Report of the

QVT 1.2 Revision Task Force

to the

OMG Domain Technical Committee

(Revision 1)

26 March 2014

Document Number:	**ptc/2014-02-02 Draft**
Task Force Chair(s):	**Edward Willink**
Chartered:	**11 December 2009 in Long Beach, CA**
Comments due date:	**13 September 2010**
Expiry date:	**4 April 2014**

Template: omg/13-05-01

Deliverables

Chosen acronym: QVT
Version number: 1.2

Specification

Revised specification (clean): ptc/2014-02-04
Revised specification (change-bar): ptc/2014-02-03

Supporting documents

Title: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification (QVT) RTF Beta document WITHOUT change bars
Doc Number: ptc/2014-02-04
Description: QVT 1.2 - Beta Specification
Status: Normative
URL: http://www.omg.org/spec/QVT/1.2/PDF

Title: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification (QVT) RTF Beta document WITH change bars
Doc Number: ptc/2013-08-12
Description: QVT 1.2 - Beta Specification with change bars
Status: Informative
URL: http://www.omg.org/spec/QVT/1.2/PDF

Machine-consumable documents

Description: Query/View/Transformation (QVT) RTF - XMI files (UML)
Doc Number: ptc/2014-02-05
Filename: 2014-02-05.zip
Status?: Normative
Depends on:

Description: Query/View/Transformation (QVT) RTF - XMI files (Ecore, Diagrams)
Doc Number: ptc/2014-02-06
Filename: 2014-02-06.zip
Status?: Informative
Depends on:

IPR Mode

IPR mode of base specification: None

If "None", Legacy IPR mode selected: RF on Limited Terms

QVT 1.2 RTF Membership

Representative	Organisation	Status
Manfred Koethe	88solutions	
Pete Rivett	Adaptive	
Bernd Wenzel	Fachhochschule Vorarlberg	
Michael Wagner	Fraunhofer FOKUS	
Jishnu Mukerji	Hewlett-Packard	
Didier Vojtisek	INRIA	
Xavier Blanc	Laboratoire Informatique de Paris 6	
Nicolas Rouquette	NASA	
Andrius Strazdauskas	No Magic, Inc.	
Edward Willink	Nomos Software	(CHAIR)
Victor Sanchez	Open Canarias, SL	
Philippe Desfray	Softeam	
Laurent Rioux	THALES	

Revision Details

Table of Contents

Disposition Summary

Disposition	Number of Occurrences	Meaning of Disposition
Resolved	18+30+22+1=71	The RTF/FTF agreed that there is a problem that needs fixing, and has proposed a resolution (which may or may not agree with any resolution the issue submitter proposed)
Deferred	4+0+25+0=29	The RTF/FTF agrees that there is a problem that needs fixing, but did not agree on a resolution and deferred its resolution to a future RTF/FTF.
Transferred	1+0+0+0=1	The RTF/FTF decided that the issue report relates to another specification, and recommends that it be transferred to the relevant RTF.
Closed, no change	9+0+5+0=14	The RTF/FTF decided that the issue report does not, in fact, identify a problem with this (or any other) OMG specification.
Closed, Out of Scope	0+0+0+0=0	The RTF/FTF decided that the issue report is an enhancement request, and therefore out of scope for this or any future FTF or RTF working on this major version of the specification. The RTF/FTF has closed the issue without making any specification changes, but RFP or RFC submission teams may like to consider these enhancement requests when proposing future new major versions of the specification.
Duplicate or merged	0+7+2+0=9	This issue is either an exact duplicate of another issue, or very closely related to another issue: see that issue for disposition.

Voting Record

Poll No.	30 December 1899	Issues included
1	22 January 2014	See reballot below
1 reballot	05 February 2014	Resolved:10937,10938,10939,10940,10941,10942, 10943,10944,10945,11058,11108,11685,11708,11825, 12368,12571,12572,12573 No Change:11061,12200,12260,12367,13988,14573, 14835,15215,15416 Deferred:10934,10935,10936,11686
2	05 February 2014	Resolved:13222,13264,13265,13268,13272,13273,13274, 13275,13277,13278,13280,13282,13283,13284,13285, 13286,13288,13290,13913,13989,14549,14619,15424, 15917,15978,18572,19096,19121,19146,19178 Duplicate:13223,13228,13251,13987,15524,19095,19174
3	19 February 2014	Resolved:11602,11826,12518,12522,12523,12524,12525, 12526,12527,13182,13183,13267,13269,13270,13276, 13279,13281,13287,13289,15977,19021,19022 No Change:12519,12520,12521,17538,18325 Deferred:11690,12213,12370,13054,13082,13103,13158, 13168,13180,13181,13252,14620,14640,15376,15390,15411, 15417,15523,15886,18323,18324,18363,18912,19019,19023 Duplicate:13266,13271
4	19 February 2014	Resolved:19208

Voter	Vote in poll 1	Vote in poll 1 reballot	Vote in poll 2	Vote in poll 3	Vote in poll 4
Manfred Koethe	YES to all	YES to all	YES to all	YES to all	Did not vote
Pete Rivett	Did not vote	Did not vote	Removed	Removed	Removed
Bernd Wenzel	YES to all	YES to all	YES to all	YES to all	YES to all
Michael Wagner	YES to all	YES to all	YES to all	YES to all	YES to all
Jishnu Mukerji	Did not vote	Did not vote	Removed	Removed	Removed
Didier Vojtisek	Did not vote	Did not vote	Removed	Removed	Removed
Xavier Blanc	Did not vote	Did not vote	Removed	Removed	Removed
Nicolas Rouquette	YES to all	YES to all	YES to all	YES to all	YES to all
Andrius Strazdauskas	Did not vote	Did not vote	Removed	Removed	Removed
Edward Willink	YES to all	YES to all	YES to all	YES to all	YES to all
Victor Sanchez	YES to all	YES to all	Did not vote	YES to all	YES to all
Philippe Desfray	Did not vote	Withdrew	Withdrew	Withdrew	Withdrew
Laurent Rioux	Did not vote	YES to all	YES to all	YES to all	Did not vote
YES,NO,Abstain Quorum	6,0,0 13/2	7,0,0 12/2	6,0,0 7/2	7,0,0 7/2	5,0,0 7/2

Summary of Changes Made

The QVT 1.2 RTF made changes that:

1). Corrected features that impeded implementation or did not serve the original intent of the specification,

2). Provided additional convenience for implementers,

3). Increased the clarity of the specification

Here is the FTF's categorization of the resolutions applied to the specification according to their impact on the clarity and precision of the specification:

Extent of change	Number of Issues	OMG Issue Numbers
Critical/Urgent - Fixed problems with normative parts of the specification which prevented implementation work.	9	10941,10943,11825,12518,12522,12527, 12571,13268,18572
Significant - Fixed problems with normative parts of the specification that raised concern about implementability.	22	10938,10942,10944,11058,11708,11826, 12523,12524,12525,12526,12572,12573, 13182,13269,13270,13281,13287,14619, 15977,15978,19021,19022
Minor - Fixed minor problems with normative parts of the specification.	15	10937,10939,10940,11108,13183,13267, 13276,13279,13280,13284,13913,13989, 19096,19121,19146
Support Text -Changes to descriptive, explanatory, or supporting material.	25	10945,11602,11685,12368,13222,13264, 13265,13272,13273,13274,13275,13277, 13278,13282,13283,13285,13286,13288, 13289,13290,14549,15424,15917,19178, 19208

Disposition: Resolved

Issue 10937: 9.18: Realized

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The concrete syntax defines 'realized', but section 10 consistently uses 'realize'.

Suggest: 'realize'

The concrete syntax defines 'realized' for a single element of a comma-separated list.

Section 10 appears to expect that 'realized' is a prefix to a comma-separated list.

Suggest: 'realize' is a comma-separated list prefix.

(semi-colon separation is available for distinct realize/not-realize.)

Resolution:

Use 'realize' as a keyword in the concrete syntax.

Use 'realized' as an adjective in editorial text, model names; e.g. RealizedVariable.

Revised Text:

In 9.18 Concrete Syntax change

```
RealizedVariable :=
"realized" VariableName ":" TypeDeclaration
```

to

```
RealizedVariable :=
"realize" VariableName ":" TypeDeclaration
```

Disposition: **Resolved**

Issue 10938: 9.17 Variable composition

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Issue 9379 made Pattern.bindsTo a non-composition.

This deprives Core of any parent for its non-realized variables.

Suggest: move BottomPattern.realizedVariables to CorePattern.variables.

CorePattern.variables then composes all variables declared for the pattern. A class-type

check of RealizedVariable/Variable derives the 'isRealized' property.

Resolution:

It is convenient to keep realized and unrealized variables separate to avoid unnecessary isRealized tests.

Just add CorePattern.variables to compose the unrealized variables.

Revised Text:

In 9.17.1 CorePattern add

```
variable: Variable [*] {composes} {opposite corePattern:CorePattern[?]}
```

Unrealized guard pattern variables are the unbound variables of the pattern. Unrealized bottom pattern variables may be used to factor out common expressions.

In Figure 9.2 add

a CorePattern to Variable composition arc for CorePattern.variable

Update QVTcore models accordingly.

Disposition: **Resolved**

Issue 10939: 9.18 Trailing |.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

GuardPatterns and BottomPatterns with Variables but without Constraints must end with a '|'.
This is inelegant. Better to only require '|' as prefix to constraints (or to require it always).
Suggest:
GuardPattern ::= Variable ("," Variable)* ["|" (Constraint ";")+]

similarly BottomPattern

Resolution:

Simple change.

Revised Text:

From 9.18 Concrete Syntax change

```
GuardPattern ::=
[Variable(“,”Variable )* “|” ]
( Constraint “;” )*
BottomPattern ::=
[ (Variable | RealizedVariable)
(“,” ( Variable | RealizedVariable)* “|” ]
( Constraint “;” )*
```

to

```
GuardPattern ::=
[Variable(“,”Variable )* ]
[“|” ( Constraint “;” )+]
BottomPattern ::=
[ (Variable | RealizedVariable)
(“,” ( Variable | RealizedVariable)*]
[“|” ( Constraint “;” )+]
```

Disposition: **Resolved**

Issue 10940: 9.18 GuardPattern assignments

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The concrete sysntax allows guard patterns to have assignments. The abstract syntax does not.

Suggest: GuardPattern uses Predicate rather than Constraint

Resolution:

Simple change.

Revised Text:

From 9.18 Concrete Syntax change

```
GuardPattern ::=
[Variable(“,”Variable )* “|” ]
( Constraint “;” )*
```

to

```
GuardPattern ::=
[Variable(“,”Variable )* “|” ]
( Predicate “;” )*
```

Disposition: **Resolved**

Issue 10941: 9.18 The middle direction packages

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

A Direction defines the packages used by each direction domain. Nothing defines the (additional) packages used by the middle area. For instance there is no place to specify the trace class model generated by the RelationToCore transformation. With many transformations and mappings in a QVTcore it does not seem appropriate to overload TransformationName or MappingName as a hook to reference the appropriate middle meta-models; different Mappings may have distinct middle meta-models; multiple transformations may share the same middle meta-model. (Core should be a first class programming language, not just one that supports the eccentricities of a RelationToCore conversion.)

Suggest: a DirectionName between "where" and "(" in Mapping. This reads very naturally:

map ...

check left (...

enforce right (...

where middle (...

But we also need to fix the Abstract Syntax. As a minimum a Mapping needs a middle-typed-model property to capture the missing information. With this addition a Mapping duplicates so much of a CoreDomain/Area, that it is more appropriate to eliminate Area altogether, merging it into CoreDomain. Area is eliminated from Mapping, and added as an additional CoreDomain referenced by the middleDomain property.

So:

CoreDomain extends Domain

CoreDomain.variables : RealizedVariable [0..*] composed

CoreDomain.bottomPattern : BottomPattern [1] composed

CoreDomain.guardPattern : GuardPattern [1] composed

Mapping extends Rule

Rule.domain : Domain [0..*] composed (must be at least CoreDomain[3])

Mapping.specification - no change

Mapping.local - no change

Mapping.context - no change

Mapping.middle : CoreDomain [1] (must be in Rule.domain)

Resolution:

The suggested changes may be convenient but they are not necessary:

- the middle model can be anonymous

- the middle area can be merged with the overall mapping.

What is needed is a way to bind a middle metamodel to its TypedModel. This can be achieved by an anonymus direction bindng

Revised Text:

In 7.11.1.2 TypedModel change

A *typed model* specifies a named, typed parameter of a *transformation*.

to

A *typed model* specifies a typed parameter of a *transformation*. Explicit external parameters have a non-null name. An implicit parameter such as the QVTc middle model may have a null name,

In 9.18 Concrete Syntax change

```
  Direction ::=
  DirectionName ["imports" PackageName("," PackageName)*]
  ["uses" DirectionName("," DirectionName)*]
to
  Direction ::=
  [DirectionName] ["imports" PackageName("," PackageName)*]
  ["uses" DirectionName("," DirectionName)*]
```

Disposition: **Resolved**

Issue 10942: 9.18 Top-level syntax.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The concrete syntax does not specify a parsing goal. If the first element (Transformation) is taken to be the goal, most of the rest of the grammar is orphaned.

Suggest:

TopLevel ::= (Transformation | Mapping)*

Resolution:

Simple change.

Revised Text:

In 9.18 Concrete Syntax add

```
TopLevel ::= (Transformation | Mapping)*
```

Disposition: **Resolved**

Issue 10943: 9.18 Anonymous Maps.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Section 10 uses anonymous maps for composed mappings. The Concrete Syntax does not allow this. Similarly, an 'in' is not appropriate for a composed mapping.

Suggest:

ComposedMapping ::= "map" [MappingName] ["refines" MappingName] MappingPatterns

where MappingPatterns is the common part of Mapping.

Resolution:

Simple change.

Revised Text:

In 9.18 Concrete Syntax change

```
   ComposedMapping ::= Mapping
to
   ComposedMapping ::=
   "map" [MappingName] ["refines" MappingName] "{"
   (["check"] ["enforce"] DirectionName "("DomainGuardPattern")" "{"
   DomainBottomPattern
   "}" )*
   "where" "(" MiddleGuardPattern ")" "{"
   MiddleBottomPattern
   "}"
   (ComposedMapping )*
   "}"
```

Disposition: **Resolved**

Issue 10944: 9.18 EnforcementOperation.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

There is no concrete syntax for enforcement operations.

Suggest: a BottomPattern Constraint may also be

'create' OperationCallExpCS

'delete' OperationCallExpCS

Resolution:

Simple change.

Revised Text:

In 9.18 Concrete Syntax change

```
  BottomPattern ::=
  [ (Variable | RealizedVariable)
  (“,” ( Variable | RealizedVariable)* “|” ]
  ( Constraint “;” )*
to
  BottomPattern ::=
  [ (Variable | RealizedVariable)
  (“,” ( Variable | RealizedVariable)* “|” ]
  ( (Constraint | EnforcementOperation) “;” )*
  EnforcementOperation ::=
  (“create” | “delete”) OperationCallExpCS
```

Disposition: **Resolved**

Issue 10945: 9.18 Typographics Issues

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Style. Use same font and presentation as 7.12.3, 8.4.

Typo. Remove '(' preceding '",'" in BottomPattern.

Typo. Remove 'e' in 'Assignement'.

Resolution:

Some aspects of the presentation are easily resolved, but replacing e.g. DirectionName to <directionName> is not worthwhile. These will all change to e.g. DirectionNameCS once OCL 2.5 aligned grammars are provided.

The '(' is needed as part of an 'optional' for which the prevailing BNF is [].

Assignement is a simple typo.

Revised Text:

Throughout 9.18 Change asymmetric double quotes ‟...″ to single quotes '...'

In 9.18 Concrete Syntax change

```
BottomPattern ::=
[ (Variable | RealizedVariable)
(“,” ( Variable | RealizedVariable)* “|” ]
( Constraint “;” )*
```

to

```
BottomPattern ::=
[ (Variable | RealizedVariable)
[“,” ( Variable | RealizedVariable)*] “|” ]
( Constraint “;” )*
```

In 9.18 Concrete Syntax change

```
Assignement ::=
```

to

```
Assignment ::=
```

Disposition: **Resolved**

Issue 11058: Consider renaming collectselect as xcollectselect

Source:

Mariano Belaunde (France Telecom R&D, mariano.belaunde(at)orange.com)

Summary:

For uniformity, collectselect should be renamed xcollectselect following the distinction between collect and xcollect (imperative version).

Resolution:

This is an unpleasant but worthwhile change. Implementations may choose to provide both spellings for transitional compatibility.

If collectselect changes, so too should collectselectOne, collectOne and selectOne, since their computation is imperative too.

Revised Text:

Change all 9 occurrences of 'collectselect' to 'xcollectselect'.

Change all 7 occurrences of 'collectselectOne' to 'xcollectselectOne'.

Change all 5 occurrences of 'selectOne' to 'xselectOne'.

Change all 2 occurrences of 'collectOne' to 'xcollectOne'.

Disposition: **Resolved**

Issue 11108: Assignment.slotExpression

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

9.3 Assignment.slotExpression should be PropertyAssignment.slotExpression (VariableAssignments have no slot)

Resolution:

Simple move already in QVTcore.xml.

Revised Text:

From 9.17.8 Assignment move

`slotExpression: OclExpression [1] {composes}`

An OCL expression identifying the object whose property value is to be assigned.

to 9.17.9 PropertyAssignment

`slotExpression: OclExpression [1] {composes}`

An OCL expression identifying the object whose property value is to be assigned.

In Figure 9.3 move the Assignment end of OclExpression.slotExpression to PropertyAssignment.

Disposition: **Resolved**

Issue 11602: Section: 7.13

Source:

Vienna University of Technology (Johann Oberleitner, joe(at)infosys.tuwien.ac.at)

Summary:

I have built a transformation engine based on QVT relations. The QVT relations example in Annex A contains a 'function' PrimitiveTypeToSqlType at the end of the example in textual syntax. This example is the only relations example in the whole spec. Nowhere is the semantics of 'function' defined nor contains the grammar of the concrete syntax a function keyword. However, 'query' is defined. Is 'function' another name for 'query'?

Resolution:

QVT Base has a Function.queryExpression.

QVT 1.1 added the mapping from queryCS to Function.

Not 'nowhere' but we can do better.

Revised Text:

In 7.11.1.5 Function change

A function may be specified

to

Since a function is side effect free, it is often called a query. A function may be specified

Disposition: **Resolved**

Issue 11685: Section: 7.11.3.

Source:

Siegfried Nolte siegfried(at)siegfried-nolte.de

Summary:

It is not clear when and how you choose a top-level and not-top-level relation. Primitive domains, the reason for them and the use of them, are not explained although they are used in the Relation language example A1.1.1

Resolution:

'top' is explained clearly in Section 7.2.2 Top-level Relations.

'primitive' is a bit of a secret.

Revised Text:

Add 7.2.4 Primitive Domains

Simple data such as configuration information or constants may be passed as parameters to a relation using primitive domains. A primitive domain is identified by primitive keyword and no domain name. A primitive domain is neither checkable nor enforceable.

```
relation Outer {
   checkonly domain source s:Source {};
   enforce domain target t:Target {};
   where {
      Inner(s,t,'target');
}

relation Inner {
   checkonly domain source s:Source {name=pn};
   enforce domain target t:Target {name=separator + pn};
   primitive domain separator:String;
}
```

Disposition: **Resolved**

Issue 11708: 9.17.12, EnforcementOperation.operationCallExp should be composes

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The definition: "operationCallExp : OperationCallExp[1]" and the corresponding "*" enforcementOperation multiplicity in Figure 9.3 provides a consistent definition of a shared reference to a Call expression. There is no indication of where this expression might be contained. It is unlikely that such expressions can usefully be shared since they must refer to invocation-site-specific variables.

Therefore:

Change the definition to:

operationCallExp : OperationCallExp[1] {composes}

and draw the composition with 0..1 parent multiplicity.

Resolution:

Simple change already in QVTcore.xml.

Revised Text:

From 9.17.12 EnforcementOperation change

```
operationCallExp: OperationCallExp [1]
```

to

```
operationCallExp: OperationCallExp [1] {composes}
```

In Figure 9.2 draw EnforcementOperation.operationCallExp as a 0..1 composition.

Disposition: **Resolved**

Issue 11825: Inconsistent Rule.transformation multiplicity/composes for Mapping

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Mapping is a Rule and Rule.transformation has unit multiplicity and is a container.

Therefore Rule.context can never be a container.

This could be fixed in a number of ways:

Fix 1: Change Rule.transformation to 0..1
--> allow hierarchical Rule containment
--> all Transformation rules are distinctly named
-- no representation change
-- minor semantic relaxation for QVTr/QVTo
-- minor semantic surprise for QVTc - composed mapping has null Mapping.transformation.

Fix 2: Change Rule.context to not composes
--> require flat Rule containment
--> Transformation can have multiple unnamed rules
-- no change for QVTc/QVTo
-- representation change for QVTc

Fix 3: Change opposite of Transformation.rule from Rule.transformation[1] to Rule.context[1]
Redefine Rule.transformation[1] as a derived property
Remove mapping.context (inherited from Rule)
-- Doesn't work: context needs to be NamedElement to be either Rule or Transformation

Fix 4: Change opposite of Transformation.rule from Rule.transformation[1] to Rule.owningTransformation[0..1]
Redefine Rule.transformation[1] as a derived property
Rule.transformation := owningTransformation
Mapping.transformation := if context then context.transformation else owningTransformation endif
-- no representation change
-- minor semantic relaxation for QVTr/QVTo

Recommend 4.

Resolution:

Rule.transformation changed to 0..1 in QVTCore.xml for QVT 1.1.

Fix 4 doesn't seem that wonderful, so go with the obvious Fix 1. And why prohibit all reuse of Rule outside a Transformation?

Revised Text:

In Fig 7.4 change Rule.transformation multiplcity from '1' to '0..1'.

In 7.11.1.4 Rule change

```
transformation: Transformation[1]
```

to

```
transformation: Transformation[0..1]
```

Disposition: **Resolved**

Issue 11826: Section 7.11.2.3: Empty CollectionTemplateExp is useful

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

7.11.2.3 and Figure 7.6 both impose a lowerBound of 1 on CollectionTemplateExp.member and CollectionTemplateExp.rest.

However, the concrete syntax permits an empty match. This empty match is exploited in the UnsharedWhenVarsToMgVars relation in Section 10.

Suggest:

Change CollectionTemplateExp.member to [0..*]

Change CollectionTemplateExp.rest to [0..1]

Add a sentence to clarify the empty match semantics.

Resolution:

Yes.

Revised Text:

In 7.11.2.3 CollectionTemplateExp change

`member : OclExpression [1..*] {composes}`

The expressions that the elements of the collection must have matches for. A special variable _ may be used to indicate that any arbitrary element may be matched and ignored.

`rest : Variable [1]`

The variable that the rest of the collection (i.e., excluding elements matched by member expressions) must match. A special variable _ may be used to indicate that any arbitrary collection may be matched and ignored.

to

`member : OclExpression [*] {composes}`

The expressions that the elements of the collection must have matches for. A special variable _ may be used to indicate that any arbitrary element may be matched and ignored. The expression may be omitted to restrict a match to an empty collection.

`rest : Variable [0..1]`

The variable that the rest of the collection (i.e., excluding elements matched by member expressions) must match. A special variable _ may be used to indicate that any arbitrary collection may be matched and ignored. The variable may be omitted to restrict a match to a collection with no elements other than those matched by the *member* expressions.

Disposition: **Resolved**

Issue 12368: Issue against QVT ptc/07-07-07 : clause 7.2.3.

Source:

NIST (Mr. Peter Denno, peter.denno(at)nist.gov)

Summary:

Text of Clause 7.2.3 is unclear:

Suggested rewrite:

(1) Add the following as the first two sentences of the first paragraph.

"In the evaluation of a Transformation, one or more domains are specified as target. The phrase 'executing in the direction of [some domain]' refers to these domains."

Then change a few words in the paragraph as suggest by the text in [] below:

"Whether or not the [change "the" to "a"] relationship maybe [should be "is"] enforced is determined by the target domain, which may be marked as checkonly or enforced. When a transformation [change "transformation" to "relation"] is enforced [change "enforced" to "evaluated"] in the direction of a checkonly domain, it is simply checked to see if [change "if" to "whether"] there exists a valid match in the relevant model that satisfies the relationship. When a transformation executes in the direction of the model of an enforced domain, if checking fails, the target model is modified so as to satisfy the relationship, i.e. a check-before-enforce semantics."

[Strike the words beginning with "i.e." "check-before-enforce" is new terminology that is neither defined nor helpful.]

Resolution:

Yes, we can certainly try to be clearer.

Revised Text:

In 7.2.3 change

> Whether or not the relationship may be enforced is determined by the target domain, which may be marked as checkonly or enforced. When a transformation is enforced in the direction of a checkonly domain, it is simply checked to see if there exists a valid match in the relevant model that satisfies the relationship. When a transformation executes in the direction of the model of an **enforced** domain, if checking fails, the target model is modified so as to satisfy the relationship, i.e., a check-before-enforce semantics.

to

> When a transformation is evaluated, the transformation executes in the direction of the domain specified as the target. Whether or not a relation is enforced is determined by the target domain, which may be marked as checkonly or enforced. When a relation executes in the direction of a checkonly domain, the domain is simply checked to see whether there exists a valid match in the relevant model that satisfies the relationship. When a relation executes in the direction of an enforced domain, if checking fails, the target model is modified so as to satisfy the relation.

Disposition: **Resolved**

Issue 12518: errors and anomalies in QVT_1.0.mdl in the 07-07-08 ZIP

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Use of automated tooling to support comparison of the models developed initially as part of the Eclipse GMT/UMLX project and being transferred to the Eclipse QVT Declarative/QVT Operational Mappings Projects reveals the following errors and anomalies in QVT_1.0.mdl in the 07-07-08 ZIP.

Since the diagrams are printed from QVT_1.0.mdl all the QVT problems also occur in 08-04-03.

Textual errors in 08-04-03 cannot be analyzed automatically. There are so many that a thorough proof read is required combined with a statement that the diagrams only are normative

Resolution:

The QVT 1.0 diagrams and models were originally from the models in the QVT_1.0.mdl file. The diagrams rely on proprietary tooling. Unfortunately some independent evolution occurred and so there were many inconsistencies.

Consistent Ecore/EMOF files from Eclipse were endorsed as the QVT 1.1 non-normative files.

For QVT 1.2, the primary non-normative files are UML models derived from the QVT 1.1 Ecore files. The diagrams are redrawn from the UML using the Open Source Papyrus tool. The diagrams are drawn to assist understanding rather than to squeeze as much as possible into the useable area. The redrawn diagrams are therefore larger/more numerous. (Unfortunately Papyrus does not support {ordered} so {ordered} text is added manually.) In all other respects diagrams and UML non-normative files should be consistent.

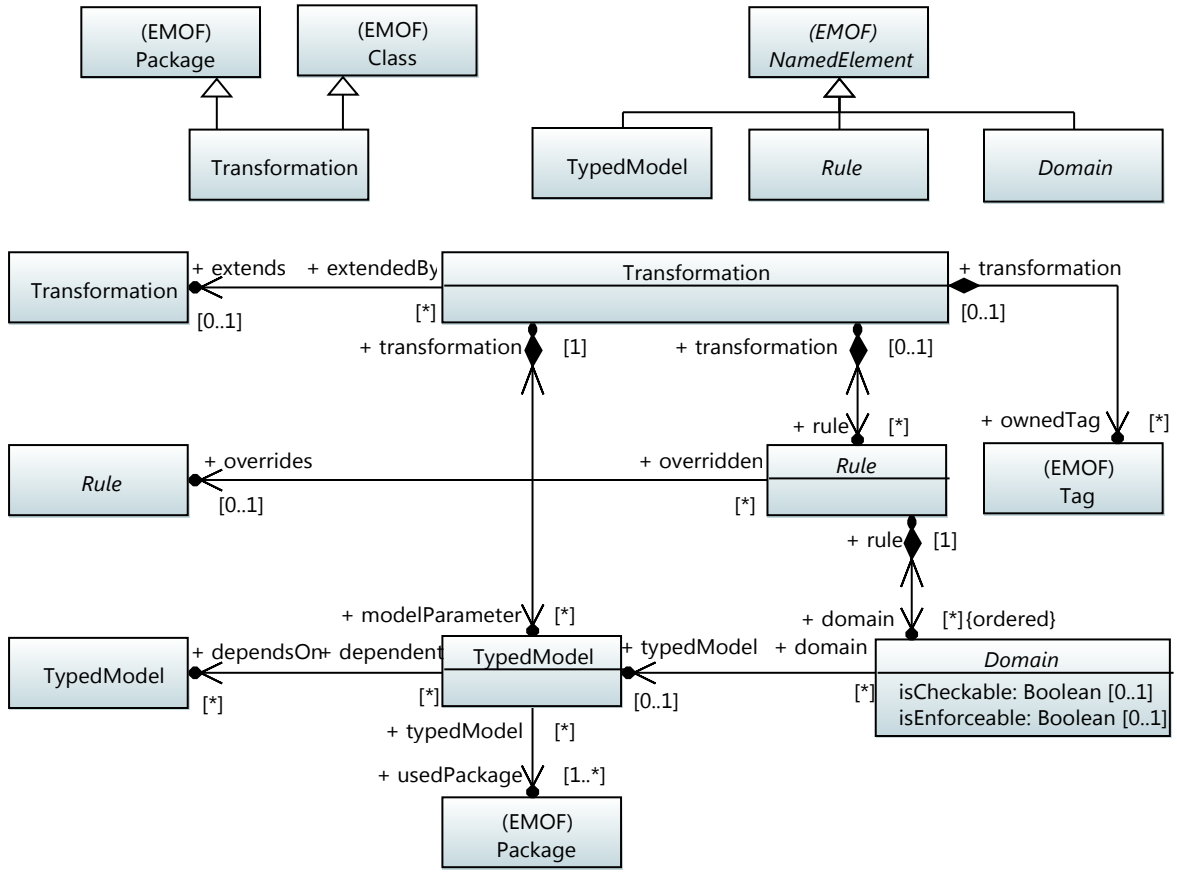
The multiplicity of unnavigable opposites was not present in the Ecore files and so is set to 0..1 for compositions, and 0..* for non-compositions by the Ecore2UML conversion. This causes a few 1 multiplicities to change to 0..1. The 1 multiplicities were needlessly restrictive prohibiting re-use of classes, so the 0..1 seems better. Perhaps we should have an issue to change navigable containers too.

TemplateParameterType is missing from the diagrams since it is not preset in the non-normative files. TemplateParameterType has no practical utility implementations are required to work magic for templates types which may or may not involve the TemplateParameterType class.

I find that when I consult the specification, I want to see all the diagrams, which is hard when they are spread thoughout the AS section. All diagrams are therefore brought to the start of their AS section.

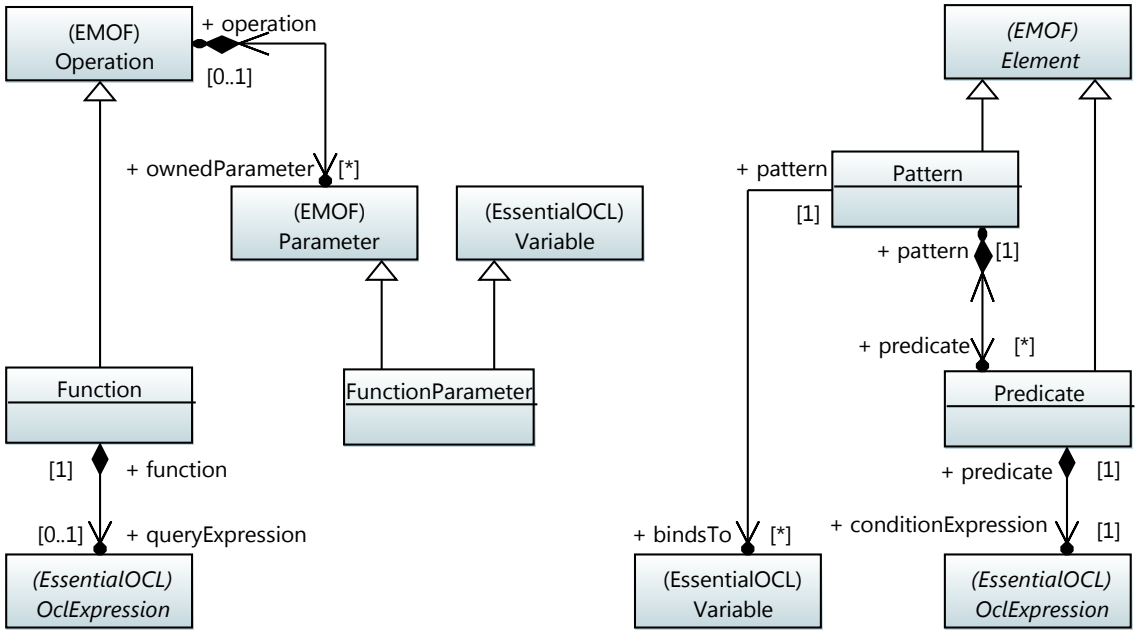
Revised Text:

Replace Fig 7.4 and 7.5 by the following two diagrams positioning the diagrams roughly where Fig 7.4 was at the start of the Abstract Syntax subclause.

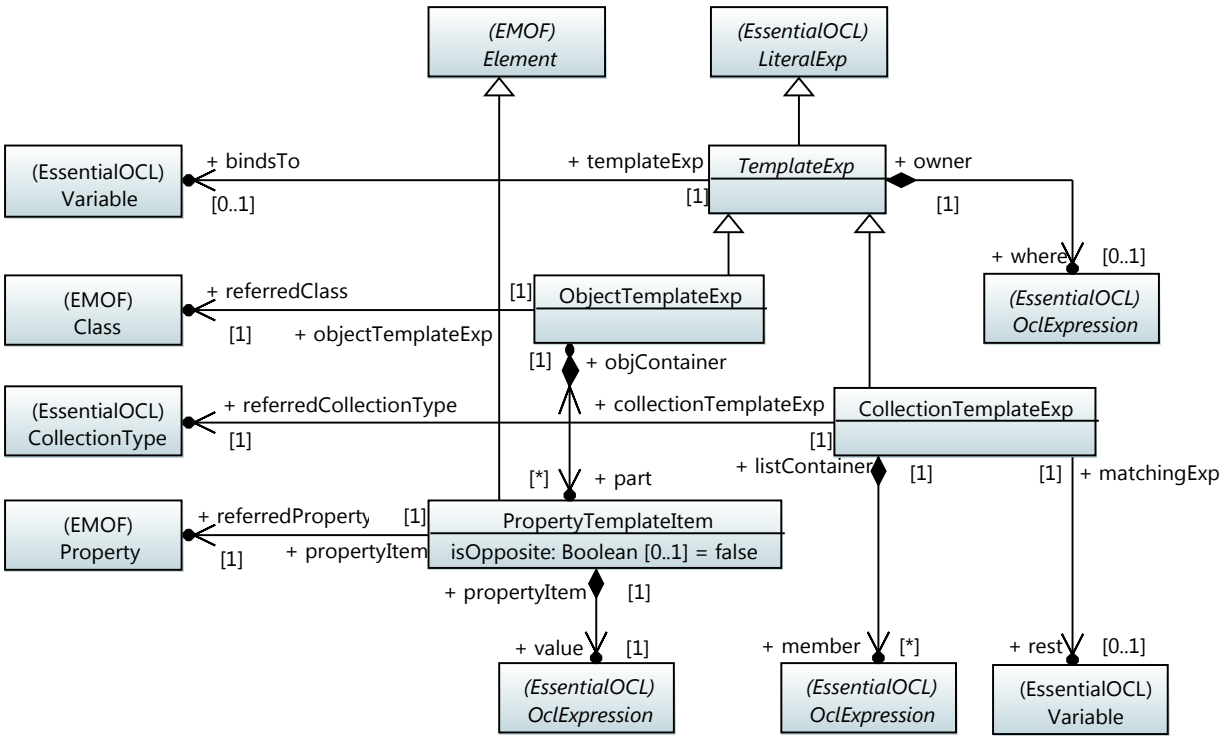


QVTBase 1

QVTBase 2

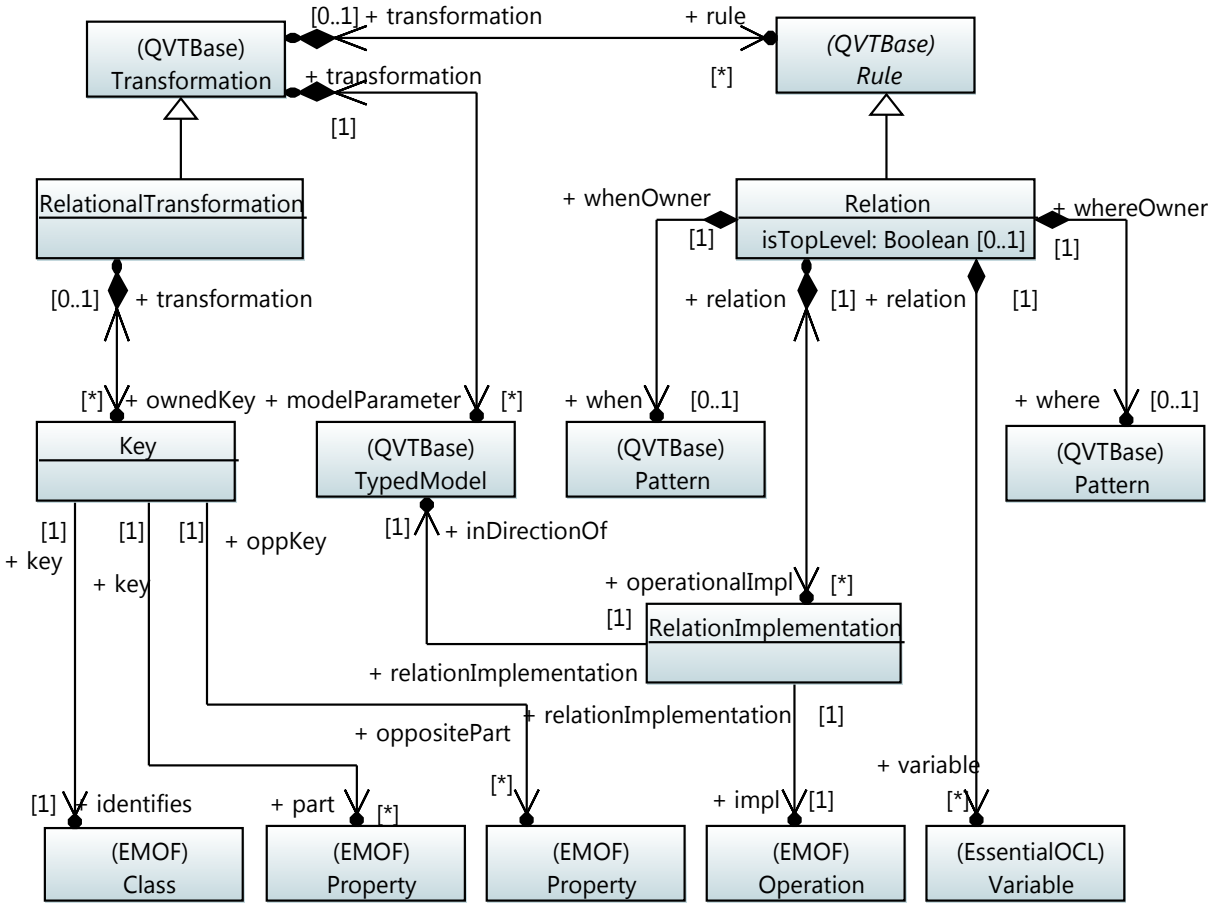


Replace Fig 7.6 by the following diagrams.

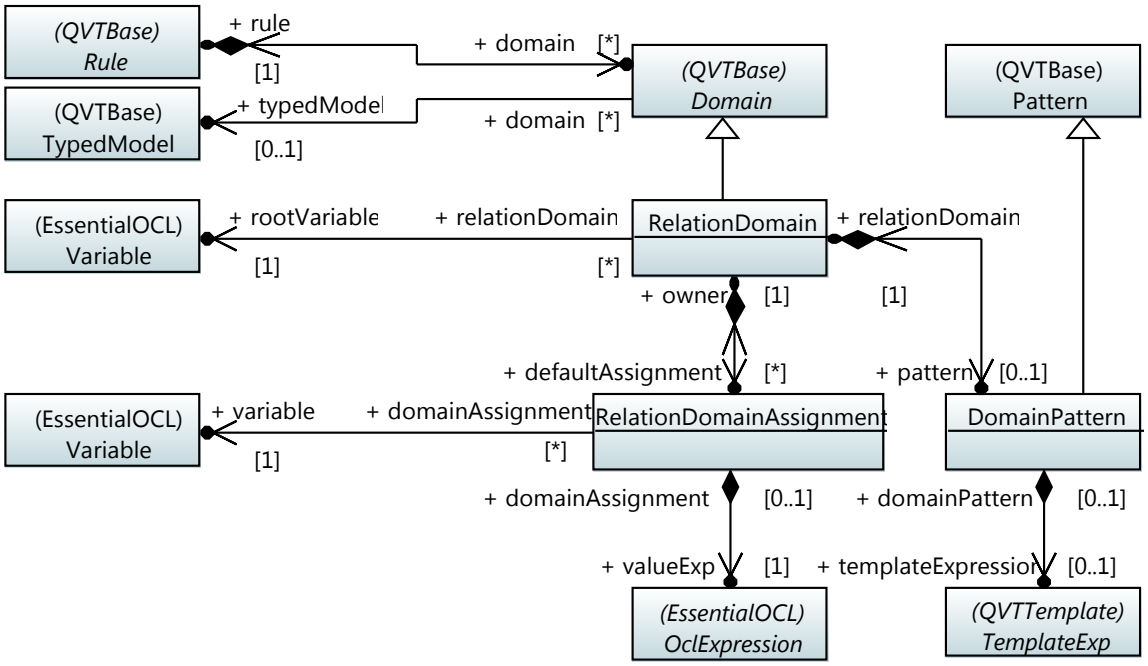


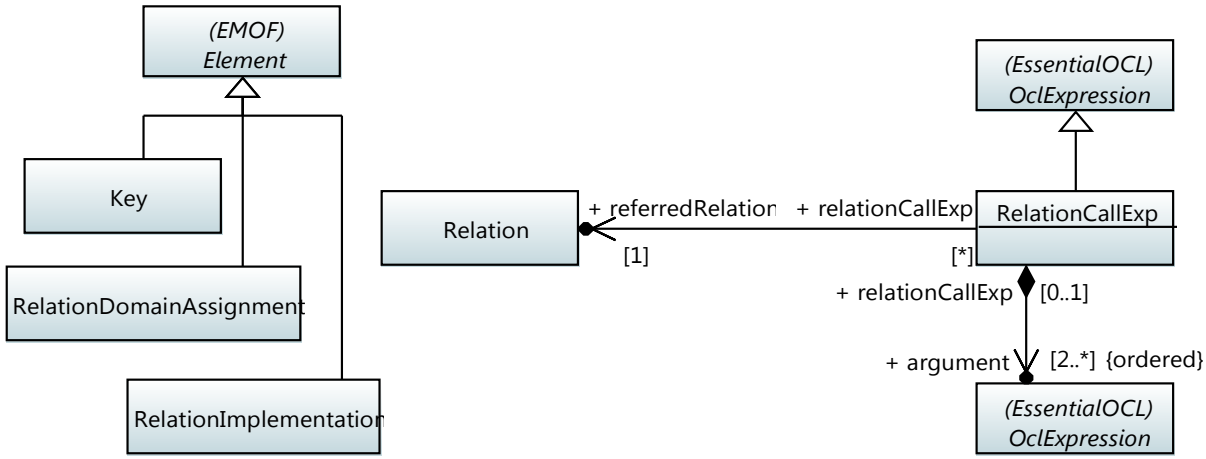
QVTTemplate 1

Replace Fig 7.7 by the following three diagrams positioning the diagrams roughly where Fig 7.7 was at the start of the Abstract Syntax subclause.



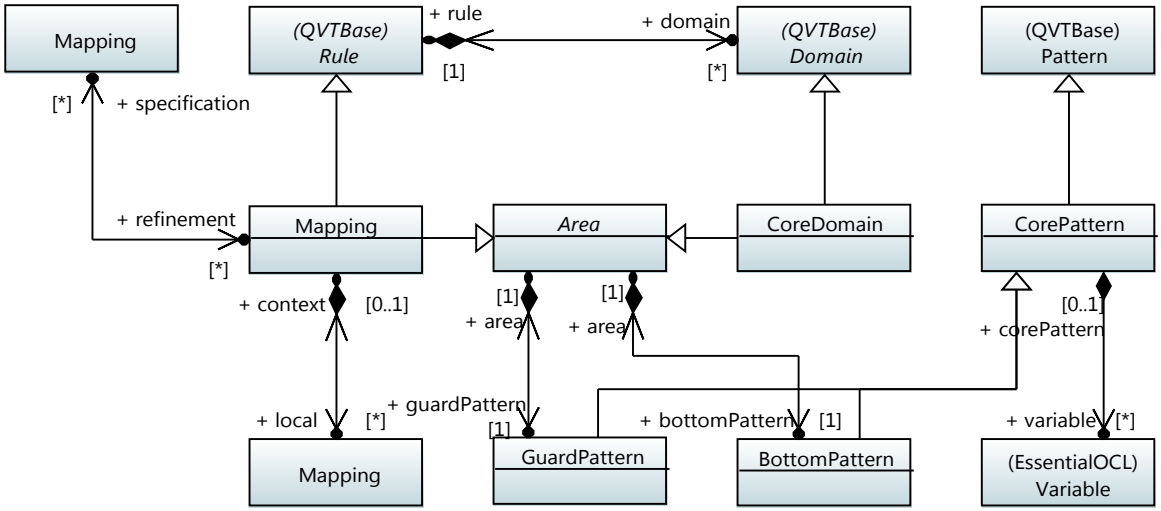
QVTRelation 1



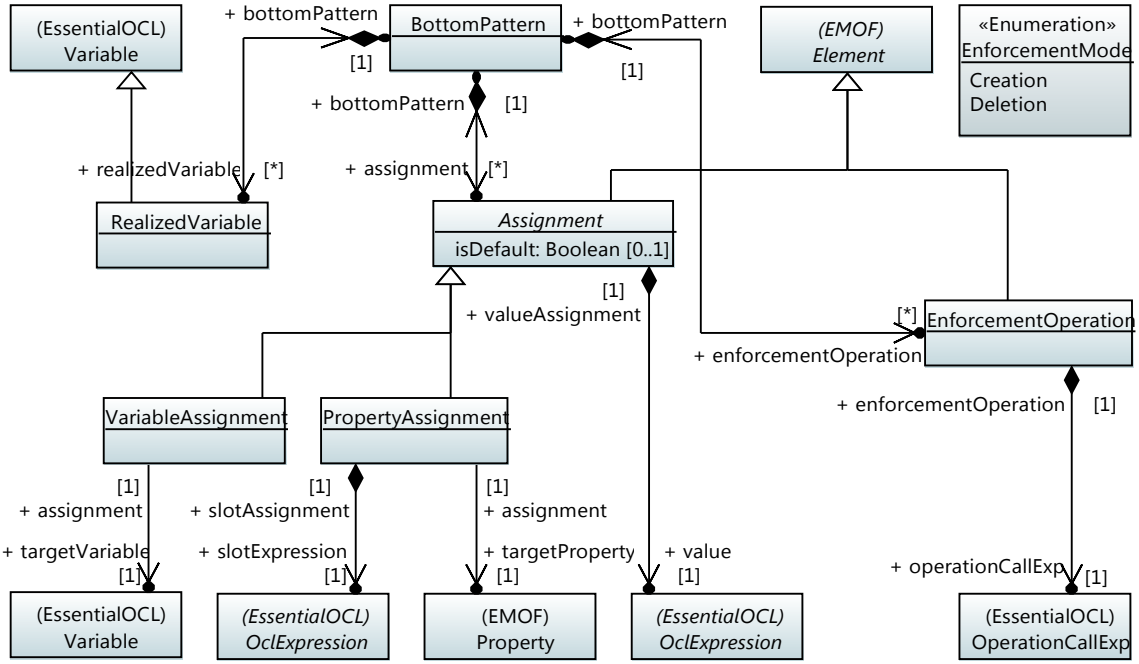


QVTRelation 2QVTRelation 3

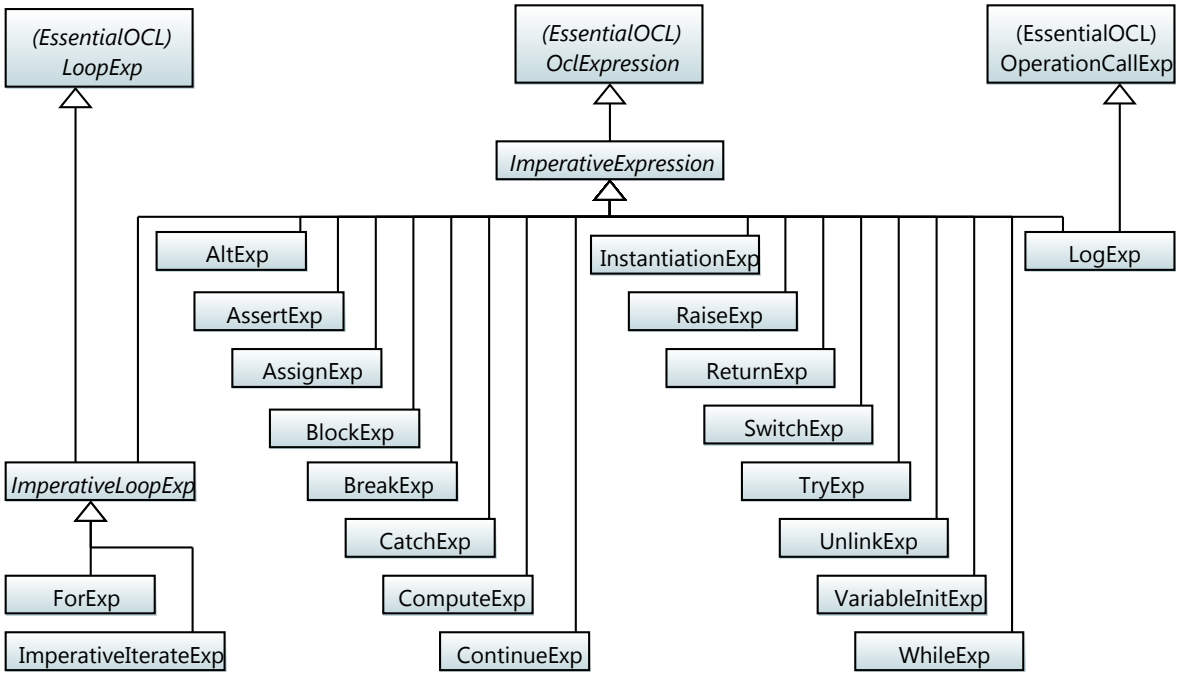
Replace Fig 9.2 and 9.3 by the following two diagrams positioning the diagrams roughly where Fig 9.2 was at the start of the Abstract Syntax subclause.



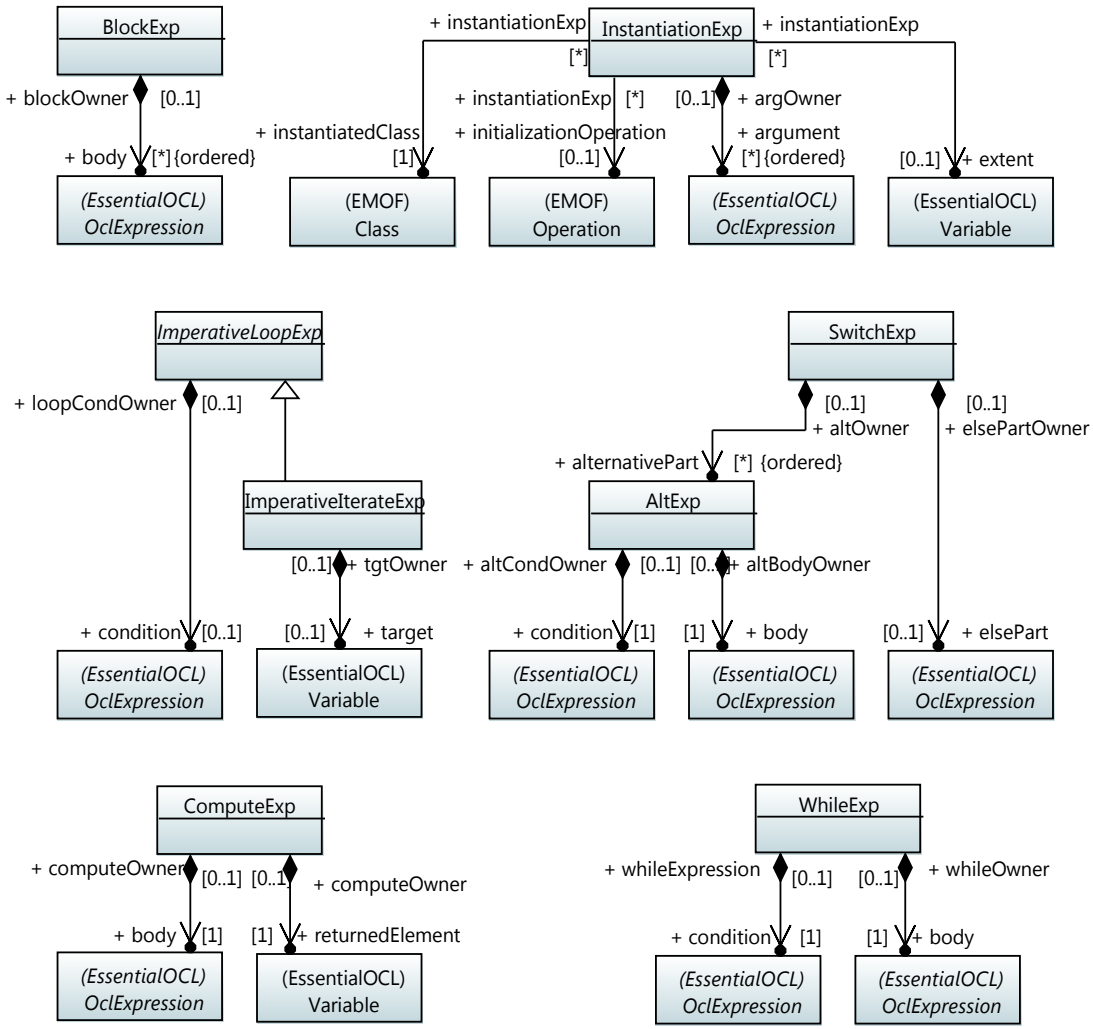
QVTCore 1



QVT Core 2

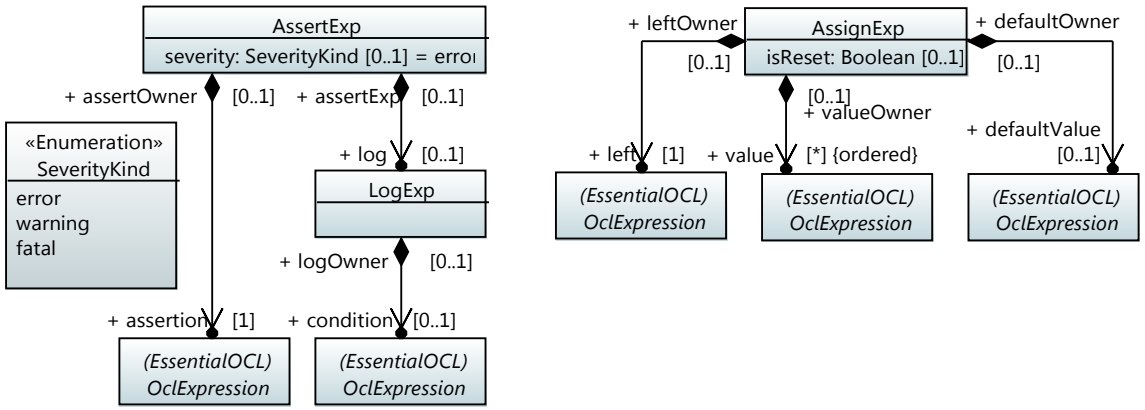


Replace Fig 8.4, 8.5, 8.6, 8.7 by the following six diagrams positioning the diagrams roughly where Fig 8.4 was at the start of the Abstract Syntax subclause.ImperativeOCL 1

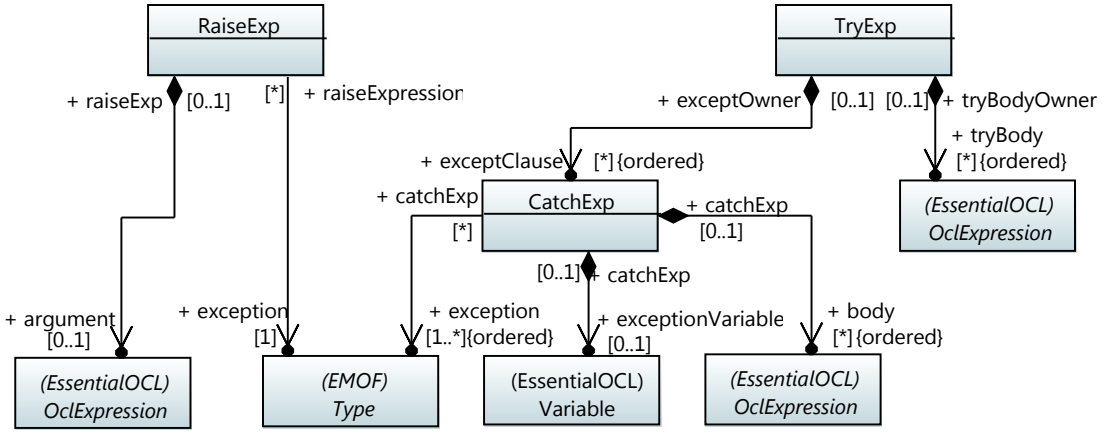


ImperativeOCL 2

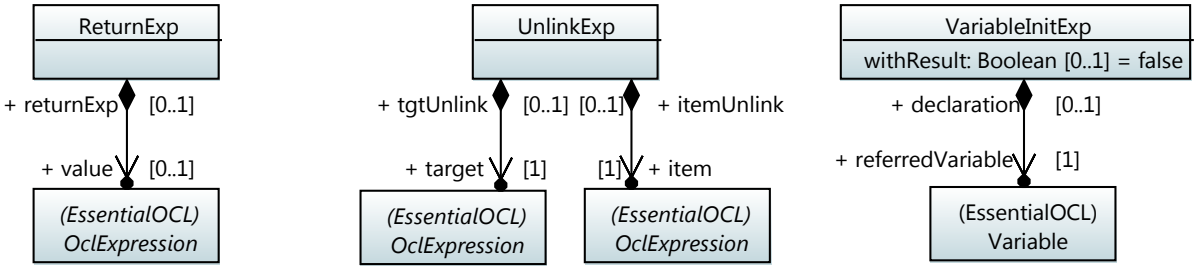
ImperativeOCL 3



ImperativeOCL 4

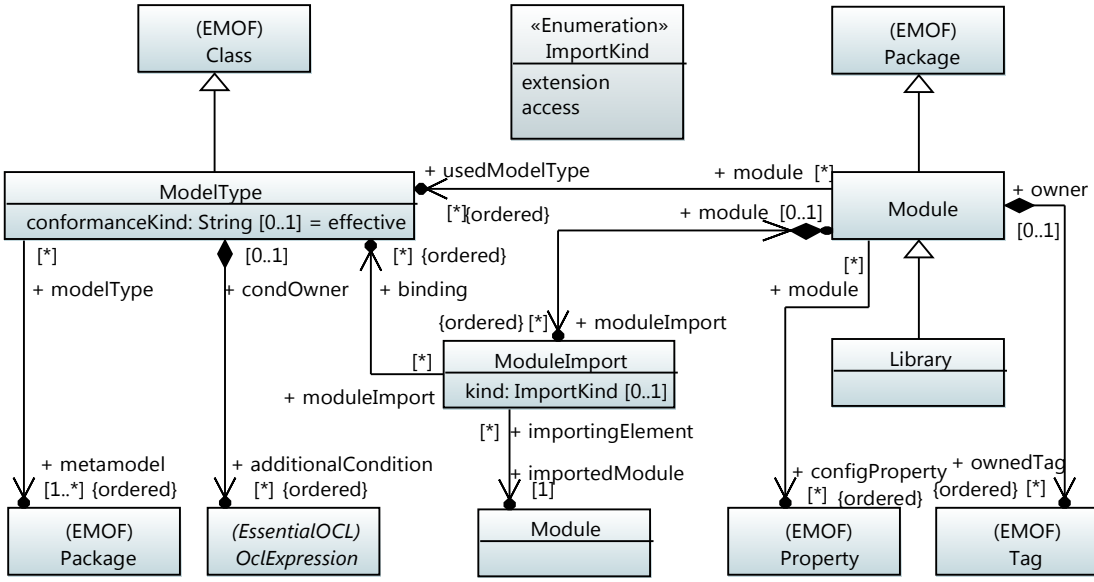


ImperativeOCL 5



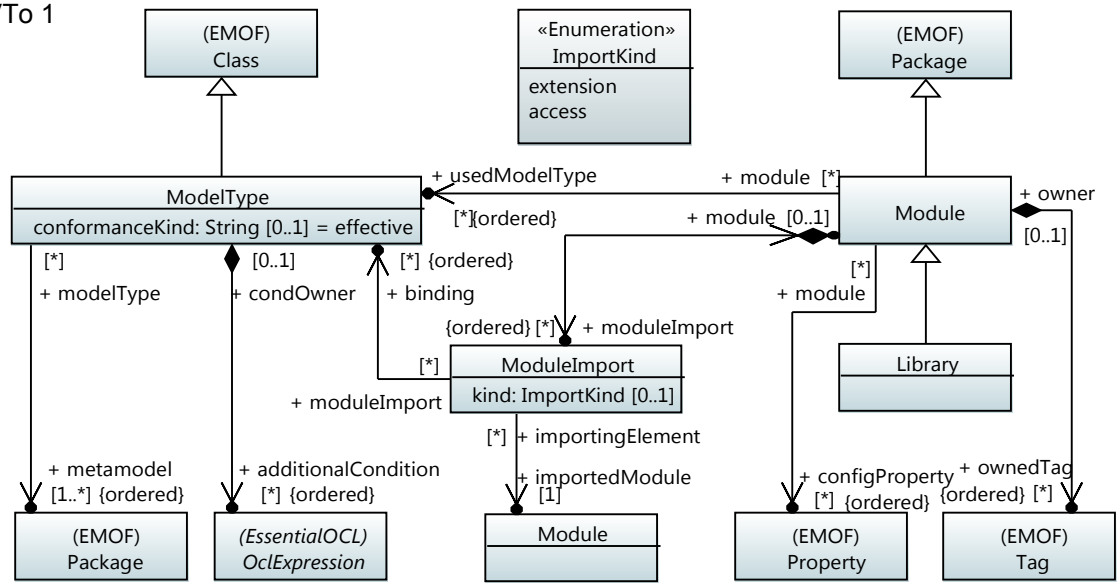
ImperativeOCL 6

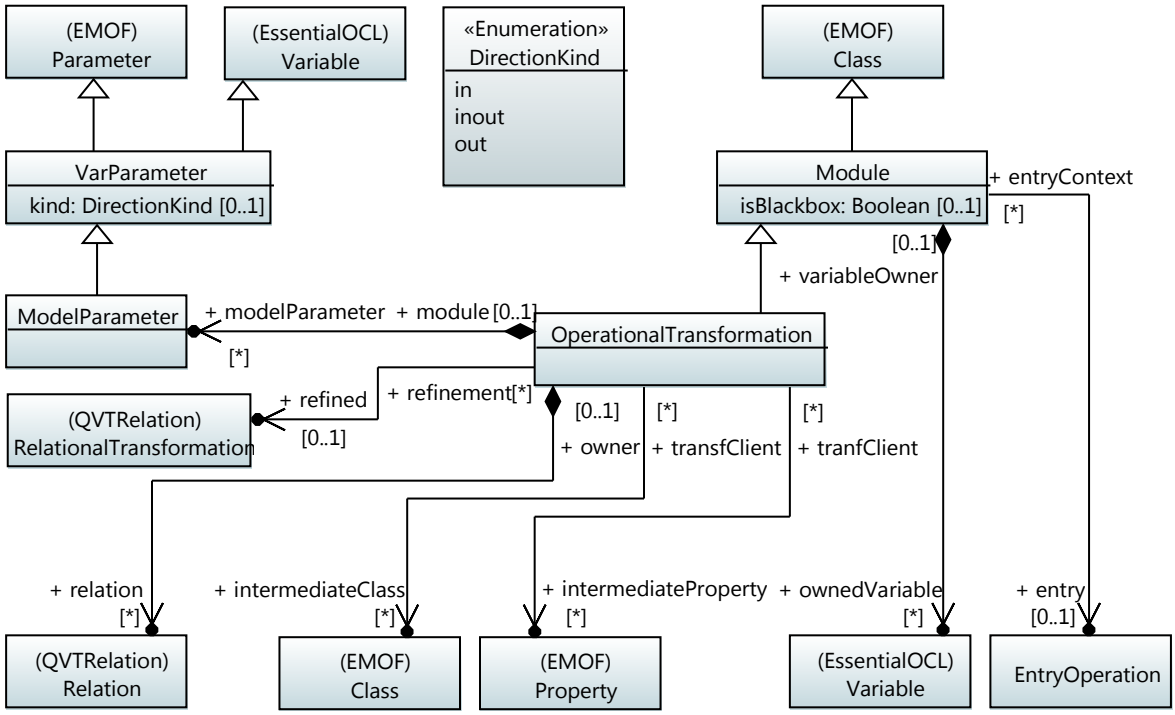
Replace Fig 8.1, 8.2, 8.3 by the following five diagrams positioning the diagrams roughly where Fig 8.1 was at the start of the Abstract Syntax subclause.



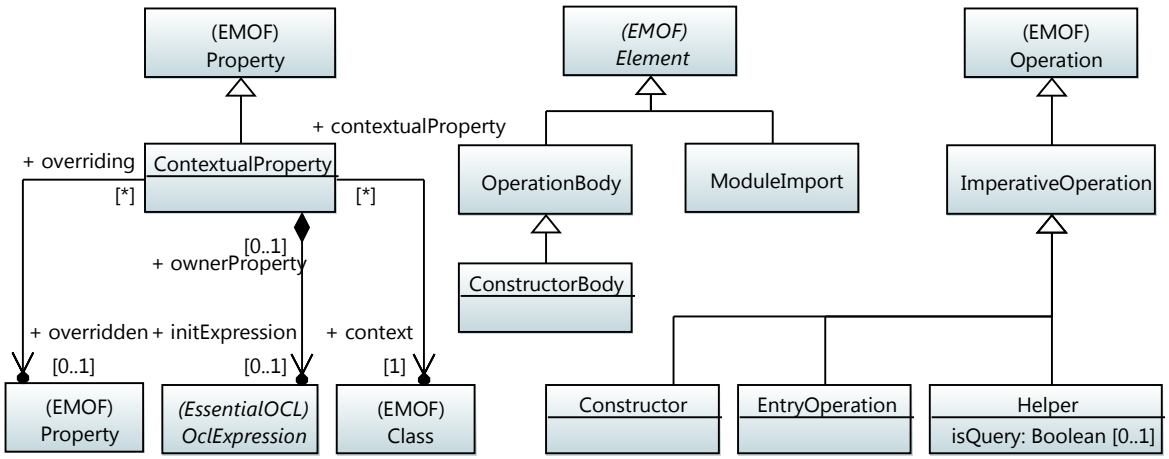
Replace Fig 8.1, 8.2, 8.3 by the following five diagrams positioning the diagrams roughly where Fig 8.1 was at the start of the Abstract Syntax subclause.

QVTo 1

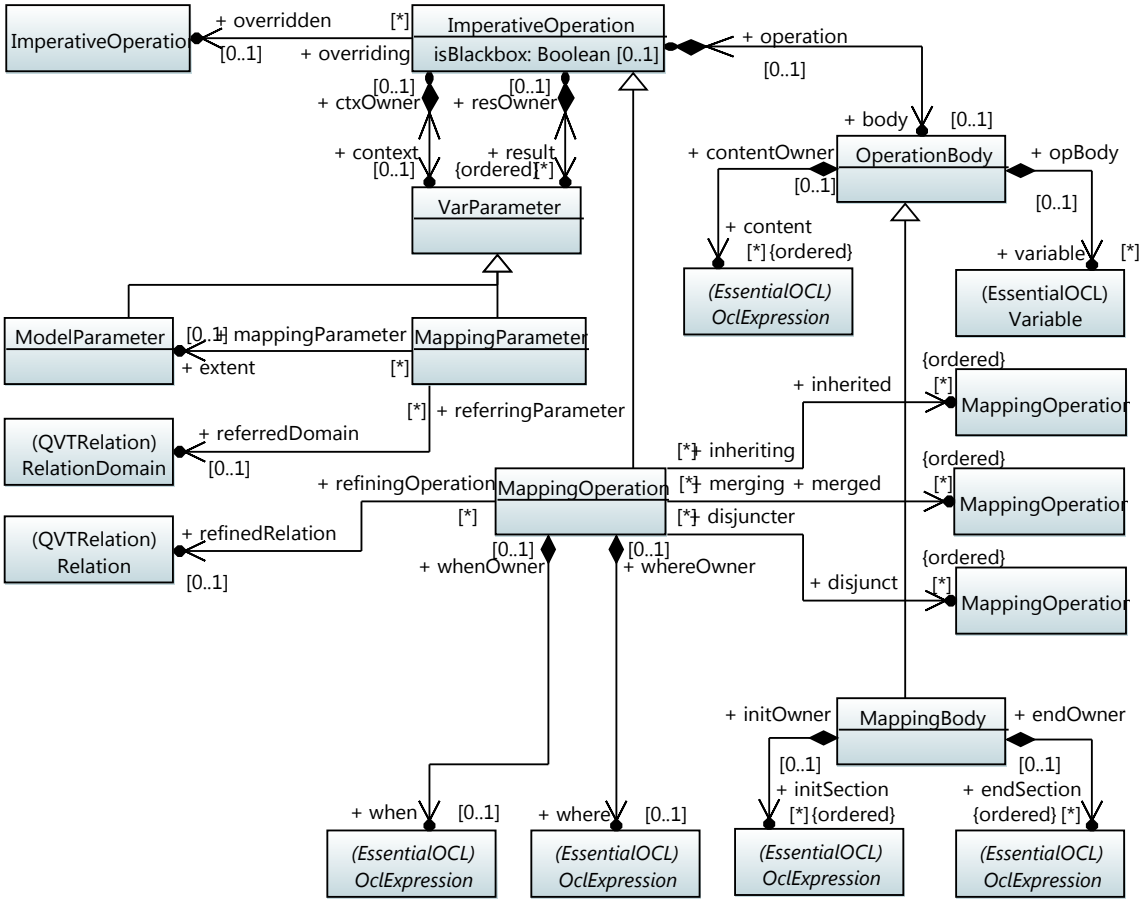




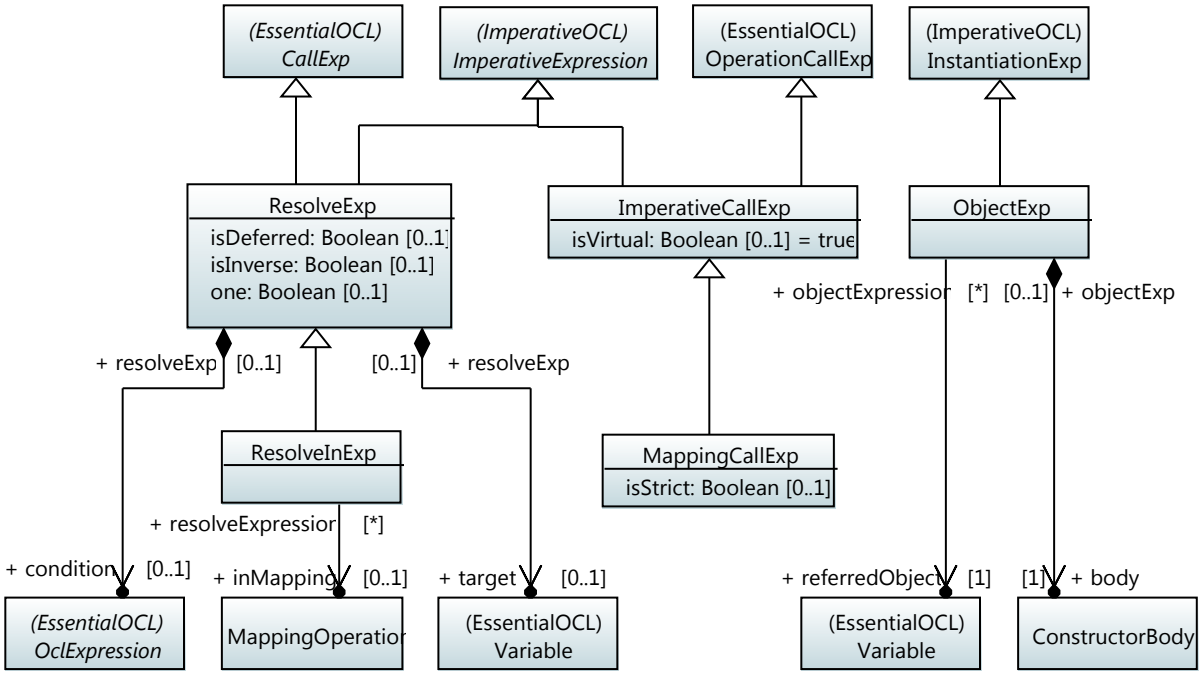
QVTo 4


(EMOF)
Property
(EMOF)
Element
(EMOF)
Operation
+ contextualProperty
+ overriding
ContextualProperty
[*]
[*]
OperationBody
ModuleImport
ImperativeOperation
[0..1]
+ ownerProperty
ConstructorBody
+ overridden
[0..1]
+ initExpression
[0..1]
+ context
[1]
(EMOF)
Property
(EssentialOCL)
OclExpression
(EMOF)
Class
Constructor
EntryOperation
Helper
isQuery: Boolean [0..1]

QVTo 2


ImperativeOperatio
+ overridden
[0..1]
[*]
+ overriding
ImperativeOperation
isBlackbox: Boolean [0..1]
+ operation
[0..1]
[0..1]
+ ctxOwner
[0..1]
+ resOwner
+ context
[0..1]
+ result
{ordered}[*]
VarParameter
+ body
[0..1]
+ contentOwner
[0..1]
OperationBody
+ opBody
[0..1]
+ content
[*]{ordered}
(EssentialOCL)
OclExpression
+ variable
[*]
(EssentialOCL)
Variable
ModelParameter
[0..1]
+ mappingParameter
+ extent
[*]
MappingParameter
[*]
+ referringParameter
{ordered}
[*]
+ inherited
MappingOperation
(QVTRelation)
RelationDomain
+ referredDomain
[0..1]
[*]
+ inheriting
+ refiningOperation
[*]
+ merging
+ merged
[*]
+ disjuncter
{ordered}
[*]
MappingOperation
(QVTRelation)
Relation
+ refinedRelation
[0..1]
[0..1]
+ whenOwner
[0..1]
+ whereOwner
+ disjunct
{ordered}
[*]
MappingOperation
+ initOwner
MappingBody
+ endOwner
[0..1]
+ initSection
[*]{ordered}
[0..1]
+ endSection
{ordered}[*]
+ when
[0..1]
+ where
[0..1]
(EssentialOCL)
OclExpression
(EssentialOCL)
OclExpression
(EssentialOCL)
OclExpression
(EssentialOCL)
OclExpression

QVTo 3



QVTo 5

Disposition: **Resolved**

Issue 12522: Errors and anomalies in QVT 1.0 07-07-08 ZIP qvtbase.ecore.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Use of automated tooling to support comparison of the models developed initially as part of the Eclipse GMT/UMLX project and being transferred to the Eclipse QVT Declarative/QVT Operational Mappings Projects reveals the following errors and anomalies in qvt_metamodel.emof.xml in the 07-07-08 ZIP.

Note that these errors and anomalies are not the same as those separately reported for the QVT_1.0.mdl from which the EMOF was notionally auto-generated.

An Ecore file resolving these anomalies is attached.

'nsPrefix' for 'QVTBase' should be 'qvtb' rather than 'qvtbase'

'nsURI' for 'QVTBase' should be ' http://schema.omg.org/spec/QVT/1.0/qvtbase.xml' rather than 'http://www.schema.omg.org/spec/QVT/1.0/qvtbase'

'name' for 'QVTBase' should be 'QVTBase' rather than 'qvtbase'

'abstract' for 'Domain' should be 'true' rather than 'false'

'abstract' for 'Rule' should be 'true' rather than 'false'

'lowerBound' for 'Domain.typedModel' should be '1' rather than '0'

'lowerBound' for 'Rule.transformation' should be '0' rather than '1'

'ordered' for 'Pattern.bindsTo' should be 'false' rather than 'true'

'ordered' for 'Pattern.predicate' should be 'false' rather than 'true'

'ordered' for 'Rule.domain' should be 'false' rather than 'true'

'ordered' for 'Transformation.modelParameter' should be 'false' rather than 'true'

'ordered' for 'Transformation.ownedTag' should be 'false' rather than 'true'

'ordered' for 'Transformation.rule' should be 'false' rather than 'true'

'ordered' for 'TypedModel.dependsOn' should be 'false' rather than 'true'

'ordered' for 'TypedModel.usedPackage' should be 'false' rather than 'true'

Unnavigable 'opposite' of 'Rule.overrides' should be modelled

Unnavigable 'opposite' of 'Transformation.extends' should be modelled

Unnavigable 'opposite' of 'TypedModel.dependsOn' should be modelled

Resolution:

These changes mostly affect non-normative files which were corrected when QVT 1.1 issued revised files based on Eclipse QVT contributions. However a few changes remain to be resolved in the main text.

No: Domain.typedModel lowerbound of 0 is correct; primitive domains have no TypedModel

Rule.transformation lowerbound change is Issue 11825.

Revised Text:

In Fig 7.4 change

- Rule.transformation multiplicity from 1 to 0..1.
- Transformation.modelParameter to not ordered.
- Rule.overriden to overridden.

In 7.11.1.1 Transformation change

```
transformation: Transformation[1]
```

to

```
transformation: Transformation[0..1]
```

In 7.11.1.4 Rule change

```
transformation: Transformation[1]
```

to

```
transformation: Transformation[0..1]
```

In the non-normative files change

- Transformation.modelParameter to not ordered.

Disposition: **Resolved**

Issue 12523: Errors and anomalies in QVT 1.0 07-07-08 ZIP qvttemplate.ecore.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Use of automated tooling to support comparison of the models developed initially as part of the Eclipse GMT/UMLX project and being transferred to the Eclipse QVT Declarative/QVT Operational Mappings Projects reveals the following errors and anomalies in qvt_metamodel.emof.xml in the 07-07-08 ZIP.

Note that these errors and anomalies are not the same as those separately reported for the QVT_1.0.mdl from which the EMOF was notionally auto-generated.

An Ecore file resolving these anomalies is attached.

'nsURI' for 'QVTTemplate' should be ' http://schema.omg.org/spec/QVT/1.0/qvttemplate.xml' rather than ' http://www.schema.omg.org/spec/QVT/1.0/qvtrelation'

'nsPrefix' for 'QVTTemplate' should be 'qvtt' rather than 'qvttemplate'

'name' for 'QVTTemplate' should be 'QVTTemplate' rather than 'qvttemplate'

'eType' for 'CollectionTemplateExp.rest' should be 'Variable' rather than 'OclExpression'

'CollectionTemplateExp.kind' should be undefined

'lowerBound' for 'CollectionTemplateExp.member' should be '0' rather than '1'

'lowerBound' for 'CollectionTemplateExp.rest' should be '0' rather than '1'

'lowerBound' for 'CollectionTemplateExp.referredCollectionType' should be '1' rather than '0'

'ordered' for 'CollectionTemplateExp.member' should be 'false' rather than 'true'

'ordered' for 'ObjectTemplateExp.part' should be 'false' rather than 'true'

Unnavigable 'opposite' of 'CollectionTemplateExp.member' should be modelled

Unnavigable 'opposite' of 'CollectionTemplateExp.rest' should be modelled

Unnavigable 'opposite' of 'PropertyTemplateItem.referredProperty' should be modelled

Unnavigable 'opposite' of 'PropertyTemplateItem.value' should be modelled

Unnavigable 'opposite' of 'TemplateExp.where' should be modelled

Resolution:

These changes mostly affect non-normative files which were corrected when QVT 1.1 issued revised files based on Eclipse QVT contributions. However a few changes remain to be resolved in the main text.

CollectionTemplateExp.rest/member lowerBound is Issue 11826.

PropertyTemplateItem.value multiplicity is right in diagram not text.

Revised Text:

In Fig 7.6 change

- Correct part/objContainer positions; should show ObjectTemplateExp.part
- Correct CollectionTemplateExp.referredCollectionType to multiplicity 1.
- Correct CollectionTemplateExp.member to multiplicity *.

- Correct CollectionTemplateExp.rest to multiplicity 0..1.

In 7.11.2.3 CollectionTemplateExp change

```
member : OclExpression [1..*] {composes}
…
rest : Variable [1]
```

to

```
member : OclExpression [*] {composes}
…
rest : Variable [0..1]
```

In 7.11.2.4 PropertyTemplateItem change

```
value: OclExpression [0..1] {composes}
```

to

```
value: OclExpression [1] {composes}
```

Disposition: **Resolved**

Issue 12524: Errors and anomalies in QVT 1.0 07-07-08 ZIP qvtrelation.ecore.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Use of automated tooling to support comparison of the models developed initially as part of the Eclipse GMT/UMLX project and being transferred to the Eclipse QVT Declarative/QVT Operational Mappings Projects reveals the following errors and anomalies in qvt_metamodel.emof.xml in the 07-07-08 ZIP.

Note that these errors and anomalies are not the same as those separately reported for the QVT_1.0.mdl from which the EMOF was notionally auto-generated.

An Ecore file resolving these anomalies is attached.

'nsURI' for 'QVTRelation' should be ' http://schema.omg.org/spec/QVT/1.0/qvtrelation.xml' rather than ' http://www.schema.omg.org/spec/QVT/1.0/qvtrelation'

'nsPrefix' for 'QVTRelation' should be 'qvtr' rather than 'qvtrelation'

'name' for 'QVTRelation' should be 'QVTRelation' rather than 'qvtrelation'

'lowerBound' for 'RelationCallExp.argument' should be '2' rather than '0'

'lowerBound' for 'RelationCallExp.referredRelation' should be '1' rather than '0'

'lowerBound' for 'RelationDomain.pattern' should be '1' rather than '0'

'containment' for 'Relation.operationalImpl' should be 'true' rather than 'false'

'transient' for 'RelationImplementation.relation' should be 'true' rather than 'false'

'ordered' for 'Key.part' should be 'false' rather than 'true'

'ordered' for 'Relation.operationalImpl' should be 'false' rather than 'true'

'ordered' for 'Relation.variable' should be 'false' rather than 'true'

'ordered' for 'RelationDomain.defaultAssignment' should be 'false' rather than 'true'

'ordered' for 'RelationalTransformation.ownedKey' should be 'false' rather than 'true'

Unnavigable 'opposite' of 'Relation.when' should be modelled

Unnavigable 'opposite' of 'Relation.where' should be modelled

Unnavigable 'opposite' of 'RelationDomain.defaultAssignment' should be modelled

Unnavigable 'opposite' of 'RelationDomainAssignment.valueExp' should be modelled

Unnavigable 'opposite' of 'RelationDomainAssignment.variable' should be modelled

Resolution:

These changes mostly affect non-normative files which were corrected when QVT 1.1 issued revised files based on Eclipse QVT contributions. However a few changes remain to be resolved in the main text.

Revised Text:

In Fig 7.7

- Remove the partial diagram artifact at the bottom of the diagram
- Correct the RelationImplementation.relation multiplicity to 1

- Correct the RelationCallExp.referredRelation multiplicity to 1
- Correct the RelationCallExp.argument multiplicity to 2..*

In 7.11.3.1 RelationalTransformation change

```
key: Key [*] {composes}
```

to

```
ownedKey: Key [*] {composes}
```

In 7.11.3.2 Relation change

```
/domain: Domain [*] {composes} (from Rule)
```

to

```
/domain: RelationDomain [*] {composes} (from Rule)
```

In the non-normative files change

- RelationImplementation.relation multiplicity to 1

Disposition: **Resolved**

Issue 12525: Errors and anomalies in QVT 1.0 07-07-08 ZIP qvtcore.ecore.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Use of automated tooling to support comparison of the models developed initially as part of the Eclipse GMT/UMLX project and being transferred to the Eclipse QVT Declarative/QVT Operational Mappings Projects reveals the following errors and anomalies in qvt_metamodel.emof.xml in the 07-07-08 ZIP.

Note that these errors and anomalies are not the same as those separately reported for the QVT_1.0.mdl from which the EMOF was notionally auto-generated.

An Ecore file resolving these anomalies is attached.

'nsURI' for 'QVTCore' should be ' http://schema.omg.org/spec/QVT/1.0/qvtcore.xml' rather than ' http://www.schema.omg.org/spec/QVT/1.0/qvtcore'

'nsPrefix' for 'QVTCore' should be 'qvtc' rather than 'qvtcore'

'name' for 'QVTCore' should be 'QVTCore' rather than 'qvtcore'

'eSuperTypes' for 'Assignment' should be 'Element' rather than nothing

'eSuperTypes' for 'EnforcementOperation' should be 'Element' rather than nothing

'abstract' for 'Assignment' should be 'true' rather than 'false'

'containment' for 'EnforcementOperation.operationCallExp' should be 'true' rather than 'false'

'containment' for 'Mapping.local' should be 'true' rather than 'false'

'transient' for 'Mapping.context' should be 'true' rather than 'false'

'Assignment.slotExpression' should be undefined

'PropertyAssignment.slotExpression' should be defined

'CorePattern.variable' should be defined

'Mapping.refinement' should be defined

'Mapping.refinement' should be the 'opposite' of 'Mapping.specification'

'ordered' for 'BottomPattern.assignment' should be 'false' rather than 'true'

'ordered' for 'BottomPattern.enforcementOperation' should be 'false' rather than 'true'

'ordered' for 'BottomPattern.realizedVariable' should be 'false' rather than 'true'

'ordered' for 'Mapping.local' should be 'false' rather than 'true'

'ordered' for 'Mapping.specification' should be 'false' rather than 'true'

Unnavigable 'opposite' of 'Assignment.value' should be modelled

Unnavigable 'opposite' of 'PropertyAssignment.targetProperty' should be modelled

Unnavigable 'opposite' of 'VariableAssignment.targetVariable' should be modelled

Resolution:

These changes mostly affect non-normative files which were corrected when QVT 1.1 issued revised files based on Eclipse QVT contributions. However a few changes remain to be resolved in the main text.

CorePattern.variable is Issue 10938

Assignment.slotExpression is Issue 11108

Revised Text:

In Fig 9.2

- add a CorePattern to Variable composition arc for CorePattern.variable

In Figure 9.3

- move the Assignment end of OclExpression.slotExpression to PropertyAssignment.
- Change the EnforcementOperation.bottomPattern multiplicity to 1.

In 9.17.1 CorePattern add

```
variable: Variable [*] {composes}
```

In 9.17.6 Mapping change

```
domain : Domain [*] {composes} (From QVTBase)
```

to

```
/domain : CoreDomain [*] {composes} (From QVTBase)
```

In 9.17.7 RealizedVariable change

```
bottomPattern : BottomPattern [*] {composes}
```

to

```
bottomPattern : BottomPattern [1]
```

In 9.17.8 Assignment change

```
bottomPattern: BottomPattern
```

to

```
bottomPattern : BottomPattern [1]
```

From 9.17.8 Assignment move

```
slotExpression: OclExpression [1] {composes}
```

An OCL expression identifying the object whose property value is to be assigned.

to 9.17.9 PropertyAssignment

```
slotExpression: OclExpression [1] {composes}
```

An OCL expression identifying the object whose property value is to be assigned.

In the non-normative files change

- EnforcementOperation.bottomPattern multiplicity to 1

Disposition: **Resolved**

Issue 12526: Errors and anomalies in QVT 1.0 07-07-08 ZIP imperativeocl.ecore.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Use of automated tooling to support comparison of the models developed initially as part of the Eclipse GMT/UMLX project and being transferred to the Eclipse QVT Declarative/QVT Operational Mappings Projects reveals the following errors and anomalies in qvt_metamodel.emof.xml in the 07-07-08 ZIP.

Note that these errors and anomalies are not the same as those separately reported for the QVT_1.0.mdl from which the EMOF was notionally auto-generated.

An Ecore file resolving these anomalies is attached.

'nsURI' for 'ImperativeOCL' should be 'http://schema.omg.org/spec/QVT/1.0/imperativeocl.xml' rather than 'http://www.schema.omg.org/spec/QVT/1.0/imperativeocl'

'nsPrefix' for 'ImperativeOCL' should be 'impocl' rather than 'imperativeocl'

'name' for 'ImperativeOCL' should be 'ImperativeOCL' rather than 'imperativeocl'

'name' for 'TryExp.exceptClause' should be 'exceptClause' rather than undefined

'containment' for 'UnpackExp.targetVariable' should be 'false' rather than 'true'

'ordered' for 'DictLiteralExp.part' should be 'false' rather than 'true'

'lowerBound' for 'ReturnExp.value' should be '0' rather than '1'

'defaultValueLiteral' for 'AssertExp.severity' should be 'error' rather than undefined

'defaultValueLiteral' for 'VariableInitExp.withResult' should be 'false' rather than undefined

'eSuperTypes' for 'SwitchExp' should be 'ImperativeExpression' rather than 'CallExp','ImperativeExpression'

Unnavigable 'opposite' of 'AltExp.body' should be modelled

Unnavigable 'opposite' of 'AltExp.condition' should be modelled

Unnavigable 'opposite' of 'AssertExp.assertion' should be modelled

Unnavigable 'opposite' of 'AssignExp.defaultValue' should be modelled

Unnavigable 'opposite' of 'AssignExp.left' should be modelled

Unnavigable 'opposite' of 'AssignExp.value' should be modelled

Unnavigable 'opposite' of 'BlockExp.body' should be modelled

Unnavigable 'opposite' of 'CatchExp.exception' should be modelled

Unnavigable 'opposite' of 'ComputeExp.body' should be modelled

Unnavigable 'opposite' of 'ComputeExp.returnedElement' should be modelled

Unnavigable 'opposite' of 'DictionaryType.keyType' should be modelled

Unnavigable 'opposite' of 'DictLiteralExp.part' should be modelled

Unnavigable 'opposite' of 'DictLiteralPart.key' should be modelled

Unnavigable 'opposite' of 'DictLiteralPart.value' should be modelled

Unnavigable 'opposite' of 'ImperativeIterateExp.target' should be modelled

Unnavigable 'opposite' of 'ImperativeLoopExp.condition' should be modelled

Unnavigable 'opposite' of 'InstantiationExp.argument' should be modelled

Unnavigable 'opposite' of 'LogExp.condition' should be modelled

Unnavigable 'opposite' of 'OrderedTupleLiteralExp.part' should be modelled

Unnavigable 'opposite' of 'OrderedTupleLiteralPart.value' should be modelled

Unnavigable 'opposite' of 'OrderedTupleType.elementType' should be modelled

Unnavigable 'opposite' of 'RaiseExp.exception' should be modelled

Unnavigable 'opposite' of 'SwitchExp.alternativePart' should be modelled

Unnavigable 'opposite' of 'SwitchExp.elsePart' should be modelled

Unnavigable 'opposite' of 'TryExp.exceptClause' should be modelled

Unnavigable 'opposite' of 'TryExp.tryBody' should be modelled

Unnavigable 'opposite' of 'UnlinkExp.item' should be modelled

Unnavigable 'opposite' of 'UnlinkExp.target' should be modelled

Unnavigable 'opposite' of 'VariableInitExp.referredVariable' should be modelled

Unnavigable 'opposite' of 'WhileExp.body' should be modelled

Unnavigable 'opposite' of 'WhileExp.condition' should be modelled

Resolution:

These changes mostly affect non-normative files which were corrected when QVT 1.1 issued revised files based on Eclipse QVT contributions. However a few changes remain to be resolved in the main text.

CatchExp.exceptionVariable/CatchExp.exception changes are Issue 15997

Remove UnpackExp is Issue 13268

Remove OrderedTupleType is Issue 13268

Remove OrderedTupleLiteralExp is Issue 13268

Remove OrderedTupleLiteralPart is Issue 13268

Add InstantiationExp.initializationOperation is Issue 19021

Revised Text:

In Fig 8.5

- Position * and {ordered} unambiguously for BlockExp.body
- Add Operation
- Add unidirectional reference from InstantiationExp to Operation
  forward role initializationOperation [0..1]
  reverse role instantiationExp [*]

In Fig 8.6

- remove spurious artifact at RHS
- add CatchExp.exceptionVariable
- change CatchExp.exception multiplicity to +
- change ReturnExp.value multiplicity to 0..1
- remove UnpackExp and its associations

In Fig 8.7

- remove OrderedTupleType and its associations
- remove OrderedTupleLiteralExp and its associations
- remove OrderedTupleLiteralPart and its associations

In 8.2.2.7 ImperativeIterateExp change

```
target : Variable [0..1]
```

to

```
target : Variable [0..1] {composes}
```

In 8.2.2.8 SwitchExp change

```
elsePart : OclExpresion {composes} [0..1]
```

to

```
elsePart : OclExpression {composes} [0..1]
```

In 8.2.2.14 CatchExp and the QVT 1.1 models add

```
exceptionVariable : Variable [0..1]
```

In 8.2.2.23 InstantiationExp add

```
initializationOperation : Operation [0..1]
```

The initialization operation that uses the arguments to initialize the object after creation. The initialization operation may be omitted when implicit initialization occurs with no arguments.

In 8.2.2.24 Typedef change

```
condition: OclExpression [1]{composes}
```

to

```
condition: OclExpression [0..1]{composes}
```

In 8.2.2.29 DictLiteralExp change

```
part : DictLiteralPart [*] {composes,ordered}
```

to

```
part : DictLiteralPart [*] {composes}
```

In the non-normative models change

- add CatchExp.exceptionVariable
- change CatchExp.exception multiplicity to +
- add InstantiationExp.initializationOperation
- remove UnpackExp
- remove OrderedTupleType
- remove OrderedTupleLiteralExp
- remove OrderedTupleLiteralPart

Disposition: **Resolved**

Issue 12527: Errors and anomalies in QVT 1.0 07-07-08 ZIP qvtoperational.ecore.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Use of automated tooling to support comparison of the models developed initially as part of the Eclipse GMT/UMLX project and being transferred to the Eclipse QVT Declarative/QVT Operational Mappings Projects reveals the following errors and anomalies in qvt_metamodel.emof.xml in the 07-07-08 ZIP.

Note that these errors and anomalies are not the same as those separately reported for the QVT_1.0.mdl from which the EMOF was notionally auto-generated.

An Ecore file resolving these anomalies is attached.

'nsURI' for 'QVTOperational' should be 'http://schema.omg.org/spec/QVT/1.0/qvtoperational.xml' rather than 'http://www.schema.omg.org/spec/QVT/1.0/qvtoperational'

'nsPrefix' for 'QVTOperational' should be 'qvto' rather than 'qvtoperational'

'name' for 'QVTOperational' should be 'QVTOperational' rather than 'qvtoperational'

'name' for 'MappingParameter.referredDomain' should be 'referredDomain' rather than 'refinedDomain'

'eType' for 'Module.entry' should be 'EntryOperation' rather than 'Operation'

'eSuperTypes' for 'ImperativeCallExp' should be 'OperationCallExp','ImperativeExpression' rather than 'OperationCallExp'

'eSuperTypes' for 'MappingOperation' should be 'ImperativeOperation' rather than 'ImperativeOperation','Operation','NamedElement'

'eSuperTypes' for 'ModelType' should be 'Class' rather than 'Class','URIExtent'

'eSuperTypes' for 'ResolveExp' should be 'CallExp','ImperativeExpression' rather than 'CallExp'

'upperBound' for 'MappingOperation.when' should be '1' rather than '-1'

'upperBound' for 'MappingOperation.where' should be '1' rather than '-1'

'lowerBound' for 'ModelType.metamodel' should be '1' rather than '0'

'defaultValueLiteral' for 'ImperativeCallExp.isVirtual' should be 'true' rather than undefined

'defaultValueLiteral' for 'ModelType.conformanceKind' should be 'effective' rather than undefined

'ordered' for 'Module.ownedVariable' should be 'false' rather than 'true'

'ordered' for 'OperationBody.variable' should be 'false' rather than 'true'

Unnavigable 'opposite' of 'ContextualProperty.initExpression' should be modelled

Unnavigable 'opposite' of 'ContextualProperty.overridden' should be modelled

Unnavigable 'opposite' of 'ImperativeOperation.overridden' should be modelled

Unnavigable 'opposite' of 'MappingBody.endSection' should be modelled

Unnavigable 'opposite' of 'MappingBody.initSection' should be modelled

Unnavigable 'opposite' of 'MappingOperation.disjunct' should be modelled

Unnavigable 'opposite' of 'MappingOperation.inherited' should be modelled

Unnavigable 'opposite' of 'MappingOperation.merged' should be modelled

Unnavigable 'opposite' of 'MappingOperation.refinedRelation' should be modelled

Unnavigable 'opposite' of 'MappingOperation.when' should be modelled

Unnavigable 'opposite' of 'MappingOperation.where' should be modelled

Unnavigable 'opposite' of 'MappingParameter.referredDomain' should be modelled

Unnavigable 'opposite' of 'ModelType.additionalCondition' should be modelled

Unnavigable 'opposite' of 'ModuleImport.importedModule' should be modelled

Unnavigable 'opposite' of 'Module.entry' should be modelled

Unnavigable 'opposite' of 'Module.ownedTag' should be modelled

Unnavigable 'opposite' of 'Module.ownedVariable' should be modelled

Unnavigable 'opposite' of 'ObjectExp.referredObject' should be modelled

Unnavigable 'opposite' of 'OperationBody.content' should be modelled

Unnavigable 'opposite' of 'OperationBody.variable' should be modelled

Unnavigable 'opposite' of 'OperationalTransformation.intermediateClass' should be modelled

Unnavigable 'opposite' of 'OperationalTransformation.intermediateProperty' should be modelled

Unnavigable 'opposite' of 'OperationalTransformation.modelParameter' should be modelled

Unnavigable 'opposite' of 'OperationalTransformation.refined' should be modelled

Unnavigable 'opposite' of 'OperationalTransformation.relation' should be modelled

Unnavigable 'opposite' of 'ResolveInExp.inMapping' should be modelled

Resolution:

These changes mostly affect non-normative files which were corrected when QVT 1.1 issued revised files based on Eclipse QVT contributions. However a few changes remain to be resolved in the main text.

ContextualProperty .initExpression is Issue 13270

ObjectExp.instantiatedClass is Issue 13276

ObjectExp. initializationOperation is Issue 19021

ObjectExp. body is Issue 19022

Revised Text:

In Fig 8.1

- Replace Operation by EntryOperation as target of Module.entry
- Change ModelType.metamodel multiplicity to 1..*

In Fig 8.2

- Disentangle MappingBody.endSection/OperationBody.content and {ordered}
- Add {ordered} to OperationBody.content

Move from 8.2.1.1 OperationalTransformation to 8.2.1.3 Module

```
ownedVariable : Variable [0..*] {composes}
```

The list of variables owned by this module.

In 8.2.1.14 ContextualProperty change

```
overridden: Property [1]
```

to

```
overridden: Property [0..1]
```

In 8.2.1.14 ContextualProperty add

initExpression: OclExpression [0..1] {composes}

In 8.2.1.15 MappingOperation change

`isBlackbox: Boolean`

to

`/isBlackbox: Boolean (from ImperativeOperation)`

In 8.2.1.21 MappingCallExp Superclasses change

`OperationCallExp`

to

`ImperativeCallExp`

In 8.2.1.24 ObjectExp change

`/instantiatedClass: Class [0..1](from InstanciationExp)`

to

`/instantiatedClass: Class [1](from InstantiationExp)`

In 8.2.1.24 ObjectExp add

`body: ConstructorBody[1] { composes }`

The constructor to execute.

In 8.2.1.24 ObjectExp add

`/initializationOperation : Constructor [0..1] (from InstantiationExp)`

The constructor that uses the arguments to initialize the object after creation. The constructor may be omitted when implicit construction occurs with no arguments.

In the non-normative files change

- ObjectExp.referredObject multiplicity to 1

Disposition: **Resolved**

Issue 12571: QVT 1.0 9.* Missing query concrete syntax.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

There is no support for queries in the QVT Core concrete syntax.

Suggest: re-use the concrete syntax of QVT Relation, except that the query is defined at global scope and so must be qualified by the name of a transformation defined in the same source unit.

example:

query txName::getStringSize (someString:String): Integer {

  someString -> size()

}

Resolution:

Simple change. Names are left vague pending resolution of Issue 10935.

Revised Text:

In 9.18 Concrete Syntax add (NB TopLevel introduced by Issue 10942)

```
TopLevel ::= (Transformation | Mapping | Query)*
Query ::= 'query' QueryName '(' [ ParamDeclaration (',' ParamDeclaration)*] ')'
':' TypeDeclaration (';' | '{' QueryOCLExpr '}')
ParamDeclaration ::= ParameterName ':' TypeDeclaration
```

Disposition: **Resolved**

Issue 12572: QVT 1.0 7.13.5 Transformation hierarchy.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The QVT Relation Concrete Syntax makes no provision for organisation of transformations in a package hierarchy; all transformations are presumed to be in root packages.

Suggest: change <identifier> to <transformationId> in both definition and reference of the <transformation> production, and define <transformationId> as <pathNameCS>.

Resolution:

Simple change.

Revised Text:

In 7.13.5 Relations Textual Syntax Grammar change

```
   <transformation> ::= 'transformation' <identifier> '(' <modelDecl> (','
   <modelDecl>)* ')' ['extends' <identifier>] '{' <keyDecl>* ( <relation> |
   <query> )* '}'
```

to

```
   <transformation> ::= 'transformation' <transformationId> '(' <modelDecl> (','
   <modelDecl>)* ')' ['extends' <transformationId>] '{' <keyDecl>* ( <relation> |
   <query> )* '}'
   <transformationId> ::= <pathNameCS>
```

Disposition: **Resolved**

Issue 12573: QVT 1.0 9.18 Missing transformation extension concrete syntax.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

There is no support for transformation extension in the QVT Core concrete syntax.

Suggest: re-use the concrete syntax of QVT Relation, allowing "extends x" to follow a transformation declaration.

example:

transformation yy::umlRdbms {

    uml imports umlMM;

    rdbms imports rdbmsMM;

} extends base1, xx::base2, base3

Resolution:

Simple change. Names are left vague pending resolution of Issue 10935.

Revised Text:

In 9.18 Concrete Syntax change

```
Transformation ::=
"transformation" TransformationName "{"
( Direction";" )*
"}"
```

to

```
Transformation ::=
"transformation" TransformationName "{"
( Direction";" )*
"}"
[ "extends"  TransformationName ("," TransformationName)*]
```

Disposition: **Resolved**

Issue 13182: QVTo Standard Library: Some operation's signatures seem to be erroneous.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

** QVTo Standard Library: Some operation's signatures seem to be erroneous. **

- the name in the signature of the operation allSubojectsOfType (8.3.4.7) has a typo. Rename correctly
- - the returned value in the signature of the operation raisedException (8.3.6.4) should better be Exception.
- the asList (8.3.8.5) operation's signature is not correct for the types OrderedSet(T), Sequence(T), Bag(T). They shouldn't have any parameter.
- -P.S: Why is this operation included in section 8.3.8 Operations List. I would recomend the creation of new sections for each collection type, instead.
- - OCLStdlib already defines toLower and toUpper operations. Since these operations may be considered as side-effects operations. I should clarify one of the possible situations: 1. toLower and toUpper are not intended to be side-effect operations. Remove them from the section. 2. toLower and toUpper are always intended to be side-effect operations, so that OCL Operations are discarded. This must be clarified. 3. both (side-effect and free side-effect) operations, are available in QVTtransformations. In this case I would change the name of QVTo std lib operations to distinguish.
- - In section (8.3.9) lastToUpper must be renamed as lastToLower.
- -P.S: Why all the QVTo Std Lib operations have a subsection number, excepting String's operations?

Resolution:

allSubojectsOfType – see Issue 13989 resolution.

raisedException – Yes. But the Status operations are misleadingly under Transformation. Add a missing section heading.

asList(T) – Yes, supersedes Issue 19146 resolution.

PS – see Issue 19146 resolution.

toLower – Yes. Make it clear that all String operations are immutable. Redirect OCL 2.4 synonyms to OCL 2.4 deprecating the synonyms. We can also fix some bad typos, Boolean rather than Integer/Real returns, references to the non-existent Float type and spurious matchXXX arguments.

lastToUpper – this seems to be a major bloat. Either name or description could change. Since the name exists with an obvious functionality, correct the description.

PS Yes

Revised Text:

After 8.3.6.3 Transformation wait insert

<<new 8.3.7 number>> Status

The following operations may be used to interrogate the Status object that synchronizes the end of a transformation.

In 8.3.6.4 Transformation raisedException change

Status::raisedException () : Class

to

Status::raisedException () : Exception

In the Issue 19146 replacement text, replace the erroneous trailing (T) parameter by an empty () parameter list in the following signatures:

```
List(T)::add(T)
List(T)::asList(T)
List(T)::clone(T)
List(T)::deepclone(T)
Collection(T)::asList(T)
Collection(T)::clone(T)
Collection(T)::deepclone(T)
Bag(T)::asList(T)
Bag(T)::clone(T)
Bag(T)::deepclone(T)
OrderedSet(T)::asList(T)
OrderedSet(T)::clone(T)
OrderedSet(T)::deepclone(T)
Sequence(T)::asList(T)
Sequence(T)::clone(T)
Sequence(T)::deepclone(T)
```

(Set needs no change.)

In the body of 8.3.9 replace

String::size () : Integer

The size operation returns the length of the sequence of characters represented by the object at hand.

Synonym operation: **length**()

by

String::length () : Integer

The length operation returns the length of the sequence of characters represented by the object at hand.

This is a synonym of the OCL String::size() operation. It is therefore deprecated.

In the body of 8.3.9 replace

String::toLower () : String

Converts all of the characters in this string to lowercase characters.

String::toUpper () : String

Converts all of the characters in this string to uppercase characters.

by

String::toLower () : String

Converts all of the characters in this string to lowercase characters.

This is a synonym of the OCL String::toLowerCase() operation. It is therefore deprecated.

String::toUpper () : String

Converts all of the characters in this string to uppercase characters.

This is a synonym of the OCL String::toUpperCase() operation. It is therefore deprecated.

In the body of 8.3.9 replace

String::lastToUpper () : String

Converts the last character in the string to a lowercase character.

by

String::lastToUpper () : String

Converts the last character in the string to an uppercase character.

In the body of 8.3.9 replace

String::matchBoolean (s:String) : Boolean •

Returns true if the string is “true,” “false,” “0,” or “1.” The method is not case sensitive.

String::matchInteger (i:Integer) : Boolean •

Returns true if the string represents an integer.

String::matchFloat (i:Integer) : Boolean

Returns true if the string represents a float.

by

String::matchBoolean () : Boolean •

Returns true if the string is “true,” “false,” “0,” or “1.” The method is not case sensitive.

String::matchInteger () : Boolean •

Returns true if the string represents an integer.

String::matchFloat) : Boolean

Returns true if the string represents a real.

This is a synonym of the matchReal() operation. It is therefore deprecated.

String::matchReal () : Boolean

Returns true if the string represents a real.

In the body of 8.3.9 replace

String::asInteger() : Boolean

Returns a Integer value if the string can be interpreted as as integer. Null otherwise.

String::asFloat() : Boolean

Returns a Float value if the string can be interpreted as as float. Null otherwise.

by

String::asInteger() : Integer

Returns a Integer value if the string can be interpreted as as integer. Null otherwise.

String::asFloat() : Real

Returns a Real value if the string can be interpreted as as real. Null otherwise.

This is a synonym of the asReal() operation. It is therefore deprecated.

String::asReal() : Real

Returns a Real value if the string can be interpreted as as real. Null otherwise.

In 8.3.9 give each operation a sub-sub-sub-section number as for other types.

Disposition: **Resolved**

Issue 13183: QVTo Standard Library. Clarification of the side-effect operations is needed.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

** QVTo Standard Library. Clarification of the side-effect operations is needed. I would explicity clarify which operations may modify the object source on which the operations are called. All the stdlib operations must clarify: 1. if the operation acts on the own object or on a copy of the object. 2. which object(s) is(are) exactly returned (the own object, a (collection of) new object(s), a (collection of) referenced object(s)) For example: String::trim operation clearly says that creates a new object copy of itself which is modified and returned. However, String::firstToUpper operation may have several interpretations.

Resolution:

Issue 19146 resolution makes the behaviour of List clear.

Dict clarified below and a typo

String clarified below.

Revised Text:

In 8.3.7.1 Dictionary get change

The null value is returned is not present.

to

The null value is returned if k is not present. Modifying the returned value modifies the value stored in the dictionary.

In 8.3.7.4 Dictionary put change

Assigns a value to a key.

to

Modifies the dictionary by assigning a value to a key. Modifying the value modifies the value stored in the dictionary.

In 8.3.7.5 Dictionary clear change

Removes all values in the dictionary

to

Modifies the dictionary by removing all values.

In 8.3.7.7 Dictionary values change

Returns the list of values in a list. The order is arbitrary.

to

Returns a new list of the values in the dictionary. The order is arbitrary. Modifying the returned list does not modify the contents of the dictionary. Modifying the values in the list modifies the values stored in the dictionary.

In 8.3.7.8 Dictionary keys change

Returns the list of keys in a list. The order is arbitrary.

to

Returns a new list of keys to the dictionary. Modifying the returned list does not modify the contents of the dictionary. Modifying the values in the list may modify the keys to the dictionary contents giving

unpredictable behaviour. If mutable types such as List or Dictionary are used as Dictionary keys, applications should take care to create clones where appropriate.

In 8.3.9 String change

All string operations defined in OCL are available.

to

All string operations defined in OCL 2.4 are available. An OCL String is immutable and so there are no operations that modify strings.

Disposition: **Resolved**

Issue 13222: Explanation of the 'Model::removeElement' operation needs clarification.

Source:

Open Canarias, SL (Mr. E. Victor Sanchez, vsanchez(at)opencanarias.com)

Summary:

Suggestion: Change original explanation text: "Removes an object of the model extent. All links that the object have with other objects in the extent are deleted." by something similar to the following one: "Removes an object from the model extent. The object is considered removed from the extent if it is not a root object of the extent, and is not contained by any other object of the extent."

Resolution:

Simple change.

Revised Text:

In 8.3.5.4 Model::removeElement change

> Removes an object of the model extent. All links that the object have with other objects in the extent are deleted

to

> Removes an object of the model extent. All links between the object and other objects in the extent are deleted. References from collections to the ends of deleted links are removed. References from non-collections are set to null.

Disposition: **Resolved**

Issue 13264: Pag 63, Section 8.2.1.1 OperationalTransformation.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: "isubclass of Module (see: ImperativeOCL package)"

Discussion: Module doesn't belong to ImperativeOCL package.

Suggestion: remove "(see: ImperativeOCL package)"

Resolution:

Simple change.

Revised Text:

In the final editorial paragrpah of 8.2.1.1 OperationalTransformation change

is a subclass of `Module` (see: `ImperativeOCL` package),

to

is a subclass of `Module`,

Disposition: **Resolved**

Issue 13265: Page 65, Notation Section 8.2.1.3 Module.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: "configuration property UML::Attribute::MAX_SIXE: String

discussion: providing a context to a configuration property doesn't seem to make sense.

suggestion remove "UML::Attribute::"

Resolution:

It's not obviously sensible, but is it actually wrong?

If there are many configuration properties, it may be convenient to partition them hierarchically. cf. java.vm.specification.vendor.

The punctuation is however confusing; add a space.

Revised Text:

In 8.2.1.1 OperationalTransformation change

> Properties that are configuration properties are declared using the **configuration** qualifier keyword.
> **configuration property** UML::Attribute::MAX_SIZE: String;

to

> Properties that are configuration properties may have hierarchical scope. They are declared using the **configuration** qualifier keyword.
> **configuration property** UML::Attribute::MAX_SIZE : String;

Disposition: **Resolved**

Issue 13267: Page 73, Section 8.2.1.10 OperationalTransformation.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: "overriden: Operation [0..1]"

discussion: the overriden operation should be an ImperativeOperation. This is correctly showd in the diagram.

suggestion: Replace "Operation" by "ImperativeOperation".

Resolution:

Yes.

Already correct in the QVT 1.1 models.

Revised Text:

In 8.2.1.10 ImperativeOperation change

overridden: Operation [0..1]

to

overridden: ImperativeOperation [0..1]

Disposition: **Resolved**

Issue 13268: Page 73: Section 8.2.1.11 Helper.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: "the invocation of the operation returns a tuple"

discussion: it could be understood as an OCL Tuple, which doesn't apply.

suggestion: Replace "a tuple" by "an ordered tuple".

Analysis:

The QVTo specification is very loose in its usage of tuple /ordered-tuple. Time to analyze whether OrderedTupleType has any utility

The Pack/Unpack use case

```
var a:A := ...
var b:B := ...
var t := Tuple{a, b};    // OrderedTupleLiteralExp
...
var (a1,b1) := t;        // UnpackExp
```

An OrderedTupleLiteralExp followed by an UnpackExp allows miscellaneous information to be packed and unpacked without naming each element.

This may sometimes give a lexical access saving in constrast to using OCL Tuples:

```
var t := Tuple{a:A = ..., b:B = ...};    // TupleLiteralExp
...
-- var a1 := t.a;
-- var b2 := t.b;
```

As the example shows, the overall saving is not necessarily large or even positive. OrderedTuples introduce a hazard from mis-aligned positional parts.

However, this functionality is sufficiently clearly specified that outright removal would be a breaking change.

Recommendation: deprecate OrderedTupleLiteralExp/UnpackExp etc as unnecessary bloat.

The OrderedTuple::at() use case

8.2.2.7: *Tuple elements can be accessed individually using the at pre-defined operation.*

Oops. The OrderedTuple::at() operation is not specified anywhere else.

In particular, if at() was specified, its return type would be data dependent demonstrating the worst characteristics of reflection.

In contrast, access to a Tuple part has a well-defined type and, if a reflective Tuple access was provided, there is a possibility that it could be type safe.

Recommendation: Eliminate the suggestion that an at operation might exist.

The Helper/Mapping return use case

This issue started because of the bad wording for helper returns. If we try to construct a complete example we highlight new problems.

The major example is in 8.1.2

```
mapping Foo::foo2atombar () : atom:Atom, bar:Bar
merges foo2barPersistence, foo2atomFactory
{
   object atom:{name := "A_"+self.name.upper();}
   object bar:{ name := "B_"+self.name.lower();}
}
```

This uses named returns and consequently an OCL Tuple return and so poses no real confusion

Continuing we find

```
   var (atom: Atom, bar: Bar) := f.foo2atombar();
```

which unpacks an OCL Tuple return. This is not supported by an UnpackExp that requires an OrderedTuple. Surely it should be

```
   var t := f.foo2atombar();
--   var atom := t.atom;
--   var bar := t.bar;
```

There does not appear to be a full example of an OrderedTuple return. Reworking the example to use one, we could elimninate the return names giving

```
mapping Foo::foo2atombar () : Atom, Bar
merges foo2barPersistence, foo2atomFactory
{
   Tuple{
       object atom:{name := "A_"+self.name.upper();},
       object bar:{ name := "B_"+self.name.lower();}
    }
}
```

and provide a specification that a list of unnamed return types e.g. "Atom, Bar" is a shorthand for "Tuple(Atom, Bar)"

A Mapping of this form cannot do piecemeal assignment to the named result variables, so a great deal is lost and very little gained.

Recommendation: implicit OrderedTuple return types have dubious benefits and are barely motivated by the specification so we can eliminate them

A return must either be an optionally named Type or an OCL Tuple of named Types.

The asOrderedTuple Use Case

The Object::asOrderedTuple() : OrderedTuple(T) library function provides a limited reflective capability with a rather poor description.

Converts the object into an ordered tuple. If the object is already an ordered type, no change is done.

What is an ordered type? Is a Sequence or List an ordered type?

If the object is an OCL Tuple, the list of attributes become the anonymous content of the ordered tuple.

Is this really intended? I would expect the values to become the content. Either way this is a non-determinstic conversion.

Otherwise, the operation creates a new tuple with a unique element being the object.

I guess it means a single element OrderedTuple containing the source, but for a class, I would really like to see the properties/values.

The above functionlity would appear to be provided by

Tuple::keys() : Set(TypedElement) to get a set of the Tuple name+type parts.
Tuple::at(TypedElement) : OclAny to get a selected part value
and extended by
OclElement::asTuple() : Tuple to give a Tuple of name+type+value parts.

Recommendation: deprecate asOrderedTuple in favour of Tuple::keys/at in OCL.

Resolution:

The usage or OrderedTuple seems misguided since an ordinary OCL Tuple does very similar things without a positional hazard and without noticeable lexical overhead.

Only the case of TupleLiteralEx/UnpackExp is plausibly defined in QVT 1.1 and neither of the QVT implementations (SmartQVT or Eclipse QVT) support OrderedTuple.

Therefore eliminate TupleLiteralExp, UnpackExp and OrderedTupleType and OrderedTupleLiteralPart.

Note that there are no editing instructions for the grammar. It is not clear that OrderedTuples were really supported at all.

Revised Text:

In 8.1.12 replace

```
var f := lookForAFooInstance();
var (atom: Atom, bar: Bar) := f.foo2atombar();
```

by

```
var f := lookForAFooInstance();
var t := f.foo2atombar();          – Tuple(atom: Atom, bar: Bar)
var atom := t.atom;
var bar := t.bar;
```

In 8.1.14 replace

These are *mutable lists*, *dictionary types*, and *ordered tuples*.

by

These are *mutable lists* and *dictionary types*.

In 8.1.14 remove

An *ordered tuple* is a tuple whose slots are unnamed. It can be manipulated in a more flexible way than tuples with named slots since it can be packed and unpacked more directly.

var mytuple := **Tuple**{1,2,3}; // an ordered tuple literal
var (x,y,z) := mytuple; // unpacking the tuple into three variables

In Fig 8.4 remove

UnpackExp

In Fig 8.7 and accompanying explanatory text remove

OrderedTupleType, OrderedTupleLiteralExp, OrderedTupleLiteralPart

In 8.2.1.15 MappingOperation replace

After call resolution, all the parameters of the mapping are passed as a tuple. The parameters include, in this order: the context parameter

by

After call resolution, the parameters of the mapping include: the context parameter

In 8.2.2.16 ReturnExp replace

If the operation declares more than one result parameter, the value is assigned to the tuple representing the collection of results. This requires that the type of the value passed to the result expressions is a tuple.

by

If the operation declares more than one return parameter, the result is a tuple with a tuple part for each return parameter. When the return expression value is omitted, this tuple is created automatically from the assignable result parameters. Alternatively the return expression value may be an explicitly constructed tuple with one part for each return parameter.

Remove 8.2.2.22 UnpackExp

Remove 8.2.2.27 OrderedTupleType

Remove 8.2.2.31 OrderedTupleLiteralExp

Remove 8.2.2.32 OrderedTupleLiteralPart

In 8.3.3.1 repr remove Object::asOrderedTuple

Disposition: **Resolved**

Issue 13269: Page 75: Section 8.2.1.13 Constructor.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: "/body: BlockExp [0..1] {composes} (from ImperativeOperation)

The expression serving to populate the object using the given parameters. This expression should necessarily be an ObjectExp instance.

discussion: This is not coherent with the ImperativeOperation definition. Body is an OperationBody, specifically a ConstructorBody.

Suggestion: Replace the text above by the following:

"/body: OperationBody [0..1] {composes} (from ImperativeOperation)

The operation body of the constructor. It should necessarily be a ConstructorBody instance.

Resolution:

Yes. But we can go all the way by modeling the ConstructorBody constraint.

Revised Text:

In 8.2.1.13 Constructor change

`/body: BlockExp [0..1]{composes}(from ImperativeOperation)`
The expression serving to populate the object using the given parameters. This expression should necessarily be an `ObjectExp` instance.

to

`/body: ConstructorBody [0..1]{composes}(from ImperativeOperation)`
The imperative implementation for this constructor.

Disposition: **Resolved**

Issue 13270: Page 75: Section 8.2.1.14 ContextualProperty.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

missed text: there is a missed text related to the initExpression association.

discussion: If we have a look to the diagram in figure 8.2, we may realize that a description of the initExpression association is missed.

suggestion: include the following text in the association's description:

initExpression: OclExpression [0..1] {composes}

An optional OCL Expression to initialize the contextual property.

Resolution:

Yes.

Already correct in the QVT 1.1 models.

Revised Text:

In 8.2.1.14 ContextualProperty add

```
initExpression: OclExpression [0..1] {composes}
```

An optional OCL Expression to initialize the contextual property.

Disposition: **Resolved**

Issue 13272: Page 83: Notation Section 8.2.1.22 MappingCallExp.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: This is called the “collect” shorthand

discussion: Since OCL provides a "collect" shorthand (f.i. doing aCollection.anOperationCall()), I would rather call "xcollect" shortand to avoid confusions.

suggestion: Replate "collect" by "xcollect".

Resolution:

OCL provides an 'implicit collect' shorthand, so 'imperative collect' is a better contrast..

Revised Text:

In 8.1.6 Inlining Mapping Operations replace

the *collect shorthand*

by

the *imperative collect shorthand*

In 8.2.1.21 MappingCallExp replace

the “collect” shorthand

by

the “imperative collect” shorthand

In 8.2.2.8 SwitchExp replace

the “collect” shorthand

by

the “imperative collect” shorthand

Disposition: **Resolved**

Issue 13273: Page 83: Notation Section 8.2.1.22 MappingCallExp.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: // shorthand of self.ownedElement->xcollect(i) i.map class2table();

discussion: It seems that the imperativeIteratExp has not been correctly notated.

suggestion: replace the text above by: // shorthand of self.ownedElement->xcollect(i | i.map class2table());

Resolution:

Yes.

Revised Text:

In 8.2.1.21 MappingCallExp replace

```
self.ownedElement->map class2table();
// shorthand of self.ownedElement->xcollect(i) i.map class2table();
```

by

```
self.ownedElement->map class2table();
// shorthand of self.ownedElement->xcollect(i | i.map class2table());
```

Disposition: **Resolved**

Issue 13274: Page 84: Notation Section 8.2.1.22 MappingCallExp.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: ->forEach (cl) cleaningTransf.map removeDups(cl);

discussion: EBNF and forExp suggest using always braces to enclose the forExp body.

Suggestion: Enclose the forEach's body with '{' and '}'. Note that there are two forExps in this section

Resolution:

Yes.

Revised Text:

In 8.2.1.21 MappingCallExp replace

```
  // first pass: cleaning the UML classes
  uml->objectsOfType(Class) // invoking the imported transformation
    ->forEach (cl) cleaningTransf.map removeDups(cl);
  // second pass: transforming all UML classes uml
    ->objectsOfType(Class)->forEach (cl)
    cl.map umlclass2javaclass (); // equivalent to: this.map
  umlclass2javaclass(cl)
```

by

```
  // first pass: cleaning the UML classes
  uml->objectsOfType(Class) // invoking the imported transformation
    ->forEach (cl) {
      cleaningTransf.map removeDups(cl);
    }
  // second pass: transforming all UML classes uml
    ->objectsOfType(Class)->forEach (cl) {
      cl.map umlclass2javaclass();
    }// equivalent to: this.map umlclass2javaclass(cl)
```

Disposition: **Resolved**

Issue 13275: Page 86: Notation Section 8.2.1.23 ResolveExp.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: // shorthand for mylist->forEach(i) i.late resolve(Table)

discussion: EBNF and forExp suggest using always braces to enclose the forExp body.

Suggestion: Enclose the forEach's body with '{' and '}'.

Resolution:

Yes.

Revised Text:

In 8.2.1.22 ResolveExp replace

```
    myprop := mylist->late resolve(Table);
    // shorthand for mylist->forEach(i) i.late resolve(Table)
```

by (and correct font size of)

```
    myprop := mylist->late resolve(Table);
    // shorthand for mylist->forEach(i) { i.late resolve(Table); }
```

Disposition: **Resolved**

Issue 13276: Page 87: Section 8.2.1.24 ObjectExp.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: /instantiatedClass: Class [0..1](from InstanciationExp)

discussion: wrong description.

Suggestion: replace the text above by "/instantiatedClass: Class [1](from InstantiationExp)"

note that in /extent reference, "(from InstanciationExp)" must be also replaced by (from instantiationExp)"

Resolution:

Yes and correct the erroneous bounds change on the spurious redefinition.

Revised Text:

In 8.2.1.24 ObjectExp and the QVT 1.1 models replace

```
/instantiatedClass: Class [0..1](from InstanciationExp)
```

Indicates the class of the object to be created or populated.

```
/extent: Variable [0..1](from InstanciationExp)
```

by

```
/instantiatedClass: Class [1](from InstantiationExp)
```

Indicates the class of the object to be created or populated.

```
/extent: Variable [0..1](from InstantiationExp)
```

Disposition: **Resolved**

Issue 13277: Page 87: Section 8.2.1.24 ObjectExp.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: "When an object expression is the body of an imperative collect expression (see xcollect in ImperativeLoopExp)"

discussion: the text should point ImperativeIterateExp instead.

suggestion: replace ImperativeLoopExp by ImperativeIterateExp.

Resolution:

Yes.

Revised Text:

In 8.2.1.24 ObjectExp replace

see xcollect in ImperativeLoopExp

by

see xcollect in ImperativeIterateExp

Disposition: **Resolved**

Issue 13278: Page 87: Notation Section 8.2.1.24 ObjectExp (03).

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: // shorthand for list->xcollect(x) object X{ … }

discussion: It seems that the imperativeIteratExp has not been correctly notated.

suggestion: replace the text above by: // shorthand for list->xcollect(x | object X{ … });

Resolution:

Yes.

Revised Text:

In 8.2.1.24 ObjectExp replace

// shorthand for list->xcollect(x) object X{ … }

by

// shorthand for list->xcollect(x | object X{ … })

Disposition: **Resolved**

Issue 13279: Page 89: Figure 8.6.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text:

1.ReturnExp::value association my have multiplicity [0..1] in the diagram

2.TryExp::catchClause association should be called exceptClause

Discussion:

1. It seems that the returned value of a return expression might be optional.

2. Since the diagram and textual descriptioon are different, I'm not sure which was the original intention of the name of this reference. Reading the description's text

and the opposite role name (exceptOwner), I guees that the name must be "exceptClause".

suggestion:

1.In ReturnExp::value association replace mutiplicity 1 by multiplicity [0..1]. In the text is well described.

2.In TryExp class change change the "catchClause" by "exceptClause"

- Page 90: Section 8.2.2.3 ComputeExp

Problem's text: body : OclExpression [1] {composes, ordered}

Discussion: Ordered doesn't make sense in a univalued reference.

Suggestion: remove "ordered".

Resolution:

Yes.

Yes.

Yes.

Already correct in the QVT 1.1 models.

Revised Text:

In Figure 8.6 replace the “catchClause” role from TryExp by “exceptClause”.

In Figure 8.6 replace the “1” multiplicity on ReturnExp.value by “0..1”.

In 8.2.2.3 ComputeExp replace

```
body : OclExpression [1] {composes, ordered}
```

by

```
body : OclExpression [1] {composes}
```

Disposition: **Resolved**

Issue 13280: Page 90: Notation Section 8.2.2.4 WhileExp.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: compute (x:MyClass := self.getFirstItem()) while (x<>null) { … }

discussion: the compute expression's body requires the enclosing braces

Suggestion: replace the text above by "compute (x:MyClass := self.getFirstItem()) { while (x<>null) { … }}"

- Page 92: Semantics Section 8.2.2.4 ForExp

Problems text:

```
Collection(T)::forEach(source, iterator, condition,body) =
  do {
    count : Integer := 0;
    while (count <= source->size()) {
      var iterator := source->at(count+1);
      if (condition) continue;
      body;
      count += 1;
    };
  };
Collection(T)::forOne(source, iterator, condition,body) =
  forEach (i | condition) {
    body;
    break;
  }
```

Discussion: It seems that ForEach and forOne expression are not correctly expressed in QVTo terms.

Suggestion: Change the text above by:

```
Collection(T)::forEach(source, iterator, condition,body) =
  do {
    count : Integer := 1;
    while (count <= source->size()) {
      var iterator := source->at(count);
     if (condition) body;
      count += 1;
    };
  };
Collection(T)::forOne(source, iterator, condition,body) =
  forEach (iterator | condition) {
    body;
    break;
  }
```

Resolution:

Yes.

Yes.

Revised Text:

In 8.2.2.4 WhileExp replace

```
    compute (x:MyClass := self.getFirstItem()) while (x<>null) { … }
```

by

```
    compute (x:MyClass := self.getFirstItem()) { while (x<>null) { … } }
```

In 8.2.2.6 ForExp replace

```
    Collection(T)::forEach(source, iterator, condition,body) =
      do {
        count : Integer := 0;
        while (count <= source->size()) {
          var iterator := source->at(count+1);
          if (condition) continue;
          body;
          count += 1;
        };
      };
```

by

```
    Collection(T)::forEach(source, iterator, condition,body) =
      do {
        count : Integer := 1;
        while (count <= source->size()) {
          var iterator := source->at(count);
          if (condition) body;
          count += 1;
        };
      };
```

and

```
    Collection(T)::forOne(source, iterator, condition,body) =
      forEach (i | condition) {
        body;
        break;
      }
```

by

```
    Collection(T)::forOne(source, iterator, condition,body) =
      forEach (iterator | condition) {
        body;
        break;
      }
```

Disposition: **Resolved**

Issue 13281: Page 93: Associations Section 8.2.2.7 ImperativeIterateExp.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: target : Variable [0..1]

discussion: composes is missed. In the diagram is correctly represented the association's feature.

suggestion: add the following text to end of the line "{composes}

Resolution:

Yes.

Already correct in the QVT 1.1 models.

Revised Text:

In 8.2.2.7 ImperativeIterateExp replace

```
target : Variable [0..1]
```

by

```
target : Variable [0..1] { composes }
```

Disposition: **Resolved**

Issue 13282: Page 95: Notation Section 8.2.2.7 ImperativeIterateExp.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: list[condition]; // same as list->xselect(i; condition)

discussion: From the specified notation, the xselect expression is not well written.

suggestion: replace ';' by '|'

Resolution:

Yes.

Revised Text:

In 8.2.2.7 ImperativeIterateExp replace

list[condition]; // same as list->**xselect**(i; condition)

by

list[condition]; // same as list->**xselect**(i | condition)

Disposition: **Resolved**

Issue 13283: Page 95: Associations Section 8.2.2.8 SwitchExp.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: elsePart : OclExpresion {composes} [0..1]

discussion: For uniformity, the multiplicty should appear after the type of the association.

suggestion: change positions between "{composes}" and "[0..1].

Resolution:

Yes, and correct spelling of OclExpression too.

Revised Text:

In 8.2.2.8 SwitchExp replace

elsePart : OclExpresion {composes} [0..1]

by

elsePart : OclExpression [0..1] {composes}

Disposition: **Resolved**

Issue 13284: Page 100: Superclasses Section 8.2.2.8 LogExp.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: OperationCallExp

discussion: From figure 8.4 and 8.6, we can guess that LogExp should inherits from both, OperationCallExp and ImperativeExpression.

suggesition: add ImperativeExpression as a LogExp's superclass.

Resolution:

Yes.

Revised Text:

In 8.2.2.19 LogExp replace

Superclasses
`OperationCallExp`

by

Superclasses
`OperationCallExp, ImperativeExpression`

Disposition: **Resolved**

Issue 13285: Page 103: Associations Section 8.2.2.23 InstantiationExp.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: // column := self.attribute->forEach new(a) Column(a.name,a.type.name);

discussion: the foreach exp is not well written:

suggestion: replace the text above by "// column := self.attribute->forEach(a) { new Column(a.name,a.type.name) };

Resolution:

Yes.

Revised Text:

In 8.2.2. 23 InstantiationExp replace

// column := self.attribute->forEach **new(**a**)** Column(a.name,a.type.name);

by

// column := self.attribute->forEach(a) { **new** Column(a.name,a.type.name); }

Disposition: **Resolved**

Issue 13286: Page 103: Figure 8.7.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem: There are two figures to represent the type extensions.

suggestion: Remove the first figure and name the sencond one as Figure 8.7 - Imperative OCL Package - Type extensions.

Resolution:

There are actually 5 sub-diagrams of which the first two are duplicated by the third and fourth; presumably an editing artefact when the fifth was introduced for ListLiteralExp.

Revised Text:

In Figure 8.7 remove the first two sub-diagrams.

Disposition: **Resolved**

Issue 13287: Page 105: Associations Section 8.2.2.24 Typedef.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: condition: OclExpression [1]{composes}

discussion: the condition is optional.

suggestion: Replace "[1]" by "[0..1]".

Resolution:

Yes.

Already correct in the QVT 1.1 models.

Revised Text:

In 8.2.2.24 Typedef replace

```
condition: OclExpression [1]{composes}
```

by

```
condition: OclExpression [0..1]{composes}
```

Disposition: **Resolved**

Issue 13288: Page 105: Associations Section 8.2.2.26 DictionaryType.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: (see DictLiteralValue).

discussion: DictLiteralValue deosn't exist. It must be DictLiteralExp.

suggestion: Replace "DictLiteralValue" by "DictLiteralExp".

Resolution:

Yes.

Revised Text:

In 8.2.2.26 DictionaryType replace

(see `DictLiteralValue`)

by

(see `DictLiteralExp`)

Disposition: **Resolved**

Issue 13289: Page 106: Associations Section 8.2.2.29 DictLiteralExp.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: part : DictLiteralPart [*] {composes,ordered}

discussion: Do the parts need to be ordered ?. The diagram doesn't show it.

suggestion: Remove ordered or update the diagram.

Resolution:

Yes. A dictionary does not need go be ordered.

Already correct in the QVT 1.1 models.

Revised Text:

In 8.2.2.29 DictLiteralExp replace

```
part : DictLiteralPart [*] {composes,ordered}
```

The list of parts contained by this dictionary.

by

```
part : DictLiteralPart [*] {composes}
The parts contained by this dictionary.
```

Disposition: **Resolved**

Issue 13290: Page 108: Section 8.3 Standard Library.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: The OCL standard library is an instance of the Library metaclass.

discussion: It should obviously refer to the QVT Operational Mapppings standard library.

suggestion: replace "OCL" by "QVT Operational Mappings".

Resolution:

Yes.

Revised Text:

In 8.3 Standard Library replace

> This section describes the additions to the OCL standard library. The OCL standard library is an instance of the Library metaclass.

by

> This section describes the additions to the OCL standard library to form the QVT Operational Mappings Library, which is an instance of the Library metaclass.

Disposition: **Resolved**

Issue 13913: Typo in 'Model::rootObjects' signature.

Source:

Open Canarias, SL (Mr. E. Victor Sanchez, vsanchez(at)opencanarias.com)

Summary:

Section '8.3.5.3 rootObjects' has a typo in the signature of the operation defined: Model::rootobjects() : Set(Element). In compliance with the surrounding definitions and with the title of the section itself, it should read 'Model::rootObjects() : Set(Element), that is, the first letter of 'Objects', capitalized.

Suggestion: Replace 'Model::rootobjects() : Set(Element)' by the new 'Model::rootObjects() : Set(Element)'

Resolution:

Yes.

Revised Text:

In 8.3.5.3 rootObjects change

```
Model::rootobjects(): Set(Element)
```

to

```
Model::rootObjects(): Set(Element)
```

Disposition: **Resolved**

Issue 13989: Typos in signatures of "allSubobjectsOfType" and "allSubobjectsOfKind".

Source:

Open Canarias, SL (Mr. E. Victor Sanchez, vsanchez(at)opencanarias.com)

Summary:

Sections 8.3.4.7 and 8.3.4.9, page 110:

Operation signatures for 8.3.4.7 and 8.3.4.9 do not correspond with the title of sections.

Please update 8.3.4.7 from "Element::subobjects(OclType) : Set(Element)" to "Element::allSubobjectsOfType(OclType) : Set(Element)".

Please update 8.3.4.9 from "Element::subobjectsOfKind(OclType) : Set(Element)" to "Element::allSubobjectsOfKind(OclType) : Set(Element)"

Resolution:

Yes.

Revised Text:

In 8.3.4.7 allSubobjectsOfType change

```
Element::subobjects(OclType) : Set(Element)
```

to

```
Element::allSubobjectsOfType(OclType) : Set(Element)
```

In 8.3.4.9 allSubobjectsOfKind change

```
Element::subobjectsOfKind(OclType) : Set(Element)
```

to

```
Element::allSubobjectsOfKind(OclType) : Set(Element)
```

Disposition: **Resolved**

Issue 14549: Wrong Chapter reference on Page 2 Language Dimension

Source:

InterComponentWare AG (Markus von Rüden, markus.vonrueden(at)icw.de)

Summary:

On Page 2 (Page 18 in PDF) there is a wrong reference in chapter 2.2 Language Dimension at 1. and 2.

"Core; The Core language is described in Chapter 11.."

"Relations: The Relations language is described in Chapter 9. ..."

It should be 9 and 7 ;)

Resolution:

The Chapter 10 reference is wrong too.

Further wrong Chapter 9/11 references exist in the main text.

The above references are manual text that do not use FrameMaker cross-reference facilities.

Loading the QVT 1.1 FrameMaker files reveals 6 unresolved cross-references.

OMG specifications have Clauses not Chapters.

Revised Text:

<Review all cross-references and correct stale definitions>

<Replace all usage of Chapter by a cross reference to the corresponding clause>

Disposition: **Resolved**

Issue 14619: QVT 1.1 Opposite navigation scoping operator (Correction to Issue 11341 resolution).

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The resolution for Issue 11341 uses '.' as a property scoping operator.

This is inconsistent with OCL for which class scoping always uses '::' e.g enumeration is class::literal operation call is class::operation.

Note the clarifying discussion in OCL, justifying Class.allInstances, explains that dot is only for operations on instances.

Resolution:

Simple correction of '.' to '::' in the grammar.

Revised Text:

In 7.13.5 change

```
   <keyProperty> ::= <identifier> | 'opposite' '(' <classId> '.' <identifier> ')'
```

to

```
   <keyProperty> ::= <identifier> | 'opposite' '(' <classId> '::' <identifier> ')'
```

and

```
   <propertyTemplate> ::= <identifier> '=' <OclExpressionCS> |
   'opposite' '(' <classId> '.' <identifier> ')' '=' <OclExpressionCS>
```

to

```
   <propertyTemplate> ::= <identifier> '=' <OclExpressionCS> |
   'opposite' '(' <classId> '::' <identifier> ')' '=' <OclExpressionCS>
```

Disposition: **Resolved**

Issue 15424: Figure 7.3.

Source:

Institute for Defense Analyses (Dr. Steven Wartik, swartik(at)ida.org)

Summary:

I am confused by something in the MOF QVT specification (version 1.1) and am not sure if it's an error or my own misunderstanding. Figure 7.3 (p. 24) has links from two instances of ObjectTemplateExp to a single instance of PropertyTemplateItem (the one in the middle). Figure 7.6 (p. 30) shows a composite association between ObjectTemplateExp and PropertyTemplateItem. Why then are there two links to the instance in Figure 7.3? Doesn't a composite association mean that a PropertyTemplateItem can be owned by only one ObjectTemplateExp?

Resolution:

The problem is that UML Object diagrams omit the decorations from instantiated relationships. Shrt of moving to a non-standard UMLX diagram, we can perhaps get away with some explanations.

Revised Text:

Above Figure 7.3 add

> Instance diagrams omit the usual UML decorations, so the reader should note that a top to bottom composition layout is used in Figure 7.3. Thus the PropertyTempateItem for 'attribute' is composed by the ObjectTemplateItem for 'Class' and composes the ObjectTemplateItem for 'Attribute'.

Disposition: **Resolved**

Issue 15917: bug in the uml to rdbms transformation example.

Source:

Xavier Dolques (xavier.dolques(at)laposte.net)

Summary:

In the UML to rdbms transformation code given in section A.2.3 :

-- mapping to update a Table with new columns of foreign keys

mapping Association::asso2table() : Table

when {self.isPersistent();}

{

init {result := self.destination.resolveone(Table);}

foreignKey := self.map asso2ForeignKey();

column := result.foreignKey->column ;

}

The mapping asso2table is supposed to update a table by adding new columns, but with the last line of the mapping, the existing columns are all replaced by the new ones. I suggest to replace the last line with :

column += result.foreignKey->column ;

Resolution:

Yes.

Revised Text:

In A.2.3 mapping Association::asso2table() change

```
column := result.foreignKey->column ;
```

to

```
column += result.foreignKey->column ;
```

Disposition: **Resolved**

Issue 15977: abstract/concrete syntax for try/catch in clauses 8.2.2.13 & 8.2.2.14 lacks support for retrieving the exception caught.

Source:

NASA (Dr. Nicolas F. Rouquette, nicolas.f.rouquette(at)jpl.nasa.gov)

Summary:

Current abstract/concrete syntax for try/catch in clauses 8.2.2.13 & 8.2.2.14 lacks support for retrieving the exception caught.

That is, QVT1.1 is currently limited to the following style of try/catch logic:

```
try {
    // ...
} except (Exception) {
    // there is no syntax to bind the actual exception caught to a variable or to retrieve it in an except expression.
};
```

One possibility would be to introduce a variable in the catch expression (clause 8.2.2.14), e.g.:

```
try {
    // ...
} except (Exception e) {
    // do something with the exception caught: e
};
```

or:

```
try {
    // ...
} except (Exception1 e1, Exception2 e2) {
    // do something with the exception caught: e1 or e2
};
```

Resolution:

Yes. Unuseable caught exceptions are clearly of limited utility.

There can only be one exception variable name for many exception types; what is its type? Cannot be any of the listed types so it will have to be the common super type. Introduce a new exceptionVariable.

Of course the lower bound on the exception type should be 1. If a catch all is required then the type can be Exception. If a finally is required then we need a wrapper or specification enhancement.

Revised Text:

In 8.2.2.13 TryExp add

> The exceptClauses are searched in order to select the first exceptClause that provides an exception type to which the raised exception conforms. If an exceptClause is selected, its body is executed.

In 8.2.2.14 CatchExp add

> The caught expression may be accessed in the body expression using the exceptionVariable whose apparent (static) type is the most derived common super type of all catchable exception types.

In 8.2.2.14 CatchExp and the QVT 1.1 models add

```
exceptionVariable : String [0..1]
```

The variable through which the caught exception may be accessed.

In 8.2.2.14 CatchExp and the QVT 1.1 models change

```
exception: Type [*] {ordered}
```

to

```
exception: Type [+] {ordered}
```

In 8.4.7 change

```
<except> ::= 'except' '(' <scoped_identifier_list> ')' <expression_block>
```

to

```
<except> ::= 'except' '(' [<identifier> ':'] <scoped_identifier>
    [',' <scoped_identifier>]* ')' <expression_block>
```

Disposition: **Resolved**

Issue 15978: clause 8.3.1.4 Exception needs to document the taxonomy of Exception types in QVT1.1.

Source:

NASA (Dr. Nicolas F. Rouquette, nicolas.f.rouquette(at)jpl.nasa.gov)

Summary:

In particular, this taxonomy should explicitly include the AssertionFailed exception type that clause 8.2.2.20 refers to for an AssertExp.

Suggest defining a String message attribute for Exception; this would facilitate retrieving the message from a raise expression (clause 8.2.2.15)

Suggest defining AssertionFailed as a subtype of Exception.

Suggest defining 2 attributes in AssertionFailed corresponding to the severity and log expressions of the AssertExp (clause 8.2.2.20)

Resolution:

Yes. We need a toxonomy.

At the root is clearly Exception and this needs no parameters.

We need to introduce a derived StringException for the standard string-valued RaiseExp, and oops we need to modify the grammar to support this shorthand.

The LogExp-valued AssertionFailed is another derivation of Exception; no severity since it is always fatal.

Revised Text:

In 8.2.2.15 RaiseExp change

> The notation uses the raise keyword with the exception name as body. The exceptions can be provided as simple strings. In that case the implicit referred exception is the user Exception defined in the QVT standard library and the string is the argument of the raise expression.
>
> myproperty := self.something **default raise** "ProblemHere";

to

> The notation uses the raise keyword followed by the exception type name and arguments for one of the expression type name constructors.
>
> myproperty := self.something **default raise** StringException("ProblemHere");
>
> The exceptions can be provided as simple strings. This is a shorthand for raising a StringException with the string as the constructor argument
>
> myproperty := self.something **default raise** "ProblemHere";

Following 8.3.1.4 Exception add additional sub-sub-sub-sections

> <new sub-sub-sub-section> **StringException**
>
> The StringException supports the simple string-valued shorthand of a RaiseExp.
>
> **Superclasses**
> Exception
> **Constructor**
> StringException**(**reason `: String)`

Attributes
reason : String [1]

The reason provided on construction.

<new sub-sub-sub-section> **AssertionFailed**

The AssertionFailed exception supports a fatal AssertExp.

Superclasses
Exception
Constructor
AssertionFailed(reason : LogExp)
Attributes
reason : LogExp [1]

The reason provided on construction.

In 8.4.7 change

```
<raise_exp> ::= 'raise' <scoped_identifier> ('(' <arg_list>? ')')?
```

to

```
<raise_exp> ::= 'raise' <scoped_identifier> ('(' <arg_list>? ')')?
                | 'raise' <STRING>
```

Disposition: **Resolved**

Issue 18572: QVT atomicity.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Clause 7.7 mandates fixed-point semantics for in-place transformations.

Please ask my bank to use a repeat-until-no-change in-place transformation for my next pay cheque:

new-balance = old-balance + deposit.

More seriously, the repeat is at the relation level, so if there are multiple applicable relations, the in-place result is specified to experience multiple updates in an indeterminate order. If relation A and then relation B are repeated to exhaustion, is relation A repeated again to accommodate relation B's changes?

Start again. QVTr and QVTc are declarative, therefore they express a single complex truth about the final outputs with respect to the original inputs. There are no observable intermediate steps. It is the responsibility of the transformation engine to ensure that the multiple actual output changes are observed as a single atomic transaction. In particular for in-place transformations, the engine must ensure that no input value is accessed after it is updated for output.

In regard to fixed-point semantics, repetition can only be determinate if it is the entire tranformation that is repeated, and whether to do so would seem to be a legitimate execution option. Therefore QVT should either not specify repetition at all, leaving it to the invoking engine, or specify it as an invocation option for a RelationCallExp.

If an in-place transformation does perform fixed-point repetition at the transformation level, it would seem that the whole repetition should still be a single atomic transaction so that outputs are never observable in an inconsistent partially transformed state between iterations. The engine must therefore iterate over candidate outputs rather than actual outputs.

Resolution:

Fixed point semantics can be achieved by a has-it-changed loop.

Revised Text:

In 7.7 In-place Transformations replace

> A transformation may be considered in-place when its source and target candidate models are both bound to the same model at runtime. The following additional comments apply to the enforcement semantics of an in-place transformation:
>
> • A relation is re-evaluated after each enforcement-induced modification to a target pattern instance of the model.
>
> • A relation’s evaluation stops when all the pattern instances satisfy the relationship.

by

> A transformation may be considered in-place when one or more source models are bound to one or more target models at runtime. Execution proceeds as if the source and target models are distinct with an atomic update of non-distinct models occurring on completion of the transformation. This implies that an implementation that operates in-place must take copies of the old state to avoid confusion with updated new state.
>
> Execution of a transformation should return a Boolean status to indicate whether any changes were made to target models. This status enables transformation applications to repeat execution until no changes occur where that is appropriate for the application.

Disposition: **Resolved**

Issue 19021: Inconsistent description about constructor names.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem:

Specification first says in the Constructor concept description: "The name of the constructor is usually the name of the class to be instantiated. However this is not mandatory. Giving distinct names allows having more than one constructor."

Later on in the Constructor notation: "The name of the constructor is necessarily the name of the context type"

This is inconsistent.

Discussion:

Indeed, the notation section statement seems to be correct since:

1. Looks like other programming languages, like Java.

2. More importantly, the instantiation expression would not be so obvious when constructing new objects, and would required to be changed.

Example:

If we have the following constructors:

constructor MyClass::MyClassConstructor(name : String) { name := name }

constructor MyClass::MyClass(name : String) { name := name + "v2" }

How can the instantiation expression refer the different constructor ?

- new MyClass("abc")

- new MyClassConstructor("abc")

- new MyClass::Constructor("abc")

The referred class in a InstantiationExp would not be enough. Changing instantiation expression to provide different name constructor doesn't seem sensible.

Proposed solution:

In section 8.2.1.13

Replace:

"A constructor does not declare result parameters. The name of the constructor is usually the name of the class to be

instantiated. However this is not mandatory. Giving distinct names allows having more than one constructor."

by

"A constructor does not declare result parameters and its name must be the name of the class to be instantiated."

Resolution:

This was discussed on https://bugs.eclipse.org/bugs/show_bug.cgi?id=421621.

Unless we abandon constructor diversity completely, the current AS imposes a needless implementation difficulty by requiring dynamic resolution of a statically known constructor. This can be

avoided by augmenting InstantiationExp.instantiatedClass with InstantiationExp.initializationOperation , which can refer to any statically determined constructor. We can therefore relax the contradictory restrictions on constructor name spelling.

Unfortunately Constructor is not available in ImperativeOCL. Promoting Constructor to ImperativeOCL would appear easy, unfortunately its superclass ImperativeOperation is also not available. Promoting ImperativeOperation requires … too hard. So we must instead introduce InstantiationExp. initializationOperation and redefine it in ObjectExp.

Revised Text:

In 8.2.1.13 Constructor notation change

The name of the constructor is necessarily the name of the context type

to

The name of the constructor is usually the name of the context type

In 8.2.1.24 ObjectExp add

`/initializationOperation : Constructor [0..1] (from InstantiationExp)`

The constructor that uses the arguments to initialize the object after creation. The constructor may be omitted when implicit construction occurs with no arguments.

In 8.2.2.23 InstantiationExp add

`initializationOperation : Operation [0..1]`

The initialization operation that uses the arguments to initialize the object after creation. The initialization operation may be omitted when implicit initialization occurs with no arguments.

In Figure 8.5 add

Operation (from EMOF)
unidirectional reference from InstantiationExp to Operation
-- forward role initializationOperation [0..1]
-- reverse role instantiationExp [*]

Disposition: **Resolved**

Issue 19022: ObjectExp Abstract Syntax misses a ConstructorBody.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem:

ObjectExp enhances an InstationExp with additional abstract syntax and semantics what respect to the association of the object to be created/updated to a variable.

Likewise, the object expression provides concrete syntax to specify the set of expressions to be executed in order to intialize/update the properties of the object to be created. However this is not reflected in the abstract sytanx.

Discussion:

Indeed, ObjectExp should contain a containment reference to comprise the block of expressions used to initialize/update the object properties.

The best AS candidate is ConstructorBody. Note that this is supported by the own description of the ConstructorBody (section 8.2.1.18):

"A constructor body contains the implementation of a constructor operation or the implementation of an inline constructor

(see ObjectExp)."

This ConstructorBody should be optional, as a result of an enhancement I'll raise in a different issue.

Proposed resolution:

Add the following association to ObjectExp in section 8.2.1.24

body: ConstructorBody [0..1] {composes}

The object expression body comprising the expressions to initialize (or update) the properties of the object to be instantiated (or updated).

Resolution:

The required association is already shown in Fig 8.3; just need to add the missing text.

Revised Text:

In 8.2.1.24 ObjectExp add

body: ConstructorBody [1] {composes}

The inline constructor body for the object to be instantiated or updated.

Disposition: **Resolved**

Issue 19096: Resolve expressions without source variable.

Source:

University of Paderborn (Christopher Gerking, christopher.gerking(at)upb.de)

Summary:

In contrast to 8.2.1.23 ResolveInExp, the prior section 8.2.1.22 ResolveExp does not explicitly state whether the source variable is optional for resolve expressions. For Eclipse QVTo, this leads to the situation the source variable is assumed to be mandatory. Consequently, the resolve expression in following snippet is invalid in Eclipse:

var p : EPackage = resolveone(EPackage);

I don't think that this restriction is desirable from the specification viewpoint.

Eclipse bugzilla: https://bugs.eclipse.org/bugs/show_bug.cgi?id=392156

Resolution:

The resolve needs a domain in which to search for target objects. For ResolveInExp this can be the traces of a designated mapping. For ResolveExp it would seem that some source objects must be supplied. However there seems no reason to prohibit a search everywhere and this could be achieved by specifying Element.allInstances(). Therefore an omitted ResolveExp sources can mean search all traces.

Revised Text:

In 8.2.1.22 ResolveExp after the first paragraph add

> The source object is optional. When no source object is provided, this expression inspects all the targets created or updated by all mapping operations irrespective of the source objects.

Disposition: **Resolved**

Issue 19121: Imprecise result types of resolveIn expressions.

Source:

University of Paderborn (Christopher Gerking, christopher.gerking(at)upb.de)

Summary:

Section 8.2.1.22 ResolveExp states the following about the result type of a resolve expression:

"If no target variable is provided, the type is either Object (the type representing all types, see Section 8.3.1) either a Sequence of Objects - depending on the multiplicity."

On top of that, Section 8.2.1.23 ResolveInExp states:

"The type of a ResolveInExp expression is computed using the same rules as for the type of a ResolveExp."

In case of a ResolveInExp, why can't we obtain the result type from the respective mapping?

Consider the following example

mapping EClass :: EClass2EPackage() : EPackage

The result of any resolveIn expression for that mapping is necessarily a subtype of EPackage. No need to cast this up to Object.

Resolution:

OclAny rather Object is of course the top type.

Revised Text:

In 8.2.1.23 ResolveInExp replace

The type of a *ResolveInExp* expression is computed using the same rules as for the type of a *ResolveExp*.

by

Type of a resolveIn expression

The type of a *ResolveInExp* expression depends on the type of the ‘target’ variable, the 'inMapping' operation and on the multiplicity indication (the ‘one’ property). The overall returned type is specified in terms of an intermediate resolved type.

If a 'target' variable is provided, the resolved-type is the type of the 'target' variable

Otherwise if an 'inMapping' is provided, .the resolved-type is the type of the 'inMapping'.

Otherwise the resolved-type is *Object* (the type representing all types, see Section 8.3.1).

If ‘one’ is true, the returned type is the resolved-type. Otherwise, the returned type is a Sequence of the resolved-type.

Disposition: Resolved

Issue 19146: Specify List::reject and other iterations.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The current specification of List has a vague claim that all OCL Collection operations are available.

Are iterations available?

Are Sequence operations and iterations not available?

Are return types adjusted to List?

The specification needs to provide a clear model defining all the operations and iterations.

Resolution:

Specifying the operations is straightforward, but highlights some nasty inconsistencies such as two versions of insertAt.

The OCL 2.4 operations are included.

From Issue 13223 clarify wording of insertAt:

From Issue 13251 add remove operations.

A model must wait till OCL 2.5 provides an extensible library model.

Revised Text:

<<Issue 13182 revises redundant (T) in some signatures.>>

Replace 8.3.8 Operations On List

All operations of the OCL Collection type are available. In addition the following are available.

add

```
List(T)::add(T) : Void
```

Adds a value at the end of the mutable list. Synonym: **append**

prepend

```
List(T)::prepend(T) : Void
```

Adds a value at the beginning of the mutable list.

insertAt

```
List(T)::insertAt(T,int) : Void
```

Adds a value at the given position. The index starts at zero (in compliance with OCL convention).

joinfields

```
List(T)::joinfields(sep:String,begin:String,end:String) :String
```

Creates a string separated by sep and delimited with begin and end strings.

asList

```
Set(T)::asList() : List(T)
OrderedSet(T)::asList(T) : List(T)
Sequence(T)::asList(T) : List(T)
Bag(T)::asList(T) : List(T)
```

Converts a collection into the equivalent mutable list.

by

The following operations can be invoked on any list. A list type is a parameterized type. The symbol T denotes the type of the values. The operations include all those of the OCL Sequence type.

=

```
List(T)::=(s : List(T)) : Boolean
```

True if self contains the same elements as s in the same order.

```
post: result = (size() = s->size()) and
  Sequence{1..size()}->forAll(i | at(i) = s->at(i))
```

<>

```
List(T)::<>(c : List(T)) : Boolean
```

True if c is not equal to self.

```
post: result = not (self = c)
```

add

```
List(T)::add(T) : Void
```

Adds a value at the end of the mutable list.

```
post: size() = size@pre() + 1
post: Sequence{1..size@pre()}->forAll(i | at(i) = at@pre(i))
post: at(size()) = object
```

append

```
List(T)::append(object: T) : List(T)
```

Returns a new list of elements, consisting of all elements of self, followed by object.

```
post: result->size() = size() + 1
post: Sequence{1..size()}->forAll(i | result->at(i) = at(i))
post: result->at(result->size()) = object
```

asBag

```
List(T)::asBag() : Bag(T)
```

Returns a Bag containing all the elements from self, including duplicates. The element order is indeterminate.

```
post: result->forAll(elem | self->count(elem) = result->count(elem))
post: self->forAll(elem | self->count(elem) = result->count(elem))
```

asList

```
List(T)::asList(T) : List(T)
```

Returns a new list that is a shallow clone of self.

```
post: Sequence{1..size()}->forAll(i | result->at(i) = self->at(i))
```

asOrderedSet

```
List(T)::asOrderedSet() : OrderedSet(T)
```

Returns a OrderedSet that contains all the elements from self, in the same order, with duplicates removed.

```
post: result->forAll(elem | self ->includes(elem))
post: self->forAll(elem | result->count(elem) = 1)
post: self->forAll(elem1, elem2 | self->indexOf(elem1) < self->indexOf(elem2)
    implies result->indexOf(elem1) < result->indexOf(elem2) )
```

asSequence

```
List(T)::asSequence() : Sequence(T)
```

Returns a Sequence that contains all the elements from self, in the same order.

```
post: result->size() = size()
post: Sequence{1..size()}->forAll(i | result->at(i) = self->at(i))
```

asSet

```
List(T)::asSet() : Set(T)
```

Returns a Set containing all the elements from self, with duplicates removed. The element order is indeterminate.

```
post: result->forAll(elem | self->includes(elem))
post: self->forAll(elem | result->includes(elem))
```

at

```
List(T)::at(i : Integer) : T
```

Returns the element of the list at the i-th one-based index.

```
pre : 1 <= i and i <= size()
```

clone

```
List(T)::clone(T) : List(T)
```

Returns a new list that is a shallow clone of self.

```
post: Sequence{1..size()}->forAll(i | result->at(i) = self->at(i))
```

count

```
List(T)::count(object : T) : Integer
```

Returns the number of occurrences of object in self.

deepclone

```
List(T)::deepclone(T) : List(T)
```

Returns a new list that is a deep clone of self; that is a new list in which each element is in turn a deepclone of the corresponding element of self.

excludes

```
List(T)::excludes(object : T) : Boolean
```

True if *object* is not an element of *self*, false otherwise.

```
post: result = (self->count(object) = 0)
```

excludesAll

```
List(T)::excludesAll(c2 : Collection(T)) : Boolean
```

Does *self* contain none of the elements of *c2* ?

```
post: result = c2->forAll(elem | self->excludes(elem))
```

excludesAll

```
List(T)::excludesAll(c2 : List(T)) : Boolean
```

Does *self* contain none of the elements of *c2* ?

```
post: result = c2->forAll(elem | self->excludes(elem))
```

excluding

```
List(T)::excluding(object : T) : List(T)
```

Returns a new list containing all elements of self apart from all occurrences of object. The order of the remaining elements is not changed.

```
post:result->includes(object) = false
post: result->size() = self->size()@pre - self->count(object)@pre
post: result = self->iterate(elem; acc : List(T) = List{} |
    if elem = object then acc else acc->append(elem) endif )
```

first

```
List(T)::first() : T
```

Returns the first element in self.

```
post: result = at(1)
```

flatten

```
List(T)::flatten() : List(T2)
```

Returns a new list containing the recursively flattened contents of the old list. The order of the elements is partial.

```
post: result = self->iterate(c; acc : List(T2) = List{} |
    if c.oclType().elementType.oclIsKindOf(CollectionType)
    then acc->union(c->flatten()->asList())
    else acc->union(c)
    endif)
```

includes

```
List(T)::includes(object : T) : Boolean
```

True if *object* is an element of *self*, false otherwise.

```
post: result = (self->count(object) > 0)
```

includesAll

```
List(T)::includesAll(c2 : Collection(T)) : Boolean
```

Does *self* contain all the elements of *c2* ?

```
post: result = c2->forAll(elem | self->includes(elem))
```

includesAll

```
List(T)::includesAll(c2 : List(T)) : Boolean
```

Does *self* contain all the elements of *c2* ?

```
post: result = c2->forAll(elem | self->includes(elem))
```

including

```
List(T)::including(object : T) : List(T)
```

Returns a new list containing all elements of self plus object added as the last element.

```
post: result = append(object)
```

indexOf

```
List(T)::indexOf(obj : T) : Integer
```

The one-based index of object obj in the list.

```
pre : includes(obj)
post : at(result) = obj
```

insertAt

```
List(T)::insertAt(index : Integer, object : T) : List(T)
```

Returns a new list consisting of self with object inserted at the one-based position index.

```
pre : 1 <= index and index <= size()
post: result->size() = size() + 1
post: Sequence{1..(index - 1)}->forAll(i | result->at(i) = at(i))
post: result->at(index) = object
post: Sequence{(index + 1)..size()}->forAll(i | result->at(i + 1) = at(i))
```

insertAt

```
List(T)::insertAt(object : T, index : Integer) : Void
```

The list is modifed to consist of self with object inserted at the one-based position index.

```
pre : 1 <= index and index <= size()
post: size() = size@pre() + 1
post: Sequence{1..(index - 1)}->forAll(i | at(i) = at@pre(i))
post: at(index) = object
post: Sequence{(index + 1)..size()}->forAll(i | at(i) = at@pre(i - 1))
```

isEmpty

```
List(T)::isEmpty() : Boolean
```

Is *self* an empty list?

```
post: result = (self->size() = 0)
```

joinfields

```
List(T)::joinfields(sep:String,begin:String,end:String) :String
```

Creates a string separated by sep and delimited with begin and end strings.

```
post: result = begin + Sequence{1..size()}->iterate(i; acc : String = '' |
    acc + if i = 1 then '' else sep endif + at(i).toString())
  + end
```

last

```
List(T)::last() : T
```

Returns the last element in self.

```
post: result = at(size())
```

max

```
List(T)::max() : T
```

The element with the maximum value of all elements in self. Elements must be of a type supporting the max operation. The max operation - supported by the elements - must take one parameter of type T. Integer and Real fulfill this condition.

```
post: result = self->iterate(elem; acc : T = self->any(true) | acc.max(elem))
```

min

```
List(T)::min() : T
```

The element with the minimum value of all elements in self. Elements must be of a type supporting the min operation. The min operation - supported by the elements - must take one parameter of type T. Integer and Real fulfill this condition.

```
post: result = self->iterate(elem; acc : T = self->any(true) | acc.min(elem))
```

notEmpty

```
List(T)::notEmpty() : Boolean
```

Is *self* not an empty list?

```
post: result = (self->size() <> 0)
```

prepend

```
List(T)::prepend(object : T) : List(T)
```

Returns a new list consisting of object, followed by all elements in self.

```
post: result->size = size() + 1
post: result->at(1) = object
post: Sequence{1..size()}->forAll(i | result->at(i + 1) = at(i))
```

product

```
List(T)::product(c2: Collection(T2)) : Set(Tuple(first: T, second: T2))
```

The cartesian product operation of *self* and *c2*.

```
post: result = self->iterate(e1; acc: Set(Tuple(first: T, second: T2)) = Set{} |
        c2->iterate(e2; acc2: Set(Tuple(first: T, second: T2)) = acc |
             acc2->including(Tuple{first = e1, second = e2})))
```

remove

```
List(T)::remove(element : T) : Void
```

Removes .all elements from self equal to element.

```
post: result = self@pre->reject(e = element)
```

removeAll

```
List(T)::removeAll(elements : Collection(T)) : Void
```

Removes .all elements from self equal to any of elements.

```
post: result = self@pre->reject(e | elements->includes(e))
```

removeAll

```
List(T)::removeAll(elements : List(T)) : Void
```

Removes .all elements from self equal to any of elements.

```
post: result = self@pre->reject(e | elements->includes(e))
```

removeAt

```
List(T)::removeAt(index : Integer) : T
```

Removes .and returns .the list element at index. Returns invalid for an invalid index.

```
pre: 1 <= index and index <= size()
post: size() = size@pre() - 1
post: Sequence{1..index}->forAll(i | at(i) = at@pre(i))
post: Sequence{(index+1)..size()}->forAll(i | at(i) = at@pre(i+1))
post: result = at@pre(index)
```

removeFirst

```
List(T)::removeFirst() : T
```

Removes .and returns .the first list element. Returns invalid for an empty list.

```
pre: 1 <= size()
post: size() = size@pre() - 1
post: Sequence{1..size()}->forAll(i | at(i) = at@pre(i+1))
post: result = at@pre(1)
```

removeLast

```
List(T)::removeLast() : T
```

Removes .and returns .the last list element. Returns invalid for an empty list.

```
pre: 1 <= size()
post: size() = size()@pre - 1
post: Sequence{1..size()}->forAll(i | at(i) = at@pre(i))
post: result = at@pre(size@pre())
```

reverse

```
List(T)::reverse() : List(T)
```

Returns a new list containing the same elements but with the opposite order.

```
post: result->size() = self->size()
post: Sequence{1..size()}->forAll(i | result->at(i) = at(size() - (i-1)))
```

selectByKind

```
List(T)::selectByKind(type : Classifier) : List(T1)
```

Returns a new list containing the non-null elements of self whose type is *type* or a subtype of *type*. The returned list element type T1 is the type specified as *type*.

```
post: result = self
    ->collect(if oclIsKindOf(type) then oclAsType(type) else null endif)
    ->excluding(null)
```

selectByType

```
List(T)::selectByType(type : Classifier) : List(T1)
```

Returns a new list containing the non-null elements of self whose type is *type* but which are not a subtype of *type*. The returned list element type T1 is the type specified as *type*.

```
post: result = self
    ->collect(if oclIsTypeOf(type) then oclAsType(type) else null endif)
    ->excluding(null)
```

size

```
List(T)::size() : Integer
```

The number of elements in the collection *self*.

```
post: result = self->iterate(elem; acc : Integer = 0 | acc + 1)
```

subSequence

```
List(T)::subSequence(lower : Integer, upper : Integer) : Sequence(T)
```

Returns a new sub-List of self starting at number lower, up to and including element number upper.

```
pre : 1 <= lower and lower <= upper and upper <= size()
post: result->size() = upper -lower + 1
post: Sequence{lower..upper}->forAll(i | result->at(i - lower + 1) = at(i))
```

sum

```
List(T)::sum() : T
```

The addition of all elements in *self*. Elements must be of a type supporting the + operation. The + operation must take one parameter of type T. It does not need to be commutative or associative since the iteration order over a list is well-defined. Integer and Real fulfill this condition.

```
post: result = self->iterate(elem; acc : T = 0 | acc + elem)
```

union

```
List(T)::union (s : List(T)) : List(T)
```

Returns a new list consisting of all elements in self, followed by all elements in s.

```
post: result->size() = size() + s->size()
post: Sequence{1..size()}->forAll(i | result->at(i) = at(i))
post: Sequence{1..s->size()}->forAll(i | result->at(i + size()) = s->at(i)))
```

<New sub-sub-section number> Iterations on Lists

There are no iterations defined for Lists since Lists are mutable and iteration domains are immutable. However the iterations defined for Sequences may be used without explicitly converting the List to a Sequence.

Invocation of one the following list iterations returning non-Lists

```
aList->iteration(...)
```

is therefore shorthand for

```
aList->asSequence()->iteration(...)
```

```
List(T)::any(i : T[?]) : T[?]
List(T)::collect(i : T[?]) : Collection(T1)
List(T)::collectNested(i : T[?]) : Collection(T1)
List(T)::exists(i : T[?]) : Boolean[?]
List(T)::exists(i : T[?], j : T[?]) : Boolean[?]
List(T)::forAll(i : T[?]) : Boolean[?]
List(T)::forAll(i : T[?], j : T[?]) : Boolean[?]
List(T)::isUnique(i : T[?]) : Boolean
List(T)::iterate(i : T[?]; acc : T2[?]) : T2[?]
List(T)::one(i : T[?]) : Boolean
```

Invocation of one the following list iterations returning Lists

```
aList->iteration(...)
```

is shorthand for

```
aList->asSequence()->iteration(...)->asList()
```

```
List(T)::reject(i : T[?]) : List(T)
List(T)::select(i : T[?]) : List(T)
List(T)::sortedBy(i : T[?]) : List(T)
```

<New sub-sub-section number> Operations on Collections

The following operations are added to the standard OCL collections

Collection::asList

```
Collection(T)::asList(T) : List(T)
```

Returns a new list containing all the elements of a collection. Whether the order is determinate depends on the derived collection type..

```
post: result->size() = size()
post: self->asSet()->forAll(e | result->count(e) = self->count(e))
```

Collection::clone

```
Collection(T)::clone(T) : Collection(T)
```

Collections are immutable so a clone returns self.

```
post: result = self
```

Collection::deepclone

```
Collection(T)::deepclone(T) : Collection(T)
```

Collections are immutable so a deep clone returns self.

```
post: result = self
```

<New sub-sub-section number> Operations on Bags

The following operations are added to the standard OCL bags

Bag::asList

```
Bag(T)::asList(T) : List(T)
```

Returns a new list containing all the elements of a bag in an indeterminate order.

Bag:: clone

```
Bag(T)::clone(T) : Bag(T)
```

Bags are immutable so a clone returns self.

Bag:: deepclone

```
Bag(T)::deepclone(T) : Bag(T)
```

Bags are immutable so a deep clone returns self.

<New sub-sub-section number> Operations on OrderedSets

The following operations are added to the standard OCL ordered sets

OrderedSet::asList

```
OrderedSet(T)::asList(T) : List(T)
```

Returns a new list that contains all the elements an ordered set in the same order.

```
post: Sequence{1..size()}->forAll(i | result->at(i) = self->at(i))
```

OrderedSet:: clone

```
OrderedSet(T)::clone(T) : OrderedSet(T)
```

OrderedSets are immutable so a clone returns self.

OrderedSet:: deepclone

```
OrderedSet(T)::deepclone(T) : OrderedSet(T)
```

OrderedSets are immutable so a deep clone returns self.

<New sub-sub-section number> Operations on Sequences

The following operations are added to the standard OCL sequences

Sequence::asList

```
Sequence(T)::asList(T) : List(T)
```

Returns a new list that contains all the elements a sequence in the same order.

```
post: Sequence{1..size()}->forAll(i | result->at(i) = self->at(i))
```

Sequence:: clone

```
Sequence(T)::clone(T) : Sequence(T)
```

Sequences are immutable so a clone returns self.

Sequence:: deepclone

```
Sequence(T)::deepclone(T) : Sequence(T)
```

Sequences are immutable so a deep clone returns self.

<New sub-sub-section number> Operations on Sets

The following operations are added to the standard OCL sets

Set::asList

```
Set(T)::asList() : List(T)
```

Returns a new list containing all the elements of a set in an indeterminate order.

Set:: clone

```
Set(T)::clone() : Set(T)
```

Sets are immutable so a clone returns self.

Set:: deepclone

```
Set(T)::deepclone() : Set(T)
```

Sets are immutable so a deep clone returns self.

Disposition: Resolved

Issue 19178: What happens when an exception is thrown by an exception handler.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

QVTo provides no guidance as to how a nested exception should be handled.

Suggest that like C++, a nested exception is ill-formed and the transformation terminates.

Resolution:

The QVTo exception mechanism is lightweight.

Throwing an exception while handling another makes it very difficult to guarantee that the handler for the first executes correctly.

Therefore declare nested exceptions as ill-formed.

Revised Text:

<< NB This issue is superseded by Issue 19208 >>

In 8.2.2.13 TryExp add

> The selected exceptClause completes before the TryExp completes. Consequently if an exception is raised during execution of the exceptClause, the execution of the exceptClause cannot complete and so execution of the transformation terminates without any further changes occurring. The trace records may be examined to determine what actions the transformation performed prior to termination.

Disposition: **Resolved**

Issue 19208: Issue 19178 resolution is rubbish

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The proposed resolution for Issue 19178 'clarifies' nested exceptions.

It is wrong. The confusion arises from C++'s problem with a nested exception in a destructor not in a handler. QVTo has no destructors so there is no problem.

Issue 19178 should be closed without change.

Do not apply chnages from Issue 19178 resolution.

Resolution:

For ease of reference, the erroneous 19178 revised text was:

> In 8.2.2.13 TryExp add
>
> The selected exceptClause completes before the TryExp completes. Consequently if an exception is raised during execution of the exceptClause, the execution of the exceptClause cannot complete and so execution of the transformation terminates without any further changes occurring. The trace records may be examined to determine what actions the transformation performed prior to termination.

We just need to make clear that the handling of nested exceptions is normal.

Revised Text:

<<Do not apply Issue 19178 revisions>>

In 8.2.2.13 TryExp add

A nested exception within an exceptClause terminates the exceptClause unless caught by a nested TryExp.

Disposition: **Resolved**

Disposition: Closed, no change

Issue 11061: Consider using asTuple instead of tuple keyword for TupleExp

Source:

Mariano Belaunde (France Telecom R&D, mariano.belaunde(at)orange.com)

Summary:

Consider using asTuple instead of tuple keyword for TupleExp since asX() is usually used for conversions.

Comment:

The TupleExp metaclass can in fact be replaced by a simple operation in the standard library.

Resolution

(1) Remove the TupleExp metaclass section description 8.2.2.21

(2) Within section 8.3.3 add the description of a new operation named asOrderedTuple defined as:

Object::asOrderedTuple : OrderedTuple(T).

Converts the object into an ordered tuple. If the object is a already an ordered type no change is done. If the object is a OCL Tuple, the list of attributes become the anonymous content of the ordered tuple. Otherwise, the operation creates a new tuple with a unique element being the object.

(3) Apply diagram update described in Appendix D.

Resolution:

There is no TupleExp or asTuple() in QVT 1.0 or 1.1 but there is an asOrderedTuple() with the resolved description. So it appears that a variant of the suggested resolution made it into QVT 1.0.

Disposition: **Closed, No Change**

Issue 12200: There is a reference to a figure that does not exist : figure 1.

Source:

THALES (Eric Maes, eric.maes(at)fr.thalesgroup.com)

Summary:

There is a reference to a figure that does not exist : figure 1.

Discussion:

Figure numbering was resolved in the QVT 1.0 FAS.

Disposition: **Closed, No Change**

Issue 12260: Section: 7.13.3 / 8.4.2.

Source:

Forschungszentrum Karlsruhe(Jens Kübler, cleanerx(at)online.de)

Summary:

The specification introduces comments by concrete syntax. Comments within the abstract syntax are not considered. This is i.e. undesirable for automated analysis of software product quality to which transformations are subject. One would again need to analyze the AST instead of the transformation metamodel. So I propose to introduce comments for transformations.

Discussion:

All Abstract Syntax elements extend EMOF::Element and so the Element::ownedComment property is available for comments.

Disposition: **Closed, No Change**

Issue 12367: Issue against QVT ptc/07-07-07 : relational grammar.

Source:

NIST (Mr. Peter Denno, peter.denno(at)nist.gov)

Summary:

Section 7.13.5 Relation BNF

The grammar rule for template is incorrect (It does not allow a

template to have embedded within it another template. The example in

Appendix A will not compile with such a grammar.)

It says:

<propertyTemplate> ::= <identifier> '=' <OclExpressionCS>

It should says:

<propertyTemplate> ::= <identifier> '=' (<template> |

<oclExpressionCS

Discussion:

No. The grammar also redefines:

```
<OclExpressionCS> ::= <PropertyCallExpCS> | <VariableExpCS> | <LiteralExpCS> |
<LetExpCS> | <IfExpCS> | '(' <OclExpressionCS> ')' | <template>
```

Disposition: **Closed, No Change**

Issue 12519: Errors and anomalies in QVT 1.0 07-07-08 ZIP qvt_metamodel.emof.xml.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Use of automated tooling to support comparison of the models developed initially as part of the Eclipse GMT/UMLX project and being transferred to the Eclipse QVT Declarative/QVT Operational Mappings Projects reveals the following errors and anomalies in qvt_metamodel.emof.xml in the 07-07-08 ZIP.

Note that these errors and anomalies are not the same as those separately reported for the QVT_1.0.mdl from which the EMOF was notionally auto-generated.

EMOF files resolving these anomalies are attached.

Discussion:

QVT 1.1 issued revised files based on Eclipse QVT contributions.

Issue 12518: QVT 1.2 is providing non-normative UML files.

Disposition: **Closed, No Change**

Issue 12520: Errors and anomalies in QVT 1.0 07-07-08 ZIP emof.ecore.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Use of automated tooling to support comparison of the models developed

initially as part of the Eclipse GMT/UMLX project and being transferred to the Eclipse QVT Declarative/QVT Operational Mappings Projects reveals the following errors and anomalies in emof.ecore in the 07-07-08 ZIP.

Note that these errors and anomalies are not the same as those separately reported for the QVT_1.0.mdl from which the Ecore was notionally auto-generated.

An Ecore file resolving these anomalies is attached.

'nsURI' for 'EMOF' should be 'http://schema.omg.org/spec/MOF/2.0/emof.xml' rather than 'http:///emof.ecore'

'name' for 'EMOF' should be 'EMOF' rather than 'emof'

'name' for 'Property.isID' should be 'isID' rather than 'isId'

'Factory' should be defined

'ReflectiveCollection' should be defined

'ReflectiveSequence' should be defined

'Comment.body' should be defined

'Factory.package' should be defined

'Element.tag' should be undefined

'eOpposite' for 'Tag.element' should be undefined

'lowerBound' for 'Operation.class' should be '0' rather than '1'

'lowerBound' for 'Type.package' should be '0' rather than '1'

'lowerBound' for 'Property.class' should be '0' rather than '1'

'ordered' for 'Class.superClass' should be 'false' rather than 'true'

'ordered' for 'Comment.annotatedElement' should be 'false' rather than 'true'

'ordered' for 'Element.ownedComment' should be 'false' rather than 'true'

'ordered' for 'Operation.raisedException' should be 'false' rather than 'true'

'ordered' for 'Package.nestedPackage' should be 'false' rather than 'true'

'ordered' for 'Package.ownedType' should be 'false' rather than 'true'

'ordered' for 'Tag.element' should be 'false' rather than 'true'

'defaultValueLiteral' for 'Class.isAbstract' should be 'false' rather than undefined

'defaultValueLiteral' for 'MultiplicityElement.isOrdered' should be 'false' rather than undefined

'defaultValueLiteral' for 'MultiplicityElement.isUnique' should be 'true' rather than undefined

'defaultValueLiteral' for 'MultiplicityElement.lower' should be '1' rather than undefined

'defaultValueLiteral' for 'MultiplicityElement.upper' should be '1' rather than undefined

'defaultValueLiteral' for 'Property.isComposite' should be 'false' rather than undefined

'defaultValueLiteral' for 'Property.isDerived' should be 'false' rather than undefined

'defaultValueLiteral' for 'Property.isReadOnly' should be 'false' rather than undefined

'Element.container()' should be defined

'Element.equals(object)' should be defined

'Element.get(property)' should be defined

'Element.getMetaClass()' should be defined

'Element.isSet(property)' should be defined

'Element.set(property,object)' should be defined

'Element.unset(property)' should be defined

'Extent.elements()' should be defined

'Extent.useContainment()' should be defined

'Factory.convertToString(dataType,object)' should be defined

'Factory.create(metaClass)' should be defined

'Factory.createFromString(dataType,string)' should be defined

'ReflectiveCollection.add(object)' should be defined

'ReflectiveCollection.addAll(objects)' should be defined

'ReflectiveCollection.clear()' should be defined

'ReflectiveCollection.remove(object)' should be defined

'ReflectiveCollection.size()' should be defined

'ReflectiveSequence.add(index,object)' should be defined

'ReflectiveSequence.get(index)' should be defined

'ReflectiveSequence.remove(index)' should be defined

'ReflectiveSequence.set(index,object)' should be defined

'Type.isInstance(object)' should be defined

'URIExtent.contextURI()' should be defined

'URIExtent.element(uri)' should be defined

'URIExtent.uri(element)' should be defined

Unnavigable 'opposite' of 'Class.superClass' should be modelled

Unnavigable 'opposite' of 'Element.ownedComment' should be modelled

Unnavigable 'opposite' of 'Package.nestedPackage' should be modelled

Unnavigable 'opposite' of 'Property.opposite' should be modelled

Discussion:

These changes affect non-normative EMOF files which were corrected when QVT 1.1 issued revised files based on Eclipse QVT contributions.

Disposition: **Closed, No Change**

Issue 12521: Errors and anomalies in QVT 1.0 07-07-08 ZIP essentialocl.ecore.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Use of automated tooling to support comparison of the models developed initially as part of the Eclipse GMT/UMLX project and being transferred to the Eclipse QVT Declarative/QVT Operational Mappings Projects reveals the following errors and anomalies in emof.ecore in the 07-07-08 ZIP.

Note that these errors and anomalies are not the same as those separately reported for the QVT_1.0.mdl from which the Ecore was notionally auto-generated.

An Ecore file resolving these anomalies is attached.

'nsURI' for 'EssentialOCL' should be ' http://schema.omg.org/spec/QVT/1.0/essentialocl.xml' rather than ' http://www.schema.omg.org/spec/OCL/2.0/essentialocl'

'name' for 'EssentialOCL' should be 'EssentialOCL' rather than 'essentialocl'

'name' for 'ExpressionInOcl.contextVariable' should be 'contextVariable' rather than 'context'

'name' for 'Variable.representedParameter' should be 'representedParameter' rather than 'bindParameter'

'NavigationCallExp' should be defined

'OpaqueExpression' should be undefined

'TypeType' should be defined

'CollectionKind::Collection' should be defined

'eSuperTypes' for 'ExpressionInOcl' should be 'TypedElement' rather than 'OpaqueExpression'

'eSuperTypes' for 'PropertyCallExp' should be 'NavigationCallExp'

'eSuperTypes' for 'AnyType' should be 'Type' rather than 'Class','Type'

'lowerBound' for 'CollectionType.elementType' should be '1' rather than '0'

'upperBound' for 'ExpressionInOcl.parameterVariable' should be '-1' rather than '1'

'abstract' for 'CollectionType' should be 'false' rather than 'true'

'containment' for 'TupleLiteralPart.attribute' should be 'false' rather than 'true'

'ordered' for 'CollectionLiteralExp.part' should be 'false' rather than 'true'

'ordered' for 'ExpressionInOcl.parameterVariable' should be 'false' rather than 'true'

'ordered' for 'LoopExp.iterator' should be 'false' rather than 'true'

'ordered' for 'TupleLiteralExp.part' should be 'false' rather than 'true'

Unnavigable 'opposite' of 'CallExp.source' should be modelled

Unnavigable 'opposite' of 'CollectionRange.first' should be modelled

Unnavigable 'opposite' of 'CollectionRange.last' should be modelled

Unnavigable 'opposite' of 'EnumLiteralExp.referredEnumLiteral' should be modelled

Unnavigable 'opposite' of 'ExpressionInOcl.bodyExpression' should be modelled

Unnavigable 'opposite' of 'ExpressionInOcl.contextVariable' should be modelled

Unnavigable 'opposite' of 'ExpressionInOcl.parameterVariable' should be modelled

Unnavigable 'opposite' of 'ExpressionInOcl.resultVariable' should be modelled

Unnavigable 'opposite' of 'IfExp.condition' should be modelled

Unnavigable 'opposite' of 'IfExp.elseExpression' should be modelled

Unnavigable 'opposite' of 'IfExp.thenExpression' should be modelled

Unnavigable 'opposite' of 'IterateExp.result' should be modelled

Unnavigable 'opposite' of 'LoopExp.body' should be modelled

Unnavigable 'opposite' of 'OperationCallExp.argument' should be modelled

Unnavigable 'opposite' of 'OperationCallExp.referredOperation' should be modelled

Unnavigable 'opposite' of 'PropertyCallExp.referredProperty' should be modelled

Unnavigable 'opposite' of 'Variable.initExpression' should be modelled

Unnavigable 'opposite' of 'VariableExp.referredVariable' should be modelled

Discussion:

These changes affect non-normative EssentialOCL files which were corrected when QVT 1.1 issued revised files based on Eclipse QVT contributions.

Disposition: **Closed, No Change**

Issue 13988: Capitalization of leading characters in multiword operation names.

Source:

Open Canarias, SL (Mr. E. Victor Sanchez, vsanchez(at)opencanarias.com)

Summary:

Section 8.3.4.11, page 110:

The operaton "Element::deepclone()" should read "Element::deepClone()". I understand that "Subobjects" do not capitalize the leading 'o' for the word "objects", but in all other cases it seems consistent to use capitals for words' leading letters. Both the title and the operation signature should be updated accordingly. This issue is similar to 13913.

The same applies to Section 8.3.7.3 (page 113), "defaultget", which would read "defaultGet", and 8.3.8.4 (page 114), "joinfields", which would read "joinFields". Again both the titles and the operation signatures should be updated accordingly.

Discussion:

The suggested spellings are better but the existing spellings are established. It does not seem worth changing.

Disposition: **Closed, No Change**

Issue 14573: A referenced picture is missing.

Source:

InterComponentWare AG (Markus von Rueden, markus.vonrueden(at)icw.de)

Summary:

"The dottet arrows in the picture below sho the dependencies between the patterns, with the following interpretatoin: ..."

But there is no picture :(

Discussion:

The figure numbering was resolved in QVT 1.1.

Disposition: **Closed, No Change**

Issue 14835: Please provide a non-null text in the Scope section of documents ptc/09-12-06 and ptc/09-12-05.

Source:

Unisys (Dr. Doug Tolbert, dtolbert408(at)gmail.com)

Summary:

Please provide a non-null text in the Scope section of documents ptc/09-12-06 and ptc/09-12-05

Discussion:

The QVT 1.0 scope text was restored for the QVT 1.0 FAS.

Disposition: **Closed, No Change**

Issue 15215: QVT1.1: Add an operation Model::getURI().

Source:

NASA (Dr. Nicolas F. Rouquette, nicolas.f.rouquette(at)jpl.nasa.gov)

Summary:

Sometimes, it is useful to get the URI corresponding to the resource of a given transformation input model parameter.

I suggest adding an operation for this purpose in clause 8.3.5 Operations on models.

Model::getURI() : String

Returns the string of the URI corresponding to the model's resource. This operation produces an empty string for a model corresponding to an output parameter

Discussion:

This seems reasonable but:

UML already provides an inherited Package::URI field so Model::getURI() would be confusing. Perhaps getExternalURI() or getDocumentURI(). However this is not a QVT issue; if there is a requirement for contextual knowledge of a Model, surely UML should provide it?

A QVT transformation hides its context and for a transformation realized by a cascade communicating through memories or pipes, what would the URI be?

Users who understand and control the context can pass the context as a parameter/control model.

More generally there is a problem in defining the URI of a transformation that generates a data-dependent number of output models. Any get context solution would address this.

Disposition: **Closed, No Change**

Issue 15416: Derived properties in QVTr.

Source:

NIST (Mr. Peter Denno, peter.denno(at)nist.gov)

Summary:

If I want to specify a uni-directional transformation using QVTr, is it ok to use "derived" properties in source domains' object templates? I guess since the transform is not meant to be run in the opposite direction, this will not create a problem?

If that is allowed, it would be a good additional feature of QVTr to allow the definition of new derived properties right in the transform, instead of having them only in the source metamodel.

For example

transform A (...) {

property Class::allSuperClass : Class {

self->closure(self.superClass) // closure might not be standard collection op but I used it just to demonstrate the point

}

relation B {

checkonly domain source c:Class {

allSuperClass = super:Class {...}

}

}

}

does this make sense?

Discussion:

Use of derived properties while checking should pose no problem. The existing text on values is sound.

Use of derived properties while enforcing is highly suspect. Derived properties are often readonly. If changeable then the derived interrelationships are an issue for the metamodel not for QVTr.

When QVTr acquires well-formedness rules, an enforeable readonly property could be diagnosed.

QVTr supports queries that can be used to emulate custom derived properties.

Disposition: **Closed, No Change**

Issue 17538: Consider submitting the QVTO profile out of UML Profile for NIEM, section 9-2 to QVT 1.2

Source:

NASA (Dr. Nicolas F. Rouquette, nicolas.f.rouquette(at)jpl.nasa.gov)

Summary:

Section 9-2 in the UML Profile for NIEM Beta2 document describes an interesting diagrammatic notation for describing the salient organization of a QVTO transformation.

Based on the notation shown in figures 9-2, 9-3, 9-4 and others, this notation clearly involves some kind of UML profile for describing a QVTO transformation whose stereotypes

include <<OperationalTransformation>>, <<MappingOperation>>, <<disjuncts>> and <<inherits>>. The figures in section 9 make a compelling illustration of the utility of a UML Profile for QVTO Transformation.

I believe this UML profile for QVTO is a novel contribution of the UML Profile for NIEM FTF; unfortunately, the document does not describe it and this QVTO Transformation profile

is not mentioned anywhere in the UML Profile for NIEM inventory or in any of the machine readable artifacts.

Discussion:

This would be an interesting enhancement and could perhaps form an Annex. However it is beyond the scope for the active members of this RTF. There has been no response from other RTF members to a couple of help-wanted requests, so this enhancement can be closed for lack of interest.

Disposition: **Closed, No Change**

Issue 18325: Intermediate data not allowed for libraries

Source:

Christopher Gerking, cgerking(at)campus.upb.de

Summary:

The QVT spec says that "an operational transformation may use for its definition intermediate classes and intermediate properties."

Is intermediate data actually restricted to plain transformations? In other words, is intermediate data unsupported for libraries? The eclipse QVTo implementation sticks to this interpretation and actually supports intermediate data only for plain transformations, which is a limitation I don't see a reason for.

Discussion:

The metamodels are very clear; no intermediate classes/properties in a Module.

Today: *8.1.3 A library contains definitions that can be reused by transformations.* Intermediate data are not inherently re-useable, but could be formulated as a metamodel if required.

Future: if the dual Package/Class inheritance of Module is resolved to make Library Package-like and OperationalTransformation Class-like, promoting intermediates to Module will create more problems to solve.

Disposition: **Closed, No Change**

Disposition: Transferred

Issue 11056: Provide the list of reflective MOF operations that are available

Source:

Mariano Belaunde (France Telecom R&D, mariano.belaunde(at)orange.com)

Summary:

Is not very clear what are the reflective MOF operations that are available to QVT operational transformation writers

Resolution:

Once OCL resolves outstanding issues regarding reflection as required by the draft OCL 2.5 RFP, the available operations should be obvious.

Disposition: **Transferred to OCL**

Disposition: Duplicate / Merged

Issue 13223: explanation of the operation: 'List(T)::insertAt(T,int).

Source:

Open Canarias, SL (Mr. E. Victor Sanchez, vsanchez(at)opencanarias.com)

Summary:

The explanation of the operation: 'List(T)::insertAt(T,int) : Void' in Section 8.3.8.3 has a mistake concerning the index associated with the first element for OCL collections. The index starts at one, '1', not zero, '0'. Suggestion: Substitute the word 'zero' by the word 'one', so that the text reads: "The index starts at one (in compliance with OCL convention)."

Resolution:

Yes. Improved wording in Issue 19146.

Disposition: **See issue 19146 for disposition**

Issue 13228: Missing operations on Lists.

Source:

Open Canarias, SL (Mr. E. Victor Sanchez, vsanchez(at)opencanarias.com)

Summary:

For 'removeAt' I specify 'one' as the starting index value. Suggestion: - List(T)::remove(T) : T Removes a value from the list. - List(T)::removeAt(int) : T Removes a value from the mutable list at the given position. The index starts at one (in compliance with OCL convention). The return value is the value removed from the mutable list. - List(T)::clear() : Void Removes all values in the mutable list.

Resolution:

Yes. Revised wording in 132 is explicitly one-based.

Disposition: **See issue 19146 for disposition**

Issue 13251: add the following operations to mutable lists.

Source:

Open Canarias, SL (Mr. E. Victor Sanchez, vsanchez(at)opencanarias.com)

Summary:

In the spirit of issue #13228, and in conversation with Mariano Belaunde, we think it is worth considering also adding the following operations to mutable lists. As specific usages of 'removeAt()': 'removeFirst()' and 'removeLast()'. And also: 'removeAll(Collection(T))'. Notice that this one includes the case of removing values specified inside mutable lists as long as ListType inherits CollectionType. Suggestion: Add the following text to section 8.3.8: - List(T)::removeFirst() : T Removes the first value of the mutable list. The return value is the value removed. - List(T)::removeLast() : T Removes the last value of the mutable list. The return value is the value removed. - List(T)::removeAll(Collection(T)) : Void Removes from the mutable list all the values that are also present in the specified collection.

Resolution:

And remove() from Issue 13228. A List is not a Collection so we need two removeAll's.

Additional operations in Issue 19146.

Disposition: **See issue 19146 for disposition**

Issue 13266: Page 72, Figure 8-2.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: in MappingParameter class: "refiningParameter" and "refinedDomain".

discussion: while a mappingOperation refines a Relation, a mappingParamer should "refer" a RelationDomain. In the text of MappingParameter section, the concept of "refers" is used several time instead of "refines".

suggestion: replace "refiningParameter" and "refinedDomain" by "referringParameter" and "referredDomain".

Resolution:

Duplicates in part Issue 12527.

Disposition: **See issue 12527 for disposition**

Issue 13271: Page 83: Section 8.2.1.22 MappingCallExp.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem's text: Superclasses OperationCallExp
discussion: It should extend ImperativeCallExp instead. The diagram is well showed.
suggestion: Replate "OperationCallExp" by "ImperativeCallExp".

Resolution:

Duplicates in part Issue 12527.

Disposition: **See issue 12527 for disposition**

Issue 13987: Minor typographical error in ImperativeIterateExp notation.

Source:

Open Canarias, SL (Mr. E. Victor Sanchez, vsanchez(at)opencanarias.com)

Summary:

Section 8.2.2.7, page 95:

The text "list[condition]; // same as list->xselect(i; condition)" should read "list[condition]; // same as list->xselect(i | condition)". Notation of imperative iterate expressions state that the condition section must be separated from prior artifacts (iterators or target initializations), where present, by a "|" character, not a semicolon ";".

Resolution:

Duplicates in part Issue 13282.

Disposition: **See issue 13282 for disposition**

Issue 15524: Rule Overriding in QVTr.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

You have reached the edge of the specification as written.

1: Yes

2: Yes

3: Yes

4: Yes

I gave some consideration to this for UMLX.

I felt that an abstract 'rule' could define a 'subrule' obligation, which would require an identical match signature, since if the override was narrower it would not fulfill the obligation and if it was wider the additional width would not be permitted by the invocation of the abstract 'rule'.

I felt that all concrete rules should always be matched to ensure that addition of extended functionality did not change previous behaviour. This complies with UMLX's all maximal matches philosophy. Keys in QVTr, Evolution Ids in UMLX can ensure that derived rules reuse inherited matches.

I think a transformation being both a package and a class introduces some difficult compatibility issues to be studied.

Transformation extension is also poorly defined giving additional imprecision when considering the combination of transformation extension and rule override.

My ideas for UMLX were not complete but I think that they may be sounder than QVTr's.

Resolution:

This issue was inadvertently raised from Issue 15417 correspondence. Close it as merged even though Issue 15417 needs further work.

Disposition: **See issue 15417 for disposition**

Issue 19095: Not possible to remove from a mutable List.

Source:

University of Paderborn (Christopher Gerking, christopher.gerking(at)upb.de)

Summary:

A List is a mutable type. However, there is only an interface for adding to a List, not for removing from a List. Analogously to the operation

List(T)::add(T) : Void

there should be something like:

List(T)::remove(T) : Void

Resolution:

Duplicates in part Issue 13251.

Disposition: **See issue 19146 for disposition**

Issue 19174: List does not support asList().

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The asList() operation is defined for all concrete collection types. Consequently, it should be defined on List itself (returning self).

Suggestion: specify the operation for the Collection type:

Collection(T)::asList() : List(T)

Add another redefinition for the List type:

List(T)::asList() : List(T)

Resolution:

The additional operation is shown in the Issue 19146 resolution..

Disposition: **See issue 19146 for disposition**

Disposition: Deferred

Issue 10934: 9.18 Undefined syntax.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The syntax for TransformationName, DirectionName, MappingName, PropertyName and VariableName is undefined. Presumably each of these is an Identifier (below) when defining, but a PathNameCS when referencing.

The syntax for PackageName is undefined. Presumably it is an OCL PathNameCS.

The syntax for ValueOCLExpr is undefined. This is presumably an OclExpressionCS.

The syntax for BooleanOCLExpr is undefined. This could be an OclExpressionCS of Boolean type but ...

The syntax for SlotOwnerOCLExpr is undefined. This could be an OclExpressionCS of Class type but ...

If BooleanOCLExpr and SlotOwnerOCLExpr are parsed as OclExpressionCS the 'default' prefix causes a major ambiguity for 'default(...).xx' as a parenthesised slot owner or function call.

It is necessary to make 'default' a reserved word within OCL expressions.

Suggest: define BooleanOCLExpr and SlotOwnerOCLExpr very narrowly.

Predicate ::= SimpleNameCS ("." SimpleNameCS)* "=" OclExpressionCS

Assignment ::= ["default"] SimpleNameCS ("." SimpleNameCS)* ":=" OclExpressionCS

Discussion:

It is hard to tackle this properly until we have the model-driven specification generated technology and clear OCL grammar that OCL 2.5 will pioneer.

Disposition: **Deferred**

Issue 10935: 9.18 Identifiers.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The syntax of identifiers is undefined.

The syntax for mapping clearly prohibits the use of a direction named 'where'.

Suggest: identifier is an OCL simpleName, less the new reserved words (check default enforce imports map realize refines transformation uses where)

Suggest: a string-literal may be used as an identifier to support awkward identifiers such as 'where'.

Discussion:

It is hard to tackle this properly until we have the model-driven specification generated technology and clear OCL grammar that OCL 2.5 will pioneer.

Disposition: **Deferred**

Issue 10936: 9.18 Undefined semantics.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The whole Concrete Syntax section deserves a much more substantial description.

In particular...

The mechanism by which a package name is located is unresolved, perhaps deliberately,

but the omission should be explicit.

What constraints exist on forward referencing of names?

Transformations and mappings could be ordered so that forward references are avoided,

but large modules benefit from an alphabetical ordering of elements, so requiring

a parser friendly order is not user friendly.

Discussion:

It is hard to tackle this properly until we have the model-driven specification generated technology that OCL 2.5 will pioneer.

Specifically the specification defines a fixed point truth with little regard as to how this may be achieved. The direction of references is therefore irrelevant.

Disposition: **Deferred**

Issue 11686: Section: A1.1.1.

Source:

Siegfried Nolte siegfried(at)siegfried-nolte.de

Summary:

top relation AssocToFKey { pKey : Key; ... when { ...; pKey = destTbl.key; } ... } In my opinion that doesn't work. pKey is of type Key, destTbl.key is of type OrderedSet key, isn't it ?

Discussion:

I think that the intention is for pKey to 'loop' over each destTbl.key, but I cannot see text to justify this.

Collection matching needs revision, in particular the untenable prohibition on nested collections.

Disposition: **Deferred**

Issue 11690: Section: 7.13.5

Source:

Siegfried Nolte siegfried(at)siegfried-nolte.de

Summary:

The "import" feature of Relations Language is not yet explained. And there is no example for it, too. For instance what does the "unit" in "import <unit>;" mean ?

Discussion:

OCL 2.5 should resolve the semantics of an import for Complete OCL. The QVTr (and QVTc) at least syntaxes should be compatible.

Disposition: **Deferred**

Issue 12213: Relations Language: how will metamodels get into a transformation scrip

Source:

Siegfried Nolte siegfried(at)siegfried-nolte.de

Summary:

Concerning to Relations Language, how will the metamodels get into a transformation script ? Is there a technique similar to Operational Mappings using metamodel declaration and modeltypes ? The RL sample transformation script in annex A.1 (pp 164) doesn't declare the metamodels. The OM sample (A.2.3) does. There is some syntax for declaring and using metamodels and modeltypes in OM (pp118), there isn't for RL (pp38).

Initial Response

I don't think QVTo is any different to QVTr.

Although A.2.3 happens to provide syntax for metamodels these have names that are distinct from the subsequent transformation and are separated by an explanatory paragraph.

How short names such as "UML" are resolved to a particular version, location and representation of a meta-model is tool-specific.

http://www.eclipse.org/gmt/umlx/doc/EclipseAndOMG08/ModelRegistry.pdf

describes one re-usable solution to the problem. It is used by QVTr and QVTc editors.

Discussion:

I now consider the failure of QVTr, QVTc and Complete OCL to provide an ability to import a Document URI is a language bug and should not depend on external implementation magic. As for Issue 11690: OCL 2.5 should resolve the semantics of an import for Complete OCL. The QVTr (and QVTc) at least syntaxes should be compatible.

Disposition: **Deferred**

Issue 12370: Section 8.7.1a production rule seems to be missing

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

a production rule seems to be missing: <module_element> ::= <modeltype> Since, a transformation file can define several transformations and libraries (modules), it is desirable having the possibility of defining modeltypes exclusively to a module. These "local" modelTypes should belong to the scope of a module, and it shouldn't be accessible to the remaining defined modules (unless the use of extension mechanisms is specified).

Discussion:

It is not clear whether this is necessary or just a convenience.

Disposition: **Deferred**

Issue 13054: MOF-QVT 1.0: 7.11.3.6 (and 7.11.1.1) BlackBox operation signature difficulties

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

In 7.11.3.6 (and 7.8) the ordering of parameters is implied but not explicit. Presumably the first is the output (enforced direction) so:

package QVTBase

context TypedModel

def: allUsedPackage() : Set(EMOF::Package)

= self.dependsOn.allUsedPackage()->asSet()->union(self.usedPackage)

endpackage

package QVTRelation

context RelationImplementation

inv RootNodeIsBoundToRootVariable : self.inDirectionOf.allUsedPackage()->includes(self.impl.ownedParameter->at(1).type._package)

endpackage

This is not satisfied by almost the last line of RelToCore in 10.3:

enforce domain core me:OclExpression{} implementedby CopyOclExpession(re, me);

which seems to have output second.

More significantly CopyOclExpession(re, me) is not meaningful as a black box signature, since it is a static function and EMOF has no way to define static functions. Perhaps it is a query, for which the declaration was omitted from the example. If this is the case, it should be made clear that black box operations must be declared internally as queries and bound externally to static operations of the transformation.

This semantic clumsiness could be resolved, if, within a transformation,

relation, domain or query, self is bound to a transformation instance. Queries would then be normal non-static operations of the transformation (class) and the implementedBy operation call would be a normal implicit call via self of a non-static transformation operation or query. (A RelationCallExp would also deviate less from OCL than it currently does.) This would also allow a transformation to extend a Utility class that provided the required black box operations. Since queries and relations are declarative, it is not obvious that there need be any prohibition on the content of an extended Class; if the Class has properties, these cannot mutate during a query or relation match, so the properties are ok; they might even permit useful behavioural tailoring. For instance an 'inherited' Boolean mangledNames property could influence the mapping of names between input and output.

The RelToCore example can then be mended by declaring that:

RelToCore(...) extends utils::CopyUtilities

and externally binding the utils model name to a package that has a CopyUtilities class with suitable a CopyOclExpession operation.

Discussion:

I would like to see a working implementation of this before resolving.

Disposition: **Deferred**

Issue 13082: current abstract syntax of ImperativeOCL introduces a couple of unclear situations

Source:

Not recorded

Summary:

Major Problem:

(1) The current abstract syntax of ImperativeOCL introduces a couple of unclear situations. This may lead to incompatible QVT implementations.

Further Problems:

(2) Control flow constructs introduced by ImperativeOCL are redundant compared with existing conventional OCL constructs.

(3) Several OCL equivalence rules break when ImperativeOCL is present.

Detailed problem description:

(1) The current abstract syntax of ImperativeOCL introduces a couple of unclear situations. This may lead to incompatible QVT implementations. In the abstract syntax, ImperativeOCL expressions / statements are inherited from OclExpression. Therefore, conventional OCL expressions may (and will) contain sub-expressions that are actually ImperativeOCL expressions. In conventional OCL, the interpretation of an expression under a given environment is a value. In ImperativeOCL, the interpretation of an expression is a value and a new environment (state,variables). This extended interpretation is not given for conventional OCL expressions, leading to undefined operational semantics of those expressions.

Example: Given the following compute expression:

```
compute(z:Boolean) {
var x : Boolean := true
var y : Boolean := true
if ((x:=false) and (y:=false)) { ... }
z := y
}
```

What is the value of this expression: is it true or false (It depends on whether the 'and' operator is evaluated short-circuit or strict). The situation is similar for the evaluation of the other logical connectives, forAll, and exists when these expressions contain imperative sub-expressions.

(2) Control flow constructs introduced by ImperativeOCL are redundant compared with existing conventional OCL constructs. Some of the new language features in ImperativeOCL such as forEach and the imperative conditional are not really necessary. Their effect can already be achieved using conventional OCL expressions:

For example:

```
company.employees->forEach(c) { c.salary := c.salary * 1.1}
```

has the same effect as

```
company.employees->iterate(c; r:OclAny=Undefined |
c.salary := c.salary * 1.1
)
```

and

```
if ( x < 0 ) { x := 0 } else { x := 1 } endif
```

is the same as

if x < 0 then x := 0 else x := 1 endif

(3) Several OCL equivalence rules break when ImperativeOCL is present.

In conventional OCL, several equivalence rules well known from logic hold. Allowing OCL expression to contain imperative sub-expressions breaks these equivalence rules.

Examples:

let x=e1 in e2 equiv. e2 { all occurences of x replaced by e1 }
e1 and e2 equiv. e2 and e1

These equivalences do not necessarily hold if e1 or e2 are allowed to have side-effects.

Proposed solution:

(A) - (The cheap solution.) State textually that conventional OCL expressions (as described in the OMG OCL spec.) are not allowed to have side effects unless used as part of a top level ImperativeOCL expression. Therefore, even in a system supporting ImperativeOCL, class invariants, and pre- and postconditions (e.g.) will not be allowed to contain ImperativeOCL sub-expressions.

State explicitly that the redundant flow control statements have been introduced (solely) to write concise imperative programs and that the side-effect free forms of conditional evaluation ('if-then-else-endif' and 'iterate') shall not be used to program side-effects (instead, the ImperativeOCL forms shall be used).

(B) - (Major rework.) Rework the abstract syntax to reuse OCL expressions by composition rather than by inheritance. Imperative expressions (=> rename to 'statements') then may contain sub-statements and OCL expressions; OCL expressions are reused unchanged from the OCL spec (no imperative sub-expressions, no side-effects).

These issues have been discussed on the MoDELS 2008 OCL Workshop, more details can be found at http://www.fots.ua.ac.be/events/ocl2008/PDF/OCL2008_9.pdf

Discussion:

A rework may be necessary to clarify whether the claim that QVT (and consequently QVTo) is an extension of OCL needs to be amended.

Disposition: **Deferred**

Issue 13103: element creation and element attachment/detachment to/from an extent

Source:

Open Canarias, SL (Mr. E. Victor Sanchez, vsanchez(at)opencanarias.com)

Summary:

Suggestion: In the Operational Mappings language, element creation and element attachment/detachment to/from an extent should be seen and treated as two different and independent activities. Once an element is created via an ObjectExp, this element is usually attached to another element immediately afterwards, which becomes its container, so the ObjectExp semantics of assigning it to an explicit or inferred extent becomes an unnecessary overhead. And also there are other times when we need to assign to an extent some model elements that may have been created at an unrelated time. They could even exist prior to the transformation execution. A case where this is relevant is with the result of a 'Element::clone()' or 'Element::deepclone()' operation execution. Is it expected that these model elements must belong by default to the same model as the original? How to clone parts of an extent with direction kind == 'in'? How to make them become part of the Set of root elements of another, different extent?

Preview Resolution:

This seems to be a misunderstanding. ObjectExp does not require a created object to be attached to an inferred extent.

This functionality seems to be present already.

Creation without attachment requires a null or null-valued extent during creation.

Attachment after creation can occur through an ObjectExp update to the required extent.

Re-attachment presumably occurs through an ObjectExp update to the new extent.

Detachment then naturally occurs through an ObjectExp update to a null-valued extent.

Preview Revised Text:

In 8.2.1.24 ObjectExp add after the first paragraph

Object creation, initialisation and residence are separate activities.

Object creation occurs when the *referredObject* has a null value; it is skipped if the *referredObject* variable references an existing object.

Object initialization always occurs, but may be trivial if the *body* is empty.

Object residence is left unchanged when the *extent* is omitted, so object creation will normally result in an object without any residence; the residence will be established as soon as the created object is put at the target end of some composition relationship. An explicit object residence may be established by specifying the model parameter for the required extent as the *extent*. Specifying an *extent* with a null value ensures that the created object has no residence; this may remove the residence of a pre-existing object.

Discussion

Discussion, primarily on Eclipse qvto-dev identified the lack of an overview of the intent of an Extent to motivate the detailed behaviours amended above in ObjectExp but not revised for InstantiationExp.

Disposition: **Deferred**

Issue 13158: QVT Relations and working with stereotypes.

Source:

Siegfried Nolte siegfried(at)siegfried-nolte.de

Summary:

QVT Relations and working with stereotypes: Is there something like a QVT-R standard library with methods on stereotypes in it? There is none in the specification; compare QVT-R, there are some methods. Are there some else options for accessing stereotypes with QVT-R?

Discussion:

OCL 2.5 should provide stereotype support that can be extended if necessary by QVTr

Disposition: **Deferred**

Issue 13168: Typedef aliases issue.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Creating aliases seems to be a good idea specially when dealing with complex types. However, for that purpose, it is not clear to me that we need to create a (useless) typedef just to represent an alias in the concrete syntax. In practice aliases are very useful when writing transformations (in concrete syntax), but they are useless in the abstract syntax (extra unnecessary classes in modules). One may still think that a typedef as an alias (no extra condition) may be needed to add operations to existing types, as specification suggests doing in order to include new operations for the OCL standard library predefined types. However, this still can be done by means of helpers. Suggestions: - Remove the statements related to aliases in the section 8.2.2.24 - Make Typedef.condition reference be mandatory in Figure 8.7 since, now, Typedef in the abstract syntax will be used to constrain existing types. - Add statements in the concrete syntax section, to clarify the use of aliases to rename existing types. Maybe, a new keyword (like alias) could be included to avoid confusions with the typedef one. - Change the grammar to adapt it to these suggestions. - Clarify in the Standard Library section that helpers will be used to add new operations to the OCL Std Lib predefined type.

Discussion:

It is hard to tackle this properly until the OCL type system is clearer. OCL 2.5 should define what it means to prepare a loaded metamodel for OCL usage. This definition can form the basis of a QVT clarification..

Disposition: **Deferred**

Issue 13180: section (8.3.2) is very confusing for the reader

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

his section (8.3.2) is very confusing for the reader. What does synonym means ? - Is just like a shorthand of the concrete syntax ? - Is just a new operation included in the QVTo Standard Library semantically equivalent?. Ideally, just one type/operation must exist (the OCL predefined type and the operation of an OCL predefined type), so that, writing Void (a type) or asType (operation) should produce the same AST as if I write OclVoid or oclAsType. With this idea, I'm really puzzled with the third paragraph of the section. Please revise this section so that it is less confusing.

Discussion:

Once OCL 2.5 provides an extensible modeled OCL Standard Library, there will be no need for this to be more than additional library definitions.

Disposition: **Deferred**

Issue 13181: ** QVTo Standard Library.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

** QVTo Standard Library. Some operation's returned values would better return the TemplateParameterType **

Several stdlib operations would be better changed to avoid doing unnecessary castings on the returned value of the operation.

For AnyType::oclAsType(oclType) operation I could write:

var aClass : Class := anotherVar.oclAsType(Class);

However, for Model::objectsOfType(OclType) operation I can't do:

var aClassSet : Set(Class) := aModel.objectsOfType(Class);

I have to do the following, instead:

var aClassSet : Set(Class) := aModel.objectsOfType(Class).oclAsType(Set(Class));

Therefore, for end-user usability, I propose exploiting TemplateParameterType and changing some QVTo Standard Library operations

Element::subobjectsOfType(OclType) : Set(T)

Element::allSubobjects(OclType) : Set(T)

Element::subobjectsOfKind(OclType) : Set(T)

Element::allSubobjectsOfKind(OclType) : Set(T)

Element::clone() : T

Element::deepclone() : T

Model::objectsOfType(OclType) : Set(T)

Model::copy() : T

Model::createEmptyModel(): T

Note: this approach is made in the Object::asOrderedTuple() : OrderedTuple(T) operation.

Discussion:

This is the approach taken by the OCL Standard Library modeling in Eclipse OCL using the OclSelf type, which has a well-defined meaning whereas TemplateParameterType is magic.

A proper QVT Library model should use the modeling capabilities to be provided by OCL 2.5.

Disposition: **Deferred**

Issue 13252: QVTo Standard Lybrary and typedefs Issue. Extending OCL predefined types.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

As interpretated from the especification (pag 104), the way of adding new operations to OCL predefined types is creating new Typedef instances

which must have the OCL predefined type as the base type. The new operations are added to this new typedef. However there are several problems:

1. The specification doesn't provide any name for these typedefs.

2. The specification doesn't specify which type (QVT typedef or OCL predefined type) should be used when referencing such OCL predefined types in a QVTo transformation.

Solution for 1).

Suggestion a: Name the typedef with the same name of the base type. This provokes name's clash with the predefined type's name, due to there are two different types from two standard libraries

which have the same name. A possible solution, would be expliciting that typedefs (aliases) will never clash its name with its base type.

Suggestion b: Name the tpyedef with a different name, such as QVToXXXXXX or XXXX_Alias.

Solution for 2).

Suggestion a: Taking the typedef as the referenced type in QVTo transformations.

Suggestion b: Taking the OCL predefined type as the referenced type in QVTo transformations.

Suggestion c: Considering resolution of issue 13168, so that only OCL predefined exists, and therefore, the only type which can be referenced.

It's a little bit weird having 2 different types (a type, and its alias typedef) which represent just the same type, specially when they are related by a reference.

My solution's preference in order are:

Suggestion c: Just having one type to refer.

Suggestion a: Since the typedef "extends" the behaviour of the predefined type (adding new operations), the former must be the referred one.

Suggestion b: The OCL predefined type is referenced, but we must take into account that the operations added to the typedef are also available.

Discussion:

It is hard to tackle this properly until the OCL type system is clearer. OCL 2.5 should define what it means to prepare a loaded metamodel for OCL usage. This definition can form the basis of a QVT clarification..

Disposition: **Deferred**

Issue 14620: QVT 1.1 Inappropriate ListLiteralExp inheritance (Correction to issue 12375 resolution).

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The resolution for Issue 12375 specifies that ListLiteralExp has CollectionLiteralExp as a superclass.

This leaves the behaviour of the inherited kind and part attributes unspecified and rather embarrassing.

ListLiteralExp (like DictLiteralExp) should inherit directly from LiteralExp.

Resolution:

The specification is indeed a mess with duplicate incompatible options for the list elements/parts.

Unfortunately the simple change prevents integer ranges in list literals.

A sensible fix needs a resultion of OCL Issue 13225 to change CollectionLiteralExp::kind to take the metamodel type as its value. QVTo can then specify ListType to a CollectionLiteralExp and eliminate ListLiteralExp altogether.

The non-normative files in QVT 1.1 used LiteralExp as the superclass, so the QVT 1.2 UML files do as well and so the diagram in Issue 12518 shows LiteralExp as the superclass.

A workaround used by Eclipse QVTo is to recognize that there is no content value difference between a Sequence CollectionLiteralExp and a List CollectionLiteralExp and so a Sequence CollectionLiteralExp is used instead,

Disposition: **Deferred**

Issue 14640: QVT 1.1 QVTr syntax mapping (correction to Issue 10646 resolution).

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The numerous problems identified in http://www.omg.org/archives/qvt-rtf/msg00094.html need to be addressed.

Apologies too for the long email. This is a very difficult area where precision is important. Provision of this resolution demonstrates the need for the resolution, but unfortunately the resolution has an erroneous precision that will make QVTr 1.1 unimplementable whereas QVTr 1.0 is just mildly ambiguous and conflicting.

Please do not include the resolution in QVT 1.1 without significantrework.

I found the definition of the "checkonly"/"enforce" to isCheckable/isEnforceable helpful, although different to three of my own intuitive guesses based on attempts to avoid errors in ModelMorf and Rel2Core examples.

The problems identified below are the result of a local review of the resolution. In the absence of a coherent Environment semantics it has not been possible to perform a global review. In particular, I was unable to review the specification for the arguably redundant bindsTo.

[Incidentally, the same resolution approach is needed for QVTc and QVTo].

Disambiguating rules

The resolution uses a similar approach to the OCL 2.0 specification, but neglects to provide any disambiguating rules although many are needed.

Environments

OCL and the resolution employ environments to carry definitions specific to particular CS constructs, so that a CS reference may be resolved with the aid of an environment or the AST.

In EssentialOCL, all definitions are resolvable with in the immutable AST with the exception of let and iterator expressions for which a nestedEnvironment() is used to carry the extra variable declaration with the aid of addElement(). The nestedEnvironment supports name occlusion from outer scopes.

In CompleteOCL, further definitions may be introduced by a definition constraint. The OCL specification provides no precise insight into how an environment changes to accommodate the definition. Should the name be added to the AST or to the environment? Is the name available for forward reference?

Environment::addElement is defined as a query operation thereby inhibiting side effects. Consequently usage such as

XX.env = YY.env.addElement(Z.name, Z.ast, false)

leaves the environment of YY unchanged and creates an extended environment for XX.

A series of such usages creates a series of progressively elaborated environments that support backward but not forward referencing. This was not a clear requirement of the QVT 1.0 specification and it prevents any variable declarations being introduced in an object template tree being resolved through environments in other object template trees or predicate expressions.

Imposition of no forward referencing seems very undesirable, particularly since the order of domains is imposed by the model parameter order. Imagine a Java program in which all methods had to be defined in a no-forward reference order.

As noted above, CompleteOCL neglected to define how AST definitions were added to the AST. QVTr must solve this problem since QVTr defines a hierarchically scoped AST rather than an annotation of a pre-existing AST.

I recommend a two stage approach. The inherited attributes section should first compute an environment from the pushed-down parent environment augmented by pull-ups from child constructs so that a complete immutable and consequently unambiguous environment is associated with each construct and then pushed-down to the children. During the pull-up the environment acquires a mapping from name to future-AST. During the push-down residual future-AST attributes are populated to give a valid AST.

Reference resolution

OCL uses lookup functions to resolve variable references. It is necessary either to overload the lookup functions so that the the distinct QVTr variable definition sites can be located in the AST, or to use some form of Environment::addElement where each definition is defined so that resolution in the environment is possible.

Details

=======

Throughout, many disambiguating rules are needed to define illegal semantics. For instance "any" is often used to select a syntactically valid value. Corresponding usage in the OCL specification has a disambiguating rule to clarify what the consequence of not "one" is.

My current best attempt at Disambiguating Rules is attached.

Environment::addElement takes three arguments, the third being a mayBeImplicit argument. This has been omitted throughout without explanation.

identifierCS

OCL defines SimpleNameCS. A degenerate mapping from identifierCS to SimplenameCS is required.

topLevelCS

The 'imported transformation' environment element is later referenced as 'imported transformations'.

Typo: TransformationListCS for transformationListCS in Synthesized attributes.

importListCS

Semantics of import conflicts must be defined.

unitCS

Typo: ast is not a Set.

Surely the import is of packages (enumerations or operations)or at least transformations (QVTo implementations) rather than necessarily relational-transformations?

transformationCS

---

ownedTag is not synthesized.

keyDeclListCS

Typo: wrong font in synthesized attributes

modelDeclCS

The [B] grammar:

modelDeclCS ::= modelIdCS ':' '{' metaModelIdListCS '}'

is missing.

keyDeclCS

Synthesized attributes appear to have experienced a copy and paste error while providing distinct part and oppositePart left hand sides.

keyPropertyCS

The synthesized attributes poke the parent.

Suggest: it would be clearer for the parent to gather and distribute children similar to the relation/query allocation by transformationCS.

relationCS

Transformation.extends does not appear to be transitive.

topQualifierCS

Suggest: a boolean or enumerated value rather than a string.

domainListCS

Typo: missing indentation.

primitiveTypeDomainCS

---

isCheckable, isEnforceable not synthesized.

objectTemplateCS

Variable needs to be added to relation to provide a container.

Variable needs to be added to relation environment to provide visibility.

collectionTemplateCS

Variable needs to be added to relation to provide a container.

Variable needs to be added to relation environment to provide visibility.

Suggest: last two if guards are redundant.

restCS

Variable needs to be added to relation to provide a container.

Non-_ named variable needs to be added to relation environment to provide visibility.

memberCS

Variable needs to be added to relation to provide a container.

Non-_ named variable needs to be added to relation environment to provide visibility.

whenCS

---

predicateListCS should be optional.

whereCS

---

predicateListCS should be optional.

ExtOclExpressionCS

This is not present in the QVTr or OCL grammar.

Presumably it represents the QVTr extension to OCL's OclExpressionCS.

However it is an extension, since at least RelationCallExpCS can be used in an ordinary OclExpressionCS using "not" or "and".

[A], [B], [C] should therefore follow on from OCL's

[A], [B], [C]..., [I].

RelationCallExpressionCS

How is a RelationCallExpressionCS distinguished from an OperationCallExpCS?

Discussion:

It is hard to tackle this properly until we have the model-driven specification generated technology that OCL 2.5 will pioneer.

Disposition: **Deferred**

Issue 15376: QVT 1.1 8.1.10 Errors in Examples

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The second example contains

"if result then return;"

which has a non-boolean condition expression and a missing endif.

In the first example, it is not clear that the revisit adds rather than overwrites.

In the third and fourth examples it is not clear why the second pass reuses the context for the first rather than creates new objects.

Discussion:

Help wanted.

Disposition: **Deferred**

Issue 15390: QVT 1.1 8 Unclear mapping operation characteristics

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

8.1.4 states "a mapping operation is always a refinement of an implicit relation" but 8.2.15 defines "refinedRelation: Relation [0..1] The refined relation, if any." Clearly a contradiction.

8.1.4. provides the REL_PackageToSchema example of how an implicit relation might be defined, but no other examples or synthesis rules are supplied.

enforce and checkonly appear in the REL_PackageToSchema example indicating that these define execution mode of a QVTo trnasformation, but there does not appear to be any description of how a transformation might be executed to for instance update an output model. Perhaps the 'output' parameter is 'inout' for create/update but 'out' for create/overwrite.

Discussion:

Help wanted.

Disposition: **Deferred**

Issue 15411: Unclear transformation rooting condition

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

"The starting point is a major flaw in all three QVT languages at present and enabled Perdita Stevens to correctly conclude in http://www.springerlink.com/content/9x368617317l3q87/ that QVTr and QVTc are incompatible. QVT currently provides no way to distinguish whether for instance a check-mode transformation is a query of whether a transformed input pattern can be discovered in the output (e.g. a database lookup), or a validation that the transformed input exactly matches the output (e.g. an already transformed check). Both facilities are useful and so when a QVT transformation is invoked the invoker needs to specify what I call the 'rooting' condition in addition to the direction

Discussion:

This requires some research work.

Disposition: **Deferred**

Issue 15417: Rule Overriding in QVTr.

Source:

Dr. Maged Elaasar, melaasar(at)gmail.com

Summary:

The abstract syntax of QVTr allows a rule to be an override of another rule.

Rule::overrides: Rule [0..1]

The rule that this rule overrides.

The concrete syntax of QVT allows it too:

<relation> ::= ['top'] 'relation' <identifier>
['overrides' <identifier>]
'{'
....
'}'

However, the only semantics I can see for 'overrides' is in clause 7.11.1.4 that says:

"A rule may conditionally override another rule. The overriding rule is executed in place of the overridden rule when the overriding conditions are satisfied. The exact semantics of overriding are subclass specific. "

Questions:

1- Whtat are the overriding conditions? are they implied or specified and if so how?

2- I have not seen any other discussion of overrding in a subclass or Rule so not sure what is meant by "The exact semantics of overriding are subclass specific"?

3- I have not seen any example of using 'overrides' what so ever in the spec, shouldn't there be one?

4 - What is the semantics of overriding? is it related to inheritance in the OO sense ? I think QVTr needs a good "inheritance" model where you can relations can be called polymorphically.

Discussion:

[Comments from inadvertently raised Issue 15524]

You have reached the edge of the specification as written.

1: Yes
2: Yes
3: Yes
4: Yes

I gave some consideration to this for UMLX.

I felt that an abstract 'rule' could define a 'subrule' obligation, which would require an identical match signature, since if the override was narrower it would not fulfill the obligation and if it was wider the additional width would not be permitted by the invocation of the abstract 'rule'.

I felt that all concrete rules should always be matched to ensure that addition of extended functionality did not change previous behaviour. This complies with UMLX's all maximal matches philosophy. Keys in QVTr, Evolution Ids in UMLX can ensure that derived rules reuse inherited matches.

I think a transformation being both a package and a class introduces some difficult compatibility issues to be studied.

Transformation extension is also poorly defined giving additional imprecision when considering the combination of transformation extension and rule override.

My ideas for UMLX were not complete but I think that they may be sounder than QVTr's.

[If Transformation is just a Class, one area for study vanishes.]

Disposition: **Deferred**

Issue 15523: QVTr already has queries but they are much less user friendly than e.g. MOFM2T's equivelent.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

QVTr already has queries but they are much less user friendly than e.g. MOFM2T's equivelent for which the first parameter is a hidden self, or indeed QVTo.

Perhaps something closer to Complete OCL would do, allowing def'd attributes or operations.

Discussion:

Perhaps it is sufficient to allow a QVTr query to be invoked with its first argument as a syntactical source rather than argument.

Disposition: **Deferred**

Issue 15886: Specification of deletion semantics.

Source:

Institute for Defense Analyses (Dr. Steven Wartik, swartik(at)ida.org)

Summary:

I'm having trouble with the semantics of DELETE on p. 189 of the QVT Specification (v1.1). It reads in part:

FORALL OBJVAR IN MAKESET(<DOMAIN_K_VARIABLE_SET>) (

…

AND BELONGSTO(OBJVAR, MAKESET(<DOMAIN_K_VARIABLE_SET>))

I guess I don't understand MAKESET and BELONGSTO. First of all, <DOMAIN_K_VARIABLE_SET> is already a set, so what's the MAKESET function do? Second, the FORALL iterates OBJVAR over the results of the same MAKESET that BELONGSTO tests. So how can BELONGSTO be false? That is, I would assume BELONGSTO is defined as follows:

BELONGSTO(e, S) ° e ÎS

except that under this definition the expression above is always satisfied.

Any and all help appreciated. Thank you very much.

Discussion:

I see no justification for not using OCL to express the Annex B semantic. The OCL could then form part of real tests.

I see little prospect of progress here till we have a conformant working QVTr implementation.

Disposition: **Deferred**

Issue 18323: Trace data for an 'accessed' transformation

Source:

Christopher Gerking, cgerking(at)campus.upb.de

Summary:

The spec should clarify the interaction of

1.) explicit instantiation + execution of 'accessed' transformations, and

2.) trace records / resolving.

The following questions are of interest:

How does the initial trace for an 'accessed' transformation look like? Does it reuse the records previously created by the 'accessing' transformation, or does an 'accessed' transformation always start on an empty trace?

How does an 'accessed' transformation affect the trace? Are the trace records post-visible that were created during execution of the 'accessed' transformation, or is the trace of the 'accessing' transformation unchanged?

Both issues heavily affect the results of resolve-operations. Resolving object references after executing an 'accessed' transformation would be very practical.

Discussion:

Help wanted.

Disposition: **Deferred**

Issue 18324: No trace data for disjuncting mapping

Source:

Christopher Gerking, cgerking(at)campus.upb.de

Summary:

Trace data creation is specified to happen "at the end of the initialization section".

For disjuncting mappings, the initialization section is never executed, which prevents any trace data from being stored.

As a consequence, no resolution via resolve-in-expressions is possible on the disjuncting mapping, due to the missing trace record. This is problematic, since disjunction should be transparent from a resolver's point of view, i.e. it should not make a difference for resolution whether a mapping disjuncts or not.

Hence, some clarification is required whether trace records are deliberately avoided for disjuncting mappings (for whatever reason), or whether trace data must be created in another place than the init section in case of a disjuncting mapping.

Discussion:

Help wanted.

Disposition: **Deferred**

Issue 18363: Undefined semantics for unsatisfied "when" and "where" in inherited mapping

Source:

Levy Siqueira, levy.siqueira(at)gmail.com

Summary:

In operational QVT, it is not clear what happens when the "when" or "where" clause of an inherited mapping does not hold.

Suggestion:

Considering inheritance is a type (or, maybe, a synonym) of generalization, it would be expected that the semantics of inheritance mimics the semantics of generalization in MOF. The UML, which defines generalization used by CMOF and EMOF, states: "... features specified for instances of the general classifier are implicitly specified for instances of the specific classifier. Any constraint applying to instances of the general classifier also applies to instances of the specific classifier." (UML Infrastructure 2.4.1, p.51). If the "when" and "where" clauses are considered as features of a mapping, the clauses of the inherited mapping should be implicitly specified. Similarly, if they are considered as constraints applying to a mapping, the clauses defined in the inherited mapping should apply to the inheriting mapping. Therefore, a possible solution in both situations is to consider that the "when" and "where" clauses must hold in the inheriting mapping.

Commentary:

An interesting discussion is if something similar to the Liskov substitution principle should be applied in this situation.

Discussion:

Yes. This is a very fundamental principle that should form part of the philosophical underpining in the first couple of paragraphs of the QVTo language exposition. (ditto QVTc, QVTr).

It is disgraceful that the specification of transformation extension and rule refinement is so poor.

For QVTc and QVTr I favour a principle that extension provides extension not replacement. But that may be too restricting.

QVTo is more permissive and practical so a pure principle may be inappropriate.

Research needed.

Disposition: **Deferred**

Issue 18912: Inconsistent multiple inheritance.

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

The pragmatic decision to define Module/Transformation as inheriting both Package and Class violates UML inheritance.

For a Package: self.nestingPackage.nestedPackages->includes(self)

For a Class:

self.package.ownedTypes->includes(self)

But self cannot have two containers.

The problem is easily resolved by extending only Package and adding those Class features that are actually required.

Discussion:

A quick experiment suggests that Class features are important but Package features are incidental, so making Transformation extend just Class and requiring a containing Package to provide context could be a good solution. Further investigation is required.

(Similar problem for FunctionParameter).

Disposition: **Deferred**

Issue 19019: List and Dict are Classes rather than DataTypes

Source:

Nomos Software (Dr. Edward Willink, ed(at)willink.me.uk)

Summary:

Intuitively List and Dict are objects, so if you pass them to a mapping/query/helper as an inout parameter, the object in the caller may be updated by the call.

This is the behaviour of a Class Instance not a DataType Value.

Please use the open https://bugs.eclipse.org/bugs/show_bug.cgi?id=420150 to discuss this topic.

Discussion:

It would be nice to clean this up, but very dangerous to do so without evaluating the consequences for a real implementation and typical transformations.

Disposition: **Deferred**

Issue 19023: Enhance ObjectExp to allow constructors invocation.

Source:

Open Canarias, SL (Mr. Adolfo Sanchez-Barbudo Herrera, adolfosbh(at)opencanarias.com)

Summary:

Problem:

ObjectExp seems to be an enhancement to the InstantationExp due to the additional semantics, however it gets limited due to the fact that every time we need to use it we have to define the constructor body (in constrast to the instantiation expression usage). Although, ObjectExp was conceived to inline object instantiations, this limitation is not justified.

However, this limitation could be removed by also allowing an ObjectExp-Constructor combination, in the same way we have for InstantiationExp. The problem relies on a flaky concrete syntax which we could enhance to exploit an ObjectExp with an already defined constructor operation:

constructor Column::ColumnConstructor (n:String,t: String) { name:=n; type:=t; }

object result1 : Column (“name”, “String”);

object result2 : Column (“age”, “Integer”);

Providing a constructor body (from ObjectExp) and a list of arguments (from InstantiationExp) should be prohibited in both, the concrete syntax and the abstract syntax. Regarding the abstract syntax this could be expression with a constraint.

context ObjectExp

inv : argument->size() > 0 implies body.oclIsUndefined()

Discussion:

This enhancement seems convenient with no apparent drawbacks, since the old ObjectExp usage remains valid.

Proposed solution:

In section 8.2.1.24 add the following subsection:

Constraints

If an object expression contains a constructor body, no arguments for a constructor are allowed (and vice versa):

context ObjectExp

inv: argument->notEmpty() > 0 implies body.oclIsUndefined() and

not body.oclIsUndefined() implies argument->isEmpty()

In section 8.2.1.24 add the following to the the end notation subsection:

Similarly to InstantiationExp, an object expression could be used to invoke a constructor operation, rather than inlining a constructor body:

object result1 : Column (“name”, “String”);

object result2 : Column (“age”, “Integer”);

Note that this notation allows us to use object expression to instantiate (or update) objects, while having a reusable constructor in order to initialize (or update) the properties of the object subject to be created (or updated).

In section 8.4.7:

Replace:

<object_exp> ::= 'object' ('(' <iter_declarator> ')')? <object_declarator>

<expression_block>

By:

<object_exp> ::= 'object' ('(' <iter_declarator> ')')? <object_declarator>

(<expression_block> | '(' (<declarator_list>)? ')')

Resolution:

InstantiationExp.initailzationOperation may provide the required AS support. CS needs protyping to determine what dosambiguation rules are required.

Disposition: **Deferred**