	<b>ATL TRANSFORMATION EXAMPLE</b>	Hugo Brunelière hugo.bruneliere@gmail.com
	<b>Software Quality Control to Bugzilla file</b>	Date 02/08/2005

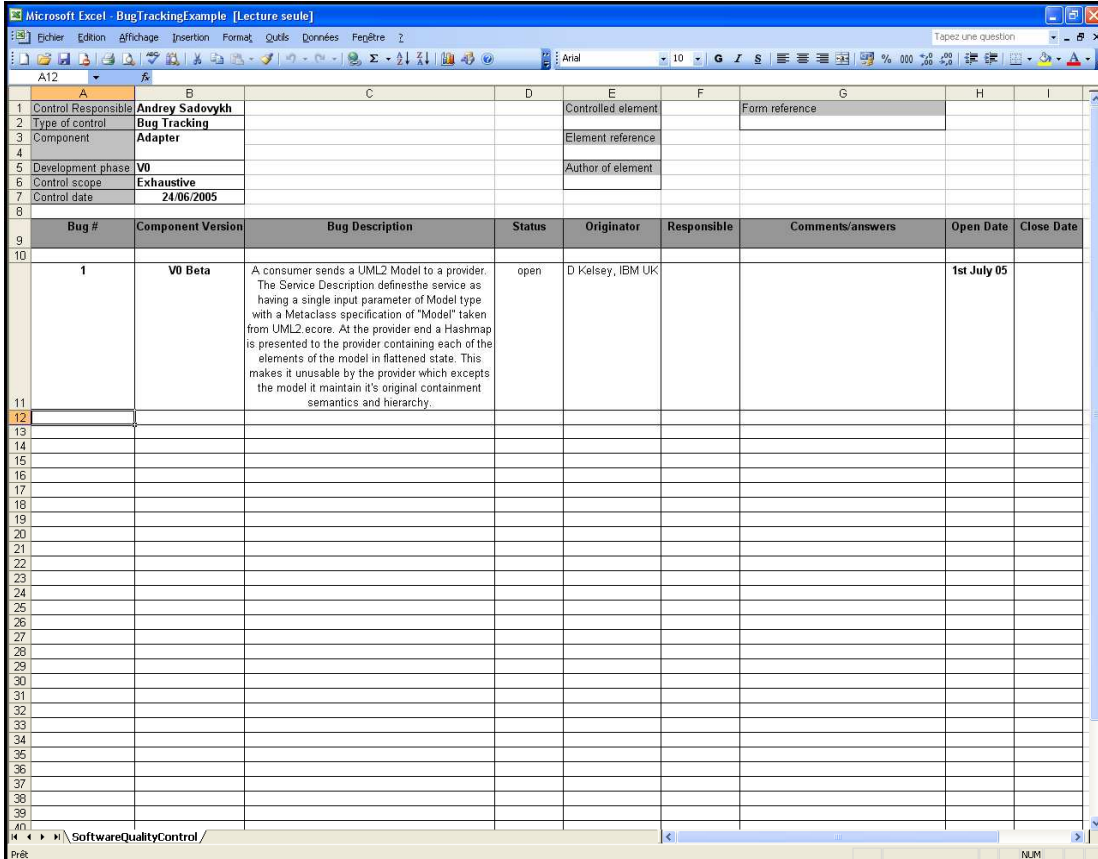
## 1. ATL Transformation Example

### 1.1. Example: Software Quality Control → Bugzilla file

The “Software Quality Control to Bugzilla file” example describes a transformation from a SoftwareQualityControl model to a simple Bugzilla XML file. Bugzilla [1] is a free "Defect Tracking System" or "Bug-Tracking System", originally developed by the Mozilla Foundation, which allows individual or groups of developers to keep track of outstanding bugs in their product effectively. 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 simple Bugzilla DTD.

#### 1.1.1. Transformation overview

The aim of this transformation is to generate a valid and well-formed XML file for Bugzilla 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]).



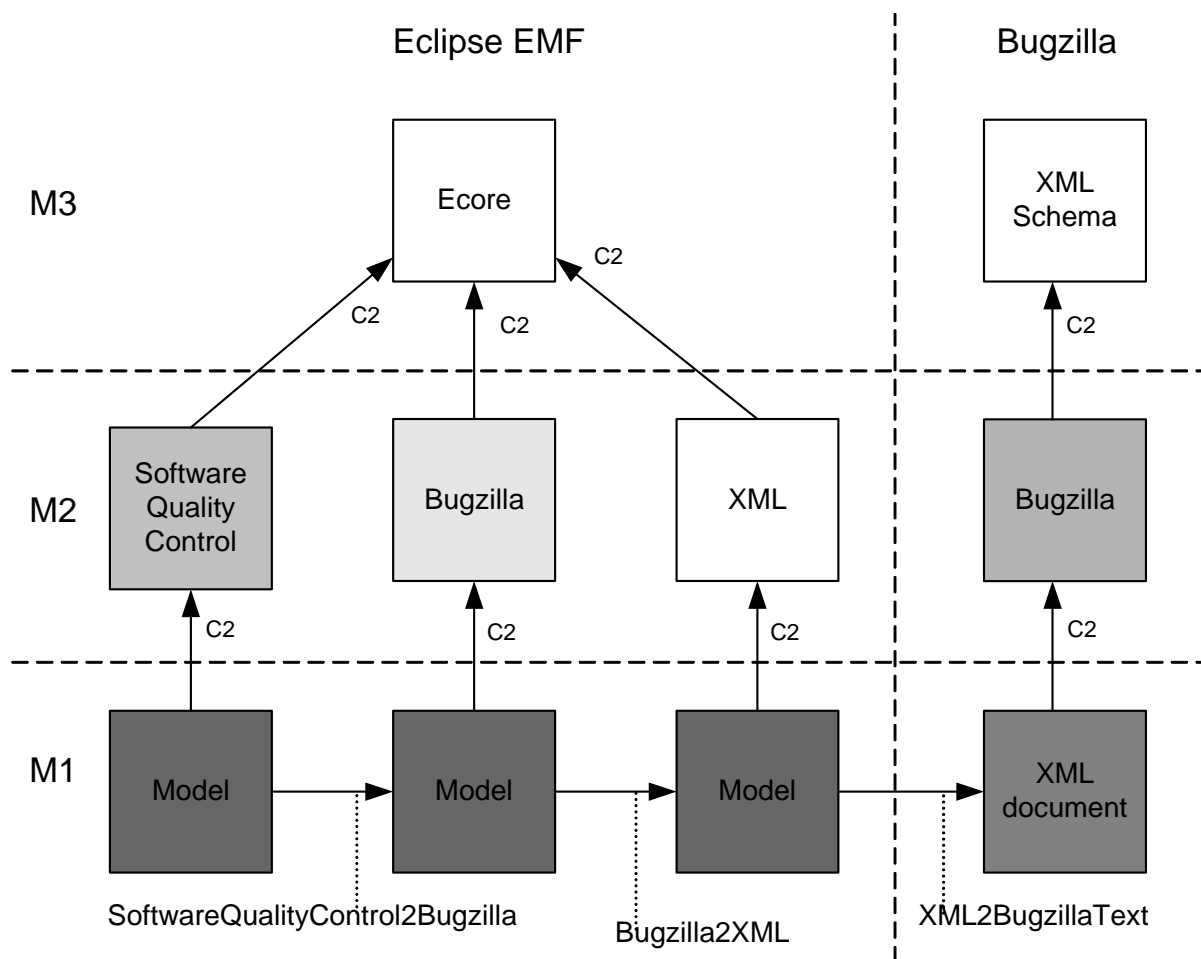
Bug #	Component Version	Bug Description	Status	Originator	Responsible	Comments/answers	Open Date	Close Date
1	V0 Beta	A consumer sends a UML2 Model to a provider. The Service Description defines the service as having a single input parameter of Model type with a Metaclass specification of "Model" taken from UML2.ecore. At the provider end a Hashmap is presented to the provider containing each of the elements of the model in flattened state. This makes it unusable by the provider which expects the model it maintain it's original containment semantics and hierarchy.	open	D Kelsey, IBM UK			1st July 05	

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


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

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

These three steps are summarized in Figure 2.

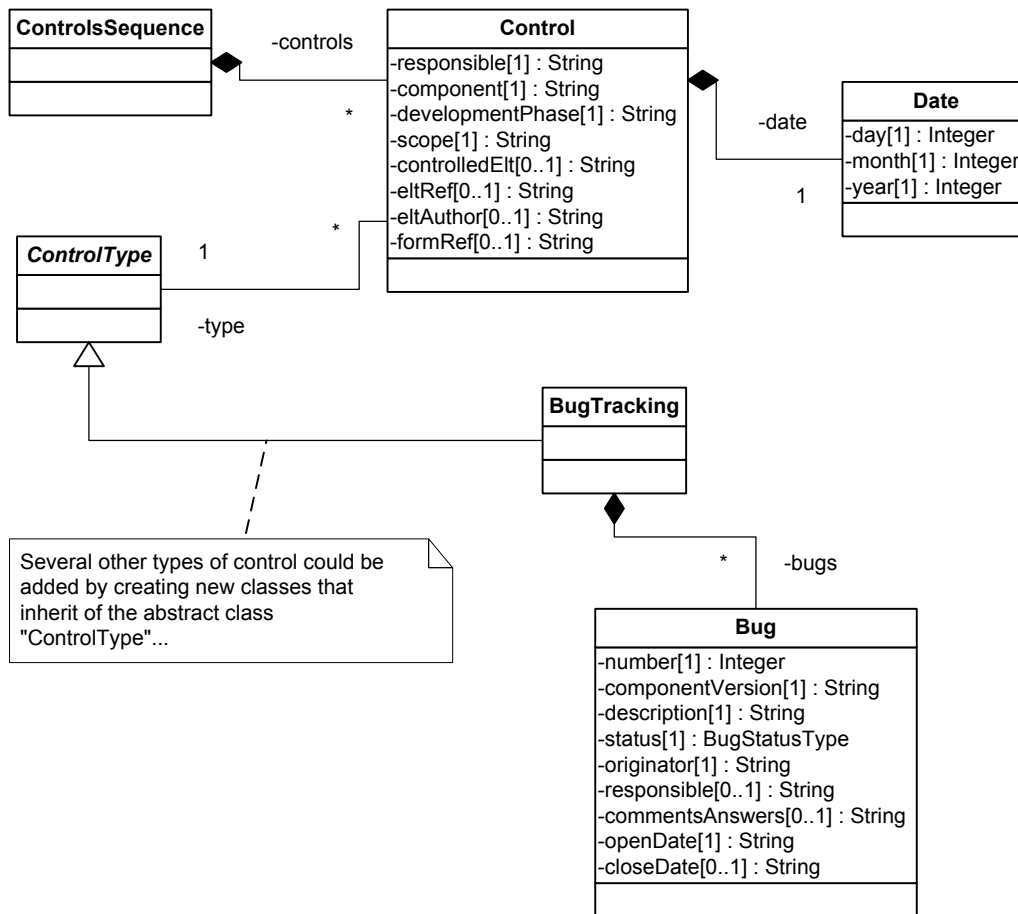


**Figure 2. “Software Quality Control to Bugzilla file” transformation’s overview**

	<b>ATL TRANSFORMATION EXAMPLE</b>	Hugo Brunelière hugo.bruneliere@gmail.com
	<b>Software Quality Control to Bugzilla file</b>	Date 02/08/2005

## 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.

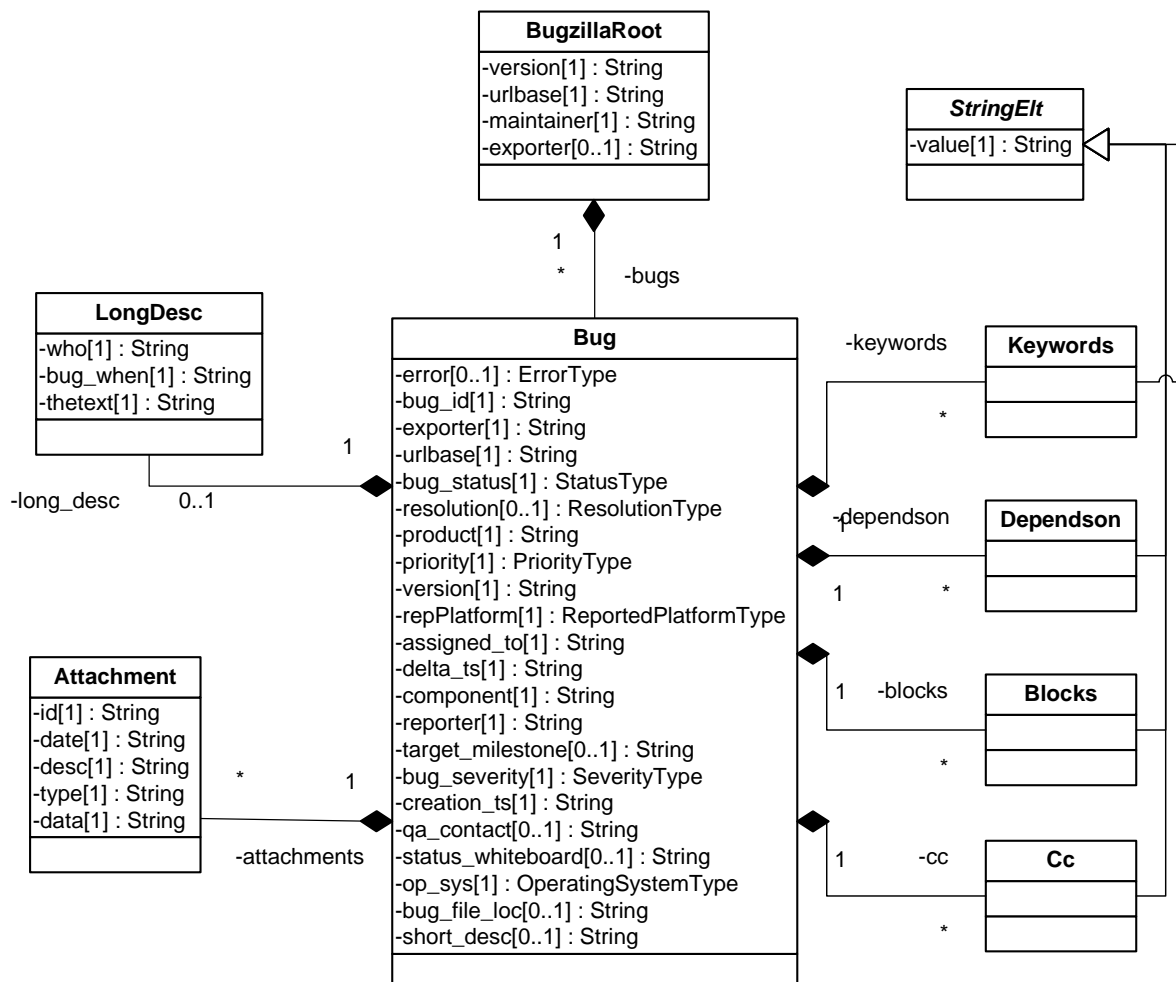


**Figure 3. The SoftwareQualityControl metamodel**

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 “Bugzilla” metamodel. Bugzilla is a free “defect-tracking” or “bug-tracking” system originally developed by the Mozilla Foundation. A huge database allows it 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, it is possible to use a Perl script to import/export bugs’ data in a simpler XML format. The data in XML files conforms to a simple DTD [3].

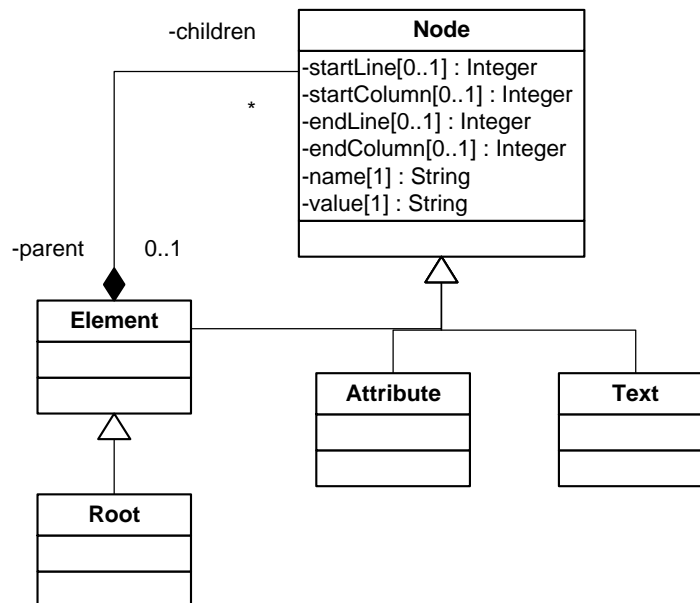
The simple Bugzilla metamodel considered here is directly inspired by this DTD. It is described in Figure 4 and provided in Appendix II in km3 format.



**Figure 4. The Bugzilla metamodel**


A Bugzilla model is a set of *Bug* elements. Each *Bug* is identified by an “id” string and contains much information about the bug itself but also about the people who deals with it, the software product that is concerned with, 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 *Attribute* is characterized by its name and its value whereas a *Text* is only assimilated to a single value.

	<b>ATL TRANSFORMATION EXAMPLE</b>	Hugo Brunelière hugo.bruneliere@gmail.com
	<b>Software Quality Control to Bugzilla file</b>	Date 02/08/2005

### 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 Bugzilla XML file whose content conforms to a simple Bugzilla DTD [3]. The input Bugzilla model of the second transformation is the output Bugzilla 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 Bugzilla


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

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

#### 1.3.2. Bugzilla to XML

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

- For the root *Bugzilla!BugzillaRoot* element, the “bugzilla” *XML!Root* element is created. The required *XML!Attribute* elements are also generated and added as children of this “bugzilla” *XML!Root* element.
- For each *Bugzilla!Bug* element, a “bug” *XML!Element* element is engendered and set as a child of the “bugzilla” *XML!Root* element. All the necessary *XML!Attribute*, *XML!Element* and *XML!Text* elements are also created and added as children of this “bug” *XML!Element* element.
- For each *Bugzilla!Keywords* element, a “keywords” *XML!Element* element and its child's *XML!Text* element are generated.
- For each *Bugzilla!Dependson* element, a “dependson” *XML!Element* element and its child's *XML!Text* element are created.
- For each *Bugzilla!Blocks* element, a “blocks” *XML!Element* element and its child's *XML!Text* element are engendered.
- For each *Bugzilla!Cc* element, a “cc” *XML!Element* element and its child's *XML!Text* element are generated.
- For each *Bugzilla!LongDesc* element, a “long\_desc” *XML!Element* element and its children's *XML!Element* and *XML!Text* elements are generated.
- For each *Bugzilla!Attachment* element, an “attachment” *XML!Element* element and its children's *XML!Element* and *XML!Text* elements are generated.

	<b>ATL TRANSFORMATION EXAMPLE</b>	Hugo Brunelière hugo.bruneliere@gmail.com
	<b>Software Quality Control to Bugzilla file</b>	Date 02/08/2005

### 1.3.3. XML to Bugzilla XML file (i.e. XML to Bugzilla 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 Bugzilla 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. SoftwareQualityControl2Bugzilla

The ATL code for the “SoftwareQualityControl2Bugzilla” transformation consists of 1 helper and 2 rules.

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

The rule *BugTracking2BugzillaRoot* allocates a Bugzilla!BugzillaRoot element only if a BugTracking element is encountered in the input SoftwareQualityControl model. This generated Bugzilla!BugzillaRoot element will be linked, thanks to a “resolveTemp(...)” method’s call, to the corresponding Bugzilla!Bug elements that will be created by the following rule during the transformation.

The rule *Bug2Bug* allocates a Bugzilla!Bug element and a Bugzilla!LongDesc element 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 SoftwareQualityControl2Bugzilla; -- Module Template
2  create OUT : Bugzilla from IN : SoftwareQualityControl;
3
4
5  -- This helper permits to convert the status value of a bug
6  -- in a right Bugzilla status type value.
7  -- CONTEXT: n/a
8  -- RETURN: Bugzilla!StatusType
9  helper def: convertStatus(bs : SoftwareQualityControl!BugStatusType) :
10 Bugzilla!StatusType =
11   if bs = #bst_open
12   then
13     #st_new
14   else
15     if bs = #bst_closed
16     then
17       #st_closed
18     else
19       if bs = #bst_skipped
20       then
21         #st_unconfirmed
22       else
23         #st_new

```

```
24     endif
25   endif
26 endif;
27
28
29
30 -- Rule 'BugTracking2BugzillaRoot'
31 -- This rule generates the root of the Bugzilla output model
32 -- if a BugTracking element exists in the input model
33 rule BugTracking2BugzillaRoot {
34   from
35     bt : SoftwareQualityControl!BugTracking
36
37   to
38     br : Bugzilla!BugzillaRoot (
39       version <- '',
40       urlbase <- '',
41       maintainer <- '',
42       --exporter <- '',
43       bugs <- bt.bugs->collect(e | thisModule.resolveTemp(e, 'bb'))
44     )
45 }
46
47
48 -- Rule 'Bug2Bug'
49 -- This rule generates a bug in Bugzilla for each
50 -- bug reported in the BugTracking element.
51 rule Bug2Bug {
52   from
53     bbt : SoftwareQualityControl!Bug
54
55   to
56     bb : Bugzilla!Bug (
57       --error <- Bugzilla!ErrorType,
58       bug_id <- bbt.number.toString(),
59       exporter <- '',
60       urlbase <- '',
61       bug_status <- thisModule.convertStatus(bbt.status),
62       --resolution <- Bugzilla!ResolutionType,
63       product <- '',
64       priority <- #pt_P1,
65       version <- bbt.componentVersion,
66       rep_platform <- #rpt_all,
67       assigned_to <- let v : String = bbt.responsible in
68         if v.oclIsUndefined()
69           then bbt.b_bugTracking.ct_control.responsible
70         else v
71         endif,
72       delta_ts <- let v : String = bbt.closeDate in
73         if v.oclIsUndefined()
74           then ''
75         else v
76         endif,
77       component <- bbt.b_bugTracking.ct_control.component,
78       reporter <- bbt.originator,
79       target_milestone <- String,
80       bug_severity <- #st_normal,
81       creation_ts <- bbt.openDate,
82       qa_contact <- bbt.b_bugTracking.ct_control.responsible,
83       --status_whiteboard <- '',
84       op_sys <- #ost_all, -- #"ost_Windows XP"
85       bug_file_loc <- String,
```



```
86     short_desc <- bbt.description,
87     keywords <- Sequence{},
88     dependson <- Sequence{},
89     blocks <- Sequence{},
90     cc <- Sequence{},
91     long_desc <- commentsAndAnswers,
92     attachment <- Sequence{}
93 ),
94 commentsAndAnswers : Bugzilla!LongDesc (
95     who <- bbt.originator,
96     bug_when <- bbt.openDate,
97     thetext <- let v : String = bbt.commentsAnswers in
98         if v.oclIsUndefined()
99         then ''
100        else v
101        endif
102 )
103 }
```

	<b>ATL TRANSFORMATION EXAMPLE</b>	Contributor Hugo Brunelière
	<b>Software Quality Control to Bugzilla file</b>	Date 02/08/2005

### 1.4.2. Bugzilla2XML

The ATL code for the “Bugzilla2XML” transformation consists of 7 helpers and 8 rules.

All the helpers have a quite similar function:

- The *getStringErrorValue* helper returns the string value corresponding to the Bugzilla!ErrorType passed in argument.
- The *getStringBugStatusValue* helper returns the string value corresponding to the Bugzilla!bugStatusType passed in argument.
- The *getStringResolutionValue* helper returns the string value corresponding to the Bugzilla!ResolutionType passed in argument.
- The *getStringPriorityValue* helper returns the string value corresponding to the Bugzilla!PriorityType passed in argument.
- The *getStringRepPlatformValue* helper returns the string value corresponding to the Bugzilla!ReportedPlatformType passed in argument.
- The *getStringSeverityValue* helper returns the string value corresponding to the Bugzilla!SeverityType passed in argument.
- The *getStringOperatingSystemValue* helper returns the string value corresponding to the Bugzilla!OperatingSystemType 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 Bugzilla 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 simple Bugzilla DTD [3].

As an example, the *BugzillaRoot2Root* rule allocates a “bugzilla” XML!Element and three or four XML!Attribute elements (one of the corresponding attributes is optional in the Bugzilla metamodel), which are children of the XML!Element, for each BugzillaRoot element of the input Bugzilla model. This “bugzilla” XML!Element will be linked, thanks to a “resolveTemp(...)” method’s call, to the “bug” XML!Element elements that will be created to represent bugs by the *Bug2Bug* rule...

```

1  module Bugzilla2XML; -- Module Template
2  create OUT : XML from IN : Bugzilla;
3
4
5  -- This helper permits to obtain the string associated
6  -- to an ErrorType value.
7  -- CONTEXT: n/a
8  -- RETURN: String
9  helper def: getStringErrorValue(ev : Bugzilla!ErrorType) : String =
10     let sev : String = ev.toString()
11     in
12         sev.substring(4,sev.size());
13
14  -- This helper permits to obtain the string associated
15  -- to a StatusType value for a bug.
16  -- CONTEXT: n/a
17  -- RETURN: String

```

```
18  helper def: getStringBugStatusValue(sv : Bugzilla!StatusType) : String =
19      let ssv : String = sv.toString()
20      in
21          ssv.substring(4,ssv.size());
22
23  -- This helper permits to obtain the string associated
24  -- to a ResolutionType value for a bug.
25  -- CONTEXT: n/a
26  -- RETURN: String
27  helper def: getStringResolutionValue(rv : Bugzilla!ResolutionType) : String =
28      let srv : String = rv.toString()
29      in
30          srv.substring(4,srv.size());
31
32  -- This helper permits to obtain the string associated
33  -- to a PriorityType value for a bug.
34  -- CONTEXT: n/a
35  -- RETURN: String
36  helper def: getStringPriorityValue(pv : Bugzilla!PriorityType) : String =
37      let spv : String = pv.toString()
38      in
39          spv.substring(4,spv.size());
40
41  -- This helper permits to obtain the string associated
42  -- to a ReportedPlatformType value for a bug.
43  -- CONTEXT: n/a
44  -- RETURN: String
45  helper def: getStringRepPlatformValue(rp : Bugzilla!ReportedPlatformType) : String
46  =
47      let srp : String = rp.toString()
48      in
49          srp.substring(5,srp.size());
50
51  -- This helper permits to obtain the string associated
52  -- to a SeverityType value for a bug.
53  -- CONTEXT: n/a
54  -- RETURN: String
55  helper def: getStringSeverityValue(sv : Bugzilla!SeverityType) : String =
56      let ssv : String = sv.toString()
57      in
58          ssv.substring(4,ssv.size());
59
60  -- This helper permits to obtain the string associated
61  -- to an OperatingSystemType value for a bug.
62  -- CONTEXT: n/a
63  -- RETURN: String
64  helper def: getStringOperatingSystemValue(osv : Bugzilla!OperatingSystemType) :
65  String =
66      let sosv : String = osv.toString()
67      in
68          sosv.substring(5,sosv.size());
69
70
71
72  -- Rule 'BugzillaRoot2Root'
73  -- This rule generates the root of the XML model
74  -- from the "BugzillaRoot" element
75  rule BugzillaRoot2Root {
76      from
77          br : Bugzilla!BugzillaRoot
78      using {
79          exporterOrNot : Sequence(String) =
```

```
80     let exp : String = br.exporter
81     in
82     if exp.oclIsUndefined()
83     then
84         Sequence{}
85     else
86         Sequence{exp}
87     endif;
88 }
89 to
90 xr : XML!Root (
91     name <- 'bugzilla',
92     children <- Sequence{v,u,m,e,
93         br.bugs->collect(e | thisModule.resolveTemp(e, 'xb'))
94     }
95 ),
96 v : XML!Attribute (
97     name <- 'version',
98     value <- br.version
99 ),
100 u : XML!Attribute (
101     name <- 'urlbase',
102     value <- br.urlbase
103 ),
104 m : XML!Attribute (
105     name <- 'maintainer',
106     value <- br.maintainer
107 ),
108 e : distinct XML!Attribute foreach(exporterVal in exporterOrNot) (
109     name <- 'exporter',
110     value <- exporterVal
111 )
112 }
113
114
115 -- Rule 'Bug2Bug'
116 -- This rule generates the XML bugs' tags
117 -- from the "Bug"s element
118 rule Bug2Bug {
119     from
120     b : Bugzilla!Bug
121     using {
122     errorOrNot : Sequence(Bugzilla!ErrorType) =
123         let err : Bugzilla!ErrorType = b.error
124         in
125             if err = #et_null
126             then
127                 Sequence{}
128             else
129                 Sequence{err}
130             endif;
131     resolutionOrNot : Sequence(Bugzilla!ResolutionType) =
132         let resol : Bugzilla!ResolutionType = b.resolution
133         in
134             if resol = #rt_null
135             then
136                 Sequence{}
137             else
138                 Sequence{resol}
139             endif;
140     targetMilestoneOrNot : Sequence(String) =
141         let tm : String = b.target_milestone
```

```

142         in
143         if tm.oclIsUndefined()
144         then
145             Sequence{}
146         else
147             Sequence{tm}
148         endif;
149 qaContactOrNot : Sequence(String) =
150     let qac : String = b.qa_contact
151     in
152         if qac.oclIsUndefined()
153         then
154             Sequence{}
155         else
156             Sequence{qac}
157         endif;
158 statusWhiteboardOrNot : Sequence(String) =
159     let sw : String = b.status_whiteboard
160     in
161         if sw.oclIsUndefined()
162         then
163             Sequence{}
164         else
165             Sequence{sw}
166         endif;
167 bugFileLocOrNot : Sequence(String) =
168     let bfl : String = b.bug_file_loc
169     in
170         if bfl.oclIsUndefined()
171         then
172             Sequence{}
173         else
174             Sequence{bfl}
175         endif;
176 shortDescOrNot : Sequence(String) =
177     let sd : String = b.short_desc
178     in
179         if sd.oclIsUndefined()
180         then
181             Sequence{}
182         else
183             Sequence{sd}
184         endif;
185     }
186 to
187     xb : XML!Element (
188         name <- 'bug',
189         children <-
190 Sequence{er,bi,ex,ub,bs,res,p,pri,v,rp,at,dts,c,rep,tarMl,bsvy,cts,qac,sw,os,bfl,sd
191 ,
192         b.keywords->collect(e | thisModule.resolveTemp(e, 'k')),
193         b.dependson->collect(e | thisModule.resolveTemp(e, 'd')),
194         b.blocks->collect(e | thisModule.resolveTemp(e, 'b')),
195         b.cc->collect(e | thisModule.resolveTemp(e, 'c')),
196         Sequence{b.long_desc}->collect(e | thisModule.resolveTemp(e,
197 'ld'))->first(),
198         b.attachment->collect(e | thisModule.resolveTemp(e, 'a'))
199     }
200 ),
201 er : distinct XML!Attribute foreach(errorVal in errorOrNot) (
202     name <- 'error',
203     value <- thisModule.getStringErrorValue(errorVal)

```

```
204     ),
205     bi : XML!Element (
206         name <- 'bug_id',
207         children <- Sequence{biv}
208     ),
209     biv : XML!Text (
210         value <- b.bug_id
211     ),
212     ex : XML!Element (
213         name <- 'exporter',
214         children <- Sequence{exv}
215     ),
216     exv : XML!Text (
217         value <- b.exporter
218     ),
219     ub : XML!Element (
220         name <- 'urlbase',
221         children <- Sequence{ubv}
222     ),
223     ubv : XML!Text (
224         value <- b.urlbase
225     ),
226     bs : XML!Element (
227         name <- 'bug_status',
228         children <- Sequence{bsv}
229     ),
230     bsv : XML!Text (
231         value <- thisModule.getStringBugStatusValue(b.bug_status)
232     ),
233     res : distinct XML!Element foreach(resolutionVal in resolutionOrNot) (
234         name <- 'resolution',
235         children <- Sequence{resv}
236     ),
237     resv : distinct XML!Text foreach(resolutionVal in resolutionOrNot) (
238         value <- thisModule.getStringResolutionValue(resolutionVal)
239     ),
240     p : XML!Element (
241         name <- 'product',
242         children <- Sequence{pv}
243     ),
244     pv : XML!Text (
245         value <- b.product
246     ),
247     pri : XML!Element (
248         name <- 'priority',
249         children <- Sequence{priv}
250     ),
251     priv : XML!Text (
252         value <- thisModule.getStringPriorityValue(b.priority)
253     ),
254     v : XML!Element (
255         name <- 'version',
256         children <- Sequence{vv}
257     ),
258     vv : XML!Text (
259         value <- b.version
260     ),
261     rp : XML!Element (
262         name <- 'rep_platform',
263         children <- Sequence{rpv}
264     ),
265     rpv : XML!Text (
```

```
266     value <- thisModule.getStringRepPlatformValue(b.rep_platform)
267   ),
268   at : XML!Element (
269     name <- 'assigned_to',
270     children <- Sequence{atv}
271   ),
272   atv : XML!Text (
273     value <- b.assigned_to
274   ),
275   dts : XML!Element (
276     name <- 'delta_ts',
277     children <- Sequence{dtsv}
278   ),
279   dtsv : XML!Text (
280     value <- b.delta_ts
281   ),
282   c : XML!Element (
283     name <- 'component',
284     children <- Sequence{cv}
285   ),
286   cv : XML!Text (
287     value <- b.component
288   ),
289   rep : XML!Element (
290     name <- 'reporter',
291     children <- Sequence{repv}
292   ),
293   repv : XML!Text (
294     value <- b.reporter
295   ),
296   tarMl : distinct XML!Element foreach(targetMilestoneVal in
targetMilestoneOrNot) (
297     name <- 'target_milestone',
298     children <- Sequence{tarMlv}
299   ),
300   tarMlv : distinct XML!Text foreach(targetMilestoneVal in targetMilestoneOrNot)
301 (
302   value <- targetMilestoneVal
303 ),
304   bsvy : XML!Element (
305     name <- 'bug_severity',
306     children <- Sequence{bsvv}
307   ),
308   bsvv : XML!Text (
309     value <- thisModule.getStringSeverityValue(b.bug_severity)
310   ),
311   cts : XML!Element (
312     name <- 'creation_ts',
313     children <- Sequence{ctsv}
314   ),
315   ctsv : XML!Text (
316     value <- b.creation_ts
317   ),
318   qac : distinct XML!Element foreach(qaContactVal in qaContactOrNot) (
319     name <- 'qa_contact',
320     children <- Sequence{qacv}
321   ),
322   qacv : distinct XML!Text foreach(qaContactVal in qaContactOrNot) (
323     value <- qaContactVal
324   ),
325   sw : distinct XML!Element foreach(statusWhiteboardVal in
statusWhiteboardOrNot) (
```

```
328     name <- 'status_whiteboard',
329     children <- Sequence{swv}
330   ),
331   swv : distinct XML!Text foreach(statusWhiteboardVal in statusWhiteboardOrNot)
332 (
333     value <- statusWhiteboardVal
334   ),
335   osv : XML!Element (
336     name <- 'op_sys',
337     children <- Sequence{osv}
338   ),
339   osv : XML!Text (
340     value <- thisModule.getStringOperatingSystemValue(b.op_sys)
341   ),
342   bfl : distinct XML!Element foreach(bugFileLocVal in bugFileLocOrNot) (
343     name <- 'bug_file_loc',
344     children <- Sequence{bflv}
345   ),
346   bflv : distinct XML!Text foreach(bugFileLocVal in bugFileLocOrNot) (
347     value <- bugFileLocVal
348   ),
349   sd : distinct XML!Element foreach(shortDescVal in shortDescOrNot) (
350     name <- 'short_desc',
351     children <- Sequence{sdv}
352   ),
353   sdv : distinct XML!Text foreach(shortDescVal in shortDescOrNot) (
354     value <- shortDescVal
355   )
356 }
357
358
359 -- Rule 'Keywords2Keywords'
360 -- This rule generates the "keywords" XML element
361 -- from the "Keywords" element
362 rule Keywords2Keywords {
363   from
364     bk : Bugzilla!Keywords
365
366   to
367     k : XML!Element (
368       name <- 'keywords',
369       children <- Sequence{kv}
370     ),
371     kv : XML!Text (
372       value <- bk.value
373     )
374 }
375
376 -- Rule 'Dependson2Dependson'
377 -- This rule generates the "dependson" XML element
378 -- from the "Dependson" element
379 rule Dependson2Dependson {
380   from
381     bdo : Bugzilla!Dependson
382
383   to
384     d : XML!Element (
385       name <- 'dependson',
386       children <- Sequence{dv}
387     ),
388     dv : XML!Text (
389       value <- bdo.value
```



```
390     )
391 }
392
393 -- Rule 'Blocks2Blocks'
394 -- This rule generates the "blocks" XML element
395 -- from the "Blocks" element
396 rule Blocks2Blocks {
397     from
398         bb : Bugzilla!Blocks
399
400     to
401         b : XML!Element (
402             name <- 'blocks',
403             children <- Sequence{bv}
404         ),
405         bv : XML!Text (
406             value <- bb.value
407         )
408     }
409
410 -- Rule 'Cc2Cc'
411 -- This rule generates the "cc" XML element
412 -- from the "Cc" element
413 rule Cc2Cc {
414     from
415         bc : Bugzilla!Cc
416
417     to
418         c : XML!Element (
419             name <- 'cc',
420             children <- Sequence{cv}
421         ),
422         cv : XML!Text (
423             value <- bc.value
424         )
425     }
426
427 -- Rule 'LongDesc2LongDesc'
428 -- This rule generates the "long_desc" XML element
429 -- from the "LongDesc" element
430 rule LongDesc2LongDesc {
431     from
432         bld : Bugzilla!LongDesc
433
434     to
435         ld : XML!Element (
436             name <- 'long_desc',
437             children <- Sequence{w,bw,t}
438         ),
439         w : XML!Element (
440             name <- 'who',
441             children <- Sequence{wv}
442         ),
443         wv : XML!Text (
444             value <- bld.who
445         ),
446         bw : XML!Element (
447             name <- 'bug_when',
448             children <- Sequence{bwv}
449         ),
450         bwv : XML!Text (
451             value <- bld.bug_when
```

```
452     ),
453     t : XML!Element (
454         name <- 'thetext',
455         children <- Sequence{tv}
456     ),
457     tv : XML!Text (
458         value <- bld.thetext
459     )
460 }
461
462 -- Rule 'Attachment2Attachment'
463 -- This rule generates the "attachment" XML element
464 -- from the "Attachment" element
465 rule Attachment2Attachment {
466     from
467         ba : Bugzilla!Attachment
468
469     to
470         a : XML!Element (
471             name <- 'attachment',
472             children <- Sequence{i,de,dc,t,da}
473         ),
474         i : XML!Element (
475             name <- 'id',
476             children <- Sequence{iv}
477         ),
478         iv : XML!Text (
479             value <- ba.id
480         ),
481         de : XML!Element (
482             name <- 'date',
483             children <- Sequence{dev}
484         ),
485         dev : XML!Text (
486             value <- ba.date
487         ),
488         dc : XML!Element (
489             name <- 'desc',
490             children <- Sequence{dcv}
491         ),
492         dcv : XML!Text (
493             value <- ba.desc
494         ),
495         t : XML!Element (
496             name <- 'type',
497             children <- Sequence{tv}
498         ),
499         tv : XML!Text (
500             value <- ba.type
501         ),
502         da : XML!Element (
503             name <- 'data',
504             children <- Sequence{dav}
505         ),
506         dav : XML!Text (
507             value <- ba.data
508         )
509 }
```

	<b>ATL TRANSFORMATION EXAMPLE</b>	Contributor Hugo Brunelière
	<b>Software Quality Control to Bugzilla file</b>	Date 02/08/2005

### 1.4.3. XML2BugzillaText

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 *BugzillaFile* 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 *BugzillaFile* helper.

The *BugzillaFile* helper returns a string which is composed of the required XML file's header and of the Bugzilla 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().BugzillaFile().writeTo('C:\\... path to be completed before using the
4  transformation ...\\BugzillaXMLfileExample.xml');
5
6  helper context XML!Root def: BugzillaFile() : String =
7      '<?xml version="1.0"?>' + '\n' + self.toString2('');
8
9
10 helper context XML!Element def: toString2(indent : String) : String =
11     let na : Sequence(XML!Node) =
12         self.children->select(e | not e.oclIsKindOf(XML!Attribute)) in
13     let a : Sequence(XML!Node) =
14         self.children->select(e | e.oclIsKindOf(XML!Attribute)) in
15     indent + '<' + self.name +
16     a->iterate(e; acc : String = '' |
17         acc + ' ' + e.toString2()
18     ) +
19     if na->size() > 0 then
20         '>'
21         + na->iterate(e; acc : String = '' |
22             acc +
23             if e.oclIsKindOf(XML!Text) then
24                 ''
25             else
26                 '\r\n'
27             endif
28             + e.toString2(indent + ' ')
29         ) +
30         if na->first().oclIsKindOf(XML!Text) then
31             '</' + self.name + '>'
32         else
33             '\r\n' + indent + '</' + self.name + '>'
34         endif
35     else
36         '/>'

```



**ATL  
TRANSFORMATION EXAMPLE**

Contributor  
Hugo Brunelière

**Software Quality Control to Bugzilla file**

Date 02/08/2005

```
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;
```

	<b>ATL TRANSFORMATION EXAMPLE</b>	Contributor Hugo Brunelière
	<b>Software Quality Control to Bugzilla file</b>	Date 02/08/2005

## 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

```

	<b>ATL TRANSFORMATION EXAMPLE</b>	Contributor Hugo Brunelière
	<b>Software Quality Control to Bugzilla file</b>	Date 02/08/2005

```

-- @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;

}

```

	<b>ATL TRANSFORMATION EXAMPLE</b>	Contributor Hugo Brunelière
	<b>Software Quality Control to Bugzilla file</b>	Date 02/08/2005

## II. Bugzilla metamodel in KM3 format

```

-- @name Bugzilla
-- @version 1.0
-- @domains Software bug tracking
-- @authors Hugo Bruneliere (hugo.bruneliere@gmail.com)
-- @date 2005/07/07
-- @description This metamodel describes the structure used by Bugzilla to
import/export bugs in XML format. Bugzilla is a free "Defect Tracking System" or
"Bug-Tracking System" which allows individual or groups of developers to keep track
of outstanding bugs in their product effectively.
-- @see bugzilla.dtd, http://www.mantisbt.org/mantis/view.php?id=4024 at the
bottom of the page

package Bugzilla {


    -- @begin Bugzilla special types

    --@comment Defines the type of error for a bug
    enumeration ErrorType {
        literal et_null;
        literal et_NotFound;
        literal et_NotPermitted;
        literal et_InvalidBugId;
    }

    -- @comment Defines the type of severity for a bug.
    enumeration SeverityType{
        literal st_null;
        -- @comment Blocks development and/or testing work.
        literal st_blocker;
        -- @comment Crashes, loss of data, severe memory leak.
        literal st_critical;
        -- @comment Loss of function.
        literal st_major;
        -- @comment A normal problem.
        literal st_normal;
        -- @comment Loss of function, or other problem where easy workaround is
present.
        literal st_minor;
        -- @comment Cosmetic problem.
        literal st_trivial;
        -- @comment Request for enhancement.
        literal st_enhancement;
    }

    -- @comment Defines the type of status for a bug.
    enumeration StatusType{
        literal st_null;
        -- @comment A new bug, when a product has voting.
        literal st_UNCONFIRMED;
        -- @comment Recently added or confirmed.
        literal st_NEW;
        -- @comment Has been assigned.
        literal st_ASSIGNED;
        -- @comment Was once resolved but has been reopened
        literal st_REOPENED;
        -- @comment Has been resolved (e.g. fixed, deemed unfixable, etc.), see the
"ResolutionType".

```

	<b>ATL TRANSFORMATION EXAMPLE</b>	Contributor Hugo Brunelière
	<b>Software Quality Control to Bugzilla file</b>	Date 02/08/2005

```

    literal st_RESOLVED;
    -- @comment The resolution has been approved by Quality Assurance.
    literal st_VERIFIED;
    -- @comment Over and done with.
    literal st_CLOSED;
}

-- @comment Defines the type of operating system on which a bug was observed.
enumeration OperatingSystemType{
    literal ost_null;
    literal ost_all;
    literal "ost_Windows 3.1";
    literal "ost_Windows 95";
    literal "ost_Windows 98";
    literal "ost_Windows ME";
    literal "ost_Windows 2000";
    literal "ost_Windows NT";
    literal "ost_Windows XP";
    literal "ost_Windows Server 2003";
    literal "ost_MacSystem 7";
    literal "ost_MacSystem 7.5";
    literal "ost_MacSystem 7.6.1";
    literal "ost_MacSystem 8.0";
    literal "ost_MacSystem 8.5";
    literal "ost_MacSystem 8.6";
    literal "ost_MacSystem 9.x";
    literal "ost_Mac OS X 10.0";
    literal "ost_Mac OS X 10.1";
    literal "ost_Mac OS X 10.2";
    literal "ost_Mac OS X 10.3";
    literal ost_Linux;
    literal "ost_BDS/OS";
    literal ost_FreeBSD;
    literal ost_NetBSD;
    literal ost_OpenBSD;
    literal ost_AIX;
    literal ost_BeOS;
    literal "ost_HP-UX";
    literal ost_IRIX;
    literal ost_Neutrino;
    literal ost_OpenVMS;
    literal "ost_OS/2";
    literal "ost_OSF/1";
    literal ost_Solaris;
    literal ost_SunOS;
    literal ost_other;
}

-- @comment Defines the type of priority for a bug.
enumeration PriorityType{
    literal pt_null;
    -- @comment Most Urgent
    literal pt_P1;
    literal pt_P2;
    literal pt_P3;
    literal pt_P4;
    -- @comment Least Urgent
    literal pt_P5;
}

-- @comment Defines the type of platform on which a bug was reported.
enumeration ReportedPlatformType{

```





ATL  
TRANSFORMATION EXAMPLE

Contributor  
Hugo Brunelière

Software Quality Control to Bugzilla file

Date 02/08/2005

```
    literal rpt_null;
    literal rpt_all;
    literal rpt_DEC;
    literal rpt_HP;
    literal rpt_Macintosh;
    literal rpt_PC;
    literal rpt_SGI;
    literal rpt_Sun;
    literal rpt_other;
}

-- @comment Defines the bug's type of resolution
enumeration ResolutionType{
    literal rt_null;
    -- @comment The bug has been fixed.
    literal rt_FIXED;
    -- @comment The problem described is not a bug.
    literal rt_INVALID;
    -- @comment This bug will never be fixed.
    literal rt_WONTFIX;
    -- @comment This bug will not be fixed in this version.
    literal rt_LATER;
    -- @comment This bug probably won't be fixed in this version.
    literal rt_REMIND;
    -- @comment This is a duplicate of an existing bug.
    literal rt_DUPLICATE;
    -- @comment This bug could not be reproduced.
    literal rt_WORKSFORME;
    -- @comment This bug has been moved to another (Bugzilla) database.
    literal rt_MOVED;
}

-- @end Bugzilla special types

-- @begin Bugzilla structure

class BugzillaRoot {
    attribute version : String;
    attribute urlbase : String;
    attribute maintainer : String;
    attribute exporter[0-1] : String;

    reference bugs[1-]* ordered container : Bug oppositeOf bug_bugzillaRoot;
}


class Bug {
    reference bug_bugzillaRoot : BugzillaRoot oppositeOf bugs;

    attribute error[0-1] : ErrorType;

    -- @comment The identification number of the bug
    attribute bug_id : String;

    attribute exporter : String;
    attribute urlbase : String;

    -- @comment The current status of this bug.
    attribute bug_status : StatusType;
    -- @comment The resolution's level of this bug.
    attribute resolution[0-1] : ResolutionType;
    -- @comment The name of the software product
```

	<b>ATL TRANSFORMATION EXAMPLE</b>	Contributor Hugo Brunelière
	<b>Software Quality Control to Bugzilla file</b>	Date 02/08/2005

```

attribute product : String;
-- @comment The bug's priority.
attribute priority : PriorityType;
-- @comment The name of the version of the software product or component
attribute version : String;
-- @comment The type of the platform on which the bug was reported.
attribute rep_platform : ReportedPlatformType;
-- @comment The current owner of this bug.
attribute assigned_to : String;
-- @comment The time at which information about this bug changing was last
emailed to the cc list.
attribute delta_ts : String;
-- @comment The name of a software product's component.
attribute component : String;
-- @comment The user who has reported this bug.
attribute reporter : String;
-- @comment The current target milestone for this bug.
attribute target_milestone[0-1] : String;
-- @comment The evaluation of this bug's severity.
attribute bug_severity : SeverityType;
-- @comment The times of the bug's creation.
attribute creation_ts : String;
-- @comment The "quality assurance" contact for this bug.
attribute qa_contact[0-1] : String;
-- @comment Some comments about the status of this bug.
attribute status_whiteboard[0-1] : String;
-- @comment The operating system on which this bug was observed.
attribute op_sys : OperatingSystemType;
-- @comment A URL which points to more information about the bug.
attribute bug_file_loc[0-1] : String;
-- @comment A short textual description of the bug
attribute short_desc[0-1] : String;
-- @comment A list of keywords relating to this bug.
reference keywords[*] ordered container : Keywords;
-- @comment Represents the bugs from which this bug depends.
reference dependson[*] ordered container : Dependson;
-- @comment ???
reference blocks[*] ordered container : Blocks;
-- @comment Represents the users who have asked to receive email when a bug
changes.
reference cc[*] ordered container : Cc;
-- @comment A long description about the bug
reference long_desc[0-1] container : LongDesc;
-- @comment Represents the attachments associated to this bug.
reference attachment[*] ordered container : Attachment;

}

abstract class StringElt {
  attribute value : String;
}

-- @comment Defines a keyword relative to a bug
class Keywords extends StringElt {}

-- @comment Defines a bug from which an other bug depends
class Dependson extends StringElt {}

-- @comment Defines ???
class Blocks extends StringElt {}

-- @comment Defines a user who have asked to receive an email when a bug changes.

```



**ATL  
TRANSFORMATION EXAMPLE**

Contributor  
Hugo Brunelière

**Software Quality Control to Bugzilla file**

Date 02/08/2005

```
class Cc extends StringElt {}

-- @comment Defines a long description about the bug
class LongDesc {
  attribute who : String;
  attribute bug_when : String;
  attribute thetext : String;
}


-- @comment Defines an attachment associated to a bug.
class Attachment {
  attribute id : String;
  attribute date : String;
  attribute desc : String;
  attribute type : String;
  attribute data : String;
}

-- @end Bugzilla structure
}

package PrimitiveTypes {

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

}
```

	<b>ATL TRANSFORMATION EXAMPLE</b>	Contributor Hugo Brunelière
	<b>Software Quality Control to Bugzilla file</b>	Date 02/08/2005

### 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 particular 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;
}
```

	<b>ATL TRANSFORMATION EXAMPLE</b>	Contributor Hugo Brunelière
	<b>Software Quality Control to Bugzilla file</b>	Date 02/08/2005

---

## References

- [1] Bugzilla official site, <http://www.bugzilla.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](http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/subprojects/ATL/ATL_examples/MicrosoftOfficeExcel2SoftwareQualityControl/ExampleMicrosoftOfficeExcel2SoftwareQualityControl%5Bv00.01%5D.pdf)
- [3] bugzilla.dtd, 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