An Overview of Aspect-oriented Component Engineering

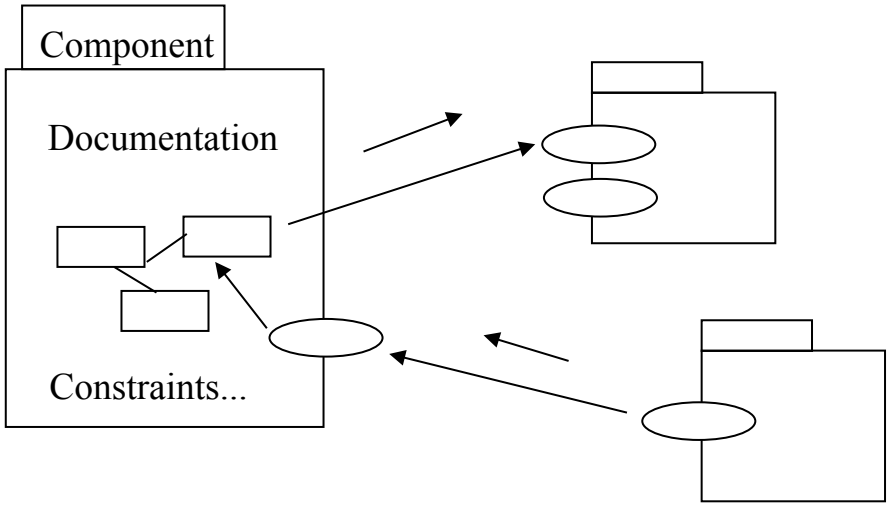
Professor John Grundy
Dept. Electrical and Computer Engineering
and Dept. Computer Science
University of Auckland, New Zealand

Outline

- What are aspects and why are they useful?
- What is AOCE?
- Requirements-level AOCE
- Design-level AOCE
- Using aspects when implementing components
- Run-time aspect usage:
  - Component/service discovery
  - Run-time integration
  - Validation of deployed components
- Tool support
- Current work
- Conclusions

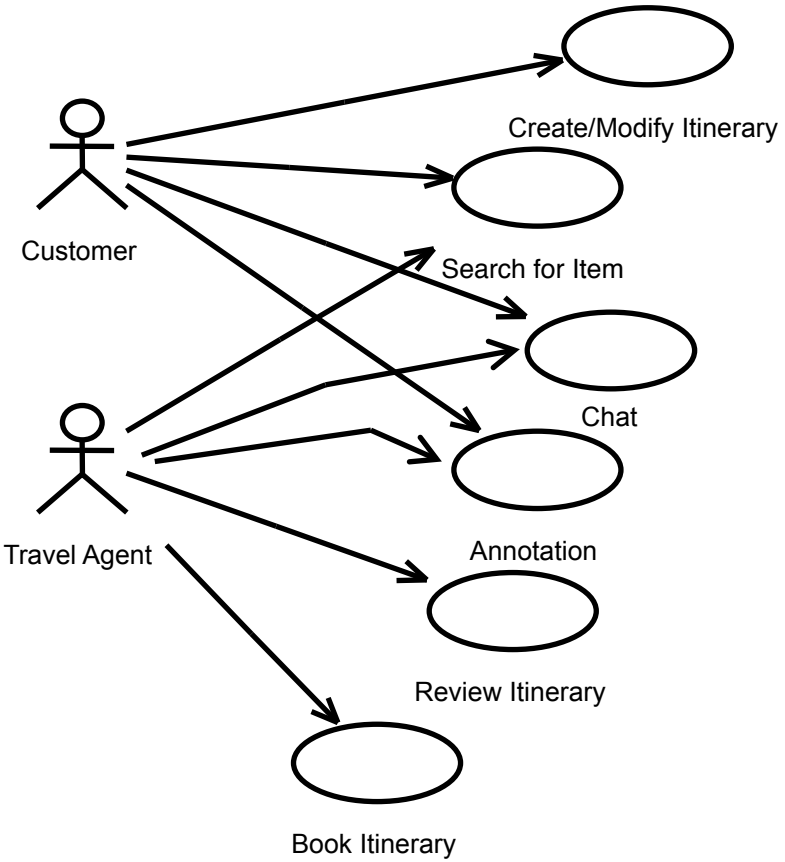
Software Components

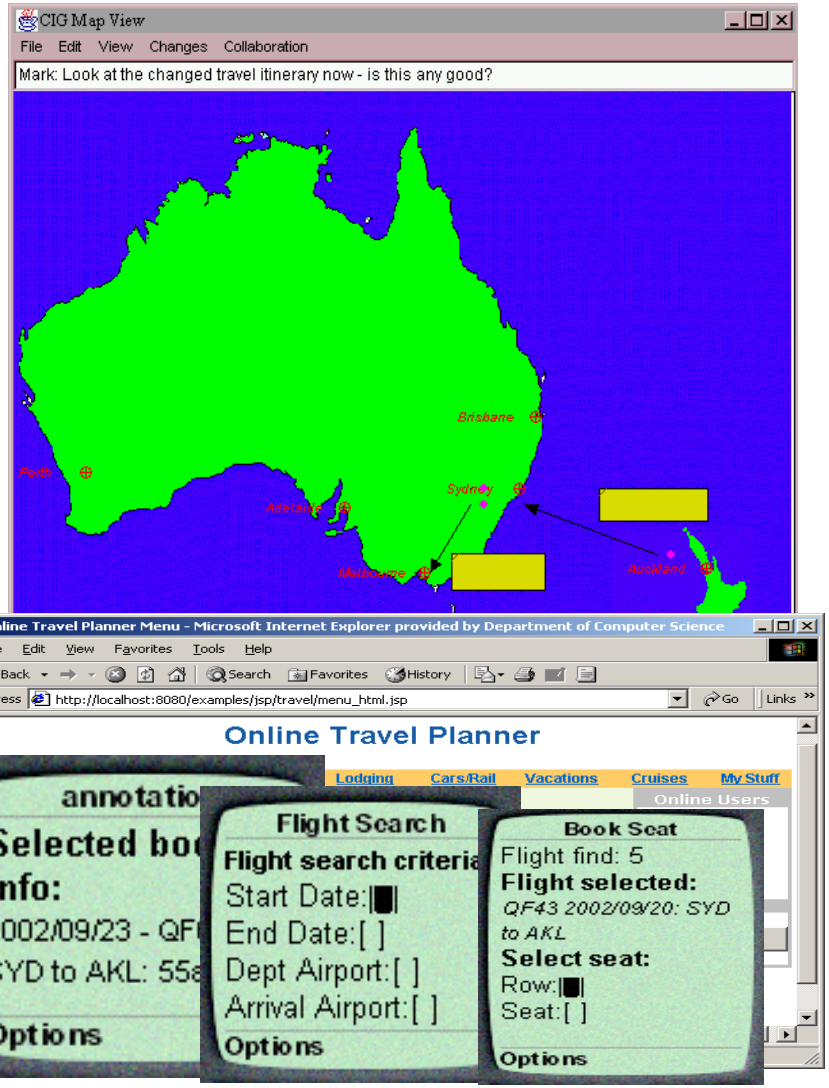
- Ideas of:
  - coarser-grained components vs objects
  - compose system from reusable parts
  - dynamic composition ie extend @ run-time



- Components interact via publicised interfaces
- Components generate events/messages
- Components have properties/methods
- Components encapsulate object(s) & information

Example...

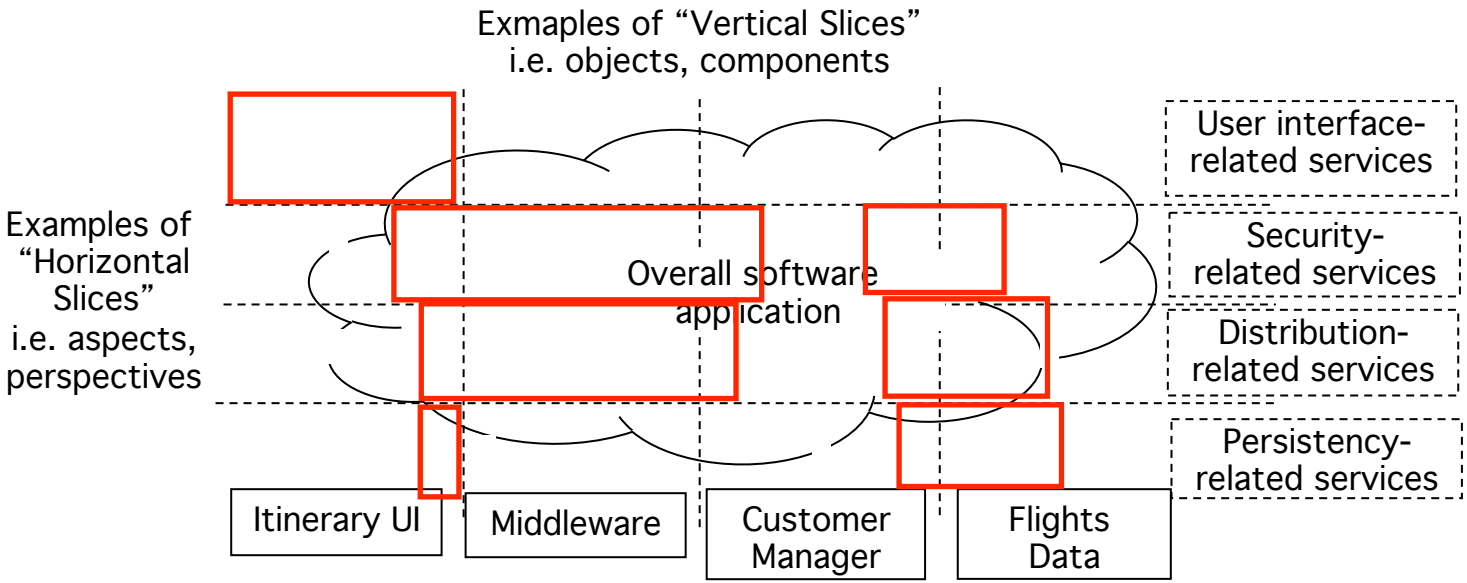

Customer
Travel Agent
Create/Modify Itinerary
Search for Item
Chat
Annotation
Review Itinerary
Book Itinerary


CIG Map View
File Edit View Changes Collaboration
Mark: Look at the changed travel itinerary now - is this any good?
Online Travel Planner Menu - Microsoft Internet Explorer provided by Department of Computer Science
Online Travel Planner
Lodging Cars/Rail Vacations Cruises My Stuff
Online Users
Flight Search
Flight search criteria
Start Date:
End Date:[ ]
Dept Airport:[ ]
Arrival Airport:[ ]
Options
Book Seat
Flight find: 5
Flight selected:
QF43 2002/09/20: SYD to AKL
Select seat:
Row:
Seat:[ ]
Options

Challenges

- 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
- We think the concept of "aspects" (cross-cutting concerns) can help…

Aspects



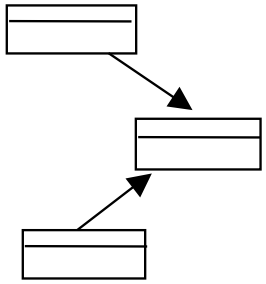
- Functional decomposition - normal approach
- Alternatives: parts of system contributing to "systemic" properties e.g. UI, persistency etc
- Systemic properties of system get spread…

Examples of Aspects

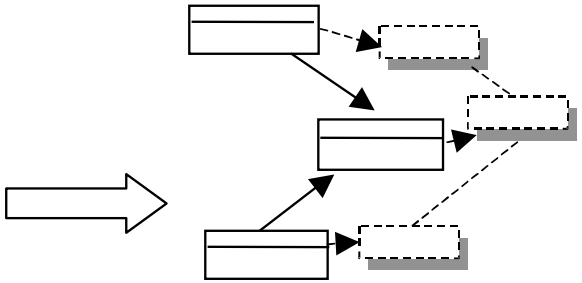
- Distribution/Remoting (networking etc)
- Persistency (data storage)
- Security (authentication, encryption, access control)
- Transactional behaviour (ACID, distributed)
- Logging
- Monitoring
- Failure recovery
- Caching
- User interfaces
- Collaboration support
- Reconfiguration support

- Key idea: these cross-cut many of the various components/ component methods in the system – how do we best handle this in requirements/design/implementation/run-time…
- How can we change the way these are handled, even @ run-time?

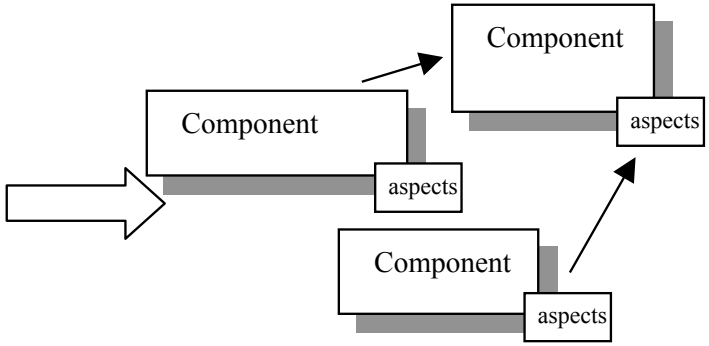
Aspect-oriented Component Engineering



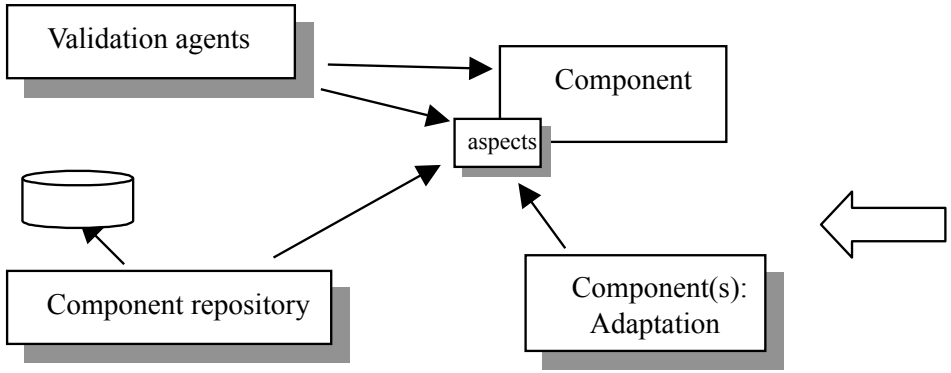
Component Specification and Design



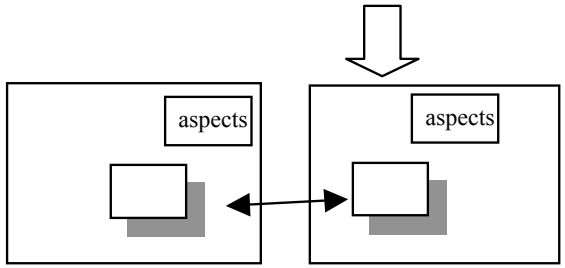
1. Extended component models with aspect information (provided/required aspect details and properties)



2. Component implementations: aspect information provide s description and way to implement reflective, de- coupled interaction



4. Component aspect encodings are accessed and u sed at run-time



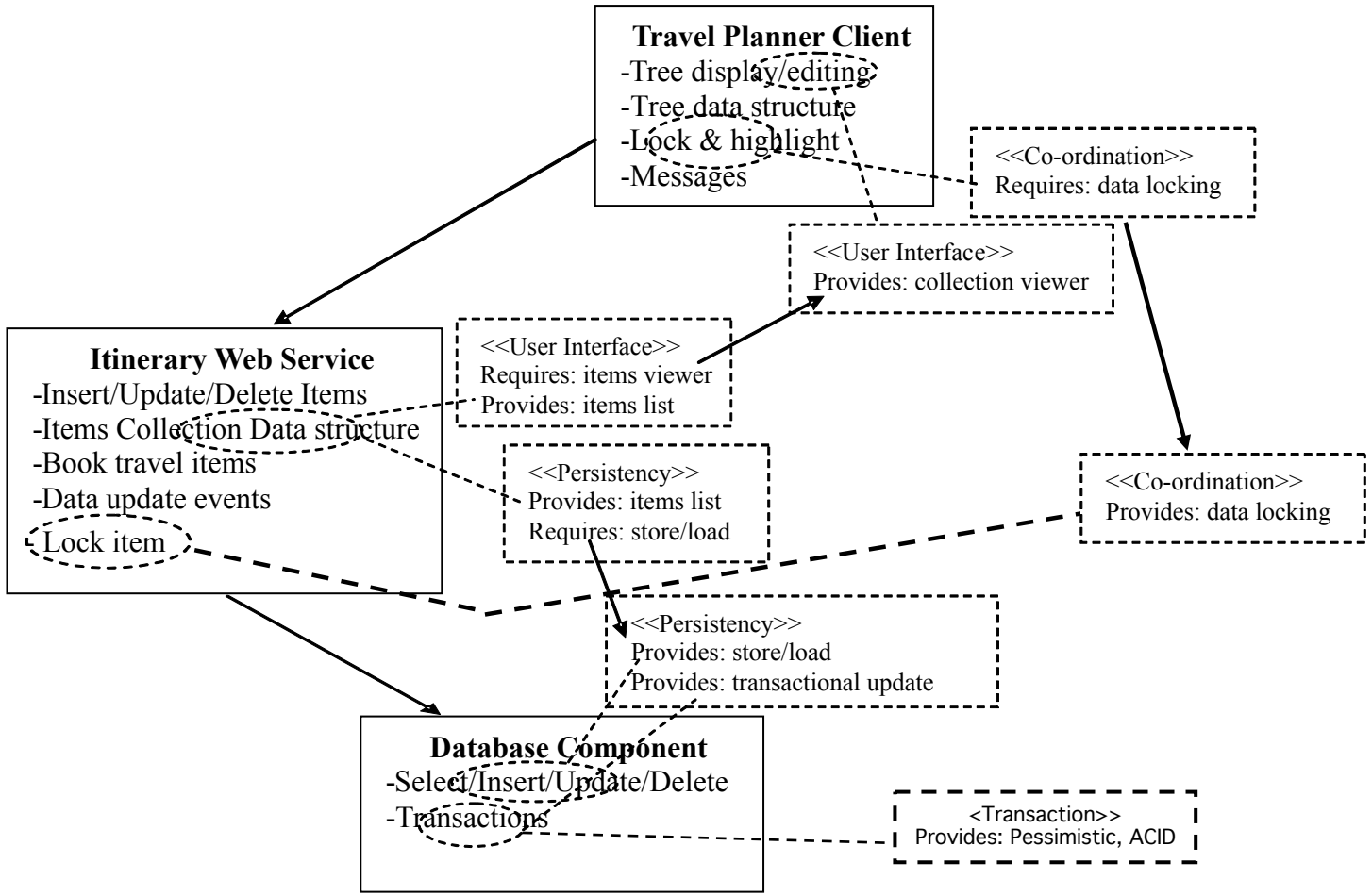
3. Components are deployed with their associated encoded aspect information

I' ll describe and illustrate SOME of these (briefly) as we go... ☺

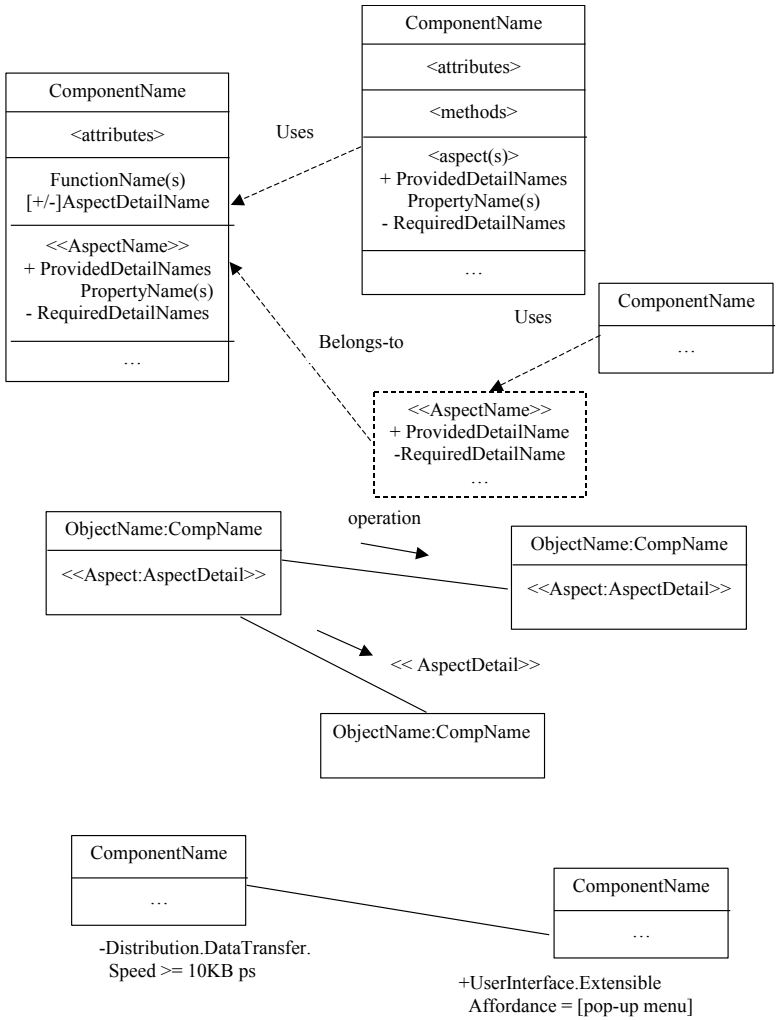
Requirements-level Aspects

- Our first foray in AOCE was to try and improve representation of these cross-cutting issues in requirements and specifications for components
- Approach taken was for each **candidate** component capture info about **possible** cross-cutting issues
- Represent these as "component aspects" that are orthogonal to requirements and specifications artefacts
- Reason about inter-related component aspects ("provided and required")
- Use to refine to design-level aspects for component implementation

An example (web services engineering using AOCE)

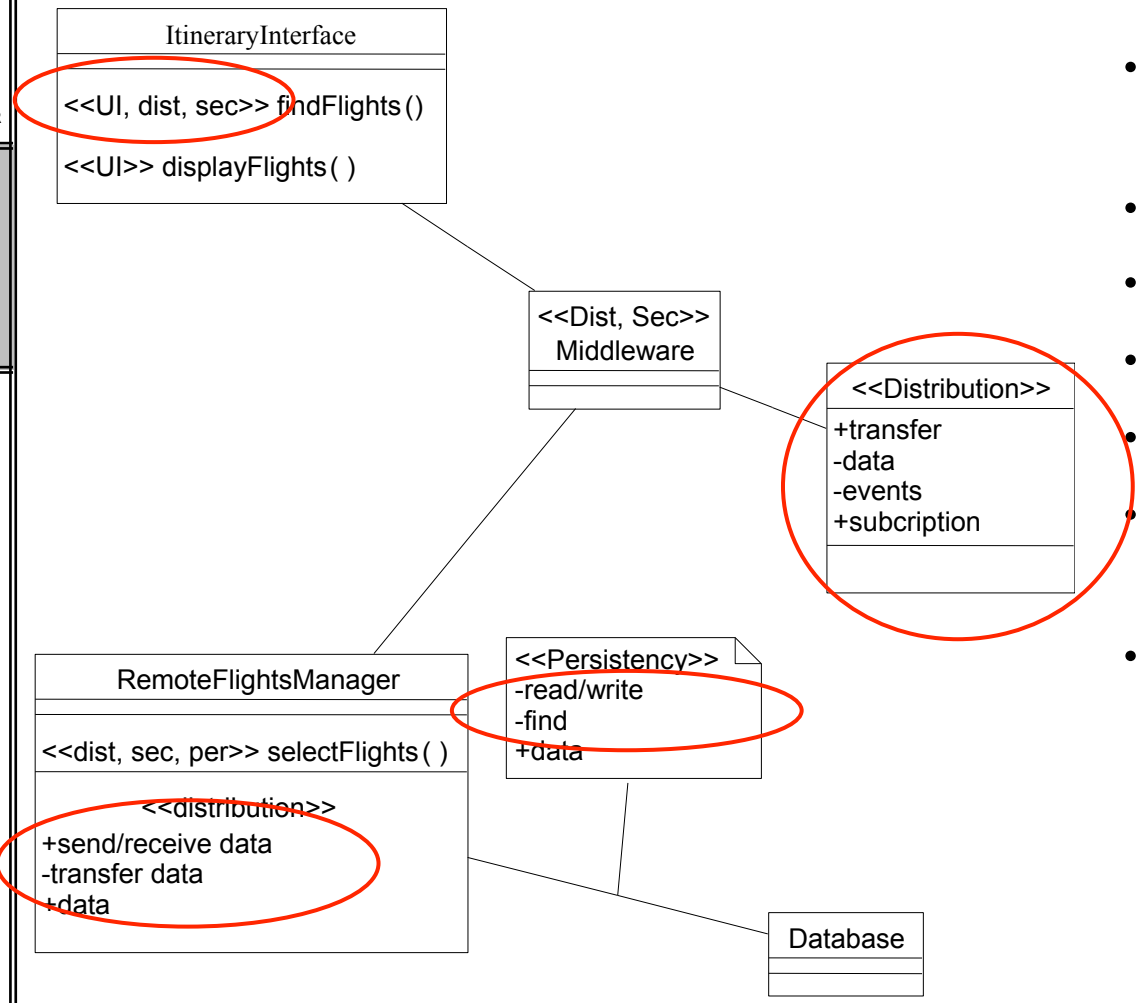


UML Extensions to Capture Component Aspects



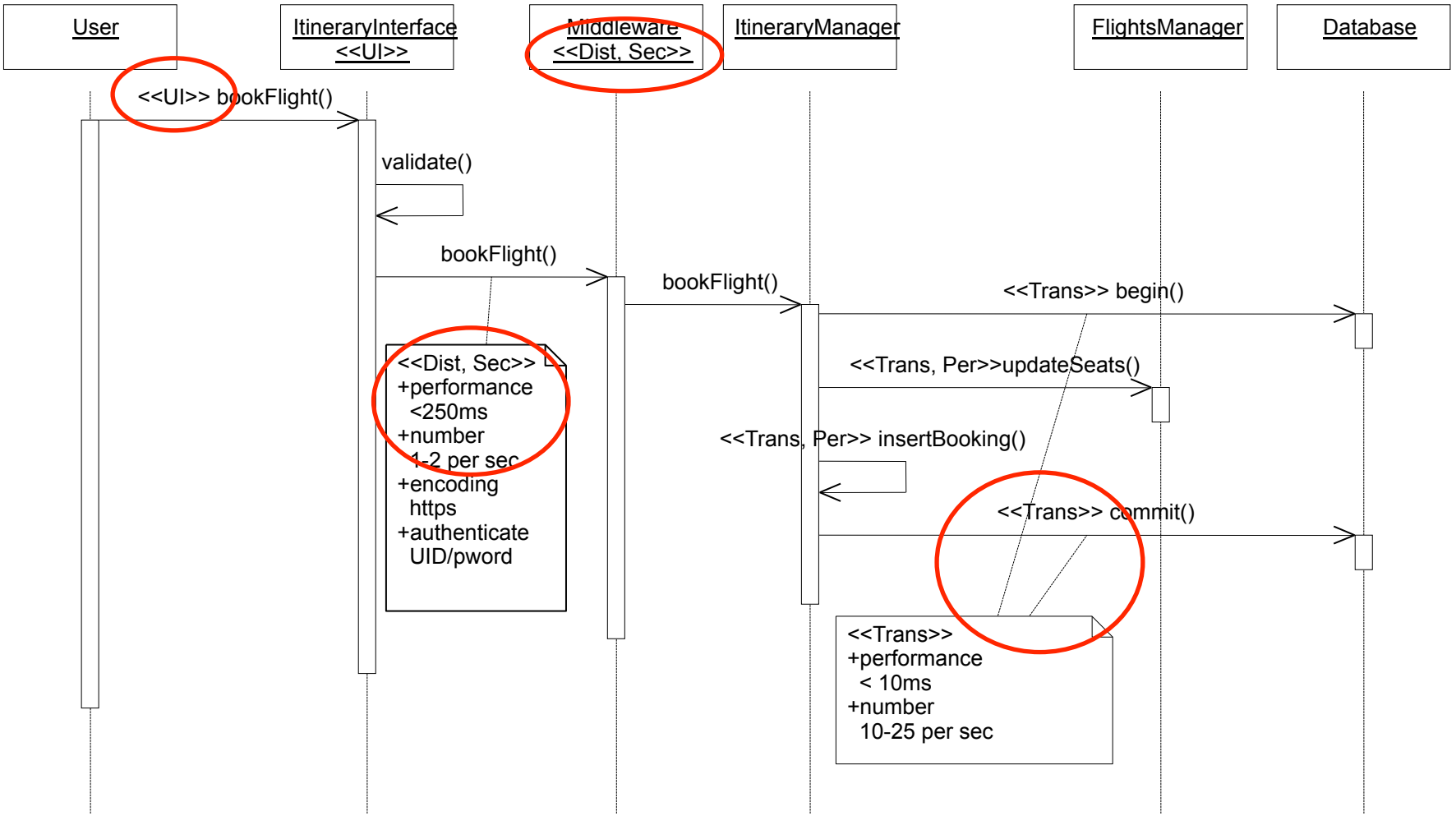
- Aspects + aspect details added to diagrams/ documentation
- Indicates where comps affected by aspects
- Multiple diagrams with different aspects = different **perspectives** (views) on specifications & designs

Aspects in Design



- Stereotypes on classes, methods
- Aspect compartments
- Aspect "icons"
- Aspect details
- Aspect detail properties
- Aspect documentation (information dialogue)
- Notes

Sequence Diagram Example



Describing Aspects - Example

```
<component name= "Itinerary Management">
    <services name="" /> <!-- no web services implementing this component />
    <components name="" />
    <property name="caching">
        <value type="boolean" />
        <getter operation="getCaching" />
        <setter operation="setCaching" />
    </property>
    …
    <operation name="findItinerary" style="rpc">
        <arg name="ID" style="in" type="LongInt" />
        <arg name="itinerary" style="out" type="itinerary:ItineraryData" />
    </operation>
    …
    <aspects namespaces="www.travelplanner.com/aspects/namespaces/itinerary" >
        <aspect name="ItineraryData"
            detail="itinerary:ItineraryDataManagement" type="provided" >
            <impacts operations="all" />
        </aspect>
        <aspect name="Persistency"
            detail="common:DataManager" type="required" >
            <impacts operations="findItineraryladdItineraryl…" />
            <property name="Performance"
                type="common:OperationSpeed">
                <common:lessThan units="ms">100</lessThan>
            </property>
        </aspect>
        <aspect name="TransactionSupport"
            detail="common:TransactionsRequired" type="required" >
            <impacts operations="findItineraryladdItineraryl…" />
            <property name="TransactionScope"
                type="common: TransactionDemarcation">
                <common:transactionState>IN_TRANS</transactionState>
            </property>
        </aspect>
        <aspect name="BookingManager"
            detail="booking:TravelBookingManager" type="required" >
            <impacts operations="addItinerarylupdateItineraryl…" />
            <property name="BookingCommittalApproach"
                type="booking:BookingCommittal">
                <booking: BookingCommittal value="BTP" />
            </property>
            <property name="Timeout" type="booking:TimeOut" >
                <booking:TimeOut units=days>
                    <max>5</max></booking:TimeOut>
            </property>
        </aspect>
        …
```

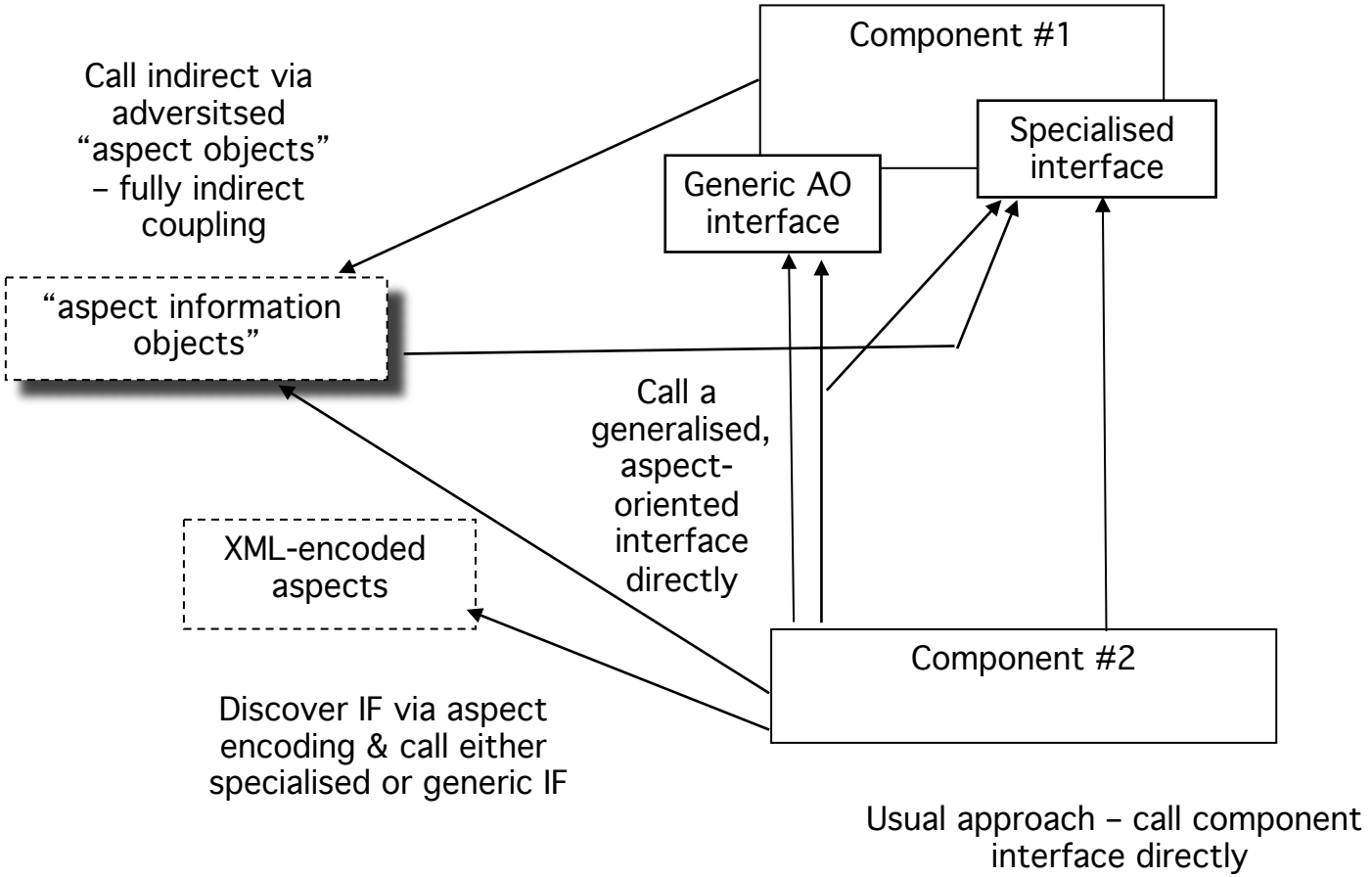
- WS component characterisation
- Low-level aspects
- Medium-level aspects
- High-level aspects

- Use in:
  - Implementing comps
  - Describing comps
  - At run-time to register/locate/ integrate/adapt/test

Component Implementation

- Need ways to use aspect information when implementing components & at run-time
- Traditional approach – inject code via AOP-style systems (Aspect-J, Hyper-J, Subject-oriented programming etc)
- However, may have COTS components (with no code); very difficult to control feature (aspect) interactions…
- Our approach – use aspects to assist in de-coupling component dependencies (can inject code, but we don't focus on that)
- Idea: aspects provide "generic" interface to accessing systemic properties of components
- Realised this via:
  - extensions to our JViews framework for building multi-view, multi-user design tools
  - extensions to EJB model in J2EE-based implementations
  - extensions to WSDL for .NET/J2EE web services systems

JViews Framework Example



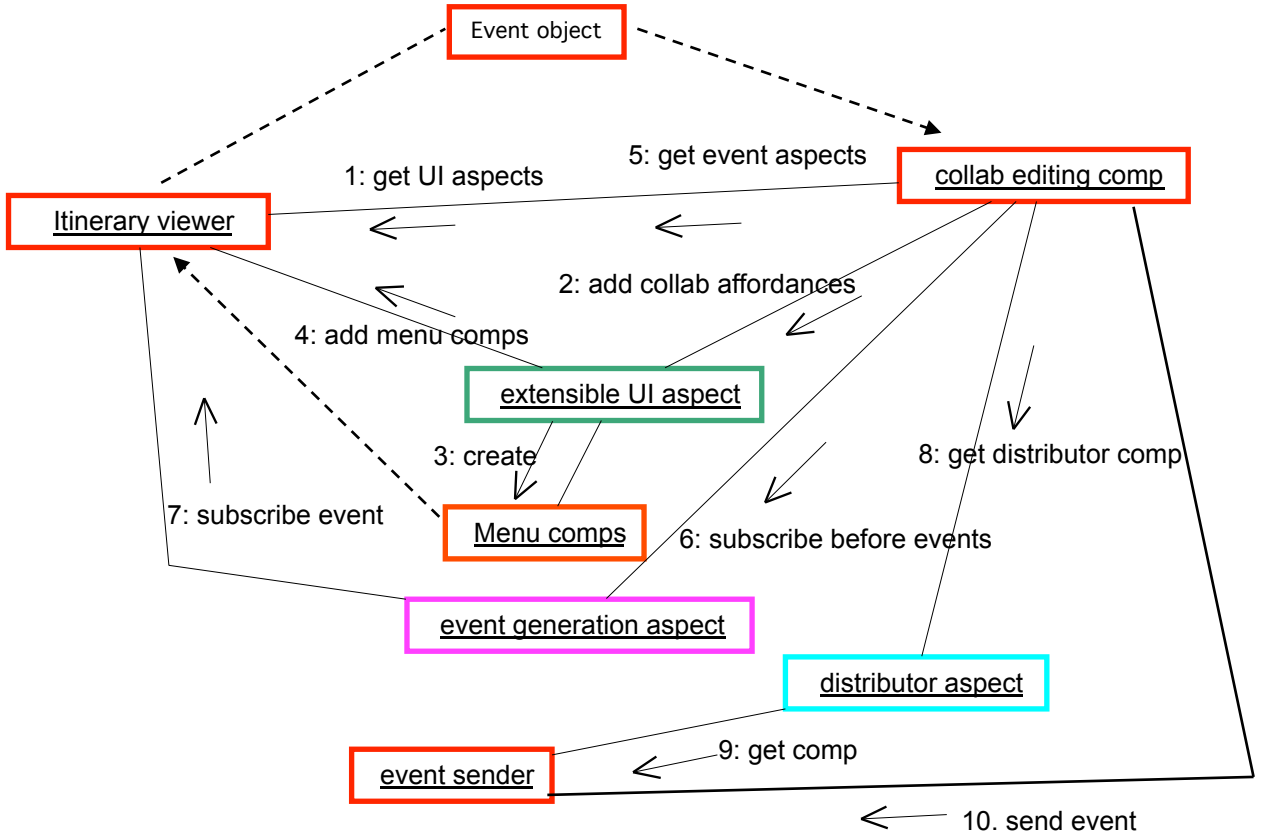
Example

Component/ object

<<Event>> aspect object

<<UI>> aspect object

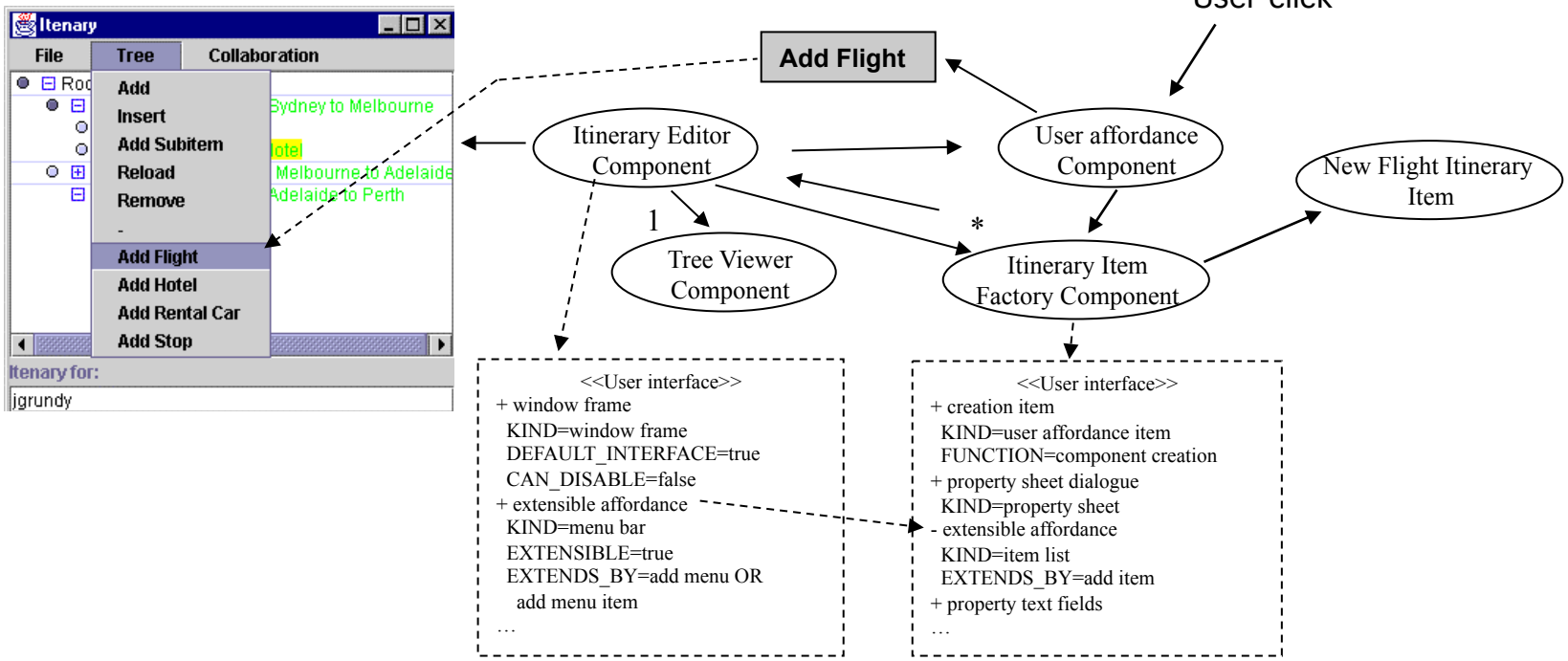
<<Remoting>> aspect object



Using Aspects at Run-time

- A number of ways of using aspects in AOCE-engineered components at run-time:
  - Add components to repository and locate via aspect information (query by cross-cut)
  - Discover (locate) and integrate with existing components
  - Adapt existing/discovered components at run-time (discover & adapt to environment)
  - Validate deployed components (synthesise tests at run-time to check components really meet their aspect-specified constraints)

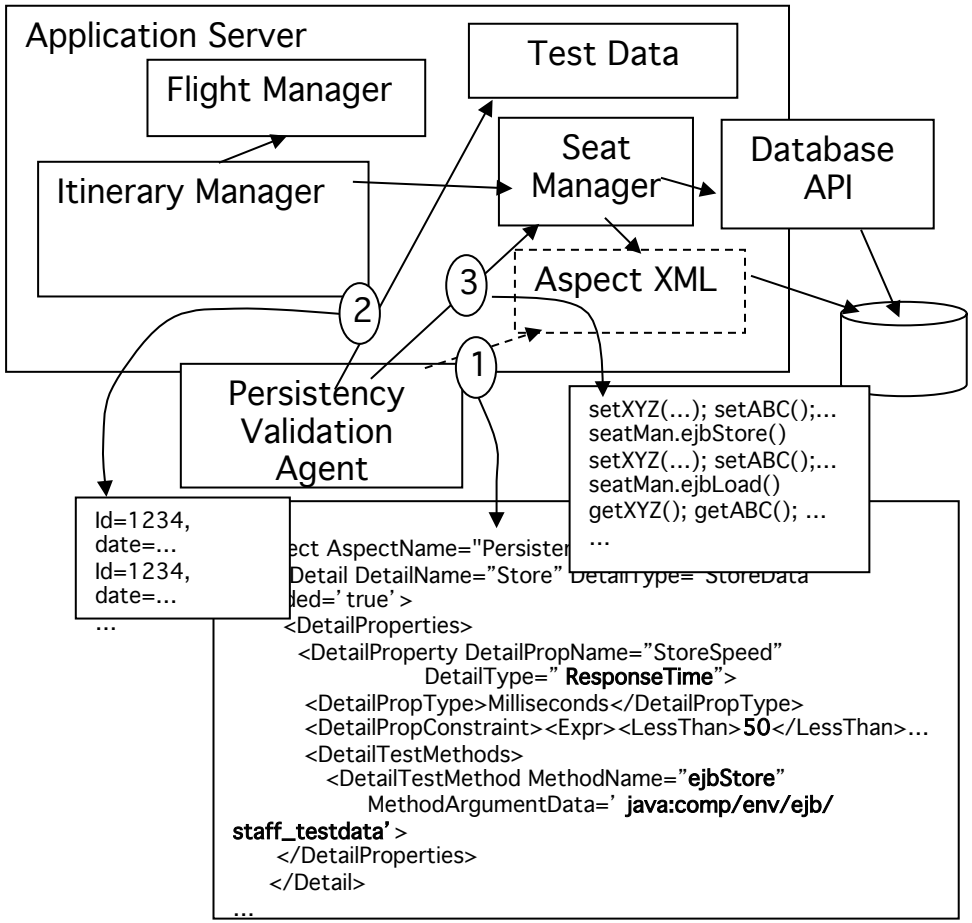
Run-time Adaptation Example



Run-time Validation Example

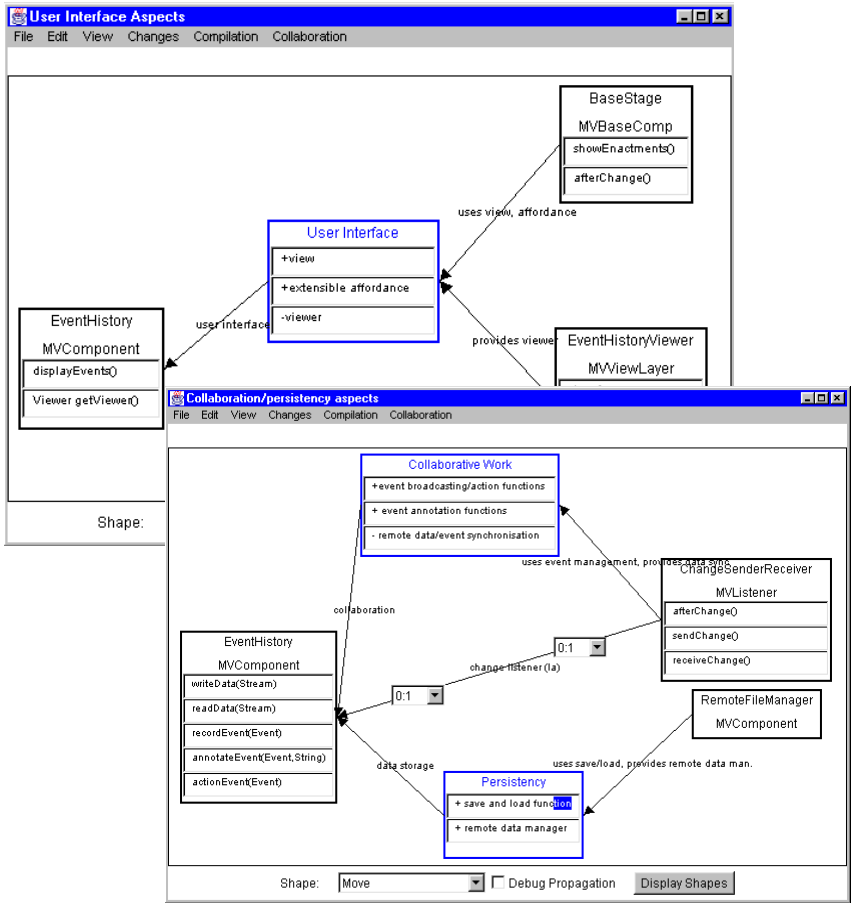
- How do we check deployed components meet their requirements?
- Our approach:
  - Characterise component behavioural/non-functional requirements with aspects
  - When deployed, inspect these characteristics
  - Synthesise tests to check these constraints have been met
- Requires more detailed information about components at design/run-time than usually present
- Built several "validation agents" – conformance check; persistency check; transaction performance check; web UI conformance/response time check; …

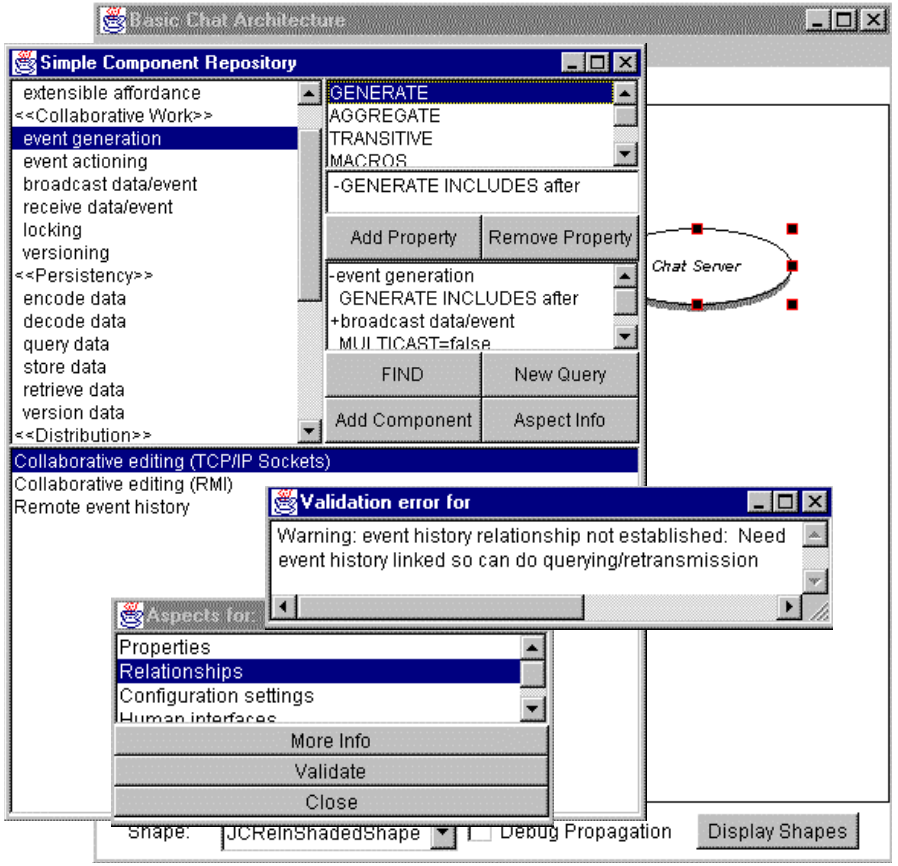
Example



- Persistency checking agent (for EJBs/DBs)
- Discovers comps and queries persistency info from their aspects (using XPath) (1)
- Gets test data (from an EJB/URL) (2)
- Synthesises tests on the deployed EJB to check persistency works (3)
- Have extended to checking TPS/ transactional behaviour/ concurrent tests…

Tools to Support AOCE





Current work

- Applying AOCE to web services engineering: AO-UDDI, AO-WSDL
- Tool support – AOWS-UML (via our Pounamu meta-tool system)
- Enriching specifications of component aspects (Alloy -> AO-WSDL), plus formal reasoning about compositions
- AOConnector abstraction for web services + aspects – de-coupled discovery, integration, composition of client/web services
- Integrating AOCE concepts into other tools (component discovery/ integration for software tools; BPEL4WS generator)
- Software architecture work + AOCE – better capture SA info
- Applying agile techniques to AOCE – eXtremeAOCE – to mitigate the "heavyweight" label
- Adaptive, multi-device UIs with AOCE techniques – another talk! ☺

Summary

- Aspects give us an orthogonal way of thinking about software component capabilities & inter-relationships – an explicit way of capturing "cross-cutting concerns"
- Our work has centred on building models of "component aspects" for requirements, design, implementation and run-time usage
- Generally AOCE seems to fit well into conventional component-based development approaches (we've used with UML, Rational Rose, J2EE and .NET, web services engineering, Wright SA specification, Perceval AOP, various implementation technologies)
- Can implement AOCE designs with or without AOP technologies
- Focus now on applying to web services/service-oriented architectures – including real deployment with industrial partners

References

Grundy, J.C. and Hosking, J.G. Developing Software Components with Aspects: Some Issues and Experiences, Chapter 25 in *Aspect-Oriented Software Development*, Prentice-Hall, October 2004, pp. 585-604.

Grundy, J.C. Multi-perspective specification, design and implementation of software components using aspects, *International Journal of Software Engineering and Knowledge Engineering,* Vol. 10, No. 6, December 2000, World Scientific Publishers, pp. 713-734.

Grundy, J.C. Aspect-oriented Requirements Engineering for Component-based Software Systems, *1999 IEEE Symposium on Requirements Engineering*, Limmerick, Ireland, 7-11 June, 1999, IEEE CS Press, pp. 84-91.

Grundy, J. and Patel, R. Developing Software Components with the UML, Enterprise Java Beans and Aspects, *2001 Australian Software Engineering Conference*, Canberra, Australia, 26-28 August 2001, IEEE CS Press, pp. 127-136.

Grundy, J.C. and Hosking, J.G. Engineering plug-in software components to support collaborative work, *Software – Practice and Experience,* vol. 32, Wiley, pp. 983-1013, 2002.

Grundy, J.C. and Hosking, J.G. Developing Adaptable User Interfaces for Component-based Systems, *Interacting with Computers*, vol. 14, no. 3, March 2002, Elsevier Science Publishers, pp 175-194.

Grundy, J.C., Ding, G., and Hosking, J.G. Deployed Software Component Testing using Dynamic Validation Agents, *Journal of Systems and Software: Special Issue on Automated Component-based Software Engineering*, vol. 74, no. 1, January 2005, Elsevier, pp. 5-14.

Singh, S., Grundy, J.C., Hosking, J.G. and Sun, J. An Architecture for Developing Aspect-Oriented Web Services, *2005 European Conference on Web Services*, Vaxjo, Sweden, Nov 14-16 2005, IEEE Press.

Singh, S. Chen, H.C. Hunter, O., Grundy, J.C. and Hosking, J.G. Improving Agile Software Development using eXtreme AOCE and Aspect-Oriented CVS, *12th Asia-Pacific Software Engineering Conference*, Taiwan, December 2005, IEEE CS Press.