Representing Dynamic Invariants in Ontologically Well-Founded Conceptual Models

Conference Paper · June 2015

CITATIONS
4

READS
655

2 authors:

John Guerson
Universidade Federal do Espírito Santo
7 PUBLICATIONS **29** CITATIONS
SEE PROFILE

João Paulo A. Almeida
Universidade Federal do Espírito Santo
148 PUBLICATIONS **1,549** CITATIONS
SEE PROFILE

Representing Dynamic Invariants in Ontologically Well-Founded Conceptual Models

John Guerson and João Paulo A. Almeida

Ontology and Conceptual Modeling Research Group (NEMO)
Federal University of Espírito Santo (UFES), Vitória ES, Brazil
{jguerson,jpalmeida}@inf.ufes.br

Abstract. Conceptual models often capture the invariant aspects of the phenomena we perceive. These invariants may be considered static when they refer to structures we perceive in phenomena *at a particular point in time* or dynamic/temporal when they refer to regularities *across different points in time*. While static invariants have received significant attention, dynamics enjoy marginal support in widely-employed techniques such as UML and OCL. This paper aims at addressing this gap by proposing a technique for the representation of dynamic invariants of subject domains in UML-based conceptual models. For that purpose, a temporal extension of OCL is proposed. It enriches the ontologically well-founded OntoUML profile and enables the expression of a variety of (arbitrary) temporal constraints. The extension is fully implemented in the tool for specification, verification and simulation of enriched OntoUML models.

Keywords: Conceptual Modeling, OntoUML, Temporal OCL

1 Introduction

In a broad perspective, conceptual modeling has been characterized as "the activity of formally describing some aspects of the physical and social world around us for purposes of understanding and communication" [20]. Many of the efforts in conceptual modeling attempt to represent a conceptualization about a given subject domain [15], which is often accomplished by capturing in a model the invariant aspects of the phenomena we perceive. These invariants may be considered *static* when they refer to structures we perceive in phenomena at a particular point in time or *dynamic* when they refer to regularities across different points in time.

Take for instance a domain about persons, their stages in life and their marriages. At a particular point in time, a number of persons will exist, each of which may be male or female, may be a child, a teenager or an adult, and may be related to someone else by marriage. The *static invariants* that may be represented in a conceptual model of this domain include the various categories of entities in a domain (in our example, "person", "man", "woman", "child", "teenager", "adult", "elder", "marriage") as well as their relations (a "child" is a "person", "marriage" may be established between two "persons", etc.). The *dynamic invariants* in turn reflect the fact that across different

points in time entities of the domain undergo change. In our example, persons are born and die, become teenagers and adults, marry, divorce, etc. Dynamic invariants represent what may change and what must remain constant in time. For example, children cannot suddenly become adults, adults cannot later in life become teenagers and elders cannot become children, teenagers or adults.

Much attention has been given to the representation of static invariants in a number of modeling notations including ER diagrams, ORM diagrams [16], and UML Class Diagrams [23]. The UML for example has been enriched with the Object Constraint Language (OCL) to capture invariant expressions [22]. With respect to the dynamic invariants, these have been mostly confined to the representation of pre- and post-conditions for operations or simple meta-attributes for features such as "read only" [23, p.125, 129]. Further, due to the strict correspondence that is often established between modeling languages and programming languages, many UML-based approaches lack support for dynamic classification (e.g. USE [13], HOL-OCL [5], UML2Alloy [1, 8]). While this facilitates the mapping to specific programming languages or formalisms, this renders these approaches less suitable to enable the expression of important conceptual structures that rely on dynamic classification (e.g., the classification of persons into life phases: child, teenager, and adult, the classification of persons into roles they play contingently such as husband and wife)[1].

In order to address the shortcomings of the UML and OCL specifications, many approaches have been proposed to extend UML and OCL with dynamic aspects. Some of these approaches address dynamic aspects as part of an overall approach to handle temporal/time aspects [3, 4, 6, 7, 9, 12, 15, 18, 19, 27]. The OntoUML [15], for example, introduces various dynamic aspects through stereotypes referring to meta-attributes of classes and properties such as rigidity and immutability. Similarly, [6] extends UML with stereotypes, augmenting it with dynamic notions of durability and frequency. Others have aimed at enriching OCL with extensions in order to cope with dynamic/temporal properties of systems. For example, some have extended OCL with Linear-Temporal Logic and Computational-Tree logic (LTL/CTL) operators [3, 7, 19, 27], created new logic formalisms [4, 9], extended OCL with temporal patterns [18], defined a Real-Time extension for OCL with a temporalized CTL [12], etc.

Despite these recent advances, a comprehensive approach to the representation of dynamic aspects in UML-based conceptual models is still lacking. This gap is addressed in this paper, in which we extend OCL with the capability to express rich dynamic invariants in OntoUML. Our approach is based on a reification of world states, with no specialized knowledge in tense logic required to use the approach.

This paper is structured as follows. Section 2 presents our running example, illustrating the various requirements for the representation of dynamics aspects. Section 3 introduces the proposed OCL extension. Section 4 revisits our running example to show how the approach meets the requirements. Section 5 discusses related work while Section 6 discusses the implementation and concluding remarks.

[1] Note that while dynamic classification is supported in principle by UML diagrams, this is not reflected in tool support and language usage, with little mention in the UML specification.

2 Requirements for the Representation of Dynamic Aspects

According to [6], the UML is a non-temporal conceptual modeling language. Thus, a UML class diagram represents the *actual* state of a system assuming that the "information base" contains only the current instances of classes and relationships. **Fig. 1** depicts a conceptual model in UML about a domain involving persons, their stages in life and marriages. UML multiplicities in the model define the allowable number of individuals to which a particular individual may be linked in any given state of the system. For example, a partner (husband or wife) can only be married to one other partner at a time. The model is silent with respect to the number of persons a person can marry in time, i.e., whether they may or may not divorce and remarry.

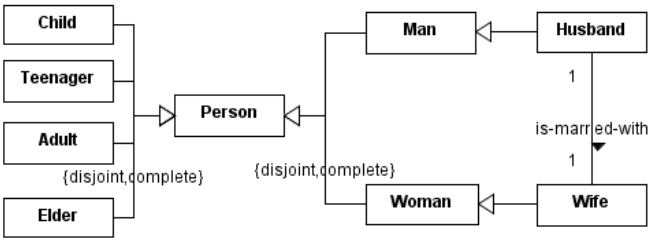

Fig. 1. UML structural conceptual model example

The model of **Fig. 2** revisits the model of **Fig. 1** employing the OntoUML profile. The profile uses class stereotypes to determine which ontological category from the UFO applies to each class [15]. This means that OntoUML can address some of dynamic aspects of this domain that are not addressed in plain UML. For example, the class Person is stereotyped as <<kind>>, meaning that it applies necessarily to its instances. Thus, a person cannot cease to be a person without ceasing to exist. This modal notion corresponds to what is called ***Rigidity*** in UFO. The consequence of rigidity in terms of dynamic aspects is that an individual of a rigid class instantiates this class throughout its life. A kind can be used in a taxonomic structure with rigid subtypes known as subkinds (e.g., Man and Woman).

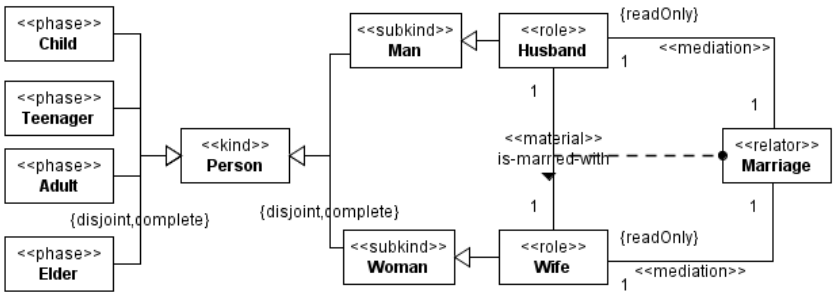

Fig. 2. OntoUML structural conceptual model example

Other examples of dynamic aspects expressed in **Fig. 2** include the stereotypes of <<roles>> and <<phase>>. Husband and Wife are stereotyped as <<role>> and Child, Teenager, Adult and Elder as <<phase>>. Roles and phases are both anti-rigid concepts (e.g. a wife can cease to be a wife without ceasing to exist). ***Anti-Rigidity*** states that a class C is anti-rigid iff for all its individuals, there will be a point in time *w* in which they exist but do not instantiate C, at *w*. The difference between the two is that the former defines contingent properties of an individual exhibited in a relational context (e.g. a person is a wife contingently and only in the context of a marriage) while the later through an intrinsic change of an individual's property (e.g. a child has the intrinsic property of being a child).

The class Marriage is stereotyped as <<relator>>. Relators can be viewed as objectified properties, as entities that "connect" other entities. They are the truthmakers of the so-called <<material>> relationships. For example, it is the existence of a particular marriage connecting man X and wife Y that makes true the relation *is-married-with*(X, Y). A derivation relationship on the other hand holds between a relator and a material relationship and exemplifies the truth-maker relation. Relators are rigid concepts and existentially dependent on the instances they connect through <<*mediation*>> relationships. A mediation is a dependency relationship that defines existential dependence from their source entity, e.g. Marriage, to their target entities, e.g., a Wife and a Husband. This means that a marriage only exists at some point in time, if wife and husband also exist at that point in time. A particular marriage then depends specifically on two "fixed" persons, i.e., the marriage between Bob and *Alice* cannot ever become the marriage between Bob and *Anna*. Mediations are thus always defined as *readOnly* at their target-side by default. From a logical point of view, this dynamic aspect of existential dependence can be viewed as a type of *immutability* (a marriage never changes their participating wife and husband). ***Immutability*** states that if an individual (e.g. marriage) exists at a point in time *w*, then at every subsequent time *w'* from *w* that the individual exists, that individual will have in *w'* the same property (e.g. same wife and husband) as it had in *w*. Finally, the classes Husband and Wife are related through exactly 1 Marriage, meaning that we represent monogamous heterosexual marriage, i.e., a partner can only be married to one partner at a time.

While some distinctions in OntoUML enable the representation of dynamic aspects of the domain, a number of other dynamic invariants cannot be expressed:

1. A person is created in the child phase.
2. An adult cannot become a child or teenager (a teenager cannot became a child).
3. An elder is the last phase of a person (it cannot become adult, teenager or child).
4. A person should eventually cease to exist at some point.

Constraints *1-3* can be viewed as more general behaviors about *classifications* or *transitions* of individuals. We named these as ***Initial***, ***General and Final Transition*** dynamic aspects, respectively. Constraint *4* in turn can be viewed as more general behavior about the *existence* of individuals. *Transient* means lasting only for a determined time. We named this as a ***Transient Existence*** dynamic aspect.

In addition to the constraints specified above, the OntoUML model still does not represent how many times a person can marry throughout his/her life, and the model

is silent about this aspect of the conceptualization. In this model, a marriage could still be transient (when it ceases to exist eventually) or permanent (when it never ceases to exist once created). The permanence view of marriages could refer, for example, to a religious conceptualization, where marriages are divine "contracts" between two people and cannot be undone. If committed to this view, a desired dynamic invariant would be:

5. If a marriage is established then it can no longer be destroyed.

We named this as a ***Permanent Existence*** dynamic aspect. On the other hand, if a marriage is transient and ceases to exist, it may be desirable in a given conceptualization to refer to ex-spouses, i.e., people who participated in a past marriage, which no longer exists in the present:

6. A person will only be an ex-husband or ex-wife if he/she was a husband/wife in a marriage in the past which no longer exists.

This is a common dynamic invariant that has been called ***Derivation by Past Specialization*** in [21].

The kinds of limitations in the expressiveness of diagrammatic languages (e.g. OntoUML, UML) we identify here are often an explicit language design decision, in order to manage the complexity of graphical representation. In order to complement the graphical representation and address the expressiveness needs, textual representations such as OCL have been proposed to enrich the model as captured in a diagrammatic language. We also take that approach and employ OCL in order to enrich OntoUML in other to support the representation of dynamic aspects of the domain.

The modeling approach is required to:

- support dynamic classification (i.e., allow for individuals to change types throughout their existence);
- enable the expression of modal constraints on types (e.g., rigidity, anti-rigidity);
- enable the expression of transition rules (constraining the order in which individuals instantiate types), transient/permanent existence and past specializations; and
- finally, enable the expression of arbitrary dynamic invariants, i.e., *invariants whose satisfaction is determined by examining the world at more than one point in time*.

Transition rules include: (i) initial type rules (determining the type that is instantiated necessarily upon creation); (ii) final type rules (determining the types that once instantiated, classify the individual); (iii) other general transition rules (constraining arbitrary order of instantiation). Transition rules may be expressed by behavior models such as with UML state chart diagrams. However, we are aiming here for a general approach to define *arbitrary* types of dynamic constraints. Further, the modeling approach must not rely on operations of classes, as these are not employed by OntoUML [15], and also not employ specialized tense/temporal logic-based operators as [3, 7, 19, 27], in order to retain its ease of use for UML/OCL modelers.

3 A Standard OCL Temporal Extension

A standard OCL invariant is a static condition that should hold for each single state of the model's instances. As a consequence, the so-called "context" of a standard invariant is a single state, and no notion of "state" is manipulated in standard OCL invariants. In order to enable the manipulation of states and consequently the representation of dynamic aspects, we *reify* the notion of "world states" (or simply "worlds") (Section 3.2). *Reification* gives the ability of referencing, quantifying and qualifying over an objectified entity (in this case, the domain's states). We use the "world" as an index to refer to the properties at a particular point in time (Section 3.3). We propose a branching world structure, which can be used to enable arbitrary reference to worlds and world paths in invariants (Section 3.4). We adjust a few standard OCL predefined operations in order to support world indexing (Section 3.5). The use of the resulting temporal extension of OCL described in this section is shown in Section 4.

3.1 Temporal Extension Approach

Our approach for extending OCL with dynamic invariants consists of using a temporalized UML/OCL model in the background with the notion of world states reified, such as illustrated by **Fig. 3**. The OntoUML model is translated into a world-reified model in plain UML. This model is enriched with constraints in standard OCL to ensure the former OntoUML model semantics. Arbitrary standard OCL constraints can then be bound to this temporalized enriched model. With this binding, only few adjustments in standard OCL are required in order for standard OCL to behave as a temporal language. Our extension employs these adjustments: (i) defining built-in operations for world-indexed navigations (Section 3.3) and (ii) creating support for world states in some pre-defined OCL object and classifier operations (Section 3.5). Using the binding with a world-reified model in background we are able to use standard OCL as if it was a temporal language.

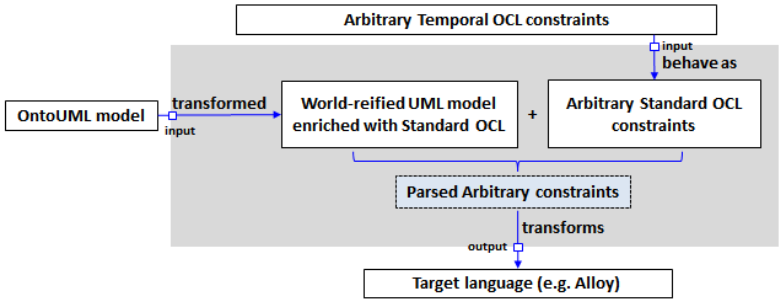

Fig. 3. Temporal Extension Approach

Temporal OCL constraints are then treated as standard OCL (with only few adjustments), and verified syntactically in order to be transformed to other languages such as Alloy [17]. The modeler expresses a conceptual model in OntoUML and Temporal

OCL and is shielded completely from this underlying support, which ultimately generates an Alloy model for simulation and validation of constraints.

3.2 Underlying the World-Reified Model of Background

The idea behind world states reification is to treat the world states (or "worlds") as entities. Thus we introduce the class "World" in this reification step. The OntoUML model example about people, their stages in life and marriages, previously depicted in **Fig. 1**, is translated into a world-reified UML model. **Fig. 4** depicts only a fragment of that resulting UML model. In this model, UML is employed as a temporal model and therefore UML classes represent individuals existing at all possible states of the world. Every former OntoUML class (e.g. the kind Person, the relator universal Marriage) now specializes UML class *Individual*, in order to support the *existsIn* relation, which holds for the worlds in which an individual exists. In this manner, all OntoUML classes can be indexed in time through this relation of existence. Note that all UML relationships in this model are *readOnly* by default since time was reified and each property change is now characterized by a change in the world states.

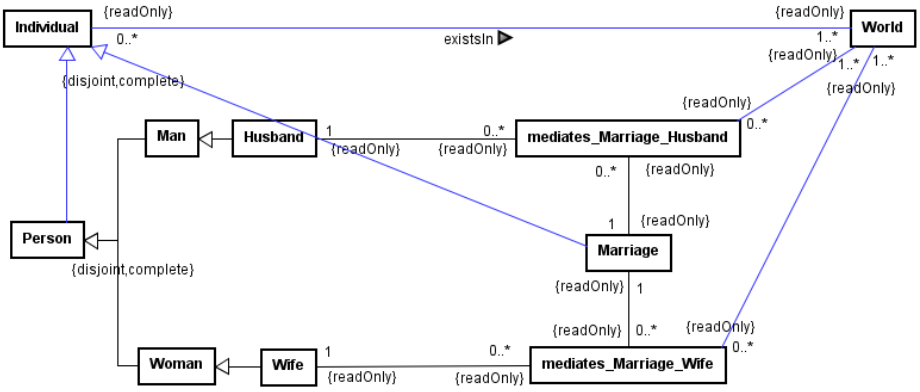

Fig. 4. A Fragment of the World-Reified UML Model of Background

In order to capture the dynamics of OntoUML relationships in this reified UML model, all former relationships are reified (translated) to a UML class, with three UML binary relationships and OCL constraints to maintain the semantics of the original OntoUML relationship. The class representing that relationship defines the worlds in which the relationship exists. For instance, the UML class *mediates_Marriage_Wife* represents the former OntoUML mediation relation between Marriage and Wife. Since all wives must be related to a marriage, it *existsIn* a non-empty set of Worlds. In each world, there may be a set of relationships of this type.

Finally, OntoUML multiplicities define *actual* multiplicity constraints (i.e. they restrict how many individuals an individual may be linked to at a single world state). We chose not to represent OntoUML's actual multiplicity constraints in this reified model using UML because only the lower actual cardinality can be represented using

a UML lifetime multiplicity (e.g. a wife has exactly one marriage at a time, which means that she has also at least one marriage in her lifetime). For this reason, we represented all OntoUML's actual multiplicity in this reified model of **Fig. 4** as additional OCL constraints. Thus, the multiplicities from Wife and Marriage to the reified mediation in the reified model are defined as simply 0..*. **Fig. 5** exemplifies these additional constraints specifying OntoUML's actual multiplicity of the mediation between Marriage and Wife (they are embedded in the world-reified model and transparent to the modeler). The first constraint states that a marriage mediates exactly one wife at a world i.e. for every world (*self*), for all marriages at *self*, the number of *mediates_Marriage_Wife* linked (at *self*) to that marriage is equal to 1. Conversely, the second constraint states that a wife is mediated by exactly one marriage at a world.

```
context World
inv marriage_mediates_one_wife_at_a_time:
    self.individual->select(i | i.oclIsKindOf(Marriage))->forAll(m |
    m.mediates_marriage_wife->select(r | r.world = self)->size() = 1)
inv wife_is_mediated_by_one_marriage_at_a_time:
    self.individual->select(i| i.oclIsKindOf(Wife))->forAll(h |
    h.mediates_marriage_wife->select(r | r.world = self)->size() = 1)
```

Fig. 5. World-Reified UML Model: Reflecting OntoUML Multiplicities using OCL

In order to maintain the actual semantics of OntoUML, additional constraints are required in our world-reified UML model (e.g., to capture the fact that relationships, relators and relata co-exist in all worlds in which they exist, to reflect the immutability of relata on which a relator depends, etc.). They are all represented in plain OCL in the world-reified model. We omit them here due to space constraints.

3.3 Temporal OCL Navigations

Usually, OCL navigations on ternary relationships can proceed in three stages: (i) navigating from a ternary relationship to each class it relates, (ii) from each related class to the ternary relationship itself, and (iii) navigating from a first related class to a second related class, filtering the result to a third related class. In our previous world-reified UML model, only (iii) is allowable filtering the result of navigation with respect to world states. (i) and (ii) are not supported because the reified ternary relationship (i.e. the UML class acting as the relationship in our reified model) is hidden from the modeler (they are an implicit construct generated only in the background), and secondly because we want to refer to properties at a specific point in time, making explicit the world state.

Fig. 6 specifies the definition of the allowable temporal OCL navigations, as (iii). The first navigation *Wife::marriage(w)* is defined from Wife to Marriage filtered by a specific world state. It returns all marriages of a wife at world *w*. The second navigation is defined from Marriage to Wife *Marriage::wife(w)*, returning the wife related to

a specific marriage at *w*. These world-indexed navigations are available to the modeler in order to refer to the relation in a particular state. Furthermore, we also enabled a temporal OCL navigation without a world parameter which returns all individuals of a property, at all possible worlds. For example, if *self* is a wife, then *self.marriage* returns all marriages of a wife in her entire life.

```
context Wife
def: marriage(w: World): Set(Marriage) =
  self.mediates_Marriage_Wife->select(m | m.world=w)->collect(marriage)->asSet()
context Marriage
def: wife(w: World): Set(Wife) =
  self.mediates_Marriage_Wife->select(m | m.world=w)->collect(wife)->asSet()
```

Fig. 6. World-Reified UML Model: Definition of Temporal OCL Navigations

3.4 World Structures

An ordered structure of world states models how the subject domain behaves in time. We adopt a structure of possible worlds inspired in Kripke structures of modal logic semantics; more specifically, we assume the branching structure previously defined in [2]. Each world has a set of immediate next worlds and at most one previous world (it is a tree with branches towards the future, capturing the notion that the future may unfold in different ways). For each world state, there is only one sequence of worlds to a future state of the world (meaning that branches do not join again). A history, i.e., a path, is comprised by a non-empty set of worlds while a world must be included in at least one history, such as depicted by **Fig. 7** using UML. This structure of worlds is a built-in part of every world-reified UML model, dictating how worlds are accessible from each other and specifying a number of pre-defined temporal operations for Worlds and Paths.

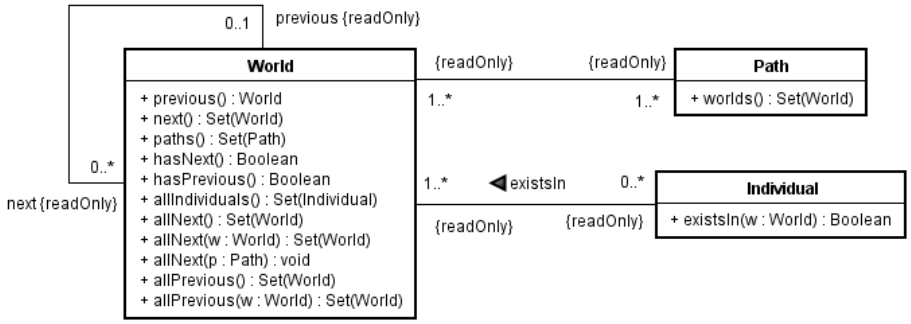

Fig. 7. World-Reified UML Model: World Structure and Temporal Operations

Differently from [2], we have reified the notion of "paths". These are useful to express constraints which are usually expressed in the CTL tense logic, quantifying not only over states but also over paths of states, both universally and existentially. Quan-

tifying universally and existentially over paths is an important feature to some dynamic properties of systems. Since Path is also an entity as World, several additional constraints are defined in standard OCL to complement this world-reified UML model in order to enforce properly the semantics of histories (paths). A history must contain exactly one initial and one terminal world, no two histories should have the same terminal world and every terminal world must be in exactly one history. Additionally, the worlds contained in a history should be derived from all previous worlds of that history's terminal world. We validated our branching world structure using the lightweight formal method of validation based on Alloy simulation and analysis [17], as a means to check the correct semantics of the reified histories (paths) that we introduced in the world structure. The following set of temporal operations are predefined: next() and previous(), which return a world's immediate next and immediate previous world; hasNext() and hasPrevious(), which checks whether a world has an immediate previous or immediate next world; allIndividuals(), which returns all existing individuals at a specific world; existsIn(), which checks the existence of an individual at a specific world; and allNext(), which returns all subsequent worlds of a particular world. This operation has two variants, allNext(w), which returns all subsequent worlds until a particular world *w* is reached (not including *w*) and allNext(p), which returns all subsequent worlds from a world, contained in a given path *p*. Analogously, allPrevious() returns all precedent worlds of a particular world. Finally, p.worlds() returns all worlds of a path *p*, and w.paths() returns all paths in which the world *w* is contained. These pre-defined temporal operations were all implemented using standard OCL (body conditions) over the world-reified UML model. For the sake of brevity, we omit here the Alloy code of our branching world structure and the implementation of these pre-defined temporal operations.

3.5 Standard OCL Operations Revisited

oclIsNew()is only used in post-conditions [22 p.154]. As our subset of OCL does not consider pre- and post- conditions (OntoUML disallows operations) oclIsNew() is not supported. Instead, we defined two temporal operations to check an individual's creation and deletion at a world *w*. oclIsCreated(w) checks if an individual exists in world *w* but does not exist in its immediate previous world. oclIsDeleted(w) checks if and individual does not exist in *w* but does exist in its immediate previous world. These are Individual's operations because existence is a characteristic of domain objects that persist in time. We also defined two additional operations for the classification of an individual at *w*. oclBecomes(C, w) checks whether an individual is classified as C at *w* but is not classified as C in *w*'s immediate previous world. oclCeasesToBe(C, w) in turn checks whether an individual ceases to be classified as C at *w*. That is, the individual does not instantiate C in *w*'s immediate previous world, but instantiate C at *w*.

There are only few adjustments to some built-in OCL object and classifier operations that need to be established due to our world reification approach. Type conformance operations must explicit the point in time in which the types are checked. Since standard OCL does not natively support world states, we include a world state parameter at oclIsKindOf(T, w), oclIsTypeOf(T, w), oclAsType(T, w) and oclType(w). The

allInstances() operation is still allowed and it returns the extension of a class at all possible worlds i.e. the set of all instances of a class independent of their actual existence in a particular point in time. Expressions such as (i) World.allInstances(), (ii) Path.allInstances(), or (iii) Individual.allInstances() are then all valid. They return respectively, the set of all possible worlds, all histories and all individuals at all worlds. Additionally, we assume a temporal operation allInstances(w) for every UML domain class. allInstances(w) returns all instances of a class at a world *w* (expressions such as World.allInstances(w) or Path.allInstances(w) are not valid constructions since worlds were reified and neither worlds nor paths exist within worlds).

4 Representing Dynamic Invariants (the Modeler's View)

In this section, we represent the dynamic aspects which were required to model as accurately as possible the conceptualization presented initially in Section 2, thereby showing how the approach satisfies the requirements. These dynamic aspects include transition rules, existence of individuals and past specializations.

Transition rules constrain the change from one (antecedent) state to another (subsequent) state. The ***Initial Transition*** rule is a peculiar type of transition rule that mentions to antecedent state. The condition holds at the first world of an individual's existence. **Fig. 8** exemplifies this rule in Temporal OCL, stating that every time a person is created, he/she is a child at that moment.

```
context Person
temp initial_transition: World.allInstances()->forAll(w |
     self.oclIsCreated(w) implies self.oclIsKindOf(Child, w))
```

Fig. 8. Temporal OCL: Initial Transition Rule

The keyword *temp* defines a temporal invariant. The "context" defines a class extension at all worlds e.g. all individuals that at some point will instantiate Person. The condition then must hold for each of these individuals. **Fig. 9** exemplifies the ***Final Transition*** in Temporal OCL. It states that for every individual that will eventually be an Elder, for every world, if that individual is an Elder at that world, then for every world after that, if the individual exists, then it instantiates Elder (i.e. there is no other possible and allowed transition for that individual before ceasing to exist).

```
Context Elder
temp final: World.allInstances()->forAll(w | self.oclBecomes(Elder, w) implies
     w.allNext()->forAll(n| self.existsIn(n) implies self.oclIsKindOf(Elder,n)))
```

Fig. 9. Temporal OCL: Final Transition Rule

Differently from initial transitions, final transition can be viewed as a more specific case of a general rule for arbitrary transitions. This general transition rule states that

an antecedent type A1 is transitioned to one or more types S1+... +SN. This means that there is no allowed transition for the instance of A1, before ceasing to exist, if not being A1 itself, or being one of the subsequent types S1, S2…SN. The final transition is just a special case of this general rule where there is no subsequent type only the antecedent type A1. **Fig. 10** exemplifies the general transition in temporal OCL stating that a teenager can only transit to teenager (i.e. A1) or to adult (i.e. S1) phases.

```
context Teenager
temp transition: World.allInstances()->forAll(w | self.oclBecomes(Teenager, w)
     implies w.allNext()->forAll(n | self.existsIn(n) implies
     self.oclIsKindOf(Teenager, n) or self.oclIsKindOf(Adult, n)))
```

Fig. 10. Temporal OCL: General Transition Rule

Fig. 11 exemplifies the ***Transient and Permanent Existence*** in Temporal OCL. The first invariant states that for every person that comes into existence, there will be at least one world after that in which that person will cease to exist. The second invariant states that every marriage, once created, will exist at all possible worlds after that. Since a marriage is existentially dependent on a husband and a wife, by implication, the roles Husband and Wife are final transitions of a person and married persons are permanent. While the individual constraints are meaningful, they are inconsistent with each other, which can be checked using our support for Alloy. The analyzer would show that there is no valid instantiation of the model with these two constraints.

```
context Person
temp transient: World.allInstances()->exists(w | self.oclIsCreated(w) and
     w.allNext()->exists(n | not self.existsIn(n)))
context Marriage
temp permanent: World.allInstances()->exists(w | self.oclIsCreated(w) and
     w.allNext()->forAll(n | self.existsIn(n)))
```

Fig. 11. Temporal OCL: Transient and Permanent Existence

Finally, **Fig. 12** exemplifies a case where ex-husbands and ex-wives are required as cases of a ***Derivation by Past Specialization***. The invariant states that for every eventual ex-wife, for every world, if an ex-wife exists at a world then there exists a set of previous worlds from *w* in which she was a wife and her related past marriage does not exist in *w*.

```
context ExWife
temp past_spec: World.allInstances()->forAll(w | self.oclIsKindOf(ExWife, w)
     implies w.allPrevious()->exists(p | self.oclIsKindOf(Wife, p) and not
     self.oclAsType(Wife, p).marriage(p).existsIn(w)))
```

Fig. 12. Temporal OCL: Derivations by Past Specializations

5 Related Work

There have been many proposals in literature that aimed at extending OCL in order to cope with dynamics/temporal aspects of systems [3, 4, 7, 9, 12, 18, 19, 27]. *Gogolla and Ziemman*'s extension of OCL [27] is based on a set of Linear Temporal Logic (LTL) operators. They introduced an environment's index to characterize the temporal evolution of the system and its current state. *Conrad and Turowski* [7] extended OCL with LTL operators to specify software contracts for business components, where contracts are represented as pre- and post-conditions. *Bill et al.* [3] presented an OCL extension named cOCL, based on Computational Tree Logic (CTL). Their verification framework consists of cOCL specifications and a model checker called MocOCL that can verify cOCL constraints. *Flake and Mueller* [12] defined a state-oriented Real-Time extension of OCL whose semantics is given through a mapping to clocked CTL logics (CCTL). They focus on the specification of real-time systems. Differently from these approaches, we do not use tense logic operators explicitly, choosing to use reification of world states to obtain the expressiveness that would be obtained with tense operators. Extensions based on modal/tense logic operators require a level of logic expertise that most modelers are not expected to have. *Distefano et al.* [9] defined an object-based extension of CTL called BOTL (Object-Based Temporal Logics), a logic formalism inspired by OCL to define specifications of static and dynamic properties in object-oriented systems. BOTL looks syntactically very similar to CTL and although BOTL's concepts are defined clearly and precisely, no tool support is actually provided. *Mullins and Oarga* [19] extended OCL with CTL operators and some first-order features. Their extension termed EOCL is largely inspired by BOTL [9] and based on the framework of *Bradfield et al.* [4]. Their SOCLe tool translates exactly one UML class diagram, one state-chart and one object diagram into an Abstract State Model (ASM) specification, which in turn is translated into an execution graph that can verify on-the-fly EOCL constraints. *Bradfield et al.* [4] proposed a formalism, termed Oμ(OCL) which requires such understanding of temporal logics (as stated by the authors) that is unrealistic to expect most developers to acquire it [4, p.2]. *Kanso and Taha* extended OCL [18] according to the set of Dwyer's temporal property patterns [10] with the explicit inclusion of events. They have fully implemented the OCL extension in an Eclipse/MDT OCL Plugin, which allows OCL temporal constraints to be defined with Ecore/UML models. However, the set of temporal patterns are not suitable to OntoUML's set of requirements, such as the initial transition dynamic aspect, usually, due to the pattern's closed/open edges of intervals. Finally, *Cabot et al.* [6] extended OCL with instant reification but solely to retrieve immediate past values of UML model properties.

6 Concluding Remarks

In this paper, we have defined a temporal extension for standard OCL to cope with dynamics in ontologically well-founded conceptual models with OntoUML. The temporal OCL extension developed requires only few adjustments to standard OCL; in

particular, to four OCL type conformance operations and the allInstances() operation. Our temporal OCL is expressive not only to represent the implicit dynamics of OntoUML (e.g. rigidity, anti-rigidity, immutability), but also to incorporate *user-defined* dynamics aspects into conceptual models, such as transitions, transience, permanence, past derivations, etc.

The extension is fully incorporated into the OLED[2] tool, which is an editor for the creation, development and validation of OntoUML structural conceptual models. We have thus extended the previous work of [14] with the support for a temporal OCL extension, which includes: (i) a temporal OCL editor with syntax highlighting and code-completion, (ii) a parser for temporal OCL constraints using Eclipse's OCL support [11] and (iii) a transformation from temporal-enriched OntoUML models into the Alloy logic-based language, enabling simulation and verification of dynamic constraints written in our temporal extension.

In the future, we plan to compare our approach with other approaches such as *Kanso and Taha*'s temporal OCL extension and their set of temporal patterns [18] and the ontology-based behavioral specification language (OBSL) [26]. These approaches trade expressiveness for ease of use, so we expect that all of the constraints that can be expressed in these approaches can be expressed with our OCL extension. We also plan to represent *Sales*' simulations scenarios for semantic anti-patterns detection [24] as a means to further demonstrate the expressivity of our extension of OCL.

Finally, we should investigate whether some of the dynamic aspects discussed here (e.g., transience and permanence) can be introduced in the graphical notation (e.g., as additional stereotypes) to improve the language's usability. We should also investigate a combination of the approach with other notations such as state diagrams which could support the specification of some of the rules (e.g., transition rules). These diagrams would ultimately be transformed into temporal OCL constraints.

Acknowledgements: This research is funded by the Brazilian Research Funding Agencies FAPES (grant number 59971509/12) and CNPq (grants number 310634/2011-3, 485368/2013-7 and 461777/2014-2).

References

1. Anastasakis, K., Bordbar, B., Georg, G., Ray, I.: On challenges of model transformation from UML to Alloy. Softw. Syst. Model. 9, 1, 69–86 (2010).
2. Benevides, A.B., Guizzardi, G., Braga, B.F.B., Almeida, J.P.A.: Validating modal aspects of OntoUML conceptual models using automatically generated visual world structures. J. Univers. Comput. Sci. 16, 2904–2933 (2011).
3. Bill, R., Gabmeyer, S., Kaufmann, P., Seidl, M.: OCL meets CTL - Towards CTL-Extended OCL Model Checking. In MoDELS, 1092, pp. 13–22 (2013).
4. Bradfield, J.C., Filipe, J.K., Stevens, P.: Enriching OCL Using Observational Mu-Calculus. In FASE, 2306, 203–217 (2002).

[2] https://code.google.com/p/ontouml-lightweight-editor/

5. Brucker, A.D., Wolff, B.: HOL-OCL: a formal proof environment for UML/OCL. In: Fiadeiro, J.L. and Inverardi, P. (eds.) 11th International Conference on Fundamental Approaches to Software Engineering, FASE 2008. pp. 97–100 Springer Berlin Heidelberg (2008).
6. Cabot, J., Olivé, A., Teniente, E.: Representing Temporal Information in UML. In UML'03, 2863, 44–59 (2003).
7. Conrad, S., and Turowski, K.: Temporal OCL Meeting Specification Demands for Business Components. In UML'01, 2185, 151–165 (2001).
8. Cunha A., Garis A., Riesco D.: Translating between Alloy specifications and UML class diagrams annotated with OCL. Softw. Syst. Model. (2013).
9. Distefano, S., Katoen, J.P., Rensink, A.: On a temporal logic for object-based systems. In Fourth International Conference on Formal methods for open object-based distributed systems IV, 49, 305-325 (2000).
10. Dwyer, M.B., Avrunin, G.S., Corbett J.C.: Patterns in property specifications for finite-state verification. In Proceedings of the 21st International Conference on Software Programming, 411–420 (1999).
11. Eclipse MDT OCL, http://www.eclipse.org/modeling/mdt/
12. Flake, S., and Muller, W.: Formal semantics of static and temporal state-oriented ocl constraints. Software and System Modeling 2(3), 164–186 (2003).
13. Gogolla, M., Bohling, J., Richters, M.: Validating UML and OCL models in USE by automatic snapshot generation. Softw. Syst. Model. 4, 4, 386–398 (2005).
14. Guerson, J., Almeida, J. P. A., Guizzardi, G.: Support for Domain Constraints in the Validation of Ontologically Well-Founded Conceptual Models. In 19th International Conference, EMMSAD, 302-316 (2014).
15. Guizzardi, G.: Ontological Foundations for Structural Conceptual Models. Telematica Instituut, The Netherlands (2005).
16. Halpin, T. and Morgan T.: Information modeling and relational databases. Morgan Kaufmann (2010).
17. Jackson, D.: Software Abstractions-Logic, Language, and Analysis, Revised Edition. The MIT Press (2012).
18. Kanso, B., and Taha, S.: Specification of temporal properties with OCL. Science of Computer Programming 96, 527-551 (2014).
19. Mullins J. and Oarga R.: Model Checking of Extended OCL Constraints on UML Models in SOCLe. In FMOODS, 4468, 59–75 (2007).
20. Mylopoulos, J.: Conceptual Modeling, Databases, and CASE: An Integrated View of Information Systems Development; chapter Conceptual Modeling and Telos; Wiley, Chichester (1992).
21. Olivé A., and Teniente, E.: Derived types and taxonomic constraints in conceptual modeling. Information Systems 27(6), 391–409 (2002).
22. OMG: OCL Specification v2.4.1 (2014).
23. OMG: UML Superstructure v2.4.1 (2012).
24. Sales T.P.: Ontology Validation for Managers, MSc Thesis, Federal University of Espírito Santo, UFES (2014).
25. Sider T.: Quantifiers and Temporal Ontology. Mind 115(457), 75-97 (2006)
26. Wiegers, R.: Behaviour Specification for Ontologically Grounded Conceptual Models. MSc Thesis, University of Twente (2014).
27. Ziemann, P., and Gogolla, M.: OCL Extended with Temporal Logic. In 5th International Andrei Ershov Memorial Conference, PSI, 2890, 351-357 (2003).