A Foundation for the Concept of Role in Object Modelling

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

Alain Wegmann
EPFL
DSC-ICA
CH-1015 Lausanne
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. Our findings and our definitions provide a basis for reconciling the positions that people have about the concept of role.

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 []. 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 []. 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 groups on ODP, SC7 WG16, 17 and 19, are currently working on a number of standards in the ODP area. Of particular importance is a standard for the ODP Enterprise Language (WG17), 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 [, 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 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 [] (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 finding compromises is not our attitude: 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). Be ing 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 looks at the current definition of role in the RM-ODP and shows why it does create problems to the RM-ODP community. Section 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 investigates the question of whether the concept of role is redundant with that of interface, and concludes by the negative. Section contains our proposal for a new definition of role, that addresses all the concerns expressed in the previous sections. Section 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 briefly relates our findings with UML.

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. [, Def. 9.14]

As with many other definitions in the RM -ODP, this definition is not easy to understand on first reading. Most of its text is related to a rather abstract example of use, rather than being definition text. 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.

1.2 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-

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*” [, p. 161].

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*” [, 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/WG17, 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.

In defence of considering roles as identifiers, it must be said that the concept of role is particularly useful when it comes to naming objects or entities. Assume that a scenario description (e.g, a UML collaboration diagram) mentions the involvement of objects o1, o2, o3 and o4. There is quite naturally an implication that exactly four objects are involved. This implication is undesirable whenever two or more of those names may possibly denote one and the same object. To avoid it, a scenario may instead speak of 4 roles, r1, r2, r3 and r4. The problem disappears, because it is well accepted that an object may perform multiple roles.

1.2 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:

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. [, 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)” [, Definition 8.6], even 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 [, 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 [, 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 This second interpretation is compatible with the UML definition of role: *“Role: The named specific behavior of an entity participating in a particular context...*” [, p. B-15].

2– This property does not seem to be true for all object models. For example, the agent role in OSI systems management is performed by a configuration of several objects, called managed objects.

Regarding the second point above, it must be said that in ODP modelling, there is always an equivalent object which abstracts any configuration. Thus, in our systems management example, the server role may be seen as being performed by just one object (an MIS-USER may be modelled in ODP by a computational object, as discussed in []). We will come back on this issue in Section .

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 does try to provide additional qualification with an example (the references to the concepts of "template for a composite object") but this extra qualification is not clear enough for being useful. Moreover, the correspondence made between role and composite object does not find any echo in the general literature about roles and object modelling.

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. We will have to look elsewhere and find another definition.

1.1 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 [], Section 6.2.1). Since a role is a behaviour, and since a behaviour is not 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 all belong to an interface, but they need not belong to any role.

1.1 Roles and Software Development Methods

To illustrate how tools can be used as tool for design, we present briefly two object-oriented software development methods:

1– OOram, a method developed by Trygve Reenskaug []. The methods supports the activity of design by finding roles that collaborate to achieve a goal (i.e., by defining a "role model"). Roles are then implemented by programming language objects.

2– Catalysis, developed by Desmond D'Souza and Allan Wills []. In their method, the authors first analyse the role of an IT application in its business environment, defining the system specification independently from the implementation details. They then implement the system by defining a collaboration of "pluggable-parts" such as programming language classes or components. They define a collaboration as a "set of related actions between typed objects playing defined roles in collaboration" [, p. 716, 2].

Other examples may be found in a survey of role mechanisms in object-oriented modelling by G. Kappel [].

2. 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 [].

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... [, Def. 9.7]*
>
> ***Instance (of a type)****: An <X> that satisfies the type.... [, 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 will now examine these two positions in turn.

1.1 Is Role a Type of Some Other Concept?

There are good reasons for considering role as a type of some other concept. One of them is that role is rather abstract: no one has ever seen an entity that is just a role instance. Also, we tend to think of names of role as denoting types rather than instances. For example, we tend to think of “Hamlet” as being the name for one role, even though there have been or will be many representations of the play by the same name. Similar examples apply to more common roles such as “client,” “server,” “manager,” etc.

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

1.1 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 role and object are different instance concepts. In fact, to be more precise, it implies 1) that role and object are different concepts and 2), that they are not first-order types (because an object is not a type).

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 3, it points to an instance concept.

4– Likewise, UML defines role as “the named specific behaviour of an entity participating in a particular context,” and thus to an instance concept [, p. B-15].

5– Trygve Reenskaug, the inventor of role-modelling for software engineering, says that roles have all the properties of objects [, p. 45]. And of course, objects are instances.

6– 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/WG17, Paris, Nov. 24 1999]. This is an alternate way of saying that there exist both a concept of role and a related concept of role type.

It follows from the above points that role is an <X> in the RM-ODP (since <X> can be anything that has a type). There are therefore several concepts related to the term role: role instance, role type, role class and role template. 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, role class or role template. In the RM-ODP, the same terminology principle applies to the concepts of object, interface, action, operation, etc.

1.1 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 all these different concepts possibly mean, and how they are related. We will address this question in Section , after we have found a suitable definition for role (the instance concept).

2. 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

3 An identifier is 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.

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. [, Def. 8.4]

Note 1 is of particular importance. It makes it clear that the definition refers to an abstraction by projection, rather than to an abstraction by conceptualization []. An ODP interface is thus a subset of the behaviour of an object, and not an object type as in UML. This subset is itself a behaviour (we refer the reader to our definition of behaviour in Section). It is important to understand that hiding an interaction turns it into an internal action, and that redundant internal actions may be removed (hiding actions is explained in []). An interface therefore includes both internal actions and interactions.

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 the 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 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. 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 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).

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 4 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, every programmer has heard of these pairs of roles: client-server, provider-subscriber, and manager- agent.

4– 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 5. 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.

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

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

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.

2. 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 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).

3- It is possible for two role types to be considered equivalent even though they are defined with respect to two different collaborative behaviour types.

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.

Most of the notes in our definition have already been discussed in Section , when we stressed the difference between role and interface. The definition itself and Note 1 add a crucial difference, motivated by the fact that role is a design tool: they make 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 3 makes it possible to modify a specification for a collaborative behaviour without changing all the roles involved. That is, those role specifications that were not affected by the modifications may be considered as remaining the same role specifications.

Regarding Note 6, we refer the reader to Section . 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 , that a role always corresponds 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 the 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*" []. In other words, a role is performed by one object, when it is performed at all.

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.

1.1 Combining Roles and Interfaces

To our knowledge, interfaces (in the ODP sense) are never explicitly considered when making role models: 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. We may therefore consider adding an extra note to our definition of role, to reflect this fact:

8 - 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 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.*

To make sense of this extra note, 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 8 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 want to 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 [] (client and server interfaces are specialisation of the general concept of interface; they are defined in the RM-ODP Architecture []).

We think that Note 8, which restricts each role to have at most one identifying interface, adds a perfectly reasonable constraint to the definition of role. Indeed, role would lose its value as a design tool if modellers were forced to deal at the same time with both roles and interfaces. On the other hand, Note 8 adds little to the understanding of the role concept, and there is little need to impose it by including it the RM-ODP standard.

2. 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.

1.2 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 *collaborative behaviour.* While object behaviours are generally infinite (composed of an infinity of actions), collaborative behaviours and thus roles are often finite (this fact contributes to make of roles an effective design tool). For example, a collaborative 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, collaborative 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 collaborative behaviours are unbounded. A modeller cannot describe them all one by one. So, rather than specifying roles or collaborative behaviours, a modeller specifies role templates or collaborative 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 [, 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....*

Collaborative Behaviour Template*: The specification of the common features of a collection of collaborative behaviours in sufficient detail that a collaborative behaviour can be instantiated using it....*

A telling example of a collaborative behaviour template is a UML use case, except for the fact that the roles in a use case are usually un-named (the names of actors, and "the system" are used instead of specific role names).

Note that it is specification, not instantiation, that is the primary purpose of role templates and collaborative 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 collaborative 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... [, Def. 9.13]*

Instance (of a type)*: An <X> that satisfies the type... [, 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.... [, Def. 9.19]*

The relationships between a template, its instantiations and its instances are quite subtle. We explain them in [].

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

To conclude this section, we observe that the current RM-ODP definition of role mentions "a template for a composite object" [, Def. 9.19]. We suppose that the ISO experts who designed this definition were thinking of "a template for a collaborative 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 collaborative behaviour).

1.2 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 , 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 6.

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.

2. The Case of UML

We are currently working on applying the results of this paper to improve the consistency of UML while extending its modelling capabilities. UML handles role better than we initially expected. The general UML literature makes it worse than it is by not using or explaining all the capabilities and subtleties of UML.

The UML definition of role, "The named specific behaviour of an entity participating in a particular context...," is both close and compatible to ours. However, "behaviour" must be understood in its ODP interpretation for the definition to make sense. Moreover, the UML definition is insufficient to fully characterize roles, because it does not explain what is meant by "in a particular context." Finally, it leaves unclear which concept of role instance or of role [class] is the one being defined. Our position is that UML should have two definitions for role, one for "role instance" and another for "role [class]", as it does for use case [, p. B-19].

Roles are important for understanding the semantics of use case models (a use case instance is quite clearly a collaborative behaviour). So what is the treatment of roles in use case models?

1– In a use case, roles are usually un-named, at least in the UML literature: the names of actors and "the system" are used instead of specific role names. This practice not only hides the link between roles and use cases, but it also makes use cases much less reusable, as we explain in [].

6 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?

2– Actors are defined in UML as “sets of roles...” There is some truth in this statement, even though an actor is obviously not a set, since it performs actions. An actor may perform many use cases, and the same use case multiple times. An actor may therefore be considered as a composition of roles, yielding an infinite behaviour. Our recommendation is to redefine actor as “a composite role...”

3– If an actor is a role, why is it possible to have a single actor attached to a use case (considering that a role is always defined with respect to other roles)? The answer, of course, is that the system is an implicit participant in every use case, and that its roles are un-named. For consistency, and for other reasons that we give in [], we think that the roles of the system should be made explicit in use case diagrams.

4– UML makes it possible to show the multiplicity of roles in use case diagrams (we speak here of role instances with respect to a use case instance, such as the number of players in a game of bridge). This capability is almost never shown in the UML literature. This is unfortunate as multiplicities are a powerful means for specifying use cases.

Roles are obviously important in collaboration diagrams. What is their treatment in UML?

UML is ambiguous about whether CollaborationRoles are role instances or role types. But if we must choose, then we would consider them as being role types. Within the context of a collaboration diagram, a role instance may indeed remain un-named, as long no other instance of the same role type exists in the diagram. In other words, the CollaborationRole name (i.e, the name preceded by a ‘/’, such as Teacher in ‘/Teacher’) serves both as an identifier for a role type and for a role instance. However, UML has no naming mechanism for a collaboration diagram in which there are several instances of a same role type (such as four players in a game of bridge). If an instance name is given, it applies to the object that performs the role, rather than to the role instance: for example, tutor / Teacher means an object named tutor playing the role Teacher [, Table 3.1]. We will investigate the practical consequences of this UML limitation in our future work.

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 issues, we hope in our future work to contribute to make UML a more powerful modelling language, while making it simpler at the same time.

2. 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 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 suggestion by UML experts that role is analog to interface was very useful to us: much of the text in our proposed new definition of role is derived from our analysis of the similarities and differences between these two concepts.

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.

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.

2. Acknowledgements

Acknowledgements go to Trygve Reenskaug for his presentation to the ISO/SG7/WG17 in Paris last year and for subsequent discussions, to William Frank for his 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. Special thanks to an anonymous reviewer for his or her valuable comments.

2. References

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

[2] D. F. D'Souza and A. C. Wills, *Objects, Components and Frameworks With UML: The Catalysis Approach*: Addison- Wesley, 1998.

[3] 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/).

[4] G. Genilloud, "On the Abstraction of Objects, Components and Interfaces," presented at the OOPSLA Workshop on Behavior Semantics, Denver, Colorado, 1999. (available at URL http://icawww.epfl.ch/).

[5] G. Genilloud, "An Analysis of the OSI Systems Management Architecture from an ODP Perspective," presented at IEEE Second International Workshop on Systems Management, Toronto, 1996.

[6] G. Kappel, W. Retschitzegger, W. Schwinger, "A Comparison of Role Mechanisms in Object-Oriented Modeling," presented at Modellierung'98, K. Pohl, A. Schürr, G. Vossen (eds), Bericht Nr. 6/98-I, Angewandte Mathematik und Informatik, Universität Münster, pp. 105-109, 1998

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

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

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

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

[11] 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/)

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

[13] 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.

[14] A. Wegmann and G. Genilloud, "The Role of "Roles" in Use Case Diagrams," presented at Third International Conference on the Unified Modeling Language (UML2000), York, UK, Oct. 2000.

[15] OMG, "OMG UML Specification v. 1.3," 1999.

[16] 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).

[17] 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).

[18] 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).

2.

Missing

OOPS-3 -- Furthermore not all relevant literature in the context of roles is addressed adequately. Missing:

Kristensen, B., B., and Osterbye, K.: Roles: Conceptual Abstraction Theory and Practical Language Issues. Theory and Practice of Object Systems, 2(3), 1996. Page 143 – 166.

Gottlob, G., Schrefl, M., and Röck, B.: Extending Object - Oriented Systems with Roles. ACM Transactions on Information Systems, 14 (3), July 1996. Page 268 – 296.

Riehle, D., and Gross, T.: "Role Model Based Framework Design and Integration." In Proceedings of the 1998 Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '98). ACM Press, 1998. Page 117-133.