Modelling Archetypes

ACCU Conference 2009
Hubert Matthews
hubert@oxyware.com

Overview

- Archetypes – modelling patterns
- Static data modelling
- Linking to dynamic behaviour of system
- Rules and constraints
- Various bits of history
- Extensions to SOA, ESB and other stuff

Four basic archetypes

- Entities – “people, place, thing”
- Transactional objects – order, loan, payment
- Descriptions/specifications – title, type objects
- Roles – borrower, authoriser

Entity classes

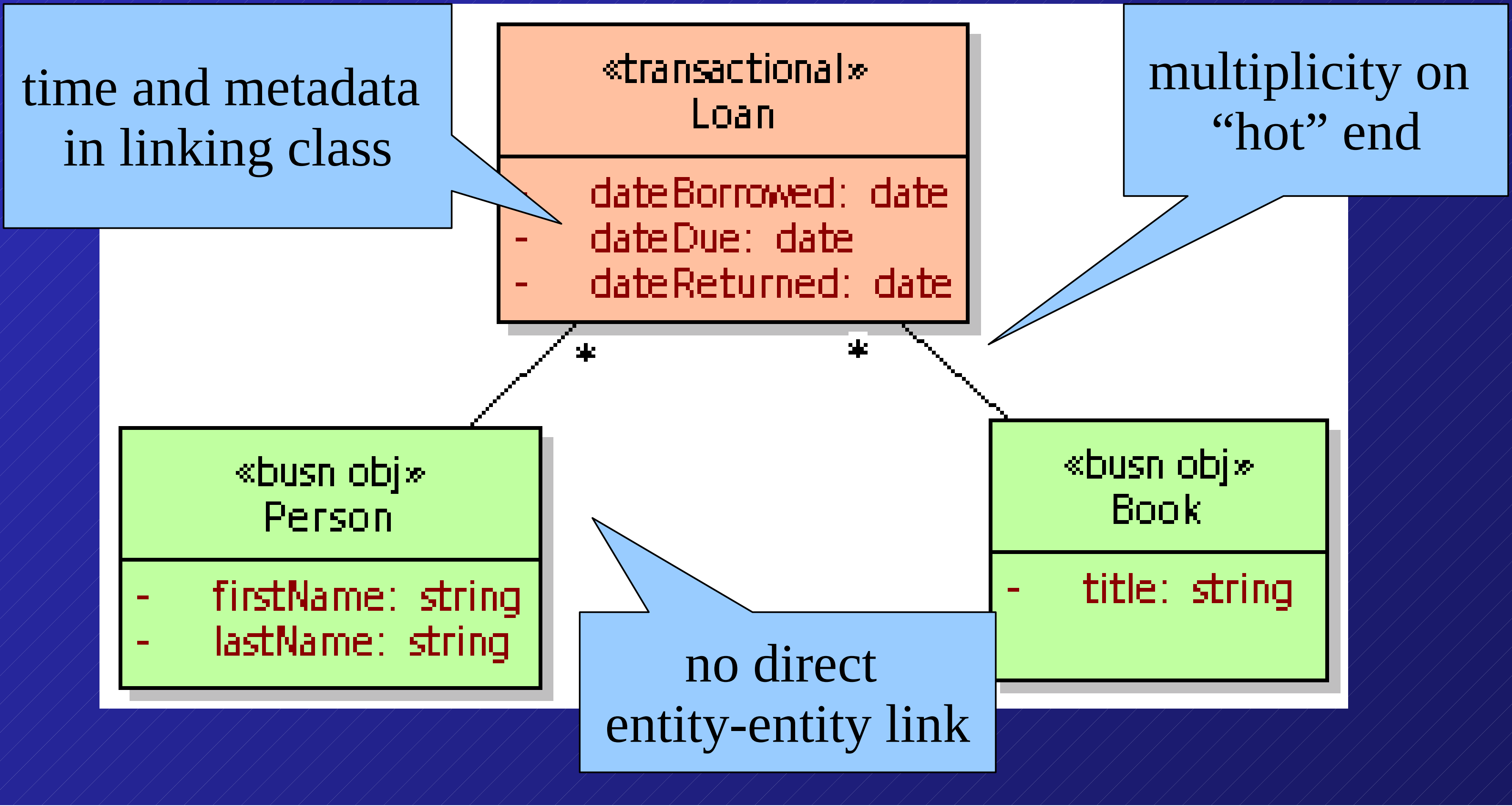
- The nouns in the standard “find the nouns” approach to OO – modelled in “green”
- Fairly static, eminently cacheable
  - No notion of time (history or future)
  - Have identities (name, ID, etc)
- Create, read, update, delete operations
  - Data only; no significant business processes
  - “Dull” use cases – get/set, edit/manage
- Often where people stop modelling (get stuck)
- Examples: customer, product, warehouse

Transactional classes

- Where the interesting stuff is!
- Related to time (look for timestamps) or states (look for status/modes)
  - Can deal with history and future, timespans
- High-volume, dynamic
- Link entities together – modelled in “pink”
- Basis of business processes
- Examples: loan, order, reservation, payment
  - Business forms are pinks that refer to green entities

Modelling in colour

- Patterns of connections between archetypes
- Use colour to denote archetypes and connection patterns to guide model building



time and metadata in linking class

«transactional»
Loan

dateBorrowed: date
- dateDue: date
- dateReturned: date

multiplicity on "hot" end

* *

«busn obj»
Person

- firstName: string
- lastName: string

«busn obj»
Book

- title: string

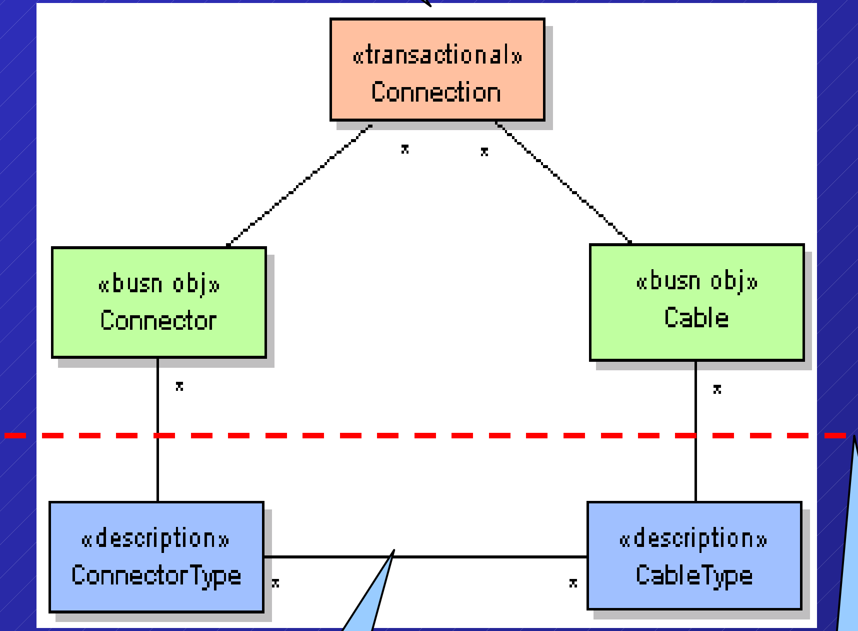
no direct entity-entity link

Description/specification objects

- Entities sometimes have associated information about their types
- Use a description or specification object - modelled in “blue” (as in “blueprint”)
  - Examples: title (book), make/model (car)
  - Catalogues are collections of blues
  - Type Object pattern
- Can be used to implement business and configuration rules in data
  - Fowler's Knowledge/Operational Split pattern

Rules in data (knowledge)

associated business process



allowed configurations (knowledge)

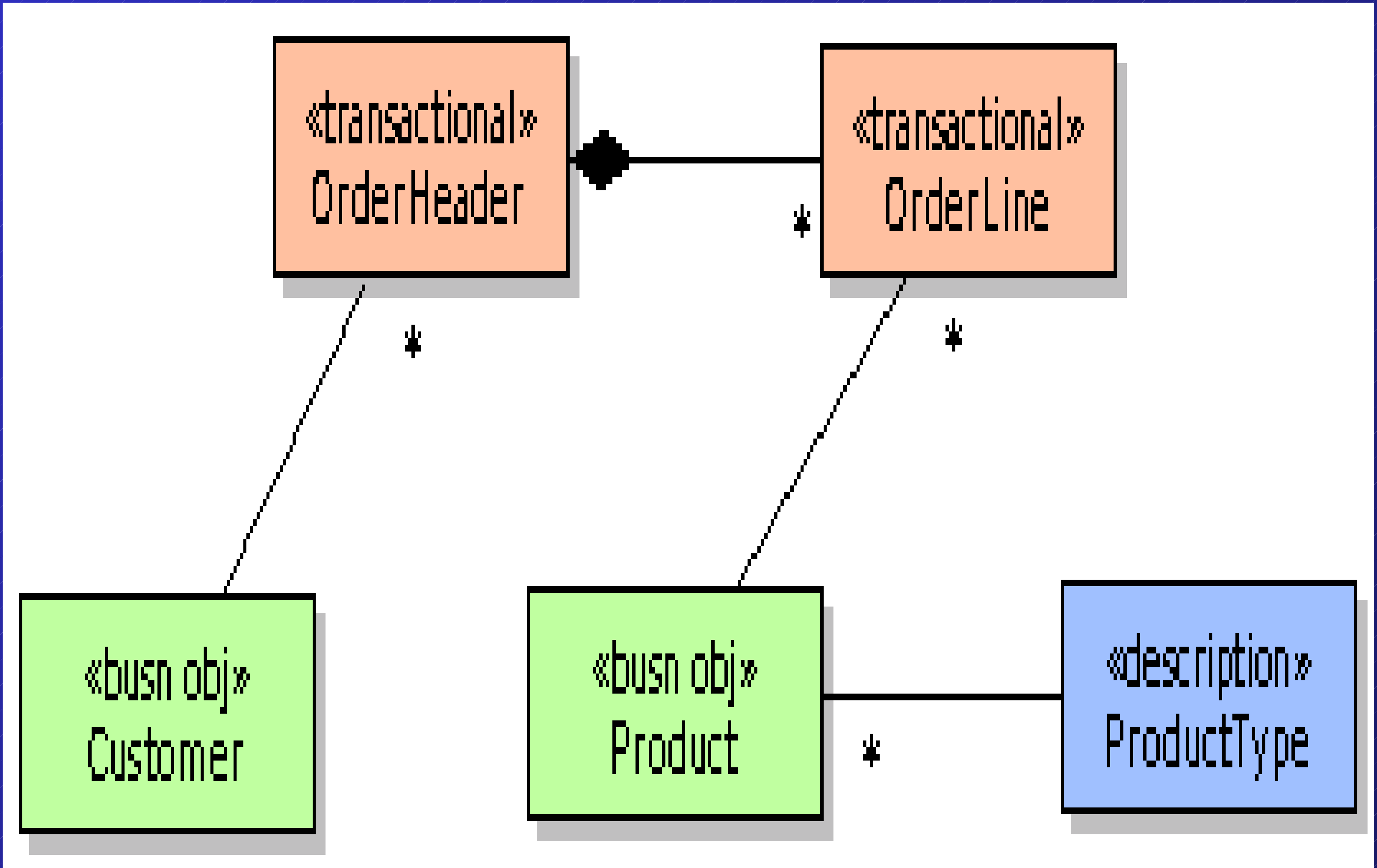
Fowler’s knowledge-operational split

- Only certain types of connector/cable pairings are valid
- Use type objects to encode rules
- Connection has 1:* to allow for time element
- Could use direct green-green link if history/future not required

Modelling guidelines

- Connect entities via a transaction ("pink")
  - Represents a step in a business process
  - Has time element, rules and constraints
  - Allows for history and future
- Connections between similar archetypes are whole-part relations (UML composition)
  - Multiplicity is 1 for whole, * for part
  - Dependent objects
- "*" multiplicity on "hot" end (pink->green->blue)
  - Great check on cardinality in database schemas

Simple order example

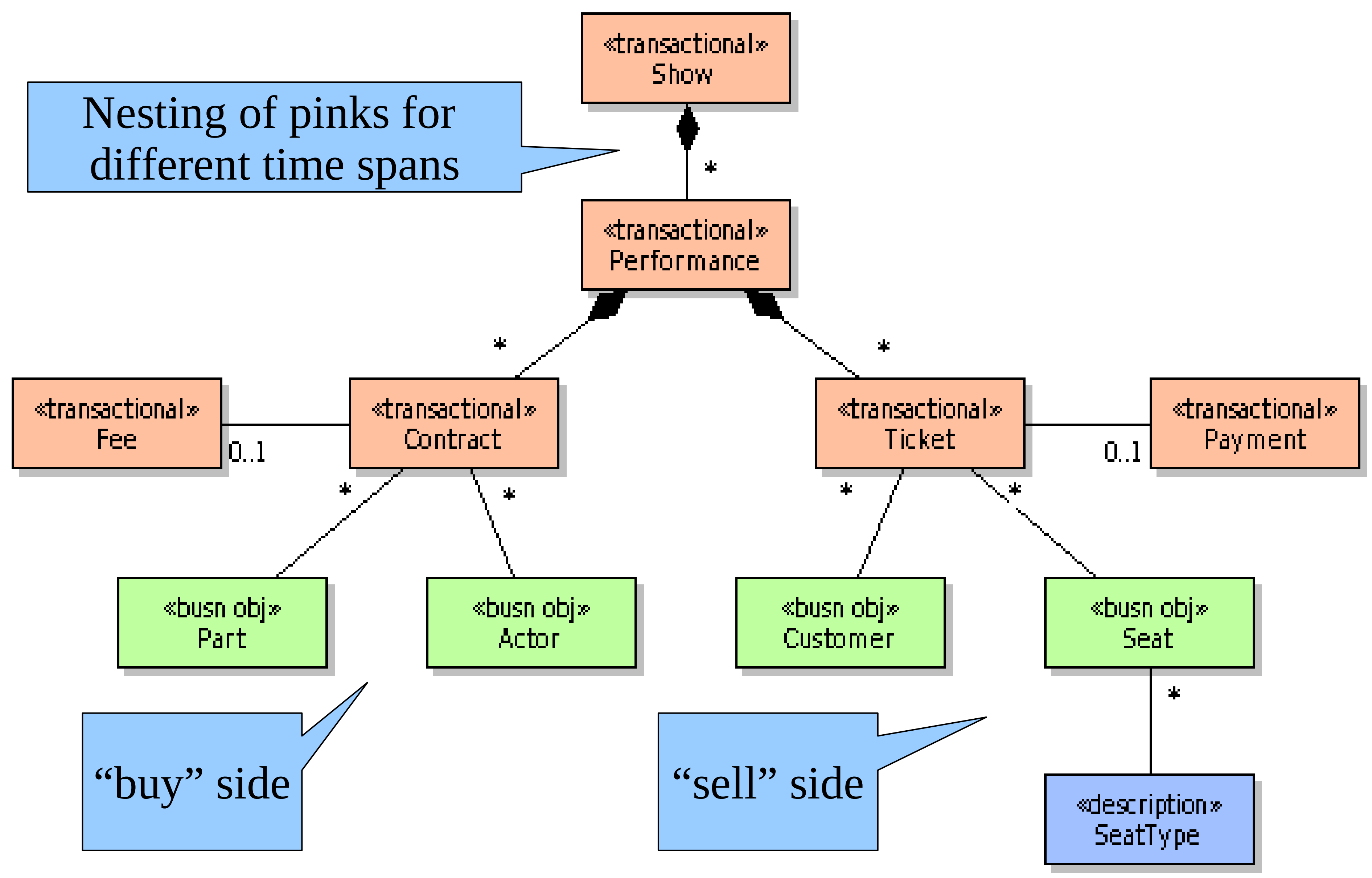


Copyright © 2009, Hubert Matthews

Common issues

- Not using transactions (“pinks”) for linking
  - Entity-to-entity links have no notion of time
  - Current state only; no history or future
  - No place for metadata – who did what when
- Confusing entities and description objects
  - Title v. Book

Theatre example

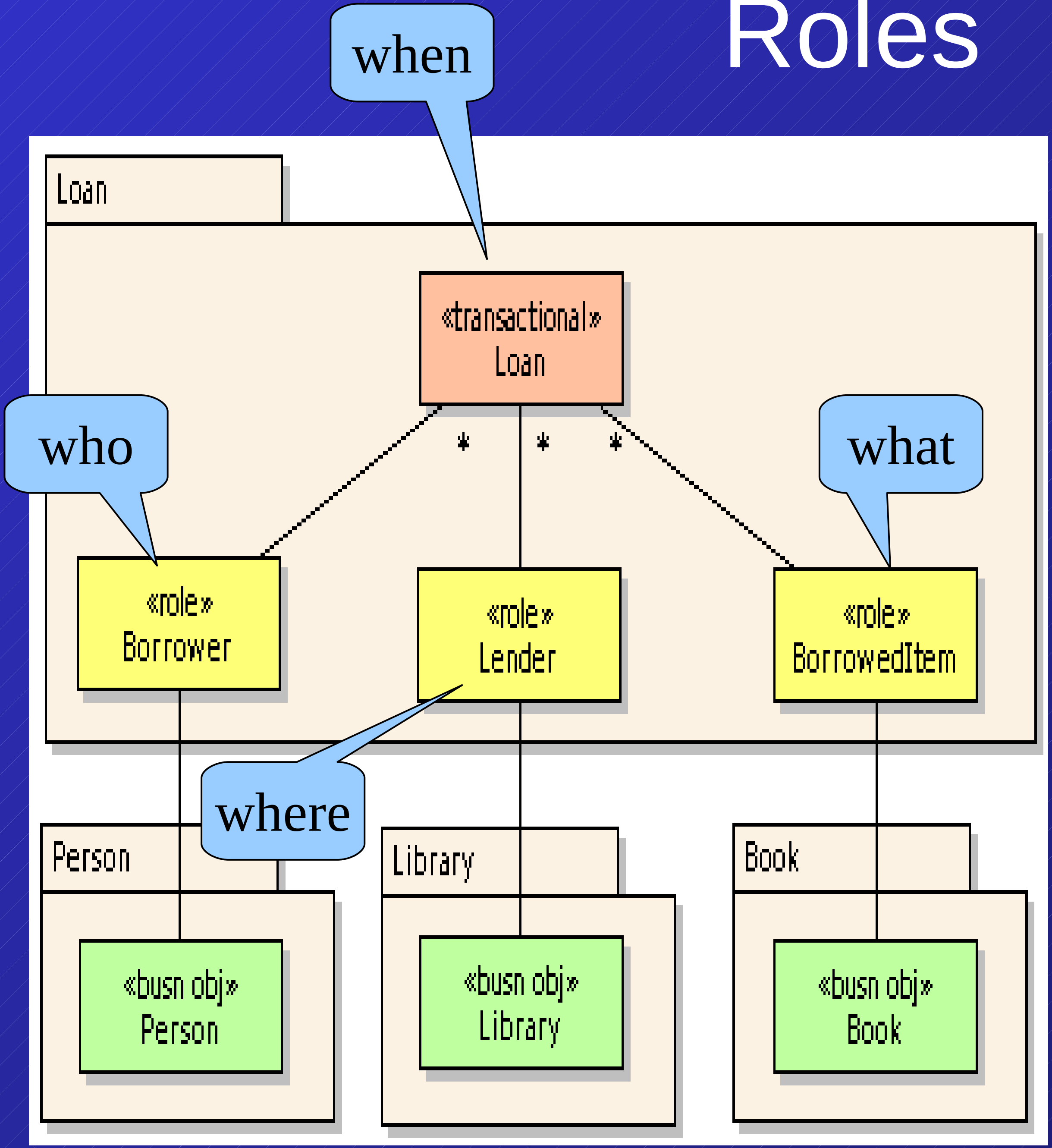


Nesting of pinks for different time spans

“buy” side

“sell” side

Roles



- Mostly associated with cross-component links
- Represent roles in a transaction
- Come between transaction and entities

Roles (2)

- An example of Proxy pattern (1:1 multiplicity across component boundary)
- Act as views on a database
  - Only details relevant to importing package
  - May also contain package-specific state
- More advanced modelling tool - not always required
- Related to Role Decoupling (a.k.a. Interface Segregation) pattern
  - E.g. Person may have roles of Doctor, Patient, Parent
  - One green, three roles
- Programming interfaces for mocks during testing

History lesson (Part 1)

- “Modelling in colour” - Peter Coad (Together, now Borland)
  - Only static data model – *no process*
  - Domain-neutral component unsuccessful attempt to include some process
  - Colours match available Post-It notes!
- Object/relational mapping tools
  - Rails/Grails/A.N.Other ORM mappers
  - Static data only – *no process*
- Domain-driven design (Evans) – *no process*
- Jackson System Development has trees for processes but no link to types/classes

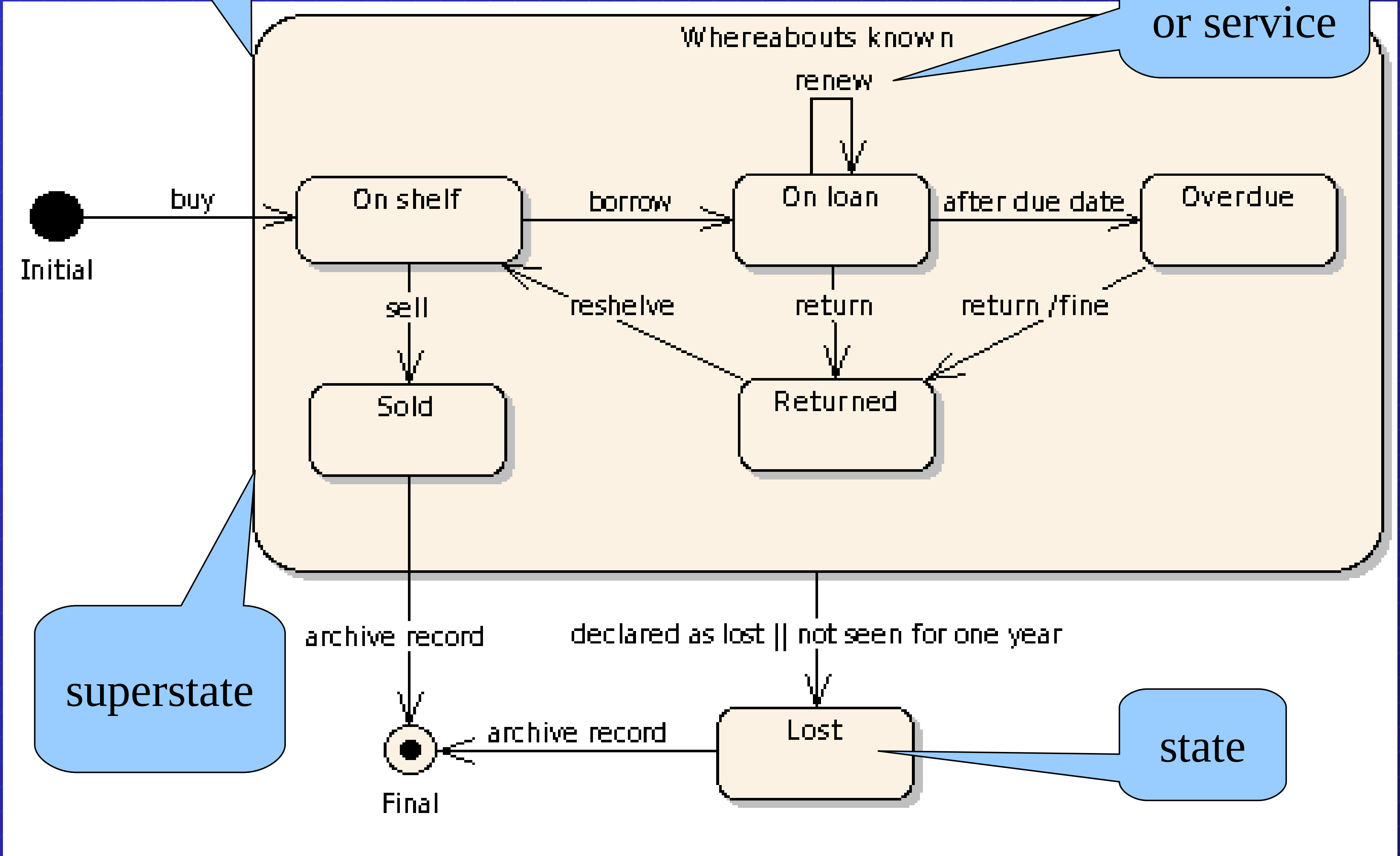
Dynamic process modelling

- Systems are built to do things, not store data
- More important than data model but not as well understood or used as often
- Key is that process model and data model must link up
  - Deep synergies between the two
  - Not often appreciated
  - Based around transactional objects ("pinks")

Statecharts v. activity diagrams

- Two approaches in UML – statecharts and activity diagrams
- Statecharts are superior for modelling processes (IMHO!)
- Activity diagram issues
  - Unhelpful semantics in UML (Petri net – requires branching)
  - Confusion over wait-on-arrows and wait-in-box
  - Encourage too much detail and drilldown
- Statecharts tend to have limited number of states that are relevant to business users
- How do you know when you have got all of the use cases/services? How can you check?

Library process



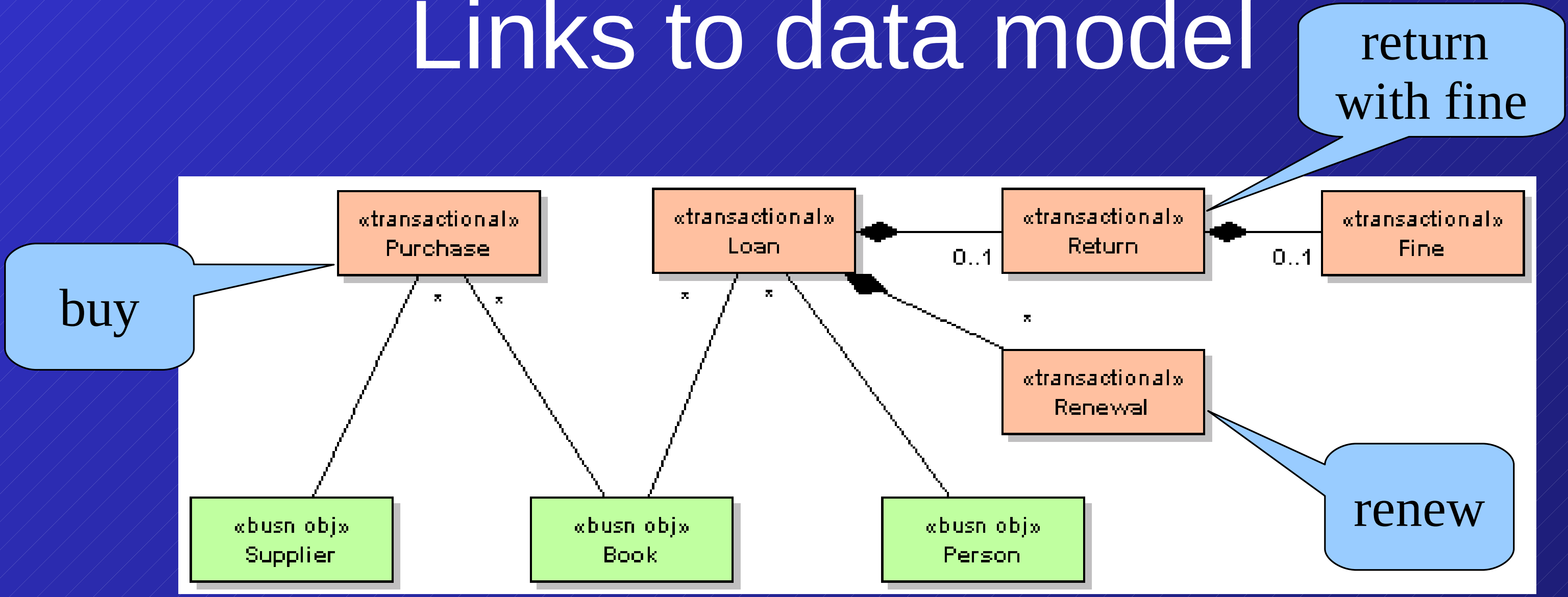
whole lifecycle in one process

use case or service

superstate

state

Links to data model



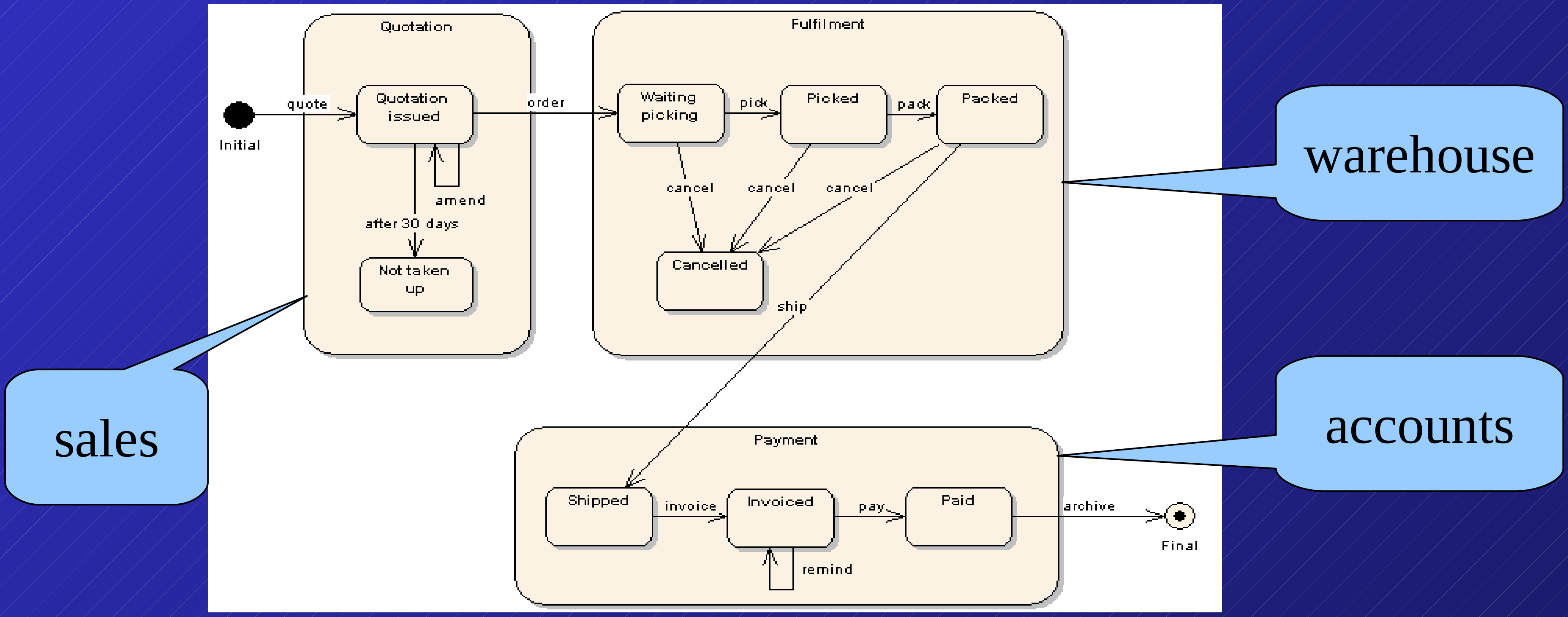
- Service/use cases have associated objects
  - Reporting, statements, audit, data mining, etc
- Some just create new “pink” objects
- Some also change existing “green” entities
  - e.g. update stock level

Major phases in processes

- Creation/setup, during operation, cleanup
  - Pensions: new business, servicing, drawdown
  - E-commerce: quotation to order, fulfilment, invoice to payment
  - Airport: before arrival, aircraft on stand, after departure
- ***Business*** transactions and contracts between phases
  - Often separate departments in a business
  - Handoff, passing of dossier/files (i.e. data flow)
- Business forms are pinks that request green information
  - “Office use only” sections are process-level pinks

Major phase examples

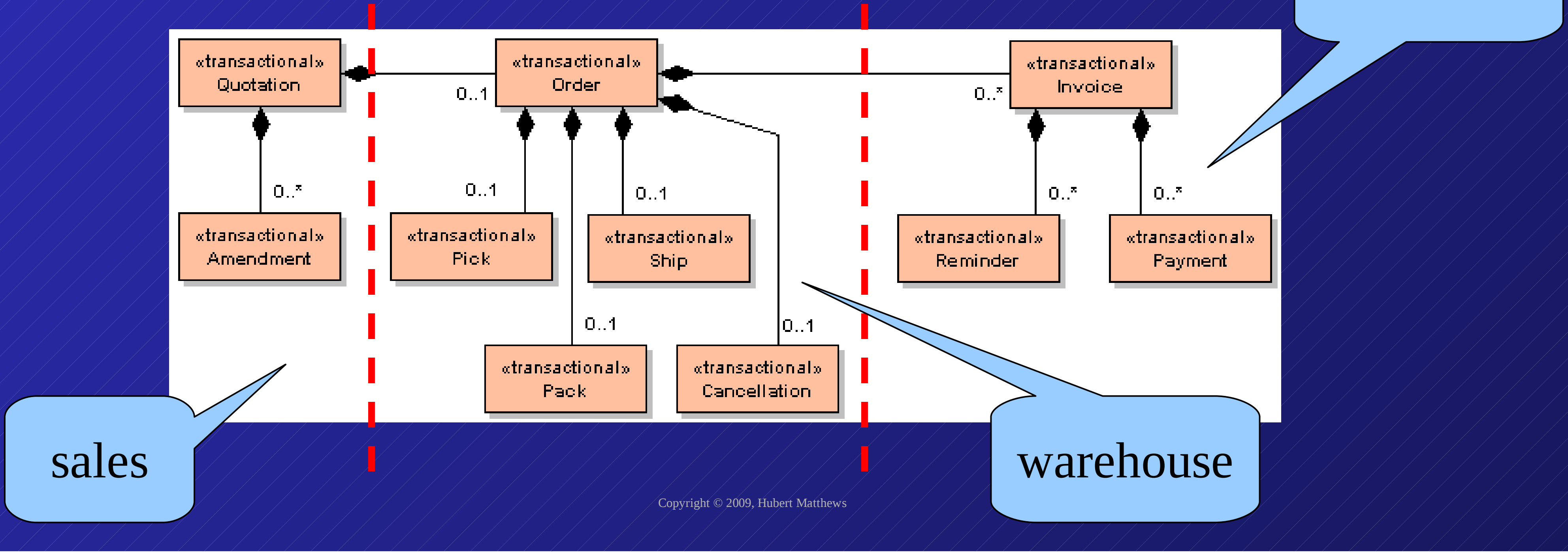
- Quotation->order, pick/pack/ship, invoice->pay



- Departmental boundaries, separate systems
- Real-world contracts at handoffs
- Source of much integration work! (“Customer” everywhere but may be different -> roles!)

Major phases and data model

- Each phase has a new top-level pink
  - Quotation, order, invoice
- Relationship across time is 1:0..1 or 1:0..*
- Lots of conditional links because things may not have happened yet



accounts

sales

warehouse

Events and “pinks”

- State machine is effectively a parser for incoming events (services/use cases)
  - Enforces ordering of business process events
  - A regular expression parser
- Jackson System Development (JSD)
  - Has entity lifecycles that describe this grammar
  - No direct links to data model, however
  - (Previous set of linked pinks is an OO JSD tree)

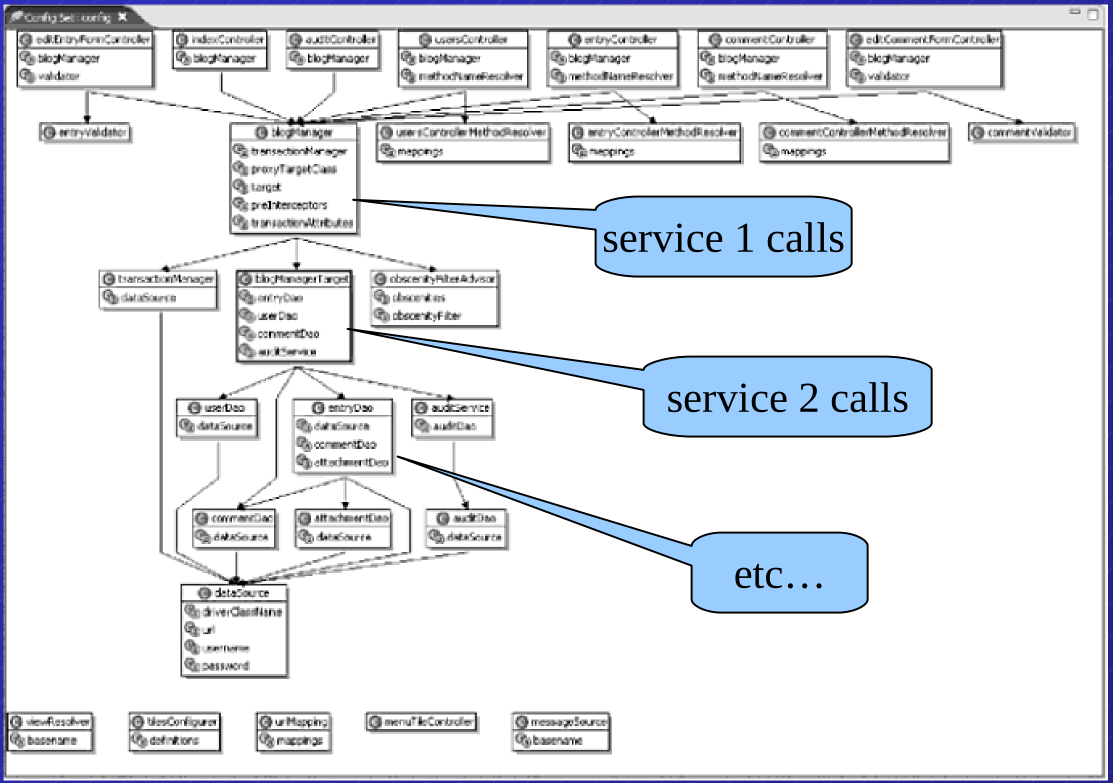
Layered systems

- Classic three-tier architecture
  - Presentation, “business logic”, data/persistence
- Everything up to now is in the data layer
- Middle layer not well understood
  - What does it do to what?
- Controllers (pieces of code) publish services that manipulate pinks (and greens)
  - Enforce process statecharts and business rules

Business rules

- Most rules are about whether a pink transaction object can be created or modified
  - Can person X borrow book Y?
- Some are read-only (access control)
  - Can person A look at bank account B?
- Implemented in controllers in middle layer
- Conceptually, controllers have a list of all possible new pinks, i.e. all allowed actions
  - May also be implemented by role objects
- *Rules are important and often overlooked*

Service-oriented architecture



service 1 calls

service 2 calls

etc…

Example of Spring dependency graph showing inter-component (i.e. service) connections

- SOA exposes middle layer
- Requires layering of services to enforce rules
- c.f. Spring’s external “wiring” of components
- *Too often people think SOA is flat and forget rules*

SOA (2)

- Archetypes help distinguish process-specific services for pinks from CRUD services for greens
- Example: Create a purchase order
  - Simple base service just creates a pink
  - Huge number of rules: budgets, preferred suppliers, approved items, payment terms, etc
  - Layered services enforce rules and manipulate pinks/greens in data layer
- Web services deal with processes and rules (verbs)
- RESTful services deal with data and often omit rules
  - CRUD access to nouns (mostly “greens”)

ESB

- Content-based routing
  - “Pink” flows through system
  - Process statechart implemented in parts by individual systems (major phases)
  - Federated collaborative approach
- Orchestration
  - Centralised management of process statechart
  - “Big box in middle” approach
- Data duplication – keeping “greens” up to date
- Similar to data-flow diagrams

BMUF (big modelling up front)?

- Lightweight models – not even attributes/fields
- Used for thinking, describing, analysing and structuring systems
  - Not used for code generation
- Agile
  - (not Scott Ambler’s “agile modelling”)

History lesson (Part 2)

- Approaches that fit this style
  - Yourdon and Schlaer-Mellor – both have objects and states but don't link the two (and no pretty colours!)
  - Jackson System Development – very close, no direct link
- Colours help a lot
  - Names for archetypes are useful, pattern names
  - Modelling rules give quick check on multiplicities, etc
  - Inspired by Coad's Modelling in Colour
- Catalysis 1 had most of this but without colours and wasn't particularly approachable
- Approach shown here is much easier and based on Catalysis 2 (shameless plug….)

Summary

- Joined-up modelling is both possible and necessary
  - Better requirements capture, easier implementation
- Agile models lead to better architectures
  - Separation of different archetypes/colours
- Transactional objects (“pinks”) are the key
- Most people focus unduly on data model but not on pinks
  - *Insufficient attention paid to process and rules*
- Lightweight models aid thinking and structure
  - Heavyweight models and code generation don’t!