Modelling for the faint-hearted

Tim Hoverd
University of York
tim.hoverd@cs.york.ac.uk
Programme

• Modelling
  – what, why

• UML
  – Objects, classes

• Usage
  – Hints and tips
Provoked...
Provoked by...

- appears to be a UML “activity diagram”
  - extrinsic pathway activity diagram
- The rounded rectangles should be “actions” or “activities”
  - Is “factor VII” an activity?
• appears to be a UML sequence diagram
  – “Blood clotting sequence diagram”
• What is “a prothrombin”
• What does the message “fibrin monomers” sent by a prothrombin to a thrombin mean?
• Shouldn’t there be something about the environment here?
  – Concentrations
  – Temperature
Models

• These diagrams are *models*  
  – well, they’re parts of a model

• Here’s another one:

  – “hybrid-pi” model of a bipolar junction transistor

• A bunch of precisely defined syntax
  – with *precise* meaning, even though some of the symbols represent things that don’t exist (†)
So what?

• We can now design:
  – *calculate* behaviour
  – without the massive complexity of doing the quantum mechanics...

• Transistors don’t *quite* work like that though
  – need to use the right model for the situation
More models

“In mechanics, it is necessary to take a real life problem and put it in mathematical language. This process is known as modelling.”

http://www.mathsrevision.net

• In general, modelling:
  – takes a real life problem and
  – represents it in a different language where
  – that language makes the problem more amenable to understanding and analysis

• Language:
  – precise
  – meaningful
  – amenable to manipulation
  – must fit the problem
  – UML is not always the right answer
But why produce a model?

• Why:
  – To get something clear in your own head
  – To communicate with others
  – To work jointly on some problem
  – To calculate what will happen in an otherwise intractable circumstance

• For all of these:
  – clarity of notation and syntax
  – (you can probably always mess it up though)

• The best models are the absolutely obvious ones
  – “Well, that’s obvious. I could have done that.”
Modelling languages

• Must be precise, with well understood semantics
• Specific purpose
• Limitations
  – Just because it works in the model doesn’t mean it will for real
• Different languages for apparently similar tasks
  – Transistor models:
    • Small signal, large signal
    • High speed, low speed
    • Hybrid-π, h-parameter
  – Computer software:
    • ER models
    • DFDs
    • ACP diagrams
    • UML notations
      – there’s lots of them
Explicit and implicit models

- Econometrics models
  - Do not necessarily contain lots of little people buying and selling things
  - They do not even represent those people explicitly
  - Their behaviour is implicit in the model

- War games
  - Do not usually contain lots of little soldiers and tanks
  - Although they may be explicitly represented
Modelling rules

• You need an explicit language
  – With explicit syntax

• But:
  1. You must know exactly what you mean by each bit of syntax
  2. A reader must know what is meant by each bit of syntax
  3. Those two semantics must be the same.

• We’re going to look at the UML notations
  – and how they’re usually used

• You can change the semantics if you want
  – As is done with the “4+1 model”
UML

• The “Unified Modelling Language”
  – it’s not a method, or a “methodology”, or agile, or BDUF, or anything else that’s popular this month

• Essentially a set of notations, for object-oriented software design and implementation

• Grew out of specific design notations, and approaches, of the three amigos
  – Booch, Rumbaugh, Jacobson
  – Heavily extended and modified since then
  – http://www.uml.org for the gory details
    • (don’t bother...)
A side-issue
Object orientation
not a side-issue

• The UML is explicitly a collection of notations for OO design
• Over 40 years old, and we’re still confused about it
  – Simula-67
  – Smalltalk-80
• Using UML notations in the “standard” way requires a good knowledge of OO
Object
Orientation

• Simple notion:
  – Systems are comprised of collections of passive objects
    • OK, I’m being deliberately extreme...
  – *Individual* objects send messages to each other, in response to messages which the original objects receive
  – objects advertise the messages they’re happy to respond to, but hide what they do about it
  – (which means that something has to send the first message...)
• If your model needs to be like that, then great
  – if not, find another notation, or use your own semantics
Objects

- Object orientation is about the behaviour of individual objects
  - “Objects have identity”; they’re not just “stuff”
- An object is asked to do something, by sending it a message
- The message receiver follows a pre-defined method of response to a message
- That response may include sending messages to other objects
  - that it already knows about
- At its core, OO is about behaviour not structure
  - the distinction between message and method is core to the available behaviour
Objects of a time

• OO appeared in the 60s and 70s
• At the time:
  – shift to user-centred systems (*Sketchpad*)
  – graphical user interfaces (*Smalltalk*)
  – move away from stepwise refinement and functional decomposition
  – information hiding
  – mythical man-month
  – **modelling** of the real world
    • “if I’m building a banking system, let’s explicitly model a bank account”
Passive and active objects

• Most objects are *passive* (usually):
  – just sit there waiting for a message to arrive
  – modelling process is really one of finding the right object to ask to do something for you
    • hopefully, it will understand
      – “I’m sorry, the object *Undercarriage* does not understand the message *lower*”...

• Active objects sit there doing stuff...

• Most OO designs are a collection of mostly passive objects
How do objects behave?

- Modelling in UML provides:
  - “interaction models”
  - to describe how objects interact with each other
  - objects know about other objects
  - and the messages they can respond to

- Describe specific interactions between a particular set of objects
  - describes a single scenario
  - horribly over-specific
  - but often useful

- Sometimes distinguish active and passive objects
Ladders and Stairs
Because...

• OO is about behaviour
  – and delegation to objects that support the desired behaviour
    • which they may well do by delegating to another object
      – which knows a nice object round the corner which...

• Which is a few more stairs...
But what are objects?

• and how do we know what other objects they know?
  – as that seems kinds of important...

• Instances of classes
  – Modelling concepts representing a set of similar objects
  – A class is an abstraction of some behaviour wrapped around some information
  – It’s not a functional abstraction

• That behaviour defined by an interface
  – the alphabet of messages that instances respond to
  – perhaps as an extension of another class’s interface
  – patterns of association between its instances

• Objects might well exist in an implementation
  – classes may well not; they’re pure model
A UML class model

- A fragment of a realistic ATC model
- **Classes**
  - the boxes
  - which have *compartments* within which are
    - the class name (which is a singular noun)
    - *Operations* (UML for message) and
    - *Attributes* (but they’re boring)
- **Associations**
  - the lines
- But what is a class?
  - It depends...
Modelling domains of discourse

• *Essential* models
  – classes represent the core concepts of the domain and their inter-relationships

• *Specification* models
  – how concepts are required to behave

• *Implementation* models
  – how concepts are implemented

• Moving between these domains is not
  – easy
  – automatable

• Rich potential for confusion:
  – *User* class in an essential model is likely about a person and their activities
  – *User* class in a specification is something that stores a password
  – *User* class in an implementation knows about password encryption
Associations

• Apparently simple lines between classes
• Give information about the objects
  — which are not on this diagram!
• What other objects they know about
• *Role names*
• *Cardinalities*
  — when do they apply?
The objects

- Two flight plan objects
- Invisible in class model
- *Responsibilities* appear
  - Where these are is a modelling choice that you make
  - Often hard to work out

- Mysterious unnamed object...
The environment

• The unnamed object is something outside the system boundary
  – something in the environment
• The environment, and what it *does* needs to be modelled
• Things to worry about:
  – What makes other things happen?
    • Users, other systems
  – What information is in the environment?
Generalisation

• Classes may be defined as a specialisation of another
  – so, a car is a vehicle

• Substitutability
  – “If you ask me for a vehicle and I give you a car, then that’s fine”
  – If birds are things that can fly then what are penguins...?
  – Or: there must be an association with the generalised concept
Object behaviour

• How to define object behaviour in general?
  – Sequence diagrams just show one particular scenario

• You could:
  – Define message signatures and wave your arms lots
    • Flight(plan:Flight Plan, aircraft:Aircraft)
  – UML Object Constraint Language
    • class invariants
    • operation pre- and post-conditions
  – Statecharts

  – All of the above...
Object Constraint Language

- A small example:
  ```
  context Flight
  inv: FlightPlan.sectors.includes(current)
  ```

- Also pre and post conditions for operations
  – reminiscent of a formal specification language like Z and VDM
Or just add a comment!
Statecharts

- Finite state machines in UML
- Often used to describe states of instances of a class
  - edges $\approx$ messages
  - you can use it for what you want
- Lots of UML syntax
Activities

• What if you need more than classes, objects, messages, etc.?
  – “the controllers do all this complicated stuff and they end up coordinating the flight into the next sector”

• Activity diagrams may be ideal...
Activity diagram

• Extension of statechart notation
• Less *software* specific than any other part of UML
• Overall view:
  – “swimlanes”
  – actions, activities, objects
Hints and tips for UML modelling

• Not a magic solution to all your modelling woes
  – it’s just a bunch of syntaxes
  – with some patchy tool support

• You could use the notations for whatever you want
  – normal use is as a way of constructing OO models
Precise

• Good models are precise – albeit abstract

• Just because it’s a model doesn’t mean that fluffy hand-waving is good

• Every single line, blob, arrow, word should mean something succinct
The system boundary and the environment

• What are you modelling and what’s outside?
  – How does what’s outside the boundary affect what’s inside?

• Should some parts of the environment be modelled?
  – for example:
    • concentrations of some substance
    • time of day...
Class modelling

• Are the classes concepts, specifications or implementations?
• A class is an *intensional* description of a set of specific instances which share the same behaviour
  – Ask yourself:
    • “what would a particular instance be called?”
    • “what behaviour would it have?”
      – That is, what messages could you send it?
• Would a class called “Oxygen” be a good model?
  – what is “an oxygen”?
  – what messages can we send to an oxygen?
• Class naming is important
  – singular noun name
  – <something> Manager and <something> Handler are not allowed
  – If you can’t think of a good name it’s a good sign that the model’s wrong
Generalisation

- ...is often a bad idea

- Which is the better model – why?
Generalisation

• ...is often a bad idea

• Which is the better model—why?

• Because it’s more abstract
Associations

• Appear on a *class* diagram
  – but say what objects are in another object’s neighbourhood
The relatives

• UML allows lots of different types of association
  – aggregation
  – composition
  – dependency
  – ternary
  – navigability
  – association classes
  – Occasionally useful but easy to over-use

• Simple rule:
  – all associations **must** have a cardinality at each end and
  – must have at least one role name (“has” is not allowed)
More UML...

• Only mentioned some parts of UML
  – Deployment diagrams
  – Use case models
  – Structure diagrams
  – Implementation diagrams
  – Protocol diagrams
  – Ports, links, constraints, etc., etc.

• The important bits have been discussed...
UML tools

• Lots of UML tools around
  – Many poor
  – Many broken
  – Some hilariously expensive
  – A couple aren’t bad
  – Whiteboards, pens and cameras are excellent

• The diagrams in these slides were done with Magicdraw.
Conclusion

• “Modelling” is a very broad term
  – make sure you know what you’re aiming to, for whom, etc.

• UML is just a set of notations that are useful for OO modelling
  – not necessarily the best choice for all circumstances
    • e.g. if you’re just looking at data structures then ER modelling might well be better
  – there are very sound reasons for going OO though
  – but you can always define your own semantics for any notation (!)