From RM-ODP to the formal behavior representation

Pavel Balabko, Alain Wegmann
Institute for computer Communications and Applications
Swiss Federal Institute of Technology – Lausanne
EPFL-DSC-ICA
CH-1015 Lausanne, Switzerland
{pavel.balabko,alain.wegmann}@epfl.ch

Abstract. In this work we consider the behavioral aspects of system modeling. In order to specify the behavior of a system, many different notations can be used. Quite often different terms in these notations are related with to same element in a system implementation. In order to relate these terms and guarantee the consistency between different notations some standard can be used. In this work we show how the Reference Model for Open Distributed Processing (RM-ODP) can be used for the purpose of the mapping of terms from different behavioral notations. In particular, we show the correspondence between terms in UML activity diagrams, UML state diagrams and Finite State Automata by means of relating them with RM-ODP terms. This allows us to consider RM-ODP as a possible meta-model for behavior specifications written in UML, which help to insure the consistency of UML models.

1 Introduction

Behavior models play a central role in system specifications. Correctly specified behavior of a system helps to avoid potential problems that might be discovered during costly testing and debugging phases. One of the means to specify the behavior of a system is to use formal methods. "Formal[1] methods are used to reveal ambiguity, incompleteness, and inconsistency in a system." [Wing90]. To use formal methods is not always easy. It requires a good knowledge of a formal notation and an ability to correctly relate formal specification terms with an implementation (or semantic domain). That is why developers often use semi-formal languages while developing systems. Such specifications are mapped more easily with their implementations.

J. Wing in [Wing90] defines formal specification language as a triple <Syn, Sem, Sat>, where Syn is called a language syntactic domain; Sem is the language semantic domain, and Sat is "satisfies relation". Satisfies relation defines the relation between syntactic terms and their semantic meanings in a semantic domain (see figure 1). Several syntactic domains can be used to describe the same system. For example, UML can be used to describe structural and behavioral aspects of the system (using UML Class and Activity/State diagrams correspondingly) and Finite State Automata (FSA) can also be used to describe behavioral aspects (for formal verification of some properties). In this case several syntactic domains can describe the same element from a system implementation. In figure 1, for example, we can see that the system element a1 (in semantic domain) is related with the concept Action in all syntactic domains.

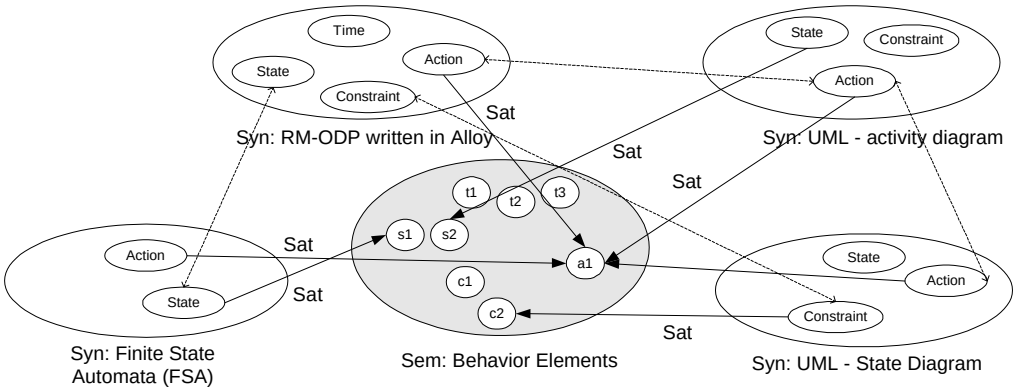


Figure 1 Formal methods: Relation between syntactic and semantic domains (Sat stands for satisfies relation).

If several syntactic domains are used (formal, as well as informal), a modeler must guarantee some form of consistency between these syntactic domains. In order to do this, the system specification can be written according to some standard that can relate different terms in several syntactic domains.
In this work we will use part 2 of the Open Distributed Processing - Reference Model RM-ODP [ISO96], which defines a generic syntactic domain for specifying distributed systems and explains its relation with a semantic

[1] A method is formal if its rules for manipulation are based on form (syntax) and not on content (semantics) [Holloway95].

domain. We will also use the formal model of RM-ODP written in the Alloy language[2]. This formal model is explained in [Naumenko01], where Naumenko shows the classification of RM-ODP concepts with the aid of the set theory and using regular predicate logic.

In this work we will consider only the behavioral aspects of different formal and semi-formal languages. In order to specify the behavior of a system, many different notations (syntactic domains) can be used (like Finite State Automata (FSA), Labeled Transition Systems (LTS), etc). Quite often different terms in these notations are related to the same element in semantic domains. In order to relate these terms, we can use the RM-ODP model. So the goal of this work is to show how RM-ODP can be related with some existing formal and semi-formal notations. For that purpose we will:

- Identify the minimum set of modeling concept from RM-ODP for behavior modeling. We have to do it since RM-ODP covers many more modeling concepts necessary for the specification of behavior.
- Define more precisely some RM-ODP behavior modeling concepts (particularly behavioral constraints and time). We have to do it since RM-ODP does not define all modeling concepts precisely enough to relate them with other existing formal notations.
- Define a set of possible abstractions that can reduce the complexity of RM-ODP behavior specifications by means of hiding irrelevant details. This can bring us to other notations. We show how an RM-ODP specification is mapped to the specification written in other languages for behavior representation. In our work we use examples with UML related notations: FSA, UML State Diagram and UML Activity Diagram.

Our work helps to relate RM-ODP with other languages for behavior modeling. This can help to spread the RM-ODP standard that can serve for mapping different formal notations. Our work also helps in the understanding of semantic meanings of formal methods since it defines a set of basic modeling concepts needed for the behavior modeling.

The structure of the paper is the following. In section 2 we start with the analysis of RM-DOP modeling concepts used in behavior specifications. In section 3 we give precise definitions for constraints of sequentiality and non-determinism. In section 4 we propose the definition for the Under-Specified Action that can be used to simplify RM-OPD models. In section 5 we show examples of specifications with Under Specified Actions and we show how to relate RM-ODP models with FSA and UML State/Activity Diagrams. We finish this work with the conclusions in section 6.

2 RM-ODP as framework for the behavior modeling

In this section we consider the minimum set of modeling concepts necessary for behavior specification. There are a number of approaches for specifying the behavior of distributed systems coming from people with different background and considering different aspects of behavior. “However, they can almost all be described in terms of a single formal model. In this model we represent a concurrent system as a triple consisting of a set of states, a set of action and a set of behavior” [Lamport90]. 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 [Broy91] 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 occurring in state changes occurring in state sequences can be understood as abstract representations of actions” [Broy91]. Here in this work we will consider both of these approaches as abstraction of the more general approach based on RM-ODP. We consider the Alloy formal definition of this approach that expresses the duality of two mentioned approaches.

2.1 Formal RM-ODP syntax made in Alloy

As we said above will use the formalism of the RM-ODP model, written in Alloy [Naumenko01]. We mainly use concepts taken from the clause 8 *“Basic modeling concepts”* of the RM-ODP part 2. These concepts are: behavior, action, time, constraints and state. According to [Naumenko01] these concepts are essentially the first-order

2 “Alloy is a little language for describing structural properties. It offers declaration syntax compatible with graphical object models, and a set-based formula syntax powerful enough to express complex constraints” [Jackson00]. See also http://sdg.lcs.mit.edu/alloy/.

propositions about model elements. We will also use some concepts (type, instance, 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. So the whole RM-ODP model is partitioned in the following way (basic interpretation concepts are not considered in this work):

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

Wegmann [Wegmann01] 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 (or semantic domain in our case)". In our work we will mainly use concepts from *"Basic modeling concepts"*. They can be linked with "Specification concepts". In our work we allow our readers to choose freely the appropriate specification concept (like type or instance, for example) for any of the basic modeling concepts. For example, the concept of action can be understood both as an instance and as a type (it depends on the choice of our readers).

As we have already said, behavior is defined as a basic modeling concept (RM-ODP, part 2, clause 8.6):

Behavior (of an object): "*A collection of actions with a set of constraints on when they may occur"*,

That can be formally represented in the following way:

```
// part of Alloy state declaration
Behavior: BasicModellingConcepts
partition Action, BehavioralConstraint: static Behavior                          // Behavior= set of actions+
                                                                                          set of constraints
corresponding_constraint (~constrained_action) : Action -> BehavioralConstraint  // constraints are defined for
                                                                                    actions

def Behavior {
all b: Behavior |                                          // for any element b from Behavior set
                ((b in Action) &&                          // (b is an Action and
                ( some b.corresponding_constraint) &&      // b has a at least one constraint and
                ( b.corresponding_constraint in Behavior)) ||  // this constraint is a Behavior element) or
                ((b in BehavioralConstraint) &&            // (b is a Constraint &&
                ( some b.constrained_action) &&            // b has a at least one action &&
                ( b.constrained_action in Behavior))       // this action is a Behavior element)
}
```

The behavior definition uses two RM-ODP modeling concepts: action and constraints

Action: "*Something which happens"*.

RM-ODP does not give the precise definition of behavioral constraints. From the definition of behavior, we can only conclude that behavioral constraints are part of the system behavior and that they are associated with the actions (see the formal definition above).

To formalize the definition of action in Alloy we have to consider two other modeling concepts. They are time and state. We can see how these concepts are related with the concept of action by looking at their definitions.

The concept of time is a fundamental concept in modeling of distributed systems. Time is introduced in RM-ODP in the following way (RM-ODP, part 2, clause 8.10):

"Location in time: *An interval of arbitrary size in time at which action can occur."*

```
// part of Alloy state declaration
instant_begin : Action –>Time!        // each action has one time point when it starts
instant_end : Action –>Time!          // each action has one time point when it finishes
```

From the RM-ODP definition of state (see below) we can see that the concept of state is dual with the concept of action and these modeling concepts cannot be considered separately:

State (of an object) (RM-ODP, part 2, clause 8.7): *At a given instant in time, the condition of an object that determines the set of all sequences of actions in which the object can take part.*

This definition shows that state depends on time and is defined for an object for which it is specified. In this work we use some simplifications. Since in this paper we consider the behavior only for one object, we never make the object explicit on the following diagrams.

```
// part of Alloy state declaration
state-existence: Time! -> State_!                    // state depends on time
```

Let's also note that "the knowledge of state does not does not necessarily allow for the prediction of the sequence of actions, which will actually occur"(RM-ODP, part 2, clause 8.7). In order to specify the sequence of action we have to use behavioral constraints. Now using already defined concepts of Time and State we can give a formal Alloy definition of Action:

```
def Action{
  all a: Action                                          // for each action a
                | a.instant_end =! a.instant_end &&      // (the beginning =! the end) &&
                (a.instant_begin.state_existance !=      // (the state before the action is not equal to
                      a.instant_end.state_existance)  &&  //    the state after action) &&
                (a.participating_object in a.participant)  //(there exists at least one associated object)
}
```

Using above presented basic modeling concepts allows us to describe the model of a behavior. In the next subsection we show an example of such behavior model.

2.2 The example of an RM-ODP model

The following example of a model corresponding to the formal RM-ODP syntax (see figure 2) is built with the Alloy Constraint Analyzer[3]. This example is a result of a semantic analysis of the formal behavior meta-model that is presented in the previous subsection. The Alloy Constraint Analyzer checks the consistency of the meta-model, generates sample configuration and visualizes it. This configuration is a valid phrase in the RM-ODP-Alloy syntax. It shows set of actions {B1, B2}, set of states {S2, S3, S4, S5}, set of constraints {B0}, set of time events {T0, T1, T2, T3} and relations between model elements. We can see that each action has time events corresponding to the beginning and to the end of the action and that each time event is associated with a corresponding state of the system. This allows us to conclude that the RM-ODP-Alloy syntax shown above is correct.

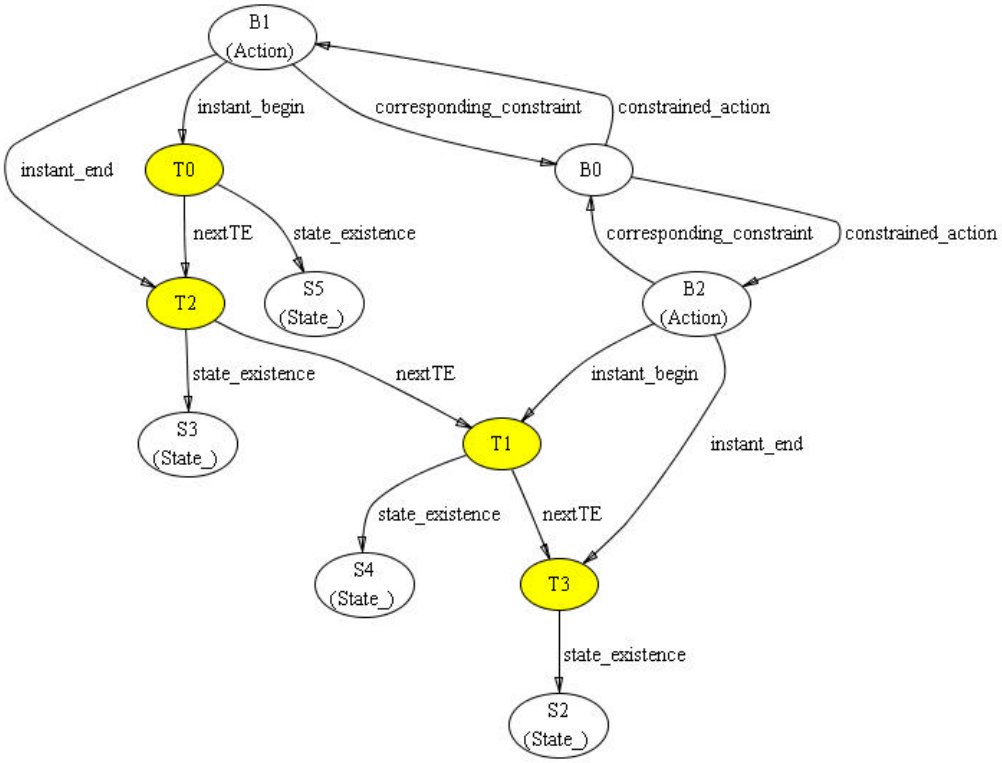


Figure 2 Example of a model build with Alloy Constraint Analyzer.

Further in our work we use similar examples based on formal RM-ODP syntax. In order to simplify reading of such diagrams and make them more compact we use a different notation. We will use ovals to represent actions, rounded rectangular to represent states. To represent time events we use small gray circles and to represent behavioral

[3] See http://sdg.lcs.mit.edu/alloy/

constraints we use stars. In our work we call diagrams built using this notation RM-ODP diagrams. Figure 3 shows an example of such diagram.

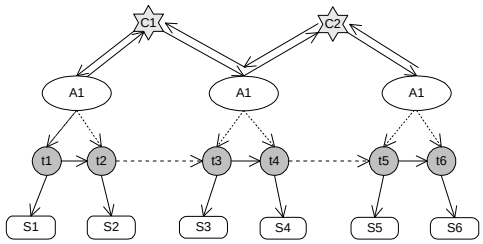


Figure 3 RM-ODP diagram.

As we can see the "pure" RM-ODP model is precise but quite bulky (it contains too many details) even if the behavior to be modeled is quite small. Fortunately, we can use a number of abstractions and simplifications to reduce the complexity of the model. Using simplifications can bring us to compact behavior models. Further in our work we show some simplifications that can bring us to some existing modeling techniques: FSM, State-charts and Activity-charts. Particularly, we show how we can simplify the modeling of time, behavioral constraints and actions.

3 Definition of time and behavioral constraints

Before considering possible simplifications of the RM-ODP model, we will define behavioral constraints more precisely. "Behavioral constraints may include for example: sequentiality, non-determinism, concurrency, real time" (RM-ODP, part 2, clause 8.6). In this work we consider constraints of sequentiality and non-determinism. The concept of constraints of sequentiality is related with the concept of time. Thus we start this section with the definition of time and then we return to the definition of behavioral constraints.

3.1 Time

Let's consider the role of time in system design. Time is fundamental in our universe. It plays two following important roles in system design:

- It serves for the purpose of synchronization. It allows for the synchronization of actions inside and between processes, the synchronization of a system with system users, the synchronization of user requirements with an actual performance of a system.
- It defines a sequence of events. It allows for the defining of the sequences of actions.

To fulfill the first goal, we have to be able to measure time intervals. However, Henri Poincaré shows in [Poincaré83] that a precise clock that can be used for time measurement does not exist in practice but only in theory. So the measurement of the time is always approximate. In this case we should not choose the most precise clocks, but ones that explain the investigated phenomena in the best way. "Simultaneity of two events or their sequentiality, equality of two durations should be defined in the way that the formulation of the physical laws is the easiest" [Poincaré83]. According to this idea we can choose different models of clocks. For example, for the actions synchronization, internal computer clocks can be used and, for the synchronization of user requirements, common clocks can be used that measures time in seconds, minutes and hours. The first role where the time can be measured in some relative units typically is not specified in formal behavior models. It is used in the modeling of non-functional requirements, which is a separate topic for research.

In our work we consider only the second role of time in system specifications. According to Poincaré we can build some special kind of clock that can be used for specifying sequences of actions. RM-ODP confirms this idea by saying that "a location in space or time is defined relative to some suitable coordinate system" (RM_ODP, part 2, clause 8.10). The time coordinate system defines a clock used for system modeling. In this work we consider a time coordinate system as a set of time events. Each event can be used to specify the beginning or end of an action. A time coordinate system must have the following fundamental properties:

- Time is always increasing. This means that time cannot have cycles.

- Time is always relative. Any time moment is defined in relation other time moments (next, previous or not related). This corresponds to the partial order defined for the set of time events.

We use the following formalization of time in Alloy: time is defined as a set of time events (Time). Time event has to be defined in relation with some other time events (partial order):

```
// part of Alloy state declaration
nextTE: Time -> Time            // defines the closest following time events for any time events
```

We will also use the followingTE relation to define the set of the following time events or transitive closure for the time event t over the nextTE relation:

```
// part of Alloy state declaration
followingTE: Time ->Time        // defines all possible following time events
```

Using followingTE we can define the following invariant that defines the transitive closure and guarantees that time event sequences do not have loops:

```
inv TImeInvariant {
        all t |                                                  // For all time events t
         ((no t.nextTE)->(no t.followingTE)) &&                  // if t does not have nextTE it also does not have followingTE
         ((some t.nextTE && no t.nextTE.followingTE)             // if t has the nextTE that does not have any followingTE
               ->(t.followingTE=t.nextTE )) &&                   //   then followingTE for t includes only nextTE for this t
         ((some t.nextTE && some t.nextTE.followingTE)           // if t has the nextTE that has some followingTE
              ->(t.followingTE=t.nextTE.followingTE + t.nextTE )) &&   //   then followingTE includes nextTE for t and followingTE for nextTE
         (t not in t.followingTE)                                // time does not have loops
}
```

This definition of time we will use in the next section when we will describe sequential constraints.

3.2 Behavioral constraints

Both approaches for behavior specification mentioned in section 2 (using the set of actions or the sequences of state changes) represent behavioral constraints implicitly. Quite often we can infer them from behavior representation, like behavior automata. For example, figure 4.b shows a specification that has constraints of sequentiality and non-determinism. We can infer that the system is specified using constraints of non-determinism by looking at state S1 that has a non-deterministic choice between two action a.

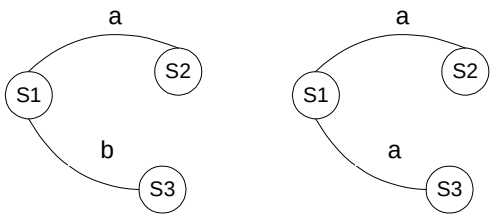


Figure 4 FSA, specification of the sequential system using: a - sequential deterministic constraints; b - sequential non-deterministic constraints.

We base our approach on RM-ODP, where behavioral constraints are represented explicitly ("Behaviour (of an object): A collection of actions with a set of constraints on when they may occur" [RM-ODP]). "The definition of Behavior should link a set of actions with the corresponding constraints" [AN]. In the following subsection we consider precise definition of constraints of sequentiality and non-determinism.

Constraints of sequentiality

We start with the analysis of constraints of sequentiality. Each constraint of sequentiality should have the following properties:

- It is defined between two or more actions.

- Sequentiality has to guarantee that one action is finished before the next one starts. Since RM-ODP uses the notion of time intervals[4] it means that we have to guarantee that one time interval follows the other one:

```
def SeqConstraints {
all sc: SeqConstraints |                                    // for all SeqConstraints sc
    some a1, a2: Action | (a1 != a2) &&                     // there are two different actions a1, a2
   (a1 in sc.constrained_action) && (a2 in sc.constrained_action) &&   // sc is defined between a1 and a2
   ( (a2.instant_begin in a1.instant_end.followingTE) ||    // a1 is before a2 or
   (a1.instant_begin in a2.instant_end.followingTE) )       // a2 is before a1
}
```

From this definition we can see that the concept of constraints of sequentiality is tightly related with the concept of time. Constraints of sequentiality should be consistent with the partial order defined in the set of time events. If this partial order is given, sequential constraints can be derived. And vice versa, if we have been given sequential constraints, the partial order in the set of time events can be derived. This allows us to use an abstraction of time while keeping the sequences of actions. We will consider this in detail in the next section. An example of sequence of actions is shown in the following picture:

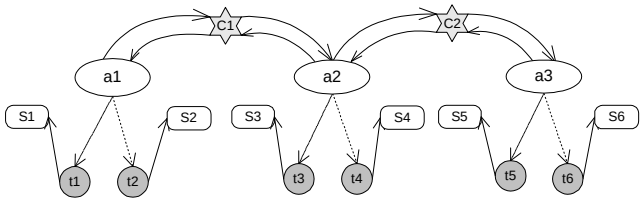


Figure 5 RM-ODP diagram: Sequence of Actions with sequentiality constraints.

Given that in figure 5, C1 and C2 are sequential constraints, we can conclude that the partial order on the set of time events is: t1->t2->t3->t4->t5->t6.

Constraints of non-determinism

In order to formalize constraints of non-determinism we considered the following definition given in [Broy91]: "A system is called non-deterministic if it is likely to have shown number of different behavior, where the choice of the behavior cannot be influenced by its environment". This means that constraints of non-determinism should be defined between a minimum of three actions. The first action should precede the two following actions and these actions should be internal (see figure 6).

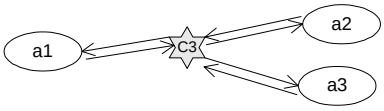


Figure 6 RM-ODP diagram: Example constraints of non-determinism

We can write this in Alloy in the following way:

```
def NonDetermConstraints {
all ndc: NonDetermConstraints |                                   // for all NonDetermConstraints
    some a1: Action |                                             // there is action a1
    some a2,a3: InternalAction | (a2 != a3)&&(a2 != a3)&& (a2 != a3) &&   // there are two different internal actions a2, a3
   (a1 in ndc.constrained_action) &&                              // sc is defined between a1
   (a2 in ndc.constrained_action) &&                              // and a2
   (a3 in ndc.constrained_action) &&                              // and a3
   (a2.instant_begin in a1.instant_end.followingTE) &&            // a1 is before a2 or
   (a3.instant_begin in a1.instant_end.followingTE)               // a1 is before a3
}
```

Let's not that, since the choice of the behavior should not be influenced by environment, actions a2 and a3 has to be internal[5] actions (not interactions). Otherwise the choice between actions would be the choice of environment.

[4] **Location in time:** "An interval of arbitrary size in time at which an action can occur" [RM-ODP].

4 Possible simplifications and where their can result

As we have seen in section 2, it is difficult to understand even a simple RM-ODP model due to its size. In order to simplify RM-ODP models we can use a set of abstractions that can be done by means of hiding details irrelevant to the purpose of modeling. In this section we will consider such a possible simplification, particularly the abstraction of time and a set of simplifications for action representation. We will also pay attention to some important information that can be lost during abstraction.

4.1 Under Specified Actions

Developers often do not use concrete time information and keep only constraints of sequentiality. Sometimes the presence of time information makes modeling correct, however "the incorporation of concrete timing properties leads to a considerable loss of abstractness" [Broy91]. For example, if several actions perform the same functionality with the only difference that they are defined for different time intervals. In that case we can define some predicate that will characterize this collection of actions. Since we have to define a predicate applied to modeling concepts, we have to use specification concepts (see section 2). Among specification concepts we can use pre- and post-conditions[6] in order to define a collection of actions with the same functionality. Using pre- and post-conditions we can define a type "Under-Specified Action " (USA). This type can be defined with the following predicate:

Under-Specified Action (USA): *"USA specifies actions with the same pre- and post-conditions".*

Using this type we can avoid specifying concrete timing properties, but in this case we have to slightly modify our model. We have seen that the concept of state is defined for each time event (see section 2). Since we make an abstraction of time, we have to relate the concept of state directly with the concept of action. We can do that because each action has only one instant_begin and one instant_end. Instead of defining a state related with instant_begin and one instant_end as it was done above we would define pre-state and post-state for each action:

```
// part of Alloy state declaration
prestate: Action! -> State_!                    // each action has one pre-state
poststate: Action! -> State_!                   // each action had one post-state
```

Let's suppose that in figure 5 actions A1 and A3 have the same pre- and post-conditions: pre-cond1 and post-cond1. In that case they will have the same type of USA: type1 (see figure 7):

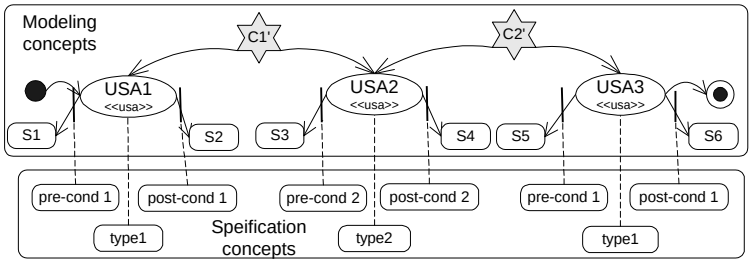


Figure 7 RM-ODP diagram: Behavior specification using Under-Specified Actions: Action structure.

Let's note that we have extended notation used for RM-ODP diagrams before. We included the symbol of pre-and post-conditions (vertical line crossing the association from an action to a state) and symbols of the beginning and the end of the action sequences. In order to make actions generic and independent of constraints of sequentiality, we keep only unidirectional links between action and constraints. It means that the sequence of actions is fully specified by constraints of sequentiality (assuming that pre-conditions are met). Lets also note, that in previous sections we left some freedom to the user to map basic modeling concepts with the specification concepts (with either instances or

---

[5] "The set of actions associated with an object is partitioned into **internal actions** and **interaction**s. 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." (RM-ODP, part 2, clause 8.3)

[6] RM-ODP gives the following definition for pre- and post-conditions (clauses 9.23 and 9.24):
Precondition: *A predicate that a specification requires to be true for an action to occur.*
Postcondition: A predicate that a specification requires to be true immediately after the occurrence of an action.

types, for example). But Under-Specified Action can be mapped only with type, since it uses the predicate to define the collection of actions. Here we fix our choice for actions to be types in models of behavior.

Behavior modeling with parameters

The next simplification we consider is the modeling of actions using parameters. UML defines parameters in the following way:

Parameter [OMG99] *"is an unbound variable that can be changed, passed, or returned. Parameters are used in the specification of operations, messages and events, templates, etc. In the meta-model, a Parameter is a declaration of an argument to be passed to, or returned from, an Operation, a Signal, etc."*

Let's see what parameter means in RM-ODP terms. Figure 8 shows a set of actions {a0, … aN} with a similar post-conditions (the difference is in the values assigned to the variable S). These post-conditions differ only in the value that is assigned to the state variable S. In the same way we have defined an Under-Specified Action type before, we can define a new Under-Specified Action type:

Under-Specified Action with parameters (USA): *USA with parameters specifies a collection actions with the same pre-conditions and with post-conditions that differ only in values assigned to system variables.*

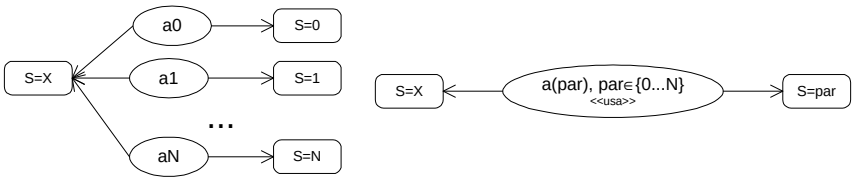


Figure 8 RM-ODP diagram: Action modeling with parameters

The action a(par), par∈{0...N} in figure 8 is the Under-Specified Action that characterize the set of actions {a0, … aN}.

4.2 Abstraction of the state information

In the previous subsection we have defined a type that characterizes the set of actions with the same pre- and post-conditions. Let's note that pre-conditions were defined on pre-states and post-conditions were defined on post-states of actions. If we continue our analysis we can see that there are many actions with different pre- and post-conditions (and pre- and post-states correspondingly) but still having very similar functionalities. We can see it if we look at the difference between pre- and post-states. Figure 9, for example shows that the post-conditions of both actions A1 and A2 specify the increment of state variable "a" by 1. This similarity allows us to extend the meaning of the under-specified action with the following predicate:

Under-Specified Action with the abstraction of states (USA): *USA with the abstraction of pre- and post-states specifies actions with post-conditions that can be expressed as a proposition applied to the action pre- and post-states.*

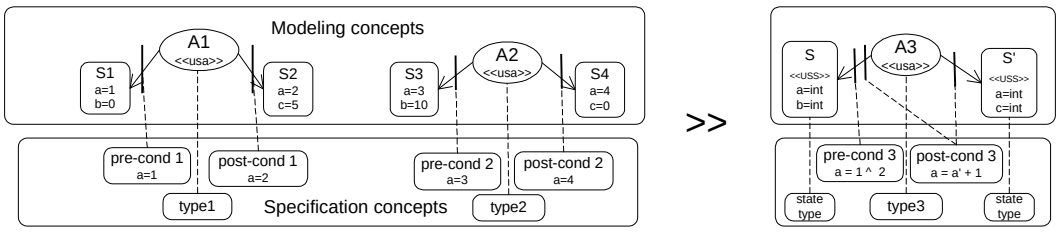


Figure 9 RM-ODP diagram: abstraction of the state information

Let's note, that in this case pre- and post-states of the under-specified action A3 become under-specified states (USS).

Under-Specified State (USS): *An under-specified pre-/post-state characterizes a collection of states for the corresponding under-specified action.*

State S, for example, characterizes a collection of states {S1, S3}. Strictly saying we have to specify state S with a={1, 3}. But typically developers use another simplification: “a:int”. This defines the type of the state variable. Using such simplifications we can see that such under-specified states would be the same for all actions and can be shown separately in the UML class diagram, for example.

5 Examples of specifications with Under-Specified Actions

In this section we show how to apply simplifications from the previous sections to an example of a behavior. Such simplifications can bring us to different existing notations for specifying system behavior. Here we show examples of UML Activity and State diagrams and with the Finite State Automata (FSA).

We start with the example (see figure 10) that specifies an object behavior with nine actions. Five of them are internal actions (they take place without the participation of the environment) and four of them are interactions (they take place with the participation of the environment of the object). Names of internal actions start with “a” and names of interactions start with “e”.

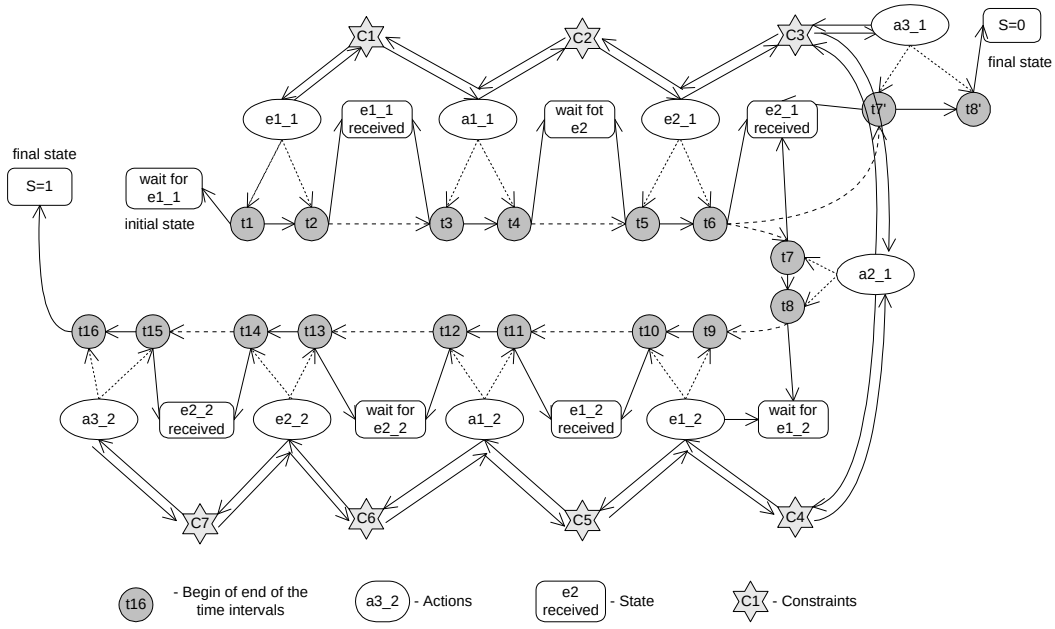


Figure 10 RM-ODP diagram: an example of a behavior.

The example also shows the system states before (pre-states) and after each action (post-states), constraints and time events. You can see that a post-state after each action is the same as a pre-state for the next action. However, in general, these pre- and post-states should be different, since some other concurrent process can change the state of a system between two actions. Here we suppose that in our system there are no concurrent processes and thus there is no other processes that can change the state of the system between two consecutive actions. Let’s also note here that actions a1_1 and a1_2; e1_1 and e1_2; e2_1 and e2_2 are similar. They perform the same functionality and the only difference between them is in the instant_begin and instant_end time points. Actions a3_1 and a3_2 also perform the same functionality not specified here, with the slight difference that a3_1 makes state variable S equal to 0, while a3_2 makes this variable equal to 1. Actions a3_1 and a3_2 also have different instant_begin and instant_end time points.

First, if we make an abstraction of time and state information in the example shown in figure 10, we get the following graphical representation:

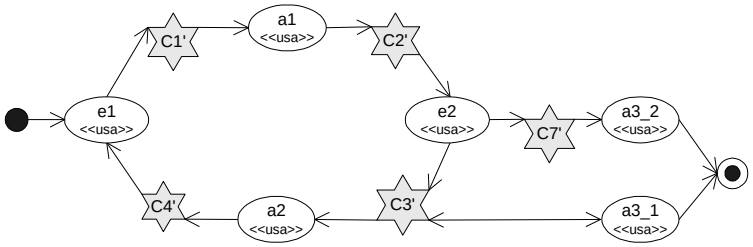
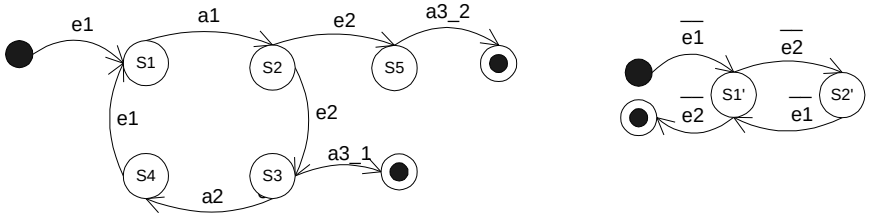


Figure 11 RM-ODP diagram: simplification of the example using time abstraction and hiding pre- and post-conditions.

Here actions a1, a2, e1 and e2 are Under-Specified Actions that characterize the following collections of action from figure 10: {1_1;a1_2}, {e1_1;e1_2}, {e2_1;e2_2}. Let's also note that constraints also become under-specified constraints in exactly the same way as actions. For example, the constraints C1' characterize the set of constraints {C1, C5} from figure 3. Now we can transform this behavior model into the Finite State Automata. It is transformed in the following way: any action becomes arc in FSM, constraints of sequentiality become states. Let's note that that just constraints of sequentiality become state in FSM but not pre- or post states. The fact is that some other concurrent process can change the state of a system between two actions. That means that pre-state of an action and post-state of consecutive action should be different. But constraints of sequentiality in RM-ODP define exactly the same meaning as states in FSA: their show all consecutive actions.

Here we have also to pay attention to the transforming of constraints of non-determinism. In order to express them in FSA we have to model action e2 twice. It shows that the system makes the internal choice between the two "branches" of behavior without being influenced by its environment. For a better illustration we also show the behavior model for the system environment. Interaction of object with its environment can be represented as a reaction between the following pairs of actions and their complements {e1,$\overline{e1}$} {e2,$\overline{e2}$}.



(a) **(b)**

Figure 12 FSM specification of the system (a) and its environment (b).

The further simplification (using modeling of actions with parameters) of our example from the figure 10 leads us to the behavior model shown in figure 13, where the post-conditions for action a3(p) is S = p. Let's note, that in all behavior models we considered above, interactions and internal actions are modeled using the same notation (the sign of oval). This allows us to define communications between systems as a reaction between an interaction a and its complement $\overline{a}$ [Milner99]. But UML uses slightly different notation that can also reduce the complexity of UML state and activity diagrams. The concept of interaction in RM-ODP corresponds to the concept of event in UML and the concept of internal action corresponds to the concept of action in UML:

"An event is something done to the object; an action is something that the object does" [Stevens00].

Event in UML is considered as an action trigger and modeled in the way it is shown on in figures 14 and 15.

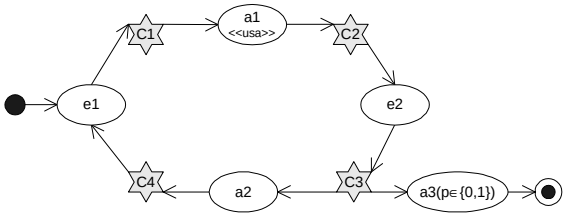


Figure 13 RM-ODP diagram: Simplification of the model using actions with parameters

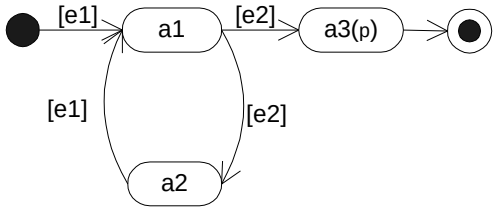


Figure 14 UML activity diagram.

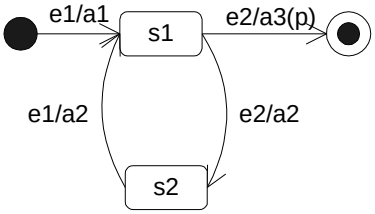


Figure 15 UML state diagram

6 Conclusion

In this work we analyzed the correspondence of RM-ODP models with the UML related behavior models by means of mapping corresponding concepts in their syntactic domains using abstraction of temporal and state information. We have seen that RM-ODP can serve as a mediator that explains the semantics of this mapping between different languages for behavior modeling. A behavior modeling case tool, that can specify different levels of abstraction and insure the consistency between these levels, can be built based on this work.

This work continues the work done by Naumenko [Naumenko01] that formalizes RM-ODP. The main contribution of this work is the precise definition of constraints of sequentiality and non-deterministic constraints and the definition of Under-Specified Actions and States that we propose to use in order to map modeling concepts used RM-ODP with concepts from other existing frameworks.

The proposed formalism still needs elaboration, however. For example, in our work we have modeled the constraints of sequentiality and non-determinism. But the modeling of constraints of concurrency and relating them with existing RM-ODP framework would be quite a large and interesting research direction.

7 References

[Broy91] 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.

[Harel97] *Harel, D. and E. Gery*, Executable object modeling with statecharts, in *IEEE Computer*. 1997. p. 31 -42.

[Holloway95] Holloway, C.M., P.S. Miner, and R.W. Butler**,** *An Informal Introduction to Formal Methods*, . May 1995, NASA Langley Research Center.

[ISO96] ISO/IEC 10746-1, 3,4 | ITU-T Recommendation X.902, *Open Distributed Processing - Basic Reference Model - Part 2: Foundations* **.** 1995-1996.

[Jackson00] Jackson, D., *Alloy: A Lightweight Object Modeling Notation*, Technical Report 797, 2000, MIT Laboratory for Computer Science: Cambridge, MA.

[Lamport90] 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.

[Milner99] Milner, R., *Communicating and Mobile Systems: the pi-Calculus*. 1999: Cambridge University Press.

[Naumenko01] 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.

[OMG99] OMG, *Unified Modeling Language Specification, v 1.3*, . 1999.

[Poincaré83] Henri Poincaré, *The value of science***,** Moscow «Science», 1983

[Stevens2000] Stevens, P. and R. Pooley, *Using UML Software Engineering with Objects and Components (Updated Edition)*. Object Technology Series. 2000.

[Wegmann01] 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, USA.

[Wing90] Wing, J.M., *A Specifier's Introduction to Formal Methods.* IEEE Computer, 1990. **23(9)**: p. 8-24.