International Journal of Computer Science and Software Engineering (IJCSSE), Volume 6, Issue 10, October 2017
ISSN (Online): 2409-4285 www.IJCSSE.org Page: 223-232

Modeling Composition of UML Profiles with Alloy

Kleinner Farias[1], Toacy Oliveira[2], Lucian José Gonçales[3] and Vinicius Bischoff[4]

[1, 2, 3] Post Graduate Program in Applied Computing (PPGCA), University of Vale do Rio dos Sinos (UNISINOS), São Leopoldo, RS 93.022-750, Brazil.

[4] Computation and Systems engineering Program (PESC/COPPE), Federal University of Rio de Janeiro (UFRJ), Rio de Janeiro, RJ, Brazil.

[1]mkleinnerfarias@unisinos.br, {[2]lucianj, [3]viniciusbishoff}@edu.unisinos.br, [4]toacy@cos.ufrj.br

ABSTRACT

Global software development teams can collaboratively create Unified Modeling Language (UML) profiles, the primary mechanism for defining domain-specific variants on top of the UML. Usually, parts of UML profiles are separately elaborated to speed up the UML tailoring process, but at sometimes the parts built in parallel need to be brought together to construct a full UML profile. Although many model composition techniques have been proposed in the last decades, no one deals with issues required to combine UML profiles, e.g., matching and integration of UML stereotypes. Consequently, little is known about how to support the composition of UML profiles. Even worse, academia and industry have overlooked the elaboration of composition methods to support the integration of UML profiles, as well as the formal representation of such methods. This study, therefore, presents a composition mechanism, as well as introduces a lightweight UML extension to support the specification of composition relationships between UML profiles. The semantics of the extension and mechanism proposed was carefully represented in Alloy, a formal modeling language based on first-order logic. Then, we used the Alloy Analyzer to check the specification generated in Alloy for some specific algebraic properties, including idempotency, uniqueness, commutativity, and associativity.

Keywords: *Alloy, Model Composition, UML, UML profile.*

1. INTRODUCTION

The Unified Modeling Language (UML) is a general purpose visual modeling language for specifying, constructing and documenting the artifacts of software systems that can be used with all major application domains and implementation platforms [1]. It has been widely used and adopted by both industry and academia as a standard the modeling language for describing software systems. In [2], Chaudron and colleagues reveal that the UML is the de facto standard for object-oriented software, and has been widely used for representing design designs through a multi-view approach. According to [1], the UML seeks to advance the state of the industry by enabling object visual modeling tool interoperability. For this, the UML specification provides a set of the human-readable notation elements, as well as providing lightweight mechanisms to fit it into particular domain application.

Specific platform and domain application need terminology, different notations, constraints and semantics in which the UML is often unable to represent them using its default elements. Thus, UML provides extension mechanisms, e.g., UML profiles, that come up with capabilities that allow metaclasses to be extended to adapt them for different purposes [1], including the ability to tailor the UML metamodel for different platforms (such as Python or NodeJS) or domains (such as Health care, Educational, finance, or even real-time, data-intensive applications).

In practice, UML profiles are formed by a set of stereotypes, tagged values, and constraints, which are able to tailor UML elements to fit the needs of specific platform or domain, preserving the semantics of the default elements. Examples of UML would be: RE-UML, a profile for component-based system requirements analysis [3]; SysML, a general-purpose modeling language for systems engineering [4]; UML2TP, a profile for designing, visualizing, specifying, analyzing, constructing, and documenting the artifacts for model-based testing approaches [5].

Profile composition can be defined as a set of activities to be performed over two input profiles, P_A (the receiving profile) and P_B (the merged profile) to produce an output-composed model, P_{AB}, the resulting profile. In other words, the composition of P_A and P_B can be understood as the integration of the content of P_A and P_B so that P_{AB} can be produced.

In global software development context, for example, separate development teams may concurrently work on partial views of an overall UML profile for allowing team members to concentrate on the parts more relevant to them. However, at sometimes the parts created in parallel need to bring together to create a full UML profile. For this reason, many model composition techniques have been proposed in the last decades [6, 7], e.g., Epsilon [8], MATA [9, 10], MoCoTo [11], and Kompose [12]. Nevertheless, none of them deals with issues required to combine UML profiles, e.g., matching and integration of UML stereotypes. Although the use of UML profile has widely increased in different research areas (e.g., [5, 13, 14, 15]), a formal semantics is still severally lacking.

Consequently, little is known about how to support the composition of UML profiles. Even worse, academia and industry have overlooked the elaboration of composition methods to support the integration of UML profiles, as well as the formal representation of such methods. In [16], the authors reinforce that while model transformation has been researched and well documented and achieved important results in the field of model-driven software development, model composition needs further investigations and efforts to support the composition of domain-specific languages. In addition, given that profile composition can been as an operation, nothing has been done to evaluate whether it holds some algebraic properties This study, therefore, presents a composition mechanism, as well as introduces a lightweight UML extension to support the specification of composition relationships between UML profiles. The semantics of the extension and mechanism proposed was carefully represented in Alloy [17, 18], a formal modeling language based on first-order logic. That is, we explain how UML profile metamodel can be specified in Alloy. Then, we used the Alloy Analyzer3 to check the specification generated in Alloy for some specific algebraic properties, including idempotency, uniqueness, commutativity, and associativity. We have chosen the Alloy language because is based on set theory, first order logic, and is strongly in influenced by object-oriented modeling notations. It is the input language to Alloy Analyzer (the tool support), which has embodied an over-the-shelf SAT solver. The Alloy Analyzer is a constraint solver, which translates constraints to be solved from Alloy into 3Alloy Analyzer: http://alloy.mit.edu/alloy/boolean constraints [18, 17].

Additionally, the Alloy language and its tool support have been successfully used for modeling: (i) aspect oriented models [19]; (ii) composition of UML models [20]; (iii) formalization of object oriented models [21, 22]; (iv) modeling critical system, including air-traffic control [23] and a proton therapy machine [24].

Contributions of this Study. Formal specification and automated analysis of UML profile metamodel and profile composition mechanism involve answering several questions. What criteria should we use for analyzing them? How can we analyze these criteria? Once UML profile metamodel is defined using natural language, how can we define it formally? How can we identify correspond parts between input profiles? What activities should we perform to merge profiles? Our contributions are derived from the answers of these questions. In particular, some contributions are listed as follows:

1. A extension of UML profile metamodel. We provide a simple extension, inserting some specific constructs into UML metamodel to support the expression of composition relationship. These constructors allow assigning composable features to profile metamodel constructs.
2. A formal semantic for UML profile metamodel. We translate UML profile specification in Alloy. Since we can analyze UML profile based on the metamodel level, our analysis can be extended to other profiles.
3. A profile composition mechanism and its formalization. We provide a composition mechanism based on strategies, including matching and composition one. Moreover, we verify some algebraic properties of this relationship. Thus, readers can make use of this mechanism knowing its properties and using it in better way.

The remainder of the paper is organized as follows. Section 2 describes Alloy modeling language. Section 3 describes the proposed extension of the UML profile metamodel. Section 4 presents the modeling of the proposed composition mechanism of UML Profiles with Alloy. Section 5 contrasts our work with the current literature. Finally, Section 6 presents some concluding remarks and future work.

2. ALLOY LANGUAGE IN A NUTSHELL

Alloy is a formal modeling language strongly typed, based on first-order logic, and set theory. It was deeply inspired on formal language Z [25] and influenced by object modeling notations logic. With Alloy and Alloy Analyzer, it is possible to represent models through graphical and textural structures. The Alloy Analyzer can generate a model diagram from an Alloy textual model, so we can use this feature to help understand large models, or to see how a model grows as new signatures are added. An Alloy model consists of the following elements [19, 17]:

1. *Signature*. It defines a set of atoms and can have a collection of declaration of relations represented as fields over defined sets of signatures. It is similar to classes in object-oriented language. Signature can extend other signature and has fields (attributes in object orientation paradigm). It is used for defining new types.

2. *Facts*. The constraints of a model that should be always held are recorded as facts. In other words, they are constraints on fields and signature. A model can have any number of facts, being identified by the keyword fact. For instance, the constraints defined in UML metamodel may be expressed by facts.
3. *Functions*. They are similar to methods in object-oriented language. Thus, they have zero or more declarations for parameters, and a declaration to specify the types of return parameter.
4. *Predicates*. They are actually functions zero or more declarations for parameters, which can return only Boolean values. They are used for expressing constraints, which can be applied for a model when needed. With predicates, we can check whether instances of a model satisfy their built-in constraints.
5. *Assertions*. An assertion is a constraint that is intended to be valid. The Alloy Analyzer checks whether this constraint is always held. Otherwise, the Alloy Analyzer returns a counterexample, in which the constraint does not hold. Using assertion, for instance, we can check whether the composition can produce a profile as a result.
6. *Module*. The Alloy models can be grouped into modules. These modules are similar to package in object-oriented language. Once a module has been defined, it can import other modules to access their contents.

The Alloy Analyzer is a constraint solver, which translates constraints to be solved from Alloy into Boolean constraints, as aforementioned. It makes two kinds of analyses, such as: (i) check for satisfying instance of the model (i.e., once defined a profile, does it represent a sound UML profile metamodel instance?); (ii) search for scenarios in which assertions are not held. Due to the undecidability of such analysis, they are parameterized by a scope, which limits the size of instances considered. Since the search for a solution is limited by a scope, the absence of an instance does not automatically show that a formula is inconsistent [17].

3. EXTENDING THE UML METAMODEL

The UML profile metamodel is defined according to the UML metamodeling approach. Thus, a metamodel is used for specifying its syntax and semantics. The typical role of a metamodel is to define the semantics for how the elements of UML profiles get instantiated. In this work, we present a simple extension for UML metamodel and consider a simplified version of the UML profile metamodel shown in Figure 1. The simplification and UML extension are both explained, as follows:

1. Alloy can express multiple inheritances. In the UML metamodel, the classes Property and Parameter have an inheritance relationship with *MultiplicityElement* and *TypedElement*. To simply, we combined *MultiplicityElement* and *TypedElement*, creating a new metaclass, so-called *TypedMultiplicityElement* [20].
2. Alloy does not support recursion. This implies, for example, that we cannot represent classes contained by other classes; likewise, profiles that are contained by other profiles.
3. We extend the UML metamodel to denote our approach. For this, *ComposableElement* extends Element (from UML) and represents the profile elements that can participate of a composition relationship. They are Class, Association, Enumeration, Parameter, Stereotype, Package, among others. For each *ComposableElement* is defined a merge rule, which is responsible for merging it. *CompositeElement* represents the *ComposableElement* that contains another *ComposableElement*. For instance, Profile contains: (i) *ownedMembers*, that are *CompositeElement*; (ii) *ownedStereotypes*, that are Stereotypes. The extension is based on the Composite pattern that allows you to build complex objects by recursively composing similar objects in a three-like manner. This pattern was previously described by Erich Gamma and colleagues in [26].

This simplification is due to Alloy suffers from poor performance when analyzing models with many signatures and fields. Because the problem is NP-complete one. We do not remove too much detail in order to ensure that our analysis produces meaningful results. Moreover, the small scope hypothesis" [17] asserts that if an assertion is invalid, it probably has a small counterexample (in a small scope).

4. MODELLING ALLOY

This section presents the modeling of UML profile composition using Alloy. For this, Section 4.1 introduces the modeling UML profile metamodel in Alloy. Next, Section 4.2 describes the composition mechanism in Alloy. Then, Section 4.3 discusses the analysis of the composition mechanism using Alloy.

4.1 Modelling UML Profile in Alloy

We represent the meaning of the elements of UML profile metamodel using Alloy models. After mapping the proposed UML profile metamodel (Figure 1) into the Alloy constructs (described in 2), an automatic analysis might be done using Alloy Analyzer. Due to space constraints, we show only part of the modeling. The multiplicities 1, 0..1, 0..* and 1..* found in UML were mapped to the following Alloy key-words one, lone, set and some, respectively. The associations between UML profile metamodel elements are translated into relations (fields) in Alloy. The modeling of stereotype and profile is showed as follows.

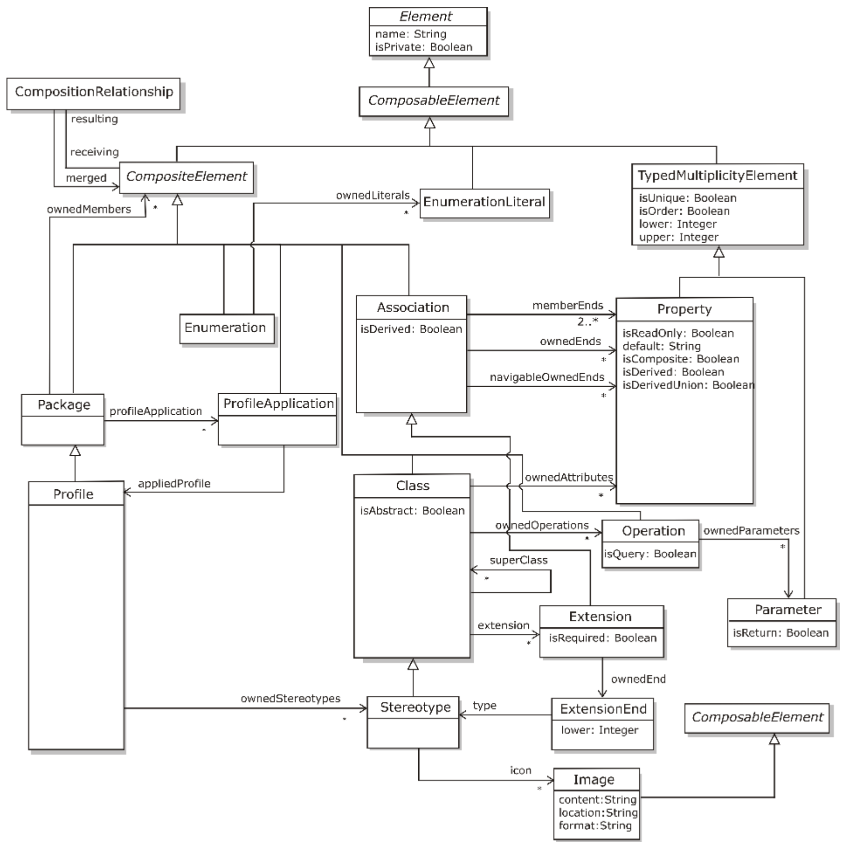

Fig. 1. Simplified UML profile metamodel and an extension for UML metamodel.

Stereotype. It is modeled (see Code 1) as a signature that extends the signature Class (line 1). The field icon represents the association between Stereotype and Image (line 2). We denote some constraints applied to Stereotype, as follows: (i) a Stereotype may only generalize or specialize another Stereotype (lines 3); (ii) so that all Image owned by a Stereotype to be distinguishable, they must have unique content, location and format (lines 4-5); (iii) each Stereotype must be owned by exactly one profile (line 6); (iv) Stereotype names should not clash with keyword names for the extended model element (lines 7-8).

```
1 sig Stereotype extends Class {
2   icon: set Image }{
3   all a: Stereotype | this.superClass in Stereotype
4    all a, b: icon | (a.@content = b.@content &&
5      b.location = b.location && a.format = b.format) => a = b
6   one p: Profile | this in p.ownedStereotypes
7   all a: Element, b:Stereotype |
8    (a not in Stereotype) => (b.@name = a.@name)
9  }
```

Code 1 : Stereotype in Alloy

Profile. It is modeled (see Code 2) as a signature that extends the signature Package (line 1) and has three fields: the *ownedStereotypes* represents the association between Profile and Stereotype (line 2); *ownedMembers* and *profileApplication* that are inherited from Package. One or more profiles may be applied at will to a package that is created from the same metamodel, which is extended by the profile. For this, *profileApplication* defines each profile that is applied to a Package [1]. So, each profile, that is applied to a Package, must be owned by exactly one profile (line 4).

```
1 sig Profile extends Package {
2   ownedStereotypes: set Stereotype
3 }{
4   one p: ProfileApplication | this in p.appliedProfile
5 }
```

Code 2: Profile in Alloy

4.2 Composition Mechanism in Alloy

For merging two profiles is needed to: analyzing the input profile elements in order to verify if they are models of valid type (Step 1); defining what profile elements are equivalent (Step 2); merging the equivalent profile elements based on a match strategy and composition strategy (Step 3).

*Step 1***.** The input profile elements are basically checked if they are some *ComposableElement* valid. For example, in the *matchOperator* predicate (see Code 3) is checked if a and b are *ComposableElement* (line 1).

*Step 2***.** Composition requires that two *ComposableElements* are equivalent, thus we created the *matchOperator* (see Code 3) for computing such equivalence. It takes two *ComposableElements* and one match strategy as parameters, and returns true if, and only if, they are equivalent. We define three kinds of match strategy that contrast between them by the number of syntactic properties of UML profile metamodel elements, which they take into account during the comparison of two elements, such as: default, only name of elements is considered; (ii) partial, a set of syntax property is considered; (ii) complete, all syntax property is considered. The *matchOperator* predicate has three parameters (line 1{2), if c is instance of *DefaultMatchStrategy*, then a and b will be passed as parameter to the predicate *defaultMatchStrategy* (line 3). In line 4 and 5, it acts similarly.

```
1 pred matchOperator(a: ComposableElement, b: ComposableElement,
2   c: MatchStrategy){
3   (c in DefaultMatchStrategy && defaultMatchStrategy[a, b]) ||
4   (c in PartialMatchStrategy && partialMatchStrategy[a, b]) ||
5   (c in CompleteMatchStrategy && completeMatchStrategy[a, b])
6 }
```

Code 3: Verifying equivalence

Step 3. We use composition strategy for determining how the composition must be performed. For each defined composition strategy, there is a set of merge rules that merge the profile elements according to the strategy. The strategies are: override, union, and merge. (i) (override specifies that every receiving profile element must override their equivalents in the merged profile. Each merged profile element that does not have equivalent in the receiving profile are copied to output model without modification; (ii) union defines that every receiving and merged profile elements must be added to output model; (iii) merge defines that every equivalent profile elements must be combined to obtain a integrated view of them. We present only override strategy in Alloy. The predicate, *overrideStrategyMerge* (see Code 4), has three *ComposableElement* as parameters: a is the receiving profile element; b is the merged profile element; c represents the resulting profile element, the result of merging between a and b (line 1-2). The predicate returns true if, and only if, the input profile elements are Class, Association, Stereotype, Enumeration, or Interface, and their merge rules return true (e.g., *stereotypeOSMergeRule* would be a stereotype merge rule).

```
1  pred overrideStrategyMerge(a: ComposableElement,
2    b: ComposableElement, c: ComposableElement){
3    (a in Stereotype  && b in Stereotype &&
4      stereotypeOSMergeRule[a, b, c]) ||
5    (a in Class && b in Class &&
6      classOSMergeRule[a, b, c]) ||
7    (a in Association && b in Association &&
8      associationOSMergeRule[a, b, c])  ||
9    (a in Enumeration && b in Enumeration &&
10     enumerationOSMergeRule[a, b, c]) ||
11   (a in Interface   && b in Interface   &&
12     interfaceOSMergeRule[a, b, c])
13 }
```

Code 4: Override strategy in Alloy

Moreover, the *compositionRelationship* predicate (see code 5) is responsible for merging receiving and merged profile according to particular composition strategy. It returns true if, and only if, the composition is valid. Otherwise, it returns false. For this, the merge predicate (line 6) must return true.

```
1 pred compositionRelationship(
2   receiving: CompositeElement,
3   merged: CompositeElement,
4   resulting: CompositeElement,
5   strategy: CompositionStrategy ){
6   merge[receiving, merged, resulting, strategy]
7 }
```

Code 5: Composition relationship in Alloy

4.2 Alloy Analysis of Model Composition Mechanism

Each analysis performed involves solving a constraint, finding an instance or a counterexample. An instance is a scenario in which both the facts and the predicates hold. On the other hand, a counterexample is an instance in which the facts hold, but the assertion does not, or the facts do not hold, so assertion fails to follow from the facts. For finding an instance or a counterexample, the Alloy Analyzer assigns values to the variables of the constraint, and then evaluates for *true* or *false*.

*Checking the algebraic properties***.** Having translated UML extension for model composition and the model composition operators in terms of the objects manipulated, we now check some algebraic properties. We may create some expectations about the composition of UML profiles. For example, should composing a model with itself return the same model? Knowing what algebraic properties the composition relationship holds, it is useful information for software designer, as they can use composition relationship more systematically, rather than based on intuition. Thus, the goal of our analysis was verifying whether the composition mechanism holds some algebraic properties and automatically finding valid snapshots of models. We checked our composition mechanism for the following properties described below (m_a, m_b and m_b are models of the same type):

∀ *receiving, merged, resulting: Profile,*

```
assert mergeOperatorIsIdempotency {
  all a, c : Profile | all match: DefaultMatchStrategy |
  all strategy: OverrideStrategy |
  mergeOperator[a, a, c, match, strategy] =>
  mergeOperator[a, a, c, match, strategy]
}
```

Code 6: Idempotency property in Alloy

∀ *match: MatchStrategy, 8 strategy: CompositionStategy*
merge(receiving, merged, resulting, match, strategy) =
merge(receiving, merged, resulting, match, strategy)

In our Alloy formalization, the property is expressed by the Code 6, in which *mergeOperator* is the predicate responsible for checking the composition. *DefaultMatchStrategy* and *OverrideStrategy* specify a particular match strategy and a merge strategy, respectively.

Uniqueness: this property verifies if the composition of two models, profile A and profile B, generates one possible output model. The composition relationship should hold this property; otherwise, it indicates that there are some problems (e.g., improper specification of merge rules) in the definition of the mechanism. Applying the uniqueness property is possible to verify problems with the formal model, since any counterexample will indicate problems with the formal model. In our approach, this property is expressed in predicate logic as:

```
assert  mergeOperatorIsUnique {
  all receiving, merged, resultingA, resultingB: Profile |
  all match: PartialMatchStrategy |
  all strategy: OverrideStrategy |
  mergeOperator[receiving, merged, resultingA,
    match, strategy] &&
  mergeOperator[receiving, merged, resultingB,
    match, strategy] =>
  matchOperator[resultingA, resultingB, match]
}
```

Core 7: Uniqueness property in Alloy

∀ *receiving, merged, resultingA, resultingB: Profile,*
∀ *match: MatchStrategy, 8 strategy: CompositionStategy*
merge(receiving, merged, resultingA, match, strategy) ∧
merge(receiving, merged, resultingA, match, strategy) →
match(resultingA, resultingB, match)

In Alloy, we express it through Code 7, in which the mergeOperator is the predicate responsible for checking the composition, *PartialMatchStrategy* and *OverrideStrategy* specify a particular match strategy and a particular merge strategy, respectively.

Commutativity: this property is a widely used mathematical term that refers to the ability to change the order of something, without changing the result produced. Once the property holds, the composition relationship may be established in any direction between the models. For a given binary function ***f:D×D->K***, it is said to be commutative if, and only if, ***f(x,y) = f(y,x)*** for every ***x, y*** ∈ ***D***. This property is expressed in predicate logic as:

```
assert mergeOperatorIsCommutative {
  all a, b, c : Profile
   | all match: DefaultMatchStrategy
   | all strategy: OverrideStrategy |
  mergeOperator[a, b, c, match, strategy] =>
  mergeOperator[b, a, c, match, strategy]
}
```

Code 8: Commutative property in Alloy

∀ *receiving, merged, resulting: Profile,*
∀ *match: MatchStrategy, 8 strategy: CompositionStategy*
merge(receiving, merged, resulting, match, strategy) =
merge(merged, receiving, resulting, match, strategy)

In Alloy, we represented it through Code 8, in which *mergeOperator* is the predicate responsible for checking the composition, *PartialMatchStrategy* and *OverrideStrategy* specify a particular match strategy and a particular merge strategy, respectively.

Associativity: The associative property is closely related to the commutative property. In this property, the order of operations does not matter as long as the sequence of the operands is not changed. Again, this property is an important for composition mechanism should have, when it is used to build domain specific language. In short, even though the models were rearranged, the result of the composition is not altered. Formally, a binary operation f on a set D is associative if, and only if, it satisfies the associative law*: f(f(x,y),z) = f(x,f(y,z))* for all x, y, z ∈ D. This property is expressed in predicate logic as:

∀ *receiving, merged, mergedA, resulting, resultingA, resultingB : Profile,*

```
assert mergeOperatorIsAssociative {
  all receiving, mergedA, mergedB,
  resulting, resultingA, resultingB: Profile |
  all match: DefaultMatchStrategy |
  all strategy: OverrideStrategy |
  mergeOperator[receiving, mergedA, resultingA,
    match, strategy] &&
  mergeOperator[resultingA, mergedB, resulting,
    match, strategy] &&
  mergeOperator[receiving, mergedB, resultingB,
    match, strategy] => mergeOperator[resultingB,
    mergedA, resulting, match, strategy]
}
```

Code 9: Associative property in Alloy

∀ *match: MatchStrategy, 8 strategy: CompositionStategy*
merge(merge(receiving, merged, resultingA, match, strategy),
mergedA, resulting, match, strategy) =
merge(receiving,
merge(merged, mergedA, resultingB, match, strategy),
resulting, match, strategy)

Code 9 represented it in Alloy, in which *mergeOperator* is the predicate responsible for checking the composition, *PartialMatchStrategy* and OverrideStrategy specify a particular match strategy and a particular merge strategy, respectively.
As the use of UML profiles for building domain specific languages substantially grows, so it stimulates, consequently, the need for manipulating them and encouraging the relationship for each other. For instance, the UML specification [1] defines some relationship such as: import, merge, apply, and so on. However, the lack of standardized formal semantics for the language does not stimulate the development of tools supporting automatic analysis and verification of the profiles. The analyses were performed using version 4.0 of the Alloy Analyzer. Tab. 1 shows the result of the analysis of profile composition based on merge (the composition strategy) according to three match strategy. As good way to use Alloy Analyzer is to start with a small scope analysis [17], so our analysis was limited to scope 2. Thus, it showed that the composition relationship is idempotency, uniqueness, commutativity and associativity for the override composition strategy. The time execution of the analysis is also showed in Tab. 1.
Nevertheless, in [15] the authors proposed an UML profile that provides specific components and stereotypes for representing data security in web applications. Furthermore, other studies, e.g., [44], come up with a general-purpose modeling language to customize the UML for systems engineering applications. In short, all these approaches supply UML meta-model extension used to improve the UML capability in expressing domain specific concepts, however, none of them take into account formal aspects of the extensions or use formal language to formalize and analyze their characteristics and limitations. Related approaches have been developed for similar purposes [45, 46].

5. RELATED WORKS

The model composition has a central role in the Model Driven Engineering being applied to address significant problems in many research areas such as database integration [27, 28], aspect oriented modeling [29, 30, 52], merging source code [31, 32], composition of web services [33, 34], UML extension for model composition [35, 36], model transformation [37, 38], model comparison [39, 40, 49], model composition [11, 41, 47], model stability [48], and composition effort [50, 51, 53].
Even so, it still needs more investigation and efforts to (1) fulfill the lack of a formalization in composition of UML Profiles, (2) providing systematic and clear semantics to compose the UML Profiles, and (3) simplify the UML profile metamodel. Despite of some works [7, 42] focused on make use of operations on the design models, i.e., the merge, override, and union, specifically, none of them has applied it in the context of composition of UML profiles. Instead, composition operations are utilized in [7] as guidelines for developers compose input models to measure the effort, and in [42] they were utilized to evaluate the developers' comprehension.
For this, in this work, we specify three kind of composition strategy [43], which are implemented by merge rules, and determine the lacking formal semantic for them. Considerable researches have been done in the area of domain specific language to deal with platforms (such as Python, Java or .Net) or application domains (such as business or development process modeling) at a abstraction high-level in order to suitably handle them. In [13], the authors presents an UML profile called DICE. This profile provides personalized components that enables to represent specific features of Big Data applications properly. Moreover, in [14] the authors provided a UML profile in order to support the development of database applications.

		Match Strategy								
		Default			Partial			Complete		
		Verify	Scope	Time	Verify	Scope	Time	Verify	Scope	Time
Properties	Idempotency	Yes	2	27m03s	Yes	2	30m18s	Yes	2	31m07s
	Uniqueness	Yes	2	12m05s	Yes	2	13m07s	Yes	2	14m41s
	Commutativity	Yes	2	22m15s	Yes	2	24m01s	Yes	2	21m09s
	Associativity	Yes	2	45m27s	Yes	2	40m05s	Yes	2	39m17s

Tab 1: The result of algebraic properties analysis

In [45], a verification approach for UML Class Diagrams is presented. The Alloy is applied in order to verify whether Class Diagrams are in compliance with UML metamodel properties. For this, the specifications in alloy are translated for enabling high-level modeling of object-oriented systems. However, this work does not define an simplified meta- model, neither composes UML profiles. At first glance, our approach differs from [46] in the sense that we model and verify UML profile metamodel, instead of state machines. In contrast, our approach specifies a formal model both of UML profile metamodel and profile composition mechanism. To sum up, none of the proposed works investigated (1) the lack of formalizations in the composition process of UML Profiles, (2) provided a clear semantics to compose the UML Profiles, or even (3) simplified the UML profile metamodel.

5. CONCLUSION

This paper presented an extension of UML profile metamodel, a composition mechanism and their formalization in Alloy. Some algebraic properties are listed and used to analyze the composition mechanism. Moreover, we explain how UML profiles can be specified in Alloy and how composition of UML profiles can be verified in the Alloy Analyzer. An initial UML extension was also presented in order to satisfy needed of composition mechanism. We argued that to create a profile is as important as to provide a mechanism that should be able to put together these profiles from different profiles by formal view. The analysis provided in this paper is sound but not complete. Since the form of analysis that underlies Alloy has limitations. As Alloy's relational logic is undecidable, the Alloy Analyzer is not able to infer, with perfect reliability, whether an assertion is valid for every possible assignment. For example, when scopes defined it limits the size of instances considered to make instance finding feasible. However, if no counterexample is found, nothing can be inferred. There is a considerable interest in academia, the numerous conferences and workshops devoted to this topic have increased, and industry in domain specific language, in particular to profiles. However, any initiative to create a formal approach of these UML variants. Alloy allowed us to formalize the UML profile metamodel and the composition mechanism operation. Some properties of the composition mechanism were analyzed in order to improve its knowledge and using. We observe that use of formal modeling language may pave the way towards a better formalization and understanding of modeling language. So, we suggest the definition of a semantic formal to all elements specified in UML metamodel. Lastly, the issues outlined throughout the paper may encourage other researchers to explore our study, as well as develop innovative techniques to minimize the side-effects of improper composition of software design models.

ACKNOWLEDGMENTS

Insert acknowledgment, if any. The preferred spelling of the word "acknowledgment" in American English is without an "e" after the "g." Use the singular heading even if you have many acknowledgments. Avoid expressions such as "One of us (S.B.A.) would like to thank ... ." Instead, write "F. A. Author thanks ... ." Sponsor and financial support acknowledgments are also placed here.

REFERENCES

[1] OMG. Unified Modeling Language: Infrastructure, version 2.5. USA: Object Management Group, 2015.
[2] M. Chaudron, W. Heijstek, and A. Nugroho, "How effective is UML modeling?", Software and Systems Modeling, Vol. 11, 2012, pp. 571-580.
[3] S. Mahmood, and R. Lai, "Re-UML: a component based system requirements analysis language", The Computer Journal, Vol. 56, No. 7, 2012, pp. 901-922.
[4] OMG. Systems Modeling Language, version 1.4. USA: Object Management Group, 2015.
[5] OMG. UML Testing Profile, version 1.2., USA: Object Management Group, 2013.
[6] K. Farias, "Empirical Evaluation of Effort on Composing Design Models". PhD thesis, Department of Informatics, PUC-Rio, Rio de Janeiro, Brazil, 2012.
[7] K. Farias, A. Garcia, K. Whittle, C. Flacha Garcia Chavez, and C. Lucena, "Evaluating the effort of

composing design models: a controlled experiment", Software & Systems Modeling, Vol. 14, pp. 1349-1365, 2014.

[8] D. Kolovos, L. Rose, A. García-Domínguez, and R. Paige, The Epsilon Book, URL: http://eclipse.org/epsilon/doc/book/, 2015.

[9] J. Whittle P. and Jayaraman, "Synthesizing hierarchical state machines from expressive scenario descriptions", ACM Transactions on Software Engineering Methodology, Vol. 19, No. 3, pp. 8:45, 2010.

[10] J. Whittle, P. Jayaraman, A. Elkhodary, and J. A. Ana Moreira, "Mata: A unified approach for composing UML aspect models based on graph transformation", Transactions on Aspect-Oriented Software Development, 2009.

[11] K. Farias, L. Gonçales, M. Scholl, T. Oliveira, and M. Veronez, "Toward an Architecture for Model Composition Techniques." in 27th International Conference on Software Engineering and Knowledge Engineering, 2015, pp. 656-659.

[12] F. Fleurey, R. Reddy, R. France, B. Baudry, S. Ghosh, and M. Clavreul "Kompose : A generic model composition tool", 2015.

[13] A. Gómez, J. Merseguer, E. Di Nitto, and D. A. Tamburri, "Towards a UML profile for data intensive applications" in proceedings of the 2nd International Workshop on Quality-Aware DevOps, 2016, pp. 18-23.

[14] P. Oleynik, and V. I. Gurianov, "Uml-profile for metamodel-driven design of database applications", Journal of Computer Science, Vol. 3, No. 11, 2016.

[15] T. Basso, L. Montecchi, R. Moraes, M. Jino, and A. Bondavalli, " Towards a uml profile for privacy aware applications", in IEEE International Conference on Computer and Information Technology,2015 , pp. 371-378.

[16] J. Bézivin, S. Bouzitouna, M. Fabro, M. P. Gervais, F. Jouault, and D. Kolovos, "A Canonical Scheme for Model Composition." in European Conference on Model Driven Architecture - Foundations and Applications, 2006, pp. 346-360.

[17] D. Jackson, Software Abstraction: Logic, Language, and Analysis. MIT Press, 2006.

[18] D. Jackson. "Alloy: a Lightweight Object Modelling Notation", ACM Transactions on Software Engineering and Methodology, Vol. 11, pp. 256-290, 2002.

[19] F. Mostefaoui, and J. Vachon. "Verification of aspect-uml models using alloy", in proceedings of the 10th International Workshop on Aspect-Oriented Modeling, 2007, pp. 41-48.

[20] A. Zito, J. Dingel, "Modeling uml 2 package merge with alloy", in proceedings of the 1st Alloy Workshop, 2006, pp. 154-164

[21] T. Massoni, R. Gheyi, and P. Borba, "A UML Class Diagram Analyzer", in Third Workshop on Critical Systems Development with UML, 2004, pp. 100-114.

[22] R. Bourdeau, and B. Cheng, "A Formal Semantics for Object Model Diagrams". IEEE Transactions on Software Engineering, 1995, pp. 799-821.

[23] G. Dennis, "TSAFE: Building a Trusted Computing Base for Air Traffic Control Software". M.S. thesis, MIT, USA, 2003.

[24] G. Dennis, R. Seater, R. Rayside, and D. Jackson, "Automating Commutativity Analysis at the Design Level", in proceedings of ACM SIGSOFT International Symposium on Software Testing and Analysis, 2004, pp. 165-174.

[25] ISO, Information Technology - Z Formal Specification Notation - Syntax, Type System and Semantics. International Standard ISO/IEC 13568, 2002.

[26] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[27] P. A. Bernstein, M. Jacob, J. Pérez, G. Rull, and J. F. Terwilliger, "Incremental mapping compilation in an object-to-relational mapping system", in proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, 2013, pp. 1269-1280.

[28] P. Shvaiko, and J. Euzenat, "Ontology matching: State of the art and future challenges", IEEE Transactions on Knowledge and Data Engineering, Vol. 25, No. 1, 2013, pp. 158-176.

[29] M. Wimmer, A. Schauerhuber, G. Kappel, W. Retschitzegger, W. Schwinger, and E. Kapsammer. "A survey on UML-based aspect-oriented design modeling" ACM Computing Surveys, Vol. 43, No. 4, 2011, pp.28-33.

[30] K. Oba, H. Wada, and J. Suzuki, "Leveraging early aspects in end-to-end model driven development for non-functional properties in service oriented architecture" Journal of Database Management, Vol. 22, 2011, pp.93-123.

[31] T.C. Lethbridge, V. Abdelzad, M. Husseini Orabi, A. Husseini Orabi, and O. Adesina, "Merging Modeling and Programming Using Umple", In Margaria, T. and Steffen, B. (eds.), Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications: 7th International Symposium, 2016, p. 187-197.

[32] Y. Nishimura, and K. Maruyama, "Supporting merge conflict resolution by using fine-grained code change history" in IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, 2016, pp. 661-664.

[33] D. Wang, Y. Yang, and Z. Mi, "A genetic based approach to web service composition in geodistributed cloud environment", Computers & Electrical Engineering, Vol. 43, 2015, pp. 129-141.

[34] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "A survey on trust and reputation models for web services: Single, composite, and communities", Decision Support Systems, Vol. 74, 2015, pp. 121-134.

[35] A. N. Perez, B. Rumpe, S. Volkel, and A. Wortmann, "Composition of heterogeneous modeling languages", in Model-Driven Engineering and Software Development: Third International Conference, MODELSWARD, pp. 45-66, 2015.

[36] M. Misbhauddin, and M. Alshayeb, "Extending the uml use case metamodel with behavioral information to

facilitate model analysis and interchange", Software & Systems Modeling, Vol. 14, No. 2, 2015, pp. 813-838.

[37] D. Strüber, J. Rubin, T. Arendt, M. Chechik, G. Taentzer, and J. Plöger, "Rulemerger: automatic construction of variability-based model transformation rules", in International Conference on Fundamental Approaches to Software Engineering, 2016, pp. 122-140.

[38] K. Lano, and S. Kolahdouz-Rahimi, "Modeltransformation design patterns", IEEE Transactions on Software Engineering, Vol. 40, No. 12, 2014, pp. 1224-1259.

[39] L. Goncales, K. Farias, M. Scholl, T.C. Oliveira, and M. Veronez, "Model comparison: a systematic mapping study", in The 27th International Conference on Software Engineering and Knowledge Engineering, 2015, pp.546-551.

[40] L. Goncales, K. Farias, M. Scholl, M. Veronez, and T.C. Oliveira, "Comparison of design models: A systematic mapping study.", in International Journal of Software Engineering and Knowledge Engineering, Vol. 25, 2015, pp. 1765-1770.

[41] J. Rubin, and M. Chechik, "N-way model merging", in proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, 2013, pp.301-311.

[42] F. Ricca, M. Di Penta, M. Torchiano, P. Tonella, and M. Ceccato, "How developers' experience and ability influence web application comprehension tasks supported by uml stereotypes: A series of four experiments", IEEE Transactions on Software Engineering, Vol. 36, No. 1, 2010, pp. 96-118.

[43] S. Clarke, "Composition of Object-Oriented Software Design Models", PhD thesis, School of Computer Applications, Dublin City University, Dublin, Irland, 2001.

[44] S. Friedenthal, A. Moore, and R. Steiner, A practical guide to SysML: the systems modeling language, Morgan Kaufmann, 2014.

[45] A. Cunha, A. Garis, and D. Riesco, "Translating between alloy specifications and UML class diagrams annotated with OCL", Software & Systems Modeling, Vol. 14, No. 1, 2015, pp. 5-25.

[46] A. Garis, A. C. R Paiva, A. Cunha, and D. Riesco, "Specifying UML Protocol State Machines in Alloy". in Integrated Formal Methods: 9th International Conference, 2012, pp-312-326.

[47] K. Farias, "Composição de UML Profiles", Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul (PUC-RS), Rio Grande do Sul, Brasil.

[48] K Farias, A Garcia, C. Lucena, Effects of Stability on Model Composition Effort: an Exploratory Study, Software & Systems Modeling, 13(4):1473-1494, 2014.

[49] K. Oliveira, T. Oliveira, "Model Comparison: A Strategy-Based Approach", 20th International Conference on Software Engineering and Knowledge Engineering, pages 912-917, 2008.

[50] K. Farias, Empirical Evaluation of Effort on Composing Design Models, 32nd ACM/IEEE International Conference on Software Engineering-Volume 2, pages 405-408, 2010.

[51] K. Farias, A. Garcia, J. Whittle, C. Lucena, Analyzing the Effort of Composing Design Models of Large-Scale Software in Industrial Case Studies, In: Proceedings of the 16th International Conference on Model-Driven Engineering Languages and Systems (MODELS'13), pages 639-655, Miami, USA, September 2013.

[52] E. Guimaraes, A. Garcia, K. Farias, On the Impact of Obliviousness and Quantification on Model Composition Effort, In: Proceedings of the 29th Symposium On Applied Computing (SAC.14), Gyeongju, Korea, March, 2014.

[53] K. Farias, Analyzing the Effort on Composing Design Models in Industrial Case Studies, In: 10th International Conference on Aspect-Oriented Software Development Companion (AOSD'11), pages 79-80, Porto de Galinhas, Brazil, 2011.