Meta-Object Programming for MDA

Dr A Clark

Dr A Evans
Model Driven Architecture

- MDA unifies multiple perspectives of the modeling space
MDA Tool Requirements

- Modelling languages becoming increasingly complex, diverse and semantically rich:
  - Software and non-software modelling.
  - Support for: model analysis, model checking, model execution, model transformation, ...
  - Support for language related facilities: parsing, model interchange, model-management.
- MDA tools must provide full support for the modelling of semantically rich languages.
- Can only be achieved by full access to a sufficiently expressive meta-modelling language.
Traditional Meta-tools

Static meta-models → Compiler → Repository → XMI, etc

Could be XMI

Meta-modelling language

Tools
Limitations

- Use of static meta-modelling language means that semantics of languages must be described at an implementation specific level, e.g. through repository API
  - Semantics cannot be viewed as being a part of the language definition.
- Meta-modelling language is not capable of expressing its own semantics:
  - Again, relies on platform specific implementation of semantics, e.g. `new()`
- Impact is to significantly reduce flexibility, interoperability and time to deployment of tools.
• OMG standard meta-modelling language.
  ◦ Cornerstone of MDA standard.

• MOF 1.0/2.0 being extended to support richer modelling abstractions:
  ◦ QVT (Query, Views, Transformations) RFP
  ◦ Versioning and Lifecycle RFP
  ◦ Facilities RFP

• Yet, still lacking full power to capture language semantics.
Solution – meta programming

- Key feature missing from MOF is: executability.
- Extend MOF with minimal set of executable primitives: create/delete object, slot update, sequential operator.
- Result:
  - MOF can express semantics of languages in a platform independent way.
  - MOF can express own semantics.
  - MOF becomes stand alone, highly flexible, executable language (programming language) for MDA tool development.
• Extends OCL with primitive action expressions:

```ocm
context TransformationInstance::addMap(left, right)
let x := Map.new(left, right) in
    self.maps := self.maps -> including(x)
end
```

```ocm
context Bank::addAccount(details)
let a = Account.new(details) in
    self.accounts := self.account -> including(a);
    self.transactions := self.transactions + 1
end
```
Example: meta-language definition and extension

- **Mappings:**
  - An extension to the MOF meta-model

  ![Diagram]

- Has rich executable semantics: e.g. pattern matching mechanism has been widely proposed.
Pattern Matching

- Implement pattern matching mechanisms via mapping to action primitives

Key benefits:
- Platform independent description of semantics
- Executable in an appropriate tool
- Flexible: semantics can be readily changed and built up in layers to define multiple languages
Executable Meta-tool Architecture

Core executable primitives

- Kernel
- EMOF
- XOCL
- Mappings
- CMOF
- UML
- CWM
- Java
- ...
PlugIn Architecture
Extensibility

Kernel

Load

Plug-in

QVT tool

Model of mapping language
**XMOF: an MDA Hub**

- Enables rapid development of tools that can support the full spectrum of languages required for MDA.
- Tools are *modelled* – giving maximum flexibility and interoperability:
  - Facilitates plug and play tool architecture
  - Tools can be exchanged as models (XMI documents) – ultimately breaks vendor dependence (tools become models, i.e. assets).
Summary

- MOF is currently not expressive enough to support modelling of semantically rich languages required for MDA
- Simple extension with execution is what is required
- MOF becomes a meta-programming environment (moves away from static, repository based approaches)
- Language definition is completely platform independent – a good basis for OMG specifications (imagine if you could download a spec and run it!)