Journal of Theoretical and Applied Information Technology

www.jatit.org

SPECIFYING BEHAVIORALCONCEPTS BY ENGINEERING LANGUAGE OF RM-ODP

[1]JALAL LAASSIRI, [1]SAÏD EL HAJJI, [1]MOHAMED BOUHDADI

[1]University Mohammed V-Agdal, Faculty of Sciences, Department of Mathematic and Informatics, Laboratory of Mathematic and Informatics and Applications, Rabat, Morocco.

ABSTRACT

Distributed systems can be very large and complex and the many different considerations which influence their design can result in a substantial body of specification, which needs a structuring framework if it is to be managed successfully. The purpose of the RM-ODP is to define such a framework. The Reference Model for Open Distributed Processing (RM-ODP) provides a framework within which support of distribution, inter-working and portability can be integrated. It defines: an object model, architectural concepts and architecture for the development of ODP systems in terms of five viewpoints. However, RM-ODP is a meta-norm, and several ODP standards have to be defined. Indeed, the viewpoint languages are abstract in sense that they define what concepts should be supported and not how these concepts should be represented using the UML/OCL meta-modeling approach. In this paper, we report on the definition and address of the syntax and semantics for a fragment of ODP object concepts defined in the RM-ODP foundations part and in the Engineering language. These concepts are suitable for describing and constraining ODP engineering viewpoint specifications. We give new approach and model it by using the RM-ODP engineering language.

Keywords: *RM-ODP, Engineering Language, Engineering viewpoint, Structural Concepts, Denotational Meta-modeling Semantics, UML/OCL.*

1. INTRODUCTION

The rapid growth of distributed processing has led to a need of coordinating framework for the standardization of Open Distributed Processing (ODP). The open distributed processing (ODP) computational viewpoint describes the functionality of a system and its environment in terms of a configuration of objects interacting at interfaces, independently of their distribution. Quality of service (QoS) contracts and service level agreements are an integral part of any computational specification, which is specified in ODP in terms of environment contracts. The Reference Model for ODP (RM-ODP) [1-4] provides such a framework. It creates an architecture supporting distribution, networking and portability. The foundations part [2] contains the definition of concepts and analytical framework for normalized description of (arbitrary) distributed processing systems. These concepts are gathered in several categories including basic modeling concepts, specification concepts, organizational concepts, and structuring concepts. The architecture part [3] contains specifications of the required characteristics that qualify distributed processing to be open. It defines a framework containing:

• Five viewpoints called: enterprise, information, computation, engineering and technology; which provide a basis for the ODP systems specification.

• A language for each viewpoint, defining concepts and rules to specify ODP systems from the corresponding viewpoint.

• Specifications of functions required to support ODP systems.

• Transparency prescriptions, showing how to use the ODP functions to achieve distribution transparency.

In other words, the first three viewpoints do not take into account neither distribution nor heterogeneity inherent problems. This principle corresponds closely to the concepts of PIM

(Platform Independent Model) and PSM (Platform Specific Model) models in MDA (Model Driven Architecture) architecture [5]. However, RM-ODP is a meta-norm [6] and can not be directly applied. Indeed, for instance, the viewpoint languages are abstract in sense that they define what concepts should be supported, not how these concepts should be represented. It is important that RM-ODP does not use the term language in its largest sense: a set of terms and rules for the construction of statements from terms; it does not propose any notation for supporting viewpoint languages.

In fact, RM-ODP provides only a framework for the definition of new ODP standards. These standards include those for ODP functions [7-8]; standards for modeling and specifying ODP systems; standards for methodology, programming, implementing, and testing ODP systems. Elsewhere, the languages Z [9], SDL [10], LOTOS [11] and, Esterelle [12] are used in RM-ODP architectural semantics part [4] for the specification of ODP concepts. Unfortunately, up to now, no formal method is suitable to specify and verify every aspect of an ODP system. The inherent characteristics of ODP systems imply the need to integrate different specification languages and to handle non-behavioral properties of ODP systems that is the QoS concepts.

There had been an amount of research to apply UML [13] as a syntactic notation for the ODP viewpoint language [14-16]. The taken approach is to give a meta-model description for the language; it is a definition of this language by itself. This is presented in terms of three views: the abstract syntax, the well-formedness rules and the modeling elements semantics. The abstract syntax is expressed using a subset of UML static modeling notations that are class diagrams. The well-formedness rules are expressed in OCL [17], a precise language based on first order-logic. OCL is used for expressing constraints on objects structure which cannot be expressed by class diagrams only. We used the meta-modeling approach [18] to define syntax of a sub-language for ODP QoS-aware enterprise viewpoint specifications.

Furthermore, a part of UML meta-model itself has a precise semantic [19] defined using denotational meta-modeling approach. The denotational approach [20] is realized by defining the instance form of every language element and a set of rules determining which instances are denoted or not by a particular language element. There are three main steps through a denotational meta-modeling approach to the semantics:

1. Define the meta-model for the model's language: object template, interface template, action template, type, and role.
2. Define the meta-model for the instances' language: objects, binders, and interfaces.
3. Define the mapping or the meaning function between these two languages.

There are good reasons for adopting the UML meta-modeling approach in context of ODP systems. The UML meta-models provide a precise core of any CASE tool. The tools include a consistency checker that makes sure that invariants defined on a model do not conflict, a consistency checker between meta-models makes sure that different system specifications are consistent and do not conflict. Besides, for testing ODP systems [2, 3], the current techniques [21, 22] are not widely accepted. A new approach for testing, named agile programming [23, 24] or test first approach [25] is being increasingly adopted. The opinion is integrating system model and testing model using UML meta-modeling approach [26]. This approach is based on the executable UML [27]. In this context, OCL is used to specify the properties that have to be tested. OCL also serves to attach constraints to UML meta-models in order to verify the coherence of meta-models and to translate the constraints into code to evaluate them on instance models.

The part of RM-ODP considered in this paper is a subset for describing and constraining the structure of ODP Engineering viewpoint specifications. It consists of modeling and specifying concepts defined in the RM-ODP foundations part and concepts in the Engineering language. The UML/OCL meta-model developed here elaborates the conceptual core of the ODP Engineering viewpoint language. We do not consider concepts for describing dynamic behavior.

The rest of the paper is organized as follows. Section 2 presents the literature review. Section 3 describes the subset of concepts considered in this work named the object model and Engineering viewpoint. Section 4 describes the meta-model for generic models, object, action, template, type/subtype, class/subclass and basic/derived class. Section 5 describes the meta-model for models instances, which are essentially object diagrams. Section 6 makes the connection between models and their instances. This introduces the basic form of the semantic approach described here. Section 7 we give approaches and model it by using the RM-ODP engineering language. A

conclusion and perspectives end the paper.

A conclusion and perspectives finalize the paper.

2. LITERATURE OF REVIEW

Behavior models play a central role in system specifications. Many specification languages can be used to specify the behavior of a business and IT systems. A system designer chooses a particular language depending on the designer's experience and on the problems he is trying to solve. For example, to show the conformance of the implementation of a system behavior with its specification, a system designer can use formal languages (for example, Pi-calculus). To visualize the state machine of a developed system, a system designer may use a UML statechart diagram or activity diagram (a variation of a state machine in which the states represent the performance of actions or subactivities [34]). The design of complex systems requires that a system designer solve many problems simultaneously (visualize a model, check the conformance of a model, etcetera), thus several specification languages should be used. This raises a problem: a system designer needs to build several independent models of the same system. This leads to the duplication of the information, which can be an additional source of errors: models done in different languages can be inconsistent.

The concept from the RM-ODP semantic domain that is necessary for the modeling of the behavior of systems was considered in [35]. The basic concepts that were used in this work are taken from the clause 8 "Basic modeling concepts" of the RM-ODP Part 2. These concepts are: action, time, and state. According to [30], these concepts are essentially the first-order propositions about model elements. Also used, some concepts (type, instance, and precondition, postcondition) from the clause 9 "Specification concepts". Specification concepts are the higher-order propositions applied to the first-order propositions about the model elements. Wegmann [31] states: "Basic Modeling Concepts and generic Specification Concepts are defined by RM-ODP as two independent conceptual categories. Essentially, they are two qualitative dimensions that are necessary for defining model elements that correspond to entities from the universe of discourse".

To explain the semantics of the generic model more clearly, the Alloy formalism was used. Alloy is a simple modeling language that allows a modeler to describe the conceptual space of a problem domain. Using Alloy, specifying the RMODP semantic domain can be obtained.

RM-ODP conceptual elements from the semantic domain can be partitioned in the following way:

```
model RM-ODP {
domain {ODP_Concepts}
state {
partition … BasicModellingConcepts,
SpecificationConcepts : static ODP_Concepts
…
      }
```

Code Fragment 1. RM-ODP model

Let's consider the minimum set of modeling concepts (Basic Modeling Concepts and Specification Concepts) necessary for the specification of systems behavior. There are a number of approaches for specifying the behavior of distributed systems coming from people with different backgrounds and considering different aspects of behavior. "However, they can almost all be described in terms of a single formal model" [32]. Based on [32], Lamport specifies the behavior of a concurrent system. A system designer has "to specify a set of states, a set of action and a set of behavior". Each behavior is modeled as a finite or infinite sequence of interchangeable states and actions.

To describe this sequence there are mainly two dual approaches. According to [33] they are:

1. "Modeling systems by describing their set of actions and their behaviors".
2. "Modeling systems by describing their state spaces and their possible sequences of state changes".

"These views are dual in the sense that an action can be understood to define state changes, and state changes occurring in state sequences can be understood as abstract representations of actions" [33]. In [35] work, he/she considers both of these approaches as an abstraction of the more general approach based on RM-ODP.

3. RM-ODP

RM-ODP is a framework for the construction of

open distributed systems. It defines a generic object model in foundations part, and an architecture which contains specifications of the required characteristics that qualify distributed processing as open. The architecture extends and specializes object concepts of foundations part. The RM-ODP architecture model consists of a set of five viewpoint models, the concepts and rules associated with the language of each model, the distribution transparency constructs, and the ODP functions. The entire RM-ODP model is based on the RM-ODP foundations of an object model, rules for specification, and rules for structuring

RM-ODP (Model Reference - Open Distributed Processing) [ISO96a] [ISO96b] [ISO98] is an international standard published by ISO/IEC. It provides a reference model for the specification of open distributed applications. The RM-ODP model can describe a system according to five viewpoints; each viewpoint is interested in a particular aspect of the system. These viewpoints are:

Enterprise. It introduces the concepts necessary to represent a system in the context of an enterprise on which it operates. It is interested to the objective and the policies of a system. A system is then represented by a community which is a configuration of enterprise objects formed to achieve a goal.

Information. It is focused on the semantics of information and the treatment carried out on information. A system is then described by Engineering objects, relationships and behavior. The description is expressed through the use of three diagrams named invariant, static and dynamic.

Computational. It allows a functional decomposition of the system. The various functions are fulfilled by objects that interact thanks to their interfaces. The basic concepts define the type of the interfaces which the computational objects support, the way in which the interfaces can be bound, and the forms of interaction which can take place. Adaptation management in multi-view systems

Engineering. It is focused on the deployment and communication of a system. It defines communication concepts like channel, stub, skeleton and deployment concepts like cluster, capsule, etc.

Technology. It describes the implementation of a system in term of configuration of technical objects representing the hardware and software components of the implementation. The goal of such a description is to provide additional information for the implementation and the test, by selecting standard solutions for the components and the communication mechanisms.

RM-ODP OBJECT MODEL (FOUNDATIONS PART)

The RM-ODP international standard [5] presents a very good architectural framework for modeling distributed systems. In our experience, unfortunately at the present time not many modelers use the standard in their everyday practice. It's a pity, considering the amount of highly qualified experts' knowledge invested in the project and the big constructive potential that its results might bring to practice if they were adequately used. We see one of the ways to promote the use of RM-ODP in formalization of its framework. The formalization requires a careful and attentive translation of the standard definitions into formal logical constructions, but once done it would allow creation of ODP-based software toolsets that could bring to modelers an "easy to be applied" version of the standard.

Generally, the term object model refers to the collection of concepts used to describe objects in an object-oriented specification (OMG CORBA), Object model [5] and RM-ODP object model [4]. It corresponds closely to the use of the term data-model in the relational data model. To avoid misunderstandings, the RM-ODP defines each of the concepts commonly encountered in object oriented models. It underlines a basic object model which is unified in the sense that it has successfully to serve each of the five ODP viewpoints. It defines the basic concepts concerned with existence and activity: the expression of what exists, where it is and what it does. The core concepts defined in the object model are object and action. An object is the unit of encapsulation: a model of an entity. It is characterized by its behavior and, dually, by its states. Encapsulation means that changes in an object state can occur only as a result of internal actions or interactions. An action is a concept for modeling something which happens. ODP actions may have duration and may overlap in time. All actions are associated with at least one object: internal actions are associated with a single object; interactions are actions associated with several objects.

Objects have an identity, which means that each object is distinct from any other object. Its identity implies that there exists a reliable way to refer to objects in a model. Depending on the RM-ODP viewpoint, the emphasis may be placed on behavior

or on states. When the emphasis is placed on behavior, an object is informally called to perform functions and offer services, these functions are specified in terms of interfaces. An interface is a subset of interactions where an object can participate. Contrary to other object models, ODP object can have multiple interfaces.

The other concepts defined in the object model are derived from concepts of object and action; those are class, template, type, subtype/ supertype, subclass/ superclass, composition and behavioral compatibility. Though, the composition of objects is a combination of two or more objects yielding a new object. An object is behaviorally compatible with a second object with respect to a set of criteria if the first object can replace the second object with no notice by the environment on the difference in object behavior on basis of that set of criteria.

A type (of an $<x>) is a predicate characterizing a collection of <x>s. The ODP notion of type is much more general than most of object models. Also, ODP permits to define several types, and dynamically change types.

A class (of an <x>) defines the set of all <x>s satisfying a type. 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 explicit class.

An <x> template specifies the common features of a collection x in a sufficient detail that an x can be instantiated using it.

RM-ODP ENGINEERING LANGUAGE

The Engineering held by the ODP system about entities in real world, including the ODP system itself, is modeled in an Engineering specification in terms of Engineering objects, and their relationships and behaviors.

Basic Engineering elements are modeled by atomic Engineering objects. More complex information is modeled as composite Engineering objects which, as any other ODP object, exhibit behavior, state, identity and encapsulation. Its type is a predicate characterizing a collection of engineering objects, which their class is the set of all Engineering objects satisfying a given type.

Engineering object template specifies the common features of an Engineering objects collection in sufficient detail that an Engineering object can be instantiated using it. It may reference static, invariant and dynamic schema.

An action is a model of something that happens in real world. Actions are instances; their types are modeled by ODP action types. An action in the information viewpoint is associated with at least one Engineering Object Class. It can be either internal action or interaction.

An invariant schema is a set of predicates on one or more Engineering objects which must always be true. The predicates constrain the possible states and state changes of the objects on which they apply.

ODP also notes that an invariant schema can specify the types of one or more Engineering objects; that will always be satisfied by whatever behavior the objects might exhibit. A static schema defines the state of one or more Engineering objects, at some point in time, subject to the constraints of any invariant schema.

A dynamic schema is a requirement of the allowable state changes of one or more Engineering objects, subject to the constraints of any invariant schema. A dynamic schema specifies how the information can evolve as the system operates. In addition to describing state changes, dynamic schema can also create and delete Engineering objects, and allow reclassifications of instances from one type to another. Besides, in the Engineering language, a state change involving a set of objects can be seen as an interaction between those objects. Not all the objects involved in the interaction need to change state; some of the objects may be involved in a read-only manner [29].

4. SYNTAX DOMAIN

We define in this section the meta-models for concepts presented in the previous section. Figure 1 defines the context free syntax for the core of object concepts, and figure 2 defines the context free syntax for the Engineering language.

In the following, we define context constraints for the defined syntax.

Context m: Model **inv**:

m.Specifier->includes All
(m.EngineeringObjectTemplates. DynamicSchema)
m.Describer ->includesAll
(m. EngineeringTemplate.StaticSchema)
m.Constrainer->includesAll
(m. EngineeringObject.InvariantSchema)

m.ActionTemplates -> includesAll (m.EngineeringObjectTemplates.action)
m.Types->includesAll(m.ActionTemplates.
Types -> union (m.EngineeringObject.Types)
We consider the concepts of subtype/supertype (RM-ODP 2-9.9) and subclass/superclass (RM-ODP 2-9.10) as relations between types and classes respectively.
Context m: model inv
m.types-> forall(t1: Type, t2: Type | t2.subtype -> includes(t1) implies t1.valid_for.satisfies_type=t2)
m.types-> forall(t1: Type, t2: Type | t1.supertype ->includes(t2) implies t1.valid_for.satisfies_type=t2)

Context a: ActionTemplate **inv**:
a.Engineeringobject.StartState <>
a.Engineeringobject.EndState

Context o: Object Template **inv**:
iot (Engineering object template) is not parent of or child of itself
not (iot.parents ->includes(iot) or iot.children->includes(iot))

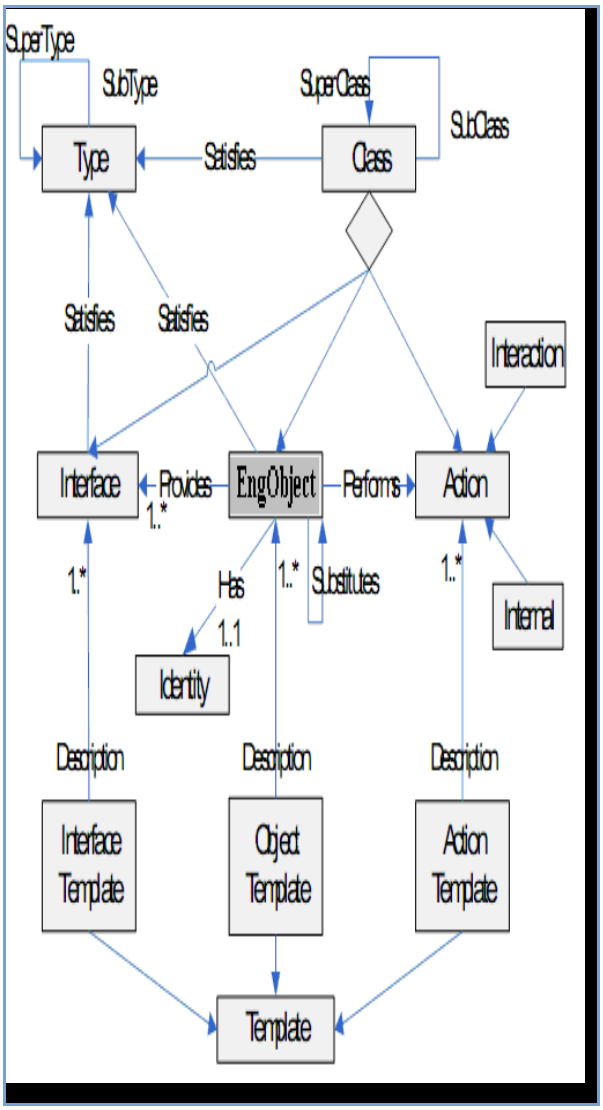

Figure 1: RM-ODP Foundation Object Model

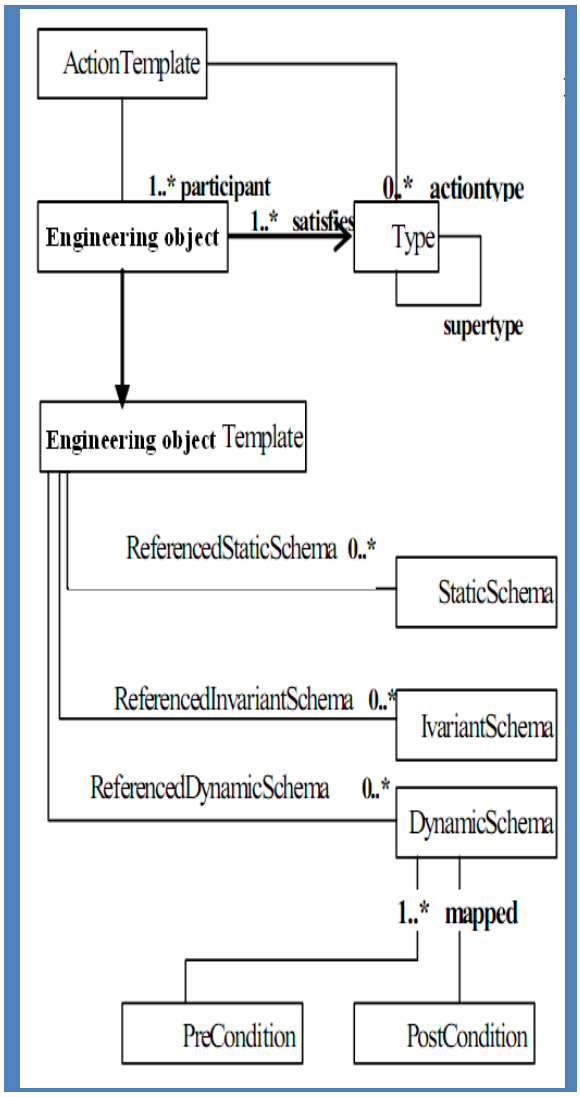

Figure 2 Engineering concepts

5. SEMANTICS DOMAIN

The semantics of a UML model is given by constraining the relationship between a model and possible instances of that model (see Figure 3). It means constraining the relationship between expressions of the UML abstract syntax for models and expressions of the UML abstract syntax for instances. We define a model to specify the ODP Engineering viewpoint. That is, a set of Engineering objects, their relationships and behaviors. This model defines Semantic Domain (figure 3).

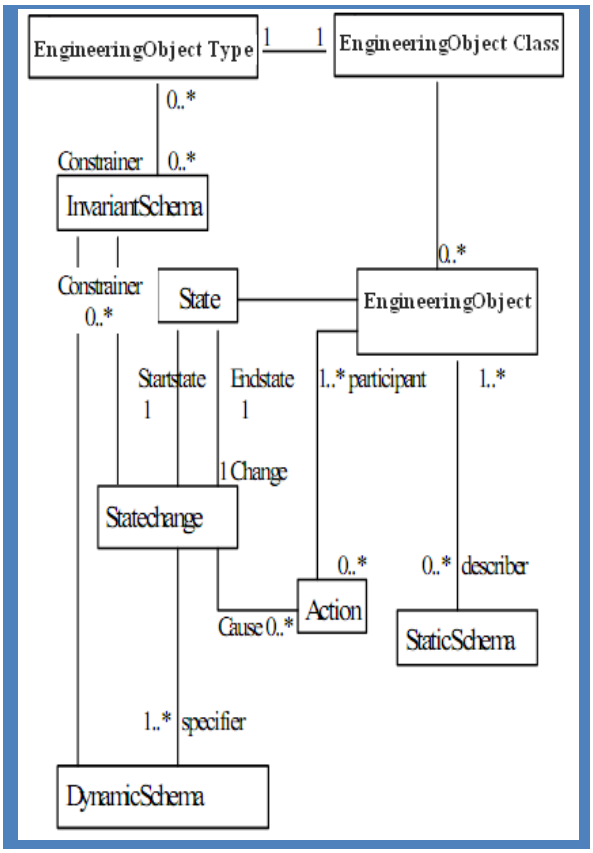

Figure. 3 Semantic Domain

A system can only be an instance of a single system model, because it is self contained and disjoint from other models. On the other side, objects are instances of one ore more object templates; they may be of one or several types. With no further constraints, it is possible for an object to change the templates of which it is an instance; thus this meta-model supports dynamic types.

There is one well-formedness rule for instances, which are given bellow:
Context s: system **inv**:
The source and target engineering objects of s'slinks are engineering objects in s
s.Engineeringobjects->includesAll(s.links.source->union(s.links.target))
Links between two Engineering objects are unique per role
s.links->forAll(l|s.links ->select (l'|l'.source=l.source&l'.target=l.target&l'.of=l.of)=l)
Declaration of "Specification concepts" (RM-ODP 2.9) in Alloy [28], time dependence.
Context Time **inv:**

forall(o:EngineeringObject ,t:Time | t.instant ->notEmpty implies o.state ->notEmpty)
Context Precondition **inv** :
Forall (prec: Dynamicschema.Precondition , o : EngineeringObject|exists(s : State) | o.mappedTo = prec and o.state_start = s)
Context Postcondition **inv** :
forall (postc: dynamicschema.Postcondition , o : EngineeringObject | exists(s : State) | o.mappedTo = postc and a.state_end = s)

6. MEANING FUNCTION

Other invariants are required to constraint the relationships between models and instances. These constitute the semantics which are the subject of this section. The semantics for the UML-based language defined by the relationship between a system model and its possible instances (systems). The constraints are relatively simple, but they demonstrate the general principle. Firstly there are two constraints related to Engineering objects and links, respectively. The first shows how inheritance relationships can force an Engineering object to be of many Engineering Object Template.

Context o: object inv:

The templates of o must be a single template and all the parents of that template

o.of->exists (t | o.of=t->union (t.parents))

The second ensures that a link connects objects of templates as dictated by its role.

Context l: link inv:

Engineering Objects which are the source/target of link have templates which are the source/target of the corresponding roles.

(l.of.source)->intersection (l.source.of) -> notEmpty and (l.of.target)->intersection(l.target.of)->notEmpty

Secondly, there are four constraints which ensure that a model instance is a valid instance of the model, it is claimed to be an instance of:

The first and second ensure that objects and links are associated with templates known in the model.

Context s: system inv:

The model, that s is an instance of, includes all object templates that s.objects are instances of.

s.of.EngineeringObjectTemplates->includesAll(s.EngineeringObjects.of)

The model, that s is an instance of, includes all EngineeringObjectClass that s.EngineeringsObjects are instances of s.of.EngineeringObjectClass ->includesAll(s.s.EngineeringsObjects.of)

The third ensures that links are associated with roles known in the model.

Context s: system **inv**:

The model, that s is an instance of, includes all the role that s.links are instances of

s.of.roles ->includesAll(s.roles.of)

The fourth constraint ensures that the system cardinality constraints on roles are observed.

Context s: system inv:

The links of s respect cardinality constraints for their corresponding role

s.links.of -> forAll(r | let links_in_s be r.instances ->intersect (s.links) in (r.upperBound -> notEmpty implies links_in_s ->size <= r.upperBound) and links_in_s->size >= r.upperbound)

The fifth ensures that reverse links are in place for roles with inverses. If a link is of a role with an inverse, then there is a corresponding reverse link

s.links->forAll (l | l.of.role.inverse ->notEmpty implies s.links ->select (l' | l'.source=l.target & l'.target=l.source & l'.of = l..of.inverse)->size=1.

7. ENGINEERING VIEWPOINT MODELING AND RMODP SPECIFICATIONS.

An engineering specification defines the infrastructure required to support functional distribution of an ODP system by:

- Identifying the ODP functions necessary to manage physical distribution, communication, processing and storage;
- Identifying the roles of different engineering objects supporting the ODP functions.

In order to do this, we specify:

1. The activities that occur within those engineering objects
2. The interactions of the engineering objects.

For achieving that, we respect the engineering language rules such as: interface reference rules, binding rules, cluster, capsule and node rules, etc

ENGINEERING OBJECTS ACTIVITIES

The functions of a software entity are:

- Transferring a software entity;
- Creating a software entity;
- Providing globally unique agent names and locations;
- Supporting the concept of a region;
- Ensuring a secure environment for software entity operations.

We specify these functions of a Software entity with the ODP functions.

ENGINEERING OBJECTS INTERACTIONS

We define three types of interactions related to interoperability:

- Remote software entity creation;
- Interaction needed for the software entity transfer;
- Software entity method invocation.

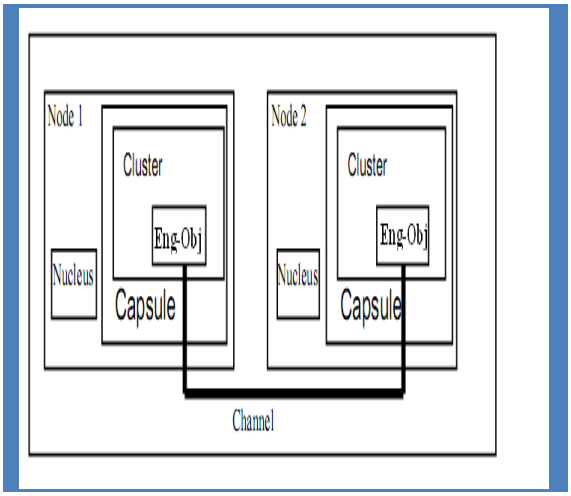

Figure 4: organization of the engineering objects

A client could be a non-software entity program or a software entity from a software entity having the same system type as the destination agent or not. This client authenticates itself to the destination software entity system and interacts with the destination software entity to request the creation of a software entity.

When a software entity transfers to another software entity, the software entity system creates a travel request providing information that identifies the destination place. In order to fulfill the travel request, the destination software entity transfers the software entity's state, authority, security credential and the code.

For example in data base System server, a channel between system client manager Object and the system server Object can be defined as illustrated in figure 5.

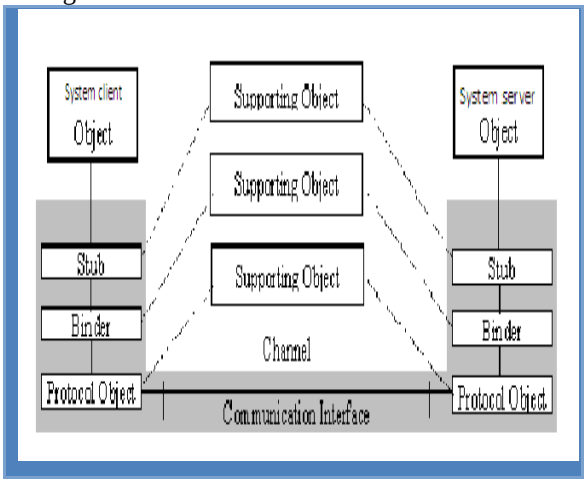

Figure.5: An example of a basic system client / system server channel.

A system client object invokes a method of another system client object or system server object if it has the authorization and a reference to the system client object.

8. CONCLUSION

The Reference Model for Open Distributed Processing (RM-ODP) provides a framework which supports distribution, inter-working and portability can be integrated. However, the ODP viewpoint languages define what concepts should be supported, not how these concepts should be represented. In addition, the UML standard has adopted a meta-modeling approach to define the abstract syntax of UML. One approach to define the formal semantics of a language is denotational: essentially elaborating the value or instance denoted by an expression of the language in a particular context. However, when we use the denotational meta-modeling approach in this paper, we defined the UML/OCL based syntax and semantics of a language for a fragment of ODP object concepts described in the foundations part and in the Engineering viewpoint language. Indeed, these concepts are suitable to define and constrain ODP Engineering viewpoint specifications. In parallel, we are applying the same approach to define a language of concepts characterizing dynamic behavior.

REFRENCES:

[1] ISO/IEC, Basic Reference Model of Open Distributed Processing-Part1: Overview and Guide to Use, ISO/IEC CD 10746-1, July 1994.

[2] ISO/IEC, RM-ODP-Part2: Descriptive Model, ISO/IEC DIS 10746-2, February 1994.

[3] ISO/IEC, RM-ODP-Part3: Prescriptive Model, ISO/IEC DIS 10746-3, February 1994.

[4] ISO/IEC, RM-ODP-Part4: Architectural Semantics, ISO/IEC DIS 10746-4, July 1994.

[5] OMG, the Object Management Architecture, OMG, 1991. http://www.omg.org

[6] M. Bouhdadi et al. A Methodology for the Development of Open Distributed Systems, Proc. JDIR'98, Paris France October 1998, pp. 200-208

[7] ISO/IEC, ODP Type Repository Function, ISO/IEC JTC1/SC7 N2057, January 1999.

[8] ISO/IEC, the ODP Trading Function, ISO/IEC JTC1/SC21, June 1995.

[9] J.M. Spivey, The Z Reference manual, Prentice Hall, 1992.

[10] IUT, SDL: Specification and Description Language, IUT-T-Rec. Z.100, 1992.
[11] ISO and IUT-T, LOTOS: A Formal Description Technique Based on the Temporal Ordering of Observational Behavior, ISO/IEC 8807, August 1998.
[12] H. Bowman et al. FDTs for ODP, Computer Standards & Interfaces Journal, Elsevier Science Publishers, Vol.17, No.5-6, 1995, pp. 457-479.
[13] J. Rumbaugh et al., the Unified Modeling Language, Addison Wesley, 1999.
[14] B. Rumpe, A Note on Semantics with an Emphasis on UML, Second ECOOP Workshop on Precise Behavioral Semantics, Technische Universitaty unchen publisher, 1998.
[15] A. Evans et al., Making UML precise, OOPSLA'98, October 1998,
[16] A. Evans et al. The UML as a formal notation, UML'98, France June 1998, LNCS 1618, Springer Berlin, 1999, pp. 336-348
[17] J. Warner and A. Kleppe, the Object Constraint Language: Precise Modeling with UML, Addison Wesley, 1998.
[18] M. Bouhdadi et al, An UML-based Meta-language for the QoS-aware Enterprise Specification of Open Distributed Systems, IFIP TC5/WG5.5 Third Working Conference on Infrastructures for Virtual Enterprises (PRO-VE'02), May 1-3 Sesimbra Portugal, Kluwer Vol. 213 (IFIP Conference Proceeding series), 2002. Collaborative Business Ecosystems & Virtual Enterprises IFIP Series Vol. 85 Springer Boston 2002.
[19] S. Kent, S. Gaito, N. Ross. A meta-model semantics for structural constraints in UML,, In H. Kilov, B. Rumpe, and I. Simmonds, editors, Behavioral specifications for businesses and systems, chapter 9, pages 123-141. Kluwer Academic Publishers, Norwell, MA, September 1999.
[20] D.A. Schmidt, Denotational semantics: A Methodology for Language Development, Allyn and Bacon, Massachusetts, 1986.
[21] Myers, G. The art of Software Testing, John Wiley &Sons, New York, 1979
[22] Binder, R. Testing Object Oriented Systems. Models. Patterns, and Tools, Addison-Wesley, 1999
[23] Cockburn, A. Agile Software Development. Addison-Wesley, 2002.
[24] Bernhard Rumpe. Agile Modeling with UML. Habilitation Thesis, Germany, 2003.
[25] Beck K. Column on Test-First Approach. IEEE Software, 18(5):87-89, 2001
[26] Briand L. and Labiche Y. A UML-based Approach to System testing. In M. Gogolla and C. Kobryn (eds): "UML" – The Unified Modeling Language, 4th Intl. Conference, LNCS 2185. Springer, 2001 pp. 194-208,
[27] Bernhard Rumpe, Executable Modeling with UML. A vision or a Nightmare? In Issues & Trends of Information Technology Management in Contemporary Associations, Seattle. Idea group Publishing, Hershey, London, pp. 697-7001. 2002 Author, Title of the Book, Publishing House, 200X.
[28] A.Naumenko, A.Wegmann, "Proposal for a formal foundation of RM-ODP concepts » conference woodpecker 2001.
[29] ISO/IEC, May 2006, Basic Reference Model of Open Distributed Processing-Use of UML for ODP system specifications, ISO/IEC CD 19793.
[30] Naumenko, A., et al. A Viewpoint on Formal Foundation of RM-ODP Conceptual Framework, Technical report No. DSC/2001/040, July 2001, EPFL-DSC ICA.
[31] Wegmann, A. and A. Naumenko. Conceptual Modeling of Complex Systems Using an RM-ODP Based Ontology. in 5th IEEE International Enterprise Distributed Object Computing Conference - EDOC 2001. 2001. Seattle, ACTION.
[32] Lamport, L. and N.A. Lynch, Distributed Computing: Models and Methods, in Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics. 1990, Elsevier and MIT Press.
[33] Broy, M., Formal treatment of concurrency and time, in Software Engineer's Reference Book, J. McDermid, Editor. 1991, Oxford: Butterworth-Heinemann. p. 23/1-23/19.
[34] OMG, Unified Modeling Language Specification, v 1.3, 1999.
[35] www.infoscience.epfl.ch/record/464/files/BalabkoW03A.pdf