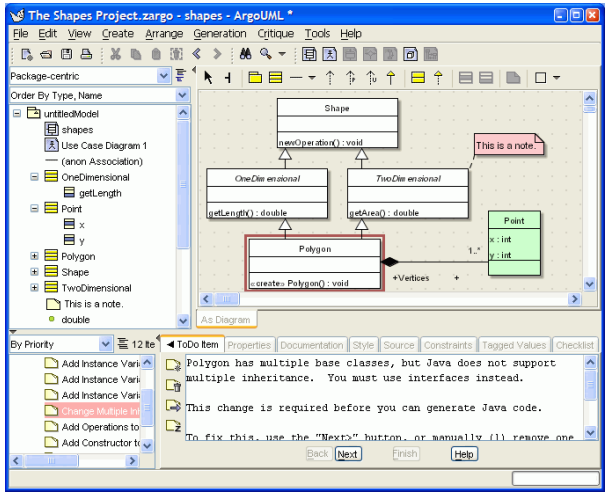
Argo

- Aims of this section:
  - Look at Argo, another software tool framework
  - Experience using Argo to develop a software tool from research prototype to near industrial strength tool
- Resources
  - ArgoUML website http://argouml.tigris.org/
    - Particularly Jason Robbin's PhD thesis and Tiziana Allegrini's dissertation
  - Cai, Y., Grundy, J.C. and Hosking, J.G. Experiences Integrating and Scaling a Performance Test Bed Generator with an Open Source CASE Tool, Proc 2004 IEEE Int Conf on Automated Software Eng, Linz, pp. 36-45.
    - Slides from the conference presn of this paper

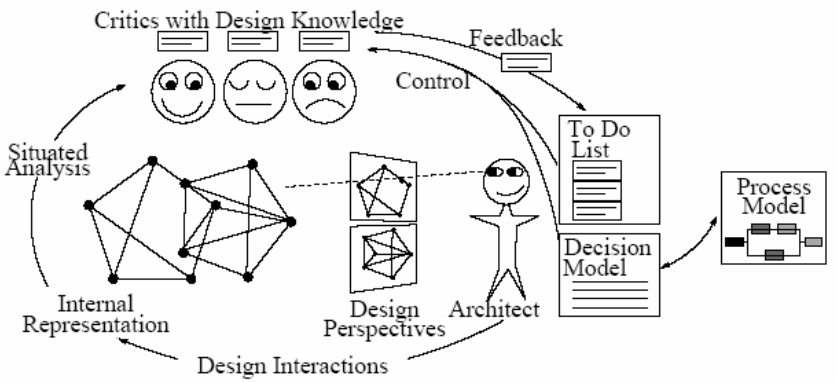
Argo and ArgoUML

- **Argo UML project goal: build an object oriented design tool that is:**
  - **a joy to use**
  - **actually helpful to designers when they are making design decisions, by offering cognitive support through critics**
  - **completely open source Java (FreeBSD license)**
  - **supporting everything in UML**
  - **modular and extensible**
  - **integrated with the web and other Tigris tools.**
- **Argo is the framework underneath the ArgoUML tool**

ArgoUML in use



Basic functionality

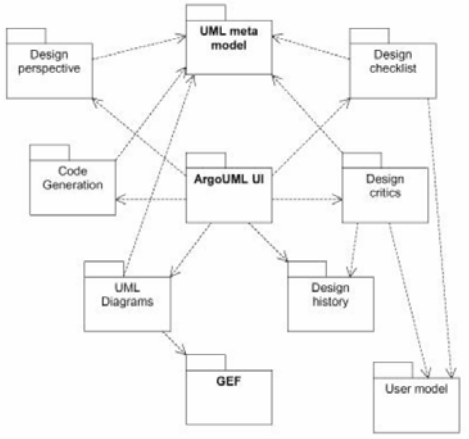


Basic functionality

- Design Perspectives
  - multiple views with consistency
- Critics
  - Multiple analysis tools which provide continual feedback on the design
- To do list
  - Feedback from critics presented here, provides active links to "criticised" design elements
- Process Model
  - Integrated process modelling (cf Serendipity) using IDEF0 notation
  - Linked to critics, so can have task specific critics

Argo Architecture

- Major Packages:
- GEF
  - Graph Editing Framework provides reusable graph editing capabilities
- UML Meta Model
  - Based on NSUML open source UML meta model
- ArgoUML UI
  - Windowing and navigation
- Design Critics
  - Support for design critic implementation and predefined critics

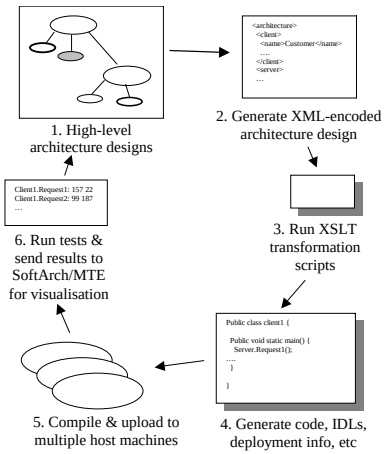


Experience Applying Argo

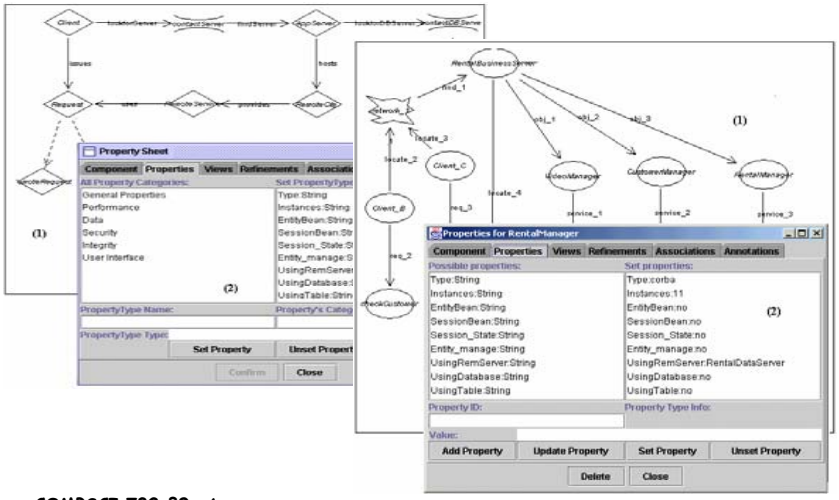
- SoftArch/MTE and its problems
- Re-engineered solution
- Experience
  - Integrating MTE with Argo/UML
  - XMI-derived model representation
  - Improvement of XSLT-based test bed generator
  - Using ANT
  - Result database
- Conclusions
  - Specific
  - Generalised

SoftArch/MTE

- SoftArch/MTE (ASE2001)
  - integrated environment to model and evaluate software architecture
  - automatically generates, compiles and deploys test bed code, runs performance tests, reports results



SoftArch/MTE (Cont'd)



SoftArch/MTE problems

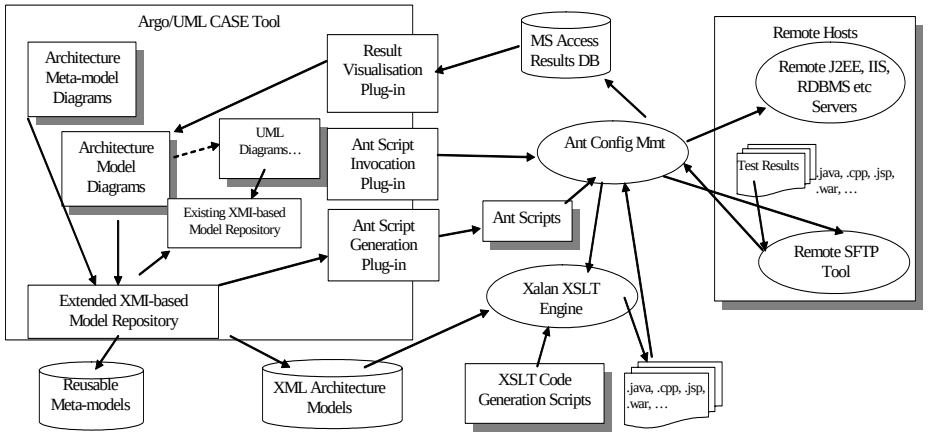
Problems when we applied SoftArch/MTE to several industrial case studies:

- custom framework (JViews)
- custom architecture notation
- custom XML representation
- non scalability of code generation approach
- custom deployment tool
- custom visualisation

Re-engineered Solution

- Use Argo/UML as base tool
  - wider user base and more robust framework
  - integration with a standard UML modelling tool
- Extend UML meta model with arch descpn/perf elements
  - base on a more standard formalism
- Develop arch perf meta model and instance modelling tools in Argo
- Use standard XMI backend model representation
- Make XSLT based code generator more generic
- Use standard deployment tool (Ant)
  - Manages test code deployment and test run
- Use standard DB (Access) for result mmt and visuln

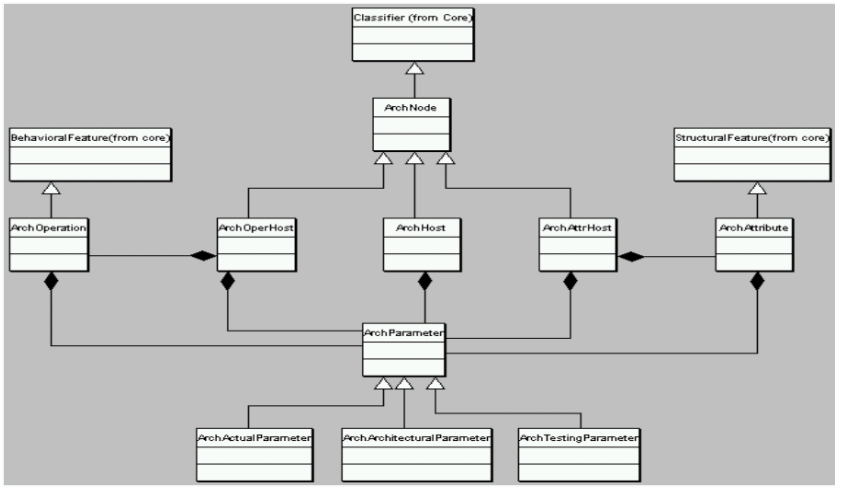
Re-engineered Solution



Integrating MTE with Argo/UML

- Choice of Argo
  - Public domain with a reasonable user base (pre Eclipse)
  - Well designed around UML meta model
    - Better maintainability
  - Designed with extendibility in mind
- Integrating MTE with Argo/UML
  - Specialise Argo's UML meta model with a set of performance-oriented architectural modelling elements
  - Develop an Argo meta-modelling tool to model and record domain-specific performance modelling knowledge
  - Develop an Argo architecture modelling tool
    - supports modelling of concrete architecture designs, which are instances of domain-specific meta-models

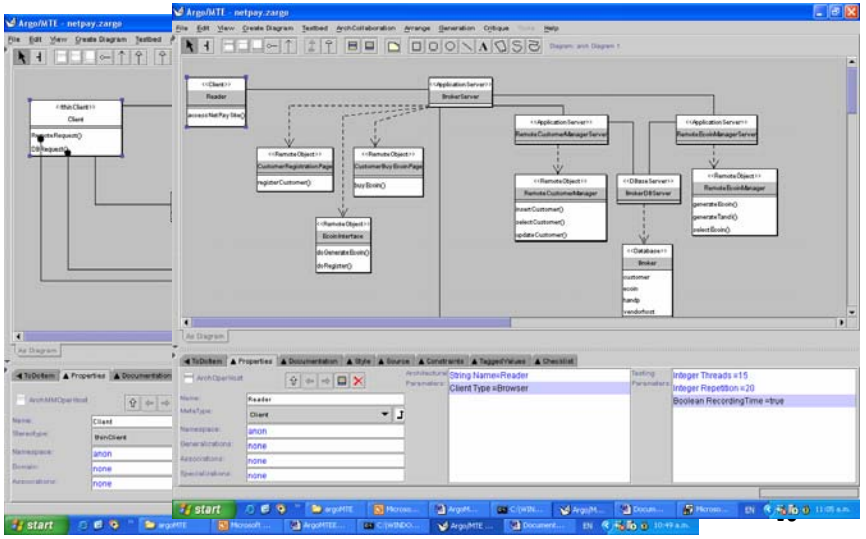
UML Meta model Extension



Elements & Attributes

Client : ArchOperHost	**ClientType (AP, TP)** **Threads(TP)**	**Type of a client e.g. browser, CORBA client.** **Number of con-current clients run for tests**
RemoteRequest : ArchOperation	**RemoteServer (AP, TP)** **RemoteObject(AP, TP)** **RemoteMethod(AP, TP)** **RecordTime(TP)** **TimesToCall(TP)** **PauseBetweenCalls(TP)**	**Name of remote server to call** **The name of remote object** **The name of remote service** **Record time for this?** **Repetitions** **Pause duration between calls**
AppServer : ArchHost	**Name (AP, TP)** **RemoteObjects(AP, TP)** **Type (AP, TP)**	**Name of server** **Objects this server hosts** **Type of the application server, e.g. CORBA, RMI, J2EE…**

Argo/MTE Modelling



XMI-derived Model Representation

- Why an XMI-derived model representation?
  - To represent architecture models in a more standardised format
  - To eventually make Argo/MTE model data exchangeable with other XMI-supporting CASE tools.
- How implemented?
  - Add tags based on performance-oriented architectural modeling elements.
  - Add a set of performance-oriented architectural modeling stereotypes and add XMI-style tags.
  - Short term have had to modify Argo XMI reader as no agreed architectural extensions to XMI
  - Later, adapt our notation and representation to evolving standards

Improved code generator

- SoftArch/MTE
  - uses two-layer model to model software architecture:
    - domain-specific meta-model →architecture model.
  - XSLT sheets mainly scripts for domain-specific meta-modeling abstraction
  - XSLT sheets need to be modified whenever meta-model changed
- Argo/MTE
  - uses three-layer model to model software architecture:
    - predefined stereotype →domain-specific meta-model →architecture model.
  - Argo/MTE XSLT sheets mainly scripts for extended UML performance-oriented architectural modeling elements and predefined stereotypes.
  - In most cases, users can generate a test bed for whatever they've modeled without touching the test bed generator.

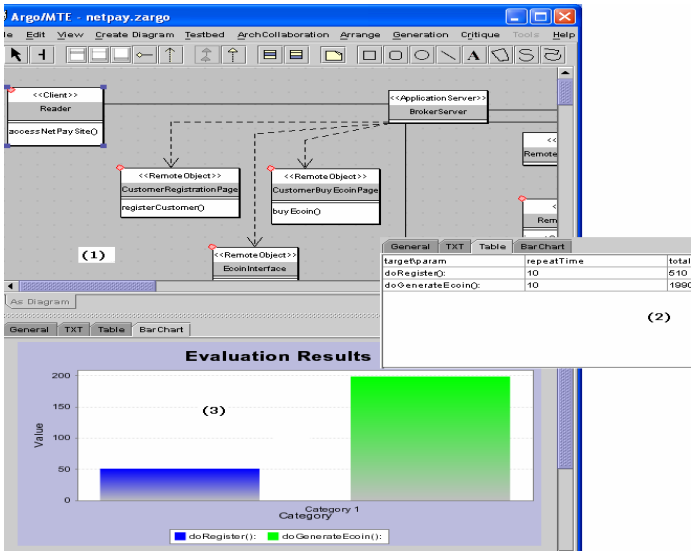
Ant automates evaluation Process

- Managing the performance evaluation is complex
  - generation, compilation, deployment, execution, result collection of test bed
  - generated test bed is a large distributed computing system
  - process is error-prone even when done manually.
- Use of Ant to manage this process.
  - Ant is designed to manage problems of priority.
  - Ant is easy to install, use, and maintain.
  - Ant build files are much easier to maintain and generate than DOS batch files (easier script generation).
  - Ant also provides a good paradigm to organize materials and files generated and needed in the evaluation process.
- Similar gains by use of SFTP for remote deployment and ACT for thin client tests

Result Database

- Choice of Access
  - Standard SQL database component
  - Visualisation scripting available
- Enables users to
  - compare different architecture models
  - compare the same architecture model working in different operational situations
  - archive the results of various refinement of an architecture model
  - export or import models from other modeling tools
  - annotate architecture component diagrams with performance results

Results visualisation



It works

- Modelled an already developed e-payment system
- Results significantly different from actual system performance
  - testbed suggested better performance than actual
- Identified error in e-payment system causing poor performance
- Corrected this and performance matched testbed

Conclusions

- Integrated Modelling Support

  Argo/MTE integrated with a standard UML-based CASE tool

  Allows test bed modelling and generation as a natural adjunct to UML modelling

  Reuses users' design notation knowledge reducing learning curve

  More appealing and effective environment than stand-alone SoftArch/MTE

- Enhanced data exchange capability

  Extended XMI model representation and extensible architecture meta-models increase chance of future model data exchange

Conclusions (cont'd)

- Better abstraction led to simpler code generation

  Addition of stereotype abstraction layer led to better reuse of code generation code & scripts

  Avoided the need for manual modification of code generation scripts

- Use of third-party tools

  Third-party tools used to coordinate:
  - test bed generation and execution process (Ant),
  - deployment (SFTP),
  - web-based client tests (ACT)
  - results management (Access)

  Much more scalable and flexible than our previous ad-hoc applications to perform these tasks.

  Particularly so for heterogeneous architectures incorporating several technologies

Generalised Conclusions

- Leverage third party tools in specialised domains
  - Complex dependency management
  - Scripting
  - Databases
  - Modelling tool implementation

  Avoid bespoke code (concentrate on your own strengths)
- Design for extendibility/reuse
  - Use abstractions to enhance reuse
  - Use plugin/API technologies to make integration easy
- Use standard representations where possible
  - Enhances user adoption
  - Enhances reuse and tool integration