This article has been published in Distributed Objects & Application (DOA 2000) September 21-23, Antwerp, Belgium

A Comparison of the Basic DO Concepts in Standardization

X. Blanc (*+), M-P. Gervais(*), J. Le Delliou(+)
(*)Laboratoire d'Informatique de Paris 6 - 8 rue du Capitaine Scott F75015 PARIS
(+)EDF Research Division - 1, av du Gnl De Gaulle F92141 CLAMART Cedex
Xavier.Blanc@lip6.fr, Marie-Pierre.Gervais@lip6.fr, Juliette.Le-Delliou@edf.fr

Abstract

This paper provides a comparison of the basic DO (Distributed Object) concepts, namely "object", "instance" and "class" used in various standards of the DO world. Two families of standards are identified: those related to the modelling aspects (MOF and UML) and those related to the architectural and implementation aspects (CORBA and Java). Moreover, the RM-ODP standard is considered as it includes both aspects.

The objective is to help out with a common understanding of these concepts. For this, we compare these concepts as defined in these standards according to the four layer architecture proposed by the MOF standard. An example is provided to illustrate the comparison.

1. Introduction

Several standards are commonly used in the DO world. We can classify them into two families: standards related to modelling aspects such as OMG's UML and MOF and those related to architectural and implementation aspects such as OMG's CORBA and Sun's Java [5][6][7][8]. Another standard covers both aspects: ISO's RM-ODP [2]. Indeed, although RM-ODP defines an architectural framework for distributed object systems, it provides a set of concepts useful in modelling too, especially concepts from the enterprise viewpoint.

Communication between people who refer to these standards is sometimes difficult because of the various meanings of the basic concepts they provide. We have realised a test within our company, asking twenty people for the definition of the basic concepts in DO, namely "Object", "Instance" and "Class". The result was amazing, we had close to twenty different definitions. The definitions were quite equivalent but little differences changed the meaning. After analysing these differences, we found that they came from the profile of the person. It appears that UML people use quite the same definition, but people using MOF or RM-ODP have another one.

Standards are partly useful because they provide a set of concepts and rules shared among a community. But the multiplicity of standards in the same area sometimes leads to some confusion since they do not use the same words with the same meaning.

This paper aims to clarify all possible interpretations of elementary concepts used in DO in comparing them in several standards. The chosen concepts are "object", "instance" and "class". The standards are those already mentioned, i.e., MOF, UML, RM-ODP, Java and CORBA. A first question that appears when comparing these concepts is: How to compare them? Which criteria can be useful to conclude that two concepts are equivalent or not? For this, we use the MOF architecture that defines the concept of "**layer**". Thanks to this concept, we classify the three concepts according to this four layer architecture and we then compare them.

Part 2 of this article explains the four layer architecture, details the concept of "layer" and presents the definition of a MOF Object, a MOF Instance and a MOF Class. Part 3 presents UML in the context of the four layer architecture and explains the concepts of UML Object, UML Instance and UML Class. Part 4 presents the RM-ODP in the context of the four layer architecture and explains the concepts of ODP Instance, ODP Object, and ODP Class. Part 5 presents Java and CORBA in the context of the four layer architecture and explains the concepts of Java Instance, Java/CORBA Object, Java Class and CORBA Interface. Part 6 presents a classification of all these concepts in the four layer architecture. And then part 7 concludes.

2. The four layer architecture

Before dealing with the four layer architecture, we present a two layer architecture. This provides the needed background to understand the concept of "layer" and then to understand the four layer architecture.

2.1 A two layer architecture

The two layer architecture deals with data and model of data.

Let us first introduce a definition of "model" taken from an English dictionary: "A model is a schematic description of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics: *a model of generative grammar; a model of an atom; an economic model".*

This definition leads us to consider two worlds (Figure 1). The world of data to be described, also called the universe of discourse or the real world (although it can be abstract and not real). And the world of the model, this world contains descriptions of the "real world".

We have the "real world" containing **data** and the "model world" containing descriptions of data. These descriptions of data are called **meta-data.**

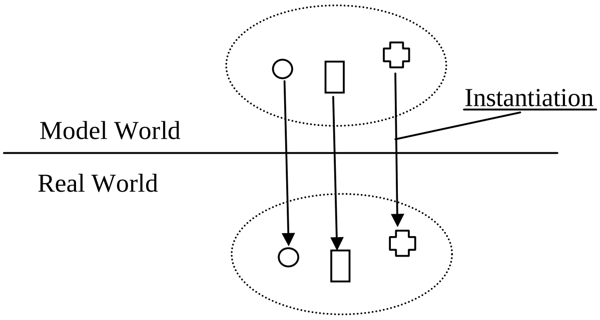

Fig 1 : The two layer architecture.

Basically, the relation between the "model world" and the "real world" defines the concept of "layer". **A model belonging to layer M_n describes data belonging to layer M_{n-1}.**

A model is composed of several elements, each of them describes elements of the lower layer. The relation between elements of the model and elements of the real world is an **instantiation**. An element of the real world is an **instance** of an element of the model.

2.2 The MOF architecture

Thanks to the two layer architecture, we can understand what a layer is. The four layer architecture described by the MOF (Meta Object Facility) standard is just a generalization of the concept of layer [6].

A model is composed of elements called meta-data. These meta-data can be considered as data. In that way, we can build a new model that describes these data.

To sum up, meta-data describe data, and a model is a set of meta-data. Considering this model as data, there is a model that describes it. This new model is a model of model, it is a **meta-model**.

Meta-models describe models as models describe data. These meta-models are composed of **meta-meta-data**. It should be noted that the term meta-model is used more than meta-meta-data.

From a two layer architecture, a three layer architecture is now built. The relationship between elements of two adjacent layers is an instantiation. Data are instances of meta-data, meta-data are instances of meta-meta-data, etc.

A four layer architecture can be built based on the three layer architecture in the same process as the previous one used to build the three layer architecture from the two layer one. The question is now: when does this process stop ? As layers are defined thanks to the relationship between described elements and descriptors, it could be possible to imagine architectures with n layers.

Of course, it is needed to define the limits of these architectures. What is the lowest layer? The MOF standard defines it as the layer that describes nothing. The M0 layer is only composed of the things to be described, it does not describe anything.

Then, the M1 layer is the layer that describes the M0 layer. Examples of models belonging to M1 layer are illustrated in the MOF standard (e.g. UML models).

The M2 layer describes the M1 layer. Examples of M2 models are the UML meta-model that defines the structure of all UML Models, or the Workflow meta-model that defines the structure of all workflows models etc.

Last, but not least, the MOF standard describes the M3 layer. Models belonging to the M3 layer describe M2 models. Actually, the MOF standard stipulates that only one model belongs to the M3 layer, called the MOF Model. Moreover, it defines that the M3 layer is the highest layer. This is possible because the MOF model can describe itself. Thus, the MOF model describes the MOF model and it is the only one model belonging to M3 layer.

To sum up :

- Data belong to the M0 layer. Data do not describe anything.
- Meta-data belong to the M1 layer. A model is composed of meta-data and it describes data. Data are instances of meta-data.
- Meta-meta-data belong to M2 layer. A meta-model is composed of meta-meta-data and it describes meta-data. Models are instances of meta-models.
- Meta-meta-meta-data belong to M3 layer. The MOF model is the meta-meta-model, it is composed of meta-meta-meta-data, and it describes meta-models. Meta-models are instances of the MOF model. The MOF model describes itself.

The figure 2 shows the four layer architecture defined by the MOF standard.

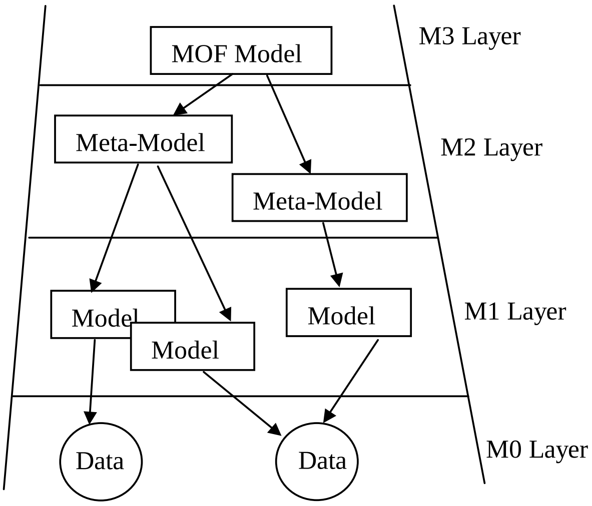

Fig 2: the four layer architecture defined in the MOF standard

2.3 Definition of "layer"

In this four layer architecture, the MOF standard defines the concept of type as follows: the type of a model is defined in its meta-model.

So, we can distinguish two meanings of this concept.

1. A model is of type Tm, the type of the model is **Tm**.
2. A type is defined by a model. A model defines the type **Td**.

These two meanings are linked because the type of a model is defined by its meta-model:

Tm(of a model) = Td (of its meta-model).

If we look at the types of models in the four layer architecture then we have:

- For M3 layer, the type of the MOF model is Tmof. The MOF model defines the type Tdmof.
- For M2 layer, the type of a meta-model is Tmm. A meta-model defines a type Tdmm.
- For M1 layer, the type of a model is Tm. A model defines a type Tdm.
- For M0 layer, the type of data is T. Data do not define anything.

Then, keeping in mind that the "type of a model is defined by its meta-model" we have:

- The type of the MOF model is the same as the type defined by the MOF model.
  (E1): *Tmof = Tdmof*

- The types of meta-models are defined by the MOF model. So Tmm = Tdmof. Applying (E1) we obtain:
  (E2): *Tmm=Tdmof=Tmof*

  i.e. the type of a meta-model is the same as the type of the MOF model.
- The types of models are defined by meta-models.
  (E3): *Tm=Tdmm*

- The types of data are defined by models.
  (E4): *T = Tdm*

These properties are very helpful to know if a model belongs to M0, M1, M2 or M3 layer and we will use it to compare the Instance, Object and Class concepts.

2.4 Object, Instance and Class in the MOF

We have presented the four layer architecture defined in the MOF standard. In this presentation, we have used several times the word "instance". In fact, the MOF defines the concepts of instance, object and class as follows.

An instance is defined as a relationship. Thus:

- Data are instances of meta-data.
- Meta-data are instances of meta-meta-data and models are instances of meta-models.
- Meta-models are instances of the MOF model.
- The MOF model is an instance of itself.

In the MOF standard, due to the encapsulation principle, meta-data are also called meta-objects. Meta-objects belong to a model, they are in the M1 layer. Applying this view to the M0 layer, a data is then considered as MOF object, which belongs to the M0 layer.

Classes in the MOF standard describe meta-objects. In other words, a class is an element of a meta-model. It belongs to the M2 layer.

According to (E2), the type of the MOF model is the same as the types of the meta-models. So, the MOF model is also composed of classes. But to distinguish the MOF model from the other meta-models, its elements are called meta-classes. Meta-classes are elements of the MOF model, they belong to the M3 layer.

The figure 3 shows the concepts of object and class as defined in the MOF standard. In the next sections, we will classify these concepts as defined by other standards in this array to compare them.

Layer	Term
M0	**Object**
M1	**Meta-object**
M2	Meta-meta-object or **Class**
M3	Meta-meta-meta-object or **Meta-class**

Fig 3: Object and class concepts in the MOF.

2.5 An example

We have chosen to illustrate the four layer architecture with a simple example. We will also use this example with the other standards to be sure of our comparison.

This example is the classic one of a cat sitting on a carpet. A cat named Pussy is sitting on a Moroccan carpet.

According to the MOF standard, the cat Pussy and the carpet are data to be described. They do not describe anything. According to (E4) they belong to M0 layer, they are objects.

Describing this cat and this carpet consists of building a model. This model will be composed of meta-objects, the model and its components belong to the M1 layer.

If we want to describe the model, for example to explain the concepts that we have used, we must build a meta-model. This meta-model describes the model, it is composed of classes. This meta-model and its components belong to the M2 layer.

If the meta-model is MOF compliant, then it is described by the MOF model that belongs to the M3 layer and that is composed of meta-classes.

The figure 4 shows the example of the cat in the context of the MOF. It shows the differences between a MOF Object and a MOF Class. The MOF objects belong to the M0 layer, they are the elements to be described. A MOF Class belongs to the M2 layer, it describes meta-objects.

Regarding to the MOF instance, it should be noted that everything is an instance of something. For example, the meta-object "the cat" is an instance of the Class "inanimate meta-object". MOF instance is a relationship, that's why it does not belong to a specific layer.

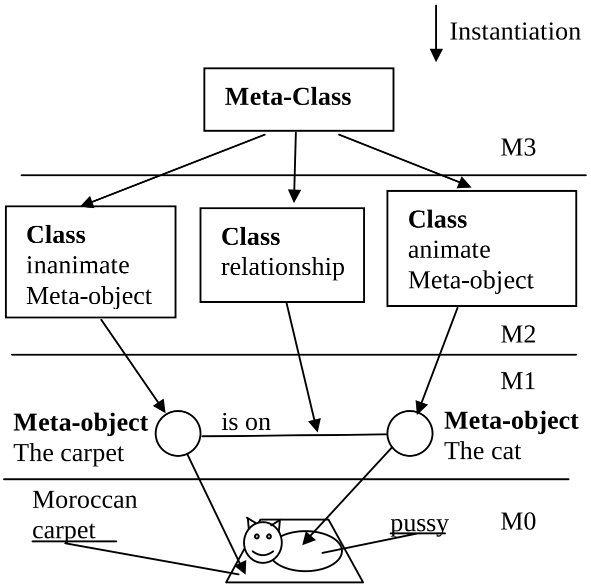

Fig 4: The example of the cat in the MOF

3. UML

UML (Unified Modelling Language) is one of the most popular language to model oriented-object applications [5]. Although UML is only a notation, there are strong definitions for UML Object, UML Class and UML Instance

UML is often used as an input when building new applications, that's why there are several mappings between UML and programming languages. These mappings will also help us to compare the concepts.

3.1 UML in the four layer architecture

In all the specifications of UML, from 1.0 to 1.3, models are used to define UML concepts. The notation used to build these models is UML.

The core of UML is the basis of all other concepts, i.e., it is sufficient to define the other parts.

As the UML core was used to define MOF model, then models contained in the UML standard are now MOF compliant. The UML specification provides the UML meta-model, which belongs to the M2 layer and which is an instance of the MOF model (Figure 5).

It should be noted that the UML meta-model is not the UML specification. For example, the UML meta-model does not deal with the notation, the UML specification does.

The UML meta-model describes the structure of all the UML models. A UML model is an instance of the UML meta-model, it belongs to the M1 layer and it describes data.

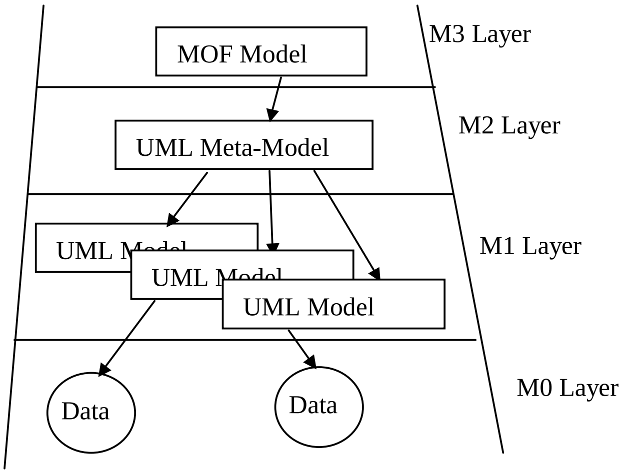

Fig 5: the UML meta-model, an instance of the MOF model

3.2 Object, Instance and Class in UML

The UML meta-model defines the concepts of UML Instance, UML Object and UML Class. These concepts are defined in the M2 layer. So, according to (E3), an element of type “UML Class”, called a UML Class, belongs to the M1 layer. This is the same for UML Instance and Object.

UML Instances, UML Objects and UML Classes belong to the M1 layer, they are MOF meta-objects, they are components of UML Models. This proposition is enforced by the fact that all of them inherit from "UML model element" that is defined in the UML meta-model as a component of the UML model. They define UML entities [4].

A UML entity is the thing to be modelled, it belongs to M0 layer. It should be noted that the concept of entity is not defined in the UML meta-model, it is mentioned in the UML specification.

More precisely, a UML instance defines an entity to which a set of operations can be applied and which has a state that stores the effect of the operations. A UML object is an instance that originates from a UML class. And a UML Class is a set of objects that share the same features.

The figure 6 classifies these concepts in the four layer architecture.

Layer	Term
M0	***Entity***
M1	***Class***, ***Instance***, ***Object***

Fig 6: UML terms in the four layer architecture

From this figure, one can notice that the UML standard does not deal with M2 and M3 layer.

This classification will help us to compare the concepts in the context of each standard. But right now, we can already say that a MOF object is totally different from a UML Object. We will detail this comparison in the part 6.

Let us go back to the example of the section 2.5. Figure 7 shows this example using the UML meta-model.

We can see in the M2 layer a part of the UML meta-model. This part defines the UML concepts of *Object*, *Class* and *Association.* Each of these UML concepts is a MOF class. Thus the UML meta-model is composed of three MOF classes, each of them is an instance of a MOF meta-class. This part of the UML meta-model defines a part of the structure of all UML models.

In this example, we have one UML model. This model is the model of the cat sitting on the carpet. It is composed of two *Classes*, namely "Carpet" and "Cat", two *Objects* "the carpet" and "the cat" and so on. Each of these elements is a MOF meta-object. Their types are defined by the UML meta-model.

In UML, an instance is an element linked to a class. It should be noted that in the UML meta-model, UML objects inherit from UML instances. This is why no instances appear in this example.

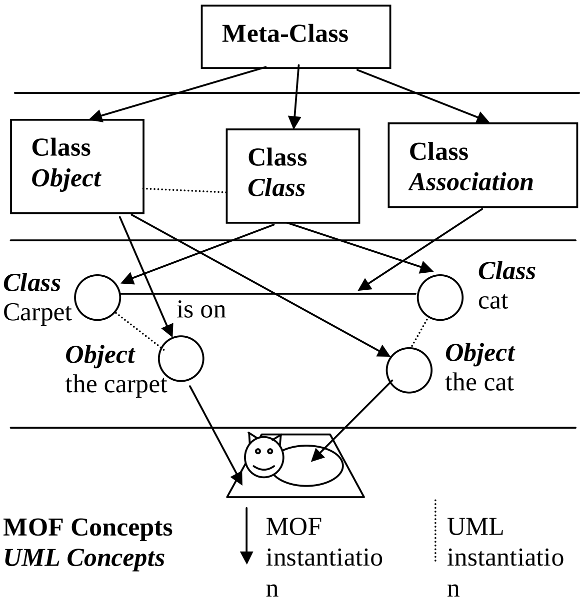

Fig 7: Example of the cat with the UML meta-model

4. RM-ODP

RM-ODP (Reference Model of Open Distributed Process) is an ISO standard [1]. It gives concepts and structuring rules for specifying open DO systems. An RM-ODP system could be an application or an organisation of human. The definitions provided by RM-ODP are rigorous and consistent. RM-ODP also defines five viewpoints that establish the separation of concerns needed to specify different facets of a system. Some concepts are specific to some viewpoints but the concepts of Class, Object and Instance are common concepts.

RM-ODP does not provide any notation to express specifications, moreover it is method-free.

4.1 RM-ODP in the four layer architecture

Some works have been done to compare RM-ODP and UML [3]. Their goal is to compare the power of expression of these two standards. Since the MOF standard is available, research has been oriented towards the construction of the RM-ODP meta-model. The next normative part of the RM-ODP will certainly include a meta-model for describing some of the RM-ODP concepts [2].

Open distributed systems specified according to the RM-ODP concepts are the things to be modelled, so they belong to the M0 layer.

A specification of an open distributed system is a model of the system as it describes the system, so it belongs to the M1 layer.

The RM-ODP standard describes the structure of all the RM-ODP specifications.

As it is for UML, the RM-ODP meta-model will not consider the whole standard, i.e., it will not define all the RM-ODP concepts. For example, it will not define the concept "entity", which is defined in that same way as UML, i.e. the thing of the real world.

The figure 8 shows the relationships between RM-ODP, the MOF and UML.

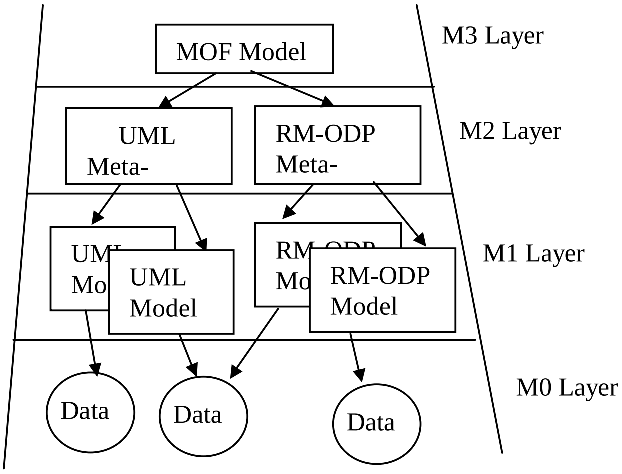

Fig 8: RM-ODP and UML in the four layer architecture

4.2 Object, Instance and Class in RM-ODP

The RM-ODP standard defines the concepts of instance, object and class.

Before defining them, in order to help the understanding, let us first introduce the concepts of entity and type:

- A RM-ODP entity is any concrete or abstract thing of interest. It is something to be modelled, it belongs to the M0 layer.
- A RM-ODP type is a predicate. RM-ODP deals with type of an <X> when an X can satisfy a type.

Then RM-ODP defines the concepts of instance, object and class.

An object is a model of an entity. Objects belong to the M1 layer.

A class is a set of elements that satisfy the same type. More precisely, RM-ODP deals with classes of <X> where <X> are elements satisfying a same type T.

An element satisfying a type is called an instance of the type.

To sum up, RM-ODP classes, instances and objects belong to the M1 layer (Figure 9).

Layer	Term
M0	Entity
M1	Class, Instance, Object

Fig 9: RM-ODP terms in the four layer architecture

We can now present our example using RM-ODP as meta-model (fig 10). This example shows the differences between RM-ODP concepts and MOF concepts. For sake of simplicity, it only deals with classes of object and types of object. Actually, the concepts of class, type and instance are not reduced to objects. As already mentioned, they can be applied to an <X> where <X> can be an action, or interfaces or other elements defined in RM-ODP. Thus classes of actions or classes of interfaces could be found. It should be noted that an instance of a type of object is an object.

The concept of association is not defined in RM-ODP. We have chosen to express it with a role. This choice has not impact on the comparison of instances, objects and classes.

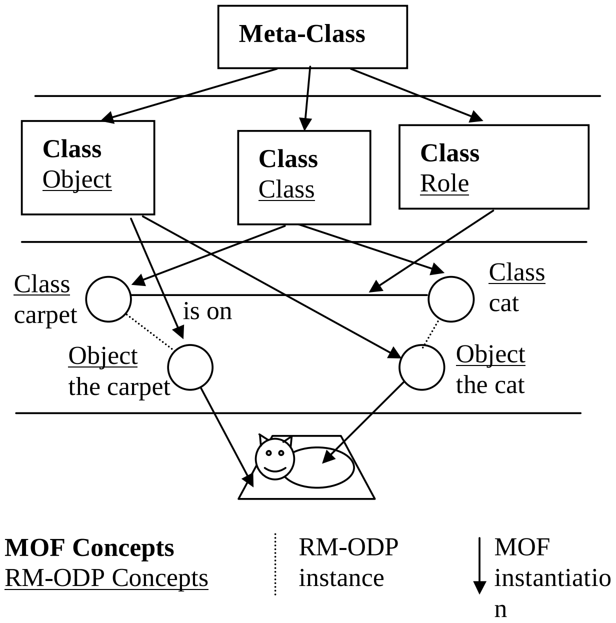

Fig 10: example of the cat with the RM-ODP meta-model

5. Java and CORBA

Java is one of the most popular programming language [7]. It is an oriented-object language that makes use of the concepts of instance, object and class.

CORBA (Common Object Request Broker Architecture) is a middleware useful for building heterogeneous DO applications [8]. Although CORBA applications are written in a programming language, there are CORBA definitions for Objects and Interfaces.

It then would seem interesting to study how these concepts are defined in such standards used in an implementation purpose rather than a modelling one. Moreover, Java programs and IDL interfaces frequently can be automatically generated from modelling languages, especially UML. Including Java and CORBA in our comparison could then provide some feedback on the concepts mapping between modelling and implementation viewpoints.

5.1 Java and CORBA in the four layer architecture

Java and CORBA are not modelling languages. So we face a problem when applying the previous classification based on the MOF architecture to a Java program and IDL interfaces. We have to determine what concerns the real world, what concerns the model and so on. We assume that the runtime is an element of the real world, it belongs to the M0 layer.

A Java program, either the source or the bytecode, describes the runtime. It belongs to the M1 layer. For CORBA, IDL interfaces are used to describe the runtime; this description is completed with a programming language (Java for example). IDL interfaces belong to the M1 layer.

The Java specification describes the structure of all Java programs. We can assume that the Java meta-model, if it would exist, would belong to the M2 layer. The CORBA specification includes the IDL grammar that describes the structure of all IDL interfaces. We can assume that the CORBA meta-model, if it would exist, would belong to the M2 layer.

The figure 11 shows Java and CORBA in the four layer architecture. It should be noted that Java sources and bytecode are considered to be equivalent.

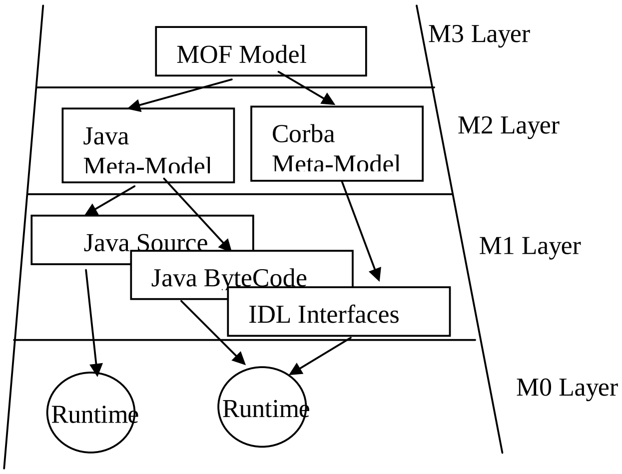

Fig 11: Java and CORBA in the four layer architecture

5.2 Object, Instance and Class in Java and CORBA

The Java specification is not so clear as the other standards for the concepts of class, instance and object. However, there is a definition commonly agreed of them. From this definition, a Java program is composed of classes while objects belong to the runtime, i.e., they are instances of classes.

The CORBA specification defines an Object as an element of the program. CORBA also uses the term "servant"[1]. Although the goal of this article is just to compare the concepts of "instance", "object" and "class", it should be noted that we introduce IDL interfaces as they model CORBA objects.

This is illustrated in Figure 12 with the example of the cat.

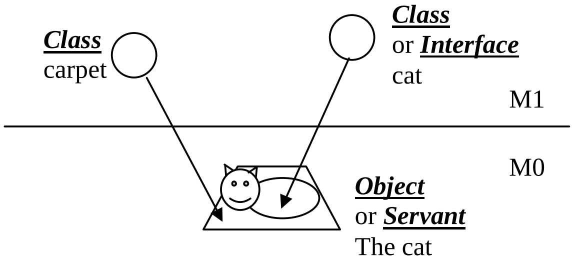

Fig 12: Example of the cat with Java and CORBA

Another point of view can be considered and then leads to other definitions of Java classes and objects. Let us consider a line of code that can be found in a Java program such as:

String _text = new String("text");

In this line of code, there is an identifier named "_text" and its type is "String". This identifier can be considered as a model of something and consequently, it belongs to the M1 layer. In the Java language, this identifier is also called an object. This means that objects can be found in the M1 layer as well as objects can be found in the M0 layer.

On the other hand, in the Java virtual machine, when an object is built, an image of the class is also built. This image is also called a class. This means that classes belong to the M0 layer as well as M1 layer.

The figure 13 shows this ambiguity. There are no strong definitions of Class, Instance and Object. Here, the same terms are used to define M0 or M1 concepts.

Layer	Term
M0	***Java Object,*** *Java Class,* ***CORBA Object***
M1	*Java Object**, Java Class, CORBA Interface***

Fig 13: Java and CORBA terms in four layer architecture. Word in bold represents the usual meaning.

6. Classification and Comparison

Figure 14 summarises all the terms presented in this paper. We can then compare them.

- Objects:
  There are several different meanings for objects according to the standards. It is clear that MOF objects cannot be compared to UML and RM-ODP ones. We will admit that, using the common definition of Java, Java objects are equivalent to MOF objects. UML and RM-ODP have quite the same definition for objects.
- Classes:
  Again, there are several different meanings for classes according to the standards. MOF classes belong to the M2 layer and they are not comparable to anything. UML, Java and RM-ODP are comparable. RM-ODP and UML classes are quite equivalent even if RM-ODP classes are more general. Java classes are more oriented programming, but they are close to UML classes.
- Instances:
  Once again there are several different meanings for instances. For the MOF, instantiation is a relationship, so an instance can belong to any layers. UML and RM-ODP instances are different. UML instances are linked to UML Class on the contrary RM-ODP instances are linked to RM-ODP type. A Java instance belongs to M0, it is a Java object.

Standard Layer	**MOF**	***UML***	RM-ODP	***Java CORBA***
M0	**Object**	***Entity***	Entity	***Object,*** *Class*
M1	**Meta-object**	***Class, Object, Instance***	Class, Object, Instance	*Object,* ***Class***
M2	**Class**			
M3	**Meta-class**			

Fig 14: Used terms in MOF, UML, RM-ODP, Java and CORBA

7. Conclusion

DO applications development is more and more complex. From modelling to implementation, several teams are involved in the same project, and each of them refers to standards devoted to its area.

The project development success is based on the mutual interaction of the involved teams and the required

---

[1] A "servant" is an CORBA object linked to a POA (Portable Object Adapter)

condition of a common understanding. Thus a traceability of concepts must be ensured between the various meanings of the same terms defined in different standards. We could argue that, as well as most of these standards deal with interoperability, the terms they use should be interoperable too.

This paper highlights the existing heterogeneity in DO standards that frequently leads to misunderstanding and confusion between people. It contributes to cross over the gap between the various meanings of concepts used in DO standards. Once the comparison achieved, it seems necessary to be rigorous when using these concepts in order to build DO applications in an effective manner. An easy way to be very clear and precise and to be well understood is to prefix the concept with the standard name. Speaking of a UML object or a CORBA object rather than an object enables the interlocutor to know exactly what the discussion is about. In this way, communication between development teams should gain in efficiency.

8. References

[1] ISO/IEC IS 10746-x — ITU-T Rec. X90x, *ODP Reference Model Part x*, 1995.

[2] ISO/IEC JTC1/SC7 CD 15414, *ODP Reference Model : Enterprise Viewpoint*, January 2000.

[3] P. Linington, *Options for Expressing ODP Enterprise Communities and their Policies by Using UML*, In Proceedings of the 3rd International Enterprise Distributing Object Computing Conference (EDOC'99), IEEE Press, Mannheim, Germany, September 1999, pp72-82.

[4] OMG: UML Specification v1.3 June 1999. www.omg.org 99-06-09

[5] G. Booch, J. Rumbauch and I. Jacobson, *The Unified Modelling Language User Guide*, Addison Wesley

[6] OMG, *MOF Specification v1.3*, July 1999 www.omg.org.

[7] J. Gosling, B. Joy and G. Steele, *The Java Specification*, Addison Wesley
java.sun.com

[8] OMG, *CORBA 2.3.1 Specification*, OMG TC Document formal/99-10-07.