Innovations Syst Softw Eng (2010) 6:55–63
DOI 10.1007/s11334-009-0120-5

ORIGINAL PAPER

Transforming OntoUML into Alloy: towards conceptual model validation using a lightweight formal method

Bernardo F. B. Braga · João Paulo Andrade Almeida · Giancarlo Guizzardi · Alessander B. Benevides

Received: 30 November 2009 / Accepted: 17 December 2009 / Published online: 2 February 2010

Abstract While conceptual modeling is strongly related to the final quality of the software product, conceptual modeling itself remains a challenging activity. In particular, modelers must ensure that conceptual models properly formalize their intended conceptualization of a domain. This paper proposes an approach to facilitate the validation process of conceptual models defined in OntoUML by transforming these models into specifications in the logic-based language Alloy and using its analyzer to generate instances of the model and assertion counter-examples. By allowing the observation of sequences of snapshots of model instances, the dynamics of object creation, classification, association and destruction are revealed. This confronts the modeler with the implications of modeling choices and allows them to uncover mistakes or gain confidence in the quality of conceptual models.

Keywords Conceptual modeling · OntoUML · Validation · Lightweight formal methods · Alloy

This work has been supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and by Fundação de Apoio à Ciência e Tecnologia do Espírito Santo (FAPES) in the scope of the INFRA-MODELA project.

B. F. B. Braga · J. P. A. Almeida (✉) · G. Guizzardi · A. B. Benevides
Ontology and Conceptual Modeling Research Group (NEMO),
Computer Science Department,
Federal University of Espírito Santo (UFES),
Vitoria, ES, Brazil
e-mail: jpalmeida@ieee.org

B. F. B. Braga
e-mail: bernardofbbraga@gmail.com

G. Guizzardi
e-mail: gguizzardi@acm.org

A. B. Benevides
e-mail: abbenevides@inf.ufes.br

1 Introduction

The practical relevance of thorough requirements analysis is emphasized by evidence provided by the empirical software engineering community, which states that it is much cheaper to find and fix software problems during the requirements and design phase than after delivery [6]. In this context, properly acquiring knowledge on a problem domain prior to detailed design has justified several efforts in conceptual modeling, which can be defined as "the activity of formally describing some aspects of the physical and social world around us for purposes of understanding and communication" [12]. In order to support such purposes, a formal conceptual model must capture a modeler's intention and convey a precise message with unambiguous semantics. This is particularly important if conceptual models are to be used effectively as a basis for the construction of an information system.

As argued in [8], the quality of a conceptual modeling language can be assessed by considering the extent to which the language supports the definition of models that capture the modeler's conceptualization of a domain. This concern has justified the revision of a portion of UML into the OntoUML conceptual modeling language. This revision enables modelers to make finer-grained distinctions between, among other things, different types of classes according to the UFO foundational ontology [8]. These ontological distinctions reflect, in turn, different manners an object can be an instance of a type. In particular, focusing on the different modal (temporal) consequences these different modes of instantiation imply.

Regardless of the quality of the conceptual modeling language employed, conceptual modeling itself remains a challenging activity, requiring additional methodological and tool support for ensuring that the modeler's intention is properly reflected in the models.

The development tool to aid the construction of OntoUML conceptual models presented in [4] has given us so far the opportunity to verify models for ontological well-formedness, i.e., adherence to ontological consistency rules defined at the language-level. While this guarantees some quality for conceptual models by enforcing ontological consistency via domain-independent syntactic rules, it does not serve to increase the modeler's confidence in the correct representation of the intended domain conceptualization, i.e., it does not support modelers in answering the question "have we built the right model for this particular domain?".

This paper proposes an approach to facilitate the validation process of conceptual models defined in OntoUML by transforming these models into specifications in the logic-based language Alloy (a "lightweight formal method" [9]) and using its analyzer to generate instances of the model and possibly produce assertion counter-examples. Validation is defined here as "the process of determining the degree to which a model is an accurate representation of the real-world from the perspective of the intended uses of the model" [17]. Our approach supports validation by allowing the observation of sequences of snapshots of model instances. We argue that the visualization of instances confronts the modeler with the implications of modeling choices. Should the instances reveal inadmissible states-of-affairs (or sequences thereof), the model may be analyzed to identify opportunities for correction in an iterative validation approach. Moreover, we believe that this can also be used as means to identify missing or over restrictive domain rules.

In this article, we build on our earlier work in [5], in which we have discussed the assessment of the modal aspects of conceptual models. Here we focus on the issue of dynamic classification, thus, we concentrate on illustrating sequences of snapshots of model instances which reveal the dynamics of object creation, classification, association and destruction.

The rest of the paper is organized as follows: Section 2 briefly describes OntoUML. Section 3 describes Alloy. Section 4 presents our transformation rules from OntoUML to Alloy. Section 5 presents instance generation and analysis. Section 6 discusses related work and Sect. 7 brings final conclusions.

2 OntoUML

Due to space limitations, we concentrate here on a fragment of the Unified Foundation Ontology (UFO) [8], with a specific focus on those distinctions that are spawned by variations in meta-properties of a modal nature. UFO's main categories are depicted in Fig. 1 and are briefly discussed in the remainder of this section by using a running example depicted in Fig. 2. Since OntoUML is a modeling language which metamodel is designed to be isomorphic to the UFO ontology, the leaf ontological distinctions in Fig. 1 appear as modeling primitives in the language (stereotyped classes and relationships in Fig. 2).

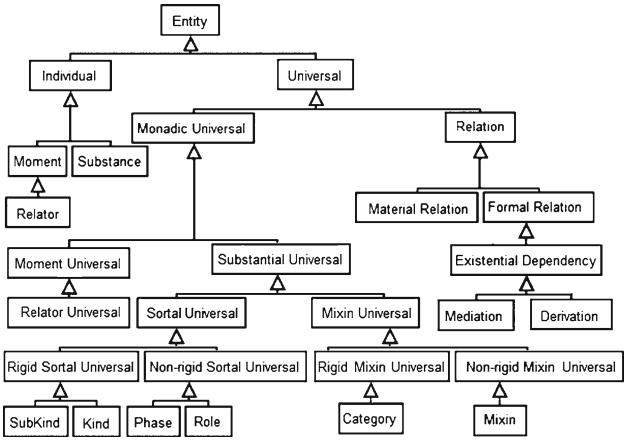

Fig. 1 Excerpt of UFO taxonomy [8]

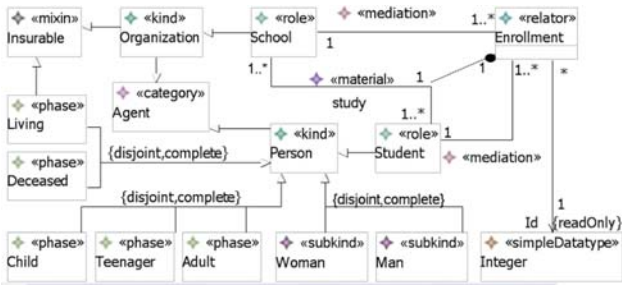

Fig. 2 Running example

2.1 Substances and moments

UFO is based on a fundamental distinction between Individuals and Universals (roughly instances and types, respectively) and, within the category of individuals, it differentiates between Substances and Moments. The distinction between Substances and Moments is based on the formal notion of existential dependence, a modal notion that can be briefly defined as follows:

Definition 1 (*existential dependence*) an individual *x* is *existentially dependent* on another individual *y* iff, as a matter of necessity, *y* must exist whenever *x* exists. In other words, in every world *w*, if *x* exists in *w* then *y* must also exist in *w*.

Substances are existentially independent individuals, i.e., there is no Entity *x* disjoint from *y* that must exist whenever a Substance *y* exists. Examples of Substances include ordinary mesoscopic objects such as a Person or a Car. Conversely, a Moment is an individual that can only exist in other individuals, i.e., that is existentially dependent on other individuals. Here, we concentrate on relational moments or relators (e.g., a covalent bond, an enrollment or a marriage).

So, a Substantial Universal is a universal whose instances are Substances (e.g., the universal Person or the universal Apple). While, a Relator Universal is a universal whose instances are individual relational moments (e.g., the particular enrollment connecting John and Organization0 in Fig. 4 is an instance of the universal Enrollment).

In addition, Kinds and Relators represent what is termed an Ultimate Sortal Universal [8]. An Ultimate Sortal Universal is a universal that supplies a principle of identity which is obeyed by its instances. A principle of identity is a principle for which we can judge whether two individuals are the same and which conditions an individual remain the same, i.e., it supplies the conditions for univocal identification and persistence of an individual [8]. For instance, in a given conceptualization, a principle of identity for Cars could be "having the same chassis number", hence, in that context, two cars are the same iff they have the same chassis number and a car c remains the same entity as long as it preserves that chassis number, irrespective of other changes it could suffer. Every individual of the conceptual model must instantiate one and only one Ultimate Sortal Class supplying the principle of identity it should obey [8]. Finally, all instances classified under a Sortal Universal obey the same principle of identity.

2.2 Rigidity

We need to define some additional modal notions (rigidity and non-rigidity) to be able to make further distinctions within Object Universal.

Definition 2 (*Rigidity*) A universal U is rigid if for every instance x of U, x is necessarily (in the modal sense) an instance of U. In other words, if x instantiates U in a given world w, then x must instantiate U in every world $w^{\backslash}$ in which x exists and that is accessible from w.

Non-rigidity is taken here to be simply the logical negation of rigidity.

2.3 Sortal universals

Sortals, as previously mentioned, are sorts of universals that carry principles of identity for their instances. Person, Car, Dog and Student are examples of Sortal Universals.

Sortal Universals that are rigid are named Kinds and subKinds. These universals define a stable backbone, a taxonomy of rigid universals instantiated by a given substance individual (the Kind being the Ultimate Substance Sortal for objects).

Within the category of non-rigid sortal universals we have a further distinction between Phases and Roles. Both Phases and Roles are specializations of Kinds or subKinds. However, they are differentiated w.r.t. their specialization conditions. For the case of Phases, the specialization condition is always an intrinsic one. For instance, in Fig. 2, a Child is a Person within a certain age. For Roles, in contrast, their specialization condition is a relational one: a Student is a Person who is enrolled in (has a study relation to) a School, etc. Formally speaking, this distinction is based on a meta-property named Relational Dependence:

Definition 3 (*Relational Dependence*) A type T is relationally dependent on another type P via relation R iff in every world w, for every instance x of T there is an instance y of P in that world such that x and y are related via R in w.

Finally, as discussed in [8], Phases (in contrast to Roles) are always defined in a partition set. For instance, in Fig. 2, the universals Child, Teenager and Adult define a phase partition for the Kind Person. As consequence, we have that in an each world w, every Person is either a Child, a Teenager or an Adult in w and never more than one of these. In addition, if x is a Child (Teenager, Adult) in w, there is always a possible world $w^{\backslash}$, accessible from w, in which x will not be a Child, in which case he will be either a Teenager or an Adult.

In summary, in the example of Fig. 2, these model distinctions (Definitions 2 and 3) are exemplified by contrasting the (Kind) universal Person, the (Role) universal Student and the (Phase) universal Teenager. Please note that, since instances of non-rigid universals may change their types, classifiers representing non-rigid universals are subject to dynamic classification.

2.4 Mixin universals

Mixins are sorts of universals that do not carry a principle of identity, instead they classify individuals that obey different principles of identity (e.g., Agent in Fig. 2 which classifies different *kinds* of entities such as Persons and Organizations). Hence, mixins are types which provide properties to (characterize) individuals which have already being individuated by sortal-supplied principles.

Mixin Universals can also be refined under more specific categories regarding rigidity. Rigid mixins are called Categories. We use the general term Mixin instead for mixins which are not rigid.

2.5 Relator universals and relations

In order to represent the relation between Student and Person, one should model Student as a role played by Person in a certain context, where he is enrolled in a School. Analogously, one should model School as a Role played by an Organization when providing educational services to a Student. This context is materialized by the Material Relation study (represented as the «material» stereotype in OntoUML), which is in turn, derived from the existence of the Relator Universal Enrollment («relator»). In other words, we can say

that a particular student x studies at a particular school y iff there is an Enrollment z that mediates x and y. This situation is illustrated in Fig. 2. The formal relations of mediation in this model represent the existential dependence of the relator on its bearers [8].

2.6 Quality structures

In UML, "a data type is a type whose instances are identified only by their value" [13]. Data types are used in OntoUML to specify what is called a quality structure in [8]. In contrast with individuals, "any instances of that data type with the same value are considered to be the same instance" [13]. In the example in Fig. 2, a data type is used to represent the identifiers of enrollments.

3 Alloy

Alloy is defined as "a structural modeling language based on first-order logic, for expressing complex structural constraints and behavior" [9]. The language is supported in a constraint solver called "Alloy Analyzer" which provides simulation and checking for an Alloy model.

A model in Alloy consists of logical constraints which are captured in *signature* and *fact* declarations. When a model is instantiated by the Alloy Analyzer, *atoms* are generated from signatures respecting the logical constraints in the model. In other words, a signature at the model level introduces a set of atoms at the instance level.

Figure 3 shows an example of an elementary Alloy model, which includes a Person signature. At the instance level, "Person atoms" are generated by the Alloy analyzer. Other signatures, such as "Organization" produce other kinds of atoms. Figure 4 depicts a sequence of instance-level states, each containing several atoms.

Signatures can include field declarations, which introduce relations between signatures. There are no top-level relations in Alloy; relations can only be declared as fields in signature declarations, e.g., in Fig. 3, signature Enrollment has a field Student, which introduces a relation Enrollment → Person. On every field or signature declaration, it is possible to use a multiplicity keyword to restrict the cardinality of the relation. The keywords are one, lone, some and set, which restricts the image to one, one or zero, one or more and zero or more elements, respectively. In Fig. 3, the field Student in signature Enrollment uses the keyword "one", which means that for every Enrollment, there will be one and only one Person associated to it via the student relation. Note that the multiplicity keyword applies only in one direction. No constraint is implied on how many Enrollments a person may relate to. Such restrictions may be added as signature facts or as facts. Facts introduce constraints which are assumed to

```
01.open util/ordering[State] as state
02.open util/relation
03.sig Person{ }
04.sig Man, Woman in Person{ }
05.fact generalization_set{
06.   disj[Man,Woman]
07.   Person = Man+Woman
08.}
09.sig Organization{ }
10.sig Enrollment{
11.   school: one Organization,
12.   student: one Person,
13.   id: one Int,
14.  derived_material_relation: student one -> school,
15.}
16.fun Agent:(Organization+Person){
17.   Organization+Person
18.}
19.sig State{
20.   exists: set (Person+Organization+Enrollment),
21.   disj Adult, Child, Teenager: set Person:>exists,
22.   disj Deceased, Living: set Person:>exists,
23.   Student: set Person:>exists,
24.   School: set Organization:>exists,
25.   Insurable: set Organization:>exists+Living,
26.   study: set Student some -> some School,
27.}   {
28.   all x:exists|x not in this.next.@exists implies
                          x not in this.^next.@exists
29.   Person:>exists = Adult+Child+Teenager
30.   Person:>exists = Deceased+Living
31.   all x:Enrollment:>exists | x.school in
        Organization:>exists and x.student in Person:>exists
32.   Student = (Enrollment:>exists).student
33.   (Enrollment:>exists).school in School
34.   all x: School | some Enrollment:>exists:>school.x
35.   Insurable = Organization:>exists+Living
36.   study in exists.derived_material_relation
37.}
```

Fig. 3 An Alloy model

be always true. Signature facts do the same but are implicitly universally quantified over the signature's set.

Signatures in Alloy can be used as a basis for the definition of *subsignatures*. The subsignature mechanism corresponds intuitively to the notion of specialization in conceptual modeling; subsignatures inherit relations and constraints of upper level signatures. For example, in Fig. 3, signatures Man and Woman are subsignatures of the Person signature (which is indicated by the keyword "in"). The sets introduced by these signatures are subsets of the Person signature. A signature that is not a subsignature is called a *top-level signature*. Each atom generated by the Alloy Analyzer belongs to one and only one *top-level* signature, although they can belong to any number of subsignatures.

4 Transformation

Our approach is based on the transformation of OntoUML models into Alloy models. The product of this transformation

is an Alloy specification that can be fed into the Alloy Analyzer to generate a sequence of instance-level states which are valid according to the language axioms. Throughout this section we use a running example shown in Fig. 3. It corresponds to the OntoUML model presented in Fig. 2.

4.1 Individuals and state transition representation

Individuals of the conceptual model are represented as Alloy atoms. In the same way Alloy *atoms* belong to one and only one *top level signature*, instances of an OntoUML conceptual model belong to one and only one ultimate sortal class. Thus, in our approach, each ultimate sortal class (i.e., each Kind and each Relator) is transformed into an Alloy signature. When the Alloy Analyzer generates atoms to find a suitable instance of the Alloy Model, each generated atom represents a unique individual of the conceptual model.

Since Alloy, in its latest version, has no built-in notion of state transition, we reify this notion by declaring a State signature, ordered with the native "util/ordering" library. By associating individuals to State atoms, we are able to represent the dynamics of states of affairs in an ordered, linear and discrete time representation.

To represent creation and/or destruction of individuals, a field *exists* is declared in the state signature, with the purpose of capturing which atoms exist in a given state. In other words, an atom x exists in a state s iff relation $s \rightarrow x$ belongs to *exist*. Further, in our view, the existence of an individual is undivided in time, i.e., if an individual is destructed at some point, it cannot exist in any subsequent states. This rule is depicted in Fig. 3, line 28 (^ denotes transitive closure and @ is used to prevent a field name from being expanded i.e to refer to x's particular "exist"field). We also constraint every ultimate sortal atom to exist in some state, but omit such rule due to space limits.

4.2 Class representation

In our approach, each class is represented as a set. An atom x representing an individual is said to instantiate a given class C if x belongs to the set that represents C. However, since OntoUML allows dynamic classification for non-rigid classes, we must represent classes differently according to rigidity. Rigid classes are represented as simple atom sets, while non-rigid classes are represented as fields of the state signature. This way, rigid classification is state-independent (as expected) and each state conveys information of the current classification of atoms by non-rigid classifiers.

SubKinds (which are rigid classes) are represented as subsignatures, i.e., subsets of a signature set. When the Alloy Analyzer populates the model with atoms, it arbitrarily includes some of them in the possible subKind sets. Categories, on the other hand, are abstract classes whose extension is equal to the sum of the extensions of the classes which subsume it. Categories are thus represented as total functions of the classes that subsume it, e.g., in Fig. 3 function Agent defines a set of atoms that instantiate the Agent class, namely, the union of Person and Organization sets.

All non-rigid classes, including roles, phases and mixins, are represented as fields of the State signature, with subtle differences in representation due to their different specialization conditions. This means that the transformation for each of these classes must introduce different constraints for the different fields. In particular, fields representing roles are constrained such that every member must be a target of the corresponding mediation relationship, reflecting the relational dependence of roles. In turn, fields representing mixins are constrained such that they are equal to the union of all sets representing classes that subsume the mixin. This is necessary since mixins are abstract classes and, similarly to categories, cannot be instantiated directly.

Generalization sets (i.e., sets of generalization relations forming a partition) in (Onto)UML are quite trivial to transform. Disjointness is represented with the *disj* keyword, either as a function in fact constraints, or for some special cases, such as phase partitioning, in the classes' declaration. Completeness is represented by equating the general class set to the union of the generalized classes' sets. The general approach is to apply these constraints in the State signature facts, such as we have done in the case of Person phase partitions (Fig. 3, lines 29 and 30). Nevertheless, if the generalization set connects rigid classes, its properties of can be represented as a simple fact, such as in the case of the Man and Woman partitions of Person (Fig. 3, line 05).

4.3 Associations

Similarly to non-rigid classes, associations are generically represented as fields of the state signature, e.g., the material relation *study* (Fig. 3, line 26), which is derived from the Enrollment relator, as mentioned in Sect. 2.5.

The cardinality of a relation can be narrowed down with the basic multiplicity keywords. Consider a relation A m → n B, where m and n are multiplicity keywords and A and B are sets. Such relation is constrained to map each member of A to n members of B and to map m members of B to each member of A. Again, since we only have four basic multiplicity keywords ("one", "lone", "some", "set" as discussed in Sect. 3), this mechanism works only for defining the most common cardinalities in conceptual modeling, namely 1, 0...1, 1...* and *. Nevertheless, the cardinalities may be further narrowed down by universal quantification of the relation and the use of the # operator.

Mediations imply existential dependency and exist throughout the extent of the relators' existence. Due to this, we can represent mediations as fields of the relator signature.

5 An example

In this section we shall guide through some instances of the model generated by the Alloy Analyzer from the specification presented in Fig. 3. Figure 4 shows a sequence of states generated by the Alloy Analyzer from the Alloy model shown in Fig. 3, which was in turn obtained from automatic transformation of the OntoUML model in Fig. 2. Each arrow represents a relation and each box represents an individual, except for Integers, where each box represents a value. Theme options have been applied to improve visualization, such as projecting over the state signature, applying different shading for Alive and Deceased phases, and hiding individuals which do not exist in the currently visible state as well as unused values. The individuals John and Mary represent two distinct instances of Person.

Our method takes advantage of the Alloy Analyzer to offer automatic instance generation, which confronts the modeler with arbitrary model population respecting the constraints in the Alloy specification. The visualization of individuals of the conceptual model, their behavior while migrating between non-rigid classes and how they associate with other individuals will either strengthen the modeler's confidence in the produced model, if faced with expected behavior, or reveal characteristics which are not intended and can then be corrected.

The first state reveals two Person atoms (John and Mary), one Enrollment atom and one Organization atom. Mary is a living (thus insurable) adult. John is a deceased (thus not insurable) child and a Student; Organization0 is his school. This raises the first question on the model: should deceased persons be allowed to be Students? We are not advocating there is a general ontological choice that should be counte-

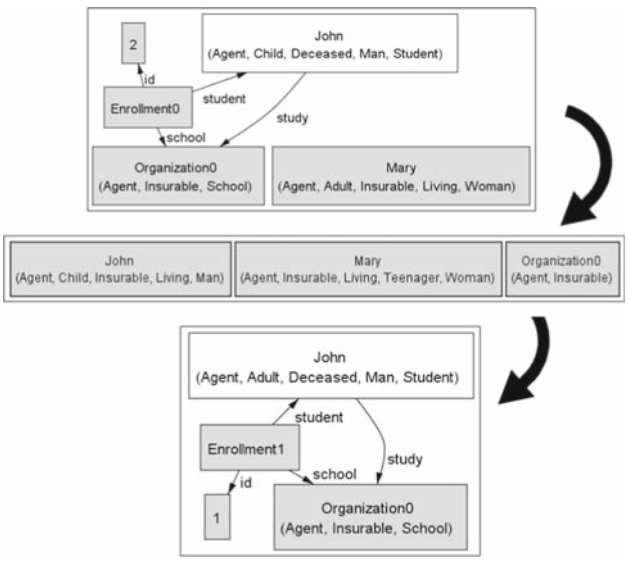

Fig. 4 A sequence of states

nanced in all conceptualizations; simply that a choice must be made and it should reflect the intended conceptualization.

In the second state, the study relationship between John and Organization0 no longer holds. Thus, Enrollment0 is destroyed, John is no longer a student and Organization0 is no longer a School. Two things emerge as unusual in this state transitioning: John came back to life and Mary turned from an Adult to a Teenager, contradicting common sense and, most likely, the modeller's intention. This suggests that some sort of control on the intrinsic reasons that cause phase changes may be necessary to ensure that transitioning occurs as intended. This is necessary since the OntoUML model does not make explicit the phase transition conditions (and thus from the perspective of the generated Alloy specification, transitions are arbitrary).

In the last state transition, Mary disappears, i.e., is destructed. This brings up certain questions on the abstraction of the domain. What is the semantics of the destruction of a substance individual? In this particular case, Mary's destruction should not be interpreted as her death since a Person's death is already modelled by a phase change. In addition, John skips adolescence and becomes an Adult straight from Childhood. This reinforces the need for controlling phase changes in the model.

The questions that emerged through arbitrary instantiation reflect modeling choices and missing domain-specific constraints that affect deeply the semantics of the conceptual model, and should be elucidated to guarantee that the model corresponds to the intended conceptualization. If it does not, constraints should be added, or the model should be corrected to improve the fit between intended conceptualization and the model.

Situations which are expected by the modeler and do not emerge in random instance generation can be searched for by the Alloy analyzer using logical constraints in the form of predicates. Assertions can be used to validate rules that are assumed to hold, as the Alloy Analyzer will try to find counter-examples, instances of the model that contradict the assertion. For example, let us consider the informal requirement that "within a school students should have a unique enrollment id". Is it safe to assume that the model complies with this requirement? With the assertion shown in Fig. 5, we check whether two different enrollment relationship always have different enrollment numbers.

Running the Alloy analyzer will reveal a counter example, shown in Fig. 6. We present only the state which violates the constraint due to space restrictions (here John and Mary share the same identifier). Labels starting with $ are used to identify variables of the assertion. We may introduce the constraint as a fact to ensure that generated enrollments cannot repeat id numbers. Note that there are two possible interpretation for the notion of "student" in the informal requirement. The first interpretation focuses on the substantial individual that

```
01.assert unique_ids{
02.  all sc: State.School|
03.  let sc_enrollments = (Errollment<:school).sc|
04.  all number: Int |
05.  lone sc_enrollments<:id.number  }
```

Fig. 5 Each id corresponds at most to one Enrollment

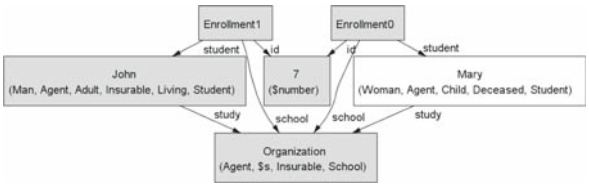

Fig. 6 A counter example to an assertion

plays the role. Under this interpretation, an id identifies a student, thus, an id identifies a person. The second interpretation focuses the facet of that person in that particular relationship (a.k.a. qua-individual[8]). We have taken the second interpretation and thus, quantify over the enrollments in Fig. 5.

Let us consider further the informal requirements that (i) "students cannot be enrolled more than once in the same school at one point in time" and that (ii) "some students may be enrolled in the same school more than once in different points in time". Figure 7 shows the assertion written to check for (i) and a predicate to search for occurrences of (ii). The predicate is of particular interest, since it specifies a constraint that requires analysis beyond a single snapshot, which is made possible by the reification of states.

Running the Alloy analyzer to verify the assertion will produce a counterexample, as shown in Fig. 8. In contrast, running the analyzer for the predicate finds instances of the

```
01.assert lone_enrollment{
02.  all s: State |
03.  all student_x : Student[s] |
04.  lone (s.exists)<:(student.student_x)
05.}
06.pred more_enrollments{
07.  some x : State.Student |
08.  #student.x > 1 and #exists.student.x >1
09.}
```

Fig. 7 An assertion and a predicate regarding student enrollment

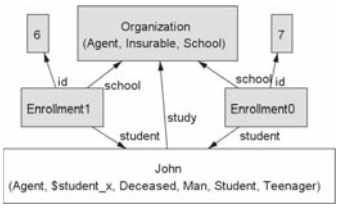

Fig. 8 A student enrolled twice in the same school

model that satisfy the constraint. We refrain from showing yet another figure due to lack of space, but Fig. 4 also complies to the predicate.

It is important to notice that failure by the Alloy Analyzer to find instances of the model that comply to a predicate, or contradict an assertion is not an assurance of their inexistence. This is because the analysis conducted by Alloy is not 'complete' in the sense that it examines only a finite space of cases [9]. This space of cases is constrained by a parameter called scope, which restricts the number of atoms of top-level signatures. Nevertheless, within the given scope, the analysis is exhaustive.

6 Related work

Several approaches in literature aim at assessing whether conceptual models comply with their intended conceptualizations. Although many approaches (e.g., [3,16]) focus on analysis of behavioral UML models, we are primarily concerned with structural models and thus refrain from further analysis of behavioral-focused work.

A prominent example is the UML Specification Environment (USE) tool proposed in [7]. The tool is able to indicate whether instances of a UML class diagram respect constraints specified in the model through OCL. Differently from our approach, which is based on the automatic creation of example state sequences, in USE the modeler must specify sequences of snapshots to gain confidence on the quality of the model [either through the user interface or by specifying sequences of snapshots in a tool-specific language called A Snapshot Sequence Language (ASSL)].

Similar to USE, [11] focuses on analysis and constraint validation of single snapshots only. Differently from our approach, [11] relies on manual translation of class diagrams. Further, they translate all classes to Alloy signatures, which suggests that no dynamic classification is possible.

The approach described in [10,1] is similar to ours in that the authors have implemented a model transformation to automatically generate Alloy specifications from UML class diagrams. Further, they introduce a notion of state transition to show sequences of snapshots. However, since they also translate all classes to Alloy signatures, dynamic classification is not accounted for. This implies in significant differences in the transformation patterns and restricts the applicability of the approach to analyze conceptual models that rely on dynamic classification.

While in this paper we use a linear notion of time, in [5] we have explored a branching-time Kripke structure to represent the dynamic aspects of the model. Each approach has several consequences. Branching time grants us with a wider view of possibilities for individuals by allowing the visualization of multiple realities. However, it requires us to

instantiate one atom for every state (or "world"), whether factual or counter-factual, and possibly extra atoms for every individual which only appear in some of the branches; thus enlarging the required scope. This may be problematic due to the fact that larger scopes increase analysis complexity and, hence, response time. By ignoring the branches and focusing on a single sequence of events, we save the computational effort and possibly reach further into the state space, populating the examples with more individuals or simply improving the performance of analysis. Naturally, this means that, in the approach presented here, each execution of the Alloy Analyzer corresponds to different possible history lines, each of which corresponds to a "path" through the branching-time structure. As a consequence, each "path" is considered in isolation and, therefore, no intersections between different history lines ("paths") are revealed.

7 Concluding remarks

Conceptual modeling constitutes a fundamental phase in Software Engineering and Database Design in which aspects of the domain of discourse are represented in diagrammatic specifications. As well understood in these fields, the quality of implementations is strongly dependent on the quality of the conceptual models from which they are derived.

In the last decade, UML has become a de facto standard for conceptual modeling in Software Engineering. However, its adequacy for that purpose is impaired due to its ambiguous semantics. For this reason, many approaches have been developed to: (i) provide formal semantics for UML, (ii) provide automated mechanisms for checking the formal consistency of UML models.

Nonetheless, none of the related works we have identified covering (i) and (ii) capture the issue of dynamic classification/object migration. Object migration has been an important issue in the literature of conceptual modeling at least since the late seventies [2] and its role in capturing subtle semantics aspects of software systems can be summarized by the following quote from [14]: "To effectively model complex applications in which constantly changing situations can be represented, a systems must be able to support the evolution …of individual objects. The strict uniformity of objects contained in a class is unreasonable…. An object that evolves by changing its type dynamically is able to represent changing situations as it can be an instance of different types from moment to moment." In addition, as discussed in depth in [8], having an explicit account for modeling and analysis of dynamic classification in a conceptual modeling language is fundamental to avoid semantic interoperability problems. As demonstrated there, for instance, the false identity of two classes in some practical model integration situations can only be spotted when the difference in modal (temporal) extensions of these classes are contrasted and made explicit.

Finally, from a real-worlds semantics perspective, there is an important difference between our work and other traditional conceptual modeling accounts of dynamic classification in the literature such as [2,14], namely, that our dynamic categories reflect a system of theoretical distinctions founded in research in formal ontology, philosophy of language and cognitive science. For this reason, these distinctions are not only precise from a logical point of view but are also cognitively warranted in the sense that they reflect the meta-level categories that we humans as cognitive subjects in fact employ to construct our conceptualizations of reality (as empirically supported by several works, see [8]). This characteristic is of great importance for conceptual modeling, in which the resulting specifications should not only be formally correct but should also be effective in supporting humans in tasks such as problem-solving, understanding and communication.

References

1. Anastasakis K, Bordbar B, Georg G, Ray I (2009) On challenges of model transformation from uml to alloy. Softw Syst Model (to appear)
2. Bachman CW, Daya M (1977) The role concept in data models. In: VLDB '1977: Proceedings of the third international conference on Very large data bases, VLDB Endowment, pp 464–476
3. Beato ME, Barrio-Solórzano M, Cuesta CE (2004) UML automatic verification tool (TABU). In: SAVCBS'04: Specification and verification of component-based systems at ACM SIGSOFT 2004/FSE-12
4. Benevides AB, Guizzardi G (2009) A model-based tool for conceptual modeling and domain ontology engineering in ontouml. In: Filipe J, Cordeiro J (eds) ICEIS. Lecture notes in business information processing, vol 24. Springer, Heidelberg, pp 528–538
5. Benevides AB, Guizzardi G, Braga BFB, Almeida JPA (2009) Assessing modal aspects of ontouml conceptual models in alloy. In: Heuser CA, Pernul G (eds) Proceedings of the first international workshop on evolving theories of conceptual modelling (EThe-CoM 2009). 28th international conference on conceptual modeling (ER 2009). Lecture Notes in Computer Science (LNCS), vol 5833. Springer, Gramado, pp 55–64
6. Boehm B, Basili VR (2001) Software defect reduction top 10 list. Computer 34(1):135–137
7. Gogolla M, Büttner F, Richters M (2007) Use: a uml-based specification environment for validating uml and ocl. Sci Comput Program 69:27–34
8. Guizzardi G (2005) Ontological foundations for structural conceptual models. Ph.D. thesis, University of Twente, Enschede
9. Jackson D (2006) Software abstractions: logic, language, and analysis. MIT Press, Cambridge
10. Maintainers (2009) UML2Alloy. http://www.cs.bham.ac.uk/~bxb/UML2Alloy
11. Massoni T, Gheyi R, Borba P (2004) A uml class diagram analyzer. In: Third international workshop on critical systems development with UML. Affiliated with 7th UML conference, pp 143–153

12. Mylopoulos J (1992) Conceptual Modeling, databases, and case: an integrated view of information systems development. In: Conceptual Modeling and Telos, chap. Wiley, Chichester, pp 49–68
13. OMG (2009) UML 2.2 superstructure specification. Technical report, Object Management Group (OMG)
14. Papazoglou MP, Krämer BJ (1997) A database model for object dynamics. VLDB J 6(2):073–096
15. Pastor O, Molina JC (2007) Model-driven architecture in practice: a software production environment based on conceptual modeling. Springer, New York
16. Schinz I, Toben T, Mrugalla C, Westphal B (2004) The rhapsody uml verification environment. In: SEFM '04: Proceedings of the software engineering and formal methods. Second international conference. IEEE Computer Society, Washington, DC, pp 174–183
17. USA Department of Defense (DoD) (2007) DoD modeling and simulation (M&S) management. Directive 5000.59