Electronic Communications of the EASST
Volume 34 (2010)

Proceedings of the 6th Educators' Symposium: Software Modeling in Education at MODELS 2010 (EduSymp 2010)

Teaching OCL Standard Library: First Part of an OCL 2.x Course

Joanna Chimiak–Opoka, Birgit Demuth

13 pages

Guest Editors: Peter J. Clarke, Martina Seidl
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Teaching OCL Standard Library: First Part of an OCL 2.x Course

Joanna Chimiak–Opoka[1], **Birgit Demuth**[2]

[1] University of Innsbruck, Austria, joanna.opoka@uibk.ac.at
[2] Technische Universität Dresden, Germany, birgit.demuth@tu-dresden.de

Abstract: Our aim is to provide a complete set of materials to teach OCL. They can be used in bachelor or master programs of computer science curricula and for training in an industrial context. In this paper we present the first part of the course related to the OCL Standard Library. This part provides model independent examples to teach OCL types and their operations. It enables users to gain a basic understanding of the OCL Standard Library, which can be used as a starting point to write model constraints (OCL specifications) or model queries. Additionally, to the content of the paper, we provide a set of OCL packages, exercise proposals and lecture slides.

Keywords: teaching material, OCL types, model independent OCL expressions

1 Introduction

The Object Constraint Language (OCL) is crucial in precise modeling [WK03]. Despite its usability it is far less frequently used in the industrial context [Amb04] than the Unified Modeling Language. We consider three main reasons for this fact: weakness of standardisation, basic tool support, and lack of social acceptance.

The OCL **standard**, also the recent 2.2 version [OMG10], has several known inconsistency and incompleteness issues, as discussed in the recent OCL Workshop [CCG+09]. A crucial problem is the discrepancy between the syntax and the semantics of the language specification. The provided syntax is for 2.2 version, but the semantic is based on [Ric02] and was defined for 1.3 version. The attempt to partially bridge the gap in the formalisation of OCL 2.1 was proposed in [BKW09]. The inconsistencies in the standard cause difficulties in understanding and implementing OCL.

Fortunately, regardless of this fact there are many **tools** supporting OCL. Currently there is a collection of OCL tools and UML tools with OCL support[1]. Developers of the tools struggle with weaknesses of the standard and have to introduce their own interpretation what leads to slight differences between implementations [GKB08]. Overcoming weaknesses of the standard in the academic context resulted in putting correctness of the implementation in the main focus and neglecting a user friendly front–end. In the commercial context OCL is rather neglected and it seems to be only an add–on to modeling tools. To partially address pragmatics of OCL tools we proposed requirements for a user friendly integrated development environment for OCL [CDSR09].

[1] List of tools at OCL Portal: http://st.inf.tu-dresden.de/oclportal/ OCL Software,
Comparison of tools by Jordi Cabot: http://www.jordicabot.com/research/OCLSurvey/index.html

The last issue, the **social acceptance** of OCL, results from the previous ones. There exists a belief that it is hard to use, learn and teach OCL. It has been shown, e.g. [Ack01], that in general, it is a *difficult, error–prone and time–consuming task* for practitioners to define OCL expressions. Moreover, *OCL expressions are often unnecessarily hard to read* [VJ00], UML/OCL models may be *difficult to understand and evolve, particularly when constraints containing complex or duplicate expressions are present* [CWO07]. Additionally, difficulties with teaching OCL were reported in [MR09]: *introducing it to non software developers is almost impossible and the professional programmers usually do not like it: it looks like a programming language, but it is not; it has first order logic semantics, but it does not look like it.*

Our aim is to **support learning and teaching of OCL**. Based on our teaching experience we designed a complete OCL course material. The course consists of several parts introducing basic types and operations, model and meta model specifications, model constraints and queries. This paper presents the first part of the course with the basic concepts. It focuses on the OCL Standard Library teaching its types by model–independent OCL expressions, however it is required to define an empty model context. According to our knowledge, our course is the first resource focusing on the core of the OCL (Section 6.2). In our opinion teaching the core of the language is important to its successful application in context of modeling and model transformation. Understanding of model–independent OCL expressions and proficiency in writing them is a prerequisite of an efficient specification of model constraints or model queries. These skills enable focusing on the content instead of on the form of the OCL expressions when writing large specification in OCL, QVT[2] or other extensions of OCL. A modular teaching approach should reduce the learning effort of newcomers to the language, as noticed in [AZH08].

In the first line we assume the course to be used in the academic context in a **computer science curricula**. An initial version of the course was successfully taught at the University of Innsbruck in model engineering course in the master program[3]. In Dresden, OCL is one of the topics in a course about advanced software engineering and is planned as part of a future course about model–driven software engineering. The improved version of the course will be taught at the University of Innsbruck and at TU Dresden. In both cases, students have strong basic skills in object–oriented software development (analysis, design including design patterns and Java programming). Additionally, the course will be freely available to academic teachers, students, professional software developers and modelers with basic skills in object–oriented software development. We provide two variants of the OCL specifications: standard and extended one. The **standard *TU Dresden* variant** is suitable for language purists and for easier tool interoperability as the following constraint shows:

```
1 context Model
  inv addRealReal:
3   let v1: Real = 1.0 in
      let v2: Real = 1.0 in
5       v1 + v2 = 2.0
```

It can be used with different tools, however due to inconsistencies in implementations we can only grant the proper usage with the tools we tested. The **extended *University of Innsbruck* variant** incorporates libraries, tests and documentation comments [CO09]. It is intended to make

[2] QVT: Query/View/Transformation http://www.omg.org/spec/QVT/

[3] http://st.inf.tu-dresden.de/oclportal/ Courses / Academic Courses / Specifying Constraints with OCL

learning OCL easier by enriching it with techniques used in programming such as the following analogous OCL **test** example shows:

```
1 modelinstance none
  test tAddRealReal:
3   let v1: Real = 1.0 in
      let v2: Real = 1.0
5   expected v1 + v2 = 2.0
```

Teachers have to decide what variant is the most appropriate one in their didactic context.

The remainder of the paper is structured as follows: Section 2 provides basic concepts crucial for teaching OCL. Sections 3–5 succesively introduce OCL types: special, primitive and composite ones. In Section 6 we describe teaching resources of our course and list alternative ones. And finally, Section 7 gives a conclusion and future work.

2 Overview

In this section we give remarks on OCL as a typed specification language with a hybrid nature.

It is important to notice, that OCL is neither a typical programming language, a formal specification language, nor a modeling language. As mentioned in [MR09]: it looks like a programming language, but it is not. OCL may be confused with programming language because of its concrete textual syntax, object–oriented paradigm and navigation style notation. On the other hand, it follows the declarative paradigm, which makes it harder to understand and to debug. Moreover, *it has first order logic semantics, but it does not look like it* [MR09]. It may make difficult to see the paradigm shift to the logic and lead to confusing OCL with a programming language. OCL is a modeling language as it is related to model engineering activities, but it is not a stand–alone language. OCL has a **hybrid nature**, i.e. expressions are defined on a underlying metamodel and evaluated for models (instances of the metamodel).

For the purpose of this paper we will only consider the basic part of OCL, which is independent of any model. In Fig. 1 we present basic types, omitting types related to modeling, such as Model Element, Enumeration, Unlimited Natural, OCL Message and OCL State. For didactic purposes we introduced several additional elements to the **type hierarchy** to make it easier to understand. The elements (Type, PrimitiveType, OCLSpecialType, CompositeType) are defined as abstract and depicted in orange in Fig. 1. Please notice that they are not defined in the standard, at least not this way. Furthermore, it is important to understand that OCL is a **typed language**. That means that types and operations must match when constructing complex OCL expressions. *Type conformance rules* are defined for pairs of types. A conformance of two types means that *Type1* ***conforms*** *to Type2 if an instance of Type1 can be substituted at each place where an instance of Type2 is expected* [WK03]. For example, Boolean is a subtype of OclAny and therefore conforms to OclAny. This should be intuitively grasped by students with Java programming experience.

In the following sections we describe three type sub–hierarchies: OCL special types (Section 3), primitive types (Section 4) and composite types (Section 5). Examples and exercises for all described OCL types and issues are provided within our course material (Section 6.1).

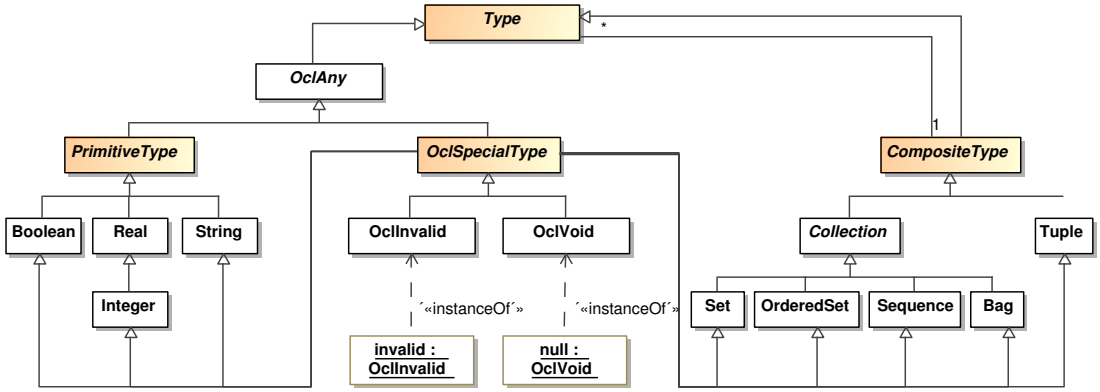

Figure 1: Hierarchy of basic types according to the OCL 2.2 Standard Specification.

3 OCL Special Types

We consider three OCL special types: OclAny (Section 3.1), OclInvalid, and OclVoid (Section 3.2). Below we give definitions of each type and point out special characteristics of them.

3.1 OclAny Type

OclAny behaves as a supertype for all the types, except composite types. *Features of OclAny are available on each object in all OCL expressions* [OMG10]. It is also important to stress that OclAny is a supertype of all primitive types and OCL special types, but not of composite types. Notice that some OCL parsers enable calling OclAny's operations on collection types but this is not standard conform. In principle OCLAny plays the role that Object plays in the Java language. Thus OCLAny offers several reflection operations.

For the OclAny type comparison two operations are defined: $=$ and $<>$. However, it is not really clear how equality ($=$) is defined. The OCL specification defines it by object identity (*same object*). Some OCL tools indeed implement equality by equality of values [SCC10]. Furthermore, several checks for concrete types are defined: oclType() returns the type of which self is an instance. oclIsTypeOf($\cdots$) checks if self is of the given type (but not a subtype of it), oclIsKindOf($\cdots$) checks if the type of self conforms to the given type (self is of the given type or a subtype of it), oclIsUndefined() checks if the type of self is of type OclUndefined (self is equal to null), and oclIsInvalid() checks if the type of self is of type OclInvalid (self is equal to undefined).[4] Additionally, there is a method for type casting: oclAsType($\cdots$).

In OCL it is possible to specify static (or class) features (attributes and operations) that are similar in Java or other object–oriented languages applicable to types themselves instead to their instances. Regarding the type system of the standard library there is only one static operation: allInstances(). This operation results in a set of all instances of that type, including all instances of its subtypes. However, the use of allInstances() is explicitly discouraged because it is difficult to find all instances of a type. In the OCL standard library type hierarchy allInstances() can only

[4] There are two additional checks oclIsNew(), oclIsInState() but they are related to model elements.

be applied for the enumeration types Boolean, OclInvalid and OclVoid.

As OclAny is an abstract type, it can be used in a declaration (variable, operation parameters and return type) but there are no direct instances of it. To teach operations defined in OclAny values of concrete types must be used.

3.2 OclVoid and OclInvalid Types

The type OclVoid is a type that conforms to all other types except OclInvalid. It has one single instance, identified as null. Similarly, *the type OclInvalid is a type that conforms to all other types except OclVoid. It has one single instance, identified as invalid.* Any property call applied on null or invalid *results in OclInvalid except for the operation oclIsUndefined() and oclIsInvalid()* [OMG10]. The semantics of both types is not yet completely specified in the current OCL specification and can be seen as problematic because that means that they conform to one another, resulting in a generalization cycle. Conceptually, there are still some issues to be solved for the next OCL 2.3 specification.[5]

For basic understanding of these concepts it is useful to make a **programming language metaphor**[5]. Null in OCL can be understood similarly to null in Java. Operation invocations on null values or operation invocations having null values as parameters result in invalid. This situation can be compared with exceptions in Java (NullPointerException).[6]

There are several special cases when dealing with null and invalid. Basically, handling of null and invalid in logical operations results in a four–valued logic (Section 4.1). It is important to make students aware of being careful when working with undefined and invalid values.

- Conversion to a collection type from null leads to an empty collection.
- Collections may contain null values but not invalid values. If a collection contains an invalid value it is invalid as a whole.
- The specification does not specify how to deal with comparison operations: equality and non–equality. However, for practical reasons, it should be possible to apply them on null and invalid values.
- Invocation of oclIsUndefined() on invalid results in false and in particular cases it can lead to false conclusions. E.g. `10.div(0).oclIsUndefined()` evaluates to true although the result is not null but invalid instead.
- Handling of null and invalid in logical operations results in a four–valued logic (Section 4.1).

As OclVoid and OclInvalid conform to OclAny, both types inherit all operations from OclAny (Section 3.1). The only specific operations are operations that redefine the OclAny type comparison operation (=) for OclVoid and OclInvalid.

[5] A discussion on *Treatment of ocl undefined and ocl invalid* at the OCL Group at LinkedIn provides insights on the practical solution used in Dresden OCL (Claas Wilke) and OMG OCL 2.2 revisions (Ed Willink).

[6] *Thus, invalid can be considered as some sort of Exception or Throwable. Unfortunately, invalid in OCL is not typed, and thus, it is not possible to find out what went wrong nor catch specific subtypes of exceptions* — from the discussion on *Treatment ...*

4 Primitive Types

In this section we describe how to deal with primitive types in OCL: Boolean, Real, Integer and String. In this part there are a few issues to be careful about, especially four–valued logic (Section 4.1) and a weak support of advanced arithmetical (Section 4.2) and text operations (Section 4.3). In Section 4.4 we show how to overcome weaknesses of OCL.

4.1 Boolean Values and Logical Operations

Basically OCL users work with the Boolean values true and false that are the instances of the Boolean type. For these standard Boolean values Boolean algebra and De Morgan's laws are applied. Additionally we have to, as explained in Section 3.2, handle two special OclAny values (null and invalid) in logical operations resulting in a four–valued logic (4VL). As the Boolean algebra is a logical calculus of truth tables, the OCL specific 4VL is also represented in a truth table (Tab. 1). Logical operations can be written both using operators by reserved keywords (such as a and b) and by infix operators (such as a.and(b)). OCL provides the basic logical operations and, or, not as well as the derived operations xor and implies. Conflicting reserved keywords and property names can be solved by using the _ (underscore) prefix. For future OCL specifications it is proposed that the above mentioned example is written as a._'and'(b).

Table 1: A truth table for the four–valued logic in OCL. Note that the truth table in the current standard [OMG10] contains errors which will be removed in the next OCL version (2.3). The presented truth table is the already updated one.

a	b	not a	a or b	a and b	a implies b	a xor b
false	**false**	**true**	**false**	**false**	**true**	**false**
false	**true**	**true**	**true**	**false**	**true**	**true**
false	null	**true**	invalid	**false**	**true**	invalid
false	invalid	**true**	invalid	**false**	**true**	invalid
true	**false**	**false**	**true**	**false**	**false**	**true**
true	**true**	**false**	**true**	**true**	**true**	**false**
true	null	**false**	**true**	invalid	invalid	invalid
true	invalid	**false**	**true**	invalid	invalid	invalid
null	**false**	invalid	invalid	**false**	invalid	invalid
null	**true**	invalid	**true**	invalid	invalid	invalid
null	null	invalid	invalid	invalid	invalid	invalid
null	invalid	invalid	invalid	invalid	invalid	invalid
invalid	**false**	invalid	invalid	**false**	invalid	invalid
invalid	**true**	invalid	**true**	invalid	invalid	invalid
invalid	null	invalid	invalid	invalid	invalid	invalid
invalid	invalid	invalid	invalid	invalid	invalid	invalid

4.2 Numbers and Arithmetical Operations

There are two types to express numbers in OCL: Real and Integer, where Integer is a subtype of Real. The basic arithmetical ($+$, $-$, $/$, $*$, abs(), min($\cdots$), max($\cdots$)) and comparison operations ($>$, $<$, $>=$, $<=$) are defined for numbers. Two types of conversion from Real to Integer are provided (floor(), round()) and additionally, in OCL 2.2, a conversion to string was introduced (toString()). There are two operations defined for the Integer type only: integer division (div($\cdots$)) and modulo (mod($\cdots$)).

4.3 Strings and Text Operations

Text operations, especially string comparison, are a weakness of OCL. The latest OCL specification [OMG10] provides several new operations on strings. In OCL 2.2, it is possible to access single or all characters of a given string: at($\cdots$) which queries the character at given position and characters() which obtains the characters of the string as a sequence. Additionally, to case conversion (toLowerCase(), toUpperCase()) an explicit operation to compare strings under case–insensitive collation was introduced: equalsIgnoreCase($\cdots$). Another useful operation is indexOf($\cdots$) which queries the index in self at which a given substring occurs in a string or zero if the substring does not occurs in the string. It can be used as a check if a given substring occurs in a string.

4.4 Extensions

Additionally, to the standard library we show two mechanisms for extending expressiveness of OCL: one within OCL (definition mechanism) and another outside of OCL (black box implementations). The black box implementations are used in QVT to allow complex algorithms to be coded in any programming language. Additional definitions can be used for advanced text operations and complex arithmetical algorithms, whereas back box extensions for regular expressions.

5 Composite Types

In this Section, we motivate and explain the introduced abstract Composite Type with the subtypes for collections and tuples (Fig. 1). Tuples and all collections are *composite* because they contain elements conform to any concrete subtype of the root type Type. The term *element* is used for an object in a collection. The type Type of the elements are designated in the OCL specification as template type 'T' in all collection operations. The elements in a tuple are understood as attributes of the tuple each attribute consisting of a name and a type. Students should be made aware that this type hierarchy is designed based on the composite design pattern. Although the type hierarchy in the OCL specification is not exactly defined as in this paper we found this representation from a conceptual and didactic point of view more comprehensible. Note that the OCL specification uses the term *part* to explain what a tuple is. But we prefer for orthogonal notions the term *element* to motivate the composite pattern.

5.1 Collection Types

OCL collection types can be explained based on the MultiplicityElement from the UML specification [OMG09]. Besides the definition of the bounds of an actual multiplicity the MultiplicityElement also *includes specifications of whether the values in a instantiation of this element must be unique or ordered.* If the upper bound of the specified interval of a multiplicity is greater than one we have an association end that results in a collection. The UML specification defines based on the Boolean properties IsUnique (default value = true) as well as IsOrdered (default value = false) of a MultiplicityElement different kinds of a collection (Set, Bag, Sequence, and OrderedSet). Collection types in OCL are the same and therewith their definitions conform to the UML specification. Tab. 2 shows the mapping of the properties to the collection types. A discussion on the Collection types hierarchy may be found in [BGH+09].

Table 2: Classification of collection types based on their properties.

PROPERTIES	IsOrdered	not IsOrdered
IsUnique	OrderedSet	Set
not IsUnique	Sequence	Bag

During a course it should be explained that there are two kinds of operations for each collection type: (basic) operations (e.g. size()) and predefined iterator expressions (e.g. forAll(···)). In [WK03] **basic operations** are classified by their meaning basically into following groups:

- equals(=) / notEquals (<>) operations,
- including / excluding operations,
- the flatten() operation,
- transformation operations (asSet(), asSequence(), asBag(), asOrderedSet()),
- typical set operations (union(···), intersection(···), difference (−), symmetricDifference(···)),
- order related operations (first(), last(), at(···), indexOf(···), insertAt(···), subSequence(···), subOrderedSet(···), append(···), prepend(···), reverse()).

For better comprehension we recommend presenting a matrix of operations with their signatures and owning types. Due to space limitations, we refer to our teaching materials (Section 6.1). A sensible issue is the difference between includes() and including() as well as excludes() and excluding(). It should be carefully explained on their signatures and examples. The including operation results in a new collection with one element added to the original collection [WK03]. The include operation results in true if the parameter object is an element in the collection. Another issue is the incompleteness of the list of all collection operations[7]. In particular it is generally recognized that some of the above listed collection operations are missing for the OrderedSet type in the OCL 2.2 specification.

Iterator expressions loop over the elements in a collection and have an OCL expression as parameter. Because iterator expressions are complex to write, often recurring iterator expressions are predefined (exists(···), forAll(···), isUnique(···), any(···), one(···), collect(···),

[7] see the OMG OCL discussion mailing list at http://www.omg.org/issues/ocl2-rtf

collectNested(···), sortedBy(···), select(···), reject(···)). Basically it is possible to add new iterator expressions in the OCL standard library. Students should study the mapping of the predefined iterator expressions to the iterate construct and learn by a few examples to write iterator expressions themselves.

5.2 Tuples

The TupleType (informally known as record type or struct) combines different types into a single aggregate type. As explained above, the elements of a TupleType are described by its attributes each having a name and a type. The type names are optional, and the order of the elements is unimportant. According to the composite pattern there is no restriction on the kind of types that can be used as elements of a tuple. Each element is uniquely identified by its name. It is possible to compose several values into a tuple. A tuple consists of named parts, each of which can have a distinct type. The values of the parts may be given by arbitrary OCL expressions.

The tuple type is crucial for the power of OCL as a query language. Before introducing this type, OCL (1.x) was not equivalent to the relational calculus [MC99]. However, there are no additional operations defined for the tuple type. The elements are accessed by their names and the OCL dot notation. It is very important to teach how to deal with single tuples and collections of tuples.

6 Teaching Resources

In this section we provide information about resources we provide for the afore described course (Section 6.1) and we compare it with other available resources for teaching OCL (Section 6.2).

6.1 Course on OCL Standard Library

The course package[8] contains lecture slides, source files with tasks for students and OCL expressions tested with the SQUAM OCL editor and Dresden OCL.

Lecture slides provide an extended version of diagrams for all types, their definitions and issues to be careful about. The slides, with the project and the set of OCL expressions, can be used as self–learning material.

The project consists of **16 packages** with **over 250 OCL expressions**. In the basic part, expressions correspond to particular OCL types (Fig. 1 in Section 2). The advanced part consists of explicit definitions of predefined iterator expressions (Section 5.1) and complex method definitions for numbers and strings (Section 4.4). There are, in total, **over 100 exercises** to test or extend existing OCL expressions or to write new one.

The source files are provided in two variants: standard and extended [CO09]. The overview of available versions in given in Tab. 3. Please note that there are unfortunately differences in the semantics how different tools evaluates OCL expressions. A benchmark provided in [GKB08] is dedicated for an older OCL specification. It points out important issues, however the OCL specific 4VL with the invalid value is not considered there. When teaching and learning OCL,

[8] http://squam.info/ocleditor/doc/OCLCourse/

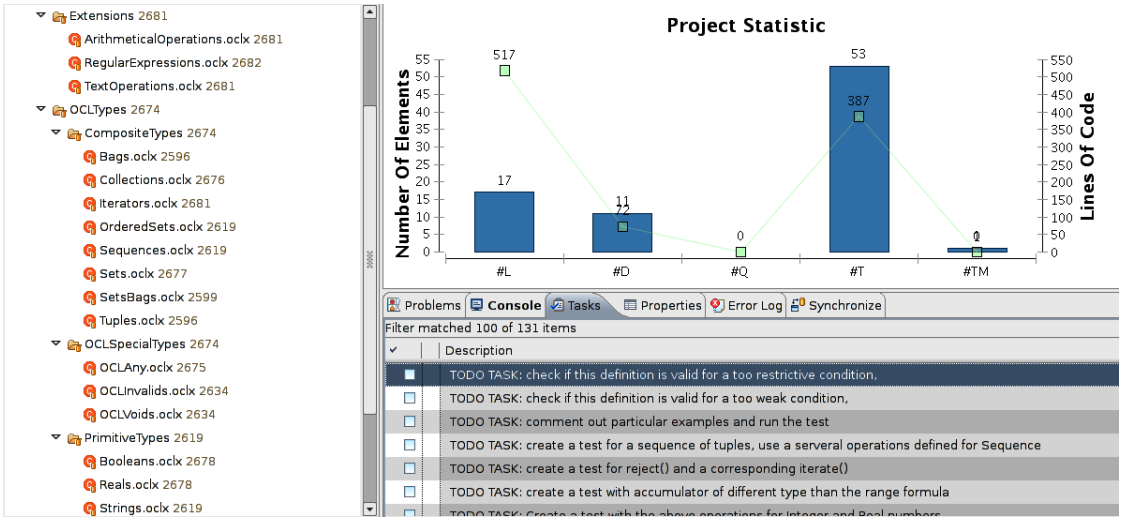

Figure 2: A screen–shot from the SQUAM OCL Editor with OCL course resources. At the left side: the project's structure in the project explorer can be seen. At the top–right side: project statistics with number of elements and lines of code. At the bottom–right side: exercises for students denoted in the OCL files with '– TODO TASK: ...' and navigable from the task view.

teachers and students should be aware of differences between standard specification and its implementations.

Table 3: Overview of OCL resources with corresponding OCL versions and tool configurations.

version	variant	OCL editor	OCL parser
OCL 2.0	extended	SQUAM OCL Editor	Galileo Eclipse MDT/OCL
OCL 2.1	extended	SQUAM OCL Editor	Helios Eclipse MDT/OCL
OCL 2.2	standard	Dresden OCL 2.2 editor	Dresden OCL 2.2 parser
tool		webpage	
Dresden OCL		http://dresden-ocl.sourceforge.net/	
SQUAM OCL Editor		http://squam.info/ocleditor/	
Eclipse MDT/OCL		http://wiki.eclipse.org/MDT/OCL	

6.2 Alternative Examples

As far as we know, our course is the only one focusing on model–independent expressions and covering the semantics and newest methods introduced in OCL 2.2. However, there are other valuable teaching resources we would like to mention here.

The well–known and most extensively used example in OCL teaching is the *"Royal and Loyal" system example* introduced and used in OCL text books as in [WK03]. The example provides an extensive set of model–dependent OCL expressions for earlier versions of OCL (up to 2.0). OCL expressions for this example are provided with several tools, among others with

Dresden OCL[9], Eclipse MDT/OCL[10], and ITP/OCL[11].

Several OCL courses (scripts, slides, OCL expressions) are available from the OCL Portal[12]. They are provided by the following teachers (tools): Heinrich Humann, Birgit Demuth, Jurriaan Hage (Dresden OCL), Lothar Schmitz (USE), and Joanna Chimiak–Opoka (OCLE). Additionally, further examples of OCL expressions are provided by Martin Gogolla together with USE[13] and as teaching materials[14] including *UML and OCL in Conceptual Modeling*[15] and *Exercises for Teaching OCL Constraints*[16].

UML tools which support OCL typically also provide OCL examples for documentation or evaluation such as Papyrus[17]. Screencasts and videos can also be helpful to learn how to use OCL in a development environment. MagicDraw UML[18] and Borland Together[19] provide such online tutorials. For further references, we will constantly update the list of OCL software and tutorials at the OCL Portal. At the same time we are looking for support of the OCL community to add their knowledge to the portal.

Besides UML model examples, students can study OCL usage for models based on other metamodels than UML, such as the Java metamodel, XSD (XML Schema), and EMF Ecore[20].

7 Conclusion

Teaching OCL is a challenging task because of an imprecise and incomplete standard specification [OMG10]. This fact has, over the years, caused confusion about the nature of OCL and differences in its semantics and implementations. As a result there is a resistance to teaching and to learning this language. Our intention was to help to overcome this resistance by providing a solid course on the core part of OCL. We provided instructions for teachers and learning materials which include OCL expressions tested with OCL tools developed at our universities.

The course presented in this paper is the first part of a larger OCL course we plan to provide. It will be integrated into and distributed with our OCL tools to give users a comprehensive example set of them. The second part of the planned OCL course will present examples how to specify *business rules* by OCL constraints on the model layer. OCL constraints for the specification of *well–formed rules* on the metamodel layer as well as model queries will be subject of the third part of the planned course. Additionally, we plan to adapt OCL expressions to be usable as a benchmark extending [GKB08] with new methods introduced in OCL 2.2 and the four–valued logic.

[9] http://dresden-ocl.sourceforge.net/4eclipse_usage.html
[10] http://wiki.eclipse.org/MDT/OCL/FAQ
[11] http://maude.sip.ucm.es/itp/ocl/examples.html
[12] http://st.inf.tu-dresden.de/oclportal/ Courses
[13] http://www.db.informatik.uni-bremen.de/projects/use/use-documentation.pdf
[14] http://www.db.informatik.uni-bremen.de/teaching/courses/ss2010_eis/
[15] http://www.db.informatik.uni-bremen.de/teaching/courses/ss2010_eis/bookConceptualModelling.pdf
[16] http://www.iem.pw.edu.pl/edusymp08/M.Gogolla_ocl.pdf
[17] http://www.papyrusuml.org
[18] http://www.magicdraw.com/files/viewlets/MD_viewlets_Validation_viewlet_swf.html
[19] http://www.borland.com/de/products/together/
[20] http://dresden-ocl.svn.sourceforge.net/viewvc/dresden-ocl/trunk/ocl20forEclipse/doc/pdf/manual.pdf, p. 112

Acknowledgements: The research herein is partially conducted within the competence network Softnet Austria (www.soft-net.at) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsfoerderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).
We would like to thank our colleagues for their support in conceptual and technical aspects of our teaching project, especially Colin Atkinson, Hannes Mösl, Claas Wilke, and Kevin Church.

Bibliography

[Ack01] J. Ackermann. Fallstudie zur Spezifikation von Fachkomponenten. In Turowski (ed.), *2. Workshop Modellierung und Spezifikation von Fachkomponenten*. Pp. 1–66. Bamberg, Deutschland, 2001. (In German).

[Amb04] S. Ambler. *The Object Primer Third Edition Agile Model-Driven Development with UML 2.0*. Cambridge, Cambridge, UK, 2004.

[AZH08] D. Akehurst, S. Zschaler, G. Howells. OCL: Modularising the Language. In *Proceedings of the Workshop Ocl4All: Workshop at MoDELS 2007*. Volume 9. Electronic Communications of the EASST, 2008.
http://www.easst.org/eceasst

[BGH+09] F. Büttner, M. Gogolla, L. Hamann, M. Kuhlmann, A. Lindow. On Better Understanding OCL Collections *or* An OCL Ordered Set Is Not an OCL Set. Pp. 276–290 in [Gho10].

[BKW09] A. D. Brucker, M. P. Krieger, B. Wolff. Extending OCL with Null-References. Pp. 261–275 in [Gho10].

[CCG+09] J. Cabot, J. Chimiak-Opoka, M. Gogolla, F. Jouault, A. Knapp. Ninth International Workshop on the Pragmatics of OCL and Other Textual Specification Languages. Pp. 256–260 in [Gho10].

[CDSR09] J. Chimiak-Opoka, B. Demuth, D. Silingas, N. F. Rouquette. Requirements Analysis for an Integrated OCL Development Environment. *Electronic Communications of the EASST: The Pragmatics of OCL and Other Textual Specification Languages 2009* 24, 2009. (presented at OCL Workshop).

[CO09] J. Chimiak-Opoka. OCLLib, OCLUnit, OCLDoc: Pragmatic Extensions for the Object Constraint Language. In Schuerr and Selic (eds.), *Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009, Denver, Colorado, USA, October 4-9, 2009, Proceedings. LNCS 5795*. Pp. 665–669. Springer Verlag, 2009. (slides).

[CWO07] A. L. Correa, C. Werner, M. de Oliveira Barros. An Empirical Study of the Impact of OCL Smells and Refactorings on the Understandability of OCL Specifications. In

Engels et al. (eds.), *MoDELS*. Lecture Notes in Computer Science 4735, pp. 76–90. Springer, 2007.

[Gho10] S. Ghosh (ed.). *Models in Software Engineering, Workshops and Symposia at MODELS 2009, Denver, CO, USA, October 4-9, 2009, Reports and Revised Selected Papers*. Lecture Notes in Computer Science 6002. Springer, 2010.

[GKB08] M. Gogolla, M. Kuhlmann, F. Büttner. A Benchmark for OCL Engine Accuracy, Determinateness, and Efficiency. In *MoDELS '08: Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*. Pp. 446–459. Springer-Verlag, Berlin, Heidelberg, 2008.

[MC99] L. Mandel, M. V. Cengarle. On the Expressive Power of OCL. In Wing et al. (eds.), *World Congress on Formal Methods*. Lecture Notes in Computer Science 1708, pp. 854–874. Springer, 1999.

[MR09] S. Moisan, J.-P. Rigault. Teaching Object–Oriented Modeling and UML to Various Audiences. Pp. 40–54 in [Gho10].

[OMG09] OMG. OMG Unified Modeling LanguageTM (OMG UML), Superstructure. version 2.2. February 2009. http://www.omg.org/spec/UML/2.2/Superstructure/PDF.

[OMG10] OMG. Object Constraint Language. OMG Available Specification. Version 2.2. Feb. 2010.

[Ric02] M. Richters. *A Precise Approach to Validating UML Models and OCL Constraints*. PhD thesis, Universität Bremen, 2002. BISS Monographs No. 14.

[SCC10] A. Sterritt, S. Clarke, V. Cahill. Precise Specification of Design Pattern Structure and Behaviour. In Kuehne et al. (eds.), *Modelling Foundations and Applications, 6th European Conference, ECMFA 2010*. LNCS 6138. Springer, 2010.

[VJ00] M. Vaziri, D. Jackson. Some Shortcomings of OCL, the Object Constraint Language of UML. In Li et al. (eds.), *TOOLS (34)*. Pp. 555–562. IEEE Computer Society, December 2000. http://dblp.uni-trier.de/db/conf/tools/tools34-2000.html#VaziriJ00.

[WK03] J. Warmer, A. Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA, Second Edition*. Addison-Wesley, 2003.