# Advanced Tools for UML:

## now and in the future

Perdita Stevens

Division of Informatics

University of Edinburgh

http://www.dcs.ed.ac.uk/home/pxs

# Objectives

At the end of this tutorial, participants will:

- understand the range of capabilities present in today's UML tools;
- have a broad overview of the capabilities and limitations of today's technology in verification and related areas, and its relevance to UML;
- have gained (including by discussion among the participants) increased understanding of what the future of support for UML could and should be.

# Plan

- Tools now: overview, shortcomings and examples
- Tools in the immediate future (XMI, SVG)
- How to evaluate a tool

*BREAK*

- State of the art beyond UML
- Tools in the future:

  - biased towards my interests, may be hijacked

Do feel free to interrupt.

# Good tool support is...

[for our purposes]

... anything that helps you get the job done better, faster, and/or for less money.

The job we will be mostly concerned with is software *design*, recorded using UML.

(But this can't exist in isolation from requirements analysis, coding, testing, configuration management etc., which may affect criteria for a good design tool.)

## What’s distinctive about...

design?

UML?

# Design

Designers put things together and bring new things into being, dealing in the process with many variables and constraints, some initially known and some discovered through designing. Almost always, designers' moves have consequences other than those intended for them. Designers juggle variables, reconcile conflicting values, and maneuver around constraints – a process in which, although some design products may be superior to others, there are no unique right answers.

Donald Schön

# Why consider the future?

Opinions differ, and today's tools do many good things.

But:

- *pencil test:* plausibility of claim that for some, pencil and paper are a better option

- *popularity of very simple tools* e.g. plain drawing tools

- a certain *antipathy to tool vendors!*

Claim: there is scope for improvement.

# What can commercial UML tools help with now?

- the mechanics of drawing and exporting UML diagrams
- eliminating errors
  - (some) syntactic errors in individual diagrams
  - (some) consistency errors between diagrams
- model- and document- linking, report generation, configuration management
- code generation and reverse engineering; limited simulation
- metric collection

NB *not* the actual design – and only given good usability and reliability

# What tools are commercially available now?

There are now about 75 UML tools listed at:

`http://www.objectsbydesign.com/tools/umltools_byCompany.html`

Prices range from free (e.g. Argo/UML) to $9000 / seat (Objecteering).

Vary widely also in: functionality; usability; performance; platform; completeness; faithfulness to standard.

# Classifying UML tools

1. Basic diagram-drawing tools (a dying breed!)

   Examples: Visio (basic version)

2. Main-stream OO CASE tools

   Examples: Rose, Together, Argo, ....

3. Specialist real-time/embedded tools: exploit more detailed models to give full code-generation, animation of state machines etc.

   Examples: Telelogic family, Rhapsody

# What’s the problem?

3 main categories of complaints:

1. Usability
2. Reliability
3. Power

Almost everyone will cite usability as the most important.

# Note of caution

*Usability is indeed key and still much underemphasised.*

But how usable is your word processor? Your window manager? Your compiler?

People will put up with “poor usability” if they must.

E.g., if the “unusable” tool does something they really want done and don’t want to do by hand.

And UML is inherently complex; perhaps the tools cannot be very easy. Perhaps power is really the key.

# Specific examples

We'll look *briefly* at examples from each of the three categories.

Visio (basic version)

Rational Rose

Argo/UML

Together

Telelogic's family or tools

# Visio

A good drawing tool: do you really need more?

Free UML templates from Pavel Hraby:

http://www.navision.com/services/methodology%20forside.asp

> “These templates have no repository or intelligence, so I like them because they get out of my way and let me draw what I’m trying to communicate.”

Martin Fowler

(The Enterprise Edition does code generation, but is not for UML accuracy fanatics.)

# Rose (Rational)

- One of the first, but
- Long criticised for being incomplete and out of date; Rose 2000e seems better.
- Features galore
  - strong emphasis on round-trip engineering
  - integration e.g. with ClearCase, MS VisualStudio
  - support for documenting use of XML, EJB...
- Real-time variant.

# Argo/UML

- Free: open source
- “Critics”: run-time criticism of design leading to to-do items for the designer to consider.
- Under active development: current version is 0.8 (seqence diagrams and reverse engineering, long missed, soon to be added)
- Two companies involved in support and development (Tigris and Gentleware)
- Good on UML accuracy: uses third-party UML metamodel

# Together

- Strong on usability
- Unusually strong focus on code, e.g. as basis for repository: real-time update, automatic layout.
- Automatically-generated seqence diagrams via built in JVM

Review based on Objects By Design criteria at www.objectsbydesign.com

# Telelogic

- Real-time focus; pedigree of impressive functionality with ObjectGeode.
- Recent acquisition of ObjectTeam's COOL:Jex.
- Still seems aimed squarely at specialists in real-time:

  > "By using UML for requirements capture and analysis, SDL, ASN.1 and MSC for real-time design and implementation, and TTCN for scriptable testing, the developer working with Telelogic's tools takes full advantage of the most appropriate languages in each phase."

`http://www.telelogic.com/solution/lang.asp`

# Visible on the horizon

I think it's pretty clear that we'll see:

- convergence with stabilising standard (inc. provision of stereotypes, tag-value pairs etc.)
- convergence of core functionality, e.g. dynamic connections with code
- more use of XMI...
- scalable vector graphics (SVG)...

And perhaps price falls?

There are plenty of other things that we *could* see, of course...

# XMI

XMI = XML Metadata Interchange Format

XML = eXtensible Markup Language

- what is it?
- what is it good for?

# What is XMI good for?

> “The main purpose of XMI is to enable easy interchange of metadata between modeling tools (based on the OMG UML) and between tools and metadata repositories (OMG MOF based) in distributed heterogeneous environments.”

- distributed working with people using different tools;
- avoiding lock-in to a tool;
- repository querying;
- generating HTML documentation.

Wider than just UML.

# What is XMI?

Spec includes:

- Design principles.
- A set of XML Document Type Definition (DTD) production rules for transforming MOF based metamodels into XML DTDs.
- A set of XML Document production rules for encoding and decoding MOF based metadata.
- Concrete DTDs for UML and MOF.

OMG doc 00-06-01

Does allow for extensions, incomplete models etc.

# Example: produce HTML from XMI

Start with your UML model saved as an XMI file.

In XSL (XML Style Language) write a pattern-matching style sheet saying what information to extract from the XMI document and how to turn it into HTML.

For example, produce automatically updated web pages detailing the attributes and operations of each class.

# Advantages

- Eliminate one of the most tedious manual documentation tasks
- Help with keeping documentation in step
- Supports distributed development

Largely open question: what *else* can we do with XMI?

# Caveats

1. XMI/UML are co-evolving and settling down.

   Different combinations of UML versions and XMI versions exist: only an exact match will enable tool-to-tool interchange.

   XMI1.0, 1.1

   UML1.1, 1.3, (1.4)

2. XMI doesn’t (fully) specify how to record graphical information; tools do this differently.

# Scalable Vector Graphics

A newish W3C standard based on XML.

Status: Candidate Recommendation (2nd August 2000)

SVG graphics:

- are smaller than e.g. JPEG, GIF
- are easily scaled, animated, connected with semantic info
- allow for raster effects and text as well as vector graphics
- are designed for Web use

`http://www.w3.org/TR/SVG/`

# SVG and UML

Tools are beginning to provide SVG output (e.g. MagicDraw, Argo/UML: Argo's is not yet release quality though.)

Seems promising as a way of publishing UML diagrams on webs.

Perhaps scope for more, e.g. semantically informed tools that operate directly on the SVGs?

Largely open question: what should the relationship be between SVG and XMI?

# Choosing a UML tool

Questions to ask yourselves include:

- what difference do you want it to make?

- what will patterns of use be?

- will you be tied in?

and especially,

- do *you* like the trial version?

Consider the Objects by Design evaluation criteria (and example review of Together/J). (www.objectsbydesign.com)

# What effects do tools have on working practices?

- + increase focus on design?
- - eat time and reduce flexibility?
- + save coding time ???
- - false sense of productivity?
- + better documentation?

# Patterns of use

Who is going to use the tool, how often, for how long?

Affects usability criteria – ease of use $\neq$ ease of learning

(Beware: hard to evaluate ease of use briefly!)

If people will not use the tool enough to become familiar with it, consider paper or a plain drawing tool instead.

# Will you be tied in?

Given the immature state of the UML tool market it’s not likely you want to choose a tool now for always.

An early aim of UML was to make it easier to move models between tools.

XMI will help you soon, but not necessarily today.

Some tools provide import/export in Rose formats - but do try it...

# Objects By Design's list

- Repository Support
- HTML Documentation
- Pick Lists
- Versioning
- Printing Support
- Exporting Diagrams
- Robustness
- New Releases
- Round-Trip Engineering
- Full UML 1.3 Support
- Data Modeling
- Model Navigation
- Diagram Views
- Scripting
- Platform

# Discussion slot

- What experience do you have with UML tools?
- What's good/bad about the tool(s) you know?
- What effect has your experience had on what you'd look for next time?
- What advice would you offer others?

# Tools for design?

So we claim that few of the capabilities of today's tools really help to solve design problems.

Why?

# Why don’t tools help with design?

Perhaps

> There’s no need.
>
> People are good at design, it isn’t a problem.

So no released software ever contains important design flaws?

# Why don’t tools help with design?

Perhaps:

> It’s impossible.
>
> Design is such an inherently human activity that nothing a tool could do could possibly help with it.

So nothing computers are good at is relevant? Keeping track of details, applying boring rules to get new information?

# Why don’t tools help with design?

Perhaps:

> There’s no demand.
>
> People who buy design tools don’t want the tools to help with design.

But if there is a problem with design and it is possible to help with it, wouldn’t people pay for that?

# Why don’t tools help with design?

Perhaps:

We just haven’t got that far yet.

Design today is “pure craft” just because we haven’t yet worked out what the right tool support is, not because none is possible.

Even though there will presumably always be an element of craft, there is lots of scope for tool support.

# Example from Constantine

Larry Constantine observed in 1993 that designers want to be able to compare alternatives.

E.g., to have two versions of the same diagram side by side.

“whiteboard with history”

Model management, consistency and UI issues... but not so hard?

# Research efforts

Much of the UML research effort has been focused on formalising and answering questions like:

- is this UML model legal?
- is this UML model a correct refinement of that one?
- does this code correctly implement this UML?
- does this code respect these OCL constraints?

In general, focus on *verification*.

# Verification and UML

Much of the research interest in these questions arises from the difficulty of answering the questions at all, not from the difficulty of having tools answer them.

That is, the UML standard itself still contains ambiguities and contradictions.

Many are being addressed (e.g., several listed in UML2.0 roadmap)

But how, in principle, should they be addressed?

# What is a “clear” UML standard?

At present the informal semantics work mostly by saying:

if the UML model is like this...

... then any system implementing the UML model is like this.

The system description is informal and largely PL-independent.

For humans, this is good, probably best possible.

# What is a “tool-friendly” UML standard?

Tool builders like to know what answers their tools should give before they try them. (Automate a mess, get an automated mess.)

But human understanding $\not\Rightarrow$ feasible tool understanding.

E.g.

1. because tools can’t brush PL differences under the carpet and see what’s important... and may not need to;
2. because tools are given more complex problems than humans can solve – that’s why we want them – which can show new problems.

# What “is” the language anyway?

Three elements:

1. syntax: what it means to be a legal utterance
2. semantics: what a legal utterance means
3. pragmatics: everything else!

# What does “means” mean?

Several different approaches:

- denotational semantics: e.g. a UML model “means” the systems that may implement it (in some fixed language, not nec. a PL).
- operational semantics: e.g. a UML model “means” what it may do (good for state machines, what about the rest of UML?)
- axiomatic semantics, e.g. Hoare rules (good for rewriting bits of UML...??)

May have several, with appropriate consistency.

# So, do we need formal UML semantics?

Not necessarily.

Argue: we have good support for programming, though no commercial PL has a complete *formal* semantics. Tool builders manage.

PLs e.g. Java have something close to a *formalisable* semantics: i.e. so unambiguous that formalisers wouldn't run across any unanswered problems.

Design languages have problems not found in PLs, though; UML is a long way off that.

# Challenge

Get something which:

- is human understandable
- is genuinely usable by tool-builders
- deals with the whole of UML (somehow)
- is correct (both: agrees with intended semantics and: doesn't introduce formal nonsenses)

Translations, from UML to something else or from UML to a subset of itself, can play a role in both. (Can damage clarity though.)

Let us consider the needs of the different questions.

# Is this UML model legal?

Reasonably easy to do;

may expose problems with the UML definition (no consistency argument);

not very obviously useful;

but underlies other capabilities, which will expect only to be applied to legal UML models.

$\Rightarrow$ formal UML syntax $\Rightarrow$ denotational semantics of OCL

# Is this UML a refinement of that?

(Refinements within UML specifically listed as to be better defined in UML2.0)

Two possible informal views:

1. Yes, iff any code that implements this also implements that;

   $\Rightarrow$ denotational semantics of UML (a UML model denotes the set of legal implementations of it)

2. Yes, iff we can tell (directly, without reference to PL) that this UML gives more detail than that;

   $\Rightarrow$ operational/axiomatic semantics of UML

# Does this code correctly implement this UML?

Would allow more reliable, and perhaps more flexible, round-trip engineering.

Work needs to be redone for each programming language.

$\Rightarrow$ denotational semantics of UML (or more than one!)

Much more realistic: UML models become tests (ACW); then a system implements a UML model iff it passes the tests...?

# Does this code respect these OCL constraints?

OCL used in definition of UML, but can also be used for contracts of components etc.

Would like to be able to verify that the contracts are adhered to;

- either statically,
- or (less ambitiously) by inserting assertions in code.

Problems with OCL: inherently inconsistent. Resolvable (cf Monday's workshop), but resolutions not yet in the standard.

# Looking outside the community

In closely related tool communities technology transfer is going on: e.g.

- from the SDL (Telelogic) and the Schlaer-Mellor (BridgePoint Automation) tool worlds ($\rightarrow$ model animation, more complete code generation).
- from the academic tool communities, e.g. groups in Munich under Broy, Tampere under Koskimies, etc. etc. ($\rightarrow$ visualisation, model-checking of temporal logic formulae, theorem proving for checking abstractions...)

Now let us consider some more diverse tool areas, in search of ideas.

# Verification tools

Models: process calculi, automata, with LTS semantics. Usually finite state.

Capabilities:

- simulation (including with non-determinacy);
- checking of many relationships (trace equivalence, bisimulation, testing equivalence, etc. etc.) between processes;
- model-checking: is a logical statement true of a process?

Being applied already in research UML tools.

# Challenges/limitations

Most industrial-strength verification techniques and tools assume systems are finite-state (though very large).

vs

Data and/or collections of objects etc. naturally modelled as infinite [families of] systems.

Finite abstractions can be used but are limiting.

True infinite-state verification is a very active area, but hard: all interesting general problems are undecidable.

Challenge: make easy questions easy.

# Editors

- macros, regexp based editing etc.
- graphical differencing, e.g. ediff
- integration with VC tools
- interactive spell-checking
- powerful customisation
- choice of efficient (keyboard) or easy (menu) interaction.

Significant influence already: UML tools with scripting languages, etc.

# Programming (languages and) tools

- type-checking
- syntax-sensitive editing
- interactive debugging
- profiling

Some simulation appearing in tools for “model-level debugging”. UML metamodel can roughly be seen as providing a type system for models...

# Refactoring tools

OO programming community: e.g. refactoring browser for Smalltalk, and now Java.

`http://st-www.cs.uiuc.edu/users/brant/Refactory/RefactoringBrowser.html`

Improve design of existing code; e.g. automate common transformations like factoring out a common superclass.

Integration with UML models?

# Patterns tools

Three main approaches:

1. Simple documentation aid: templates for structures of patterns. Exists in many tools.
2. Assistance in changing design to use a pattern – help avoid slips. E.g. ADEPT project Ó Cinnéide, UCD.
3. Some tools also try to identify where patterns should be used: but is this possible in principle?

# Static analysis tools

General term for tools that analyse code without evaluating it.

- enforcement of coding standards
- dependency analysis
- Y2K bug detection

Some ability to deal with incomplete code.

Largely open question: how to adapt these techniques, e.g. slicing, to UML

# Supporting Software Design

How can the practice of mainstream software design be well supported by formal techniques?

Two possible approaches:

1. “Money no object”: how can formality help raise the ceiling, increase the maximum dependability of systems? “Formal methods”
2. How can formality help *without* raising cost or requiring software engineers to be more mathematically oriented than they are now?

I’m personally most interested in 2.

# Open questions 1

How can what-if experiments be made easier? To what extent could a tool suggest scenarios which should be explored, provide support in exploring the dynamic implications of earlier decisions, etc.?

# Open questions 2

How can design by contract be supported?

- Languages for appropriate contracts, which a tool can treat as formal statements, without requiring the designer to learn a formal language?
- Unfinished design? Tool must support the identification of (in)dependent elements, and must not simply give up in the case of a possibly violated contract.

# Open questions 3

How can we ease the use and development of components and other chunks of designs, such as product-line architectures (PLAs) and frameworks?

- complex interfaces (not just single operations)
- restrictions on how choices are resolved (e.g. who chooses?)
- exploring, and perhaps changing or making explicit, the assumptions made about the environment.

# Games for software design

Technology imported from verification tools.

Tool can help designer explore *what happens if* certain choices are made...

Allow tool to challenge in areas which are considered robust; understand what needs to be done; incorporate that in detailed design.

Experimental: but reason to hope useful esp. in presence of non-determinism (uncontrolled choices).

# Design by contract

At present most contracts are written in English.

Automated support for contracts in OCL is beginning.

What about

- temporal properties (deadlock, liveness...)?
- user-friendliness; can graphical syntax help?
- unfinished design? Must not simply give up.

# Product-line architectures etc.

Challenges include:

- representation (of PLA, its instantiation, the relationship);
- how to help with design of PLA: eliminating flaws that *could* arise with *some* instantiation;
- how to help with instantiation: what does “correct” mean and how to deal with flexibility here?

Problems also relevant to frameworks, components, configurability...

# Discussion slot

- Ideally, what would you like a design tool to help with?
- Is it feasible today?
- If not, why not?

# Last words

I hope you've got something out of today.

Please visit

`http://www.dcs.ed.ac.uk/home/pxs/UML2000`

(next week!) and fill in the evaluation form.

Moving on:

I am just beginning a 5-year period of researching advanced UML tool capabilities full-time.

I'm keen to talk with developers and users of UML tools: feel free to drop me an email any time (Perdita.Stevens@dcs.ed.ac.uk).