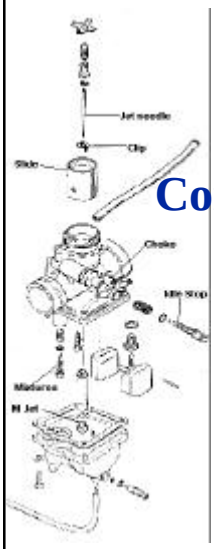
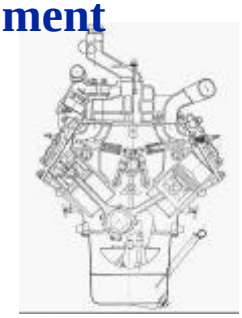


Modeling for Component Based Development with Catalysis

Alan Cameron Wills



TriReme
Component technology

http://www.trireme.com

The presenter

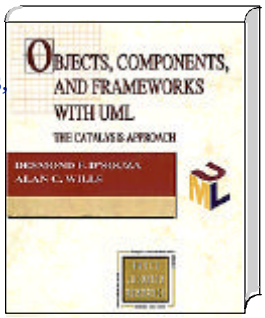
- Alan Cameron Wills
  - working in objects since 1986, software since 1972
  - joint developer of Catalysis design method
  - frequent presenter and contributor to development of the field
  - worked with clients in a variety of fields in Europe and US
  - director of TriReme
    - consultancy, mentoring, training in
      - design methods
      - ob ject and component technology
      - migration strategy

Catalysis

- Development of reusable designs and components
- UML notation
  - Component Based Development
  - Reuse of models & designs
  - Context-flexible process patterns
  - Traceable development
  - Scalable
  - Coherence between models

http://www.trireme.com/catalysis
http://www.catalysis.org/

- Users — BankAmerica, Loral, BNR, Sterling, Siemens, GPT, Eurofighter, …
- Recommendations: Gartner, Butler CBD
- Tools: Sterling COOL, Platinum, ...
- Book: ACW & Desmond D'Souza
  - www.amazon.com & all good bookshops



Addison-Wesley 0-201-31012-0

Agenda

- Components
  - why, what, how
- Catalysis techniques
  - meeting methods requirements of CBD

Component Based Development with UML/Catalysis

Alan Cameron Wills

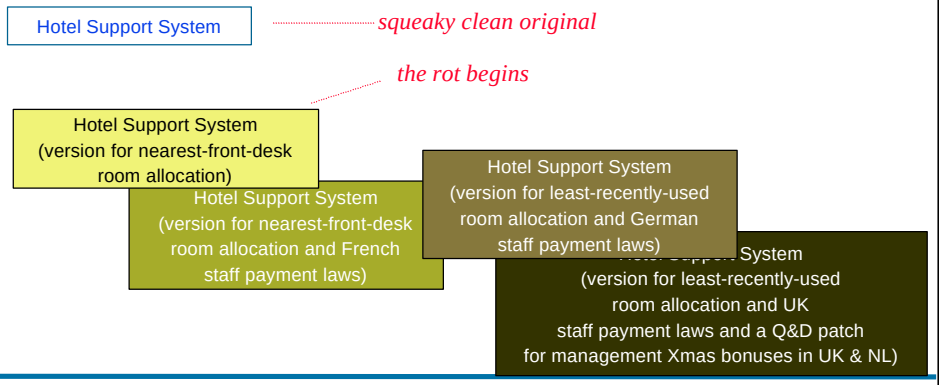
1: What is Component Based Development?

TriReme Object technology

http://www.trireme.com

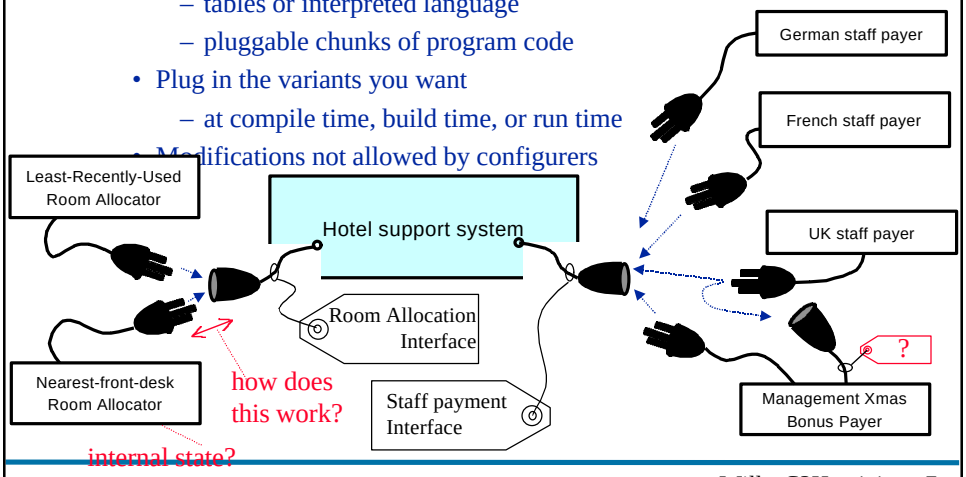
How not to get Flexibility

- Modify the software
  - problem: divergent versions –> rising maintenance costs
  - problem: mods inconsistent with original clean architecture –> increasing wartiness & interdependency of parts
  - software degenerates over time



How to get flexibility

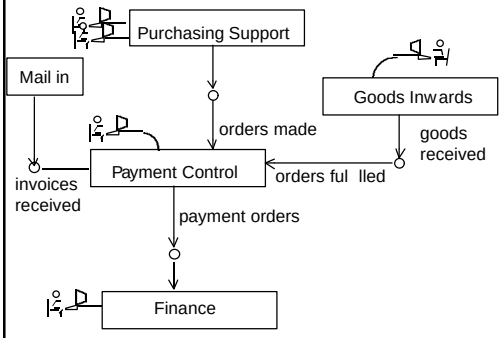
- Build with pluggable parts
  - Separate variable behaviour into separate parts
    - parameters
    - tables or interpreted language
    - pluggable chunks of program code
  - Plug in the variants you want
    - at compile time, build time, or run time
  - Modifications not allowed by configurers



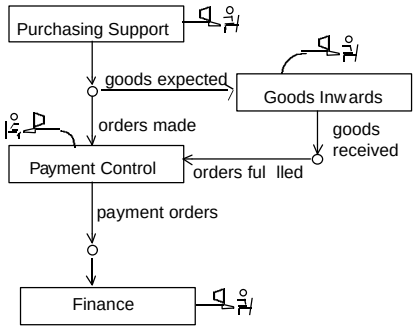
Federated architecture

- Software structure mirrors business
  - easier to keep software up to date with business
  - localised control of software support

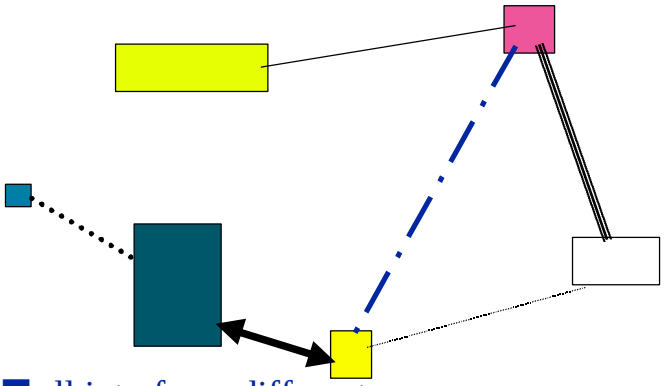
Before reorganisation



After reorganisation



Contrast typical large evolved setup



… 100's of subsystems & individually-crafted interfaces…

different:

•ways of representing business concepts

•protocols and media TCP, RMI, CORBA, card decks, ...

- all interfaces different
- ---> can't easily rearrange or substitute one standard module with another
- ---> no flexibility

not component

Where do we see components?

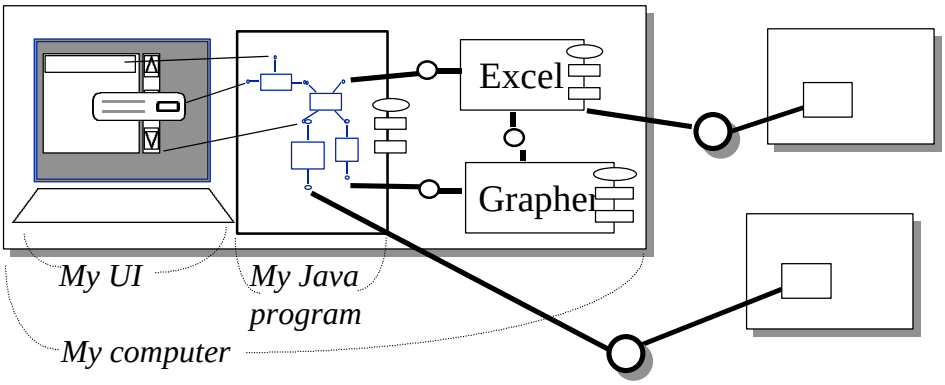
- different scales and boundaries ...

Java AWT

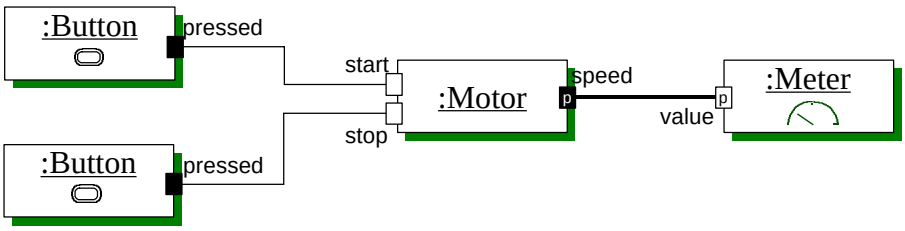
Java Beans, 'Parts'

COM, OLE, Unix pipes

CORBA, DCOM, Java RMT

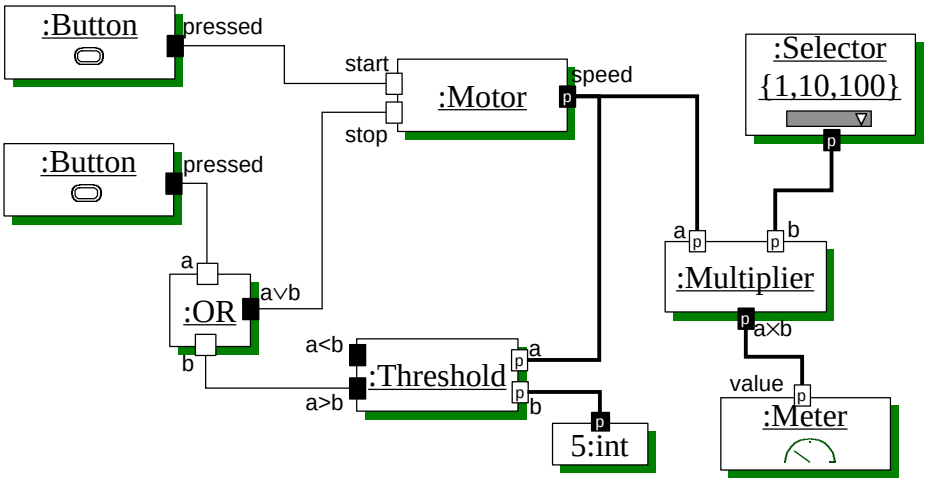


Component interfaces



- Important properties of component interfaces:
  - each interface general enough to be plugged with many others
  - each component well-specified enough to be able to use without looking inside it
- (Notation: UML-RT)

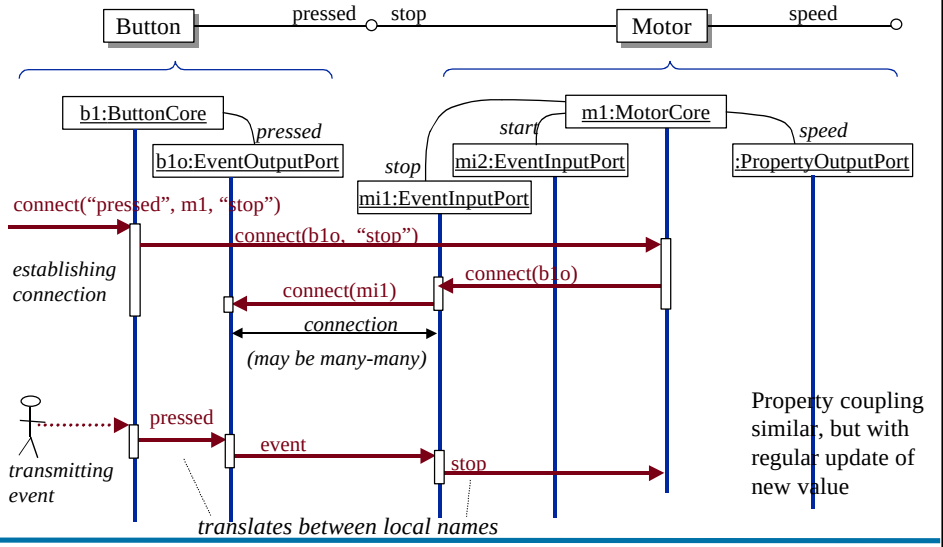
Plug in more bits from the kit...



- Visual tools make plugging these parts easy
  - E.g. Digitalk PARTS, VisualAge, Java Beans DK

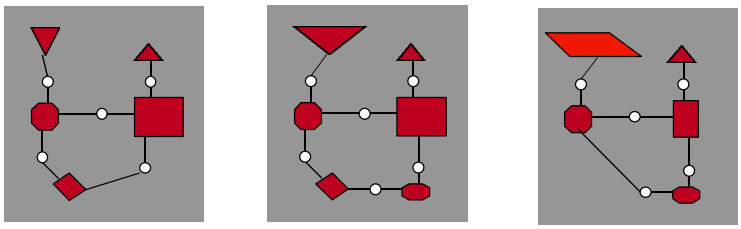
Connector abstracts protocol

- One possible implementation of these connectors:



Components come in kits

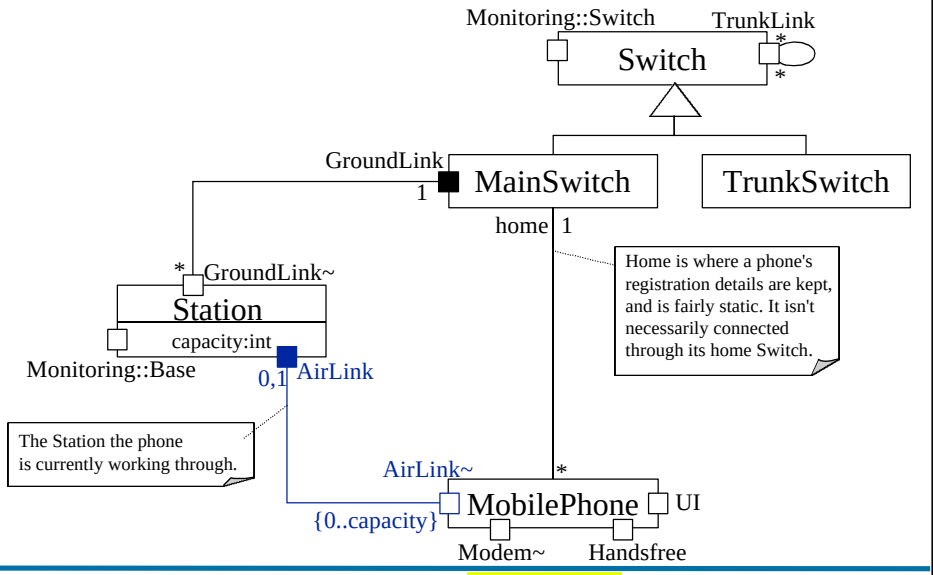
- **Families of products**
  made from **kits of components**



- **Component kit architecture = definition of interfaces (including models of what they talk about)**

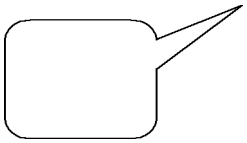
Component = a part [conforming to the interface standards] of a kit

Component assembly example



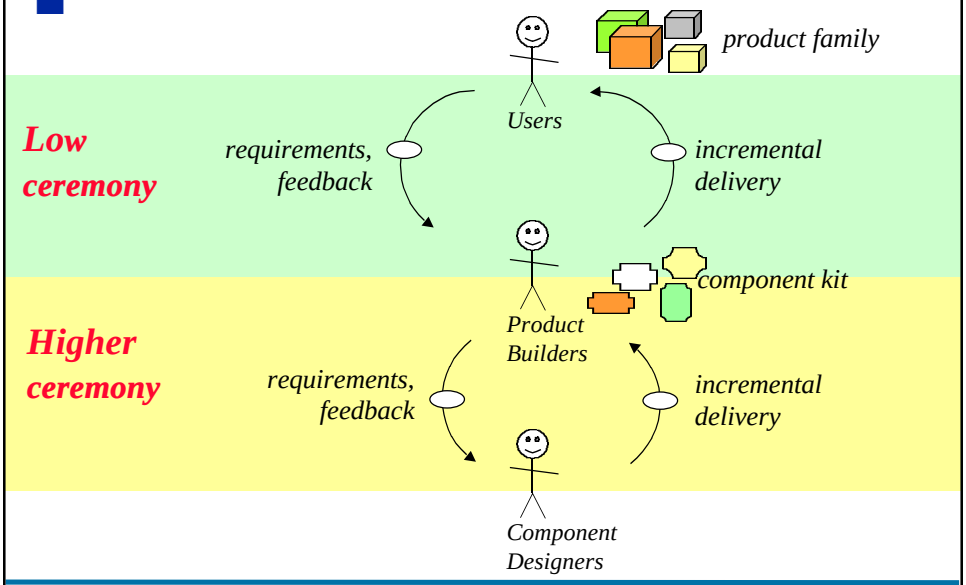
Modelling for CBD

- Specify and design separately:
  - each class of component
  - each class of connector
    - define message protocols



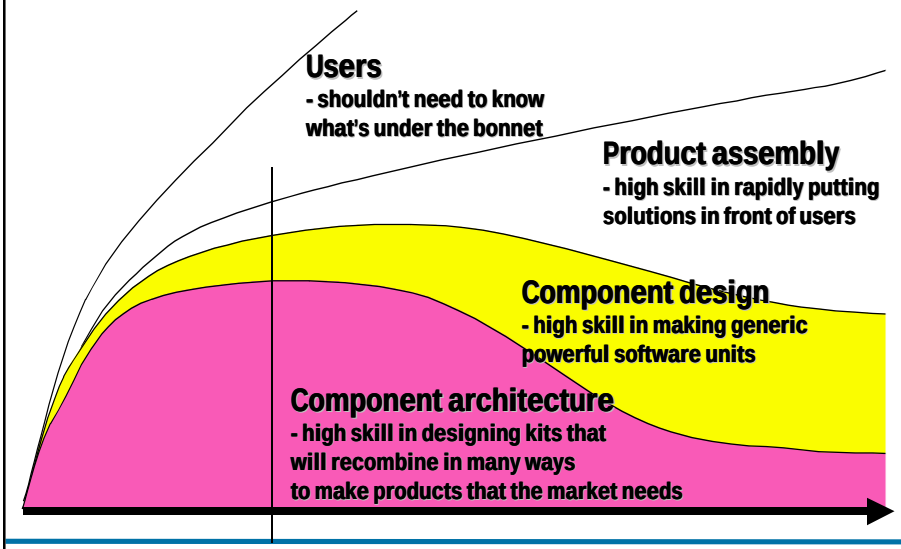
    - define common models

RAD loops



Drivers, mechanics, engineers, & architects

- Separation of skills as industry matures



Choosing component boundaries

- Criteria for deciding component separation:
  - Separation of concerns --- separation of variable features
    - Work out what varies from one installation to another
  - Support local to business users, hardware, other external agents
    - Operational/mobile/on holiday when they are
  - Bandwidth --- keep info close to where it's used
    - Caching/replication/reduncancy appropriate? Needs synchronisation.
  - Continuity of service --- reduce the chance of everything down; --- keep a number of independent units --> portable, etc;
    - ==> caching, replication
  - Openness --- ability for components to cope with more types as they come along
    - ==> reflection ==> extensible representation (tag->value etc)
  - Existing assets

Agenda

- Components
  - why, what, how
- Catalysis techniques
  - meeting methods requirements of CBD

Abstract actions & objects
Specifying components
Modeling connectors
Conformance of components to business model
Model templates

Catalysis = UML ++

- OO analysis and design

4 Components Component Based Development
  - rapid development of families of products from component kits

4 Reuse Reuse of models & designs
  - frameworks and patterns

4 Precision Unambiguous specification

4 Abstraction
  - for high-integrity design and early exposure of important issues

4 Traceability Traceability: business model → specifications → code
  - maintainability, strong quality assurance

4 Scalability Scalable through precise abstraction
  - treat large components and transactions as single objects

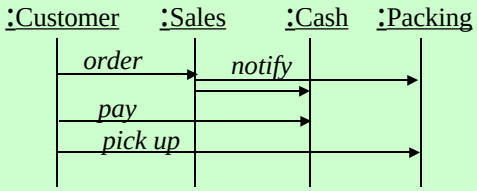
4 Coherence Coherence between UML models
  - strong cross-checking helps consistency and completeness

4 Process Process patterns
  - combine appropriate techniques to cover many situations

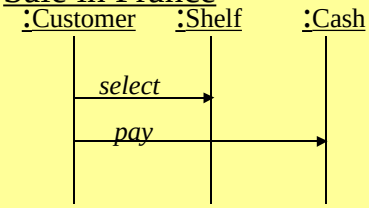
Use cases

- Descriptions of sequences achieving a defined goal

Sale in Sweden

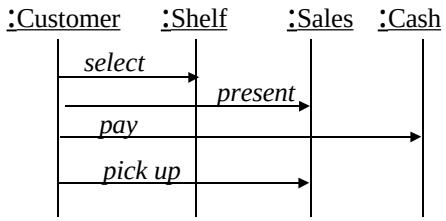


Sale in France



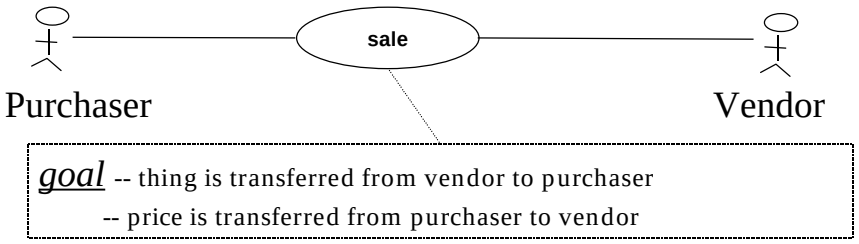
- Different sequences may achieve same goal --- which is more important

Sale in UK



Action

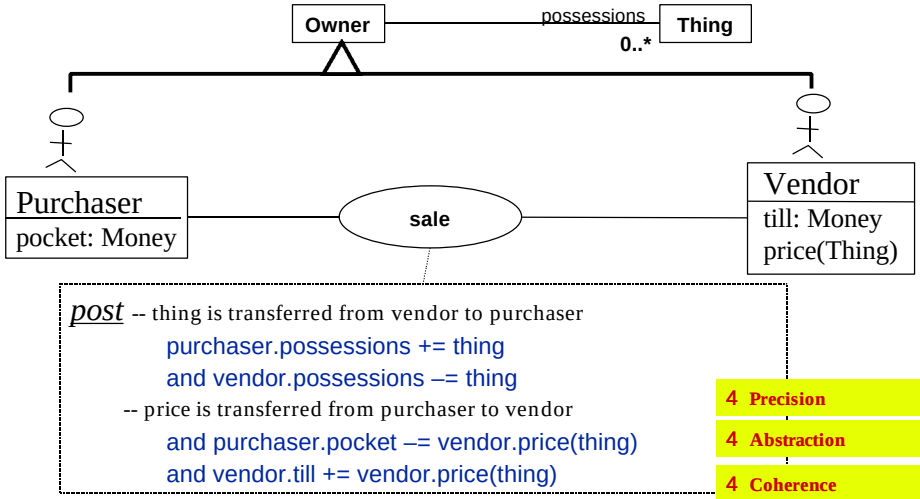
- Action = use-case|activity|task|operation|transaction|…
- Goals more general than steps that achieve them
  - and important to document anyway



- Actions are about interactions/transactions between people, departments, software or hardware components

Action generalises use-case

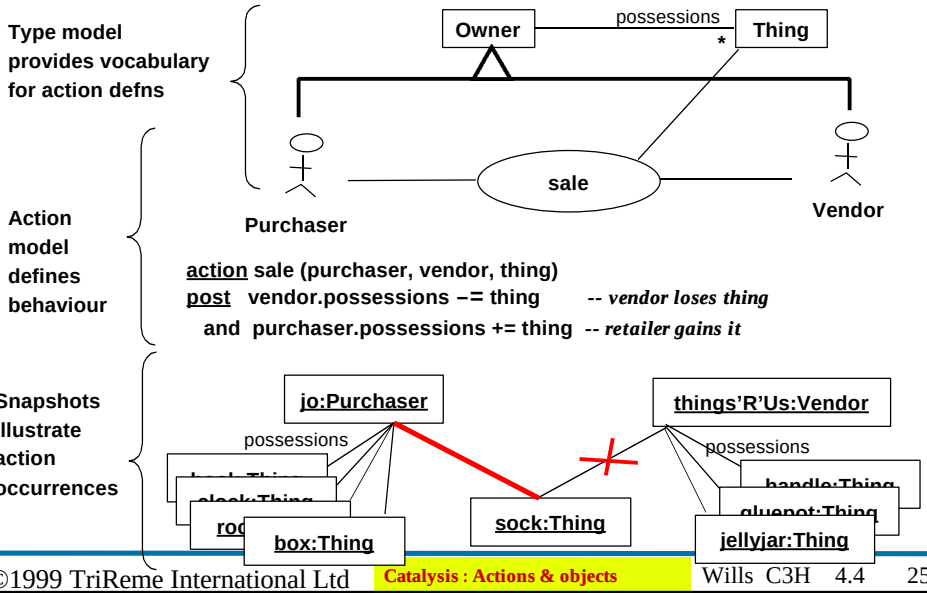
- Associations and attributes of the types provide the vocabulary in which to express what the action achieves



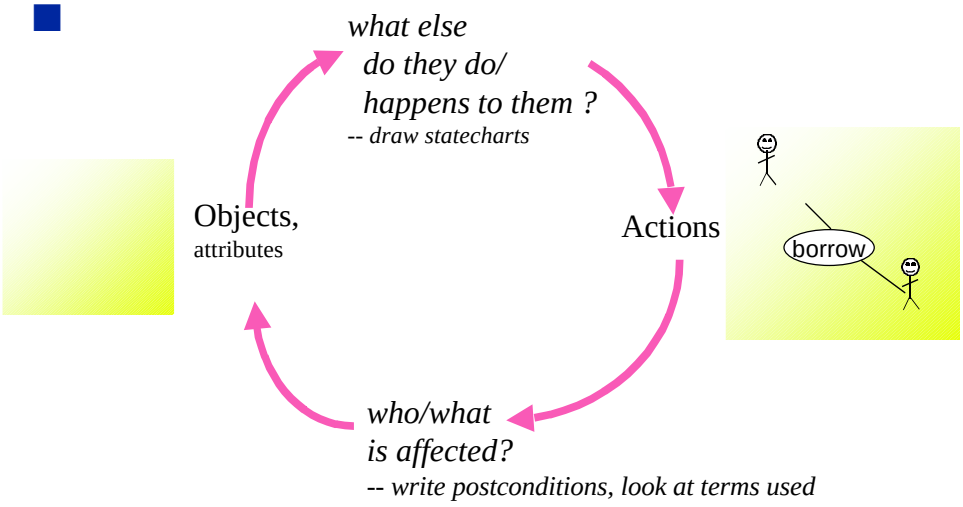
4 Precision
4 Abstraction
4 Coherence

High Integrity Design: Actions

- Actions represent definable changes of state in actors

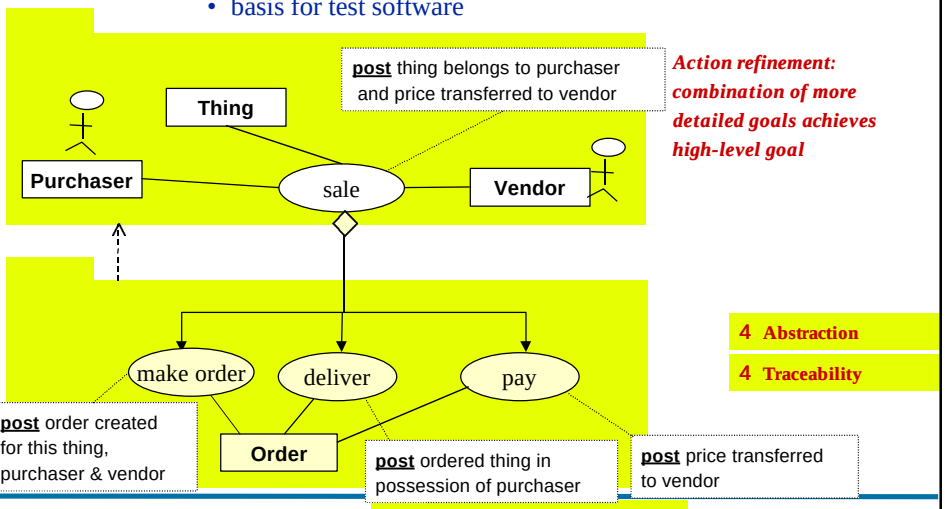


Business Model construction



High Integrity Design: Refinement

- Traceable relationship between levels of detail
  - can (optionally) be written precisely using OCL or Java
    - basis for test software

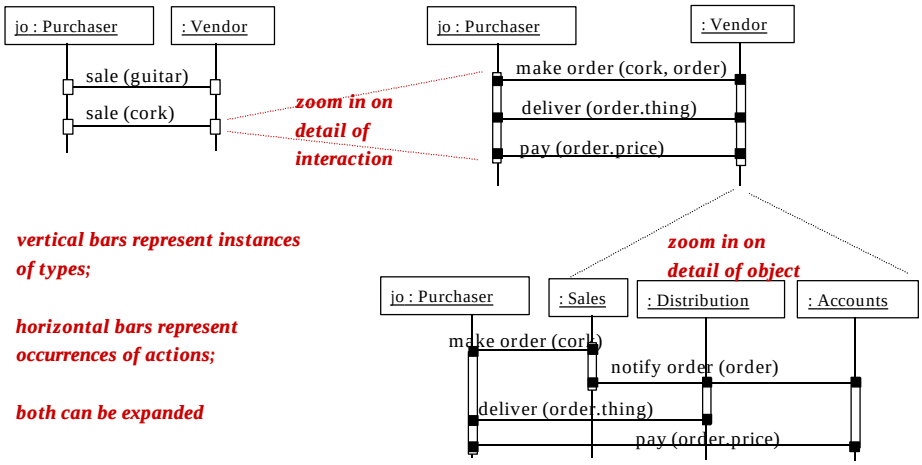


Action refinement: combination of more detailed goals achieves high-level goal

4 Abstraction
4 Traceability

High Integrity Design: Refinement

- Traceability from business goals through software design
- Large objects and actions treated same as small ones



vertical bars represent instances of types;

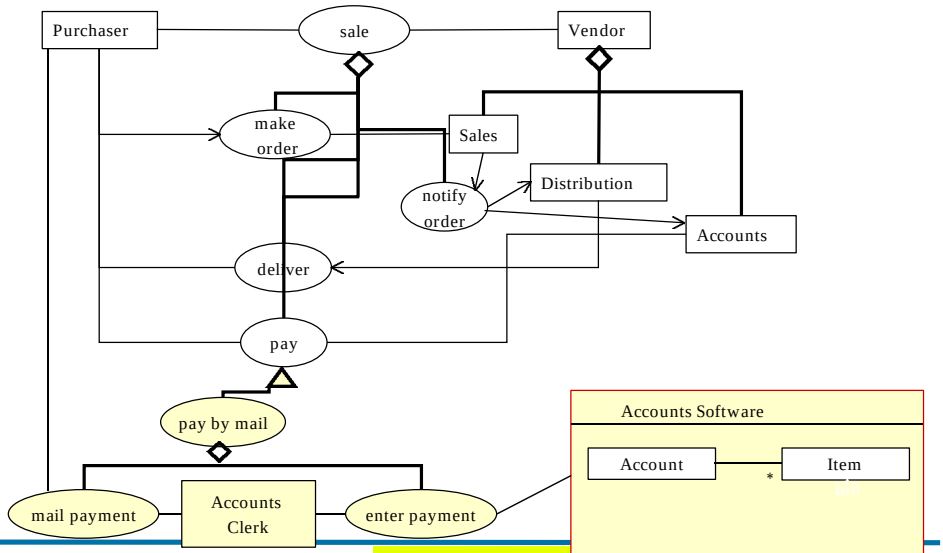
horizontal bars represent occurrences of actions;

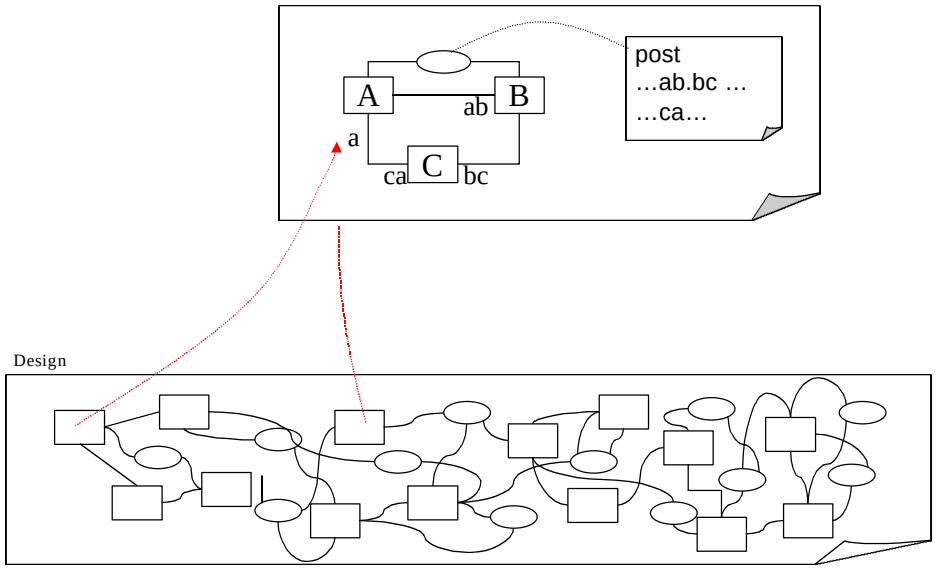
both can be expanded

4 Traceability 4 Scalability Catalysis : Actions & objects Wills C3H 4.4 28

Object & action Refinement

- Traceability from business goals through software design





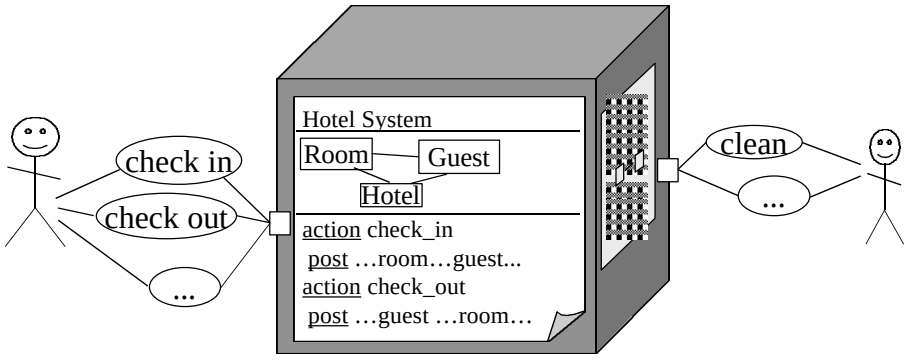
Agenda

- Components
  - why, what, how
- Catalysis techniques
  - meeting methods requirements of CBD

Abstract actions & objects
Specifying components
Modeling connectors
Conformance of components to business model
Model templates

Specifying the black box

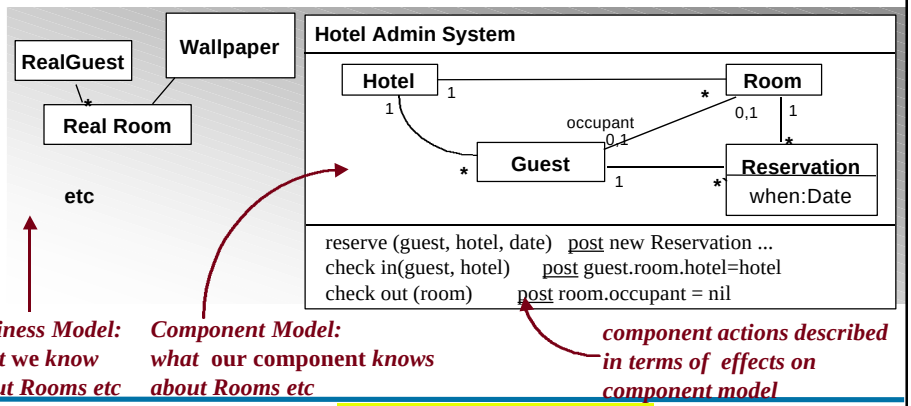
- Spec is a label on the side of the box
  - doesn't tell you what's inside
  - does tell you what behaviour to expect



  - tells implementors what they have to aim for

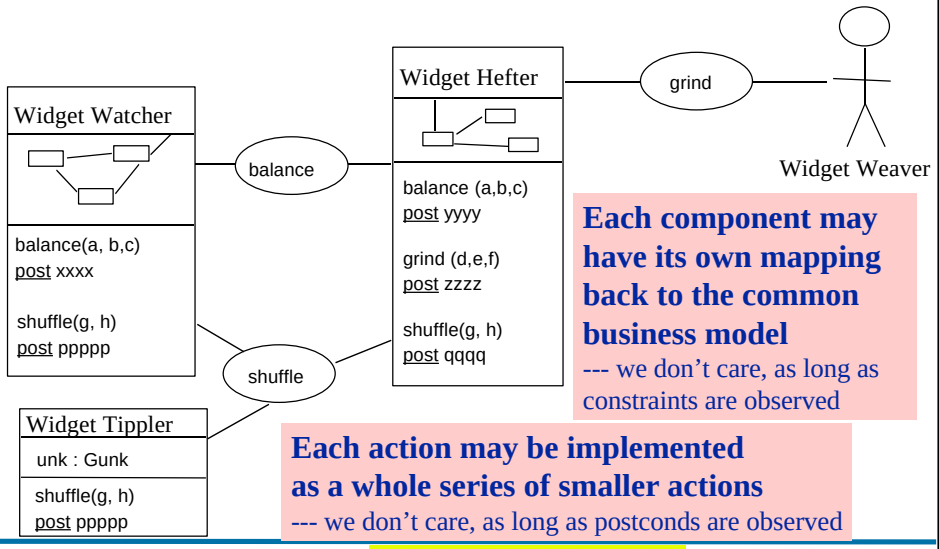
Business model => component model

- Business model was about world surrounding our system
- Component model is what our component knows about the world
  - just the stuff relevant to its responsibilities



Communicating components

- Components interact using biz concepts



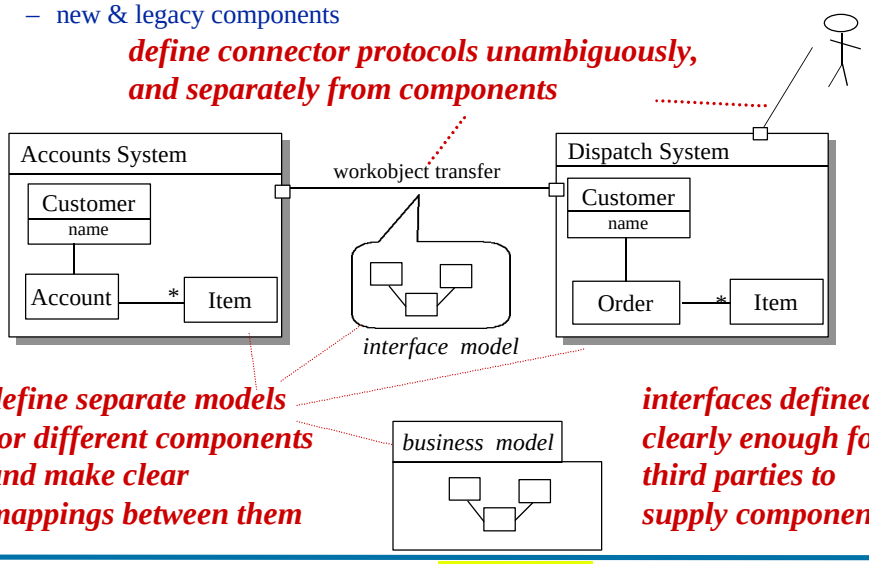
Each component may have its own mapping back to the common business model
--- we don't care, as long as constraints are observed

Each action may be implemented as a whole series of smaller actions
--- we don't care, as long as postconds are observed

Modeling for Cmpt Architectures

- Families of products generated from a component kit
  - new & legacy components

define connector protocols unambiguously, and separately from components



define separate models for different components and make clear mappings between them

interfaces defined clearly enough for third parties to supply components

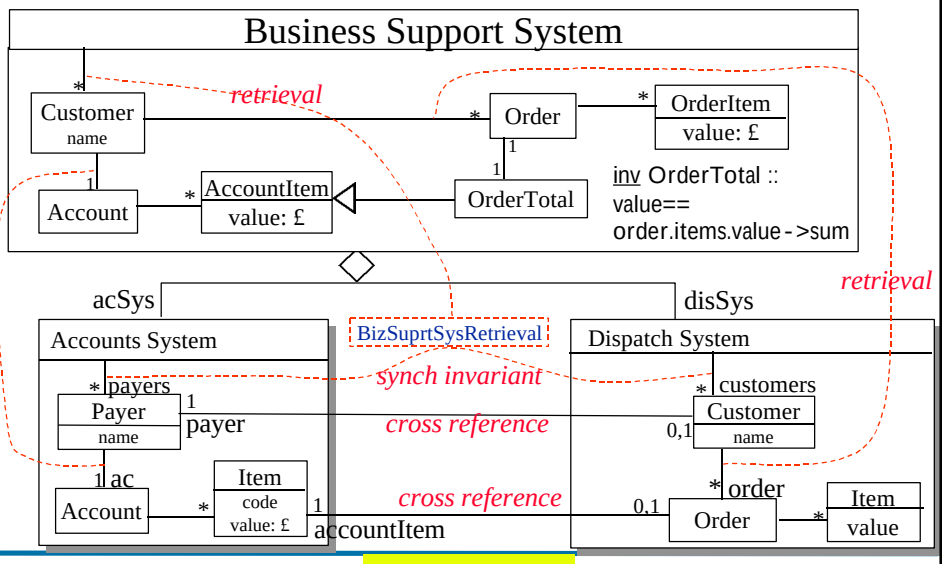
Agenda

- Components
  - why, what, how
- Catalysis techniques
  - meeting methods requirements of CBD

Abstract actions & objects
Specifying components
Modeling connectors
Conformance of components to business model
Model templates

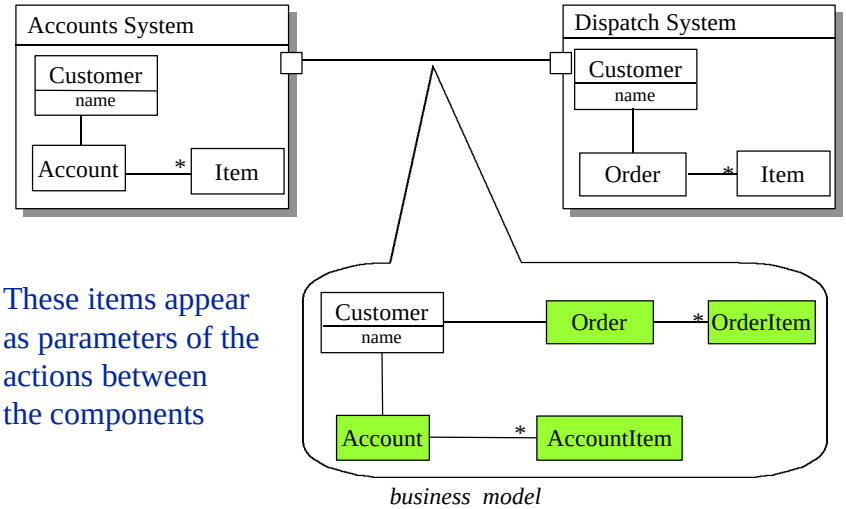
Mapping Component Models

- 'Retrieve' business model from components



Business model

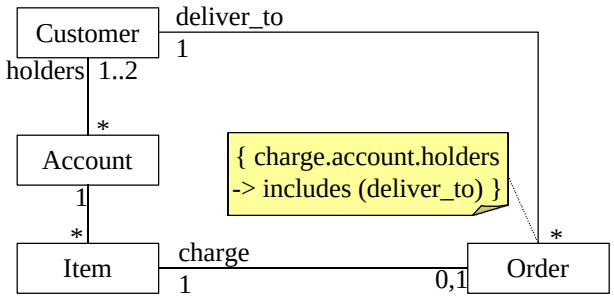
- Different components may have different internal models, but must speak a common language



- These items appear as parameters of the actions between the components

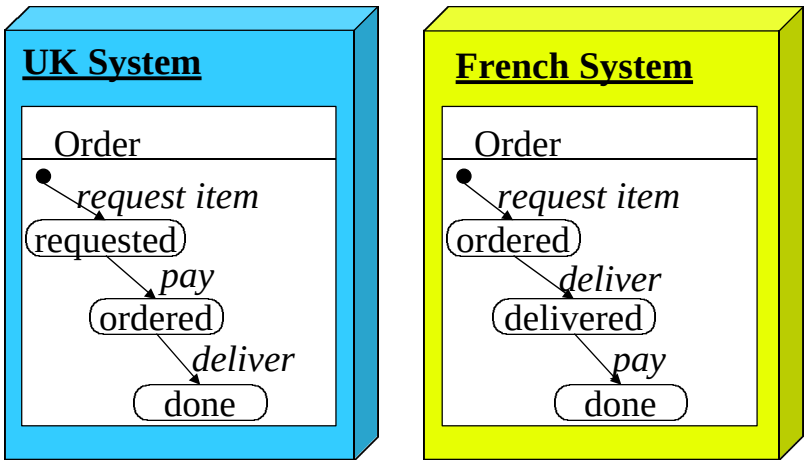
Business rules

- Each component should observe the rules about what's allowed
  - at least when communicating with other components
- Static constraints --- written informally or in OCL/C++/...
  - e.g. *an Order shall be charged to an Account of the same Customer to whom it is to be delivered*



Dynamic rules

- Rules about the allowed transitions



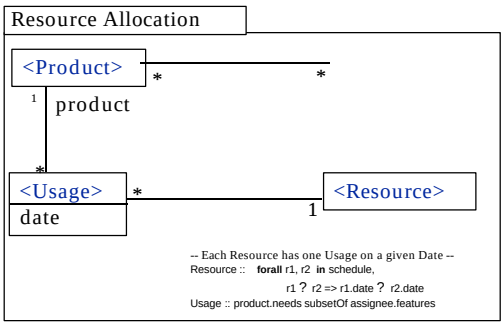
  - what happens if these systems pass orders to each other to fulfill?

Agenda

- Components
  - why, what, how
- Catalysis techniques
  - meeting methods requirements of CBD

Abstract actions & objects
Specifying components
Modeling connectors
Conformance of components to business model
Model templates

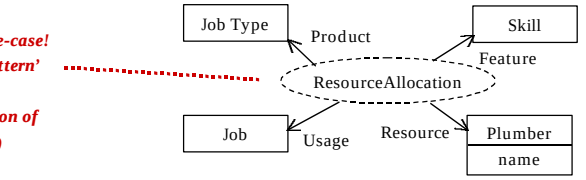
Templates



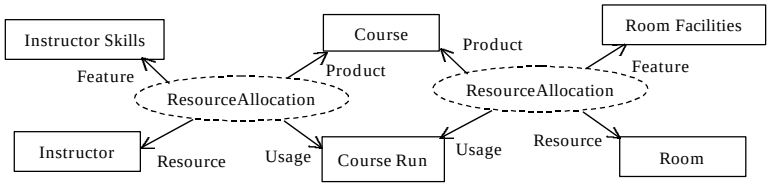
Template application

Pictorial notation

(not a use-case! UML 'pattern' indicates application of template)

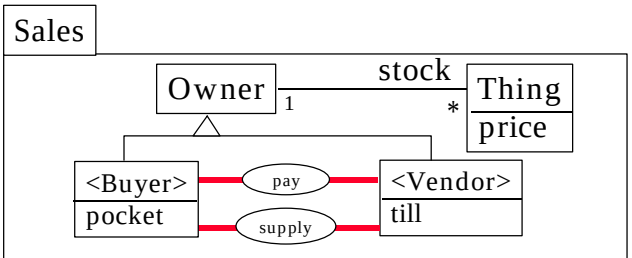


Quoting template adds macro-instantiated attributes etc to existing types



Multiple application

Collaborations

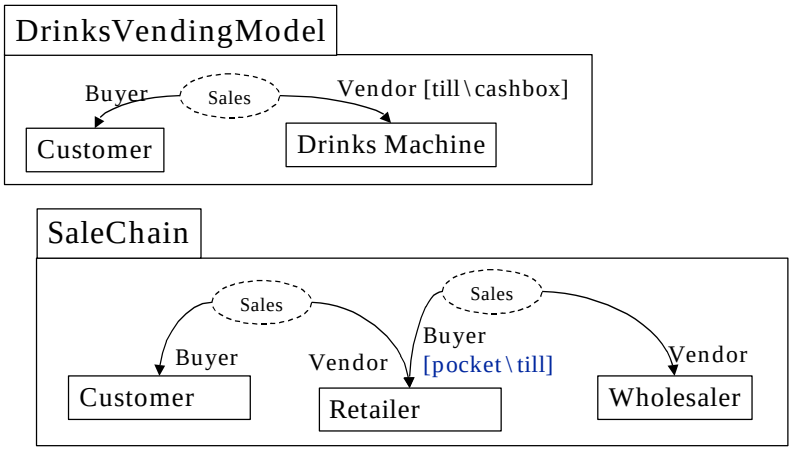


action pay (buyer, vendor, amount)
pre buyer.pocket > amount
post buyer.pocket –= amount
and vendor.till += amount

action supply (vendor, buyer, thing)
pre vendor.stock **includes** thing
post vendor.stock –= thing
and buyer.stock += thing

Instantiating Generic Models

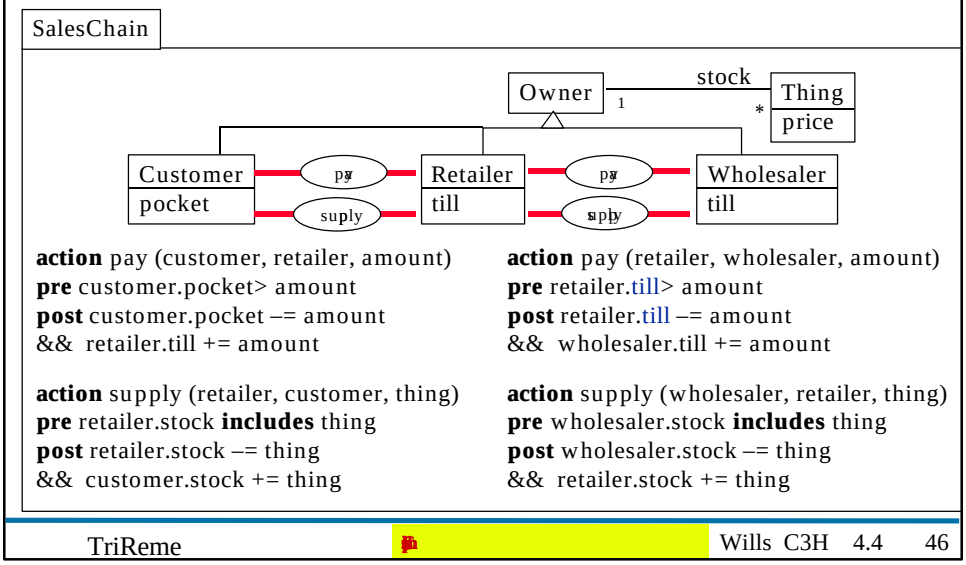
- Templates provide protocols



4 Abstraction

4 Components

Unfolded composite



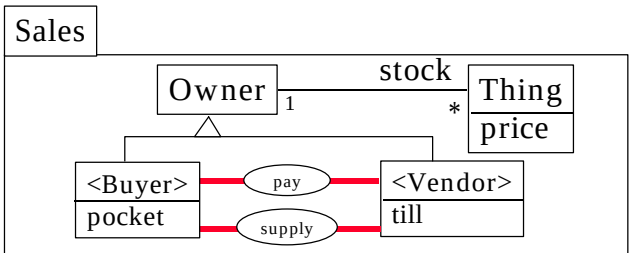
action pay (customer, retailer, amount)
pre customer.pocket> amount
post customer.pocket –= amount
&& retailer.till += amount

action pay (retailer, wholesaler, amount)
pre retailer.till> amount
post retailer.till –= amount
&& wholesaler.till += amount

action supply (retailer, customer, thing)
pre retailer.stock **includes** thing
post retailer.stock –= thing
&& customer.stock += thing

action supply (wholesaler, retailer, thing)
pre wholesaler.stock **includes** thing
post wholesaler.stock –= thing
&& retailer.stock += thing

Effects: dynamic business rules



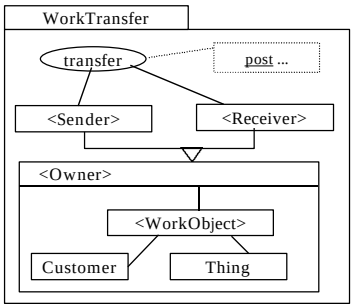
Effect:
An additional clause on the postcondition of every other action affecting a Buyer --- even from other templates.
<trigger>===><reqmnt>

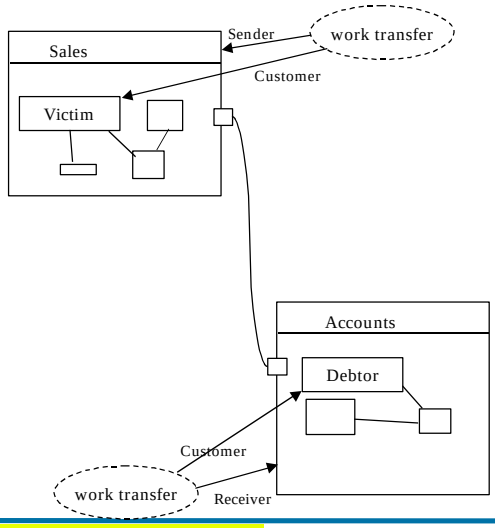
action supply (vendor, buyer, thing)
…

Buyer:: **effect** *restock* (thing:Thing)
stock[thing].size < 10 ==>
thereis v:Vendor, [[supply(v,this, thing)]]

Connector definition

- Model in connector framework mapped to implementing component's model





allows the different components to have different models, and to have much bigger models than the connector framework is concerned with

4 Components

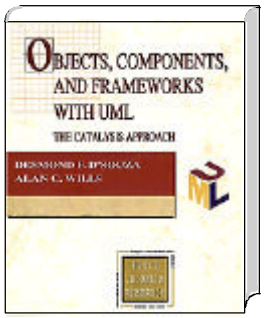
Catalysis: strengths

- Component Based Development
  - powerful interface abstractions, strongly specified
  - mapping between differing component models
- High IntegrityDesign
  - robust, reliable software
  - rigorous specifications define test harnesses
  - clear semantics, coherence between UML models
  - traceable refinements
- OO Analysis & Design
  - precise models expose gaps & inconsistencies early
  - abstractions good for whiteboard discussion and documentation
  - fractal method: same techniques at all levels

4 Components
4 Abstraction
4 Traceability
4 High Integrity
4 Precision
4 Coherence
4 Traceability
4 Precision
4 Abstraction
4 Process

www.trireme.com

Catalysis



Addison-Wesley 0-201-31012-0

TriReme
Component technology

http://www.trireme.com