Softw Syst Model (2016) 15:757–781
DOI 10.1007/s10270-014-0438-9

REGULAR PAPER

Empirically evaluating OCL and Java for specifying constraints on UML models

Tao Yue · Shaukat Ali

Received: 12 November 2013 / Revised: 11 September 2014 / Accepted: 13 October 2014 / Published online: 21 November 2014

Abstract The Object Constraint Language (OCL) has been applied, along with UML models, for various purposes such as supporting model-based testing, code generation, and automated consistency checking of UML models. However, a lot of challenges have been raised in the literature regarding its applicability in industry such as extensive training, slow learning curve, and significant effort to use OCL due to lack of familiarity of practitioners. To confirm these challenges, empirical evidence is needed, which is severely lacking in the literature. To build such preliminary evidence, we report a controlled experiment that was designed to evaluate OCL by comparing it with Java; a programming language that has also been used to specify constraints on UML models. Results show that the participants using OCL perform as good as the participants working with Java in terms of three objective quality metrics (i.e., completeness, conformance and redundancy) and two subjective metrics (i.e., applicability and confidence level). In addition, the participants using OCL performed consistently well for all the constraints of varying complexity, while fluctuating results were obtained for the participants using Java for the same constraints. Based on the empirical evidence, we can conclude that it does not make much difference to use OCL or Java for specifying constraints on UML models. However, the participants working with OCL performed consistently well on specifying constraints of varying complexity suggesting that OCL can be used to model complicated constraints (commonly observed in industrial applications) with the same quality as for simpler constraints. Moreover, additional analyses on the constraints when using Java and OCL tools revealed that tools are needed to specify fully correct constraints that can be used to support automation.

Keywords OCL · Java · Controlled experiment · Empirical study · Constraints

Communicated by Prof. Antonio Vallecillo.

T. Yue (✉) · S. Ali
Certus Software V&V Center, Simula Research Laboratory,
P.O. Box 134, 1325 Lysaker, Norway
e-mail: tao@simula.no

S. Ali
e-mail: shaukat@simula.no

1 Introduction

The Object Constraint Language (OCL) [1] was defined to append extra semantics to UML models in addition to the ones already enforced by the UML metamodel itself. OCL can be used to support various modeling activities for different purposes, such as providing precise meaning to state invariants on states and guards on transitions of UML state machines for automated test data generation [2–7]. Moreover, OCL can be used to specify constraints at various MOF-levels (M3, M2, and M1) for different purposes such as querying a subset of model elements from a model [1,8], evaluating/validating [8–13] a specified constraint (e.g., for automated test oracles), and solving [4,14–29] (e.g., for automated test data generation).

While working with several industrial partners on diverse model-based engineering projects using UML and its extension and OCL, we found that the use of OCL with models can support various model-based engineering activities including: automated model-based testing, consistency checking, and configuration in the context of produce line engineering [5,6,20,30]. Moreover, OCL is also widely used as the language for writing constraints in many commercial modeling tools such as IBM Rational Software Architect (RSA) and Magic Draw [31]. However, successfully applying OCL in

practice comes with a cost, i.e., additional effort (in terms of training and required tool support) is required to specify constraints using OCL and being declarative language in nature, its formalism hinders its applicability in practice. To convince our industrial partners (who are not familiar with OCL) to invest in terms of training and tool support, especially when there are alternatives such as Java with which practitioners are more familiar, it is very important to provide evidence via rigorous empirical study to tell which constraint specification language is better and why.

Java is a commonly used programming language, and it may be used to specify constraints on UML models (e.g., [32,33]), and people in industry are usually familiar with Java. However, as it was not defined for the purpose of specifying constraints on UML models, it is not so straightforward, as compared to OCL, in terms of, e.g., traversing model elements in UML models. There exist tools for both OCL and Java [9,12,30,34,35] for evaluating, querying, solving, and parsing formally specified constraints. For example, commercial tool IBM RSA [33] and open source tool Papyrus [32] both allow users to specify/validate their constraints specified both in Java and OCL against UML models.

Since there is no evidence in the literature suggesting that OCL is better than Java or vice versa in terms of specifying constraints on UML models, we conducted a controlled experiment to investigate this. Our main goal is to collect a body of evidence based on which we can recommend Java or OCL to our industrial partners for writing constraints on UML models in their respective applications. Moreover, we aim to build/gather preliminary evidence about OCL/Java for specifying constraints that is missing in the current literature (Sect. 5) that can be used by researchers and other practitioners to select a language for specifying constraints to solve their respective problems.

The controlled experiment was conducted with 29 fully trained graduate students taking a graduate student course in 'Empirical Software Engineering' at Beihang University, Beijing, China. The course was given by the authors of the paper. Two case studies were used in the experiment. Quality measures (e.g., *Completeness*, *Conformance* and *Redundancy*) were defined to evaluate constraints specified by the experiment participants. Results show that the participants working with OCL and Java performed equally well. Additionally, we observed that the participants using OCL performed consistently well for all the constraints of varying complexity, which is not the case for the participants using Java for the same constraints. Thus, we suggest using OCL for specifying constraints in industrial applications of model-based engineering since constraints in industrial applications are complex, as we observed in our industrial applications.

To further investigate the constraint specifications in Java and OCL using tools, we performed additional analyses. We selected 100 constraints with 100 % conformance and inputted/entered them to the tools to identify errors. Results show that more errors were identified in Java specifications as compared to OCL, and further lead to the conclusion that tools are needed to specify fully correct constraint specifications that can be used for supporting automation.

The rest of the paper is organized as follows. Section 2 provides details on the experiment planning. Section 3 reports and discusses experimental results. Section 4 points to possible threats to validity in our experiment. Section 5 reports the related work and we conclude the paper in Sect. 6.

2 Experiment planning

This section discusses the planning of the experiment according to the definition and reporting template defined by Wohlin et al. [36]. Section 2.1 provides experiment definition and hypotheses formulation. Section 2.2 provides details on the participants and training for the experiment. Section 2.3 provides details on the material that we used for the experiment. Section 2.4 provides metrics that we used to assess the quality of specified constraints. Last, Sect. 2.5 discusses the design of the experiment and its execution.

2.1 Experiment definition and hypotheses formulation

The objective of our experiment is to compare OCL and Java with respect to their applicability of specifying constraints on UML class diagrams. Applicability is assessed according to two criteria: the quality of specified constraint specifications and participants' subjective opinions on the applicability of these two languages. We measure the quality of OCL and Java constraint specifications from three complementary points of view: *Completeness*, *Conformance*, and *Redundancy*. The subjective opinions (*Applicability* and *ConfidenceLevel*) were collected through two four-point Likert scale questions of the questionnaires: (1) To which extent the constraint is easy to specify and (2) To which extent do you feel confident to apply a language (Java or OCL) to specify constraints. The independent variable that we concern is *Method* (OCL vs. Java). There is one factor that is also interesting to take into account when statistical analyses are conducted: *Constraint complexity*. The detailed discussion of the five dependent variables and related measurement is provided in Sect. 2.4.

Based on the above variables, we can formulate the following null hypothesis (H_0) to be tested for each dependent variable: there is no significant difference between OCL and Java in terms of the five dependent variables. None of the expected differences between OCL and Java can a priori be certain to be in a specific direction. This therefore leads to the definition of two-tailed hypotheses (H_1) and it is stated as: OCL results in different quality of constraints or differ-

Table 1 Hypotheses

Dependent Variable	Null Hypothesis ($\mathbf{H}_0$)	Alternative Hypothesis ($\mathbf{H}_1$)
Completeness	Complt(OCL) = Complt(Java)	Complt(OCL) ≠ Complt(Java)
Conformance	Confor(OCL) = Confor (Java)	Confor(OCL) ≠ Confor(Java)
Redundancy	Redun(OCL) = Redun(Java)	Redun(OCL) ≠ Redun(Java)
Applicability	Appli(OCL) = Appli(Java)	Appli(OCL) ≠ Appli(Java)
ConfidenceLevel	Confi(OCL) = Confi(Java)	Confi(OCL) ≠ Confi(Java)

ent responses to the two questions in the questionnaire when compared to Java. Hypotheses are provided in Table 1.

2.2 Participants and training

The controlled experiment was conducted at Beihang University, Beijing, China. The participants in the experiment were 29 graduate students taking a short-term but intensive graduate course in 'Empirical Software Engineering' at the Department of Computer Science and Engineering. The course was given by the authors of the paper. The students in this degree already hold a Bachelor in Computer Science and have already been exposed to the UML and OCL notations and have used Java for multiple course projects.

The participants were trained by the authors of this paper. Two three-hour sessions, as part of the course curriculum, were given on the following topics: (1) Recap of UML class diagrams since the participants were already familiar with this topic preceding the training, (2) Introduction to OCL, and (3) Recap of Java. Each topic was accompanied with several examples and interactive class assignments. The participants were given a questionnaire with eight questions before the experiment sessions to collect information about their knowledge and experience on UML class diagrams, OCL, and Java. The questionnaire is provided in "Appendix D" for reference. The collected questionnaire responses were computed (by giving equal weight to each questionnaire) to obtain a single value for each participant, indicating his/her background on UML, OCL, and Java in general. These values were then used as the basis to group the participants into two blocks and therefore ensure better homogeneity across the two groups involved in the experiment. The experiment was part of a series of compulsory laboratory exercises that were part of the course curriculum.

2.3 Materials

We used two systems in the experiment: Banking System and Video Conferencing System (VCS) [5]. Banking System is an extended version of the Banking System from the OCL 2.2 specification [1]. The rationale of choosing this system as one of the case studies is because the context is relatively easy to understand. We selected VCS as the second case study as it is a real industry case study of Video Conferencing System (VCS) developed by Cisco Systems, Norway. This case study is part of a project aiming at supporting automated, model-based testing of a core subsystem of a VCS called Saturn [5].

For Banking System, a bank has several employees and customers. Each customer can have at most two accounts in a bank: One is saving account and the other is current account. A customer must be employed in a company or owns a company in order to have a bank account. An employee of a bank can also be its customer having at the most two accounts in the bank. For the VCS case study, the core functionality to be modeled manages the sending and receiving of multimedia streams. Audio and video signals are sent through separate channels, and there is also a possibility of transmitting presentations in parallel with audio and video. One conference participant can send presentation to all others, in parallel to the ongoing video call. The core functionality was used as part of the experiment.

In the answer sheet provided to the participants for each system and each method, we provided (1) a brief description of the system, (2) the class diagram on which constraints should be specified, (3) the description of each attribute of the classes in the class diagrams, and (4) a list of constraints that the participants should specify using either OCL or Java. Hence, we designed four answer sheets for the four combinations of the two methods and the two case study systems. The content of the answer sheets designed for Banking System and VCS is provided in "Appendices A and B", respectively.

2.3.1 Complexity metrics

To enable the participants to tackle increasingly more complex constraints to smooth the learning curve, we ordered the constraints according to their complexity, which is measured by applying these four metrics sequentially:

1. Maximum number of traversals in all the clauses of a constraint ($n_{traversals}$)
2. Number of required attribute types (n_{types})
3. Order of the complexity of the attribute types ($o_{typeComplexity}$), and
4. Number of clauses ($n_{clausesRequired}$).

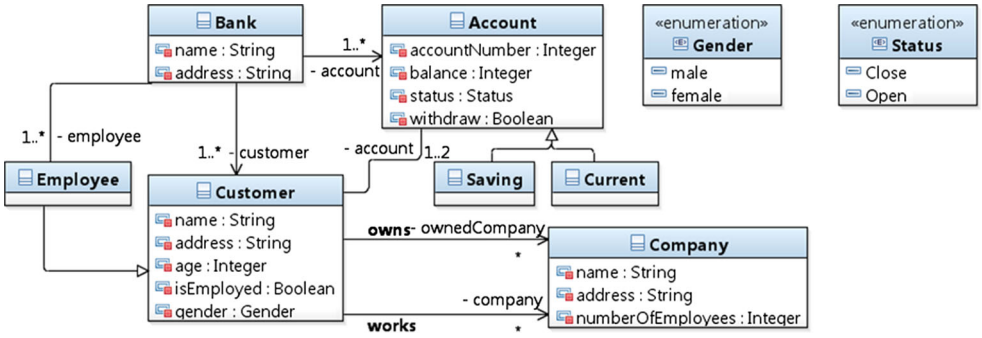


Fig. 1 Class diagram of Banking System

Table 2 Values of complexity metrics and specifications in OCL and Java of selected constraints

ID	Specification in English	$n_{traversals}$	n_{types}	$o_{typeComplexity}$ (Low→High)	$n_{clausesRequired}$
A	All customers of the bank must be employed.	1	Boolean	N/A	1
	`self.customer->forAll(c:Customer|c.isEmployed)`				

	`for (int i=0;i<b.customer.length;i++)`
	`{`
	`  if (b.customer[i].isEmployed==false)`
	`  {`
	`  return false;`
	`  }`
	`}`
	`return true;`				

B	All accounts in the bank must have unique account numbers and each account must be linked to exactly one customer.	2	Integer (2)	N/A	2
	`self.account->isUnique(account.accountNumber) and self.account->forAll(a:Account|a.customer->size()=1)`				

	`for (int i=0;i<b.account.length-1;i++)`
	`{`
	`  for (int j=i+1;j<account.length;j++)`
	`  {`
	`  if (account[i].accountNumber==account[j].accountNumber)`
	`  return false;`
	`  }`
	`}`
	`for (int i=0;i<account.length;i++)`
	`{`
	`  if (account[i].customer.length>1)`
	`  return false;`
	`}`
	`return true;`				

C	A customer of the bank can only have a saving account when he/she has a current account but not vice versa.	3	Boolean (2), Integer (2)	Boolean, Integer	4
	`self.customer->forAll(c|c.account->size()=1 implies c.account->first()->oclIsTypeOf(Current) xor c.account->size()=2 implies c.account->select(c->oclIsTypeOf(Saving)=1))`				

	`for (int i=0;i<customer.length;i++)`
	`{`
	`  if (customer[i].account.length ==1 && customer[i].account[0] instanceof Saving)`
	`  return false;`
	`  if (customer[i].account.length ==2 && account[0] instanceof Saving && account[1] instanceof Saving || account[0] instanceof Current && account[1] instanceof Current)`
	`  {`
	`  return false;`
	`  }`
	`}`
	`return true;`				

Using Banking System as the example to explain the above metrics, we provide the class diagram of the system in Fig. 1 and three constraints specified in English, along with values to the complexity metrics are provided in Table 2.

$n_{traversals}$ is defined as a step from the context class to the farthest class on whose primitive attributes a constraint is specified. For example, Constraint A presented in Table 2 has one traversal from `Bank` (the context class) to `Customer`.

Table 3 Questionnaire design

Measure	Statements	Scale
ClassDiagramUnderstandability	I understood the provided class diagram clearly	4-point Likert scale: Completely agree, Generally agree, Generally disagree, Completely disagree
ConstraintUnderstandability	I understood the constraint properly	
Applicability	The constraint was easy to specify with the requested language	
ConfidenceLevel	I am confident to apply the requested language	

Constraint B contains two traversals from Bank (the context class), via `Account`, to `Customer`. Constraint C is the most complex one among the three as it has three traversals from the context class `Bank`, via `Customer` and `Account`, to `Saving` or `Current`.

There are four types of primitive attributes appearing in the given constraints: *Boolean*, *Enumeration*, *Integer*, and *String*, which are ordered (from the simplest to the most complex) according to their complexity in terms of specifying constraints. For example, as shown in Table 2, Constraint A involves one attribute `isEmployed: Boolean` owned by class `Customer`. Constraint B is associated with one integer attribute: `accountNumber: Integer` (of class Account) and an integer operation `size()/length` (in Java) (returning an integer) to check that each account is linked to exactly one account. For Constraint C, we need one integer operation `size()/length` in Java to check that a customer can only have a saving account when he/she has a current account, i.e., `account->size()=2`. Moreover, we need one Boolean operation `oclIsTypeOf()/instanceof` in Java to check the type of account. For checking the reverse, again we need one Boolean and one Integer.

Number of clauses ($n_{clausesRequired}$) is defined as the total number of clauses required in a constraint specification. We first ordered the constraints according to the maximum number of traversals ($n_{traversals}$), then the number of required variable types when the same maximum number of traversals appears in two or more constraints. If n_{types} is equal for two or more constraints, we further check $O_{typeComplexity}$, then $n_{clausesRequired}$.

2.3.2 Post-questionnaire

A post-questionnaire was distributed after the students finished the tasks of specifying constraints in each round. The objective of the questionnaire is to obtain subjective opinions of the participants on the applicability of Java and OCL and their confidence level (*Applicability* and *ConfidenceLevel*) of applying the requested method. As shown in Table 3, the questionnaire has four statements on a 4-point Likert scale question. The first statement is defined for each system. The other three statements are asked for each constraint of each system. The third statement requires the participants to rate each constraint according to the extent to which they perceive it to be easy to apply (*Applicability*). The last statement was used to obtain the participants' subjective opinions on their confidence after each constraint was applied (*ConfidenceLevel*). Notice that the same questionnaire is used for collecting information for both methods. The questionnaire is presented as a table, with instructions, to the students as shown in "Appendix C".

2.4 Dependent variables and measurement

As previously discussed, in total, we defined five dependent variables. Their measurement is described below. To illustrate each dependent variable, we use Banking System as the running example.

2.4.1 Quality of constraints

We measure the quality of a constraint from three aspects: *Completeness, Conformance*, and *Redundancy*. These quality metrics are used for measuring both OCL and Java constraints. In this section, we discuss how these three aspects are measured using a set of metrics. Several OCL and Java constraints specified by the students during the experiment are provided in "Appendix E" for reference.

Completeness ($Completeness_{Constraint}$): This metric measures the percentage of the specified clauses of a constraint specification, with the formula below:

$$1 - \frac{Number\ of\ missing\ clauses}{n_{Clauses}},$$

where $n_{Clauses}$ is the total number of clauses expected from a constraint specification. For example, if a specification of Constraint B (Table 2) only describes clause "all accounts in the bank must have unique account numbers," then the completeness of this specification is $1-(1/2)=50\,\%$ since the constraint contains the other clause: "each account must be linked to exactly one customer."

Conformance ($Conformance_{Constraint}$): This metric measures the conformance of a constraint specification, using the formula below:

$$\frac{\text{Completeness}_{\text{Traversal}}+\text{Conformance}_{\text{Iteration}}+\text{Conformance}_{\text{Condition}}}{x}*\text{Completeness}_{\text{Constraint}}. \quad (1)$$

$Completeness_{Traversal}$, $Conformance_{Iteration}$ and $Conformance_{Condition}$ are the three aspects for measuring the conformance of a constraint, with equal weights. Traversals in a clause of the constraint are the steps required for traveling from the context model element (e.g., a class) to the destination model element, as we discussed in Sect. 2.3.1. We therefore define the metric below to calculate the overall completeness of the traversals of a constraint:

$$Completeness_{Traversal}=\frac{\sum_{i=1}^{n_{Clauses}}\left(1-\frac{Number\ of\ missing\ traversals\ in\ clause\ i}{Total\ number\ of\ traversals\ required\ for\ clause\ i}\right)}{n_{Clauses}}.$$

For example, if a specification of Constraint B (Table 2) only describes clause "each account must be linked to exactly one customer," and the specification only describes the traversal from `Bank` to `Account`, then the completeness of traversals for this particular specification is calculated as: $\frac{\left(1-\frac{1}{2}\right)+0}{2}=25\,\%$.

$Conformance_{Iteration}$ indicates whether manipulating (or iterating over) a collection of objects, which is frequently needed for constraints specified on class diagrams due to the multiplicities on associations, is correct. The following metric is used to compute its values:

$$Conformance_{Iteration}=\frac{\sum_{i=1}^{n_{Clauses}}(Conformance_{Iteration}i)}{n_{Clauses}},$$

in which, $Conformance_{Iteration}i$ is the conformance of iteration for each clause, further defined as: if the iteration is totally wrong, a value 0 is assigned; if it is partial correct, 0.5 is assigned; if it is fully correct, 1 is assigned.

$Conformance_{Condition}$ takes into account the conformance of the condition required for each clause in a constraint specification. As for conformance of iteration, if it is fully correct, partially correct, or fully wrong, in terms of corresponding to the provided input, 1, 0.5, and 0 are assigned, respectively. Examples to calculate conformance based on students' solutions are provided in "Appendix E".

In Formula (1), x is determined by whether iterations are required to specify a constraint. In our experiments, five constraints of VCS do not have iterations. Therefore, for these cases, x equals 2; otherwise 3. The average conformance of the three aspects times the completeness gives us an overall conformance of the constraints, as shown in Formula (1).

Redundancy ($Redundancy_{Constraint}$), for specifying a constraint, extra clauses are considered as redundant clauses:

$$\frac{Number\ of\ extra\ clauses}{Number\ of\ extra\ clauses+n_{Clauses}-Number\ of\ missing\ clauses}.$$

2.4.2 Applicability and confidence level

Applicability and *ConfidenceLevel* are two subjective measures used to assess the two languages (i.e., OCL and Java), and they are based on the responses to two 4-point Likert Scale questions of the post-questionnaire, from 1 (Completely disagree) to 4 (Completely agree).

2.4.3 Complexity of constraints

We measure the complexity of constraints as an ordinal variable with three levels: *Low, Medium*, and *High*. As we discussed in Sect. 2.3, a set of criteria (i.e., $n_{traversals}$, n_{types}, $O_{typeComplexity}$ and $n_{clausesRequired}$) were used to order 10 constrains for each case study system. To define the complexity of constraints across two case study systems, we first ordered the 20 constraints (10 for Banking and 10 for VCS) using the same set of criteria. Then, this order is divided into three groups, which correspond to three levels of our constraint complexity measurement. The division is based on the objective of balancing the number of constraints following into each level. As the result, for VCS, three, four and three constraints follow into three categories (*Low, Medium*, and *High*), respectively. For Banking, four, two and four constraints are classified into *Low, Medium*, and *High* categories, respectively. In total, eight, six and six constraints are at *Low, Medium*, and *High* levels for all the 20 constraints of the two systems. For example, according to this mechanism, the complexity of Constraint A, Constraint B and Constraint C presented in Table 2 are classified as *Low, Medium*, and *High*, respectively.

Table 4 Experiment design

Round	System	Group 1	Group 2	Obtained data points	
				OCL	Java
1	Banking System	OCL	Java	160	130
2	Video Conferencing System (VCS)	Java	OCL	130	160
				580	

2.5 Experiment design and execution

The design of our experiment is summarized in Table 4. We used a within-subjects design[1] since we have two systems and two languages (Java and OCL). During the training sessions (Sect. 2.2), each subject was equally trained to understand the two languages: OCL and Java. Based on the results of a questionnaire (Sect. 2.2), the experiment groups were formed through randomization and blocking to obtain two comparable groups of 16 students each (*Group 1* and *Group 2*) with similar proportions of students from each block. In the first round, *Group 1* was asked to specify constraints of Banking using OCL, whereas *Group 2* was asked to use Java instead.

Such a within-subjects design offers two main advantages. First, with it, we can reduce the error variance due to individual differences in human performance, which is quite common in software engineering tasks. This is due to the fact that the same group of students is exposed to both languages across the different systems. Second, within-subjects designs provide more statistical power as compared to a between-subjects design [36] as it leads to more observations for each treatment. Potential threats from within-subjects designs are "carryover" effects. To address this, for each system, each group was given a different treatment in such a way that ordering effects were counterbalanced: languages, i.e., OCL and Java occurred once in a different order across the two groups. For example, as shown in Table 4, in round 1, *Group 1* was asked to specify constraints for Banking in OCL, whereas *Group 2* to specify constraints for Banking in Java. Note that in the experiment, in the first round, Banking was used, and in the second round, VCS was used. The purpose is to enable the participants to tackle increasingly more complex models and constraints. With a within-subjects design, a matched pair analysis can be applied by comparing the performance of subjects with themselves across treatments (Sect. 3.2).

[1] A within-subjects design offers two main advantages. First, we can reduce the error variance due to individual differences in human performance, which is quite common in software engineering tasks. This is due to the fact that the same group of students is exposed to all OCL specification approaches across the different case studies. Second, within-subjects designs provide more statistical power as it leads to more observations for each treatment Wohlin et al. [36].

As we previously discussed in Sect. 2.2, in our experiment, we have 32 participants enrolled in the experiment after the training sessions, which were divided into two groups, each of which has 16 participants. During the experiment, all the 16 participants of Group 1 participated in the experiment and 13 out of 16 participants from Group 2 participated in the experiment and completed the tasks. Each participant was asked to specify 10 constraints either using OCL or Java for each system. Therefore, we obtained in total 580 data points and their decomposition is provided in Table 4 for reference.

At the beginning of the experiment, an answer sheet containing a brief description of the system, the class diagram on which the constraints will be specified, and the instruction of tasks to perform was distributed to the participants. The participants were given 15 minutes to read the answer sheet and had an opportunity to raise questions on the answer sheet. Then, the authors of the paper explained the system and its class diagram to all the participants. After all these preparation activities, the first constraint was given to the students via a classroom projector screen. At the same time, a 10 minutes timer was triggered. This process repeated 10 times until all the constraints were specified. Notice that before the experiment was conducted, two students were asked to specify the constraints and the average time spent on specifying one constraint was around 10 minutes. After that, the post-questionnaire was distributed. Fixing the time for task execution tends to yield more differences in task effectiveness, but then results cannot be used to study time differences across treatments [36].

The students used pens during the experiment to record the results on the provided answer sheets ("Appendices A and B"), which were collected after each task. We understand that it would be closer to reality to use OCL and Java tools in the experiment. However, we considered that selecting which tools to use would form an internal threat to validity, as there exist various OCL and Java tools in the market and applying which one, even in reality, heavily depend on the application context and there is no unified answer. It is also worth noting that the scope of this experiment is to evaluate how well OCL and Java can be used to specify constraints, not to evaluate particular tools.

The authors of the paper carefully checked the collected answer sheets and evaluated the derived constraints based on the defined quality metrics (Sect. 2.4.1). The data were encoded into a JMP [37] data file to perform the statistical analysis.

3 Results and discussion

In this section, we present results and discussions to test the hypothesis formulated in Sect. 2.1. We first provide descriptive statistics of the dependent variables in Sect. 3.1. In Sect. 3.2, we report the results of the univariate analysis that

we conducted to test the significant difference of OCL and Java in terms of the five dependent variables by *Method* and by *Complexity* of constraint, respectively. To further analyze whether the objective quality measures and the two subjective measures correlate to each other, we conducted correlation analysis (Sect. 3.3).

3.1 Descriptive statistics

The descriptive statistics for all the dependent variables are provided in Tables 5, 6, and 7. The overall observation is that regardless which method was used, the participants performed well in terms of the three quality metrics: high *Completeness* (91 and 89 % for OCL and Java, respectively), reasonable *Conformance* (71 and 73 % for OCL and Java, respectively), and low *Redundancy* (2 and 4 % for OCL and Java, respectively). Another observation is that there is no big difference between OCL and Java by looking at the mean values of the three quality metrics (rows 5, 7 and 9 of Tables 5, 6). To confirm the significance, we performed statistical tests, which will be discussed in Sect. 3.2.

When looked into mean values of each method of the three constraint complexity levels, the participants who were using OCL performed consistently well across all the levels (Table 5). There are 6 and 10 % differences between *Medium* and *High* and between *Medium* and *Low*, for Java, in terms of *Completeness*, and 13 and 12 % differences between *Medium* and *High* and between *Medium* and *Low*, for Java, in terms of *Conformance* (Table 5). This result indicates that the participants performed differently when they were specifying different levels of complexity of constraints using Java. Further statistical analysis conducted to check the significance and results is reported in Sect. 3.2.

Regarding the two subjective, Likert scale measures, the participants subjectively thought constraints with higher complexity were more difficult to specify and they had less confidence. This observation applies to both Java and OCL. For example, for OCL, constraints with *Low*, *Medium* and *High* complexity, received 51, 31, and 25 % *Applicability* as shown in Table 7. Further statistical analysis was conducted to check the significance and results are reported in Sect. 3.2.

3.2 Univariate analysis

3.2.1 Dependent variables by method

Due to the fact that the distributions of all the continuous dependent variables strongly depart from normality as the results of the Shapiro–Wilk *W* test [38] showed, we performed nonparametric, Matched Pair, Wilcoxon rank sum

Table 5 Descriptive statistics for *Completeness*, *Conformance*, and *Redundancy*—OCL

Measures	Complexity Level								
	High			Medium			Low		
	Mean (%)	*N*	Std.	Mean (%)	*N*	Std.	Mean (%)	*N*	Std.
Completeness	90	103	0.25	91	84	0.23	93	103	0.21
				91 % (average), 290 data points					
Conformance	72	103	0.32	71	84	0.3	70	103	0.32
				71 % (average), 290 data points					
Redundancy	1	99	0.07	3	83	0.12	2	100	0.08
				2 % (average), 282 data points					

Table 6 Descriptive statistics for *Completeness*, *Conformance*, and *Redundancy*—Java

Measures	Complexity Level								
	High			Medium			Low		
	Mean (%)	*N*	Std.	Mean (%)	*N*	Std.	Mean (%)	*N*	Std.
Completeness	90	100	0.23	84	90	0.28	94	100	0.19
				89 % (average), 290 data points					
Conformance	78	100	0.31	65	90	0.34	77	100	0.31
				73 % (average), 290 data points					
Redundancy	4	97	0.13	3	87	0.11	5	99	0.15
				4 % (average), 283 data points					

Table 7 Descriptive statistics for *Applicability* and *ConfidenceLevel*

Complexity level	Method	Measure	Completely agree (%)	Generally agree (%)	Generally disagree (%)	Completely disagree (%)	Count
Low	Java	Applicability	51	34	13	2	97
		ConfidenceLevel	36	37	24	3	100
	OCL	Applicability	51	34	13	2	103
		ConfidenceLevel	41	39	19	1	103
Medium	Java	Applicability	51	34	13	2	86
		ConfidenceLevel	36	37	24	3	90
	OCL	Applicability	31	36	29	5	84
		ConfidenceLevel	20	44	30	6	84
High	Java	Applicability	36	19	36	9	97
		ConfidenceLevel	20	29	38	13	100
	OCL	Applicability	25	30	36	9	103
		ConfidenceLevel	23	25	38	14	103

Table 8 Two-tailed matched pair Wilcoxon test with the significance level $\alpha = 0.05$ (dependent variables by method)

Group	Measures	Wilcoxon test		
		Mean difference (OCL–Java) (%)	*DF*	Prob > |*Z*|
1	Completeness	6.4	159	**0.0071**
	Conformance	−2.5	159	0.5255
	Redundancy	−3.9	151	**0.0042**
	Applicability	−1.3	149	0.9022
	ConfidenceLevel	8.1	159	0.3312
2	Completeness	−4.5	129	0.1544
	Conformance	−1.5	129	0.6105
	Redundancy	0.3	122	0.7969
	Applicability	3.1	129	0.6473
	ConfidenceLevel	9.3	129	0.1689
1 + 2	Completeness	1.5	289	0.4059
	Conformance	−2	289	0.4006
	Redundancy	−20	274	**0.0227**
	Applicability	0.7	279	0.8029
	ConfidenceLevel	8.6	289	0.1048

test [38], and results are reported in Table 8. Each row reports on each dependent variable measure for each group of participants or the two groups together (1+2). Columns show the mean differences, degree of freedom (*DF*), and corresponding probability for the Wilcoxon test.

For *Group 1* who used OCL to specify constraints for Banking in the first round and specified constraints for VCS using Java in the second round, as shown in Table 4, the matched pairs were formed based on the data collected for these two tasks. A pair in our context is the same student using OCL to specify a constraint at a level of complexity in the first round and specifying a constraint using Java in the second round of equivalent. The same strategy was followed for matching the results of applying OCL and Java by *Group 2*.

The participants in *Group 1* performed significantly better when they were using OCL than Java in terms of quality metrics *Completeness* and *Redundancy*, as shown in Table 8 (the values highlighted in bold in Column 5). No significant difference was observed between two methods regarding *Conformance* though when Java was used, the participants performed slightly better than when they were using OCL (notice the negative value in Row 4 and Column 3: −0.025). Regarding the participants' subjective opinions on the applicability of the methods and their confidence of applying them, no significant difference was observed between the two methods. When looking at the results of *Group 2*, no significant difference can be observed between the two methods for any of the five dependent variables.

Table 9 Wilcoxon test with the significance level $\alpha = 0.05$ (dependent variables by Complexity)

Method	Measures	Pearson Chi-square test (Prob>ChiSq)	Wilcoxon test (p value)		
			High-Medium	Medium-Low	High-Low
OCL	Completeness	N/A	0.825	0.4736	0.3228
	Conformance		0.7061	0.9523	0.7001
	Redundancy		0.5092	0.3529	0.7609
	Applicability	**0.0001**	N/A		
	ConfidenceLevel	<**0.0001**			
Java	Completeness	N/A	0.1533	(−) **0.0036**	0.1150
	Conformance		**0.0051**	(−) **0.0089**	0.8744
	Redundancy		0.7528	(−) 0.4874	(−) 0.6994
	Applicability	**0.0004**	N/A		
	ConfidenceLevel	**0.0064**			

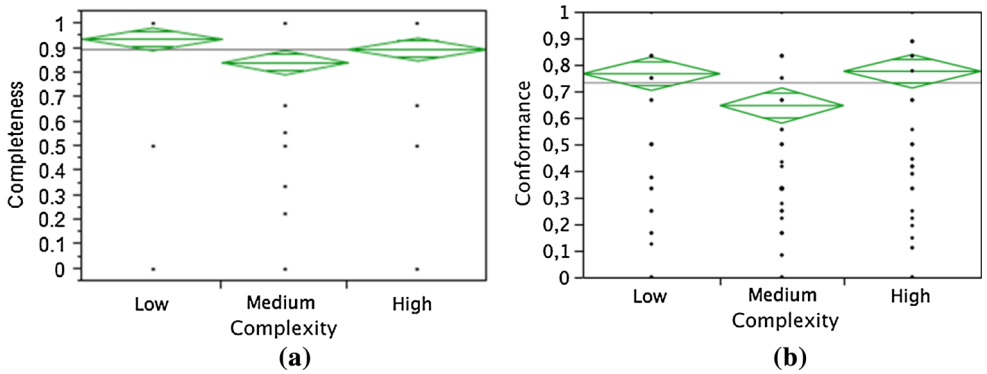


Fig. 2 Mean diamonds graph for Completeness (**a**) and Conformance (**b**) by Complexity of Java

When the results from the two groups are combined, OCL yielded better performance than Java in terms of *Redundancy*, implying that the participants working with Java significantly introduced more redundant clauses as compared to the participants working with OCL.

3.2.2 Dependent variables by Complexity

As discussed in Sect. 2.4, we classified all the 20 constraints of the two systems into three categories: *Low, Medium*, and *High*. For continuous data, to test whether the dependent variables are significantly different given different levels of complexity, we performed the Kruskal–Wallis one-way analysis of variance test [38]. It is a nonparametric equivalent of the one-way ANOVA test, since the distributions of all the continuous dependent variables strongly depart from normality as the results of the Shapiro–Wilk W test showed. To compare each pair of complexity levels, we performed the Wilcoxon Signed-Rank test [38]. Results are provided in Table 9. For ordinal data, the Pearson Chi-square test [38] was performed and results are also reported in Table 9.

As shown in Table 9 (Rows 5, 6, 10 and 11, and Column 3), for *Applicability* and *ConfidenceLevel*, significant differences were observed between any two *Complexity* levels. When we further looked into the details, we observed that for more complex constraints, the participants had significantly less confidence and thought the given method was significantly more difficult to apply. This observation is consistent for both OCL and Java.

Regarding the three quality measures with continuous data, results of the Wilcoxon pair tests show that there is no significant difference for any constraint complexity level pair for any measure of OCL, as shown in Table 9 (Rows 3–5 and Columns 4–6). This implies that OCL consistently performed well (with 91 % *Completeness*, 71 % *Conformance*, and 2 % *Redundancy* on average) for all constraints at the different levels of complexity.

For Java, as shown in Table 9, significant difference was observed for Completeness between pair *Medium-Low* in favor of *Low* (notice that '(−)' attached to the p values in the table indicates the direction of the differences). As shown in Fig. 2, constraints with the *Medium* complexity specified using Java obtained lower *Completeness* than the ones with *High* complexity though no significant difference was identified (see Row 8, Column 4 of Table 9). As shown in Row 9 and Columns 4 and 5 of Table 9, constraints with the *Medium*

Table 10 Correlation analysis among dependent variables with the significance level $\alpha = 0.05$

Variable	by Variable	Spearman ρ	Prob > $\lvert\rho\rvert$
Completeness	Applicability	0.1846	**<0.0001**
Conformance	Applicability	0.3266	**<0.0001**
Redundancy	Applicability	−0.0309	0.4672
ConfidenceLevel	Applicability	0.8046	**<0.0001**
ConfidenceLevel	Completeness	0.1814	**<0.0001**
ConfidenceLevel	Conformance	0.2938	**<0.0001**
ConfidenceLevel	Redundancy	−0.0679	0.1069

complexity specified by the participants using Java have significantly lower *Conformance* than constraints with the other two levels of complexity (on average 13 or 12 % lower than the constraints classified as the *High* or *Low* complexity, respectively, as shown in Table 6). This result indicates that the complexity of constraints has impact on the quality of specified constraints when Java was used. Recall that in our experiment, the ten constraints for each system were ordered from the simplest one to the most complex one. The participants started from the simplest ones (*Low*) to more complex ones (*Medium* and *High*). For constraints with low complexity, the participants performed well. Gradually along with the increase in the complexity, their performance decreased. But they gained experience of applying Java for specifying constraints after finished roughly two third of the constraints, which eventually leads to the fact that they performed well for the constraints with the *High* complexity.

3.3 Correlation analysis

It is also interesting to know whether there are correlations between the quality measures (i.e., *Completeness*, *Conformance* and *Redundancy*) and the measures measuring the participants' subjective opinions on the applicability of two methods: *Applicability* and *ConfidenceLevel*. To this end, we conducted the nonparametric Spearman's ρ test [38]. Results are reported in Table 10. Spearman's ρ is used to determine the wellness of dependence relationship between two dependent variables. The value of ρ ranges from −1 to +1. When the value is 0 this means that there is no dependence between two variables. A positive value means the value of one dependent variable increases as the value of the second dependent variable increases. A negative value of ρ shows increasing the value of one dependent variable decreases the value of the second dependent variable. In addition to reporting ρ, a p value is often reported to show the significance of the relationship.

From Table 10, Rows 1–2 and Column 4, one can observe that *Completeness* and *Conformance* are significantly correlated with *ConfidenceLevel* and *Applicability*, as the p values are less than 0.05. This result indicates that the objective measures of the quality of constraints and the subjective opinions of the participants on the two methods are nicely consistent. In other words, a participant who was more confident to apply a method to specify a constraint and thought the method was easier to apply specified constraints with higher quality. Significant correlation was identified between *ConfidenceLevel* and *Applicability*, which implies that when participants were confident also thought the methods were easy to apply.

3.4 Additional analysis

We also conducted additional analyses to understand how far the student's derived constraint specifications are from the fully correct specifications in the sense that they are ready to be processed by tools for supporting automation. From the constraint specifications derived by the students, we selected 25 specifications that achieved 100 % conformance for each system and each method. In total, 100 constraint specifications were selected and inputted to IBM RSA for checking OCL constraints and Eclipse IDE for Java development for checking Java specifications.

Out of the 25 selected OCL constraints for the Banking System, 12 of them contained errors that need to be fixed before being used for supporting automation. For VCS, fifteen OCL constraints have errors identified. We report the identified error types and number of errors in Table 11. One can see that most of the errors are due to syntactically incorrect reference to enumeration literals. For example, in OCL, an enumeration literal is referred as "*Enumeration Name:: Enumeration Literal*." Some students mistakenly referred to enumerations in one of the following ways: (1) Referring to the enumeration literal as in Java, i.e., "*Enumeration Name. Enumeration Literal*"; (2) Referring only with the name of the enumeration literal, i.e., "Enumeration Literal."

For the selected 25 Java specifications for the Banking System, ten out of them contained one or more errors. For VCS, nine out of 25 contained one or more errors; three of them were caused by the incorrect design of the class diagram. As it can be seen in Fig. 5, the cardinalities from *Saturn* to *SIP* and *H*323 classes are exactly one. In the correct design, it should have been "0..1." In addition for VCS, we found seven instances where the students referred to objects in a syntactically wrong way (J1 in Table 12). For example, an object *s* of type *Saturn* was given to the students as the starting point for traversal, but in these seven instances, the students did not start the traversal from *s*. We report the identified error types and number of errors in Table 12. In total, fifteen errors were identified in the 10 constraints for Banking and 20 errors were identified in the 9 constraints for VCS.

Based on the total number of errors observed, it seems that constrains specified in Java contain more errors than constrains specified in OCL. However, to further confirm this,

Table 11 Errors observed when tools were used (OCL)

ID	Error type	Number of errors	
		Banking	VCS
O1	Referring to an enumeration literal in a syntactically wrong way	4	11
O2	Referring to a subtype in a syntactically wrong way	1	0
O3	Missing 'self.'	1	1
O4	Spelling mistakes	2	0
O5	Missing either a left or right bracket	1	0
O6	Missing "endif"	1	0
O7	'.' is used instead of '->' when referring to size()	1	0
O8	Redundant if-then-else-endif statements	1	0
O9	The class diagram provided is not fully correct	N/A	3
Total		**12**	**15**

Table 12 Errors observed when tools were used (Java)

ID	Error type	Number of errors	
		Banking	VCS
J1	Referring to an object in a syntactically wrong way	0	7
J2	Referring to an enumeration literal in a syntactically wrong way	0	1
J3	Misused uppercase or lowercase when referring to objects	6	4
J4	'Boolean' is used instead of 'bool'	5	1
J5	Missing semicolon	1	2
J6	Missing double quotation marks for strings	0	4
J7	Misspelling	1	1
J8	Extra bracket	1	0
J9	Used function size() to obtain the size of an array instead of ".length"	1	0
Total		**15**	**20**

another controlled experiment is needed with students specifying constraints directly in Java and OCL tools. We plan to conduct such experiment in the future. Another observation is that tools are needed for specifying fully correct OCL and Java constraints. We can also learn from the results of the experiment that the error types reported in Tables 11 and 12 are easy to fix with tool support.

3.5 Overall discussion

Based on the results presented in Sects. 3.1, 3.2 and 3.3, we can observe that the performance of the participants specifying constraints using both OCL and Java is equally well as shown in Table 5. The mean *Completeness* for OCL is 91 and 89 % for Java, whereas mean *Conformance* for OCL is 73 and 71 % for Java as shown in Table 5. Moreover, the specified constraints have low mean *Redundancy*, i.e., 1 % for OCL and 4 % for Java (Table 5).

Based on the results presented in Sect. 3.2, we observed that the participants who worked with OCL performed consistently well to specify constraints of varying complexity; however this was not the case for Java. Based on these results, it is apparent that it does not make much difference to use OCL or Java for specifying constraints on UML models. However, since the performance of the participants working with OCL is not affected by the complexity of constraints, we recommend using OCL for specifying constraints on UML models. Even when one has to specify complex constraints, as is the case in most of the industrial applications, we expect the better performance with OCL as compared to Java.

Notice that the purpose of the experiment is to compare OCL and Java in terms of specifying constraints on UML Models at the design time and we were not interested in studying the runtime details of Java and OCL. Moreover, some of OCL evaluators such as Dresden OCL [10] and Eclipse OCL [8] translate OCL constraints into Java for evaluation at the

backend and thus all the runtime issues of Java are the same as for OCL.

It is important to point it out that the motivation of the work is to test the capability of human subjects in terms of specifying constraints using OCL and Java. We do not aim to use, in the context of this controlled experiment, manually derived specifications for any particular purpose (e.g., generating test data). Therefore, evaluating constraints is out of the scope of this experiment. In addition, evaluating an OCL constraint and the execution of a Java program require OCL evaluators and Java compilers, respectively, and thus are out of the scope of this controlled experiment.

In this experiment, we only measure the semantic conformance of derived OCL and Java constraint specifications against provided English specifications and class diagrams. Since we focus on comparing OCL and Java in terms of specifying constraints by mentally understanding UML class diagrams and constraints written in English, measuring syntactic conformance of these constraint specifications is not within the scope of this experiment. This is due to the reason that a tool can easily check syntactic conformance of OCL and Java constraint specifications, but it cannot ensure their semantic conformance against requirements and validating their semantic conformance has to be manual, which leads to the definition of the complexity metrics as discussed in Sect. 2.3.1.

4 Threats to validity

Below, we discuss the threats to validity of our controlled experiment based on the concepts discussed in [36]. Conclusion validity threats are concerned with factors that can influence the conclusion that can be drawn from the results of the experiments. As with most controlled experiments in software engineering, our main conclusion validity threat is related to the sample size on which we base our analysis. To deal with this, our experiment design required modeling 10 constraints per system (20 in total for two systems) to maximize the number of observations within time constraints. The other concern is that the quality of constraints specification can be interpreted in various ways, depending on one's subjective opinion. However, we made an effort to minimize subjective judgments by proposing a set of objective metrics to measure the quality of constraints. By doing so, subjective perceptions can be reduced to minimum and the comparison of constraints derived by different participants becomes possible.

Internal validity threats exist when the outcome of results is influenced by confounded factors and are not necessarily due to the application of the treatment being studied. Through our experiment design, we have tried to minimize the chances of other factors being confounded with our primary independent variable: the use of OCL and Java. We used a within-subjects design and matched pairs analysis since the strength of this design is that the variation due to differences in participants is eliminated as each participant acts as its own control. We avoided any biased assignment of participants to groups by randomization and blocking based on questionnaire results. The experiment participants were provided with constraints that are written in English as the input, which are inherently ambiguous. Therefore, it might have the impact on the quality of the derived OCL and Java constraints. However, it is worth noticing that the participants using either OCL or Java for the same system were provided with the same set of constraints in English. Therefore, we do not expect this threat having any impact on the comparison of OCL and Java. Another concern is the proficiency of the English language of the students. Explaining the constraints in English by the authors to the participants during the experiment reduces the potential impact of proficiency of the English of participants on their performance. We also avoided the possible impact of this factor by using the within-subjects design and matched pairs analysis.

Regarding construct validity, the main threat is that we were not able to investigate all features of OCL (such as specialized operations including *oclInState*) in this experiment due to the nature of our case studies. This will require replications with different systems, which we plan to conduct in the future.

The main threat to external validity is typical of controlled experiments in artificial settings: Are the participants representative of software professionals? Many practitioners have anyway very little knowledge of OCL in general, and hence require training. Note also that we chose a group of experienced graduate students with an advanced educational background (Sect. 2.2). In addition, some studies [39–41] have been reported on the performance, for various tasks, of trained software engineering students when compared with professional developers. These differences were not statistically significant when compared to junior and intermediate developers, thus suggesting that there is no evidence that students trained for the tasks at hand may not be used as participants in place of professionals.

5 Related work

OCL is a standard language that is widely accepted for writing constraints on UML models. OCL is based on first-order logic and set theory and provides various constructs (e.g., collection operations) to define constraints in a concise form. The language allows modelers to write constraints at various levels of abstraction and for various types of models. For example, it can be used to write class and state invariants, guards in state machines, constraints in sequence

diagrams, and pre- and post-conditions of operations. Our several industrial case studies have shown the benefits that it can bring to solve various industrial problems such as supporting automated model-based test case generation and automated model-based consistency checking and configuration in the context of produce line engineering [5,6,15,18–20,26,28,30]. OCL is also being used as the language for writing constraints on models in many commercial MBT tools such as CertifyIt [13] and QTronic [42].

Java is a programming language that has been very widely used and supported by a lot of tools. However, the modeling community does not often notice that Java can also be used as a constraint language to specify constraints in UML models. Considering Java is widely known and practiced by software developers and has a large number of tools available in the market, gradually it becomes an option, in addition to OCL, to specify constraints on UML models. This observation is supported by the fact that some market-leading UML modeling tools such as IBM RSA [33] and Papyrus [32] support using both OCL and Java to specify constraints. It is, however, rarely reported in the literature, with scientific evidence, which of these two languages (i.e., OCL or Java) is better in terms of specifying constraints on UML class diagrams.

In the rest of the section, we discuss several representative tools that provide support for using OCL and/or Java for specifying constraints (Sect. 5.1), followed by the related work reporting controlled experiments empirically evaluating the impact of OCL in UML-based maintenance and the understandability of OCL (Sect. 5.2).

5.1 Tools that implement OCL and/or Java for specifying constraints

In UML models, constraints can be specified using different types of languages such as natural language, programming languages (e.g., Java and C++), and OCL. Some existing open source and commercial tools such as IBM RSA [33] and Papyrus [32] provide modeling environments for users to specify constraints in UML models with various languages. However, OCL and Java are two commonly implemented constraint specification languages and some modeling tools also support automated validation of constraints specified in OCL and/or Java. In the following section, we briefly discuss some widely used modeling tools that have OCL and/or Java implemented as their constraint specification languages.

IBM RSA [33] allows one to specify constraints either using OCL or Java. One argument for supporting both languages is that "Java might be easier to use to express complex constraints, and offer great flexibility" and "OCL is more consistent with how OMG defines UML constraints" [43]. Notice that this argument is not supported with any scientific evidence. The empirical study, we conducted exactly aims to test which one is easier to use and which one can handle complex constraints better. The results reported in Sect. 3 reveal that OCL performs significantly better than Java in terms of handling complex constraints, which is not consistent with the argument provided in [43]. Open source modeling tool Papyrus [32] allows users to specify constraints using OCL, Java, natural language, C, and C++. However, as mentioned in [44], to make specified constraints usable by Papyrus, constraints must be written in OCL or Java such that specified constraints can be validated automatically.

OneModelica [45] is the IDE designed for the Modelica modeling language. In [46] a study was reported to compare OCL and Java in the context of OneModelica for Modelica code validation. OCL and Java were compared to each other regarding two aspects: readability of constraints as well as execution performance. The first comparison aspect is closely relevant to the objective of the controlled experiment reported in this paper. The authors of the paper [46] concluded, via subjective language concept comparison, that the readability of OCL constraints is "very good" as compared to Java. However, more software developers can understand Java and tool support is a quite important benefit as compared to OCL. This conclusion conforms to what we observed from the controlled experiment. However, it is important to notice that the controlled experiment we conducted is a scientific way to provide evidence and the results of the experiment were analyzed and evaluated with more objective metrics (e.g., conformance, completeness) instead of a very subjective evaluation of readability.

MagicDraw [31], Enterprise Architect [47], and argoUML [48] are another three UML modeling tools that provide capability of specifying OCL constraints and validating them. None of them, however, support specifying constraints using Java.

5.2 Controlled experiments

Briand et al. [49,50] conducted a controlled experiment to evaluate the impact of OCL in UML-based maintenance, from the perspective of using OCL on model comprehension and maintainability. The motivation was to assess the benefits (precision) that OCL brings when applying it in UML-based development, considering the additional effort required and extra formality introduced. Results show that an initial learning curve is required to gain significant benefits when using OCL in combination with UML diagrams. To compare with our experiment, we evaluate the applicability of OCL in combination with UML by comparing it with Java, which can equivalently do the same thing. Our motivation is to collect evidence and provide arguments in a scientific way which of these two languages is better. Therefore, we can recommend it to our industrial partners.

Correa et al. reported a controlled experiment in [51] to evaluate the impact of bad OCL expressions and their refactoring on the understandability of OCL specifications. Results show that most refactoring significantly improves the understandability of OCL specification. We did not find any other work relating to empirical evaluation of OCL or Java.

Harald Störrle reported in [52] the results of a series of controlled experiments to evaluate the usability of the OCL Query API (OQAPI), which was designed for the purpose of improving the usability of OCL for supporting querying. Experiment results show that OQAPI is easy to use in terms of facilitating user querying using OCL.

Based on the above-related work, we can conclude that the controlled experiment reported in this paper is one of the first experiments that were exclusively designed to compare OCL and Java for specifying constraints on UML models. The results of the experiment provide some evidence that can be used by practitioners and academics to choose a language for specifying constraints for their specific problems.

6 Conclusion

The Object Constraint Language (OCL) has been widely used along with UML models for various purposes such as supporting model-based testing and configuration of products in a product line. From the last several years, we have been working on various industrial projects on model-based engineering (MBE), which involved using the OCL. One of the major challenges that we faced is the limited evidence about the applicability of OCL in the literature as compared to, e.g., Java. Such evidence is important to convince the industrial partners about the use of OCL in the industry.

To collect some evidence about the use of OCL, we reported a controlled experiment that was conducted to evaluate the "applicability" of OCL by comparing it with one of the most commonly used programming languages in terms of applying them to specify constraints on UML class diagrams. We looked at applicability from two aspects: the quality of specified constraints in terms of completeness, conformance, and redundancy, and subjective opinions of participants on the applicability and their confidence of applying the two languages.

Experiment results showed that both OCL and Java are equally good: Completeness and conformance of the specified constraints were high, and there were very few redundant clauses in the specified constraints. We also observed that the applicability of OCL is not impacted by the complexity of constraints. This observation gives us confidence that OCL scales well when it is used for specifying complex constraints, which are commonly seen in industrial settings. However, this is not a case for Java whose performance is influenced by the complexity of constraints.

Moreover, we performed additional analyses where we took 100 constraints in Java and OCL that have 100 % conformance. These constraints were inputted to OCL and Java tools to identify additional errors in their specification. Results show that the constraints specified in Java contain more errors than in the ones in OCL. These results suggest that tools are absolutely needed for specifying fully correct OCL and Java constraints.

Based on the results of the experiment, we recommend using OCL for specifying constraints on UML models for addressing large-scale industry problems, especially for industrial contexts where Java is not used as the development language.

Appendix A: Answer sheet of the Banking System case study

In this Appendix, we present the answer sheet that was provided to the students during the controlled experiment for Banking System, including the instruction, the system description, the class diagram that OCL constraints should be applied on, and the 10 constraints written in English.

A.1 Instruction for specifying constraints using OCL

Check the provided system description and the class diagram, and specify the given constraints using OCL with the following format:

```
context Bank
      inv:
```

A.2 Instruction for specifying constraints using Java

Check the provided system description and the class diagram, and specify the given constraints using Java. All the attributes in the class diagram are public, which means that they can be directly accessed with objects. Please provide the specification of each constraint as the body of the following function:

```
public boolean constraint (Bank b){
}
```

A.3 System description

This case study is an extended version of Banking System case study from the OCL 2.2 specification. A bank has several employees and customers. Each customer can have at most two accounts in a bank: One is saving account and the other is current account. A customer must be employed in a company or owns a company in order to have a bank account. An

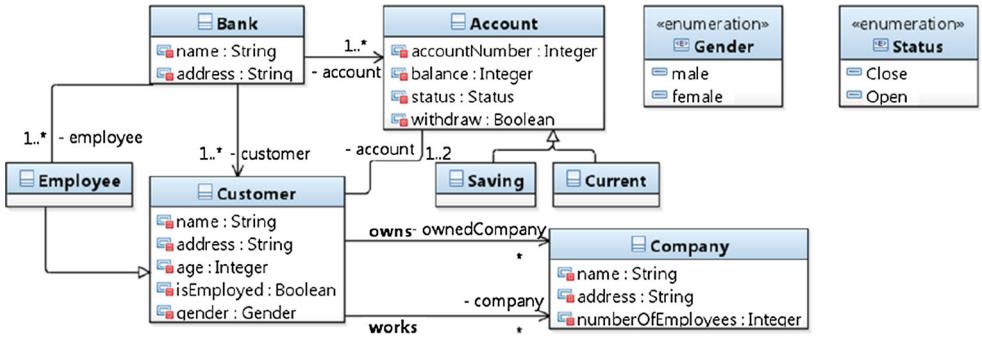


Fig. 3 Class diagram for Banking System

Table 13 Description of each attribute

Class	Attribute	Description
Bank	Name	Name of a bank
	Address	Address of a bank
Customer	Name	Name of a customer
	Address	Address of a customer
	isEmployed	True if a customer is employed or owns a company; false otherwise
	Gender	Male or female
Account	accountNumber	A unique account number of a customer
	Balance	Amount in an account
	Status	Close if a bank account is closed down; open otherwise
	withdraw	True if a certain amount can be withdraw from an account; false otherwise
Company	Name	Name of a company
	Address	Address of a company
	numberOfEmployees	Number of employees in a company

employee of a bank can also be its customer having accounts in the bank.

A.4 Class diagram

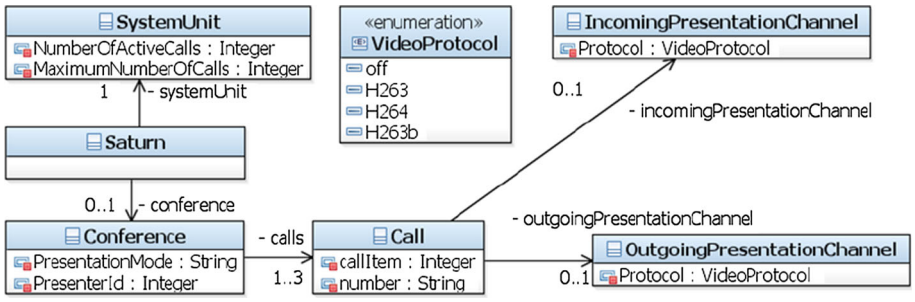
Figure 3 shows a class diagram modeling the Banking System. Table 13 provides the description of each attribute of a class in the class diagram.

A.5 Constraints

1. All customers of the bank must be employed.
2. All customers and employees of the bank must be 18 years or older.
3. If all the accounts of a customer have balance less than or equal to 0, then these accounts should be all closed.
4. A customer is either employed in a company or owns his/her company.
5. All accounts in the bank must have unique account numbers and each account must be linked to exactly one customer.
6. Each customer of the bank either owns at least a company or work in a company that has more than one employee.
7. The bank does not allow their customers to withdraw money from their saving accounts.
8. In the bank, there should be gender equality in its employees, i.e., the number of male employees should be equal to the number of female employees.
9. An employee of the bank must not work in another company, but may own a company with exact one employee.
10. A customer of the bank can only have a saving account when he/she has a current account but not vice versa.

Appendix B: Answer sheet of the Video Conferencing System (VCS) case study

In this Appendix, we present the answer sheet that was provided to the students during the controlled experiment for Video Conferencing System, including the instruction, the system description, the class diagram that OCL constraints should be applied on, and the 10 constraints written in English.



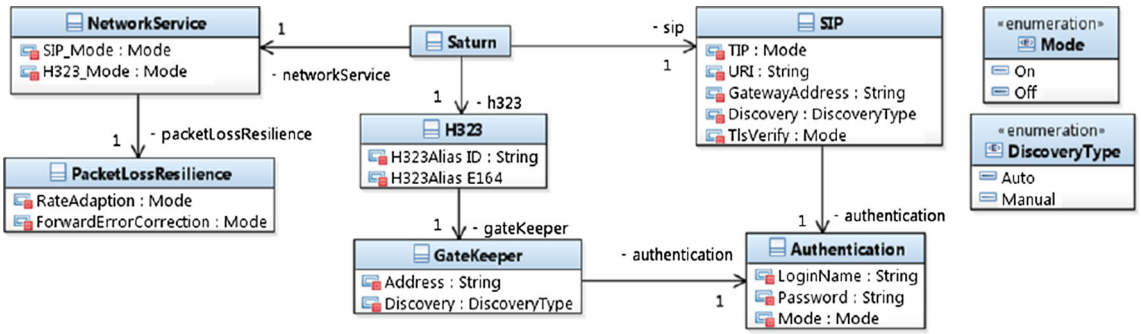


Fig. 4 Class diagram for Saturn (Part I)

Fig. 5 Class diagram for Saturn (Part II)

Table 14 Description of each attribute

Class	Attribute	Description
System Unit	NumberOfActiveCalls	Holds number of VCSs in a videoconference
	MaximumNumberOfCalls	Maximum number of calls supported by Saturn (3 in this case)
Conference	PresentationMode	Saturn is "off" when none of VCSs is presenting. Saturn is "Receiving" when a VCS other than Saturn is presenting. Saturn is "Sending" when Saturn is presenting
	PresenterId	A non-negative Id of the VCS currently presenting
Call	CallItem	It is a non-negative Id of a call to a VCS
	Number	Number or IP address of a VCS that is in a videoconference with Saturn
IncomingPresentationChannel	Protocol	Protocol used for presentation, which is: Off, H263, H264, and H263b
OutgoingPresentationChannel	Protocol	

B.1 Instruction for specifying constraints using OCL

Check the provided system description and the class diagram, and specify the given constraints using OCL with the following format:

```
context Bank
      inv:
```

B.2 Instruction for specifying constraints using Java

Check the provided system description and the class diagram, and specify the given constraints using Java. All the attributes in the class diagram are public, which means that they can be directly accessed with objects. Please provide the specification of each constraint as the body of the following function:

```
public boolean constraint (Saturn s){
}
```

B.3 System description

Our case study is part of a project aiming at supporting automated, model-based testing of a core subsystem of a video conferencing system (VCS) called Saturn. The core functionality to be modeled manages the sending and receiving of multimedia streams. Audio and video signals are sent through separate channels, and there is also a possibility of transmitting presentations in parallel with audio and video. Only one conference participant can send presentations at a time and all others receive it.

B.4 Class diagram

The functional behavior of Saturn consists of a set of class diagrams and a set of UML state machines. An excerpt of class diagram for Saturn is provided in Fig. 4. The UML class diagram is meant to capture information about APIs and system (state) variables, which are required to generate executable test cases in our application context. In this figure, however, we do not show APIs, since we do not need them in this context. Figure 5 is also a class diagram for Saturn capturing various configuration parameters. The *Saturn* class in Figs. 4 and 5 is the same, and we present two separate class diagrams for the purpose of clarity. Table 14 shows the description of each attribute.

B.5 Constraints

1. Saturn should be either in SIP mode or H323, but not in both modes at the same time.
2. Number of active calls for Saturn ranges from 0 to 3.
3. When Saturn is presenting, the presenter's ID should be a non-negative integer.
4. For each call, call item should be a non-negative integer and call number shouldn't be an empty string ("").
5. For each call, call item and number should be unique.
6. In H323 mode, rate adaption and forward error correct mode should be enabled at the same time.
7. When Saturn is in SIP mode then all the information related to SIP protocol shouldn't be empty.
8. When Saturn is presenting, protocols of all the outgoing presentation channels should not be off.
9. When Saturn is receiving presentation, exactly one of input presentation channels should have protocol not equal to off.
10. When Saturn's presenting or receiving presentation, exactly one of the presentation channels should have protocol not equal to "Off".

Appendix C: Post-questionnaire

Please put a ($\surd$) in the corresponding column. You are strongly encouraged to refer to the list of constraints you were provided with. (The constraint numbers in the following tables match the numbers provided in the list of restriction specifications).

A: Completely agree; B: Generally agree; C: Generally disagree; D: Completely disagree

Constraint#	I understand the provided class diagram clearly				I understood the constraint properly				The constraint was easy to specify with the requested language.				I feel confident to apply the language			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
1																
2																
3																
4																
5																
6																
7																
8																
9																
10																

Appendix D: Pre-questionnaire

Levels of agreement: Strongly agree, Agree, Neither agree nor disagree, Disagree, and Strongly disagree

Likert Scale Questions:

1. I have good knowledge on UML class diagram modeling.
2. I have good knowledge on writing OCL expressions.
3. I have good knowledge on programming using Java.

Open Questions:

4. How many courses have you taken that taught UML? What are these courses?
5. How many courses have you taken that taught OCL? What are these courses?
6. How many Java-programming projects have you conducted in the past? What are they?
7. How many OCL and UML related courses have you conducted in the past? What are they?
8. What other kinds training on UML, OCL, or Java have you received in the past?

Appendix E: Examples of OCL and Java constraints

In this Appendix, we provide some examples of OCL and Java constraints taken from the answer sheets submitted by

the students during the controlled experiment. We also provide the evaluation results based on the metrics defined in Sect. 2.4.1.

E.1 OCL, Banking System

Banking System:

Constraint A: All customers of the bank must be employed.

Student A:

```
Context Bank
inv: self.customer->iterate(c:Customer|
c.isEmployed = true)
```

Completeness = 1, Conformance = 1, and Redundancy = 0

Student B:

```
Context Bank
inv: for each self.employee.isEmployed
= true
```

Completeness = 1, Conformance = 0.67, and Redundancy = 0

Constraint B: All accounts in the bank must have unique account numbers and each account must be linked to exactly one customer.

Student A:

```
Context Bank
inv: self.amount->isUnique(account
Number) and self.account.customer->size()
=1
```

Completeness = 1, Conformance = 0.83, and Redundancy = 0

Student B:

```
Context Bank
inv: if for all self.account.account
Number is unique and for each self.
account.account
```

Completeness = 0.5, Conformance = 0.25, and Redundancy = 0

Constraint C: A customer of the bank can only have a saving account when he/she has a current account but not vice versa.

Student A:

```
Context Customer::Saving:Saving
inv: self.current->size()>=1
```

Completeness = 0.25, Conformance = 0.097, and Redundancy = 0

Student B:

```
Context Customer
inv: if self.current.size()>0 then
self.saving.size()>0
```

Completeness = 0.5, Conformance = 0.11, and Redundancy = 0

E.2 OCL, Video Conferencing Systems

Constraint D: Saturn should be either in SIP model or H323, but not in both modes at the same time.

Student C:

```
Context Saturn
inv: (self.networkService.SIP_Mode =
On and self.networkService.H323_Mode
= Off) or (self.networkService.SIP_mode
= Off and self.networkService.H323_Mode
= On)
```

Completeness = 1, Conformance = 0.75, and Redundancy = 0

Student D:

```
Context Saturn
inv: self.h323->size() + self.sip
->size() =1
```

Completeness = 1, Conformance = 1, and Redundancy = 0

Constraint E: For each call, call item and number should be unique.

Student C:

```
Context Saturn
inv: self.conference.calls->isUnique
(callItem) and self.conference.calls->
isUnique(number)
```

Completeness = 1, Conformance = 0.33, and Redundancy = 0

Student D:

```
Context Saturn
inv: self.conference.calls->isUnique
(CallItem) and self.conference.calls->
isUnique(number)
```

Completeness = 1, Conformance = 0.33, and Redundancy = 0

Constraint F: When Saturn's presenting or receiving presentation, exactly one of the presentation channels should have protocol not equal to "Off".

Student C:

```
Context Saturn
inv: OutgoingPresentationChannel::
Protocol <> Off or Incoming
PresentationChannel::Protocol <> Off
```

Completeness = 0.5, Conformance = 0.28, and Redundancy = 0

Student D:

```
Context Saturn
inv: if self.conference.PresenationMode <> Off then
      (self.conference.call -> One
(c|c.incomingPresentationChannel.Protocol <> Off) and
      self.conference.call-> forAll
(c|c.outgoingPresentationChannel.Protocal <> Off))
      or
(self.conference.call -> One (c|c.outgoingPresentationChannel.Protocol <>
Off) and
      self.conference.call-> forAll
(c|c.incomingPresentationChannel.Protocal <> Off) and)
```

Completeness = 1, Conformance = 1, and Redundancy = 0

E.3 Java, Banking System

Constraint A: All customers of the bank must be employed.

Student C:

```
public Boolean constraint (Bank b) {
      for (int i=0; i<b.customer.size(); i++){
      if (!b.customer(i).isEmployed)
            return false;
      continue;
}
return true;
}
```

Completeness = 1, Conformance = 1, and Redundancy = 0

Student D:

```
public Boolean constraint (Bank b){
      for (int i=0; i<b.customer.length; i++){
                  if (b.customer[i].current.length>0 &&
      b.customer[i].saving.length==0)
return false;
      }
return true;
}
```

Completeness = 1, Conformance = 0.67, and Redundancy = 0

Student D:

```
public Boolean constraint (Bank b){
      for (int i=0; i<b.customer.size(); i++){
      if (b.customer[i].account.size()==1 && b.customer.account[0]
instance of Saving)
            return false;
      else if (b.customer[i].account.size()==2 &&
b.customer.account[0] instance of Saving
&& b.customer.account[1] instance of Saving)
            return false;
}
return true;
}
```

Completeness = 1, Conformance = 1, and Redundancy = 0

E.4 Java, Video Conferencing System

Constraint D: Saturn should be either in SIP model or H323, but not in both modes at the same time.

Student A:

```
public Boolean constraint (Saturn){
      if (Saturn.networkService.SIP_Mode == On &&
     Saturn.networkService.H323_Mode = Off) return true;
else if (Saturn.networkService.SIP_Mode == Off &&
     Saturn.networkService.H323_Mode = On) return true;
else return false;
}
```

Completeness = 1, Conformance = 1, and Redundancy = 0.5

Student B:

```
public Boolean constraint (Saturn){
      return((Saturn.networkService.SIP_Mode == On
         && Saturn.networkService.H323_mode == Off) ||
             (Saturn.networkService.SIP_Mode == Off
                  && Saturn.networkService.H323_Mode == On));
}
```

Completeness = 1, Conformance = 1, and Redundancy = 0

Constraint E: For each call, call item and number should be unique.

Student A:

```
public Boolean constraint (Call){
      for (int i=0; i<call.length(); i++) {
            for(int j=0; j<i; j++){
      if (call[i].callItem == call[j].callItem &&
call[i].number == call[j].number ) return false;
}
}
return true;
}
```

Completeness = 1, Conformance = 0.83, and Redundancy = 0

Student B:

```
public Boolean constraint (Call){
      Boolean IsUnique = true;
      for (int i=0; i<call.lenght(); i++) {
            for (int j=i+1; j<call.length(); j++){
            if call[i].callItem == call[j].callItem ||
                  call[i].number == call[j].number
                  IsUnique = false;
            }
      }
      return IsUnique;
}
```

Completeness = 1, Conformance = 0.83, and Redundancy = 0

Constraint F: When Saturn's presenting or receiving presentation, exactly one of the presentation channels should have protocol not equal to "Off".

Student A:

```
public Boolean constraint (Saturn){
      int number =0;
      for (int i=0; i<Saturn.conference.call.length(); i++){
            if(Saturn.conference.call[i].incomingPresentationChannel.Protoc
      ol !=Off) number++;

      if(Saturn.conference.call[i].outgingPresentationChannel.Protocol
!=Off) number++;
      }
      if number ==1 return true; else return false;
}
```

Completeness = 0.5, Conformance = 0.5, and Redundancy = 0.5

Student B:

```
public Boolean constraint (Saturn){
      int channels=0;
      int IsOff =0;
      if          (Saturn.conference.PresentationMode==”sending”          ||
Saturn.conference.PresentationMode == ”receiving”) {
      for (int i=0; i<call.size(); i++){
      for(int j=0; j<call[i].incomingPresentationChannel.size(); j++){
      channels++;
      if (call[i].incomingPresentationChannel[j].Protocol == “Off”)
            IsOff++;
}
      for(int j=0; j<call[i].outgoingPresentationChannel.size(); j++){
      channels++;
      if (call[i].outgoingPresentationChannel[j].Protocol == “Off”)
            IsOff++;
}
}
if (channels-IsOff = 1) return false;
else return true;
}
}
```

Completeness = 1, Conformance = 1, and Redundancy = 0.5

References

1. Object Management Group (OMG), http://www.omg.org/spec/OCL/2.2/
2. Lefticaru, R., Ipate, F.: Functional search-based testing from state machines. In: Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation. IEEE Computer Society (2008)
3. Binder, R.V.: Testing Object-oriented Systems: Models, Patterns, and Tools. Addison-Wesley, Reading (1999)
4. Iqbal, M.Z., Ali, S., Yue, T., Briand, L.: Experiences of applying UML/MARTE on three industrial projects. In: France, R.B., Kazmeier, J., Breu, R., Atkinson, C. (eds.) ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS), vol. 7590, pp. 642–658. Springer, Berlin (2012)
5. Ali, S., Briand, L.C., Hemmati, H.: Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems. Softw. Syst. Model. **11**, 633–670 (2012)
6. Ali, S., Briand, L., Arcuri, A., Walawege, S.: An industrial application of robustness testing using aspect-oriented modeling, UML/MARTE, and search algorithms. In: ACM/IEEE 14th International Conference on Model Driven Engineering Languages and Systems (Models 2011) (2011)
7. Drusinsky, D.: Modeling and Verification Using UML Statecharts: A Working Guide to Reactive System Design, Runtime Monitoring and Execution-based Model Checking, Newnes (2006)
8. http://www.eclipse.org/modeling/mdt/?project=ocl
9. Chiorean, D., Bortes, M., Corutiu, D., Botiza, C., Cârcu, A.: OCLE (2010)
10. http://www.dresden-ocl.org/index.php/DresdenOCL
11. http://sourceforge.net/apps/mediawiki/useocl/index.php?title=The_UML-base_Specification_Environment
12. Egea, M.: EyeOCL Software (2010)
13. Smarttesting, http://www.smartesting.com/
14. Bordbar, B., Anastasakis, K.: UML2Alloy: a tool for lightweight modelling of discrete event systems. In: IADIS International Conference in Applied Computing (2005)
15. Aertryck, L.v., Jensen, T.: UML-Casting: Test synthesis from UML models using constraint resolution. Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL '2003) (2003)
16. Distefano, D., Katoen, J.-P., Rensink, A.: Towards model checking OCL. ECOOP-Workshop on Defining Precise Semantics for UML (2000)
17. Clavel, M., Dios, M.A.G.d.: Checking unsatisfiability for OCL constraints. In: The Proceedings of the 9th OCL 2009 Workshop at the UML/MoDELS Conferences (2009)
18. Bao-Lin, L., Zhi-shu, L., Qing, L., Hong, C.Y.: Test case automate generation from uml sequence diagram and ocl expression. In: International Conference on Computational Intelligence and Security, pp. 1048–1052 (2007)
19. Benattou, M., Bruel, J., Hameurlain, N.: Generating Test Data from OCL Specification. Citeseer, Princeton (2002)
20. Aichernig, B.K., Salas, P.A.P.: Test case generation by OCL mutation and constraint solving. In: Proceedings of the Fifth International Conference on Quality Software. IEEE Computer Society (2005)
21. Cabot, J., Claris, R., Riera, D.: Verification of UML/OCL class diagrams using constraint programming. In: Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop. IEEE Computer Society (2008)
22. Berardi, D., Calvanese, D., Giacomo, G.D.: Reasoning on UML class diagrams. Artif. Intell. **168**, 70–118 (2005)
23. Winkelmann, J., Taentzer, G., Ehrig, K., Ster, J.M.: Translation of restricted OCL constraints into graph constraints for generating meta model instances by graph grammars. Electron. Notes Theor. Comput. Sci. **211**, 159–170 (2008)
24. Kyas, M., Fecher, H., Boer, F.S.D., Jacob, J., Hooman, J., Zwaag, M.V.D., Arons, T., Kugler, H.: Formalizing UML models and OCL constraints in PVS. Electron. Notes Theor. Comput. Sci. **115**, 39–47 (2005)
25. Krieger, M.P., Knapp, A., Wolff, B.: Automatic and efficient simulation of operation contracts. In: 9th International Conference on Generative Programming and Component Engineering, (2010)
26. Weißleder, S., Schlingloff, B.-H.: Deriving Input Partitions from UML Models for Automatic Test Generation. Models in Software Engineering, pp. 151–163. Springer, Berlin (2008)
27. Gogolla, M., Bttner, F., Richters, M.: USE: a UML-based specification environment for validating UML and OCL. Sci. Comput. Program. **69**, 27–34 (2007)
28. Brucker, A.D., Krieger, M.P., Longuet, D., Wolff, B.: A specification-based test case generation method for UML/OCL. In: Proceedings of the 2010 International Conference on Models in Software Engineering, pp. 334–348. Springer, Oslo, Norway (2011)
29. Ahrendt, W., Baar, T., Beckert, B., Giese, M., Habermalz, E., H, R., #228, hnle, Menzel, W., Schmitt, P.H.: The KeY approach: integrating object oriented design and formal verification. Proceedings of the European Workshop on Logics in Artificial Intelligence, pp. 21–36. Springer (2000)
30. Ali, S., Iqbal, M.Z., Arcuri, A., Briand, L.: A Search-based OCL constraint solver for model-based test data generation. In: Proceedings of the 11th International Conference On Quality Software (QSIC 2011). IEEE Computer Society, pp. 41–50 (2011)
31. http://www.nomagic.com/products/magicdraw.html
32. http://www.eclipse.org/modeling/mdt/papyrus/
33. IBM Rational Software Architect. http://www-03.ibm.com/software/products/en/ratisoftarch
34. Arcuri, A., Fraser, G.: On parameter tuning in search based software engineering. In: International Symposium on Search Based Software Engineering (SSBSE). Springer's Lecture Notes in Computer Science (LNCS) (2011)
35. IBM, http://www-01.ibm.com/software/awdtools/library/standards/ocl-download.html
36. Wohlin, C., Runeson, P., Höst, M.: Experimentation in Software Engineering: An Introduction. Springer, Berlin (1999)
37. http://www.jmp.com/
38. Sheskin, D.J.: Handbook of Parametric and Nonparametric Statistical Procedures. Chapman and Hall/CRC, Boca Raton (2007)
39. Höst, M., Regnell, B., Wohlin, C.: Using students as subjects—a comparative study of students and professionals in lead-time impact assessment. Empir. Softw. Eng **5**, 201–214 (2000)
40. Arisholm, E., Sjoberg, D.I.K.: Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. IEEE Trans. Softw. Eng. **30**, 521–534 (2004)
41. Holt, R.W., Boehm-Davis, D.A., Shultz, A.C.: Mental representations of programs for student and professional programmers. In: Gary, M.O., Sylvia, S., Elliot, S. (eds.) Empirical Studies of Programmers: Second Workshop, pp. 33–46. Ablex Publishing Corp, New York (1987)
42. CONFORMIQ, http://www.conformiq.com/qtronic.php
43. Hesari, S., Behjati, R., Yue, T.: Towards a systematic requirement-based test generation framework: industrial challenges and needs. In: 21st IEEE Requirements Engineering Conference, pp. 261–266 (2013)
44. Nie, K., Yue, T., Ali, S., Zhang, L., Fan, Z.: Constraints: the core of supporting automated product configuration of cyber-physical systems. In: Moreira, A., Schätz, B., Gray, J., Vallecillo, A., Clarke, P. (eds.) Proceedings of 16th international conference on model

driven engineering languages and systems, vol 8107, pp. 370–387 (2013)
45. http://www.onewind.de/OneModelica.html
46. Strach, R., Mareike, S.: Static validation of modelica models for language compliance and structural integrity. In: Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, Linköping University Electronic Press, Linköpings universitet, University of Nottingham, Nottingham, UK (2013)
47. http://www.sparxsystems.com/
48. http://argouml.tigris.org/
49. Briand, L., Labiche, Y., Yan, H., Di Penta, M.: A controlled experiment on the impact of the object constraint language in UML-based maintenance. In: Software Maintenance, 2004, Proceedings. 20th IEEE International Conference on, pp. 380–389. IEEE (2004)
50. Briand, L.C., Labiche, Y., Di Penta, M., Yan-Bondoc, H.: An experimental investigation of formality in UML-based development. IEEE Trans. Softw. Eng. **31**, 833–849 (2005)
51. Correa, A., Werner, C., Barros, M.: An empirical study of the impact of OCL smells and refactorings on the understandability of OCL specifications. In: Model Driven Engineering Languages and Systems, pp. 76–90 (2007)
52. Störrle, H.: Improving the usability of OCL as an ad-hoc model querying language. In: 13th International Workshop on OCL, Model Constraint and Query Languages (OCL 2013), pp. 83–92 (2013)

Tao Yue is a senior research scientist of Simula Research Laboratory and an associate professor at University of Oslo, Oslo, Norway, where she leads the expertise area of Model Driven Engineering (MDE). She has around 16 years of experience of conducting industry-oriented research with a focus on MDE in various application domains such as Avionics, Maritime and Energy, and Communications in several countries including Canada, Norway, and China. Her main research area is software engineering, with specific interested in requirements engineering, model-based development, requirements-based testing, model-based configuration and variability modeling, and empirical software engineering.

Shaukat Ali is currently a research scientist in Certus Software Verification and Validation Center, Simula Research Laboratory, Norway. He has been affiliated to Simula Research Lab since 2007. He has been involved in many industrial and research projects related to Model-based Testing (MBT) and Empirical Software Engineering since 2003. He has experience of working in several industries and academic research groups in many countries including UK, Canada, Norway, and Pakistan. Shaukat has been on the program or organisation committees of several international, IEEE and ACM conferences such as ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, European Conference on Modeling Foundations and Applications and System Analysis and Modelling Conference.