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

John Grundy and Rakesh Patel

University of Auckland

New Zealand

Outline

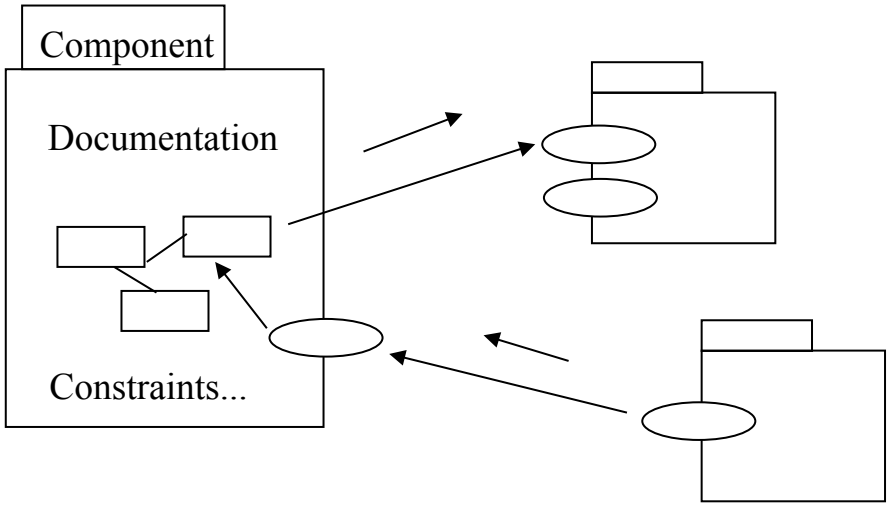
- What is a Software Component?
- What are aspects; component aspects?
- What is Aspect-oriented component engineering?

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

- Conclusions and future work

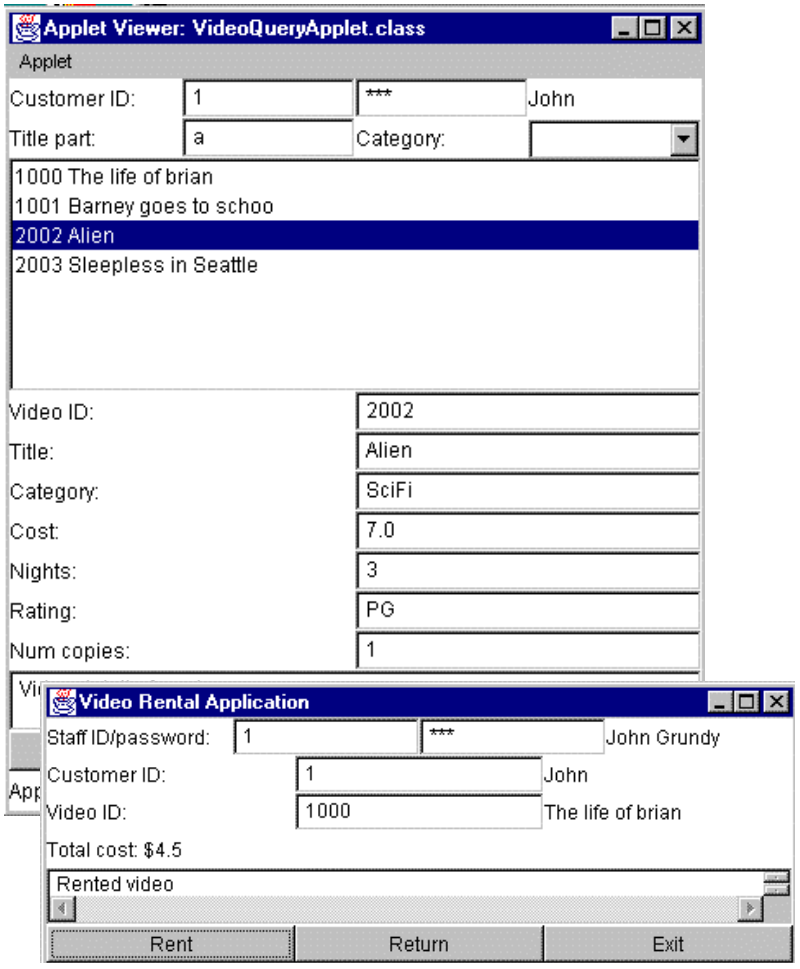
Software Components

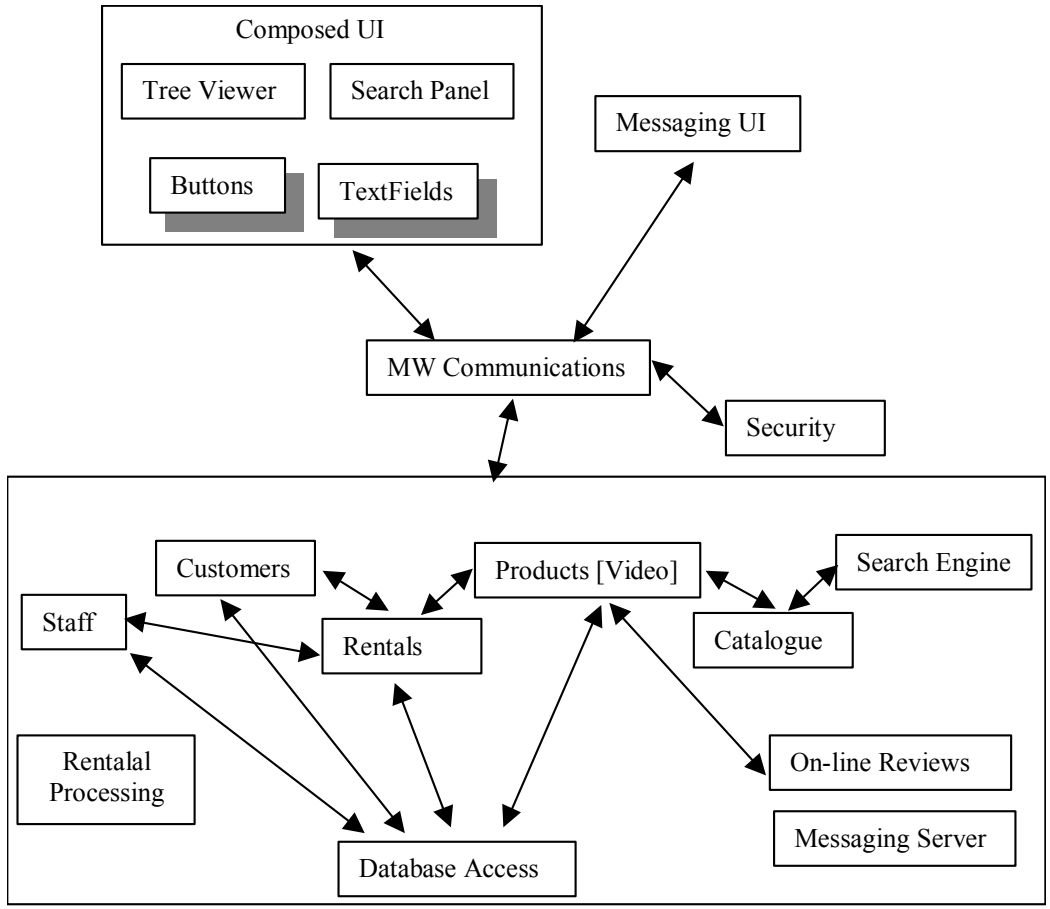
- Ideas of:
  - coarser-grained components vs objects
  - compose system from reusable parts
  - dynamic composition ie extend @ run-time



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

Example...

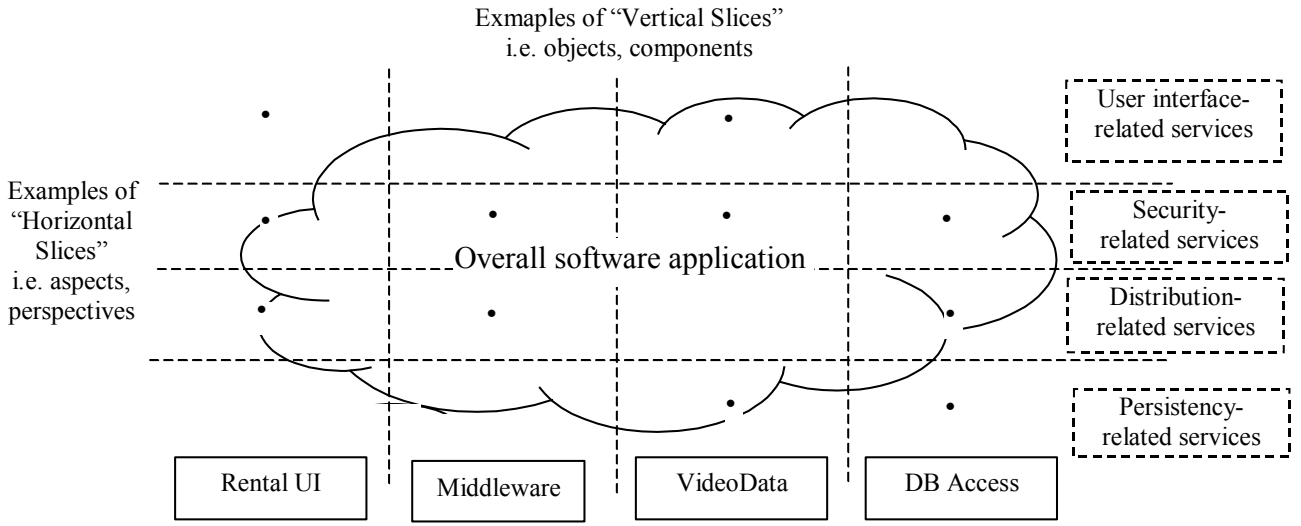

Applet Viewer: VideoQueryApplet.class
Applet
Customer ID:
1
***
John
Title part:
a
Category:
1000 The life of brian
1001 Barney goes to schoo
2002 Alien
2003 Sleepless in Seattle
Video ID:
2002
Title:
Alien
Category:
SciFi
Cost:
7.0
Nights:
3
Rating:
PG
Num copies:
1
Video Rental Application
Staff ID/password:
1
***
John Grundy
Customer ID:
1
John
Video ID:
1000
The life of brian
Total cost: $4.5
Rented video
Rent
Return
Exit


Composed UI
Tree Viewer
Search Panel
Buttons
TextFields
Messaging UI
MW Communications
Security
Customers
Products [Video]
Search Engine
Staff
Rentals
Catalogue
Rentalal Processing
On-line Reviews
Messaging Server
Database Access

Challenges

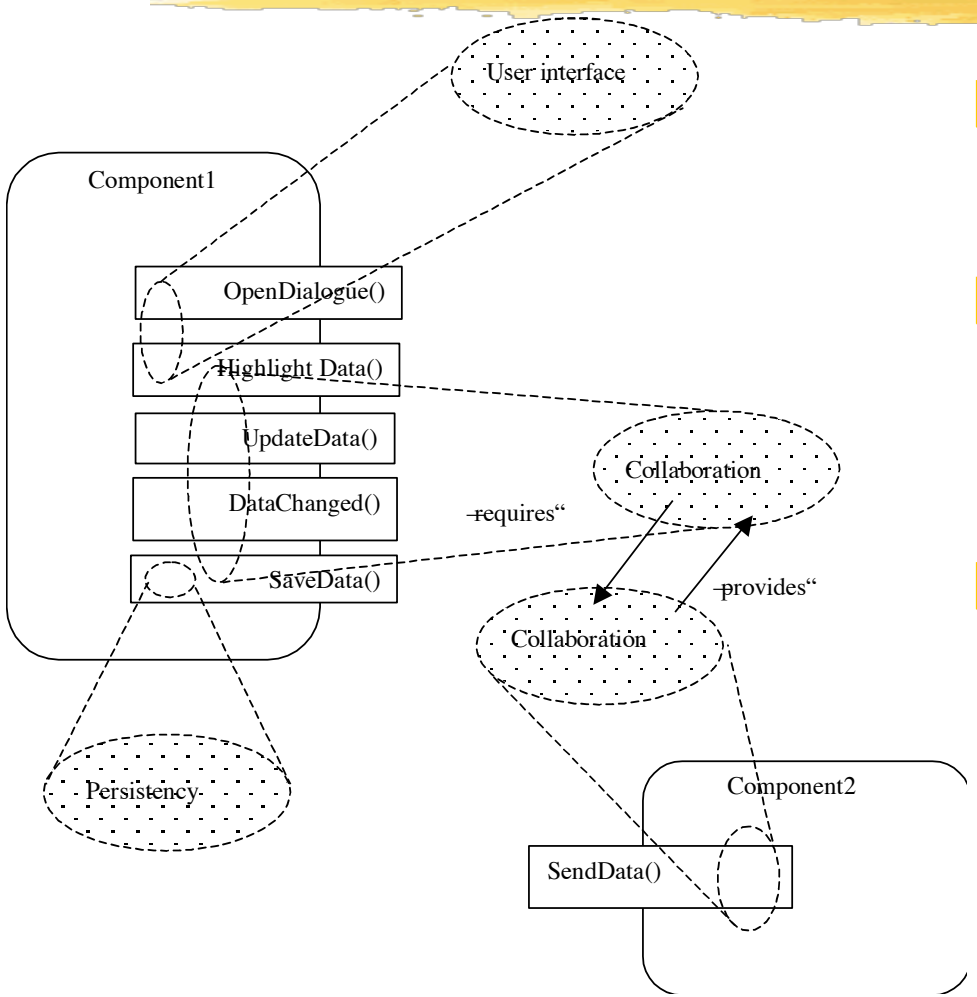
- Example methods: Select Perspective™, COMO, Catalysis™, Aspect-oriented Component Engineering…
- Example technologies: OpenDoc, EJBs, COM+
- Issues when engineering components:
  - How to identify components vs objects?
  - How to compose components?
  - How to make "reusable", "tailorable", "adaptable"?
  - 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



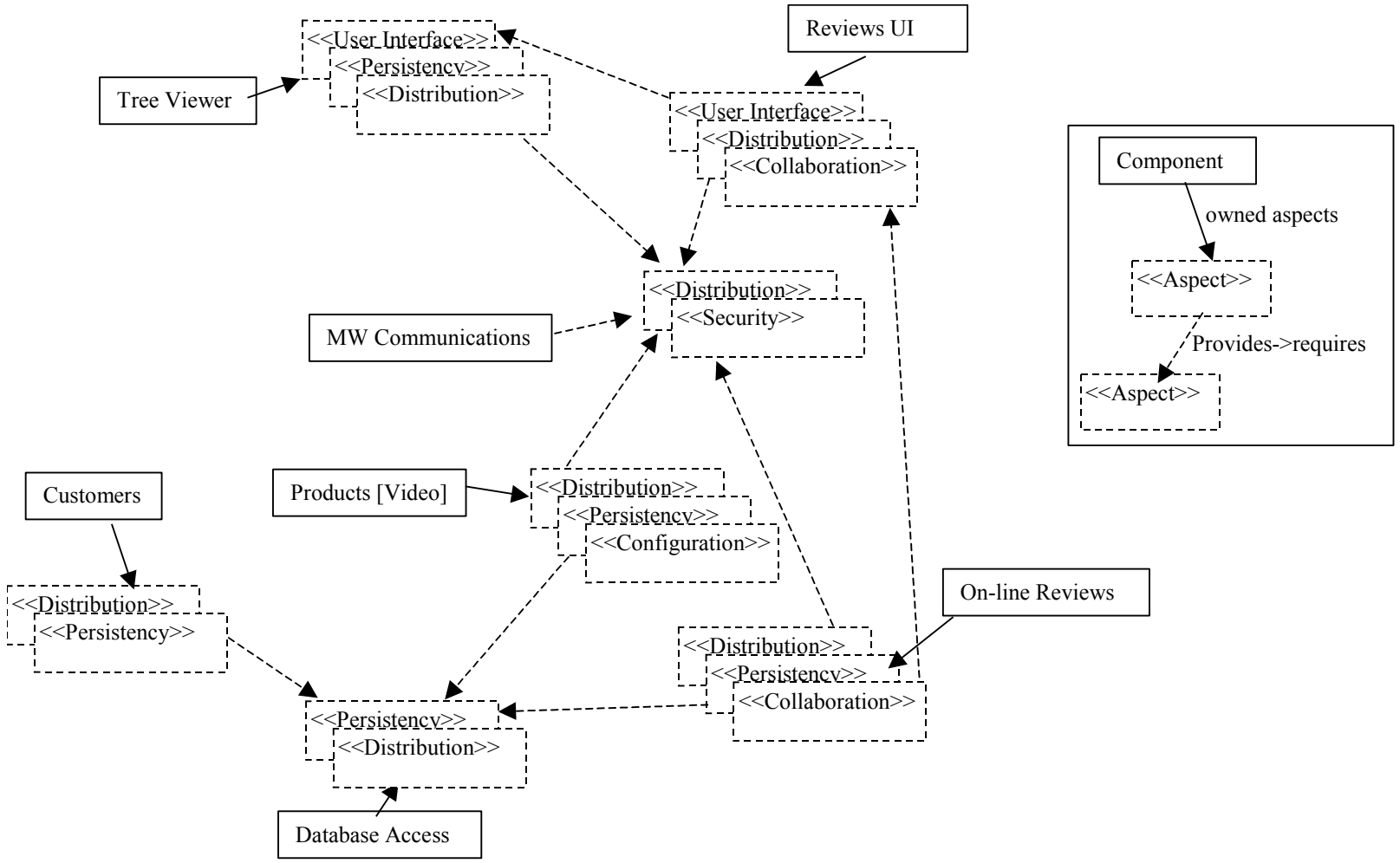
- Functional decomposition - normal approach
- Alternatives: parts of system contributing to "systemic" properties e.g. UI, persistency etc
- Systemic properties of system get spread...

Component Aspects

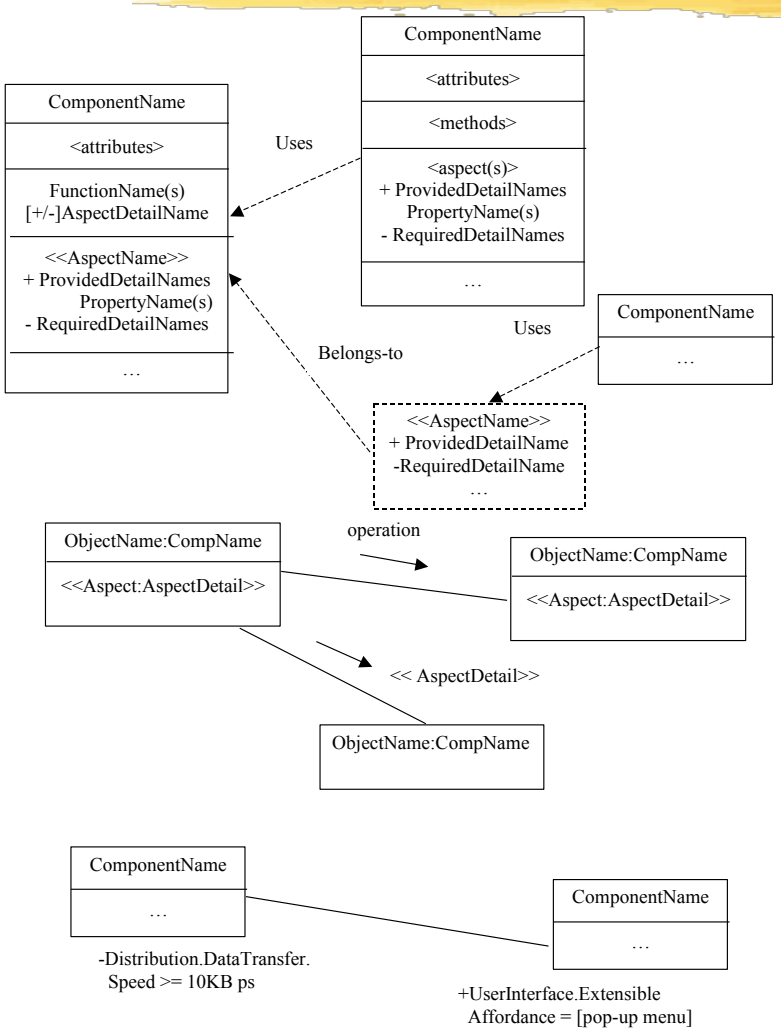


- Component = set of methods etc
- Methods and behaviour impacted by >1 systemic aspect
- Aspects give various “perspectives” on comp. behaviour

Example

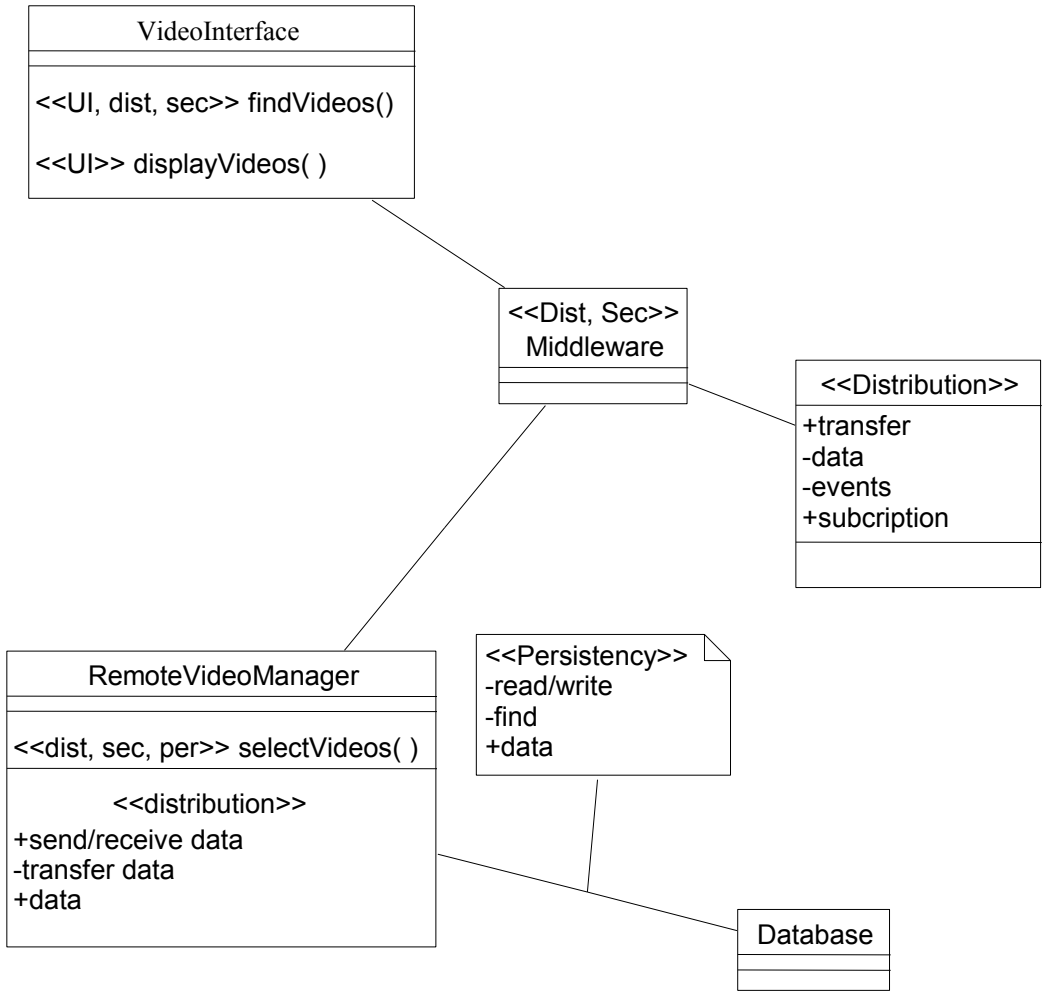

Reviews UI
<<User Interface>>
<<Persistency>>
<<Distribution>>
Tree Viewer
<<User Interface>>
<<Distribution>>
<<Collaboration>>
Component
owned aspects
<<Aspect>>
Provides->requires
<<Aspect>>
<<Distribution>>
<<Security>>
MW Communications
Customers
Products [Video]
<<Distribution>>
<<Persistency>>
<<Configuration>>
On-line Reviews
<<Distribution>>
<<Persistency>>
<<Distribution>>
<<Persistency>>
<<Collaboration>>
<<Persistency>>
<<Distribution>>
Database Access

AOCE with UML



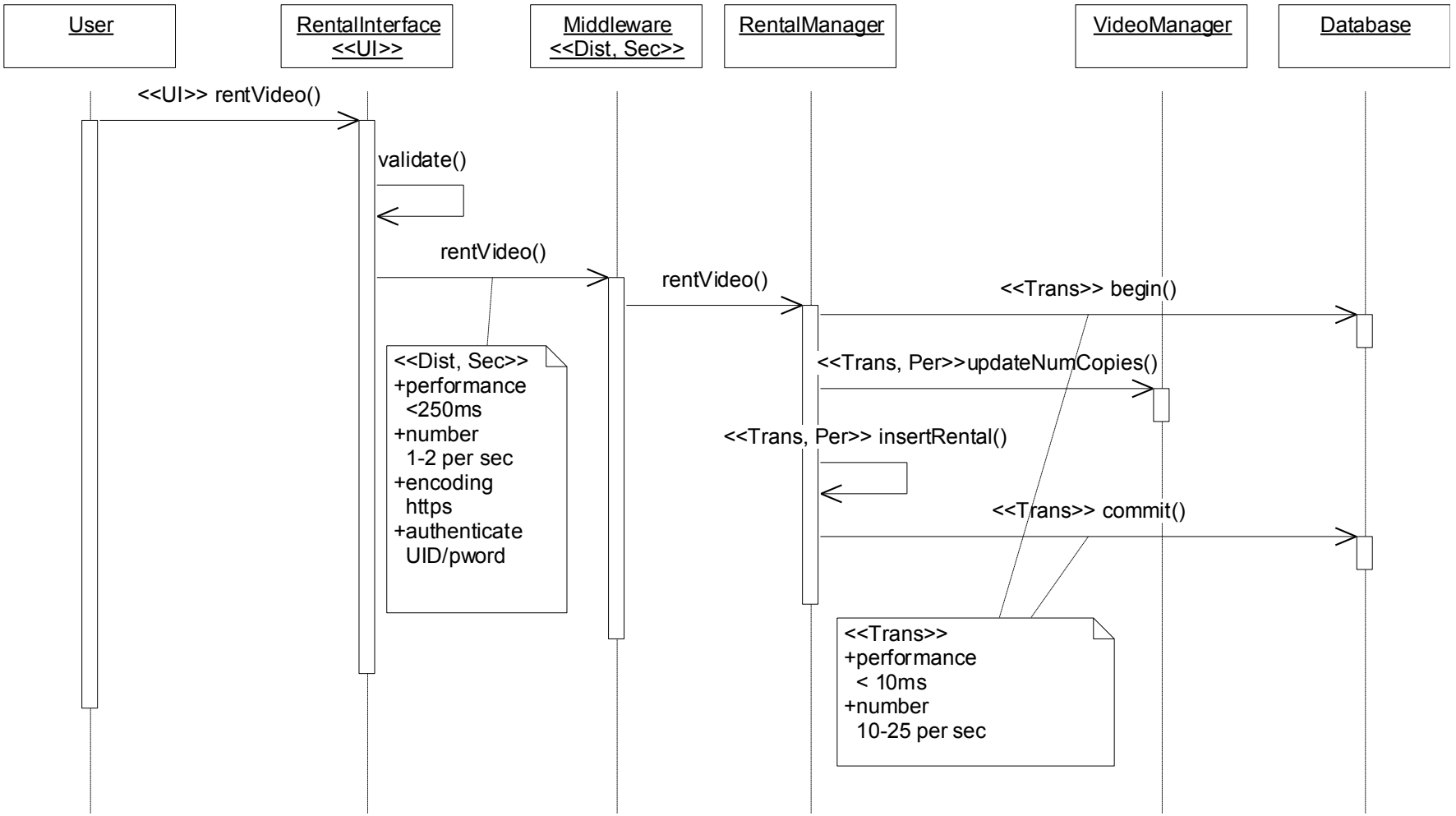
- Aspects + aspect details added to diagrams/ documentation
- Indicates where comps affected by aspects
- Multiple diagrams with different aspects = different **perspectives** (views) on specifications & designs

Examples (from Rose)

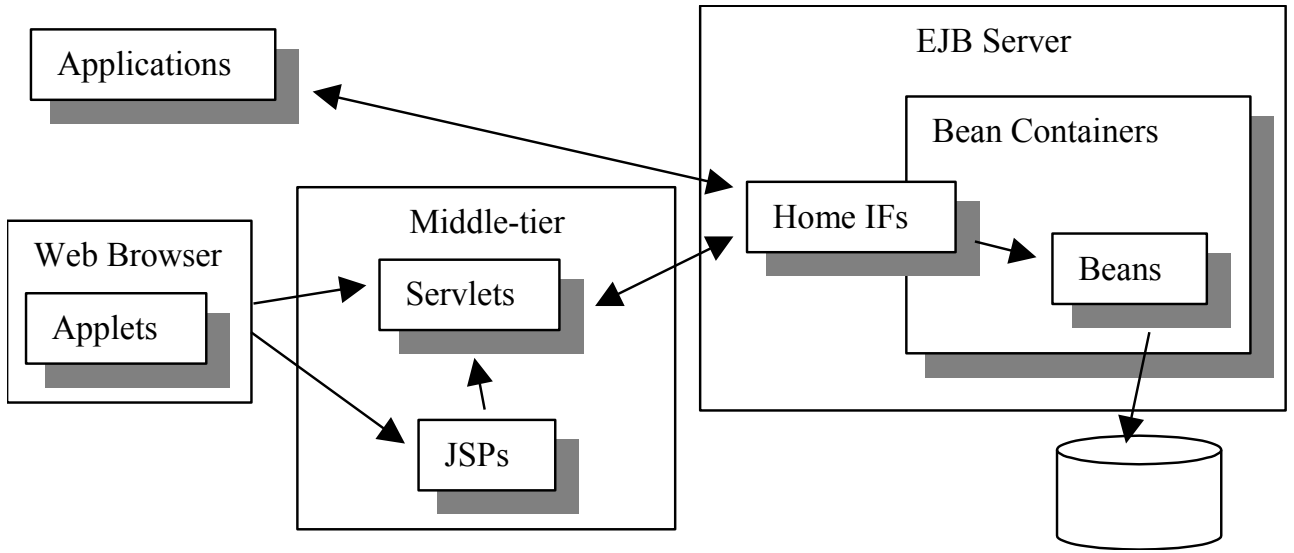


- Stereotypes on classes, methods
- Aspect compartments
- Aspect "icons"
- Aspect details
- Aspect detail properties
- Aspect documentation (information dialogue)
- Notes

Example #2

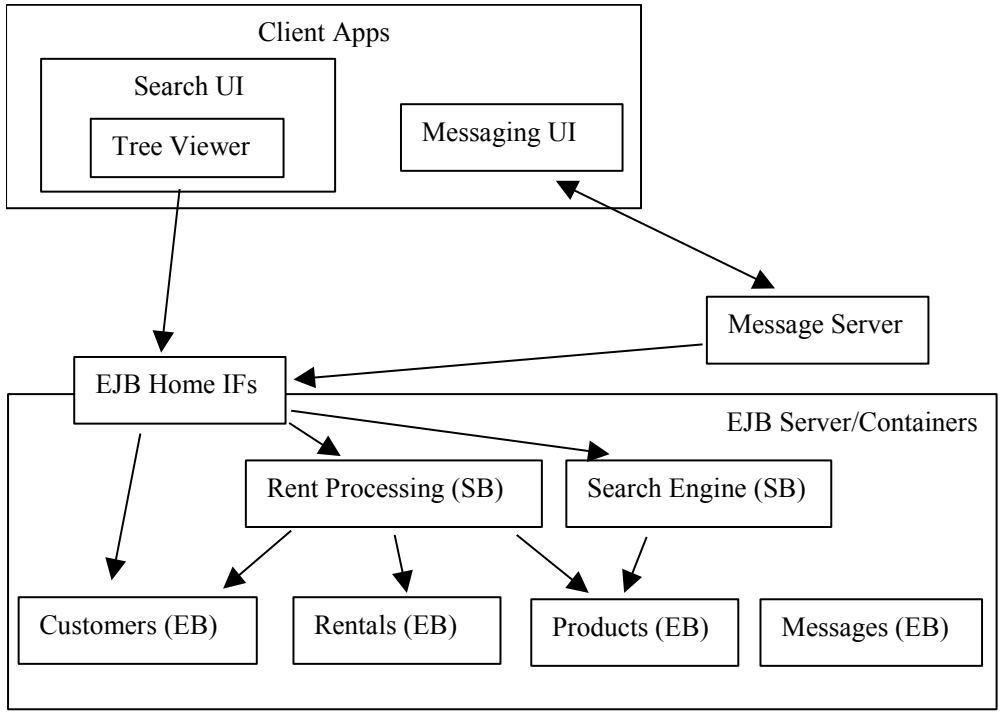

User
RentalInterface
<<UI>>
Middleware
<<Dist, Sec>>
RentalManager
VideoManager
Database
<<UI>> rentVideo()
validate()
rentVideo()
rentVideo()
<<Trans>> begin()
<<Dist, Sec>>
+performance
<250ms
+number
1-2 per sec
+encoding
https
+authenticate
UID/pword
<<Trans, Per>>updateNumCopies()
<<Trans, Per>> insertRental()
<<Trans>> commit()
<<Trans>>
+performance
< 10ms
+number
10-25 per sec

Implementing Components



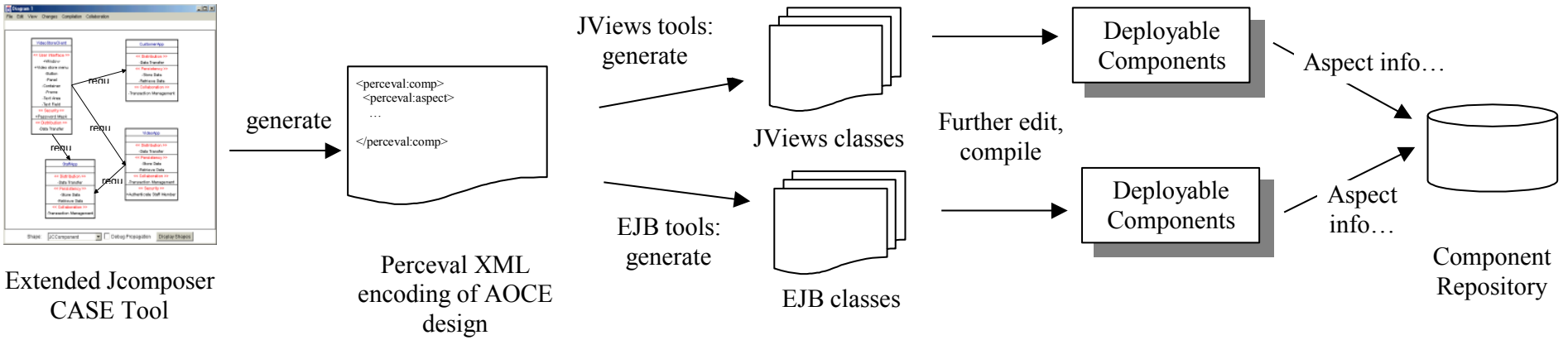
- JViews (see paper)
- Perceval (see paper)
- EJBs: Java server comp model
  - Well-defined structure
  - Isolates many systemic properties
  - Still problems designing EJB components

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

- Grundy, J. and Patel, R. Developing Software Components with the UML, Enterprise Java Beans and Aspects, In Proceedings of the 2001 Australian Software Engineering Conference, Canberra, Australia, 26-28 August 2001, IEEE CS Press.
- Grundy, J.C. Multi-perspective specification, design and implementation of components using aspects, International Journal of Software Engineering and Knowledge Engineering, Vol. 10, No. 6, December 2000, World Scientific.
- Grundy, J.C., Mugridge, W.B. and Hosking, J.G. Constructing component-based software engineering environments: issues and experiences, Information and Software Technology Vol 42, No. 2, Special Issue on Constructing Software Engineering Tools, Elsevier Science Publishers.
- 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.
- Grundy, J.C., Hosking, J.G., Mugridge, W.B. Inconsistency Management for Multi-view Software Development Environments, IEEE Transactions on Software Engineering: Special Issue on Managing Inconsistency in Software Development, Vol. 24, No. 11, 1998, IEEE CS Press.