



Lecture 5: 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

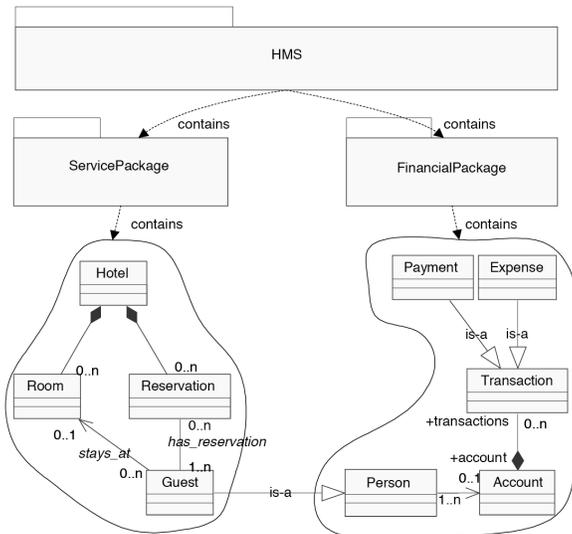
Requires sophisticated tools

“tripless” - the model is the code





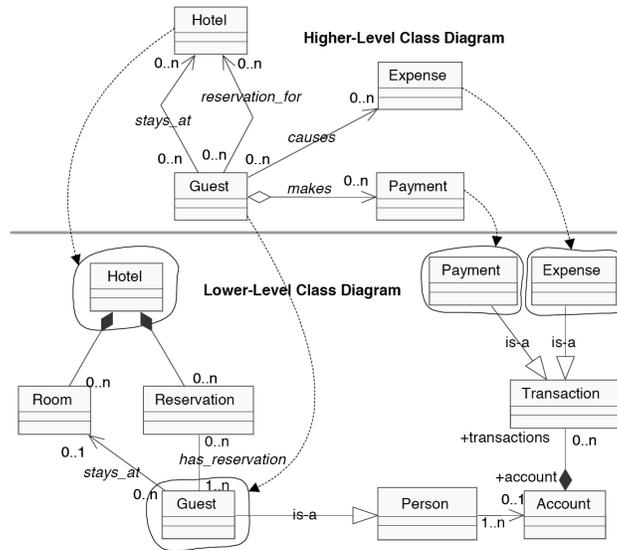
Package Decomposition



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



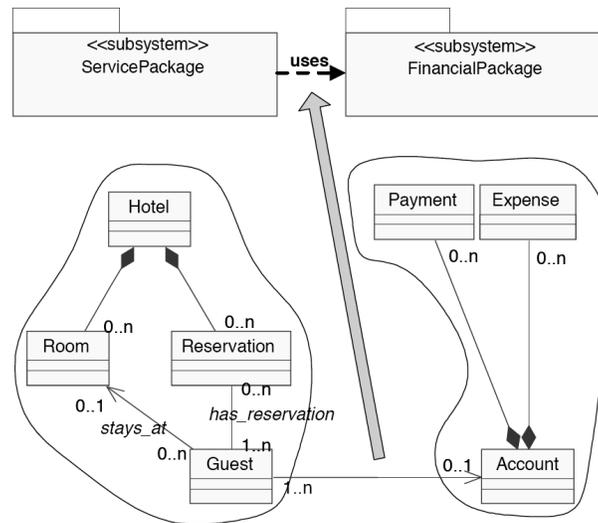
Class Abstraction



Source: from Egved "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

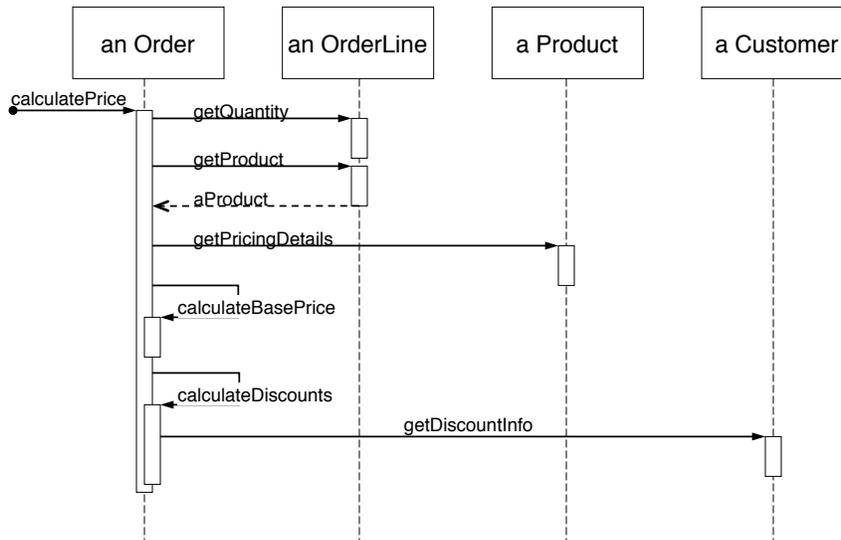
- Execution traces
- State machines models of complex objects

E.g. Function of the code

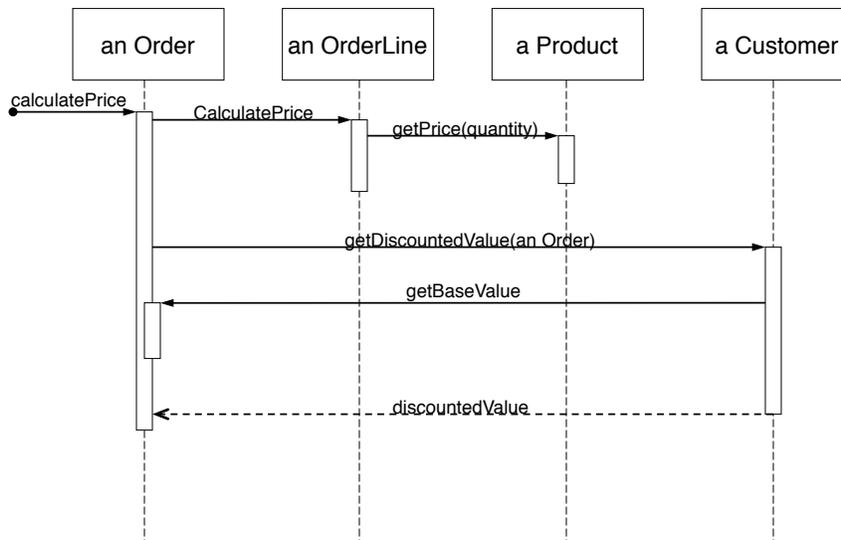
- What functions does it provide to the user?



Sequence Diagrams

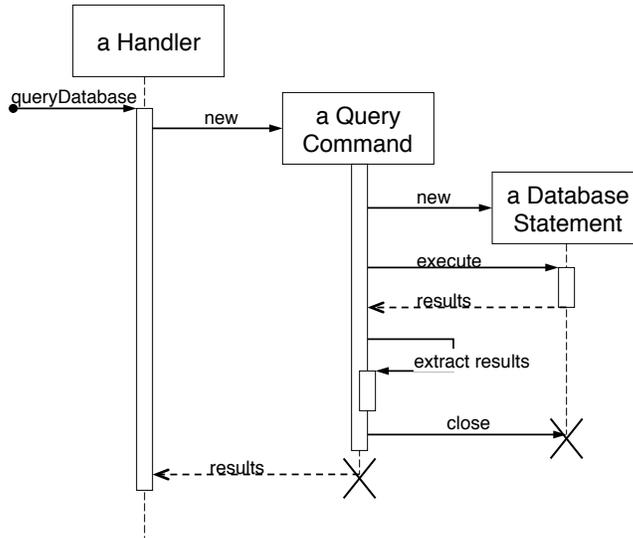


Design Choices...

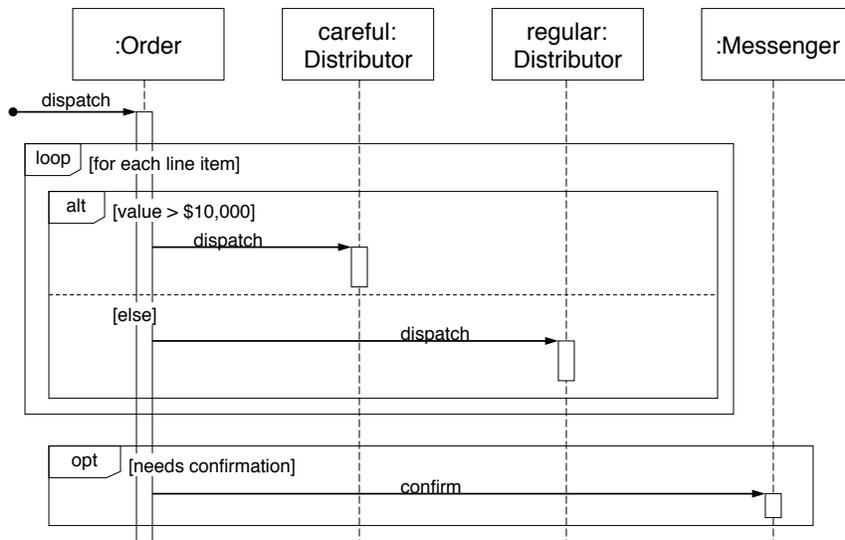




Creating and Deleting Objects



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

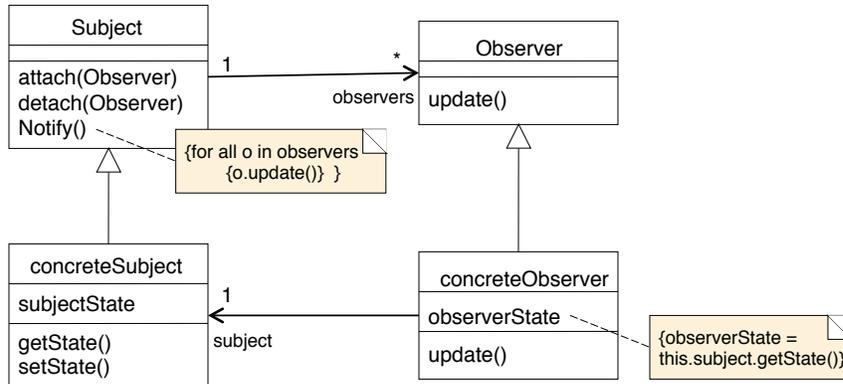




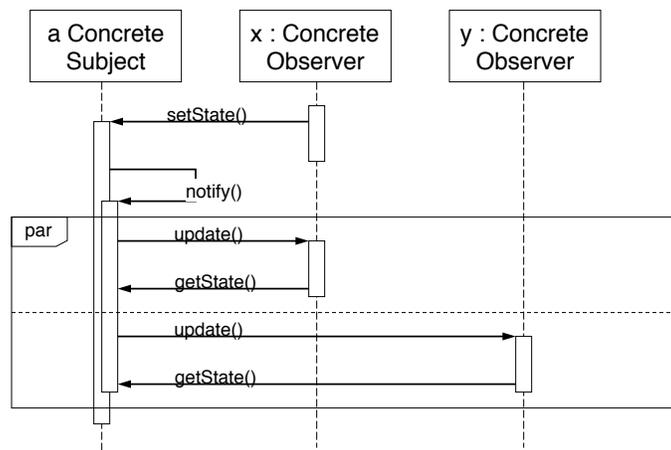
Modeling a Design Pattern

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

