Expressing Model Constraints Visually with VMQL

Störrle, Harald

Published in:
2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)

Link to article, DOI:
10.1109/VLHCC.2011.6070399

Publication date:
2011

Expressing Model Constraints Visually with VMQL

Harald Störrle
Department of Informatics and Mathematical Modeling,
Technical University of Denmark
Richard Petersens Plads, 2800 Lyngby, Denmark
hsto@imm.dtu.dk

***Abstract*—OCL is the de facto standard language for expressing constraints and queries on UML models. However, OCL expressions are very difficult to create, understand, and maintain, even with the sophisticated tool support now available. In this paper, we propose to use the Visual Model Query Language (VMQL) for specifying constraints on UML models. We examine VMQL's usability by controlled experiments and its expressiveness by a representative sample. We conclude that VMQL is less expressive than OCL, although expressive enough for most of the constraints in the sample. In terms of usability, however, VMQL is superior to OCL, although the experimental evidence we present here is not as compelling as the one we presented when comparing VMQL and OCL on model querying.**

I. Introduction

While the Unified Modeling Language (UML) has been the "*lingua franca of software engineering*" for a long time now, uptake of the Object Constraint Language (OCL) accompanying UML has been very limited or even downright disappointing, when looking at industrial applications. Even in the UML standard itself, only slightly more than half of all constraints are expressed in OCL, even though there is now excellent tool support for OCL. Previous work (cf. [18]) indicated that there are three reasons for this: the poor usability of OCL as a language, the media gap between the all textual OCL and the mainly visual UML, and the inadequate level of abstraction provided by OCL. VMQL has been introduced to overcome these shortcomings for the task of model querying. In this paper we apply VMQL to the task of expressing model constraints which requires some extensions to VMQL.

The work reported in this paper starts with the observation that queries and constraints are just two sides of the same coin. Assume that $\alpha_M : ME \rightarrow bool$ is a property of model elements me of the domain ME of model elements, where models are just sets of model elements. Then, α_M may serve as a constraint on M, because asking for the (hopefully empty) set of model elements that violate the constraint expressed by α_M means to compute the set $\{me \in M | \neg\alpha_M(me)\}$.

But α_M may also serve as a query on M, because asking for the (hopefully non-empty) set of all model elements satisfying the query expressed by α_M just means to compute $\{me \in M | \alpha_M(me)\}$, i.e., the dual of the constraint in M: indeed the other side of the same coin in a very strict sense. So, we tried to find out whether the Visual Model Query Language (VMQL, see [17], [18]) would serve as a constraint language as well, and if so, how it would compare to OCL, the de facto standard for expressing constraints on UML-like models today.

The most important qualities of model constraint languages like OCL and VMQL are expressiveness and usability, that is, the set of expressions (of whatever kind) that may be expressed in the respective language, and the ease with which a language may be learned and applied, both actively and passively (i.e., reading and writing). Other qualities like genericity and practicality (including tool support) can not be discussed in this paper due to a lack of space (see [18] for an in-depth treatment). In the next section, we introduce VMQL, and then go on to compare it with OCL regarding expressiveness and usability, respectively.

II. The Visual Model Query Language

Application domain experts often lack a computer science or engineering background, yet are key modelers in model-based development projects. Participating in two very large scale industrial projects of this type, the author realized that current model querying facilities were not suitable for being used by application domain experts. As a consequence, the Visual Model Query Language (VMQL, see [17], [18]) was created as an easy-to-use visual model querying facility. Since it is based on concrete rather than abstract syntax, it is easy to transfer to another language, from the users's point of view.

Basically, a VMQL query is a model fragment in the modeling language used in the project, annotated by constraints. Fig. 1 shows some introductory examples that will be discussed shortly. Basically, all annotations are comments with the ≪vmql≫ stereotype that are translated into predicates evaluated during query execution. Table I presents those VMQL constraints used in this paper; see [18] for a more complete language definition, its implementation, and the technical details of query processing, and see [16], [15] for the underlying technology.

Now consider the examples of Fig. 1 in greater detail. Query 1a simply looks for a concrete class named "Foo" contained in a package called "Bar". Query 1b does the same for abstract classes, and Query 1c takes the transitive closure of the containment relationship into account. One might also interpret these queries as looking for a package named "Bar" that contains the class "Foo", or as looking for a containment relationship between the two: all VMQL query elements have the same weight per se.

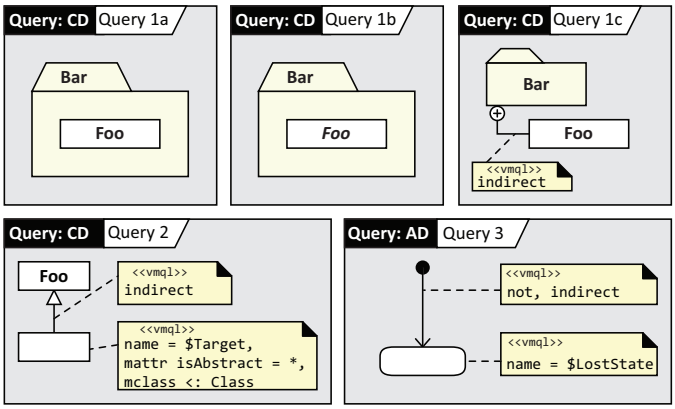


Fig. 1. Sample VMQL queries.

The user may single out individual elements by probing for some of their features, usually the name. This is demonstrated by Query 2, which looks for all subclasses of "Foo" and binds their name to the variable "Target", the intended result of the query. Observe that Query 2 looks for concrete as well as for abstract subclasses of "Foo". This is achieved by the meta-model constraint `mattr isAbstract = *` which asserts that the meta attribute "isAbstract" may have any value (signified by the asterisk). Also, Query 2 finds not just instances of the UML meta class Class that satisfy these conditions, it also finds instances of more specific meta classes, such as AssociationClass. This is achieved by the constraint `mclass <: Class`. The commas between the three parts mean conjunction.

VMQL may equally well be used for other diagram types, e.g., Activity, Sequence, or State Machine diagrams. For instance, Query 3 looks for all Actions that are not connected to the initial node.

III. Expressing constraints with VMQL

Let us now look at some examples of VMQL constraints (cf. Fig. 2). These examples are taken from an insurance example created in an industrial project (see [17] for more details). The first has been established as a means to prevent fraud. It is to be applied to process descriptions. Ensuring that process instances follow their descriptions is the responsibility of the workflow execution system. The rule is, that two independent assessments of a claim are made before any money is actually paid out. In order to implement this rule, the constraint asks for the "assess claim" action to occur twice in activity diagrams before the "award payment" occurs. In the VMQL model, thus, there are two "assess claim" actions that are connected by ActivityEdge arrows to the same "award payment" action. In order to make sure that the "assess claim" actions in the query match with the *same* "assess claim" action in the constrained model, they are constrained as being "distinct", enforcing that there are two independent claim assessements.

Observe that the rule is (and should be) unspecific about the ordering of these two assessments. Whether they are carried out in parallel, or in any order is irrelevant at the level of rules. Also, we really do not care just when the assessments are done: right at the start of the process, just before the payment, or at any other time. So, we only state that the assessments need to occur before the "award payment" action. This is implemented by the two "indirect" constraints: as long as there is a path from the assessments action to the payment action, the rule is satisfied. Obviously, this rule in itself is not enough, as nothing is said about the outcome of the assessment, but that is subject to another business rule.

Evaluating this VMQL constraint on some model will try

Constraint	Meaning
`match`	Restricts the name of the constrained model element by a wild card expression or regular expression, e.g., `match pa?tern*`
`distinct`	Enforces that a set of constrained model elements of the same type are pairwise distinct.
`once`	Enforces that a solution occurs only once in the set of all solutions.
`all`, `all as set`	Aggregates all solutions for the constrained elements into a single solution for the whole query, and builds up collection variables indicated by terms initiated by `$$`. Adding the option `as set` removes duplicates.
`steps`	Defines the length of a path between two connected model elements. Only one type of relationship may occur on the path. Applicable values are integers > 0 or `*` for arbitrary length > 0. Applicable to elements that are subclasses of Relationship, e.g., `steps = 3`, `steps < 3`, `steps = *`, `steps = 2; 3`, or `steps > 1, steps < 4`.
`indirect`	Defines a path of arbitrary length between two connected model elements; alias for `steps = *`.
`mclass`	Allows the constrained element to be of a different meta class than actually specified in the query. E. g., the annotation `mclass = Class; Component` allows the annotated element to match with classes as well as components, `mclass <: Classifier` will allow matching with all sub meta-classes of Classifier, and `mclass = *` matches with any meta-class.
`mattr`	Constrains the value of a meta attribute that has or has not a representation in the concrete syntax. If given a value expression, the meta attribute's value must conform to it. If given a variable (i. e. any expression starting with `$`), the value of the meta attribute is bound to that variable if possible. Variable may be bound several times, but only once for any meta-attribute, e. g., `mattr isRoot = true`, `mattr aggregationKind = composition; none`, `mattr isAbstract = *`, or `mattr name = $N`.
`name`	Alias for `mattr name`.
`precision`	Reduces the precision level used in the model matching to values below 1.
`strict`	Enforces that a query element must match exactly with a result element, i.e., the binding is bijective rather than injective.
`not`	Prevents that a result contains a match for the constrained model element of the query.
`optional`	Allows that a constrained model element of a query may or may not appear in the result.
`either`	Allows a set of alternatives for a constrained model element of a query to appear in the result. Applicable only to non-empty sets of model elements.
`in`, `sum`, `min`, `not_in`,...	The usual functions to be used on collection variables (similar to OCL).
`context`	Anchors the VMQL expression to some element, binding the element identifier as the result.
`assert`	Report model elements of the given type only if they do *not* satisfy the query (implies context).

TABLE I
Overview of VQML constraints (see [18] for semantics).

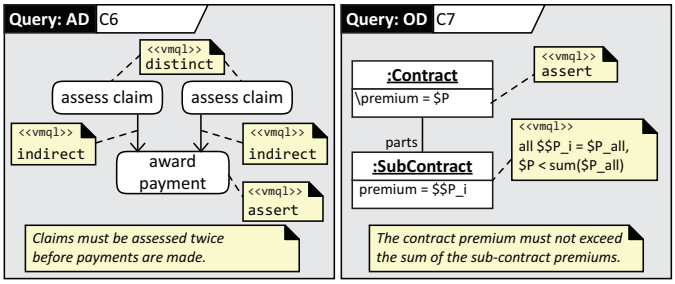


Fig. 2. Sample VMQL constraints.

to find model elements matching those in the VMQL model. Any match that satisfies this rule violates the business rule and should be reported. This is achieved by the "assert" constraint which collects as the result set all matches containing a match for the constrained elements but failing to satisfy the overall query.

The second constraint in Fig. 2 expresses another business rule, this time expressed on an object diagram. A contract may consist of several subcontracts, each of which may have a specific premium. However, the premium of the overall contract must not exceed the sum of the individual premiums. In order to implement this rule, the constraint "all" collects all bindings of the constrained elements that are compatible for each single binding of all the unconstrained elements. Note that variables preceded by `$$` rather than `$` are container variables, that is, they collect arbitrarily many values. Container variables may be aggregated by special purpose functions like "sum", "for all", "map" and so on. Thus, the function "sum" sums up the elements in `P_i` as `P_sum`. The "assert" constraint allows us, as before, to express the constraint without a negation, which would put additional cognitive load on the modeler. It also implies a "context" constraint which ensures that the result set is composed of matches of the constrained elements rather than other parts of the constraint.

Again, the rule is as unspecific as possible: obviously, some kind of iteration is necessary although the order is really absolutely irrelevant. More than that, we do not even require the collection of all subcontracts to be a set, say, let alone whether it is ordered. All of these details are not specified on the constraint, but would be in an OCL constraint analogous to this one. As we can see from these examples, expressing a constraint in VMQL means to specify a counter example (or violation) of the constraint and then look for it.

While translating OCL constraints to VMQL, it became obvious that there are two very different classes of constraints. The first class expresses a constraint for a meta classes with a visual concrete syntax representation, for instance, UseCase or State. For these constraints, expressing VMQL constraints is straightforward indeed. Fig. 2 and Fig. 3 show some examples. More than 90% of the OCL expressions in the sample were of this type.

The second class of OCL expressions (less than 10% in our sample) are constraints on meta classes that can not be displayed with reference to the concrete syntax. The reason for this might be that these constrained elements have no concrete

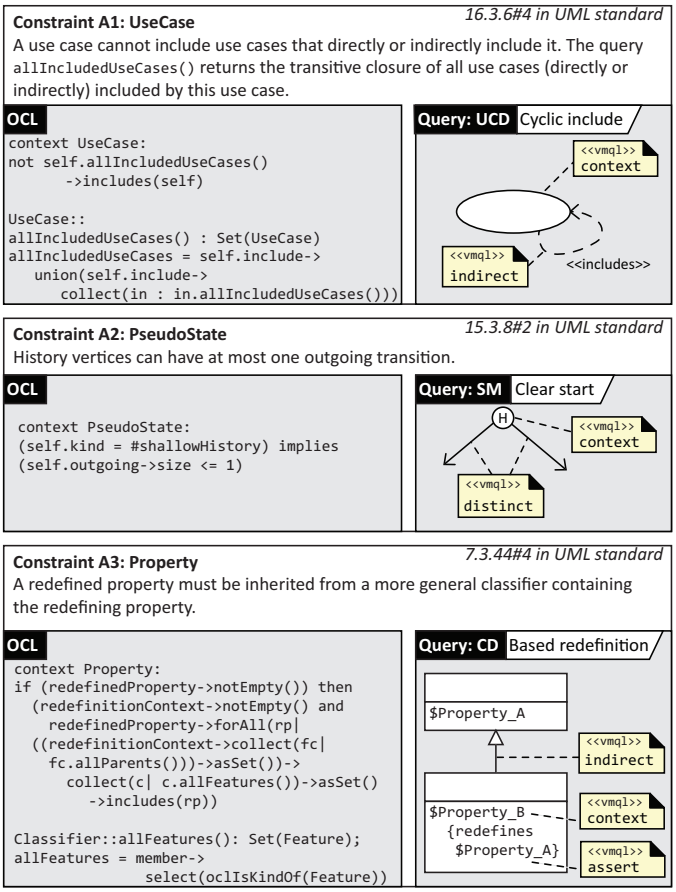


Fig. 3. Some constraints taken from the UML standard expressed in different ways: prose (top), OCL (left), and VMQL (right).

syntax representation at all (e.g., Classifier), they have only a textual representation (e.g., Feature), or an ambiguous representation (e.g., most subclasses of Action look like Action). Expressing these OCL constraints with VMQL is possible by expressing it as an annotated object diagram at the M2 meta level. but one cannot take advantage of a visual concrete syntax doing so. Instead, constraints have to be expressed as object diagrams showing instances of the meta model. Fig. 4 shows some examples.

Intuitively, these examples are not as convincing as those shown in Fig. 3, but should still be much easier to understand than their OCL counterparts. Unfortunately, our experimental data do not contain enough queries of this type to allow a statistically significant interpretation of the results.

IV. Constraint-extensions to VMQL

While examining the OCL constraints in [12], it became clear that VMQL as defined in [18] did need some modifications and extensions to be able to express constraints elegantly and concisely. See Table I for a complete list of the annotations VMQL features now. There are four extensions over the previous version of VMQL we found useful for expressing constraints elegantly.

First, we introduced the "all" constraint together with collection variables and some special functions like "sum", "map" and so on. This extension actually extends the expressiveness

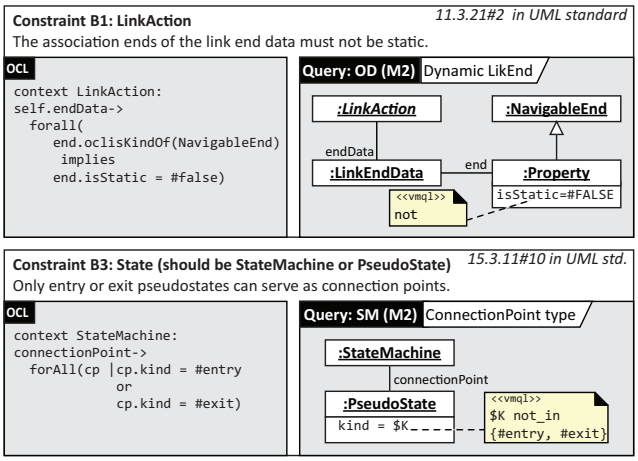


Fig. 4. More constraints from the UML standard (reference in top right corner). They are expressed by the prose descriptions (top), and the OCL expression (left) used in the UML standard, and as a VMQL diagram (right).

of VMQL, and is also very useful for querying purposes. Second, extending the constraint expressions in several ways, e.g. allowing value ranges for attributes (see e.g., [12, 15.3.11, constraint 10]), inequalities rather than just equations, and interval expressions for multiplicities does not extend the expressiveness, but makes constraints much more compact. Third, a special form of variable binding was introduced by the `context` annotation. Fourth, the handling of negation was improved by introducing the `assert` annotation which indicates that only results of the VMQL query shall be reported where the asserted element is not present. Observe, that the first two of these changes are true extensions while the second two are just syntactic sugar for existing features. However, the empirical results reported below indicate that the convenience of the last to modifications translates into a better comprehension of VMQL expressions.

V. Expressiveness of VMQL

In [18] we have compared the expressiveness of VMQL and OCL from a logical point of view and concluded that they are incomparable. In particular, [11] and [3] have shown that pure OCL 2.0 expressions represent the primitive recursive functions. Adding the capability of custom-defined functions (i.e. using the OCL keyword `def`) makes OCL Turing-complete. Since VMQL currently lacks a facility to define recursion or unbounded loops, it is certain that it is not Turing-complete. In fact, it is easy to find primitive recursive functions that may not be simulated in VMQL, so in the mathematical sense, VMQL is less expressive than OCL.

However, the theoretical limits of expressiveness of languages are rarely encountered in practical applications. For instance, many SQL implementations lag behind the current SQL standard quite considerably, but that does not stop them from being used quite successfully in practical applications. Similarly, many UML tools lag behind the current UML standard quite considerably. Again, this does not stop them from being used for many practical purposes.

That means that that for many practical applications, the theoretical expressiveness deficit of VMQL will not be an issue, as long as it is capable of expressing *the "right" constraints*, i.e., those that cover a large percentage of the most important constraints. Of course, this would depend on the application, maybe the modeling style, and quite possibly also the modeling level (M_0 to M_3, in OMG terminology). It is thus difficult to argue which constraints are the right ones to judge the expressiveness of a constraint language.

So, to evaluate the expressiveness of VMQL for modeling constraints from a practical perspective, we looked at one of the best known and most popular source of OCL constraints, the UML standard itself [12]. This document contains 473 constraints, but only 249 of these (52.6%) are expressed using OCL. Of these, 91 (approx. 40%) are very simple constraints that are at most one line of OCL, without negation, iteration, quantifiers, if-then-else, or function calls. These can be translated to VMQL trivially. The remaining 158 constraint are still too numerous to translate into VMQL manually. So we took a random sample by selecting every third OCL constraint, thus yielding 52 exemplars. Of these, 50 could be easily translated into VMQL; only two presented problems: one OCL constraint contained syntactical errors that made it impossible to guess the intended meaning, and one was so convoluted (and maybe also contained errors) that we could not understand the OCL expression in the first place.

Since our sample was drawn randomly, we may look at the process of translating random non-trivial OCL expressions from the UML standard to VMQL as a stochastic variable X. Since there is a fixed number of trials each of which has only two outcomes, and the individual samples are stochastically independent, X is binomial. The probability p of X indicates the likelihood of being able to translate any OCL constraint into VMQL. We will call this the translation coverage. Using the standard procedure, we can compute the confidence interval of X from a lower bound l to 1. For convenience, however, we have used the approximative method implemented in R's `binom.test` which gives more conservative estimates.

So, for a confidence level of $\alpha = 0.95$ we yield $l = 0.9199$ and a success probability $p = 0.9807$ for p-value smaller than 10^{-13}. That is to say that from our sample we may conclude that while we really expect a translation coverage in excess of 98%, we can safely (that is, with confidence of 95%) assume at least a 92% translation coverage. Observe, that this is a very conservative estimate, because the reason for failing to translate one OCL expression into VMQL comes from OCL/UML, rather than VMQL. Also keep in mind that this conversion rate only applies to those 60% of OCL-expressions that are non-trivial; for the remaining 40%, the translation coverage is 100%.[1] In total, thus, we *expect* a translation coverage around 99% and can *assert* a translation coverage of more than 95%.

So, assuming that the OCL-usage in the UML standard is representative for the OCL-usage in general, we conclude

[1]Sampling another 27 constraints from the UML in an interest-directed (i.e., a non-random) yielded another translation failure, which, again, could be attributed to UML/OCL rather than VMQL.

that almost all if not all of the OCL expressions occurring in practical modeling can be expressed as VMQL.

VI. Usability of VMQL

Usability is understood to mean the ease with which a language may be learned and applied actively and passively (i.e., in reading and writing). Obviously, this quality can only be evaluated empirically. Thus, we have designed and executed two controlled experiments comparing individual performance (accuracy and response times), personal preference, and cognitive load when reading and writing model constraints using VMQL and OCL, respectively. VisualOCL had to be excluded from this comparison, for several reasons. Firstly, creating valid VisualOCL is difficult given the available definitions and tools—there are three different Eclipse plugins to create VisualOCL expressions, but none of them is at the same time usable enough, covers a sufficient amount of VisualOCL, and produces adequate output. Also, VisualOCL expressions even for small examples requires very much space area so that we could not include them in our questionnaires, or in this presentation here.

A. Method

In previous experiments on model queries (cf. [18]), we could clearly show that subjects performed substantially better when using VMQL than when using OCL for model querying. While model querying and expressing model constraints are dual in a technical sense as we have argued above, they are substantially different from a user perspective. The languages are used in a different way, and different parts of the language are being used, as is underscored by the fact that we had to enhanced VMQL before being able to use it to cover typical UML constraints. So, we cannot simply assume that previous results on the usability of VMQL for *model querying* will hold for reading and writing *model constraints*, too. Therefore we created a new controlled experiment the results of which we report in this paper.

1) Standard demographic data (e.g., sex, age bucket, occupation, experience with UML and OCL, etc.) was collected in both experiments.
2) In the first task, subjects were asked to consider a model constraint in the respective language and select the correct prose description out of three options given. There were eight such constraints, all of which were expressed both in VMQL and OCL. We recorded scores for right/wrong answers and normalized scores to a scale from 0 to 10. Subjects also recorded the total time taken for all items using the same language
3) In the second task we asked subjects to read four prose descriptions of model constraints and asked them to translate it into VMQL and OCL, respectively. The same constraints are used in the writing task, than in the reading task, but the tasks are arranged intermittantly (i.e., first reading and writing of one language, than the other), and subjects can not look at previous pages of the questionnaire. Thus, we can exclude that writing is actually recall based on short term memory. We recorded a score between 0 and 4[2] for the accuracy subjects achieved in each expression, summed the score for all items using the same language, and normalized scores to a scale from 0 to 10. Subjects also recorded the total time taken for all items using the same language.
4) In the third task, we measured subjective assessment of the two languages using different instruments. In the first experiment, we asked for preference (fun, confidence) and cognitive load (effort, readability/writability). In the second experiment, we asked only for cognitive load (effort, difficulty) for reading/writing both languages. Observe that subjective assessments have been found to be very reliable indicators of objective load (cf. [6]).

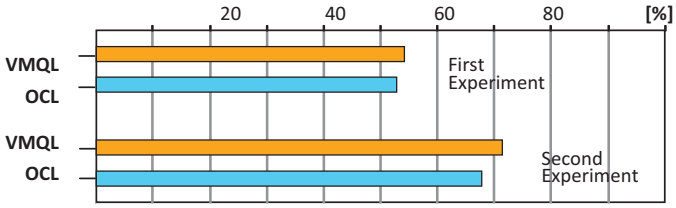
There were two equally sized experimental groups that, within each task, either presented the VMQL, or the OCL tasks first. Subjects were randomly assigned to those groups. The first experiment was run in May 2010 with 9 participants, including students, scientists, and practitioners. Detailed analysis of this experiment revealed several issues with the instructions and some of the questions, e.g., misunderstandings due to double negation. We improved the experimental setup accordingly. Also, the results from the preference task were very similar to previous results on this issue (cf. [18]), so we removed this set of questions and instead expanded the cognitive load questions for increased reliability. A follow-up experiment with these improvements was conducted in February/March 2011 with 7 participants all of which were Master's students at the time. All subjects participated voluntarily and anonymously without reward. No oral instructions were given: the questionnaire contained a brief instruction sheet as its first page. The experiments were run in Munich, Berlin, and Kongens Lyngby without the author being present. Results based only on one of the experiments is explicitly marked below.

B. Observations

The experimental data obtained were analyzed using R [13]. Analysis of the first experiment showed that participants performed better under the VMQL condition than under the OCL condition, but to a much lesser degree than we were led to expect from earlier experiments (cf. [18]). Detailed study of the errors subjects made revealed issues with the instructions and the questions that lead to an improved experimental setup, as discussed above. More importantly, we also found two notational shortcomings in VMQL as it was at the time being.

1) In most OCL constraints, the `context:` subexpression identifies the focus of attention, while no such element was present in the VMQL constraints. As a consequence, some participants interpreted e.g. Constraint 2 (see Fig. 3, top) as a constraint on includes-relationships rather than on use cases.

[2] 0:nothing recognizable at all; 1: some elements of an answer; 2: major errors; 3: minor errors; 4: exactly right answer



RIGHT ANSWERS [%]	VMQL	OCL	ADVANTAGE
EXPERIMENT 1	54.2	52.8	+2.6
EXPERIMENT 2	71.4	67.9	+5.3
IMPROVEMENT	+31.9	+28.6	

Fig. 5. Improvements in reading accuracy between the two experiments.

2) There are two different styles of expressing constraints: either, one might describe a wanted constellation and declare it right, or one might declare an unwanted constellation and declare it wrong. In the previous version of VMQL, only the former could be expressed conveniently. In order to express the latter style, the VMQL constraints had to be expressed indirectly (e.g., by using additional negations not directly visible in the prose description), which led to misinterpretations.

In order to address these issues we enhanced VMQL by the `context` and `assert` annotations (cf. the last two lines in Table I), updated the experimental conditions accordingly, upgraded the cognitive load measurement in the third task, and repeated the experiment. Fig. 5 shows the differences in results between the two experiments. Clearly, improving the instructions and questions had a marked effect on both conditions, VMQL and OCL. Also, the advantage of VMQL more than doubled in the second experiments, i.e., with the new `assert` and `context` annotations. Nevertheless, the difference between the results for VMQL and OCL are still much smaller than what we expected after the first usability experiments on VMQL (cf. [18]).

Looking more closely at the accuracy results from both experiments together (see Fig. 6, a), it becomes apparent that there is a wide variance for all indicators. While relatively little difference between accuracy for different languages is found, there obviously is substantial difference in the writing tasks. The exact data (see Table II) confirm this. Using a one-sided Wilcoxon test, we find no significant difference for reading accuracy ($p = 0.47$), but a highly significant difference in favor of VMQL for writing accuracy ($p = 0.0013$). Looking at the response times (see Fig. 6, b, and Table II, b), we find slightly larger, but still small differences in favor of reading VMQL, but against VMQL for writing (both, again, without statistical significance: $p = 0.57$ and $p = 0.82$, respectively). When relating the reading response time with the reading accuracy, we find a much large difference in the increase, but it is still not statistically significant ($p = 0.74$).

We now turn to subjective assessment of load and preference. We asked several questions capturing different aspects but they turned out very similar, i.e., scores for fun, confidence, effort, readability/writability, or difficulty showed very similar

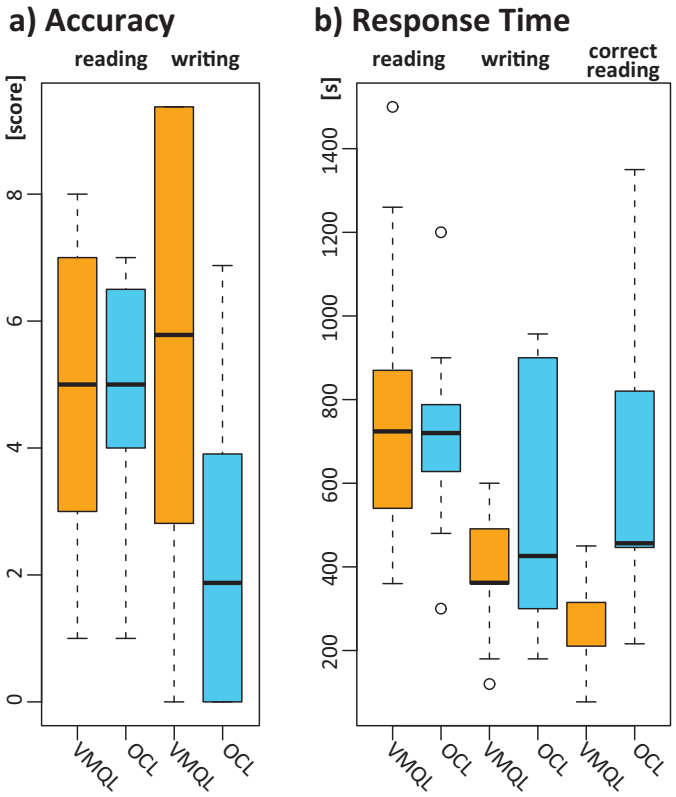


Fig. 6. Summary of experimental results wrt. subject performance measured by accuracy and response time (data from both experiments).

results. Since this task was considerably changed from the first to the second experiment, we ignore here those questions that have been asked in only one of the experiments and focus on the classic "effort" question that were asked in both experiments. We have normalized the different scales used in both experiments to a common range of $0 \ldots 10$ in Table II c). Remarkably, the results we obtained here are much more in favor of VMQL than OCL, both for reading and writing. However, a divergence between objective performance (i.e., accuracy) and preference/cognitive load seems to be a common phenomenon (cf. [6, p. 521]).

C. Interpretation

The wide individual variance we observed throughout our experiments is very likely a consequence of the small sample size, and the differences between the two experiments. However, there also seem to be unrelated factors like personal taste and drawing skills. For instance, in follow up interviews or free form comments, several subjects commented on the usage of pen and paper as inappropriate ("*can't write OCL without syntax completion*" or "*creating a drawing is always more difficult because it takes up more space*").

We attribute the generally better performance in the second experiment to the improvements we made to VMQL. There is a notable advantage of VMQL over OCL in the writing task. Since the constraints we have asked subjects to write have been shown before in the reading task, this is really a recall task. Better performance for VMQL through its visual

TABLE II
MEASUREMENT DETAILS OF THE BOX PLOTS PRESENTED IN FIG. 6: ACCURACY IN TERMS OF CORRECT ANSWERS FOR READING WRITING (TOP), RESPONSE TIME FOR READING, WRITING, AND READING RELATIVE TO THE NUMBER OF CORRECT ANSWERS (MIDDLE), AND COGNITIVE LOAD ASSESSMENTS (BOTTOM; THESE MEASUREMENTS WERE TAKEN ONLY IN THE SECOND EXPERIMENT).

A) ACCURACY [SCORE: 1..8]

CORRECT ANSWERS	OCL		VMQL		ADVANTAGE
	μ_o	σ	μ_v	σ	$\mu_v - \mu_o$
READING	4.71	1.83	4.76	2.36	+1.3%
WRITING	3.56	3.27	9.06	5.19	+154.5%

B) RESPONSE TIME [S]

TIME PER ITEM	OCL		VMQL		ADVANTAGE
	μ_o	σ	μ_v	σ	$\mu_v - \mu_o$
READING	725.14	201.88	776.62	317.74	+7.1%
WRITING	452.25	320.05	408.25	57.20	-9.7%
CORRECT READING	168.19	69.77	217.90	137.58	+29.6%

C) COGNITIVE LOAD [SCORE: 1..10]

SUBJECTIVE EFFORT	OCL		VMQL		BENEFIT
	μ_o	σ	μ_v	σ	$\mu_v - \mu_o$
READ	2.59	0.91	2.21	0.97	-17.2%
WRITE	3.19	0.96	2.23	0.96	-43.0%

notation, however, cannot be attributed to visual short term memory since the experimental setup placed either the OCL reading task or the OCL writing task between them. So, there was an average of more than 11 minutes between reading the VMQL constraint and recalling it.

This leaves us with two phenomena to be explained: (1) the unexpectedly small margin in reading performance, and (2) the unexpectedly large margin in writing performance.

Concerning the smaller margin in reading tasks (1) we have two explanations. Firstly, we really did expect that the performance on the constraint-related tasks tested in this experiment would be worse than for the query-related tasks tested in previous experiments. After all, OCL was designed specifically for expressing constraints, while VMQL was designed for expressing queries. So, the nature of the tasks in this experiment ought to favor OCL, as it would have favored VMQL in our earlier experiments.

Secondly, in comparing the relative performance, we have to take into the account prior knowledge of the participants. On average, participants rated themselves to have a higher competence in OCL and Logics than in VMQL (0.8 and 2.4 compared to 0.2 on a 5-point Likert scale from 0 "don't know language" to 4 "expert knowledge"). This explanation was further confirmed by comments made by study participants during follow-up interviews, such as: "*I really felt I didn't even have the knowledge for an educated guess [in the VMQL tasks]*".

Together, these factors certainly would explain the unexpectedly small margin in the comprehension task, but not the unexpectedly good performance in production task (2). Our follow-up interviews hinted at a possible explanation: "*the graphics [VMQL] are really good for getting an overview, but most constraints were so small this was not essential*", as one participant put it. In fact, the tasks presented here were much simpler and quicker (about half the duration) to complete than those in previous experiments, which might be a bias towards OCL. Consequently, we will have to study the influence of expression complexity to subject performance in the future.

Having said all that, even as they are the experiments confirmed again that VMQL is superior in terms of motivating people to actually write or read constraints, which is certainly a good basis for further investigation and improvement.

D. Threats to validity

The number of participants is sufficient to derive valid conclusions, but the number and selection of different constraints used in the tasks might bias the outcome. We have tried to prove a representative set of constraints covering the whole range of different styles and applications, but that is of course a subjective judgment. However, the results are plausible, and completely in line with the subjective assessment by the study participants who unanimously preferred VMQL over OCL in terms of fun, a judgment that would be very hard to bias by the choice of constraints.

Another bias might come from the formulation of the prose descriptions. In order to avoid such bias, the prose descriptions were drawn verbatim from the UML standard for those six out of eight constraints we took from the UML. So if there is a bias, it is present in the field, not in our experiments.

Since we use the same queries for both languages, the order of the languages participants work on may create a learning effect and thus bias the results. In order to avoid this, half of the participants worked on VMQL first, and the other half worked on OCL first. If there is a learning effect it would equally affect the results of both languages, thus not skew the relative scores.

VII. RELATED WORK

While there is an abundance of work on consistency checking of models using formal methods (see [10]), much less attention has been paid to actually specifying model constraints and queries. There are three major approaches to visually specifying properties for UML models.

Constraint Diagrams have been proposed by [9]. In conjunction with a Z-like framework, they are a powerful visual language for expressing constraints, although they seem to be restriced to UML class diagrams or similare structures. Also, there seems to be no evaluation of their practical value in a realistic setting (cf., however, [7], [8]).

VisualOCL have been proposed by [2] not as a language in its own right but as a visualization on top of OCL (cf. [1], [4]). This approach has the advantage that the "back end" (that is, OCL) can be used as a black box, including formal semantics, tools and so on, and VisualOCL can be applied to any existing body of OCL constraints. The translation between OCL and VisualOCL is rigorously defined, and so, together with formal semantics for OCL, VisualOCL is formally defined. VisualOCL covers almost the entire OCL, thus warranting almost the same expressiveness as OCL (though for a deprecated

version only). There is also tool support for VisualOCL, albeit only academic prototypes. [5] compares Constraint Diagrams and VisualOCL and suggests that VisualOCL diagrams are easier to understand than Constraint Diagrams for simple OCL expressions. There is no evaluation of the usability of VisualOCL, in particular not in comparison with OCL. Recall, however, that the prohibitive space consumption of VisualOCL prevented it being included in this study.

Query Models have been proposed by Stein et al. (see [14] and subsequent works). This approach is certainly the one closest to VMQL. Like Constraint Diagrams and VMQL, Query Models use a variant of the host language (UML) to express additional constraints. Like VisualOCL, Query Models are supposed to be translated into OCL, although this translation is undefined, and it is not at all clear how this could be done (there is no generic definition and only a handful of examples). There is no tool support for executing Query Models (read: checking constraints expressed as Query Models), and no evaluation of their usability or expressiveness.

VIII. Discussion

The Visual Model Query Language (VMQL) has first been proposed in [17] and elaborated in [18]. Originally intended as a vehicle for end users to issue ad-hoc queries to (large) models, it soon became apparent, that it might also be used for expressing constraints, for queries and constraints are just two different usages of predicates on model elements. While VMQL is less expressive as OCL (e.g., functional abstraction), this seems to be a theoretical rather than a practical limitation, as we were able to translate almost all OCL constraints from a significant sample of all the OCL constraints in the UML standard. Also, VMQL is highly generic (cf. [18]) and may be applied to many other visual languages, including future DSLs or legacy languages beyond MOF-based languages, and thus outside the reach of OCL. Additionally, VMQL surpasses OCL in terms of usability, as is indicated by the findings of controlled experiments we conducted as part of this work, and previous work. The reliability of our findings is somewhat limited by the small number of experiment participants and the small set of tasks they worked on. On the other hand, no other competing approach (most notably, QueryModels and VisualOCL) has ever even tried to demonstrate its usability.

The process of applying VMQL to constraints rather than queries, its original domain, exhibited a number of shortcomings and deficits of VMQL as it was at the time. Most of these were quite easy to fix and resulted in an improved and extended VMQL as presented in this paper. The question remains whether it is possible to increase the expressiveness of VMQL to completely match that of OCL (i.e., not just practically, but also theoretically), and how this may be achieved. Simply embedding OCL into VMQL would of course sidestep this issue altogether, but this option comes at the price of loosing the usability advantage VMQL has over OCL. Thus, adding a capability for functional abstraction and recursion might be a better option, but it is not clear how this could be achieved without deteriorating understandability. We hope to address this issue with our future work.

Other ongoing work focuses on strengthening the tool support, extending the empirical results (including industrial case studies, and more experimental evidence), and understanding better just why VMQL affords superior usability than OCL in the first place. The answer to this question will allow us to further improve VMQL, but it might also help to improve OCL, or provide better OCL tool support.

References

[1] P. Bottoni, M. Koch, F. Parisi-Presicce, and G. Taentzer, "Consistency Checking and Visualization of OCL Constraints," in *Proc. 3^{rd} Intl. Conf. Unified Modeling Language (≪UML≫'00)*, ser. LNCS, B. Selic, S. Kent, and A. Evans, Eds., no. 1939. Springer Verlag, 2000, pp. 294–308.

[2] ——, "A Visualisation of OCL using Collaborations," in *Proc. 4^{th} Intl. Conf. Unified Modeling Language (≪UML≫'01)*, ser. LNCS, M. Gogolla and C. Kobryn, Eds., no. 2185. Springer Verlag, 2001, pp. 257–271.

[3] M. Cengarle and A. Knapp, "OCL 1.4/5 vs. 2.0 expressions formal semantics and expressiveness," *Software and Systems Modeling*, vol. 3, no. 1, pp. 9–30, 2004.

[4] K. Ehrig and J. Winkelmann, "Model Transformation from VisualOCL to OCL Using Graph Transformation," *Electron. Notes Theor. Comput. Sci.*, vol. 152, pp. 23–37, 2006.

[5] A. Fish, J. Howse, G. Taentzer, and J. Winkelmann, "Two Visualizations of OCL: A Comparison," Univ. of Brighton, Tech. Rep. VMG.05.1, 2005.

[6] D. Gopher and R. Braune, "On the psychophysics of workload: Why bother with subjective measures?" *Human Factors*, vol. 26, no. 5, pp. 519–532, 1984.

[7] J. Howse and S. Schuman, "Precise Visual Modeling: A Case-Study," *Software and Systems Modeling*, vol. 4, no. 3, pp. 310–325, 2005.

[8] J. Howse, S. Schuman, G. Stapleton, and I. Oliver, "Diagrammatic Formal Specification of a Configuration Control Platform," *Electronic Notes in Theoretical Computer Science*, vol. 259, pp. 87–104, 2009.

[9] S. Kent, "Constraint Diagrams: Visualizing Invariants in Object-Oriented Models," in *Proc. Intl. Conf. Object-Oriented Programming, Systems, and Languages (OOPSLA97)*. ACM Press, 1997, pp. 327–341.

[10] F. J. Lucas, F. Molina, and A. Toval, "A systematic review of UML model consistency management," *Inf. Softw. Technol.*, vol. 51, no. 12, pp. 1631–1645, December 2009.

[11] L. Mandel and M. V. Cengarle, "On the Expressive Power of OCL," in *Intl. Conf. Formal Methods (FM'99)*, ser. LNCS, vol. 1708. Springer Verlag, 1999.

[12] OMG, "OMG Unified Modeling Language (OMG UML), Superstructure, V2.2 (formal/2009-02-02)," Object Management Group, Tech. Rep., Feb. 2009.

[13] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2011. [Online]. Available: http://www.R-project.org

[14] D. Stein, S. Hanenberg, and R. Unland, "Query Models," in *Proc. 7^{th} Intl. Conf. Unified Modeling Language (≪UML≫'04)*, ser. LNCS, T. Baar, A. Strohmeier, A. Moreira, and S. J. Mellor, Eds., no. 3273. Springer Verlag, 2004, pp. 98–112.

[15] H. Störrle, "A PROLOG-based Approach to Representing and Querying UML Models," in *Intl. Ws. Visual Languages and Logic (VLL'07)*, ser. CEUR-WS, P. Cox, A. Fish, and J. Howse, Eds., vol. 274. CEUR, 2007, pp. 71–84.

[16] ——, "A logical model query interface," in *Intl. Ws. Visual Languages and Logic (VLL'09)*, P. Cox, A. Fish, and J. Howse, Eds., vol. 510. CEUR, 2009, pp. 18–36.

[17] ——, "VMQL: A Generic Visual Model Query Language," in *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'09)*, M. Erwig, R. DeLine, and M. Minas, Eds. IEEE Computer Society, 2009, pp. 199–206.

[18] ——, "VMQL: A Visual Language for Ad-Hoc Model Querying," *J. Visual Languages and Computing*, vol. 22, no. 1, Feb. 2011.