

Applying aspect-extended UML Modelling to *e*

Darren Galpin



Never stop thinking

The Problem

- We have testbenches which have been developed over many years, involving many people, much knowledge, and much re-use.
- Re-use typically involves adding new functionalities and modifying old ones -> many extensions and overrides.
- Testbenches are often moved between sites and teams – how to bring up new teams quickly, especially as documentation on testbench is often poor.
- Knowledge is often lost along the way. Contractors are used, people leave, knowledge isn't documented before it is lost. Very different programming styles can be used.
- Testbenches are often very complex..... Might have redundant code as the RTL changes.
 - Need some way of making the knowledge capture and bring up easier...

The Solution – UML (?)

“The Unified Modeling Language (UML) is a graphical language for visualizing, specifying and constructing the artifacts of a software-intensive system. The Unified Modeling Language offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components. ”

Wikipedia

- Provides a graphical way of viewing a system, so should make it easier to see the program flow and interaction.
- There are commercial tools available, so can possibly just load the code in and view the result (wishful thinking....)
- Could design the code in tool, annotate, and use UML database to feed back the effects of changes. UML can write skeleton code for the first design, which can later be filled in.

Issues

- UML was written to handle object oriented systems, but *e* is aspect oriented. How do we handle this?

- Use Theme/UML.
 - Developed by Trinity College Dublin to handle AspectJ.
 - AspectJ supports method and class extensions.
 - However, not yet supported by a commercial tool.

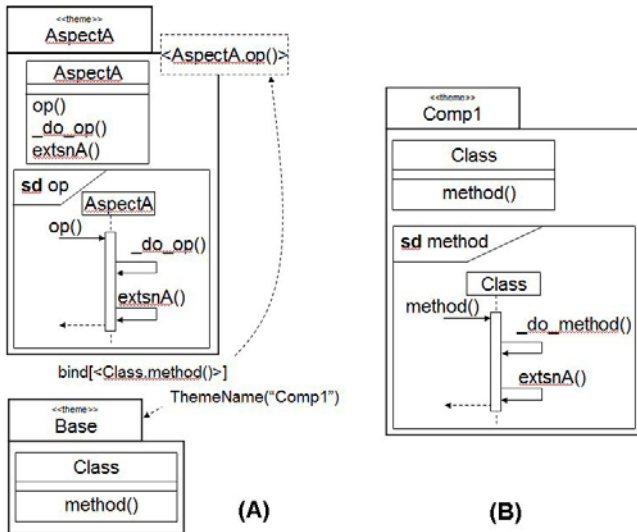
- Can we use Theme/UML to model *e*?

Problem 1 – Units and Structs

- AspectJ, like Java, is based on classes. *e* is based on units and structs.
- Units and structs are very similar, contain similar constructs and have similar built in methods (pre-generate(), run(), etc...)
- Handle this by having a top level class which has two child classes – a unit and a struct. Common methods are placed into the base class, any specific methods are added by extension into the child unit and struct “classes”.

Problem 2 – extending methods

- In Theme/UML, a thread of functionality is known as a theme. An aspect theme adds crosscutting behaviour which modifies the base execution (in otherwords, “is also”, “is first” or “is only” extension).



- Wish to bind new aspect theme to the base theme, extending method() with extsnA() via “is also” extension.
- The resulting composed object is shown on the right.
- Have to manually draw these at the moment – laborious for complex testbenches needing to capture multiple extensions.

How to handle type extension?

- Can simply extend types in e , but cannot in Aspect J.

- Example:

```
type dog_type : [pug,poodle];  
extend type dog_type : [bulldog];
```

- Can create a new base dog class, with child classes for each dog_type.
 - In e , the child class does not exist until generated.
 - Constraints can be added so that a sub-type might never be generated.
 - Difficult to graphically show this in UML.
- Root problem is that AspectJ just isn't as extensible as e

Handling constraints

- Variables in e have constraints to restrict what is generated. For example:

```
type bark_type : [yelp, growl, howl];
type dog_kind : [poodle, pug, bulldog];
struct dog {
  bark : bark_type;
  dog_kind : dog_kind;
  keep soft bark_type in [yelp,growl];
};
extend poodle dog {
  keep bark_type=yelp;
};
```

- Constraints can be added to UML via Object Constraint Language. Allows ranges and implications to be added to variables.
- OCL cannot have aspects added to it – Theme/UML has not considered this. Hence the extension cannot be modelled. What about soft and hard constraints?

Timed Behaviour

- TCM's, TE's and Coverage objects – model them all as methods.
 - All instantiated within a unit/struct.
 - Event is a method that raises a flag when emitted or when a sequence of other events is observed.
 - A TE is a piece of boolean logic that evaluates to true or false and raises a flag.
 - Cover objects are methods that increment counters for coverage purposes.

- But how to distinguish between `method()` and `method()@clk`?

- In addition, different extension rules apply for the different “methods”. E.g. “is only” not allowed for cover objects.

- Other timed behaviour: run-time generation, destruction of structs, Load order.

So, in summary

■ UML doesn't work.....

- Full Aspect-orientation does not map well, so cannot support full feature set of e .
- Aspect-orientation is not commercially supported, so need to manually draw AOP diagrams. Too much work....

■ So don't bother with it?

- Diagramming small parts of the system can be useful. UML does support polymorphism, so can use the diagrams to describe sub-sections of the code.
- A good diagram is worth a thousand words.
- Diagrams are language neutral.....

For further info see "Modelling Hardware Verification Concerns Specified in the e-Language: An Experience Report", by Darren Galpin, Cormac Driver and Siobhán Clarke, submitted to AOSD 2009.