UML big picture

Perdita Stevens

School of Informatics
University of Edinburgh

Plan

- History of UML (quick reminder)
- Parts of UML
- How it all fits together
- UML as a language
- Consistency: what does it mean, do we need it?
- Defining UML
- Metamodelling

Whence UML?

Recall: 1980s/early 90s: explosion of OO

Plethora of gurus, each with own company, tool, book, modelling notation

including Booch, Rumbaugh, Jacobson, the "three amigos" who originated UML, the Unified Modeling Language... ...and Coad, Odell, Schlaer and Mellor, Wirfs-Brock...

Standardisation controversy

Journal of Object Oriented Programming, vol 6 no 4, July-August 1993, includes both:

- a guest editorial by Jacobson entitled "Time for a cease-fire in the methods war";
- an Open Letter to the Industry entitled "Premature methods standardization considered harmful", signed by 8 methodologists (but *not* Jacobson)

Resolution

1994: Rumbaugh joined Booch at Rational (Booch's company, since bought by IBM).

"The methods war is over: we won" said Booch and Rumbaugh.

1995: proposal for a Unified *Method*. Jacobson joined them at Rational.

OMG pulled the community together into standardising UML 1997 UML1.0... March 2015, UML2.5...

Persistent issues around what "standardising" should mean: how standard, how formal?

Parts of UML

Parts of UML you now know about:

- Use case diagram: summarise requirements
- Class diagram: static structure
- Sequence diagrams: how objects interact (inter-object behaviour)
- State diagrams: how an object's state changes when it receives messages (intra-object behaviour)
- Activity diagrams: workflow; how activities are organised.

The rest of UML

Several other diagram types we don't cover, including:

- Three other kinds of interaction diagrams! (Sequence diagrams are the most useful, and I reckoned one was enough.)
- Deployment diagrams, to show how processes in the eventual system are deployed to hardware.
- Package diagrams, to show how the overall system and its namespace is organised.

Object Constraint Language: a textual language for constraints, i.e. expressions that may be true or false in a context.

Facilities for defining profiles, i.e. variants of UML for particular domains: may be briefly mentioned later.

You don't need to use it all!

No project uses all of UML; many users only ever use a small subset, e.g., just class diagrams and sequence diagrams.

Depends on needs of project: e.g., embedded system development often uses state diagrams extensively, business software development seldom needs them.

(Several recent papers have investigated what parts of UML are used how much, but they use convenience samples...)

We'll come back at the end of the course to talk about how the available tools influence modelling.

Consistency

A source of confusion and bugs is different diagrams/documents contradicting one another.

A set of diagrams is consistent if there is at least one way to implement a system that they all describe.

E.g. classes and operations mentioned in a sequence diagram had better also occur in the class diagram.

This is embedded in the definition of the modelling language: an inconsistent set of diagrams will not constitute a legal model (just as a Java "program" that doesn't compile isn't really a Java program).

UML as language

A language, which can be considered a set of possible utterances, has:

- syntax: is an utterance part of the language? (Legal, grammatical.)
- semantics: what does a legal utterance mean?
- pragmatics: what conventions do people use, e.g. which of the legal utterances seem most natural?

UML syntax

Rules of diagrams, such as you've been learning.

Defined in the UML standard using metamodelling. This is (largely) automatable: tools can and to some extent do check that your UML diagram is legal UML.

("largely" and "to some extent" because some of the rules are complicated, and many are not checked in any tool)

UML semantics

Think of a diagram as a statement about a system.

The job of UML semantics is to explain what a diagram says about the system.

Typically infinitely many software systems could be described by a given diagram – e.g., class diagram says what classes there are and what operations they offer, but says nothing at all about what those operations do.

Defined in the UML standard using English. Huge mountain of work on formal semantics, motivated by desire to provide better tools.

UML pragmatics

Covers how UML is typically used; which varies between contexts. You've heard me talk about "what people usually do".

Example: layout of class diagrams has no meaning in UML, but people arrange them with the most important classes in the middle, with as few crossing lines as they can manage, with generalisation arrows going up, with a preference for association names to be read from left to right.

Language designers may think they can ignore pragmatics, but sometimes not: e.g., need to record diagram layout if models are to be portable between tools in practice.

How to define syntax of a graphical language

We have to give abstract syntax, i.e.

- what the parts are (what kinds of model element are there? e.g. Class, Association, Generalization...)
- how they can be related (e.g. you might specify that any Generalization must have a Class which is its supertype and a Class that is its subtype)
- what the restrictions are (e.g., no Class can be connected to itself by a chain of Generalizations).

and concrete syntax: how is all this represented graphically?

(Class represented by rectangle, Generalization by triangle-headed line, the triangle connected to the supertype Class and...)

Metamodelling

Defining the abstract syntax of a modelling language is very similar to describing the objects and their relations in an OO system!

"This model is correct in the modelling language"

is a very similar question to

"This collection of interlinked objects is described by this class diagram"

So we can define UML's abstract syntax in UML... or better, in a small subset of UML... or better, in MOF (Meta Object Facility).

MOF

Meta Object Facility

Standardised by OMG in order to help with the definition of the abstract syntax of UML.

Think of it as a small subset of UML class diagram notation: includes classes, attributes, associations, generalization.

Sufficient to express the metamodel of a real modelling language: simple enough to be shared by many modelling languages and form the basis of technology for model repositories, transformations etc.

Warning: this is a simplified story – the actual relationship between MOF and UML is now somewhat incestuous! We will see a bit more about language definition later in the context of domain-specific modelling languages, when we'll talk about the OMG 4 level Metamodel Hierarchy.

Advantages of metamodelling

- Reasonably understandable, yet reasonably precise, definition of abstract syntax
- Helps structure tools
- Makes it possible to define other modelling languages using the same metamodelling language... on to domain-specific modelling languages, model-driven development etc.