Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 304 (2014) 81–96

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Towards a $\mathbb{K}$ Semantics for OCL

Andrei Arusoaie Dorel Lucanu

Department of Computer Science
Alexandru Ioan Cuza University
Iaşi, Romania

Vlad Rusu

Inria Lille Nord Europe
Villeneuve d'Ascq, France

Abstract

We give a formal definition to a significant subset of the Object Constraint Language (OCL) in the $\mathbb{K}$ framework. The chosen subset includes the usual arithmetical, Boolean (including quantifiers), and string expressions; collection expressions (including iterators and navigation); and pre/post conditions for methods. Being executable, our definition provides us, for free, with an interpreter for the chosen subset of OCL. It can be used for free in $\mathbb{K}$ definitions of languages having OCL as a component We illustrate some of the advantages of $\mathbb{K}$ by comparing our semantical definition of OCL with the official semantics from the language's standard. We also report on a tool implementing our definition that users can try online.

Keywords: Object constraint language, Formal executable semantics, $\mathbb{K}$ semantic framework

1 Introduction

The Object Constraint Language (OCL) is a textual language for writing constraints over UML models. It has been designed at IBM in the mid nineties, and has been incorporated in the UML standard starting from UML version 1.1.

OCL is intended to be a formal specification language. It is used for adding precision that UML models lack. Its OMG standard [15] defines the syntax of the language, and, to some extent, its semantics. However, despite many academic works on OCL (some of which are referenced in the Related Works section) there is curently no commmercial tool that offers full support for it.

[1] Email: andrei.arusoaie@gmail.com
[2] Email: dlucanu@info.uaic.ro
[3] Email:Vlad.Rusu@inria.fr

http://dx.doi.org/10.1016/j.entcs.2014.05.004
1571-0661/© 2014 Elsevier B.V. All rights reserved.

Part of the problem is the standard, which is imprecise, incomplete, and flawed in some places. This is common to many languages for which the semantics is informally described in manuals declared as standards (see, e.g., the definition of C [9]). Some of these problems include the nondeterministic cast operations from unordered to ordered sets and bags, the question of whether a singleton set $\{a\}$ and the set element a should be distinct or not (answered differently in different pages of the standard [15]), and issues due to the presence of an *invalid* value.

In this paper we report on work in progress towards a formal semantics of OCL in the $\mathbb{K}$ framework [22]. $\mathbb{K}$ is a semantical framework mainly intended for defining formal operational semantics for programming languages. $\mathbb{K}$ semantics is executable, meaning that programs in the defined languages can be executed, tested, and in the near future, formally verified [4].

We have defined both *static* parts of OCL (corresponding to well-formedness constraints and queries on UML models) and *dynamic* parts (corresponding to pre/post conditions of methods). Moreover, we allow for evaluating OCL expressions on *symbolic* models, i.e., models which attributes may have symbolic values [5]. This allows us, for example, to compute the condition (on the variables) under which a concrete instance of a symbolic model, satisfiying given OCL constraints, exists; and to compute the conditions under which a given sequence of method calls is feasible. These conditions can then be passed to constraint solvers to check whether they are satisfiable. A negative answer is a useful feedback to users: they need to revise their models and OCL expressions in order to make them consistent with each other.

Regarding the static part, the defined OCL fragment consists of the usual arithmetical, Boolean (including quantifiers), string operations, and collection operations (including navigation via UML associations and iterators), along with let-in and if-then-else constructs. Expressions may have a scalar type (integer, Boolean, string, or UML classes) or a collection type (bags or sets). This is a significant fragment of OCL, allowing users to write most of the useful well-formedness constraints on UML models, and which we may extend in the future when new OCL standards decide on some open issues and fix erroneous ones (such as "ordered sets" with an unkown order). Regarding the dynamic part, all pre/post condition constructions using the defined static part are also defined, as well as the specific constructions for postconditions (for referring to values in the pre-condition and to returned values).

The main advantages of $\mathbb{K}$ formal semantics definitions are their expressiveness, executability, and modularity. Each OCL construction is typically defined using one or two $\mathbb{K}$ rules. We have benefited a lot from $\mathbb{K}$ features such as the syntactical substitution, the automatic generation of $\mathbb{K}$ rules for evaluating argument of *strict* operations, and the automatic context transformers, which require users to only provide $\mathbb{K}$ rules with minimal information for matching and rewriting.

We illustrate these advantages by comparing our $\mathbb{K}$ semantics of the let-in OCL operation with its official semantics from the *Annex A : formal semantics (informative)* of the OCL standard [15]. We also show how the symbolic OCL evaluation is obtained

[4] The language-independent *matching logic* has been defined [20] and is being implemented.

[5] To our best knowledge ours is the only approach dealing with symbolic evaluation of OCL.

Download English Version:

https://daneshyari.com/en/article/423686

Download Persian Version:

https://daneshyari.com/article/423686

Daneshyari.com