A New Definition for the Concept of Role, and Why it Makes Sense

Guy Genilloud
EPFL
DSC-ICA
CH-1015 Lausanne
guy.genilloud@epfl.ch

Alain Wegmann
EPFL
DSC-ICA
CH-1015 Lausanne
alain.wegmann@epfl.ch

Presented at the OOPSLA workshop on behavioral semantics, Oct.15 2000

Abstract

There is widespread agreement in the object community that the concept of role is important for object modelling, but little agreement about what is to be understood by "a role". In this paper, we present a new definition for the concept of role in the context of ISO's RM-ODP Foundations for object modelling. We show that the concept of role is similar to that of interface, but that there important differences between these two concepts. We also provide definitions for concepts, related to the role concept, that may also be called roles: role type and role object type. We then make the case for our definitions, showing that they are largely compatible with assertions that exist in the literature about roles.

1. Introduction

There is widespread agreement in the object community that the concept of role is important for object modelling, but there is little agreement about what is to be understood by "a role." In a recent survey paper, Friedrich Steimann shows that the academics are very divided on this topic [12]. Standard bodies are also facing the issue of finding a proper definition for the concept of role. In particular, some members of the ISO group SC7/WG17, who is working on a standard for the ODP Enterprise Language, have found that the current definition of role in the RM-ODP Foundations [18, Def. 9.14] is very obscure if not flawed, so much that it does not help them. Within the OMG, the UML Revision Task Force found itself involved in a long and intense thread of email discussions after Jürgen Boldt of the OMG sent a message on August 11, 1999 where he said "*So what are roles? It seems that in UML they are not types (or classifiers), but a positive definition (other than the equally vague glossary entry) would seem imperative!*" (see for example the contribution by Trygve Reenskaug [10]). To our knowledge, these discussions remained rather inconclusive and await a proper resolution. The OMG has very recently set up an ad-hoc working group with mission to write a reflection paper on the topic of roles [16].

Our purpose with the present paper is firstly to present a new definition for the concept of role in the context of ISO's RM-ODP Foundations for object (with the hope that this definition will also be of interest to the OMG community), and secondly to show that this definition corresponds to expectations or understanding that people have about the concept of role. This paper is organized as follows. Section 2 presents the current RM-ODP definition of role, and our proposal for a new definition of role. Section 3 then discusses what makes roles similar to interfaces, and what makes them different. Section 4 provide definitions for related concepts: role type, and role object type. Finally, Section 5 shows that our definitions are largely compatible with the 15 assertions that Friedrich Steimann has collected in his survey paper [12].

2. Definitions of Role and Interface

2.1 The current definition of interface in the RM-ODP

Interface: *An abstraction of the behaviour of an object that consists of a subset of the interactions of that object together with a set of constraints on when they may occur. Each interaction of an object belongs to a unique interface. Thus the interfaces of an object form a partition of the interactions of that object.*

NOTES

1 - An interface constitutes the part of an object behaviour that is obtained by considering only the interactions of that interface and by hiding all other interactions. Hiding interactions of other interfaces will generally introduce non-determinism as far as the interface being considered is concerned.

2 - The phrase "an interface between objects" is used to refer to the binding (see 13.4.2) between interfaces of the objects concerned. [18, Def. 8.4]

This definition is based on another important definition, that of behaviour[1]:

Behaviour: *A collection of actions (that is, interactions or internal actions) with a set of constraints on when they may occur... (the rest of this definition is as that of [18, Definition 8.6])*

It is essential to realize that by an interaction, the RM-ODP makes here reference to an interaction occurrence (a.k.a. an interaction instance), and not to an interaction template nor to an interaction type. Likewise, the definition above is that of an instance of an interface, and it is not an object: interface is therefore an independent instance concept in the RM-ODP Foundations. Importantly, an interface in ODP is not an object type, as it is in UML.

Note 1 is of particular importance, as it makes it clear that the definition refers to an abstraction by projection, rather than to an abstraction by conceptualization [14]. Hiding an interaction does not mean to ignore it, but rather to turn it into an internal action (hiding actions is explained in [7, p.121]). An interface is identified with a set of interactions in which the object can participate [17, Section 7.1.2], but it is more than just this set of interactions; it is a complete behaviour, albeit a less deterministic one than that of its object [2].

An interface does not consist only of those interactions that identify it. The non-normative overview to the RM-ODP explains that *an interface represents a part of the object's behaviour related to a particular subset of its possible interactions* [17, Section 7.1.2]. To take an informal example, an attribute of an object that is accessed by both read and write methods of an interface belongs to that interface; on the other hand, an attribute that has absolutely no influence on the methods of an interface does belong to that interface (the ODP definition covers also all the intermediate cases).

2.2 A New Definition for Role

The current definition of role of the RM-ODP [18, Definition 9.14] is unsatisfactory, as testified by several debates and issues that arose during SC7/WG17 working sessions on the ODP Enterprise Language. It is partially in response to this situation that we proposed in [3] a new definition for role in the RM-ODP, taking into account the common insight that the concept of role is analog to that of interface:

Role: *An abstraction of the behaviour of an object that consists of a subset of the interactions of that object together with a set of constraints on when they may occur. A role always belongs to a specific larger behaviour that involves other roles, called a* **collaborative behaviour***.*

NOTES

1- The collaborative behaviour of a role represents the specific context in which it is defined, together with other roles. All the interactions of a role are with the roles with which it is defined. And all the actions in a collaborative behaviour belong to one or more of its roles.

2- The roles in a collaborative behaviour may be of the very same type (e.g. as in the couple player-player), or of different types (e.g., as in the couple producer-consumer).

[1] The RM-ODP Foundations only provide a definition for "Behaviour (of an object)" [18, Definition 8.6]. In [3], we showed that a slightly more general definition of behaviour was needed, for several reasons.

[2] See [4] for a semi-formal example of the non-deterministic behaviour of an interface vs. the deterministic behaviour of an object.

3- A role is fully identified by the object that performs it, and by a set of interactions (the intersection of the object behaviour and a collaborative behaviour). It is possible for one role instance to belong to multiple collaborative behaviours.

4- A role constitutes the part of an object behaviour that is obtained by considering only the interactions of that role and by hiding all other interactions. A role may contain internal actions.

5- An action occurrence of a role may also belong to other roles.

6 - Role being a specification concept, objects may be defined partially or completely without making use of roles. Therefore, the interactions of an object need not belong to any defined role.

7 - A role maps to at most one object within the model where it is defined. However, an object may have a decomposition relation with a configuration of objects in a model at a lower level of abstraction. In this indirect way, a role may map to a set of objects.

As for the definition of interface, the interactions and internal actions of a role are to be understood as action occurrences.

3. Interface vs. Role

3.1 Differences between Interface and Role

Both role and interface are identified with an object and a set of interactions of that object, and both are defined as the abstraction by projection of the behaviour of that object. And in both cases, the abstraction consists in hiding some of the interactions of the object (those that are not identified as belonging to the role or to the interface). Role and interface are therefore similar concepts

Role and interface are different concepts in that the considerations that apply to the allocation of interactions to them are different: the set of interactions of an interface is usually not the set of interactions of some role, and vice-versa. These sets of interactions tend to be different for the following reasons:

1– The RM-ODP defines interface as a basic modelling concept, with the consequence that all interactions are associated with interfaces. On the other hand, both the current RM-ODP definition and our proposed new definition make role a specification concept. This means that roles are a design tool, which may or may not be used: objects may be defined partially or completely without making use of roles. Therefore, an interaction is necessarily associated with interfaces, but it is not necessarily associated with any roles.

2– By the definition of interface, each interaction (occurrence) of an object belongs to a unique interface. This restriction does not apply to roles: an object performing the same interaction with respect to two different roles should be allowed, whenever possible, to perform this interaction just once, not twice. For example, consider an interaction that consists of emitting an event notification on a multicast channel: emitting a notification twice for the same event would serve no purpose; it would just be confusing. Another example is a person who makes the same trip for two different purposes with respect to two different roles (e.g, a prince making an official visit to Switzerland, and making a private visit to his daughter in her boarding school). In summary, the roles of an object, unlike its interfaces, do not form a partition of the interactions of that object.

3– An important consideration for defining interfaces, that originates from notation or from technology restrictions, is to partition interfaces between server interfaces (for accepting operation invocations) and client interfaces [3] (for invoking operations). Such a partitioning is not applicable to roles because performing a role generally implies accepting operation invocations from other roles, and making operations invocations to other roles.

[3] Readers unfamiliar with the concept of client operation interface will find explanations in [19, Section 7.2.2.3]. Quite simply, a client operation interface is an interface from which an object invokes operations on other objects. ODP Objects may have several interfaces, and even several interfaces of the same type.

The UML ignores the concept of a client interface, and as a result tends to forget that objects may invoke operations on other objects. These omissions and errors invalidate the UML definitions of behaviour and abstraction. For example, Clemens Szyperski showed in his famous book on components that a contract for an object (pre- and post-conditions for each operation, and invariants) is not always sufficient for providing a valid abstraction of this object [13, Chapter 5]. Following on this observation, G. Genilloud showed that the abstraction of an object or a component corresponds instead to its specification, and that an object specification is quite different from a contract [4].

4– An interface is fully defined on the basis of the object to which it belongs; it is thus independent of the interfaces which will ever be bound to it. On the other hand, it is well accepted that roles are always defined relatively to other roles in a specific context, as pointed out by the definition of role in UML. It is for this very reason that our definition of role makes reference to the concept of a collaborative behaviour.

5– Object modellers tend to specify finite collaborative behaviours (in a sequence diagram, a collaboration diagram or a use case textual specification, only one instance of behaviour is represented, and it is usually finite). Most roles are therefore finite behaviours. On the other hand, the set of interactions of an interface is typically defined on the basis of typing considerations (e.g., all instances of the AddUser operation may be defined as belonging to the manager interface of an object) or of naming considerations (that is, on how the object is identified by other objects — see Section 3.2). If an interface remains active forever, the number of interactions that it includes is unbounded.

6– An interface is an abstraction of an object behaviour. As explained in the definition, this abstraction consists of hiding the interactions that happen at other interfaces, and generally introduces non-determinism in the resulting behaviour. This non-determinism is arbitrary and non-intentional: crucial information about the way an object handles service requests may well be lost when performing this kind of abstraction. On the other hand, abstracting an object to a role does not lose as much crucial information: roles are indeed specified independently of objects: they are sufficient by themselves (with other roles) and make sense in the context they are defined. Because crucial information may be lost when abstracting an object to an interface, finding the behaviour of an object from his set of interfaces can be a very difficult task. Synthesizing roles to find an object behaviour is much easier, as the ROOA and the OOram software engineering methods demonstrate [1] [9].

We conclude that role and interface are similar but different concepts. Roles and interfaces are usually different behaviours, and for good reasons. However, it is not impossible to have a particular role and a particular interface that are essentially equivalent (i.e., both are identified by the same object and by the very same set of interactions). It is possible to define an interface whose behaviour is finite (e.g., the interface is deleted after a given sequence of interactions has occurred), and conversely, it is possible to define roles whose behaviour is infinite. For example, UML actors are composite roles which are typically infinite: an actor may engage in an unbounded number of use case instances (collaborative behaviours) of different types; therefore, an actor is the composition of an unbounded number of role instances, leading to an infinite behaviour.

3.2 Mapping Roles to Interfaces

In an ODP model, it is usually admitted that objects refer each other by referencing their interfaces. Since roles (instances) refer to other roles, the question arises as to whether there is a need for referencing objects by the roles they play? We think not.

In the reality, we often experience situations where we play roles with respect to other persons, who play related roles. Yet we almost never feel the need to name the instances of the roles with which we interact[4]. We typically use the person's name, or the role type name, such as "Waiter" in a restaurant, and that is good enough. In object models, role identifiers are not needed for about the same reasons that they are not needed in reality.

In object models, roles must be mapped to objects for being performed. When doing this mapping, roles can be mapped to interfaces in such a way that every reference to a role is substituted by a reference to an interface. There is always a solution to make this mapping work:

- There is the case of objects that may perform only one role at a time. In a collaborative behaviour, any reference to the role may therefore be mapped to a reference to one interface that also contains the desired interactions. It does not matter that this interface contains interactions that belong to other roles since those belong to other temporal contexts.
- A common situation is for an object to perform multiple roles at a time, and to remove ambiguity about which role it plays by considering the arguments of its interactions. For example, an invoice identifier may be sufficient for identifying a particular sales transaction, or shall we say, a particular sales collaborative behaviour. In this situation, it is possible to map many roles (instances) to a same interface (instance).
- In fact, there are situations which tolerate some ambiguity about which role instance is referred to. For example, think of immutable operations that always return the same results whenever they are called.
- As for the remaining cases, the ODP model allows for considerable flexibility in defining interfaces. For example, it makes it possible to arrange for having interfaces that correspond exactly to roles.

[4] This is probably the reason why we mostly think of roles as types, rather than as instances.

Note that we do not suggest that a role be mapped to just one interface. A role may be mapped to more than one interface of an object. What is required is that any reference to the role in an interaction between roles (so as to enable a role to discover the existence of a third role) is always mapped into a reference to the same interface, that we call the *identifying interface* for the role. For example, consider the quite common object models where interfaces are either of the client or of the server kind (i.e., they contain only invocations or receptions, but not both), and where at most two objects participate in an interaction. Our suggestion implies that to each role corresponds at most one server interface — the *identifying interface* for the role. On the other hand, the number of client interfaces for a role is unrestricted, since objects never need to refer explicitly to client interfaces of other objects.

From the above discussion, we observe that role models are more easily implemented by objects that may have multiple interfaces, in particular if those objects also have the capability of instantiating and deleting interfaces on themselves.

4. Roles as Types

ISO experts working on the ODP Enterprise Language are divided on this question: is role a type concept or an instance concept? UML experts are puzzled by the same question. We believe this to be a false question to ask, for two reasons:

1– It is quite common in language to use a term to either mean a thing (an instance), or the type of that thing. For example, consider the two sentences: “He came with his wife’s car”, and “With the Mini, Austin invented a new car”. In the first sentence, the term “car” refers to an instance, in the second it refers to a type. Thus, the same term “car” may denote quite different concepts. As another example, consider the sentence: “A domain object is an object that represents an entity from the problem domain”. In this sentence, the term “domain object” denotes a type of objects rather than a specific instance, even though most definitions of object clearly define object to be an instance concept. For an extended coverage of this topic, see the book of George Lakoff [6].

2– The second reason is that to every instance concept, there is a related type concept, and vice-versa. This is very apparent from the ODP definitions of instance and type in the RM-ODP Foundations.

> ***Type (of an <X>):*** *A predicate characterizing a collection of <X>s. An <X> is of the type, or satisfies the type, if the predicate holds for that <X>. A specification defines which of the terms it uses have types, i.e. are <X>s. In RM-ODP, types are needed for, at least, objects, interfaces and actions... [18, Def. 9.7]*
>
> ***Instance (of a type)****: An <X> that satisfies the type.... [18, Def. 9.18]*

In fact, we understand the true question facing ODP and UML experts to be this one: Is role a type applicable to some other modelling concept? Or is role an independent modelling concept (and therefore both the concepts of role instance and role type exist)? We examined these two positions in [3]. What we found is that the term role may denote an instance concept (role), a type for role instances (role type) or a type for objects (role object type).

4.1 Role Types

Defining role as an instance makes role an <X> in the RM-ODP Foundations. We thus immediately have a definition for the concept of *role type*:

> ***Type (of a role):*** *A predicate characterizing a collection of roles. A role is of the type, or satisfies the type, if the predicate holds for that role...*

We are aware that many people tend to think of roles as types rather than as instances. We fully acknowledge that in many linguistic contexts, the term role, or the name for a role, denotes a role type rather than a role. We even acknowledge that the term role may be understood as denoting a role object type, a concept which we will now define.

4.2 Role Object Types

When we think of a teacher, we think of a person and not of a role. We therefore tend to think of “Teacher” as a type of person, or in other words, as a person type rather than as a role type. Likewise in object models, role types may be used to type objects and not just roles.

Strictly speaking, role types are only applicable to roles, and as such, they are of little interest to those who see roles only as types, that is, as *object types*. However, a role type is applicable to objects in the sense that it implies an object type. Consider a role type, say *R(x)*, which is a predicate of the form “*x* is a role, AND *x* is characterized by...”. This role type implies an object type *O(y)*, of the form "*y* is an object, AND *y* performs one or more roles *x* SUCH THAT *x* is characterized by..." (in both predicates, the ellipsis denotes the very same characteristics). We call an object type such as *O(y)* a *role object type*.

Role object types are neither subtypes or supertypes of other (more classical) object types: Teacher is neither a subtype nor a supertype of Person. It might be that all teachers are persons, but this fact would be better modelled with an explicit constraint, rather than by making Teacher a subtype of Person (as is usually done). In [5], Kilov and Ross introduce the concept of dynamic supertype, a generalization of the ODP concept of supertype [5]. Using this modelling technique, Teacher would thus be a dynamic supertype of Person.

The RM-ODP leaves it to specific notations or software engineering methods to decide whether instances are explicitly typed, and exactly how they are typed. This means that we need not completely agree here on what role types and role object types exactly are. There is indeed more than one way to define a role object type on the basis of a role type. Considering the role type *R(x)* above, we could define an alternate role object type O'(y) in this way: "y is an object, AND y may perform one or more roles x SUCH THAT x is characterized by...". O(y) implies that an object of its type actually performs the role and is clearly a dynamic type; O'(y) alleviates this constraint and is therefore more static. In fact, we certainly make use of this flexibility without necessarily realising it. For example, different persons are likely to come up with different answers to the question: which persons are teachers (those who are teaching right now, those who have taught in the past, or those who are likely to teach in the near future)?

The points we need to make are 1) that a role type may be derived from the specification of a role, and 2) that one or several object types may easily be derived from a role type. Loosely speaking, we might say that a role type may be applied to objects or to roles, as we choose.

Intuitively, the idea of role as an object type makes perfect sense. It is indeed easy to conceive of a collection of objects that are susceptible to perform a particular role. Readers familiar with Java or UML have already encountered a similar idea, as interface in both this languages is a type concept applicable to objects. In a sense, what we did in this section is to take the very same idea and apply it to the concept of role.

5. Generality and Consistency of our Definition of Role

To demonstrate the generality of our definition of role, and its consistency with experts findings about roles, we check it against the properties of roles that Friedrich Steimann collected and commented in its extensive study of the literature [12].

1– A role comes with its own properties and behaviour.

Yes. Our definition is compatible with the fact that a role instance has a behaviour. Regarding properties, it must be understood that they are not mentioned in the RM-ODP foundations, the reason being that it is redundant to speak of both properties and behaviour (properties may be considered as an indirect way of specifying behaviour). If an object has the property of being blue, then this property only matters when an object reports that it is blue (or more generally, when the blue property affects in some way the interactions of that object).

2– Roles depend on relationships. As suggested by the work of Sowa and Guarino, a role is meaningful only in the context of a relationship.

Yes. Our definition is in fact more precise as it makes roles depend on collaborative behaviours, involving several other roles. To such a collaborative behaviour corresponds obviously a relationship. On the other hand, we are not sure that just any relationship is adequate for defining roles. It is widely accepted that roles are performed. So, if no collaborative behaviour may be related to some relationship, then there are no roles in this relationship. For example, consider a relationship that says "never works with". What would be the roles of such a relationship?

3– An object may play different roles simultaneously. This is one of the most broadly accepted properties of the role concept. Because a role is usually regarded as a type, it amounts to the multiple classification of objects.

Yes. There is nothing in our definition of role that would prohibit an object of simultaneously playing several instances of different role types. And role object types, as we define them, certainly enable a multiple classification of objects.

[5] In the RM-ODP, supertype/subtype are defined in such a way that "A type A is a subtype of a type B, and B is a supertype of A, if every <X> which satisfies A also satisfies B." On the other hand, "A type A is a dynamic subtype of a type B, and B is a dynamic supertype of A, if every <X> which satisfies A *may* also satisfy B."

4– <u>An object may play the same role several times, simultaneously.</u> This is an equally fundamental finding, a frequent example of which being an employee holding several employments. Unlike with different roles, however, it does not correspond to multiple classification. The main reason to distinguish multiple occurrences in the same role is that each occurrence of the object in a role is associated with a different state (item 10). For example, an employee has one salary and one office address per job.

Yes. There is nothing in our definition that would prohibit an object of simultaneously playing several instances of a same role type. Moreover, our suggestion of mapping roles (instances) to interface (instances) enables to associate different states (or portions of state) with different roles. For example, different occurrences of the GetSalary operation may report values that depend on which interface they belong to.

Note: Friedrich Steimann finds himself in the need of speaking of role instances (other than objects), perhaps without being aware of it. Occurrence and instance being considered as synonyms in the RM-ODP Foundations, it is clear to us that a role occurrence is nothing else than a role instance. This confirms that we are right to define role as an instance, and that a role instance is not an object.

5– <u>An object may acquire and abandon roles dynamically.</u> This is a dynamic property of the role concept that comes close to object migration or dynamic (re)classification. However, the two are not necessarily the same; for example, Wieringa et al. make an explicit distinction between dynamic classification and role playing.

Yes. There is no problem since the RM-ODP Foundations place absolutely no restrictions on the behaviour of an object. And it is well known that an ODP object may instantiate or delete interfaces dynamically.

6– <u>The sequence in which roles may be acquired and relinquished can be subject to restrictions</u>. For example, a person can become a teaching assistant only after becoming a student. The usual sequence specifying formalisms are in use.

Yes. The RM-ODP Foundations, being not an actual modelling technique, leave total freedom as to how the behaviour of an object is to be specified.

7– <u>Objects of unrelated types can play the same role.</u> Although a fundamental observation complementing those of items 3 and 4, it is not acknowledged by all authors.

Yes. Of course, “play the same role” is to be understood here as “play roles of the same type.” Even though we define role object types (whose purpose is precisely to relate objects playing different roles of the same role type), we do not demand that the object type system relates objects by the roles they play.

8– <u>Roles can play roles.</u> This mirrors the condition that an employee (which is a role of a person) can be a project leader, which is then a role of the employee (but also another role of the person, although only indirectly). A rather technical subtlety that seems to require that roles are themselves instances.

Yes and no. Yes in the sense that our definition allows for composite roles to be defined, or for roles to include other roles. No in the sense that only an ODP object may perform actions, and therefore roles. We think that our definition fully satisfies the intention of this assertion.

9– <u>A role can be transferred from one object to another.</u> It may be useful to let a concrete role dropped by one object be picked up by another, or even to specify the properties of a concrete role without naming a particular role player. For example, the salary of an open position may be specified independently of the person that will be employed.

Yes and No. Yes if role is to be understood in the sense of a type (see our response to assertion 5). No if role is to be understood in the sense of a particular role instance (or occurrence): our definition makes a role a projection of an object, and therefore, it does not make it possible to consider that a role is transferred from one object to another. However, we shall point out that, in a model, transferring a “role” from an object to another object would need to be modelled explicitly: extra interactions and extra roles would appear within the context of the defining collaborative behaviour, or within the context of another (backstage) collaborative behaviour. Obtaining the same functionality is therefore possible with our definition of role. The only constraint that it imposes in this matter is that more roles are defined. Still, we acknowledge that the idea that “a role instance is transferred from one object to another” makes sense. Our current definition is excessively narrow in that respect and we will try to loosen it in our future work [6].

Note: for the second time, Friedrich Steimann finds himself in the need of speaking of role instances (other than objects). This time, he uses the term “concrete role.”

[6] We will examine defining role as “the projection of a composition of objects”, the composed objects being those that perform the role. In this way, we would fall back on our current definition whenever a role is performed by a single object.

10– The state of an object can be role-specific. The state of an object may vary depending on the role in which it is being addressed. Together with item 4, i.e., the possibility of one object playing the same role multiply at the same time, this seems to suggest that each role played by an object should be viewed as a separate instance of the object.

Yes. This is possible in the RM-ODP Foundations fitted with our definition of role. As we pointed out in the discussion of the fourth assumption, mapping roles to interfaces enables to associate different states with different roles.

11– Features of an object can be role-specific. Attributes and behaviour of an object may be overloaded on a by-role basis, i.e., different roles may declare the same features, but realize them differently. If an object plays several of these roles simultaneously, it responds according to the role in which it is being addressed.

Yes. We would not say that the reason is overloading, but rather the fact that different roles may be mapped to different interfaces of an object.

12– Roles restrict access. When addressed in a certain role, features of the object itself (or of other roles of the object) remain invisible. This corresponds to an object having different perspectives, facets, or aspects.

Yes, in some sense. It is not the roles of an object, but its interfaces that restrict the interactions that may take place with it. But it is possible to define interfaces on the basis of role types, and even on the basis of role instances.

13– Different roles may share structure and behaviour. This usually means that role definitions inherit from each other, but sometimes also that the definitions of roles rely on features of the objects playing them (delegation).

Yes. We explicitly stated that two role instances may share some action occurrences. And of course, it is possible for two different actions of two different roles to be instantiations of the same action template. In fact, two different role instances may well be instantiations of the same behaviour template.

Note that our definition of role makes no reference to structure as structure is linked to state, and as state is the dual of behaviour.

14– An object and its roles share identity. In the object-oriented world this entails that an object and its roles are the same, a condition that has been paraphrased as "a role is a mask that an object can wear."

Yes and no. No in the sense that an object and its roles are not exactly the same things, and this is true no matter on how we define roles (if roles and objects were the very same things, the concept of role would have no interest whatsoever and it would not deserve a definition; even in the interpretation that a role is a mask than an object can wear, a role and an object are different things). Yes, in the sense that identifying a role in an object model identifies the object that performs this role (note that this may be true only at a given time, rather than at all times — see assertion 9). And sometimes, identifying an object in a given context identifies a specific role of that object. See also Section 5.1.

15– An object and its roles have different identities. This view, which is quite singular, is a tribute to the so-called counting problem. It refers to the situation in which instances counted in their roles yield a greater number than the same instances counted by the objects playing the roles. For example, the number of passengers taking a certain means of transportation in one week may be greater than the number of individual persons travelling with that means during the same period.

Yes, since an object and its roles are not exactly the same things.

5.1 A Note about Identity

We think that the last two assertions in the previous section are ill-conceived, as they reflect a common misunderstanding of the idea that objects have identity.

The Oxford English Dictionary provides a good definition of identity: "*2. a. The sameness of a person or thing at all times or in all circumstances; the condition or fact that a person or thing is itself and not something else; individuality, personality.*" Therefore, an object having identity simply means that this object is distinct from any other object, and in fact, from anything else.

It is clear that if an object is distinct from any other object, then there is something that makes it distinct from any other object. For this reason, many people like to think that each object has a kind of an invisible attribute, that they call "identity" or OID (object identifier). Many object data bases or programming languages comfort them in that belief. Indeed, if every OID is assigned to a single object, it contributes to make that object distinct from all other objects. However, having exactly one OID and having identity are different properties. For an object to have identity, it is sufficient that it has exactly one OID, but this is not a necessary condition. For example an object may have not just one, but several OIDs. Consider RIDs (role identifiers), the like of OIDs for roles. All RIDs are associated with an object, the object that performs the role. Now, every RID of an object makes that object distinct from all other objects. An RID is also an OID. In that sense, we may say that an RID is shared between an object and one of its roles.

On the early days of the Object Management Group, a group of architects from various domains has looked at what properties may be considered as essential for objects, and “having exactly one OID” was not one of those they retained. On the other hand, they retained this essential property:

An object can be identified directly or reliably. Object identification is direct in the sense that a request names the object without describing it. Object identification is reliable in the sense that repeating an identification refers to the same object.... A value that identifies an object is called an object reference. [11, p.34]

CORBA objects may have not just one, but several object references associated to them. And CORBA is unable of saying whether two different CORBA object references refer to a same CORBA object or to two different objects [7]. The reasons for this apparent limitation are explained in [8] and in [15].

In summary, the fact that objects have identity is quite different from the idea that they have exactly one OID assigned to them. This difference matters when considering roles, in particular when trying to count roles (instances) rather than objects.

6. Summary and Conclusions

When considering the debate on what is a role, we found that all the experts were right in a sense or another when they claimed that role is an instance, or that role is a type (for objects). Indeed, our findings are that the term role may denote an instance concept (role instance), a type for role instances (role type) or a type for objects (role object type). A role instance is therefore not a type, and role is to be considered as an <X> in the RM-ODP. Special care should therefore be used to make sure that there is no ambiguity about which concept is meant by the term ‘role’ (or a name for a role) in a particular context.

The insight by UML experts that role is analogous to interface was very useful to us: our proposed new definition of role is derived from our analysis of the similarities and differences between these two concepts. And the insight turned out to be a very good one, as we have shown in Section 5.

Further confirmation that we are on the right track with of our definition of role was given to us by the Roles Work Group of the Business Objects Domain Task Force (BODTF) of the OMG. They worked independently of us, and yet they came to analogous results. In their working document (a survey of the concept of role in many different domains), they came up with this conclusion:

The common thread of the concept of role appears to be the behaviors (e.g., state and methods) associated with a principal in the context of a particular activity....
A principal may take on the same role for multiple instances of the same activity. Each of these roles may have different state. Consequently, it must be possible to distinguish one role from others to associate the proper role state with the activity context. In other words, a role must be identifiable. [16, Summary]

7. Acknowledgements

The RM-ODP standard, while imperfect with respect to its definition of role, was invaluable to us. It helped us understand what are the modelling concepts related to role, and how they are related. For example, from our definition of role, we automatically obtained the related definitions of role template and role type. And from the RM-ODP definition of type, we understood how a role object type may be derived from a role type.

Acknowledgements go to Trygve Reenskaug for his presentation to the ISO/SG7/WG17 in Paris last year and for subsequent discussions, to William Frank and to Joaquin Miller for their various contributions and support, and to all members of the ISO/SG7/WG17 or the UML revision task force, for providing us both with an interesting issue and with important leads for our analysis and proposed resolution.

8. References

[1] R. G. Clark and A. Moreira, “Formal Specifications of User Requirements,” *Automated Software Engineering*, vol. 6, pp. 217-232, 1999.

[7] In ODP terminology, a CORBA object is really an interface. The question that cannot generally be answered by CORBA is therefore whether two interface identifiers refer to a same interface or to two different interfaces.

[2] G. Genilloud and A. Wegmann, “On Types, Instances, and Classes in UML,” presented at the ECOOP Workshop on UML Semantics, Sophia-Antipolis, Cannes, France, 2000. (available at URL http://icawww.epfl.ch/).

[3] G. Genilloud and A. Wegmann, “A Foundation for the Concept of Role in the RM-ODP,” in proceedings of the 4th International Enterprise Distributed Object Computing Conference (EDOC 2000), Makuhari, Japan, 2000.

[4] G. Genilloud, “On the Abstraction of Objects, Components and Interfaces,” in proceedings of the OOPSLA Workshop on Behavior Semantics, pp. 93-104, Denver, Colorado, 1999. (this paper is also available at URL http://icawww.epfl.ch/).

[5] H. Kilov and J. Ross, *Information modeling : an object-oriented approach*: Prentice Hall, 1994.

[6] G. Lakoff, *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind*. Chicago: The University of Chicago Press, 1987.

[7] R. Milner, *Communication and Concurrency*: Prentice-Hall, 1989.

[8] M. L. Powell, “Objects, References, Identifiers and Equality,” SunSoft, Inc., White Paper OMG TC Document 93.7.5, July 2 1993.

[9] T. Reenskaug, O. A. Lehne, and P. Wold, *Working with Objects: The OOram Software Engineering Method*: Prentice Hall, 1995.

[10] T. Reenskaug, “UML Collaboration and OOram semantics (New version of a green paper,” vol. 2000, 2nd ed, Nov. 8, 1999, http://www.ifi.uio.no/~trygver/documents/)

[11] A. Snyder, “The Essence of Objects: Concepts and Terms,” *IEEE Software*, vol. 10, pp. 31--42, Jan. 1993.

[12] F. Steimann, “On the representation of roles in object-oriented and conceptual modelling,” *Data and Knowledge Engineering*, vol. 35, pp. 83-106, 2000.

[13] C. Szyperski, *Component Software : Beyond Object-Oriented Programming*. Addison-Wesley, 1998.

[14] M.-N. Terrasse and M. Savonnet. “Metamodeling with the UML: An Approach to the Formalization of the UML Metamodel,”. in the 5th CAiSE/IFIP 8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design, EMMSAD'00. June 2000.

[15] A. Watson, “Identity, Equality and Location,” Object Expert, vol. 2, pp. 21-24, 1997.

[16] BODTF, “A Green Paper on Roles (version 0.2),” August 14, 2000.

[17] ISO/IEC and ITU-T, “Information technology — Open Distributed Processing — Reference model: Overview,” in Standard 10746-1, Recommendation X.901, 1998. <http://isotc.iso.ch/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards>

[18] ISO/IEC and ITU-T, “Information technology — Open Distributed Processing — Reference model: Foundations,” Standard 10746-2, Recommendation X.902. 1996. <http://isotc.iso.ch/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm>

[19] ISO/IEC and ITU-T, “Information technology — Open Distributed Processing — Reference model: Architecture,” Standard 10746-3, Recommendation X.903. 1996. <http://isotc.iso.ch/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm>.

Ninth OOPSLA Workshop on

Behavioral Semantics

OOPSLA 2000

Minneapolis, Minnesota, USA — October 15, 2000

Edited by

Kenneth Baclawski
Haim Kilov

Northeastern
UNIVERSITY
COLLEGE OF COMPUTER SCIENCE