MDT OCL Goes Generic

Introduction to OCL and Study of the Generic Metamodel and API

Christian W. Damus
IBM™ Rational™ Software
OCL Committer
Agenda

• Introduction to OCL
• Case Study: Problem Description
• Case Study: Generic OCL Implementation
• Reflections and Future Work
• Summary
• Q & A
About the Object Constraint Language

- OCL is designed to be a simple language for specification of constraints and queries in models, accessible to non-programmers
  - Relatively small grammar but rich expressiveness

- A member of the Unified Modeling Language™ (UML®) family of specifications at OMG

- OCL expressions have no side-effects
  - No assignment semantics
  - All data types are immutable
  - Can be used to specify side-effects, depending on the placement of OCL in the model

- Tightly coupled to UML and the Meta-Object Facility (MOF™)
Constraints and Queries

• Invariant constraints

\[
\text{context Borrower} \\
\text{inv max_fine: self.loan->}\text{collect(} \\
\text{overdueDays() * rate())-}>\text{sum()} \leq 30.0
\]

-- © Copyright 2008 IBM Corp. All rights reserved.
-- This source code is made available under the terms of the Eclipse Public License, v1.0

• Operation pre-conditions

\[
\text{context Library::lend(b : Borrower, c : ItemCopy) : Loan} \\
\text{pre circulates: c.item.isCirculating()} \\
\text{pre undamaged: not c.isDamaged} \\
\text{pre not_overdue: b.loan->}\text{forall(overdueDays() = 0)}
\]

-- © Copyright 2008 IBM Corp. All rights reserved.
-- This source code is made available under the terms of the Eclipse Public License, v1.0
Constraints and Queries

• Operation post-conditions

    context Library::lend(b : Borrower, c : ItemCopy) : Loan
    post borrowed: b.borrowed = b.borrowed@pre->including(c)
    post new_loan: result.oclIsNew() implies Loan^make(b, c)
    post loan: result.overdueDays() = 0 and
                result.borrower = b and
                result.copy = c

-- © Copyright 2008 IBM Corp.  All rights reserved.
-- This source code is made available under the terms of the Eclipse Public License, v1.0
Constraints and Queries

Query:

context Library:
-- find all books in the library that are not available to be
-- borrowed, together with the number of copies that are
-- currently out and the number that are damaged

def unavailableBooks() : Set(Tuple(book : Book,
                                        out : Integer,
                                        damaged : Integer)) =

    let unavailable : Set(Book) =
        self.book->select(b : Book |
                           b.copy->forAll(borrowedBy->notEmpty() or isDamaged)) in
    unavailable->collect(b | Tuple{
                               book = b,
                               out = b.copy->select(borrowedBy->notEmpty())-size(),
                               damaged = b.copy->select(isDamaged)-size()})

-- © Copyright 2008 IBM Corp.  All rights reserved.
-- This source code is made available under the terms of the Eclipse Public License, v1.0
Demo

• Quick demonstration of querying a UML model using OCL
EMFT OCL 1.0 Overview

- MDT OCL’s first release is in the EMFT Project
- Metamodel and related APIs are based on Ecore
1.0 API: Metamodel

• Based on early drafts of the OCL 2.0 Abstract Syntax Model (AST) specification

• Includes implementations of the standard Types and Expressions packages
  • Types defines several metaclasses for specialized classifiers
  • Expressions defines the structure of an OCL expression

• Adds packages to model Queries, Utilities, and UML constructs that have no analogues in Ecore
1.0 API: Metamodel

- Types package
  - The AnyType metaclass extends EClassifier; its sole instance is OclAny, providing the top of OCL's lattice-structured type hierarchy
  - The VoidType metaclass extends EClassifier; its sole instance is OclVoid, providing the bottom of the lattice; at this time, the void literal is named null
  - The CollectionType metaclass extends EDataType with an elementType : EClassifier association
  - The MessageType metaclass extends EClass with mutually exclusive referredOperation : EOperation and referredSignal : Signal associations, and owning attributes corresponding to the operation parameters or signal properties
  - The TupleType metaclass extends EClass defining records, whose attributes are named parts
1.0 API: Metamodel

- Types package (cont’d)

  - The PrimitiveType metaclass extends EDataType; its subtypes PrimitiveString, PrimitiveReal, PrimitiveInteger, PrimitiveBoolean have instances String, Real, Integer, Boolean, respectively
1.0 API: Metamodel

• Expressions package

  • The OclExpression abstract metaclass extends TypedElement (see below) and is the root of a rich expression hierarchy, including operation-call, property-call, iteration, conditional, and literal expressions. OclExpression has a property type: EClassifier, the static result type of the expression

  • The Variable metaclass also is a TypedElement, optionally representing an EOperation EParameter (as in pre-condition constraints)

  • The MessageExp defines message expressions and has mutually exclusive associations calledOperation : CallOperationAction and sentSignal : SendSignalAction

  • The TupleLiteralExp defines tuple literal expressions comprising a list of TupleLiteralPart that look like Variables except that they represent properties of the TupleType
1.0 API: Metamodel

- Expressions package (cont’d)
1.0 API: Metamodel

- UML package
  - The `TypedElement` abstract metaclass represents the UML construct of the same name; Ecore's `ETypedElement` is not used
  - The `Constraint` metaclass represents the UML construct of the same name
  - The `CallOperationAction` and `SendSignalAction` metaclasses are required by message expressions. They are peculiar in the OCL for being the only constructs from the UML's Actions package that it uses.
1.0 API: Metamodel

• Utilities package
  • A number of interfaces (in the Ecore sense) defining services required by the OCL parser
  • The PredefinedType interface is extended by all of the metaclasses in the Types package whose instances are the types in the OCL Standard Library. It provides operations such as:
    ▸ getOperations() – obtains the EOperations of a standard library type
    ▸ getRelationshipTo(EClassifier) – tests type conformance
  • The Visitable interface mixes an accept(Visitor) operation into the Expressions metaclasses for AST visitation
  • The ASTNode interface its subinterfaces mix in textual location information into the Expressions metaclasses
1.0 API: Metamodel

• Query package

  • Defines a single Query metaclass with a reference to an expression: OclExpression
  • The generated QueryFactory has additional operations that construct a Query by parsing the concrete syntax (text) of an OCL expression
1.0 API: Environment

- The Environment interface is loosely based on the environment construct defined by OCL
  - Provides resolution (look-up) of elements in the model that are referenced by name in OCL expressions
  - Records variable definitions
- Nested environments define nested scopes
  - Outer scope is classifier context
  - Inner scopes are operation contexts, let expressions, etc.
1.0 API: Parsing and Evaluation

- Create a query or a constraint:

Query query = QueryFactory.createQuery(
    "package library context Borrower " +
    "inv: self.loan->select(overdueDays() > 0).borrowed.item " +
    "endpackage");

Query constraint = QueryFactory.createQuery(
    "package library context Borrower " +
    "inv: self.loan->collect(overdueDays()*rate())->sum() <= 30.0 " +
    "endpackage");

- Evaluate it:

System.out.printf("Overdue items: %s", query.evaluate(borrower));
if (constraint.check(borrower))
    System.out.printf("Fine not over the maximum.");
System.out.printf("Borrowers over maximum fine: %s",
    constraint.reject(allBorrowers));

/* © Copyright 2008 IBM Corp. All rights reserved.  
This source code is made available under the terms of the Eclipse Public License, v1.0 */
Agenda

• Introduction to OCL

• **Case Study: Problem Description**

• Case Study: Generic OCL Implementation

• Reflections and Future Work

• Summary

• Q & A
Problem: What about UML?

• UML class models more or less map to Ecore
  ♦ Good enough for many applications

• Several UML constructs, and corresponding OCL capabilities, have no correspondents in Ecore
  ♦ Static feature calls
  ♦ Message expressions – Ecore does not model signals nor actions
  ♦ Association class navigation
  ♦ Qualified association end navigation
  ♦ Non-navigable association end navigation
  ♦ oclIsInState() and transition guards

• Serialization not practical
OCL Dialects – Relationship to MOF and UML

• OCL has two important dialects

• CompleteOCL targets (and extends) the Unified Modeling Language (UML) metamodel
  • CompleteOCL expressions are UML TypedElements and are defined in the context of UML Classifiers

• EssentialOCL targets (and extends) the Essential Meta-Object Facility (EMOF) metamodel
  • EssentialOCL expressions are EMOF TypedElements and are defined in the context of EMOF Types

• UML2’s UML metamodel corresponds closely to UML

• EMF’s Ecore metamodel corresponds closely to EMOF
Agenda

- Introduction to OCL
- Case Study: Problem Description
- Case Study: Generic OCL Implementation
- Reflections and Future Work
- Summary
- Q & A
Solution: Generic OCL Implementation

- MDT OCL 1.1 implements the Abstract Syntax Model without reference to Ecore

- Uses Ecore’s new-in-2.3 support for generic types to parameterize the metamodel and API, defining type parameters for the modeling constructs required by OCL expressions, including
  - Classes, data types, enumerations, and other classifiers
  - Attributes, association ends, operations, parameters
  - States, call-operation actions, send-signal actions
  - Constraints
MDT OCL 1.1 Overview

- Generic AST not based on Ecore
- Separate Ecore and UML-based specializations of the AST
OCL 1.1: Generic Metamodell

• Types package
  • OCL defines several metaclasses for specialized classifiers
    ▷ In Java™, Ecore, and UML, a class cannot specialize a type parameter
  • The CollectionType\(<C>\) metaclass is a specialized data type with an elementType : C association to the “classifier” type parameter
  • The MessageType\(<C, 0, P>\) metaclass is a specialized classifier with mutually exclusive referredOperation : 0 and referredSignal : C associations, and owning attributes corresponding to the operation parameters or signal properties
  • The TupleType\(<O, P>\) metaclass is a specialized data type defining records, whose attributes are named parts
  • The PrimitiveType\(<O>\) metaclass is a specialized data type with instances String, Real, Integer, UnlimitedNatural, Boolean
OCL 1.1: Generic Metamodel

Expressions package

- The OclExpression\(<C>\) abstract metaclass is the root of a rich expression hierarchy, including operation-call, property-call, iteration, conditional, and literal expressions. An expression’s type : \(C\) is the static result type of the expression.

- The Variable\(<C, PM>\) metaclass also is a typed element, optionally representing an operation parameter (as in pre-condition constraints).

- The MessageExp\(<C, COA, SSA>\) defines message expressions and has mutually exclusive associations calledOperation : \(COA\) and sentSignal : \(SSA\) to “call operation action” and “send signal action” constructs, respectively.

- The TupleLiteralExp\(<C, P>\) defines tuple literal expressions comprising a list of TupleLiteralPart\(<C, P>\) that look like Variables except that they represent properties of the TupleType\(<O, P>\).
OCL 1.1: Generic Metamodel

- UML package
  - Not needed. All UML dependencies are captured as type parameters

- Query package
  - Omitted. Queries are not modeled as elements in OCL

- Utilities
  - Similar to the 1.0 release, with the addition of:
    - TypedElement<C> interface: provides the getType() : C and setType(C) operations required by the OCL parser
    - ExpressionInOcl<C, PM> metaclass: as defined by the OCL 2.0 specification, which does not provide a package namespace for it
OCL 1.1: Generic Environment

• Over all, the generic Abstract Syntax Model has type parameters for the following constructs: C, O, P, EL, PM, S, COA, SSA

• These are named elements that the Environment is responsible for looking up in the subject model. Thus, the Environment interface binds them together for consistency throughout the API:

  Environment<PK, C, O, P, EL, PM, S, COA, SSA, CT, CLS, E>

• Additional parameters in the Environment include:
  • PK (for packages) which are only indirectly referenced by OCL expressions but directly referenced by context declarations
  • CT for the “constraint” metaclass
  • CLS and E for the class and element/object types in the evaluation of expressions
OCL 1.1: Parsing and Evaluation

• Parsing and evaluation using the Query and QueryFactory is replaced by a façade: org.eclipse.ocl.OCL<PK, C, …>
  • OCL encapsulates the Environment<PK, C, …> of a particular subject model

• Input text is provided by an OCLInput

• The OCL can create a Query for repetitive evaluation
OCL 1.1: Metamodel Bindings

- OCL 1.1 defines **bindings** for the UML and Ecore metamodels
- Abstract Syntax models that extend the generic OCL model:
  - Bind generic type parameters to the target metamodel’s metaclasses
  - Merge in the generalizations to the target metamodel’s metaclasses
- Environment implementations:
  - Similar type parameter substitutions
  - Metamodel-specific algorithms for looking up model elements
  - Other metamodel-specific capabilities (e.g., CompleteOCL support, additional compliance points, optional custom capabilities)
import org.eclipse.ocl.ecore.OCL;
import org.eclipse.ocl.ecore.Constraint; // on behalf of Ecore

OCL ocl = OCL.newInstance();
OCLInput input = new OCLInput(
    "package library context Borrower " +
    "inv: self.loan->collect(overdueDays() * rate())->sum() <= 30.0 " +
    "endpackage");
List<Constraint> constraints = ocl.parse(input);
OCL.Query query = ocl.createQuery(constraints.get(0));
OCLExpression<EClassifier> body = query.getExpression();
System.out.printf("Constraint violations for %s: %s",
    body.toString(),
    constraint.reject(allBorrowers));

/* © Copyright 2008 IBM Corp. All rights reserved. This source code is made available under the terms of the Eclipse Public License, v1.0 */
import org.eclipse.ocl.ocl.uml.OCL;
import org.eclipse.uml2.uml.Constraint;  // defined by UML

OCL ocl = OCL.newInstance();
OCLInput input = new OCLInput(
    "package library context Borrower "+
    "inv: self.loan->collect(overdueDays() * rate())->sum() <= 30.0 "+
    "endpackage");
List<Constraint> constraints = ocl.parse(input);
OCL.Query query = ocl.createQuery(constraints.get(0));
OCLExpression<Classifier> body = query.getExpression();
System.out.printf("Constraint violations for %s: %s",
    body.toString(),
    constraint.reject(allBorrowers));
Agenda

• Introduction to OCL

• Case Study: Problem Description

• Case Study: Generic OCL Implementation

• Reflections and Future Work

• Summary

• Q & A
Pioneering Ecore Genericity

• Early test case for Java 5.0 code generation with generics
  • Helped to identify problems
  • Provided requirements such as support for wildcard operation parameters and parameterized XyzSwitch classes

• Even provided test cases for bugs in JDT compiler’s handling of generic types!
  • Reporting of spurious occurrences of the “operation \(x\) has the same erasure as \(y\) but does not override it” involving multiple inheritance of interfaces
Pioneering UML Templates

- Early test case for metamodel transformations of generic types
- Ecore to UML conversion:
  - Ownership of template binding classes
  - Representation of wildcards
  - Sharing of templates and bindings
- UML to Ecore conversion:
  - Stereotype definitions («eGenericType», «eTypeParameter», etc.)
  - Code generation from UML templates
Benefits over Alternative Approaches

• Separate implementations of EssentialOCL for Ecore and CompleteOCL for UML
  • Two distinct abstract syntax models
  • Two distinct environment, parsing, evaluation APIs
  • Difficult to implement clients and extensions targeting both metamodels

• Reflection in terms of Object (no generic bindings)
  • Awkward for clients and extensions alike to perform casts
  • No type safety
Large Generic Signatures

• Long parameter lists can make code hard to read. e.g.,

```java
EnvironmentFactory<PK, C, O, P, EL, PM, S, COA, SSA, CT, CLS, E> factory;
OCL<PK, C, O, P, EL, PM, S, COA, SSA, CT, CLS, E> ocl =
OCL.newInstance(factory);
```

• Java needs an alias or typedef construct

• Mitigated by the metamodel-specific type parameter bindings. e.g.

```java
import org.eclipse.ocl.ecore.OCL;

OCL ocl = OCL.newInstance(new EcoreEnvironmentFactory());
```

• Limiting from an API evolution perspective

• Generic type signatures changes are API-incompatible
Compliance

• Serialization
  • One of the goals of the 1.1 release is to provide serialization
  • Serialization compliance not well defined by the OCL specification
  • Generic AST model not interesting; focus on metamodel bindings

• Standard library
  • OCL standard types in 1.0 release have no URI
  • TypeResolver\(<C, 0, P>\) API
    • Instantiates standard library types on the fly
    • Stores them in a Resource (persistent Environment)
  • Standard library not well defined by the OCL specification
    • Problems with TypeType, ElementType
Extensibility

• 1.0 release not designed for extensibility

• 1.1 extensibility focuses on metamodel bindings

• 1.2 release attempts to provide rudimentary extensibility of:
  • Grammar (contributed by QVT implementation in GMT UMLX)
  • Environment

• Outstanding extensibility issues in:
  • Standard library
  • Validation
Agenda

• Introduction to OCL

• Case Study: Problem Description

• Case Study: Generic OCL Implementation

• Reflections and Future Work

• Summary

• Q & A
Summary

• OCL is a powerful language for constraints and queries
• Generic AST model binds to Ecore, UML, and other metamodels
• Like most solutions to most practical problems, there are trade-offs
• There is still much to be done. Contributions are welcome!
Resources

• Object Constraint Language (OCL) Specification

• MDT OCL Newsgroup
  ✓ news://news.eclipse.org/eclipse.modeling.mdt.ocl

• Eclipse Corner Articles:

• MDT OCL source code & unit tests in Eclipse CVS
Agenda

• Introduction to OCL
• Case Study: Problem Description
• Case Study: Generic OCL Implementation
• Reflections and Future Work
• Summary

• Q & A
Questions?

• I may have some useful answers
Thank you!
Legal Notices

Copyright © IBM Corp., 2008. All rights reserved. Source code in this presentation is made available under the EPL, v1.0; remainder of the presentation is licensed under Creative Commons Att. Nc Nd 2.5 license.

IBM and the IBM logo are trademarks or registered trademarks of IBM Corporation in the United States, other countries, or both.

Rational and the Rational logo are trademarks or registered trademarks of International Business Corporation in the United States, other countries, or both.

UML, Unified Modeling Language, and MOF are trademarks or registered trademarks of Object Management Group, Inc.

Java and all Java-based trademarks, among others, are trademarks or registered trademarks of Sun Microsystems in the United States, other countries, or both.

Eclipse and the Eclipse logo are trademarks of the Eclipse Foundation, Inc.

Other company, product, and service names may be trademarks or service marks of others.

THE INFORMATION DISCUSSED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION, IT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, AND IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, SUCH INFORMATION. ANY INFORMATION CONCERNING IBM’S PRODUCT PLANS OR STRATEGY IS SUBJECT TO CHANGE BY IBM WITHOUT NOTICE.