Bugs - Bug #1450

comment conversion fail

06/15/2012 10:53 AM - Ovidiu Maxiniuc

Status:	WIP	**Start date:**	06/15/2012
Priority:	Normal	**Due date:**	
Assignee:	Ovidiu Maxiniuc	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

History

#1 - 07/13/2012 11:55 AM - Greg Shah

- Description updated

- Assignee changed from Greg Shah to Ovidiu Maxiniuc

- Status changed from New to WIP

- Subject changed from Comment conversion fail to comment conversion fail

GES: I moved this here from the "Description" field in the original issue. OM originally wrote this.

Sometimes, when converting a 4GL program into java the comments cause the generated java source broken.

Attached are the important files needed to reproduce the bug. I noticed that the problem seems to be the comments at the beginning of an included file.

As soon as the first line
/*aaa*/
is removed from the 'chrdat.i', the conversion process is successful.
Also configuring the conversion process to drop the comments will also fix the problem.
I did not observed if other comments from the main and included files are causing the conversion to fail.

#2 - 07/13/2012 12:00 PM - Greg Shah

Some thoughts:

1. I think it is very unlikely that this has anything to do with the comment being inside an include file. It is possible, but is not likely. Try to recreate the problem without the include. This can easily be done by just using the calendar.p.cache file that is created from the conversion process. It is likely the placement of the comment in the fully pre-processed code that is the issue, rather than the fact that the comment came from an include file.

2. Strip the final calendar.p.cache source code down to the minimum that recreates the problem.

3. With comment processing, the trickiest part is the determination of which converted code the comment should be attached to. In other words, where should that comment emit? This is dependent upon TRPL code that inspects the 4GL AST nodes near the comment, to assess what the comment is commenting on. Then based on this assessment, the TRPL rules must calculate an attachment point in the Java AST. It is most likely this TRPL code which is just lacking some rules for the specific case at hand.

Look carefully at rules/convert/comments.rules

I originally wrote that code. Ask me any questions that you have.

#3 - 07/19/2012 10:21 AM - Ovidiu Maxiniuc

I observed that (sometimes) the antlr fails to parse correctly progress code if the inline comment is not separated by a space.
Is this a known issue ?
Ex:

```
        define var b as char init "b".
        b = "a" + b./*this will not be parsed correctly*/
```

Adding an white space after . will remove the issue.

#4 - 07/19/2012 10:24 AM - Ovidiu Maxiniuc

If a comment is written on the last line of .p file (I am not sure if this is the cause, must double-check) the antlr lexer will add an additional <DOT> in the output (which does not have any correspondent into the source file).
However, this will not affect the converter because the parser will drop <DOT> tokens when analyzing the statements.

#5 - 07/19/2012 10:37 AM - Ovidiu Maxiniuc

Can you give me some ideas about how can I analyze/debug dynamic loaded classes from rules ?
At the moment I do it manually, using directly the .rules sources and sometimes I have to effectively search through some files including (imported and default workers, common .rules files).

If the case I can put a breakpoint in helper/workers if I see a call to one from a rule/action. However, I can only see class name (ex: CE001) and a call to execute() method.

Also, in Rule.resolver.variables I can search for some variables.

This is normal, but I wonder if you have some more tricks here.

I saw that there are traces that can be enabled by altering the debug level, I never tried yet, but it looks to me very verbosely. It should be possible to call some print(String) function from a rule, maybe it is already implemented ?

#6 - 07/19/2012 10:41 AM - Ovidiu Maxiniuc

At this time I believe that the bug is somewhere at the later steps, at "convert/brew" iteration.

#7 - 07/19/2012 10:47 AM - Greg Shah

This is in regard to the problem:

b = "a" + b./* comment failure without whitespace */

Progress 4GL lexing and parsing is quite complex. Their design included many ambiguities and conflicts that are resolved by using some undocumented rules that are implemented deep in the Progress compiler. The lexing and parsing of the "." DOT character is especially troublesome. Progress uses the DOT for many different purposes. They require the DOT to be followed by whitespace in some cases to allow their lexer/parser to resolve the ambiguity (I am guessing here at the intention since I have never actually seen their lexing/parsing code, but I know that it helped me out when I was writing the code <g>).

However, in this specific case, I think this may be a bug. Until we have access to a 4GL dev environment, I can't test this specific case. But based on my review of the com/goldencode/p2j/uast/progress.g, I don't see that we intended this behavior.

Please open a new issue for this. Add my comments in this history entry plus the following information to the problem description:

1. What exactly is the failure? If there is a stack trace or are error messages... please add them. I wonder if this is a failure in the preprocessor or if this is really a problem in the lexer/parser. On a quick glance, I don't see an obvious problem in the parser.

2. Under what circumstances does this case NOT fail? Some things to try:
- whitespace follows the comment
- a standalone expression follows the comment
- an assignment follows the comment
- a language statement follows the comment
- the assignment of b = "a" + b. is changed to be a standalone expression (b. is legal enough)
- the assignment of b = "a" + b. is changed to be a language statement

#8 - 07/19/2012 10:50 AM - Greg Shah

This is in regard to history record #4 above about the extra DOT added to the end of a file.

The additional DOT token added in the preprocessor at the end of a file is done intentionally. Based on our testing of how Progress works (and having seen many working customer programs that do this), we found that Progress does not require you to "properly" terminate your last line of code. There is always an implicit DOT there. We implement this "feature" by having the preprocessor add this to the output stream.

#9 - 07/19/2012 11:19 AM - Greg Shah

This is in regard to your history entry #5 above, looking for ideas on debugging TRPL.

1. A quick and dirty way to get output from specific points in a rule is to add temporary print statements. Use printf() and printfln() to get output to the console. sprintf() and fprintf() can also be used. All of these are found in pattern/CommonAstSupport.java. Example code:

```
<action>printf('The token type of the current node is %d.\n', type)</action>
```

2. The dynamically loaded classes (like CE001) are an artifact of the current approach to TRPL implementation. We dynamically generate a class for every TRPL expression and then store that class in memory. There is no Java "source" for it, we create the bytecode directly from the expression syntax. If your debugger can operate at the bytecode level, you could set a breakpoint in the bytecode directly, but that is almost never needed. We would only do that if we were debugging our expression "compiler" itself. The best thing to do is to set breakpoints in Java code (maybe in a worker, maybe in some supporting class) that can be known to be triggered near the point of the failure. If needed, add a "throwaway" <rule></rule> that has a very specific expression that only evaluates to true in the specific case you care about. Then inside that rule, place an action that gets you into some Java code that you control. Then set your breakpoint in that code. For example:

```
<rule>expression that detects the failure condition goes here
   <action>myVar = create("MyDebuggingClassHelper")</action>
   <action>myVar.myHelper(parm1, parm2, parm3)</action>
</rule>
```

In this case, you would have had to add a variable definition for myVar and the MyDebuggingClassHelper would have to be found in the classpath OR you can fully qualify it in the create() like com.goldencode.p2j.util.MyDebuggingClassHelper. You could set your breakpoint in the myHelper() method.

#10 - 07/19/2012 11:21 AM - Greg Shah

Please post the simplified 4GL source that you are working with. You can just put the code into the history entry using pre tags if it is small enough.

#11 - 07/20/2012 12:00 PM - Ovidiu Maxiniuc

Even though I though this is more complicated (even related to included files), the cases in which this bug occurs are rather simple. I identified two cases for now, but I will continue to look for other.

1. A comment before the first term:

```
b = /*zero*/ "1" /*one*/ + "2" /*two*/.
```

The /*zero*/ comment will be treated as one of the terms of the concatenation and an extra , after it.
The solution would be that in convert/brew.rules:651:

```
(type == method_call and nextChildIndex > 1)
```

to add some constraints to skip the , if first child is a STAR_COMMENT.

2. The case when the comment occurs before assign operator:

```
DEFINE VAR a AS int.
a /*zero*/ = 0.
```

The assignment (=) will be replaced in java source with comma (,) operator, also because the assign method call sees the comment as an extra term.

#12 - 07/23/2012 11:23 AM - Ovidiu Maxiniuc

I managed to convert simple test codes into java, both above cases.

I encountered difficulties when adding extra rules. One particularly annoying cause was the CommonAstSupport.Library.childAt(long type,long index) always seems to returned false. I replaced it with this.getChildAt(0).type == star_comment and things went smooth. However, later I realized my really stupid mistake: I was switching the parameters. I kept the first working code as I believe that there is no difference from the performance point of view.

As an additional note I observed that some comments are duplicated, some are slightly moved and other dropped.

#13 - 07/24/2012 09:09 AM - Greg Shah

In order to evaluate the correctness of your solution, please do the following:

1. For each failing case above, please post the snippet of the AST (from the .p.ast XML file) that corresponds to the failing line. Make sure that the entire subtree associated with the failing line of code is posted. Post them in a PRE tag in the issue history.

2. Is it true that only the /*zero*/ comment is a problem? The /*one*/ and /*two*/ comments work OK?

#14 - 07/25/2012 06:28 AM - Ovidiu Maxiniuc

1. Here is the ast for 1st issue:

```
    <ast col="0" id="1327144894491" line="0" text="assignment" type="ASSIGNMENT">
      <ast col="3" id="1327144894492" line="4" text="=" type="ASSIGN">
        <annotation datatype="java.lang.Long" key="peerid" value="1335734829089"/>
        <ast col="1" id="1327144894497" line="4" text="b" type="VAR_CHAR">
          <annotation datatype="java.lang.Long" key="oldtype" value="2332"/>
          <annotation datatype="java.lang.Long" key="refid" value="1327144894479"/>
          <annotation datatype="java.lang.Long" key="peerid" value="1335734829090"/>
        </ast>
        <ast col="0" id="1327144894502" line="0" text="expression" type="EXPRESSION">
          <ast col="26" id="1327144894503" line="4" text="+" type="PLUS">
            <annotation datatype="java.lang.Long" key="peerid" value="1335734829091"/>
            <ast col="14" id="1327144894508" line="4" text="&amp;quot;1&amp;quot;" type="STRING">
              <annotation datatype="java.lang.Long" key="peerid" value="1335734829092"/>
            </ast>
            <ast col="28" id="1327144894513" line="4" text="&amp;quot;2&amp;quot;" type="STRING">
              <annotation datatype="java.lang.Long" key="peerid" value="1335734829093"/>
            </ast>
          </ast>
        </ast>
      </ast>
    </ast>
```

and the second one:

```
    <ast col="0" id="1327144894516" line="0" text="assignment" type="ASSIGNMENT">
      <ast col="12" id="1327144894517" line="5" text="=" type="ASSIGN">
        <annotation datatype="java.lang.Long" key="peerid" value="1335734829094"/>
        <ast col="1" id="1327144894522" line="5" text="a" type="VAR_INT">
          <annotation datatype="java.lang.Long" key="oldtype" value="2332"/>
          <annotation datatype="java.lang.Long" key="refid" value="1327144894467"/>
          <annotation datatype="java.lang.Long" key="peerid" value="1335734829095"/>
        </ast>
        <ast col="0" id="1327144894527" line="0" text="expression" type="EXPRESSION">
          <ast col="14" id="1327144894528" line="5" text="0" type="NUM_LITERAL">
            <annotation datatype="java.lang.Long" key="peerid" value="1335734829096"/>
          </ast>
        </ast>
```

```
      </ast>
    </ast>
```

They look fine to me. I followed the process just up to .jast, and tags were also looking fine.
The problem was that the comment was interpreted as a normal parameter to a method call when it should normally be ignored.

2. Yes, I can confirm that both /*zero*/ comments are a problem and /*one*/ and /*two*/ comments work OK. Here is the generated code:

```
   /*one*/
   b.assign(concat( /*zero*/ , "1" /*one*/ , "2" /*two*/));
   /*zero*/
   a /*zero*/, 0);
```

#15 - 07/25/2012 08:47 AM - Greg Shah

Good. Please add the relevant snippets of the .jast file to the history. I need to review those to understand how safe your fix is.

#16 - 07/25/2012 11:43 AM - Ovidiu Maxiniuc

After further testing I realized that double star-comment weren't handled correctly, ie. code like:

```
a /*zero*/ /*another zero*/ = 0.
```

were still broken in java source.

So I had to re-count non-comment parameters for method-call, static-method-call and constructor and act on the new index. The xml code is a bit more complex now, having a couple of <while> tags.

#17 - 07/25/2012 11:49 AM - Ovidiu Maxiniuc

Back with your answers of #note-15:
a /*zero*/ = 0.

```
 <ast col="0" id="1335734829089" line="0" text="assign" type="METHOD_CALL">
   <annotation datatype="java.lang.Long" key="peerid" value="1327144894492"/>
   <ast col="0" id="1335734829090" line="0" text="a" type="REFERENCE">
     <annotation datatype="java.lang.Long" key="peerid" value="1327144894497"/>
   </ast>
   <ast col="0" id="1335734829140" line="0" text="zero" type="STAR_COMMENT">
     <annotation datatype="java.lang.Boolean" key="non-ws-b4" value="true"/>
     <annotation datatype="java.lang.Boolean" key="non-ws-after" value="false"/>
   </ast>
   <ast col="0" id="1335734829091" line="0" text="0" type="NUM_LITERAL">
     <annotation datatype="java.lang.Long" key="peerid" value="1327144894505"/>
   </ast>
```

```
</ast>
```

and b = /*zero*/ "1" /*one*/ + "2" /*two*/.

```
<ast col="0" id="1335734829092" line="0" text="assign" type="METHOD_CALL">
  <annotation datatype="java.lang.Long" key="peerid" value="1327144894508"/>
  <ast col="0" id="1335734829093" line="0" text="b" type="REFERENCE">
    <annotation datatype="java.lang.Long" key="peerid" value="1327144894513"/>
  </ast>
  <ast col="0" id="1335734829094" line="0" text="concat" type="STATIC_METHOD_CALL">
    <annotation datatype="java.lang.Long" key="peerid" value="1327144894517"/>
    <ast col="0" id="1335734829141" line="0" text="zero" type="STAR_COMMENT">
      <annotation datatype="java.lang.Boolean" key="non-ws-b4" value="true"/>
      <annotation datatype="java.lang.Boolean" key="non-ws-after" value="true"/>
    </ast>
    <ast col="0" id="1335734829095" line="0" text="1" type="STRING">
      <annotation datatype="java.lang.Long" key="peerid" value="1327144894522"/>
    </ast>
    <ast col="0" id="1335734829143" line="0" text="one" type="STAR_COMMENT">
      <annotation datatype="java.lang.Boolean" key="non-ws-b4" value="true"/>
      <annotation datatype="java.lang.Boolean" key="non-ws-after" value="true"/>
    </ast>
    <ast col="0" id="1335734829096" line="0" text="2" type="STRING">
      <annotation datatype="java.lang.Long" key="peerid" value="1327144894527"/>
    </ast>
    <ast col="0" id="1335734829144" line="0" text="two" type="STAR_COMMENT">
      <annotation datatype="java.lang.Boolean" key="non-ws-b4" value="true"/>
      <annotation datatype="java.lang.Boolean" key="non-ws-after" value="false"/>
    </ast>
  </ast>
</ast>
```

#18 - 07/25/2012 12:20 PM - Greg Shah

Please post the snippets of code (inline in the history) that you wrote for the solution.

#19 - 07/26/2012 06:21 AM - Ovidiu Maxiniuc

To fix the issue, add the following in convert/brew.xml,
line 117:

```
   <variable name="methodCallParamCounter"  type="java.lang.Integer" />
   <variable name="methodCallParamCounter2" type="java.lang.Integer" />
   <variable name="methodCallParamFound"    type="java.lang.Boolean" />
```

line 633:

```
         <rule>type == static_method_call or type == constructor
            <!-- use comma only after non-comment childrens -->
            <rule>this.getChildAt(nextChildIndex - 1).type != star_comment
               <!-- check if this is the last child having type != star_comment -->
               <action>methodCallParamCounter = nextChildIndex</action>
               <action>methodCallParamFound = false</action>
               <while>methodCallParamCounter &lt; numImmediateChildren
                  <rule>this.getChildAt(methodCallParamCounter).type != star_comment
                     <action>methodCallParamFound = true</action>
                  </rule>
                  <action>methodCallParamCounter = methodCallParamCounter + 1</action>
               </while>

               <rule>methodCallParamFound
                  <!-- if not the last parameter, add the comma or space-->
                  <rule>type == static_method_call and text == 'System.out.println'
                    <action on="true" >fprintf(fid, ' + \" \" + ')</action>
                    <action on="false">fprintf(fid, ', ')</action>
                  </rule>
               </rule>
            </rule>
         </rule>

         <rule>type == method_call
            <rule>this.getChildAt(nextChildIndex).type != star_comment
               <!--if passed 2nd non-comment then put the name of the method
                  then use comma only after non-comment childrens AND
                  check if this is the last child having type != star_comment-->
               <action>methodCallParamCounter = 0</action>
               <action>methodCallParamCounter2 = 0</action>
               <while>methodCallParamCounter &lt; nextChildIndex
                  <rule>this.getChildAt(methodCallParamCounter).type != star_comment
                     <action>methodCallParamCounter2 = methodCallParamCounter2 + 1</action>
                  </rule>
                  <action>methodCallParamCounter = methodCallParamCounter + 1</action>
               </while>
               <!-- now methodCallParamCounter2 contains the real
                     child index (excluding comments)-->

               <rule>methodCallParamCounter2 == 1
                  <!-- put the name of the method after 1st non-comment child
                        (this is the object on which the method is invoked)-->
                  <action>fprintf(fid, '.%s(', text)</action>
               </rule>

               <rule>methodCallParamCounter2 &gt; 1
                  <!-- put comma between other children (method parameters) -->

                  <action>methodCallParamCounter = nextChildIndex</action>
                  <action>methodCallParamFound = false</action>
                  <while>methodCallParamCounter &lt; numImmediateChildren
                     <rule>this.getChildAt(methodCallParamCounter).type != star_comment
                        <action>methodCallParamFound = true</action>
                     </rule>
                     <action>methodCallParamCounter = methodCallParamCounter + 1</action>
                  </while>

                  <rule>methodCallParamFound
                     <!-- if not the last parameter, add the comma -->
```

```
                    <action>fprintf(fid, ', ')</action>
                  </rule>
               </rule>

            </rule>
         </rule>
```

and comment out/remove the following lines 648-661 (the old code).

#20 - 07/26/2012 06:03 PM - Greg Shah

I have reviewed the proposed solution. I believe all the looping code is unnecessary. You are making things more difficult for yourself than is needed. The key here is to remember that eventually, the tree walking will traverse all nodes. So it is a rare solution that must manually walk the tree using a loop. We do it, but it is rare.

I also believe that the logic is incorrect. In order to decide on whether or not to emit a comma, you should never have to look ahead any more than nextChildIndex. You DO need to know if there has been a PRIOR parameter emitted or not. That is the key to the solution.

The idea of <next-child-rules> is that these rules only execute when moving from one child node to another child node. Consider this subtree:

STATIC_METHOD_CALL |-PARM1 |-PARM2 |-PARM3

The STATIC_METHOD_CALL parent node has 3 children, index 0 is PARM1, index 1 is PARM2 and index 2 is PARM3.

A rule in the <next-child-rules> section will get processed when the tree walking moves between PARM1 and PARM2 (nextChildIndex will be 1 at that time) and will get processed a 2nd time when the tree walking moves between PARM2 and PARM3 (nextChildIndex will be 2 at that time). These rules are perfect for deciding when to emit commas between parameters because you can easily reference both the old node using this.getChildAt(nextChildIndex - 1) and the new node using this.getChildAt(nextChildIndex).

An important point here: during the <next-child-rules> processing, the this node is the parent of the next child being processed.

In the original code, note that the part that emits the comma is protected by checking if the next child node (to which we are walking) is not a comment:

```
   <rule>this.getChildAt(nextChildIndex).type != star_comment
```

In the type == static_method_call case where we are moving from PARM1 to PARM2, this code properly checks to see if PARM2 is not a comment. If it is a comment, then no comma is emitted. And only if we get to a PARMX next child that is not a comment, do we emit a comma. Since we are in-between child nodes here, emitting the comma will show up in the output between the Java code that is emitted for those 2 child nodes.

The problem is that this code is missing a check for whether the PREVIOUS children are comments. In the simplest case (e.g. /*zero*/ parm1), something like and this.getChildAt(nextChildIndex - 1).type != star_comment would solve the problem.

But this check alone is not sufficient, because while it will solve some problems, it would also mis-process something like parm1 /*zero*/ /*one/ parm2. So the solution does need to be smarter. But I still think it can be simpler than using loops.

All we need to know is that there has been at least 1 non-comment child of our current parent. We won't emit the comma until we get to a nextChildIndex that is not a comment. And since we never get a <next-child-rules> call except on the 2nd and all subsequent children, if there is any prior non-comment children AND the nextChildIndex points to a non-comment, then we know that we MUST emit a comment. If BOTH of these conditions don't hold, then we MUST NOT emit a comma.

The simplest solution is to let the tree walking do the work for us. Create a new variable (named something like parmCounter) to be used as a counter. That variable gets incremented in a <next-child-rules> rule that checks this:

```
   <rule>this.getChildAt(nextChildIndex - 1).type != star_comment
      <action>parmCounter = parmCounter + 1</action>
   </rule>
```

This would be done OUTSIDE and ABOVE the current protection code:

```
   <rule>this.getChildAt(nextChildIndex).type != star_comment
```

Then this protection code would be modified to look like this:

```
   <rule>this.getChildAt(nextChildIndex).type != star_comment and
         parmCounter &gt; 0
```

I haven't looked carefully at all the cases inside that block, so perhaps that new parmCounter > 0 code must be placed in a more nested location. But that code should solve the problem for the static_method_call case. Perhaps the method_call case needs to be something like parmCounter > 1, to ensure that the instance reference is avoided.

The trick here is that we would need to push the current state of this var when we descend (use a <descent-rules>) from a static_method_call/method_call/constructor and reset the var to 0. Something like this:

```
   <descent-rules>
      <rule>parent.type == static_method_call or
            parent.type == method_call or
            parent.type == constructor

         <!-- save off the previous scope's data -->
         <action>addScope("parameter_processing")</action>
         <action>addDictionaryLong("parameter_processing", "parm_counter", parmCounter)</action>
         <action>parmCounter = #(long) (0)</action>

      </rule>
   </descent-rules>
```

And of course, on the ascent, the values must be popped to restore them. Something like this:

```
   <ascent-rules>
      <rule>parent.type == static_method_call or
            parent.type == method_call or
            parent.type == constructor

         <!-- restore the previous scope's data -->
         <action>parmCounter = lookupDictionaryLong("parameter_processing", "parm_counter")</action>
         <action>deleteScope("parameter_processing")</action>

      </rule>
   </ascent-rules>
```

The reason for this push/pop processing is so that method calls can be arbitrarily nested (method calls inside sub-expressions that are parameters to another method call).

Let me know if this makes sense, if it works and if you have any questions.

#21 - 07/27/2012 10:27 AM - Ovidiu Maxiniuc

That was my first approach to fix this bug. However, I discovered that (see note 16) the code can have mode than one comment in the list of method call.
Consider this:

```
b /*pre-assign*/ = /*post-assign*/
   /*pre 1st*/ "1" /*post first*/  +
   /*pre 2nd*/ "2" /*two*/ +
   /*three*/ "3" /*and */ /*so on*/ /*and on*/.
```

The .jast will look something like:

```
<ast text="assign" type="METHOD_CALL">
   <ast text="b" type="REFERENCE"/>
   <ast text="pre-assign" type="STAR_COMMENT"/>
   <ast text="concat" type="STATIC_METHOD_CALL">
      <ast text="post-assign" type="STAR_COMMENT" />
      <ast text="1" type="STRING">
      <ast text="post first" type="STAR_COMMENT">
      <ast text="2" type="STRING">
      <ast text="two" type="STAR_COMMENT">
      <ast text="3" type="STRING">
      <ast text="and on" type="STAR_COMMENT">
      <ast text="so on" type="STAR_COMMENT">
      <ast text="and " type="STAR_COMMENT">
   </ast>
</ast>
```

At this moment is very difficult (if not impossible) to put the commas where needed. I know that this is an extreme example, but as long as the Progress language allows it should be converted to a compilable java source code.
Switching compact = false settings in the p2j file configuration will make the concat a STATIC_METHOD_CALL with the same issues.

#22 - 07/27/2012 11:00 AM - Greg Shah

I believe my described solution takes this issue into account. The parmCounter would never be incremented for comments. So long as the emission of commas is protected by the parmCounter, there should not be an issue.

Look back at my previous post. Let me know where specifically you see that it doesn't handle the case you mention.

#23 - 08/16/2012 11:54 AM - Ovidiu Maxiniuc

You were right. After re-reading your post and analyzing the issue I was able to fix the issue without using a looping. I'm sure that my old implementation was correct, but it was enough slow because the children were counted each time.
The new implementation also count non-comment parameters but on-the fly, and puts a comma BEFORE the subsequent parameters skipping, of course the first one. I believe that the code is more compact end easy to understand now.

It goes like this:
On descent and ascent I use a scope in stack to save/restore counter value for nested method calls as you mentioned in comment #20 and, on next child the handling is replaced by the following code:

```
<!-- count non-comment children -->
<rule>type == static_method_call or
      type == method_call or
      type == constructor
   <action>childCounter = childCounter + 1</action>

   <rule>childCounter &gt; 1
      <!-- add comma only BEFORE 2nd or more non-comment child -->
      <action>shouldAddComma = true</action>
   </rule>

   <rule>childCounter == 2 and type == method_call
      <!-- BUT not if 1st child it's the direct object -->
      <action>shouldAddComma = false</action>
   </rule>

</rule>

<rule>shouldAddComma
   <rule>type == static_method_call
      <rule>text == 'System.out.println'
         <action on="true" >fprintf(fid, ' + \" \" + ')</action>
         <action on="false">fprintf(fid, ', ')</action>
      </rule>
   </rule>

   <rule on="true">type == method_call and childCounter &gt; 2
      <action>fprintf(fid, ', ')</action>
   </rule>

   <rule on="false">type == method_call and childCounter == 2
      <action>fprintf(fid, '.%s(', text)</action>
   </rule>

   <rule>type == constructor
      <action>fprintf(fid, ', ')</action>
   </rule>
</rule>
```

It is important to mention here how the counter is initialized/resetted after pushing its value into stack:

```
<!-- reset parameter configuration for future nested list(s) -->
<action>shouldAddComma = false</action>
<rule>this.getChildAt(0).type == star_comment
   <!-- count the first child if not a comment -->
   <action on="true" >childCounter = #(long) (0)</action>
   <action on="false">childCounter = #(long) (1)</action>
</rule>
```

This will handle correctly the 1st child (whether it is a comment or other kind).

#24 - 10/23/2012 08:34 AM - Ovidiu Maxiniuc

- File om_upd20120817a.zip added

Attached the update archive.

#25 - 10/24/2012 10:27 AM - Greg Shah

Feedback on the proposed fix:

1. The update zip should have the full path for brew (p2j/rules/convert/brew.xml).

2. The copyright date needs to be updated.

3. A history entry must be added to the header of the file.

4. Minor typo: "procesisng".

5. The code itself looks OK. After fixing the above issues, please post the updated zip into this Redmine task. Then regression test the conversion change with Majic as documented in today's update to #1443. Post the results here.

#26 - 10/24/2012 10:42 AM - Greg Shah

- File om_upd20120726a.zip added

I am uploading the original update. This is for historical purposes only and is very different from the improved solution posted in om_upd20120817a.zip. Don't use it.

Files

calendar.p	94 Bytes	06/15/2012	Ovidiu Maxiniuc
chrdat.i	236 Bytes	06/15/2012	Ovidiu Maxiniuc
p2j.cfg.xml	2.04 KB	06/15/2012	Ovidiu Maxiniuc
om_upd20120817a.zip	8.21 KB	10/23/2012	Ovidiu Maxiniuc
om_upd20120726a.zip	7.85 KB	10/24/2012	Greg Shah