Role is an <X>:
a Foundation for the Concept of Role

Guy Genilloud, Alain Wegmann

May 2000

Technical Report DSC/2000/023

EPFL-DSC
CH-1015 Lausanne *http://dscwww.epfl.ch*

Role is an <X>:
a Foundation for the Concept of Role

April 2000

Guy Genilloud, Alain Wegmann
EPFL
DSC-ICA
CH-1015 Lausanne
guy.genilloud@epfl.ch, alain.wegmann@epfl.ch

ABSTRACT

Standardization experts in object modelling are having difficulties with defining the concept of role; for example, they are not sure of whether role is a type or an instance concept. This issue is a source of confusion in the UML standard, and prevents ISO experts to reach consensus and finalize a language for ODP enterprise modelling. In this paper, we make an in-depth analysis of the problem, find its likely causes, and come up with a proposal for a new ODP definition of role, as well as with definitions of related concepts. We expect our definitions and our results not only to achieve consensus among ISO delegates, but also to be a basis for improving the UML standard and related software engineering processes.

Keywords

Modelling concept, role, object modelling, ODP, UML.

1. INTRODUCTION

The Reference Model of Open Distributed Processing (RM-ODP) is both an ISO/IEC standard and an ITU recommendation. Its purpose is to serve as a coordinating framework and a very general architecture for the development of standards related to distributed processing, interworking, and portability [10]. The RM-ODP Part 2, called Foundations, provides the definitions of the concepts and an analytical framework for normalized description of (arbitrary) distributed processing systems [11]. In other words, Part 2 provides the foundations for the object modelling needs that arise when building distributed processing systems, from business modelling to a technical realization.

The ISO group JTC1/SC7/WG3 is currently working on a number of ODP standards. Of particular importance is a standard for the Enterprise Viewpoint Language, which is concerned with modelling the purpose, scope and policies governing the activities of a system. There is consensus that the concept of role is central to this language. However, ISO experts have considerable difficulties to agree on the very meaning of the concept of role. They find the current definition of role in the RM-ODP Foundations [11, Def. 9.14] to be very obscure if not flawed, so much that it does not help them.

ISO delegates are not the only experts to struggle with understanding the concept of role. For example, the UML Revision Task Force of the OMG 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!*" To our knowledge, these discussions remained rather inconclusive.

Our purpose with this paper is to find and to propose a new definition for the concept of role in the RM-ODP (with the perspective that settling the ODP issue will benefit to the object modelling community at large). We take into consideration the different meanings that are attributed to the concept of role in well-known contexts such as RPC protocols (where we speak of client and server roles), network or systems management (manager and agent roles), UML and OOram [5] (OOram is the SW engineering process that

contributed most to the introduction of the concept of role in UML).

Our hope is to find a definition that is agreeable to most people, and therefore that serves them in reconciling their positions. But achieving consensus is not really our concern: we are looking for a definition that is meaningful, sensible, and consistent with the rest of the RM-ODP framework. Since we are concerned with the very foundations of object modelling, we expect our findings to be useful to designers of software engineering processes or languages (whether programming, specification or modelling). Being ourselves interested in improving UML, we will make a preliminary analysis of the impact of our findings for UML.

1.1 Organization of this Paper

This paper is organized as follows. Section 2 looks at the current definition of role in the RM-ODP and shows why it does create problems to the RM-ODP community. Section 3 addresses the ongoing debate on whether role is an instance or a type concept, shows that it is a false debate, and centres it on the real issue. Section 4 investigates the question of whether the concept of role is redundant with that of interface, and concludes by the negative. Section 5 contains our proposal for a new definition of role, that addresses all the concerns expressed in the previous sections. Section 6 shows how the concept of template addresses the problem of specifying a potentially infinite number of roles, and discusses the type concepts related to roles. Section 7 briefly shows the consequences of our findings for the case of UML. It exposes a number of oddities, inconsistencies and limitations with respect to role modelling.

2. CURRENT DEFINITION

The current RM-ODP definition of role is:

> ***Role:*** *Identifier for a behaviour, which may appear as a parameter in a template for a composite object, and which is associated with one of the component objects of the composite object.*
>
> *Specification of a template as a composition of roles enables the instantiation process to be explained as the association of a specific component of the resultant composite object with each role. The association of a component object with a role may result from the actualization of a parameter. [11, Def. 9.14]*

As many others definitions in the RM-ODP, this definition is not easy to understand on first reading. But even people accustomed to the RM-ODP definitions (and who learned to appreciate their consistency and very general applicability) struggle to make full sense of this particular one. They tend to make two different interpretations of the definition, which we consider in turn.

2.1 Role as an Identifier

The first interpretation of the definition of role is to consider that a role is an identifier for a behaviour. That is, a role is a name that unambiguously denotes a behaviour in some naming context[1]. The RM-ODP Part 1, an explanatory and non-normative document, confirms this interpretation when it says: "*A role identifies, in a template for a composite object, a behaviour to be associated with one of the component objects*" [10, Section 7.2.5].

However, as observed by William Frank, saying that a role is a name makes the concept of role trivial: "*Since names for individual things are arbitrary and without any consistent semantic content, to say that a role is name for a behavior implies that the concept of a role carries no more meaning than the concept of a behavior (in fact, has even less meaning, since it is only a name)*" ["Foundation for Role Concept in the ODP Enterprise Viewpoint", expert contribution to ISO/SG7/WG3, 8 Dec. 1999].

Indeed, to say "Plays a role" is natural, but to say "Plays a name" is meaningless. Moreover, who would think of the characteristics of a role as being its character set and its string length? Quite clearly, a role may have a name, but is not a name.

2.2 Role as a Behaviour

The definition of role may also be considered to say that a role is a named behaviour[2]. There is some explanatory text in the RM-ODP Part 1 document that tends to confirm this second interpretation:

[1] This interpretation is compatible with the way G. Booch, J. Rumbaugh and I. Jacobson introduce the concept of role: "*A role names a behavior of an entity participating in a particular context*" [1, p. 161].

> *A role may correspond to a subset of the total behaviour of a component object. When an object is viewed in terms of a role, only a named subset of its actions is of interest, and other actions are abstracted away, possibly to other roles. A component object may have several roles at a given time depending upon its interactions, and may take different roles at different times. These roles may be associated with interfaces. [10, Section 7.2.5]*

This text establishes a possible correspondence between a role and a subset of the behaviour of an object. If a role is a behaviour rather than a name, then this text just says that a role *is* a subset of the behaviour of an object (the subset of a behaviour being itself a behaviour).

At this point, we face the problem of finding a suitable definition for behaviour. The RM-ODP only has a definition for "Behaviour (of an object)" [11, Definition 8.6], event though it does make reference to the concept of behaviour independently of that of object (for example, Definition 9.1 speaks both of the composition of objects and of the composition of behaviours). Fortunately, the generalized definition of behaviour that we need can be obtained very simply by removing the restriction "of an object" from [11, Definition 8.6], the body remaining unchanged:

> ***Behaviour:*** *A collection of actions with a set of constraints on when they may occur... (the rest of this definition is as that of [11, Definition 8.6])*

With this generalized definition, we have no problem of considering role as a behaviour. We turn our attention to an important property of a role: that of *being a subset of an object behaviour*. However, by itself, this property does not say much about roles, for two reasons:

1. This property is true of all behaviours, not just roles. Indeed, any behaviour can be made into a subset of the behaviour of some object. A constructive proof is as follows: find a set of objects that participate in all the actions of the behaviour, and compose these objects.
2. A role does not always correspond to a subset of the behaviour of one object. Indeed, some roles, such as the agent role in systems management, are performed by a configuration of several objects. A constructive proof is as follows: consider a non-trivial role (i.e., a role that includes more than one action), and an object that performs this role (the role is a subset of the behaviour of this object); it is possible to decompose this object into a configuration of objects, in such a way that the actions of the role are performed by multiple objects.

The point we would like to make is the following. We have two sensible ideas: that a role is a behaviour that is named in some context, and that a role is a subset of the behaviour of an object. But these two ideas are not sufficient by themselves to distinguish roles from other behaviours. That is, defining a particular subset of an object's behaviour and giving it a name should not be sufficient to turn into a role. The RM-ODP makes references to the concepts of "template for a composite object", and of "component object", but it fails to explain why the notion of role should have anything to do with that of composite object. In the general literature about roles and object modelling, we are not aware of any references to such a correspondence.

In summary, we feel that we are on the right track with this second interpretation of the RM-ODP definition of role, but we have failed to capture the full sense of that definition.

2.3 Role Is a Tool for Design

There is one element in the RM-ODP definition of role that we have yet to consider. By including the definition of role in its Section 9, the RM-ODP Foundations standard categorizes role as a *specification concept*. The implication is that role addresses notions such as type and class that are necessary for reasoning about specifications, or is a general tool for design (see the explanations for specification concepts in [10, Section 6.2.1]. Since a behaviour is an instance rather than a type, we find that a role is a general tool for design.

Importantly, role does not belong to the minimum set of concepts that form the basis for ODP system descriptions. The ODP essential concepts, called the *basic modelling concepts*, are: object, environment (of an object), action, internal action, interaction, interface, activity, behaviour (of an object), state (of an object), communication, location in time, location in space, and interaction point. Thus, unlike these concepts, role is not fundamental in object modelling: objects may be defined partially or completely without the use of the concept of role, which is just a design tool. For example, the interactions of an object must belong to an interface, but they need not belong to any defined role.

---

[2] This second interpretation is compatible with the UML definition of role: "*Role: The named specific behavior of an entity participating in a particular context...*" [9, p. B-15].

3. IS ROLE AN INSTANCE OR A TYPE?

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 [3].
2. The second reason is that to every instance concept, there is a related type concept, and vice-versa. This is much 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... [11Def. 9.7]*

   Instance (of a type*): An <X> that satisfies the type.... [11, Def. 9.18]*

In fact, we understand the true question facing ODP and UML experts to be this one: Is role an independent modelling concept (and therefore both the concepts of role instance and role type exist)? Or is role a type applicable to some other modelling concept? We will now examine these two positions in turn.

3.1 Is Role an Independent Concept?

Role is an independent modelling concept if the very notion of a role instance makes sense in some linguistic contexts. We think that it does, for all these reasons:

1. We often speak of "performing a role", or of "a role being performed". It makes sense to perform a behaviour (a collection of actions), but it does not make sense to perform a type (a predicate).
2. The sentence "An object may perform more than one role" probably makes sense to just about everybody. It implies that a role and an object are different instance concepts.
3. The RM-ODP definition of role appears to us as poorly formulated and unclear. But given the time, efforts and expertise invested in its conception, we suspect it to be mostly right. Whether it points to an identifier or to a behaviour[1], it points to an instance concept. Likewise, the UML defines role as a behaviour and thus as an instance concept.
4. Trygve Reenskaug, the inventor of role-modelling for software engineering, says that roles have all the properties of objects [5, p. 45]. And of course, objects are instances.
5. Regarding the question of whether roles are types, Mr. Reenskaug's intuition is that "*Roles are not types, but can be typed*" [presentation to ISO/SG7/WG3, Paris, Nov. 24 1999]. It follows that role is an <X> in the RM-ODP (since <X> can be anything that has a type). The are therefore several concepts related to the term role: role instance, role type, role class and role template. In the RM-ODP, the same goes for terms such as object, interface, action, operation, etc.

In summary, the term role can be understood in some contexts to denote an instance concept (and not an instance of some other concept, such as object). But in other contexts, the same term role may well denote a type concept (a predicate), a class concept (a set of instances) or a template concept (a specification). The RM-ODP principle is that the term "role" is to be understood as denoting the instance concept, unless it is clear from the context that we are talking of a concept of role type or role template.

3.2 Is Role a Type of Some Other Concept?

If role is a type applicable to some other instance concept, then the question is to know what is this other

[1] An identifier is indeed a term which, in a given naming context, refers unambiguously to an entity (a type refers indirectly to a set of entities). A behaviour, by its RM-ODP definition, it is a collection of actions and actions, by definition, are individual occurrences or events, rather than types or templates which apply to those events. Thus, a behaviour, if it were to happen, by definition only happens once.

concept. In the RM-ODP modelling framework, all actions are attributed to objects (systems, users, and components being modelled as objects). Object is therefore the very likely answer to our question.

Confirmation to that effect is given from the literature. For example, Li and Wong write that "*In particular, an object at any time point is described not only by an instance of a class, but also an instance of every role type whose role it currently plays*" [4, Section 1]. Trygve Reenskaug also uses the term role to denote an object type: "*A role in an object specification is called an object type, which is a specification of a set of objects with identical, externally visible properties*" [5, p. 14].

We observe that Mr. Reenskaug uses the same term "role", in different contexts, to refer to an instance concept or to a type applicable to objects. Such a practice is quite common and in fact difficult to avoid. For example, the UML standard uses the terms "actor", "action," "event," etc. to denote either an instance/occurrence concept, or a type concept. More to the point, UML attempts to avoid this situation by defining different terms for a type/specification concept, and its related instance/occurrence concept. For example a stimulus is the occurrence of a message. Yet, UML often talks of "sending a message" when it should talk of "sending a stimulus" ("sending a specification" is certainly not what is meant).

3.3 One Term for Several Related Concepts

We have found 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). The question is to know what these different concepts possibly mean, and how they are related. We will address this question in Section 6.2, after we have found a suitable definition for role.

4. INTERFACE VS. ROLE

As we mentioned, the UML revision task force of the OMG had long email discussions in the second half of 1999 trying to better understand roles. One question that arose in the discussions was that of the relation between the concepts of role and interface. This is indeed an interesting question, which we will investigate now on the basis of the RM-ODP definition of interface:

> ***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. [11, Def. 8.4]*

This definition makes an interface a subset of the behaviour of an object. This subset is itself a behaviour (we refer the reader to our definition of behaviour in Section 2). Our initial thought is that the definition of interface captures an important characteristic of the concept of role — role and interface are indeed analog concepts. However, on closer examination, we find important differences between interface and our intuitive understanding of role:

1. Each interaction of an object belongs to a unique interface. This restriction does not apply to roles: an object performing a same interaction with respect to two different roles should be allowed, whenever possible, to perform this interaction just once, not twice. The definition of role must allow optimization of this kind to be possible. For example, consider an interaction that consists in emitting an event notification on a multicast channel: emitting a notification twice for a same event would serve no purpose; it would just be confusing. An analog example applies in the real world: consider a person who plays two different roles with respect to you, both of which require her to notify you of any change of address; you would not expect this person to notify you twice of a single change of address. Another example is a person who makes a 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).

2. Performing a role generally implies accepting operation invocations, but it also implies making operations invocations on other roles. However, interfaces are often constrained to be server or client operation interfaces[1] for technological constraints. This reason alone is sufficient for preserving a distinction between interfaces and roles, unless technological constraints could be removed, which of course is more than unlikely.
3. 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, roles are always defined relatively to other roles in a specific context, as pointed out by the definition of role in UML. For example, we tend to think of roles in pairs: client-server, manager-agent, provider-subscriber, etc.
4. An interface is an abstraction on 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[2]. On the other hand, abstracting an object to a role does not introduce non-determinism. Roles are indeed specified independently of objects: they are sufficient by themselves (with other roles) and make sense in the context they are defined. Thus, the non-determinism that a role may have exists also in its object behaviour, and is intentional.
5. 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.

5. A NEW DEFINITION FOR ROLE

We propose a new definition for role in the RM-ODP, taking into account the points that we have just raised.

> ***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 is thus a behaviour.*
>
> *NOTES*
>
> *1- A role belongs to a specific larger behaviour that involves other roles, that we call a* ***roles community behaviour****. Thus, a role is defined relatively to other roles within a specific context. These other roles 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). All the interactions of a role are with the roles with which it is defined.*
>
> *2- Roles are generally specified first and attributed to objects only later: roles are sufficient by themselves (together with their relative roles) and fully make sense in the context they are defined.*
>
> *3 - 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. Abstracting an object to a role does not introduce non-determinism. The non-determinism that a role may have exists also in the object behaviour, and is intentional.*
>
> *4 - An interaction of a role may also belong to other roles.*
>
> *5 - Roles are defined independently of interfaces. That is, roles have explicit identifiers only for other roles, not for interfaces. This imposes a constraint when mapping roles to interfaces, as*

[1] Readers unfamiliar with the concept of client operation interface will find explanations in [12, Section 7.2.2.3]. They may also look at the concept of port in [5]. 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 [7, 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 [2].

[2] For a concrete example, see the discussion of interface in [2].

*each role may have at most one **identifying interface**. A role may nevertheless be mapped to more than one interface of an object, but any reference to the role in an interaction between roles (so as to enable a role to discover the existence of a third role) may only be mapped into a reference to its identifying interface.*

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.

Most of the notes in our definition have already been discussed in Section 4, when we stressed the difference between role and interface. Note 1 adds an extra difference, and is motivated by the fact that role is a design tool: it makes it impossible to consider the existence of a role irrespectively of at least some other role. For the purpose of object modelling, a role that has no interactions with other roles is indeed of no interest —what happens in an object and is not observable in any way may as well not happen.

Note 5 was not discussed yet. It reflects a new concern: to our knowledge, interfaces (in the ODP sense) are never explicitly considered when making role models. We therefore introduce the rule that roles always reference each other directly. This is unlike ODP objects or components (whether COM, CORBA or EJB) which reference each other by referencing their interfaces, and not necessarily always the same interface. To make sense of Note 5, the reader should consider interfaces which are either of the client or of the server kind (i.e., they contain only invocations or receptions, but not both). Assume also for this example that at most two objects participate in an interaction. Note 5 implies that to each role corresponds at most one server interface — this interface is 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. We do not refer to the concepts of client or server interfaces in our definition of role because these concepts are not defined in the RM-ODP Foundations [11] (client and server interfaces are specialisation of the general concept of interface; they are defined in the RM-ODP Architecture [12]).

Regarding Note 6, we refer the reader to Section 2.3. We will not justify this point further in this paper, as nobody seems to suggest that roles are absolutely essential for object modelling.

Note 7 addresses our disagreement, already expressed in Section 2.2, that a role always correspond to a subset of the behaviour of one object. Our reason was that an object may always be decomposed into a configuration of objects. However, the role concept is a design tool which is used with respect to a given model, and it remains that a role maps to at most one object with respect to this specific model.

Indeed, the concept of role is introduced by modellers for applying the separation of concerns principle: the goal for the modeller is to obtain partial views of objects. When specifying a community of roles and their joint behaviour, the modeller has in mind a particular object model and an abstraction level. She determines the maximum number of objects which may be involved in the joint behaviour, and she defines as many roles as necessary. When doing so, she may consider attributing several roles to a same object. But she certainly does not intend to attribute a role to several objects. Otherwise, she would simply define more roles. We find confirmation of this point in a recent work of Trygve Reenskaug: "*In an instance of a collaboration, each ClassifierRole maps onto at most one object*" [6].

Therefore, roles serve their purpose with respect to a given object model. When a modeller decomposes objects in a model to obtain a more refined model, she has essentially two options: 1) consider that roles have served their purpose with respect to the more abstract object model — she just forgets about them, and roles are no longer defined in the refined object model; 2) decide to keep using roles in the new model — she must decompose roles alongside objects, such that roles in the new model are all fully contained within objects.

6. ROLE IS AN <X>

Our proposed new definition of role for the RM-ODP Foundations makes role an independent instance concept. It follows that "role" is substitutable for <X> in the generic definitions of the RM-ODP Foundations. We thus automatically obtain definitions for important concepts related to role, in particular role template and role type.

6.1 Templates

According to our definition, a role is a behaviour, and more specifically a subset of the behaviour of an object, as well as a subset of a *roles community behaviour* (or *community behaviour* for short). While

object behaviours are generally infinite (composed of an infinity of actions), community behaviours and thus roles are often finite (this fact contributes to make role an effective design tool). For example, a community behaviour may be expressed by an interaction diagram in UML, and those diagrams tend to contain only a finite number of interactions.

Thus a role, very much like a method occurrence in a programming language, tends to be ephemeral. But of course, objects generally play the same role again and again (not the very same role according to our definition, but the same role indeed). Likewise, community behaviours tend to occur again and again in a model, even though the objects behind the roles may change at each occurrence of the community (for example, consider the couple client server with respect to operations). In short, the number of roles and the number of community behaviours are unbounded. A modeller cannot describe them all one by one. So, rather than specifying roles or roles community behaviours, a modeller specifies role templates or community behaviour templates (likewise, a programmer specifies methods, which are templates of method occurrences).

The definition of a role template can be obtained by substituting role for <X> in the generic definition of "<X> template" in the RM-ODP [11, Def. 9.11]. We thus obtain:

> ***Role Template**: The specification of the common features of a collection of roles in sufficient detail that a role can be instantiated using it....*
>
> ***Roles community behaviour Template**: The specification of the common features of a collection of roles community behaviours in sufficient detail that a roles community behaviour can be instantiated using it....*

A telling example of a roles community behaviour template is a UML use case, except for the fact that roles are anonymous in use cases (they are replaced by actors and the system).

Note that it is specification, not instantiation, that is the primary purpose of role templates and community behaviour templates. There need not be an explicit action of instantiation of a role template for a role instance to exist (and likewise for a community behaviour — think of use case in UML). In the RM-ODP Foundations, it is made clear that not all *instances* of a template are *instantiations* of that template. Instantiation and instance are indeed different concepts:

> ***Instantiation (of an <X> template):** An <X> produced from a given <X> template and other necessary information. This <X> exhibits the features specified in the <X> template. <X> can be anything that has a type... [11, Def. 9.13]*
>
> ***Instance (of a type)**: An <X> that satisfies the type... [11, Def. 9.18]*

The type that relates instantiation to instance is a template type:

> ***Template type (of an <X>):** A predicate defined in a template that holds for all the instantiations of the template and that expresses the requirements the instantiations of the template are intended to fulfil.... [11, Def. 9.19]*

In his own terms, Trygve Reenskaug confirms the existence of related instance and template concepts of community behaviour when he speaks of *instance level collaborations* and *specification level collaborations* [6].

To conclude this section, we observe that the current RM-ODP definition of role mentions "a template for a composite object" [11, Def. 9.19]. We suppose that the ISO experts who designed this definition were thinking of "a template for a community behaviour" (since their definition of behaviour was restricted to the behaviour of an object, they needed to refer to a composite object for denoting a community behaviour).

6.2 Types

Our proposed definition of role for the RM-ODP Foundations makes role an independent instance concept. This new definition may thus cause discomfort to those experts who see role as purely a type concept. But as we explained in Section 3, there should be no reason for this discomfort. We fully acknowledge the validity and importance of the concept of role as a type. We define role as an instance concept because we have compelling reasons for defining so. Firstly, role must be defined in a way that is consistent with the way similar concepts, such as interface, are defined in the RM-ODP Foundations. Secondly, there is a clear need for a concept of role instance, and defining role as an instance allows for the existence and use of a role type concept, whilst the opposite is not quite true[1].

Defining role as an instance makes role an <X> in the RM-ODP Foundations. We thus 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....*

Strictly speaking, role types are only applicable to roles, and as such, they may be of little interest to those who see roles as object types. However, a role type implies an object type: a role type, say *R(x)*, is a predicate of the form "*x* is a role, AND *x* is characterized by..."; the corresponding object type, say *O(y)*, is 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 a type such as *O(y)* a *role object type*.

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. For example, a role object type *O'(y)* may be defined from *R(x)* 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. The points we need to make are 1) that a role type may be derived from the specification of a role, and 2) that an object type 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.

7. FURTHER WORK

We are currently working on applying the results of this paper to improve the consistency of UML while extending its modelling capabilities. The good news is that the UML definition is both close and compatible to ours, but it is also terse and incomplete, and thus not sufficient. However, knowing the important characteristics of role, we find interesting shortcomings or oddities related to role in UML.

For example, a role is always related to some other roles. But what is the truth of this in use case diagrams? Roles are anonymous in these diagrams, being replaced by actors and the system (or are the roles of the system named by the use cases?). This leads us to question what actors exactly are, since they are obviously more than just sets of roles (in fact, we should say "more than just sets of role types").

As another example, the duality instance/concept is not fully acknowledged in UML. In particular, use case diagrams do not make it possible to show the multiplicity of roles with respect to use cases (we speak here of role instances and use case instances, such as the number of players in a party of bridge). Interestingly, this shortcoming comes from the UML notation, not from the structure of the meta-model. Regarding collaboration diagrams, we find that identifiers for roles (CollaborationRoles) are in fact the identifiers for role types: if there is more than one instance of a role type in a collaboration diagram, we face a modelling problem with UML.

These questions and their answers are important because they lead towards a better integration of use case diagrams with class and collaboration diagrams. By addressing these relatively minor use case issues, we hope to make UML a more powerful modelling language, while making it simpler at the same time [8].

8. SUMMARY AND CONCLUSIONS

In this paper, we addressed the issue and questions that standardization experts are facing with the concept of role. We found that experts were all right in a sense 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), a type for role instances (role type) or a type for objects (role object type). The suggestion that role

[1] In the UML, *message* has been defined as a template (specification) rather than an instance. People who found a need for a message instance concept had to pick another term: *stimulus*. But few people are comfortable with this terminology, and the UML standard keeps talking of sending or receiving a message. If role was a reserved term for the type concept, what term should be used for the instance concept?

is analog to interface was also useful, because it lead us to our proposed new definition of role for the RM-ODP Foundations standard [11]. This standard itself, while imperfect with respect to its definition of role, was invaluable to us. It helped us understand what are modelling concept and how they may be 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.

We expect our definitions and our results not only to achieve consensus among ISO delegates, but also to be a basis for improving the UML standard and related software engineering processes.

9. ACKNOWLEDGEMENTS

Acknowledgements go to Trygve Reenskaug for his presentation to the ISO/SG7/WG3 in Paris last year and for subsequent discussions, to William Frank for his various contributions and support, and to all members of ISO/SG7/WG or UML revision task force, for providing us both with an interesting issue and with important leads for our analysis and proposed resolution.

10. REFERENCES

[1] G. Booch, I. Jacobson, and J. Rumbaugh, *The Unified Modeling Language User Guide*: Addison Wesley Publishing Company, 1998.

[2] G. Genilloud, “On the Abstraction of Objects, Components and Interfaces,” presented at OOPSLA Workshop on Behavior Semantics, Denver, Colorado, 1999.

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

[4] Q. Li and R. K. Wong, “Multifaceted object modeling with roles: A comprehensive approach,” *Information Sciences*, vol. 117, pp. 243-266, 1999.

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

[6] 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/)

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

[8] A. Wegmann, G. Genilloud, “The Role of “Roles” in Use Case Diagrams”, Technical Report (in preparation).

[9] OMG, ″OMG UML Specification v. 1.3,″ 1999.

[10] ISO/IEC and ITU-T, "Open Distributed Processing - Basic Reference Model - Part 1: Overview and Guide to Use," Standard 10746-1, Recommendation X.901.1996 (http://isotc.iso.ch/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards).

[11] ISO/IEC and ITU-T, ″Open Distributed Processing - Basic Reference Model - Part 2: Foundations,″ Standard 10746-2, Recommendation X.902. 1995. (http://isotc.iso.ch/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm).

[12] ISO/IEC and ITU-T, ″Open Distributed Processing - Basic Reference Model - Part 3: Architecture,″ Standard 10746-3, Recommendation X.903. 1995. (http://isotc.iso.ch/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm).