7
Consistent Semantics for ODP Information and Computational Models

Joubine Dustzadeh, Elie Najm
Ecole Nationale Supérieure des Télécommunications
46, rue Barrault, 75634 PARIS CEDEX 13, France
{joubine, najm}@email.enst.fr

Abstract

We tackle two important ODP viewpoints, namely the information and the computational. We first provide formal semantics for object diagrams of some popular application development methodologies (such as OMT and Fusion) and show how these notations support ODP information modeling. We also formalize an essential part of the ODP computational viewpoint including the concepts of distributed objects and concurrent serializable activities. We then complement the two semantics by providing rules for consistent mappings between the two models.

Keywords

ODP, viewpoint consistency, information modeling, OMT, actions, computational modeling, serializability

1 INTRODUCTION

Distributed systems are inherently complex and their description in one single large specification is not desirable in practice. To deal with this complexity, system specifications need to be decomposed through a process of separation of concerns, and based on different levels of abstraction. Specification decomposition is a recurrent concept that is found in many architectures for distributed systems. For instance, the reference model of Open Distributed Processing – ODP considers five viewpoints: *enterprise*, *information*, *compu-*

Formal Description Techniques and Protocol Specification, Testing and Verification
T. Mizuno, N. Shiratori, T. Higashino & A. Togashi (Eds.)

tational, *engineering* and *technology*. Specifications of systems in ODP can be made from any of these viewpoints and different – formal or informal – specification languages may be used.

The ODP information viewpoint provides a highly abstract model of real world entities and their relationships along with the constraints that govern their behavior. ODP does not prescribe specific languages to be used for information modeling partly due to the high level of abstraction required to capture information structures and due to the possible reflective nature of information. Informal object models (such as OMT [2]) are largely used for information modeling. In addition to the problems related to use of informal specification languages, the correspondence between specifications from different viewpoints (and particularly between the information and computational viewpoints) poses a considerable challenge in ODP. To better address these issues, the semantics of information models merits particular attention in ODP. The semantics of information models in ODP includes: (i) the semantics of the information specification (when considered to be isolated from other viewpoints) and (ii) the examination of how the information specification relates to the computational structure of the system.

How this Paper is Organized

Section 2 discusses various notations used for information modeling in ODP. Section 3 introduces the object model of OMT and provides a formal semantics for it. Section 4 introduces and formalizes the concept of information actions. Section 5 introduces a model of the ODP computational viewpoint. Section 6 provides a framework for valid mappings between the information and the computational models. Finally, some concluding remarks are given in section 7.

2 NOTATIONS FOR INFORMATION MODELING IN ODP

There exist several notations and methods that are candidates for information modeling within ODP. Three important requirements should be considered carefully: (i) the ability to directly capture the information model, (ii) abstractness, and (iii) friendly graphical user interface. An ideal notation for the ODP information model should support the principles of object-orientation and should also offer an easy-to-use interface to the service specifier for the task of mapping real world entities into information objects. Yet, this notation should have a formally defined semantics for the sake of analyzability, tool support, and compatibility between the information specification and the specifications made in other viewpoints. Information structures and relationships can be naturally captured by a user friendly graphical notation.

Graphical interfaces are also well suited to support incremental specification. As an information specification progresses, one can modify and refine objects and relations in an iterative fashion.

Z is being considered as highly appropriate for information modeling. Although Z is not fully object oriented, it has those object features that match the needs of information modeling. Static, invariant and dynamic schemas can be directly mirrored in a Z specification. Moreover, Z provides a good basis for relating specifications in other viewpoints or languages [10]. There is a large community of Z users and researchers. The object model of some object-oriented analysis methods, like e.g. OMT or Fusion [3], are also well suited for ODP information modeling. Although they are not really formal, they have been used because of their appealing graphical syntax and because they are part of fully integrated development methodologies.

3 FORMAL OBJECT-ORIENTED INFORMATION MODELING

Various surveys indicate that the *object model* of OMT is the most common object model that is practiced for the purpose of information structuring in ODP. However, as mentioned earlier, the object model of OMT lacks a formal semantics. The example illustrated in figure 1 shows how an OMT information specification may lead to ambiguity and therefore to multiple interpretations. The dangers of such an ambiguous information specification include:

- several possible (and semantically different) implementations of the same specification.
- impossible realization (implementation) of the specification (figure 1).

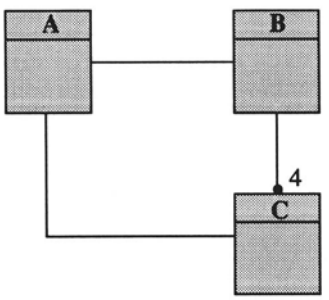

Figure 1 Example of an Incorrect Object Diagram in OMT

This OMT diagram uses some basic OMT concepts such as objects and binary associations (and their cardinality constraints). It defines three classes

A, B, C and three associations between these classes.

We have run a survey in the relevant mailing list* and news group* on the Internet and requested that OMT experts examine the semantical correctness of the diagram illustrated in figure 1. The post included the following questions:

- for each instance a1, is there exactly one instance b1?
  Let us represent it as following: <a1,b1>
- for b1, are there exactly four instances c1,...,c4?
  <b1,c1> <b1,c2> <b1,c3> <b1,c4>
- What about a1 and c-i? (since there is a one-to-one association between A and C)?

Surprisingly enough, this simple diagram resulted in several different interpretations and a long debate was initiated on the semantical correctness of OMT diagrams in general.

3.1 Formalization of OMT Object Diagrams

We choose the *object model* of OMT as a representative of all object models that are appropriate for information modeling in ODP. Despite this choice, the formal rules outlined in our approach remain applicable to other object models with minor modifications. The challenge of formalizing the OMT object model is twofold: definition of an abstract syntax of the language and definition of the semantical function for the elements of the language. Thus, our formalization method consists of the following steps:

Definition of a Formal Abstract Syntax : we first need to provide an abstract linear syntax that captures exactly the permissible OMT diagrams. This syntax should exist independently of any graphical OMT editor. It will be defined by a combination of BNF productions and a set of static semantics rules. We introduce the domain $\mathcal{U\_CRS}$ of *Unflattened Class Relation Systems* consisting of *unflattened classes** (translated from OMT classes) and *relations* (reflecting OMT basic associations and aggregation relations). Let $\mathcal{D}$ represent all OMT diagrams and $\mathcal{U\_CRS}$ represent the set of unflattened class relation systems. The linearization function $\mathcal{L}$ can be represented as following:

$$\mathcal{D} \xrightarrow{\mathcal{L}} \mathcal{U\_CRS}$$

*otug@rational.com
*comp.object
**unflattened* means that classes may contain inheritance links to other classes

Flattening Process : in order to completely linearize the OMT diagrams translated into unflattened class relation systems, we use a flattening function $\mathcal{F}$ to remove all inheritance links from unflattened classes. This process leads to a new domain $\mathcal{CRS}$ whose elements are called *Class Relation Systems.* The flattening function can be represented as following:

$$\mathcal{U\_CRS} \xrightarrow{\mathcal{F}} \mathcal{CRS}$$

The flattening process also allows the definition of some static semantics rules for unflattened class relation systems (for instance, inheritance may not occur recursively).

Instance Systems : at this stage, we define the semantical domain $\mathcal{IS}$ for class relation systems. Elements of this domain are called *Instance Systems.* A generic element of $\mathcal{IS}$ is composed of instances of objects and instances of associations. Each instance system represents a possible state of the system that is specified by the OMT diagram. The semantics of an OMT diagram is defined in terms of the set of allowable configurations of instance systems.

3.2 Definition of a Formal Abstract Syntax

Our first concern here is to give an abstract syntax to the OMT notation. Specification of a syntax for a model implies the definition of an alphabet and the permissible combinations of its symbols. Unflattened class relation systems are defined by a BNF grammar rule as described below. U_CLASS is intended to be the linear representation of a class in an OMT object diagram, and RELATION a linear representation of both OMT associations and aggregations. U_CLASS and RELATION are defined below:

- U_CRS ::= U_CLASS | U_CRS U_CLASS | U_CRS RELATION
- U_CLASS ::= $\langle name : \aleph, parset = \pi, abstract : \mathbb{B}, attlist = \eta \rangle$
- RELATION ::= $\langle class_1 : \aleph, role_1 : \tilde{\aleph}, card_1 : \mathbb{F}\,\mathbb{N}, name : \tilde{\aleph}, card_2 : \mathbb{F}\,\mathbb{N}, role_2 : \tilde{\aleph}, class_2 : \aleph, attlist = \eta \rangle$

Notations and Conventions

- since we are not concerned here with syntactic constraints of different names and types, we introduce the given set $\aleph$ whose internal structure is transparent in our model. All the permissible names by OMT for classes, attributes, operations, operation arguments, associations and roles are drawn from this set. However, since in OMT role names and relation names may be omitted, we introduce the following notation to reflect absence of role names or association names.

$$\tilde{\aleph} ::= \aleph \mid \varepsilon \mid \kappa$$

where ε is a special name to represent absence of a name and κ indicates aggregation. Thus, for an aggregation relation, $r.role_1 = \kappa$ and $r.name = \kappa$.

- *attlist* $= \eta$ represents a list of attributes tagged by their name and their type. $\eta ::= \langle x_1 : t_1, \ldots, x_n : t_n \rangle$.
- *name* is the class name (resp. relation name) in CLASS (resp. in RELATION).
- *parset* is the set of names of the immediate parents (superclasses). If a class does not have any superclasses, *parset* will be an empty set. For simple inheritance, *parset* contains only one element (the name of the superclass) and for multiple inheritance it is composed of as many names as there are immediate parents. We will restrict our study to simple inheritance.
- *abstract* is a boolean. it holds the value true for an abstract class and false otherwise.
- $class_1$ and $class_2$ are the names of the classes that are related to each other via a relation.
- $role_1$ and $role_2$ are the role names in an association. For an aggregation, the corresponding role name (next to the aggregation diamond) is set to κ. If there is no role name specified in an association, $role_i$ will be ε.
- $card_1$ and $card_2$ are the multiplicity constraints defined for an association and they take their values in $\mathbb{F}\,\mathbb{N}^*$. For a one-to-one association, both $card_1$ and $card_2$ are the set $\{1\}$. In the case of a multiplicity *many* placed on the $class_i$ side of an association, if no set is numerically specified, $card_i$ is $\mathbb{N}$.

Well-formedness of Unflattened Class Relation Systems

Now that we have defined a formal syntax for unflattened class relation systems, we can provide formal rules to ensure their well-formedness. The BNF syntax allows construction of elements that may not correspond to any OMT diagrams. Rules of well-formedness should be defined to eliminate inconsistent unflattened class relation systems. These rules of well-formedness are also needed for a correct construction of OMT object diagrams. For example, two attributes within the same class may not have the same name, or two classes may not have the same name.

Definition 1 *An unflattened class relation system ucrs is well-formed iff it satisfies rules UCRS.WF defined below*[*].

- UCRS.WF.1 : a binary association can exist only between classes already defined.
- UCRS.WF.2 : the class name should be unique.
- UCRS.WF.3 : the attribute names should be unique within a class.
- UCRS.WF.4 : roles are mandatory for associations between two objects of the same class.

---

[*] $\mathbb{F}$ indicates a finite set. Thus, for instance, $\mathbb{F}\,S$ is the type whose elements are finite sets of elements of type S.

[*] A formal representation of these rules is given in the appendix (figure 5).

- UCRS.WF.5 : aggregation is anti-symmetric (it implies non-symmetry between the roles).
- UCRS.WF.6 : no role name should be the same as an attribute name of the source class.
- UCRS.WF.7 : all role names on the far end of associations attached to a class must be unique.
- UCRS.WF.8 : inheritance must not be cyclic.
- UCRS.WF.9 : only concrete classes may be a leaf class in an inheritance tree.

3.3 Flattening Process

Once unflattened class relation systems have been checked against the above well-formedness rules, a flattening process $\mathcal{F}$ may begin in order to remove inheritance links and abstract classes. In the appendix, we propose an algorithm describing $\mathcal{F}$. Note that the flattening process maybe achieved based on different parsing algorithms used for trees and forests in graph theory.

The flattening process leads to a new domain $\mathcal{CRS}$ whose elements are called *Class Relation Systems.* The BNF representation of $\mathcal{CRS}$ is the following:

- CRS ::= CLASS | CRS CLASS | CRS RELATION
- CLASS ::= $\langle name : \aleph, attlist = \eta \rangle$

3.4 Instance Systems

In our model, an instance system is viewed as a collection of allowable interrelated object instances. A valid instance system is one where all the constraints and rules expressed in the object model are reflected in the object instances and in the dependencies between object instances. Multiplicity constraints in associations between object instances are examples of such consistencies.

Syntax of Instance Systems

At a given point in time, an instance system can be considered as a collection of record structures composed of values (attributes) of given types. The effect of operations (defined in OMT classes) is reflected by an evolution of the state space of an instance system, that is, a transformation of values in the record structures and of the collection of records (e.g. create and delete records). Instance systems are defined by the following BNF grammar rule:

- InstanceSystem ::= information_element | information_element InstanceSystem
- information_element ::= object_instance | association_instance
- object_instance ::= $\langle id : \delta, name : \aleph, attset = \beta \rangle$
- association_instance ::= $\langle id : \delta, name : \aleph, id_1 : \delta, role_1 : \tilde{\aleph}, id_2 : \delta, role_2 : \tilde{\aleph}, attset = \beta \rangle$

Where $attset = \beta$ represents a list of attributes tagged by their name and their value. $\beta ::= \langle x_1 = v_1, \ldots, x_n = v_n \rangle$.

Well-formedness of Instance Systems

Similarly to a class relation system, an instance system must satisfy a number of well-formedness and static semantics rules. A non-exhaustive list of these rules are given below (a formal representation of these rules may be found in the appendix, figure 6):

- IS.WF.1 : instances must have a unique identity.
- IS.WF.1 : an association between two object instances requires that both object instances exist.
- IS.WF.3 : attribute names should be unique within an instance.
- IS.WF.4 : aggregation is anti-symmetric.

Semantics of Class Relation Systems

Now that we have given the well-formedness rules for instance systems, we can define the semantics of a class relation system in terms of the set of allowable configurations of instance systems.

Notations

Let *crs* be a well-formed class relation system, c a class in *crs* and r a relation in *crs*. Let *is* be a well-formed instance system, ω an object instance in *is* and ρ an association instance in *is*. For $i \in \{1, 2\}$, we introduce the following notations to facilitate the expression of the semantics rules for class relation systems.

Meaning

- $crs.r.i$ denotes the unique class referenced by $r.class_i$
- $is.\rho.i$ denotes the unique object referenced by $\rho.id_i$
- $\omega : c$ means that object instance ω is of class c
- $\rho : r$ means that association instance ρ is of class r
- $is.r.i.id$ denotes the set of object instances related to ω (with identifier id) via associations of type r

Notation		Formal Definition
$crs.r.i$	$=$	$\{c \in crs \mid c.name = r.class_i\}$
$is.\rho.i$	$=$	$\{\omega \in is \mid \omega.id = \rho.id_i\}$
$\omega : c$	$\Leftrightarrow$	$((\omega.name = c.name) \wedge \forall j :$ $(\omega.attset_j.name = c.attset_j.name) \wedge$ $(\omega.attset_j.type = c.attset_j.type))$
$\rho : r$	$\Leftrightarrow$	$((\rho.name = r.name) \wedge \forall j :$ $(\rho.attset_j.name = r.attset_j.name) \wedge$ $(\rho.attset_j.type = r.attset_j.type))$
$is.r.i.id$	$=$	$\{\omega \in is \mid \exists (\rho : r) \in is$ such that: $(\rho.id_i = id) \wedge (\rho.id_{\bar{i}} = \omega.id)\}$

- $\bar{i} = 2$ if $i = 1$ and $\bar{i} = 1$ if $i = 2$.

Definition 2 *We define valid correspondence between class relation systems and instance systems by relation $\mathcal{VC}$ such that $\mathcal{VC} \subset \mathcal{CRS} \times \mathcal{IS}$ and $(crs\ \mathcal{VC}\ is)$ iff the rules VC.1, VC.2, VC.3 hold:*

(VC.1): Any object instance in *is* is an instance of a class in *crs*.
(VC.2): Any association instance in *is* is an instance of a relation in *crs*.
(VC.3): The multiplicity constraints defined for any relation r in *crs* are respected in *is*.

These rules are formally expressed in figure 2.

(VC.1):	$\forall \omega \in is,\ \exists c \in crs \mid \omega : c$
(VC.2):	$\forall \rho \in is, \exists r \in crs \mid (is.\rho.i : crs.r.i))$
(VC.3):	$\forall r \in crs, \forall id,\ card\{is.r.i.id\} \in r.card_{\bar{i}}$

Figure 2 Semantics Rules for Instance Systems

Definition 3 *Semantics of class relation systems is given by the following rule:*

$$[\![crs]\!] \widehat{=} \{is \mid crs\ \mathcal{VC}\ is\}$$

where $\mathcal{VC}$ is a valid correspondence between a class relation system (translated from an OMT diagram) and an instance system.

Definition 4 *A sufficient condition* for the equivalence between two class relation systems crs_1 and crs_2 is:*

$$[\![crs_1]\!] = [\![crs_2]\!]$$

Now let us examine again the example of figure 1. In our semantics, the diagram of figure 1 denotes either an empty set of instances or a set of an infinity of instances of classes `A`, `B` and `C`. Thus, this diagram denotes only degenerated cases and should be discarded.

4 ACTIONS IN INFORMATION MODELING

In many realistic case studies, one may have to group a certain number of objects that participate simultaneously in the same action. While most object models of popular object-oriented methodologies such as OMT and Booch offer powerful constructs to staticly structure information entities, they fall short in expressing collective behaviors for groups of objects. As an example, the OMT object model which is one of the most commonly used specification languages for the information viewpoint does not support the concept of *multi-object actions.*

While one of the most commonly recognized advantages of the object approach is to improve the modularity and incrementability of a specification, and to allow encapsulation of data and individual behavior of objects, traditional object-oriented specification tools mostly focus on the specification of individual behavior of objects and usually neglect collective behaviors of objects. This approach could be accepted at a more program-oriented abstraction level, that is, at a stage of specification where computational objects and their interfaces have already been identified. In contrast, a multi-object action approach allows for a more expressive specification of collective behaviors in the sense that decisions on how the participating objects in an action make joint decisions on whether to commit in joint actions, elementary computations or exchange messages to coordinate them can be left to later stages of design [4]. For this reason, we choose a multi-object action approach, à la Fusion, for modeling dynamic schemas in the ODP information viewpoint.

*Class relation systems can be further equated whereby attributes and relations are interchangeable.

4.1 Semantics of Information Actions

Let us consider a class relation system *crs* (resulting from the flattening of some OMT diagram). We extend the information model of this *crs* with the following elements:

- *Inv* represents an invariant of the system. When an instance system *is* satisfies the invariants, we have: $Inv(is) = \text{true}$.
- $[\![crs]\!]_{Inv}$ denotes a restriction to $[\![crs]\!]$ such that: $\forall\, is \in [\![crs]\!]_{Inv} \Rightarrow Inv(is) = \text{true}$
- A denotes an information action that is defined on information objects. An action A embodies a pre-condition predicate A_{Pre}, a body part A_B and a post-condition predicate A_{Post}.

$$A = (A_{Pre}, A_B, A_{Post})$$

where: $A_B : \mathcal{IS} \longrightarrow \mathcal{IS}$, $A_{Pre} : \mathcal{IS} \longrightarrow \mathbb{B}$, $A_{Post} : \mathcal{IS} \longrightarrow \mathbb{B}$

Note that in this semantics, information actions are considered as atomic. In other words, the body part A_B of an action A operates on the whole instance system without being interrupted by or interleaved with other actions. Moreover, A_B may be considered as an imperative language whose syntax need not be explicitly given in our study.

Definition 5 (Valid Information Actions) *An information action A is valid iff:*

$$\forall\, is \in [\![crs]\!]_{Inv}, A_{Pre}(is) = true \;\Rightarrow$$
$$A_{Post}(A_B(is)) = true \;\wedge\; Inv(A_B(is)) = true$$

5 THE COMPUTATIONAL MODEL

As seen earlier, RM-ODP allows for a multi-viewpoint specification of distributed systems. In the current state of the RM-ODP, the least natural transition seems to lie in the passage from the information model to the computational model.

An action (considered at the information viewpoint level) corresponds to the execution of a computational activity which can involve several computational objects. Also, because of the multi-thread property of computational objects, several activities (relative to different actions) may run in the same computational object in parallel. The global state of the system changes due to the execution of computational activities. One can then observe, by taking *snapshots* at given moments in time, the changes of the global state of the

system due to executions of computational activities. These snapshots – taken at *relevant* moments – will be recorded in *object logs* and will contain enough information to allow checking for the consistency of the global state of the system against the invariant schemas defined at the information viewpoint level.

Actions are specified in an atomic way in the information model. Designers need not worry about concurrency control at this stage. However, the corresponding computational activities could run in an interleaved fashion within computational objects. In other words, computational activities may have to access and share the system resources in a concurrent way. More specifically, variables and attributes of objects may be accessed for *read* or *write* operations concurrently. Also, during the execution of an action, new objects may be created. That leaves us with a concern for the consistency checking of the global state of the system at the end of execution of a given action. A consistent global state is one where the cardinality constraints, the invariants and the post-conditions remain satisfied.

We consider a computational model that is a subset of the computational model defined by RM-ODP. We consider only implicit bindings plus some new concepts such as object logs and activities. By introducing these new concepts, we hope to bring some clarity into the passage from the information model and to make a contribution in extending the RM-ODP computational model. A description of the computational terms that we consider in our model is given in the following:

Computational Elements : a computational element can be either an object or a message.

Objects : an object has multiple interfaces. Objects and interfaces within objects can be created and deleted dynamically.

Messages : messages can be interrogation requests, announcement requests and interrogation replies. A generic type of message is proposed to represent all kinds of messages. Each message is sent on the account of an activity and contains the identifier of this activity. Messages can be created and deleted dynamically.

Object Logs : in order to ensure that current executions of actions do not violate the consistency of the system, we build execution logs for activities. To each object, an execution log is associated and the log gets updated: (i) every time an attribute of the object is updated (write) and (ii) every time an attribute of the object is read. The snapshots recorded in the object logs must always satisfy the following rules:

- a computational activity may not run while the preconditions of the corresponding action are not satisfied.

- after the execution of a computational activity, the postconditions of the corresponding action as well as all invariants schemas defined at the information viewpoint must be verified.

5.1 Syntax of Computational Terms

A distributed computation Ω is a collection of *distributed elements* which may evolve in parallel. The parallel operator is denoted as $||$. A distributed element can be either an object or a message μ. This abstraction does capture the understanding that a distributed program, from the point of view of the computational model, is merely a set of asynchronously interacting objects. Figure 3 defines a formal abstract syntax for computational terms.

$$
\begin{array}{lcl}
\Omega & ::= & \theta \mid \mu \mid \Omega \,||\, \Omega \\
\theta & ::= & \langle id : \delta, attset = \beta, class = c, actset = \tau, loglist = \lambda \rangle \\
\mu & ::= & \langle tgt = u, src = v, act = \alpha, cont = m \rangle
\end{array}
$$

Figure 3 Syntax of Computational Terms

In figure 3, the following conventions apply:

- θ is a computational object.
- $\theta.id$ is the unique identifier of a computational object. δ is a generic type for all identifiers.
- $\theta.attset = \beta$ is the set of all attributes within an object. β takes the form of a list of attributes tagged by their name and their value: $\beta ::= \langle x_1 = v_1, \ldots, x_n = v_n \rangle$.
- $\theta.class = c$ represents the corresponding class name in the information viewpoint. c takes its value in $\aleph$.
- $\theta.actset = \tau$ is the set of all activities that are present in an object. τ takes the form of a list of activity identifiers: $\tau ::= \langle \alpha_1 \rangle \cdots \langle \alpha_n \rangle$.
- $\theta.loglist = \lambda$ is the list of all log entries that have been recorded for an object. λ takes the form of a list $\lambda ::= \langle l_1 \diamond l_2 \diamond \cdots \diamond l_n \rangle$ of log entries l_k where :

$$
\diamond ::= \begin{cases} \langle\rangle \diamond l = \langle l \rangle \\ \langle l_1 \diamond \cdots \diamond l_k \rangle \diamond l = \langle l_1 \diamond \cdots \diamond l_k \diamond l \rangle \end{cases}
$$

Each entry l_k (interchangeably denoted $[\lambda]_k$) contains an activity identifier

α_j, the type of the operation performed by the activity (read or write), and an attribute x_i whose value v_i has been read or updated by α_j. In case of a read operation, we use the notation $l_k = \alpha_j^r(x_i = v_i)$ where v_i holds the value read by α_j, and in case of a write operation, we use the notation $l_k = \alpha_j^w(x_i = v_i)$ where v_i holds the new value written by α_j. When there is no confusion, v_i may be omitted.

- $\mu.tgt$ is the target interface of message μ. In case of an interrogation request (or an announcement) $\mu.tgt$ holds a value u which is the interface reference of the target interface.
- $\mu.src$ is the source interface of message μ. In case of an interrogation response, $\mu.src$ holds a value v which is the reference of an interface previously invoked.
- $\mu.cont = m$ denotes the remaining content of the messages. This part is not relevant in our study (see [5]).

5.2 Serializability of Computational Activities

Definition 6 (Serializability) *Let Ψ be a set of computational activities. An interleaved execution ξ of Ψ is said to be serializable if and only if ξ generates the same result as some serial (sequential) execution of Ψ.*

Definition 7 (Causal Precedence) *For a given object θ, we define a causal precedence between two activities α_i and α_j – denoted $\alpha_i \overset{\theta}{\rightsquigarrow} \alpha_j$ – if there exists an attribute x of θ such that :*

- *x is read by α_i before x is updated (written) by α_j, OR*
- *x is updated by α_i before x is read by α_j, OR*
- *x is updated by α_i before x is updated by α_j.*

This definition can be formalized as following:

$$(\alpha_i \overset{\theta}{\rightsquigarrow} \alpha_j) \iff \exists x \mid \alpha_i^r(x) \overset{\theta}{\rightsquigarrow} \alpha_j^w(x) \vee \alpha_i^w(x) \overset{\theta}{\rightsquigarrow} \alpha_j^r(x) \vee \alpha_i^w(x) \overset{\theta}{\rightsquigarrow} \alpha_j^w(x)$$

Object Logs can serve to track precedence relations between activities:

$$\begin{aligned}(\alpha_i \overset{\theta}{\rightsquigarrow} \alpha_j) \iff & \exists l_n, l_m \in \theta.loglist \mid n < m, \exists x \in \theta.attset \mid \\ & (l_n = \alpha_i^r(x) \wedge l_m = \alpha_j^w(x)) \vee \\ & (l_n = \alpha_i^w(x) \wedge l_m = \alpha_j^r(x)) \vee \\ & (l_n = \alpha_i^w(x) \wedge l_m = \alpha_j^w(x))\end{aligned}$$

Let Ψ be a set of computational activities and let $\rightsquigarrow$ be the associated

dependency relation. The serializability theorem can then be expressed as following:

Theorem 1 (Serializability) *An execution ξ of Ψ is a serializable execution iff the transitive closure of the dependency relation $\rightsquigarrow$ is a partial order.*

Thus, we can define computational consistency: a distributed computation Ω and a collection of activities ξ are computationally consistent iff ξ is serializable.

5.3 Computational Consistency

Let Ω be a distributed computation and ξ an execution of a set Ψ of computational activities. In order for Ω to be consistent (from a computational viewpoint) ξ must be serializable. The serializability of ξ implies that ξ generates the same result as some sequential execution of Ψ. Let us represent the serialized (sequence of) activities by Ψ_s:

$$\Psi_s = \{\alpha_1, \ldots, \alpha_i, \ldots, \alpha_n\}$$

For this serialized execution (of Ψ_s), we use $\longrightarrow_s$ to reflect the serialized nature of the system transitions. The evolution of the global state of the system can therefore be represented by:

$$\Omega_{init} \xrightarrow{\alpha_1}_s \Omega_1 \cdots \xrightarrow{\alpha_i}_s \Omega_i \xrightarrow{\alpha_{i+1}}_s \cdots \xrightarrow{\alpha_n}_s \Omega_n$$

Or, more simply: $(\Omega_{init}, \longrightarrow_s, \Psi_s)$.

At the end of each activity α_i, we have: $\Omega_i = \Omega_{init}.\alpha_1 \ldots \alpha_i$

Note that within this serialization, at the end of execution of an activity α_i, there are no messages left (since there are no present activities), and the system is composed of instances of computational objects only.

6 ARTICULATION BETWEEN THE TWO MODELS

In order to carry on in our study of articulation between the information and computational models, we need to make a choice for the mapping between information objects and computational objects. We choose to map each computational object to an information object. However, our results could be obtained also with more general forms of mappings. In our mapping, associations between information objects are represented by special attributes. In order to capture this mapping, we can extend our formal abstract syntax of computational objects (figure 3) and add a new label *θ.relset* as following:

$$\theta ::= \langle id : \delta, class = c, attset = \beta, relset = \varphi, actset = \tau, loglist = \lambda \rangle$$

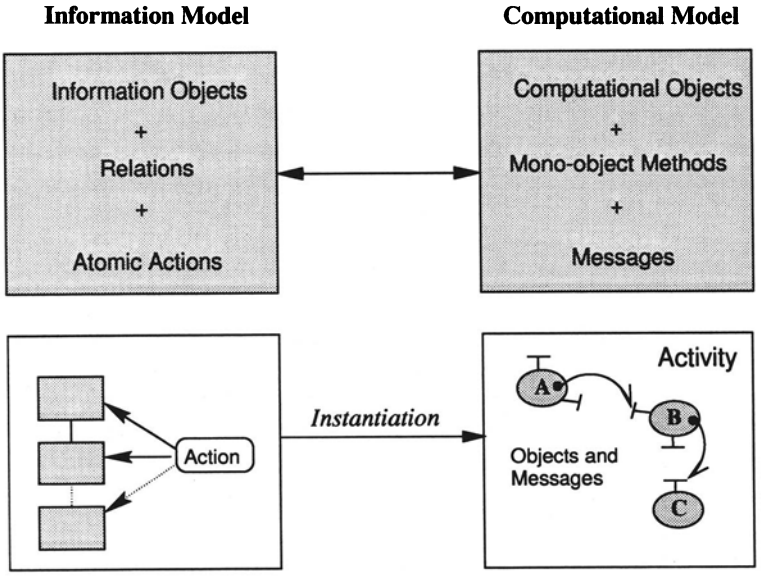

Figure 4 Relation between the Information and Computation Models

- $\theta.relset = \varphi$ represents a list of relation attributes tagged by a relation name r, a role name and a list of identifiers of computational objects to which θ is associated via relation r. Each relation attribute $[\varphi]_i$ may be represented by: $[\varphi]_i = \langle relname = r, role : \aleph, idlist : \mathbb{F}\,\delta \rangle$

We can now consider a function $\mathcal{H} : \Theta \longrightarrow \mathcal{IS}$ defined on instances of computational objects, that would yield elements of *instance systems*, i.e. instances of information objects and instances of information relations.

An Information/Computation Consistency Property

Besides being consistent (serializable) from a computational viewpoint, distributed computations must also satisfy the invariants defined in the information viewpoint. More specifically, (i) for each computational activity α_i, the pre-conditions of the corresponding information action A^{α_i} must be satisfied before the activity is able to be executed, and (ii) at the end of execution of each activity, the post-conditions of the corresponding information action as well as the global information invariants must be satisfied. These rules are formalized in the following:

Definition 8 *We define valid correspondence between an information representation* $(cs, Inv, \Gamma = \{A_1, \ldots, A_k\})$ *and a computational representation* $(\Omega_{init}, \longrightarrow_s, \Psi^* = \{\alpha_1, \ldots, \alpha_n\})$ *iff :*

$$\begin{aligned} \forall \alpha_i \in \Psi_s, \mathcal{H}(\Omega_{init}.\alpha_1 \ldots \alpha_i) \in [\![cs]\!]_{Inv} \quad & \wedge \\ A^{\alpha_i}_{Pre}(\mathcal{H}(\Omega_{init}.\alpha_1 \ldots \alpha_{i-1})) = \text{true} \quad & \wedge \\ A^{\alpha_i}_{B}(\mathcal{H}(\Omega_{init}.\alpha_1 \ldots \alpha_{i-1})) = (\mathcal{H}(\Omega_{init}.\alpha_1 \ldots \alpha_i)) \quad & \wedge \\ A^{\alpha_i}_{Post}(\mathcal{H}(\Omega_{init}.\alpha_1 \ldots \alpha_i)) = \text{true} & \end{aligned}$$

7 CONCLUSION

In the study presented here, we aimed at providing a formal basis for the ODP information and computational models. In particular, we first provided a formal semantics for a hybrid OMT/Fusion object model. OMT is a widely used graphical notation for information modeling in ODP. We used OMT's class/relation paradigm as a notation to describe the structure of information of systems. Much research is now being carried out to endow these models with formal semantics, either directly (like our approach) or through a translation to Z [9] or through a translation to an Abstract Data Types language [8]. As for the dynamic aspects of the information model, we choose the Fusion model of actions, which we find more suited for the description of multiple object interactions.

We then considered the computational model for which we extended an existing formalization with the concepts of activities and object logs. Computational activities may run concurrently on collections of computational objects and thus, we provided consistency rules which define acceptable interleaving of activities. For that purpose, we choose the well known serializability property to characterize the valid interleavings.

The computational model, endowed with its intra-consistency rules, can then be checked against the information model specification. It is important to observe the state changes of a distributed system and to make sure that the system never reaches a state where the rules and invariant schemas defined in higher levels of abstraction could be violated. Thus, in our study, we observe a system at the computational level to ensure its consistency (i) from a computational viewpoint, and (ii) with respect to the rules and invariants imposed by its information specification.

A future direction to our study would be the provision of Information to Computational transformations that preserve the consistency of the source and target models. Such transformations would need to cover two main issues: (i) mapping the global state, assignment oriented, approach of programming into a distributed, message passing style; and (ii) introducing mechanisms, such as transactional processing monitors, in order to ensure the intra-computational consistency for multiple, concurrent, and interleaved activities.

REFERENCES

[1] UIT-T X.901 | ISO/IEC 10746-1 Part 1 Overview: "Reference Model of Open Distributed Processing" – July 1994.

[2] Object-Oriented Modeling Technique – J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorenson: Prentice Hall, Englewood Cliffs, N.J. 1991.

[3] Object Oriented Development: The Fusion Method – D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes and P. Jeremaes: Prentice Hall, 1994.

[4] Fundamentals of Object-Oriented Specification and Modeling of Collective Behaviors – Reino Kurki-Suonio: (Eds. H. Kilov, W. Harvey), Kluwer Academic Publishers 1996.

[5] A Formal Semantics for the ODP Computational Model – E. Najm and J-B. Stefani: Computer Networks and ISDN Systems, Volume 27, 1995.

[6] Formal Support to ODP – Joubine Dustzadeh, Elie Najm: INDC'96, Sixth IFIP/ICCC Conference on Information Network and Data Communication, June 1996. Chapman & Hall.

[7] Semantics of Information Models for Open Object-Based Distributed Systems – Joubine Dustzadeh: Ph.D. thesis, Dec. 1996, ENST Paris.

[8] Robert H. Bourdeau, Betty H. C. Cheng: "A Formal Semantics for Object Model Diagrams" – IEEE Transcations on Software Engineering, October 1995.

[9] B. W. Bates, J-M. Bruel, R. B. France, M. M. Larrondo-Petrie: "Formalizing Fusion Object Oriented Analysis" – FMOODS'96, March 1996.

[10] H. Bowman, E.A. Boiten, J. Derrick and M.W.A. Steen: "Viewpoint Consistency in ODP, a General Interpretation" – FMOODS'96, March 1996.

APPENDIX

In figure 5, *ucrs* denotes a generic unflattened class relation system, uc, uc_1 and uc_2 denote classes and r, r_1 and r_2 denote relations.

In figure 6, *is* represents an instance system, ie, ie_1 and ie_2 represent information elements (instances of object or association) within an instance system, ω_1 and ω_2 represent object instances, and ρ is an association instance.

In figure 7, F represents a forest composed of inheritance trees. T denotes an inheritance tree. $\sigma(T)$ is a boolean predicate on T which returns true if T is a singleton tree and false otherwise. R_T is the root of tree T. N_j^T denotes the j-th (immediate) subclass (node) of the root R_T.

(UCRS.WF.1): $\forall r \in ucrs,\ \exists uc_1, uc_2 \in ucrs$ such that :
$(r.class_1 = uc_1.name) \wedge (r.class_2 = uc_2.name)$

(UCRS.WF.2): $\forall uc_1, uc_2 \in ucrs, (uc_1 \neq uc_2) \Rightarrow (uc_1.name \neq uc_2.name)$

(UCRS.WF.3): $\forall uc \in ucrs, \forall a_1, a_2 \in uc.attlist,$
$(a_1 \neq a_2) \Rightarrow (uc.a_1.name \neq uc.a_2.name)$

(UCRS.WF.4): $\forall r \in ucrs, (r.class_1 = r.class_2) \Rightarrow$
$(\{r.role_1, r.role_2\} \neq \{\varepsilon, \varepsilon\})$

(UCRS.WF.5): $\forall r \in ucrs, (r.role_i = \kappa) \Rightarrow (r.role_{\bar{i}} \neq \kappa)$

(UCRS.WF.6): $\forall r \in cs, \exists c \in cs \wedge \exists i \in \{1,2\}$ such that:
$((r.class_i = c.name) \wedge (\forall a_j \in c.attributes, r.role_i \neq c.a_j.name))$

(UCRS.WF.7): $\forall r_1, r_2$ such that: $\{r_1.class_1, r_2.class_2\} = \{r_2.class_1, r_2.class_2\}$
(if $(r_1.class_1 = r_2.class_1) \Rightarrow$
$(r_1.role_1, r_1.name, r_1.role_2) \neq (r_2.role_1, r_2.name, r_2.role_2)$)
(if $(r_1.class_1 = r_2.class_2) \Rightarrow$
$\{r_1.role_1, r_1.name, r_1.role_2\} \neq (r_2.role_{\bar{1}}, r_2.name, r_2.role_{\bar{2}})$)

(UCRS.WF.8): There is no sequence $uc_1, \ldots, uc_n \in ucrs$ such that:
$(\forall i < n,\ (uc_{i+1}.name \in uc_i.inherits)) \wedge$
$(uc_1.name \in uc_n.inherits)$

(UCRS.WF.9): $\forall uc \in ucrs, (uc.abstract = \mathsf{true}) \Rightarrow$
$(\exists uc' \in ucrs \mid uc.name \in uc'.inherits)$

Figure 5 Some Well-formedness Rules for Unflattened Class Relation Systems

(IS.WF.1): $\forall\, ie_1, ie_2 \in is, (ie_1 \neq ie_2) \Rightarrow (ie_1.id \neq ie_2.id)$

(IS.WF.2): $\forall \rho \in is,\ \exists \omega_1, \omega_2 \in is$ such that : $(\rho.id_1 = \omega_1.id) \wedge (\rho.id_2 = \omega_2.id)$

(IS.WF.3): $\forall\, ie \in is, \forall\, a_1, a_2 \in ie.attlist,$
$(ie.a_1 \neq ie.a_2) \Rightarrow (ie.a_1.name \neq ie.a_2.name)$

(IS.WF.4): $\forall \rho \in is, \{\rho.role_1, \rho.role_2\} \neq \{\kappa, \kappa\}$

Figure 6 Some Well-formedness and Static Semantics Rules for Instance Systems

let F be the forest of all inheritance trees;

begin
  while $\exists\ T_i \in F\ \mid\ \sigma(T_i) =$ false **do**
    $\forall j \in \{1, \ldots, k\}$ **do**
      $N_j^{T_i}.attlist := N_j^{T_i}.attlist \bigcup R_{T_i}.attlist$;
      create a relation to N_i^T for each relation of R_{T_i};
      remove parent link $N_j^{T_i}.parset$;
    od
    if $(R_{T_i}.abstract =$ false$)$ **then**
      $F := (F - T_i) \bigcup R_{T_i} \bigcup \{N_1^{T_i}, \ldots, N_k^{T_i}\}$;
    fi
    if $(R_{T_i}.abstract =$ true$)$ **then**
      remove relations of R_{T_i};
      $F := (F - T_i) \bigcup \{N_1^{T_i}, \ldots, N_k^{T_i}\}$;
    fi
  od
end

Figure 7 A Flattening Algorithm for Inheritance Trees