) contains \( \text{player}_t(q, x, z) \) where \( q :: \text{Person} \) and \( z \) is the role instance.

We can also state that the \text{Trainer}'s role instance is created iff, before of the execution of the role addition's action, the actor had already been a player:

\[ \Box(\forall t' \leq t \text{ player}_t(Person, x, z) \supset \\
\quad ([\text{Trainer, Person} \hookrightarrow_t x] \exists y \text{ Trainer}_{t+1}(Person, x, y)) \]

Where \( x \) is an instance of \text{Person} (i.e. \( x :: \text{Person} \)).

At this stage we can also model the situation in which the trainer has the power to acknowledge the hours of training updating the respective attribute in \text{Tom}:

\[ \Box(\top \supset \forall x \geq 0 \\
\quad (\text{set(trainer, Tom, hours}\_\text{of}\_\text{training}, \text{I}_{\text{Attr}}(\text{Tom, hours}\_\text{of}\_\text{training}) + x)) \top) \]
Chapter 4

Comparing Different Role’s Accounts

4.1 Social Roles

The model introduced in Chapter 3 is able to describe portions of the social role’s properties identified in Masolo et al. [2004].

4.1.1 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 Universals can play the same role Universal”. In our model this is represented as:

\[
\{\text{Human, Frog}\} \subseteq \text{Players} \\
\text{Fantasy Village} \in \text{Contexts} \\
\text{Prince} \in \text{Roles} \\
\text{RO} = (\text{<Prince, Fantasy Village>}) \\
\text{PL} = (\text{<Human, Prince>,< Frog, Prince>})
\]

\(^1\)For an analysis at the Individual level see “A role can be played by different entities, simultaneously”.

48
2. Roles are anti-rigid and they have “dynamic” properties:

- **An entity can change role**: At the 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 the role instance is a concept founded on three elements: the player, the context and the session in which the interaction takes place. In this view it is possible to have an actor $a$ that interacts with $c$ through two role instances $x$ and $y$ of the same universal $R$, each interaction is identified by the two different sessions $s$ and $s'$ associated respectively with $x$ and $y$. However it remains the fact that 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 also 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 within the same institution.

- **A role can be played by different entities, simultaneously**: This sentence can be translated in the following way: “Two player individuals (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:

$$R_t(i,a,o) \land \neg R_t(i,b,c) \land 
\neg R_{t'}(i,a,o) \land R_{t'}(i,b,c)$$

3. **Roles have a relational nature**: Here the term role is defined as a *founded* concept. In general, $\alpha$ is founded on a property $\beta$ if, necessarily, any *definition* of $\alpha$ ineliminably involves $\beta$, which is external to $\alpha$. For instance,
in our model the role class \( R \) is definitionally dependent on another entity \( C \) if the RO relation has a couple \(< R, C >\) where \( C \) is a context. If we want to represent that all roles are founded on a unique context, we can add the following constraint:

\[
R \in \text{Roles} \iff \exists ! C \in \text{Contexts} : < R, C > \in \text{RO}
\]

### 4. Roles are linked to contexts:
At the universal level we have the RO relation that binds roles and contexts.

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. Therefore we introduce role instances only through \( I_{\text{Roles}} \), such that if we have \( R(i, a, s, x) \) we say that the role instance \( x \) is strictly dependent on \( i, a \) and \( s \). In fact if we delete only one of the elements linked to \( x \) in \( I_{\text{Roles}} \) we have the immediate expiration of \( x \). Also anti-rigidity is a key property of the model, hence an object \( a \) playing a role \( R \) maintains its identity even after role instance \( R(i, a, o) \) ceases to exists. In other words we can represent the following integrity rule:

\[
AR(R) = \forall a, o, t(R_t(i, a, o) \rightarrow \exists t' (\text{exists}_{t' < t}(a) \land \neg R_{t'}(i, a, o)))
\]

Where AR stands for anti-rigidity.

### 4.2 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’ identifying the weaknesses which arise from others modelling approaches. To formalise his approach he defines a role-oriented modelling language called LODWICK [Steimann, 2000].

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
4.2 Steimann’s Approach

taking the place held by the role filled). We already showed in Section 3.3 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 Lod-
wick’s role concept, and then show how his approach could be modelled in our
framework. To do this several role’s features are taken from different works in
literature by Steimann and then discussed from the LODWICK’s 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 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 prop-
erties 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:

\[ R(i, a, x) \rightarrow a = x \]

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

Where the first formula states that role instances have no identity, and the
second one expresses 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\(^1\):

\[ R(i, a, x) \rightarrow (\text{attr}(x) = v \leftrightarrow (\text{attr}(R) \lor \text{attr}(a) = v)) \]

2. Roles depend on relationships: “Yes. Roles occupy the places of relation-
ships, and the relative part of a role’s intension captures which relationships
an object must participate in to be considered playing the role.”

\(^1\)The same holds for Op.
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 let the actor interact with the institution. Informally, we can say that the role instance implicitly defines a one-way association (actor $\rightarrow$ 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:

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

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

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

$RO = (\langle \text{husband, Woman} \rangle, \langle \text{wife, Man} \rangle)$

$PL = (\langle \text{Man, husband} \rangle, \langle \text{Woman, wife} \rangle)$

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:

$\forall x, y \text{ husband}(\text{Caterina, Mario, x}) \leftrightarrow \text{wife}(\text{Mario, Caterina, y})$

This relationship could be depicted in the following way:

![Diagram of Mario and Caterina engaging in a two-way relationship through the roles husband and wife.]

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 and 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:

$$R(i, a, x) \land R'(j, a, y) \land x \neq y \land i \neq j$$

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 institutions so that:

$$R(i, a, x) \land R(j, a, y) \land i \neq j$$

5. **An object may acquire and abandon roles dynamically:** “Yes. Roles are assumed by an object as associations with that object are added, and relinquished as associations are removed from the dynamic extensions of relationships.”
4.2 Steimann’s Approach

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 responsibility of the dynamic model.”

This is quite a subtle subject, but we can handle it exploiting the Requirements set. Suppose that we are in a bank 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 \( \pi_{\text{Req}}(t, \text{director}, \text{Leonard}) \) contains the following predicate:

\[ \text{employee}_t(\text{bank}, \text{leonard}, o) \]

7. Objects of unrelated types can play the same role: “Yes. This is one of the cornerstones of Lodwick’s role formalization; it follows from the definition of the role-filler relations linking the type and the role hierarchy.”

This point can also be easily 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 having an element in both Players and Roles sets, we can let \( \text{Players} \cap \text{Roles} = \emptyset \) 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. In the second case, 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 role instances 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, it is possible to model that if an actor plays a role $a$ 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 restricts 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 and 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.”

Instead, we can have: $R(i, a, o) \rightarrow a \neq o$. 
As explained in S. Sandhu & J. Coyne [1996], starting in the 1970s, computer systems featured multiple applications and served multiple users, leading to heightened awareness of data security issues. System administrators and software developers alike focused on different kinds of access control to ensure that only authorized users were given access to certain data or resources. One kind of access control that emerged is role-based access control (RBAC).

---

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
- $AU \subseteq U \times AR$: user-administrative role assignments
- $PA \subseteq P \times R$: permission-role assignments
- $AP A \subseteq AP \times AR$: 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:
  $$roles(s) \subseteq \{ r : R \mid (\exists r' \geq r) \cdot ([user(s), r'] \in UA \cup AU A) \}$$
- $permissions : R \rightarrow 2^{P \cup AP}$, 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.
4.3 RBAC Model

A role is chiefly a semantic construct forming the basis of access control policy. With RBAC, system administrators create roles according to the job functions performed in a company or organization, grant permissions (access authorization) to those roles, and then assign users to the roles on the basis of their specific job responsibilities and qualifications.

A role can represent specific task competency, such as that of a physician or a pharmacist. A role can embody the authority and responsibility of, say, a project supervisor. Authority and responsibility are distinct from competency. A person may be competent to manage several departments but have the responsibility for only the department actually managed. Roles can also reflect specific duty assignments rotated through multiple users, for example, a duty physician or a shift manager. RBAC models and implementations should conveniently accommodate all these manifestations of the role concept.

Roles define both the specific individuals allowed to access resources and the extent to which resources are accessed. For example, an operator role might access all computer resources but not change access permissions; a security-officer role might change permissions but have no access to resources; and an auditor role might access only audit trails.

There are a few elements which need 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 [S. Sandhu & J. Coyne, 1996] 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
permissions $\subseteq \text{Attr}$.

- **Sessions**: Users establish sessions during which they may activate a subset
  of the roles they are enable to play. Each session maps one user to pos-
sibly 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 associa-
tion 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 exist 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 $R$:

$$R(i,a,s,x) \rightarrow (\exists v: \text{attr(x)} = v \iff (\text{attr(R)} \vee \text{attr(a)} = v))$$
• 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 discriminates who and how can change the universal level.

4.4 Zope Model

Zope is an open source application server for building content management systems, intranets, portals, and custom applications. With Zope you can update your web site from web interface. To allow this, Zope also features a tightly integrated security model. Built around the concept of “safe delegation of control”, Zope’s security architecture also allows you to turn control over parts of a web site to other organizations or individuals.

Zope is a multi-user system. In Zope, users have only the capabilities granted to them by a Zope security policy. As the administrator of a Zope system, one has the power to change your Zope system’s security policies to whatever suits his business requirements.

Once a user has been authenticated, Zope determines whether or not he has access to the resource which is being protected. This process is called authorization. The process of authorization involves two intermediary layers between the user and the protected resource: roles and permissions.

Users have roles which describe “what they can do” such as “Author”, “Manager”, and “Editor”. These roles are controlled by the Zope administrator. Users may have more than one role, and may have a different set of roles in different
contexts. Zope objects have permissions which describe “what can be done with them” such as “View”, “Delete objects”, and “Manage properties”.

A context in Zope is a “place” within the Zope object hierarchy. In essence, a context can be thought of as an object’s “location” within the Zope Object Database, described by its “path”. Each object that exists in the Zope Object Database which has a web-manageable interface can be associated with its own security policy. In fact, most Zope objects acquire their security policies from their containers because it makes a given security policy easier to maintain. Only when there are exceptions to the “master” security policy in a context are individual objects associated with a differing policy. In essence, security policies map roles to permissions in a context; in other words they say “who” can do “what”, and “where”.

For example, the security policy for a Folder (the context) may associate the “Manager” roles (the role) with “Delete objects” permission (the permission). Thus, this security policy allows managers to delete objects in this folder. If objects created within this folder do not override their parents’ security policy, they acquire this policy. Subobjects within subfolders of the original folder have the same policy unless they override it themselves.

As already said, in Zope users have one or more roles, once an user gets an authorization it enters the system with at least one role active (for example Manager), even if an user enters the system without being authorized he has a role active (anonymous).

We already talked about roles and permissions during the RBAC section, the main difference is that in RBAC permission are directly attached to roles while Zope has a security policy which dynamically binds permissions to roles depending on which context the user is playing the role.

It seems that roles in Zope are just like user pools, playing a role implies being part of a set of users which obtain permissions through the security policy. We can see the security policy as a function \( \rho \) that, giving a role and a context returns all the permissions assigned to the role player within the specific context.

In RBAC the fact that a role has a permission was modelled through the presence of a specific attr in the role instance, this was sufficient because once an attribute was defined at the universal level for the role class, each role instance
had that permission no matter of the context. With Zope is different, different role instances of the same role universal in different contexts can have totally different permissions. To model this in a simple way we introduce for instance a special attr that we call perm. The attribute perm can be seen as a bit map. Each permission is represented by one bit. If a permission is active in the context linked with the role instance the bit is 1, 0 otherwise.

For example suppose that a Zope system has three permission defined: “View”, “Delete object” and “Manage properties” (without losing generality we suppose that the set of properties is an ordered set), and that Tom enters the system playing the role “Web mantainer”. In this case Tom will access object in Zope only through the Web mantainter role. Now suppose that for Web maintainer in context “Folder” the permission associated are “View” and “Manage properties”. When Tom will interact with “Folder” creating the role instance x of Web mantainer, perm in x will be such that:

\[ \text{perm}(x) = 101 \]

Where the first bit says that View is activated, the second one that delete object is not and the third that manage properties in asssociated with that role instance.

Zope offers the possibility to dynamically define new permissions for a certain object in the hierachy, now suppose that the administrator defines a new permission \( \alpha \) for the object “Folder”, \( \alpha \) is a totally new permission, hence we add a bit to perm so that if \( x \) is the role instance of Web mantainer and \( \alpha \) is activated we write:

\[ \text{perm}(x) = 1011 \]

Where the fourth bit represents the newly added property \( \alpha \). Every interaction in Zope is modelled through a Zope Object (context) so we are sure that Tom interacting in a Zope system will always play one or more roles.

In Zope, permissions flow through hierarchies is not linear, there could be points in the hierarchies where inheritance of the security policy is no longer valid, or where the “master” security policy is redefined. To model how permission are transmitted throw the object hierarchy, we use the session concept, such that each role instance linked with the same player share the same session. The set
of integrity rules in such sessions will define how perm’s value of different role instances are set depending on the security policy\textsuperscript{1}. In other words we can say that permission flow in Zope is state dependent, in the sense that is so flexible that cannot be defined in a static way as hierarchies impose, to define such a flow we need to reason about each singular state of the interaction between the user and the Zope object (i.e. the state of the role instance).

\textsuperscript{1}The idea to merge security policy with session’s concept should be explained in a more complete way.
Chapter 5

Modeling Roles in Agent Organizations

5.1 Introduction

In the last years, the usefulness of roles in designing agent organizations has been widely acknowledged. Witness not only the papers appeared at AAMAS, IAT, but also the creation of specialized workshops which have agent organizations (COIN, ROLES, AOSE, NorMAS, etc.) among their topics.

Many different models have been designed. Some of them use roles only in the design phases of a MAS [Zambonelli et al., 2003], while other ones consider roles as first class entities which exist also during the runtime of the system [Colman & Han, 2007]. Some of them focuses on how roles are played by agents [Dastani et al., 2004], other ones on how roles are used in communication among agents in organizations [Boella et al., 2006].

This heterogeneity of the way roles are defined and used in MAS risks to be a danger for the interoperability of agents in open systems, since each agent entering a MAS can have a radically different notion of role. Thus, the newly entered agents cannot be governed by means of organizations regulating the MAS. Imposing to all agent designers a single notion of role is a strategy that cannot have success. Rather, it would be helpful to design both multiagent infrastructures that are able to deal with different notions of roles, and to have agents which are
able to adapt to open systems which use different notions of roles in organizations. This alternative strategy can be costly if it is not possible to have a general model of role that is compatible, or can be made compatible with other existing concepts.

In this chapter we generalize and merge two models of roles used in multiagent systems, in order to promote the interoperability of systems. The research question is: How to combine the model of role enactment by Dastani et al. [2004] with the model of communication among organizational roles of Boella et al. [2006]?

We answer these questions by extending to agents a metamodel of roles developed for object oriented systems Genovesi [2007]. The relevant questions, in this case, are: how to introduce beliefs, goals and other mental attitudes in objects, and how to pass from the method invocation paradigm to the message passing paradigm.

Then we specialize the metamodel to model two existing approaches and we show how they can be integrated in the metamodel since they deal with complementary aspects. We choose to model the proposals of Boella et al. [2006] and Dastani et al. [2004] since they are representative of two main traditions. The first tradition is using roles to model the interaction among agents in organizations, and the second one is about role enactment, i.e., to study how agents have to behave when they play a role.

From one side, organizational models are motivated by the fact that agents playing roles may change, for example a secretary may be replaced by another one if she is ill. Therefore, these models define interaction in terms of roles rather than agents. In Boella et al. [2006] roles model the public image that agents build during the interaction with other agents; such image represents the behavior agents are publicly committed to. However, this model leaves unspecified, how given a role, its player will behave. This is a general problem of organizational models which neglect that when, for example, a secretary falls ill, there are usually some problems with ongoing issues (the new secretary does not know precisely the thing to be done, arrangements already made etc.). So having a model of enacting and deacting agents surely leads to some new challenges, which could not be discussed, simulated or formally analyzed without this model.
5.2 Agents and Roles

In contrast, the organizational view focuses on the dynamics of roles in function of the communication process: roles evolve according to the speech acts of the interactants, e.g. the commitment made by a speaker or the commands made by other agents playing roles which are empowered to give orders. In this model roles are modeled as sets of beliefs and goals which are the description of the expected behavior of the agent. Roles are not isolated, but belongs to institutions, where constitutive rules specify how the roles change according to the moves played in the interactions by the agents enacting the roles.

Dastani et al. [2004] focuses, instead, on how roles are played by an agent, and, thus, on the private aspects of roles. Given a role described in terms of beliefs, goals, and other components, like plans, their model describe how these mental attitudes become the beliefs and goals of the agents. In this model roles are fixed descriptions, so they do not have a dynamics like in the model of Boella et al. [2006]. Moreover, when roles are considered inside organizations new problems for role enactment emerge: for example, how to coordinate with the other agents knowing what they are expected to do in their role, and how to use the powers which are put at disposal of the player of the role in the organization. The same role definition should lead to different behaviors when the role is played in different organizations.

In contrast, it specifies the internal dynamics of the agents when they start playing (or enacting in their terminology) a role or shift the role they are currently playing (called the activated role). So they model role enacting agents: agents that know which roles they play, the definitions of those roles, and which autonomously adapt their mental states to play the roles.

Despite the apparent differences, the two approaches are compatible since they both attributes beliefs and goals to roles. So we study by means of the metamodel how they can be combined to have a more comprehensive model of roles.

5.2 Agents and Roles

Since the aim of this chapter is to build a metamodel to promote interoperability, we make minimal assumptions on agents and roles.
The starting point of our proposal is a role metamodel for object orientation. The relation of objects and agents is not clear, and to pass from object to agents we take inspiration from the Jade model [Bellifemine et al., 2001].

Agents, differently than objects, do not have methods that can be invoked starting from a reference to the object. Rather, they have an identity and they interact via messages. Messages are delivered by the MAS infrastructure, so that agents can be located in different platforms. The messages are modeled via the usual send-receive protocol. We abstract in the metamodel from the details of the communication infrastructure (whether it uses message buffers, etc.).

Agents have beliefs and goals. Goals are modeled as methods which can be executed only by the agent itself when it decides to achieve the goal.

As said above, we propose a very simple model of agents to avoid controversial issues. When we pass to roles, however, controversial issues cannot be avoided.

The requirements to cope with both models of roles we want to integrate are:

- Roles are instances, associated in some way to their players.
- Roles are described (at least) in terms of beliefs and goals.
- Roles change over time.
- Roles belong to institutions, where the interaction among roles is specified.
- The interaction among roles specifies how the state of roles changes over time.

In Boella et al. [2006] roles are used to model interaction, so agents exchange messages according to some protocol passing via their roles. This means that the agent have to act on the roles, e.g., to specify which is the move the role has to play in certain moment. Moreover, roles interact with each other.

Dastani et al. [2004]’s model specifies how the state of the agent changes in function of the beliefs and goals of the roles it plays. However, it does not consider the possibility that the state of the role change and, thus, it ignores how the agent becomes aware of the changes of beliefs and goals of the role.

To combine the two models we have to specify how the interaction between an agent and its role happens when the agent changes the state of the role or
the state of the role is changed by some event. A role could be considered as an object, and its player could invoke a method of the role. However, this scenario is not possible, since the roles are strictly related to the institution they belong to, and we cannot assume that the institution and all the agents playing roles in the institution are located on the same agent platform. So method invocation is not possible unless some sophisticated remote method invocation infrastructure is used. Moreover, the role have to communicate with its player when its beliefs and goals are updated. Given that the agent is not an object, the only possibility is that a role sends a message to its player. As a consequence, we decide to model the interaction between the agent and the role by means of messages too.

Finally, we have to model the interaction among roles. Since all roles of an institution belongs to the same agent platform, they do not necessarily have to communicate via messages. To simplify the interaction, we model communication among roles by means of method invocation.

The fact that roles belong to an institution has another consequence. According to the powerJava model of roles in object oriented programming languages, roles, seen as objects, belong to the same namespace of the institution. This means that each role can access the state of the institution and of the sibling roles. This allows to see roles as a way to specify coordination [Baldoni et al., 2006c].

In a sense, roles are seen both as objects, from the internal point of view of the institution they belong to, and as agents, from the point of view of their players, with beliefs and goals, but not autonomous. Their behavior is simply to:

- Receive the messages of their players.
- Execute the requests of their player of performing the interaction moves according to the protocol allowed by the institution in that role.
- Send a message to their players when the interaction move performed by the role itself or by some other role results in a change of state of the role.
5.3 Roles in Multiagent Systems

Since here we have been talking about objects as cornerstone of the individual level, now in order to switch from objects to agents, it must be underlined that an object of the meta-model does not necessarily overlap with object in OO programming. We used the terms object to refer to individuals, and terms like attribute and operation to talk about state and behavior of an entity.

In order to be as much general as possible, we define elements of the meta-model with only those features that are essential to talk about roles and leave the possibility to specify the abstract model depending on which account of role we want to grasp. This approach gives us the possibility to talk about object and agent using the same framework, and specifying each time which are the characteristics of role’s player. In moving from objects to agents we need to state the following:

- **Attributes** are complex properties of the agent which describe its internal features as well as its mental attitudes (belief, goals, plans etc.).
- **Operations** are actions that the agent does in the system.
- **Agents** at individual level are supposed to be autonomous so they cannot be forced to execute an action from an external entity.
- The only way to interact between agents is through message passing.
- The system in which agents interacts is represented by a unique institution.
- **Role instances** are linked with one and only one system. In order to express this point we add into the constraint of every snapshot the following integrity rule:
  \[ r \in \text{I}\_\text{roles} \iff \exists! c \in \text{I}\_\text{contexts} : <r, c> \in \text{I}\_\text{RO} \]

For the sake of generality, we prefer not to specify how agents reason on the basis of their mental attitudes; what we want to model is how mental attitudes evolve as a consequence of playing a role and what are the elements on which the agent have to carry out its reasoning process.
It is important to understand that the meta-model is not a framework for agent specification, the elements listed above are the basic features that we think are fundamental to talk about role in MAS, but of course they are not sufficient to utterly model agents.

5.4 Enact and Deact Roles

In Dastani et al. [2004], the problem of formally defining the dynamics of roles, is tackled identifying the actions that can be done in a open system such that agents can enter and leave. In this setting roles have existence outside the agents in the implemented system, so “agents are not completely defined by the roles they play” [Dastani et al., 2004]. This view leads to a definition of roles that sees them as strictly linked with a system (context), instantiable and with their own proper identity.

In Dastani et al. [2004] four operations to deal with role dynamics are defined: enact and deact, which mean that an agent starts and finishes to occupy (play) a role in a system, and activate and deactivate, which means that an agent starts executing actions (operations) belonging to the role and suspends the execution of the actions.

Although is possible to have an agent with multiple roles enacted simultaneously, only one role can be active at the same time.

Before diving into modeling the four basic operations to deal with roles, we need to match with our framework a few concept defined in Dastani et al. [2004], following we report a list of elements together with their definition and then how they fit in our meta-model:

- Multiagent system: In Dastani et al. [2004] roles are taken into account at the implementation level of open MAS, they belong to the system which can be entered or left by agents dynamically. In our framework is possible to view a system as a context to which are linked all roles that can be played by the agents.
5.4 Enact and Deact Roles

- **Agent role**: A role is a tuple \(\langle \sigma, \gamma, \omega \rangle\). Where \(\sigma\) are beliefs, \(\gamma\) goals and \(\omega\) rules representing conditional norms and obligations. This definition specifies a role “in terms of the information that becomes available to agents when they enact the role, the objectives or responsibilities that the enacting agent should achieve or satisfy, and normative rules which can for example be used to handle these objectives” [Dastani et al., 2004]. With this view we define, for roles of our framework, a set of complex attributes \{beliefs, goals, plans, rules\} \(\in I_{Attr}\) together with the operations that represent actions that an agent can carry out when it activates the roles instance choosing it from the set of roles it is playing.

- **Agent type**: We consider an agent type “as a set of agent roles with certain constraints and assume that an agent of a certain type decides itself to enact or deact a role”. To talk about agent types we use classes introduced in the framework as a specification of agent instances at the individual level, with this in mind we use the PL relationship to link agent classes to agent roles (role’s classes) so that the set of roles that an agent can enact (play), is constrained by \(I_{PL}\).

- **Role enacting agent**: “We assume that role enacting agents have their own mental attitudes consisting of beliefs, goals, plans, and rules that may specify their conditional mental attitudes as well as how to modify their mental attitudes. Therefore, role enacting agents have distinct objectives and rules associated to the active role it is enacting, and sets of distinct objectives and rules adopted from enacted but inactive roles”. In our framework we define a role enacting agent as a instance \(x\) having a set of attributes \(A\) that represent the internal structures used to deliberate.

\[
A = \{\text{beliefs}_a, \text{objectives}_a, \text{plans}_a, \text{rules}_a, \text{enacted\_roles}[], \text{active\_role}\} \in \pi_{I_{Attr}}(x)
\]

The \text{enacted\_roles} attribute is a role ordered record where each entry with index \(i\) corresponds to a triple \(\langle \sigma_i, \gamma_i, \omega_i \rangle\) which represents the set of beliefs, objectives, plans and rules associated to roles instance \(i\) enacted by \(x\).
5.5 The Public Dimension of Roles

As introduced above, the model in Dastani et al. [2004] identifies four operations to deal with role dynamics, in order to grasp the fundamental ideas proposed in the cited paper, we redefine the \textit{enact}, \textit{deact}, \textit{activate} and \textit{deactivate} operations respecting their original meaning. Given a role enacting agent $x$, a role instance $i :: R$ played by $x$ in context $c$ such that,

$$\text{beliefs}_r, \text{objectives}_r, \text{plans}_r, \text{rules}_r \in \pi_{\text{Attr}}(i)$$

$$\{\text{beliefs}_a, \text{objectives}_a, \text{plans}_a, \text{rules}_a, \text{enacted\_roles}[i], \text{active\_role}\} \in SA \pi_{\text{Attr}}(x)$$

Next we report the semantics of each operation exploiting the \texttt{set} primitive:

$$\langle x.\text{enact}_t(i) \rangle \varphi \subset \langle R, s \leftarrow x; \text{set}_t(x, x, \text{beliefs}_a, \text{beliefs}_a \cup \text{beliefs}_r) \rangle$$

$$\langle x.\text{deact}_t(i) \rangle \varphi \subset \langle R, s \leftarrow x; \text{set}_t(x, x, \text{enacted\_roles}[i], \text{null}) \rangle$$

$$\langle x.\text{activate}_t(i) \rangle \varphi \subset \langle \text{set}_t(x, x, \text{active\_role}, \text{enacted\_roles}[i]) \rangle$$

$$\langle x.\text{deactivate}_t(i) \rangle \varphi \subset \langle \text{set}_t(x, x, \text{active\_role}, \text{null}) \rangle$$

In order to be coherent it must be respected a logical order in the execution of these operations, as in Dastani et al. [2004]:

- each operation $\text{deact}(i)$ is preceded by a $\text{enact}(i)$.
- each operation $\text{deactivate}(i)$ is preceded by only one instruction $\text{activate}(i)$ that is not preceded by another $\text{activate}(j)$.

5.5 The Public Dimension of Roles

In Boella & van der Torre [2007] roles are introduced inside institutions to model the interaction among agents. In Boella et al. [2006] the model is specifically used to provide a semantics for agent communication languages in terms of public mental attitudes attributed to roles.

The basic ideas of the model are:
5.5 The Public Dimension of Roles

- Roles are instances with associated beliefs and goals attributed to them. These mental attitudes are public.

- The public beliefs and goals attributed to roles are changed by speech acts executed either by the role or by other roles. The former case accounts for the addition of preconditions and of the intention to achieve the rational effect of a speech act, the latter one for the case of commands or other speech acts presupposing a hierarchy of authority among roles.

- The agents execute speech acts via their roles.

This model has been applied to provide a semantics to both FIPA and Social Commitment approaches to agent communication languages Boella et al. [2006]. This semantics overcome the problem of the unverifiability of private mental attitudes of agents.

- In order to maintain the model simple enough, we model message passing extending the dynamic model with two actions (methods) send\((x, y, sp)\) and receive\((y, x, sp)\). Were send\((x, y, sp)\) should be read as the action carried by \(x\) of sending a speech act \((sp)\) to \(y\) and similar considerations hold for the receive action.

- A role only listens for the messages sent by the agents playing it:

\[
\langle \text{listen}(r) \rangle \varphi \subset \langle \text{P; played_by}(r, x)?: \text{receive}(r, x, sp); \text{D}\rangle \varphi
\]

These rules define a pattern of protocol where \(P\) and \(D\) have to be read as possible other actions that can be executed before and after the receive.

- The reception of a message from the agent has the effect of changing the state of other roles. For example, a command given via a role amounts to the creation of a goal on the receiver if the sender has authority (within the system) over it\(^1\).

\(^1\)request\((r, r’, act)\) is a speech act that has to be read as following: role \(r\) asks to \(r’\)’s player to do \(act\).
\[ \Box (\text{authority}_{sys}(r, \text{request}) \supset [\text{receive}(r, x, \text{request}(r, r', \text{act}))] \mathcal{G}_t' (\text{act})) \]

- To produce a speech act, the agent has to send a message to the role specifying the illocutive force, the receiver and the content of the speech act:

\[ \langle \text{communicate}(a) \rangle \varphi \subset \langle P; \text{send}(x, r, \text{sp}); D \rangle \varphi \]

### 5.6 The Combined Model

The two models presented above model complementary aspects of roles: the public character of roles in communication and how agents privately adapt their mental attitudes to the roles they play.

In this section we try to merge the two approaches using the metamodel we presented. On the one hand, the model of Boella et al. [2006] is extended from the public side to the private side, by using Dastani et al. [2004] as a model of role enacting. In this way, the expectations described by the roles resulting from the interaction among agents can become a behavior of agents and they do not remain only a description.

On the other hand, the model of Dastani et al. [2004] is made more dynamic. In the original model the role is given as a fixed structure. The goals of agent can evolve according to the goal generation rules contained in it, but the beliefs and goals described by the role cannot change. This is unrealistic, since during the activity of the agent enacting its role, it is possible that further information are put at disposal of the role and that new responsibilities are assigned, etc.

This problem can be overcome by the merging with the model of Boella et al. [2006] and by the addition of a further element, which is anyway necessary in Boella et al. [2006]'s model.

First of all, in Dastani et al. [2004] roles cannot change since they are not related to a more extensive context. Instead, in Boella et al. [2006], roles belong to institutions together with other roles. Sibling roles and the institution they belong to are the sources of changes for the role. Second, in Boella et al. [2006],
5.6 The Combined Model

The changes of roles are described by the effects of the speech acts which can be performed via roles. These two elements can be added to Dastani et al. [2004]'s model without apparent contradictions.

The missing element is that both models do not consider the problem of how the player of a role become aware of the changes in the state of the played role as a consequence of the actions of other roles. Furthermore, in Dastani et al. [2004] a role is given as known by the agent playing the role. This is not a realistic assumption, in particular, when the state of the role changes over time, but also the way an agent comes to know the initial state of the role must be explicitly modeled. Otherwise, all roles instances must be assumed to be publicly known in advance.

In order to merge the two models within the same framework, we need to add (complex) actions which are able to grasp the dynamics introduced in Boella et al. [2006] and Dastani et al. [2004]. Interactions among agents is done through message passing and, in particular, through actions send and receive introduced in section 6. Next we are going to introduce all the speech-acts and complex actions which are needed to grasp the combined model and then we introduce a running example to clarify their use defining a course of actions in the dynamic model defined in section 3.2.

An agent who wants to play a role within an open system has to ask to the system for a role instance; this process is handled by two speech act: ask_to_play(R) and accept_to_play(r, A), where the first one is sent from the agent to the system in order to ask to play a role of type R, whereas the second is sent from the system to the agent, together with the identifier of the role instance r and a set A of other role instances present in the system, in order to inform the agent with which roles is possible to interact. Next we report the two effect rules associated:

\[
\Box(\top \supset \text{[receive}(s, x, \text{ask}_r \text{to}_p \text{lay}(R)); \text{send}(s, x, \text{accept}_s \text{to}_p \text{lay}(r, A)])
\]

played_by_sys(r, x, s) \quad (5.5)

\[
\Box(\top \supset \text{[send}(x, s, \text{ask}_r \text{to}_p \text{lay}(R)); \text{receive}(x, s, \text{accept}_s \text{to}_p \text{lay}(r, A)])
\]

played_by_ag(r, x, s)) \quad (5.6)
Where $s$ is the system, $x$ the agent, and $r$ a role instance of type $R$. In this section we use $x, y, z \ldots$ to denote agents, $s$ for the system and $r, r', r'' \ldots$ for role instances. Notice that $\text{played}_\text{by}_\text{sys}(r, x, s)$ and $\text{played}_\text{by}_\text{ag}(r, x, s)$ refer to two different infrastructures; in Rule 5.5 is the system that, after having acknowledged the agent request, knows that $x$ is going to play $r$, whereas in Rule 5.6 is the agent that becomes aware of the play relation between $x$ and $r$. To link the two predicates with the logical model introduced in Section 3 we have that:

$$\text{played}_\text{by}_\text{sys}(r, x, s) \land \text{played}_\text{by}_\text{ag}(r, x, s) \rightarrow R(s, x, r)$$

When we are dealing with a single system we can omit $s$ writing $\text{played}_\text{by}_\text{sys}(r, x)$ and $\text{played}_\text{by}_\text{ag}(r, x)$.

To enact a role, an agent, provided the identifier of the role instance it wants to enact, has to send a message to the role and to wait till the role replies with the information about the state of the role: its beliefs, goal, plans, etc. When the state is received, the agent can enact the role in the same way described by Rule 5.1 in Section 5. In order to model such interaction we introduce two complex actions $\text{tell}_\text{enact}$, $\text{accept}_\text{enact}$ and two speech acts $\text{accept}_\text{enact}$ and $\text{inform}_\text{enact}$. Following the specification of the complex actions:

$$\langle \text{tell}_\text{enact}(x, r) \rangle \varphi \subset \langle \text{played}_\text{by}_\text{ag}(r, x)\rangle; \langle \text{send}(a1, r1, \text{enact}(x, r)) \rangle \varphi \quad (5.7)$$

$$\langle \text{accept}_\text{enact}(r, x) \rangle \varphi \subset \langle \text{receive}(r, x, \text{enact}(x, r)); \text{played}_\text{by}_\text{sys}(r, x)\rangle; \langle \text{send}(r, x, \text{inform}_\text{enact}(<\text{beliefs}, \text{objectives}, \text{plans}, \text{rules}>)) \rangle \varphi \quad (5.8)$$

When the agent receives the specification of the role he wishes to enact, it can internalize them as in Rule 5.1:

$$\square (\top \supset [\text{receive}(x, r, \text{inform}_\text{enact}(<\text{beliefs}, \text{objectives}, \text{plans}, \text{rules}>))])$$

$$\mathcal{B}^n(\text{beliefs}) \land x.\text{enacted_roles}[r] = <\text{objectives}, \text{plans}, \text{rules}>)^1 \quad (5.9)$$

In this combined view is possible that role’s specifications change dynamically, in that case it is up to the role to send a message to its player each time its state is updated:

$$\langle \text{update}_\text{state}(r, x) \rangle \varphi \subset \langle \text{played}_\text{by}_\text{sys}(r, x)\rangle; \langle \neg G^t_1(q) \land G^{t+1}_1(q) \rangle; \langle \text{send}(r, x, \text{inform}_\text{goal}(q)) \rangle \varphi \quad (5.10)$$
Last but not least, we need to model the deactment of a role respecting the formalization as in Rule 5.2, therefore we introduce two speech acts deact, ok_deact and a complex action confirm_deact defined as follows:

\[
\langle \text{confirm\_deact}(r, x) \rangle \varphi \subset \langle \text{receive}(r, x, \text{deact}); \text{played\_by} \text{sys}(r, x)?; \\
\text{send}(r, x, \text{ok\_deact}) \rangle \varphi
\]

(5.11)

After sending the ok_deact, the system will not consider anymore agent x as player of r:

\[
\Box (\top \supset [\text{confirm\_deact}(r, x)] \neg \text{played\_by} \text{sys}(r, x))
\]

(5.12)

If it is possible for the agent to deact the role, it will receive an ok_deact from its role:

\[
\Box (\top \supset [\text{receive}(x, r, \text{ok\_deact})] \text{x\_enacted\_roles}[r] = \text{null} \land \neg \text{played\_by}\text{ag}(r, x))
\]

(5.13)

Fig. 5.1 depicts two agents which interact through roles in an open system. At time t the system has already agent_B that enacts role r2 as represented by the black arrow which goes from agent_B to r2. The system evolves as following:

- At time t+1 agent_A asks to institution system_C to play a role of type R1:

\[
\text{send}_{t+1}(\text{agent\_A, system\_C, ask\_to\_play(R1)})
\]
5.6 The Combined Model

- At time $t+2$ system$_C$ replies to agent$_A$ assigning to him the role instance r1:
  \[\text{send}_{t+2}(\text{system}_C, \text{agent}_A, \text{accept}\_\text{to}\_\text{play}(r1, \{r2\}))\]

- At time $t+3$ agent$_A$ wants to enact (internalize) role r1:
  \[\text{tell}\_\text{enact}_{t+3}(\text{agent}_A, r1)\]

- At time $t+4$ role r1 receives the speech act from agent$_A$ asking for enactment and accepts it, replying to agent$_A$ with its specifications:
  \[\text{accept}\_\text{enactment}_{t+4}(r1, \text{agent}_A)\]

- Once that agent$_A$ has enacted the role as in Rule 5.1 it decides, at time $t+5$, to activate it \(^1\) and then to ask to the agent playing r2 to do an action act. In other words:
  \[\text{send}_{t+5}(\text{agent}_A, r1, \text{request}(r1, r2, \text{act}))\]

When r1 receives a send from agent$_A$ asking for an act of r2, first it checks if the sender has the authority in the system to ask such an act, if so r2 acquires the goal to do act:

\[\Box (\text{authority}_{\text{sys}}(r', \text{act}) \supset [\text{receive}(r, \text{agent}_A, \text{request}(r, r', \text{act}))] G' (\text{act}))\]

Is important to underline that because role internals are public to other roles in the same system, it is always possible for r1 to check or modify r2’s goals. So, at time $t+6$ we have:
\[\text{receive}_{t+6}(r1, \text{agent}_A, \text{request}(r1, r2, \text{act}))\]

- Now that r2 has updated its internal state (i.e. its goals) it must inform its player agent$_B$:
  \[\text{update}\_\text{state}_{t+7}(r2, \text{agent}_B)\]

Where update\_state is modelled as in Rule 5.10

---

\(^1\)Activating a role means to take into account its specification during the private agent deliberation process, so there is no need to introduce a public action in the dynamic model to represent the activation of a role.
• At time $t+8$ agent $A$ decides to deact the role $r_1$:

$$\text{send}_{t+8}(\text{agent}_A, r_1, \text{deact})$$

• Finally, at time $t+9$, $r_1$ confirm the deact:

$$\text{confirm} \text{\_} \text{deact}_{t+9}(r_1, \text{agent}_A)$$
Appendix A

Can roles play roles?

Roles types depend on an accidental relationship to some other entity (the context, the player). The idea to let roles play roles stems from the observation that an employee (a role played by a person) can become a project leader during its existence, where project leader is played by the role employee and only indirectly is a role of person.

Roles are conceptual elements used to model interactions, in fact, they are strictly linked with relationships; in Genovese [2007] we make a stronger statement: roles refer to the state of an interaction. This view underlines that roles are the right tool to model a relationship and that participants in a relationship are roles and not natural types. With this view in mind it seems at least fuzzy to let the state of an interaction (a role) participate in another interaction in another context.

A.1 Role as a Founded Concept

In ontology theory [Guarino & Welty, 2001] a role is a founded concept, which means that its existence relies on something which is external to it (the player and the context [Genovese, 2007; Loebe, 2005]), this implies that when the player ceases to exist the same happens for the role being played; for instance when a student graduates, his university ID ceases to exist. Role players are instances of natural types which are characterized by semantic rigidity and lack of foundation [Guarino, 1992], for example Person is a natural type, because an individual, if
A.2 The Counting Problem

a person, will always remain (and always has been) a person, and being a person is independent of the existence of any relationship. Student is a role since to be a student enrollment in a university is required, and finishing studies does not lead to a loss of identity [Steimann, 2000].

If we accept the view that roles play roles we have to reject the above ontological definition; in fact think of the following situation in which Mario, who is an employee for a big software house, becomes a project leader. If we think of employee playing the role project leader and then suppose that one day Mario stops playing the role employee because it has to concentrate itself on being a project leader, at this point the player of project leader ceases to exist but not the role played because Mario is still a project leader. We have violated the foundation of roles.

A.2 The Counting Problem

Role individuals allow to solve the famous counting problem. The problem refers to an appropriate understanding of a combination of sentences like the following:

1. Air company Y serves 900 passengers a month.
2. Every passenger is a person.
3. Air company Y servers 100 persons a month.

At first sight it seems that these three sentences are incoherent one with each other; the solution is to know what we are counting in each sentence. In the first one we refer to the number of roles individuals, whereas in the third one we count the number of player individuals of the role passenger; so if Mario goes from Turin to Tokyo and then he comes back, the company serves two passengers (two role individuals) but only one person (Mario). The second sentence has to be read at the universal level: every role passenger must be linked with a player (of type) person [Loebe, 2005].

Consider for example a soldier of an army that becomes a general, by letting the role soldier be the player of the role general. The same holds for the role
A.3 Conclusions

Now we can have the following survey written by the general for his superiors:

- The royal army is composed as follows:
  - 300 captains,
  - 5 general,
  - 1 million soldiers,
  - total: 1 million persons.

This view if not completely wrong is at least counterintuitive. The problem is that every general or captain is played by another role soldier which is itself played by a person, and when we count roles instead of player individuals we raise a contradiction.

A.3 Conclusions

We argue that by letting roles playing roles we do not add expressive power to the role notion, on the contrary we put in discussion the foundation of roles and add ambiguity in the evaluation of the counting problem.
References


REFERENCES


Colman, A. & Han, J. (2007). Roles, players and adaptable organizations. *Applied Ontology*. 64


REFERENCES


