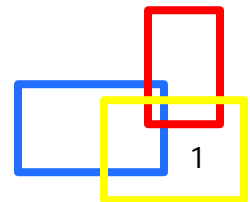




VERSANT

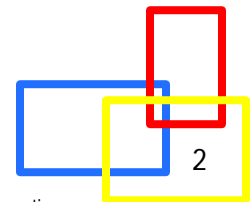
A domain model-centric approach to J2EE
development

Keiron McCammon
CTO
Versant Corporation



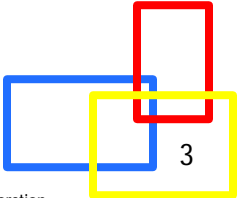
Overview

- What is a domain model centric approach?
- When should you consider using it?
- How would you use it?
 - ◆ Architecture
 - ◆ Design
 - ◆ Implementation
 - ◆ EIS Integration





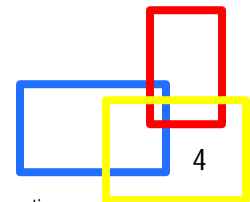
What



Implementing Business Logic

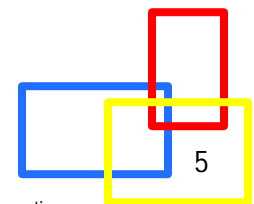
- Using objects to model the business domain
- Model supports complex application logic
 - ◆ Its more than data validation/verification
- Persistence is secondary consideration
 - ◆ Focus is the business logic not the data
- Essentially its an OO application

Process-centric approach versus data-centric





When



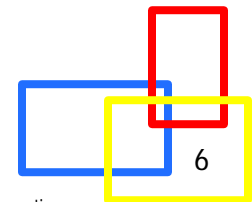
Data-centric vrs Process-centric

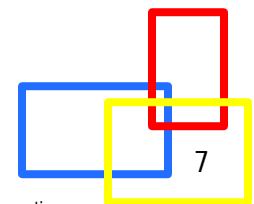
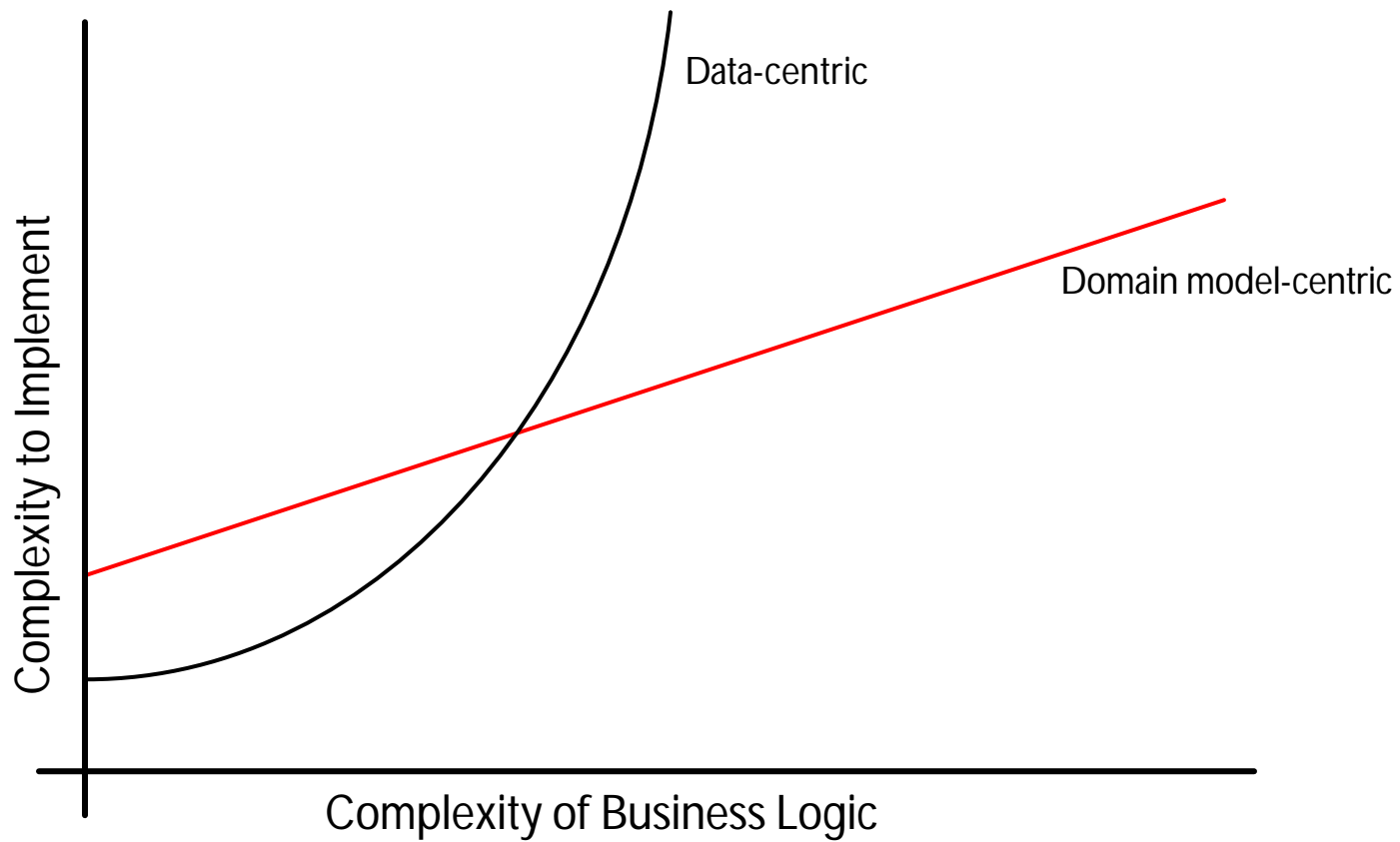
Data-centric

- Data-driven
- Ad-hoc query based, decision support
- Record oriented, batch processing
- Wrapping existing database

Process-centric

- Process-driven
- Navigational access
- Complex application logic
- Object-oriented
- Building a new application





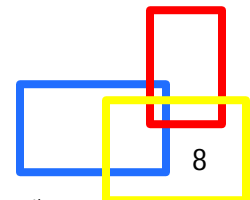
Pro's & Con's

Pro's

- Fully leverage OO benefits
- Long term payoff as application complexity increases

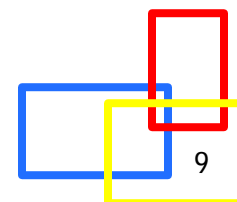
Con's

- Requires OO expertise
- Higher cost of entry for simply application



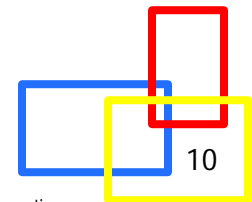


How



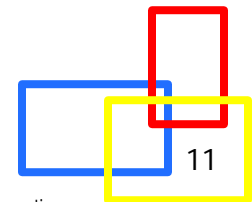
Application Data & Enterprise Data

- Application Data is the domain model objects, distinct from Enterprise Data
 - ◆ Application data modeled as objects
 - ◆ Enterprise data already exists in other systems
- Interact with application data and transact with enterprise data
- Application data doesn't need to leave the middle-tier



What about the database?

- Still need to persist application data
- Interest is in storing/retrieving objects not rows/columns



Alternative Solutions

■ O/R Mapping Tools

- ✓ Reduces coding effort
- × Proprietary
- × Mapping overhead

■ DAO Design Pattern

- × Extra coding effort
- × Mapping overhead

- ◆ <http://developer.java.sun.com/developer/restricted/patterns/DataAccessObject.html>

■ EJB 2.0 CMP

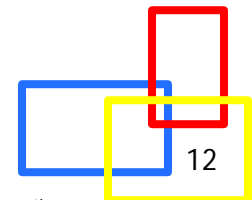
- ✓ Reduces coding effort
- ✓ J2EE standard
- × Mapping overhead
- × Ease of use

■ Java Data Objects

- ◆ <http://jcp.org/jsr/detail/012.jsp>
- ◆ Transparent object persistence
- ◆ Could use ODB as middle-tier database

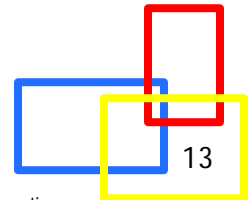
- ✓ No coding effort
- ✓ Java standard
- ✓ Open choice of datastore

- <http://www.versant.com/products/enjin/index.html>



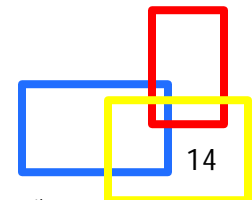
Contentious Conjecture...

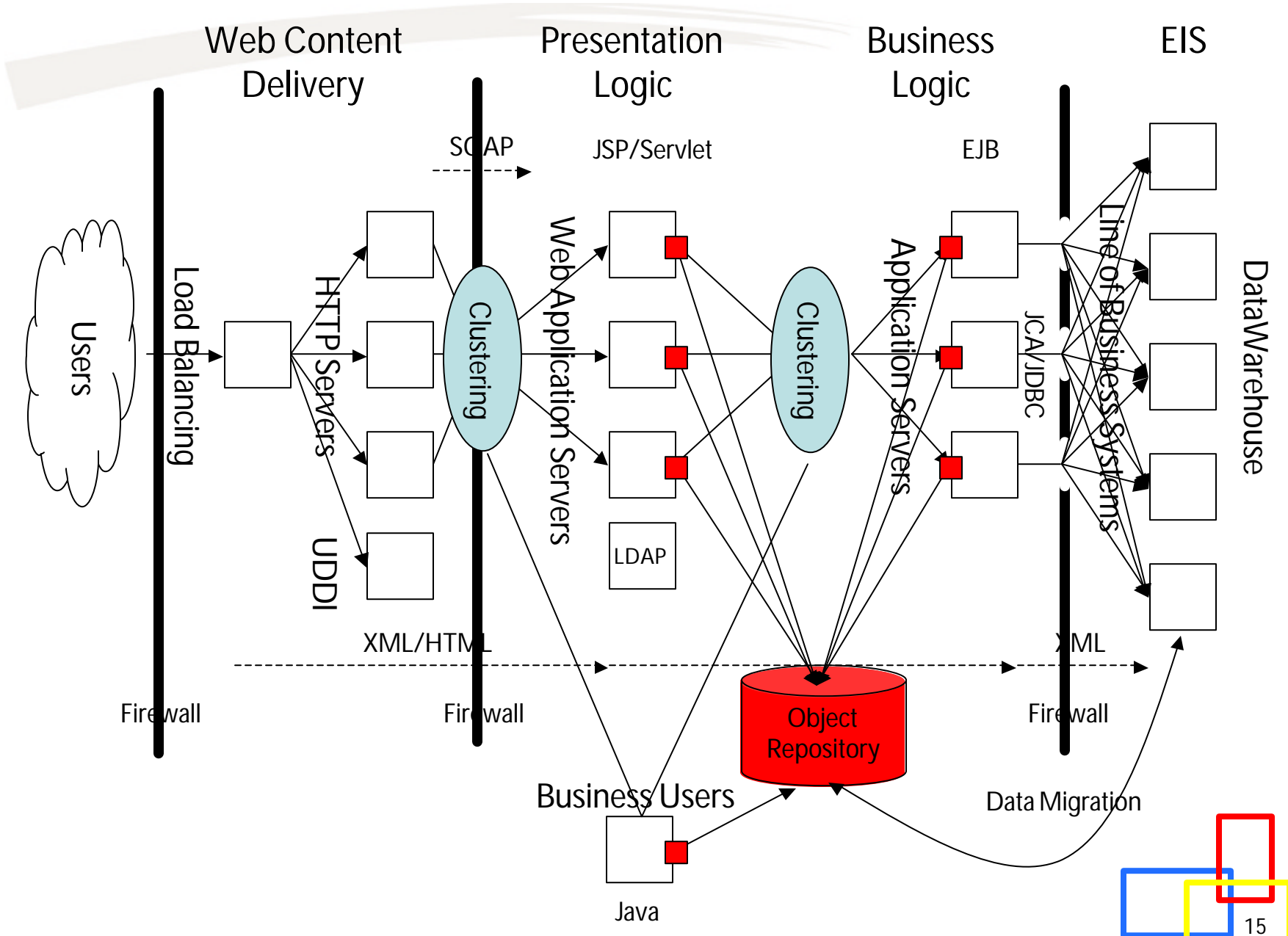
“J2EE doesn’t offer much support for domain model-centric applications”






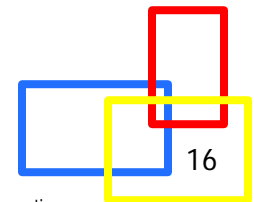
How (Architecture)





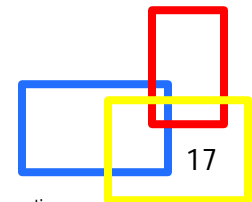


How (Design)



Domain vrs Component Models

- Crucial to distinguish between Domain & Component models
 - ◆ Domain model
 - Models the data and relationships as Objects
 - ◆ Component model
 - Models external, remote interfaces
- Merging the two leads to “fine grained” EJBs, every Java class being an EJB
 - ◆ Fine structure is too inefficient for heavy weight components
- Focus on “coarse grained” EJBs
 - ◆ Coarse structure diminishes benefits of OO approach
 - ◆ But good for defining interfaces & enterprise services



Modeling the Domain

■ Data Objects

- ◆ Objects that represent “real-world” entities

 - Customer; Order; Product

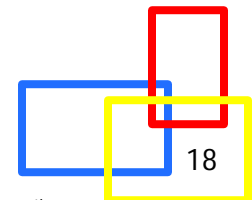
- ◆ And their relationships

 - Customer “can have many” Orders

■ Built using Java Classes

- ◆ Light-weight

- ◆ Intra-VM access only



Modeling the Components

■ Enterprise Java Beans

◆ Interfaces that represent interactions

- A customer “can place” an Order
- Provides a view or “façade” onto the domain model

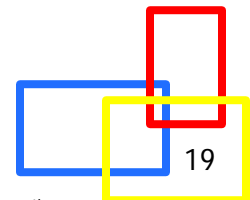
◆ And their enterprise services

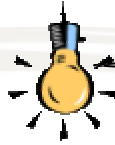
- Security; naming; transaction management

■ Built using Enterprise Java Beans

◆ Heavy-weight

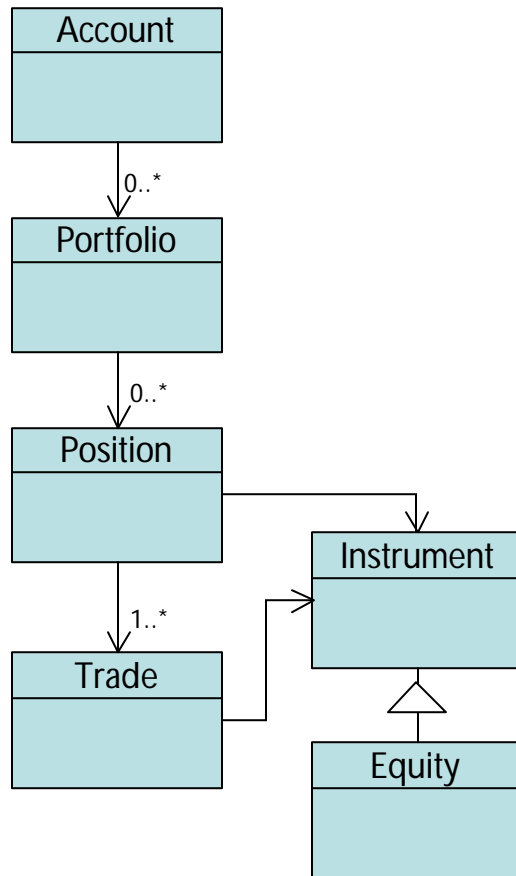
◆ Inter-VM access, distributed





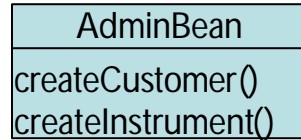
Combines benefits of both Object modeling and EJB development

Domain model

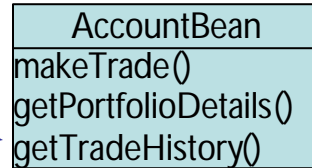


UML

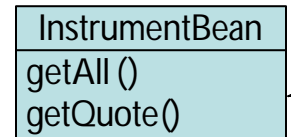
Component Model



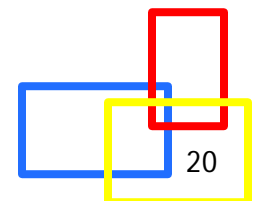
Façade for Account/Instrument
Separates business methods for managing domain model




Façade for Account
Customer interactions, hides underlying navigational model

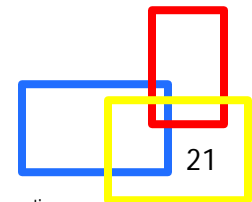


Façade for Instrument
Manipulates all Instruments, hides underlying inheritance model



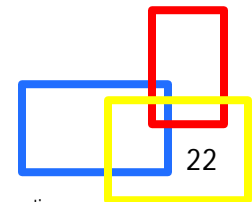


How (Implementation)

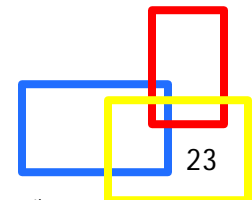
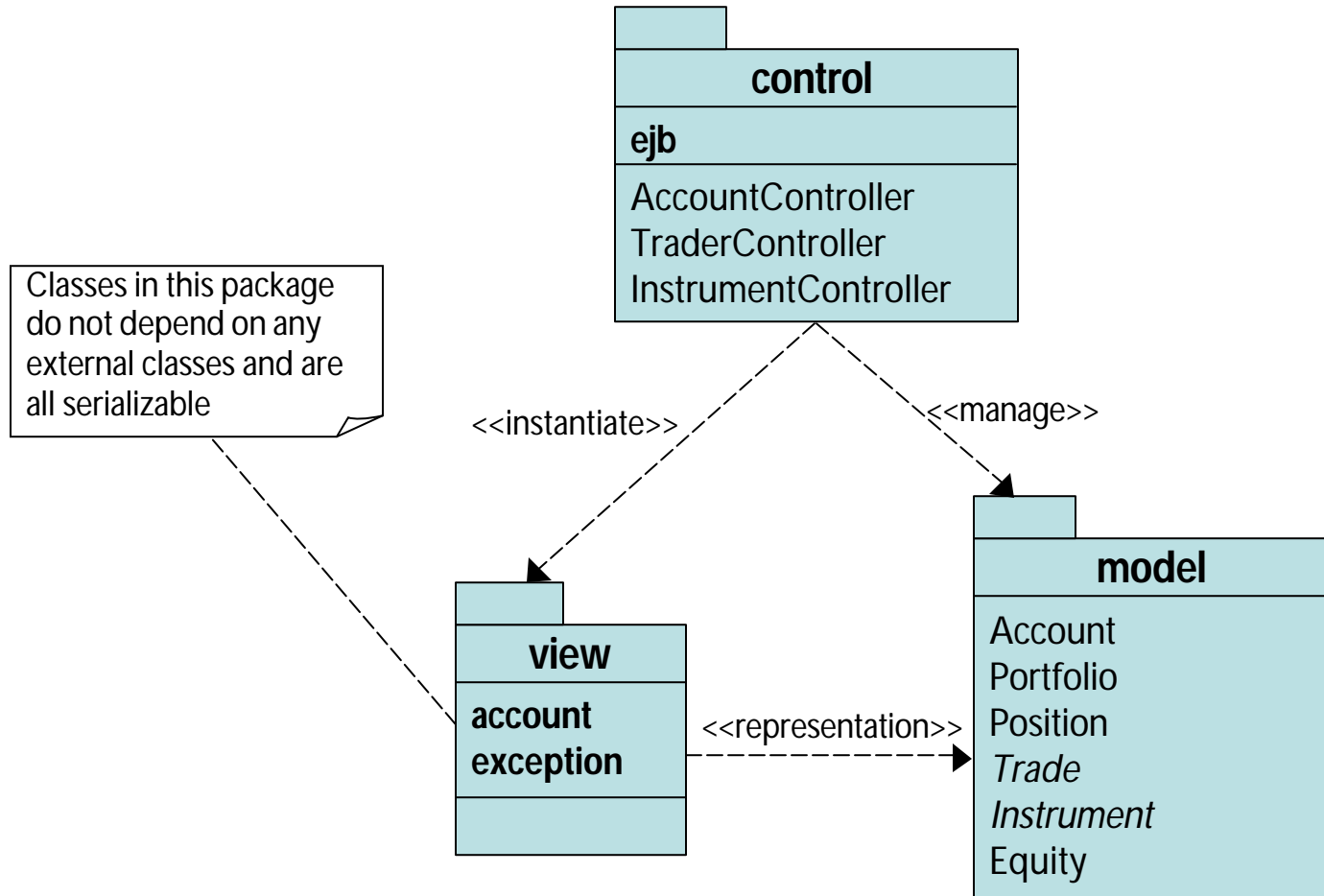


Software Layering

- Separate implementation and isolate dependencies
 - ◆ Simplifies development, testing, debug, maintenance
- Loosely-coupled components
 - ◆ Isolate client versus server-side classes
 - ◆ Simplifies distributed deployment
- Three main layers
 - ◆ Model
 - ◆ View
 - ◆ Control

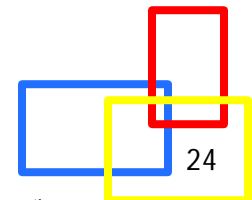


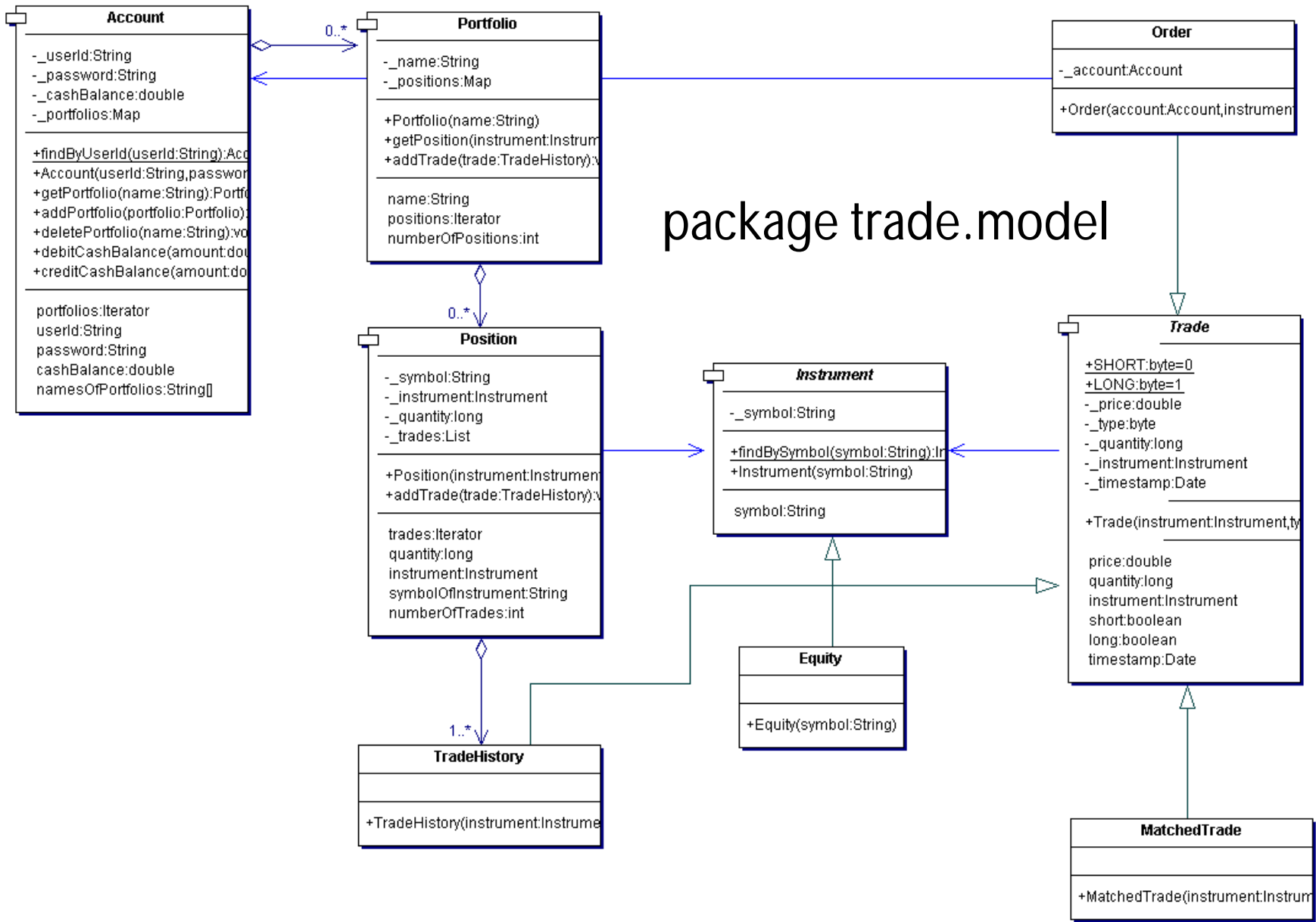
Software Layers



Model (Domain Object Model)

- Data Objects
 - ◆ Complex data, complex data relationships
 - ◆ Fine-grained
 - ◆ Persistent and transactional
- No business logic
 - ◆ Mainly getters/setters
- Independent of View and Controller layers
 - ◆ Although may throw exceptions

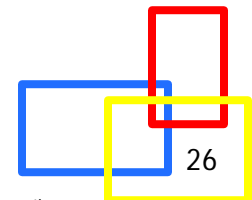




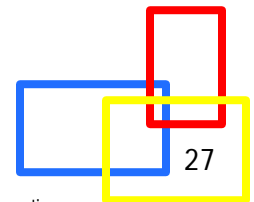
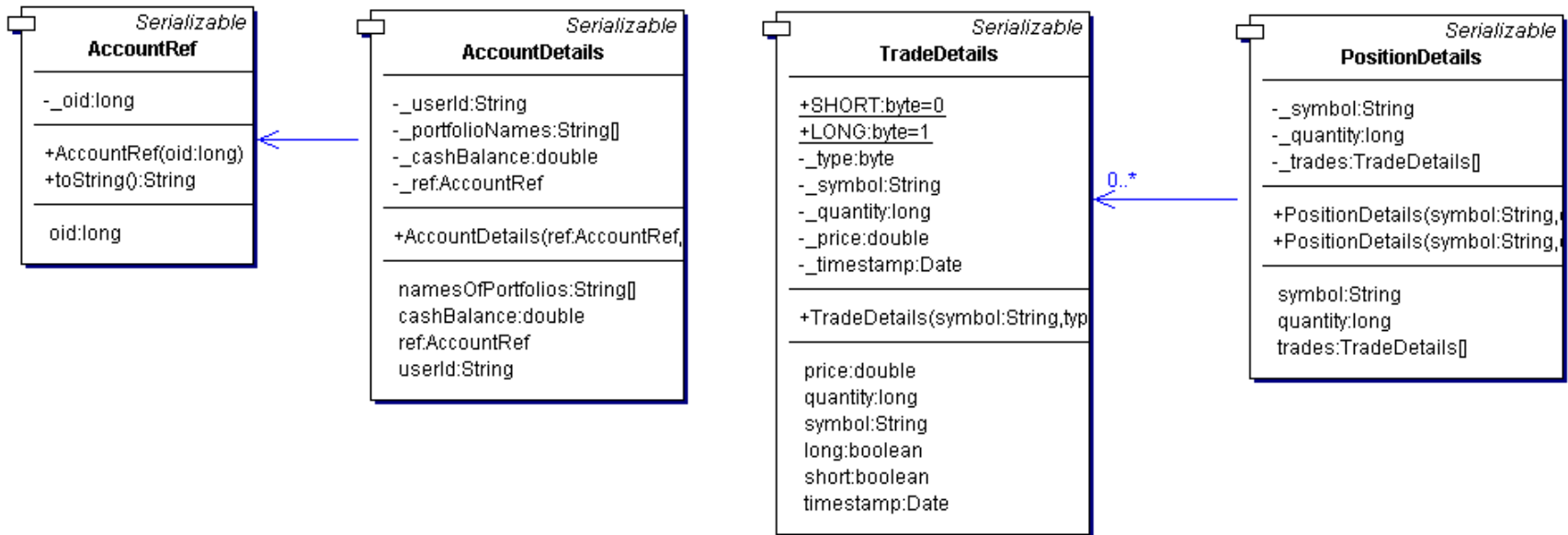
package trade.model

View (External Data Interchange)

- Light-weight Java classes
 - ◆ Serializable, non-persistent, non-transactional
 - ◆ Simple data and simple data relationships
 - ◆ Ideally immutable
 - ◆ No external dependencies
- No business logic
 - ◆ Mainly getters/setters
- Instantiated by Controller layer, representation of Model layer

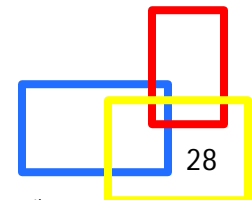


package trade.view



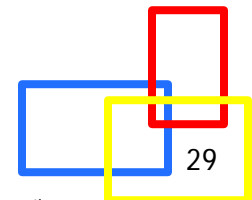
Control (Business Logic)

- External Interfaces
 - ◆ Business Logic
 - ◆ Coarse-grained operations on domain model
 - ◆ Handles persistence and transactions
- Manages the Model Layer and instantiates the View Layer



Control (cont'd)

- Implement business logic as normal Java classes
 - ◆ Simplifies unit testing and debugging
 - ◆ Can be re-used with JSP/Servlets
- EJBs layer on top of business logic classes
 - ◆ Add inter-component interactions
 - ◆ Add transaction propagation/management
 - ◆ Supports distributed deployment



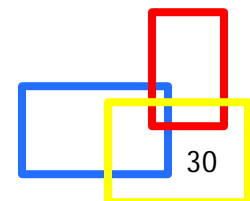
package trade.control

ejb
+AccountSessionBean +TradeSessionBean

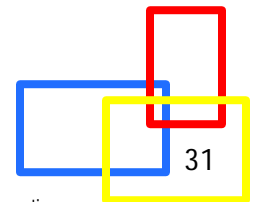
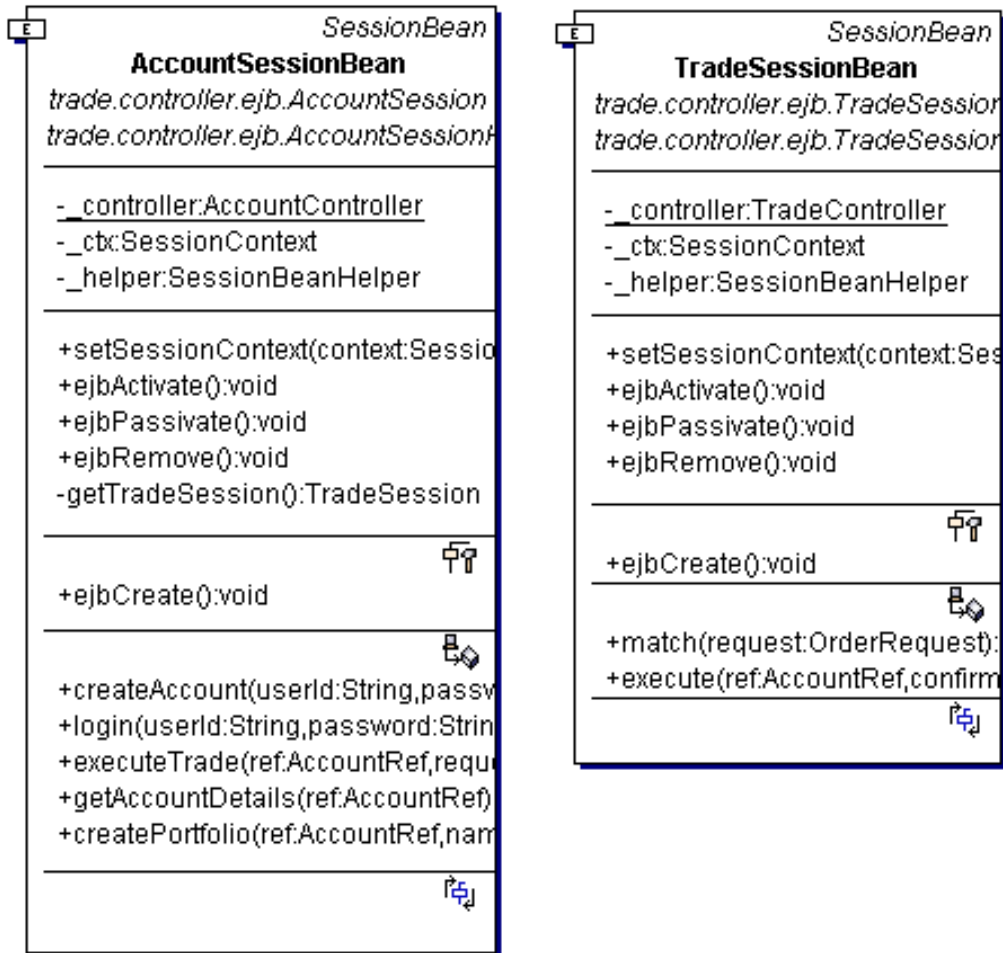
AccountController
+login(userId:String,password:Strin +creditAccount(ref:AccountRef,depo +debitAccount(ref:AccountRef,debit +getAccountDetails(ref:AccountRef, +getPositionDetailsForPortfolio(ref +createPortfolio(ref:AccountRef,nar +getTradeDetailsForPortfolio(ref:Ac +addTrade(ref:AccountRef,confirma +createAccount(userId:String,pass +deleteAccount(ref:AccountRef):voi -getAccount(ref:AccountRef):Accoun -getInstrument(ref:InstrumentRef):I

TradeController
-_rand:Random
+TradeController() +getQuote(ref:InstrumentRef):Quot +getQuotes(refs:InstrumentRef[]):Q +match(request:OrderRequest):Ord +execute(ref:AccountRef,confirmati -getAccount(ref:AccountRef):Accoun -getInstrument(ref:InstrumentRef):I -getMatchedTrade(confirmation:Ord

InstrumentController
+findBySymbol(symbol:String):Instr +createEquity(symbol:String):Instru

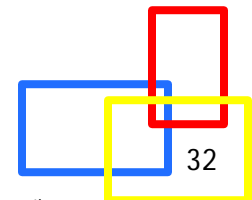


package trade.control.ejb

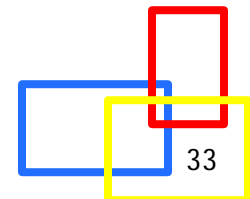
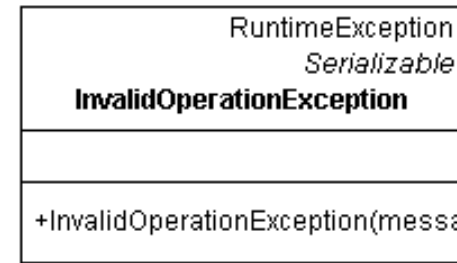
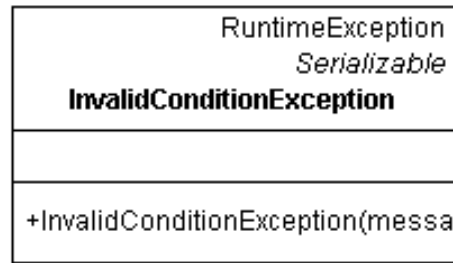
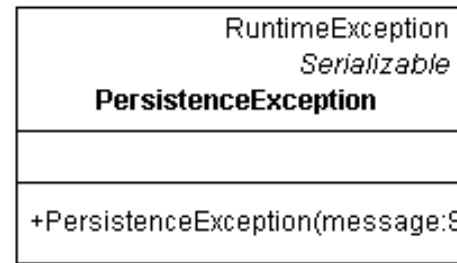
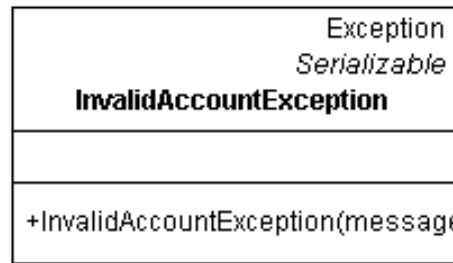


Exceptions

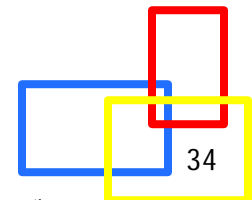
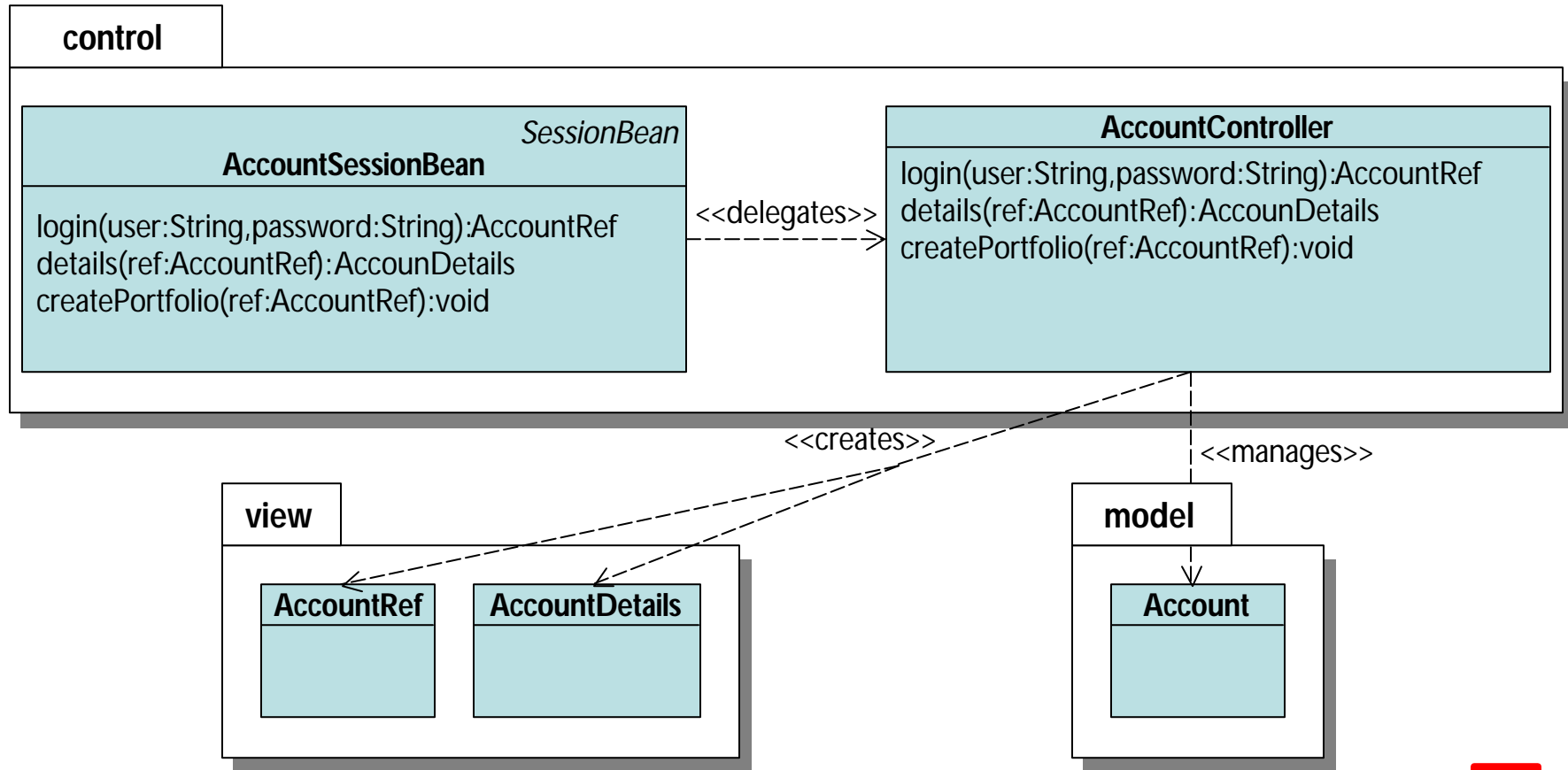
- Identify internal versus external exceptions
 - ◆ Internal exceptions never thrown outside of Control Layer
- External exceptions belong to View Layer
 - ◆ No internal dependencies

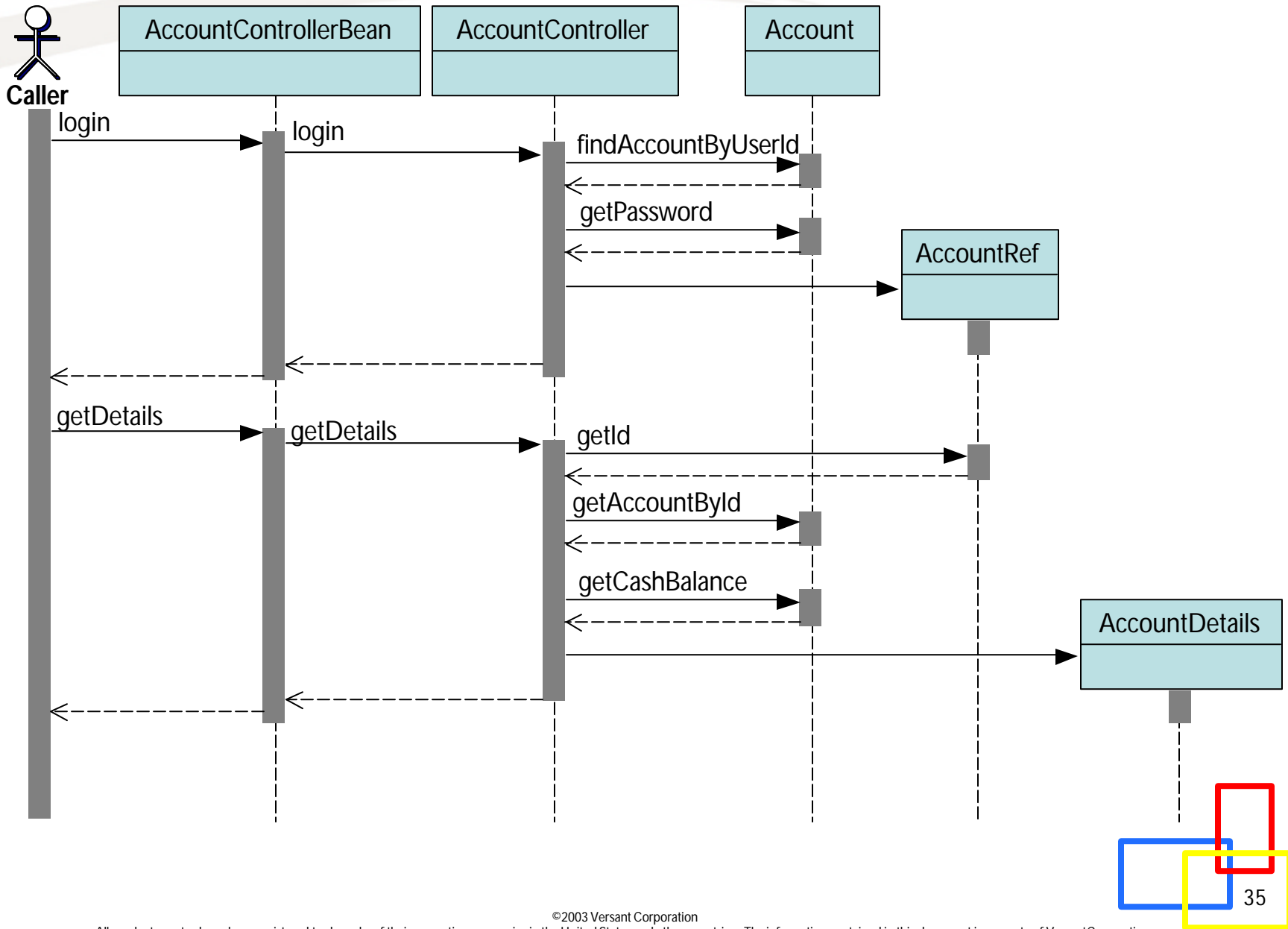


package trade.view.exception



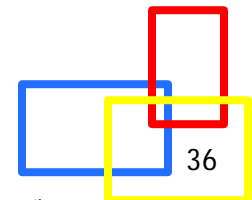
An Example...






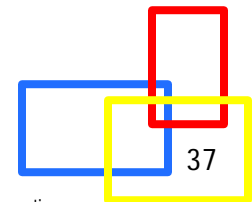
Roles & Responsibilities

- J2EE + light-weight object persistence allows the Domain model to be separated from the Component model
- Domain model is implemented as Java objects
- J2EE manages the Component model
 - ◆ Components manipulate the domain model as Java objects
 - Components are stateless SessionBeans



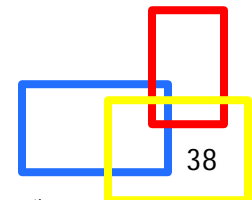


How (EIS Integration)



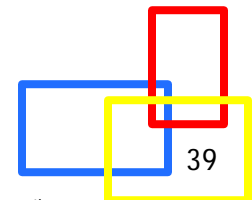
Enterprise Coordination

- Coordination of “Application Data” with Enterprise Information Systems (EIS)
- Different approaches depending on needs
 - ◆ Synchronous
 - Distributed Transaction Management (XA)
 - ◆ Asynchronous
 - Omni or bi-directional
- Connectivity
 - ◆ Point-to-point
 - JDBC/JCA
 - ◆ Message-oriented
 - JMS



Synchronous Coordination

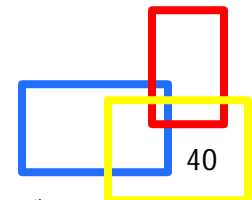
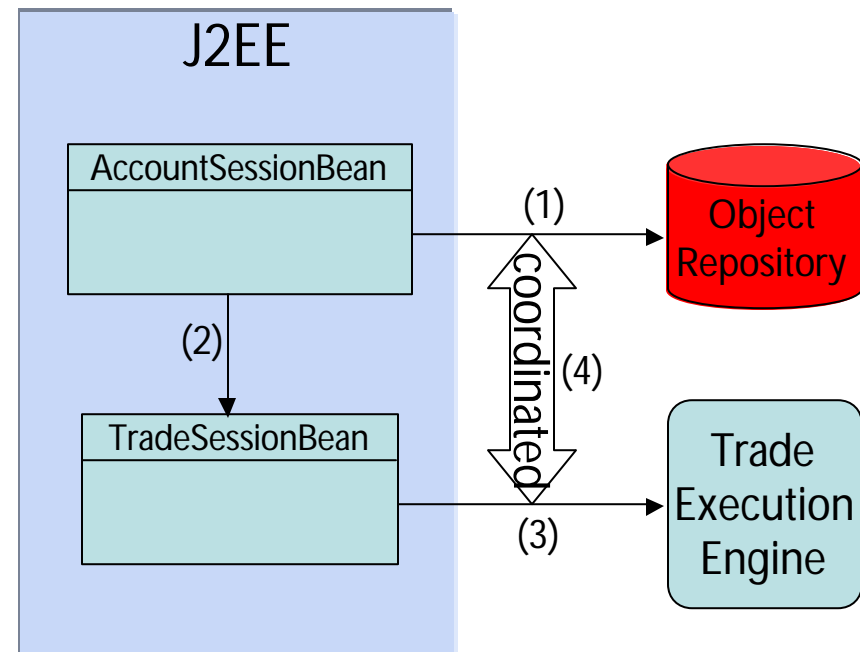
- J2EE provides a Java Transaction Service
 - ◆ Distributed Transaction Manager
- Use J2EE architecture to build application
 - ◆ “Application Data” managed in the middle-tier
 - ◆ EIS connectivity via JDBC/CMP/JCA/JMS
- Transform objects to/from EIS representation as required
 - ◆ Tables/XML/...
 - ◆ Cache EIS data as “Application Data” for long running transactions
- Synchronization controlled by J2EE
 - ◆ Just need to call appropriate EJBs
- Message-driven Beans facilitate near-synchronous coordination



Synchronous

Sequence of events:

- (1) AccountSessionBean updates Account
- (2) AccountSessionBean calls TradeSessionBean to execute a trade
- (3) TradeSessionBean executes trade
- (4) J2EE commits transaction



Considerations

Pros

- Guaranteed consistency

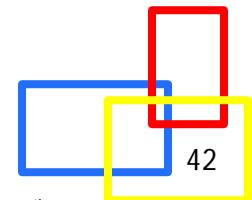
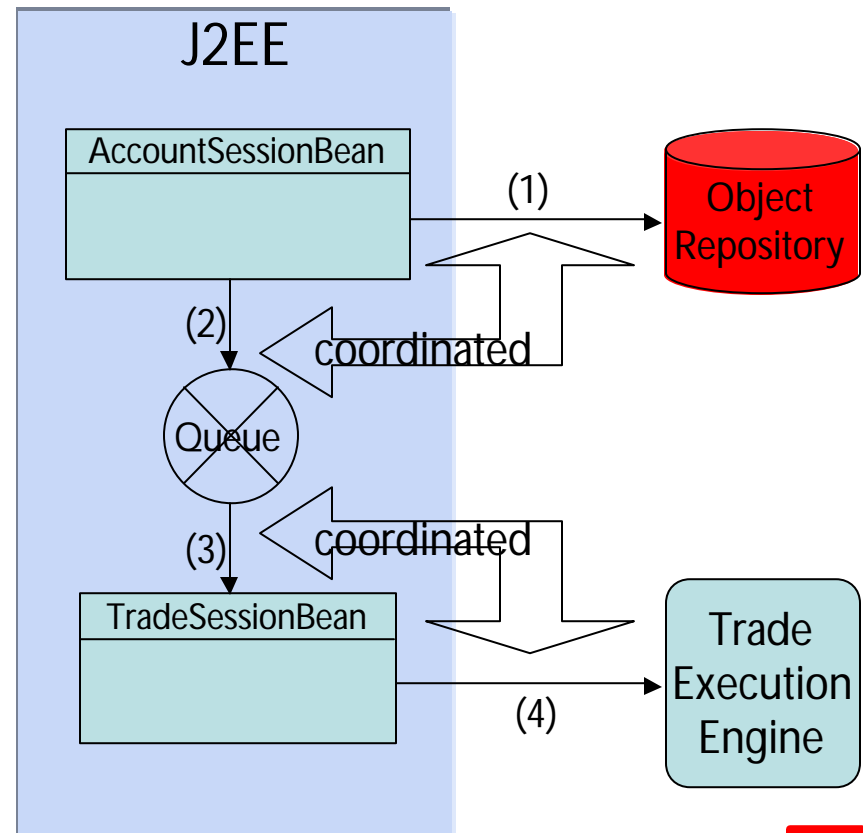
Cons

- Expensive
- Prone to unavailability of external systems
 - ◆ Unless using JMS
- Prone to performance bottlenecks of external systems

Using Message-driven Beans

Sequence of events:

- (1) AccountSessionBean updates Account
- (2) AccountSessionBean sends message to TradeSessionBean
- (3) TradeSessionBean reads message
- (4) TradeSessionBean executes trade



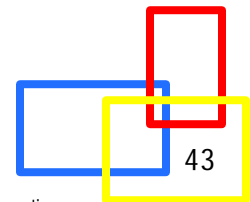
Considerations

Pros

- De-couples middle-tier from enterprise systems

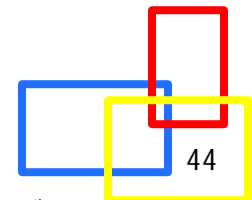
Cons

- Not synchronous
- Additional complexity



Asynchronous Coordination

- Middle-tier persistence guarantees long term storage
 - ◆ It's a database!
- Use J2EE architecture to build application
 - ◆ "Application Data" managed in middle-tier
- Periodically propagate "business transactions" from middle-tier to enterprise systems and vice-versa
 - ◆ Time-based; # of transactions; ...
- Omni-directional
 - ◆ From or to the middle-tier
- Bi-directional



Propagating changes from Middle-tier

■ Within J2EE

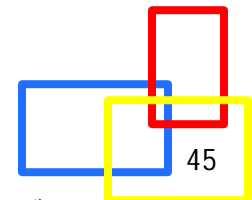
- ◆ External event triggers EJB to perform synchronization
 - EIS connectivity via JDBC/CMP/JCA/JMS
- ◆ Transform objects to EIS representation
 - Tables/XML/flat-files

■ Outside J2EE

- ◆ Batch processing
 - External application periodically exports changes from middle-tier to appropriate EIS representation

■ Suitable for propagating new “data”

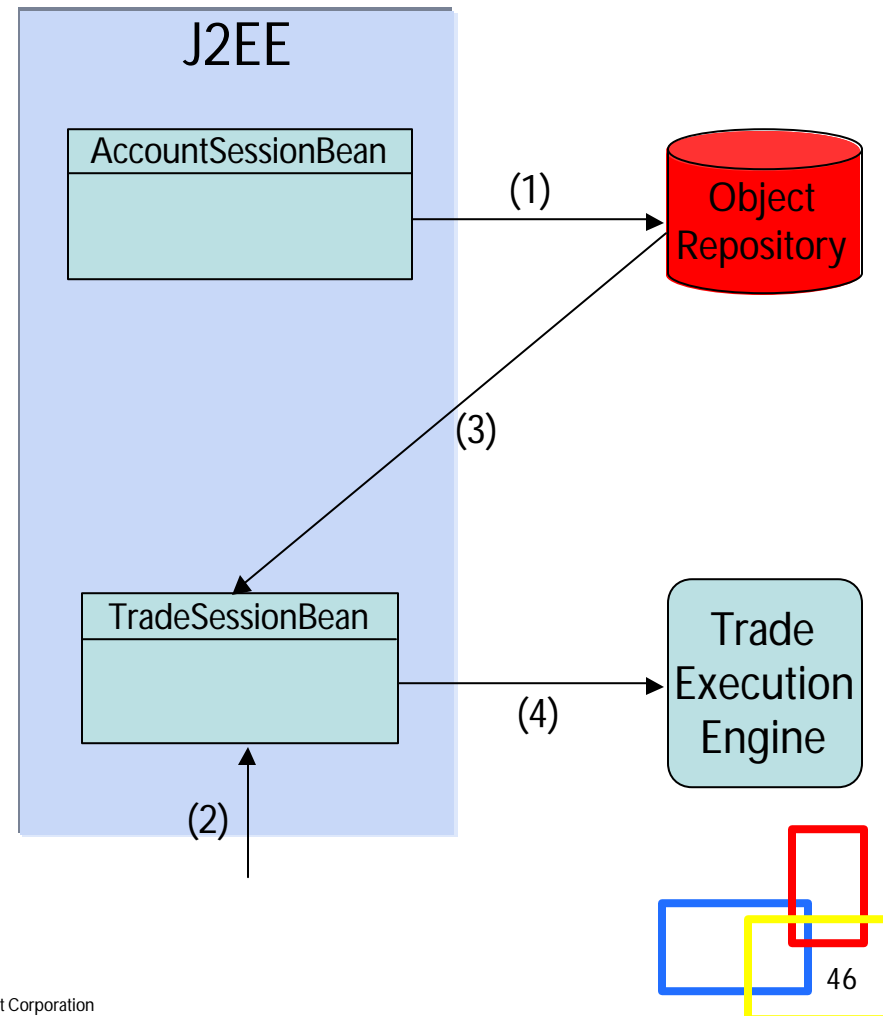
- ◆ No conflicts, “data” owner by middle-tier



Within J2EE

Sequence of events:

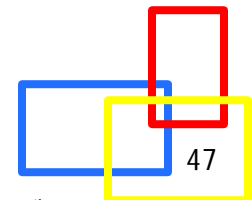
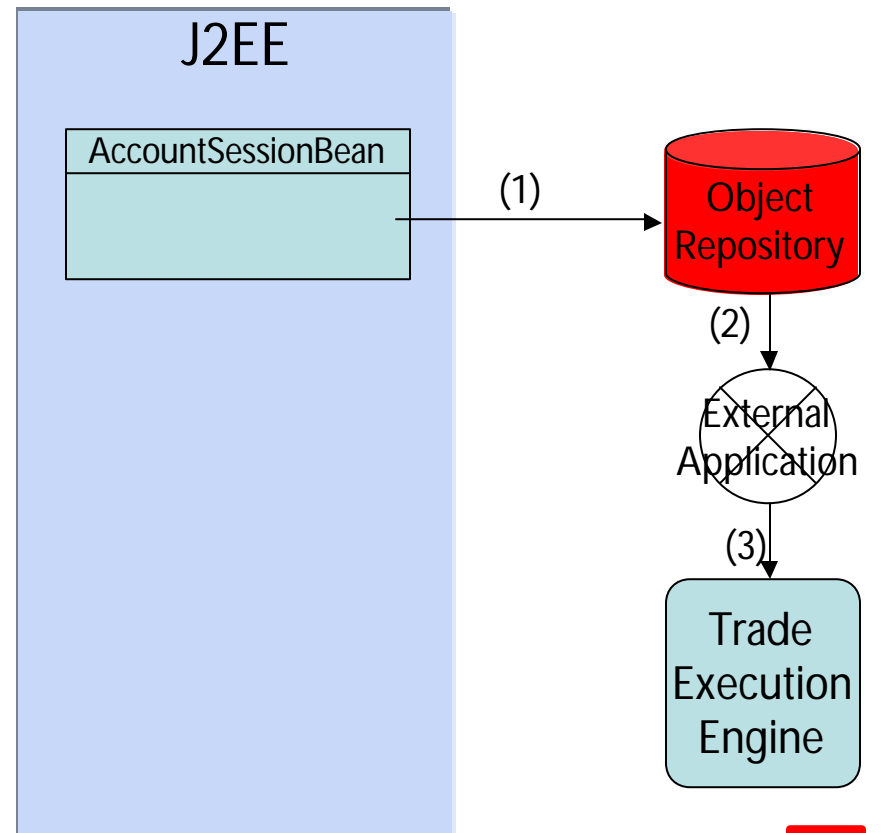
- (1) AccountSessionBean updates Accounts
- (2) An external trigger calls TradeSessionBean
- (3) TradeSessionBean gets trades from middle-tier
- (4) TradeSessionBean executes batch of trades



Outside J2EE

Sequence of events:

- (1) AccountSessionBean updates Account
- (2) External application reads updates
- (3) External application executes batch of trades



Propagating changes to the Middle-tier

■ Within J2EE

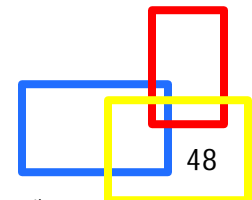
- ◆ External event triggers EJB to perform synchronization
 - EIS connectivity via JDBC/CMP/JCA/JMS
- ◆ Transform EIS data to appropriate object representation

■ Outside J2EE

- ◆ Batch processing
 - External application periodically exports changes from EIS to appropriate object representation

■ Suitable for propagating new “data” or changes to reference data

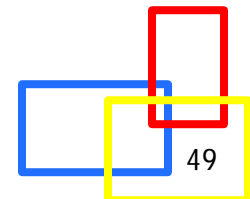
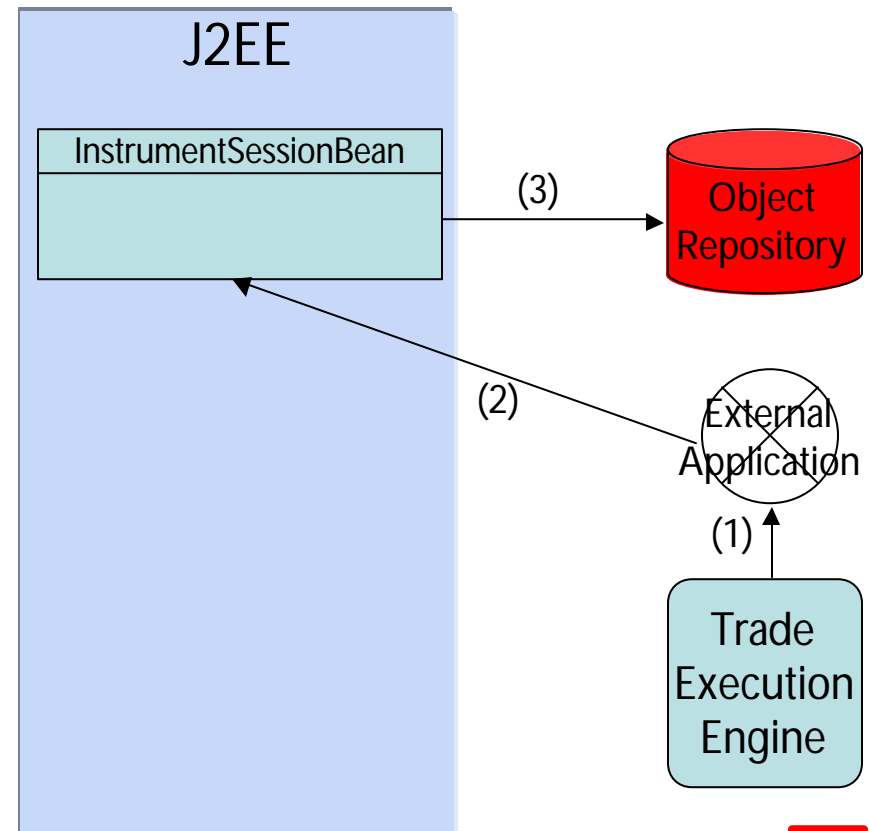
- ◆ No conflicts, “data” owned by Enterprise



Within J2EE

Sequence of events:

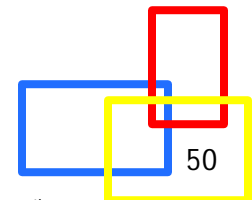
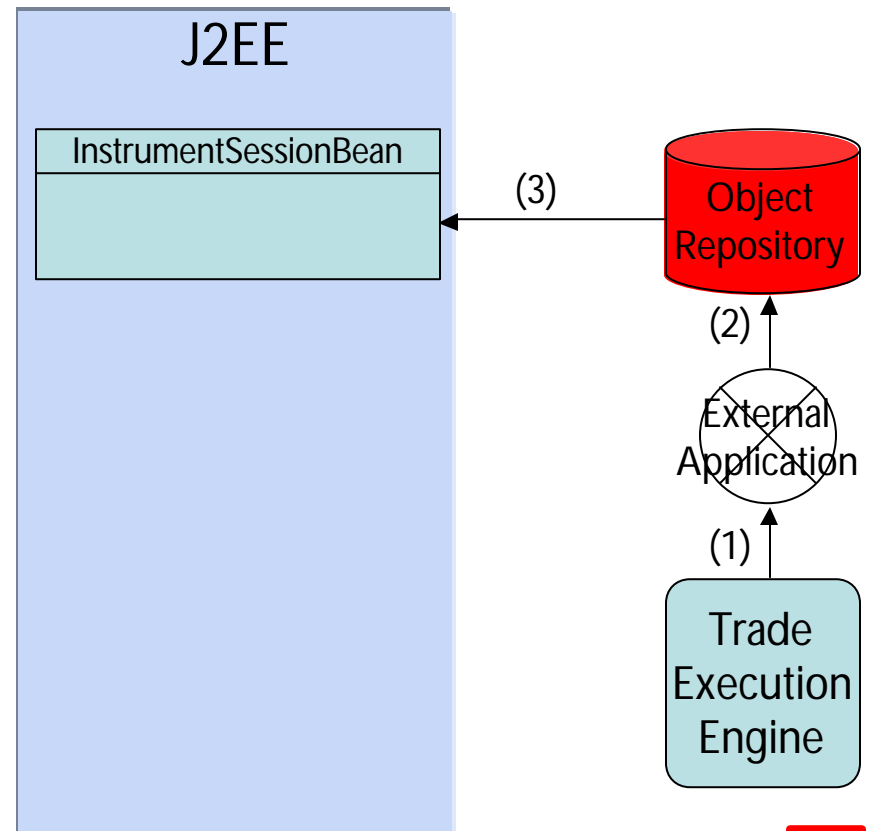
- (1) External system sends message to InstrumentSessionBean
- (2) InstrumentSessionBean reads message
- (3) InstrumentSessionBean creates new instrument in the middle-tier



Outside J2EE

Sequence of events:

- (1) External application reads changes from external system
- (2) External application creates new instruments in the middle-tier
- (3) InstrumentSessionBean can read new instruments



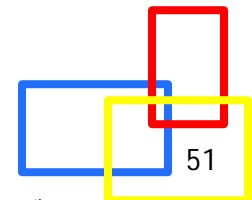
Considerations

Pros

- Decouples middle-tier from enterprise systems
- Possible to coordinate with batch-oriented systems

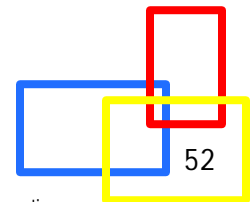
Cons

- Not synchronous
- Not bi-directional



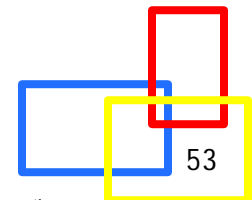
Bi-directional Coordination

- Replication of changes to/from middle-tier
 - ◆ Data changed both in the middle-tier and enterprise system
- Introduces potential for conflicting updates
 - ◆ Best approach is to avoid need for bi-directional updates



Summary

- Domain model-centric approach is good for process-centric applications
 - ◆ Keep object management orthogonal to the component modeling
- Domain model can easily support many use-cases and applications
- Software layering simplifies development and reduces software dependencies
- Utilize J2EE capabilities for EIS Integration



Interesting Design Patterns

Sun Java Center J2EE Patterns

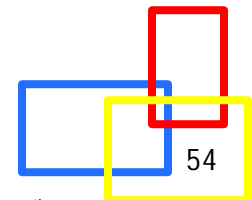
<http://developer.java.sun.com/developer/restricted/patterns/J2EEPatternsAtAGlance.html>

- Session Façade
- Value Object
- Data Access Object
- Value Object Assembler

Martin Fowler's Information System Architecture

<http://martinfowler.com/eaCatalog/>

- Domain Model
- Data Transfer Object
- Remote Façade





Thank you

