



lecture 4: sequence & use case diagrams

csc302h
winter 2014

your department turns 50 this year!



check out this article in UofT Magazine:

<http://www.magazine.utoronto.ca/time-capsule/paving-the-way-for-the-information-highway-calvin-gotlieb-j-n-patterson-hume-beatrice-worsley/>



- reverse-engineering models from software & design discovery
- software evolution
 - (Lehman) program types
 - S/P/E-type: only really care about E-type (embedded) when discussing software evolution
 - laws of software evolution (also Lehman)
- cost of software aging. ways to improve longevity. reducing maintenance costs for each type of development (recall pie chart)

- how tools can help
 - code browsing
 - refactoring (for greater clarity)
 - documentation
 - design discovery (uml model generation)
- what tools can't do
 - tell you what the developer was thinking
 - make a bad developer good



sequence diagrams



Modeling Software Behaviour

- (briefly: making UML abstractions...)
- **UML sequence Diagrams**
- **Comparing Traces**
- **Explaining Design Patterns**
- **Style tips**



Uses of UML

As a sketch

Very selective - informal and dynamic

Forward engineering: describe some concept you need to implement

Reverse engineering: explain how some part of the program works

As a blueprint

Emphasis on completeness

Forward engineering: model as a detailed spec for the programmer

Reverse engineering: model as a code browser

Roundtrip: tools provide both forward and reverse engineering to move back and forth between program and code

As a Programming language

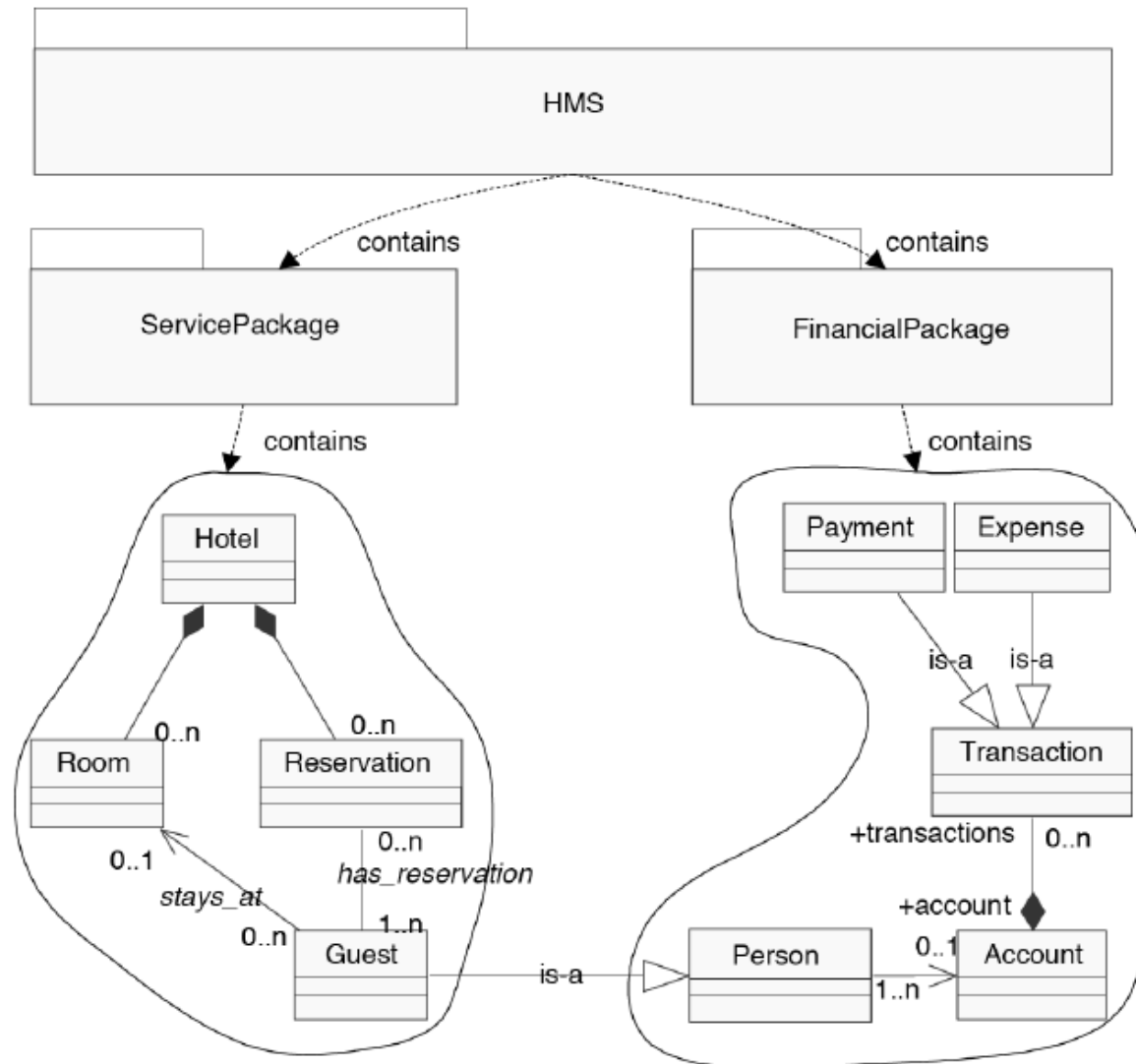
UML models are automatically compiled into working code

Requires sophisticated tools

“tripless” - the model is the code



Package Decomposition

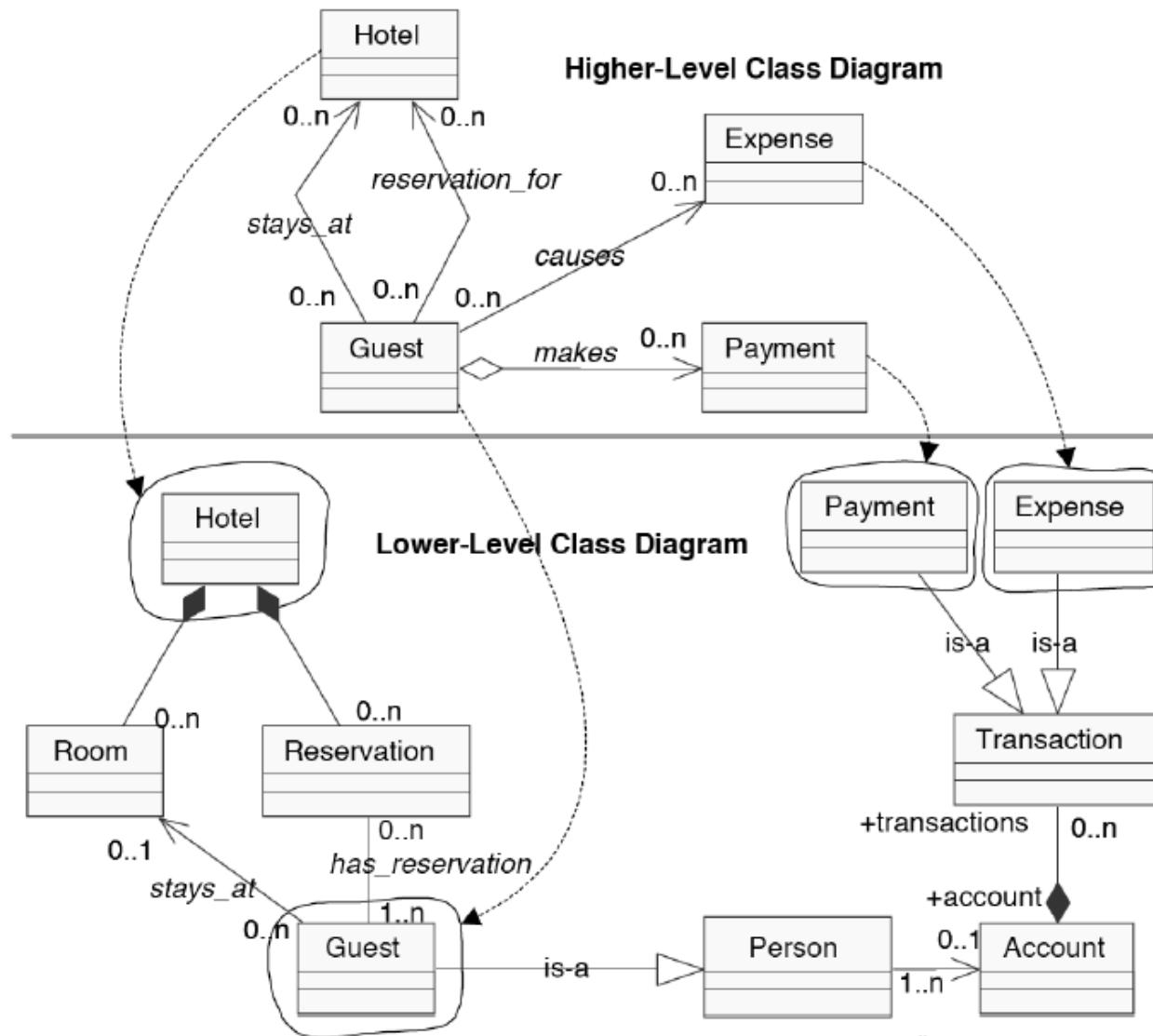


Source: from Egged "Automated Abstraction of Class Diagrams, TSE 2002"





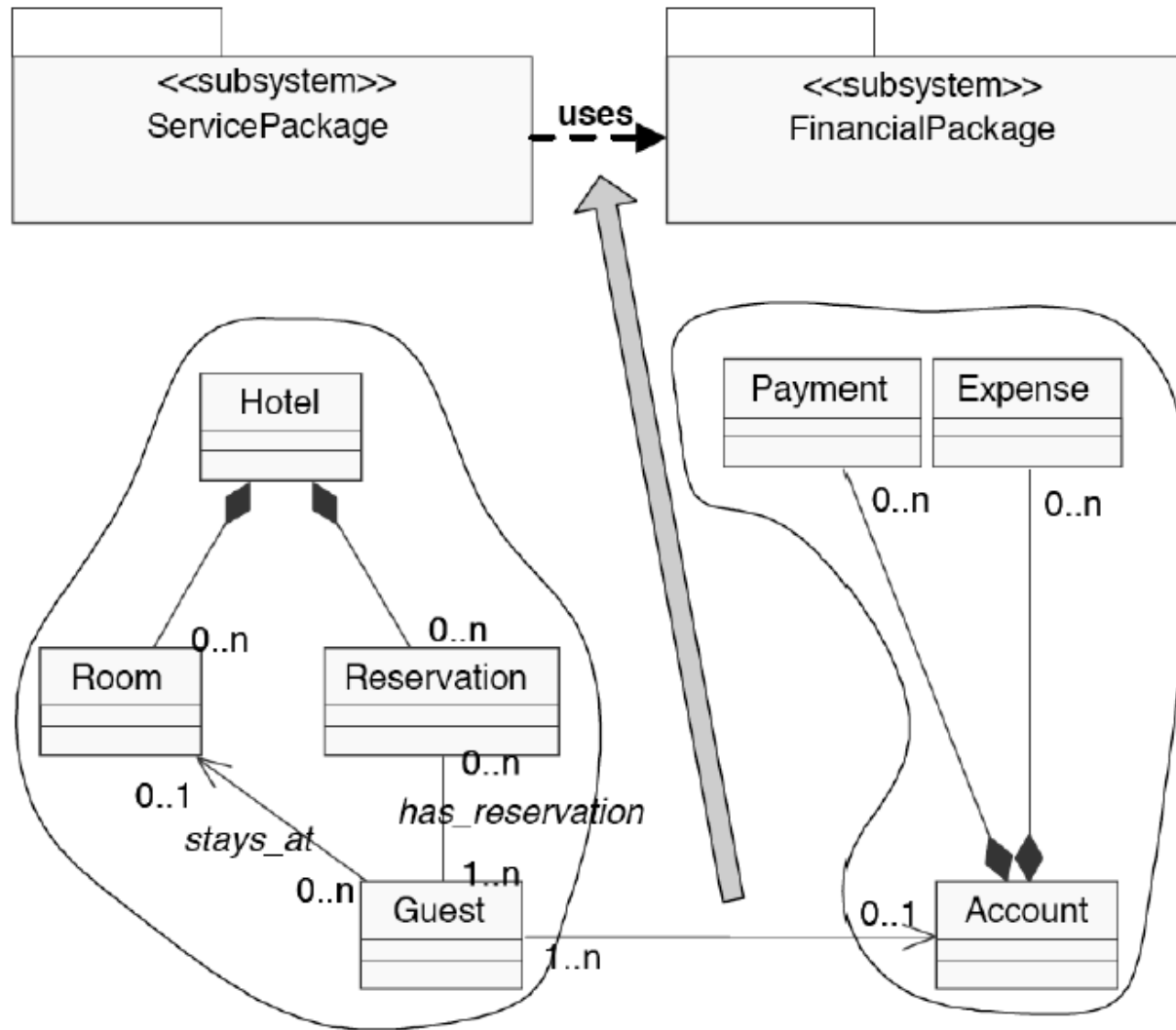
Class Abstraction



Source: from Eged *Automated Abstraction of Class Diagrams*, TSE 2002



Finding Dependencies



Source: from Egyed "Automated Abstraction of Class Diagrams, TSE 2002"





Things to Model

E.g. Structure of the code

Code Dependencies

Components and couplings

E.g. Behaviour of the code

Execution traces

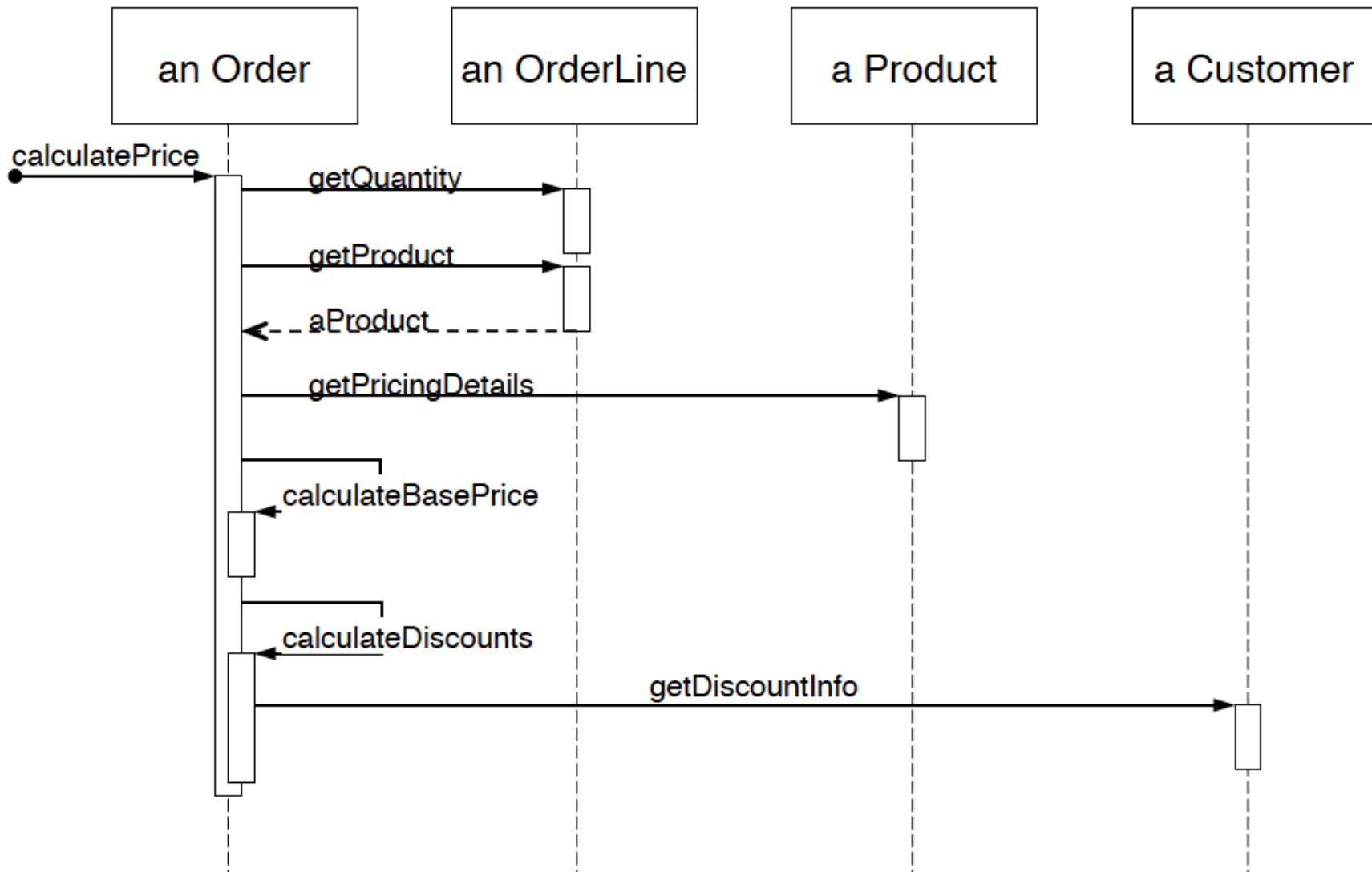
State machines models of complex objects

E.g. Function of the code

What functions does it provide to the user?

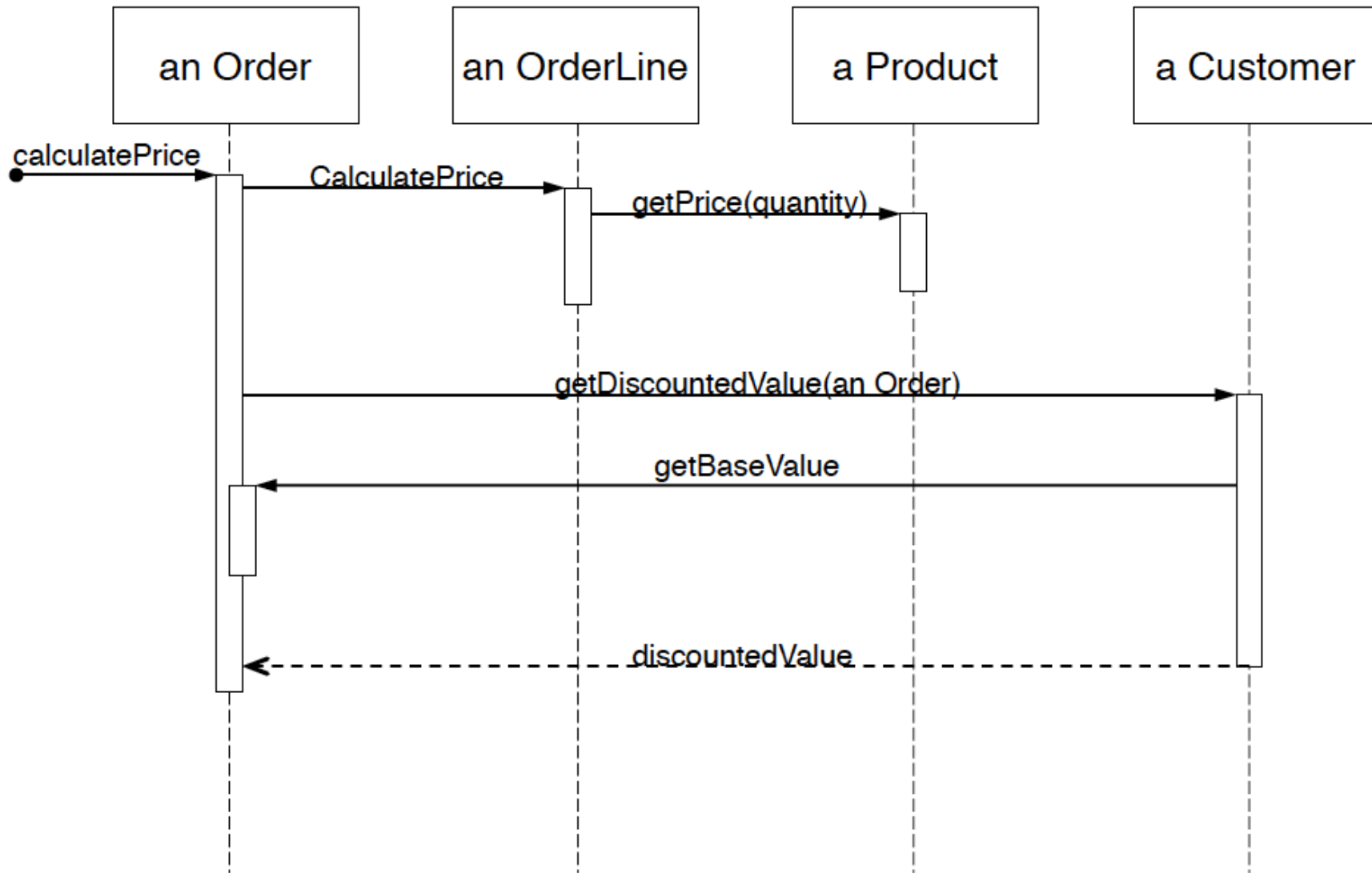


Sequence Diagrams



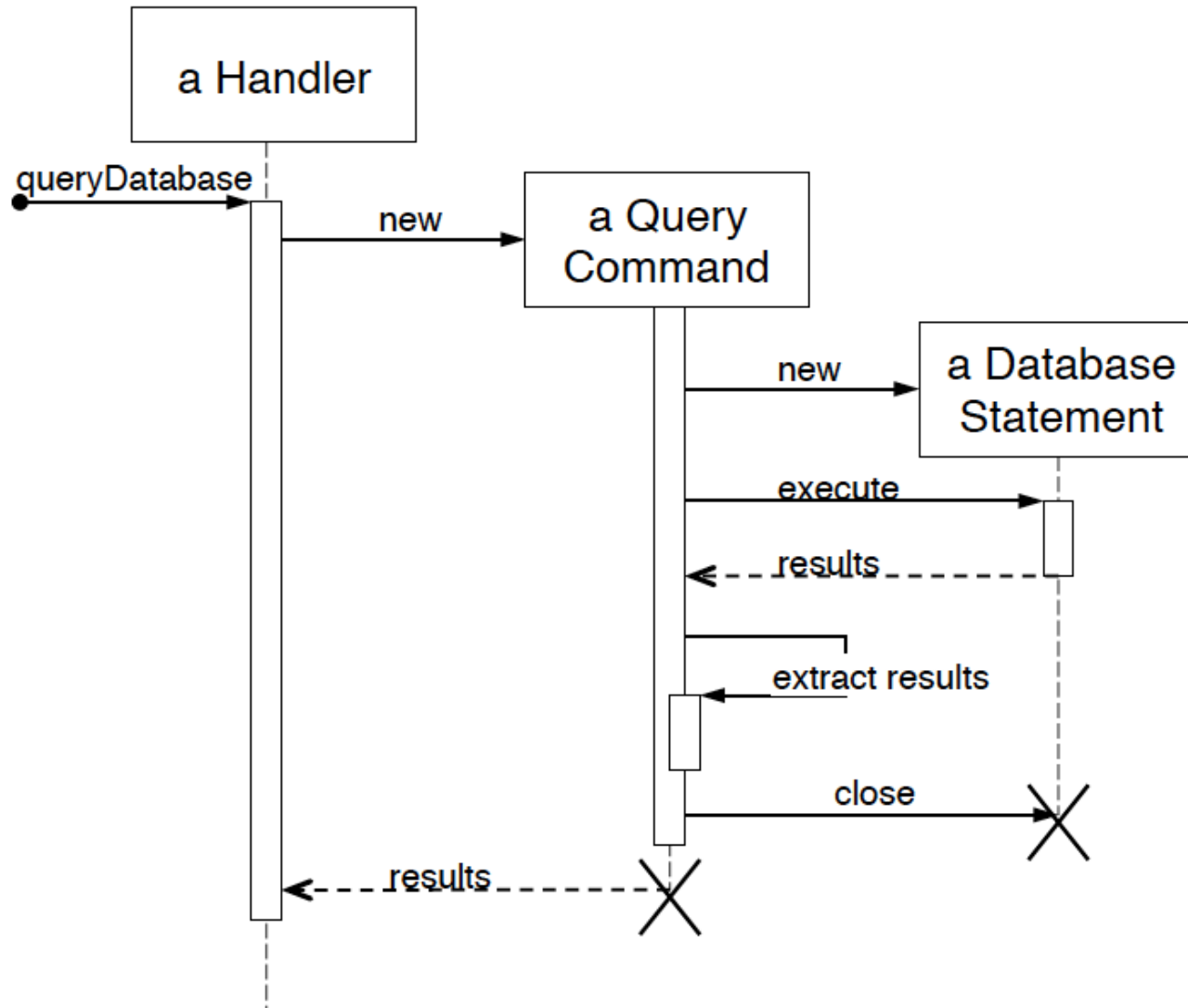


Design Choices...



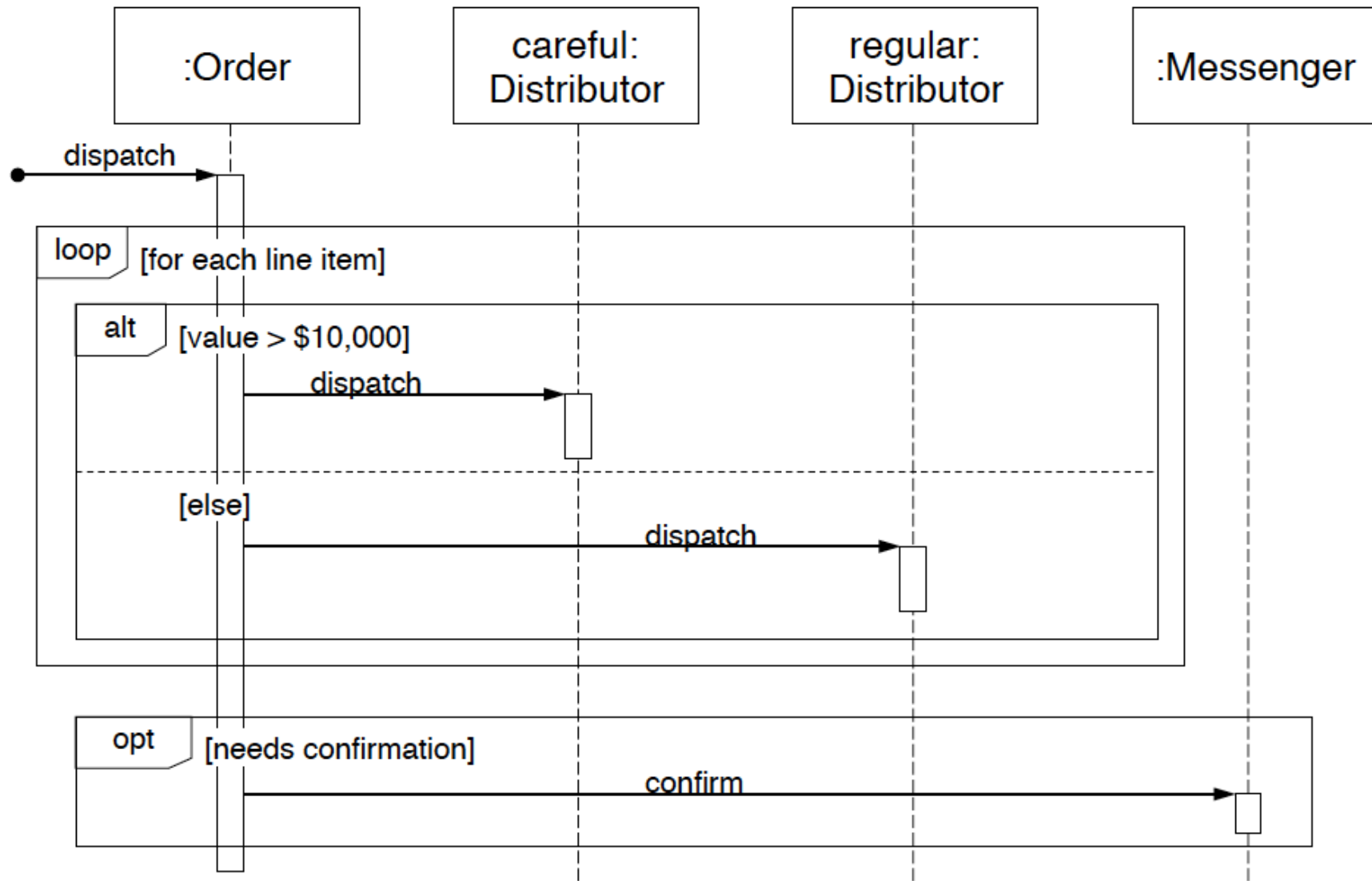


Creating and Deleting Objects





Interaction Frames





Interaction Frame Operators

Operator	Meaning
alt	Alternative; only the frame whose guard is true will execute
opt	Optional; only executes if the guard is true
par	Parallel; frames execute in parallel
loop	Frame executes multiple times, guard indicates how many
region	Critical region; only one thread can execute this frame at a time
neg	Negative; frame shows an invalid interaction
ref	Reference; refers to a sequence shown on another diagram
sd	Sequence Diagram; used to surround the whole diagram (optional)



When to use Sequence Diagrams

Comparing Design Options

Shows how objects collaborate to carry out a task
Graphical form shows alternative behaviours

Assessing Bottlenecks

E.g. an object through which many messages pass

Explaining Design Patterns

Enhances structural models
Good for documenting behaviour of design features

Elaborating Use Cases

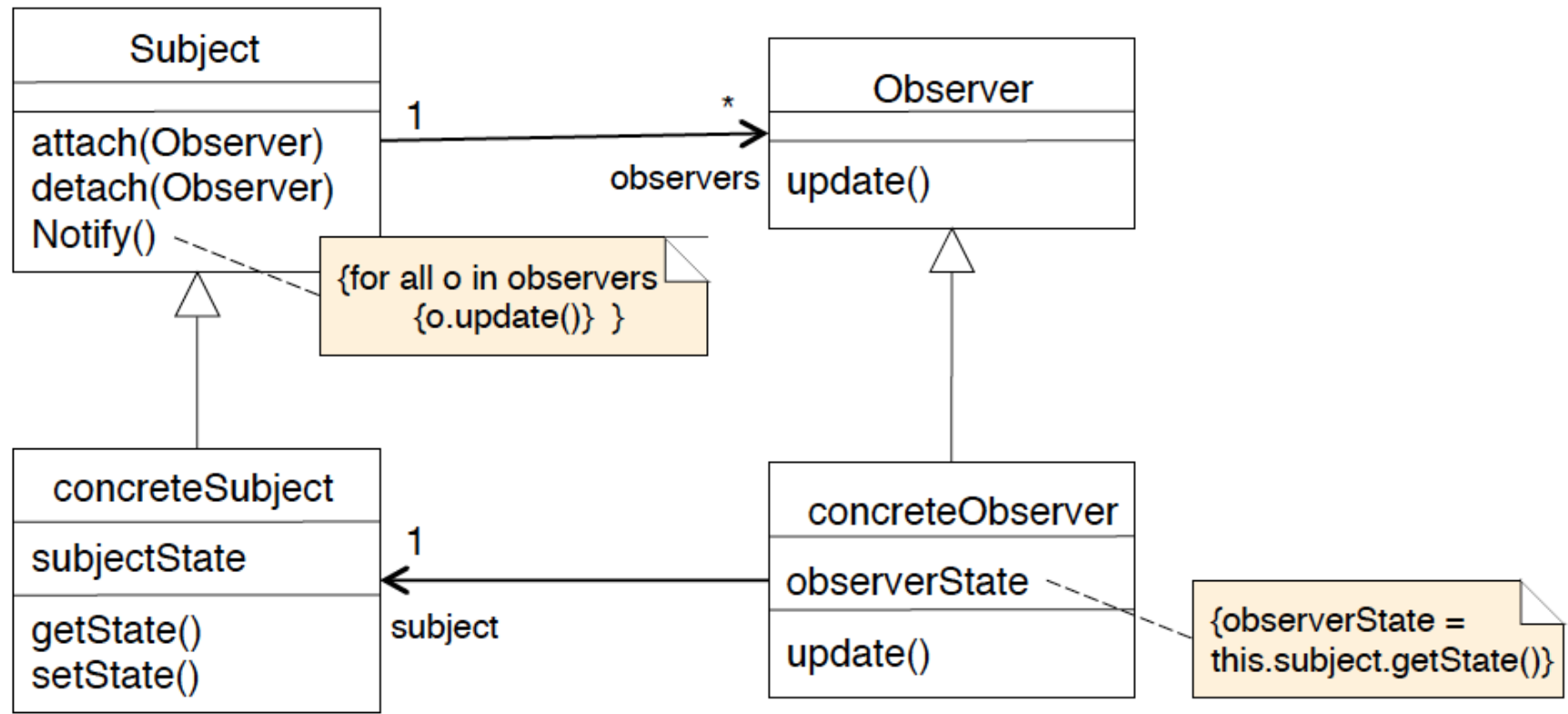
Shows how the user expects to interact with the system
Shows how the user interface operates



Modeling a Design Pattern

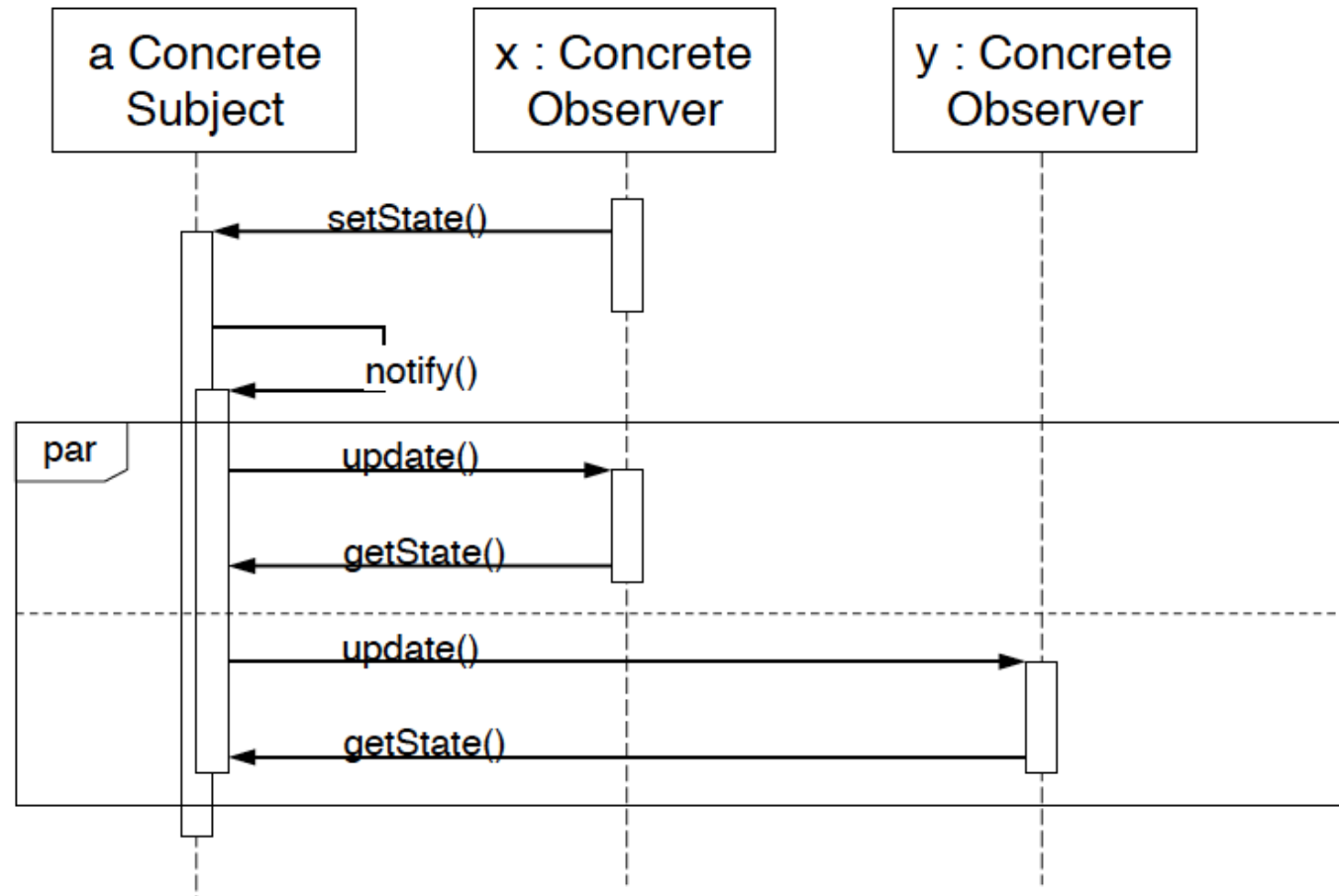
E.g. Observer Pattern

For a one-to-many dependency, when you need to maintain consistency
The subject pushes updates to all the observers





Sequence Diagram for Observer





Style Guide for Sequence Diagrams

Spatial Layout

Strive for left-to-right ordering of messages

Put proactive actors on the left

Put reactive actors on the right

Readability

Keep diagrams simple

Don't show obvious return values

Don't show object destruction

Usage

Focus on critical interactions only

Consistency

Class names must be consistent with class diagram

Message routes must be consistent with (navigable) class associations



use case diagrams

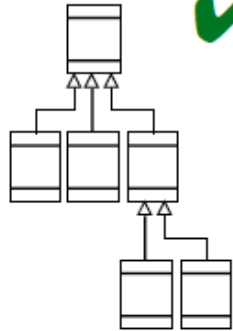


“Use Case”-Driven Design

- **User Stories in Agile Development**
- **Introducing UML into the Software Process**
- **Domain Models**
- **Use Cases**

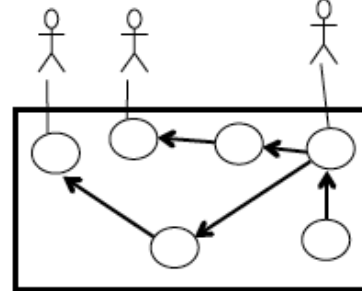


Refresher: UML Notations



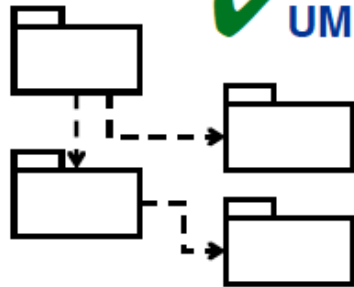
UML Class Diagrams

information structure
relationships between data items
modular structure for the system



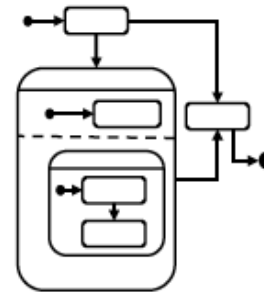
Use Cases

user's view
Lists functions
visual overview of the main requirements



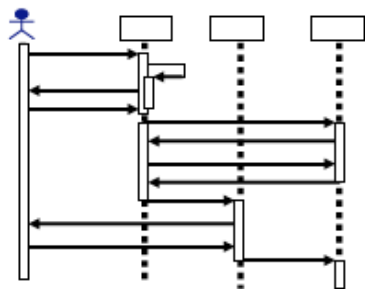
UML Package Diagrams

Overall architecture
Dependencies between components



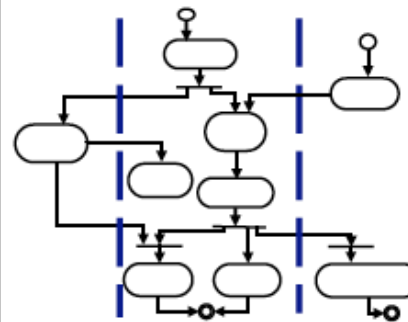
(UML) Statecharts

responses to events
dynamic behavior
event ordering, reachability, deadlock, etc



UML Sequence Diagrams

individual scenario
interactions between users and system
Sequence of messages



Activity diagrams

business processes;
concurrency and synchronization;
dependencies between tasks;



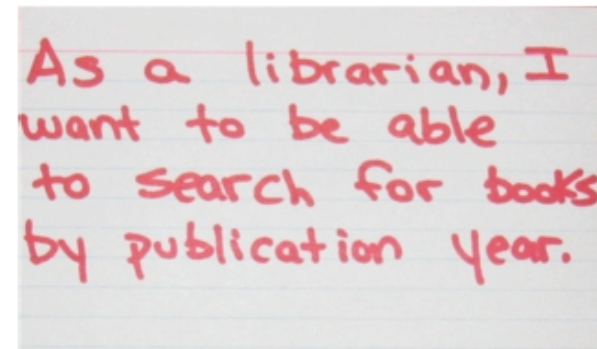
What do users want?

User Stories

Used in XP, Scrum, etc.

Identify the user (role) who wants it

Typically written on notecards



(User Interface) Storyboards

Sketch of how a user will do a task

Shows the interactions at each step

Commonly used in UI Design



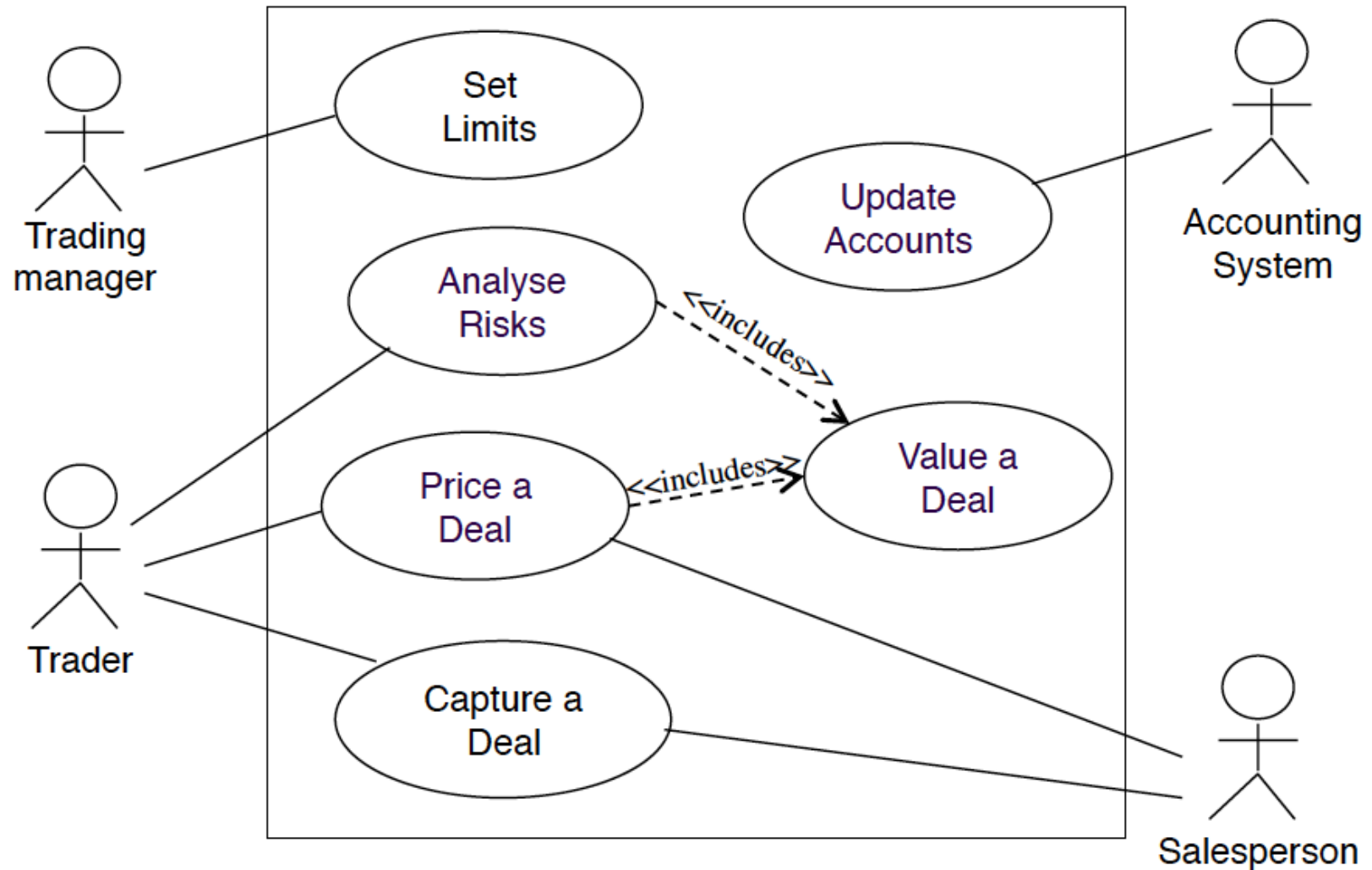
Use Cases

Sets of user features

UML diagram shows inter-relationships



Use Case Diagram

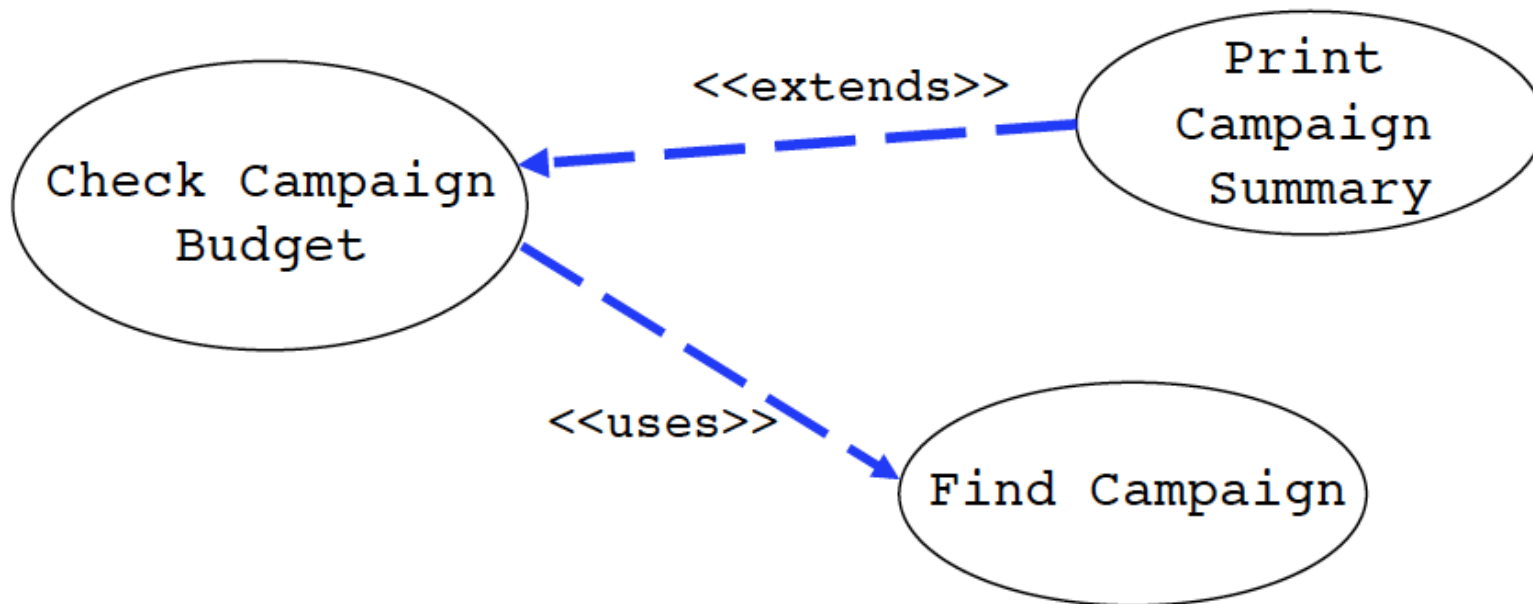




Relationships between Use Cases

<<extends>> when one use case adds behaviour to a base case
used to model a part of a use case that the user may see as optional system behavior;
also models a separate sub-case which is executed conditionally.

<<uses>>: one use case invokes another (like a procedure call);
used to avoid describing the same flow of events several times
puts the common behavior in a use case of its own.





Using Generalizations

Actor classes

Identify classes of actor

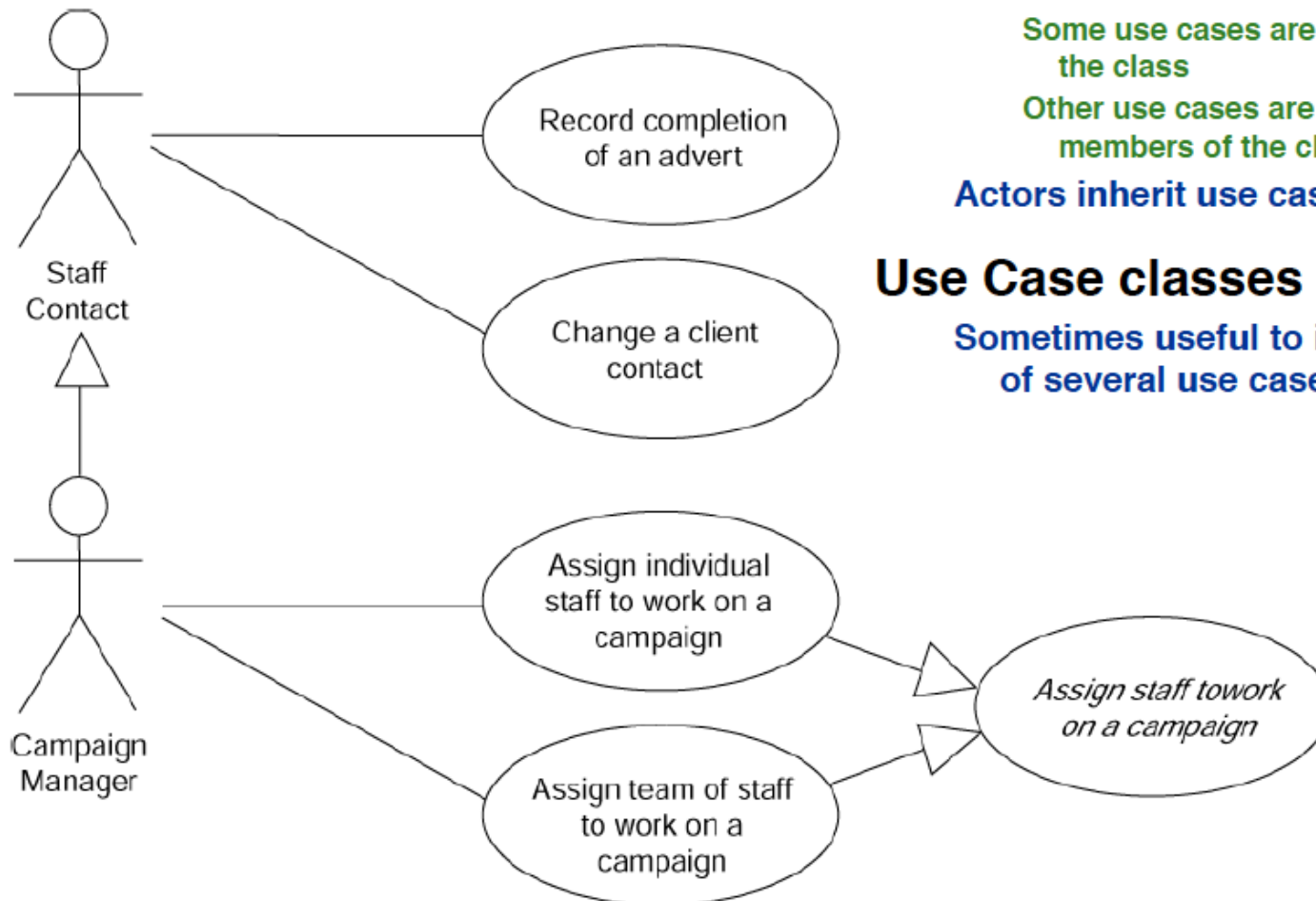
Where several actors belong to a single class
Some use cases are needed by all members in the class

Other use cases are only needed by some members of the class

Actors inherit use cases from the class

Use Case classes

Sometimes useful to identify a generalization of several use cases





Describing Use Cases

For each use case:

a “flow of events” document, written from an actor’s point of view.
describes what the system must provide to the actor when the use case is executed.

Typical contents

- How the use case starts and ends;
- Normal flow of events;
- Alternate flow of events;
- Exceptional flow of events;

Documentation style:

Choice of how to elaborate the use case:

- English language description
- Activity Diagrams - good for business process
- Collaboration Diagrams - good for high level design
- Sequence Diagrams - good for detailed design



Detailed Use Case

Buy a Product

Main Success Scenario:

1. Customer browses catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information (address, next-day or 3-day delivery)
4. System presents full pricing information
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

Extensions:

3a: Customer is Regular Customer

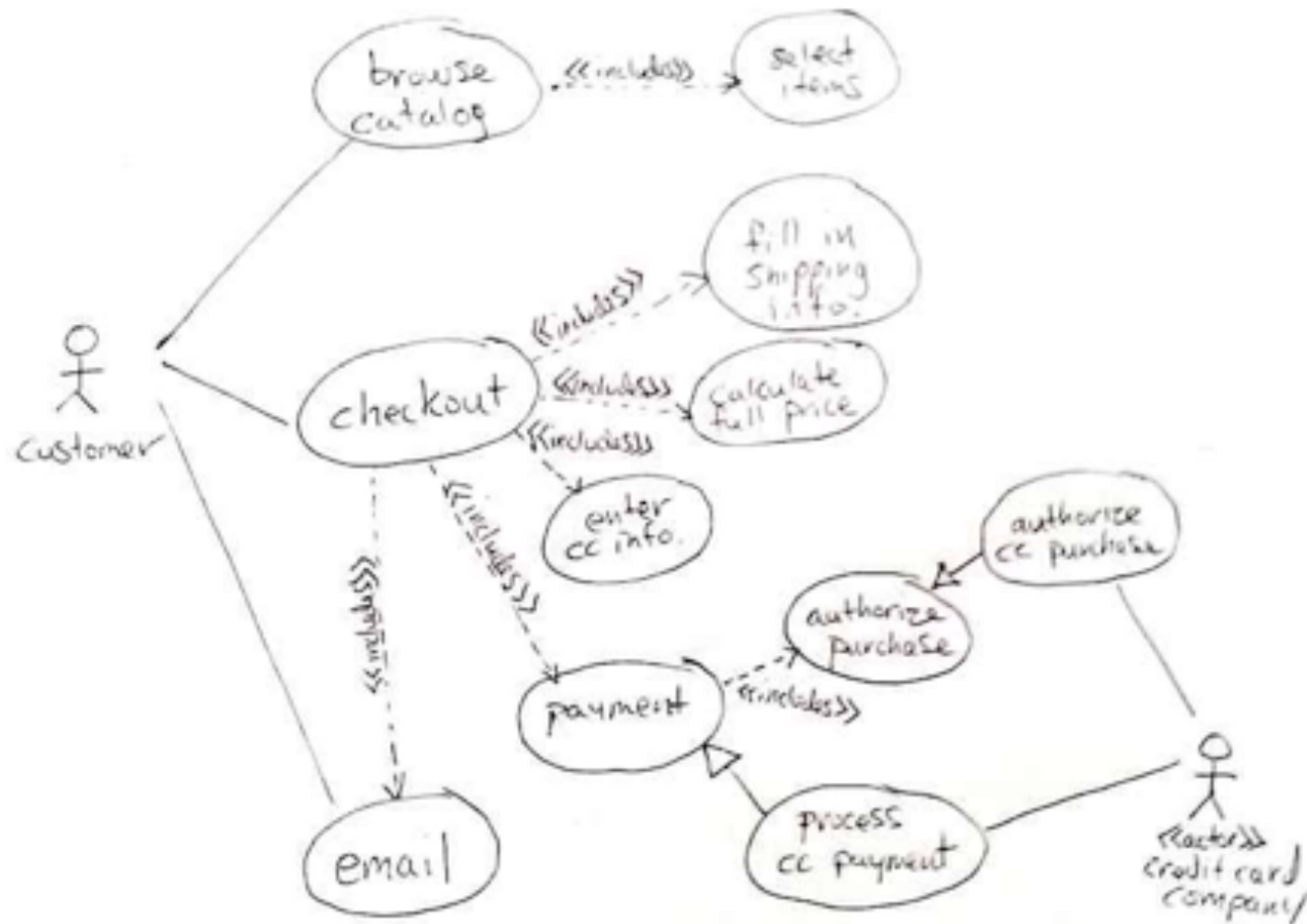
- .1 System displays current shipping, pricing and billing information
- .2 Customer may accept or override these defaults, returns to MSS at step 6

6a: System fails to authorize credit card

- .1 Customer may reenter credit card information or may cancel



detailed use case to diagram





Finding Use Cases

Browse through existing documents

noun phrases may be domain classes

verb phrases may be operations and associations

possessive phrases may indicate attributes

For each actor, ask the following questions:

Which functions does the actor require from the system?

What does the actor need to do ?

Does the actor need to read, create, destroy, modify, or store some kinds of information in the system ?

Does the actor have to be notified about events in the system?

Does the actor need to notify the system about something?

What do those events require in terms of system functionality?

Could the actor's daily work be simplified or made more efficient through new functions provided by the system?