Synthesising Call Sequences from OCL Operational Contracts*

Hao Wu
Maynooth University
haowu@cs.nuim.ie

ABSTRACT

The Unified Modeling Language (UML) is widely used by software engineers in different phases of software development cycle. It allows them to visualise and depict a system into different diagrams. Among these diagrams, UML class diagrams are used to capture the structure of a system including classes, attributes and associations. The set of operation calls defined in a UML class diagram then capture the behaviour of a system. These operation calls typically constrain the inputs and outputs via a set of pre or postconditions (operational contracts) written in Object Constraint Language (OCL). Hence, a sequence of operation calls conforming to pre or postconditions is crucial to analyse, verify and understand the behaviour of a system. In this paper, we propose a new technique for synthesising call sequences from a set of operational contracts. This technique works by reducing a synthesis problem to a satisfiability modulo theories (SMT) problem. The preliminary results show that our technique is capable of synthesising call sequences at a large scale.

CCS CONCEPTS

• **Software and its engineering → Model-driven software engineering**; **Unified Modelling Language (UML)**; **Software verification**;

KEYWORDS

UML, OCL, SMT Solver, Sequence Synthesis

ACM Reference Format:
Hao Wu. 2019. Synthesising Call Sequences from OCL Operational Contracts. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3297280.3297612

1 INTRODUCTION

The Unified Modelling Language (UML) is a general-purpose modelling language for visualising and building a software system at an abstract level. It provides users with different diagrams for modelling both structural and behavioural aspects of a system. For example, a class diagram is used to describe the relationships among entities or to model the transitions between states in a state machine. The Object Constraint Language (OCL) on other hand, provides a way of specifying necessary constraints that cannot be expressed by the diagrams. These constraints allow users to formally express rules over the structural and behavioural aspects of a system. For example, a class invariant is used to specify a property for all instances or a precondition to constrain inputs of an operation call.

Although the combination of UML and OCL makes models more formal, the tasks associated with verification or reasoning are quite challenging [4, 23]. Recently, numerous approaches have been proposed to tackle this challenge [3, 7, 21, 22]. Many of these focus on checking the consistency of a UML class diagram with a set of OCL constraints (structural aspects of a system) whilst a few of them consider operational contracts (behavioural aspects of a system) [18, 19]. In general, operational contracts are specified as pre/postconditions of an operation call and they capture constraints over different behaviours and system states. In this paper, we present an approach that uses satisfiability modulo theories (SMT) to synthesise call sequences from a set of OCL operational contracts. The synthesised call sequences are very helpful for users to verify or test a software system at an abstract level.

In this paper, we propose a novel technique that reduces synthesising call sequences from OCL operational contracts to a satisfiability modulo theories (SMT) problem. Our technique works by encoding OCL operational contracts into a set of first order formulas that can be efficiently solved by an SMT solver [9]. More specifically, this paper reports our initial idea and preliminary work.

2 RELATED WORK

Exisiting approaches to generating/synthesising call sequences from OCL operational contracts are focusing on using filmstripping [12, 14]. In general, the filmstripping approach translates the source model into so-called flimstrip models. These models essentially are UML class diagrams annotated with OCL constraints. These constraints are frame conditions that specify the changes to the model. One of the major advantages using filmstrip is that the low level (SAT solver) is not explicitly exposed to users. Thus, it does not require users to have the knowledge at the solver level. However, the cost here is the substantial manual interaction at model level. Therefore, the filmstripping approach is suitable for very dedicated tasks.

Though the idea of tackling operational contracts via SMT solving is not new [18, 19], the existing SMT-based approaches are quite limited by proposed SMT encodings. In comparison, our proposed approach here aims to improve existing SMT based approaches by proposing a first-order encoding. Our preliminary results show that this encoding has a potential of allowing users to synthesising call sequences at a much larger scale.

*Produces the permission block, and copyright information

Cabot et al. propose a systematic approach that uses constraint progamming (CP) to program the verification of UML models annotated with OCL operational contracts into a constraint satisfaction problem [6, 13]. The main advantage is that CP provides a high-level language so that a particular constraint problem is programmable. Their approach provides a variety of properties to be checked. Instead of designing a number of different verification properties, our work focuses on improving existing SMT-based approaches by proposing a first-order encoding so that synthesising sequences at large scale is possible. In particular, our encoding possesses high expressiveness and flexibility meanwhile it maintains the performance.

Alloy as a model finding tool, is widely used in the modelling community for verifying UML class diagrams [15, 20]. However, much of the work uses Alloy as its basis focuses on verifying/solving structural constraints of a system [2, 10, 16]. For example, Kyriakos et, al. maps a range of OCL constructs to Alloy's specificaiton [2], and Kuhlmann et, al. integrates kodkod (Alloy's solving engine) into the USE modelling tool and this enables them to be able to verify and analyse UML models annotated different types of OCL constraints [20].

Other approaches have sought to formalise UML and OCL into different types of formalisms, including interactive theorem provers such as Isabelle and KeY [1, 5, 8]. For example, Brucker et, al. shows that OCL can be translated to high-order logic (HOL) and proved by using Isabelle while Kyas et, al. formalises UML models with OCL into PVS [17].

3 OUR PROPOSED APPROACH

In this section, we describe our propose approach to synthesising call sequences from OCL operational contracts. Formally, in order to synthesise a call sequence, the user must provide:

- a UML class diagram of the form $\langle C, A, \Gamma, \Delta \rangle$, where C is a set of classes, A is a set of associations, Γ defines a set of class invariants and Δ is a set of operation calls. Each operation $\delta \in \Delta$ specifies a set of pre/postconditions: *Pre* and *Post*.
- the length k of a call sequence, where k is the number of operation calls in a sequence.
- a set of user-specified properties P.

The objective here is to discover a sequence of operation calls $G_{\langle k,P \rangle}$, that has a length of k with properties P. Visually, such a sequence $G_{\langle k,P \rangle}$ can be viewed as follows:

$$G_{\langle k,P \rangle} : S_0 \xrightarrow{\delta_1} S_1 \xrightarrow{\delta_2} S_2 \xrightarrow{\delta_3} S_3 \cdots \dashrightarrow^{\delta_k} S_k$$

where each δ_i in a sequence represents an operation call selected from Δ, and each S_j is a state that is triggered by an δ_i in the previous state S_{j-1}. S_0 is the initial state and S_k is the final state that is derived by the kth operation call (δ_k) in a sequence. Note that δ_k is called in S_{k-1} and triggers S_k. Thus, a call sequence of length of k creates a total of $k+1$ states. In each state S_j, there is *exactly one* operation call δ_i invoked to make the transition to the next state S_{j+1}. Therefore, this transition captures all possible sequences that have a length of k. Our idea of searching for a sequence of operation calls with properties P is to reduce it to an SMT problem. Formally, we cast $G_{\langle k,P \rangle}$ to the following formula:

$$G_{\langle k,P \rangle} \overset{\text{def}}{=} \Phi \wedge \Psi \tag{1}$$

where Φ encodes all possible call sequences with a length of k and Ψ encodes a set of user-defined properties. Here, we define Φ as the synthesis constraints and Ψ as the property constraints. In general, the search space for finding a call sequence with respect to pre/postconditions is $|\Delta|^k$. We say a sequence is found/synthesised when an SMT solver successfully solves Formula 1 or a counterexample is discovered when the solver solves $\Phi \wedge \neg\Psi$.

The Φ in Formula 1 represents a set of formulas $\phi_{\langle i,j \rangle}$ and each encodes a δ_i at state j. Precisely, each $\phi_{\langle i,j \rangle}$ is defined as follows:

$$\phi_{\langle i,j \rangle} \overset{\text{def}}{=} Pre \wedge Post \wedge Inv \tag{2}$$

where *Pre* imposes a precondition on the objects from the previous state and *Post* constrains what needs to be achieved in the current state [1]. *Inv* encodes a set of constraints that indicate the class invariants (Γ) should hold at all states.

3.1 State-based Functions

Our idea of representing an object in each state is to use a state-based (uninterpreted) function. To be precise, we map each attribute and association defined in a class diagram to a *state-based* uninterpreted function F. In general, this F has an n-ary form:

$$F : T_1 \times T_2 \times \ldots \times T_{n-1} \times INT \longrightarrow T_n$$

The type of each argument depends on different features defined in a model except for the last one. Here, we define the last argument as an integer type that represents a state j. The return type of F also depends on a particular feature. We then use this F as one part of the encodings for *Pre*, *Post* and *Inv* in Formula 2. For example, we can define a function $F_{value} : INT \times INT \longrightarrow INT$. Then the formula $F_{value}(o, 2) = 10$ encodes an integer type attribute *value* of an object o at state 2 is assigned to a value of 10,

3.2 Collection

We allow unbounded collection data types by utilising *Array* theory defined in SMT. In particular, we encode collection data types such as *Bag* and *Set* directly into an *Array* and map operations over an *Array* at a particular state to a set of state-based uninterpreted functions. For example, we can use *Array* theory to encode a collection of *items* as follows:

$$items \overset{\text{def}}{=} (Bag\ \ INT) \quad \text{and} \quad (Bag\ \ T) \overset{\text{def}}{=} (Array\ \ INT\ \ T)$$

where *items* is a bag of integers specifying a particular object id, *INT* in (*Bag T*) indicates an index and T denotes a particular type. We now can further define a state-based uninterpreted function *Select* that allows us to select an element from a collection at a particular state.

$$Select : (Bag\ \ INT) \times INT \times INT \longrightarrow T$$

For example, $Select(items,\ 0,\ 1) = item0$ means that the first element in the collection *items* at state 1 must be *item*0.

4 EVALUATION RESULTS

Table 1 shows the initial evaluation results of our proposed approach. In general, we are able to synthesise a call sequence with respect to the pre/postconditions within *one* second if k (the length of a call sequence) is set to less than 50. Thus, in order to effectively

[1] If we consider state j as our current state.

Length	**Company** [11]		**TrafficLights** [19]		**Bank** [18]		**Marriage** [12]		**Onlineshop**	
	$o = 10, c = 30$		$o = 3, c = 9$		$o = 8, c = 24$		$o = 4, c = 12$		$o = 7, c = 8$	
	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size
$k = 60$	1.34	2717	0.20	602	5.78	1558	1.16	2579	1.11	792
$k = 80$	1.41	4956	0.34	812	8.57	2078	1.96	3456	1.42	1051
$k = 100$	3.93	6196	0.66	1024	16.82	2598	2.44	4315	2.29	1315
$k = 120$	7.47	7437	0.73	1246	19.24	3118	4.76	5175	2.54	1527
$k = 140$	8.51	8678	2.23	1442	32.93	3928	5.94	6035	3.04	1832

Table 1: The evaluation results of synthesis constraints for each model. k here denotes the length of synthesised call sequence (This also means that we have k+1 number of the states). o denotes the number of created objects and c indicates the number of pre/postconditions considered in each triggered state. The time unit is seconds. Size measures the number of generated formulas.

evaluate our approach, we set k from 60 to 140 with an interval of 20. As it can be seen, our technique is able to synthesise call sequences with a number of objects within a reasonable amount of time. Interestingly, we notice that the number of generated SMT formulas is not always proportional to the solving time. The performance is mainly affected by the types of constraints specified in pre/postconditions. For example, the *Company* model in Table 1 specifies multiple constraints over collection data types (sets) as pre/postconditions while the *Bank* model has additional numerical constraints for its pre/postconditions. In fact, we notice that *Bank* model imposes a great challenge on the SMT solver, though it produces relatively less formulas. This is because the operational contracts of this model are quite complex than other models. In particular, it models the transactions between different accounts and banks using quantifiers over numeric constraints such as all recipient's account receives a certain amount of money transfer.

5 CONCLUSION

In this paper, we propose a new way of synthesising call sequences by reducing it to an SMT problem. In the future, we plan to extend this proposed approach in three aspects: (1) Allowing much larger scale synthesis. Existing approaches are limited by generating call sequences at smaller scale. (2) Enable property based synthesis. This can allow users not only to synthesis call sequences but also with respect to different types of properties. (3) Optimise our first-order encoding and we are currently investigating a new technique that can help us to significantly reduce the number of formulas generated.

REFERENCES

[1] Wolfgang Ahrendt, Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt. 2007. KeY: A Formal Method for Object-Oriented Systems. In *Formal Methods for Open Object-Based Distributed Systems*. Springer Berlin Heidelberg, 32–43.

[2] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. 2007. UML2Alloy: A Challenging Model Transformation. In *ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems*. Springer, Nashville, TN, 436–450.

[3] Mira Balaban and Azzam Maraee. 2013. Finite Satisfiability of UML Class Diagrams with Constrained Class Hierarchy. *ACM Transcation on Software Engineering and Methodology* 22, 3, Article 24 (2013), 42 pages.

[4] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. 2005. Reasoning on UML Class Diagrams. *Artificial Intelligence* 168, 1-2 (2005), 70–118.

[5] Achim D. Brucker and Burkhart Wolff. 2009. Semantics, calculi, and analysis for object-oriented specifications. *Acta Informatica* 46, 4 (01 Jul 2009), 255–284.

[6] Jordi Cabot, Robert Clarisó, and Daniel Riera. 2014. On the verification of UML/OCL class diagrams using constraint programming. *Journal of Systems and Software* 93 (2014), 1–23.

[7] Manuel Clavel, Marina Egea, and Miguel Angel GarcÃŋa de Dios. 2009. Checking Unsatisfiability for OCL Constraints. *Electronic Communication of the European Association of Software Science and Technology* 24 (2009).

[8] Carolina Dania and Manuel Clavel. 2016. OCL2MSFOL: A Mapping to Many-sorted First-order Logic for Efficiently Checking the Satisfiability of OCL Constraints. In *19th International Conference on Model Driven Engineering Languages and Systems*. ACM, 65–75.

[9] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: an efficient SMT solver. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, Budapest, Hungary, 337–340.

[10] Ana Garis, Alcino Cunha, and Daniel Riesco. 2011. Translating Alloy Specifications to UML Class Diagrams Annotated with OCL. In *9th International Conference on Software Engineering and Formal Methods*. Springer, Montevideo, Uruguay, 221–236.

[11] Martin Gogolla, Fabian Büttner, and Mark Richters. 2007. USE: A UML-based specification environment for validating UML and OCL. *Science of Computer Programming* 69, 1-3 (2007), 27–34.

[12] Martin Gogolla, Frank Hilken, Khanh-Hoang Doan, and Nisha Desai. 2017. Checking UML and OCL Model Behavior with Filmstripping and Classifying Terms. In *11th International Conference on Tests & Proofs*. 119–128.

[13] Carlos Alberto González Pérez, Fabian Buettner, Robert Clarisó, and Jordi Cabot. 2012. EMFtoCSP: A Tool for the Lightweight Verification of EMF Models. In *Formal Methods in Software Engineering: Rigorous and Agile Approaches*. Zurich, Suisse.

[14] Frank Hilken and Martin Gogolla. 2016. Verifying Linear Temporal Logic Properties in UML/OCL Class Diagrams Using Filmstripping. In *2016 Euromicro Conference on Digital System Design*. 708–713.

[15] Daniel Jackson. 2002. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering Methodologies* 11, 2 (2002), 256–290.

[16] Mirco Kuhlmann and Martin Gogolla. 2012. From UML and OCL to Relational Logic and Back. In *15th International Conference on Model Driven Engineering Languages and Systems*. Springer, 415–431.

[17] Marcel Kyas, Harald Fecher, Frank S. de Boer, Joost Jacob, Jozef Hooman, Mark van der Zwaag, Tamarah Arons, and Hillel Kugler. 2005. Formalizing UML Models and OCL Constraints in PVS. *Electronic Notes in Theoretical Computer Science* 115 (2005), 39–47.

[18] N. Przigoda, C. Hilken, R. Wille, J. Peleska, and R. Drechsler. 2015. Checking concurrent behavior in UML/OCL models. In *18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 176–185.

[19] M. Soeken, R. Wille, and R. Drechsler. 2011. Verifying dynamic aspects of UML models. In *Design, Automation Test in Europe*. 1–6.

[20] Emina Torlak and Daniel Jackson. 2007. Kodkod: a relational model finder. In *13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, Braga, Portugal, 632–647.

[21] Hao Wu. 2016. Generating Metamodel Instances Satisfying Coverage Criteria via SMT Solving. In *The 4th International Conference on Model-Driven Engineering and Software Development*. 40–51.

[22] Hao Wu. 2017. Finding Achievable Features and Constraint Conflicts for Inconsistent Metamodels. In *13th European Conference on Modelling Foundations and Applications*. Springer, 179–196.

[23] Hao Wu. 2017. MaxUSE: A Tool for Finding Achievable Constraints and Conflicts for Inconsistent UML Class Diagrams. In *Integrated Formal Methods*. Springer, 348–356.