Introduction to UML
(Unified Modelling Language)
Part three

UML Class diagrams: tricks and tips

Written by: Robin Beaumont  e-mail: robin@organplayers.co.uk
Date last updated: Thursday, 06 October 2011Version: 1

This document is part of a complete uml course at: http://www.robin-beaumont.co.uk/virtualclassroom/contents.html

Contents
1. Learning Outcomes ......................................................................................................................................... 1
2. Introduction ..................................................................................................................................................... 2
3. Confusing Instances and Classes ................................................................................................................... 2
4. Do not Include Lookup/Descriptor classes ................................................................................................... 3
5. Unnecessary Complexity ............................................................................................................................... 4
6. Poor Aesthetics ............................................................................................................................................... 5
7. Use Multiple diagrams .................................................................................................................................. 6
8. Export from the case tool using the appropriate file format ......................................................................... 6
9. Summary ......................................................................................................................................................... 7
10. References ...................................................................................................................................................... 8

1. Learning Outcomes
This document aims to provide you with the following skills and information. After you have completed it you should come back to these points, ticking off those you feel happy with.

<table>
<thead>
<tr>
<th>Learning outcome</th>
<th>Tick box</th>
</tr>
</thead>
<tbody>
<tr>
<td>Be able to discuss the problem of confusing instances for classes in uml class diagrams</td>
<td></td>
</tr>
<tr>
<td>Be able to explain what lookup(descriptor classes are and how they are managed in uml class diagrams</td>
<td></td>
</tr>
<tr>
<td>Be able to demonstrate how unnecessary complexity in uml class diagrams can be avoided</td>
<td></td>
</tr>
<tr>
<td>Be able to recognise and correct uml class diagrams with poor aesthetics</td>
<td></td>
</tr>
<tr>
<td>Be able to describe how the use of multiple diagrams may improve poor aesthetics</td>
<td></td>
</tr>
<tr>
<td>Be able to discuss the various export options available in case tools, including those that are most appropriate.</td>
<td></td>
</tr>
</tbody>
</table>
2. Introduction

This very short chapter investigates some of the common mistakes people make when developing UML class diagrams.

3. Confusing Instances and Classes

Consider the following uml class diagram with four classes: DOCTOR, PATIENT, DRUG_TREATMENT and DRUG, from a previous chapter.

Repeating the sentence from the original scenario:

"Initially the system will be concerned solely with drug treatment. Each patient is required to take a variety of drugs a certain number of times per day and for varying lengths of time."

According to our uml class diagram on the left:

- An instance of DRUG_TREATMENT is related to many DRUG instances.
- A DRUG instance is related to a single DRUG_TREATMENT instance.

For a variety of reasons this is the best way of modelling the association. However, let's consider another way the associations may have been modelled.

A different modeller argues that while a particular DRUG_TREATMENT is associated with one or more DRUGs a particular DRUG is associated with one or more DRUG_TREATMENTS. While this may seem more logical it should be realised that now if we change the particular value in a instance of DRUG we may (depending upon how many DRUG_TREATMENT instances it is related to) be changing several DRUG_TREATMENT instances at the same time!

Why has the confusion occurred? The problem is the result of three things:

1. I think the first problem are the terms used in the original scenario if we replaced the above quote with what I feel to be equivalent clinical terms we have “Initially the system will be concerned solely with drug prescriptions (=drug treatment). Each patient is required to take a variety of doses (=drugs) a certain number of times per day and for varying lengths of time.” Do you feel happier with the equivalent terms?

2. Secondly the modeller is getting confused between classes and instances. The modeller is correct in thinking that a particular DRUG_TREATMENT (prescription) has one or more DRUGs (doses). S/he is working at the instance level (good) and thinking about a particular DRUG_TREATMENT for a particular patient. However, s/he then states that a particular DRUG is associated with one or more DRUG_TREATMENTS. Here s/he has forgotten that s/he is dealing with instances. While we are talking about a particular patient’s particular drug treatment, s/he has now moved to the level of thinking about all patients’ relationships to drugs. In this case the following is correct, A PATIENT takes many DRUG(s) and a DRUG (not dose) is associated with many patients.

However this is not what we are trying to model in this scenario as we are interested in individual patients and their drugs (i.e. doses) that are individually prescribed to them.

To highlight the difference it helps to consider instances of the various classes, let's do that now.

Let's assume john who is prescribed (DRUG_TREATMENT) morphine as required (prn). He has been given n DRUG (doses) instances. I have shown two instances in detail, instance dose1 and dose2. Clearly it makes no sense having any of these DRUG instances being associated with more than one DRUG_TREATMENT (drug prescription).

I hope this convinces you of the folly in considering the possibility that a single instance of DRUG (dose) is associated with more than one DRUG_TREATMENT (drug prescription).

3. Another reason why the modeller may have become confused is related to the concept of the lookup table, which we will discuss next.
4. Do not Include Lookup/Descriptor classes

While the topic of lookup tables is very much concerned with the low level design, the concept seems often to impinge upon people’s decision making when they are just beginning to draw uml class diagrams. This is probably because we are all familiar with the concept and therefore feel that it must be represented somewhere in the class model.

A lookup table is something used by the users of databases to provide a pick list of possible entries for a particular attribute. It might be something like a disease code or name of a local doctor.

Consider the class diagram on the previous page PATIENT -> DRUG_TREATMENT -> DRUG. Let’s assume a particular doctor wishes to have the ability to select a drug from a pre-defined drug list, which seems entirely reasonable.

We may think it is sensible to add another class to the diagram, DRUG_DICTIONARY, which would provide certain values to any particular DRUG instance. In other words certain attribute values for any DRUG instance would be obtained by looking up and copying the values from the DRUG_DICTIONARY class. You might think that this is just another one to many association, but look below and think about it:

In this situation all we want to do is copy certain attribute values from the DRUG_DICTIONARY into a particular DRUG instance.

This process of copying values from another table but not really associating with the class is how a lookup table works, [technically we say it lacks referential integrity constraints]. Such details as lookup tables are not usually shown. Thinking about it this makes sense because just looking at the class diagram opposite suggests there are several other places we could have lookup tables: one to show the doctors, one for the drug treatments and even one for selecting patients. The class diagram would start to look very cluttered.

Another reason why it makes sense not to show the lookup tables is because frequently these tables are simply queries on the attribute you are currently working with. For example you may want to add a new drug attribute value for a new instance of a DRUG based upon all the values that exist in the instances that already exist. You may remember that when we started to consider which where appropriate classes, one of the exclusion criteria where those possible classes we came up with that were simply the result of processing information from other classes.

Just one final warning: although I have indicated that it is not a good idea to show lookup tables in a uml class diagram, there may be the odd occasion to show one. For example, in the above the DRUG_DICTIONARY might be something important in your model such as the British National Formulary or something similar. You would probably then include it and add some text in the diagram providing additional information.

You should begin to realise now that there is not one correct or perfect uml class diagram. Much depends upon the unique situation in which each is developed.

The uml solution to the problem of lookup tables is the enumerated type so we could simply create a new enumerated type called data_dictionary, however an enumerated type only consists of a single list of values.

Lookup tables are also called descriptor tables by modellers who specialise in developing a special kind of database called a data warehouse. (Nelson G S 2001 Implementing a Dimensional Data Warehouse with the SAS System SUGI26 proceedings. See: http://www2.sas.com/proceedings/sugi22/DATAWARE/PAPER129.PDF)

Hernandez 1997 p344-349 calls them Validation tables and recommends that such classes be distinguished in the model (he was discussing ERDs) from other classes by having a distinctive marking.
5. Unnecessary Complexity

An historical aside – Occam’s Razor

Entia non sunt multiplicanda præter necessitatem – entities ought not to be multiplied except from necessity

Which means that all unnecessary facts or constituents in the subject being analysed are to be eliminated. Occam dissected every question as with a razor. William of Occam, the Doctor Singularis et Invincibilis (d. 1349), the great Franciscan scholastic philosopher, was probably born at Ockham, Surrey, UK. Occam being the latinized form of the name.

From Ebenezzor Cobham Brewers Dictionary of Phase & Fable (Wordsworth edition revised by Ivor Evans 1993)

One of the main problems with class diagrams is the desire to introduce a multitude of associations and classes when thoughtfulness (along with a large amount of scrap paper!) can usually produce a model with much more clarity. Here are some examples.

Unnecessary entities and relationships

Consider the following:

A COURIER delivers one or more SAMPLEs.
Each sample can have zero or more TESTs carried out on it.
Each TEST produces zero or more FINDINGs.
Each sample is signed for by the RECEIVING_TECHNICIAN.
The sample is then managed by a SUPERVISOR_TECHNICIAN who also confirms any FINDINGs.
The TESTs are carried out by JUNIOR_TECHNICIANs.

From the above information one possible uml class diagram is shown opposite. This all looks very complex. The classes COURIER, SAMPLE, TEST and FINDING seem sufficiently different. However, how different are the RECEIVING_TECHNICIAN, SUPERVISOR_TECHNICIAN and JUNIOR_TECHNICIAN? I even wonder what would happen on a Sunday afternoon, when they probably all would be the same person.

Obviously the modeller has tried to present the information that is collected at each stage concerning the TECHNICIAN input, but I wonder if the following might be just as suitable.

To indicate that a particular grade of technician should only carry out certain tasks a small amount of text could be added to the accompanying text and it is important to realise that a uml class diagram cannot express all the complexities you may wish to model.

Looking again at the uml class diagram there still appears to be a large number of associations from the TECHNICIAN class to various others. Do we need all these? To help answer this question I would like to digress slightly and consider two different aspects:

- Independent versus sequential associations
- Multiple level associations

Avoid loop associations - one may be redundant

Carter 1995 p.49 (discussing ERDs) recommends that if there are a number of associations which form a loop in your class diagram if possible the loop should be broken. For example in the SAMPLE -> TECHNICIAN -> TEST situation in the uml class diagram above you should consider very carefully if you need all three associations. In the above diagram you do but there are situations where you do not. For example think about the three classes CUSTOMER, INVOICE and PAYMENT and assume we have associations between all three. However in this situation the association between Customer and PAYMENT is no really necessary as any payment information can be obtained by going through the Invoice which obviously will exist before a payment is made.
Independent versus sequential associations

Consider the following two class diagrams:

![Class Diagram Example]

Clearly the above example is rather manufactured as hopefully the right hand side is the option that would be chosen. However, it does highlight a valuable point.

Multiple level relationships

This is the situation in the COURIER -> SAMPLE -> TEST -> FINDING class diagram where the TECHNICIAN class has an association at three levels. Considering the information above about dependency, you should begin to realise that if we had a mandatory association (1:n) rather than an optional one (0:n) the situation would be made easier. Why? Because with a mandatory association we know that a parent instance would always possess at least one child instance. Let's presume the class diagram opposite was changed so that the associations between SAMPLE -> TEST and TEST and FINDING were mandatory (1:n). If this were the case we would know that every SAMPLE would be related to at least one TEST and FINDING instance. Given this information we might then revise the class diagram to have only one association from TECHNICIAN which would then probably be to FINDING.

While the above is one possible method of making the class diagram much clearer, it might not be appropriate to the situation. The question is: what is the real situation and once again what is one trying to achieve, does one want to know who processed the sample at each stage or not?

6. Poor Aesthetics

The diagram below shows a poor version of the one above.

![Poor Aesthetic Diagram Example]

The problems include:
- Classes too close together or far apart
- Lines crossed
- Inappropriately positioned multiplicities

Another common problem is making the diagram too small on the printed page - you should not need to zoom into 200% to see what the diagram is saying! This can be solved by using the strategy discussed in the next section.
Another problem is the inappropriate placement of the classes within the drawing canvas; the canvas should always be made to fill the objects on the diagram. The diagram opposite is very poor because of the excessive white space around the classes. The modeller simply needs to adjust the drawing frame.

7. Use Multiple diagrams

Often classes develop over time and end up with a fair number of attributes and operations. As mentioned in a previous chapter it is possible to suppress the attribute and operations compartments of the class symbol on the diagram and this is sometimes sensible to do so in an overview diagram where you are attempting to show all the classes together. You can then on subsequent pages have an expanded diagram of possibly individual classes with an accompanying narrative. For example, say we had a UML overview class diagram of a GP practice (only showing the class names), we would then possibly on subsequent pages have one or more pages devoted to each class which would include an expanded picture of the specific class with all its attributes and operations displayed.

8. Export from the case tool using the appropriate file format

Most case tools allow a variety of methods to export/save diagrams for use in documents. The two basic types of image are bitmap or vector based; vector images are preferred and allow you to edit the image once in the word processor.

The screenshot opposite is for the popular case tool MagicDraw. Most of the options produce bitmap images however the .wmf option does produce a vector image, however often word processors pre-process the image so that what has been saved in the case tool is not what is automatically imported into the word processor!
Clearly the most satisfactory result is for the wmf format file, however all formats require some manipulation to obtain optimal results, once you have found something you like I advise you to write down the settings so that you can use it again.

9. Summary

This chapter has described some of the common errors people make when developing uml class diagrams, it has focused on two main aspects, the actual validity of the diagrams along with methods to improve their aesthetics.

Exercise 1


Exercise 2

Time: 360 minutes (6 hrs)

It is very important to practice the skill of developing class diagrams, and to help you I have provided a number of more complex exercises than those you have encountered so far.

Go to my main website and download the “Scenarios for practicing modelling techniques” document: http://www.robin-beaumont.co.uk/virtualclassroom/contents.html

This document contains about fifteen scenarios.

Select two or three scenarios to:
- Develop a uml class diagram for each scenario
- Provide a description of each case
- List a number of attributes for each plus a description of each attribute
- List a set of constraints / assumptions - You do not have the luxury of being able to question the client?

Hints:
Remember to use your own ‘expert domain knowledge’ where appropriate. But always state clearly any assumptions / constraints you have made for each of the models.
10. References

Carter John 2000 Database design & programming with Access, SQL and visual basic McGraw Hill


Date C J. 1995 (6th ed.) An introduction to database systems. Addison-Wesley.


Hernandez M J 1997 Database design for mere Mortals. Addison - Wesley


Blaha M Rumbaugh 2005 (2nd ed.) Object-Oriented Modelling and design with UML. [basically this is the seocnd edition of Rumbaugh et al 1991]Prentice Hall.

End of document