1. ATL Transformation Example

1.1. Example: Software Quality Control → Mantis Bug Tracker file

The "Software Quality Control to Mantis Bug Tracker file" example describes a transformation from a SoftwareQualityControl model to a simple Mantis XML file. Mantis Bug Tracker [1] is a free web-based bug-tracking system written in PHP that uses a MySQL database. The transformation is based on a Software Quality Control metamodel which describes a simple structure to manage software quality controls (and more especially bug-tracking). The input of the transformation is a model which conforms to the SoftwareQualityControl metamodel. The output is an XML file whose content conforms to a Mantis XML schema.

1.1.1. Transformation overview

The aim of this transformation is to generate a valid and well-formed XML file for Mantis Bug Tracker from a SoftwareQualityControl model. Figure 1 gives an example of a simple Microsoft Office Excel workbook whose content is a particular representation for "bug-tracing" or "bug-tracking" (which is the type of software quality control that interests us for our example). The bugs' information contained in the single worksheet of this workbook has been previously injected into a SoftwareQualityControl model thanks to the "MicrosoftOfficeExcel2SoftwareQualityControl" transformation (see [2]).

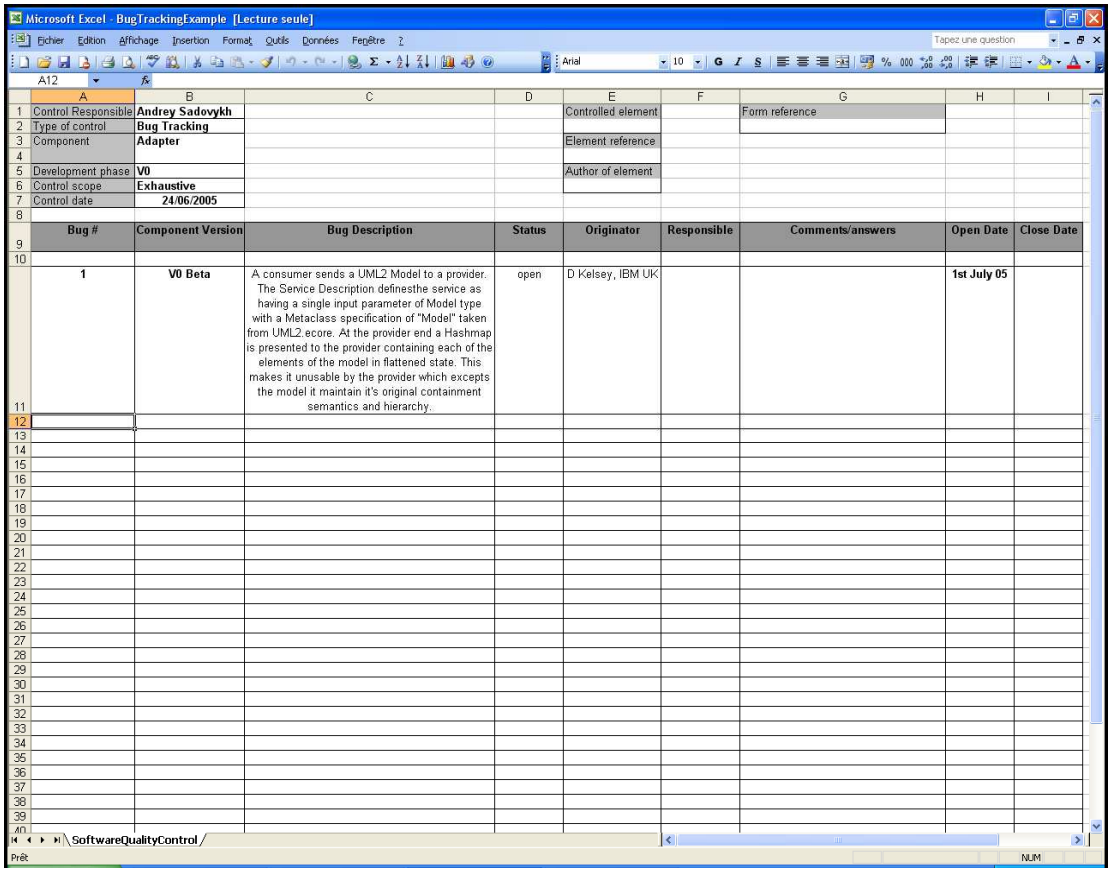

Figure 1. An example of a simple Excel "bug-tracking" representation.

To make the “SoftwareQualityControl to Mantis Bug Tracker file” global transformation we proceed in three steps. Indeed, this transformation is in reality a composition of three transformations:

- from SoftwareQualityControl to Mantis
- from Mantis to XML
- from XML to Mantis XML file (i.e. XML to Mantis text)

These three steps are summarized in Figure 2.

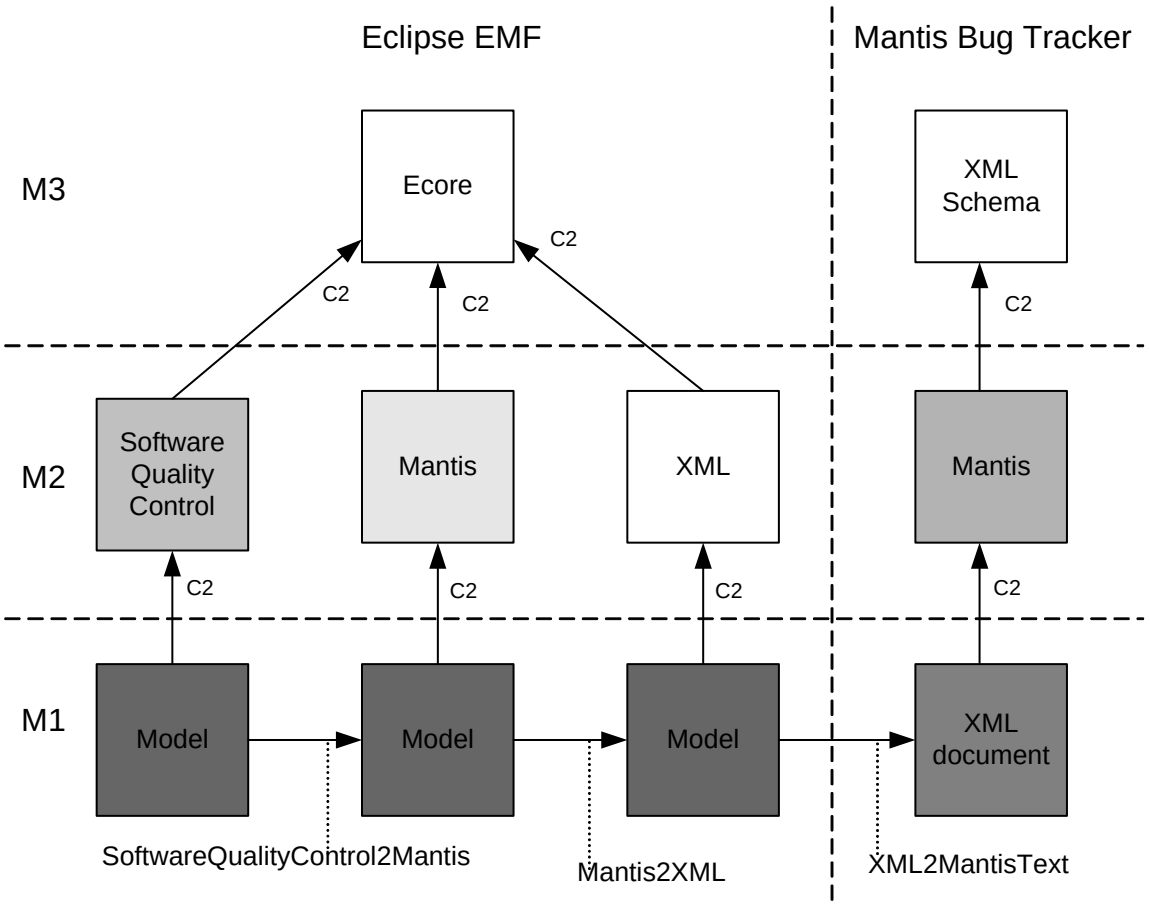

Figure 2. “Software Quality Control to Mantis Bug Tracker file” transformation’s overview

1.2. Metamodels

The transformation is based on the "SoftwareQualityControl" metamodel which describes a simple structure to manage software quality controls and more especially bug tracking. The metamodel considered here is described in Figure 3 and provided in Appendix I in km3 format. Note that we present in this documentation the current version of this metamodel that has been created for our particular example: it could be improved in order to allow handling several other types of quality control.

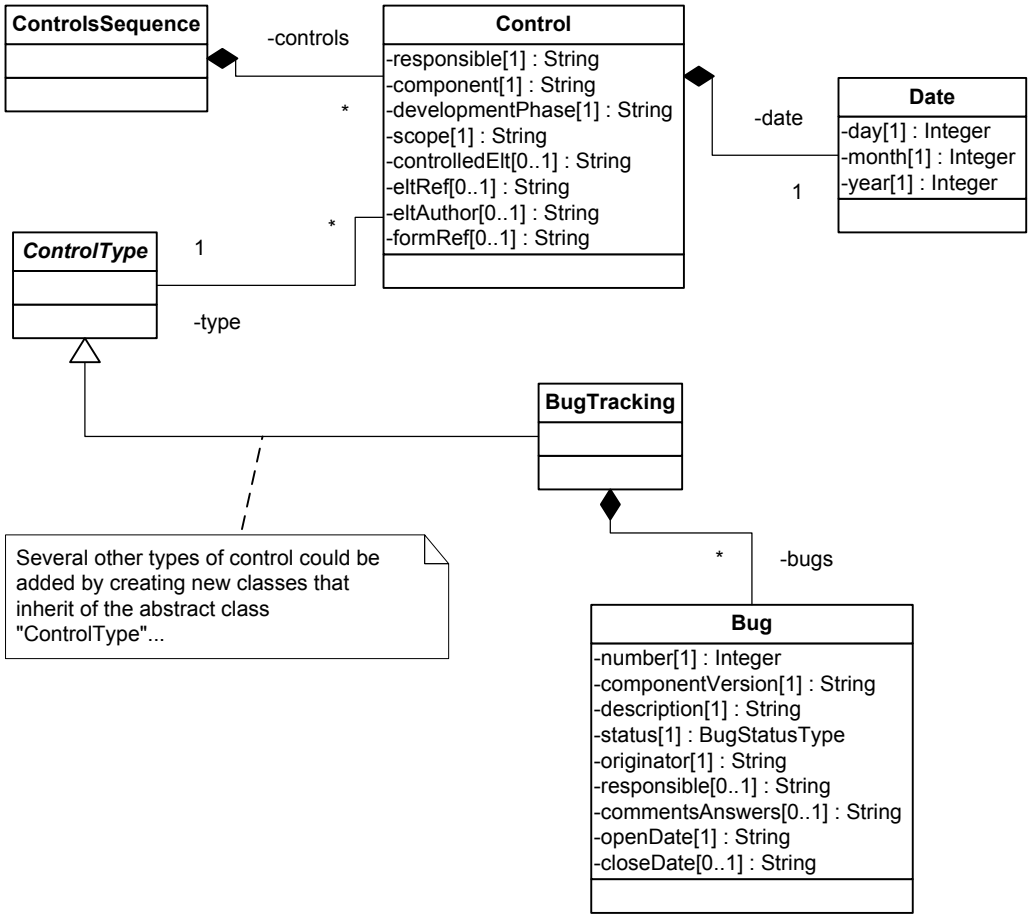

Figure 3. The SoftwareQualityControl metamodel

A "SoftwareQualityControl" model is composed of several *Control* elements. Each *Control* is defined by specific information about the component and the element which are concerned, about the person who is responsible for the control, the date, etc. The main information is the type of the control. It determines what kind of actions has been performed and consequently what kind of data has been saved. In the case of our example, we only create *BugTracking* type but it could have a lot of other control types. In this type, the control consists of a set of *Bug* elements in which each *Bug* is identified by a unique number. A *Bug* is characterized by several specific fields such as its description, its status…

The transformation is also based on the “Mantis” metamodel. A huge database allows Mantis to store a big amount of information about a lot of bugs. These data are too complex to be easily handled by SQL requests. However, Mantis allows importing/exporting bug data from/into XML files. The data in XML files conforms to an XML schema [3].

The Mantis metamodel considered here is directly inspired by this XML schema. It is described in Figure 4 and provided in Appendix II in km3 format.

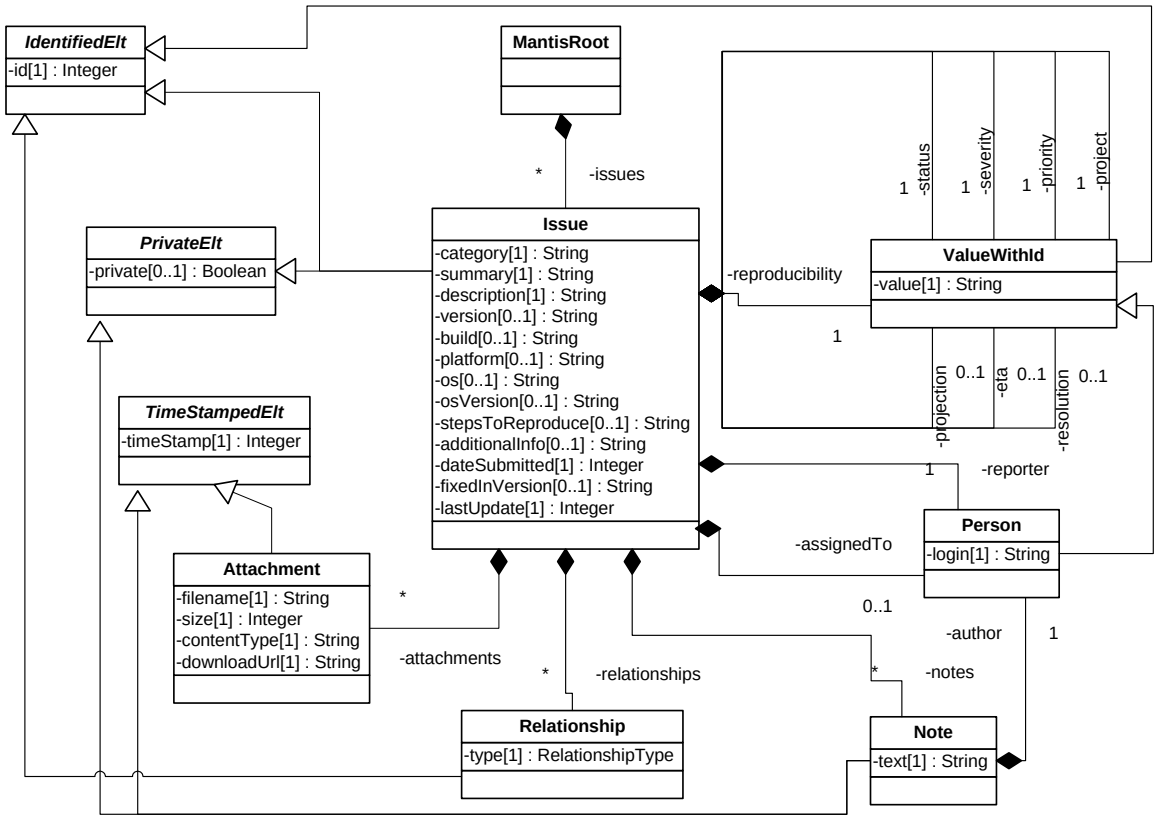

Figure 4. The Mantis metamodel

A bug in Mantis is named an *Issue*. Consequently, a Mantis model is a set of *Issue* elements. Each *Issue* is identified by a unique number. An *Issue* contains much information about itself, its software product, etc.

The last metamodel used by this transformation is a simple XML metamodel which is necessary to export models into XML files. This metamodel is presented in Figure 5 and provided in Appendix III in km3 format.

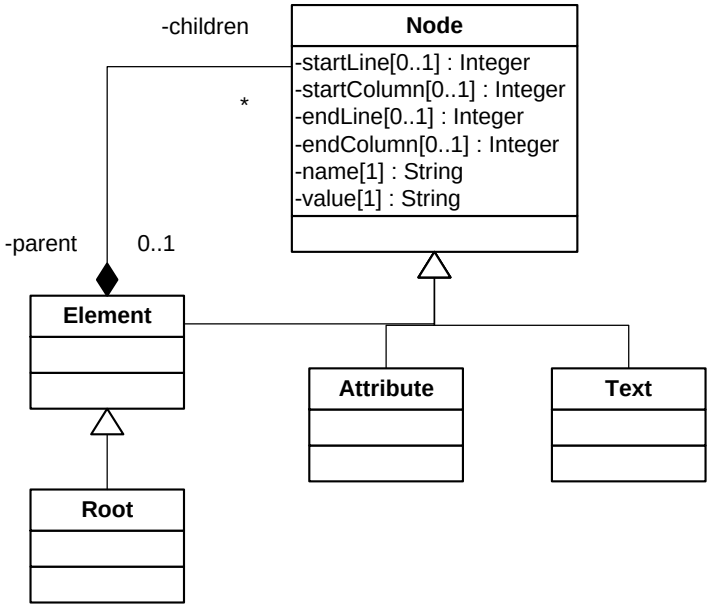

Figure 5. A simple XML metamodel

Each element of an XML document is a *Node*. The root of a document is a *Root* element which is an *Element* in our metamodel. Each *Element* can have several children (nodes) that can be other *Element*, *Attribute* or *Text* elements. An *Element* is usually identified by its name and defined by its children. An *Attribut*e is characterized by its name and its value whereas a *Text* is only assimilated to a single value.

1.3. Rules Specification

The input of the global transformation is a model which conforms to the SoftwareQualityControl metamodel (described in Figure 3); the output is a Mantis XML file whose content conforms to the Mantis XML schema [3]. The input Mantis model of the second transformation is the output Mantis model generated by the first transformation. The input XML model of the third transformation is the output XML model engendered by the second transformation.

1.3.1. SoftwareQualityControl to Mantis

These are the rules to transform a SoftwareQualityControl model into a Mantis model:

- For each *SoftwareQualityControl!BugTracking* element, a *Mantis!MantisRoot* element is created. It will be linked to the corresponding *Mantis!Issue* elements that will be generated during the transformation by the following rule.
- For each *SoftwareQualityControl!Bug* element, a *Mantis!Issue* element is engendered. The attributes and the sub-elements of this generated element are correctly initialized in this rule.

1.3.2. Mantis to XML

These are the rules to transform a Mantis model into an XML model:

- For the root *Mantis!MantisRoot* element, the “mantis” *XML!Root* element is created. The required *XML!Attribute* elements are also generated and added as children of this “mantis” *XML!Root* element.
- For each *Mantis!Issue* element, an “issue” *XML!Element* element is engendered and set as a child of the “mantis” *XML!Root* element. All the necessary *XML!Attribute*, *XML!Element* and *XML!Text* elements are also created and added as children of this “issue” *XML!Element* element.
- For each *Mantis!Attachment* element, an “attachment” *XML!Element* element and its children’s *XML!Element* and *XML!Text* elements are generated.
- For each *Mantis!Relationship* element, a “relationship” *XML!Element* element and its children’s *XML!Element* and *XML!Text* elements are generated.
- For each *Mantis!Note* element, a “note” *XML!Element* element and its children’s *XML!Element* and *XML!Text* elements are generated.

1.3.3. XML to Mantis XML file (i.e. XML to Mantis text)

There are no rules defined for this step but only an ATL query (and the associated ATL helpers) that allows generating a valid and well-formed Mantis XML text file from an XML model. The aim of this query is to extract each of the elements that compose the input XML model into an output XML file. Look at the “ATL Code” following section to get more details about this ATL query.

1.4. ATL Code

There is one ATL file coding a transformation for each of the three steps previously detailed. In this part we will present and describe more precisely the ATL code associated to each implemented transformation.

1.4.1. SoftwareQualityControl2Mantis

The ATL code for the “SoftwareQualityControl2Mantis” transformation consists of 2 helpers and 2 rules.

The *convertStatus* helper returns the string value corresponding to the SoftwareQualityControl!BugStatusType value passed in argument.

The *getResponsibleName* helper returns the string value corresponding to the name of the person who is responsible of the context’s SoftwareQualityControl!Bug element. Note that if the “responsible” attribute is not valued, the name of the control’s responsible is returned (i.e. the “responsible” attribute’s value of the SoftwareQualityControl!Control element associated to the context’s SoftwareQualityControl!Bug element in the input model).

The rule *BugTracking2MantisRoot* allocates a Mantis!MantisRoot element only if a BugTracking element is encountered in the input SoftwareQualityControl model. This generated Mantis!MantisRoot element will be linked, thanks to a “resolveTemp(…)” method’s call, to the corresponding Mantis!Issue elements that will be created by the following rule during the transformation.

The rule *Bug2Issue* allocates a Mantis!Issue element and all the required Mantis!ValueWithId, Mantis!Note and Mantis!Person elements for each SoftwareQualityControl!Bug element of the input model. The attributes of the generated elements are simply valued in the rule, if necessary, by traversing the input model and by thus recovering the sought values.

```
1  module SoftwareQualityControl2Mantis; -- Module Template
2  create OUT : Mantis from IN : SoftwareQualityControl;
3
4
5  -- This helper permits to convert the status value of a bug in string
6  -- CONTEXT: n/a
7  -- RETURN: String
8  helper def: convertStatus(bs : SoftwareQualityControl!BugStatusType) :  String =
9     let sv : String = bs.toString()
10    in
11       sv.substring(5,sv.size());
12
13 -- This helper permits to get the name of the person who is responsible for the
14 bug.
15 -- If the "responsible" field is not valued, the responsible of this bug is the
16 -- control responsible.
17 -- CONTEXT: n/a
18 -- RETURN: String
19 helper context SoftwareQualityControl!Bug def: getResponsibleName() :  String =
20    let rv : String = self.responsible
21    in
22       if rv.oclIsUndefined()
23       then
24          self.b_bugTracking.ct_control.responsible
25       else
26          rv
27       endif;
28
```

```
29
30
31    -- Rule 'BugTracking2MantisRoot'
32    -- This rule generates the root of the Mantis output model
33    -- if a BugTracking element exists in the input model
34    rule  BugTracking2MantisRoot {
35       from
36          bt : SoftwareQualityControl!BugTracking
37
38       to
39          mr : Mantis!MantisRoot (
40             issues <- bt.bugs->collect(e | thisModule.resolveTemp(e, 'mi'))
41          )
42    }
43
44
45    -- Rule 'Bug2Issue'
46    -- This rule generates a issue in Mantis for each
47    -- bug reported in the BugTracking element.
48    rule  Bug2Issue {
49       from
50          bbt : SoftwareQualityControl!Bug
51       using {
52          commentsAnswersOrNot : Sequence(String) =
53             let ca : String = bbt.commentsAnswers
54             in
55                if ca.oclIsUndefined()
56                then
57                   Sequence{}
58                else
59                   Sequence{ca}
60                endif;
61       }
62       to
63          mi : Mantis!Issue (
64             id <- bbt.number,
65             project <- proj,
66             category <- '',
67             priority <- prior,
68             severity <- sev,
69             status <- stat,
70             reporter <- rep,
71             summary <- '',
72             description <- bbt.description,
73             version <- bbt.componentVersion,
74    --       build <-,
75    --       platform <-,
76    --       os <-,
77    --       osVersion <-
78             reproducibility <- reprod,
79    --       stepsToReproduce <-,
80    --       additionalInfo <-,
81             dateSubmitted <- 0, -- the date is an integer value in a specific format :
82    how to convert?
83             assignedTo <- at,
84    --       projection <-,
85    --       eta <-,
86    --       resolution <-,
87    --       fixedInVersion <-,
88             attachments <- Sequence{},
89             relationships <- Sequence{},
90             notes <- Sequence{note},
```

```
91            lastUpdate <- 0 -- this date is not mentionned in any field in the Software
92    Quality Control metamodel
93          ),
94          proj : Mantis!ValueWithId (
95             id <- 0,
96             value <- bbt.b_bugTracking.ct_control.component
97          ),
98          prior : Mantis!ValueWithId (
99             id <- 0,
100            value <- ''
101         ),
102         sev : Mantis!ValueWithId (
103            id <- 0,
104            value <- ''
105         ),
106         stat : Mantis!ValueWithId (
107            id <- 0,
108            value <- thisModule.convertStatus(bbt.status)
109         ),
110         rep : Mantis!Person (
111            id <- 0,
112            value <- bbt.originator,
113            login <-''
114         ),
115         reprod : Mantis!ValueWithId (
116            id <- 0,
117            value <- ''
118         ),
119         at : Mantis!Person (
120            id <- 0,
121            value <- bbt.getResponsibleName(),
122            login <-''
123         ),
124         note : distinct Mantis!Note foreach(commentsAnswersVal in
125   commentsAnswersOrNot)(
126            timestamp <- 0,
127            author <- aut,
128            text <- commentsAnswersVal
129         ),
130         aut : distinct Mantis!Person foreach(commentsAnswersVal in
131   commentsAnswersOrNot)(
132            id <- 0,
133            value <- bbt.originator,
134            login <-''
135         )
136   }
```

1.4.2. Mantis2XML

The ATL code for the “Mantis2XML” transformation consists of 1 helper and 5 rules.

The *getRelationshipTypeStringValue* helper returns the string value corresponding to the Mantis!RelationshipType passed in argument.

Each implemented rule follows the same principle: an XML!Element (with some associated other XML!Element, XML!Attribute or XML!Text elements) is allocated for each element of the Mantis input model. These generated XML elements are correctly linked from the ones to the others (thanks to “resolveTemp(…)” method’s calls) in order to construct an XML model whose content conforms to the Mantis XML schema [3].

As an example, the *MantisRoot2Root* rule allocates a “mantis” XML!Element and three XML!Attribute elements, which are children of the XML!Element, for each MantisRoot element of the input Mantis model. This “mantis” XML!Element will be linked, thanks to a “resolveTemp(…)” method’s call, to the “issue” XML!Element elements that will be created to represent issues (bugs) by the *Issue2Issue* rule…

```
1   module Mantis2XML; -- Module Template
2   create OUT : XML from IN : Mantis;
3
4
5   -- This helper permits to obtain the string associated
6   -- to an RelationshipType value.
7   -- CONTEXT: n/a
8   -- RETURN: String
9   helper def: getRelationshipTypeStringValue(rt : Mantis!RelationshipType) : String =
10     let rv : String = rt.toString()
11     in
12        rv.substring(4,rv.size());
13
14
15
16  -- Rule 'MantisRoot2Root'
17  -- This rule generates the root of the XML model
18  -- from the "MantisRoot" element
19  rule  MantisRoot2Root {
20     from
21        mr : Mantis!MantisRoot
22
23     to
24        xr : XML!Root (
25           name <- 'mantis',
26           children <- Sequence{att1,att2,att3,
27                        mr.issues->collect(e | thisModule.resolveTemp(e, 'xi'))
28                        }
29        ),
30        att1 : XML!Attribute (
31           name <- 'xmlns',
32           value <- 'http://www.mantisbt.org'
33        ),
34        att2 : XML!Attribute (
35           name <- 'xmlns:xsi',
36           value <- 'http://www.w3.org/2001/XMLSchema-instance'
37        ),
38        att3 : XML!Attribute (
39           name <- 'xsi:schemaLocation',
```

```
40            value <- 'http://www.mantisbt.org mantis.xsd'
41         )
42   }
43
44
45   -- Rule 'Issue2Issue'
46   -- This rule generates the XML issue's tags
47   -- from the "Issue" element
48   rule  Issue2Issue {
49      from
50         mi : Mantis!Issue
51      using {
52         privateOrNot : Sequence(String) =
53            let priv : Boolean = mi.private
54            in
55               if priv.oclIsUndefined()
56               then
57                  Sequence{}
58               else
59                  Sequence{priv.toString()}
60               endif;
61         versionOrNot : Sequence(String) =
62            let vv : String = mi.version
63            in
64               if vv.oclIsUndefined()
65               then
66                  Sequence{}
67               else
68                  Sequence{vv}
69               endif;
70         buildOrNot : Sequence(String) =
71            let bv : String = mi.build
72            in
73               if bv.oclIsUndefined()
74               then
75                  Sequence{}
76               else
77                  Sequence{bv}
78               endif;
79         platformOrNot : Sequence(String) =
80            let pv : String = mi.platform
81            in
82               if pv.oclIsUndefined()
83               then
84                  Sequence{}
85               else
86                  Sequence{pv}
87               endif;
88         osOrNot : Sequence(String) =
89            let ov : String = mi.os
90            in
91               if ov.oclIsUndefined()
92               then
93                  Sequence{}
94               else
95                  Sequence{ov}
96               endif;
97         osVersionOrNot : Sequence(String) =
98            let ovv : String = mi.osVersion
99            in
100              if ovv.oclIsUndefined()
101              then
```

```
102                Sequence{}
103             else
104                Sequence{ovv}
105             endif;
106          stepsToReproduceOrNot : Sequence(String) =
107             let strv : String = mi.stepsToReproduce
108             in
109                if strv.oclIsUndefined()
110                then
111                   Sequence{}
112                else
113                   Sequence{strv}
114                endif;
115          additionalInfoOrNot : Sequence(String) =
116             let aiv : String = mi.additionalInfo
117             in
118                if aiv.oclIsUndefined()
119                then
120                   Sequence{}
121                else
122                   Sequence{aiv}
123                endif;
124          fixedInVersionOrNot : Sequence(String) =
125             let fivv : String = mi.fixedInVersion
126             in
127                if fivv.oclIsUndefined()
128                then
129                   Sequence{}
130                else
131                   Sequence{fivv}
132                endif;
133          assignedToOrNot : Sequence(Mantis!Person) =
134             let atv : Mantis!Person = mi.assignedTo
135             in
136                if atv.oclIsUndefined()
137                then
138                   Sequence{}
139                else
140                   Sequence{atv}
141                endif;
142          projectionOrNot : Sequence(Mantis!ValueWithId) =
143             let projv : Mantis!ValueWithId = mi.projection
144             in
145                if projv.oclIsUndefined()
146                then
147                   Sequence{}
148                else
149                   Sequence{projv}
150                endif;
151          etaOrNot : Sequence(Mantis!ValueWithId) =
152             let ev : Mantis!ValueWithId = mi.eta
153             in
154                if ev.oclIsUndefined()
155                then
156                   Sequence{}
157                else
158                   Sequence{ev}
159                endif;
160          resolutionOrNot : Sequence(Mantis!ValueWithId) =
161             let resv : Mantis!ValueWithId = mi.resolution
162             in
163                if resv.oclIsUndefined()
```

```
164             then
165                Sequence{}
166             else
167                Sequence{resv}
168             endif;
169      }
170      to
171         xi : XML!Element (
172            name <- 'issue',
173            children <- Sequence{idAtt,privAtt,proj,cat,prior,sev,stat,rep,sum,desc,
174                                 vers,buil,plat,o,overs,repro,sTr,addInfo,dateSub,
175                                 assi,proje,e,res,fiv,
176                                 mi.attachments->collect(e | thisModule.resolveTemp(e,
177   'xa')),
178                                 mi.relationships->collect(e | thisModule.resolveTemp(e,
179   'xrs')),
180                                 mi.notes->collect(e | thisModule.resolveTemp(e, 'xn')),
181                                 lastUp }
182         ),
183         idAtt : XML!Attribute (
184            name <- 'id',
185            value <- mi.id.toString()
186         ),
187         privAtt : distinct XML!Attribute foreach(privateVal in privateOrNot)(
188            name <- 'private',
189            value <- privateVal
190         ),
191         proj : XML!Element (
192            name <- 'project',
193            children <- Sequence{projIdAtt,projVal}
194         ),
195         projIdAtt : XML!Attribute (
196            name <- 'id',
197            value <- mi.project.id.toString()
198         ),
199         projVal : XML!Text (
200            value <- mi.project.value
201         ),
202         cat : XML!Element (
203            name <- 'category',
204            children <- Sequence{catVal}
205         ),
206         catVal : XML!Text (
207            value <- mi.category
208         ),
209         prior : XML!Element (
210            name <- 'priority',
211            children <- Sequence{priorIdAtt,priorVal}
212         ),
213         priorIdAtt : XML!Attribute (
214            name <- 'id',
215            value <- mi.priority.id.toString()
216         ),
217         priorVal : XML!Text (
218            value <- mi.priority.value
219         ),
220         sev : XML!Element (
221            name <- 'severity',
222            children <- Sequence{sevIdAtt,sevVal}
223         ),
224         sevIdAtt : XML!Attribute (
225            name <- 'id',
```

```
226           value <- mi.severity.id.toString()
227         ),
228         sevVal : XML!Text (
229           value <- mi.severity.value
230         ),
231         stat : XML!Element (
232           name <- 'status',
233           children <- Sequence{statIdAtt,statVal}
234         ),
235         statIdAtt : XML!Attribute (
236           name <- 'id',
237           value <- mi.status.id.toString()
238         ),
239         statVal : XML!Text (
240           value <- mi.status.value
241         ),
242         rep : XML!Element (
243           name <- 'reporter',
244           children <- Sequence{repIdAtt,repLogAtt,repVal}
245         ),
246         repIdAtt : XML!Attribute (
247           name <- 'id',
248           value <- mi.reporter.id.toString()
249         ),
250         repLogAtt : XML!Attribute (
251           name <- 'login',
252           value <- mi.reporter.login
253         ),
254         repVal : XML!Text (
255           value <- mi.reporter.value
256         ),
257         sum : XML!Element (
258           name <- 'summary',
259           children <- Sequence{sumVal}
260         ),
261         sumVal : XML!Text (
262           value <- mi.summary
263         ),
264         desc : XML!Element (
265           name <- 'description',
266           children <- Sequence{descVal}
267         ),
268         descVal : XML!Text (
269           value <- mi.description
270         ),
271         vers : distinct XML!Element foreach(versionVal in versionOrNot)(
272           name <- 'version',
273           children <- Sequence{versVal}
274         ),
275         versVal : distinct XML!Text foreach(versionVal in versionOrNot)(
276           value <- versionVal
277         ),
278         buil : distinct XML!Element foreach(buildVal in buildOrNot)(
279           name <- 'build',
280           children <- Sequence{builVal}
281         ),
282         builVal : distinct XML!Text foreach(buildVal in buildOrNot)(
283           value <- buildVal
284         ),
285         plat : distinct XML!Element foreach(platformVal in platformOrNot)(
286           name <- 'platform',
287           children <- Sequence{platVal}
```

```
288         ),
289         platVal : distinct XML!Text foreach(platformVal in platformOrNot)(
290            value <- platformVal
291         ),
292         o : distinct XML!Element foreach(osVal in osOrNot)(
293            name <- 'os',
294            children <- Sequence{oVal}
295         ),
296         oVal : distinct XML!Text foreach(osVal in osOrNot)(
297            value <- osVal
298         ),
299         overs : distinct XML!Element foreach(osVersionVal in osVersionOrNot)(
300            name <- 'osVersion',
301            children <- Sequence{oversVal}
302         ),
303         oversVal : distinct XML!Text foreach(osVersionVal in osVersionOrNot)(
304            value <- osVersionVal
305         ),
306         repro : XML!Element (
307            name <- 'reproducibility',
308            children <- Sequence{reproIdAtt,reproVal}
309         ),
310         reproIdAtt : XML!Attribute (
311            name <- 'id',
312            value <- mi.reproducibility.id.toString()
313         ),
314         reproVal : XML!Text (
315            value <- mi.reproducibility.value
316         ),
317         sTr : distinct XML!Element foreach(stepsToReproduceVal in
318   stepsToReproduceOrNot)(
319            name <- 'stepsToReproduce',
320            children <- Sequence{sTrVal}
321         ),
322         sTrVal : distinct XML!Text foreach(stepsToReproduceVal in
323   stepsToReproduceOrNot)(
324            value <- stepsToReproduceVal
325         ),
326         addInfo : distinct XML!Element foreach(additionalInfoVal in
327   additionalInfoOrNot)(
328            name <- 'additionalInfo',
329            children <- Sequence{addInfoVal}
330         ),
331         addInfoVal : distinct XML!Text foreach(additionalInfoVal in
332   additionalInfoOrNot)(
333            value <- additionalInfoVal
334         ),
335         dateSub : XML!Element (
336            name <- 'dateSubmitted',
337            children <- Sequence{dateSubVal}
338         ),
339         dateSubVal : XML!Text (
340            value <- mi.dateSubmitted.toString()
341         ),
342         assi : distinct XML!Element foreach(assignedToVal in assignedToOrNot) (
343            name <- 'assignedTo',
344            children <- Sequence{assiIdAtt,assiLogAtt,assiVal}
345         ),
346         assiIdAtt : distinct XML!Attribute foreach(assignedToVal in assignedToOrNot)(
347            name <- 'id',
348            value <- assignedToVal.id.toString()
349         ),
```

```
350         assiLogAtt : distinct XML!Attribute foreach(assignedToVal in assignedToOrNot)
351     (
352             parent <- assi,
353             name <- 'login',
354             value <- assignedToVal.login
355         ),
356         assiVal : distinct XML!Text foreach(assignedToVal in assignedToOrNot) (
357             parent <- assi,
358             value <- assignedToVal.value
359         ),
360         proje : distinct XML!Element foreach(projectionVal in projectionOrNot) (
361             name <- 'projection',
362             children <- Sequence{projeIdAtt,projeVal}
363         ),
364         projeIdAtt : distinct XML!Attribute foreach(projectionVal in projectionOrNot)
365     (
366             name <- 'id',
367             value <- projectionVal.id.toString()
368         ),
369         projeVal : distinct XML!Text foreach(projectionVal in projectionOrNot) (
370             parent <- proje,
371             value <- projectionVal.value
372         ),
373         e : distinct XML!Element foreach(etaVal in etaOrNot) (
374             name <- 'eta',
375             children <- Sequence{eIdAtt,eVal}
376         ),
377         eIdAtt : distinct XML!Attribute foreach(etaVal in etaOrNot) (
378             name <- 'id',
379             value <- etaVal.id.toString()
380         ),
381         eVal : distinct XML!Text foreach(etaVal in etaOrNot) (
382             parent <- e,
383             value <- etaVal.value
384         ),
385         res : distinct XML!Element foreach(resolutionVal in resolutionOrNot) (
386             name <- 'resolution',
387             children <- Sequence{resIdAtt,resVal}
388         ),
389         resIdAtt : distinct XML!Attribute foreach(resolutionVal in resolutionOrNot) (
390             name <- 'id',
391             value <- resolutionVal.id.toString()
392         ),
393         resVal : distinct XML!Text foreach(resolutionVal in resolutionOrNot) (
394             parent <- res,
395             value <- resolutionVal.value
396         ),
397         fiv : distinct XML!Element foreach(fixedInVersionVal in fixedInVersionOrNot)(
398             name <- 'fixedInVersion',
399             children <- Sequence{fivVal}
400         ),
401         fivVal : distinct XML!Text foreach(fixedInVersionVal in fixedInVersionOrNot)(
402             value <- fixedInVersionVal
403         ),
404         lastUp : XML!Element (
405             name <- 'lastUpdate',
406             children <- Sequence{lastUpVal}
407         ),
408         lastUpVal : XML!Text (
409             value <- mi.lastUpdate.toString()
410         )
411     }
```

```
412
413
414    -- Rule 'Attachment2Attachment'
415    -- This rule generates the attachment's XML tags
416    -- from the "Attachment" element
417    rule  Attachment2Attachment {
418       from
419          ma : Mantis!Attachment
420
421       to
422          xa : XML!Element (
423             name <- 'attachment',
424             children <- Sequence{fileN,si,cType,ts,dlU}
425          ),
426          fileN : XML!Element (
427             name <- 'filename',
428             children <- Sequence{fileNVal}
429          ),
430          fileNVal : XML!Text (
431             value <- ma.filename
432          ),
433          si : XML!Element (
434             name <- 'size',
435             children <- Sequence{siVal}
436          ),
437          siVal : XML!Text (
438             value <- ma.size.toString()
439          ),
440          cType : XML!Element (
441             name <- 'contentType',
442             children <- Sequence{cTypeVal}
443          ),
444          cTypeVal : XML!Text (
445             value <- ma.contentType
446          ),
447          ts : XML!Element (
448             name <- 'timestamp',
449             children <- Sequence{tsVal}
450          ),
451          tsVal : XML!Text (
452             value <- ma.timestamp
453          ),
454          dlU : XML!Element (
455             name <- 'downloadUrl',
456             children <- Sequence{dlUVal}
457          ),
458          dlUVal : XML!Text (
459             value <- ma.downloadUrl
460          )
461    }
462
463
464    -- Rule 'Relationship2Relationship'
465    -- This rule generates the relationship's XML tags
466    -- from the "Relationship" element
467    rule  Relationship2Relationship {
468       from
469          mr : Mantis!Relationship
470
471       to
472          xrs : XML!Element (
473             name <- 'relationship',
```

```
474            children <- Sequence{typ,rid}
475         ),
476         typ : XML!Element (
477            name <- 'type',
478            children <- Sequence{typVal}
479         ),
480         typVal : XML!Text (
481            value <- thisModule.getRelationshipTypeStringValue(mr.type)
482         ),
483         rid : XML!Element (
484            name <- 'id',
485            children <- Sequence{ridVal}
486         ),
487         ridVal : XML!Text (
488            value <- mr.id
489         )
490   }
491
492
493   -- Rule 'Note2Note'
494   -- This rule generates the note's XML tags
495   -- from the "Note" element
496   rule  Note2Note {
497      from
498         mn : Mantis!Note
499      using {
500         privateOrNot : Sequence(String) =
501            let priv : Boolean = mn.private
502            in
503               if priv.oclIsUndefined()
504               then
505                  Sequence{}
506               else
507                  Sequence{priv.toString()}
508               endif;
509      }
510      to
511         xn : XML!Element (
512            name <- 'note',
513            children <- Sequence{privAtt,auth,ts,tex}
514         ),
515         privAtt : distinct XML!Attribute foreach(privateVal in privateOrNot)(
516            name <- 'private',
517            value <- privateVal
518         ),
519         auth : XML!Element (
520            name <- 'author',
521            children <- Sequence{authId,authLog,authVal}
522         ),
523         authId : XML!Attribute (
524            name <- 'id',
525            value <- mn.author.id.toString()
526         ),
527         authLog : XML!Attribute (
528            name <- 'login',
529            value <- mn.author.login
530         ),
531         authVal : XML!Text (
532            value <- mn.author.value
533         ),
534         ts : XML!Element (
535            name <- 'timestamp',
```

```
536             children <- Sequence{tsVal}
537         ),
538         tsVal : XML!Text (
539             value <- mn.timestamp.toString()
540         ),
541         tex : XML!Element (
542             name <- 'text',
543             children <- Sequence{texVal}
544         ),
545         texVal : XML!Text (
546             value <- mn.text
547         )
548     }
```

1.4.3. XML2MantisText

The ATL code for this transformation consists in 4 helpers and 1 query.

Contrary to rules that are implemented to generate a model from another model, a query allows calculating output text files from an input model (see [4]). This is the reason why we need to use queries for this type of transformation: generating an XML file from an XML model. The implemented query gets the XML!Root of the XML model and calls the *MantisFile* helper on it. It recovers the string value returned by this helper (corresponding to the generated XML text) and writes it into an XML file located in the path passed in argument. The parsing of all input model's elements is recursively made from the *MantisFile* helper.

The *MantisFile* helper returns a string which is composed of the required XML file's header and of the Mantis XML file's content. This content is generated by the *toString2* helper called on the XML!Root element of the XML model.

There are three *toString2* helpers with different contexts. The XML!Attribute one simply returns the name and the value of an attribute in the correct string format. The XML!Text one only returns the string value contained in a text node. The XML!Element one returns the valid and well-formed content of the output XML file by parsing recursively all the elements of the input XML model (note that it sometimes calls the XML!Attribute and XML!Text *toString2* helpers).

```
1   query XML2Text = XML!Root.allInstances()
2        ->asSequence()
3        ->first().MantisFile().writeTo('C:\\ … path to be completed before using the
4   transformation …\\MantisXMLfileExample.xml');
5
6   helper context XML!Root def: MantisFile() : String =
7      '<?xml version="1.0" encoding="ISO-8859-1"?>'+'\n'+ self.toString2('');
8
9   helper context XML!Element def: toString2(indent : String) : String =
10     let na : Sequence(XML!Node) =
11        self.children->select(e | not e.oclIsKindOf(XML!Attribute)) in
12     let a : Sequence(XML!Node) =
13        self.children->select(e | e.oclIsKindOf(XML!Attribute)) in
14     indent + '<' + self.name +
15     a->iterate(e; acc : String = '' |
16        acc + ' ' + e.toString2()
17     ) +
18     if na->size() > 0 then
19        '>'
20        + na->iterate(e; acc : String = '' |
21           acc +
22           if e.oclIsKindOf(XML!Text) then
23              ''
24           else
25              '\r\n'
26           endif
27           + e.toString2(indent + '  ')
28        ) +
29        if na->first().oclIsKindOf(XML!Text) then
30           '</' + self.name + '>'
31           else
32              '\r\n' + indent + '</' + self.name + '>'
33        endif
34     else
35        '/>'
```

```
36      endif;
37
38
39    helper context XML!Attribute def: toString2() : String =
40      self.name + '="' + self.value + '"';
41
42
43    helper context XML!Text def: toString2() : String =
44      self.value;
```

I. SoftwareQualityControl metamodel in KM3 format

```
-- @name   SoftwareQualityControl
-- @version  1.0
-- @domains  Software, Quality control, Software life cycle
-- @authors  Hugo Bruneliere (hugo.bruneliere@gmail.com)
-- @date   2005/07/04
-- @description  This metamodel describes a simple structure to manage software
quality control and especially bug tracking. It is based on a simple Excel table
representation.

package SoftwareQualityControl {

   -- @begin Controls' general information

   -- @comment Defines the format for the dates (DD/MM/YY).
   class Date {
      attribute day : Integer;
      attribute month : Integer;
      attribute year : Integer;
   }

   -- @comment Defines a sequence of controls. This is the root container.
   class ControlsSequence {
      reference controls[*] ordered container : Control oppositeOf
c_controlsSequence;
   }

   -- @comment Defines a control (general information, type, details...)
   class Control {
      reference c_controlsSequence : ControlsSequence oppositeOf controls;

      -- @comment The surname and name of the person who is responsible for this
control.
      attribute responsible : String;
      -- @comment The name of the component which is concerned by this control.
      attribute component : String;
      -- @comment The name of the development phase during which the control takes
place.
      attribute developmentPhase : String;
      -- @comment The scope of this control, for example "Exhaustive".
      attribute scope : String;
      -- @comment The date of this control (in the format : DD/MM/YY).
      reference date container : Date;
      -- @comment The name of the specific element which is controlled.
      attribute controlledElt[0-1] : String;
      -- @comment The reference of this specific element.
      attribute eltRef[0-1] : String;
      -- @comment The author's name of this specific element.
      attribute eltAuthor[0-1] : String;
      -- @comment The form reference for this control.
      attribute formRef[0-1] : String;

      -- @comment The type of this control. The data contained in a "Control"
element depends on the type of this control.
      reference type : ControlType oppositeOf ct_control;
   }

   -- @end Controls' general information
```

```
   -- @begin Specific information for types of control

   -- @comment Defines the abstract concept of type of control. It exists several
types of control. Each class which represents a type of control must inherit of
this class.
   abstract class ControlType {
      reference ct_control[*] : Control oppositeOf type;
   }

   -- @comment Defines a special control type which is bug tracking.
   class BugTracking extends ControlType {
      -- @comment Represents the different bugs tracked during the control.
      reference bugs[*] ordered container : Bug oppositeOf b_bugTracking;
   }

   -- @comment Defines a bug and the associated information.
   class Bug {
      reference b_bugTracking : BugTracking oppositeOf bugs;

      -- @comment The bug identification number
      attribute number : Integer;
      -- @comment The version of the component from which the bug has been detected.
      attribute componentVersion : String;
      -- @comment The complete description of the bug.
      attribute description : String;
      -- @comment The current status of the bug
      attribute status : BugStatusType;
      -- @comment The name of the person who find the bug.
      attribute originator : String;
      -- @comment The name of the person who is responsible for this bug.
      attribute responsible[0-1] : String;
      -- @comment Special comments or possible answers to correct this bug.
      attribute commentsAnswers[0-1] : String;
      -- @comment The date when the bug has been indexed.
      attribute openDate : String;
      -- @comment The date when the bug has been resolved.
      attribute closeDate[0-1] : String;
   }

   -- @comment Defines the type of status for a bug.
   enumeration BugStatusType {
      literal bst_open;
      literal bst_closed;
      literal bst_skipped;
   }

   -- @end Specific information for types of control

}


package PrimitiveTypes {

   datatype Integer;
   datatype String;
   datatype Boolean;
   datatype Double;

}
```

II. Mantis metamodel in KM3 format

```
-- @name   Mantis
-- @version  1.1
-- @domains  Software, Quality control, Bug tracking
-- @authors  Hugo Bruneliere (hugo.bruneliere@gmail.com)
-- @date   2005/07/11
-- @description This metamodel describes the structure used by Mantis, a web-based
bugtracking system written in PHP and using MySQL database, to import/export data
in XML.
-- @see    mantis.xsd, http://www.mantisbt.org/mantis/view.php?id=4024 at the bottom
of the page

package Mantis {

   -- @begin Special types

   -- @comment Defines the different possible types of relationship between two
bugs.
   enumeration RelationshipType {
      literal "rt_related to";
      literal "rt_parent of";
      literal "rt_child of";
      literal "rt_duplicate of";
      literal "rt_has duplicate";
   }

   -- @end Special types


   -- @begin Mantis general structure

   -- @comment Defines the root element that contains the bugs.
   class MantisRoot {
      reference issues[*] ordered container : Issue oppositeOf i_mantisRoot;
   }

   -- @comment Defines the abstract concept of an element with an identifying
number.
   abstract class IdentifiedElt {
      attribute id : Integer;
   }

   -- @comment Defines the abstract concept of an element with a boolean that can
indicate if this element is private or not.
   abstract class PrivateElt {
      attribute private[0-1] : Boolean;
   }

   -- @comment Defines a bug (a bug is an "Issue" in Mantis).
   class Issue extends IdentifiedElt, PrivateElt {
      reference i_mantisRoot : MantisRoot oppositeOf issues;

      -- @comment All the information related to a bug.
      reference project container : ValueWithId;
      attribute category : String;
      reference priority container : ValueWithId;
      reference severity container : ValueWithId;
      reference status container : ValueWithId;
```

```
      reference reporter container : Person;
      attribute summary : String;
      attribute description : String;
      attribute version[0-1] : String;
      attribute build[0-1] : String;
      attribute platform[0-1] : String;
      attribute os[0-1] : String;
      attribute osVersion[0-1] : String;
      reference reproducibility container : ValueWithId;
      attribute stepsToReproduce[0-1] : String;
      attribute additionalInfo[0-1] : String;
      attribute dateSubmitted : Integer;
      reference assignedTo[0-1] container : Person;
      reference projection[0-1] container : ValueWithId;
      reference eta[0-1] container : ValueWithId;
      reference resolution[0-1] container : ValueWithId;
      attribute fixedInVersion[0-1] : String;
      reference attachments[*] ordered container : Attachment;
      reference relationships[*] ordered container : Relationship;
      reference notes[*] ordered container : Note;
      attribute lastUpdate : Integer;
   }

   -- @comment Defines an element composed of an identifier associated to a value.
   class ValueWithId extends IdentifiedElt {
      attribute value : String;
   }

   -- @comment Defines a person by using his identifier, his login and his complete
name (contained in the attribute "value").
   class Person extends ValueWithId {
      attribute login : String;
   }

   -- @coment Defines a relationship between two bugs.
   class Relationship extends IdentifiedElt  {
      attribute type : RelationshipType ;
   }

   -- @comment Defines the abstract concept of an element with a "timestamp" value.
   abstract class TimeStampedElt {
      attribute timestamp : Integer;
   }

   -- @comment Defines a note (a comment) associated to a bug.
   class Note extends TimeStampedElt, PrivateElt {
      reference author container : Person;
      attribute text : String;
   }

   -- @comment Defines an attachment of type "file" associated to a bug.
   class Attachment extends TimeStampedElt {
      attribute filename : String;
      attribute size : Integer;
      attribute contentType : String;
      attribute downloadUrl : String;
   }

   -- @end Mantis general structure

}
```

```
package PrimitiveTypes {

   datatype Integer;
   datatype String;
   datatype Boolean;
   datatype Double;

}
```

III. XML metamodel in KM3 format

```
-- @name   XML
-- @version  1.1
-- @domains  XML
-- @authors  Peter Rosenthal (peter.rosenthal@univ-nantes.fr)
-- @date   2005/06/13
-- @description This metamodel defines a subset of Extensible Markup Language (XML)
and particulary XML document. It describes an XML document composed of one root
node. Node is an abstract class having two direct children, namely ElementNode and
AttributeNode. ElementNode represents the tags, for example a tag named xml:
<xml></xml>. ElementNodes can be composed of many Nodes. AttributeNode represents
attributes, which can be found in a tag, for example the attr attribute: <xml
attr="value of attr"/>. ElementNode has two sub classes, namely RootNode and
TextNode. RootNode is the root element. The TextNode is a particular node, which
does not look like a tag; it is only a string of characters.

package XML {
   abstract class Node {
      attribute startLine[0-1] : Integer;
      attribute startColumn[0-1] : Integer;
      attribute endLine[0-1] : Integer;
      attribute endColumn[0-1] : Integer;
      attribute name : String;
      attribute value : String;
      reference parent[0-1] : Element oppositeOf children;
   }

   class Attribute extends Node {}

   class Text extends Node {}

   class Element extends Node {
      reference children[*] ordered container : Node oppositeOf parent;
   }

   class Root extends Element {}
}

package PrimitiveTypes {
   datatype Boolean;
   datatype Integer;
   datatype String;
}
```

References

[1] Mantis Bug Tracker official site, http://www.mantisbt.org/

[2] ExampleMicrosoftOfficeExcel2SoftwareQualityControl[v00.01].pdf, http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/subprojects/ATL/ATL_examples/MicrosoftOfficeExcel2SoftwareQualityControl/ExampleMicrosoftOfficeExcel2SoftwareQualityControl%5Bv00.01%5D.pdf

[3] mantis.xsd, file available at http://www.mantisbt.org/mantis/view.php?id=4024

[4] ATL User manual, “4.1 Queries and the Generation of Text” subsection, http://www.eclipse.org/gmt/, ATL subproject, ATL Documentation Section