

# OCLASM: Towards an ASM-based formalism for OCL

Gergely Mezei, Tihamér Levendovszky, Hassan Charaf

Budapest University of Technology and Economics  
Goldmann György tér 3., 1111 Budapest, Hungary

## 1 Introduction

Domain-specific, visual languages can accelerate the development and maintenance of software systems. The customization abilities of domain-specific languages made them very popular in almost all fields of software engineering. Besides the advantages, these languages have weaknesses as well: they have the tendency to be imprecise, incomplete, and sometimes even inconsistent. The solution to the problem is to extend the visual definitions by textual constraints. There are several textual constraint languages, the Object Constraint Language (OCL) is possibly the most popular among them. OCL was originally developed to create precise UML diagrams [1] only, but the flexibility of the language makes it possible to reuse OCL in more generic language engineering as well.

We have implemented a metamodeling and model transformation tool, the Visual Modeling and Transformation System (VMTS, [2]). In this tool we use constraint both in modeling and in creating validated model transformations [3]. In our approach, we try to create an efficient constraint handling solution. We have implemented an OCL Compiler [4] and created three optimization algorithms [5] that can reduce the number of model queries. However, we found that our algorithms are not precisely defined, we had to correct them several times. That is why we have decided to formalize the algorithms and prove the correctness of the algorithms formally, if it is possible.

OCL is a formal language, it has a formalism [6] based on set theory. This formalism does not define all OCL constructs as elaborated in [7]. Moreover, the original formalism does not describe the dynamic behavior of the constraints, namely, how the system states evolve. Most of these issues are solved in the approach presented in [8]. The approach is based on Abstract State Machines (ASMs) and defines the dynamic behavior, which is unavoidable in proving the correctness of our algorithms. However, the constraint expressions do not appear in the abstract state machine directly, instead, it uses the original formalism (the set theoretical approach) and defines traces to be notified on changes. This representation is not practical to describe our algorithm, because it changes the constraint expression itself, thus, we need a higher level of abstraction in formalization. Therefore, we have decided to create a new formalism for OCL.

Abstract State Machine (ASM) is a flexible formalization technique that provides a concise way to define system semantics and dynamic behavior. ASM

supports modularization and its notation is similar to programming languages, which is useful in formalizing algorithms given by pseudo code. We have found that ASM offers a practical solution to formalize our algorithms and OCL itself. We have used ASM to formalize the OCL dialect that we are using and the OCL optimization methods that we have created. This paper introduces OCLASM, our OCL formalism, in a nutshell. The paper focuses on the basic ideas, while more details about our formalism can be found in [9] and [2].

## 2 OCLASM: An ASM-Based OCL Formalism

We have created OCLASM to describe a dialect of OCL and the dynamic behavior of the constraint during the evaluation. OCLASM acts as an interpreter for the OCL constraints. The states of OCLASM represent the state of execution at a certain point of time. A state can be considered as an internal state of the evaluating environment, which is evolved by the rule set of OCLASM. States describe, for example, which expression is under evaluation or which local variables are available. The rules of OCLASM are used to navigate between the states when running the validation. OCL cannot change the underlying model by the definition [1], thus, model queries are applied as monitored functions. The expressions of the constraints are obtained by shared functions, because the constraint can be changed e.g. by our optimization algorithm. The execution of the constraint is a step-by-step execution of constraint expressions by using the rules of OCLASM.

We use OCLASM not only on UML-compliant models, but in an n-layer metamodeling environment. This means that the underlying model extends the original model structure definition used in OCL: it consists of model nodes, attributes and relationships (not restricted to UML types). Attributes are either of primitive types or complex attributes containing several sub-attributes. Complex attributes can be nested hierarchically in a tree representing the attributes. Navigation on this tree is applied by a new expression type, the *AttributeNavigation*. The OCL notation of *AttributeNavigation* is the same as the notation of navigation between model items in the standard OCL: steps of the navigation are separated by dots, if the destination is not uniquely determined by the name used in the expression, the result is a set.

The extension of the underlying model structure means that OCLASM can be used not only to describe the dynamics of constraint evaluation, but to formalize constraints for domain-specific models as well.

### 2.1 Functions

Both model nodes and attributes are identified by a unique ID (a literal expression), the universe of *IDs* is shared between the two different constructs. OCLASM defined monitored functions to retrieve attributes, model items and the navigation of model items from the underlying model definition. Since we use

metamodel-based model definitions instead of built-in, hard-coded model definitions, it can be useful to support *typeof* function which retrieves the metamodel item of an item.

Besides the model management, it is necessary to provide functions in OCL which can query and change the syntax tree representing the constraint. We refer to basic syntactic constructs (programming statements and expressions) available in OCL as *Phrases*. OCLASM defines pointers to nodes of the syntax tree and offers child-parent functions to navigate from and to sub-trees. Recall that these functions are shared functions, because they are usually defined by the outer systems, but sometimes it is required to change them from inside as well.

OCLASM also contains numerous dynamic functions. These functions help to store the current state of the evaluation environment when the rules are applied. There are functions to obtain the current expression under evaluation, or the type, name of a certain expressions. Local variables of the constraint are handled similarly.

## 2.2 Transition rules

Transition rules describe how the states of OCLASM change over time by evaluating expressions and executing statements of the input program. The main idea is to create a rule for each type of language constructs available in OCL. For example OCLASM has rules for iterate, navigation, or variable declaration actions. These rules describe the semantics of the expression and manage dynamic functions. OCLASM has a central rule called *eval*. This rule has a switch-case block with many branches, more precisely for each type of language constructs, *eval* has a branch. It checks the type of the parameter *Phrase* and executes the appropriate branch by calling the rule associated with the *PhraseType*. Therefore, *eval* acts as a mapping function between *Phrases* and rules of OCLASM. Moreover, *eval* helps calling the rules with the appropriate parameters (it adds sub-expressions as parameters, using the *Child* function). Updating the value of *CurrentPos* is also handled by *eval*.

The initial position of OCLASM sets the *CurrentPos* to the start position of the outermost constraint expression and sets the value of all other dynamic functions to *undef*. A run of OCLASM is started by calling *eval* for the start position. When the run of the state machine of OCLASM is finished, the outermost expression holds a single value. If the evaluated constraint was an invariant, then this value shows whether the model was valid.

## 2.3 Using the formalism

Using OCLASM, we have formalized our constraint relocation algorithm. The algorithm can relocate the constraint into a new context, where the evaluation of the constrain requires less model queries. The algorithm has been formalized as a rule in OCLASM, which is applied before the evaluation of the constraint begins. The rule decorates and modifies the *Phrases* of the constraint just as

a compiler would do. The result is correct if the modified constraint finds the same models to be valid as the original constraint. Using the formalization, it is possible to localize the changes between the two constraint expressions and to express their equivalency by showing that although the rules are different, they always produce the same result. The formalized algorithm along with the proof can be found in [2].

### 3 Conclusions and future work

This paper has given a very short introduction on our formalism on OCL. The formalism is based on ASM, which has provided a very useful theoretical basis to describe our approach. The paper introduced the basic concepts, the structure of the abstract state machines of the formalism and it has discussed how the states evolve. Functions and the role of the rules have also been presented.

It seems that the advantages, mainly the flexibility of and the compactness of the ASM technique makes it practical to reuse ASM in other fields of our research. At the moment, we are working on describing our complete attribute and maybe topological structure by ASM as well. This could produce a complete n-layer metamodeling approach that is based fully on formal constructs.

### References

1. J. Warmer, A. Kleppe, Object Constraint Language, The: Getting Your Models Ready for MDA, Second Edition, Addison Wesley, 2003
2. VMTS Web Site, <http://vmts.aut.bme.hu>
3. L. Lengyel, T. Levendovszky, H. Charaf, Validated Model Transformation-Driven Software Development, International Journal of Computer Applications in Technology (IJCAT), Special Issue on: Concern Oriented Software Evolution
4. G. Mezei, T. Levendovszky, H. Charaf, A New Formalism Technique for OCL, 5th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence and Informatics, 2007
5. G. Mezei, T. Levendovszky, H. Charaf, An Optimizing OCL Compiler for Metamodeling and Model Transformation Environments, Working Conference of Software Engineering, 2006
6. Object Management Group Object Constraint Language Specification, <http://www.omg.org/docs/ptc/03-10-14.pdf>
7. S. Flake: Towards the Completion of the Formal Semantics of OCL 2.0. ACSC pp 73-82, 2004
8. S. Flake, W. Müller: An ASM Definition of the Dynamic OCL 2.0 Semantics. UML 226-240, 2004
9. T. Levendovszky, G. Mezei, H. Charaf, Formalizing the Evaluation of OCL Constraints, Acta Polytechnica Hungarica, Volume 4, Issue 1, 2007, ISSN 1785-8860, pp. 89-110.