Marama

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

History

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

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

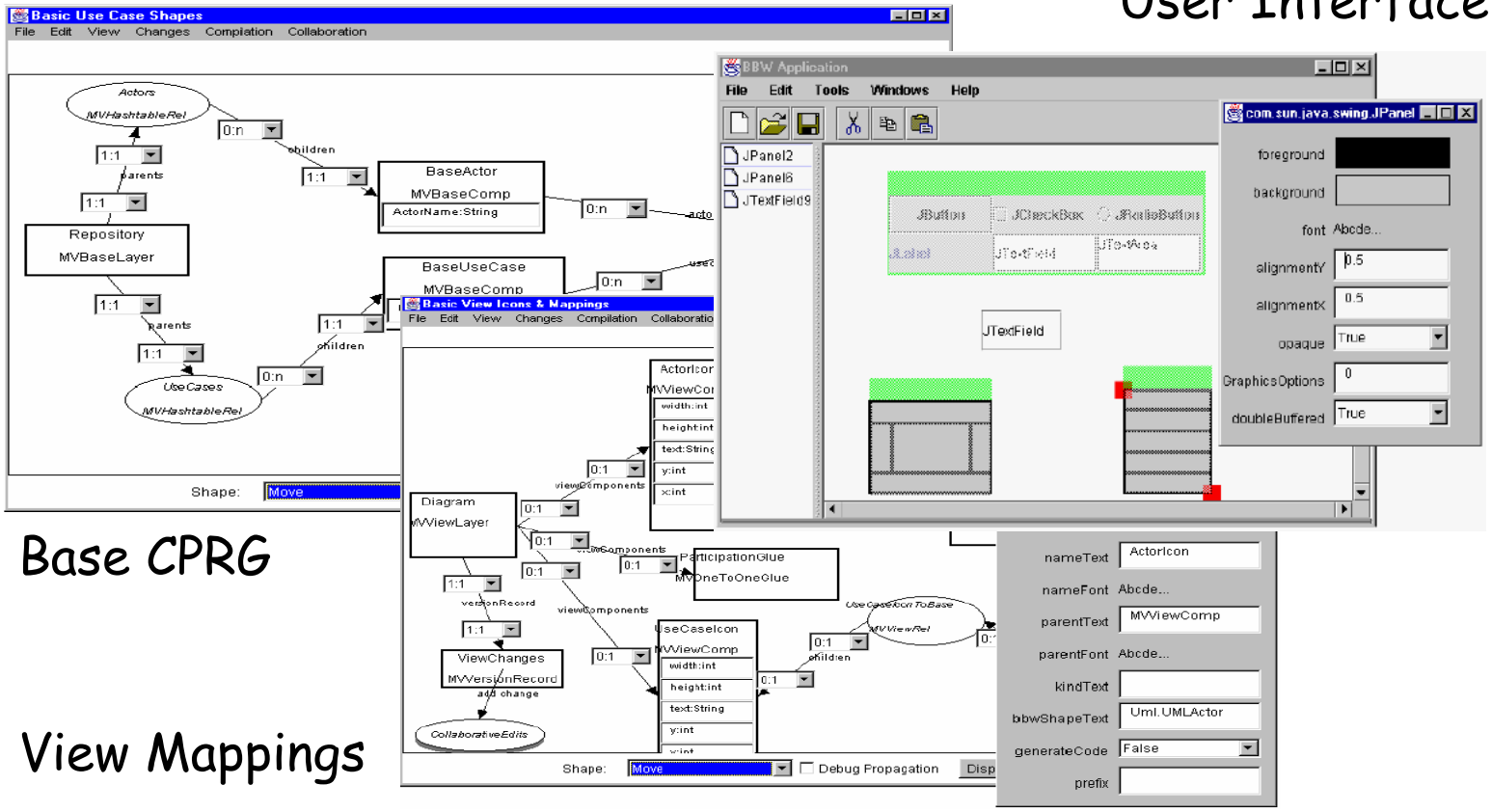
JViews/JComposer

- 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

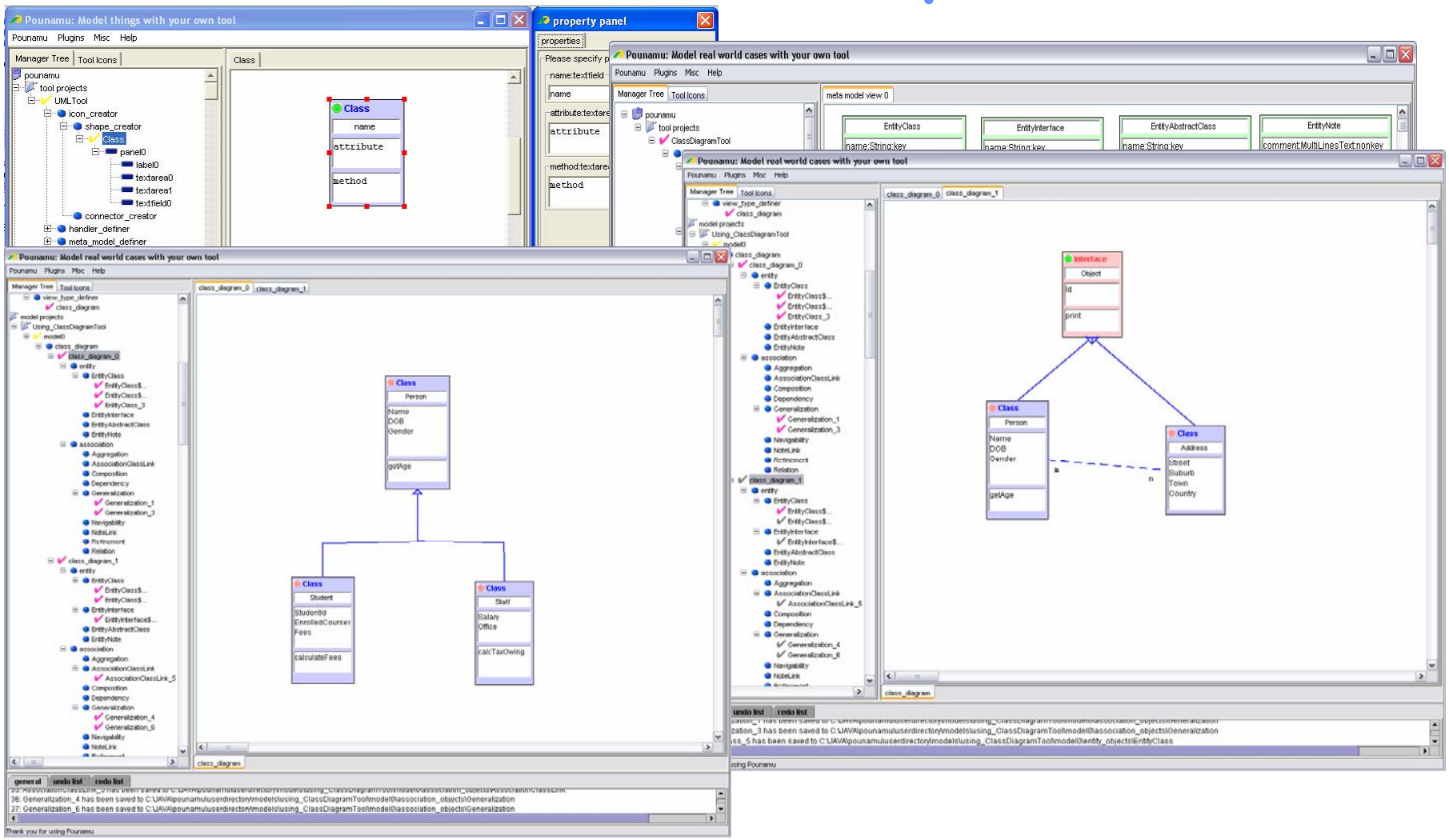
Problems

- 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 Example



Pounamu Problems

- Provided some good things:
  - Quick to build prototype DSVL tools
  - Used for wide range of exemplar systems e.g. software architecture, performance engineering tools; UML tools; UI design tools; project management tools; software process modelling tool; stats design tool; Jimi Hendriks GuitarGeek.com set-up DSVL… ☺
- BUT
  - Easy(ish) to define shapes, meta-model views BUT event handlers required Java scripting/APIs
  - All custom-built so lots of effort to maintain ourselves
  - Rather clunky UI
  - Good extension e.g. full web services API but all our own proprietary APIs

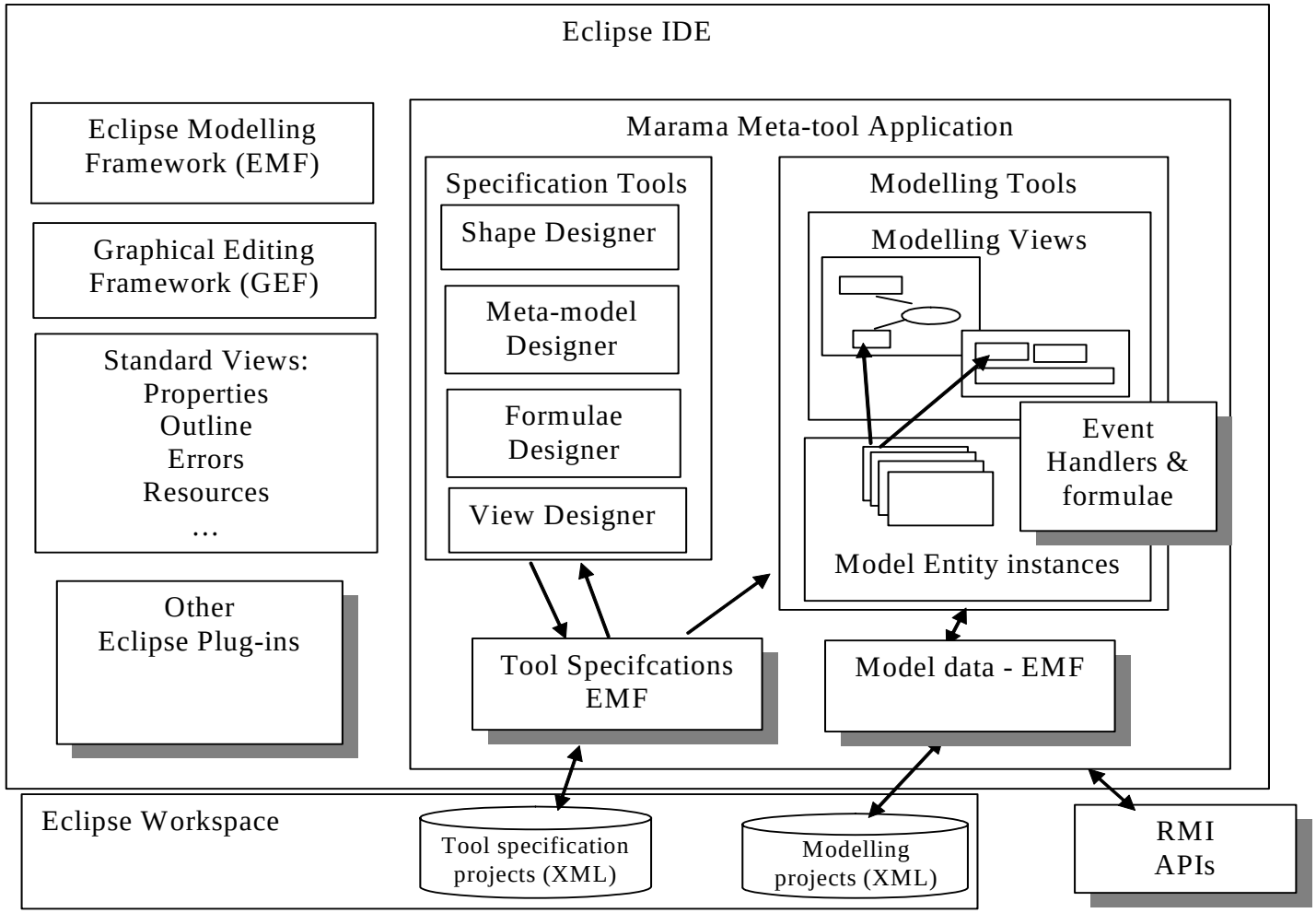
Marama & Marama meta-tools

- Developed Eclipse-based plug-ins (Marama) to read Pounamu DSVL tool specs and generate Eclipse-based tools (by Grundy, in 2005)
- This produced much richer, stable, integrated IDEs as leaverages Eclipse capabilities for many complex IDE issues e.g. UI integration, resource management, control integration, etc
- Then decided to "retire" Pounamu (be thankful! ☺ - CS732 has used for 3 years and SE462 and SE710 two years ☺…)
- Karen Liu and Jun Huh developed(ing) "Marama meta-tools" – a Marama-based set of tools to define new DSVLs
- John G has hacked a few extensions to Marama (next lecture) e.g. thin-client diagramming via SVG+JavaScript, collaborative work support via diagram diffing/merging, MaramaSketch hand-drawn support for diagram entry

Marama components

- Meta model designer tool
  - Specifies tool meta models incl. constraint-based formulae over the model
- Shape and connector creator tool
  - Used to define icons, connectors and associated properties
- View type designer tool
  - Specifies an editor for a set of shapes, connectors and handlers, and their relationship to a meta model
- Event handlers
  - Specifies dynamic behaviour in response to events (eg shape creation). Currently done via formulae, "Kaitiaki" event specification tool or Java code using Marama APIs
- Model projects
  - Instances of a specified tool in use

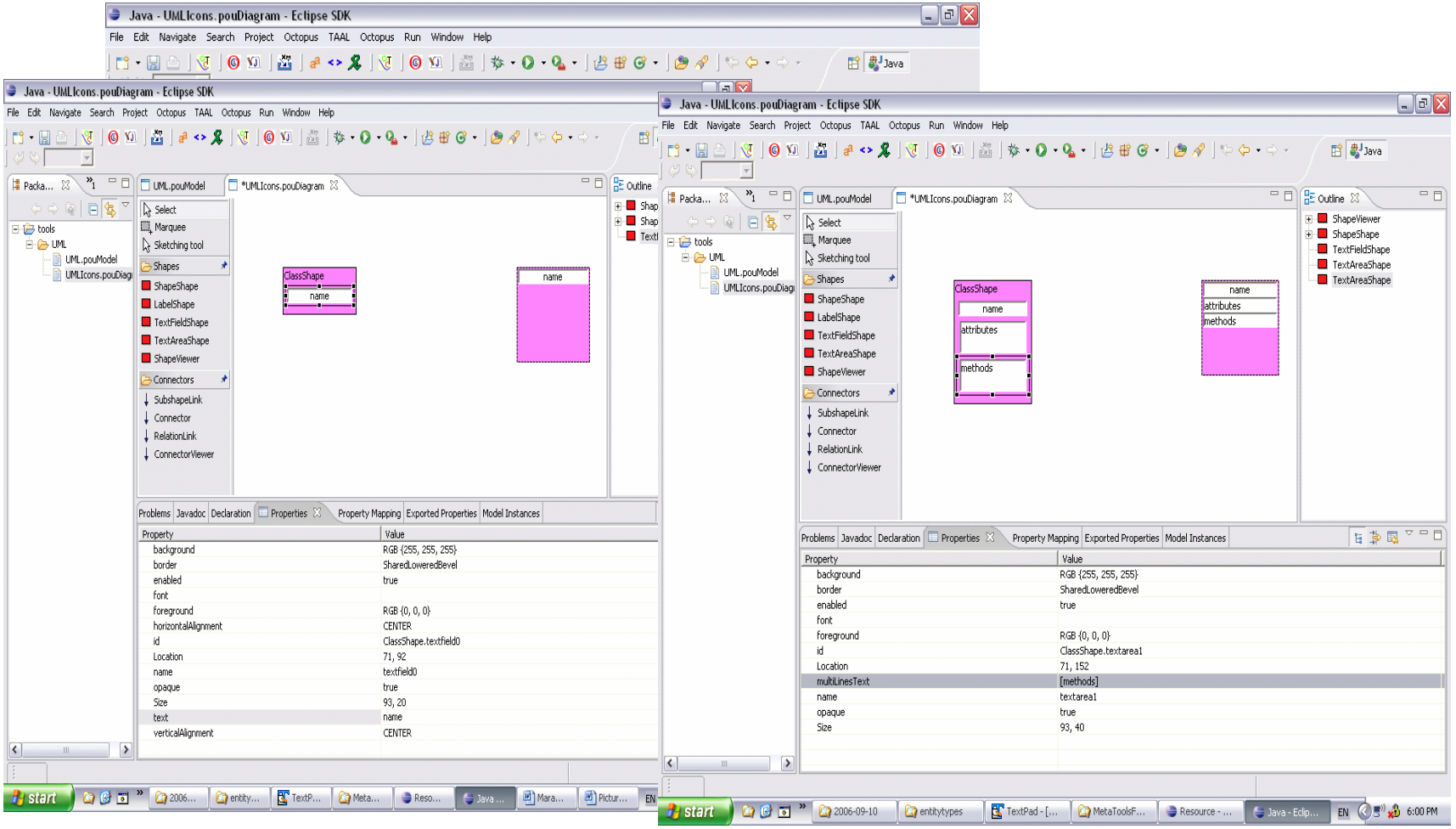
Marama approach



Simple example of usage

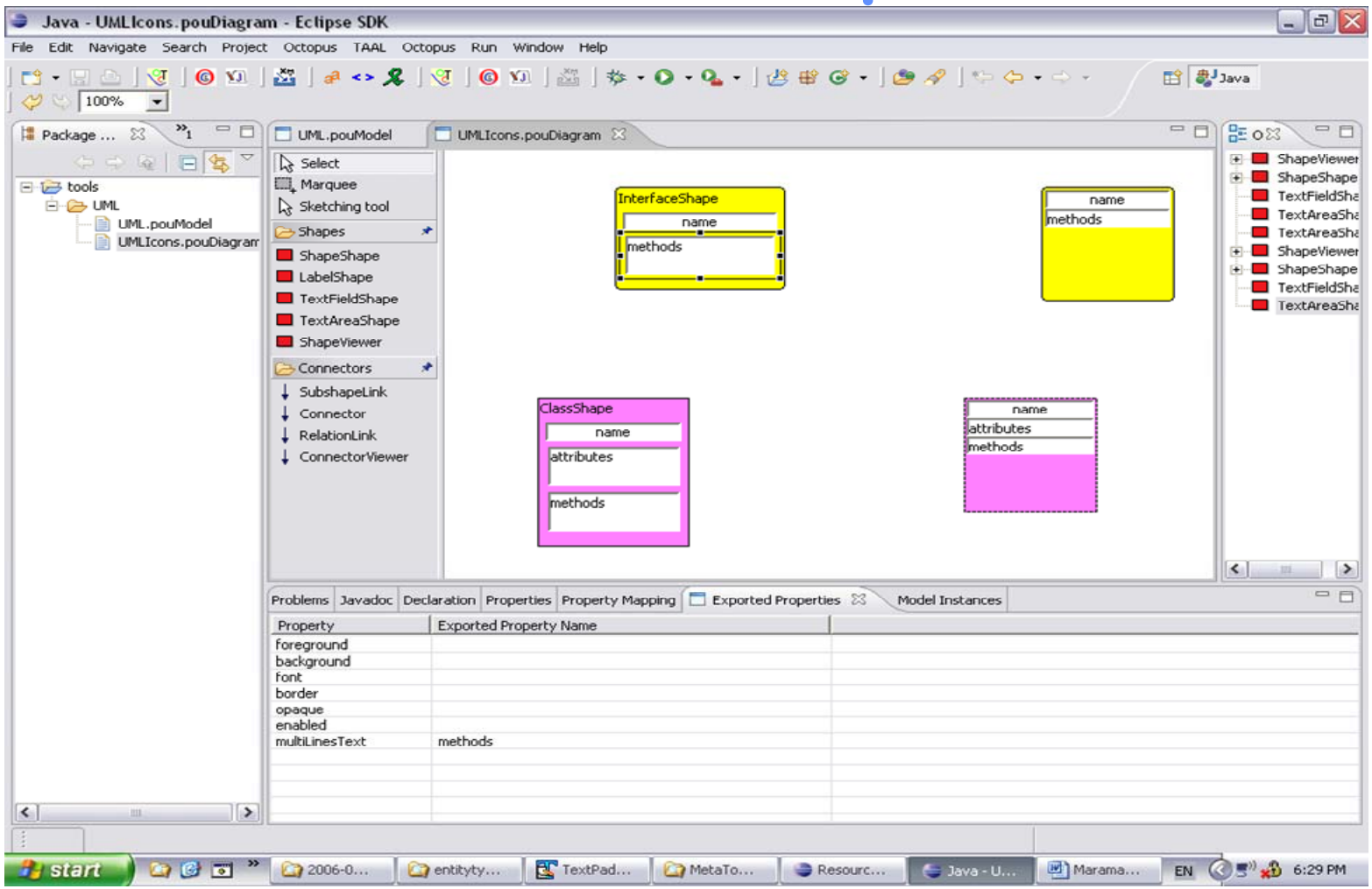
- Define a very simple UML modelling tool.
- Class diagrams have at least one type of shape and one type of connector
  - Class icon – rectangle with three regions, name, attributes and operations
  - Generalisation –unidirectional arrow
    - Connects class to class
- Start by defining shape for class
  - 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



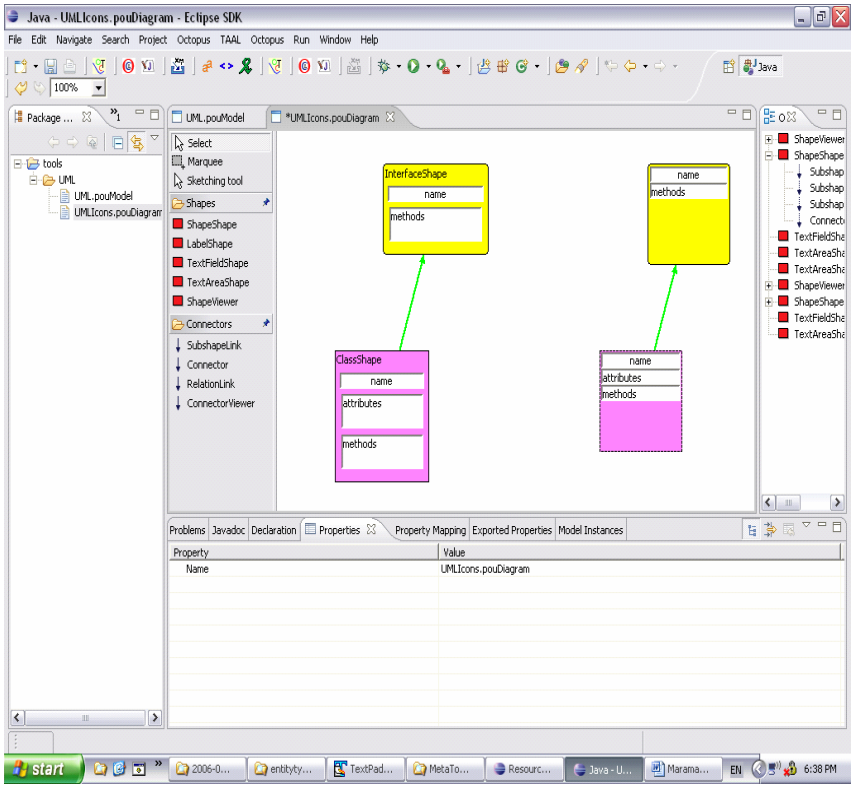
Repeat for other shapes (eg interface, note)

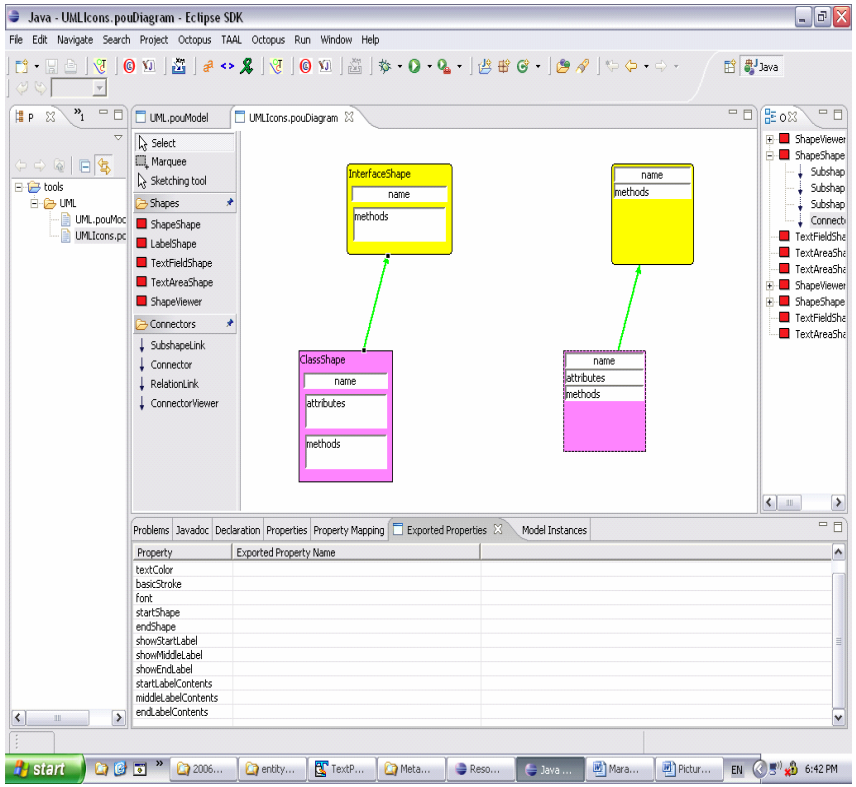
Other shapes



Connector Designer

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



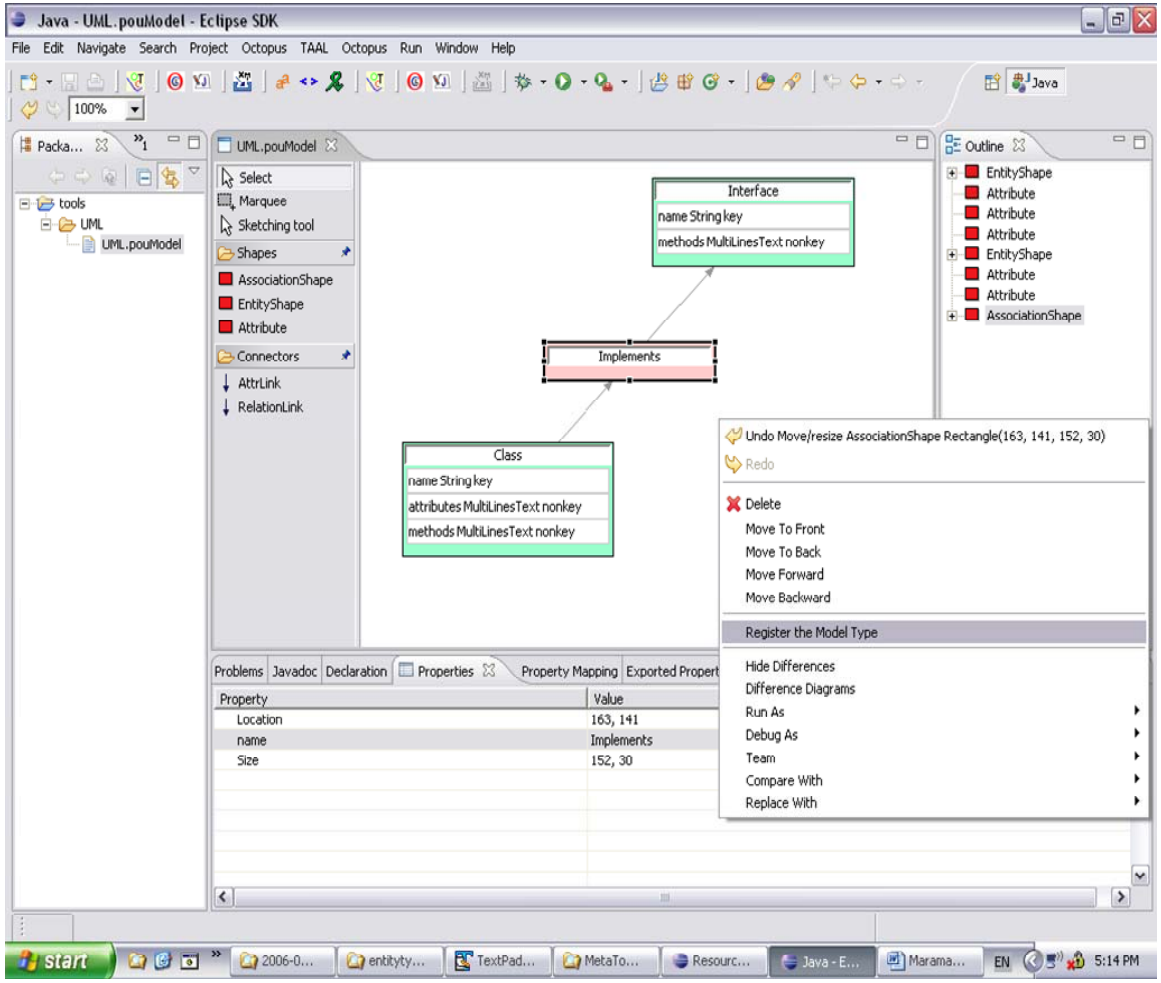


Repeat for other connectors (eg association)

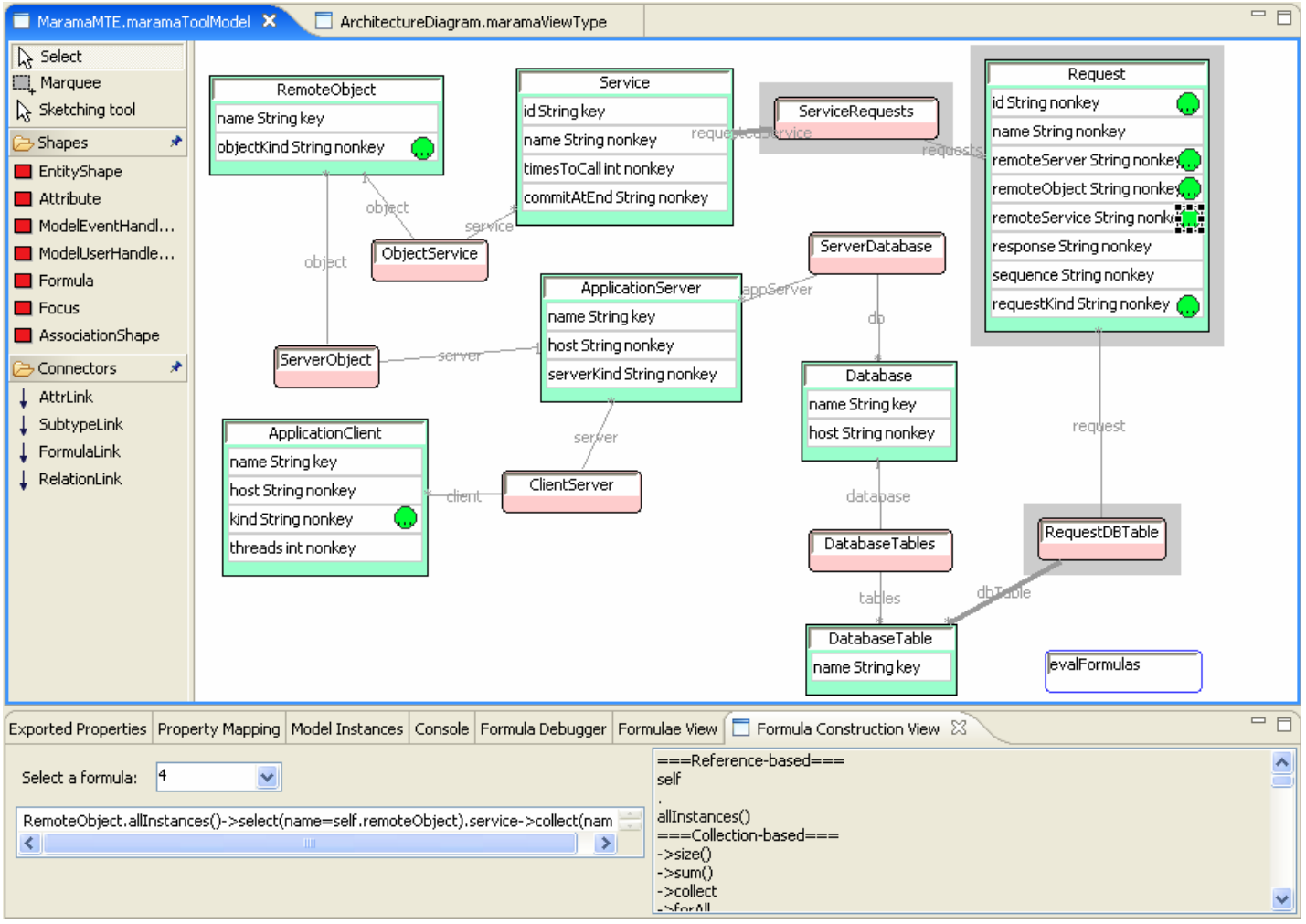
Meta model

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

Meta Model Designer



Formula Designer



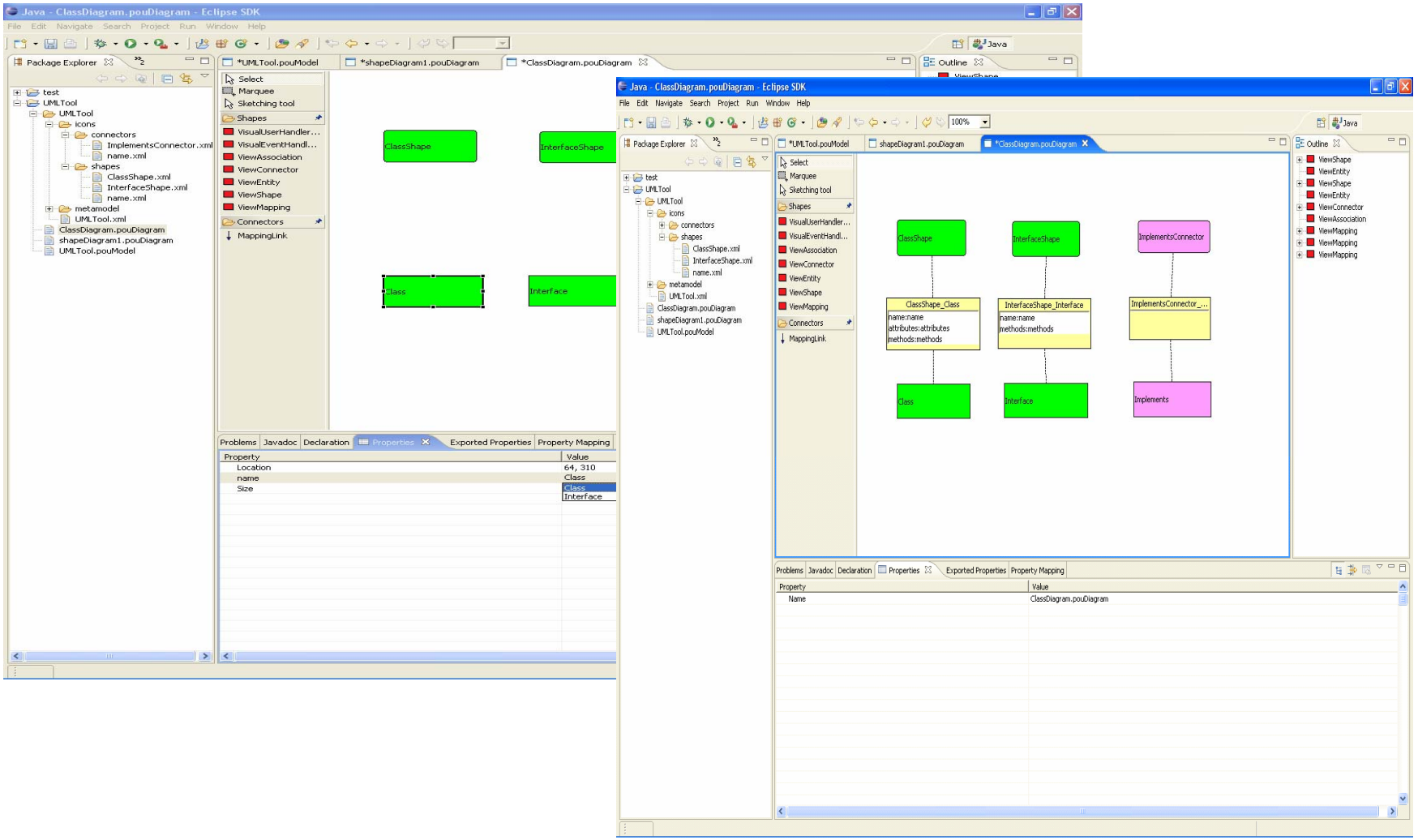
Review

- To date have defined:
  - A shape (class icon) and connector (implements) GUI components (shape & connector tools)
  - An underlying repository consisting of types for entities representing classes, linked by implements relationship types (meta model tool)
  - Formulae/event handlers providing behavioural specification and/or constraints on model/editors
- 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, implements
  - 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 formulae or event handlers

View Type Definer

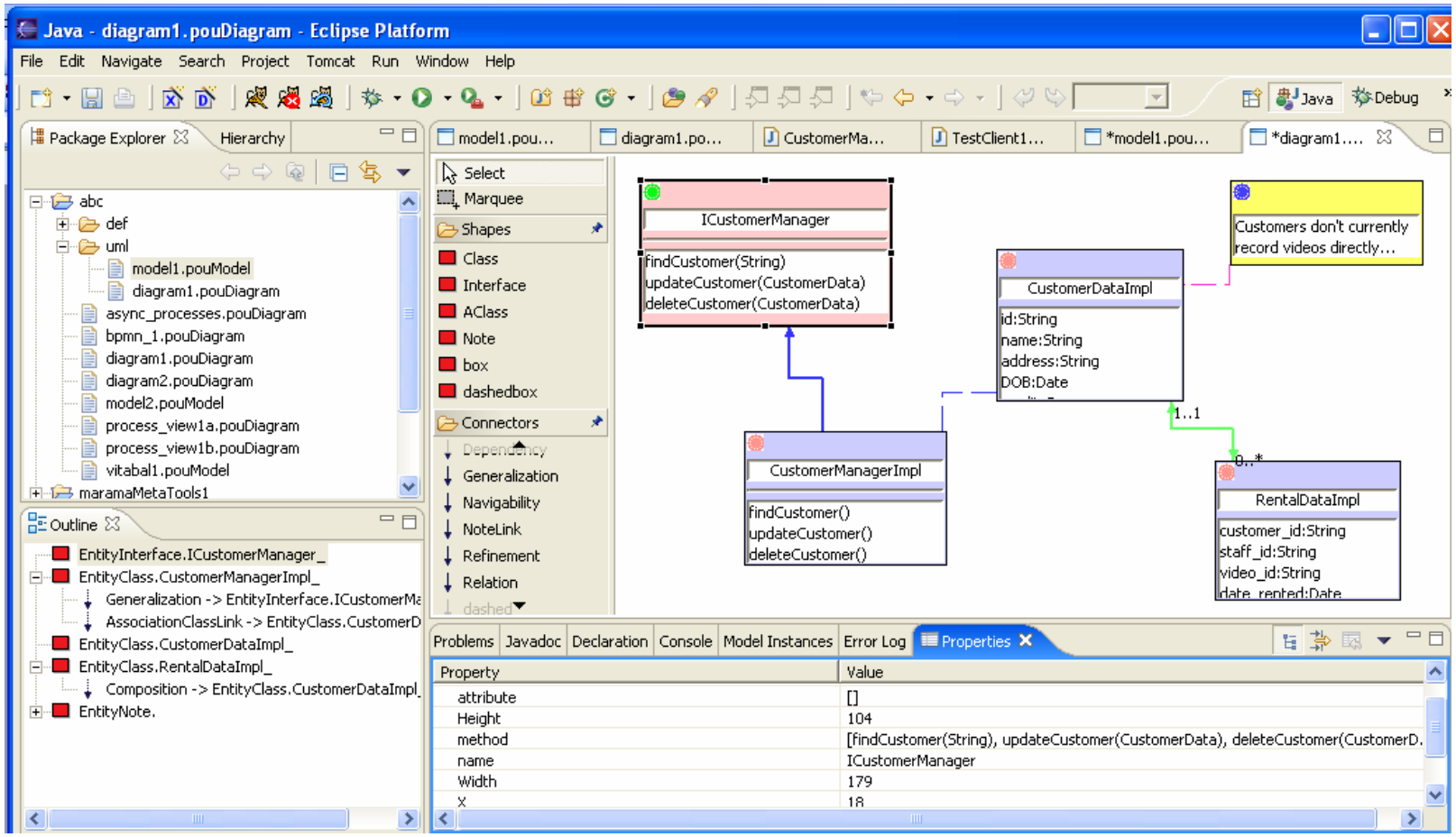


Models

- Then register everything and save the tool
- Can then create models using the tool
- Create a model project, then views

- See Marama tutorial and example tool development
- See example Whole/Part tool and more complex MaramaMTE tool off Marama downloads web page

Model projects



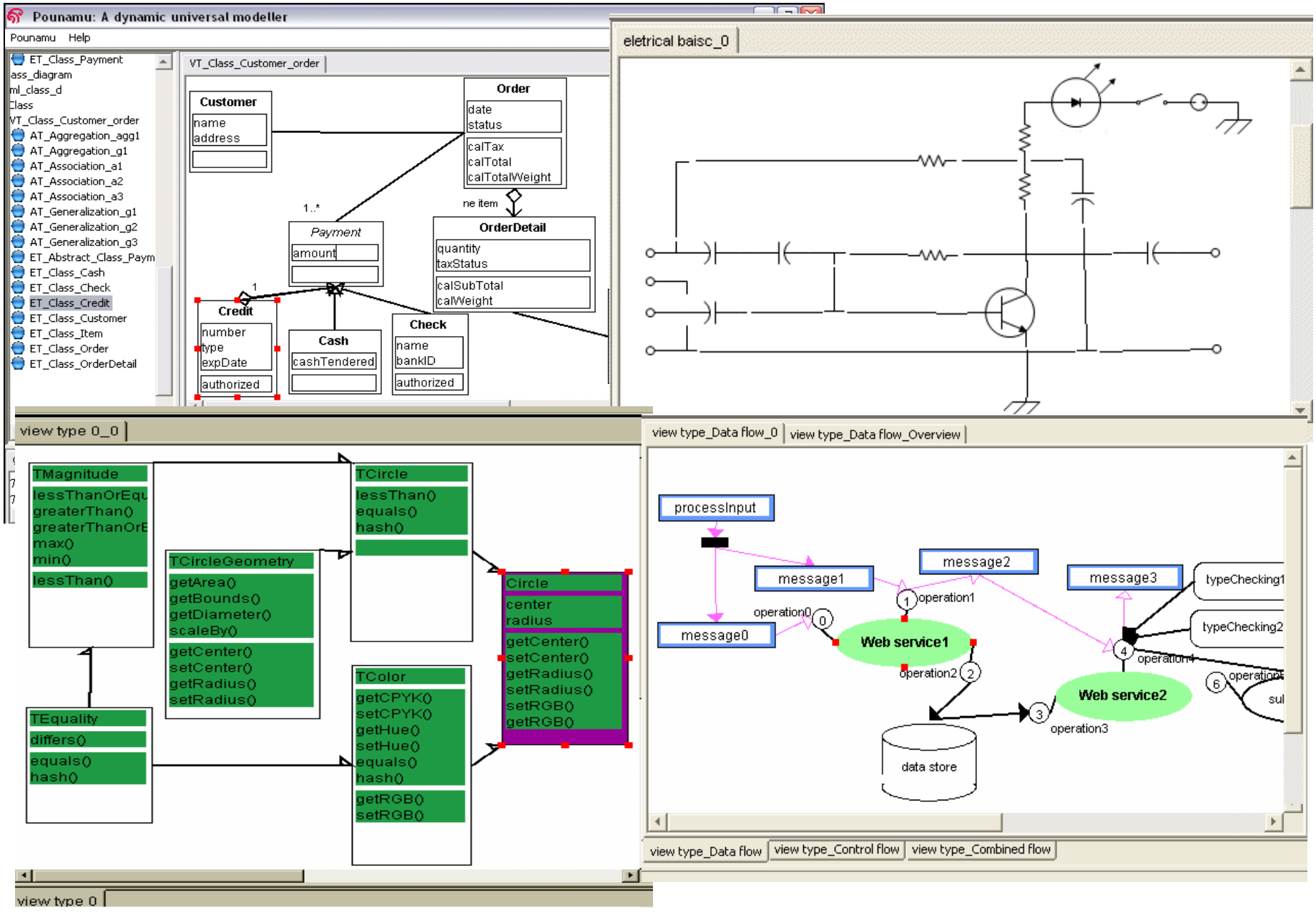
Modification, integration, extension

- Marama is live,
  - Changes to a tool specification are immediately reflected in executing models using that tool (well – usually have to close/reopen the editing views in the in-use tool… ☺)
- Formulae and Handlers provide behavioural extension capability
  - Formulae compield to OCL & interpreted
  - Handlers via API, code modified in the invoking Eclipse
- EMF data structures and Marama APIs provide internal integration with other Marama tools and other Eclipse plug-ins
  - Can have multiple Marama tools communicate
  - Can control/exchange data with other Eclipse plug-ins
- RMI interfaces provide external integration capability
  - 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 XSLT-based backends manipulating the XML save format
  - Eg for code generation and reverse engineering
  - MaramaVMLPlus data transformation tool being integrated into Marama meta-tools to support this "nicely"…

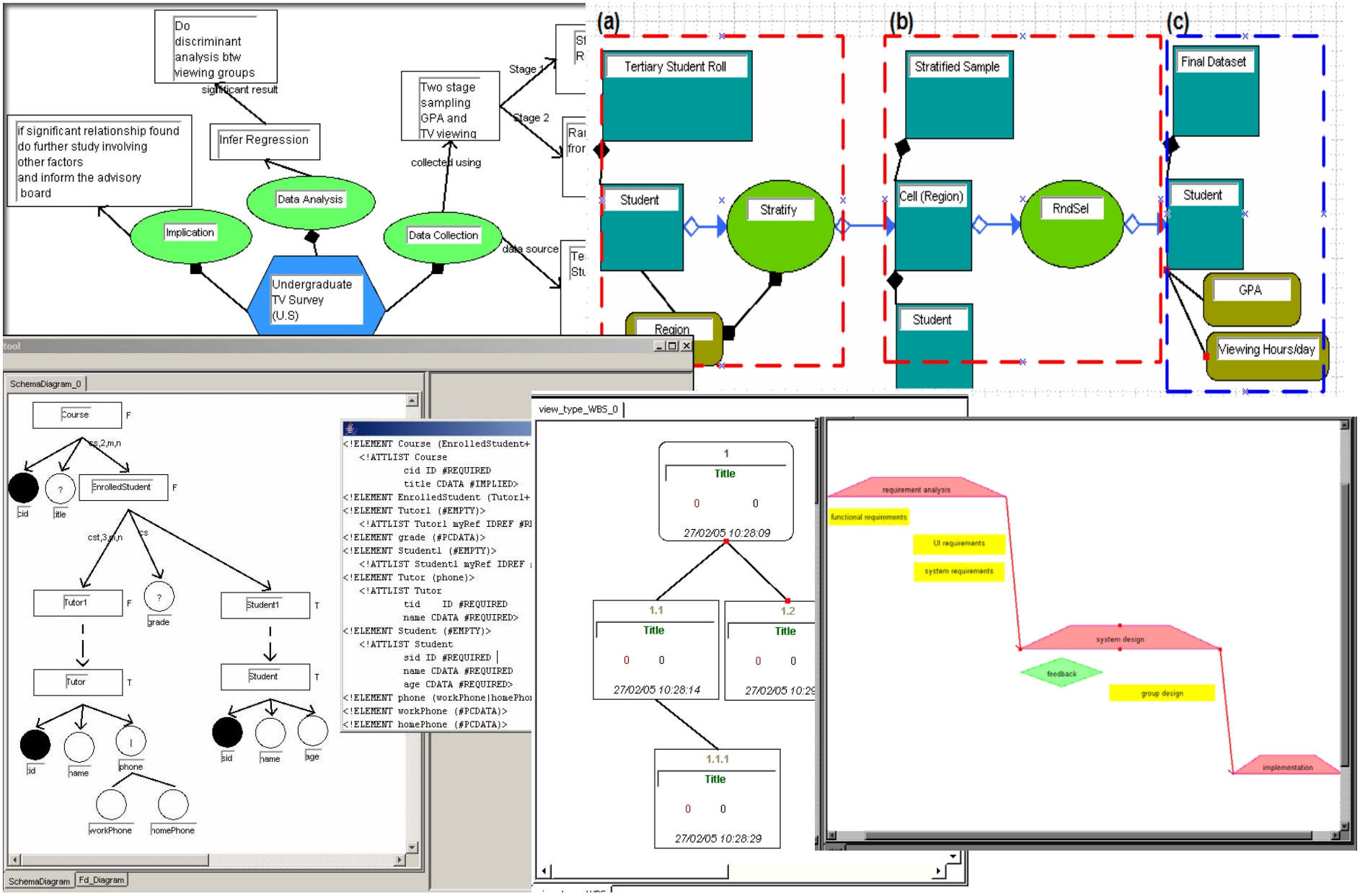
Examples

- 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
  - SDL extensions – Chul Hwee Kim MSc thesis
  - Aspect Oriented Comp Eng tool – Santokh Singh PhD
  - Web services composition tool – Karen Liu PhD
- Marama has been used to develop:
  - VMLPlus data mapping tool – Jun Huh – commercialisation project
  - Web services composition, Formulae designer – Karen PhD
  - Architecture modelling and performance engineering – Rainbow Cai PhD
  - Enterprise modelling tool, BPMN tool – Richard Li PhD
  - UI design tool, business process modelling tool – John Grundy, consulting work

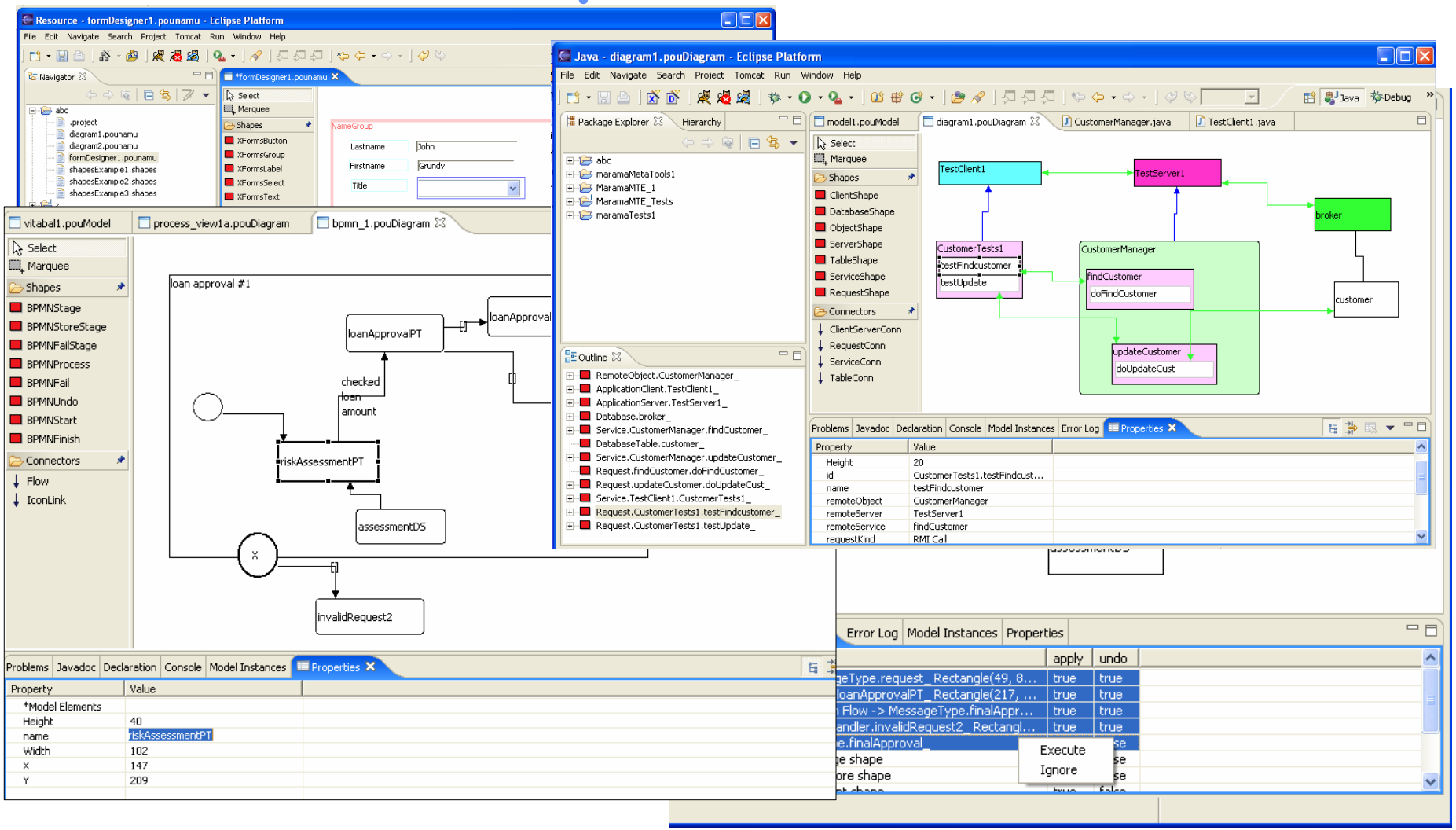
Examples - Pounamu



Examples - Pounamu



Examples - Marama



Pounamu & Marama Development

Pounamu:

- 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)
- 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)

- Marama:
  - Development of Eclipse-based DSVL editing tools
    - John Grundy
  - Development of meta-tools incl. formulae; improvement of editing tools
    - Karen Liu (PhD), Jun Huh, John Grundy
  - Import/export support using MaramaVMLPlus
    - Jun Huh
  - Various tools using Marama:
    - MaramaVMLPlus – data mapping tool – Jun Huh (Research Assistant)
    - MaramaMTE, MaramaArch – Rainbow Cai (PhD)
    - Enterprise Modelling & BPMN tools – Richard Li (PhD)
    - MaramaFormula, Kaitiaki – Karen Liu (PhD)
    - MaramaCritics – Norhayati Md. Ali (PhD)

Assignment

- Develop a prototype DSVL-based tool with Marama

- Download Marama (see web page), Eclipse if necessary
- Install Marama (see tutorial)
- Play with Marama design tools (see tutorial)
- Design & prototype DSVL tool with Marama

- Write short (3 page, IEEE format) paper on your DSVL tool and experiences with Marama