Insights on the Use of OCL in Diverse Industrial Applications

Shaukat Ali[1], Tao Yue[1], Muhammad Zohaib Iqbal[2,3], and Rajwinder Kaur Panesar-Walawege[1]

[1] Simula Research Laboratory, P.O. Box 134, Lysaker, Norway
[2] National University of Computer & Emerging Sciences, Islamabad, Pakistan
[3] SnT Luxembourg, Luxembourg
{shaukat,tao,rpanesar}@simula.no,
zohaib.iqbal@nu.edu.pk

Abstract. The Object Constraint Language (OCL) is a widely accepted language, standardized by OMG, for specifying constraints at various meta levels (e.g., meta-models and models). Despite its wide acceptance, there is a lack of understanding about terminology and purposes for which OCL can be used. In this paper, we aim to reduce this gap and provide guidance for applying OCL in practical contexts and we report our experience of applying OCL for different industrial projects in diverse domains: Communications and Control, Oil and Gas production, Energy Equipment and Services, and Recycling. Based on our experience, first, we unify the commonly used terminology in the literature for applying OCL in different ways for addressing diverse industrial problems. Second, we report the key results of the industrial application of OCL. Finally, we provide guidance to researchers and practitioners for choosing an appropriate meta level and purpose for their specific industrial problem at hand.

Keywords: Object Constraint Language, Industrial Applications, Constraint Solving, Constraint Parsing.

1 Introduction

The Object Constraint Language (OCL – http://www.omg.org/spec/OCL/2.3.1/) is the Object Management Group's (OMG) standard language for specifying constraints on models. Constraints can be specified at all the meta levels provided by the Meta-Object Factory (MOF – http://www.omg.org/mof/)—the OMG's framework for meta-modeling. Thus, constraints can be specified on meta-meta models (e.g., an implementation of MOF), meta-models (e.g., UML meta-model), customized profiles on meta-models (e.g., MARTE profile for UML – http://www.omgmarte.org/), and models (e.g., UML models).

The OCL has been used in industrial projects for various purposes, such as for configuration management in energy and maritime and seismic acquisition [1] and test case generation in communication and control [2, 3]. OCL is also being used as the

D. Amyot et al. (Eds.): SAM 2014, LNCS 8769, pp. 223–238, 2014.

language for writing constraints on models in many commercial Model-Based Testing (MBT) tools such as CertifyIt[1] and Fokus!MBT[2].

For the past several years, we have used OCL in several industry-driven research projects. The most significant projects include: model-based functional and robustness testing of embedded systems and communication and control systems, configuration of product lines of large-scale integrated control systems, and certification of subsea production systems according to safety standards. In this paper, we present our experience of applying the OCL in these domains. Our key findings are: 1) a small subset of OCL can be sufficient for a given industrial application; 2) specification and enforcement of constraints at the different MOF meta levels works in the same way; 3) evaluation of constraints was the most common purpose for the use of OCL. Based on our findings, we present guidelines for practitioners to choose the right meta level and purpose to apply OCL for their particular problem. Notice that all the definitions and discussions presented in this paper are within the context of our industrial applications, and may need to be adapted to other contexts.

The contributions of this paper can be summarized as follows: 1) clear and precise definitions of commonly used terminology related to the use of OCL; 2) a clear relationship among the different purposes (e.g., OCL solving and evaluation) that OCL can be used for; 3) key results from our industrial applications of OCL; 4) a detailed discussion that can guide practitioners in choosing when to apply OCL for a particular purpose and at which meta level. These contributions are aimed at reducing the gap between the academic understanding of OCL and its industrial application.

The rest of the paper is organized as follows: Section 2 presents various classifications of our OCL applications, Section 3 reports results from our industrial applications, Section 4 provides discussion, and Section 5 concludes the paper.

2 Classification of Various OCL Applications

This section provides an overview of our industrial applications (Section 2.1), definitions and examples in Section 2.2 and Section 2.3, and the relationships between various purposes for which OCL can be used in Section 2.3.

2.1 Overview

We use a conceptual model to discuss the overall picture of our experience of applying OCL in various projects (Fig. 1). We characterize our applications mainly from two aspects: 1) *Purpose* of applying OCL: e.g., *Constraint Solving* and *Constraint Evaluation*, and 2) *Meta Level*, at which OCL constraints are applied. Moreover, we discuss each application (e.g., *TestDataGeneration*) based on the type(s) of models on which OCL was used (e.g., *Structural* and *Behavioral* model) and the types of diagrams used for each type of model (e.g., UML class diagrams as structural models).

[1] http://www.smartesting.com/en/product/certifyit
[2] http://www.fokusmbt.com/index.html

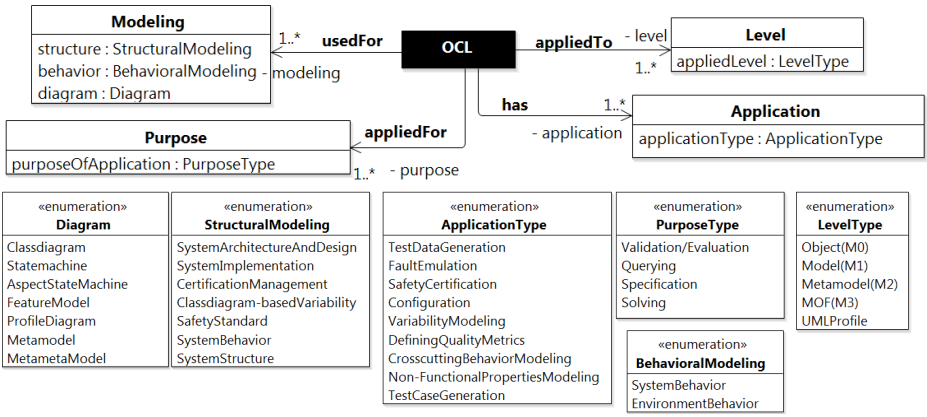


Fig. 1. Conceptual model of OCL applications

2.2 Definitions

This section presents definitions of the terms that we use in the rest of the paper.

Meta Levels. Meta-Object Facility (MOF) is a standard defined by Object Management Group (OMG) for model-driven engineering. MOF is designed as a four-level architecture, which allows modeling at four levels: meta-meta level (*M3*), meta level (*M2*), model level (*M1*), and Object level (*M0*). In other words, we define the term *Meta Levels* as a set of these four levels: *Meta Levels = {M3, M2, M1, M0}*.

Specification Levels. Specification levels are a subset of *Meta Levels*, on which OCL constraints can be specified: *Specification Levels = {M3, M2, M1}*.

Enforcement Levels. Enforcement levels are a subset of *Meta Levels*, at which OCL constraints are enforced (e.g., evaluated, solved). An enforcement level is one level lower than the level at which constraints are specified. It is defined as *Enforcement Level = {M2, M1, M0}*.

Purposes of Using OCL. In this section, we provide definitions and examples of the various purposes for which we have used OCL.

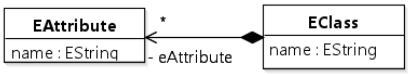
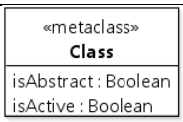
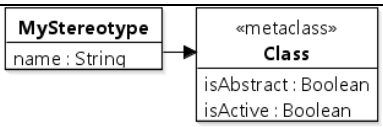
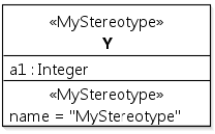
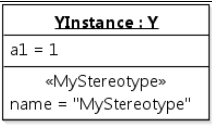
Constraint Specification (CSpec). Given a model *M* at one of the *Meta Levels*, *CSpec* means defining a constraint *C* on *M*. Based on the example given in the first row of Table 2, we define a constraint *((2/self.a1 > 0) and self.a2 > 0)* on class *X* (at *M1* level). We also show examples of OCL constraints at each meta level in Table 1. For example, in the third column of Table 1, we define a constraint on the definition of stereotype *MyStereotype (self.name = '')* in a profile diagram at the *M2* level.

Constraint Parsing (CP). Given a model *M* at one of the *Meta Levels* and a constraint *C* specified on *M*, *CP* means parsing *C* and obtaining an abstract syntax tree of *C* for further manipulation (e.g., calculating branch distances to generate test data from OCL constraints using a search algorithm [4]). An example of *CP* is shown in

the second row of Table 2, where an abstract tree of (*(2/self.a1 > 0) and self.a2 > 0*) is shown.

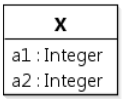
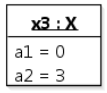
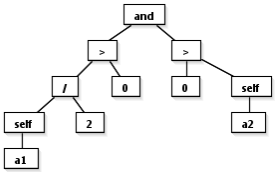
Constraint Evaluation/Validation (CE). Given a model *M* at one of the *Specification Levels*, an instance o_i of *M* at one level lower (belonging to *Enforcement Levels*) than the level of *M*, and a constraint *C* in OCL, *CE* means checking whether *C* is satisfied, dissatisfied, or results in an error situation by o_i. In OCL, satisfaction of a constraint means, the constraint is evaluated to *true*, dissatisfaction means the constraint evaluates to *false*, or *undefined* when a constraint results in an error situation. An example of *CE* is shown in the third row of Table 2, where a constraint is defined on a UML class *X* at the *M1* level and is evaluated on its three instances at the *M0* level. First instance (o_1), satisfies the constraint and hence it evaluates to *true*, the second instance (o_2) evaluates to *false*, and the third instance (o_3) evaluates to *undefined* since (*2/self.a1*) results in being divided by 0, and overall the constraint evaluates to *undefined*.

Table 1. Examples of OCL constraints at various levels

Level	Example	Constraint
M3		**Specification:** **context** EClass **inv**: self.name <> ''
M2		**Evaluation/Validation:** true
		Specification: **context** Class **inv**: self.isActive
		**Specification:** **context** MyStereotype **inv**: self.name = ''
M1		**Evaluation/Validation:** false
		Specification: **context** EClass **inv**: self.a1 > 0
M0		**Evaluation/Validation:** true

Constraint Solving (CSolv). Given a model *M* at one of the *Specification Levels*, an instance *O* of *M* at one meta level lower than *M*, i.e., belonging to *Enforcement Levels*, and a constraint *C* defined on *M*, *CSolv* means finding at least one instance of *M*, which evaluates *C* to be *true*, *false*, or *undefined*. An example of *CSolv* is shown in Table 2, row 5. Given the constraint *C= (2/self.a1 > 0) and self.a2 > 0* defined on class *X* (at *M1* level), a constraint solver provides an instance o_i (at *M0* level) satisfying *C*. In this particular example, o_i can be instance *x4* shown in the table. Notice that the constraint solver may provide multiple instances depending on the application.

Table 2. Examples of various purposes of OCL

Example	Model (M)	Instance (o_1)	Instance (o_2)	Instance (o_3)
A class *X* with two Integers *a1* and *a2*, and with three instances available: *x1*, *x2*, and *x3*.				
Specification	**context** X **inv**: (2/**self**.a1 > 0) **and** **self**.a2 > 0	N/A		
Parsing				
Evaluation/Validation		true	false	undefined
Solving				
Querying				

OCL Querying (OQ). Given a model *M* at one of the *Specification Levels*, a set of its instances $O = \{o_1, o_2, o_3, ..o_n\}$ at one meta level lower (belonging to *Enforcement Levels*), OCL querying *OQ* returns one or more instances of *M*, which satisfy the constraint specified in *OQ*. An example is shown in Table 2, row 6, where given a constraint *C= (2/self.a1 > 0) and self.a2 > 0* and a set of instances $O = \{o_1, o_2, o_3\}$, a constraint querying returns instances from *O* that satisfies *C*. In this particular example, such an instance is o_1 *(x1)*.

2.3 Relationships between Various Purposes of Using OCL

OCL Querying. Fig. 2 shows the relationship among OCL querying, evaluation, and specification. The first step is specification of a constraint *C* at *M3*, *M2*, or *M1* level. A query in OCL then returns a model at one meta level lower (*M2*, *M1*, or *M0*) level using OCL evaluation.

OCL Solving. Fig. 3 shows the relationship of how OCL solving is related to the purposes for which OCL can be used. The first step in OCL solving is to specify a constraint *C* on a model at *M3*, *M2*, or *M1* level. OCL solving then starts with a random instance of a model at one meta level lower, i.e., *M2* (*C* is specified at *M3*), *M1* (*C* is specified at *M2*) or *M0* (*C* is specified at *M1*) level. This instance is then evaluated using OCL evaluation. If the instance satisfies *C* the OCL solving stops. Otherwise OCL solving is guided towards another instance (using OCL parsing and OCL

querying) using for example a search algorithm (see [4] for details) and a new instance is generated, which is again evaluated by OCL evaluation. OCL solving continues until an instance is found that satisfies *C*.

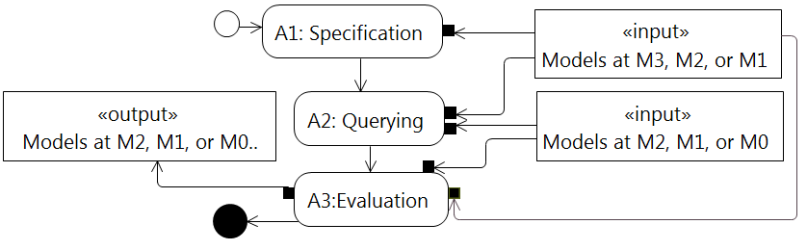


Fig. 2. OCL querying

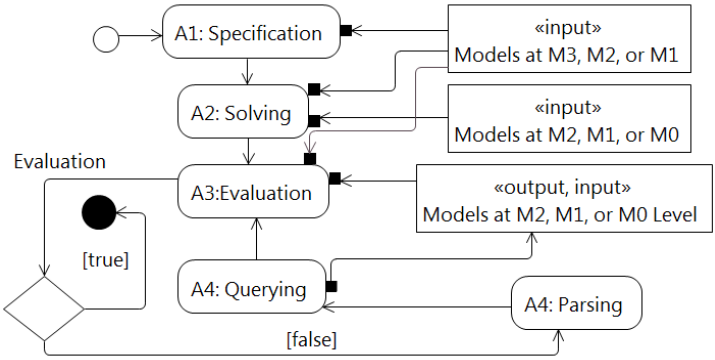


Fig. 3. OCL solving

3 Industrial Applications

In this section, we present our industrial applications of OCL based on the concepts and definitions presented in Section 2.

3.1 Model-Based Testing of Video Conferencing Systems

In this section, we discuss six applications of OCL, which are related to testing a commercial Video Conferencing System (VCS) developed by Cisco Systems.

Case Study Description. Our first case study is a VCS called *Saturn* developed by Cisco Systems Inc, Norway. The core functionality of *Saturn* manages establishing and disconnecting video conferences. In total, *Saturn* consists of 20 subsystems such as audio/video subsystems [5]. Each subsystem can run in parallel to the subsystem implementing the core functionality. *Saturn's* implementation consists of more than three million lines of C code. Our second case study is about a product line of VCSs

called *Saturn Product Line*, developed in Cisco Systems Inc, Norway. The Saturn family consists of various hardware codecs ranging from C20 to C90. C20 is the lowest end product with minimum hardware and has lowest performance in the family.

Table 3. Mapping of applications to various aspects of OCL

#	Application	Case Study	Model Elements	#Constraints	Constructs/ Operations	Types of Attributes
A1	Test Data Generation	VCS	Guards	144	-	Enumeration, Integer, Boolean, String
		MSM	(Guards, Change Events)	(11, 3)	select, forAll, implies, oclInState	Integer, Boolean, String, Enumeration, NFP_Real
		BRE		(11,1)		
A2	Test Oracle Generation	VCS	State Invariants	100	select, collect, forAll, exists, includes, excludes	Enumeration, Integer, Boolean, String
		MSM/ BRE	Guards	3	select, forAll, oclInState	Integer, Boolean, NFP_Real
A3	Fault Emulation	VCS	Change Events	57	select, collect	Enumeration, Integer, Boolean, NFP_Real, NFP_Percentage
A4	Crosscutting Behavior Modeling	VCS	Change Events	57	select, collect	Enumeration, Integer, Boolean, NFP_Real, NFP_Percentage
			State Invariants	10	-	Enumeration, Integer, Boolean, NFP_Real, NFP_Percentage
A5	Specifying Non-Functional Properties	VCS	Pointcuts	12	-	Enumeration, Integer, Boolean, String
			Advice	144	-	Enumeration, Integer, Boolean, String
A6	Variability Modeling	VCS	Configuring UML State Machine	52	select, forAll, exists, includes, excludes	Enumeration, Integer, Boolean, String
			Configuring Aspect State Machines	44	select, forAll, exists, includes, excludes	Enumeration, Integer, Boolean, String, NFP_Real, NFP_Percentage
A7	Safety Certification	SPCS	Stereotypes	218	select, collect, forAll, exists, includes	No variables used.
A8	Configuration	SCM	Package, Stereotype, Class, TemplateSignature, Dependency	6	select, forAll, allInstances,	Integer, Boolean, String

Problem Description. The first problem in this project is about supporting automated, model-based robustness testing of *Saturn*. *Saturn* should be robust enough to handle the possible abnormal situations that can occur in its operating environment and invalid inputs. For example, *Saturn* should be robust against hostile environment conditions (regarding the network and other communicating VCSs), such as high percentage of packet loss and high percentage of corrupt packets. Such behavior is very important for a commercial VCS and must be tested systematically and automatically to be scalable. More details on the robustness behavior of *Saturn* and its modeling can be found in [5]. The second problem in this project emerged while working with model-based robustness testing discussed in the last paragraph. We wanted to significantly reduce the amount of modeling effort required for MBT by devising a product line modeling and configuration methodology since Video Conferencing Systems (VCSs) are product lines.

Objectives. 1) *Test Data Generation (A1)* aims to solve OCL constraints to generate data required to generate executable test cases. 2) *Test Oracle Generation (A2)* has the objective of evaluating OCL constraints to determine if the execution of a test case passed or failed. 3) *Fault Emulation in Environment (A3)* is to solve OCL constraints defined on the environment of a real-time embedded system with the goal of generating the data that violates the constraints so that various faults can be emulated in the environment to test the robustness of a system. 4) *Specifying Non-Functional Properties (NFPs) with MARTE (A4)* aims to specify constraints on NFPs defined in the UML MARTE profile using OCL. 5) *Crosscutting Behavior Modeling (A5)* was proposed to model crosscutting behavior using Aspect State Machine (ASM) [2, 5, 6]. OCL queries are used to model Pointcuts [7] (a feature in Aspect-Oriented Modeling)— modeling elements of a standard UML state machine, on which an ASM should be weaved. 6) *Behavioral Variability Modeling (A6):* The objective of this application is to model and resolve various types of variability that exist in UML state machines with the ultimate aim of reducing the modeling effort required for MBT of different products in a product line.

Solution. Saturn consists of 20 subsystems. To model the functional behavior, for each subsystem, we modeled a class diagram to capture APIs and state variables. In addition, we modeled one or more state machines to capture the behavior of each subsystem. On average each subsystem has five states and 11 transitions, with the biggest subsystem having 22 states and 63 transitions. Note that, though an individual subsystem may not look complex in terms of number of states and transitions, all subsystems run in parallel to each other and therefore the space of system states and possible execution interleaving are very large.

Saturn's robustness behavioral models consist of five aspect class diagrams and five aspect state machines. An ASM is a UML state machine extended with a UML profile for AOM called AspectSM [5]. The largest ASM specifying robustness behavior has three states and ten transitions, which would translate into 1604 transitions in standard UML state machines without having AspectSM applied. The modeling of ASM is systematically derived from a fault taxonomy [5] categorizing different types of faults (faults in the environment such as communication medium and media

streams that lead to faulty situations in the environment). Each ASM has a corresponding aspect class diagram modeling different properties of the environment using the MARTE profile, whose violations lead to faulty situations in the environment.

Saturn Product Line family also consists of 20 subsystems and each subsystem has at least one configurable state machine specifying its functionality and on average such state machine has five states and 11 transitions. Saturn product line family models also consist of 124 hardware configuration parameters and 99 software configuration parameters.

Results. Table 3 provides a summary of the key results of applying OCL for all the applications of all the projects. For each application, we report on which model elements OCL was specified and how many constraints were there in our industrial case studies. In addition, for each application we provide OCL constructs and operations used and also types of attributes used in the constraints. For example, for *A3*, we modeled 57 change events with OCL *Select* and *Collect* operations. In addition, we used attributes of types: *Enumeration*, *Integer*, *Boolean*, and a couple of *NFPs* from MARTE. In all the applications, we used relational and logical operations, and hence we do not mention them explicitly in the table.

3.2 Safety Certification

Case Study Description. This case study concerns the certification of the software used in a subsea production control system (SPCS) developed by a large energy company in Norway. SPCS is a complex safety-critical system consisting of a myriad of equipment types. An oil field consists of subsea oil wells that have an assembly of control valves, pressure gauges and chokes attached to them that control the flow of oil. These are all housed on a structure called a template attached to which is a system of steel tubes, electrical and fiber optic cables that transport power and communication signals from the surface to the subsea equipment. Finally there is equipment to carry the oil to the surface. SPCS controls this entire system by sending and receiving data between the surface and the subsea equipment thus allowing the engineers at the surface to control and monitor the sub-sea equipment.

Problem Description. SPCS are subject to various industry and governmental regulations and undergo a process of certification by a third-party certification. In our case the SPCS was subject to a certification process against the IEC61508 standard for electrical, electronic, or programmable electronic systems that are used in safety-critical environments. The supplier of the system provides evidence that the system is compliant with the criteria set in the requisite standard. Hence, there should be a consistent interpretation of the standard being used by all parties involved. Without this explicit interpretation there can be problems between the certifier and the supplier due to the variance that exists. A systematic procedure is also needed for creating the necessary evidence, such that the supplier can properly interpret the standard in the context of its application domain and verify whether sufficient evidence exists to satisfy all the requirements of the standard [8].

Objective. *Certification Standards Modeling (A7).* The objective of using OCL is to assist system suppliers in establishing a relationship between a domain model of a safety-critical application and the evidence model of a certification standard.

Solution and Results. A conceptual model of the evidence requirements of a safety standard is created. This conceptual model is used as the basis for a UML profile of the standard. The UML profile is used for stereotyping the elements of a domain model of the system to be certified. When a stereotype from the profile is applied to a domain model element, it shows how that element fulfills the requirements from the standard. OCL constraints are added to the stereotypes to ensure certain properties of the stereotypes as well as to guide system developers in refining the domain model. When the OCL constraints associated with a stereotype are validated, they will start the guidance process for augmenting the domain model with other stereotypes. This may require the domain model to be updated so that the stereotype constraints are satisfied. Table 3 summarizes our results of applying OCL for certification in row A7.

3.3 Architecture Variability Modeling for Supporting Automated Product Configuration

Case Study Description. This case study is a product line of subsea control modules (SCMs) developed by FMC Technologies, Norway. SCMs control all the equipment and services located in the subsea, but communicates (via Network) with the topic control units. SCMs are deployed with software, which can be configured differently according to customers' requirements, some of which include environment factors (e.g., depth of the seabed), to control the subsea wells. An SCM contains subsea electronic modules, software applications deployed on them, and mechanical and electrical devices that are controlled and monitored by the software. The software application deployed to the control modules is configured mainly based on the number, type, and details of devices (e.g., sensors) connected to and controlled by the subsea electronic module on which the software application is deployed.

Problem Description. Integrated Control Systems (ICSs) are typically large-scale, highly configurable systems of systems such as SCMs. Such systems consist of large number of subsystems typically geographically distributed and connected through network. A family of ICSs share the same software code base, which is configured differently for each product to form a unique installation and, therefore, a large number of interdependent variability points are introduced by both hardware and software components. Due to the complexity of such systems and inadequate automation support, product configuration is typically error-prone and costly, and therefore an automated product configuration support is needed.

Objective. This application is about specifying the guidelines as OCL constraints for the purpose of automated product configuration in the context of ICSs (*A8*).

Solution and Results. We developed a UML-based product line modeling methodology (named as SimPL) that provides a foundation for supporting semi-automated product configuration in the specific context of ICSs [9]. The SimPL profile together

with inherent features of UML (i.e., templates and packages) enables comprehensive modeling of variability points, tracing variability points to software and hardware model elements, and grouping and hierarchically organizing the variability points. As part of the SimPL methodology, we defined guidelines for modeling each view (e.g., software view, hardware view). To guide users through the process of applying SimPL, a modeling environment was constructed to automatically enforce six OCL constraints that correspond to these guidelines. Table 3 summarizes our results of applying OCL for specifying and evaluating constraints that correspond to modeling guidelines proposed as part of SimPL (Row A8).

3.4 Environment Model-Based Testing

Case Study Description. We apply the environment model-based testing to two industrial case studies. The first case study from WesternGeco is of a very large and complex control system for marine seismic acquisition. The system controls tens of thousands of sensors and actuators in its environment. The timing deadlines on the environment are in the order of tenths of seconds. The system was developed using Java. The second case study is an automated bottle-recycling machine developed by Tomra AS. The system under test (SUT) was an embedded device 'Sorter', which was responsible to sort the bottles into their appropriate destinations. The system communicated with a number of components to guide recycled items through the recycling machine to their appropriate destinations. It is possible to cascade multiple sorters with one another, which results in a complex recycling machine. The SUT was developed using C. Both the systems are Real-Time and Embedded Systems (RTESs) and were running in environments that enforce time deadlines in the order of tenths of seconds with acceptable jitters of a few milliseconds in response time.

Problem Description. RTESs typically work in environments comprising large numbers of interacting components. The interactions with the environment can be bound by time constraints. Violating such time constraints, or violating them too often for soft real-time systems, can lead to serious failures leading to threats to human life or the environment. For effective testing of industrial scale RTESs, systematic automated testing strategies that have high fault revealing power are essential. The system testing of RTESs requires interactions with the actual environment. Since the cost of testing in real conditions tends to be high, environment simulators are typically used for this purpose. For the industrial systems of WesternGeco and Tomra, we applied one such approach for black-box system level testing based on the environment models of the systems. These models were used to generate an environment simulator [10, 11], test cases, and obtain test oracles [3]. For test case generation, we applied various testing strategies, including search-based testing [12], adaptive random testing [13], and a hybrid approach combining these two strategies [12].

Objectives. *1) Test Data Generation (A1).* The objective of this application is to generate test data by solving OCL constraints in order to reach states in the environment that represent a failure of the SUT (the "error" states). *2) Test Oracle Generation*

(A2). The objective of this application is to evaluate OCL constraints to determine if the execution of a test case reached the "error" states or not.

Solution and Results. For the purpose of environment model-based testing, the environment of the SUT was modeled using our proposed UML & MARTE Real-time Embedded systems Modeling Profile (REMP) [14]. REMP provided extension to the standard UML class diagram and state machine notations, and used the MARTE profile for modeling timing details and non-deterministic events. The models developed were constrained by OCL for the purposes mentioned in the previous section. The structural details of an RTES environment were modeled as an environment domain model, which captures the information of various environment components, their properties, and their relationships. The behavioral details of the environment were modeled using the state machine notation annotated with REMP. Such state machines contain information of the nominal behavior of the components, their robustness behavior (e.g., breakdown of a sensor), and "error states" that should never be reached (e.g., hazardous situations). Table 3 summarizes the results of applying OCL in our context (rows A1 & A2).

4 Overall Discussion

In this section, we provide an overall discussion together with guidelines for practitioners based on our experience of applying OCL.

4.1 Selecting a Subset of OCL

From Table 4, we can see that in most of the applications, *select*, *collect*, and *forAll* were the most frequently used operations. Based on this observation, we can conclude that even though OCL provides a rich collection of constructs and operations, in practice the complete specification is not usually required. This means that for applying OCL in industrial applications one can select a well-defined subset of OCL that is sufficient to serve a required purpose. Note that this is similar to the use of a subset of UML and MARTE in practice as suggested in [1]. This also means that less training is required to teach the subset of OCL, which aids its adoption in industry.

4.2 Choosing a Meta Level

From the last column in Table 4, we can see that six out of eight applications are related to MBT, all of which required specifying constraints at M1 and enforcing these at M0. This observation is perfectly explainable because when dealing with test case generation we are very close to the system/software design and implementation (low level of abstraction). Recall that constraints specified at M1 correspond to the actual system variables of the design or implementation while at the M0 level these constraints are enforced based on the runtime values of the variables.

Table 4. Mapping of OCL applications to various purposes and meta level*

App.	Industry	Case Study	Domain	Modeling	Diagrams	Purpose	(Spec., Enf.)
A1	CCS, EES, REC	VCS, MSM, BRE	RTES	System Behavior, System Structure	CDs & SMs	CSolv	(M1, M0)
A2	CCS, EES, REC	VCS, MSM, BRE	RTES	System Behavior, System Structure	CDs & SMs	CE	(M1, M0)
A3	CCS	VCS, MSM, BRE	RTES	Environment Behavior	CDs, SMs, & ASMs	CE, CSolv	(M1, M0)
A4	CCS	VCS	RTES	System Behavior, System Structure, Environment Behavior	CDs & SMs	OQ	(M1, M0)
A5	CCS	VCS	RTES	System Behavior, System Structure, Environment Behavior, Architecture	CDs	CSolv, CE	(M1, M0)
A6	CCS	VCS	RTES	Class Diagram-based, State Based Variability	CDs & SMs	CSolv	(M1, M0)
A7	OGP	SPCS	ICS, RTES	Safety Standard	Profile, CDs	CE	(M2,M1)
A8	OGP	SCM	ICS, RTES	Architecture	CDs	CE	(M2,M1)

* CCS: Communication and Control System, EES: Energy Equipment and Services, REC: Recycling, OGP: Oil and Gas Production, VCS: Video Conferencing System, MSM: Marine Seismic Acquisition, BRE: Bottle Recycling, SPCS: Subsea Production Control System, RTES: Real-Time Embedded System, ICS: Integrated Control System, CD: Class Diagram, SM: State Machine, ASM: Aspect State Machine, Profile: UML Profile, SCM: Subsea Control Module, MM: Metamodel

For A7 and A8, as we were dealing with UML profiles, therefore we specified the constraints at the M2 level and these were enforced at the M1 level. Notice that in these two applications, our problems were at a higher meta level than implementation, i.e., architecture and design modeling of product lines for supporting configuration (A7) and standard modeling for supporting safety certification (A8). In these two cases, the resulting models to which the profiles were applied were UML class diagrams, which are at the M1 level.

Based on the above observations, we can conclude that constraint specification and enforcement at all applicable levels works in the same way (i.e., specified at one level and enforced in one level lower) and with pretty much the same set of OCL constructs. The only challenge, as far as we can see, is to select a right meta level for specifying constraints, which heavily depends on the problem to be solved. If the problem is related to the implementation, the most appropriate meta level is the pair (M1, M0) as is the case for (A1-A6). If we are dealing with UML profile, the obvious choice is to specify constraints at the M2 level and they will be automatically enforced at the profiled M1 level models. Moreover, the specification at the highest meta level (M3) is needed to enforce constraints at the M2 level, which is commonly used to define meta-models. This is suggested when there is a need in a particular

industry to define a new MOF-based domain specific language to solve a particular problem in hand.

4.3 Choosing Diagram

In all our applications, class diagrams were used as the basis for modeling attributes that required specifying OCL constraints. In addition, for the applications where behavior was required to be modeled, we used state machines as our case studies exhibit state-based behavior. Of course, other behavioral diagrams (e.g., sequence diagrams) can also be used in other contexts. Based on this observation, we can then conclude that though choosing an appropriate diagram depends on application contexts; however at a minimum a UML class diagram representing various concepts required at various meta levels is needed to hold attributes required for specifying OCL constraints. Moreover, choosing a particular diagram does not impact what OCL constructs are applied and which meta level to use.

4.4 Selecting a Purpose of OCL

In our applications, the most common use of OCL was to perform evaluation (6 out of 8 applications) followed by solving (4 out of 8). In addition, recall that specification of constraints is required in solving, evaluating, parsing, and query as we discussed in Section 2. This observation can be explained from the fact that to support automation, e.g., test data generation, the specified constraints are required to be evaluated and/or solved. Of course, if an application is only for the purpose of bringing additional precision to models, specification of constraints is sufficient. Notice that as we discussed in Section 2.3., the most important step is OCL evaluation as it is also required for OCL solving and thus suggesting that OCL evaluation is at the core of any automated constraints manipulation activity. This is the reason that a wide variety of OCL evaluators exist, such as OCLE 2.0 [15], OSLO [16], IBM OCL parser [17], and EyeOCL Software (EOS) evaluator [18]. In all our applications except A7 and A8, we chose EOS as it is one of the most efficient evaluators for OCL. Notice that for A4 and A9, where we used OCL for querying, we again used EOS. For A7 and A8, we used the OCL evaluator built-in in IBM Rational Software Architect, because it has a good support for enforcing the constraints specified on UML profiles on M1 level models.

Several OCL solvers exist in the literature that translate OCL into other formalisms [19-24] such as Alloy and Satisfiability Problem (SAT) to solve them. In our industrial applications, we developed our own OCL Solver called EsOCL [4] based on search algorithms since the existing solvers either did not handle important features of OCL such as collections or their operations [19, 20], were not scalable, or lacked proper tool support [21].

5 Conclusion

This paper presents our experiences of applying the Object Constraint Language (OCL) on six industrial case studies. The case studies belong to diverse industrial

domains including Communication and Control, Energy Equipment and Services, Recycling, and Oil and Gas Production. In these case studies, OCL is applied solving various industrial problems including model-based testing, safety certification, and automated product configuration. The results of the industrial case studies showed that a well-selected subset of OCL notations was sufficient for various problems for various purposes including constraint evaluation, solving, and querying. We found that OCL constraint specification and enforcement at various meta levels of MOF works in the same way, i.e., specified at M_x level and enforced at M_{x-1} where x={1, 2, 3}. OCL evaluation is a fundamental activity and is the core of all our industrial applications. Based on our findings, we presented guidelines for practitioners that can help them choose an appropriate purpose of OCL and meta level.

Acknowledgments. Muhammad Zohaib Iqbal was partly supported by ICT R&D Fund, Pakistan (ICTRDF/MBTToolset/2013) and by National Research Fund, Luxembourg (FNR/P10/03).

References

1. Iqbal, M.Z., Ali, S., Yue, T., Briand, L.: Experiences of Applying UML/MARTE on Three Industrial Projects. In: France, R.B., Kazmeier, J., Breu, R., Atkinson, C. (eds.) MODELS 2012. LNCS, vol. 7590, pp. 642–658. Springer, Heidelberg (2012)
2. Ali, S., Briand, L., Arcuri, A., Walawege, S.: An Industrial Application of Robustness Testing using Aspect-Oriented Modeling, UML/MARTE, and Search Algorithms. In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 108–122. Springer, Heidelberg (2011)
3. Arcuri, A., Iqbal, M., Briand, L.: Black-Box System Testing of Real-Time Embedded Systems Using Random and Search-Based Testing. In: Petrenko, A., Simão, A., Maldonado, J.C. (eds.) ICTSS 2010. LNCS, vol. 6435, pp. 95–110. Springer, Heidelberg (2010)
4. Ali, S., Iqbal, M.Z., Arcuri, A., Briand, L.: Generating Test Data from OCL Constraints with Search Techniques. IEEE Trans. Softw. Eng. 39(10), 1376–1402 (2013)
5. Ali, S., Briand, L.C., Hemmati, H.: Modeling Robustness Behavior Using Aspect-Oriented Modeling to Support Robustness Testing of Industrial Systems. Software and Systems Modeling 11(4), 633–670 (2012)
6. Ali, S., Yue, T., Briand, L.C.: Does Aspect-Oriented Modeling Help Improve the Readability of UML State Machines? Software & Systems Modeling 13(3), 1189–1221 (2014)
7. Laddad, R.: AspectJ in Action: Practical Aspect-Oriented Programming. Manning Publications (2003)
8. Panesar-Walawege, R.K., Sabetzadeh, M., Briand, L.: Supporting the verification of compliance to safety standards via model-driven engineering: Approach, tool-support and empirical validation. Information and Software Technology 55(5), 836–864 (2013)
9. Behjati, R., Yue, T., Briand, L., Selic, B.: SimPL: A Product-Line Modeling Methodology for Families of Integrated Control Systems. Information and Software Technology 55(3), 607–629 (2013)
10. Iqbal, M.Z., Arcuri, A., Briand, L.: Code Generation from UML/MARTE/OCL Environment Models to Support Automated System Testing of Real-Time Embedded Software. Simula Research Laboratory, Technical Report (2011-04) (2011)

11. Iqbal, M.Z., Arcuri, A., Briand, L.: Environment modeling and simulation for automated testing of soft real-time embedded software. Softw Syst. Model. 1–42 (2013)
12. Iqbal, M.Z., Arcuri, A., Briand, L.: Combining search-based and adaptive random testing strategies for environment model-based testing of real-time embedded systems. In: Fraser, G., Teixeira de Souza, J. (eds.) SSBSE 2012. LNCS, vol. 7515, pp. 136–151. Springer, Heidelberg (2012)
13. Iqbal, M.Z., Arcuri, A., Briand, L.: Automated System Testing of Real-Time Embedded Systems Based on Environment Models. Simula Research Laboratory, Technical Report (2011-19) (2011)
14. Iqbal, M.Z., Arcuri, A., Briand, L.: Environment Modeling with UML/MARTE to Support Black-Box System Testing for Real-Time Embedded Systems: Methodology and Industrial Case Studies. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010, Part I. LNCS, vol. 6394, pp. 286–300. Springer, Heidelberg (2010)
15. Chiorean, D., Bortes, M., Corutiu, D., Botiza, C., Cârcu, A.: OCLE. (September 2009), http://lci.cs.ubbcluj.ro/ocle/
16. Hein, C., Ritter, T., Wagner, M.: Open Source Library for OCL (2009)
17. Drusinsky, D.: Modeling and Verification using UML Statecharts: A Working Guide to Reactive System Design, Runtime Monitoring and Execution-based Model Checking. Newnes (2006)
18. Egea, M.: EyeOCL Software (September 2009), http://maude.sip.ucm.es/eos/
19. Aertryck, L.V., Jensen, T.: UML-Casting: Test synthesis from UML models using constraint resolution. Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL 2003) (2003)
20. Benattou, M., Bruel, J., Hameurlain, N.: Generating test data from OCL specification. In: Proceedings of the Workshop:Workshop on Integration and Transformation of UML Models at ECOOP 2002 (WITUML) (2002)
21. Bao-Lin, L., Zhi-shu, L., Qing, L., Hong, C.Y.: Test case automate generation from UML-sequence diagram and OCLexpression. In: International Conference on Computational Intelligence and Security, pp. 1048–1052 (2007)
22. Clavel, M., Dios, M.A.G.D.: Checking unsatisfiability for OCL constraints. In: Proceedings of the Workshop: The Pragmatics of OCL and Other Textual Specification Languages at MoDELS 2009, Electronic Communications of the EASST, vol. 24 (2009)
23. Kyas, M., Fecher, H., Boer, F.S.D., Jacob, J., Hooman, J., Zwaag, M.V.D., Arons, T., Kugler, H.: Formalizing UML Models and OCL Constraints in PVS. Electron. Notes Theor. Comput. Sci. 115, 39–47 (2005)
24. Brucker, A.D., Krieger, M.P., Longuet, D., Wolff, B.: A specification-based test case generation method for UML/OCL. In: Dingel, J., Solberg, A. (eds.) MODELS 2010. LNCS, vol. 6627, pp. 334–348. Springer, Heidelberg (2011)