Pounamu

- Aim of section:
  - Examine Pounamu, a meta tool for constructing extensible visual environments
    - Very much alpha software
- Contents
  - Historical development
  - Pounamu usage example
  - Pounamu structure
  - Applications
  - Assignment

History

- Original interest: Visual class diagramming tool ISPEL
- Led to long term interest in frameworks and tools for constructing such systems

Ispel 1991 → MViews 1993 → JViews 1997 → JComposer 1998 → Pounamu 2003

Frameworks for constructing multi-view multi-notation environments

Meta tools for specifying and constructing multi-view multi-notation environments

Note: see later lecture on Evolving Frameworks Pattern Language

Meta modelling

- What's a meta model?
  - A model that defines/describes a model
  - Eg the UML meta model describes abstract concepts such as class type, association type, generalisation type, etc, that have instances in a particular model, (eg customer class, order class, customer-order association, customer-organisation generalisation)
- What's a meta tool?
  - A tool that allows you to define meta models which can be used to *generate* environments for modelling using instances of the meta models

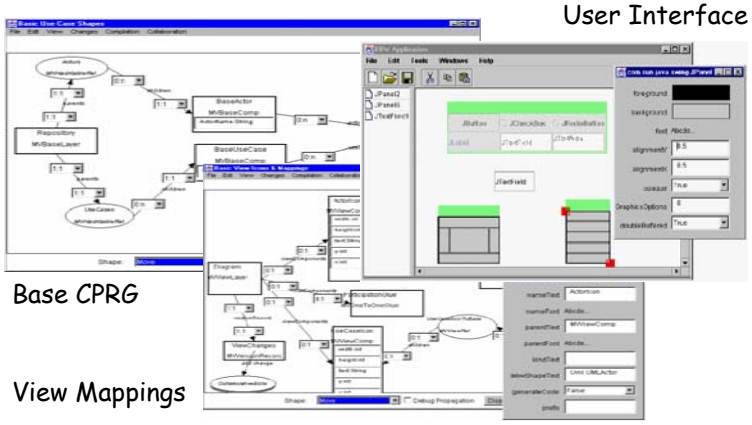
JViews/JComposer

- JViews provided a framework for constructing multi-view environments
  - Change Propagation and Response Graphs
  - 3 layer model: base, view and display layers
  - Much programming required for instantiation: many classes, many components, complex but repetitive programming
- JComposer/BuildByWire: 2 tools to allow much of a JViews environment to be generated
  - BuildByWire: constraint based GUI component specifier
  - JComposer: meta modeller, for specifying JViews base and view layer components and relationship to BBW GUI components

JComposer/BuildByWire

- Used to specify and generate JViews-based environments

User Interface



Base CPRG

View Mappings

+ Backend code generator

Problems

- Heavyweight GUI components
- Complex tools
- Heavyweight framework
- Based on bespoke event model
- Customisation difficult
- Dynamic behaviour difficult to add – much programming still needed
- Strong compile/utilise cycle

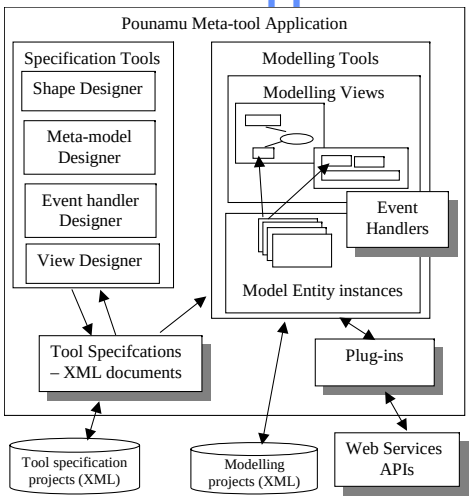
Pounamu

- Pounamu overarching design requirements
  - Simplicity of use.
    - It should be very easy to express the design of a visual notation, and generate an environment to support modelling using the notation.
  - Simplicity of extension and modification.
    - It should be possible to rapidly evolve proof of concept tools by modification of the notation, addition of back end processing, integration with other tools, and behavioural extensions (eg complex constraints).
- Led to a much more lightweight structure, with extensibility, customisation strongly built in, plus web services interface

Pounamu components

- Shape creator and connector creator tools
  - Used to define icons, connectors and associated properties
- Event Handler Designer tool
  - Specifies dynamic behaviour in response to events (eg shape creation). Currently Java code using API
- Meta model designer tool
  - Specifies tool meta models
- View type designer tool
  - Specifies an editor for a set of shapes, connectors and handlers, and their relationship to a meta model
- Model projects
  - Instances of a specified tool in use

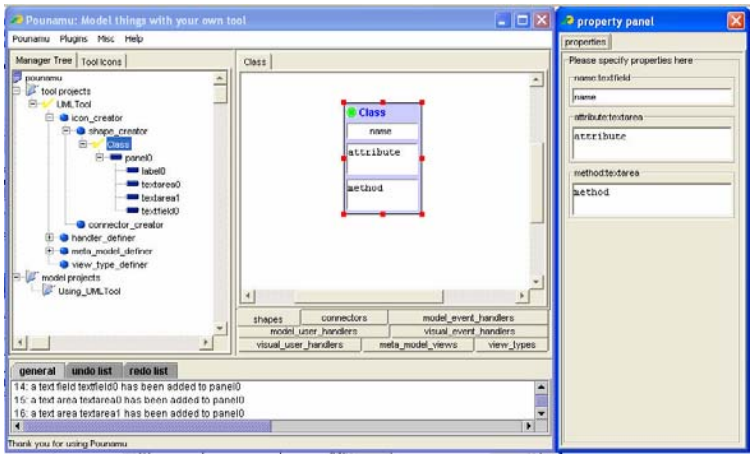
Pounamu approach



Simple example of usage

- Define a very simple UML modelling tool.
- Class diagrams have at least one type of shape and one type of connector
  - Class icon – rectangle with three regions, name, attributes and operations
  - Generalisation –unidirectional arrow
    - Connects class to class
- Start by defining shape for class
  - Rectangular border panel containing
    - textfield for name
    - rectangle plus multi valued textfield for each of attributes and operations
  - Allow the textfields to be seen by the underlying tool model

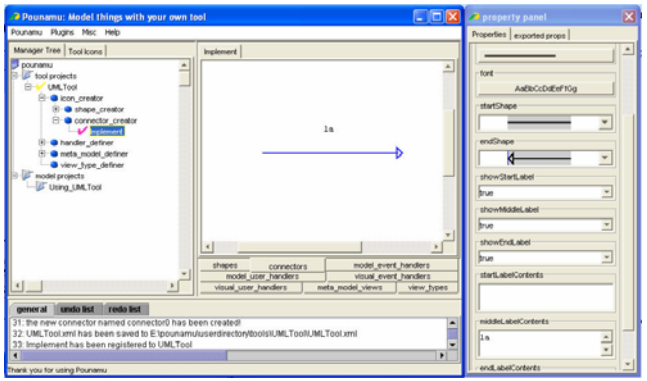
Shape Creator



Repeat for other shapes (eg interface, note)

Generalisation

- Create the generalisation connector using the connector tool
  - Arrow on end of connector

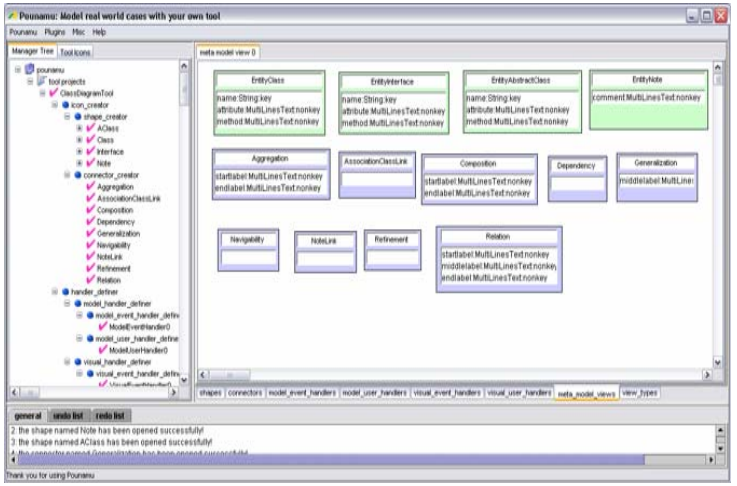


Repeat for other connectors (eg association)

Meta model

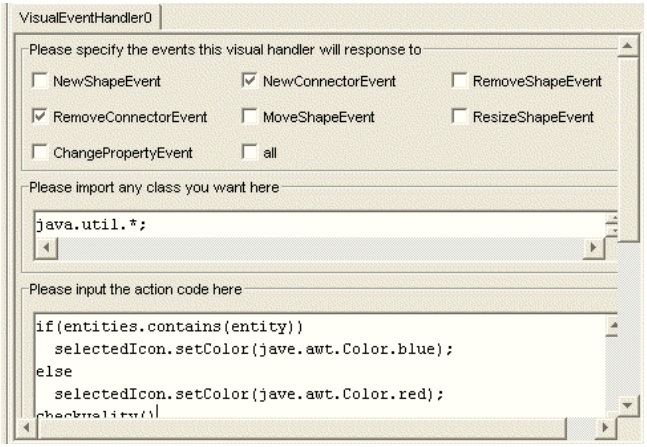
- Need to define the meta model for the underlying tool shared repository
- Pounamu meta modeller uses an entity relationship model
- Implement entities:
  - Class entity, with name (key), attribute and method attributes
  - Interface entity, note entity, etc
- Implement associations:
  - Generalization (for generalisation), with middlelabel attribute
  - Aggregation, Composition, etc

Meta Model Definer



Handler

- Add a very simple handler that colours entities to indicate various state changes
- Uses API to access Pounamu data structures



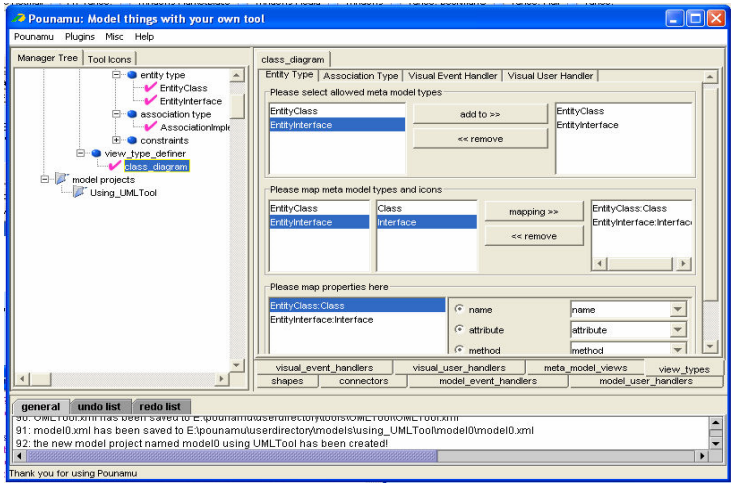
Review

- To date have defined:
  - A shape (class icon) and connector (generalisation) GUI components (shape & connector tools)
  - An underlying repository consisting of types for entities representing classes, linked by implements relationship types (meta model tool)
  - An event handler that responds to the addition or deletion of connectors (event handler tool)
- Missing
  - A way of defining the editor for a class diagram
    - ie allowable icons and connectors and what handlers are relevant
  - A way of connecting things added to an editor window to the underlying repository
- The latter provided by the View Specification tool

View specification

- Now specify a class diagram editor using the view specification tool
  - Specify shapes, connectors, handlers that can be created in the view:
    - Class, generalisation
  - Specify meta model entities and associations associated with the view
    - Class, implements
  - Specify mappings from meta model components to view components
    - Both at a component level and then at an attribute level
    - Simple 1-1 mappings – more complex mappings require handlers

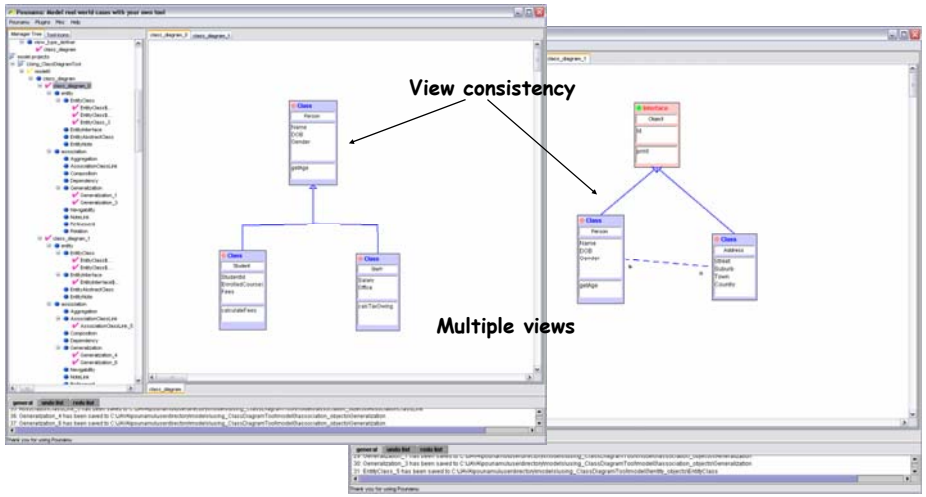
View Type Definer



Models

- Then register everything and save the tool
- Can then create models using the tool
- Create a model project, then views
- See Pounamu tutorial and example class diagram tool

Model projects



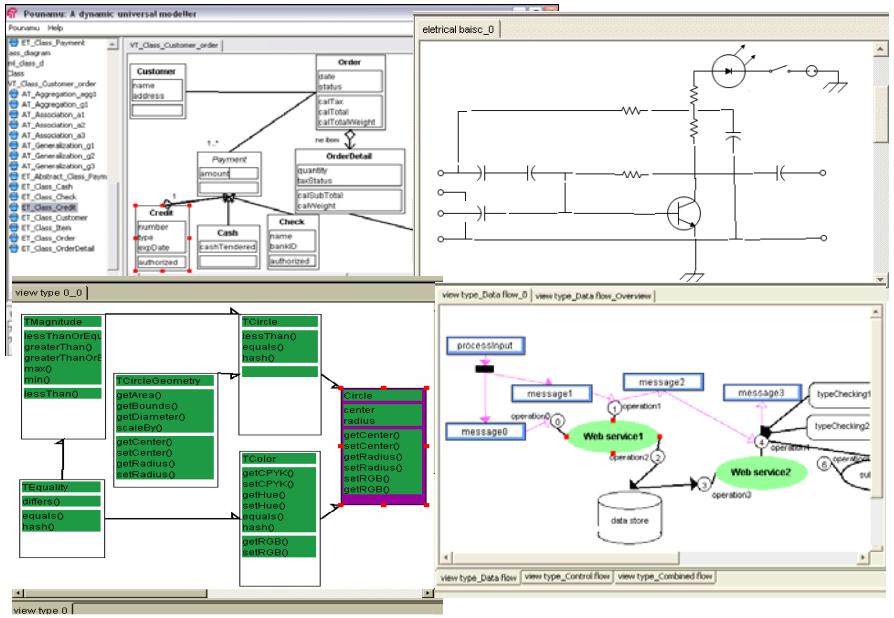
Modification, integration, extension

- Pounamu is live,
  - Changes to a tool specification are immediately reflected in executing models using that tool
- Handlers provide behavioural extension capability
  - Via API, can extend tool behaviour significantly
  - Handlers compiled and installed on the fly
- RMI and Web services interfaces provide external integration capability
  - Can add plug ins
  - Have used for developing generic thin client and mobile phone modeller interfaces, process modelling and enactment tool, collaboration and group awareness tools, integration with project management tool
- Can add backends manipulating the XML save format
  - Eg for code generation and reverse engineering

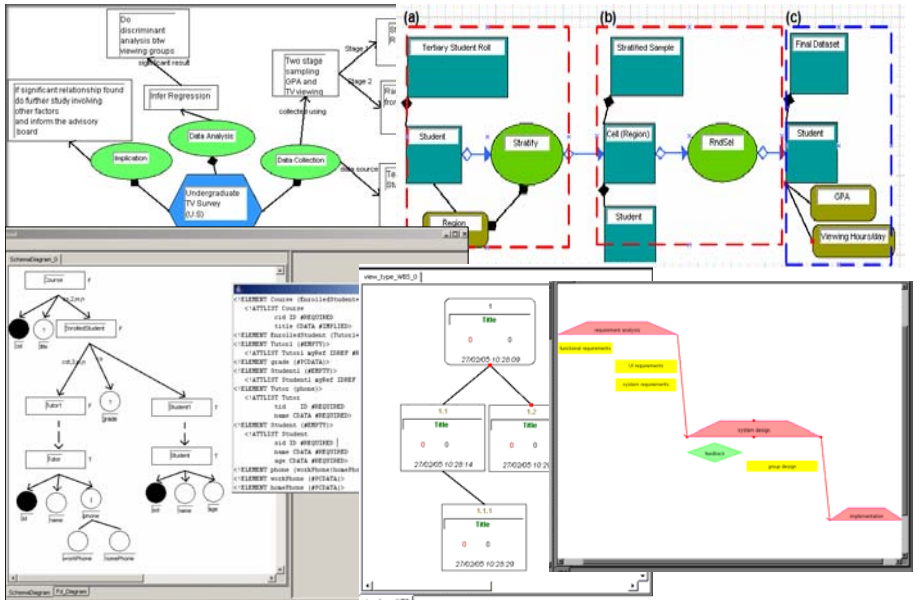
Examples

- Pounamu has been used to develop a wide variety of other tools
  - UML tool (all major views) – Karen Liu project
  - Process modelling tool – Therese Helland MSc
  - Circuit Designer – Nianping Zhu
  - Traits design tool – Blazej Kot project
  - ORA-SS Tool – Nodira Khoussainova project
  - SDL stats survey tool – Chul Hwee Kim project
  - Project scheduling tool – Jun Ho Huh & Nader Hosseini-Sianaki BE(SE) project
- Current projects
  - SDL extensions – Chul Hwee Kim MSc thesis
  - Aspect Oriented Comp Eng tool – Santokh Singh PhD
  - Web services composition tool – Karen Liu PhD

Examples

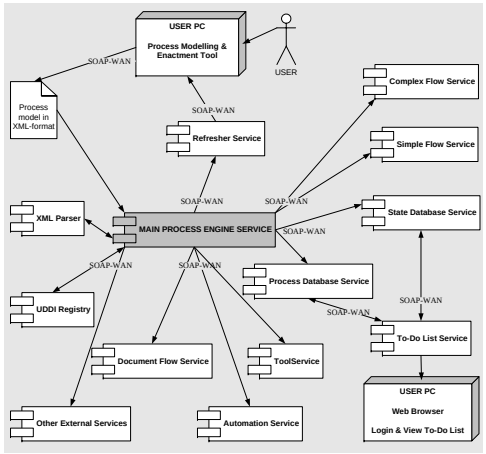


Examples

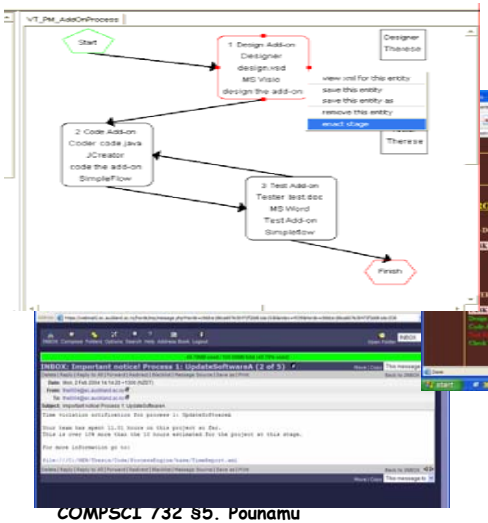


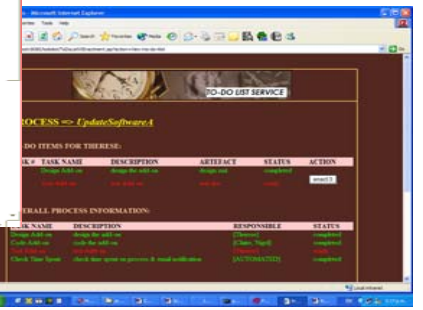
IMÅL Process Modeller

- Developed by Therese Helland (MSc)
- Pounamu specified process modelling tool
- Backend services oriented architecture integrating with process engine, decision support (Idiom), and other office automation tools (Infopath, To do list server)
- Process modelling views reused to visualise process model enactments



IMÅL Process Modeller





Pounamu Development

- Research funded by New Economy Research Fund
- Design and development of core system
  - Nianping Zhu, John Grundy, John Hosking
- Shape definer extensions: Xiaomin Tian (Project)
- Thin Client interface: Feng Luo (Project) Penny Cao (MSc)
- Collaboration interface: Akhil Mehra (Project & MSc)
- Property sheet extensions: Blazej Kot (summer schol)
- Web services interface: Therese Helland (MSc), Penny, Nianping, Akhil
- Mobile phone interface: Joe Zhao (MSc)
- Visual event handler definition: Karen Liu (PhD), Kelvin Jin (MSc)
- Zoomable user interface: Karen Liu (summer schol)