DRAFT (97/08/05) — submitted to
the OOPSLA'97 workshop (Nr. 27) on OO Behavioural Semantics
October 6, 1997 - Atlanta, Georgia, USA

Object Modelling and Common Objects in the RM-ODP Information Language

Guy Genilloud
Swiss Federal Institute of Technology of Lausanne (EPFL)
EPFL-DI-LIT, CH-1015 Lausanne, Switzerland
guy.genilloud@di.epfl.ch

Abstract

This paper argues that the Reference-Model for Open Distributed Processing (RM-ODP) is a suitable basis for developing new standards pertaining to domain architectures, common business facilities, and common objects. However, the RM-ODP is difficult to understand, and there appears to be several divergent interpretations of it. This paper proposes and discusses an interpretation of information modelling that is both fully consistent with the ODP foundations, and compatible with some of the best practice in object-oriented software engineering.

1 Introduction

Large companies in business domains such as manufacturing, healthcare, insurance, and finance, are currently addressing an important but daunting task: the standardisation of computing facilities that are specific to business domains, and the definition of architectures within which these facilities can be successfully developed and used. This standardisation implies agreements about objects that are common to a business domain, or even across multiple business domains — these objects are called “common business objects”.

However, in the absence of a reference implementation architecture and a common software engineering process, there is considerable difficulty in agreeing on just what an object is. Moreover, agreements must first be found at an abstract (implementation independent) and semantical level, prior to an implementation level. Otherwise, the use of common objects may result in an increased level of confusion, rather than in the desired improvements in commonness, complexity and reuse.

This paper argues that the Reference-Model for Open Distributed Processing (RM-ODP) is a suitable basis for addressing the problems mentioned above. The RM-ODP includes a rich set of modelling concepts that are applicable to all kinds of (object-based) models, and it provides definitions that are both precise and mutually consistent [X.902]. It also defines a set of five viewpoint languages that are a sufficient basis for addressing the construction and use of large distributed systems [X.903].

However, the RM-ODP suffers from two deficiencies: it is difficult to apprehend, and it is sometimes excessively general (being terse and overly open is often the price to pay for reaching International consensus, and for obeying ISO rules). As a result, its significance and usefulness are not yet well perceived. Moreover, there are several divergent interpretations of the RM-ODP viewpoint languages.

This paper proposes an interpretation of information modelling that is both fully consistent with the ODP foundations, and compatible with some of the best practice in object-oriented software engineering (e.g., Z, [Catalysis], and [Fusion]). Specifically, we argue that computational models should be seen as refinements[1] of information models, much in the same way than engineering models can be seen as refinements of computational models (which is probably less controversial).

1. By refinement, we only mean that a “refined specification” must be behaviorally compatible with a more abstract specification. There is no requirement that one specification be derived from the other, nor event expressed in the same notation.

2 Improving Concepts and Terminology

The fuzziness that surrounds common business objects and related concepts is linked to the use of improper terms, as well as to insufficient qualification of the context in which the terms are used:

- By improper terms, we refer to the fact that the term 'object' is often used when the terms 'object type', 'object class', 'object template', or yet 'interface' would be more appropriate — for example, people tend to use the term 'common business object' when they actually mean 'common business class'.

  This problem is partially due to the fact that different computer science communities (information analysis, database, programming languages, operating systems) have developed their own perspective of object-orientation. These perspectives are incomplete, and can be inconsistent with one another.

  Another problem is that the usual "object-oriented terminology" has drifted away from English. This leads to clashes between the "object-oriented terms" and the English terms, which remain necessary and which are indeed used. For example, most languages or methods define 'class' as "an object template from which instances might be instantiated." This definition matches none of the uses of the word 'class' in English.

- By insufficient qualification of the context, we mean that it is not clear in what kinds of models the objects and related concepts are supposed to be used.

  Software engineering tells us that several models, at different levels of abstraction, are necessary to address the complexity of building large systems: an analysis model, a design model, an implantation model, and perhaps other models yet. Accordingly, it is quite useful to qualify objects as "*analysis objects*" or "*implementation objects*", since these objects are not quite the same. However, this practice is only possible within a given software engineering process, and we have no universal agreement on such a process.

The RM-ODP answers the first problem with a Foundations document [X.902]: it includes a rich set of modelling concepts, and it provides definitions that are both precise and mutually consistent. Importantly, the definitions are applicable to all kinds of object-based models, and they are independent of a notation, software engineering process, or implementation.

The RM-ODP answers the second problem by defining a set of five viewpoint languages (enterprise, information, computational, engineering and technology) that are a sufficient basis for addressing the modelling of large distributed systems [X.903]. Importantly, ODP defines these languages independently of any software engineering process: the semantics of information and computational models, for example, are explained without any direct or indirect reference to an implementation.

2.1 The RM-ODP Foundations

From the very beginning, ISO and ITU experts agreed that object-orientation concepts would be used heavily for specifying and building distributed systems. However, they immediately faced the problem that each of the different communities involved (information analysis, database, programming languages, operating systems) has its own perspective of object-orientation.

To avoid misunderstandings, the RM-ODP Foundations document provides a rigorous definition for each of the concepts commonly encountered in object-oriented models [X.902]. It underlies a *basic object model* which is *unified* in the sense that it has been extended successfully to serve each of the five ODP viewpoints — it is the very invariant that has been applied equally well to enterprise modelling, information modelling, computational modelling, et cetera. Because of its generality of application and its precision, the ODP basic object model captures the essence of what it means to be object-oriented.

Some of the essential characteristics of the RM-ODP Foundations are as follows:

- An object-based *model* is a set of interacting objects that are all active, which means that they all participate in actions.
- *Objects* are the units of encapsulation, characterized by their behaviour and their state. *Encapsulation* means that changes in an object state can only occur as a result of internal actions or interactions.

  Objects have an *identity*, which means that each object is distinct from any other object. Object identity implies that there exists a reliable way to refer to objects in a model[1]. There is no implication that all objects have a unique name in some "well-known" naming context.

- An *action* is a concept for modelling something which happens. ODP actions may have a duration and may overlap in time (allowing an action to model the exchange of a multimedia stream, e.g., a composite television signal).

  All actions are associated with at least one object: *internal actions* are associated with a single object; *interactions* are actions associated with several objects.

  Synchronous interactions provide a rendez-vous mechanism between objects: several objects participate jointly in such an action, and they may change their states simultaneously.
- An *interface* is a subset of the interactions in which an object can participate. In contrast with other object models, an ODP object can have multiple interfaces (the capsule of an ODP object is a set of interfaces). Like objects, interfaces can be instantiated and deleted (some objects have extendable capsules).
- A *template* is a specification from which *instantiations* can be created (this concept is usually called 'class' in the literature on object-orientation). Thus, an object template is a specification for the common features of a set of objects; an interface template is a specification of the common features of a set of interfaces; an action template is a specification of the common features of a set of actions.
- A *type* is a predicate. Objects and interfaces (as well as other concepts such as actions and relations) can be typed with any predicate, but are commonly typed on the basis of the (specification) templates of which they are *instances*[1]. The ODP notion of type is a much more general than that of most object models. For example, many programming languages define the type of an object on the basis of their implementation — two alternate implementations of a class specification have then a different type, which is undesirable. Note also that ODP allows an object to have several types, and to dynamically change types.

  An *object class*, in the ODP meaning, represents the collection of objects that satisfy a given type. Many object models do not clearly distinguish between a specification for an object and the set of objects that fit the specification. ODP makes the distinction between template and class explicit.

  A *subclass* is a subset of a class. A subtype is therefore a predicate that defines a subclass. ODP subtype and subclass hierarchies are thus completely isomorphic.

It is important to note that an action may be atomic or non-atomic at a given level of abstraction (i.e., within a given object model). *Atomic actions* cannot be subdivided into other actions. *Non-atomic actions* are essentially a notational convenience. This paper is essentially concerned about atomic interactions between objects. Therefore, all the actions that we discuss are atomic. We will use the term '*multi-party interaction*' to denote atomic synchronous interactions between two or more objects.

2.2 Compatibility with Classical and Generalized Object Models

As explained by Kilov and Ross, there are two broad categories of object models [KR94]:

- *Generalized object models* do not distinguish a recipient from other request parameters [X3H7]. In generalized models, a request is defined as an event which identifies an operation and optional parameters. For example, an `AddToInventory(part1,lot2,bin3)` request does not give special significance to either the part, the lot, or the bin. These objects participate uniformly in the request.
- *Classical* or *messaging object models* do distinguish a recipient. In classical models, a request is defined as an event which identifies an operation, a recipient, and optional parameters. Either the part, the lot, or the bin could be designed to be the recipient of the `AddToInventory` request. The request to a specific recipient is called a message. A common syntax places the recipient first: `part1.AddToInventory(lot2, bin3)`.

The RM-ODP Foundations are compatible with generalized object models in the sense that multi-party interactions correspond to "*generalized requests*". However they are not completely equivalent: a multi-party interaction just does not occur when one of the objects involved is in a state that is incompatible with it; a generalized request

1. This is analog to the notion of *referential integrity* in CORBA.

1. A predicate, called a *template type*, is associated with an object template or an interface template. This predicate characterises the template instantiations, usually by describing their suitability for some purpose. Objects and interfaces need not be instantiations of a given template to be instances of its associated type.

always occurs, but no behaviour is actually specified in case the preconditions are not satisfied (this behaviour is left for refinement). The generalized object model semantics can be emulated in ODP by specifying several multi-party interactions.

The RM-ODP Foundations are compatible with classical object models in the sense that interactions can be constrained, as does the ODP computational language (see Section 3.1).

2.3 The RM-ODP Viewpoint Languages

A model is a description, or a view, of an existing or of an hypothetical system. Any model of a system is an abstraction of that system, in the sense that it suppresses (or does not specify) irrelevant detail that is of no interest to the users of the model. Since detail is suppressed in a model, there is a need to make several models of a system in order to cover the interests and needs of all the people who have a stake in that system (e.g., buyers, users, managers, implementers).

Considering the current practice and research in distributed computing and system design techniques, the RM-ODP architecture has defined five viewpoint languages, called *enterprise*, *information*, *computational*, *engineering* and *technology*, that allow to express the different kinds of models that are needed to analyse, specify, build and maintain distributed systems. It is important to note that the RM-ODP uses the term '*language*' in its broadest sense: "a set of terms and the rules for the construction of statements from the terms". The RM-ODP does not propose any *notation* for supporting the viewpoint languages[1]— notations are the scope of future ODP standards.

The ODP languages are all object-oriented, and they each specialise the RM-ODP basic object model in various ways. The objects are only defined with respect to the models in which they appear, and no single object may be part of two models in different viewpoints (although strong correspondences are possible). Objects in enterprise models are called *enterprise objects*, objects in information models are called *information objects*, et cetera.

The adoption of a particular viewpoint allows a perception of a system which emphasizes on one particular concern, while ignoring other characteristics that are temporarily irrelevant to that concern:

- *Enterprise models* focus on the *role* of the distributed system within an organisation or community (the system typically appears as a single enterprise object). Enterprise modelling is important for understanding the requirements placed on the system, and for recording the motivation behind a system specification.
- *Information models* focus on the *semantics* of the information that is processed by the system, irrespective of how the system is built. These models are easily comprehensible because information objects, which essentially model the system state, are familiar to the users (they are abstractions of real-world entities).
- *Computational models* focus on the *components* of a distributed system, more specifically on their interfaces and on the interactions that occur at these interfaces. Computational objects are loosely coupled components that can be built independently and that can be distributed over a network. Computational models remain abstract in the sense that resource usage (CPU, threads, memory, communication) is typically not considered at that level.
- Finally, the complex problems linked to *physical distribution* and *resource usage* are only visible in *engineering* and *technology models*, which focus on structure and implementation.

The RM-ODP defines five viewpoint languages, but it does not state that only five models are always sufficient to specify and build a distributed system. There can be an interest of making more that one model within a viewpoint. For example, it can be useful to make a computational model where some service is provided by a single computational object (to show how it will be used), and another model where the same service is provided by a composition of computational objects (to show how it will be distributed).

Today, there is agreement within the ODP community at large both on the interest of using viewpoints and on the specific five projections that have been selected by the RM-ODP. However, there is still debate regarding the precise scope of the viewpoints (see for example the discussion in [ETSI95]). In the second part of this paper, we present our views on the respective scopes of the computational and information viewpoints.

---

1. However, ISO and ITU are working on an amendment to the RM-ODP which will explain how to use existing formal definition notations (LOTOS, Z, SDL and Estelle) for producing models in the ODP viewpoints. This amendment does not imply that the RM-ODP mandates the use of formal description techniques for ODP systems.

3 Scope of Computational and Information Models

The information, computational, and engineering models all describe a system in terms of a set of interacting objects. The differences in those models come from different object composition rules, and from the implementation issues that are addressed by those models. For example, the use of processing resources such as CPU usage, memory, and communication (and therefore object location) must be addressed in engineering models, whereas it needs not be in computational models.

An engineering model describes therefore a same computation than a computational model, but provides more information with respect to the actual implementation of the system. In that sense, the engineering model of a system is a refinement of its computational model. This idea is probably well understood and well accepted.

Similarly, we propose that computational models are refinements of information models, but only in the sense of behavioural compatibility — we do not imply that computational models must be derived from information models, nor that they must be expressed in a similar notation.

We also propose that the main difference between computational models and information models are that (atomic and synchronous) multi-party interactions are allowed only in information models.

3.1 Computational Models

The ODP computational language defines rules that computational models must follow. A fundamental idea is that computational models identify loosely coupled components and their interactions, such that an implementation with an "engineering virtual machine" (e.g. CORBA) is possible without much transformations. For this very reason, the computational language constrains the interactions in which objects may participate to signals, operations, and flows:

- *Signals* are synchronous interactions that involve at most two objects and that allow only one-way communication. As all synchronous interactions, they are always perceived identically by both participants in the interaction. Hence, there is no concept of partial failure of a signal.

  Although the RM-ODP is not explicit on this point, objects cannot in general communicate directly using signals. What objects can do is to instantiate a so-called *binding object* between them (a binding object is typically an abstraction of a communications protocol) — signals can then be exchanged with the binding object, which propagates them with a delay to the other objects.

- *Operations* are interactions analog to requests in CORBA, or to messages in object-oriented languages. They involve at most two objects, and they are classified in two types: *announcements* are operations for which no outcome is reported to the invoker; *interrogations* are operations for which an outcome (*termination*) is always reported to the invoker. This reported outcome may be the real termination of the operation, but it can also be an exception reporting an engineering infrastructure failure (e.g., a communication failure).

  Operations are *asynchronous*[1] in the sense that the invocation and termination might be delivered some time later (if at all) after they have been submitted. In fact, operations might sometimes be synchronous, but the assumption to be made is that they are asynchronous.

- *Flows* are interactions modelling the conveyance of information from a producer object to a consumer object. They are typically used to model continuous interactions including the special case of an analogue information flows. As for operations, the assumption to be made is that flows are asynchronous.

Interactions in computational models are constrained because implementing synchronous interactions (including signals) in a distributed way is inefficient or excessively difficult — allowing arbitrary interactions would defeat the very purpose of the computational viewpoint. Restraining interactions is also a way to preserve a clear separation between computational modelling and information modelling, as discussed in Section 3.2.2.

In fact, some *ODP transparencies*, such as the *transaction transparency*, can relax the constraints imposed by the computational language in specific and controlled ways. We do not consider the impact of the ODP transparen-

1. By '*asynchronous*', we only mean that the delay between submission and delivery is variable. An operation invocation may be either *blocking* (the thread waits for the termination) or *non-blocking* (the thread is allowed to execute other actions before attempting to receive the termination). This is an implementation issue.

cies in this paper.

3.2 Information Models

The RM-ODP is excessively terse regarding the information language.

In particular, it fails to convey the intent of information modelling: the statement that “an information specification defines the semantics of information and the semantics of information processing in an ODP system” can be interpreted in different ways.

A useful interpretation can be obtained by noting that ODP defines *information* as “the knowledge that is exchangeable amongst users in a given universe of discourse.” In this context, information may be seen as “the knowledge necessary to make use of a system or part of the system” [CC95]. An information model specifies thus *what a user must know about a system, to make a proper use of it*. Note that a user of a system may be a person or another system — in the latter case, the users of the information model are the people who specify, design or implement that other system.

Simplicity and abstraction (independence of implementation) are major concerns of information modelling, whilst executability of models is not. Users need not be aware of all the components of a system, of all the possible states of these components, nor of all the interactions between those components. They are only interested in the system states that they can perceive. Since these states are necessarily a subset of the effective system states, information models can and should be simpler than computational models.

3.2.1 Computational Models Used as Information Models

A computational model may be considered and may even be used as an information model. Such a model is probably not the simplest nor the most abstract information model that can be made, but it represents a valid specification of a system, and users may get enough knowledge from it to use the system.

Indeed, it sometimes makes sense to use a computational model as an information model. For example, if there are just two components to consider, revealing the system components does not dramatically increase the complexity of the model. Moreover, revealing components in an information model can be useful when they tend to enter in well-known failure modes.

3.2.2 The Information Language Specification Technique

The RM-ODP information language proposes a specification technique analog to Z, [Fusion], and [Catalysis]. It does this with only a few rules and definitions, saying little more than the following:

- *Static schemas* capture information structure (global state) at some point in time.
- *Dynamic schemas* specify legible state changes in multiple information objects, using preconditions and postconditions.
- *Invariant schemas* constrain the possible states and state changes of the objects to which they apply.
- A global state transition may involve several information objects.

The above rules raise two important questions:

1. How should information models be expressed in notations that do provide neither dynamic nor invariant schemas (e.g. LOTOS)?
2. How can dynamic schemas and invariant schemas specify legible state changes in several objects simultaneously, when encapsulation is an essential property of objects (as noted in the ODP Foundations)?

The answer to both questions is that “multi-party interactions are allowed in information models.” In fact, this answer underlies a fundamental difference between information and computational modelling.

Objects cannot change their states simultaneously when they interact only through operations, flows, or through signals and binding objects. The reason is that operations and flows are not perceived by objects at the same time, and that binding objects introduce a delay in communication. However, objects can change their states simultaneously by jointly participating in a multi-party interaction (as in LOTOS) — multi-party interactions are indeed atomic and synchronous.

Multi-party interactions are much more powerful interactions than signals and operations. Firstly, a single multi-party interaction can consistently change the states of several objects, which is often what is desired. And sec-

ondly, there is much less uncertainty with multi-party interactions than with operations (or with signals and binding objects): a multi-party interaction occurs and produces the right effects, or it just does not occur.

By themselves, multi-party interactions allow information models to be more abstract and simpler to understand than computational models. However, the information language allows for an even greater level of abstraction by not requiring interactions to be listed explicitly: an information object may participate in all interactions that one can think of, except those that conflict with a dynamic or an invariant schema. In other words, information modelling typically proceeds by excluding invalid interactions (adding constraints), rather than by listing valid interactions.

3.2.3 Encapsulation of Information Objects

Information objects are not simply "lists of attributes". They do have a behaviour, which may be specified using static schemas, dynamic schemas, and invariant schemas. Because an information object is encapsulated, the behavioural constraints in those schemas cannot be overridden nor violated.

An interesting dynamic schema to be specified for an information object is that of an *event* — other information objects may participate in that event, i.e., be told when it occurs and learn its parameters. Unlike computational objects, information objects can observe events exactly when they occur.

3.2.4 Explicit Interactions in Information Models

Whereas interactions between information objects need not be specified, it is not unreasonable to specify some interactions explicitly. In particular, it can be appropriate to specify the interactions of the system and its environment (with specific information objects, that we call *interface information objects*). Otherwise, there are simply too many potential interactions and state changes to consider.

Current practice within ITU is to specify the system interactions within the computational model only [G851]. In fact, the system environment is not specified, and the information model is therefore incomplete (how can one make use of a system, without knowing any of its interfaces?). This "information model" is therefore only usable in conjunction with a computational model. It would be more correct to state that the information model includes *interface information objects*, which are specified (by reference) in the computational model.

An object template may specify one or several interfaces, and still be an interesting information object template. There is nothing wrong with emphasizing some interactions, provided other interesting and valid interactions are not excluded.

3.2.5 Multiple Information Views of a System

The RM-ODP defines five viewpoint languages, but it does not imply that systems are specified with just five models. Since a distributed system has typically different kinds of users, it is interesting to make several information models, or *information views*, of that system (see [Catalysis] for a good example of using multiple information views in object modelling).

Within information views, some interactions between the system and its environment may be abstracted using schemas. In that case, invariant or dynamic schemas may refer to information objects within the system, as well as to information objects outside of the system (see [EPFL1588], section 4.5.1).

3.2.6 Reuse of Information Object Templates

The information language specification technique is not only very expressive. It also makes information object templates (units of specification) easily reusable.

For example, consider a multi-party interaction that is shared by a clerk object, an inventory object and a customer shipment object; the occurrence of this interaction models that a clerk decides to ship some goods to a customer, that these goods are withdrawn from the inventory, and that they are added to the shipment. This multi-party interaction occurs only if the three objects are in a state which allows it to happen.

Now, suppose that new requirements indicate that a shipment implies an immediate payment. The model can accommodate this change very easily by having two more objects participate in a revised multi-party interaction: a customer account object and a vendor account object. Constraints are added in the dynamic and invariant schemas, but the templates of the clerk object, the inventory object and the customer shipment object remain unchanged.

The above example illustrates the interest of information modelling for defining reusable common business objects, or more specifically, common information business object templates.

4 Conclusion

Object modelling, according to the RM-ODP viewpoint languages, can help develop business domain architectures and reusable object specifications. Indeed, information models are a powerful technique for specifying the systems or components of a business architecture, and information object templates are easily reusable in different specification contexts. However, the RM-ODP is difficult to understand, and there appears to be several divergent interpretations of it. This paper proposes an interpretation of information modelling that is both fully consistent with the ODP foundations, and compatible with some of the best practice in object-oriented software engineering.

This paper has presented an interpretation of ODP information modelling that is compatible with some of the best practice in software engineering. More specifically, we see ODP information modelling as being analog to the analysis phase in recent object-oriented methods (e.g., [Catalysis], and [Fusion]). This analogy should considerably help in promoting and explaining the RM-ODP. The main benefits of ODP with respect to object-oriented methods are that it is an international standard, that it is based on stronger “object foundations”, and that its “high-level object models” are more implementation independent. The RM-ODP focuses on modelling, alleviating the need to agree on a specific aspects of software engineering.

5 Acknowledgements

Jean-Bernard Stefani gave me some useful hints to understand the RM-ODP information model. My position and arguments in this paper were influenced and improved by email discussions with Haim Kilov, Elie Najm, Richard Sinnott, and Marcos Rogerio Salvador. Haim Kilov convinced me to write this paper, and provided useful advice and comments.

6 Bibliography

[Catalysis] D. D'Souza and A. Wills, “Catalysis — Practical Rigor and Refinement: Extending OMT, Fusion, and Objectory,” available as http://www.iconcomp.com/papers/catalysis/catalysis.frm.html, 1995.

[CC95] H. Christensen and E. Colban, “Information Modelling Concepts,” Technical Report, Telecommunications Information Networking Architecture Consortium (TINA-C), April 1995.

[CORBA-2] OMG, “The Common Object Request Broker: Archictecture and Specification (2.0),” Object Management Group. 1995.

[EPFL1588] G. Genilloud, “Towards a Distributed Architecture for Systems Management — PhD. Thesis 1588,” Computer Science. Swiss Federal Institute of Technology of Lausanne (EPFL). 1996.

[ETSI95] ETSI, "Transmission and Multiplexing (TM) -- The Application of ODP to the Management of a Transport Network (Draft)," Work Item No: DTR/TM-2221. 1995.

[Fusion] D. Coleman, Patrick Arnold, S. Bodoff, C. Dollin, et al., Object-Oriented Development: The Fusion Method, Prentice Hall, 1994.

[G851] ITU, “Application of the RM–ODP Framework to the Management of the Transmission Network,” Draft Recommendation G.851–01. Q30/15. 1995.

[KR94] H. Kilov and J. Ross, Information modeling : an object-oriented approach, Prentice Hall, 1994.

[X3H7] ANSI, "The X3H7 Object Model Features Matrix," Technical Report X3H7-93-007v10, available as http://info.gte.com/ftp/doc/activities/x3h7/by_model/OODBTG.html, February 14, 1995.

[X.902] ISO/IEC and ITU-T, “Open Distributed Processing - Basic Reference Model - Part 2: Foundations,” Standard 10746-2, Recommendation X.902. 1995.

[X.903] ISO/IEC and ITU-T, “Open Distributed Processing - Basic Reference Model - Part 3: Architecture,” Standard 10746-3, Recommendation X.903. 1995.