Developing Software Components with the UML, Enterprise Java Beans and Aspects

John Grundy and Rakesh Patel
University of Auckland
New Zealand
Outline

- What is a Software Component?
- What are aspects; component aspects?
- What is Aspect-oriented component engineering?

- Using AOCE+UML
- Implementing AOCE designs with EJBs
- Basic tool support

- Conclusions and future work
Software Components

- Ideas of:
  - coarser-grained components vs objects
  - compose system from reusable parts
  - dynamic composition ie extend @ run-time

- Components interact via publicised interfaces
- Components generate events
- Components have properties/methods
- Components encapsulate object(s); information
Example...

Challenges

- Example methods: Select Perspective™, COMO, Catalysis™, Aspect-oriented Component Engineering...
- Example technologies: OpenDoc, EJBs, COM+
- Issues when engineering components:
  - How to identify components vs objects?
  - How to compose components?
  - How to make “reusable”, “tailorable”, “adaptable”?
  - How to reason about composed systems (statically and dynamically)
  - Reliability, trustability, performance etc issues
  - Plus all the usual: impl meets design meets spec etc
Aspects

- Functional decomposition - normal approach
- Alternatives: parts of system contributing to “systemic” properties e.g. UI, persistency etc
- Systemic properties of system get spread...

ASWEC Presentation (c) John Grundy 2001
Component Aspects

- Component = set of methods etc
- Methods and behaviour impacted by >1 systemic aspect
- Aspects give various “perspectives” on comp. behaviour
Example

Tree Viewer

MW Communications

Products [Video]

Customers

Database Access

Reviews UI

Component

owned aspects

Provides->requires

<<Aspect>>

<<Aspect >>
Aspects + aspect details added to diagrams/documentation

Indicates where comps affected by aspects

Multiple diagrams with different aspects = different perspectives (views) on specifications & designs
Examples (from Rose)

- Stereotypes on classes, methods
- Aspect compartments
- Aspect “icons”
- Aspect details
- Aspect detail properties
- Aspect documentation (information dialogue)
- Notes
Implementing Components

- JViews (see paper)
- Perceval (see paper)
- EJBs: Java server comp model
  - Well-defined structure
  - Isolates many systemic properties
  - Still problems designing EJB components
Example

- Clients = JSPs + JavaBeans (UI aspects)
- Servers = EJBs + EJB container/server
- Designing for persistency, security, distribution
- Transactions, threads
- Aspects used to:
  - aid IF design
  - identify responsibilities
  - reason about JavaBean vs EJB vs container provides/requires
  - document/test Beans
Tool support

- JComposer = UML-based CASE tool + aspects
- Generate “inter-change” format (Perceval - XML)
- Use XSLT to generate skeleton code for EJBs, Jviews
- Further implement using JDK, Jbuilder etc
- Deploy and run
- Store for reuse in Component Repository (see ACSC’ 2000)
Conclusions & Future Work

- Engineering software components challenging:
  - Identifying components; component responsibilities
  - What does each provide/require? Constraints?
  - Reasoning about inter-component behaviour etc

- Aspects help:
  - when specifying/designing/implementing/reusing comps

- Currently working on:
  - automated testing components using EJBs/aspects
  - aspects + conventional CASE e.g. Rose
  - aspects + software architecture abstractions (SoftArch)
References