Submission for XP2005 Workshop:
"Agile Development with Domain Specific Languages"

Author Matthew Fowler
Director
New Technology/*enterprise* Ltd, London

My interest in the above area stems from developing the JeeWiz product, which has a model-driven system generator and high-value transformations. We can automate any system development activity that has an observable pattern or technique. We use as input any model format - UML, XML, XML-Schema/WSDL - so DSLs will be a natural extension to this.

We achieve a level of system generation that enables a highly agile development process, effectively reversing the dependency chain of non-agile software projects.

We have created about 10 modelling "languages", using our own meta-modelling facilities. I am interested in understanding existing DSL formats and underlying concepts so that we can
- accept DSM's as input for the generator
- create DSLs from our existing meta-models.

Contribution:

To achieve agile development using modelling, we have found it necessary to compose DSLs that are rich enough to describe a whole (multi-tier) application, and this is most effectively done through composition of smaller modelling "languages" or fragments of same.

A number of techniques have worked well for us, and were implemented in JeeWiz V2 (two years ago). These are sketched below and I can give more details at the workshop if required:

1. DSL fragments

   Often, meta-classes have similarities - this could be in their properties, children, associations or behaviour. To support re-use of these similarities, we have a "factor" feature. This allows these similarities to be factored out into distinct files and then included in the appropriate meta-classes. For example, a "security roles factor" allows a meta-class to have lists for roles that are allowed, or not allowed, to access the resource.

   Fragments are specifically not meta-class inheritance - this is a mix-in approach, but only requiring the 'innards' of a meta-class to be specified rather than a whole meta-class.

2. One man's domain is another man's platform

   In the same way that inheritance allows us to construct richer behaviour in objects incrementally, so meta-classes can inherit from others to construct richer modelling concepts. For example, an 'Entity' meta-class can inherit from a 'BusinessObject' meta-class which in turn inherits from the 'Class' meta-class.

   We group meta-classes into meta-models (e.g. the 'Class' meta-class goes into one meta-model, 'Entity' into another, screen 'Page' into yet another). This helps us chunk the functionality relating to one domain (or platform!) - in our case, Java classes for the meta-classes go into one

Java jar. It also means we can then easily load many meta-models as the modelling environment requires - we build complex modelling/generating environments by specifying an ordered set of meta-models.

We explicitly allow meta-class inheritance to span meta-models so the 'domain' language provided by a meta-model can be used as a 'platform' by higher level (e.g. 'Entity' relies on the concept of 'Class', which is in another meta-model).

We explicitly allow use of the same named concepts (e.g. 'Entity') in related meta-models: the meta-classes can inherit. This means that the same modelling concept takes on different connotations depending on the set of meta-models we are currently using.

3. Constructing DSLs

   The previous point describes the underpinnings of a modelling environment, but it is too complex for modelers to use.

   Therefore, to build a DSL based on meta-models, we collapse the meta-models and create a single "aggregate" meta-model, and create the DSL from that. This means that with concepts like 'entity', that appear in multiple meta-models, the user sees only one modelling concept with the aggregate of all inherited and mix-in properties.

4. Localising DSLs

   Different modelling environments and customers have different needs. To accommodate these, the 'collapsing' process described in the previous section also uses a "filter" model, which is used to adapt the complete modelling language/DSL to the local environment. This can remove unwanted modelling concepts - such as properties, relations and dependencies - or add additional ones as appropriate.

   This approach allows us to support the meta-models to be used in different modelling environments and customer architects to adapt them to the local skills and modelling standards.

Question Is DSM about modelling types or instances, or both?

Web References http://www.jeewiz.com/product.html