Literate programming meets UML

Dr. Alun Moon
School of Informatics
University of Northumbria
Newcastle upon Tyne, UK
alun.moon@unn.ac.uk

Abstract

This work is an ongoing small project to apply the benefits of *literate programming* to UML. Literate programming is a powerful tool in that it places the emphasis on the documentation of the algorithm, and allows the code to be developed in a logical order. UML is a useful graphical notation to describe features of a software system. However, it lacks the ability to document the code and algorithm in detail. This gap can be filled by literate programming. Elements of UML can usefully enhance the documentation part of a web, with "a picture worth a thousand words". Finally the process of *tangling* a web into a program is applied to the UML to create a final diagram from fragments throughout the web. The diagrams are 'enhanced' by having TEX available to typeset the text.

1 Introduction

Literate programming is a powerful tool in that it places the emphasis on the documentation of the algorithm, and allows the code to be developed in a logical order. UML (Uniform Modeling Language) is a useful graphical notation to describe features of a software system. However, it lacks the ability to document the code and algorithm in detail. This gap can be filled by *literate programming*. Elements of UML can usefully enhance the documentation part of a web, with "a picture worth a thousand words".

METAPOST has been used to develop the graphical part of the system; macros for TEX are included in the web document. METAFONT has all the geometrical tools to allow a diagram to be built up, and its equation solving mechanism allows the elements to be defined in relation to each other. METAPOST also has facilities for typesetting text, making it the suitable tool to use.

1.1 No existing packages

Existing packages on CTAN such as PSTRICKS have many of the layout tools and arrow decoration needed for UML. This project is in part a learning exercise in writing METAPOST and TEX macro packages. The TEX components are written for plain TEX, as this is what CWEAVE produces.

2 Conventions

These tools were developed with Java in mind as the language. Java and UML feature heavily in the teaching within the School at Northumbria University. Some form of literate programming may be introduced to the undergraduates, if only just the concept of writing documentation, to help emphasise design in software engineering.

Although Java allows multiple classes in a source file, for the purposes of this tool only one is allowed. Each web file generates one Java file, which compiles to one class. Multiple classes may be possible later. This keeps the management of the diagram elements simple.

3 Design of the macros

The initial set of macros have a slightly *object oriented* feel about them. Class names are used as suffix parameters making a readable file. As the diagrams become more complex, additional data structures are used to ease processing by METAPOST. The TEX macros write material to a `.uml` file which is post-processed to create METAPOST input files, much as an index is processed with makeindex.

3.1 Tangled or Weaved?

Are UML diagrams tangled or weaved? The answer is a bit of both. They are weaved as they form part of the documentation, and include TEX material. They are tangled as the material is defined in the order of the web file, but has to be rearranged into a program or hierarchical order.

```
\def\private{$-$} \def\public{$+$}
\def\package{$\phantom{+}$}
\def\protected{$\sim$}

\def\classformatproperties#1{%
    \vbox{\halign{##\hfil\cr #1 }}}

% List macros after Knuth in
% The TeXbook, page 378
\def\leftlist#1{%
    \def\\##1{\relax##1\cr}%
    \vbox{\halign{##\hfil\cr#1}}}

\def\classformatlist#1{\leftlist#1}
```

Figure 1: TeX macros for class diagram contents.

4 Class diagrams

The TeX and METAPOST macros are shown in figures 1 and 2.

The METAPOST class is built up as a picture. The `class` macro takes three arguments: pictures for the title, attributes and operations of the class. These are given as `btex`...`etex` formatted pictures. Once all the attributes and operations are known, the class has a fixed size. The code declares three points as suffixes to the class name. The pair `reg` is a registration point, used to position the class when finally drawing it. The two pairs `top` and `bot` are points to connect inheritance arrows to. The picture variable `pic` holds the picture of the formatted class for drawing. The points for the inheritance arrows are a fixed distance from the left edge of the class only because I prefer to align the edges of the boxes.

The TeX macros are used to format the contents of a class. There is a set of symbols for the access qualifiers, to allow for easy alignment. The attributes and operations can be formatted using the `\classformatproperties` macro, where the elements are separated by `\cr` tokens. The `\classformatlist` macro formats a list of elements, with the list in the form suggested by Knuth in *The TeXbook* (Knuth, 2000, p. 378).

4.1 Alignment

The attributes and operations are aligned in a `\vbox` using `\halign`. One of the macros above must be used. The TeX macros writing the `.uml` file write out fragments of METAPOST. If the `\halign` macro was used then the `#` symbol in the template is expanded by `\write` to `##`.

```
vardef class@#(expr title)(expr attributes)
   (expr operations) :=
 save x,y;
 scantokens("pair " & str @# & " top");
 scantokens("pair " & str @# & " bot");
 scantokens("pair " & str @# & " reg");
 scantokens("picture " & str @# & " pic");
 @#pic := nullpicture;
 @#reg + right scaled 1cm = @#top;
 @#top-z0 = @#bot-z6;
 pen ln; ln = pensquare scaled 1pt;
 z0 = origin;
 x1-x0 = x3-x2 = x5-x4 = x7-x6
   = max(width title, width attributes,
         width operations, 2cm) + 1pc;
 x0 = x2 = x4 = x6;
 y0-y1 = y2-y3 = y4-y5 = y6-y7 = 0;
 y0-y2 = 1.5pc + height title;
 y2-y4 = 1pc + height attributes;
 y4-y6 = 1pc + height operations;

 addto @#pic doublepath z0--z1--z7--z6--cycle
  withpen ln;
 addto @#pic doublepath z2--z3 withpen ln;
 addto @#pic doublepath z4--z5 withpen ln;
 addto @#pic also title shifted (z2+(.5pc,.75pc));
 addto @#pic also attributes shifted
  (z4+(.5pc,.5pc)-llcorner attributes);
 addto @#pic also operations shifted
  (z6+(.5pc,.5pc)-llcorner operations);
enddef;
```

Figure 2: METAPOST code for a class.

4.2 Example

An example class diagram is shown in figure 3, and the code that generated it in figure 4.

5 Sequence diagrams

The sequence diagram has been developed in a simple human-friendly form, and a complex machine form. The simple form allows simple sequence diagrams to be drawn. There is a limitation: only one method per class can be drawn.

Unlike class diagrams where classes can be laid out on a grid, elements of sequence diagrams affect not only the position but also the size of other elements. For this reason the points that form an element must be declared before it can take part in the diagram. Sequence diagrams have three main sections in the code: declaration, creation and drawing.

6 Modifying CWEB

The original plan was to modify CWEB to work with Java and UML. This has not been pursued as the author has learned much more about CWEB. The modifications if any are likely to be minor, and there may be a better route using TeX macros or other tools, for instance:

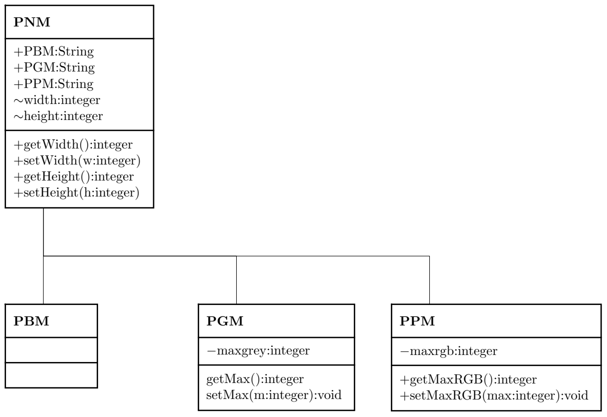

Figure 3: Sample class diagram.

- CWEB produces C++, which is close enough to Java. A web file using the `@s` mechanism to modify the syntax to Java is given in appendix A.
- UML creation can be done largely through TEX macros via an intermediate `.uml` file, just as indexes are produced to be read as a set of macros, after sorting and cross-referencing.
- By choosing good macro names and calling conventions, a language such as Perl can be very useful, especially if helpful data is put into comments in the web source and intermediate files.
- A simple `sed` script (`sed -e 's/^#/\/\//'`) converts the `#` line pragmas into line comments. (Can anyone come up with a version of `javac` that can make use of the `#` line pragmas?)

7 Web UML meta-tools

The web meta-tools for UML are currently in a primitive state. Most of the effort is currently on getting a good set of TEX macros. The METAPOST data structures are undergoing a major revision which fundamentally changes the internals of the tools. Two tools are needed to do the tangling:

- class builder — to collect attribute and operation lines and write the TEX/METAPOST class macro.
- sequencer — to arrange the sequences, write all the sections, declarations, creation, and drawing.

8 Data structures and macros

The data structures and macro calling conventions are undergoing a major revision. The macros presented here work well, but have a limiting simplicity, especially the sequence diagram, which has the following limitations:

```
beginfig(0)
 class.pnm(btex \bf PNM etex)
 (btex \classformatlist{
   \\{\public PBM:String}
   \\{\public PGM:String}
   \\{\public PPM:String}
   \\{\protected width:integer}
   \\{\protected height:integer}} etex);
 (btex \classformatlist{
   \\{\public getWidth():integer}
   \\{\public setWidth(w:integer)}
   \\{\public getHeight():integer}
   \\{\public setHeight(h:integer)}} etex);

 class.pbm(btex \bf PBM etex)(btex ~ etex)
   (btex ~ etex);

 class.pgm(btex \bf PGM etex) (btex \classformatlist{
  \\{\private maxgrey:integer}} etex)
  (btex \classformatlist{
   \\{getMax():integer}
   \\{setMax(m:integer):void}} etex);

 class.ppm(btex \bf PPM etex) (btex \classformatlist{
  \\{\private maxrgb:integer}} etex)
  (btex \classformatlist{
  \\{\public getMaxRGB():integer}
  \\{\public setMaxRGB(max:integer):void}} etex);

 pnm.reg = origin;
 pnm.bot - pbm.top = (0,1in);
 ppm.reg - pgm.reg = pgm.reg - pbm.reg = (2in,0);

 forsuffixes $=pnm,pbm,pgm,ppm: drawclass$ ; endfor;

 draw pbm.top connect pnm.bot ;
 draw pgm.top connect pnm.bot;
 draw ppm.top connect pnm.bot;
endfig;
```

Figure 4: METAPOST code for a class diagram.

- only one call per sequence element can be made;
- each sequence element can be called by only one other.

This is due to the use of suffix names for the elements.

8.1 Revised structure

In the revised structure a sequence block would be referred to as, for instance, l_2s_3, meaning the third sequence block down in the second swim-lane. This makes for nearly unreadable METAPOST code for a complex diagram, but does allow complex diagrams to be built by the meta-tools. Losing the name to refer to an element allows no restrictions on the number of calls to an operation.

```
vardef sequ@#(text call_list) =
  @#.n = .5[@#.nw,@#.ne];
  @#.s = .5[@#.sw,@#.se];
  @#.ne - @#.nw = @#.se - @#.sw = @#.ce - @#.cw
   = @#.re - @#.rw = (seq_width,0);
  @#.nw - @#.cw = @#.rw - @#.sw = @#.ne - @#.ce
   = @#.re - @#.se = (0,seq_width);
  @#.nw - @#.sw = (0,whatever);
  if (length(str call_list) >0):
   @#.ce + (seq_space,0) = call_list.nw;
   @#.re + (seq_space,0) = call_list.sw;
  else:
   @#.ce = @#.re;
  fi;
 enddef;
```

Figure 5: Sequence diagram element.

```
declaresequence.main; declaresequence.bezier;
declaresequence.bernstein; declaresequence.binomial;
declaresequence.fact;

sequ.main(bezier); sequ.bezier(bernstein);
sequ.bernstein(binomial); sequ.binomial(fact);
sequ.fact();

main.nw = origin;

beginfig(0)
 pickup pensquare scaled 1pt;
 drawsequence.main;
 drawsequence.bezier; drawsequence.bernstein;
 drawsequence.binomial; drawsequence.fact;
 drawarrow main.ce--bezier.nw;
 drawarrow bezier.ce--bernstein.nw;
 drawarrow bernstein.ce--binomial.nw;
 drawarrow binomial.ce--fact.nw;
endfig;
```

Figure 6: Sequence diagram usage.

9 Teaching

CWEB is being introduced to colleagues in the school and suggested for use on a Masters in embedded systems. There are issues in relation to UML as ANSI C or MISRA C are the preferred choices of language. Is there a neat way of generating header files without too much repetition in the WEB source?

Literate programming has also been suggested as a way to help undergraduate students think about the design (engineering) of program code, by concentrating on the documentation rather than the coding.

References

Knuth, Donald. *The TEXbook*. Addison-Wesley, 2000.

A Java web file

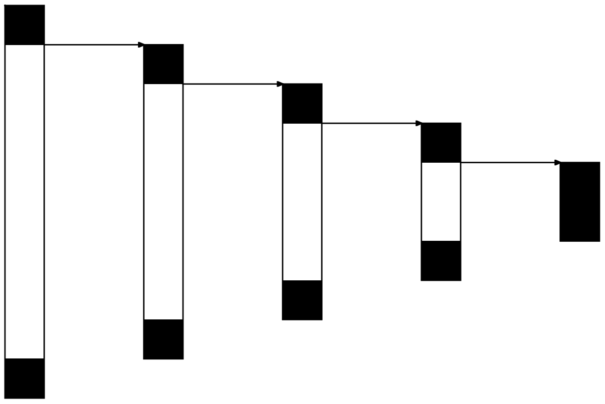

Figure 7: Sequence diagram.

```
% NULL->null
@s null NULL

% Java keywords *not* in CWEB
@s abstract int          @s interface int
@s boolean int           @s native int
@s byte int              @s package int
@s extends int           @s strictfp int
@s final int             @s super int
@s finally if            @s synchronized int
@s implements int        @s throws int
@s import include        @s transient int
@s instanceof sizeof

% CWEB keywords *not* in Java
@s and variable          @s namespace variable
@s and_eq variable       @s not variable
@s asm variable          @s not_eq variable
@s auto variable         @s offsetof variable
@s bitand variable       @s operator variable
@s bitor variable        @s or variable
@s bool variable         @s or_eq variable
@s clock_t variable      @s pragma variable
@s compl variable        @s ptrdiff_t variable
@s const_cast variable   @s register variable
@s define variable       @s reinterpret_cast variable
@s defined variable      @s sig_atomic_t variable
@s delete variable       @s signed variable
@s div_t variable        @s size_t variable
@s dynamic_cast variable @s sizeof variable
@s elif variable         @s static_cast variable
@s endif variable        @s struct variable
@s enum variable         @s template variable
@s error variable        @s time_t variable
@s explicit variable     @s typedef variable
@s export variable       @s typeid variable
@s extern variable       @s typename variable
@s FILE variable         @s undef variable
@s fpos_t variable       @s union variable
@s friend variable       @s unsigned variable
@s ifdef variable        @s using variable
@s ifndef variable       @s va_dcl variable
@s include variable      @s va_list variable
@s inline variable       @s virtual variable
@s jmp_buf variable      @s wchar_t variable
@s ldiv_t variable       @s xor variable
@s line variable         @s xor_eq variable
@s mutable variable
```