Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Beyond OCL:
Model Queries for Mere Mortals

Università di Trento, 23.5.2014

Problem

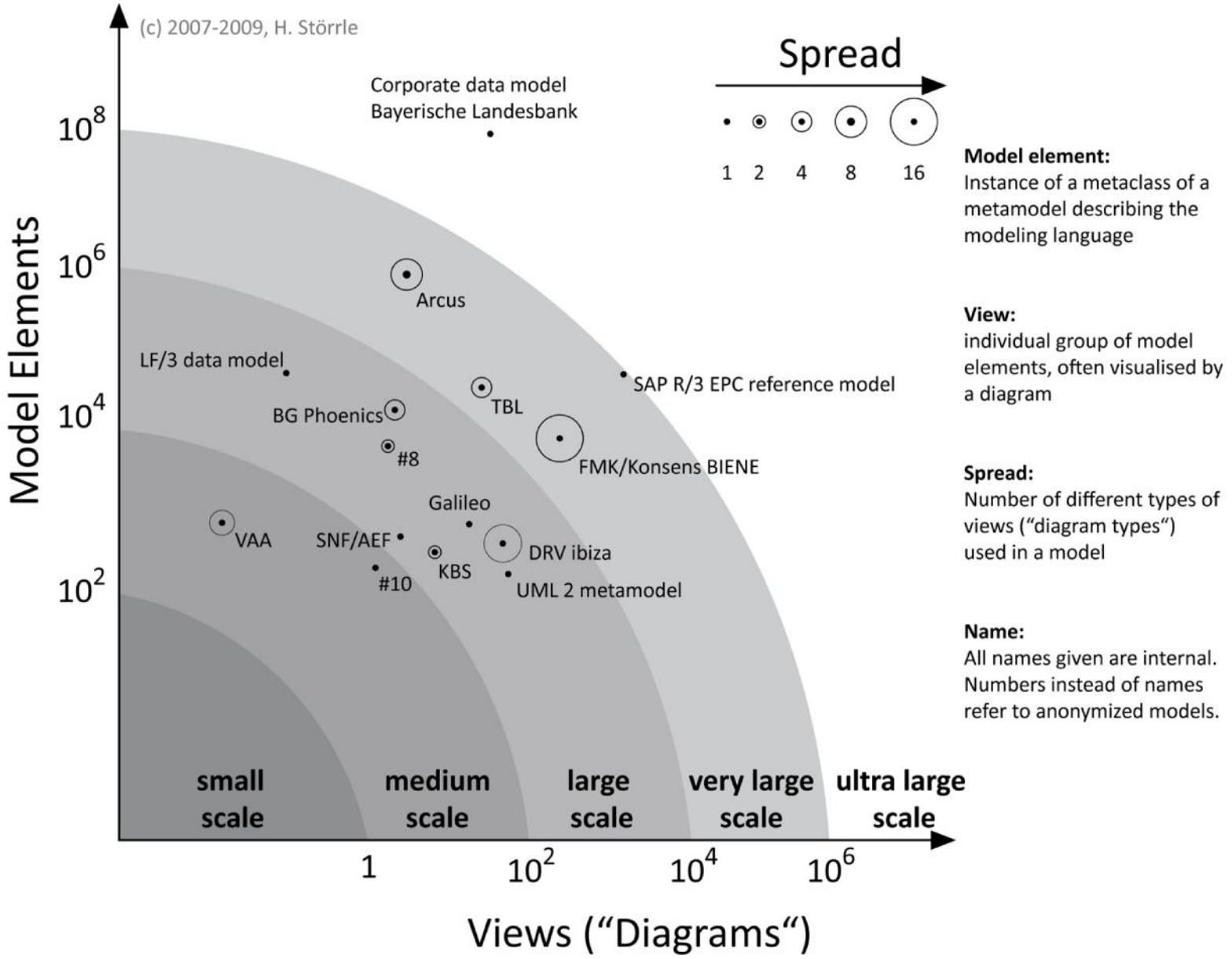
- **In Model Based Development or Business Process Reengineering, large systems of analysis level models are created in a variety of languages.**
  - These models are often created by (or in collaboration with) domain experts, who usually do not have a CS background.

- **The models are a significant asset though often a "buried treasure" – finding information in large model bases may not be not easy.**
  - As one client once put it: *"The process of modeling really made us understand our processes, but now our repository is kind of a black hole, really."*

- **There are a number of practical ways for querying models.**
  1. Look through the whole model (or the diagrams presenting it) manually.
  2. Select one from a number of predefined queries (or visualizations).
  3. Do a full-text search, possibly applying one of a finite selection of filters, or regular expressions.
  4. Program the query using a query API of the modeling tool at hand.
  5. Use a model query language to formulate a query.

- **In a UML context, OCL is the "natural" model query language.**

Models can become large



Model element:
Instance of a metaclass of a metamodel describing the modeling language

View:
individual group of model elements, often visualised by a diagram

Spread:
Number of different types of views ("diagram types") used in a model

Name:
All names given are internal. Numbers instead of names refer to anonymized models.

Part 1

Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Beyond OCL:
Model Queries with VMQL

Università di Trento, 23.5.2014

VMQL – Basic Idea

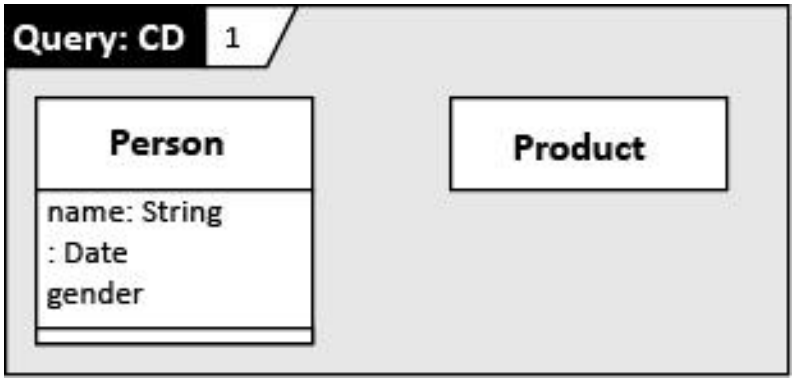
- **If people can model, they understand the modeling language.**
  - Using this vocabulary and concrete syntax surely is acceptable to them.
  - Some slight additions may be acceptable.

- **This will work for querying languages for any visual language.**
  - With suitable implementation, this VMQL is a concrete query facility for all languages, including future DSLs.

- **A similar idea has been proposed but not implemented [1,2].**
- **For process models this problem has been studied more intensely in recent years. [3,4]**

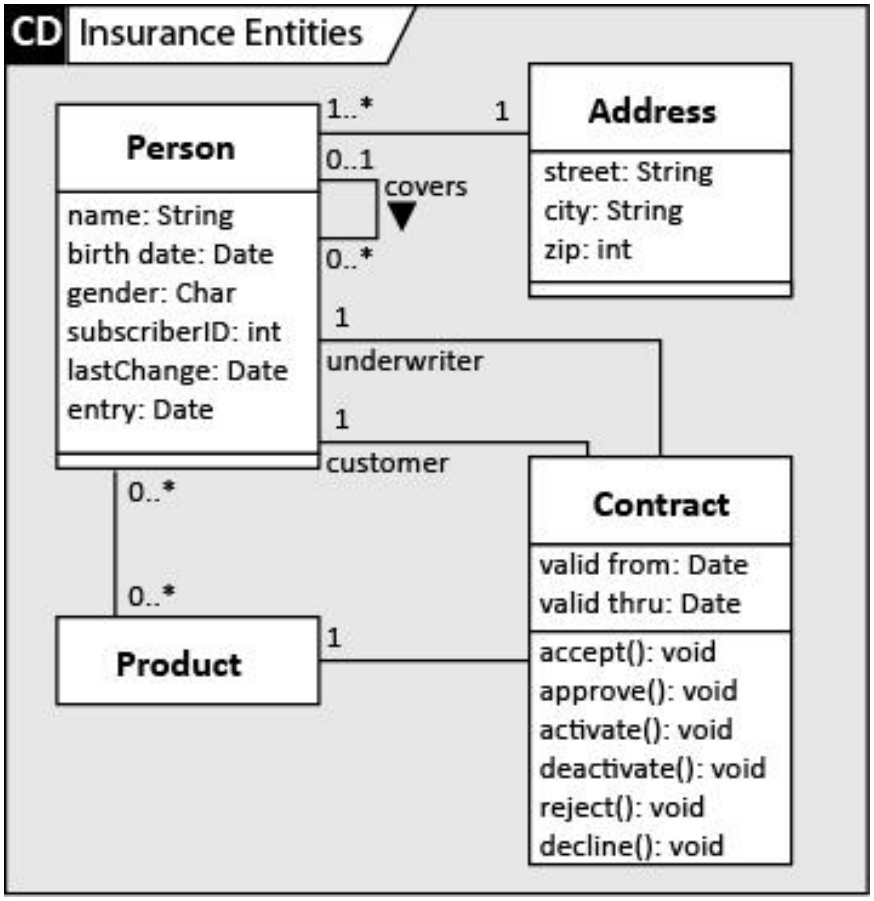
1) Stein, D., Hanenberg, S., Unland, R.: Query models. Proc. UML. LNCS 3273 (98–112) Springer, 2004
2) Stein, D., Hanenberg, S., Unland, R.: A Graphical Notation to Specify Model Queries for MDA Transformations on UML Models. Proc. MDAFA'03. LNCS 3599 (77–92) Springer 2005
3) Beeri, C., Eyal, A., Kamenkovich, S., Milo, T.: Querying Business Processes. Proc. 32nd VLDB (343–354) 2006
4) Awad, A., Decker, G., Weske, M.: Efficient Compliance Checking Using BPMN-Q and Temporal Logic. Proc. BPM 2008. LNCS 5240 (326–341) Springer 2008

Example 1: Ground Query

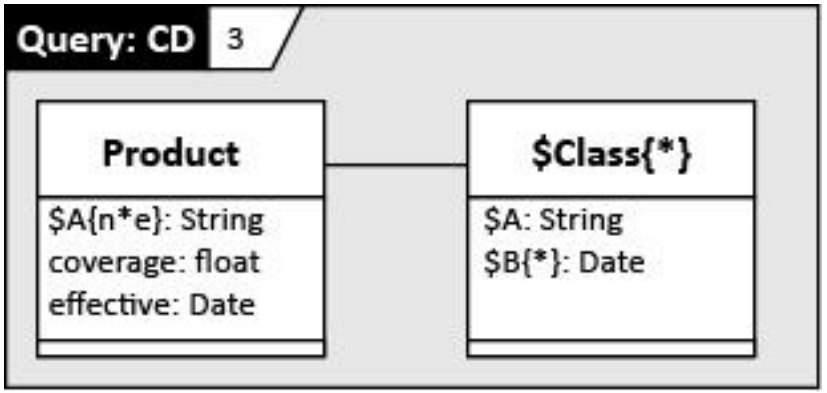


Find a class named ‘Product’,
and a class named ‘Person’ with
- a string attribute called ‘name’
- an attribute with type date
- an attribute named gender

Binding		
Product	↔	Product
Person	↔	Person
Person.name	↔	Person.name
Person.<#1>	↔	Person.birth date
Person.gender	↔	Person.gender

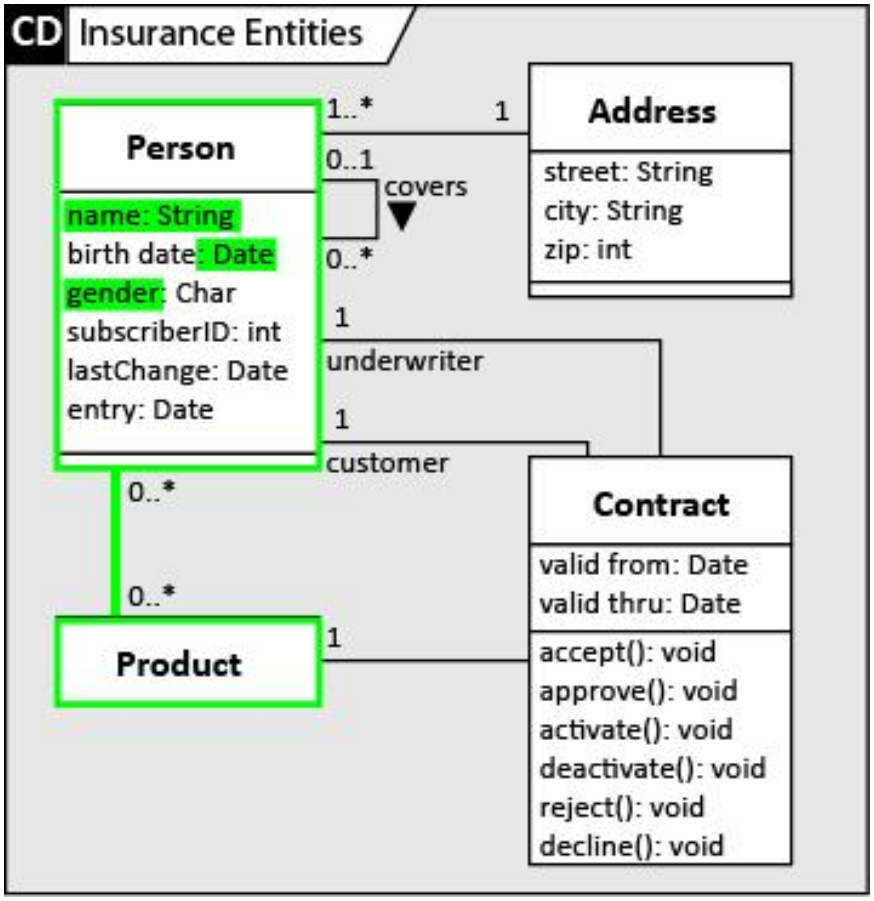


Example 2: Variables and Patterns



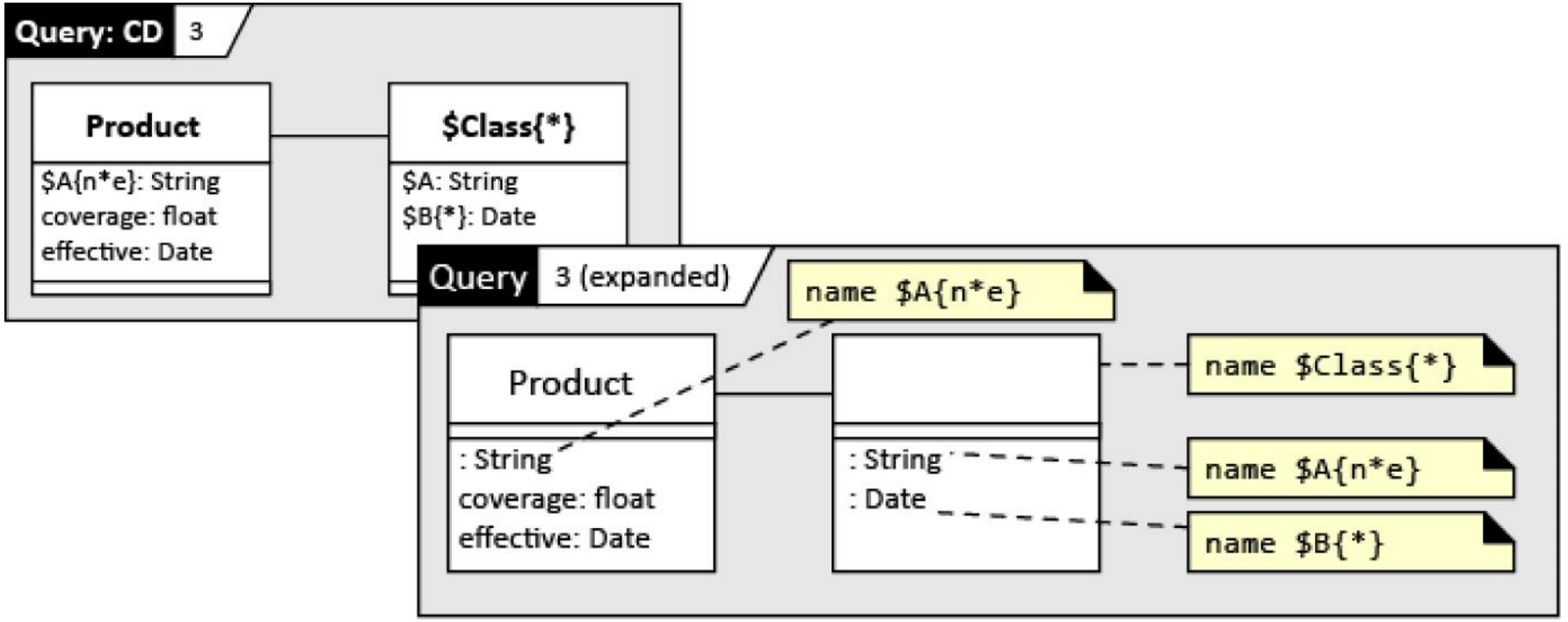
Find two associated classes with some attributes as shown above. One of the classes is called ‘Product’, report the other one’s name in variable ‘Class’.

Binding	Class	A	B
1:	“Person”	“name”	“birth date”
2:	“Person”	“name”	“last change”
3:	“Person”	“name”	“entry”



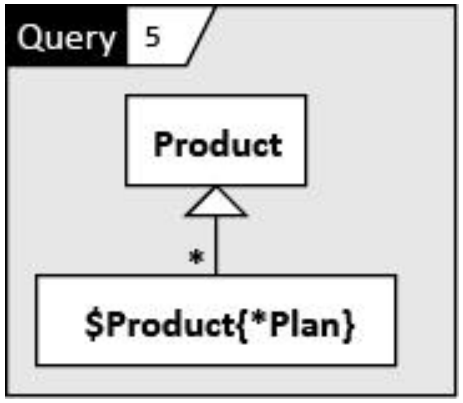
Syntactic Sugar

1. **Technically, using variables and patterns in the query model has defined a set of constraints.**
   1. We have provided syntactic sugar to hide this from the user.
2. **Constraints are captured as comments annotating arbitrary model elements.**



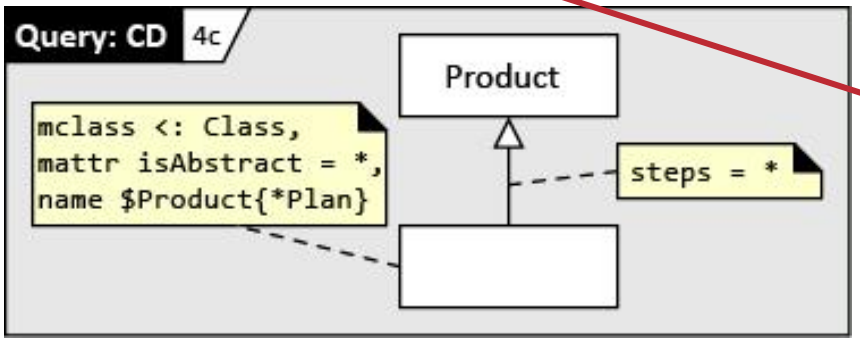
Example 3: Paths, Meta Model Access

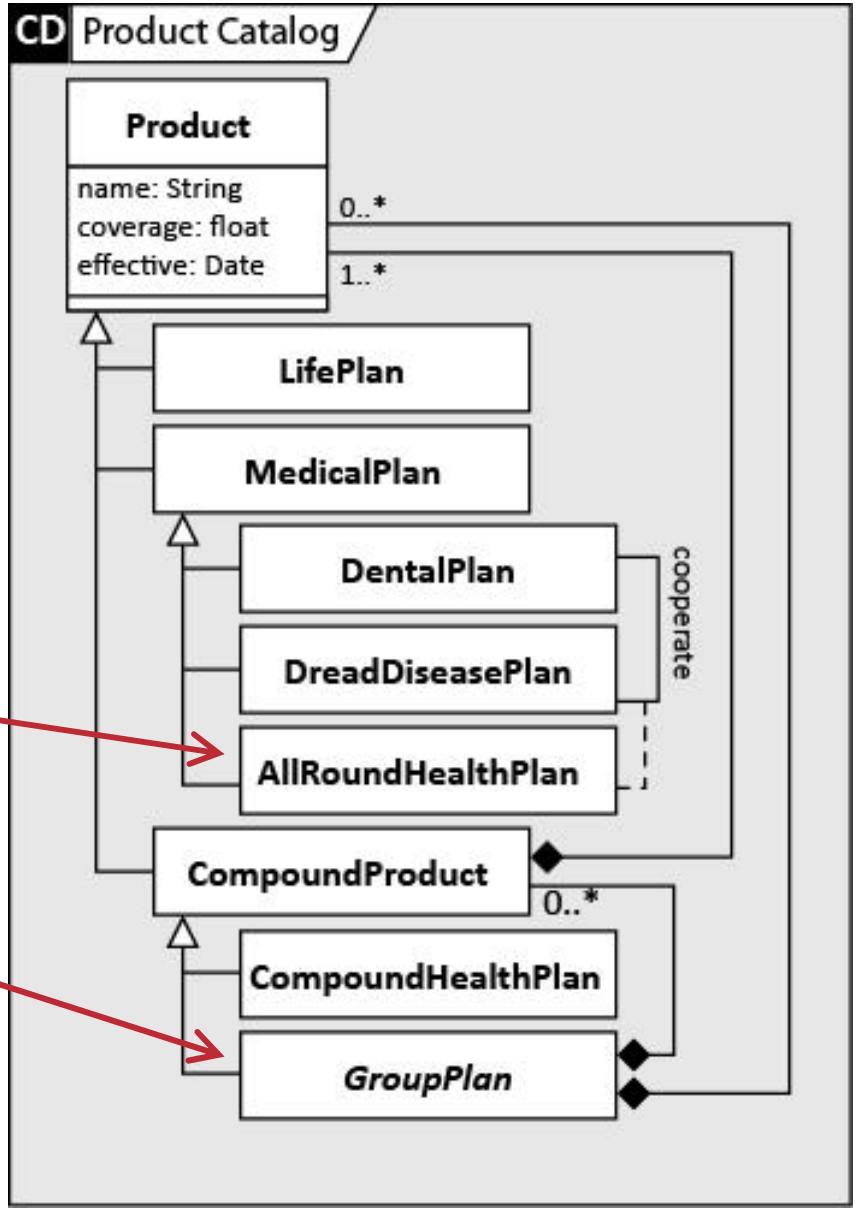
Intention: Find all plans!



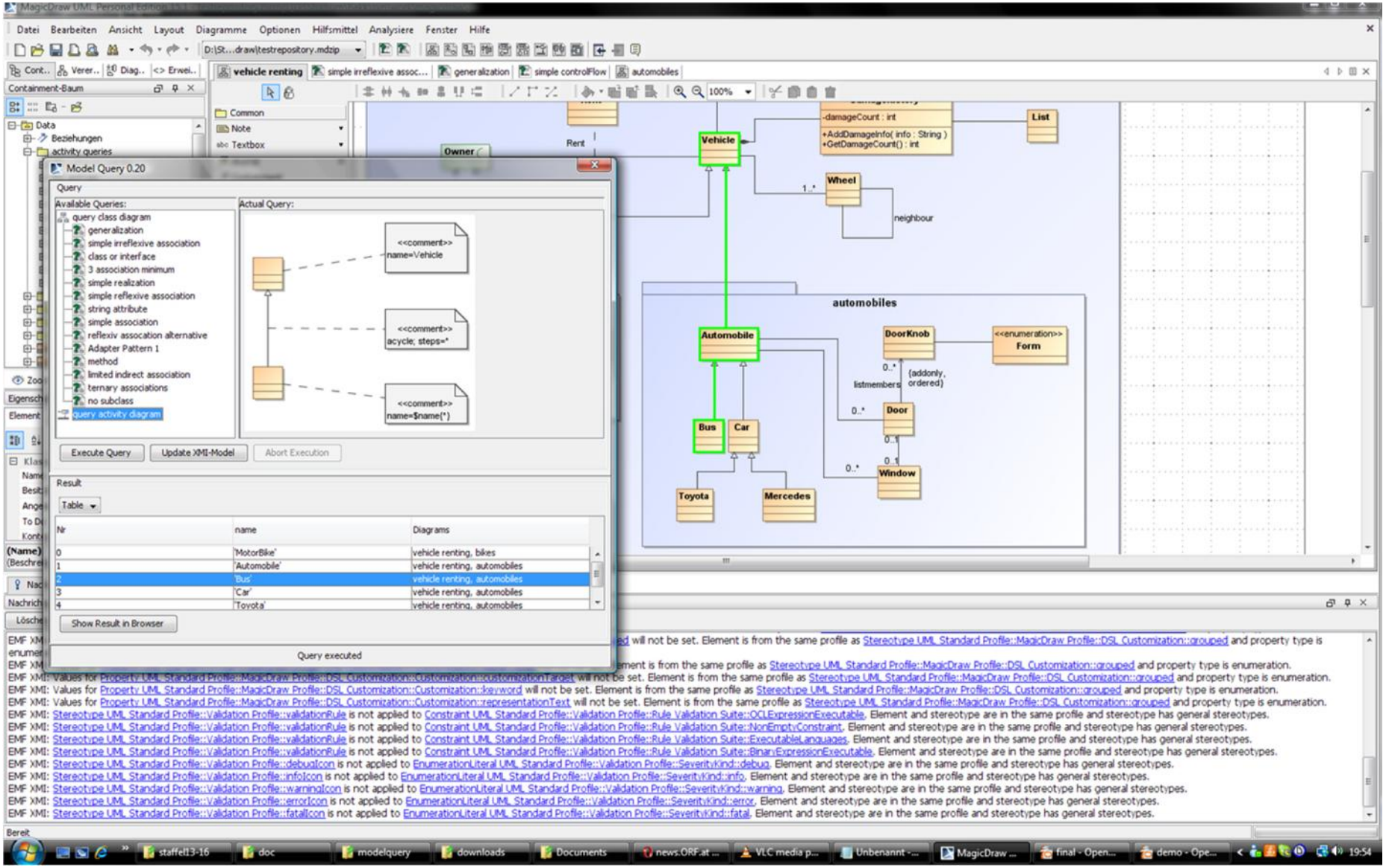
Find all subclasses of 'Product' whose name ends in 'Plan'.

…but doesn't find 'AllRoundHealthPlan' and 'GroupPlan'

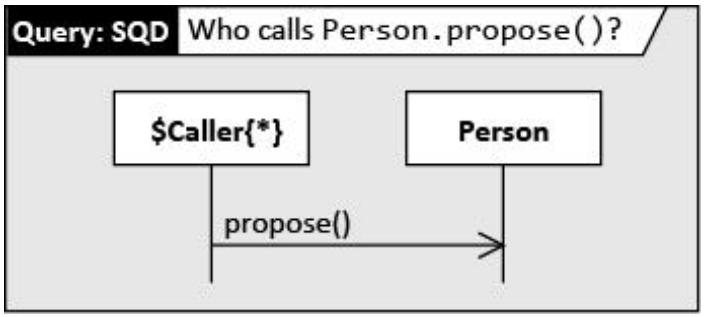


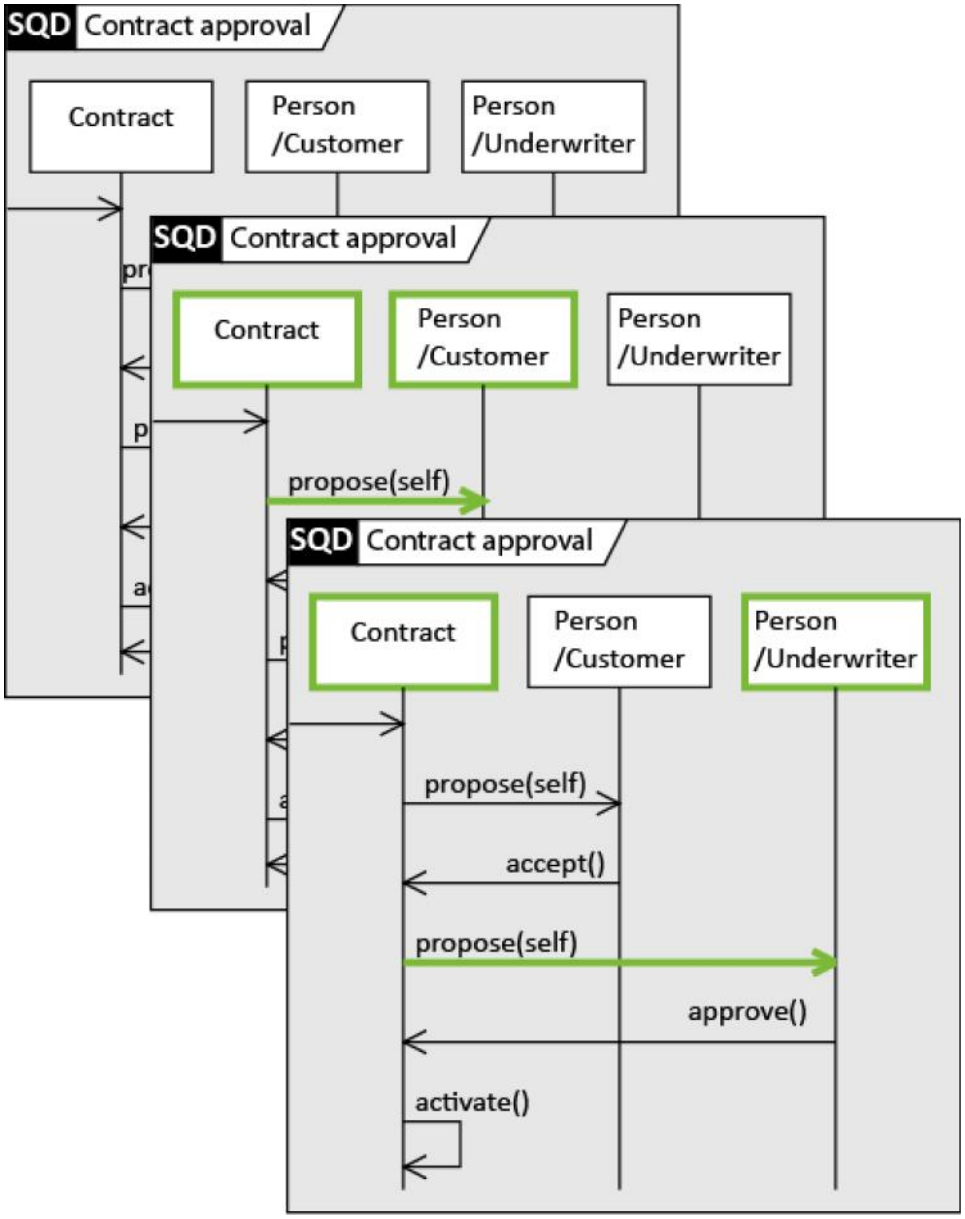


Implementation of VMQL in ModelQuery

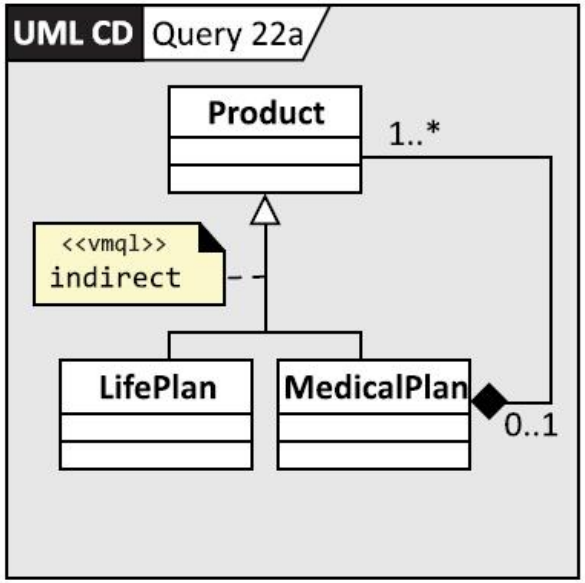


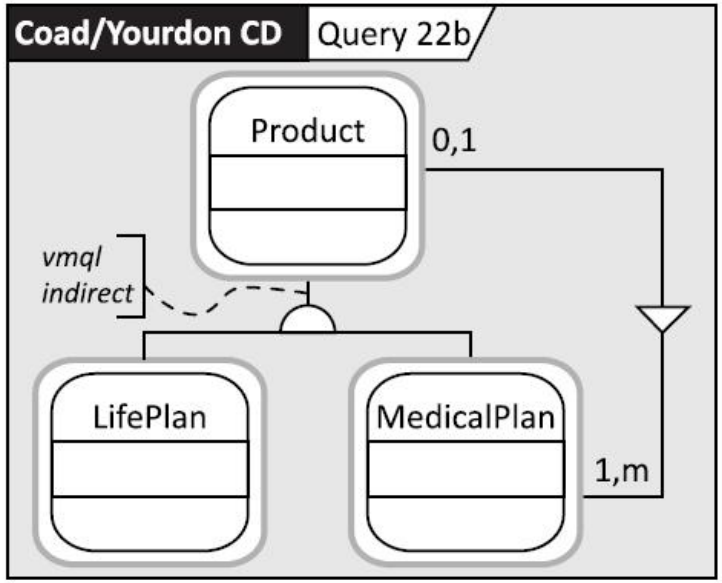
Actually, VMQL works for any UML Notation

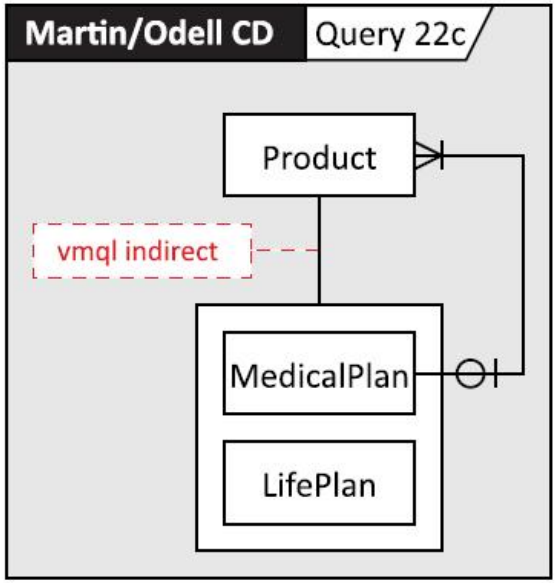


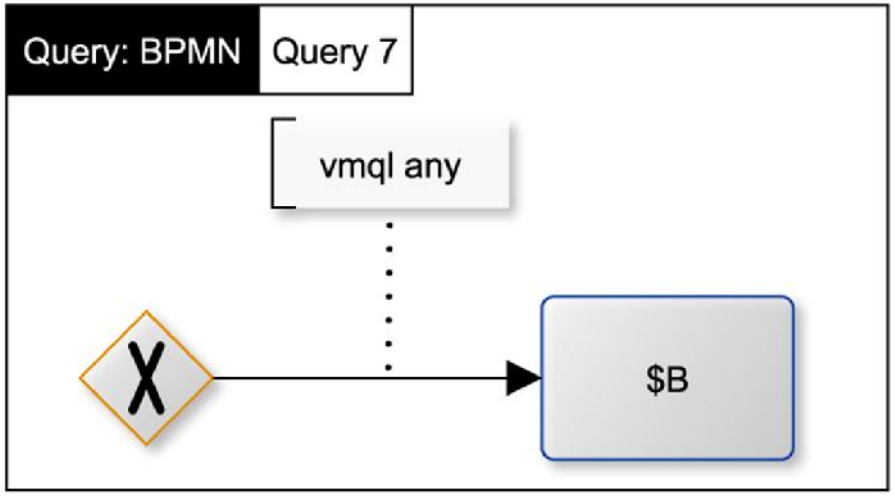


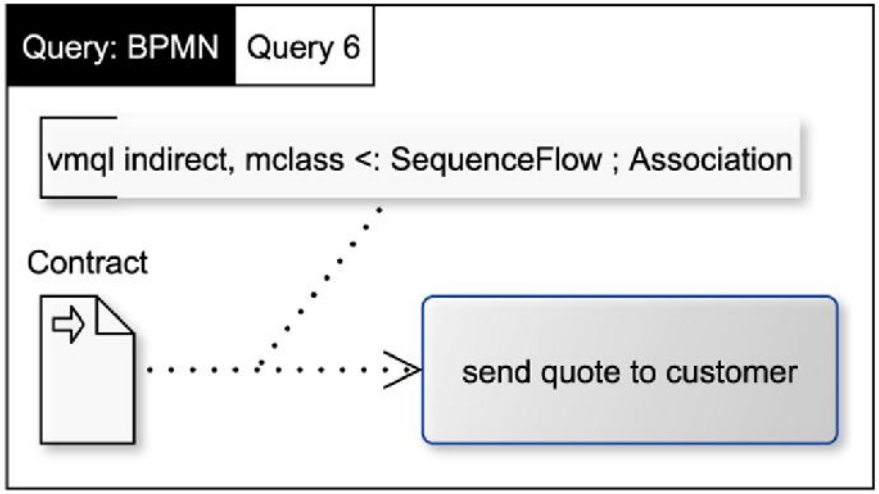
In fact, any Modelling Language











Come to think of it: not just on Queries

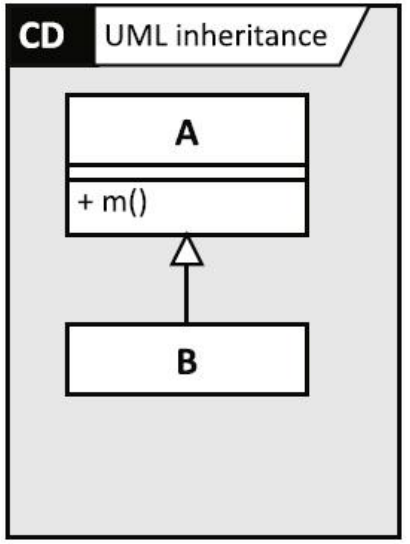
- **Apart answering simple questions about a model, a query may also serve other purposes:**
  - A query may also express a property (including temporal properties) we may want to check, metrics we may want to compute, or (design) patterns we might want to detect.
  - With suitable base predicates, queries may be used as version control primitives ("find the difference between…").
  - A (negated) query is a model constraint or consistency condition.
  - A query is the "left hand side" of a model transformation, with suitable base functions, it may also express the "right hand side" (→ VMTL).
  - VMTL should also work as a language for representing (and executing) diagrammatic inferences.

- **These additional use cases require only few and minor conceptual additions.**
  - The implementation is lagging behind, though.

Limitations of VMQL

- **VMQL only works for modeling languages with a meta model and comment-boxes.**
  - Notational elements without concrete syntax representation, or non-syntactic features of a language (e.g., layout) cannot be queried.
  - Result presentation relies on the preexistence of diagrammatic presentations of the model.

- **VMQL is purely syntactic: semantic relationships cannot be expressed directly.**
  - Querying for the methods of B does not yield m().
  - Unless you put in some semantic muscle, i.e., predicates that capture some wanted property.



Part 2

Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Beyond OCL:
Model Queries for Mere Mortals

Università di Trento, 23.5.2014

Experimental Setup

- **Subjects**
  - Mostly students (all levels), some practitioners
- **Tasks**
  1) Given a query in language X, select the correct out of three given natural language descriptions;
  2) Given a query in English, select matches out of 16 queries in language X;
  3) Given a query in English, produce a query in language X.
- **Controlled Variables**
  - Main: query language
  - Noise: task type, task, population
- **Measurements**
  - Score (Writing: checklist assessment)
  - Time (self timed)
  - Cognitive Load (several different for corroboration)
  - Preference (several different for corroboration)

Post Interview Remarks

- **After the experiments, some participants were willing to share their feelings.**
- **Some also added written comments on the questionnaires during the experiment.**
- **These perceptions are in line with the objective findings, thus corroborate the overall outcome.**

Quotes from participants:

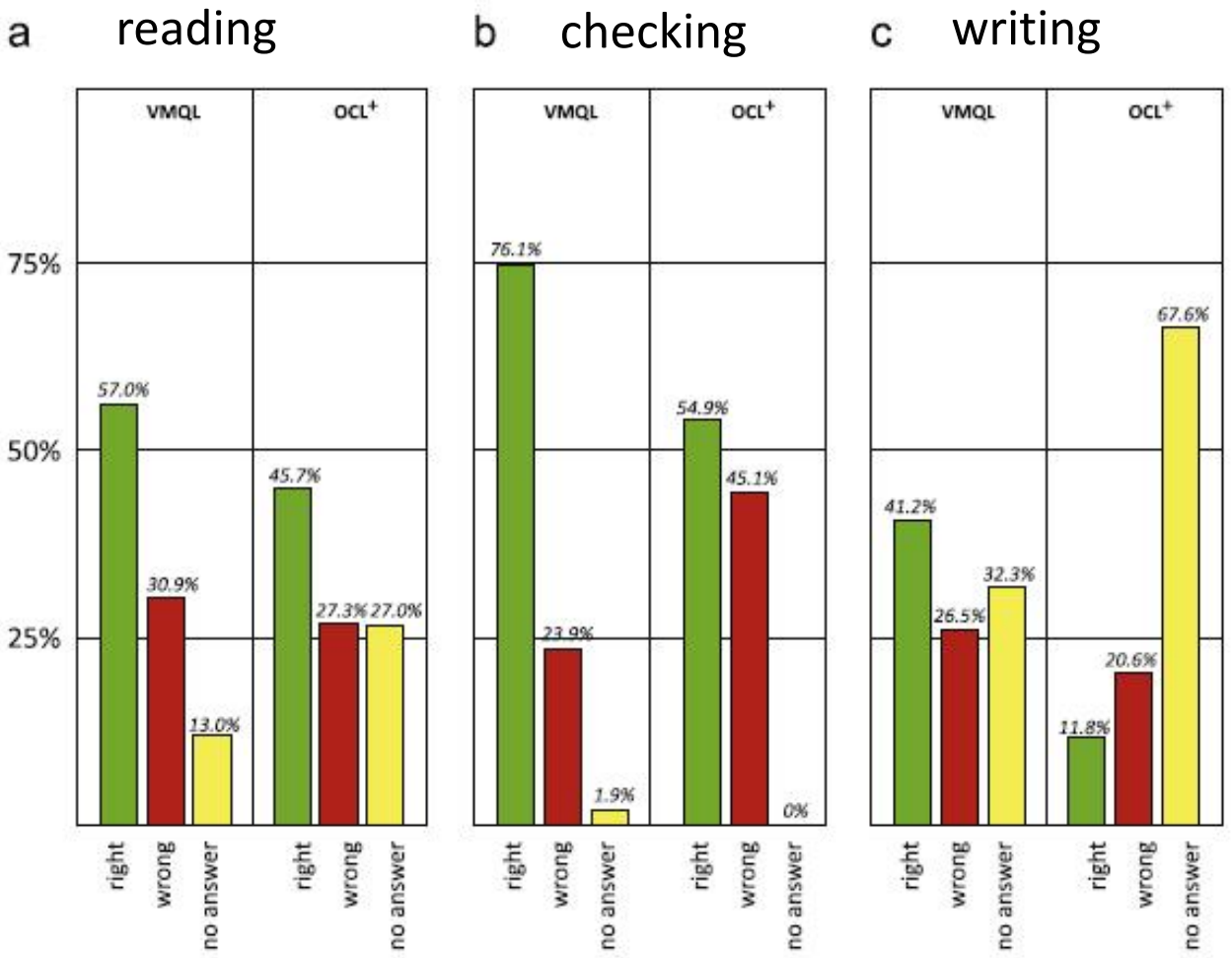
"After [the other] languages, it's hard to get yourself to work on [OCL]. [It] is rather, well, relatively complicated, I kept thinking, jeez, why does it have to be quite as complicated. [The others] are quite easy in comparison, these are easy to understand."

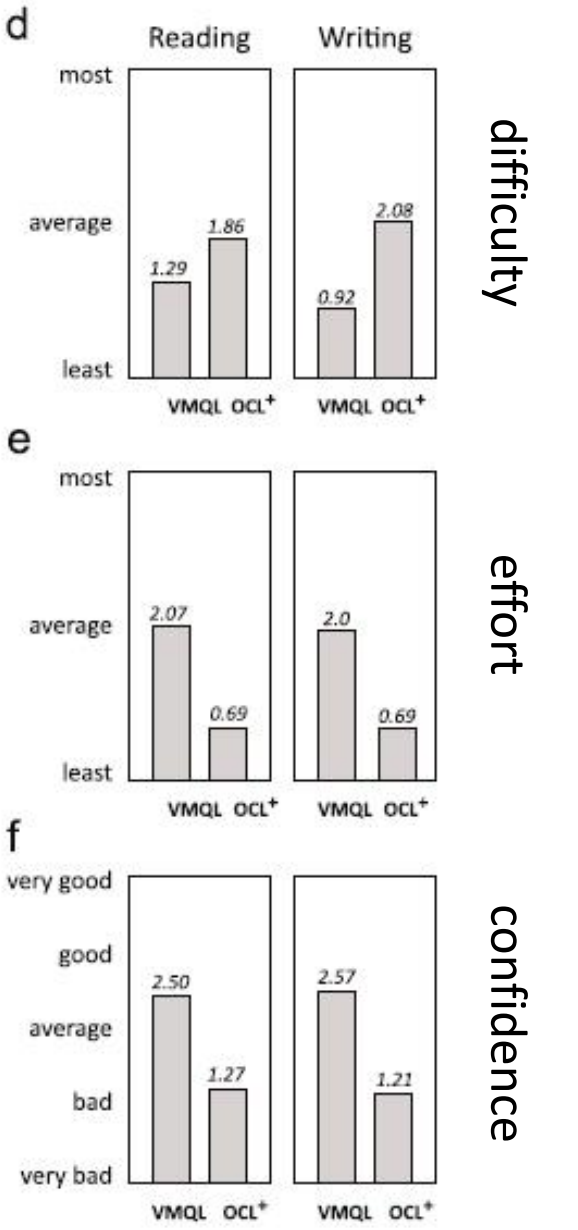
"All in all it was ok...I found OCL horrible"

"[the other language] was ok, but [OCL] is diffcult to understand, you have to follow the algorithm. That's ok, it works, but it's more effort."

"[OCL] was really pissing me off".

VMQL/OCL+ for Queries





Participants

	students	pros	sum
male	15	6	21
female	10	1	11
sum	25	7	32

Task completion

	tasks	students	pros	average
Task A (reading)	10	88.1%	100%	90.7%
Task B (reading)	12	95.3%	100%	96.3%
Task C (writing)	8	57.8%	-	57.8%

languages A-D

VMQL/OCL+ for Constraints

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%

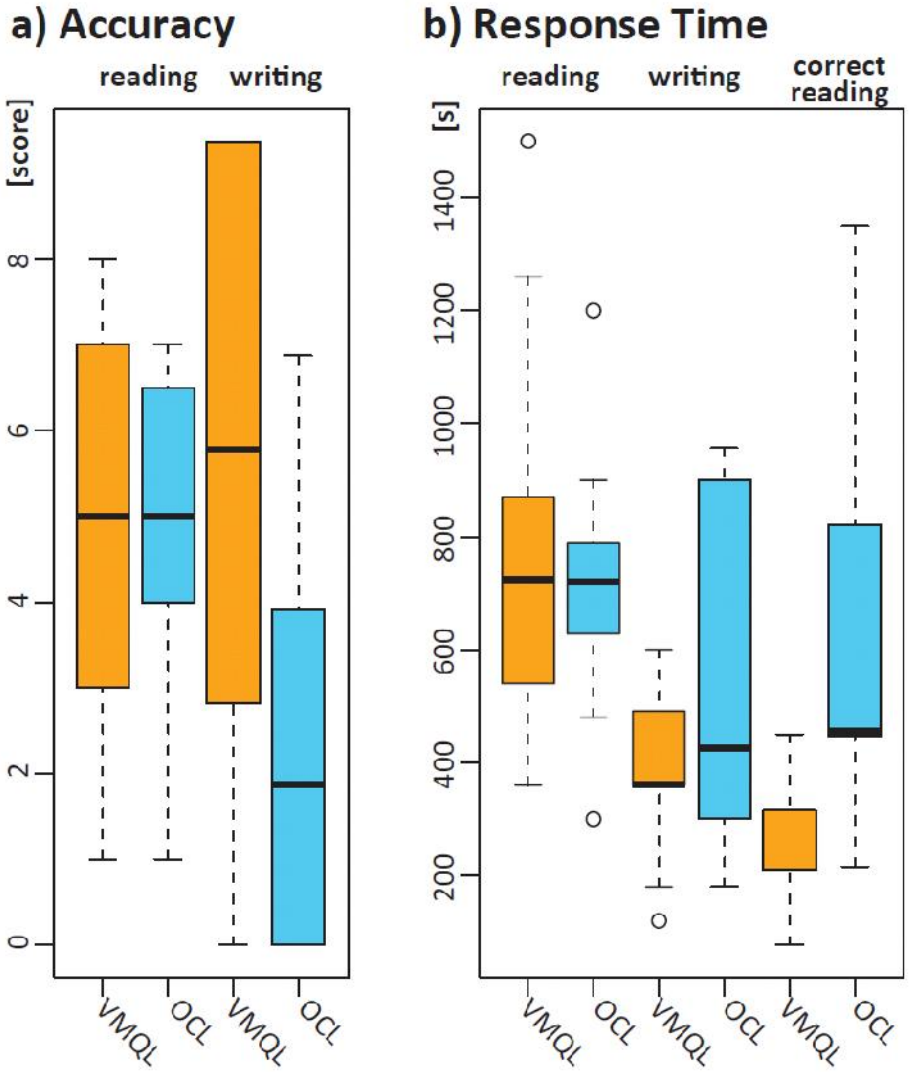
B) RESPONSE TIME [S]

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%

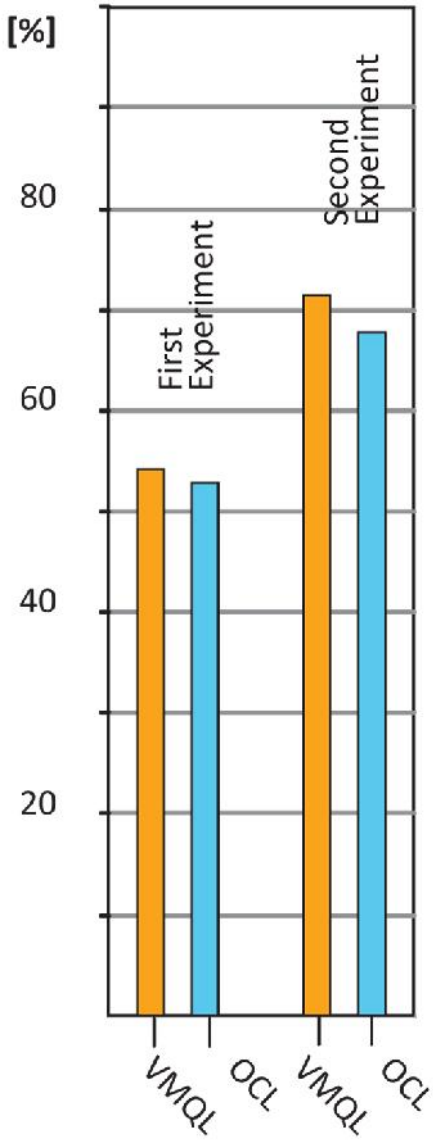
C) COGNITIVE LOAD [SCORE: 1..10]

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%

VMQL/OCL+ for Constraints



Only a) writing is significant



9+7 participants

Part 3

Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Factoring out differences: visual/textual vs. low/high level of abstraction

Università di Trento, 23.5.2014

MOCQL (1/3)

- **Suppose, you want to find all classes that capture some kind of address. In OCL this could be expressed as**

```
Class.allInstances()-> select(c |
            c.name.contains_substring("Address"))
```

under the assumption that string-simiularity function is available in some query API .

- **In MOCQL, on the other hand, we can simply say**

```
show all classes $C named like "*Adress" in M_1
```

which, we believe, is easier to understand.

- **Instead of the above query, one could also say any of the following equivalent MOCQL queries**

```
show all classes $C named like "*Adress" in M_1
in M_1 show all classes $C named like "*Adress"
show all classes $C where $C.name is similar to "*Adress" in M_1
```

- **MOCQL offers plenty of syntactical variants so that vague recall will (often) still yield the expected result. In OCL you have to recall precisely the right syntax.**

MOCQL (2/3)

- **Suppose, you want to restrict your query to contain only abstract classes. In OCL this could be expressed as**

```
Class.allInstances()-> select(c |
            c.isAbstrac=true &&
            c.name.contains_substring("Address") )
```

- **In MOCQL, on the other hand, all we have to do is add ‘abstract’ in the right place:**

```
in M_1 show all abstract classes $C named like "*Address"
```

MOCQL (3/3)

- **Suppose we were to look for abstract classes that do not have subclasses.**
  - This is an example where the typical built-in search facilities of most modeling tools will fail
- **In OCL, we would have to write something like the following.**

```
Class.allInstances()->select(c | c.isAbstract=true).
    intersection(c | c.general->isEmpty())
```

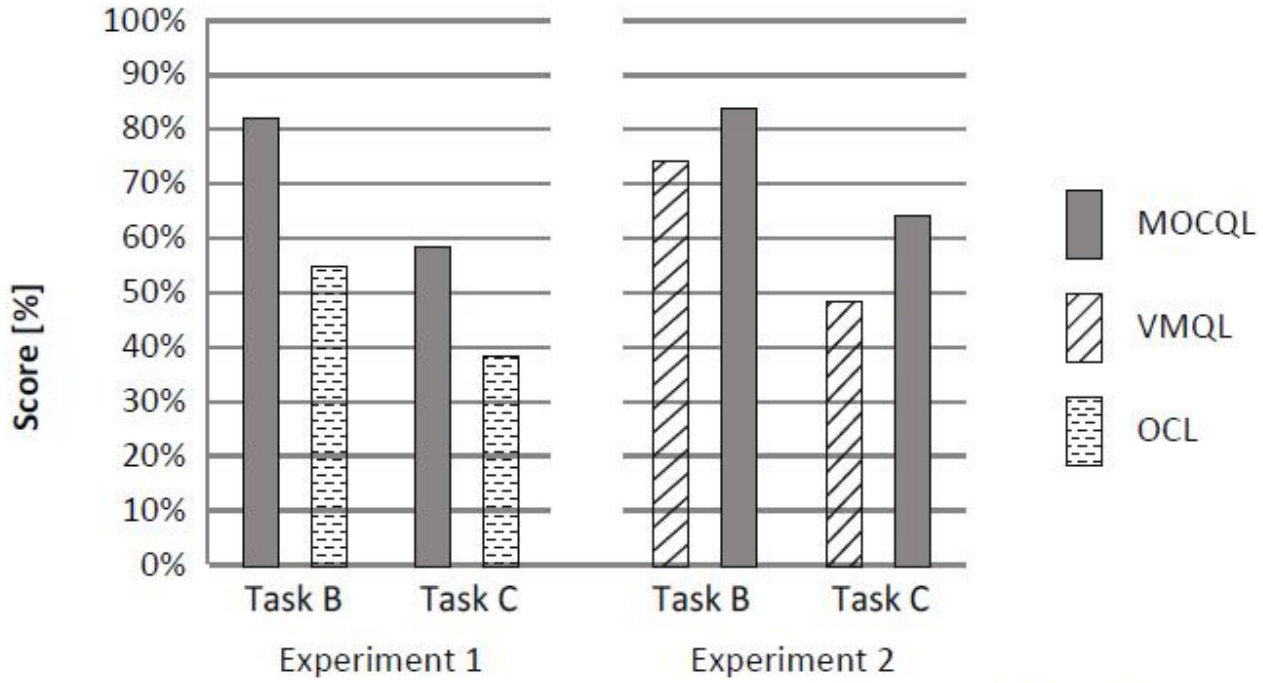
- **Substantial understanding of details of the UML meta model are required here (e.g., the property "general"), the different navigational operators (dot vs. arrow).**

- **In MOCQL, we can simply write**

```
show all abstract classes $C where there are
    no classes $SUB such that $C generalizes $SUB in M_1
```

VMQL/OCL+/MOCQL for Querying

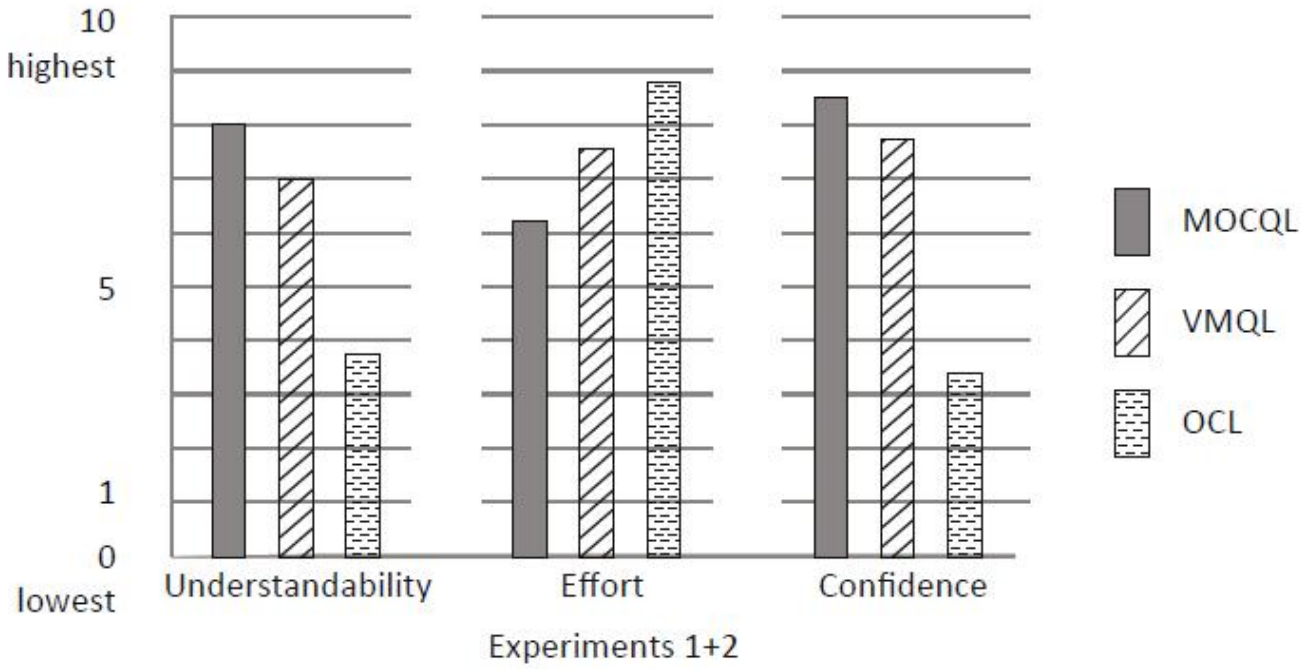
Language	Experiment 1 Task B	Experiment 1 Task C	Experiment 2 Task B	Experiment 2 Task C
MOCQL	82.1%	58.7%	83.9%	62.7%
VMQL	-	-	74.2%	49.0%
OCL	54.8%	38.1%	-	-



Hypothesis	Task	Significance
Experiment 1: Subjects perform better using OCL than MOCQL	B	$p < 10^{-7}$ ***
	C	$p < 10^{-5}$ ***
Experiment 2 Subjects perform better using VMQL than MOCQL	B	$p = 0.0033$ *
	C	$p = 0.059$ ·

VMQL/OCL+/MOCQL for Constraints

Language	Understandability		Effort		Confidence	
	μ	σ	μ	σ	μ	σ
MOCQL	8.0	1.88	6.2	3.38	8.5	2.20
VMQL	7.0	2.28	7.5	3.30	7.7	2.90
OCL	3.8	1.80	8.8	1.78	3.3	1.55



Part 4

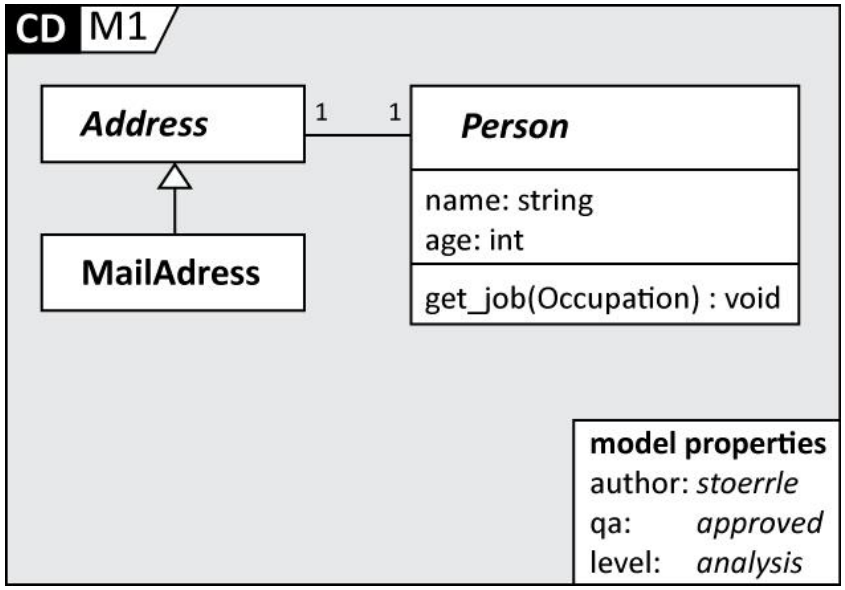
Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

The other way round: Improving the Usability of OCL by the OCL Query API (OQAPI)

2013 OCL Workshop (at MODELS'13), Miami, FL, USA

OQAPI provides convenience

1. **Find all classes named “Address”!**
2. **Find all classes named “Address” or so!**



1. **With OQAPI:**

```
classes() -> named(‘Address’)
classes() -> named_like(‘Ad?ress’)
```

2. **Plain OCL:**

```
Class.allInstances() -> select(c | c.name.contains_substring("Address"))
???
```

3. **OCL does not provide *easy* access to**
   1. Selection by name
   2. Pattern matching/wildcards in string expressions
   3. Selection of instances by type

OQAPI hides the meta model

Does "A" have any associations?

Query: Is the class named "A" associated to any other class?

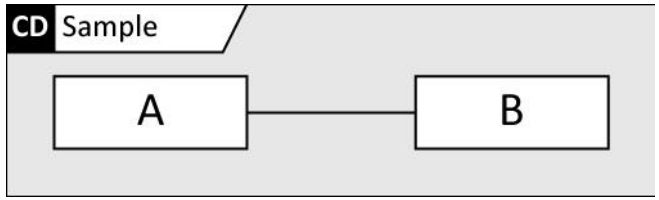
With OQAPI:

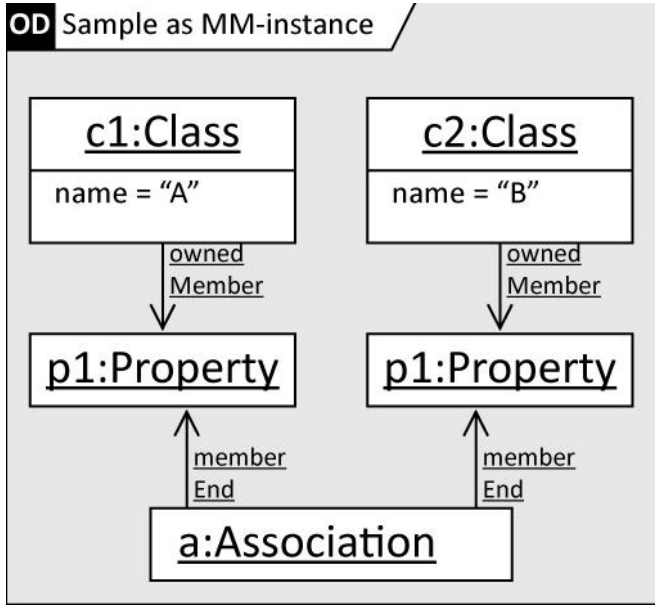
```
classes()->named('A')->associated_to()
->notEmpty()
```

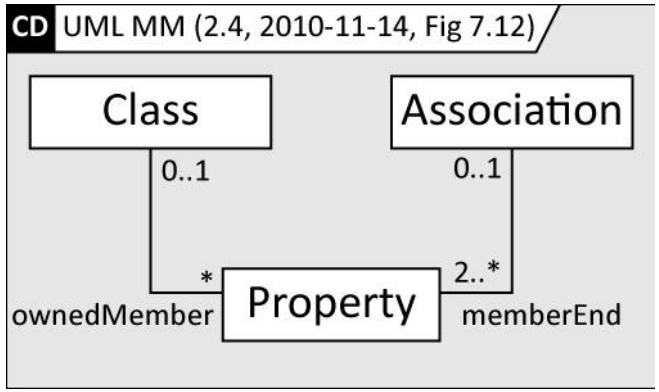
Plain OCL:

```
let end1 = c1.ownedMember,
    end2 = c2.ownedMember,
    assc = Association.allInstances()
 in collect(a | assc->includes(a)
              and a.memberEnd->intersects(end1)
              and a.memberEnd->intersects(end2)
             )->notEmpty()
```

- **This is poor OCL code, experts can do better!**
- **This is not OCL's fault, it's the complex UML MM!**







Observations 1 & 2: Scores, Cognitive Load

1. **Observations**
   1. Understandability is improved through the usage of OQAPI.
   2. Writability cannot be assessed because of lack of data.
   3. Effort goes down and confidence goes up when using OQAPI.
2. **Interpretation**
   1. The results are significant, though not highly significant, at high effect size, so we may expect much higher significance through increasing n.

	OCL (n=18)	OQAPI (n=25)	Improvement [%]	Significance (p-Value, level)	Effect size (Cohen's d, level)
Understandability					
μ	3.76	4.78	27.13%	0.097 .	-0.559 M
σ	1.80	1.84	2.22%		
Writability					
μ		5.53			
σ		2.53			
Effort					
μ	8.75	6.25	-28.57%	0.015 *	0.918 L
σ	1.78	3.23	81.69%		
Confidence					
μ	3.33	5.23	57.14%	0.013 *	-0.881 L
σ	1.55	2.50	61.29%		

Remarks

All measures normalized to 0..10

Cohen's interpre-tation for effect size

Two-tailed t-test (same results for Wilcoxon)

Threats to Validity

Study participants

- might not be representative of modelers in general
- number is relatively low (but recall: almost no previous work).

Tasks	student BSc	student MSc	other	Sum
male	14	15	1	30
female	2	2	0	4
Sum	16	17	1	34

completion rate: ~80..95%
(depending on task)

Sample models and queries

- might not be representative of „real life“ models

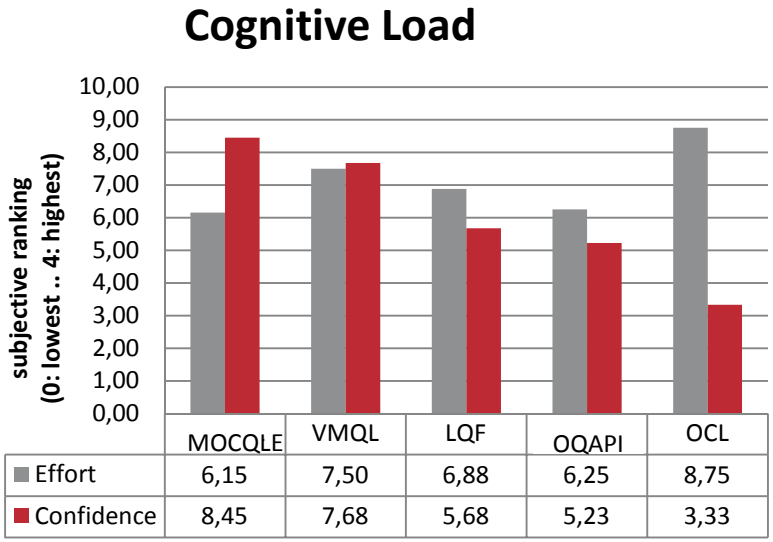
Experimental Procedure

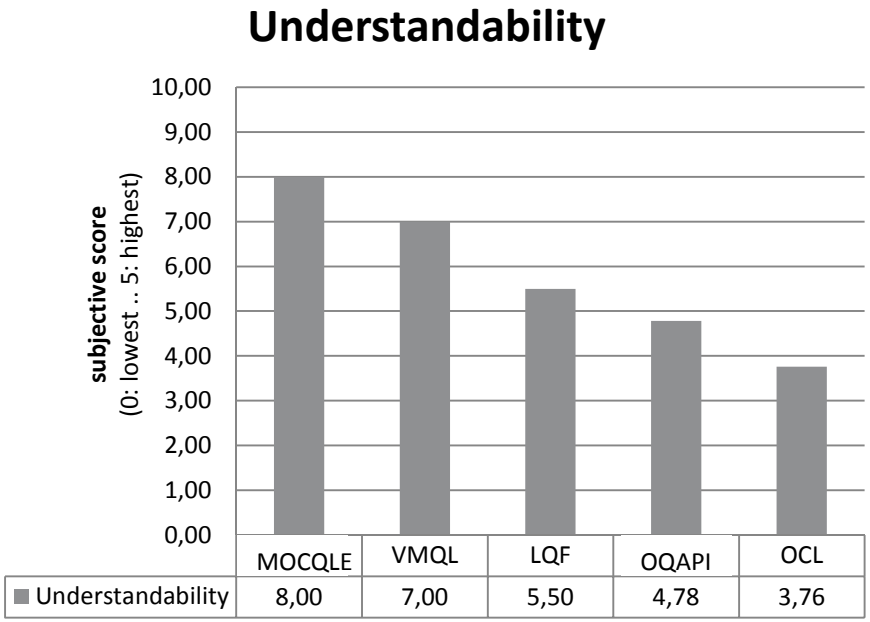
- pen&paper, so the task setting might not be realistic enough

Experimental Tasks

- tasks involved textual answer options, which might favor a textual query notation

Comparison: OCL, OQAPI, VMQL, MOCQLE





Publications in this Thread

1. **MOCQL: A Declarative Language for Ad-Hoc Model Querying**. Proc. Eur. Conf. Model Driven Development Foundations and Applications (ECMFA), Springer 2013

2. **Improving the Usability of OCL as an Ad-hoc Model Querying Language.** Proc. OCL-Workshop @MODELS, ACM 2013

3. **MQ-2: A Tool for Prolog-based Model Querying**. Proc. Eur. Conf. Model Driven Development Foundations and Applications (ECMFA). Springer 2012, with R.V. Acretoaie

4. ***VMQL: A Visual Language for Ad-Hoc Model Querying,*** Journal of Visual Languages and Computing, Elsevier, 22(1), 2011, 3—29

5. **Expressing Model Constraints Visually with VMQL,** Proc. Intl. Symp. Visual Languages/Human Centric Computing (VL/HCC), IEEE 2011

6. ***VMQL: A Generic Visual Model Query Language***, Proc. Intl. Symp. Visual Languages/Human Centric Computing (VL/HCC), IEEE 2009

7. ***A logical model query interface***. Proc. Intl. Ws. Visual Languages & Logic (VLL), 2009

8. ***A PROLOG-based Approach to Representing and Querying UML Models***, Proc. Intl. Ws. Visual Languages and Logic 2007 (VLL)

COME SIGN UP FOR MY UPCOMING
SUMMER SCHOOL ON EMPIRICAL RESEARCH
METHODS IN SOFTWARE ENGINEERING
http://www.imm.dtu.dk/~hsto/ERMSE/

Prof. Dr. Harald Störrle

Software Engineering Section
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 303b, Room 056
DK-2800 Kgs. Lyngby

Tel **0045 4525 3757**
EMail **hsto@dtu.dk**
Web **www.compute.dtu.dk/~hsto**

DTU