Documentation and examples of consdiag.py version 1.0

Manuel Gutierrez Algaba, irmina@ctv.es

December 1998

1 Copyright issues

2 Introduction

When I wrote this program there wasn't any automated utility for drawing Rumbaugh OO boxes. Of course, you can use xfig or any other drawing program for this purpose. But this utility has two major advantages, It's easier to use and it's faster to 'draw'. On the other hand, the kinds of available drawings are limited to:

- isolated classes
- trees (and just trees, no graphs :( ) of classes related by their inheritances
- simple associations of classes

It's not much , but...
And this is an utility written in python.

`http://www.python.org`

I imagine that it could be written in TEX but It's 10 times easier to use python. And what's more important TEX programmers have a model , if they want to do the translation.
Another point, the python code could be improved, I've duplicated code in several parts, because I thought the classes were really different each other, ..., in the end, they've resulted to be very similar in the way they work.

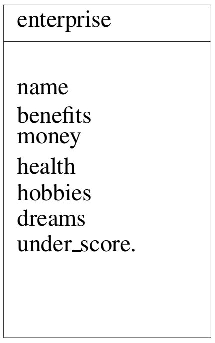
And, it generates tex code. So if you want it in postscript, gif or whatever, use the programs dvi, gs or grab directly from a window!

3 How to use it

3.1 Using it in a python code

Just use a "from consdiag import * " and this piece of python code (for example). In drawing3.tex is kept the following TEX instructions that generates the drawing 2:

Figure 1: Single figure




```
c = a_simple_class()
c.do_name_clase('person')
c.do_attributes(['name' , 'sex', 'money', 'health',
    'hobbies', 'dreams'])
c.do_functions(['dream', 'love', 'play soccer', 'diving'])
c.index_of_class(3)

c.do_lines()
c.generate_latex_code('drawing3.tex')
```

3.2 Sources of help

Well, the best you can do is to take a look at the end of consdiag.py where you can see how it generates the draw. Secondly, you should take a look at the source of this document, that is: less consdiag.tex Two important details:

- Don't forget to include the command: `\usepackage{texdraw}` at the start of your document
- The figures are included using a simple `\input` command.

And that's all.

4 Some examples of it

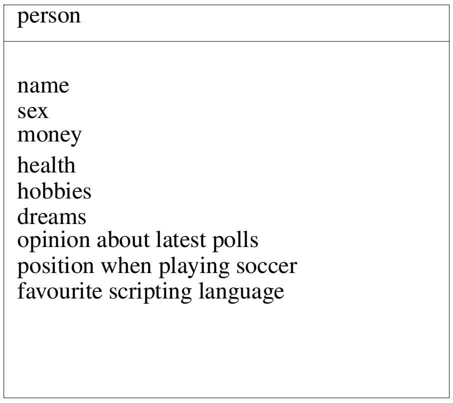
4.1 A simple class

The most basic figure you can face when dealing with OO classes is a simple class, with some attributes. And that's what you can see in figure 1.

4.2 A simple class with long names

Let's see what happens to figure 1 when you want to include a really long attribute. If you were working with a traditional drawing utility, you would have to redraw all the

Figure 2: Single figure with long names



box. But with this utility no. That’s what you got (figure 2). So, this utility is starting to pay, isn’t it?

4.3 A simple class with attributes and mark of cardinality

Well, it’s time to see a class at its full extent. A class may have functions, and it may be included in more than one drawing. In that case a mark of cardinality (a number that says how many times that class has been drawn) is shown. An example of such a class is what you can see at figure 3. In this case the cardinality is 3. This is figure is done with the python code:

```
c = a_simple_class()
c.do_name_clase(’person’)
c.do_attributes([’name’ , ’sex’, ’money’, ’health’,
    ’hobbies’, ’dreams’])
c.do_functions([’dream’, ’love’, ’play soccer’, ’diving’])
c.index_of_class(3)

c.do_lines()
c.generate_latex_code(’drawing3.tex’)
```

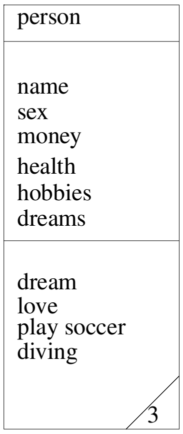
4.4 Inheritances examples

Well, till now, it’s a very simple program, let’s face a more interesting example. A inheritance of classes. figure 4

This is achieved typesetting :

```
c = a_simple_class()
c.do_name_clase(’person’)
c.do_attributes([’name’ , ’sex’, ’money’, ’health’,
’hobbies’, ’dreams’])
c.do_functions([’dream’, ’love’, ’play soccer’, ’diving’])
```

Figure 3: Single figure with attributions, functions and marks




```
c.do_lines()
d = a_simple_class()
d.do_name_clase('work')
d.do_attributes(['salary' , 'post',
'labor union', 'productivity', 'qualifications',
 'capabilities'])
d.do_functions(['work', 'strike', 'be ill', 'be fired'])

d.do_lines()
e = union_derivation()
e.add_union(c,d)
e.generate_latex_code('drawing4.tex')
```

Easy, uh ? :)

But much more complex drawings are also easy to do as shows 5 , this is a picture of a mother class that has several daughter classes.

Regretfully, it IS NOT possible (right now) to inheritate from more than one class. Instead , you can use indices to mark several inheritances.

And , well, you can use several levels of inheritances, just be a bit careful about how you build the code, take this (it makes figure 6) as example:

```
d = a_simple_class()
d.do_name_clase('worker')
d.do_attributes(['salary' , 'post',
'labor union', 'productivity', 'qualifications',
 'capabilities'])


d.do_lines()
f = a_simple_class()
```

Figure 4: Simple inheritance

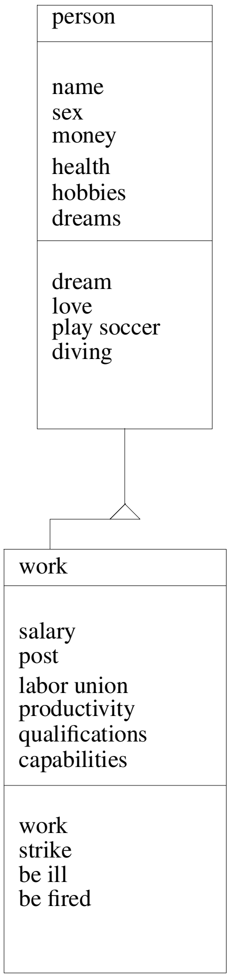
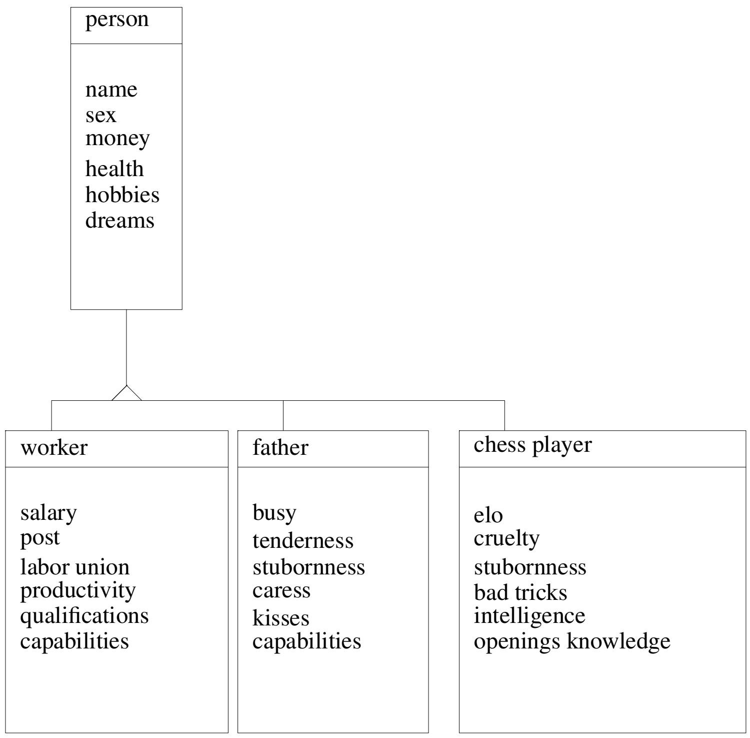


Figure 5: Several inheritances



```
f.do_name_clase('computer scientist')
f.do_attributes(['languages' , 'preferred OS',
'preferred editor', 'hackerism', 'ATBB'])

f.do_lines()

h = union_derivation()
h.add_union(d,f)

c = a_simple_class()
c.do_name_clase('person')
c.do_attributes(['name' , 'sex', 'money', 'health',
'hobbies', 'dreams'])
c.do_lines()

e = union_derivation()
e.add_union(c,h)
e.generate_latex_code('drawing6.tex')
```

Finally a remainder: You can combine this features so you can get several levels and with several sons, but... watch the space! The piece of paper has its limitations.

4.5 Associations examples

Well, I got a bit upset when I realize that fact of multiple inheritances, so associations haven't got so complicated, and in fact , you can do only associations between simple classes and associations, so you can't mix associations and derivations, that would have been fine. figure 7

4.6 Aggregations too!

You can do aggregations, just replace "derivation" in the previous examples for "agregation" and that's all. As example look at figure 8.

5 Caveats and bugs

If you want to do very complex drawings as:

- several levels of inheritances/agregations and each other with lots of sons,
- really deep trees of inheritances, more than four are probably a really bad idea,
- several nested associations,

then, probably you'll get a shit as outcome. This utility is not clever, just try to to set the classes in a logical, predetermined way.

Apart from this caveat, there'll be some bugs, . . . , or perhaps not. python is rather robust!

Figure 6: Several levels of inheritances

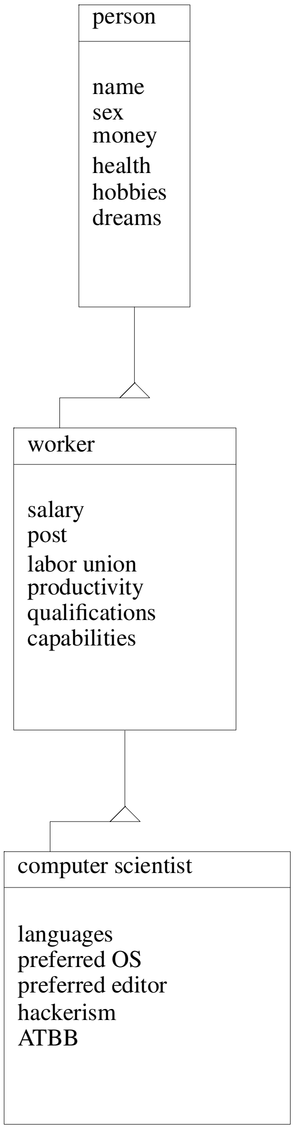


Figure 7: Simple association

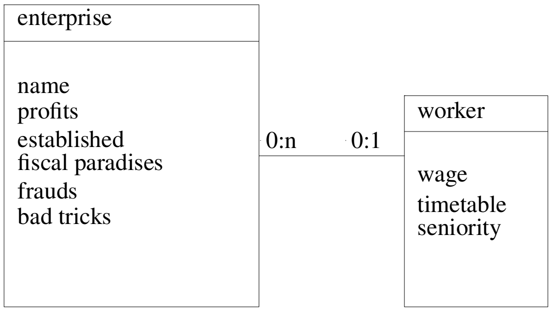
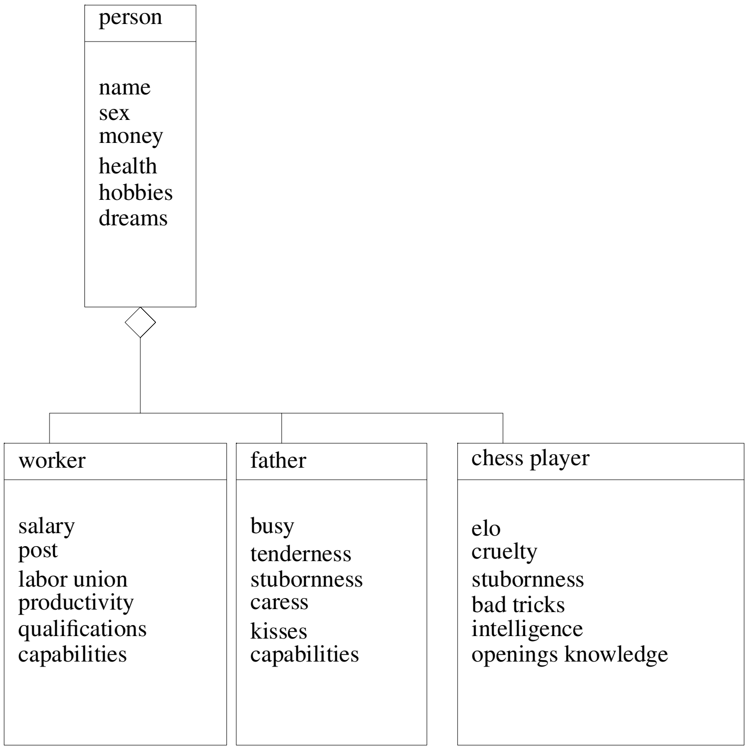


Figure 8: Multiple aggregation



6 Bye bye

I hope this documentation helps you to use this utility. It's not difficult and greatly profitable. And if you want to get similar drawings or improve some of them just take a look at the code, once you get accostumed to it , you'll find it quite logical.