UML and Meta Modelling

- Topics:
  - UML as an example visual notation
  - The UML meta model and the concept of meta modelling
  - Model Driven Architecture and model engineering
    - The AndroMDA open source project
  - Applying cognitive dimensions to assist in designing a UML tool
    - How to mitigate some of the problems inherent in UML

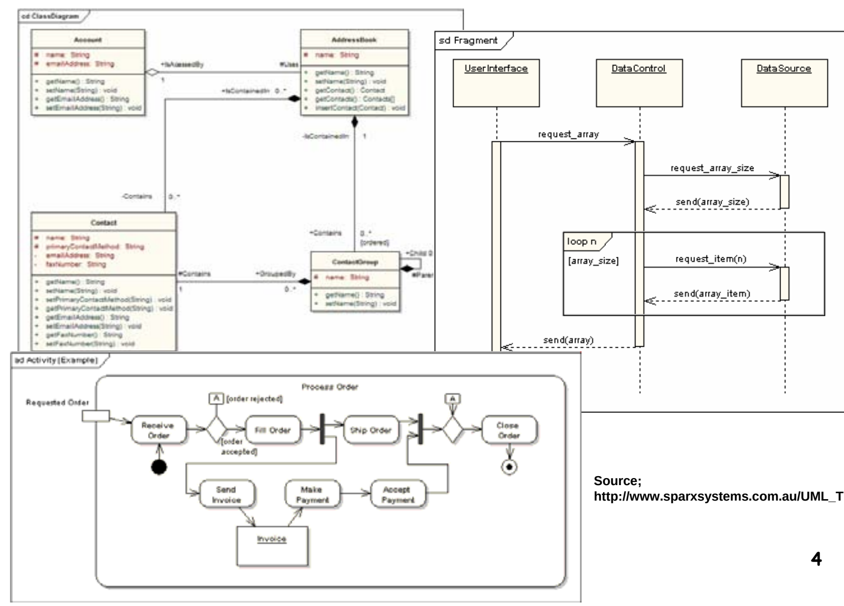
The Unified Modelling Language

- Notation(s) for describing object oriented models
  - can be used for describing implementations, designs, and analyses
  - incorporates and extends elements from several earlier modelling notations
  - early development primarily by Rational Software Inc (now owned by IBM), but now developed by OMG (UML 2.0 in process of release)
- Has a variety of diagram types expressing both static and dynamic aspects
  - class diagrams
  - package diagrams
  - use cases
  - sequence and collaboration (now called communication) diagrams
  - state & activity diagrams
  - etc (12 diagram types in all)
- Plus Object Constraint Language (OCL) for expressing more complex constraints
- Sources:
  - UML Distilled, Martin Fowler, Addison Wesley
  - UML specifications from http://www.uml.org/

Notation vs. Methodology

- UML is a set of notations
  - Used to model OO systems
  - Define a set of overlapping models using the various diagrams each expressing a different view or viewpoint on the system modelled
  - Described by a meta-model ie a model to describe a model
- But also need to know how to go about constructing a model
  - i.e. a methodology for using the notation
  - Eg RUP – Rational Unified Process
- Will primarily look at UML notation, rather than modelling methodologies, but will touch on Model Driven Architecture approach

Example Diagrams



Source; http://www.sparxsystems.com.au/UML_Tutorial.htm

Diagram Perspectives

- Diagrams are used for multiple purposes with different semantics
- When interpreting them you need to know the perspective being used
- Eg Class diagrams
- Conceptual
  - diagram represents concepts in domain
  - may or may not relate to implementation classes
  - typically used in analysis
- Specification
  - software interfaces, i.e. types rather than classes
  - typically used in design and documentation
- Implementation
  - laying bare implementation details
  - only occasionally used for detailed understanding

Constraints

- Much of UML is about specifying constraints: eg relationship between things, multiplicity of associations, exclusivity of subclasses
- A variety of keyword based constraints are included in UML
  - subtypes: {complete} {incomplete} {disjoint} {overlapping}
  - association ends or attributes:
    - {ordered} {unordered} {sorted}
    - {changeable} {addOnly} {frozen}
  - timing of messages (standard functions)
    - startTime stopTime executionTime
- Additional textual constraints can be specified informally using notes
- But more formal constraints can be specified using the Object Constraint Language (OCL)

OCL

- A formal language
- Pure expn language - uses a declarative style
  - specifies constraint, not what to do if violated
  - side effect free
  - strongly typed
- Used to specify, eg,
  - pre and post conditions on operations and invariants, eg:

    context Company inv enoughEmployees : self.numberOfEmployees > 50
    context Company::setCreditLimit(limit: int)
    pre: limit >= 0
    post: creditLimit >= 0
  - constraints on navigation of associations
- Also used to specify UML meta-model semantics (see later)

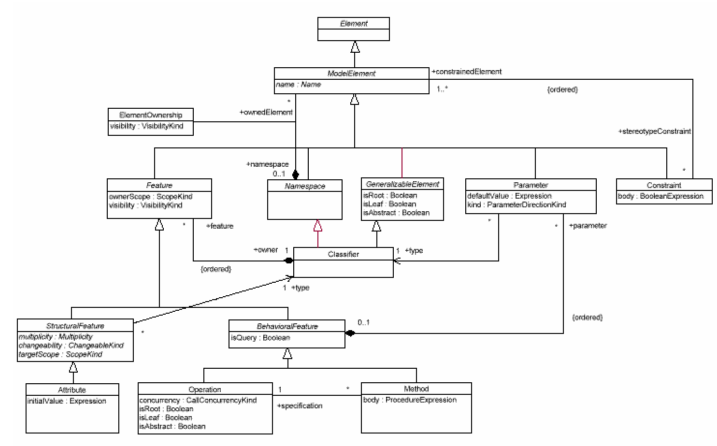
UML meta-model

- Need a formal specification of UML's syntax and semantics to allow:
  - uniform understanding of what models mean
  - tool makers to design UML tools that implement semantics consistent with those of other tools
  - interchange of models between tools (by specifying interchange formats)
- Such a formal specification is a *meta-model* as it describes the form that its instances (individual UML models) can take
- But how do we specify the meta-model?
  - Answer (simple): Use UML to define itself
  - Answer (complex): Define the UML meta-model using a meta-modelling language.

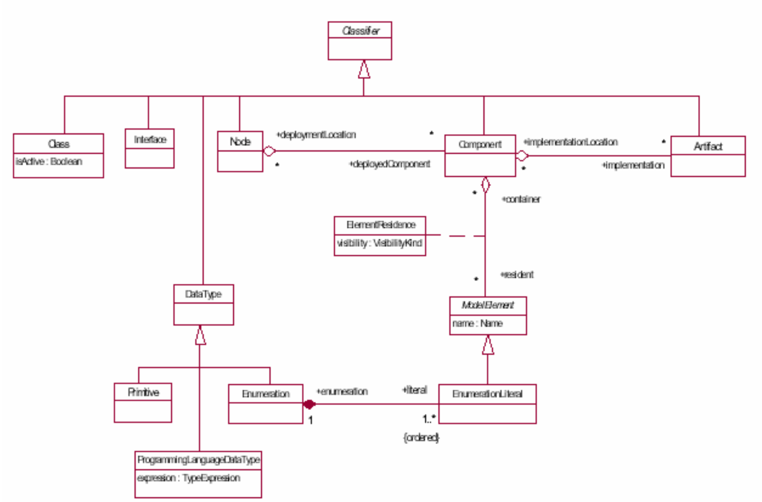
UML specification

- The formal UML specification is at http://www.uml.org/
- This does not specify the exact surface syntax for UML (ie exact icons etc), rather it specifies UML in an abstract syntax-like form
- The specification makes extensive use of UML diagrams (particularly class diagrams) supplemented by OCL for more detailed semantics.
- The definition is in terms of *packages* defining common and more specialised diagram components/concepts (the following is UML 1.5 – these have changed in UML2.0)
  - eg Core Backbone package defines fundamental concepts
  - eg Core Classifiers package defines entity-like things (eg classes, interfaces)
  - eg State Machines package defines extensions to cover state diagrams

Core Backbone



Core Classifiers

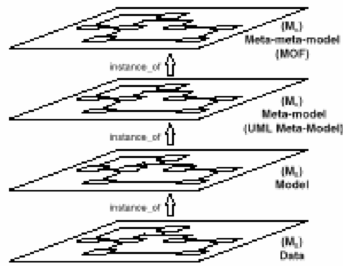


Meta-meta-modelling

- Although it appears as if UML defines itself, this is not actually the case.
- The specification actually uses a meta-modelling language
  - this is itself object oriented and has many concepts in common with UML
  - called Meta Object Facility (MOF)
  - common with OMG CORBA IDL specification work
  - also used for the Common Warehouse Metamodel (CWM)
- But how is this meta-modelling language specified?
  - Answer: using itself (defining a meta-meta-model)

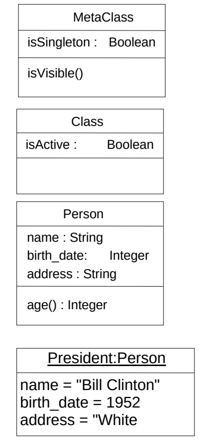
4 Layer Model

- This approach leads to a four layer approach to the modelling
- meta-meta-model (M3): defines the MOF notation
- meta-model (M2): defines UML notation using MOF
- user model (M1): a UML model of a particular problem domain
- data (M0): typical objects instantiating the UML model
- Note: could use M3 instead to define M2 for ER modelling; M1 a typical ER model; M0, typical ER data.



4 layer model

- Typical examples of elements at each level:
- M3: MOF MetaClass
- M2: UML Class, instance of MOF Class; very similar to MOF concept of a Class
- M1: Person, a typical instance of UML Class
- M0: President:Person, a typical instance of Class Person
- From C. Atkinson, Supporting and applying the UML conceptual framework.



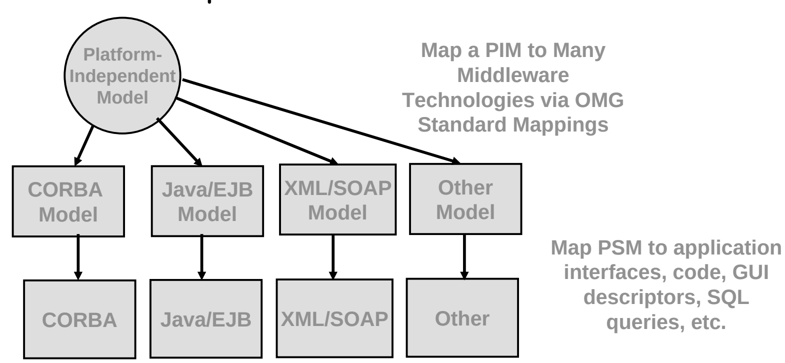
Advantages of meta modelling

- *Consistency of interpretation using more formal semantics*
  - *Although MOF approach not nearly as unambiguous as other specification approaches*
- *Possibility of interchange standards based on meta model specification*
  - *Can interchange models between tools*
  - *XMI is the defined interchange standard based on MOF*
    - *Essentially MOF in XML (makes for verbose interchange files)*
- *Can use meta models as schema for semantic data to be stored in a repository*
- *Can define extensions that reuse parts of the existing model*
  - *Eg did this with our DPML work (see later)*
- *Can use meta models to specify tools*
  - *If have appropriate tool building tools can generate the tool from the meta model (this is what we do with our JComposer and Pounamu tools) or a system from a model (MDA approach)*

See www.metamodel.com

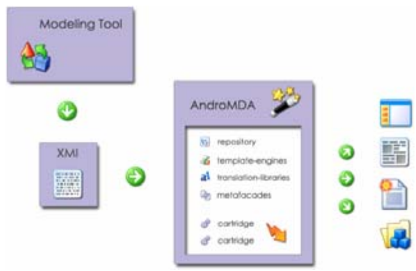
Model Driven Architecture (MDA)

- *Generate systems from models (see http://www.omg.org/mda/)*
  - *Start with Platform Independent (UML) Model (PIM)*
  - *Generate a Platform Specific (UML) Model from PIM*
  - *Generate implementation from PSM*



Example MDA system

- AndroMDA http://www.andromda.org/
  - open source code generation framework
  - follows the Model Driven Architecture (MDA) paradigm.
  - takes PIm model(s) from CASE-tool(s) and generates fully deployable applications and other components.
  - Currently limited to J2EE PSMs
  - Uses concept of a "cartridge" which defines the PIM->PSM translation for a given PSM



MDA – Critique

- Example of "model engineering": treats software development as a set of transformations between successive models
- MDA specializes model engineering by using MOF and associated UML models. Relies on UML Profiles which are specified using MOF
- PSMs are likely to be very difficult to construct – hard enough to program in J2EE or .NET by hand
- Problem of debugging generated code
- Domain oriented programming where you generate systems from domain specific languages is more likely to provide real advantage
  - See Pounamu and other meta tools shortly
- From D.Thomas, MDA: Revenge of the Modelers or UML Utopia, IEEE Software May-June 2004

Towards UML Evaluation

- How would we go about evaluating UML?
  - As a notation or set of notations?
  - As an adjunct to a methodology such as RUP?
- Could conduct experiments with user populations
  - Eg survey based approach
  - Need careful experimental design with hypotheses to test
    - Eg people do not use notational element X because of Y
- Could use cognitive dimensions to evaluate notation
  - But needs to be done in the context of a particular environment (ie a UML tool such as Rational Rose)
  - Also difficulties as really a set of notations
  - Could turn problem around and look at requirements for a UML tool based on Cognitive Dimension framework (6.1 of CD paper)

Requirements for a UML tool

- Abstraction gradient
  - Will always be high for UML as it is a very rich collection of notations
  - Could minimise by offering subset of notation to novice users
- Hidden Dependencies & Visibility
  - Multiple diagrams with multiple notations
  - Strong need for consistency between diagrams, but this leads to many hidden dependencies
  - Could offset by navigation tools to move rapidly between elements that are being kept consistent (partial remedy – see CD paper)
- Viscosity
  - Key issue here is insertion and deletion of new elements and how this affects consistency management
  - Also automatic layout considerations, direct versus dialog box editing etc
    - Many of these issues are UI related rather than notational

Requirements for a UML tool

- Closeness to mapping
  - Appears to be good for class and interaction diagrams and poss package diagrams
  - Other types of diagram are typically less used by programmers. Poss this is due to difficulty in mapping to eventual implementation in programmer's mind
  - Depends critically on designer's background
  - Support for refinement from conceptual->implmn
- Progressive evaluation
  - UML is not "executed" in the same way as other VLs
  - Issues here with code generation (of stub classes)
    - Regeneration after user additions to stub classes
  - "Simulation" of sequence diagrams?
  - Support for refinement from conceptual->implmn

Requirements for a UML tool

- Premature Commitment
  - Many issues here
  - Eg need for a class before adding a method or association (dangling association)
  - Support for refinement from conceptual->specn->implmn
  - Layout – having to decide a generalisation is likely to occur and allow space for it to avoid re-laying diagram out
- Error proneness
  - A likely problem here is the overloaded use of the notations for conceptual, specification, & implementation
    - Could minimise by appropriate diagram annotation to indicate perspective (not done in any of the tools that I am aware of, but could be considered part of MDA initiative)

Requirements for a UML tool

- Consistency
  - Some difficulties due to multiple notations
  - Strong attempt made to reuse elements in multiple diagrams (eg class, object notation in sequence and interaction diagrams)
  - However areas where notations is strongly different (eg operations in class diagrams versus seq diagrams, state diagrams)
  - Crossing to completely dissimilar notations (eg state or activity diagrams) creates a significant consistency hurdle
  - Some difficulties also due to multiple perspectives

Summary

- UML is a big and general purpose set of visual notations
  - Causes difficulties that need mitigation in tool design
- It has wide adoption as the lingua franca for software design
  - Hence reduces closeness of mapping issues – software designers brought up with UML
- Introduced the concept of meta modelling
  - For defining semantics of UML
  - As a more general purpose approach to high level modelling
  - As the basis of tool generators
  - As the basis for model driven design
- Next lecture introduce the Pounamu meta tool