



THE UNIVERSITY OF AUCKLAND  
[www.auckland.ac.nz](http://www.auckland.ac.nz)

# Some Recent Directions in Automated Software Engineering Research

Professor John Grundy

Dept Electrical and Computer  
Engineering & Dept Computer Science



# Outline

- ❖ What is “Automated” Software Engineering?
- ❖ Some recent ASE research @ UoA:
  - Meta-tools
  - Performance engineering via test-bed generation
  - Component discovery/integration/validation
  - Collaborative work
  - Adaptive user interfaces
  - Software architectures to support this stuff
- ❖ Conclusions

# Automated Software Engineering

- ❖ Generative – try and generate code from high-level, abstract descriptions (models)
- ❖ Component-based – “build applications from bits” approach; ultimately “autonomous agents” composition
- ❖ Adaptive – components/agents discover environment and adapt to the circumstances they find themselves in
- ❖ Dynamic – ideally can do the above at run-time while the software is in use
- ❖ Formalisms necessary – specifications we can reason with; generate code from; verify vs validate models/code

# Why?

- ❖ Code is too low-level for tasks we want it for – who wants to write code anyway?
- ❖ Engineering is about building models of problems/products – can we have models higher level than code? If so, can do much more with them than with program code...
- ❖ Can generate huge code base from small abstract models (if all goes to plan...)
- ❖ Some successes – domain-specific languages; IDEs; 4GLs; rapid prototyping tools; CASE/CAD tools; hardware synthesis
- ❖ Still lacking sufficient formal models for practical use
- ❖ Validation/verification become crucial issues
- ❖ Can get emergent behaviours esp. with agent-based systems

# Examples from our work

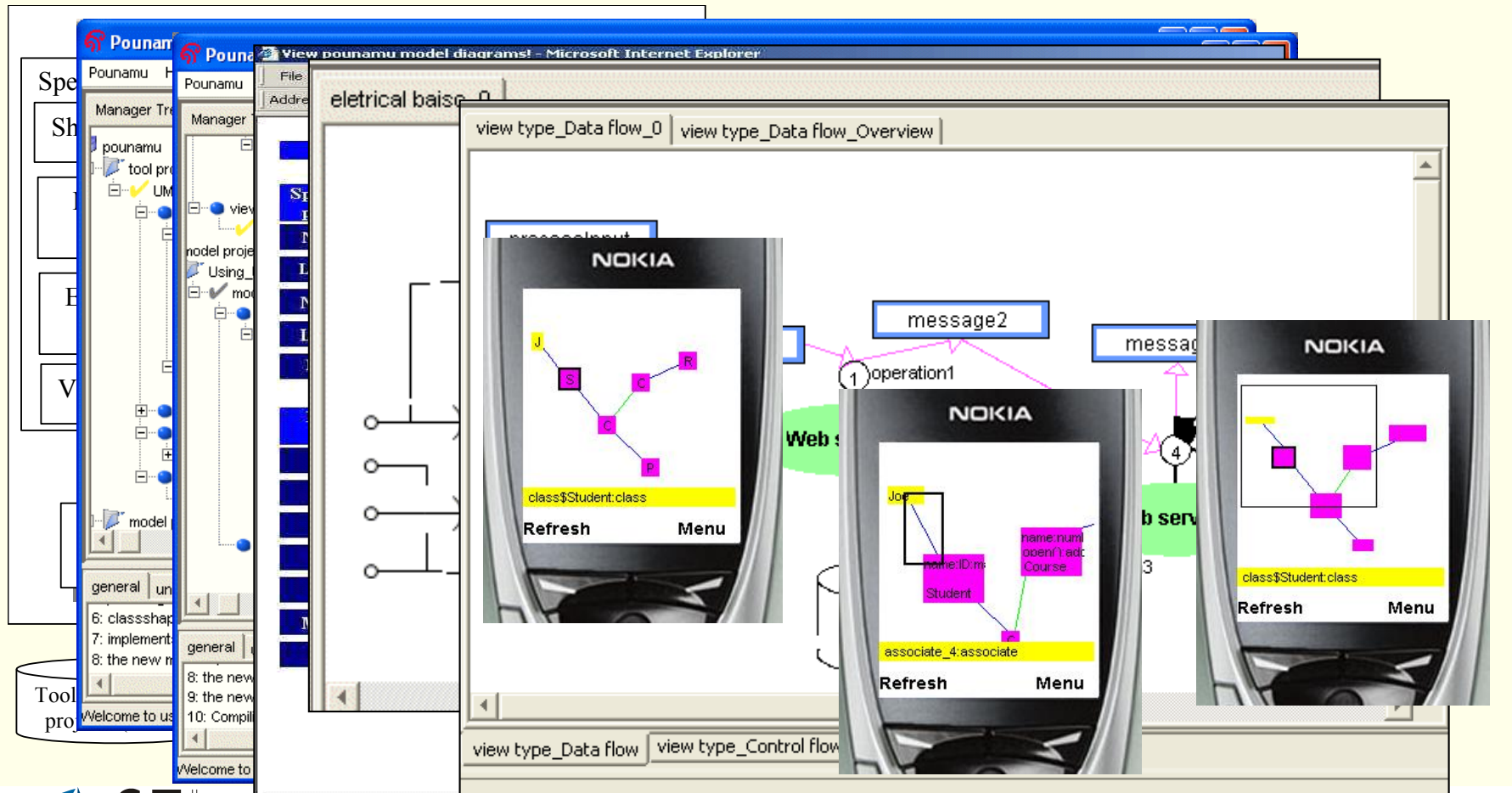
- ❖ Pounamu (a meta-CASE tool):
  - Specifying & evolving software tools
- ❖ Argo/MTE (a test-bed generator):
  - Performance engineering via test bed code generation
- ❖ Aspect-oriented component engineering:
  - Adaptive component-based systems
  - Deployed component validation
- ❖ Adaptive user interfaces:
  - E-whiteboard, PDAs, mobile phones etc.
- ❖ Some of our current/future directions...

## Meta-tools: Pounamu

- ❖ A meta-tool (tool for building tools...)
- ❖ Specify visual, multiple view “tools”/applications
- ❖ Framework provides for dynamic, run-time tool modifications
- ❖ Various plug-in extensions: web-based & mobile PDA/phone-based diagramming; collaborative work support; web services APIs; dynamic tool integration

# Pounamu example

HCC 2003  
HCC 2004  
AUIC 2005  
ASWEC 2005



## Code generation: Argo/MTE

- ❖ For performance engineering – very difficult to estimate likely system performance during design
- ❖ Our approach: generate real performance test-bed (code etc) from software architecture model
- ❖ Run performance tests and visualise results
- ❖ Extension to ArgoUML open-source CASE tool



# Argo/MTE example

ASE 2001  
SEKE 2004  
ASE 2004  
ASE J 2005

The screenshot displays the ArgoUML interface with a UML class diagram. Red circles and arrows highlight specific elements: (1) the Reader class, (2) the Broker class, (3) the BrokerServer class, and (4) the EcoInPage and ArticleInterface classes. Below the diagram, an 'Evaluation Results' bar chart shows the performance of two methods: doRegister() and doGenerateEcoIn().

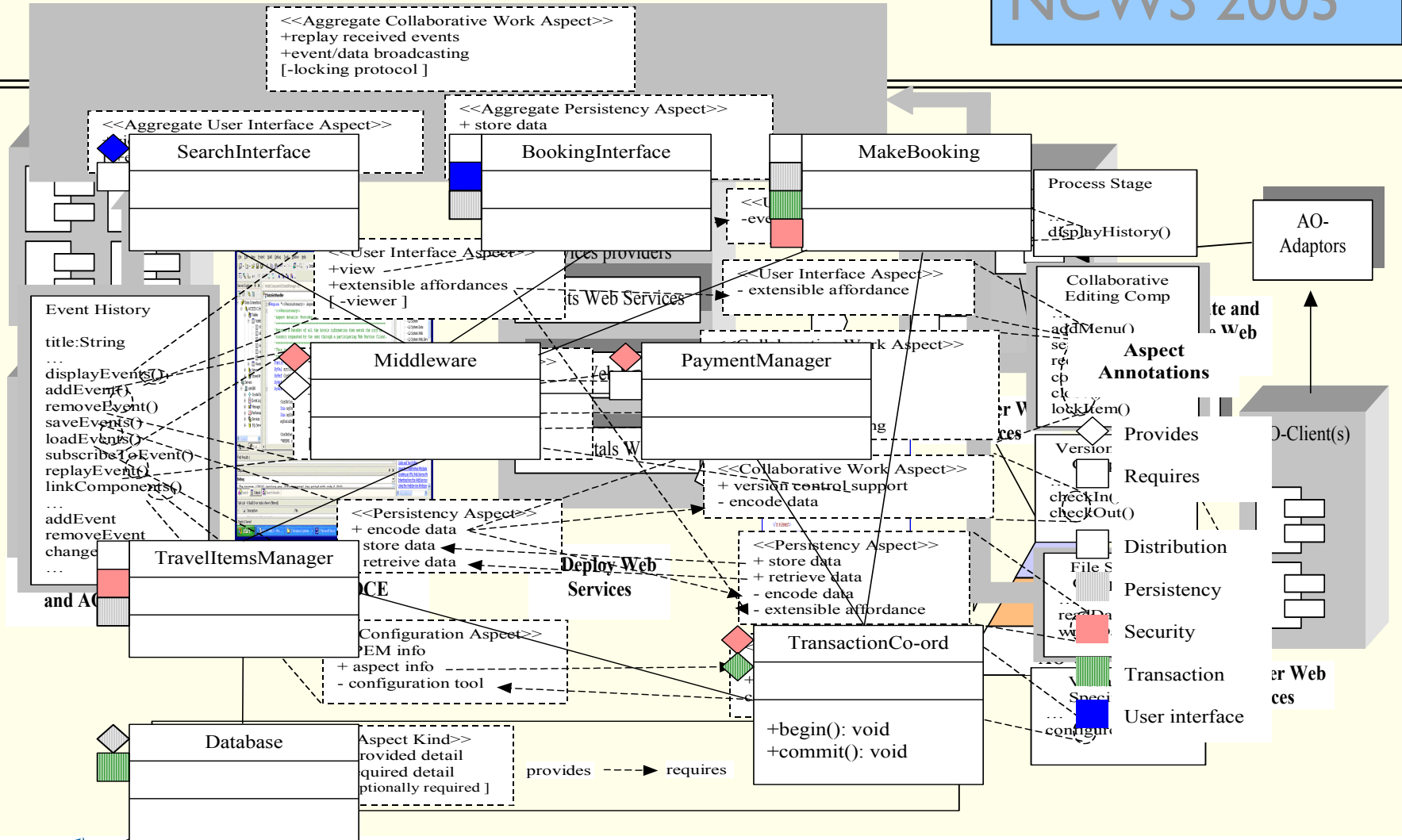
Method	Value
doRegister()	50
doGenerateEcoIn()	190

## Components: AOCE

- ❖ Software components = “build from bits” model
- ❖ Problem – understanding/characterising the components to use
- ❖ Our solution – apply aspect-oriented techniques to identify cross-cutting concerns to characterise
- ❖ Developed method, basic tool support
- ❖ Can apply at run-time for dynamic adaptation too

# AOCE Examples

IJSEKE 2000  
S-P&E 2002  
NCWS 2003

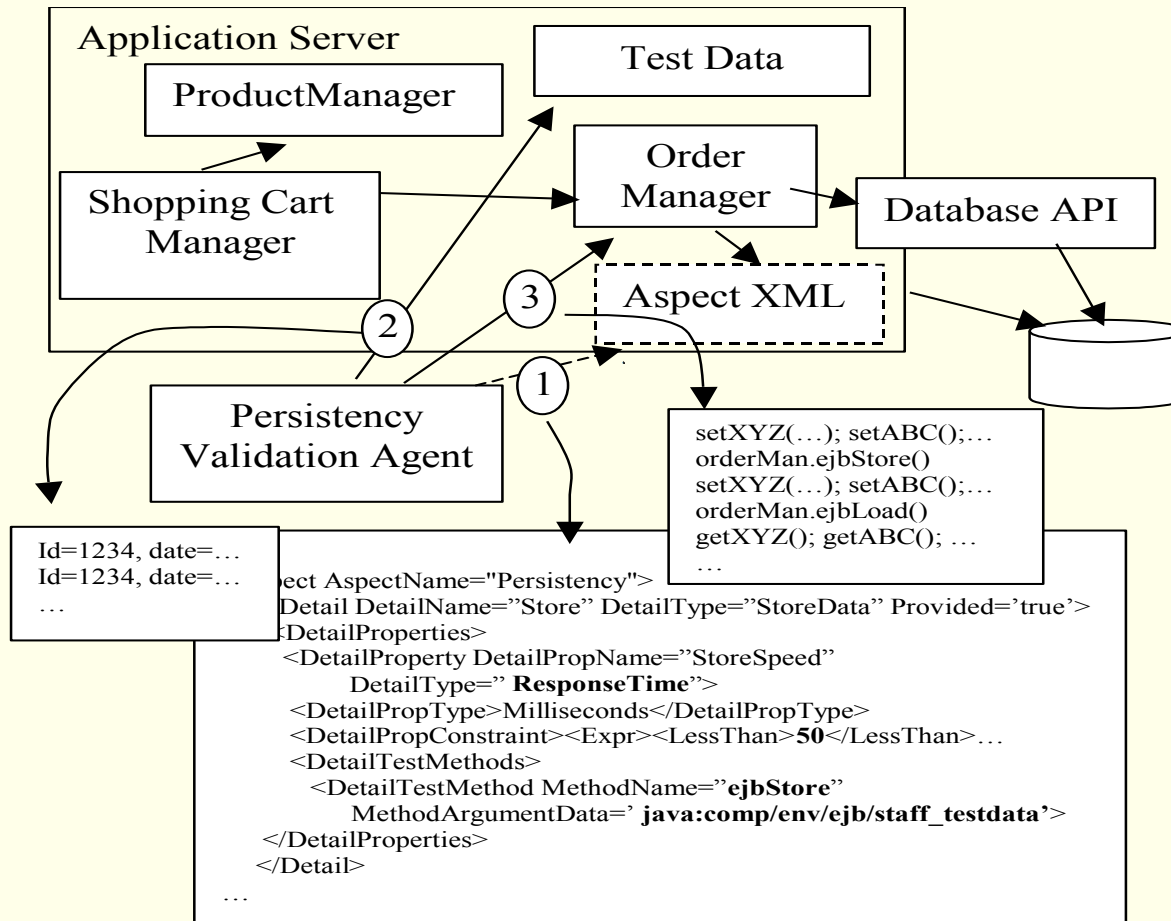


# Component Validation

- ❖ How do we check deployed components meet their requirements?
- ❖ Our approach:
  - Characterise component behavioural/non-functional requirements
  - When deployed, inspect these characteristics
  - Synthesise tests to check these constraints have been met
- ❖ Requires more detailed information about components at design/run-time

# Component Validation Example

ASE 2002  
NCWS 2003  
JSS 2004



# Some of our current work

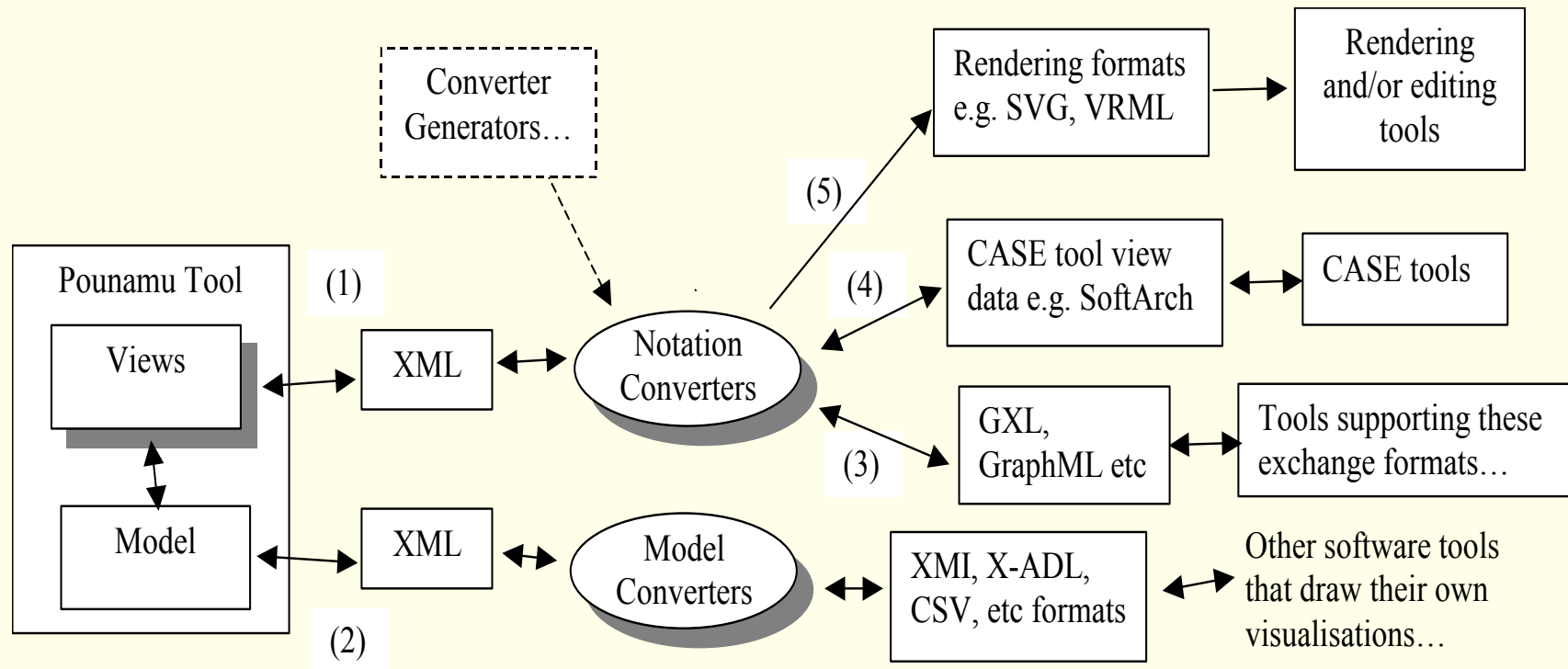
- ❖ Adaptive user interfaces for tools:
  - Web browsers
  - Mobile devices
  - Sketching-based interfaces
- ❖ AOCE for web services
- ❖ Collaboration “agents”/components
- ❖ Visual languages/tools for integration
- ❖ DSLs for Data and notation translations

# Adaptive UIs

IwC 2002  
HCC 2003  
MUI 2003  
IV 2003



# How achieved





# Extensible Software Tools

IST 2000  
S-P&E 2002  
ASWEC 2005

The screenshot displays the jEdit IDE interface with several windows open:

- Registry Browser (1):** A window for browsing registry objects. It shows a search for 'CodeLogic' and a table of results.
 

Object Type	Key	Name	Description
Organization	1740c573-7417-4...	CodeLogic	Java Tools Devel...
- Organization: CodeLogic (2):** A detailed view of the registry object, showing organization information, primary contact information (Shona Chin), and classification details.
- Service Bindings:** A window showing available services for the selected organization, including CVS, CodeLint, and JRefactor.
- Tools Integration with Web Services:** A dialog box showing a list of available web services (JCodeLint, JRefactor, JEditCVS, JICQChat) and a detailed description of CodeLint.
 

CodeLint is a Web Service for JEdit that can identify syntax & semantic errors in your Java & C/C++ source code & Class files. CodeLint can detect numerous bugs/errors/inconsistencies that are not detected by your compiler. It can identify casual errors such as wrong hex/octal/char value to wrong assumption about operator precedence, finalize method not calling super.finalize to Critical Errors/Bugs like possible deadlock conditions in your code & a lot more.

# Web Services-extended AOCE

NCWS 2003  
ASAW 2004

Search for flights - Microsoft Internet Explorer

Address: https://hemus/software306/p2/Project/FlightWebApplication2/Flights/CustomerP

Welcome

Search for Flights

Airlines: Qantas Airways-QANZ

Departure: Auckland

Destination: Dunedin

SEARCH

From: Thursday, 21 August 2003

To: Friday, 22 August

August 2003

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

August

S	M	T	W	T
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30
31				

AspectQueriesWF - Microsoft Internet Explorer

Address: http://localhost/AOUIDDI\_BUI/AspectQueriesComponent/AspectQueriesWF.aspx

## Aspect Details Queries

Web Service:

Component Name:

Aspect Type:

Aspect Name:

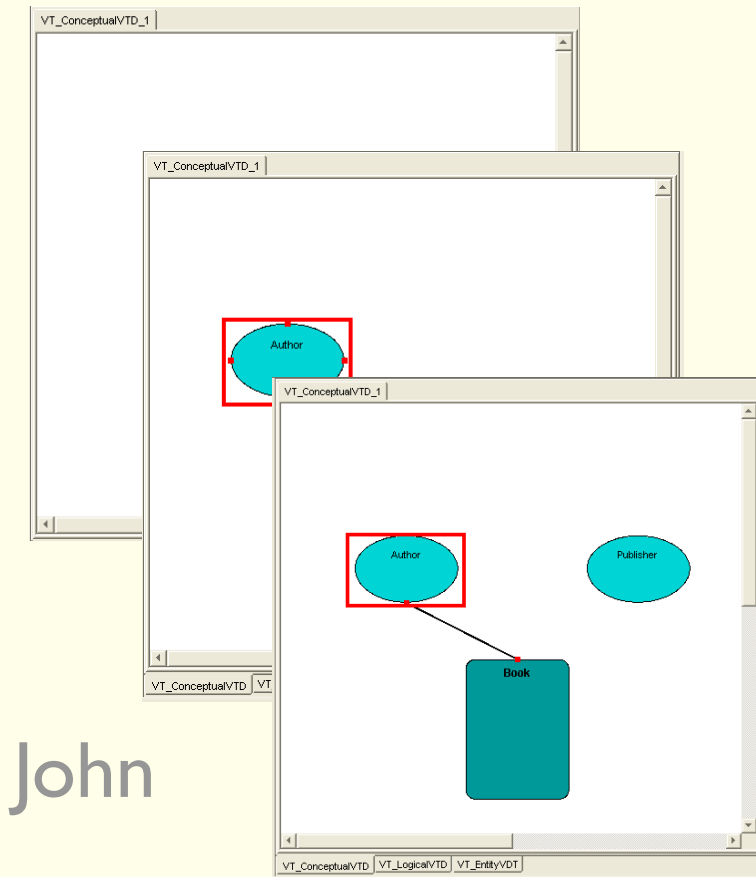
Full Aspect Details Required:

Details of aspects:

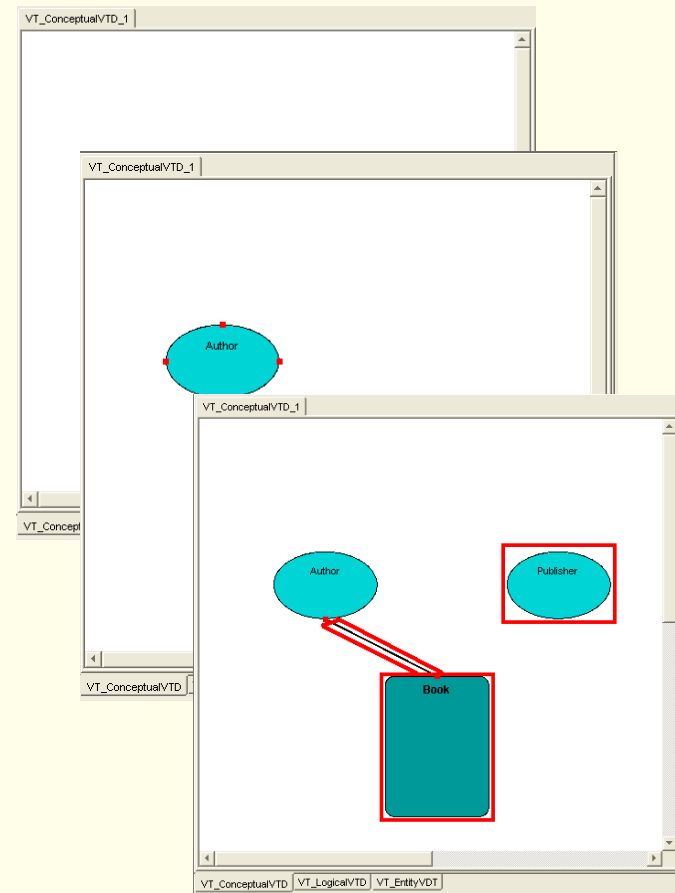
```
3=ALDetails.Length
System.String[]
numMatches=2 out of 2 http://localhost/hrwebservice
HotelsDataManagementComponent Persistence HotelsDataSetfromCityCountry
DataSet requestStringAspectDetails= data retrieval:select:true*performance:500
selects in 2.5ms:required responseStringAspectDetails= data
retrieval:select:true*performance:500 selects in 2.5ms:required
System.String[]
```

# Collaboration Components

S-P&E 2002  
WoDiSEE 04

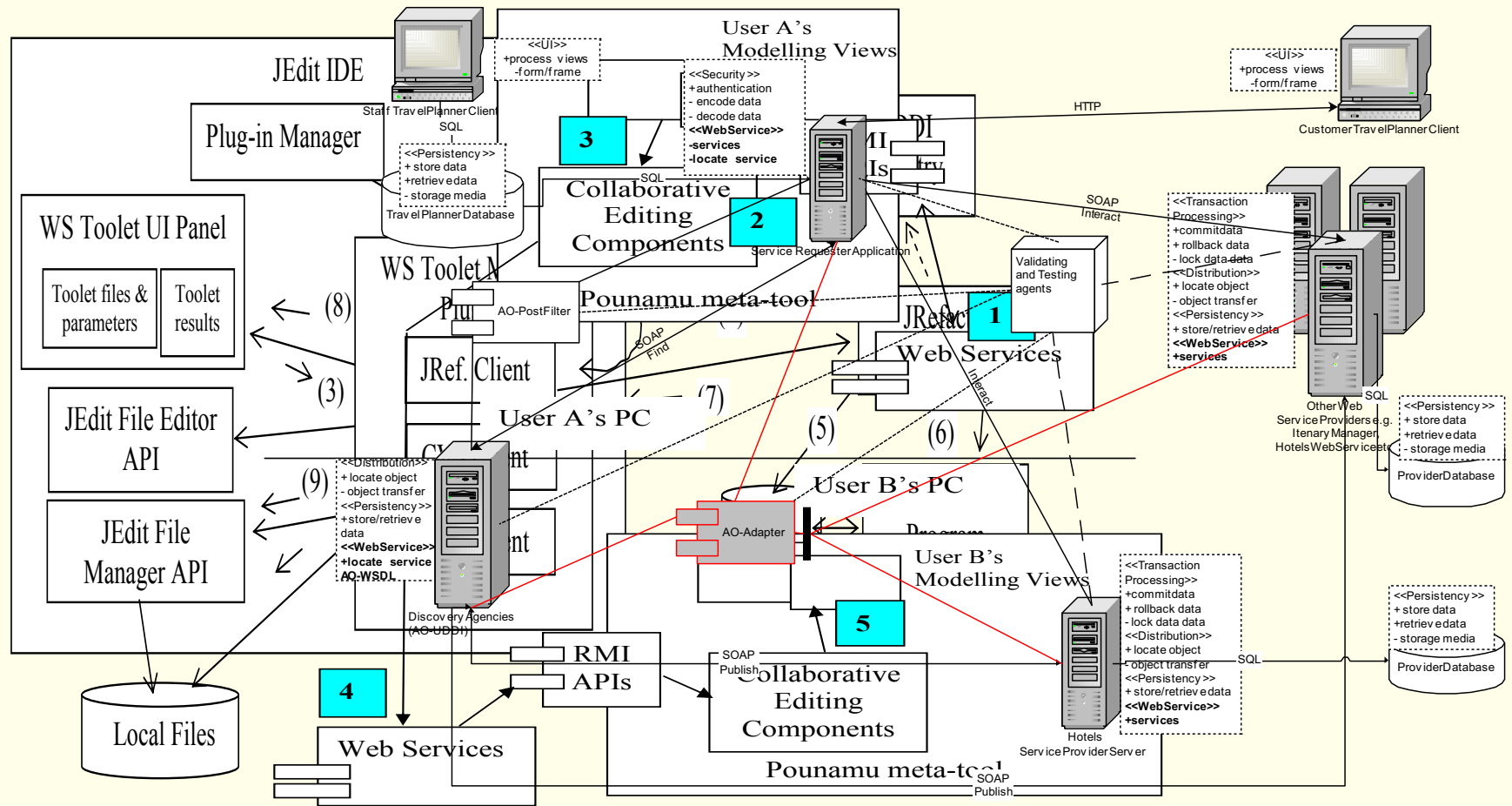


John



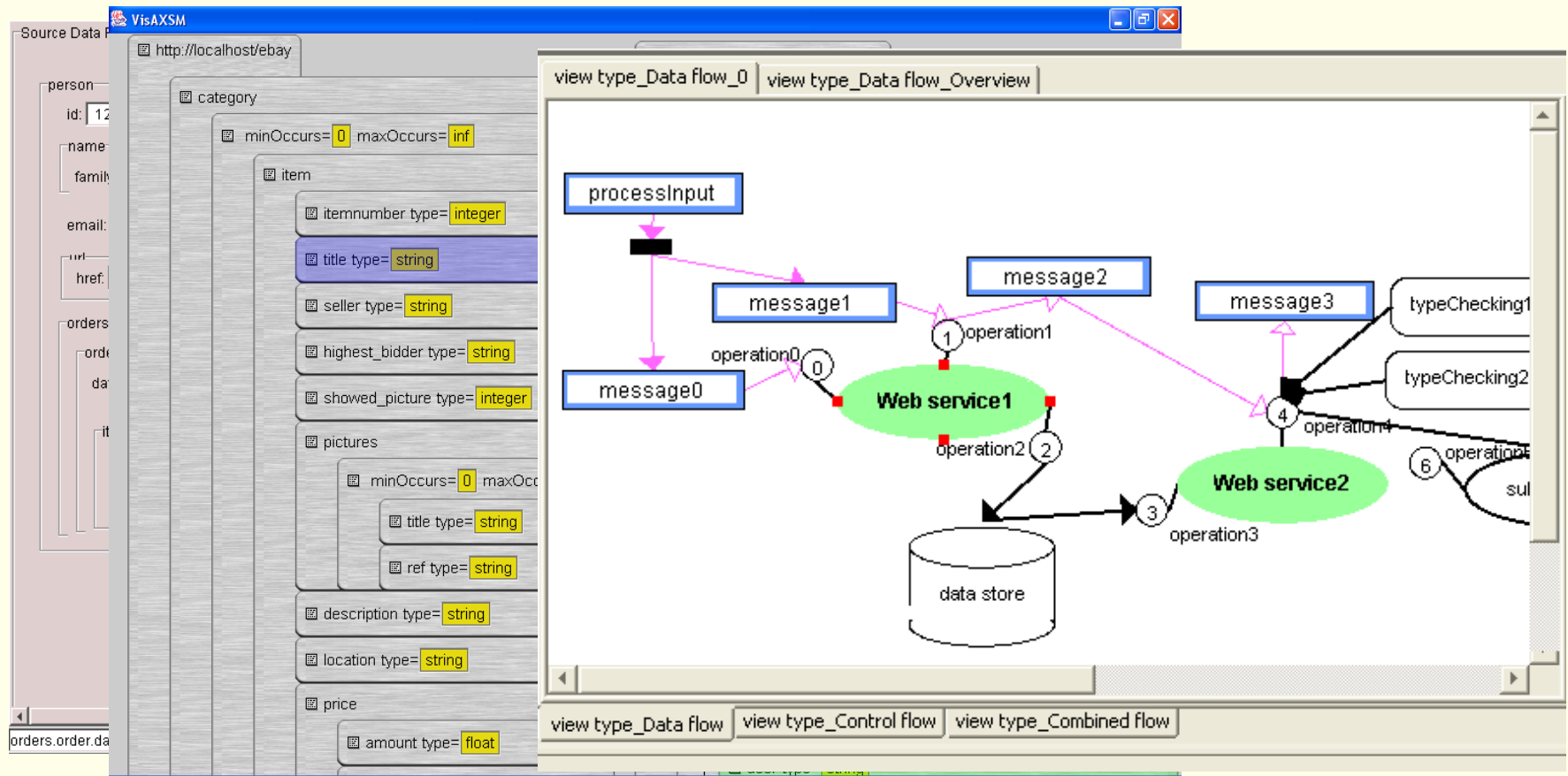
Mark

# How achieved/achieving...



# Integration Visual Languages

HCC 2002  
ASE 2004  
JVLC 2004



# Conclusions

- ❖ Automated Software Engineering = generate/adapt software from high-level models
- ❖ Our work focuses on component-based system composition, synthesis of UIs, and architectural enhancements to assist these
- ❖ Promising results to date in these areas
- ❖ Future work includes more formal specifications of components; better tools to support composition/integration/synthesis
- ❖ Commercialisation of some of this research underway