Conceptual Modeling of Complex Systems Using an RM-ODP Based Ontology

Alain Wegmann, Andrey Naumenko
Institute for computer Communications and Applications
Swiss Federal Institute of Technology – Lausanne
EPFL-DSC-ICA
CH-1015 Lausanne, Switzerland
{alain.wegmann, andrey.naumenko}@epfl.ch

Abstract

The development of business and information systems requires a significant amount of modeling. The current modeling languages and tools have difficulties supporting the modeling of systems spanning through multiple organizational levels. The use of inadequate modeling abstractions is one of the important causes for these difficulties. This paper proposes an ontology that defines the concepts needed for object-oriented modeling and gives a graphical example. The ontology is based on RM-ODP and relies on Constructivism and System Theory. The proposed ontology allows the definition of development methods, modeling languages and tools that are applicable to complex systems. This can lead to significant productivity improvements in the business and software development communities.

1. Introduction

The e-economy (e.g. development of the business to customer or business to business applications) and the latest evolution in information technologies (e.g. business protocols, components,...) strongly affect the way enterprises are organized and how information systems are developed and used. To adapt themselves to these new requirements, the enterprises need to re-engineer their overall operations. This re-engineering effort can span through multiple organizational levels such as, for example: the market level (e.g. supply chain), the company level (e.g. business processes), the information system level (e.g. system integration), the software application level (e.g. component-based application), and the software component level (e.g. a Java developed component). We use the term “complex system” to designate the set of all interacting entities found in these various organizational levels [16, 7]. In most development projects, each organizational level is addressed by a different group of professionals each with their own discipline (working methods, terminology, etc). The challenge posed to these communities of professionals is to develop more competitive companies faster. Our overall research goals address the discovery and development of modeling abstractions, tools and methods that target this challenge. To improve the way professionals develop complex systems, it is important to be able to have an adequate representation of the subject of interest (i.e. the complex system). We call this representation the “model”. It represents what the developer defines as the system of interest that she perceives in her reality. The developer manipulates the model through views that are abstractions of the model made for a specific purpose. These views correspond to the artifacts usually present in the development processes. Examples of these artifacts are UML diagrams [19].

To be able to build our model, we need a precise ontology that defines the modeling constructs. As was noted in [15], there are multiple examples of domain specific or even application specific ontologies that are used for e-commerce and web-based applications. An ontology should define a vocabulary of basic terms, a precise specification of their meaning and relations between them [20]. In our approach, we define an ontology applicable for modeling any kind of system. We base our work on the ISO/ITU standard “Reference Model for Open Distributed Processing” (RM-ODP) [10]. The vocabulary defined by RM-ODP is sufficient however incomplete: some terms are missing, some definitions, those defining relationships between concepts in particular, can be improved. This is what this paper presents. As our goal is to model complex systems, we also include in our approach the principles issued from “Constructivism” and “System Theory”. These two theories are an important corpus of knowledge describing the key principles needed for modeling systems. These theories were developed in the 1950s by multidisciplinary teams who were studying living and artificial systems.

As a result, this paper presents an ontology that can be used for system modeling in the development of any kind of application (e. g. business, software, system science, etc). Comparing it to other work, an advantage of our solution is that it is based on the RM-ODP ISO/ITU standard. This standard has demonstrated its usefulness for the modeling of distributed systems.

We can also compare our work with [21]. They base their ontological foundations on the works of Bunge [4, 5]. Their approach is interesting and has many similarities with our results. But they omit several issues, such as the relation between what they call "entities" ("concrete things" in reality) and "conceptual things (i.e. mathematical concepts such as sets and functions)". The fact they omit these relationships can be explained by the absence of their formal definitions (in the form of predicates that can be compared).

By adopting our ontology, which is based on Constructivism and System Theory, the development community could (1) improve the definition of the existing modeling languages such as UML, (2) develop tools that truly supports the modeling of complex systems, and (3) can tightly link the methods to the tools and the notation. This can lead to significant productivity improvements for the software community.

This paper is structured as following: Section 2 – development method and theoretical foundations, Section 3 – interpretation and extension of RM-ODP, Section 4 – application of the ontology, Section 5 – impacts, Section 6 - Conclusions.

2. Theoretical Foundations

The main "theory" that we use as foundation in our work comes from computer science. It is the "Reference Model - Open Distributed Processing" (RM-ODP) [10]. RM-ODP is an ISO/ITU standard approved in 1996. It provides the definitions and relations between concepts useful to describe object-oriented distributed systems. It positions itself as a "meta-standard" for object-oriented modeling standards. The Object Management Group community adopted in 1998 this standard as a base for describing CORBA systems.

To be able to interpret RM-ODP in the context of complex system modeling, we base our work on Constructivism and on System Theory.

Constructivism [12] is an epistemology (i.e. "the study of the nature of knowledge" [1]). It was developed in the 20th century. Constructivism takes its roots in Kant's belief that intuition is an essential part of human understanding. By taking a constructivist approach, we acknowledge the fact that models are valid in the context of the people or systems that develop or use them. The consequence of this fact is the coexistence of multiple models of the same universe of discourse. In practice, for each system of interest, we define a sub-model that represents its corresponding view of the universe of discourse. The model that represents the complex system is actually an assembly of sub-models (one per system of interest). In summary, in line with Constructivism, we put an emphasis on making explicit the context of the views used by the developer. In addition, we allow the developer to capture relationships between the sub-models.

Systems theory was initially developed in the middle of the 20th century [2]. It is a "trans-disciplinary study of the abstract organization of phenomena, independent of their substance, type, or spatial/temporal scale of existence. It investigates both the principles common to all complex entities and the models that can be used to describe them" [1]. System Theory is a constructivist theory. By including the System Theory in our approach, we recognize the commonalities between the various organizational levels and the fact that each organizational level depends on each other. In addition, we leverage on the principles identified in living systems to understand how to structure software and business systems in a better way. One of the principles we found especially useful is the "teleological operation principle". This principle states "all phenomena (which can be modeled) are perceived as teleologic actions (i.e. actions aimed at achieving a project or a goal)" [12]. An example of the application of this principle is the fact that the model is developed to achieve a specific goal (i.e. the project's goal). If a different goal needs to be achieved, a different model might have to be developed. As a consequence, the tools should support these model variations.

RM-ODP was developed for modeling distributed systems. Even though RM-ODP does not refer to System Theory and Constructivism, the existing RM-ODP definitions are compatible with the principles defined in these two theories. By making explicit the relationships between System Theory, Constructivism and RM-ODP, we can better understand how to interpret and use the standard.

3. RM-ODP as ontology for object-oriented modeling

The RM-ODP standard [10] is composed of four parts. Part 1 is an overview of RM-ODP and is non-normative. Part 2 defines the fundamental concepts needed for modeling of ODP systems. Part 3 presents an application of part 2 for particular specification languages. Part 4 is an attempt for formalization of the previous parts done in

Lotos [13], ACT ONE, SDL-92, Z and ESTELLE languages. We focus our research on part 2 and we will make few references to part 3. Parts 1 and 4 are not in the scope of this work.

Part 2 of RM-ODP has 15 sections. The sections 1 to 4 introduce the context, references and abbreviations. Section 5 introduces the categorization of ODP concepts. This section is needed to understand how all the following sections relate to each other. The sections 6 ("basic interpretation concepts"), 8 ("basic modeling concepts), 9 ("specification concepts") are central to our work and are discussed in this document. The sections 7 and 10 to 15 define supplementary concepts that are beyond the scope of this work (with exception of few concepts found in 11 ("contract") and 13 ("client" and "server")).

3.1. Basic Interpretation Concepts

The section 6 of part 2 "basic interpretation concepts" introduces the concepts needed for the interpretation of the concepts defined in the sections 8 ("basic modeling concepts") and 9 ("specification concepts"). The section 6 defines the concepts of:

- "**universe of discourse**" that corresponds to what is perceived as being reality by the developer.
- "**entity**: any concrete or abstract thing of interest" [clause 6.1].
- "**proposition**: an observable fact or state of affairs involving one or more entities, of which it is possible to assert or deny that it holds for those entities." [clause 6.2].

The section 6 defines also "system: something of interest as a whole or as comprised of parts" [clause 6.5]. The notion of system allows the developer to consider a group of entities either as one entity or as multiple related entities. If a component of a system is itself a system, it is called a "sub-system" of the system in which it is a component[1].

When modeling, the developer represents what she finds interesting in the universe of discourse. To explain this representation, we define the following terms:

- **"model"**: representation of the universe of discourse made for a specific purpose.
- **"model element"**: in the model the representation of an entity from the universe of discourse.
- **"quality"**: in the model the representation of a proposition from the universe of discourse.

The Fig. 1 illustrates the relationships between the universe of discourse and the model. Fig. 1 also illustrates the model views (or artifacts) that represent what a development tool shows to the developer. The model views are abstractions of the model made using a modeling language. They represent the entities and their qualities in a given context. Note that a quality in the model is a predicate that characterizes a model element. Similarly a proposition in the universe of discourse is a predicate characterizing an entity. For example, a quality such as "object" or "environment" characterizes a model element as the predicate for its being a "model of an entity"; "action" characterizes a model element for its being "something that happens".

[1] We call the system, in which a subsystem is a component, a "supra-system". This term is not defined in RM-ODP and we take it from [16].

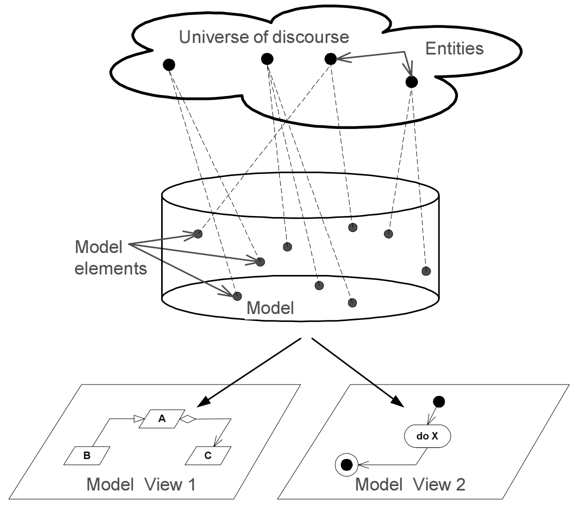


Figure 1. Relationships among what is found in the universe of discourse, in the model and in the model views.

3.2. Basic modeling concepts

The section 8 of part 2 "basic modeling concepts" defines the concepts needed to describe in the model the propositions about the entities of interest found in the universe of discourse.

An entity can be modeled either as an object, or as part of an object (if the object represents a system), or as part of an object's environment. RM-ODP defines:

- "**object**: a model of an entity…" [clause 8.1].
- "**environment** (of an object): the part of the model which is not part of that object" [clause 8.2].

It is important to understand these two definitions from a constructivist standpoint. An object always interacts with its environment (and never directly with another object). By stating this, we acknowledge the fact that each object has its own model of its environment and that the environment mediates the communication between the different objects. Note that, as explained in Section 2, we call "sub-model" the model describing an object. The "model", representing the overall complex system, is composed of many "sub-models".

RM-ODP gives additional precisions on what is an object. It states that an object is "characterized by its behavior and, dually by its state" [clause 8.1]. Note that using our interpretation we can also state that the environment is characterized by its behavior and dually its state. To characterize an object or its environment, we have defined two kinds of information [18]:

- "**behavioral information**" describing the behavior
- "**structural information**" describing the state

The behavioral information and the structural information are a partition of the information on the object. Information is defined in RM-ODP as "any kind of knowledge, that is exchangeable amongst users, about things, facts, concepts and so on, in a universe of discourse" [clause 3.2.5].

RM-ODP defines the following behavioral information elements:

- "**behavior**: a collection of actions, with a set of constraints on when they may occur" [clause 8.6],
- "**action**: something which happens." [clause 8.3].

RM-ODP further defines the concept of action by stating "the set of actions associated with an object can be partitioned into internal actions and interactions. An internal action always takes place without the participation of the environment of the object. An interaction takes place with the participation of the environment of the object." [clause 8.3]

RM-ODP puts a great deal of emphasis on the behavioral information. However, they put fewer considerations on the structural information and they do not define how the structural information is structured. The only definition they have is:

- "**state**: at a given instant of time, the condition of an object that determines the set of all sequences of actions in which the object can take part". [clause 8.7]

Our goal is to have the same level of details in the structural information as in the behavioral information. For this reason, it is necessary to add two concepts, belonging to the structural information, and which are dual to actions and behavioral constraints. These concepts are:

- "**structural information element**": at a given instant in time something perceived by an object or its environment or exchanged between the object and its environment. The set of structural information elements associated with the object or the environment can be partitioned into attributes and parameters. Attributes are accessed by internal actions. Parameters are accessed or modified exclusively within interactions.
- "**structural constraint**": relationship between two or more structural elements. Constraints might include, for example, reference between structural elements or existence within the life cycle of another structural element.

The proposed definitions, by formulating that information elements can be within objects (as attributes) and the corresponding information can be exchanged between objects and their environment (through parameters in interactions), allow describing the information flow between objects.

To be able to precisely model the exchanges between an object and its environment (and thus between objects), we need to explicitly add the concepts of:

- "**client interaction**: interaction initiated by an object towards its environment."
- "**server interaction**: interaction initiated by the environment towards an object".

With the client interaction, the object externalizes information. At the beginning of the client interaction, the values of a given collection of attributes are copied in the corresponding parameters. With the server interaction, the object internalizes information. In that case, when the server interaction completes, the values of the parameters are copied in the corresponding attributes. The concepts of "client" and "server" are defined in the section 13 of part 2.

The classification of the concept of role, initially defined as a Specification Concept, raises an issue. RM-ODP defines role as an identifier of a behavior [clause 9.14]. In our interpretation [9] roles and interfaces are two kinds of behavior abstractions (role is a subset of actions and behavioral constraints of an object participating to a collective behavior; interface is a subset of interactions and behavioral constraints of an object). For this reason, we believe that role and interface should be in the same category. We suggest putting both concepts in the basic modeling concepts (where interface is currently defined). This is justified by both concepts being the specializations of a behavior. Such classification has the advantage of allowing an application of the specification concepts to the role and to the interface (thus allowing, for example, the definition of role instance and role type).

It is important to note the role of time in the definition of the basic modeling concept. Time is needed to define state (information at a specific time). Note the state for a moment in time will be different from the state in the previous moment in time even if no structural element has changed comparing with the state of the previous moment in time (this allows modeling of "null" actions – i.e. actions performing no changes). Time is also needed to define actions (difference of state at different moments of time), and interaction (information exchanged between an object and its environment between the beginning of the interaction and its completion).

3.3. Specification concepts

The section 9 of part 2 "specification concepts" defines the means to be used by a developer to describe in the model the propositions about the propositions describing the entities of interest from the universe of discourse.

For our discussion, we classify the specification concepts in three categories to reflect their role in the modeling task. (1) The "generic specification concepts" used mainly to represent the creation/destruction and the classification of model elements. Such concepts include type, instance, and class. (2) The "abstraction/refinement specification concepts" used mainly to relate groups of model elements at different levels of detail. Such concepts include composition and decomposition. (3) The "schemas" that define the set of predicates needed to define a model element.

3.3.1 Generic Specification Concepts

The generic specification concepts include type, class, subtype / supertype, subclass / superclass, template, instantiation (of a template), introduction, creation, deletion, instance, template type, template class, derived class/base class) [clauses 9.7 - 9.21].

The main specification concepts are:

- "**type**: a predicate characterizing a collection of <X>" [clause 9.7],
- "**instance**: an <X> that satisfies a type" [clause 9.18],
- "**class**: the set of all <X> satisfying a type" [clause 9.8].
- "template": the specification of the common features of a collection of <X>s in sufficient details that an <X> can be instantiated using it." [clause 9.11]

The use of these terms is illustrated in the following example: A behavior instance defines an actual occurrence of a behavior. A behavior class defines a set of behavior instances that share common characteristics. A behavior type defines the common characteristics of the behavior occurrences that belong to the behavior class. A behavior template defines the features of a behavior in a way that allows its instantiation. These concepts are presented in more details in [8].

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 with the prepositions defining them. This is why we consider them as orthogonal as illustrated in Fig. 2.

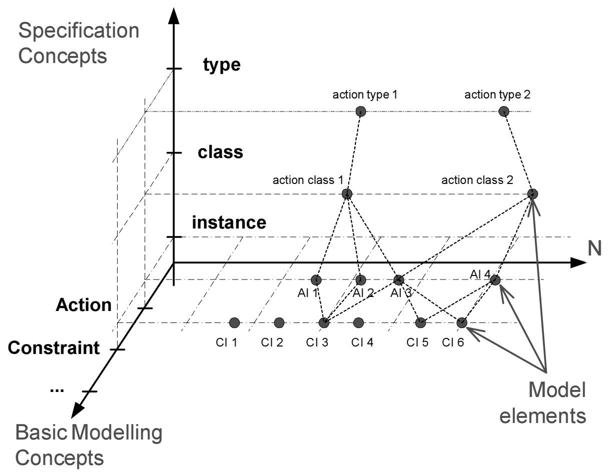


Figure 2. Illustration of use of the basic modeling concepts and the specification concepts for definition of model elements.

The example on Fig. 2 shows several model elements that are defined either as different action types, classes and instances, or as different constraint instances. Applying the RM-ODP definition for behavior ("a collection of actions with a set of constraints on when they may occur" [clause 8.6]), we see, for example, that the action instances AI1, AI2, AI3 with the constraint instance CI3 represent an instance of a behavior type and the action instances A3, A4 with the constraint instances CI3, CI5, CI6 represent another instance of another behavior type.

3.3.2 Abstraction/Refinement Specification Concepts

With our interpretation of RM-ODP, we consider that model elements at various levels of details coexist in the model. In addition, we consider that the relationships between these levels of details are formally established only in the model (and not in the universe of discourse). This means that the developer ultimately has the responsibility to establish these relationships. For this reason, we consider it important for the specification concepts to include:

- "**refinement**: the process of transforming one specification into a more detailed specification." [clause 9.5], and
- "**abstraction**: the process of suppressing irrelevant detail to establish a simplified model, or the result of that process". [clause 6.3]

Currently, "abstraction" is defined as a basic interpretation concept. As both concepts define complementary modifications that can be applied to a model: both concepts should be defined together and both definitions should have the same structure. This means that they should both refer to the process and the result of the process.

"Abstraction" and "refinement" are very general concepts. They describe all modeling tasks that add or remove details (such as the creation/destruction of an object). A special kind of refinement/abstraction is the composition/ decomposition. These concepts are defined as:

- "**composition**:
  a) … a combination of two or more objects yielding a new object …, or
  b) … a combination of two or more behaviors yielding an new behavior…" [clause 9.1]
- "**decomposition**:
  a) … the specification of a given object as a composition.
  b) … the specification of a given behavior as a composition" [clause 9.3]

We suggest modifying these two definitions to make them generic. Currently they are only defined for behaviors and objects but they can be applied to any basic modeling concepts such as roles, interfaces, state, activity, etc. Note, for consistency reasons, it would be useful to explicitly define in generic terms the concept of "component" [clause 9.1]) and "composite" [clause 9.2]. For a more details discussion on abstraction / refinement versus composition / decomposition, refer to [17,18].

As it is defined in the ODP standard [clause 9.1], "composition (of objects) is a combination of two or more objects yielding a new object". If we are interested in the nature of this combination then we should specify the mechanisms that would allow it to yield a new object. We define this mechanism as the "composition constraints". They are a set of structural and behavioral constraints that allow the resulting composite object to fulfill its mediation responsibilities with regard to the component objects participating in the composition. An example of composition constraint will be given in Section 4. In summary, a composite object is the result of the composition of two or more component objects with the corresponding composition constraints. A component object is defined by its structural and behavioral limits. These limits are necessary and sufficient for it to participate in a composition. The structural limits are defined by the object's external state specification. The behavioral limits are determined by the object's interfaces specification.

3.3.3 Schemas

The schemas[2] are used to define concrete model elements. A schema is a mean to group predicates together.

As indicated in Catalysis [6], in system development, first class modeling concepts are: objects and actions. It is thus important to determine what set of predicates (i.e. schemas) are needed for the definition of these two basic modeling concepts. For this reason, we consider that the schemas needed to define an object, as well as the ones needed to define an action, need to be present in RM-ODP part 2.

To specify the behavioral and structural information of an object, we can refer to the clause 6.1 in RM-ODP part 3 [10]. It introduces the necessary schemas needed to define an object[3]. These schemas are:

- "**invariant schema**: a set of predicates on one or more information objects that must always be true. The predicates constraint the possible states and state changes of the objects to which they apply." [10, part 3, clause 6.1.1]
- "**static schema**: a specification of the state of one or more information object at some point in time, subject to the constraints of any applicable invariant schemata." [10, part 3, clause 6.1.2]
- "**dynamic schema**: a specification of the allowable state changes of one or more information object, subject to the constraints of any applicable invariant schemata." [10, part 3, clause 6.1.3]

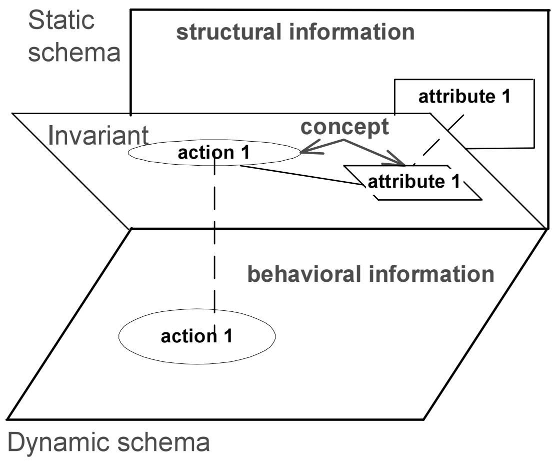


Figure 3. Relationship between invariant, static and dynamic schema

[2] "Schema" is a concept defined in part 3 of RM-ODP and takes its roots in "Z" [14]
[3] Note that objects are called "information objects" in part 3 of RM-ODP, section 6.

These schemas are illustrated in Fig. 3. The invariant schema can be interpreted as the mapping between the behavioral information and the structural information (i.e. state). This property is explained by the object nature, exhibiting dually its state and its behavior. By representing both structural and behavioral information in the invariant, the developer can make a more precise model. In particular, she can specify in what context things exist or are referenced. For example, in Fig. 3, "attribute 1" is referenced during "action1" execution. This will be further illustrated in Section 4.

Note that concept of "being always true" (present in the definition of invariant) has an implicit reference to a context. An invariant is always true in the context in which it is defined. Such context is typically the lifetime of an information object (as said in [clause 9.22]).

After having specified the object, we need to specify the actions. An action can be defined by the concepts of:

- "**pre-condition**: a predicate that a specification requires to be true for an action to occur." [clause 9.23]
- "**post-condition**: a predicate that a specification requires to be true immediately after the occurrence of an action." [clause 9.24]
- "**invariant**: a predicate that a specification requires to be true for the entire lifetime of a set of objects." [clause 9.22]

Although correct, these definitions could be improved to make more explicit the context in which they are applied. The first fix could be to change the concept "specification" into "action specification" or "contract" in order to make explicit their applications on actions (only actions allow us to specify in a single construct something at two points in time - before the action and after the action). Note that RM-ODP defines "contract" as "an agreement governing part of the collective behavior of a set of objects" [clause 11.2.1]. The second change can be made on improving the definition of the relationship between the pre- and the post-conditions. We have discussed the fact that system modeling cannot be done independently from the developer's goal. So when a developer defines an action specification, she defines what is the result of an action (post-condition) depending on the context in which it occurs (pre-condition). So the pre-condition should rather be considered as a condition for the post-condition to be true (and not necessarily as the condition for the action to occur). Note that the definition of invariant [clause 9.22] is redundant with the definition of invariant schema [part 3, clause 6.1.1]. One could be omitted.

While modeling actions, we found that policies play a very important role in having a complete action specification. We define:

- "**policy**": predicate that states conditions valid at specific moments of time during an action occurrence.

This definition is in agreement with the RM-ODP definition "Policy: A set of rules related to a particular purpose" [clause 11.2.7]. A policy for an action is essentially a constraint of any kind that is relevant with regard to the action. Policies can be used to make explicit the design goals and design choices for action refinements. For example, a policy for the operation of a software application might state that at some point in time its user will have to key-in sequentially several identifiers (i.e. the policy in the normal course of events for the application execution) or another policy might specify that if an identifier is incorrect, the application should ask the user to enter a new identifier (i.e. the policy on an alternative course of events). The composition constraints that were considered previously in this Section are another example of a policy. These new definitions of pre-condition, policies and post-condition are very close to the ones recommended by Alan Wills in [22]. The only difference is that in the action contract we propose to use policies instead of exceptions (policy is a more general concept that encompasses the exception).

It is interesting to describe how policies help in the definition of contracts for interactions, - a problem that is not yet solved [19]. A client interaction modifies the state of the environment during its execution. At the interaction completion, the values of the attributes of the object performing the client interaction are not changed. The post-conditions can only state that the action has occurred and that the object has not changed the values of its attributes. However, the policy states that during the interaction, information will be transferred to the environment. For a server interaction, the corresponding policy states that information comes from the environment and the post-condition states that the attribute values have changed. In summary, the introduction of policies allows us to have an elegant solution that keeps post-condition free from defining state changes in other objects than the one of interest.

3.4 Contribution Overview

Table 1 presents an overview of the concepts presented in this paper. All new and refined concept definitions are compatible with the current RM-ODP definitions. Our work mostly consisted in deriving useful details from RM-ODP definitions. This allows for its use as an ontology for definition of modeling languages applied in the context of system modeling.

The proposed ontology has been tested in two large case studies including multiple organizational levels (market, company, application, programming language classes). It is now used extensively to define our development process and structure our development tools. In addition, we validated the ontology by making a formal model in Alloy [11] of the basic interpretation concepts and the basic modeling concepts [18].

Table 1. Concept Overview

<u>**Basic interpretation concepts**</u>		**Section**
universe of discourse		3.1
entity		3.1
proposition		3.1
system		3.1
sub-system		3.1
supra-system	new concept	3.1
model	refined definition	3.1
model element	refined definition	3.1
quality	new concept	3.1
		3.1
sub-model	new concept	3.1

<u>**Basic modeling concepts**</u>

object		3.2
environment		3.2
information		3.2
behavioral information	new concept	3.2
structural information	new concept	3.2
behavior		3.2
behavioral constraint		3.2
action		3.2
internal action		3.2
client interaction	refined definition	3.2
server interaction	refined definition	3.2
state		
structural constraint	new concept	3.2
structural information element	new concept	3.2
attribute	new concept	3.2
parameter	new concept	3.2
role	new classification	3.2
interface		3.2

<u>**Specification concepts**</u>

instance		3.3.1
type		3.3.1
class		3.3.1
template		3.3.1

refinement		3.3.2
abstraction	new classification	3.3.2
composition	refined definition	3.3.2
decomposition	refined definition	3.3.2
component	refined definition	3.3.2
composite	refined definition	3.3.2
composition constraint	new concept	3.3.2

invariant schema (object, environment)	refined definition	3.3.3
static schema (object, environment)	refined definition	3.3.3
dynamic schema (object, environment)	refined definition	3.3.3
invariant	removed	3.3.3
pre-condition (action)	refined definition	3.3.3
post-condition (action)	refined definition	3.3.3
policy (action)	new use	3.3.3

4. Application

After having presented the RM-ODP concepts at a rather abstract level, we now illustrate these concepts by working through a more tangible example: a piece of Java code. Even if the example is quite pragmatic, all presented concepts are applicable at entities belonging to any organizational level. The same concepts can be used to model a supply chain, an IT system architecture or software components.

The notation used is not UML but is inspired by UML. The notational elements are similar. The major difference consists in the fact that we put different kinds of UML diagrams into one view. This allows relating the notational elements between the "diagrams".

4.1. Example Introduction

Let us consider a Java application that consists of a window ("Frame1") with a button ("button1"). Exhibit 1 illustrates the application code.

```
public class Frame1 extends Form
{      int i;
       X   x;
       Button button1 = new Button;

       public Frame1() // Constructor
       {      super();
              this.x = new X();    }

       private void button1_click()
       {      this.i = this.x.getA(); }
public class X
   {   int a;
       X() // Constructor
       {      this.a = 1;   }
       public int getA()
       {      return (this.a);    }
   } // X
} // Frame1
```

Exhibit 1. Java code example: window with button

When a user clicks on the button, the method "button1_click()" is invoked. This method performs the assignment "this.i = this.x.getA()". Let us consider what is happening while the assignment is executed. As we see in the code, an object of type "Frame1" (let's assume that it is identified as "f") is composed of several parts. It includes an object[4] instance of type "int" that is referenced as "i" within "f". The instance is identified as

[4] We use the words "object" and "type" correspondingly to the RM-ODP definitions. In java there is a slight difference, namely "an object is a class instance or an array" [3], which doesn't include an instance of int that is defined as a primitive type. The int should be wrapped either in the Integer or in an array to be instantiated as a real java object.

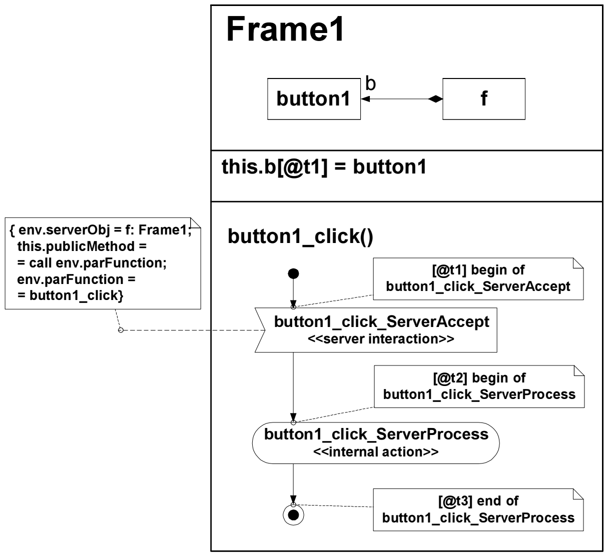
"i1" and is automatically created in the "Frame1" constructor.

In addition, it includes an object instance of type "X" that is referenced as "x". The instance is identified as "x1" and is explicitly created by the statement "this.x = new X()". These parts are initialized during the construction of "f", which means that within the method "button1_click()" we are referencing already existing objects "i1" and "x1".

We present the component object specification followed by the composite object specification of "Frame1".

4.2. Example of a Component Object Specification

The Fig. 4 represents the component object "f" of the type "Frame1". Note that "f" is supposed to be a component of a larger system that is not represented here.



Frame1
button1
b
f
this.b[@t1] = button1
button1_click()
{ env.serverObj = f: Frame1; this.publicMethod = = call env.parFunction; env.parFunction = = button1_click}
[@t1] begin of button1_click_ServerAccept
button1_click_ServerAccept <<server interaction>>
[@t2] begin of button1_click_ServerProcess
button1_click_ServerProcess <<internal action>>
[@t3] end of button1_click_ServerProcess

Figure 4. Example of ODP-based UML compatible graphical notation: Frame1 external representation

It is interesting to describe the way the object "f" is specified (illustration of Section 3.3.3):

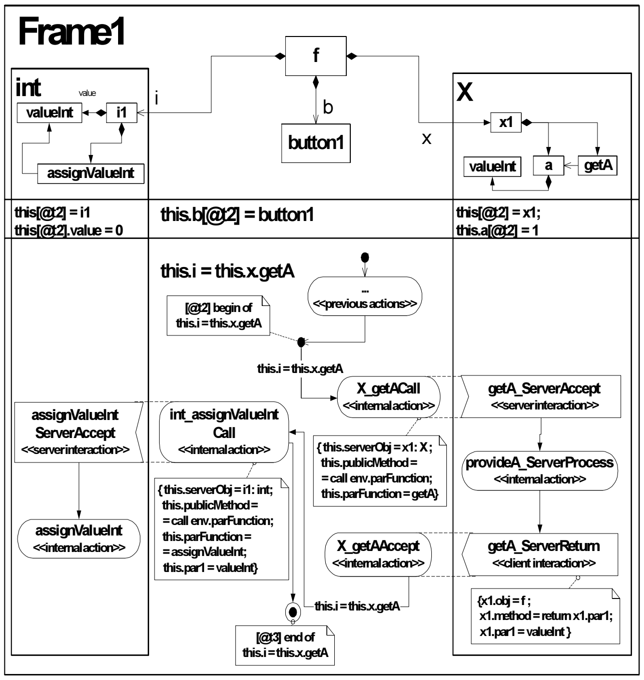
The upper pane represents the invariant schema of an object. The invariant shows that, within the object "f", a button exists. Only the button is represented, as the other objects are not visible from outside the object "f".

The middle pane corresponds to the static schema of an object containing the structural part of the invariant information. It states that at time [@t1] (i.e. immediately before the "button1_click" action) "f" object refers to its "button1" object as "this.b". Note that "this" is a keyword representing the object "f". The static schema can of course only be represented for specific moments in time that must exist within the corresponding object lifecycle.

The lower pane represents the dynamic schema containing the behavioral part of the object information. The dynamic schema represents a certain part of the object behavior that it exhibits during its lifecycle. The behavioral part presents that "f" accepts the button click from the environment (by executing a server interaction) and then executes the corresponding server processing. Note the comment outside the object box "f"; it represents the parameter value coming from the environment.

4.3. Example of a Composite Object Specification

The Fig. 5 presents the same object "f" but as a composite object. As presented in Section 3.3, we not only consider the composite object as a refinement of the component object but also as a different representation of the same part of the universe of discourse.



Frame1
f
int
value
valueInt
i1
i
b
button1
x
X
x1
valueInt
a
getA
assignValueInt
this[@t2] = i1
this[@t2].value = 0
this.b[@t2] = button1
this[@t2] = x1;
this.a[@t2] = 1
this.i = this.x.getA
...
<<previous actions>>
[@t2] begin of this.i = this.x.getA
this.i = this.x.getA
X_getACall <<internal action>>
getA_ServerAccept <<server interaction>>
assignValueInt ServerAccept <<server interaction>>
int_assignValueInt Call <<internal action>>
{ this.serverObj = x1: X ; this.publicMethod = = call env.parFunction; this.parFunction = getA}
provideA_ServerProcess <<internal action>>
{ this.serverObj = i1: int; this.publicMethod = = call env.parFunction; this.parFunction = = assignValueInt; this.par1 = valueInt}
assignValueInt <<internal action>>
X_getAAccept <<internal action>>
getA_ServerReturn <<client interaction>>
this.i = this.x.getA
{x1.obj = f ; x1.method = return x1.par1; x1.par1 = valueInt }
[@t3] end of this.i = this.x.getA

Figure 5. Example of ODP-based UML compatible graphical notation: Frame1 internal representation

It is interesting to compare the representation of the component object and the one of the corresponding composite object. We can see two object boxes corresponding to the "i1" and "x1" objects inside the object box representing the composite object "f". Note that all model elements shown in "f" and not in "x1" or "i1" correspond to the composition constraints presented in Section 3.3.2.

All the objects (including "f") are defined with the three panes (invariant, static, dynamic). Note that the elements shown inside the "i1" (respectively "x1") object box represent the sub-model of "i1" (respectively "x1").

Considering "i1" and "x1", we see that as they are declared independently, the object "i1" exists inside the context of "f" and doesn't have any relation with the object "x1". Analogously, "x1" doesn't have any relation with "i1". So, because of this independence, "i1" is not able to have a direct communication with "x1". Nevertheless both "i1" and "x1" exist within the same object "f"; so communicating with "f" they can transmit information to each other under the condition that "f" is fulfilling the corresponding composition constraints defined in Section 3.3.2.

Within the method "button1_click()", which belongs to object "f" (and not to the "button1"), "f" performs assignment "this.i = this.x.getA()". Here it is intended to assign ("=") a value to its "i1" object. The value that it will assign to "i1" should be further found within the "x1" object by calling its "getA()" method. This equation expresses the composition constraint for the assignment. Knowing this constraint (as part of the code of "f"), "f" performs an internal action to execute the "getA()" method of its object "x1". From the point of view of "x1", this internal action is perceived as a server interaction coming from "x1" environment (i.e. from "f"). This is an illustration of the Constructivist approach presented in Section 2: each object has a different perception of the same action occurrence. It also illustrates the concept of server interaction presented in Section 3.2. The comment attached to the server interaction illustrates the passing of the structural parameters (which are essentially values such as "f.par1= valueInt"), and of the behavioral parameters (which are essentially actions to be made with values such as: "f.parFunction = assignValueInt").

The object "x1" executes the processing associated with the requested internal action and executes a client interaction returning the parameter "valueInt". This value represents the integer value found in the "a" attribute of the "x1" object. The object "f" (i.e. the environment of "x1") perceives this sever interaction as being another internal action. Now, having received the resulting parameter from its "x1" and having the "this.i = this.x.getA()" composition constraint as an instruction for what needs to be done with "valueInt", "f" object executes yet another internal action to assign the value of the "i1" object to the received "valueInt". This internal action is perceived by "i1" as a server interaction with "valueInt" as parameter. And now it is "i1" who performs the local action assigning its "valueInt" to the received parameter value.

5. Impact

In this paper, we present an ontology that defines the concepts necessary for realizing object-oriented models and we illustrate the use of our ontology in a graphical model. In this Section, we detail the kind of effects that this ontology can have on the development environment. The development environment is defined as the methods, the tools and notations used in the context of a development project that requires the modeling of complex systems.

The development environment should be able to manage multiple sub-models and the relationship between the model elements found in the different sub-models. We define a sub-model as being the part of the model describing one system of interest. An example of the application of this principle can be illustrated by the modeling of a sale transaction. Given a "seller" object and a "buyer" object being components of a "market" composite object. The "sale" action occurrence belongs to the "market" object model. The "sell" action occurrence belongs to the "seller" object sub-model. The "buy" action occurrence belongs to the "buyer" object sub-model. Of course all three occurrences represent their parts of the same thing happening in reality. This example demonstrates the basic principle of multiple viewpoints on a same subject matter that can be found in Constructivism and is supported by RM-ODP.

In modeling it is quite frequent to have dual information. The development environment should be able to deal with dualities. For example, as illustrated in Section 1 and 2, state and behavior are dual. Sometimes, developers consider dual information as being redundant and their goal in this case is to avoid this redundancy. Based on our experience, we claim that this is not a redundancy but essential information that is important to be able to understand the models. The tools used in system development should manage this duality automatically.

Section 4 illustrates the duality between structural and behavioral information (visible in the invariant schema). Another example can be given in the context of the "market" object. The "seller" object can be considered as a component object when the developer wants to specify the market. The "seller object" can also be considered as a composite object if the developer is interested in documenting the business processes taking place within the "seller" object. The representations of an object as a component (of a larger system) or as a composite (showing the object parts) should be considered as two representations, which are dual to each other. The tools should allow the developer to toggle from one representation to the other.

6. Conclusion

This paper relates to the methods, tools and notations used for the development of business and software systems. We use the Catalysis method, UML notation and existing commercial tools both for developing such systems (in collaboration with our industrial partners) and for teaching object-oriented developments. In both cases, we experience difficulties that can be related to the fact that the notation, the method, the tools, and the developers have a different understanding of what object-oriented modeling means. In this paper, we propose an ontology, based on an international standard, that defines the fundamental concepts needed for object-oriented modeling. This ontology is based on RM-ODP, a telecommunication standard. We also use Constructivism and System Theory to interpret this ontology. Our concrete contributions consists in (1) making explicit the relationships between the various sections of RM-ODP part 2, (2) in introducing the concepts of: structural information element, structural constraint, composition constraint, client interaction, server interaction, and (3) in defining what is found in an invariant (i.e. structural and behavioral elements), by explaining the role of policies in the action specification. By defining the above concepts, the mapping of RM-ODP to existing methods and notation is drastically simplified. By understanding this ontology, the developer can understand how to interpret the methods, and notations and can configure the tools to support the development of systems in a more integrated way. If method designers, modeling language designers, and tool designers adopt this ontology, then the development environment could become significantly more productive. Early indications of this can be seen in our experience. Teaching object-oriented methods to undergraduates has been considerably simplified since we based our method and our interpretation of UML on our ontology. Our tools have become significantly more usable since we captured the relationship between the UML artifacts using our ontology.

7. Acknowledgements

Guy Genilloud (FSArch, Neyruz, Switzerland) introduced us to RM-ODP and Catalysis. John Donaldson (Compaq Professional Services, Geneva, Switzerland) and Fréderic Bouchet (Nortel Networks, Geneva, Switzerland) contributed to the identification of the modeling issues in business and information systems. William Frank (FSArch, Jersey City, USA), Eric Schwarz, and René Berger helped us to realize the importance of the philosophical and fundamental aspects hidden behind system design. Otto Preiss (ABB, Baden, Switzerland) contributed with his valuable reviews and comments.

8. Bibliography

[1]. R. Audi (Editor). *The Cambridge Dictionary of Philosophy*, 2nd edition. Cambridge University Press, September 1999, isbn 0-521-63722-8.

[2]. L. v. Bertalanffy. *General system theory: foundations, development, applications*. George Braziller, New York, 1969, isbn 0-8076-0453-4.

[3]. G. Bracha, J. Gosling, B. Joy, G. Steele. *The Java Language Specification*, Second Edition. Addison Wesley, June 2000, isbn 0-201-31008-2.

[4] M. Bunge, *Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World.* D. Reidel Publishing Co., Inc., New York, NY,1977.

[5] M. Bunge, *Treatise on Basic Philosophy: Vol. 4: Ontology II: A World of Systems*. D. Reidel Publishing Co., Inc., New York, NY, 1979.

[6]. D.F. D'Souza, A.C. Wills. *Objects, Components, and Frameworks with UML: the Catalysis Approach.* Addison-Wesley, 1999, isbn 0-201-31012-0, www.catalysis.org.

[7] D. Durand, *La systémique*, Presse Universitaire de France, 1998, isbn 2-13-044622-1

[8]. G. Genilloud, A. Wegmann, On Types, Instances, and Classes in UML, *Proceedings of ECOOP'2000, Workshop on Defining Precise Semantics for UML*, Sophia Antipolis, Cannes, France, June 2000.

[9]. G. Genilloud, A. Wegmann, A Foundation for the Concept of Role in the RM-ODP, *Proceedings of 4th International Enterprise Distributed Object Computing Conference (EDOC 2000)*, Makuhari, Japan, September 2000.

[10]. ISO/IEC 10746-1, 2, 3, 4 | ITU-T Recommendation X.901, X.902, X.903, X.904. *Open Distributed Processing - Reference Model.* OMG, 1995-96, http://isotc.iso.ch/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm.

[11] D. Jackson, *Alloy: A Lightweight Object Modelling Notation, Technical Report 797*, MIT Laboratory for Computer Science, Cambridge, MA, February 2000, http://sdg.lcs.mit.edu/~dnj/

[12]. J. L. Le Moigne. *Le Constructivisme, T. I "Les Fondements".* Ed. ESF, coll. Communication et Complexité, 1994, isbn 2-7101-1079-2.

[13] L. Lorippo, M. Faci, M. Haj-Hussein. *An Introduction to LOTOS : Learning by Examples*. Computer Networks and ISDN Systems, 23: 325-342, 1992

[14] J. McDermid, *Software Engineer's Reference Book*, Butterworth Heinemann, 1993, isbn 0-7506-0813-7.

[15] D. McGuinness. "Conceptual Modeling for Distributed Ontology Environments." *Proceedings of ICCS 2000*. Darmstadt, Germany, August 2000.

[16] J. G. Miller, *Living Systems*, University Press of Colorado, 1995, isbn 0-87081-363-3

[17] A. Naumenko, A. Wegmann. *Abstraction in Relations Between Artifacts Used in Software Development Processes. EPFL-DSC Technical Report*, April 2001.

[18] A. Naumenko, A. Wegmann, G. Genilloud, W. F. Frank. Proposal for a formal foundation of RM-ODP concepts. *Proceedings of ICEIS WOODPECKER – 2001*, Setubal, Portugal, July 2001.

[19] OMG. *Unified Modeling Language Specification.* Version 1.3, June 1999, http://www.omg.org/uml.

[20] H. Smith. *Frequently Asked Questions. Ontology.Org*, 1998-2001, http://www.ontology.org/main/papers/faq.html

[21] Y. Wand, V. C. Storey, R. Weber. An ontological analysis of the relationship construct in conceptual modeling. *ACM Transactions on Database Systems*, Volume 24, Issue 4 (1999), pp 494-528.

[22] A. Wills. Modeling Traits for e-Commerce, *ACW tutorial, OOPSLA 2000*, Minneapolis, USA, October 2000.