A Textual Notation Generator for MOF Models

Jim Steel, DSTC
steel@dstc.edu.au
Introduction

• Need for textual languages for models
• Existing metamodelling standards and techniques
• A more generic approach
• Language customisations
• Prototype and applications
• Summary
Meta-modeling

• Metadata can now be modeled using the Meta-Object Facility (MOF)
• The metadata can be exchanged using the XML-based Model Interchange (XMI) – currently XMI 1.1
• For human construction and consumption, current solutions involve handwritten languages or tools
The need

- XMI is often too verbose for human construction/consumption without tools
- Textual offer advantages over graphical tools (search/replace, complexity)
- Hand-written tools or languages take time to develop, and require maintenance if the metamodel changes
XMI: Example

<XMI.content>
<Notification>
  <Notification.EventTypeRepository xmi.id='n1'>
    <Notification.EventTypeRepository.contents>
      <Notification.EventType xmi.id='n2' domain='room' name='entry' private=true />
      <Notification.EventType xmi.id='n3' domain='room' name='announce' private=false />
    </Notification.EventTypeRepository.contents>
  </Notification.EventTypeRepository>
</Notification>
</XMI.content>
An automated approach

• Based on a metamodel, generate a language automatically
  – Common representations for packages, classes, associations, references, etc
  – Producer for inspection of a MOF-compliant repository’s metadata
  – Consumer for altering metadata and/or populating such a repository
Customisations

- An automatically generated language has no knowledge of the domain
- Providing a limited number of language customisations can alleviate this
  - Identifying attributes
  - Attribute Representations
  - Association Representations
Example

Notification {
    EventTypeRepository "x01" {
        private EventType "entry"
        { domain: "room" }
        EventType "announce" { domain: "room" }
    }
}

Prototype

• Developed against DSTC’s dMOF product
• Generates an XSL stylesheet for conversion from XMI for production
• Generates a JavaCC grammar and backend for consumption
• Customisations specified in a generated language and stored in a repository
• Currently in use within DSTC
Applications

• Simple process for developing a language
  – obtain a metamodel
  – define language customisations
  – automatically generate parser/stylesheet

• Usefulness
  – Gives a rapidly deployable language for evolving/changing metamodels
  – Lightweight alternative to integrated tools