Modelling Objects

1 Characteristics of Objects

Objects have a distinct set of characteristics, not all of which readily lend themselves to modelling

Some key characteristics are:

- They are uniquely identifiable runtime entities
- Objects can be composed, that is, the object's data variables may themselves be objects
- The implementation of an object can be reused and extended through inheritance, to define other objects
- OO code can be polymorphic, using generic code that will work with related but different types

Inheritance and polymorphism are both difficult to include in a diagrammatic model

2 The UML

The Unified Modelling Language (UML) is now more or less the de facto standard for describing object oriented systems

Essentially it draws together (unifies) three earlier modelling forms, and does not do it particularly well. Empirical evidence suggests that is is not very widely used for developing models, although sometimes it is used for formalising them.

However, it has extensive documentation and tool support, so hard to avoid using it in some form

The practices of ”design methods” developed in the ”structured design” era tend to begin with the creation of single-viewpoint models, For example:

- Start with a functional model using DFDs
- Elaborate this into greater detail
- Possibly augment with STDs
- Transform the DFDs into Structure Charts to produce a constructional model

In contrast, OO methods tend to begin by identifying ”real-world” objects and all their interactions. So from the start, the design model needs to describe many characteristics, based upon several viewpoints

A major criticism of the UML is that the outcome is a set of notations that are:

- Overly complex and unwieldy
- Lack any overarching ”viewpoints” concept, making it hard to integrate them in any sense
- There is also some overlap between many of the notations (diagram forms) because of the diverse set of forms used in the earlier OO modelling that were merged into the UML

3 Statecharts

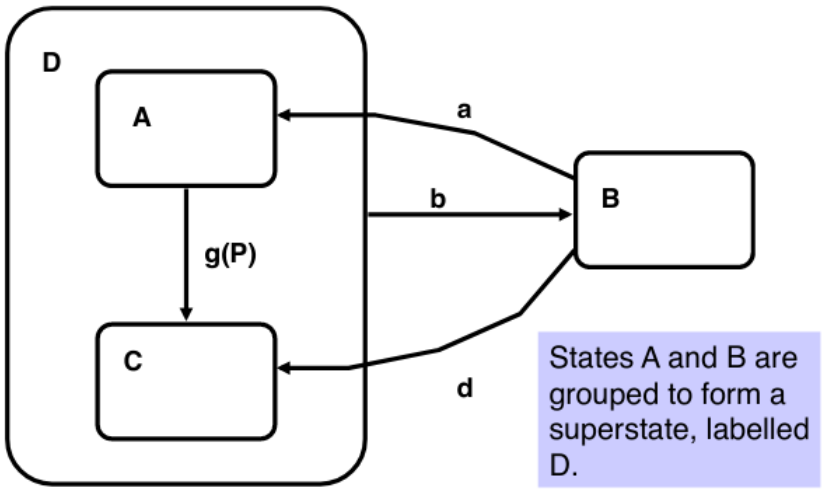
The Statechart has a mathematical underpinning. Claimed to result in a model that is relatively unambiguous and less dependent upon text to provide semantic interpretation than most diagrammatic forms

The notation supports:

- Clustering of states to form super-states
- Orthogonality of states
- Stepwise refinement of states

Much of this is now incorporated in the UML

3.1 Hierarchy in statecharts


D
A
a
b
B
g(P)
C
d
States A and B are grouped to form a superstate, labelled D.

4 The Sequence Diagram

Definition: Sequence Diagram

A diagram that shows object interactions arranged in time sequence. In particular, it shows the objects participating in an interaction and the sequence of messages exchanged

A sequence diagram visually describes a pattern of interaction arranged in time order

The sequence diagram describes the system from a functional viewpoint: it is largely concerned with how interaction amongst objects is organised

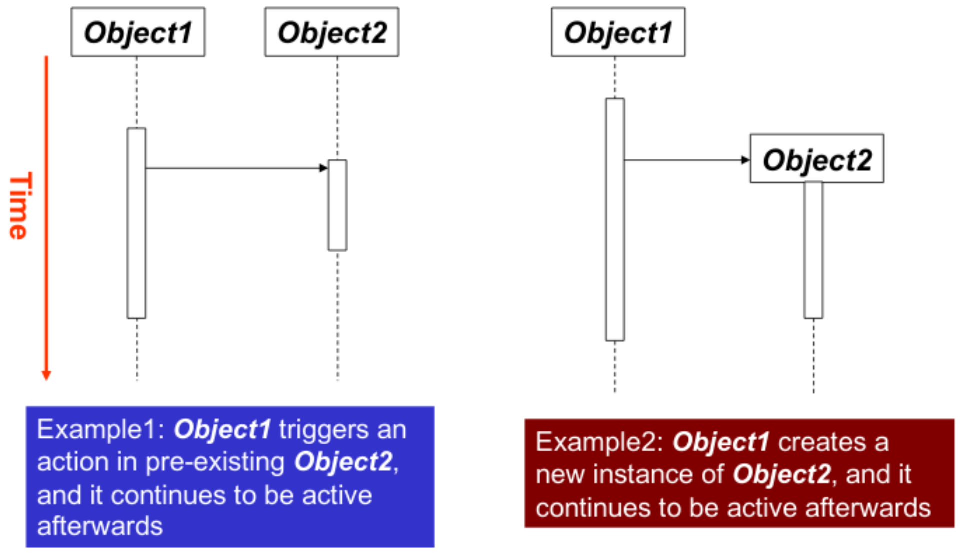
4.1 Variations

The sequence diagram can have two roles:

- Descriptor form - mostly used for the purposes of systems analysis and tries to describe all possible scenarios for object interaction
- Instance form - Mostly used in requirements modelling and design, where we might want to model the specific object interactions that occur in one actual scenario

4.2 Notation

- The vertical dimension is used to indicate time, generally proceeding down the page
- The horizontal dimension represents each of the objects that are to take part in a pattern of interaction: each object is shown in a separate column
- An object symbol (a rectangular text-box) appears in its column at the point in time that the object is created. The object's lifeline descends from it and is a dashed line if the object is acquiescing or a long thin rectangle if it has an active focus of control
- Messages between objects are shown as solid horizontal arrows between the lifelines of the sender and recipient


Object1
Object2
Time
Object1
Object2
Example1: *Object1* triggers an action in pre-existing *Object2*, and it continues to be active afterwards
Example2: *Object1* creates a new instance of *Object2*, and it continues to be active afterwards

4.3 Additional features

Sequence diagrams can have lots of optional features including:

- Timing constraints
- Distinguishing between asynchronous control and procedural control
- Showing deletion of objects

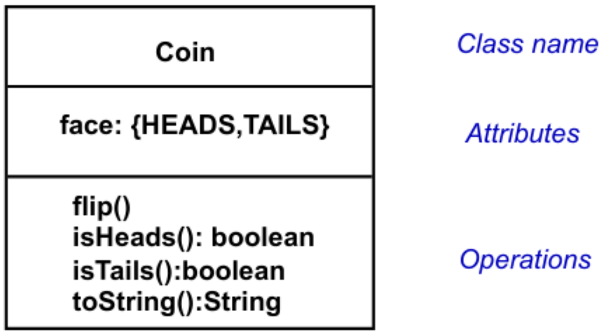
The advice is to keep it simple and use the instance form for any initial modelling sketches

5 The class diagram

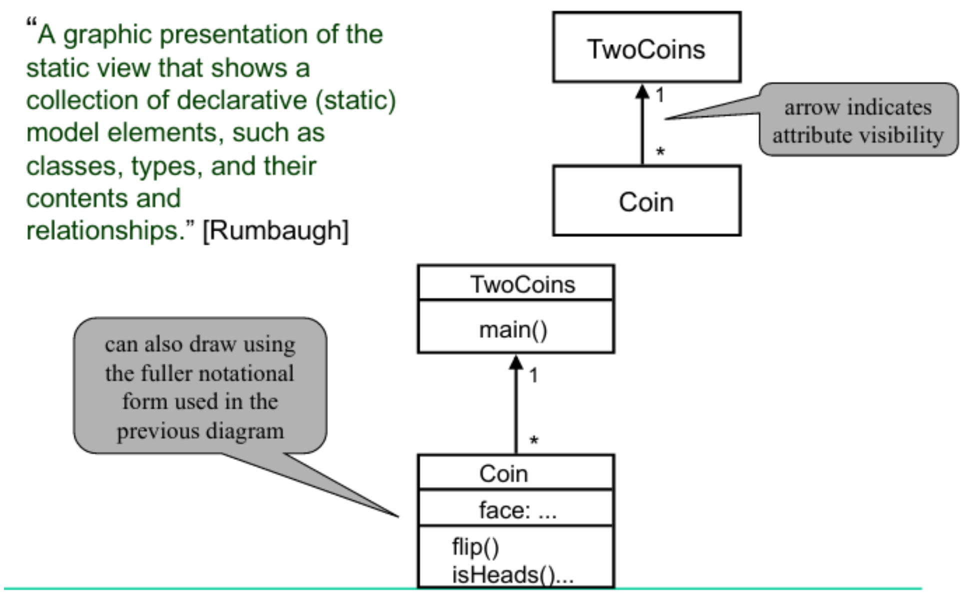
Can be used to model objects that do things and also data objects. (here it acts as an entity-relationship diagram, although empirical studies show that it provides a poorer visualisation than an ERD)

Different levels of detail in the formalisation (can add information about methods, data etc in the boxes)

This notation is widely used, including by many tools



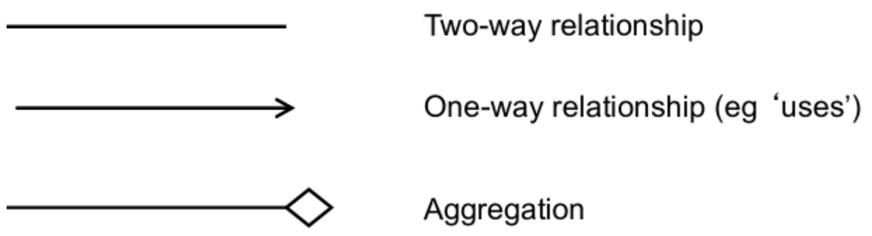
Converted to UML it would look like this

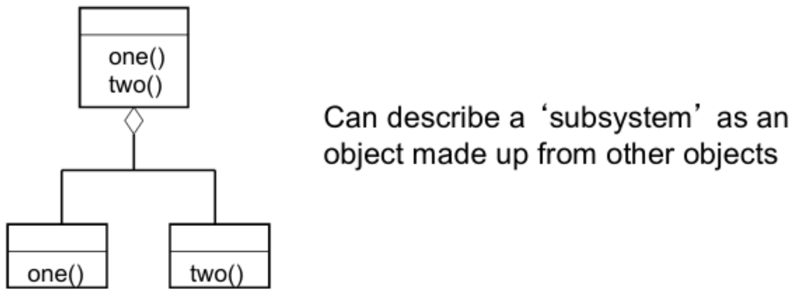


"A graphic presentation of the static view that shows a collection of declarative (static) model elements, such as classes, types, and their contents and relationships." [Rumbaugh]

TwoCoins
1
arrow indicates attribute visibility
*
Coin
TwoCoins
main()
can also draw using the fuller notational form used in the previous diagram
1
*
Coin
face: ...
flip()
isHeads()...

The arrow heads mean as follows




one()
two()
one()
two()
Can describe a 'subsystem' as an object made up from other objects

6 Analysis

The abstract nature of diagrammatic models makes it difficult to conduct formal and rigorous analysis

Models typically address ideas about:

- Information
- Behaviour
- Architecture
- Application domain
- Business/enterprise

Analysis usually addresses such characteristics as:

- Completeness and consistency
- Correctness
- Dependability

6.1 How do we a analyse such models?

Various techniques, just mention two here

- Tabulation for cross checking (completeness)
  - A bit like the earlier example of an STT. If we tabulate different relationships, we can check our model for any omissions, or inconsistencies. A simple but often quite effective method
- Use of walkthroughs and scenarios (correctness)
  - Usually a team thing, involving walking through a model working out how it will behave in response to various stimuli and under specific conditions
  - The basis is usually one or more scenarios of use