YUML and YPL Database Manual

Don Batory
batory@cs.utexas.edu
November 2016

1 YUML

Yuml is a free web service that draws UML class diagrams given a Yuml input specification. As this is a for-profit company, the ″free″ service comes with some strings attached. Namely, it will produce a pretty class diagram for you provided that your specification is not too complicated. (I found other problems with Yuml, but this is a story for another day).

Familiarize yourself with Yuml:

- go to the Yuml Class Diagram Web site
- type in this spec
  `[student|name]has-loves[course|name]`
- and Yuml returns this absolutely gorgeous diagram:

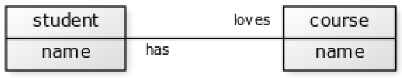

Fig. 1: Student-Course Diagram.

Try drawing your own diagrams. When you feel comfortable, proceed to the next section.

1.1 Yuml Specifications

A Yuml specification is elegant. Here is a BNF of a subset of Yuml that `MDELite6` uses. Literals (a.k.a. tokens) are in single ‘quotes’.

```
         // YumlSpec is 1 or more lines
YumlSpec : Line+ ;

         // each line defines a box or connection
Line : Box | Connection ;

Box : '[' Class ']' ;

         // read left-2-right, ignore subscripts
Connection : BoxName [End1] [Role1] DashType
             [Role2] [End2] BoxName ;

         // BoxName = class or interface name
BoxName : '[' String ']'
        | '[' 'interface;' String ']'
        ;

DashType : '-'   // solid line
         | '-.-' // dashed line
         ;

End : '<>' // aggregation
    | '++' // composition
    | '^'  // inheritance
    | '<'  // left arrow
    | '>'  // right arrow
    ;

         // String that has no ']' and quote chars
Role : String ;

         // name only, name+meths only, name+flds+meths
Class : Name
      | Name '|' String
      | Name '|' String '|' String
      ;

         // String that has no ']' and quote chars
Name : String ;
```

Note that a String token is mentioned above. This not a Java String, but one that is devoid of the characters:

- comma ′,′
- left brace ′[′
- right brace ′]′
- less than ′<′
- greater than ′>′
- minus ′-′

Further, a semicolon ″;″ means new line. Some hints:

- Since Yuml doesn’t like ″[]″ as in ″String[]″, I use ″#″ – so ″String[]″ becomes ″String#″.
- Since Yuml doesn’t like commas (as in ″foo(int x, int y)″), I simply use blanks between types – like ″foo(int int)″.
- Since Yuml has no indicator to distinguish static from non-static, I simply preface the names of static members with an underscore – like ″_bar()″.

So, the following Yuml specification (a sentence in the above language):

```
[Interface;Closable|close()]
[Interface;NetworkChannel|
    bind();getLocalAddress();getOption();
                setOption();supportedOptions()]
[MyClass|_MyClass();close()]
[Interface;Closable]^-.-[MyClass]
[YourClass]<>-3>[MyClass]
[interface;Closable]^-[Interface;NetworkChannel]
```

Produces the beauty of Figure 2.

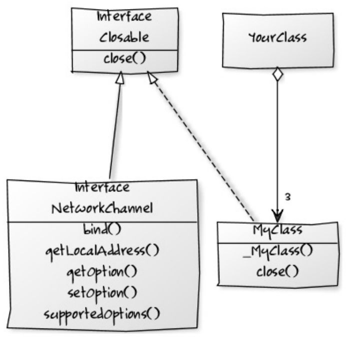

Fig. 2: Another Yuml Diagram.

Warning! Do not read the above specification too deeply! `'Interface;Closable'` is a String. The word `'Interface'` means nothing to Yuml. It could just as well have been `'George'`, which also means nothing to Yuml. What Yuml does understand is `';'` (semicolon), which means add a new line. So `'Interface;Closable'` produces a 2-line name in the above figure. And the string `'bind();getLocalAdddress()'` means print strings `'bind()'` and `'getLocalAddress()'` on separate lines.

1.2 The YPL Schema

Here is the YPL schema (`ypl.schema.pl`), which can encode YUML diagrams as a database of tuples:

```
dbase(yuml,[yumlClass,yumlInterface,yumlAssociation]).

table(yumlClass,[id,"name","fields","methods"]).
table(yumlInterface,[id,"name","methods"]).
table(yumlAssociation,["name1","role1","end1",
       "name2","role2","end2"]).
```

Here is a `MDL.ClassYumlParser` translation of (*ie,* the database of tuples that encodes) the specification of Figure 1:

```
dbase(ypl,[yumlClass,yumlInterface,yumlAssociation]).

table(yumlClass,[id,"name","fields","methods"]).
yumlClass(c0,'student','name','').
yumlClass(c1,'course','name','').

table(yumlInterface,[id,"name","methods"]).

table(yumlAssociation,[id,"name1","role1","end1","name2","role2","end2"]).
yumlAssociation(id0,'student|name','has','','course|name','loves','').
```

And here is a `MDL.ClassYumlParser` translation of the specification of Figure 2:

```
dbase(ypl,[yumlClass,yumlInterface,yumlAssociation]).

table(yumlClass,[id,"name","fields","methods"]).
yumlClass(c2,'MyClass','_MyClass();close()','').
yumlClass(c3,'YourClass','','').

table(yumlInterface,[id,"name","methods"]).
yumlInterface(c1,';NetworkChannel','').
yumlInterface(c4,';Closable','').
yumlInterface(c0,';Closable','').

table(yumlAssociation,[id,"name1","role1","end1","name2","role2","end2"]).
yumlAssociation(id0,';Closable','','^','MyClass','','').
yumlAssociation(id1,'YourClass','','<>','MyClass','3','>').
yumlAssociation(id2,';Closable','','^',';NetworkChannel','','').
```

1.3 YPL Constraints

There indeed are YPL constraints. I have not posted them, as they are good examples for homework assignments. If you are not in my classes, you may contact me for hints.