QVT 1.2 Revision Task Force

Ballot 1

"Preview 1"

1 January 2014

QVT 1.2 RTF Formal Issues Resolution Vote - Ballot No. 1

Poll start date: Wednesday, 8 January 2014 (01:00 AM EDT - 06:00 GMT)

Poll closing date: Wednesday , 22 January 2014 (07:00 PM EDT - 24:00 GMT)

What is being voted on: Proposed issue resolutions for the set of issues listed in the tables on the following pages. The proposer is listed for each resolution, the full text of the issues and corresponding resolutions can be found in the section following the issue tables.

- Only officially registered RTF members in good standing are allowed to vote
- Voters who do not vote in two successive ballots lose their good standing and will be removed from the RTF membership; they can only be reinstated by a TC vote
- Quorum for a vote is half the registered RTF members
- Simple majority (of non-abstaining votes) decides the vote
- Votes should be sent via email to the chair of the RTF(ed@willink.me.uk) as well as to the qvt-rtf@omg.org list.
- Members can submit their vote anytime between the poll start date and the poll closing date.
- During the polling interval, members are allowed to change their votes. The most recent vote will be assumed to supersede all previous votes cast by a member.
- The possible ways to vote are:
  - Yes
  - No
  - Abstain (Note: “Abstain” does not influence the voting result but *does* count towards quorum)
- Votes can be cast either for individual issues or for the entire block (but not a mix of both)

Block Vote

<Company name> votes {Yes | No | Abstain} for the entire block of proposed issue resolutions identified below and specified in the appendix to this document.

Individual Issues Vote (NB: ONLY if you choose not to use the Block Vote option above!) <Company name> votes as follows on the proposed issue resolutions specified in and specified in the appendix to this document. Then vote for one issue resolution per line, like this:

12345 {Yes | No | Abstain}

No Vote

A brief explanation for each No vote should be provided.

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: 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 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 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 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**

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 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".

Resolution:

An OCL tuple is a reasonable return and likely to be the resolution for a similar problem when invoking multi-out UML operations from OCL. Why should the result be ordered?

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**

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: 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**