Getting value from UML tools

Perdita Stevens

Division of informatics

University of Edinburgh

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

What kind of session is this?

Not finished, tried-and-tested “do it this way and all will be well”.

Rather, discussion of a set of problems and a technology that can form part of a solution.

The beginning of an interesting story, not the end.

Feel free to ask questions and offer comments as we go.

Plan

- UML tools state of the art
- UML tools in the development process
- XMI the technology
- Using XMI with UML
- And the effect will be...?
- Conclusion

UML tools state of the art

You start!

Do you use a UML tool?

Which one? How did you choose?

What do you like about it?

What do you hate about it?

How much is it used and by whom?

UML tools state of the art

Current UML tools can help with:

- 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, CM
- code generation and reverse engineering; limited simulation
- metric collection

UML tools in the development process

UML tools in the development process 1

Picking out the most relevant points:

- model- and document- linking, report generation, CM
- code generation and reverse engineering; limited simulation

Several of the largest tool vendors have put much emphasis on tool integration.

Good, provided you’re happy with the tools your vendor considered and the kind of integration they support.

UML tools in the development process 2

So why do tools so often gather electronic dust?

- they’re better at post hoc recording of design than at helping with design (but that’s a another talk...);
- it takes too much effort to keep UML models and other documentation in step, and can tie you to one tool;
- they’re hard to integrate with your own other tools and processes.

XMI is just one step in addressing this problem – but an important one.

Models are not malleable

Programmers can easily manipulate their code in minutes or hours, e.g. from my own experience

- “grep”ing to find relevant sections of code quickly;
- writing small scripts to find/correct minor problems;
- extracting comments and formatting them for use in an in-house, non-standard help system;
- extracting information about method signatures etc. for insertion in documentation.

And this matters because...?

Why should we care that models are not malleable, tractable?

Because this is what lies behind the rush to code and behind the mistrust of modelling.

If developers are to *trust* their models, they need to *own them, hit their heads against them, falsify them.*

There is no fundamental reason why this has to imply UML as programming language.

Indeed, there are good reasons why it should not.

Take-home slogan

We’re used to easy things involving UML models being hard.

XMI can help make easy things easy

(and some hard things possible: a few words about that at the end)

XMI the technology

XMI the technology

XMI = XML(-based) Metadata Interchange Format (OMG standard)

XML = eXtensible Markup Language (W3C standard)

> “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.”

But actually, it can do a lot more than that.

Necessary background

Before we can talk about how XMI can help we need to review the basic technologies involved in UML and XML.

I assume that probably:

- everyone knows what UML is!

  - but that perhaps not everyone knows how it’s defined?

- everyone has heard some XML hype

  - but that perhaps not everyone has delved behind it?

How UML is defined

Two main documents within the OMG UML standard:

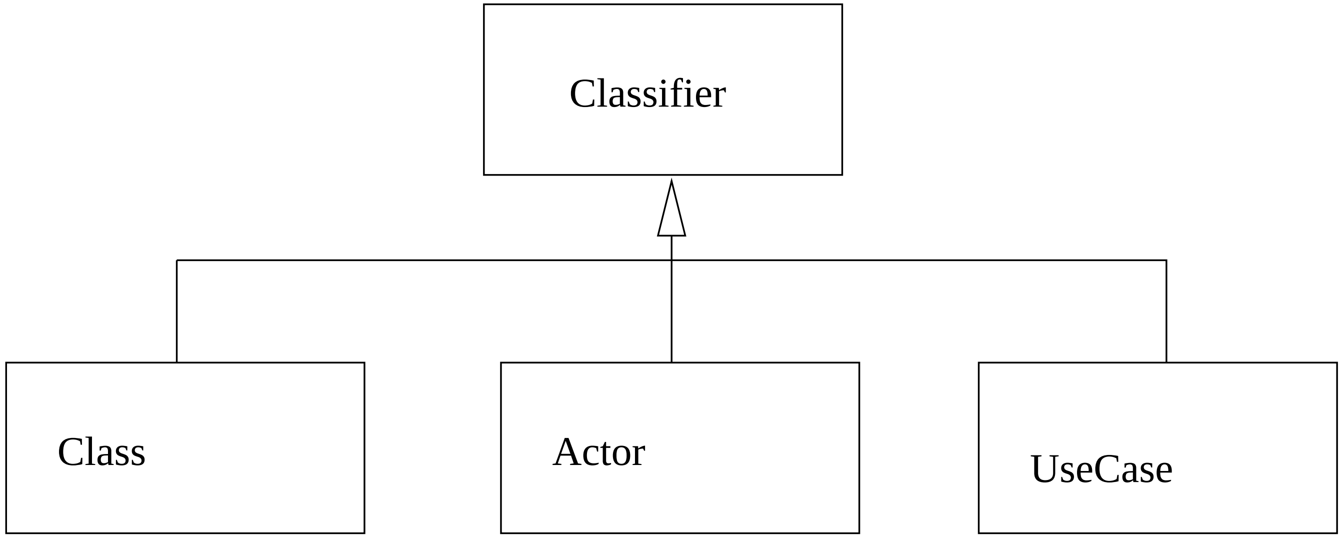
- Notation Guide : informal explanation of notation (concrete syntax) and its connection to abstract syntax.
- Semantics : *semi-formal specification of abstract syntax*, plus further explanation of semantics.

Semantics takes precedence over Notation Guide in cases of conflict – theoretically.

Plus: definition of the Object Constraint Language (OCL).

UML semantics document: syntax

The **abstract syntax** describes (in UML!) the relationships between kinds of UML model elements (the *metamodel*). For example, use cases, actors and classes are all said to be examples of classifiers:



Just enough XML

XML stands for

eXtensible *Markup* Language

but it’s more revealing to think of it as

(eXpressive?) *META* Language

Its strength is that you can easily use it to define simple languages for describing domain-specific structured data.

XML documents are trees...

described as plain text. e.g.

```
<XMI.header>
  <XMI.documentation>
    <XMI.exporter>Together</XMI.exporter>
    <XMI.exporterVersion>4.0</XMI.exporterVersion>
  </XMI.documentation>
  <XMI.metamodel xmi.name = ’UML’ xmi.version = ’1.1’/>
</XMI.header>
```

Defining languages in XML

To define a domain specific XML language you record which XML documents are valid *in your context* by defining a DTD or schema.

Good simplicity/power tradeoff is XML's secret of success:

- XML tools (parsers, editors,...) only have to understand XML; they don't *have* to know anything about your particular XML language.
- Or they may: e.g. a validating parser checks that a document matches its DTD as well as checking that it's proper XML.

UML + XML = ?

So a UML model is not just boxes and lines: it’s *structured data*, structured according to the UML metamodel.

For example, if there’s a generalization there must be two generalizable elements, the subtype and supertype.

XML is a way of defining languages of structured data.

So they are a natural match. But how do they fit together?

Homing in on XMI

XMI = XML Metadata Interchange

an OMG standard.

Most importantly,

XMI is a way to save UML models in XML.

In fact it’s more general.

UML is a MOF-based metamodel;

XMI shows how to save any MOF-based metamodel in XML

Short digression on MOF

MOF = Meta Object Facility

an OMG standard

MOF is a simple language for defining languages, e.g., UML.

(Strictly UML’s metamodel does not fit MOF exactly, so UML also has a *interchange metamodel* which does – hack!)

MOF looks remarkably like a small subset of UML; notated with UML.

OMG 4 level metamodel architecture

META-LEVEL	MOF TERMS	EXAMPLES
M3	meta-metamodel	“MOF Model”
M2	meta-metadata metamodel	UML Metamodel
M1	metadata model	UML Models
M0	data	Modelled systems

Why isn’t MOF just the same as UML?

From OMG-MOF1.3

> The key differences are due to different usage scenarios of MOF and UML. The MOF needs to be simpler, directly implementable, and provide a set of CORBA interfaces for manipulating meta objects. The UML is used as a general-purpose modeling language, with potentially many implementation targets.

MOF provides an “open-ended information modeling capability.”

Back to XMI

So, XMI is an attempt to take full advantage of

- UML’s world dominance
- UML’s definition in MOF
- XML’s world dominance

and produce *a standard way to save UML models in XML*, in order to provide modellers with the ability to *move UML models between tools*.

(Extras for free: e.g. same for any other MOF-based language.)

What’s in XMI1.1?

- 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.)

Does allow for extensions, incomplete models etc.

And that means for UML...?

Because you can look up or down the metamodel architecture,

UML can be regarded as:

- an XML document that conforms to a DTD describing MOF;
- an XML DTD to which UML models must conform.

The latter is the more important for most people.

Concretely, what is XMI for UML?

1. A way of turning UML models into XML documents
2. and a DTD that those documents should conform to.

UML1.4 (released Feb 2001) includes an official UML DTD.

UML1.3 did not, and that’s a problem, though not necessarily a show-stopper.

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 (yet) specify how to record graphical information; tools do this differently.

Using XMI with UML

Using XMI for UML

1. Most obvious use: interoperability of UML tools.
2. Currently active: “heavyweight” interoperability of tools for different purposes, e.g., UML tools with requirements management tools
3. Less noticed so far: “lightweight” tool interoperability, mini-tools, direct model analysis and manipulation.

The final point is what I want to focus on today.

It provides modellers with powers that programmers take for granted.

Examples

We’ll consider three example tasks:

- synchronisation of model with documentation (in brief)
- analysis of a model (in detail)
- integration of a UML tool with a third party tool (in brief)

NB we are not considering any tasks that involve altering a model today – this is harder than it might be because XMI doesn’t include diagram layout info.

Example 1: 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.

A full tutorial description of this is at `http://www.objectsbydesign.com`.

Limitations of the XSL approach

Good where the structure of the target document closely matches (part of) the structure of the source document.

But

- never intended to be a general purpose scripting language
- scripts with much control flow can be clumsy, hard to read
- performance problems, at least with naive approaches

Beware tasks that turn out to be more complex than you thought, where XSL turns out not to be the right tool part way through!

Alternatives

1. Your favourite general purpose language, with an XML parser written for that language (almost all have them).

   It helps if the language has good file and text processing facilities.

   My favourite is Perl so that's what remaining examples will use.

2. A special purpose XML programming language, e.g. XDuce.

   Research prototypes at present, but worth watching.

Example 2: simple model analysis

Identifying public attributes of classes in a UML model.

Perl, with a plain XML parser as the only addition.

30-odd lines; totally straightforward (if you’re familiar with references and data structures in Perl!)

[There are many ways of doing this task of course; open question is what support beyond the XML parser is most useful.]

The general technique

The XMI parser returns a tree; specifically, a pair

$$(tag, content)$$

for the top level of the document, where $content$ is in detail

$$(attributes, (tag, content)*)$$

Use two mutually recursive functions to walk down the tree, picking out the information we need.

(This applies to any problem, not just this one.)

Chunk 1: setup

```
#!/usr/local/bin/perl
use XML::Parser;
my $file = shift;
die "Can’t find file \"$file\""
               unless -f $file;
my $parser = new XML::Parser
                  (Style => Tree,
                   ErrorContext => 2);
my $pairref = $parser->parsefile($file);
```

Chunk 2: abbreviations and kickoff

```
$VISIBILITY =
’Foundation\.Core\.ModelElement\.visibility’;
$CLASS = ’Foundation\.Core\.Class$’;
$NAME =
’Foundation\.Core\.ModelElement\.name$’;
$ATTRIBUTE = ’Foundation\.Core\.Attribute’;

mypair(undef, undef, @$pairref);
```

Chunk 3: start of `sub mypair {`

```
my ($recclass, $recattr, $tag, $content)
                                   = @_;
return $content unless $tag;
if ($tag=˜/$ATTRIBUTE/) {
  my $attr = {};
  myarray($recclass, $attr, @$content);
  print "$$attr{name}... $$recclass{name}"
    if $$attr{visibility} =˜ ’public’;
  }
```

Chunk 4: middle of `sub mypair {`

```
elsif ($tag=˜/$VISIBILITY/ && $recattr)
  {$$recattr{visibility} =
         ${$$content[0]}{’xmi.value’};}
elsif ($tag=˜/$NAME/ && $recattr)
  {$$recattr{name} = @$content[2];}
```

Chunk 5: end of `sub mypair {`

```
elsif ($tag=~/$CLASS/)
  {myarray({}, $recattr, @$content);}
elsif ($tag=~/$NAME/) {
  $$recclass{name} = @$content[2]
      unless $$recclass{name};
}
else {
  myarray($recclass, $recattr, @$content);
}
```

Chunk 6: roll down the tree

```
sub myarray {
  my ($class, $attr, $attributes, @rest)
                                  = @_;
  while (@rest) {
    mypair ($class, $attr,
            shift @rest, shift @rest);
  }
}
```

Review of technique

We used an XML parser to analyse the XMI file recording the model.

Then we wrote a simple script to walk down the tree gathering the information we needed.

In this case, all we wanted to do was print it out.

(Yes, there are much easier ways of doing that *particular* task! I was just trying to illustrate the generate technique.)

Example 3: tool integration

e.g. the Edinburgh Concurrency Workbench (CWB), for exploring behaviour as expressed in state machines.

General idea:

- extract information from the XMI file,
- process it appropriately
- write an input file for the CWB

(Beyond the scope of this introduction: also, use the CWB to add information into the XMI file: two-way integration.)

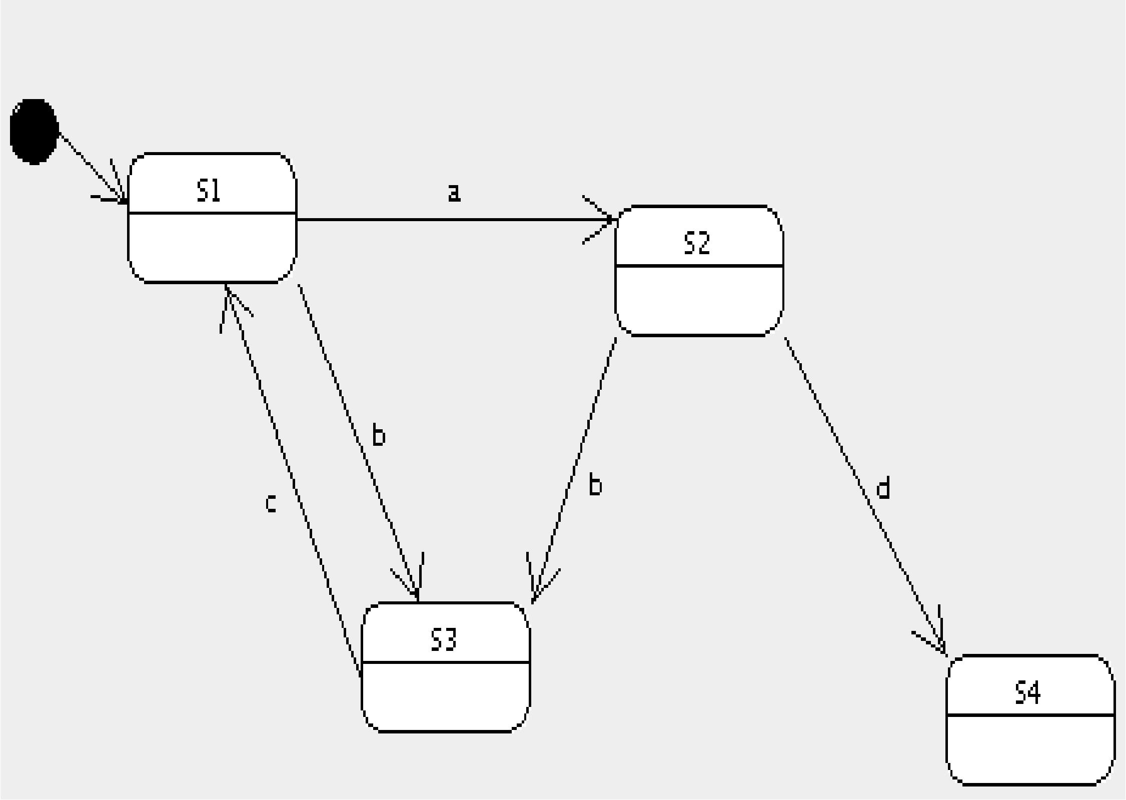
General technique

Very similar to what we did before, but can’t deal with the information we find quite as locally.

We build up data structures to record the information as we find it, then use these at the end to write the CWB input file.

136 lines; details at the XMI Hackers’ Homepage.

E.g., script translates this...



...to this

```
agent S3 = c.S1;
agent S4 = 0;
agent S1 = a.S2 + b.S3;
agent S2 = b.S3 + d.S4;
```

i.e. an input file for the Edinburgh Concurrency Workbench.

Details of this particular translation don’t matter: the point is that I could *easily* extract from the model what was relevant to my needs and reformat it appropriately.

And the effect will be...?

XMI in the development process

I’ve spent most of the session talking about the benefits that may accrue from XMI: developers have more power than they have had to integrate the use of a UML tool into the development process.

Power is always dangerous!

Let’s consider the risks too.

And the effect will be...?

Risks

Rapidly developed scripts may

- contain bugs
- rely heavily on assumptions about the environment in which they're used

For example, it's natural to develop a script by inspecting an XMI file exported by your tool. This is programmer-time efficient, but may mean the resulting script works only for your current tool.

Is this a problem?

And the effect will be...?

Contexts of XMI use

Must notice I’ve been talking about two kinds of development here:

1. development that, while cheap, would go through the usual software process: e.g. model-document linking, tool integration
2. guerrilla programming, the kind of thing that a developer might do alone to make his/her life easier

Both have their place – the main danger is uncontrolled migration of a script from being type 2 to being type 1!

And the effect will be...?

Dependability

Risk may be acceptable if the alternative is:

- doing the same task by hand: boring repetitive tasks are also error-prone
- doing something that takes much longer
- failing to integrate valuable tools

This is the same kind of trade-off you already do in other circumstances.

And the effect will be...?

XMI is not just for easy things

For example, if you use an unpopular programming language your UML tool does not have code generation built in.

But once the information of your UML model is recorded in an XMI file, anyone can develop a code generator for your language – not just your UML tool vendor.

Definitely a type 1 tool though! And for most organisations, uneconomic to develop until XMI adoption is more standard and reliable than it is today.

And the effect will be...?

Open tools

I’ve put the emphasis on what a developer can do quickly, alone.

But remember you don’t have to do it all yourself: XMI is an open standard.

Perhaps the future lies in individuals sharing mini-tools that help them.

This will probably work best if the tools shared each do one thing, well.

I am beginning to collect such resources at `http://www.dcs.ed.ac.uk/home/pxs/XMI/`

And the effect will be...?

Conclusion

XMI can help to put you in charge of your UML tool, instead of the other way round.

More work needs to be done on how best to support the use of XMI, to maximise the benefits and minimise the risks.

Some good things can already be done. Others are in the future.

More information: books

- David Carlson, Modeling XML Applications with UML
- Steve Brodsky’s forthcoming book on XMI in Java
- mine on using XMI to get more value from UML tools

If you might be interested in commenting on drafts of my book, please send me email: `perdita@stevens-bradfield.com`

More information: web

Links to all the other XMI sites I know of are on my new webpage

The XMI Hackers’ Homepage

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

More things are gradually appearing there too... e.g., the slides I’ve used today will be there from next Monday.

The End

Thank you for being here.

Any more questions?