Introduction to the Unified Modeling Language (UML)

Part 2

Giancarlo Succi

Part 2

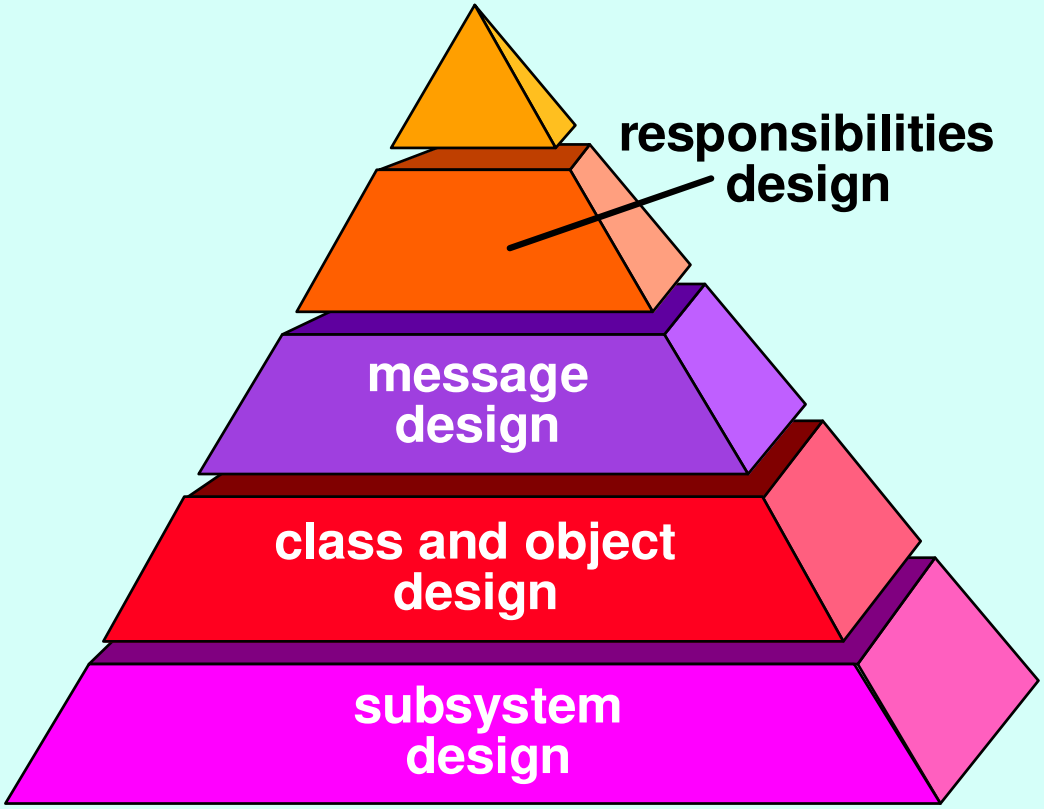
- Object Oriented Design in UML
- Components in UML
- Features for real-time system analysis and design in UML
- Introduction to the Object Constraint Language
- Introduction to Refactoring

Object Oriented Design in UML

OOD Topics

- The Idea of OOD
- Design Issues
- OOA vs. OOD
- Statechart Diagrams
- Activity Diagrams
- Sequence Diagrams
- Collaboration Diagrams
- Example of plugging everything together
- Systems and Sub-Systems
- Package Diagrams
- Design Patterns

Object-Oriented Design



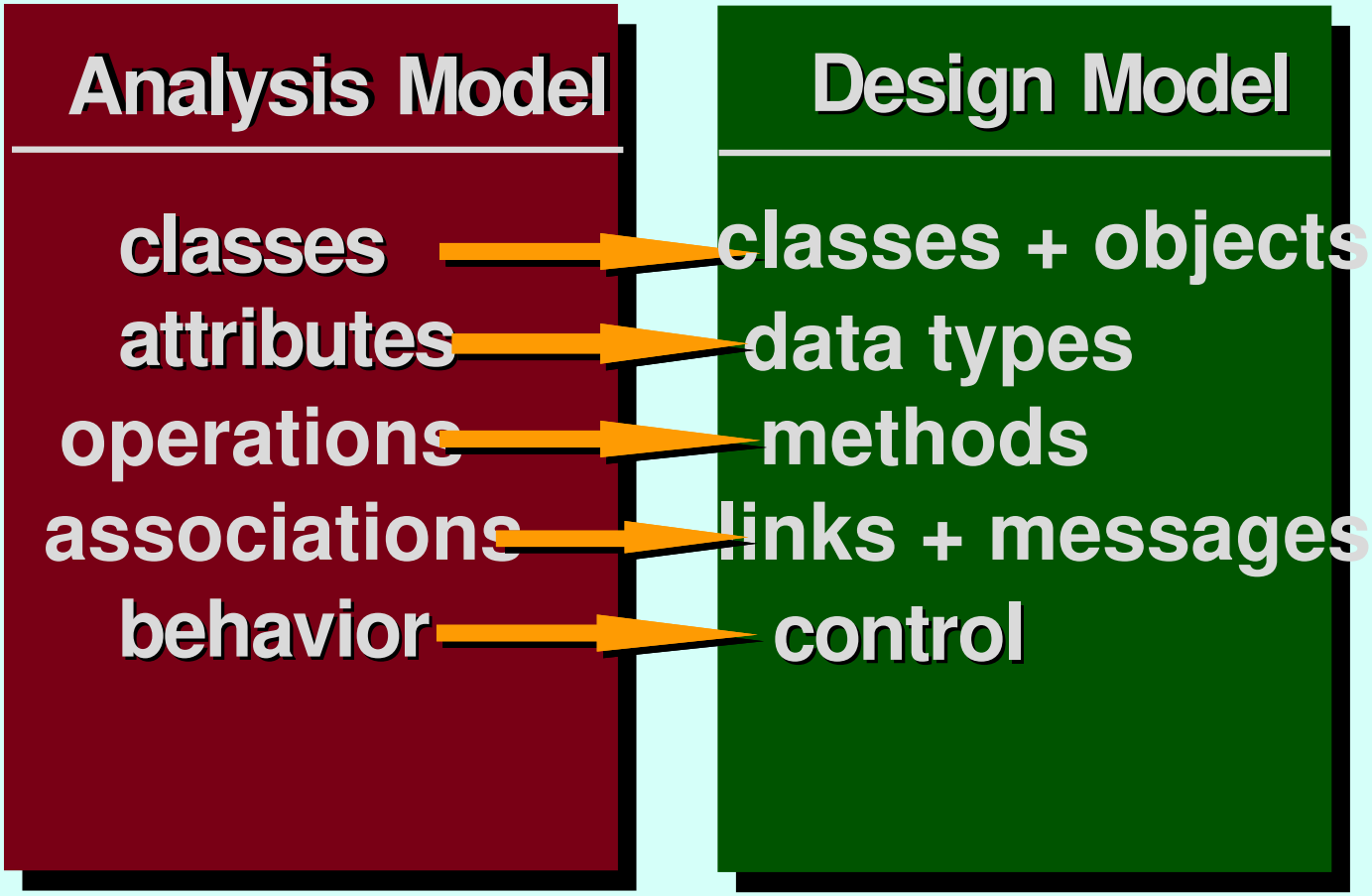
Design Issues

- ***decomposability***—the facility with which a design method helps the designer to decompose a large problem into sub-problems that are easier to solve;
- ***composability***—the degree to which a design method ensures that program components (modules), once designed and built, can be reused to create other systems;
- ***understandability***—the ease with which a program component can be understood without reference to other information or other modules;
- ***continuity***—the ability to make small changes in a program and have these changes manifest themselves with corresponding changes in just one or a very few modules;
- ***protection***—a architectural characteristic that will reduce the propagation of side affects if an error does occur in a given module.

Ingredients for OOD

- ***Problem domain component***—the subsystems that are responsible for implementing customer requirements directly;
- ***Human interaction component*** —the subsystems that implement the user interface (this included reusable GUI subsystems);
- ***Task Management Component***—the subsystems that are responsible for controlling and coordinating concurrent tasks that may be packaged within a subsystem or among different subsystems;
- ***Data management component***—the subsystem that is responsible for the storage and retrieval of objects.

OOA to OOD Relationship



What distinguishes OOD from OOA?

- Level of detail
  - Names are fixed
  - Fixed signatures for messages
  - Multiplicity & its realization
  - Visibility
  - Algorithms for methods
  - More detailed sequence/collaboration diagrams
  - .....
- Additional diagram notations

Therefore...

- In OOD we still have class diagrams, but they are refined to match the design of the system

In addition to class diagrams, we have several other diagrams:

Structural Diagrams

- ✓ Object Diagrams
- ✓ Deployment Diagrams

Behavioral Diagrams

- ✓ Sequence Diagrams
- ✓ Collaboration Diagrams
- ✓ Statechart Diagrams
- ✓ Activity Diagrams

Diagrams that we will not discuss

UML includes other diagrams that will not be discussed in the present course:

- ☒ Component Diagrams

Structural Diagrams for OOD in UML

Class Diagrams

- Their structure is the same as for OOA

Object Diagrams

- They deal with objects, instances of classes
- They are absolutely homomorphic to class diagrams
- Given this, we will not analyze them in deep

Behavioral Diagrams for OOD in UML

- Statechart Diagrams
  - Describe the evolution of the states of any classifier
  - Commonly used for objects
- Activity Diagrams
  - Describe the evolution of activities in the system
- Sequence Diagrams
  - Describe the interactions between objects by time ordering
- Collaboration Diagrams
  - Describe the interactions between the objects by organizations

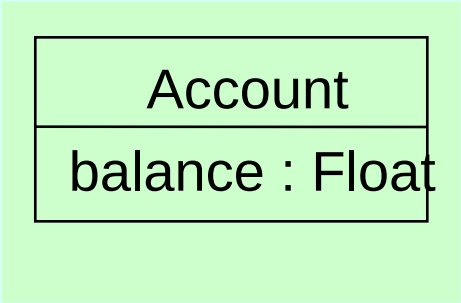
Interaction Diagrams

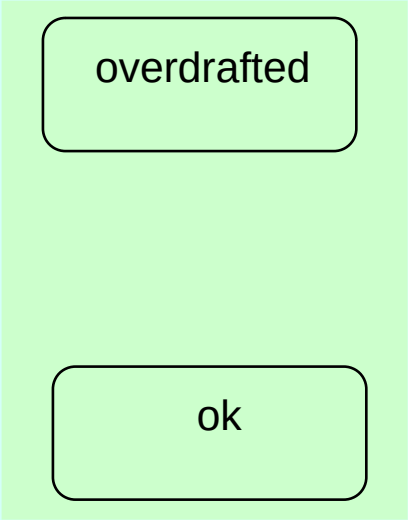
Statechart diagrams vs Interaction diagrams

- **Interaction diagrams:** show how objects interact with each other
- **Statechart diagrams:** Shows the behavior of *one* object
  - how does it change its state based on the messages it receives
  - narrowly focused, fine-grained
- **Other names:**
  - State transition
  - State diagrams
  - Harel diagrams

Object states

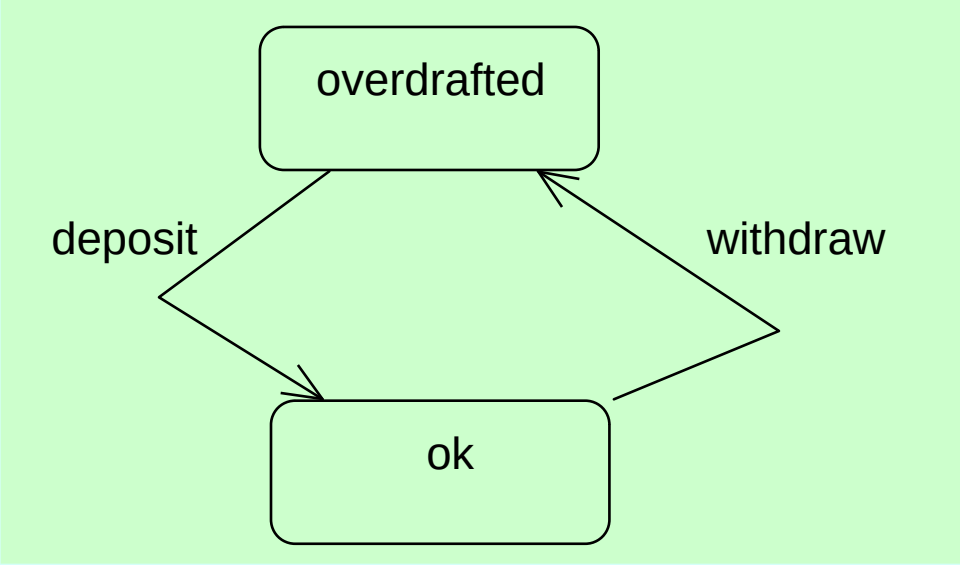
- State
  = set of values that describe an object at a specific moment in time
- State is determined based on the attribute values





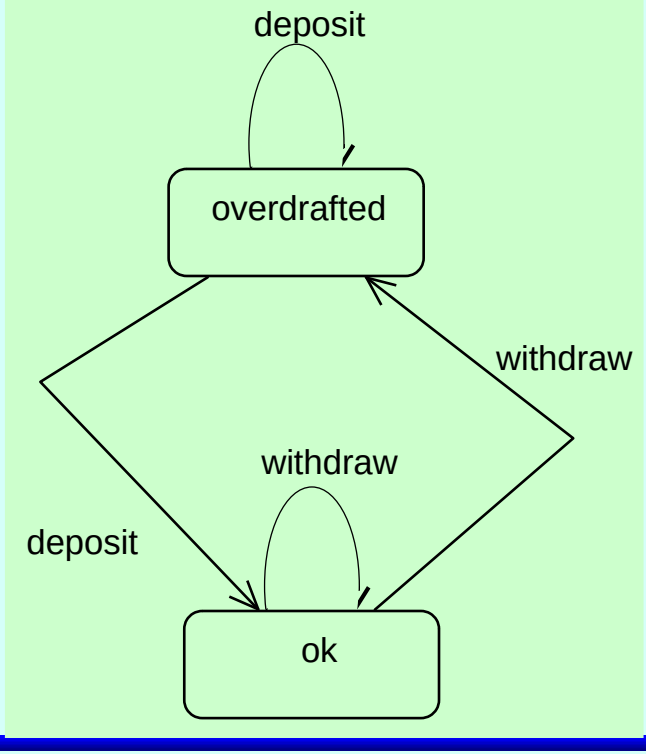
State changes (1)

- States may be changed when an event occurs



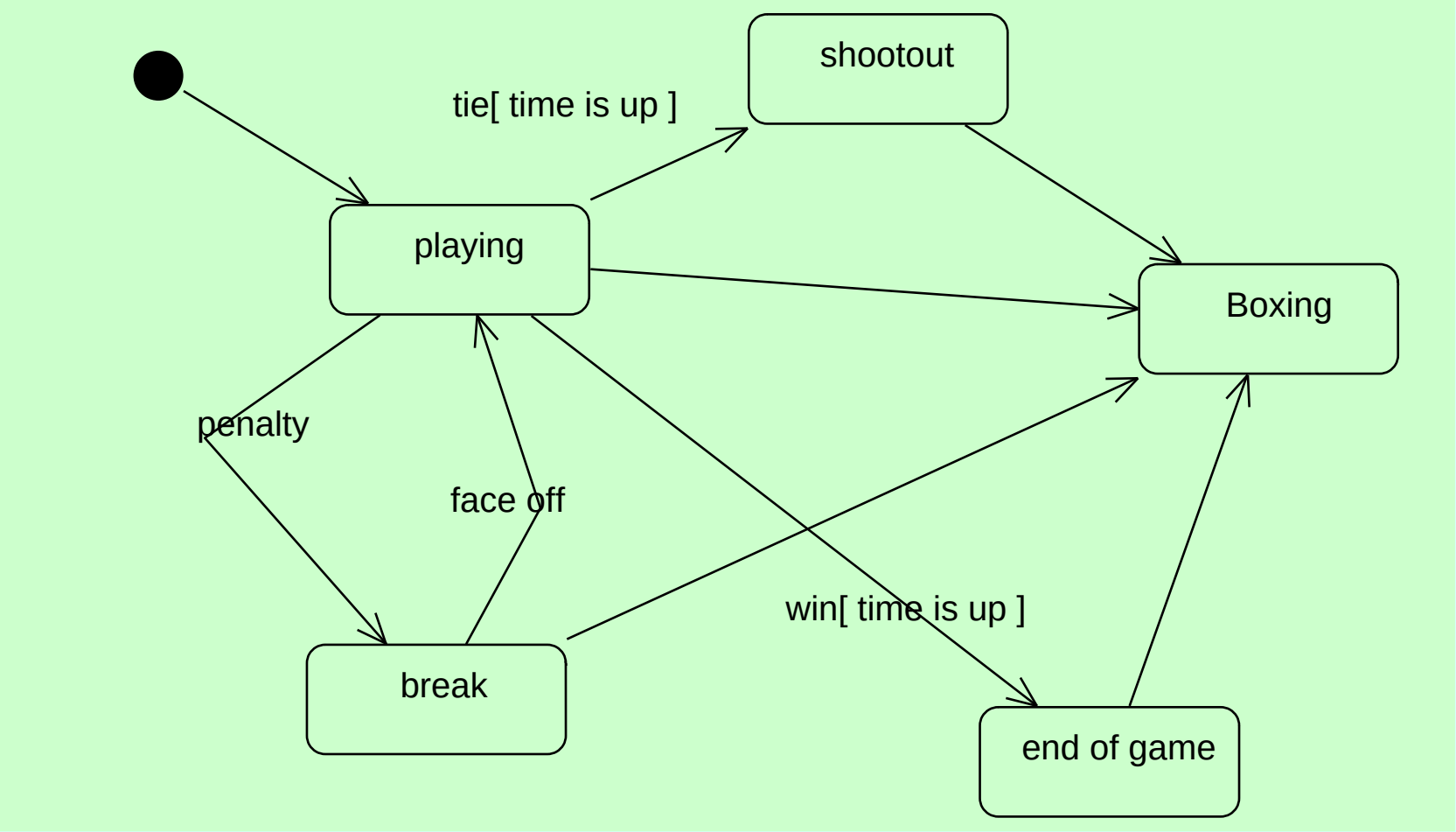
State changes (2)

- Events: Messages received
- Events may or may not change the state

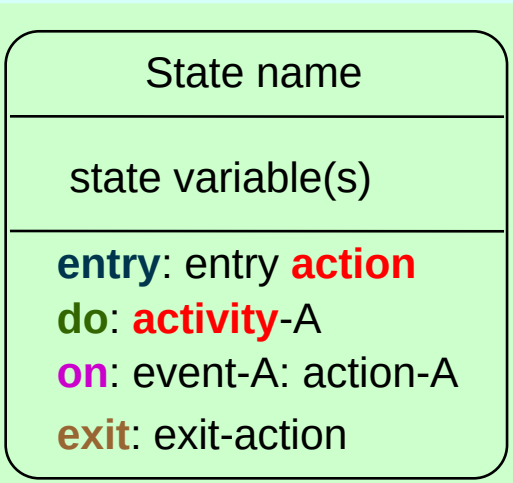


Example of Statechart Diagrams: States of a hockey game

States of a hockey game



State diagram notation (1)



- Activity: Can take longer and be interrupted
- Action: Occur quickly

  ➡ what means “quickly”?

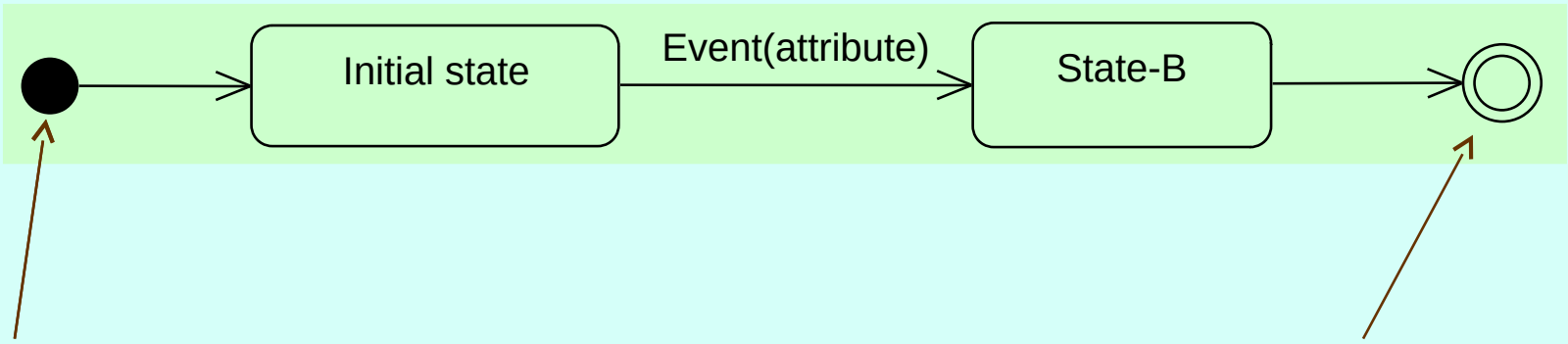
- entry: an action that is performed on entry to the state
- do: an ongoing activity performed while in the state (example: display window)
- on: an action performed as a result of a specific event
- exit: an action performed on exiting the state

State diagram notation (2)

- **Event: message send**
- **Guard condition:**
  - **Transition only occurs when guard evaluates to true**
  - **Guards of transition exiting one state are mutually exclusive**
- **Action: Processes considered to occur quickly and are not interruptible**

Each part can be omitted!

State diagram notation (3)

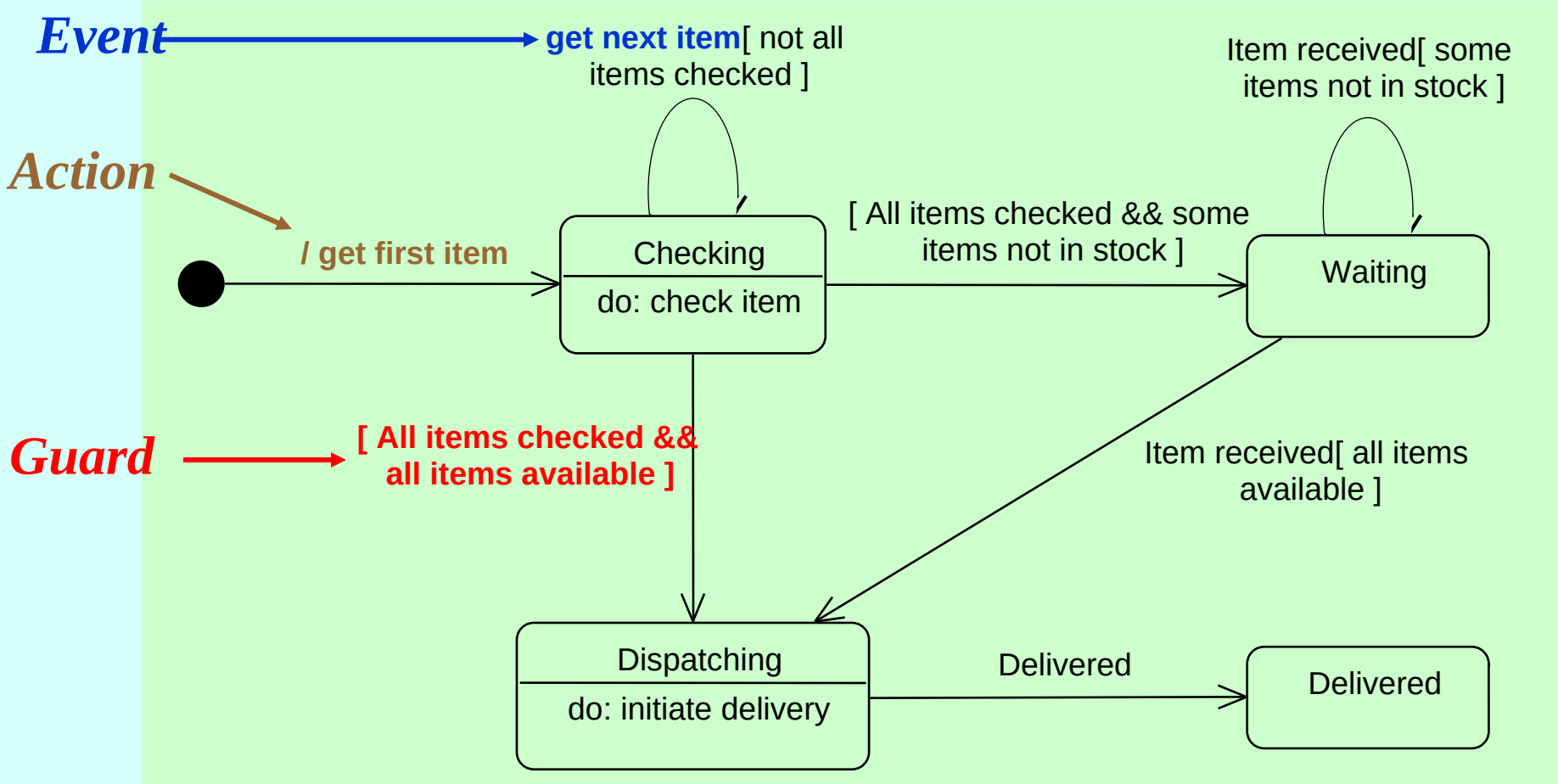


Start State

End State

Example of Statechart Diagrams (2): Order Management

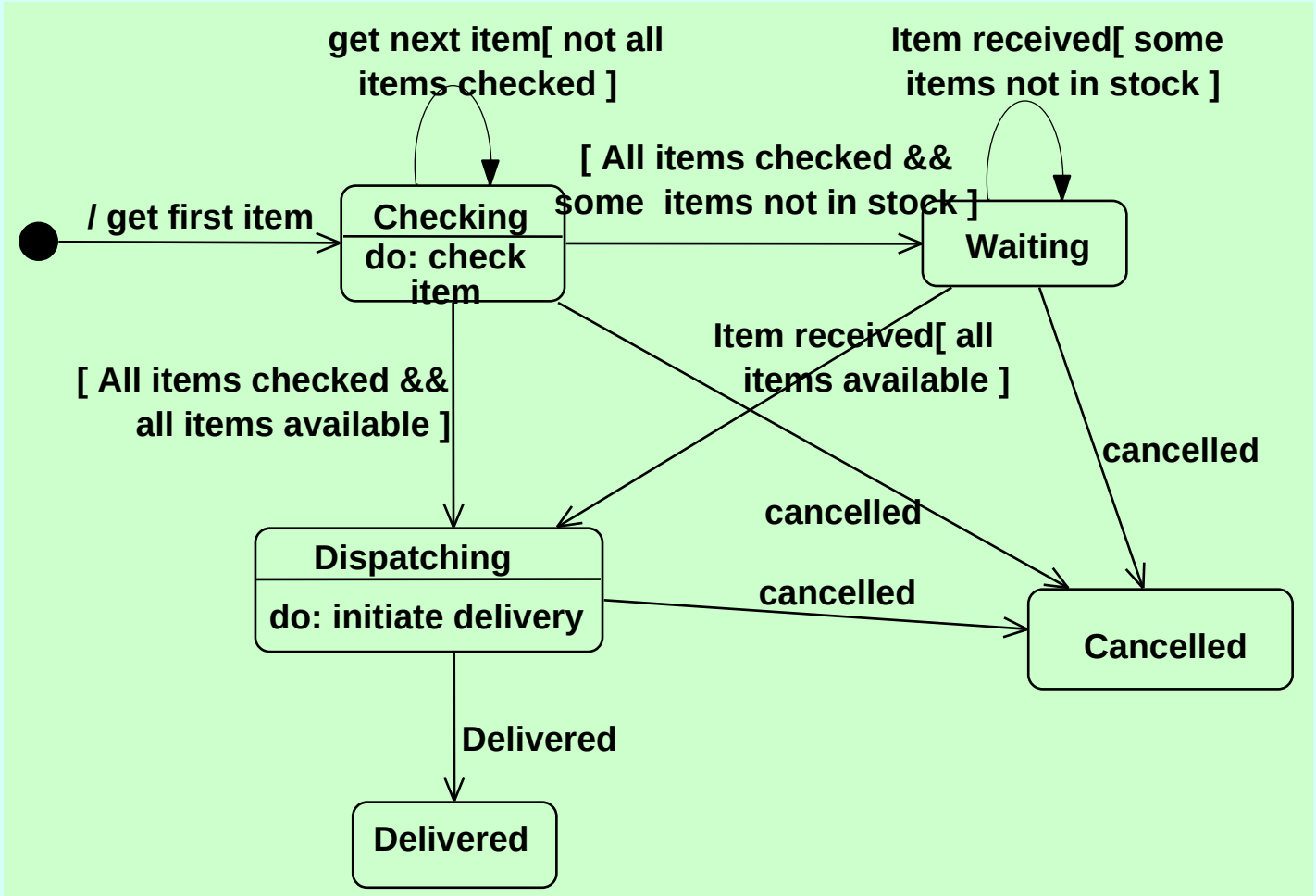
State transitions for an order



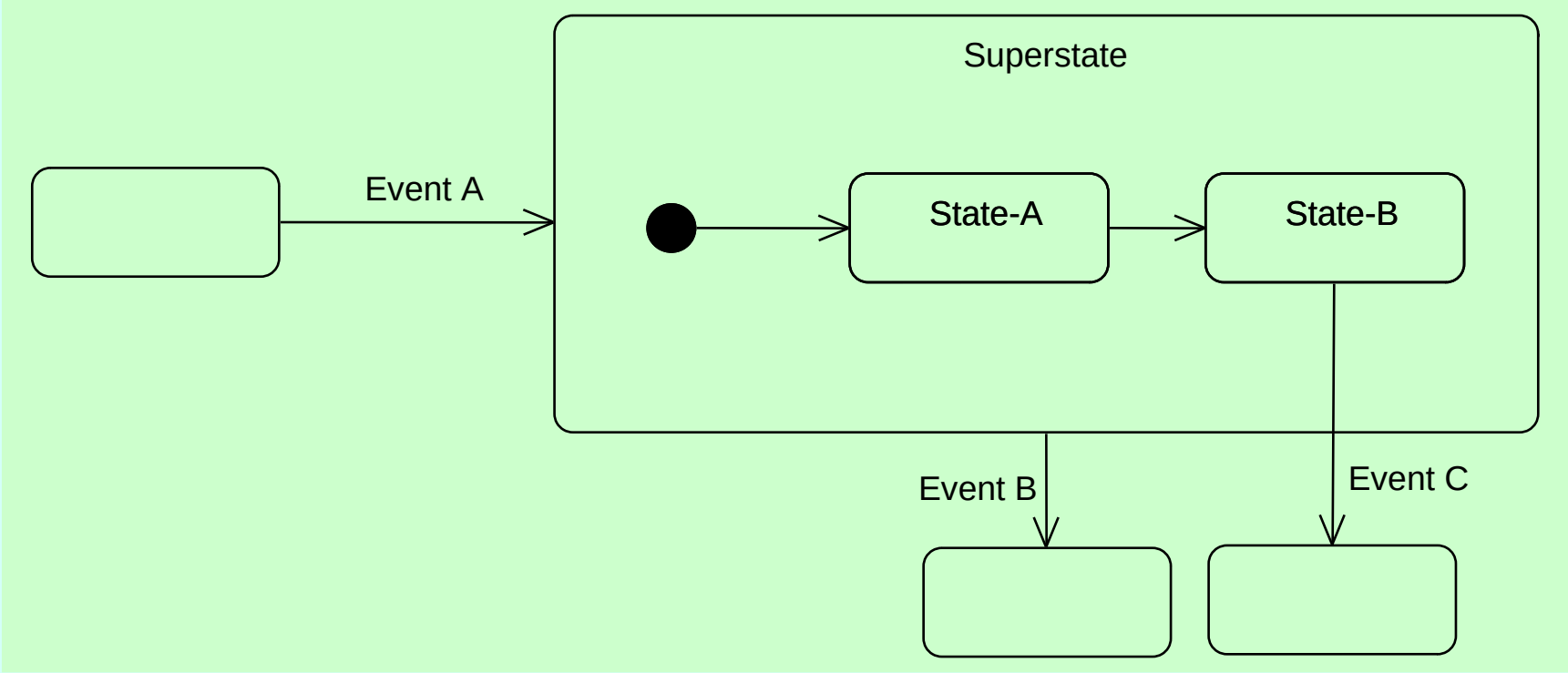
Problem: Cancel the order

- Want to be able to cancel an order at any time
- Solutions
  - Transitions from every state to state “cancelled”
  - Superstate and single transition

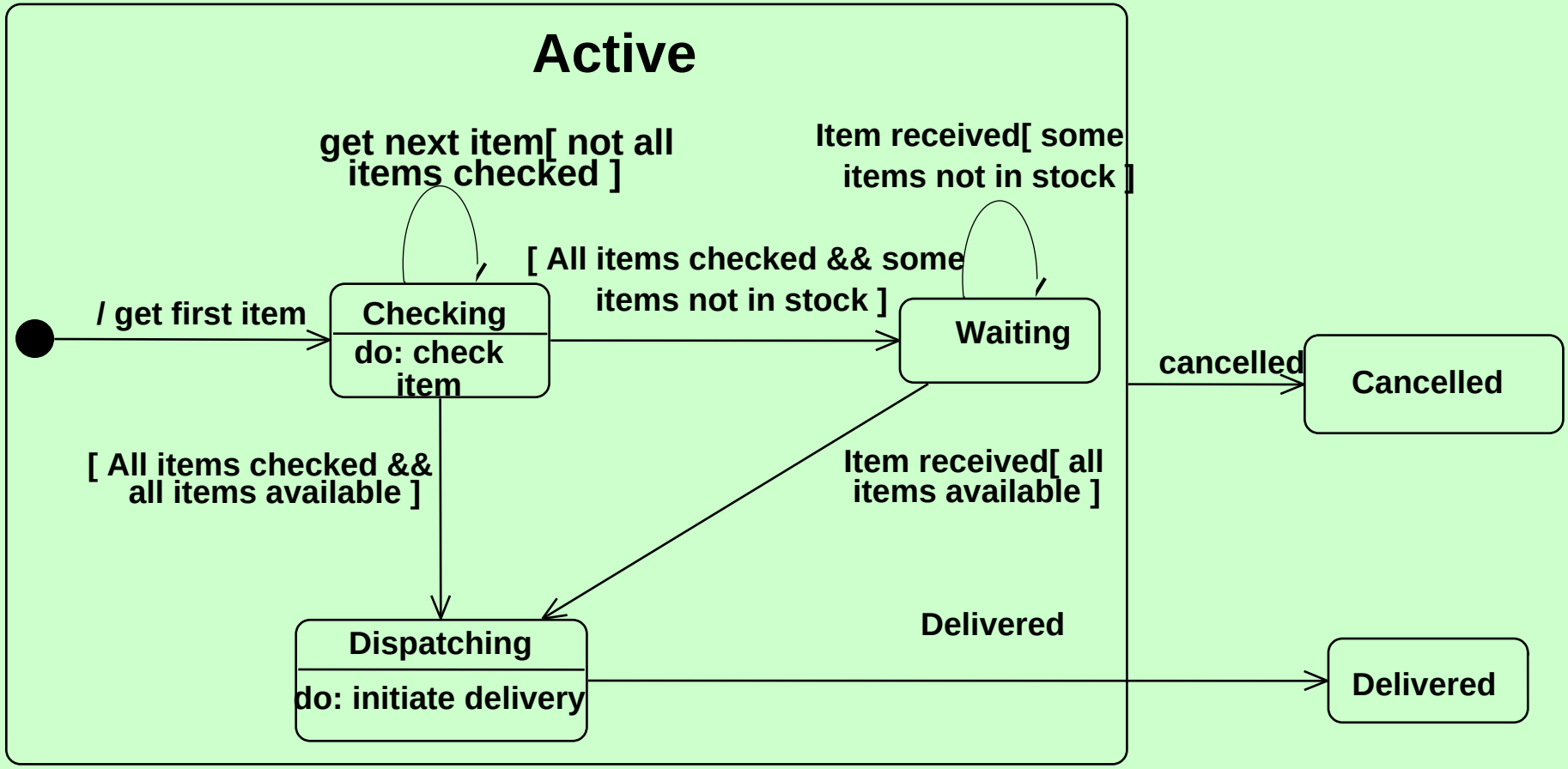
Transitions to “cancelled”



State diagram notation (4)



Superstate / Substates



Final Note

- Statechart diagrams DO NOT HAVE TO refer to classes or objects, they can also refer to subsystems etc…
- … However, often this is the most widely use of statechart diagrams...

Proposed Exercises

- Define the statechart diagram of a basketball game
- Define a statechart diagram of your software development process

Possible solution

- First, we create the diagram
- Then, we need to set the appropriate use cases and classes
- Eventually, we move to the statechart diagrams

Topics not covered

- Nested states vs. Concurrent states
- History states
- Internal transitions

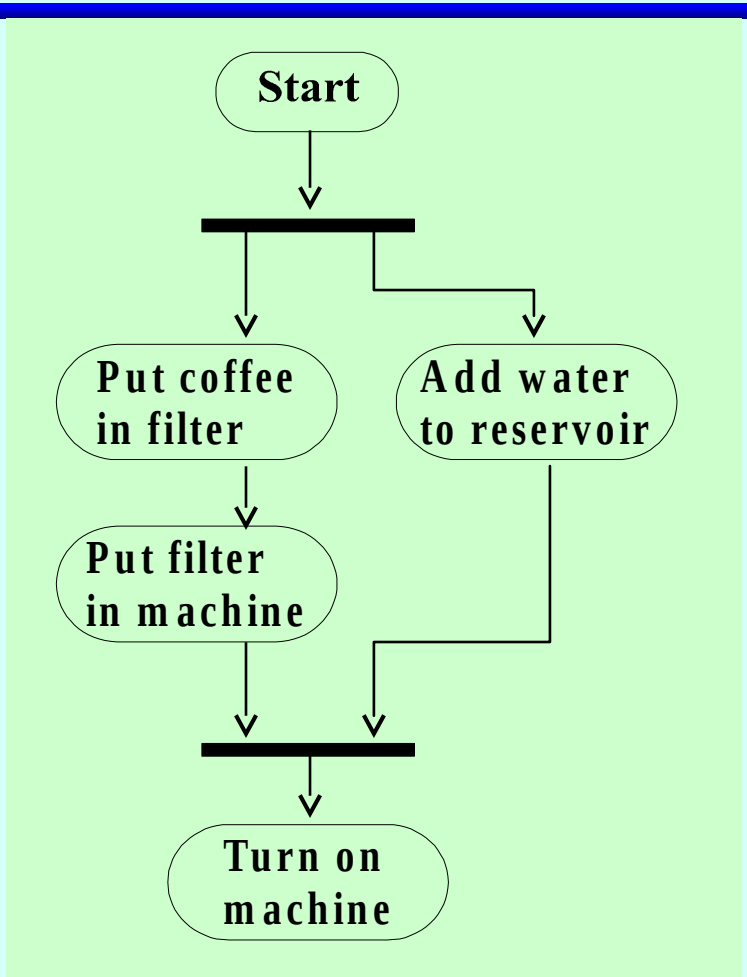
Activity diagrams

- Used to describe
  - workflow
  - parallel processing
- Activities
  - conceptual: task to be done
  - specification/implementation: method on a class
- Similar to Petri nets

Example of activity diagrams

- The Coffee Pot

Activity diagram

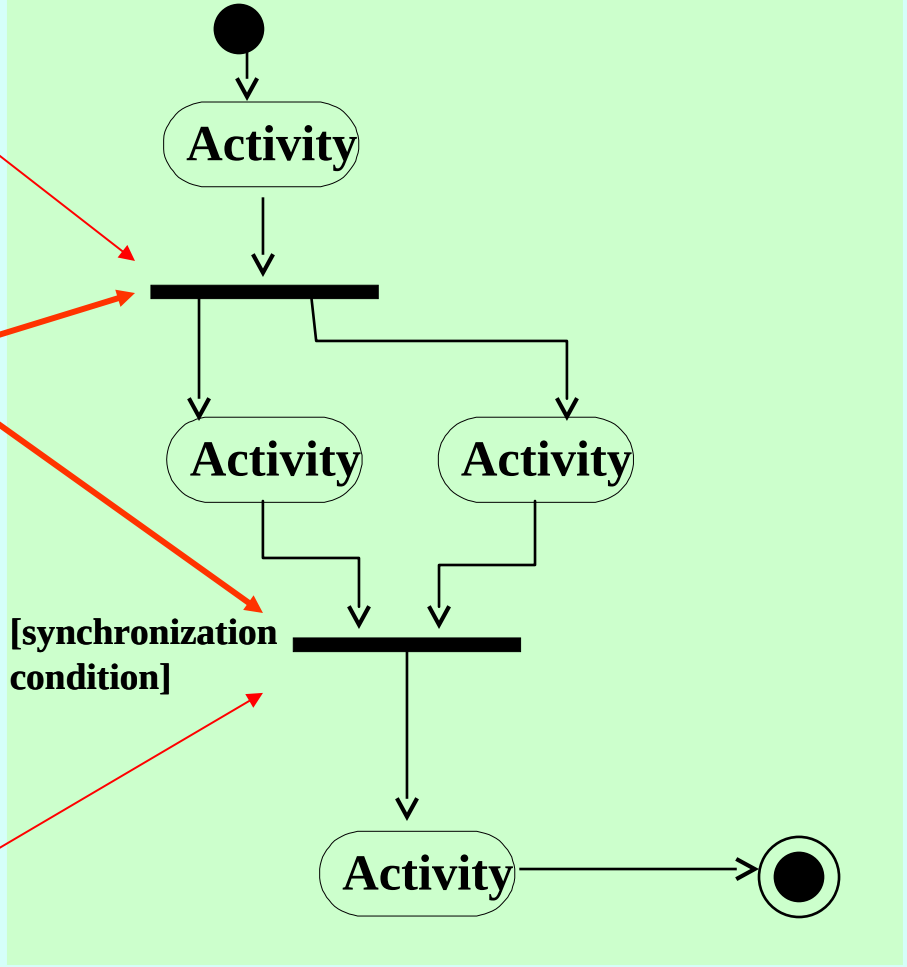


Structure of activity diagrams

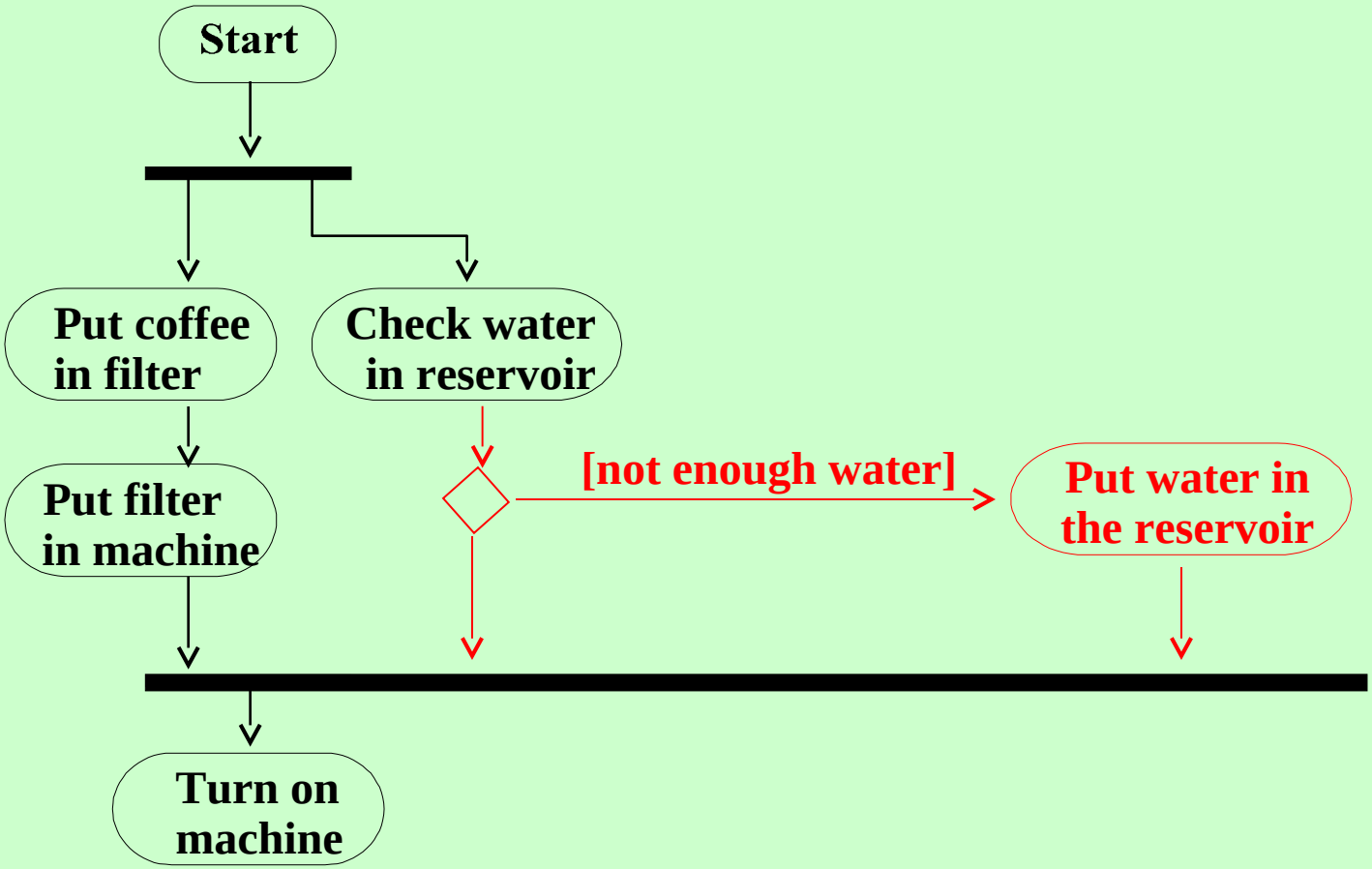
Synchronization bar:

- Activities can be carried out in parallel (any order)
- Incoming: all predecessors have to be finished

Activity diagram shows partial order of activities



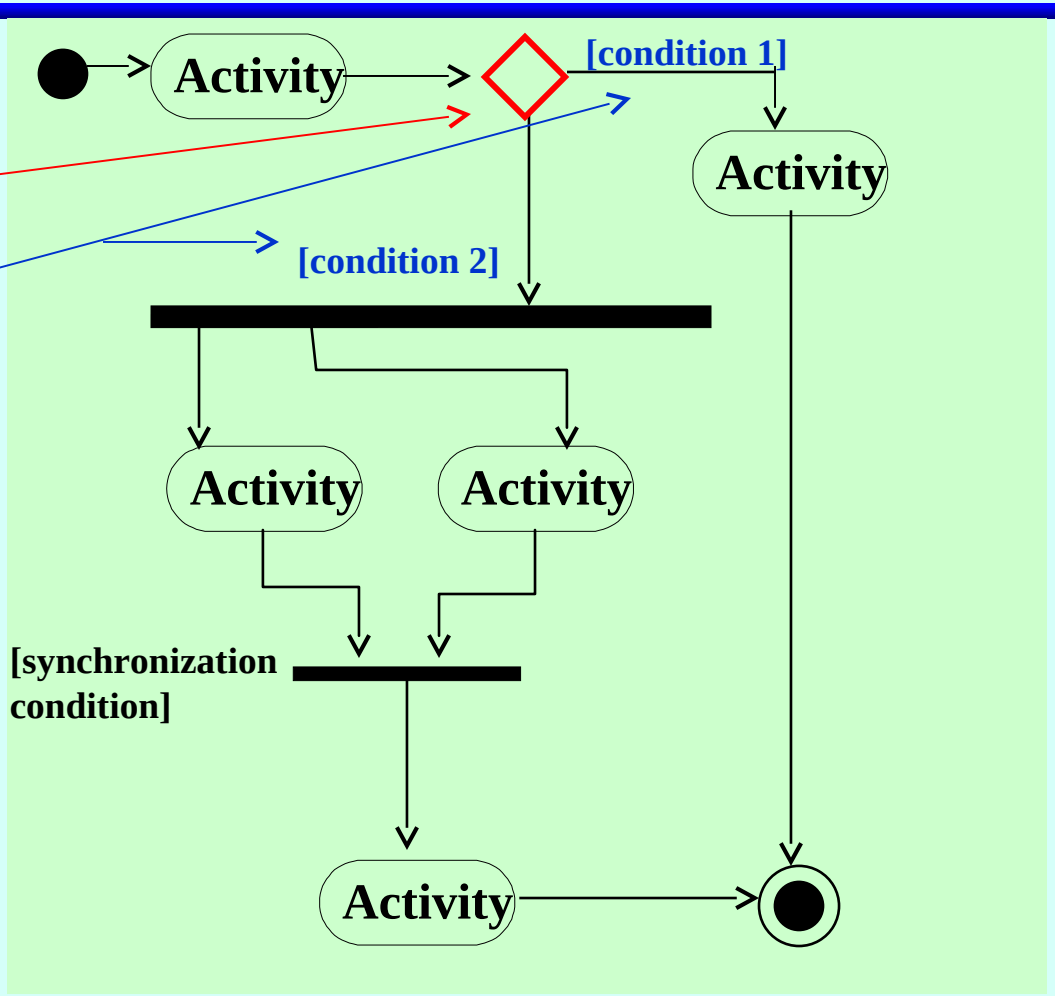
Conditions in activity diagrams



Structure of activity diagrams

Branch

Guard expression



Proposed Exercise

- Define an activity diagram for the previously defined basketball game
- Define an activity diagram for your software development process … mmm …

Topics not covered

- Activities vs. Actions
- Swimlanes and Object flows
- Changes in an object's state

Continuation of the Proposed Exercise (1)

Develop the relevant OO Statechart Diagrams and Activity Diagrams for a system supporting the reservation and scheduling for taxi drivers that was discussed in the first day of the course.

Continuation of the Proposed Exercise (2)

Develop the relevant OO Statechart Diagrams and Activity Diagrams for the system that manages the search, the selection, and the purchase of books at Chapters (http://www.chapters.ca) that was discussed in the first day of the course

Continuation of the Proposed Exercise (3) *Stocks Trading Service*

Develop the relevant OO Statechart Diagrams and Activity Diagrams for…

- By connecting to the service, a user can connect to different banks to acquire stock prices. The system also allows the user to perform some trend and prediction analysis of prices. If users are interested in ordering some stocks, they can choose to order them immediately or with a delay. They can also either bid at a single price or within a range of prices.
- The system should handle the situation where the connection to a bank is down, there is a conflict of bids, or if a particular stock is no longer available.

Continuation of the Proposed Exercise (4) N*etwork Printing Service*

Develop the relevant OO Statechart Diagrams and Activity Diagrams for…

- There is a super high-resolution color laser printer available on the network for users to print documents to. The service allows users to preview the output of their document on their screens. In addition, the user can also view the status of the printer to see whether there are other documents waiting to be printed and whether there are any problems with the printer (such as paper jams, out of paper, low on toner, etc…). In addition, users can monitor their own print jobs and delay or delete jobs as they see fit.
- To use this service, a user needs to have the proper authorization and print quota to print. A system administrator manages users and their print quotas.

Continuation of the Proposed Exercise (5) *Component Brokerage System*

Develop the relevant OO Statechart Diagrams and Activity Diagrams for…

- This system essentially acts as a broker for software components. When developers have completed development of their software, they can deploy them as reusable software components for others to use. By connecting to the system over the Internet, these developers can submit their components to the system. An administrator then reviews the component for its functionality and ways of connecting to other components, categorizes it, and publishes it in a publicly-viewable area.
- Customers (such as other developers) can then connect to the public system and browse/search the components. When they have found something useful, they can download it for use on their own machine.
- Later, the providers of the components can connect to the system and view the download statistics of their components. They can also add/remove components from the system.

Continuation of the Proposed Exercise (6) *Bug Tracking System*

Develop the relevant OO Statechart Diagrams and Activity Diagrams for…

- Developers use this system to track bugs in an on-going software project. Developers who find bugs can submit a report. Other developers can then assign the bug to a particular developer (especially the developer responsible for the software module) to fix it. In addition, users can browse/search all the bugs in the system so far.
- An administrator manages users to restrict access to the bug tracking system. In addition, the administrator should also be able to generate reports on the state of the project in the form of a set of web pages updated daily at 2am.

Interaction diagrams

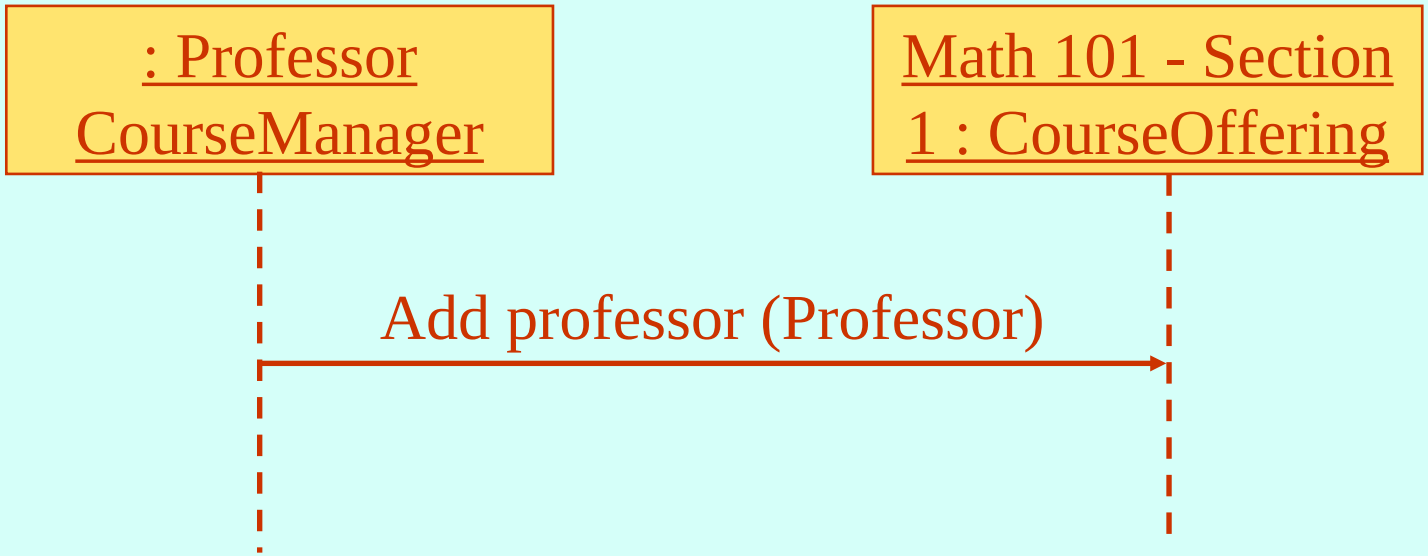
- Focus on “real” entities, the objects
- Details how object communicate one another
- 2 views:
  - Time-based view
  - Organization-based view

Sequence diagrams

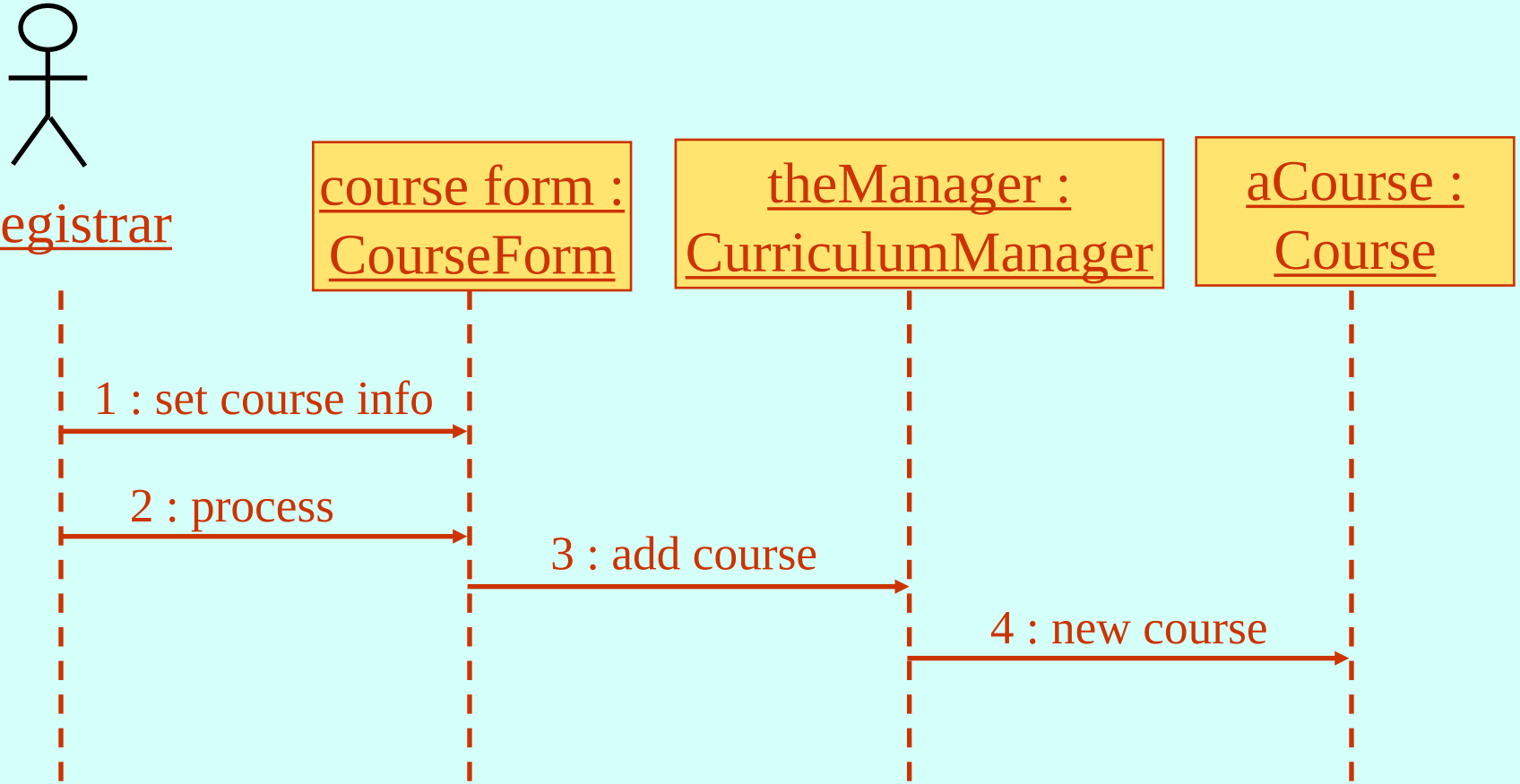
- Shows object interactions arranged in time sequence
- It focuses on
  - objects (and classes)
  - message exchange to carry out the scenarios functionality
- The objects are organized in an horizontal line and the events in a vertical **time** line

Timelines

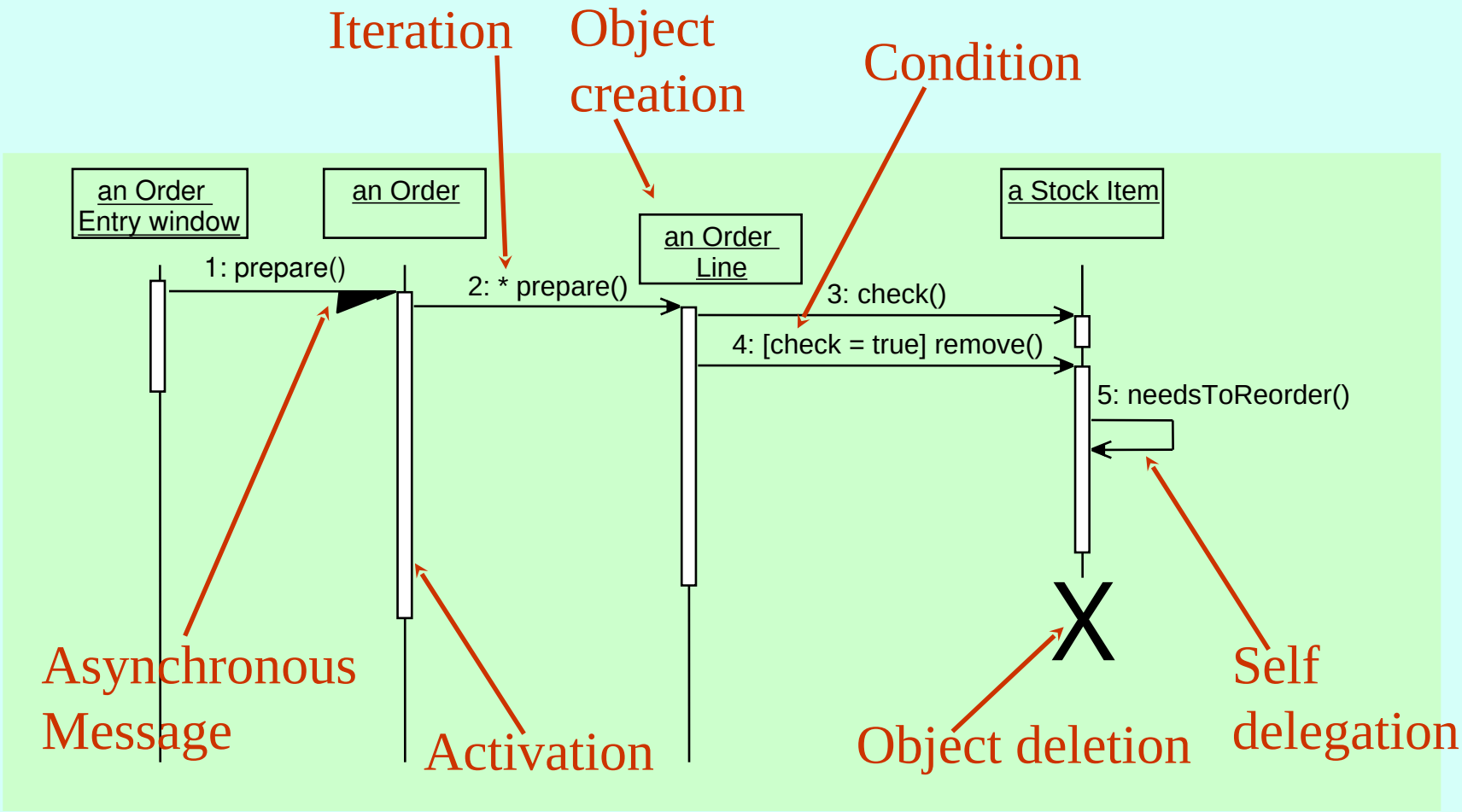
- Messages point from client to supplier



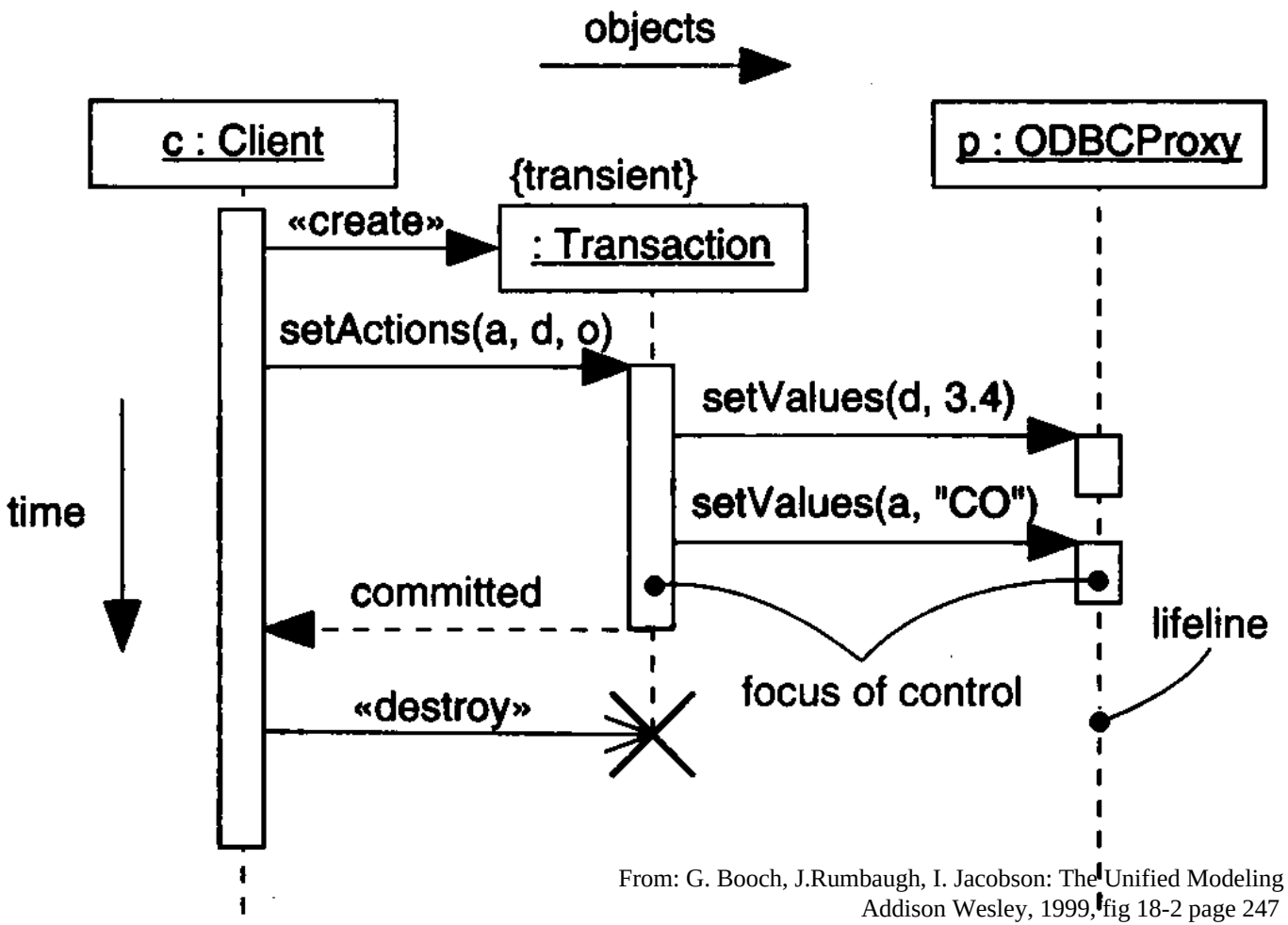
Example: Sequence diagram



Sequence diagrams: More details



Example of a transaction



From: G. Booch, J.Rumbaugh, I. Jacobson: The Unified Modeling Language User Guide. Addison Wesley, 1999, fig 18-2 page 247

Content of sequence diagrams

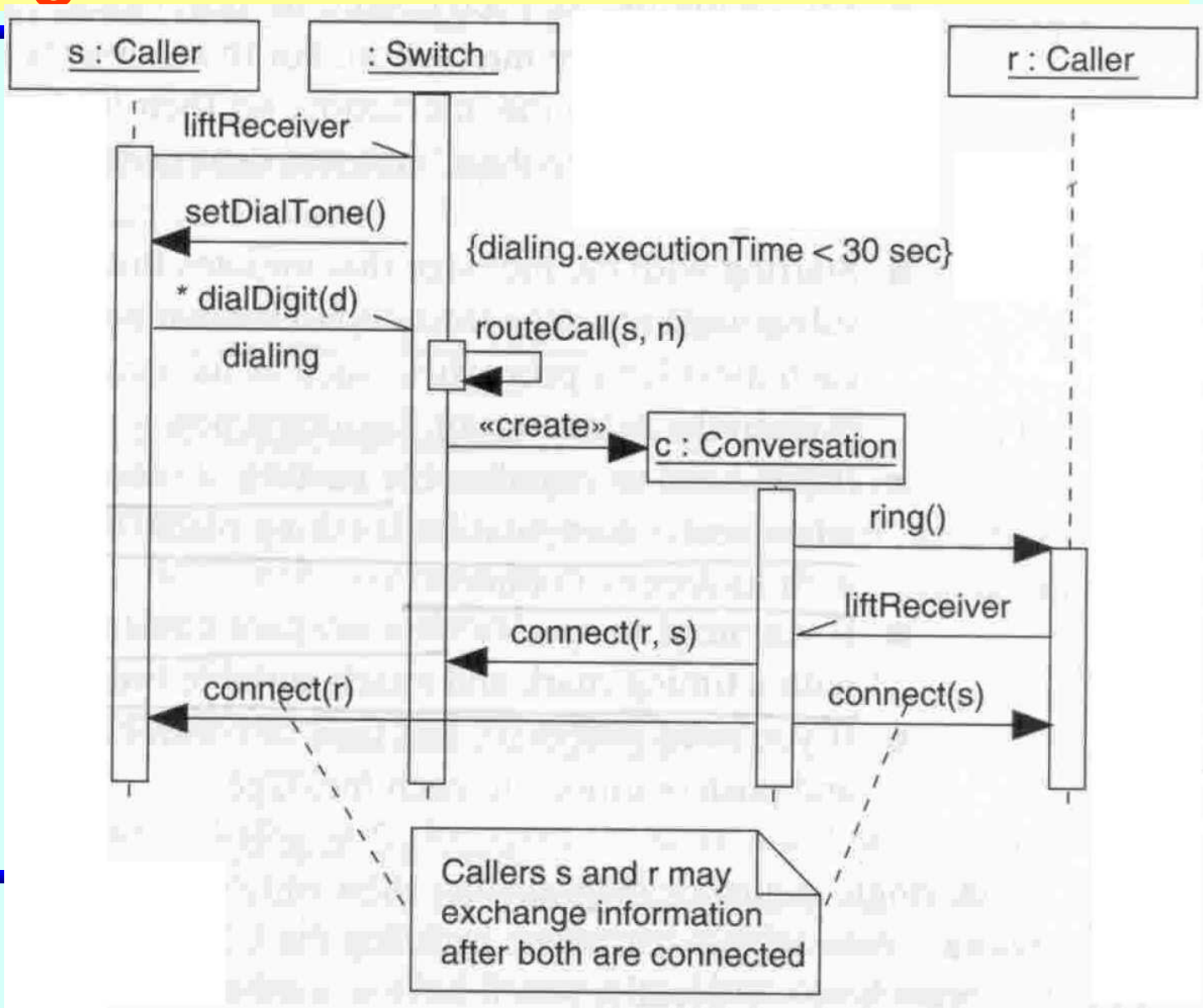
- Objects
  - they exchange messages among each-other
- Messages
  - Synchronous: “call events,” denoted by the full arrow
  - Asynchronous: “signals,” denoted by a half arrow
  - There are also «create» and «destroy» messages

Asynchronous messages

- Does not block the caller
- Can do 3 things:
  - Create a new thread
  - Create a new object
  - Communicate with a thread that is already running

Complete example of sequence diagrams

From: G. Booch, J.Rumbaugh, I. Jacobson: The Unified Modeling Language User Guide. Addison Wesley, 1999, fig 18-4 page 252



Complexity and sequence diagrams

- **Diagrams are meant to make things clear**
- KISS

  = keep it small and simple
- Conditional logic
  - simple: add it to the diagram
  - complex: draw separate diagrams

Proposed Exercises

- Use the sequence diagram to model how you receive the request to develop a new piece of code
- Use the sequence diagram to model how a message is sent over an Ethernet connection

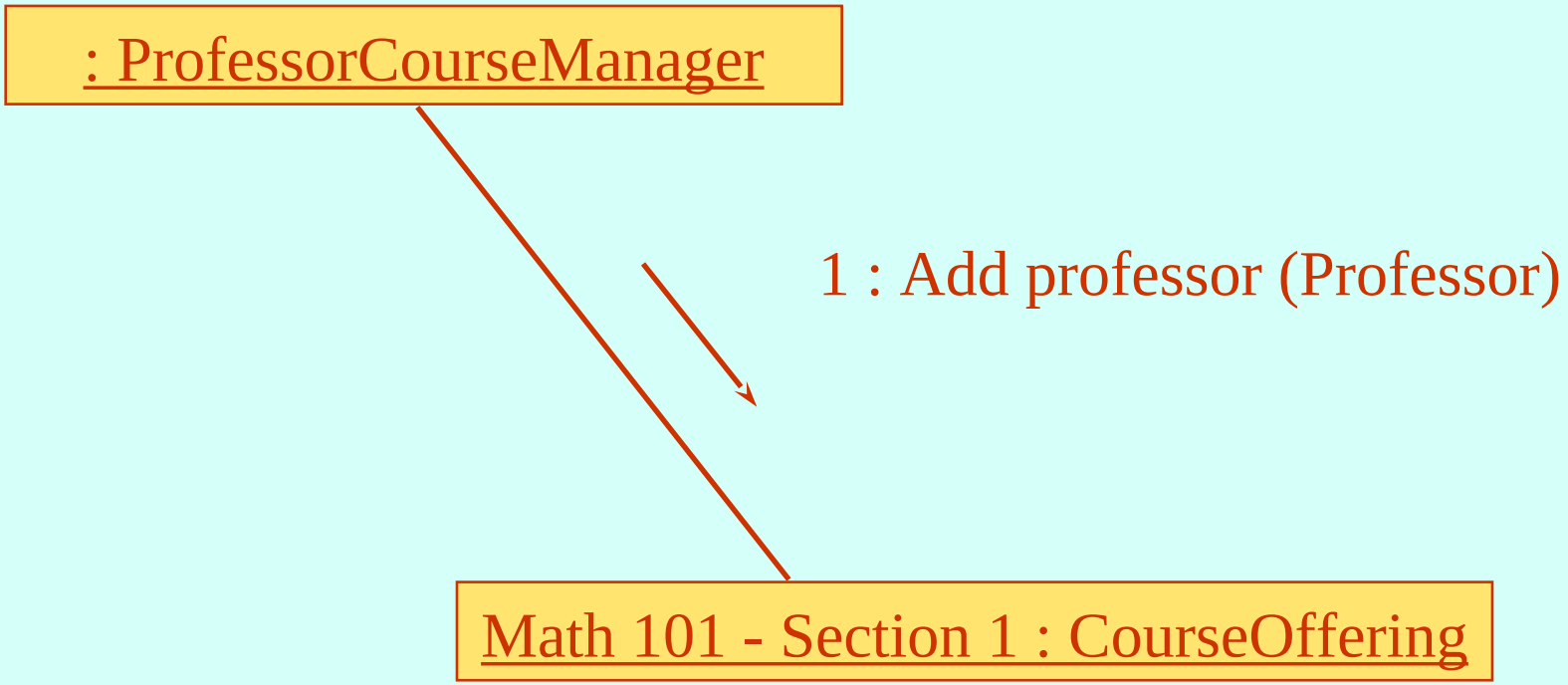
Where are the boundaries?

- A boundary shapes communication between system and outside world
  - e.g. user interface or other system
- It may be useful to show boundary classes in interaction diagrams:
  - capture interface requirements
  - do NOT show how the interface will be implemented

Collaboration diagrams

- Show how objects interacts with respect to organizational units (boundaries!)
- Sequence of messages determined by numbering
  - 1, 2, 3, 4, …..
  - 1, 1.1, 1.2, 1.3, 2, 2.1, 2.1.1, 2.2, 3 (shows which operation calls which other operation)

Collaboration diagram basics



Comparing sequence & collaboration diagrams

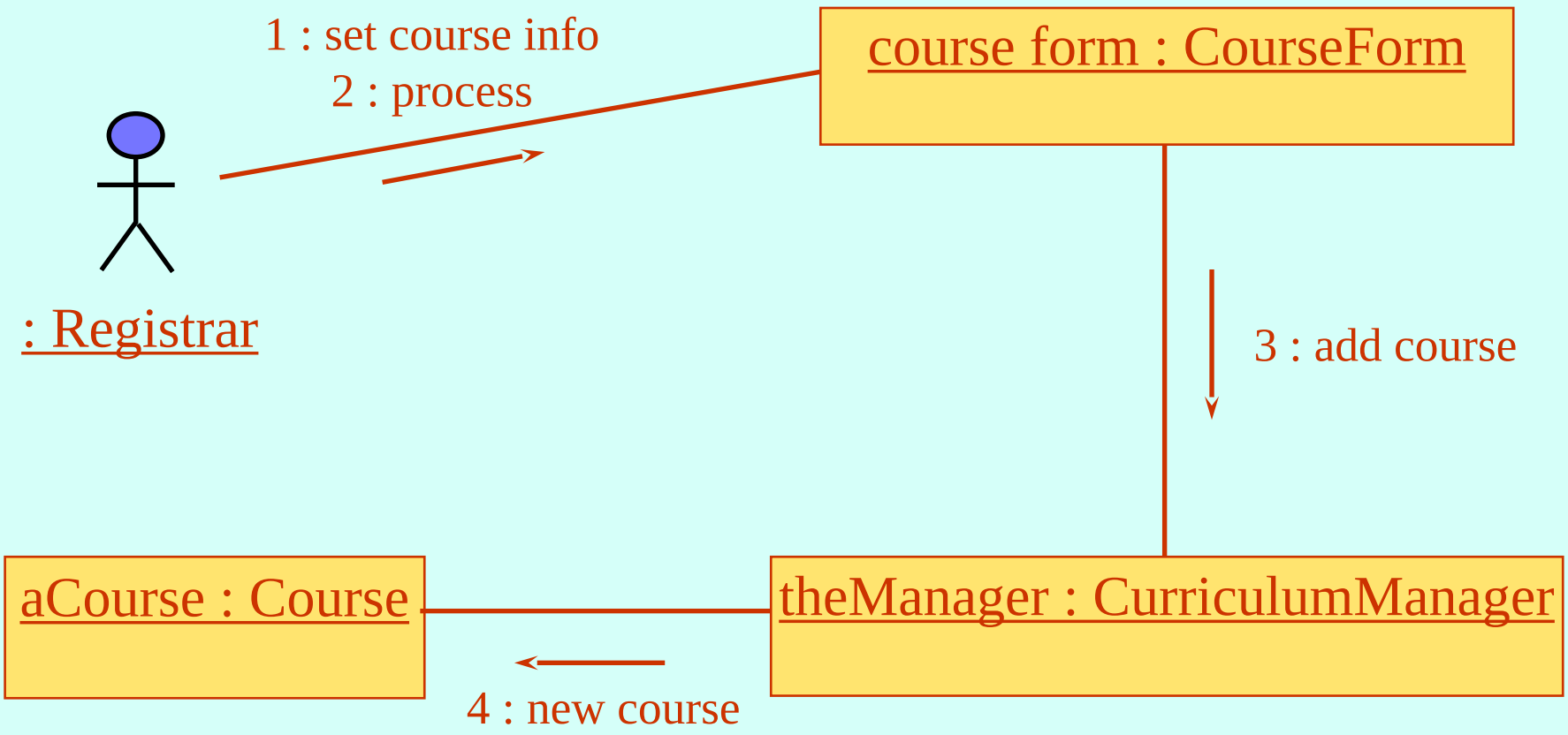
- Sequence diagrams are best to see the flow of time
  - Sequence of messages more difficult to understand in collaboration diagrams
- Flow of control by organization is best seen through collaboration diagrams
  - Layout of collaboration diagrams may show static connections of objects
- Complex control is difficult to express anyway!!!

Content of collaboration diagrams

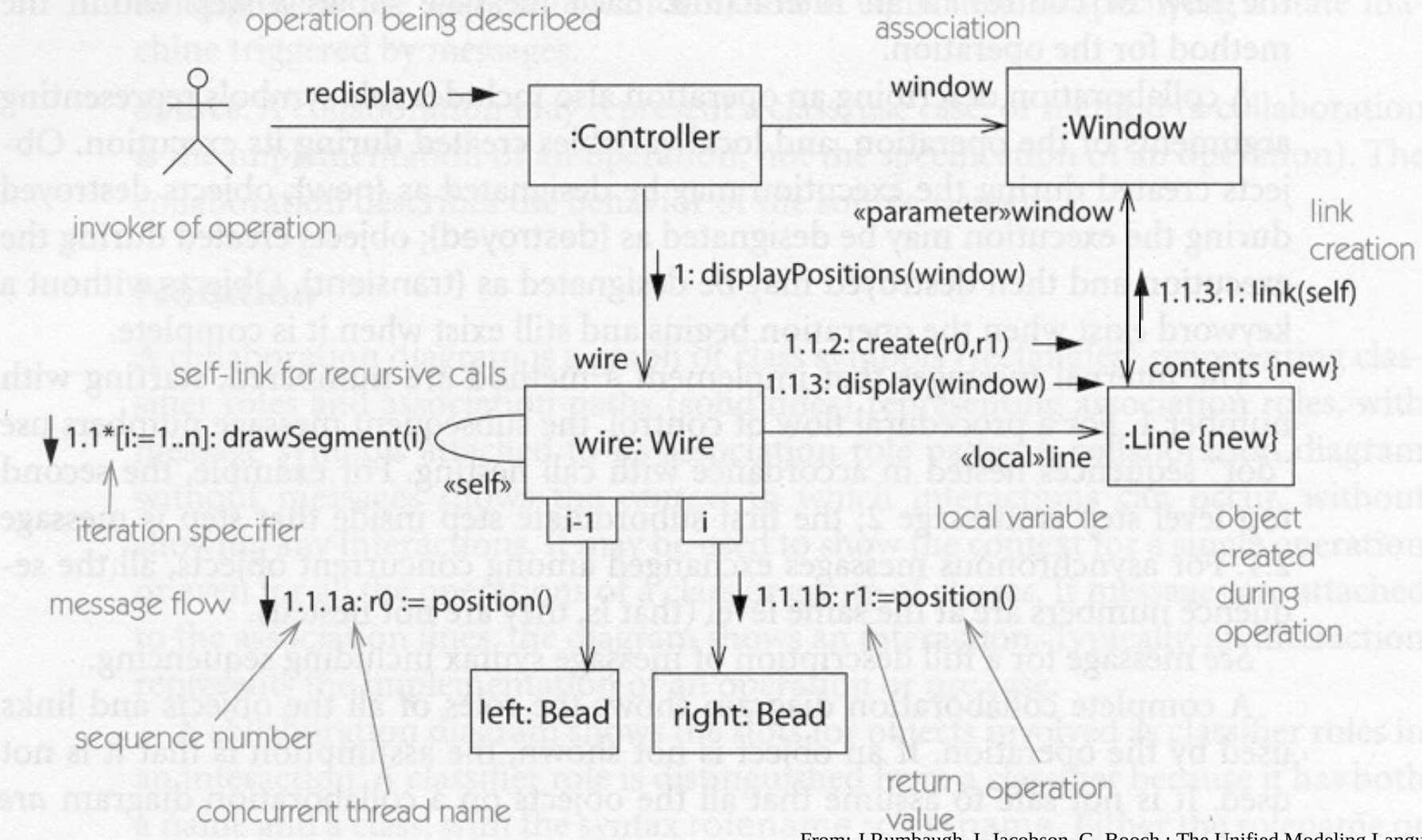
- Objects
  - they exchange messages among each-other
- Messages
  - Synchronous: “call events,” denoted by the full arrow
  - Asynchronous: “signals,” denoted by a half arrow
  - There are also «create» and «destroy» messages
- Messages are numbered and can have loops

Almost same stuff as sequence diagrams!!!!

Collaboration diagram example



Collaboration diagrams can become VERY complex…



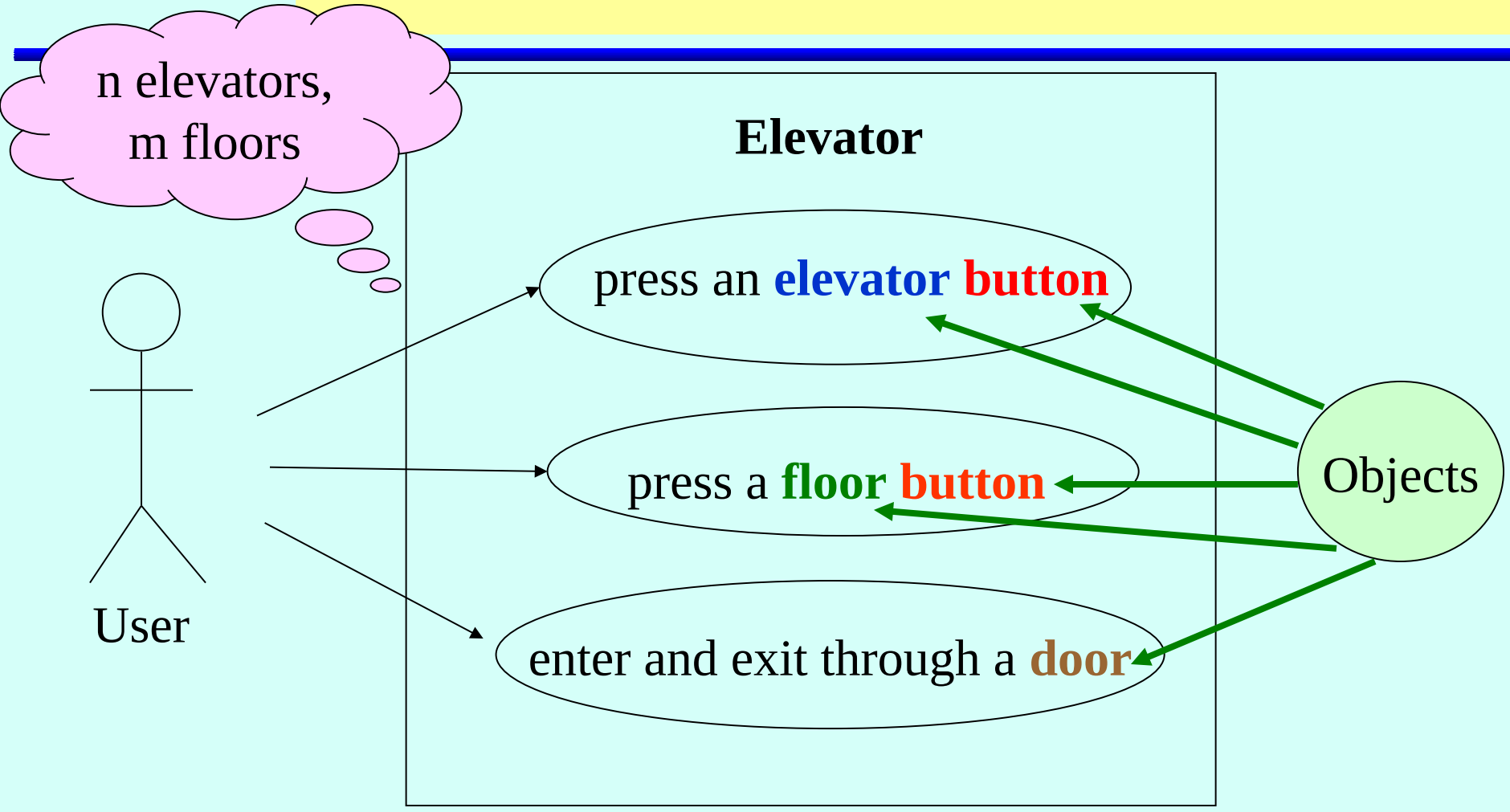
From: J.Rumbaugh, I. Jacobson, G. Booch : The Unified Modeling Language Reference Manual. Addison Wesley, 1999, fig 13-51 page 202

Proposed exercise

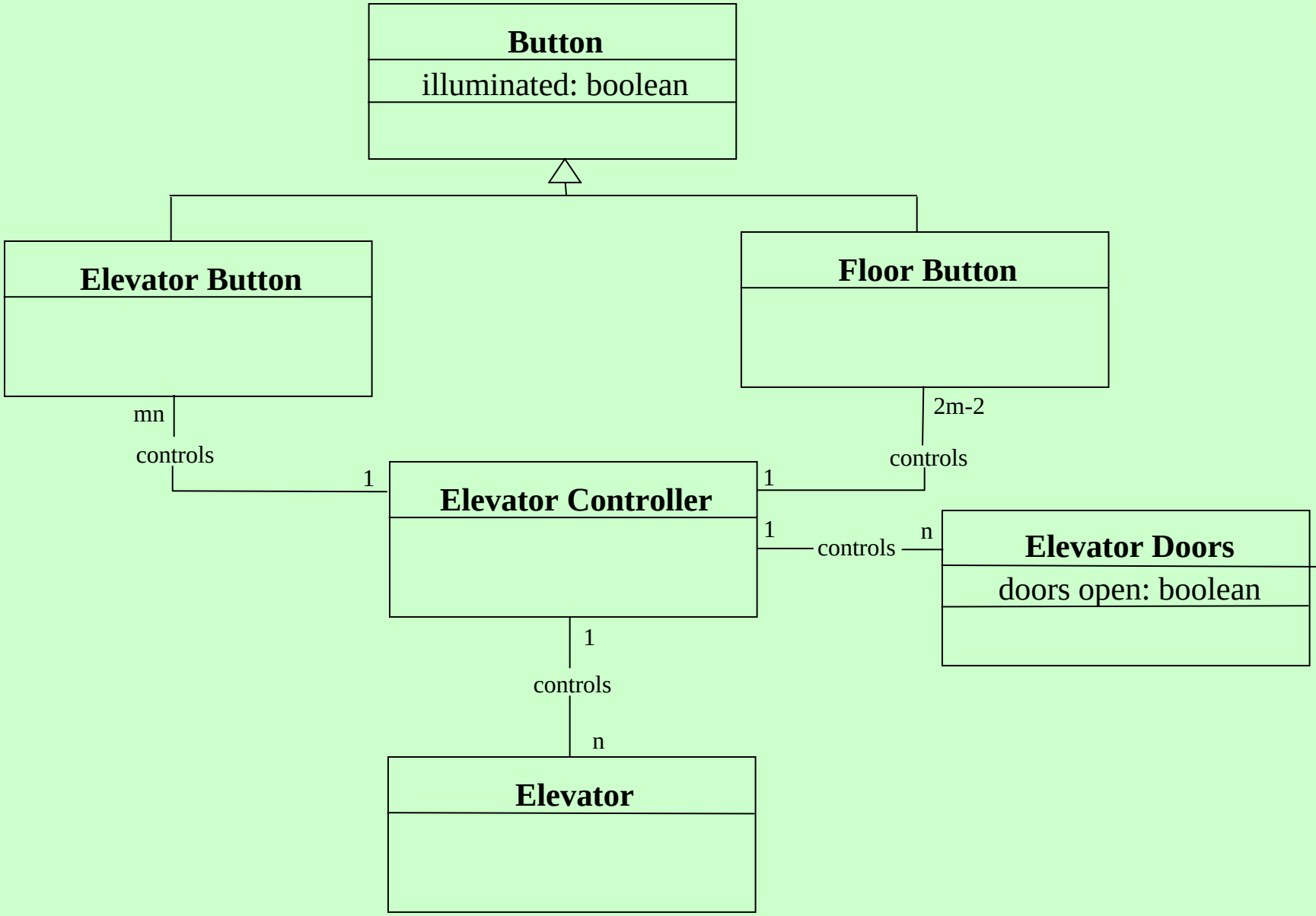
- Use the sequence diagram to model how you receive the request to develop a new piece of code -what is the difference with before?
- Use the sequence diagram to represent how a cellular phone moves from cell to cell and handle roaming

A Comprehensive Example: The Elevator

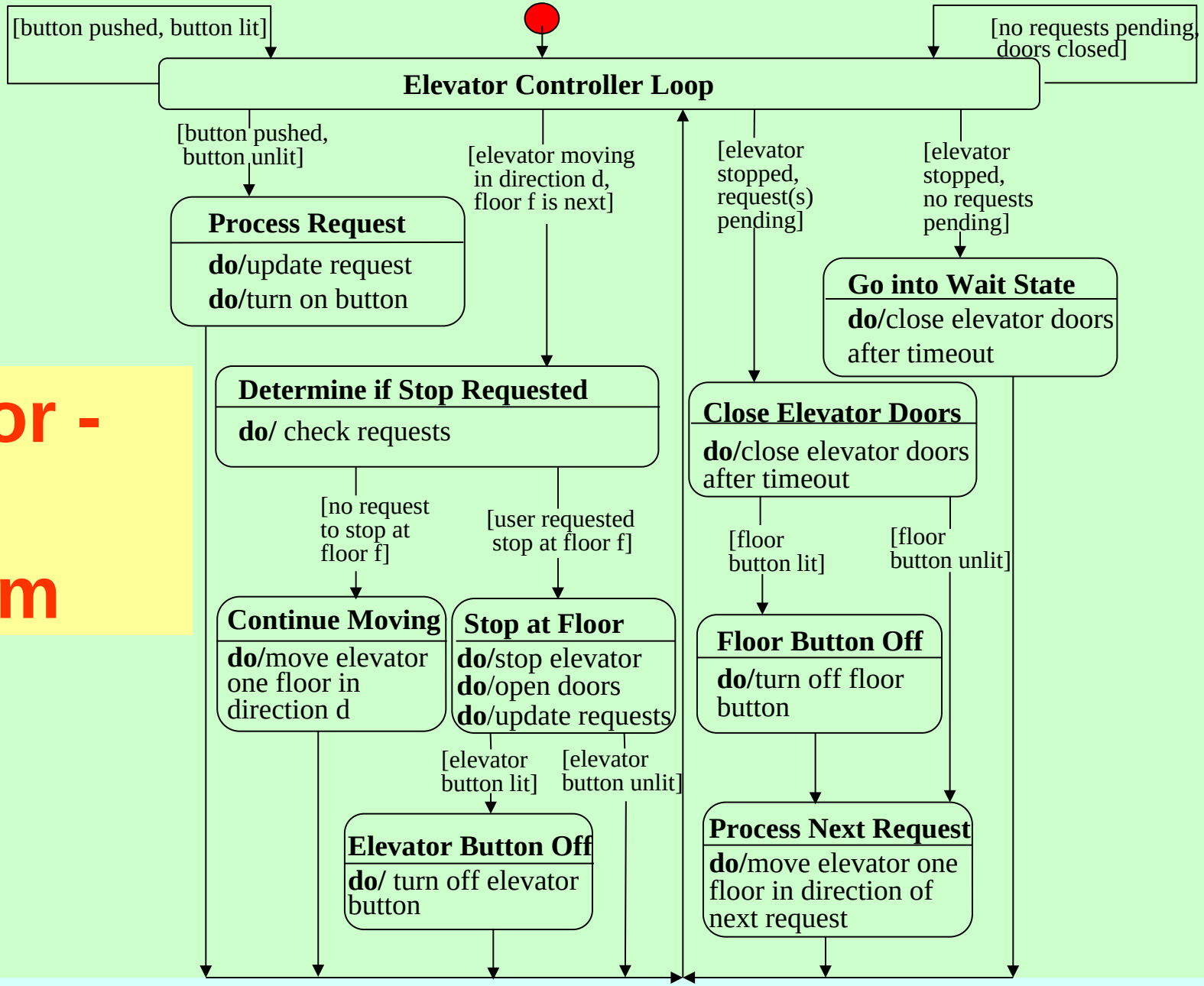
Elevator -- Use Case



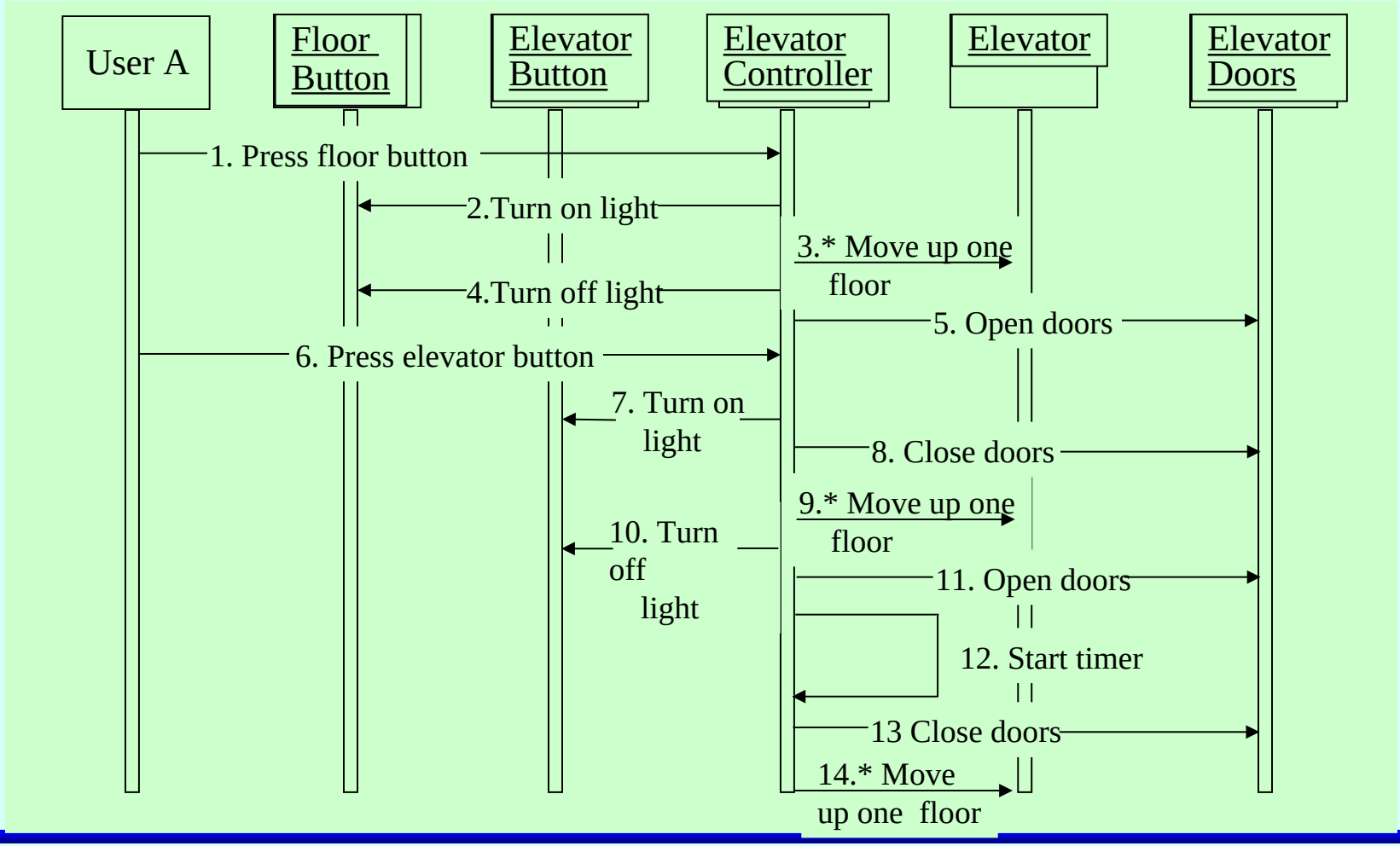
Elevator - First Class Diagram



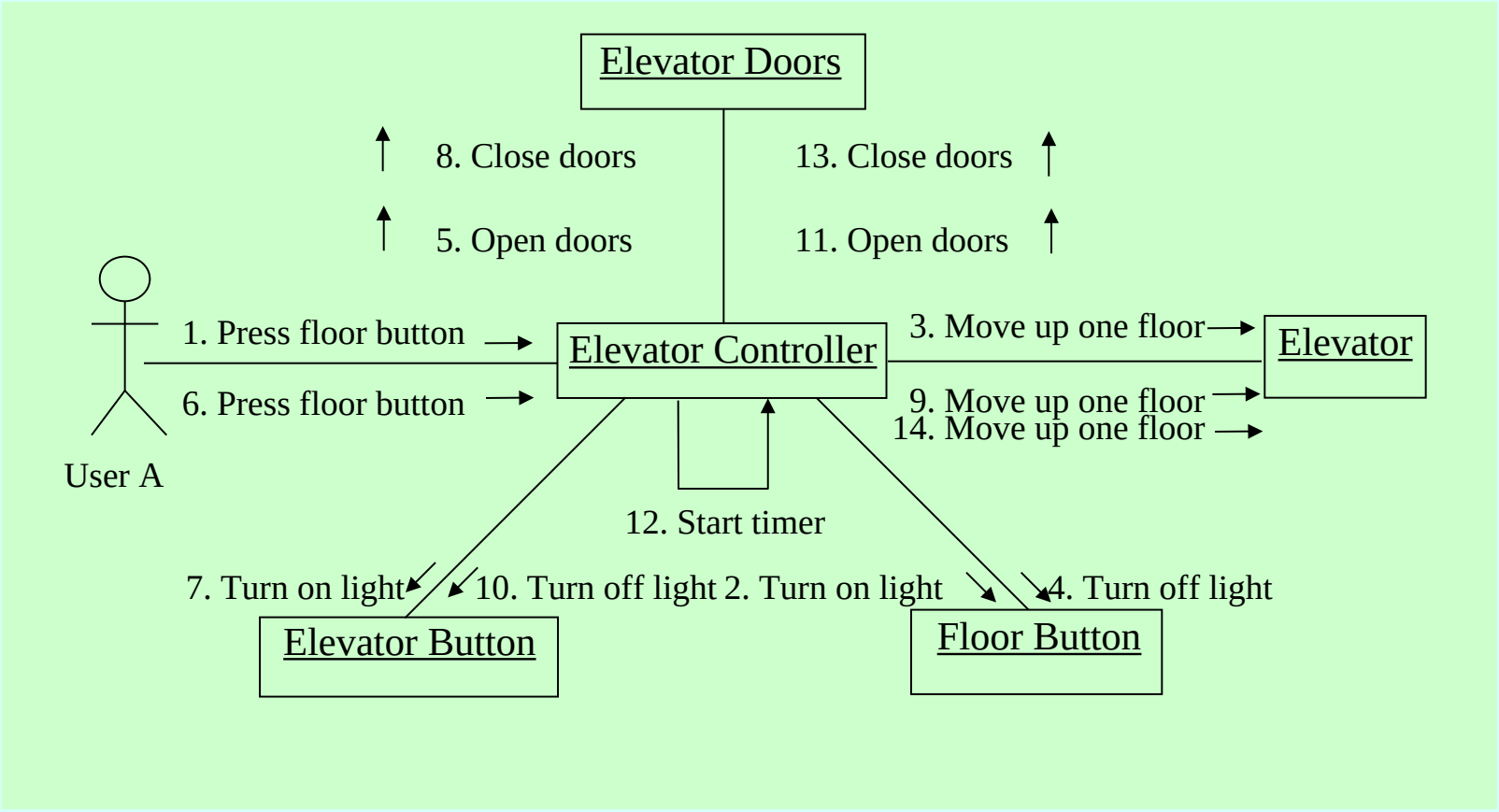
Elevator - State Diagram



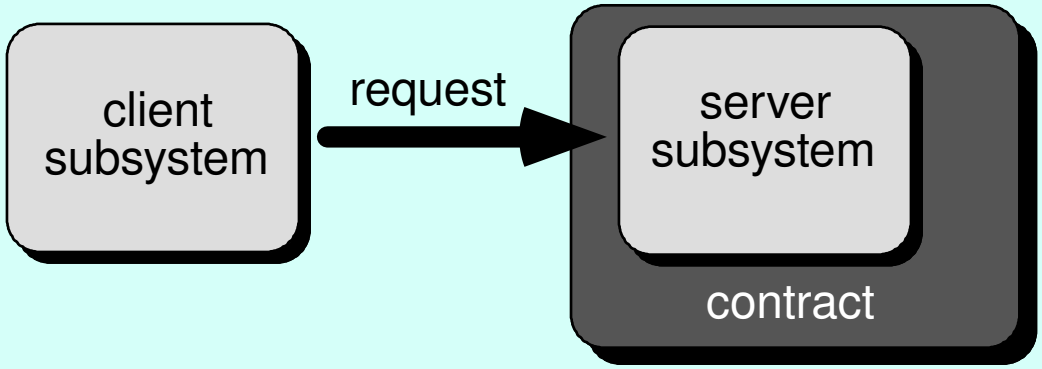
Elevator - Sequence Diagram

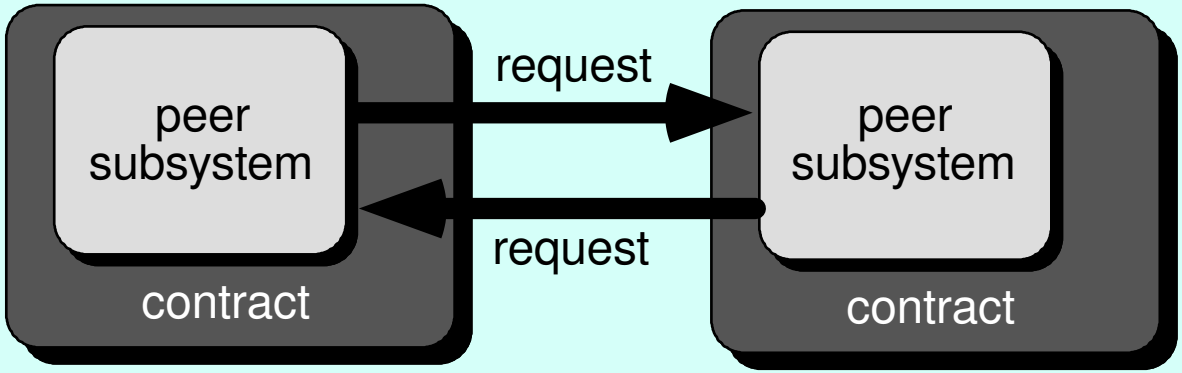


Elevator - Collaboration Diagram

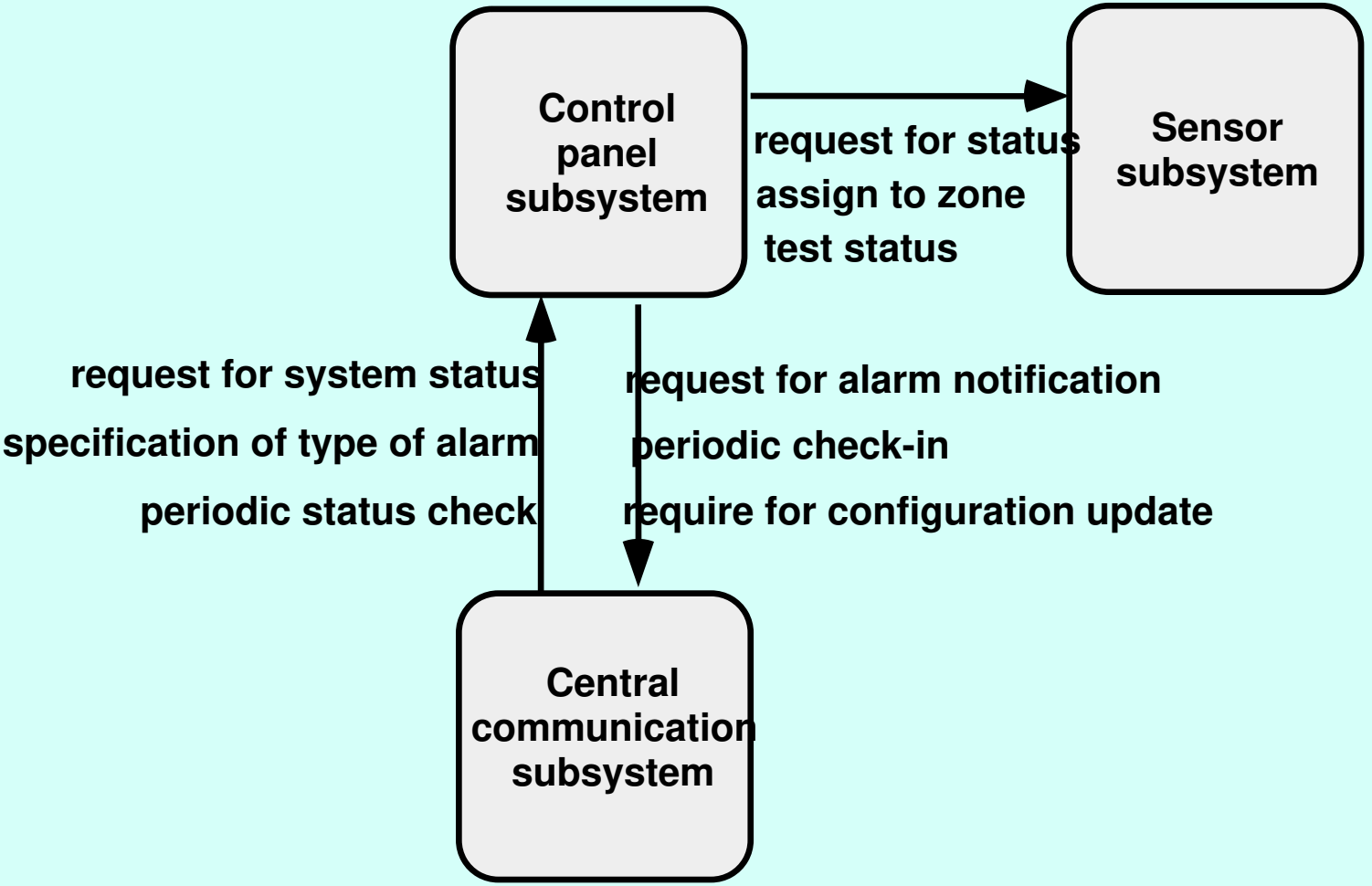


System Design





Systems and Sub-Systems



How to break a system into smaller subsystems?

- Roman principle: Divide & conquer
  - Split up a large system into manageable parts
- Structured methods: functional decomposition
- OO: Group classes into higher level units

Packages / Components

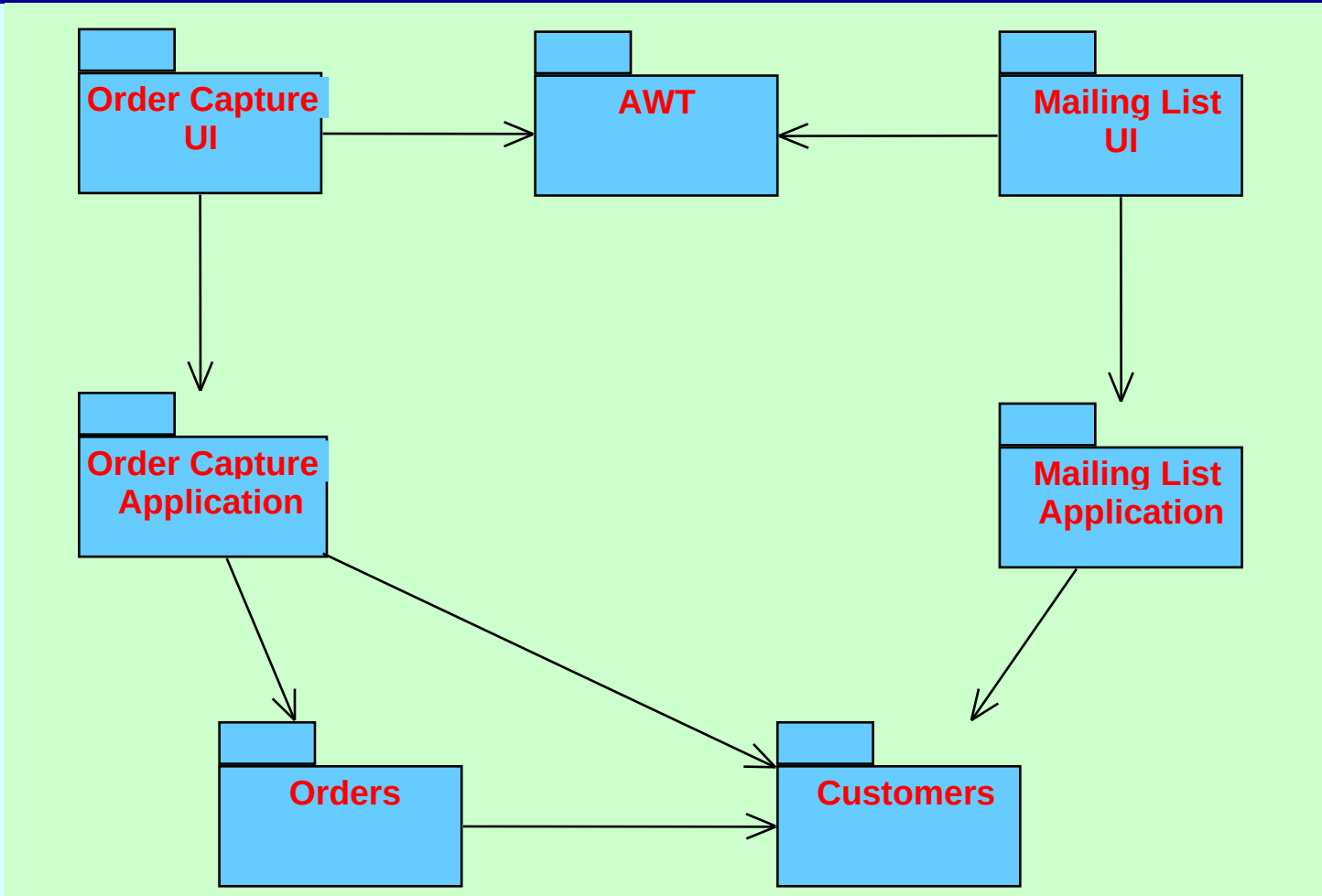
Packages

- Show packages
- Show dependencies between packages

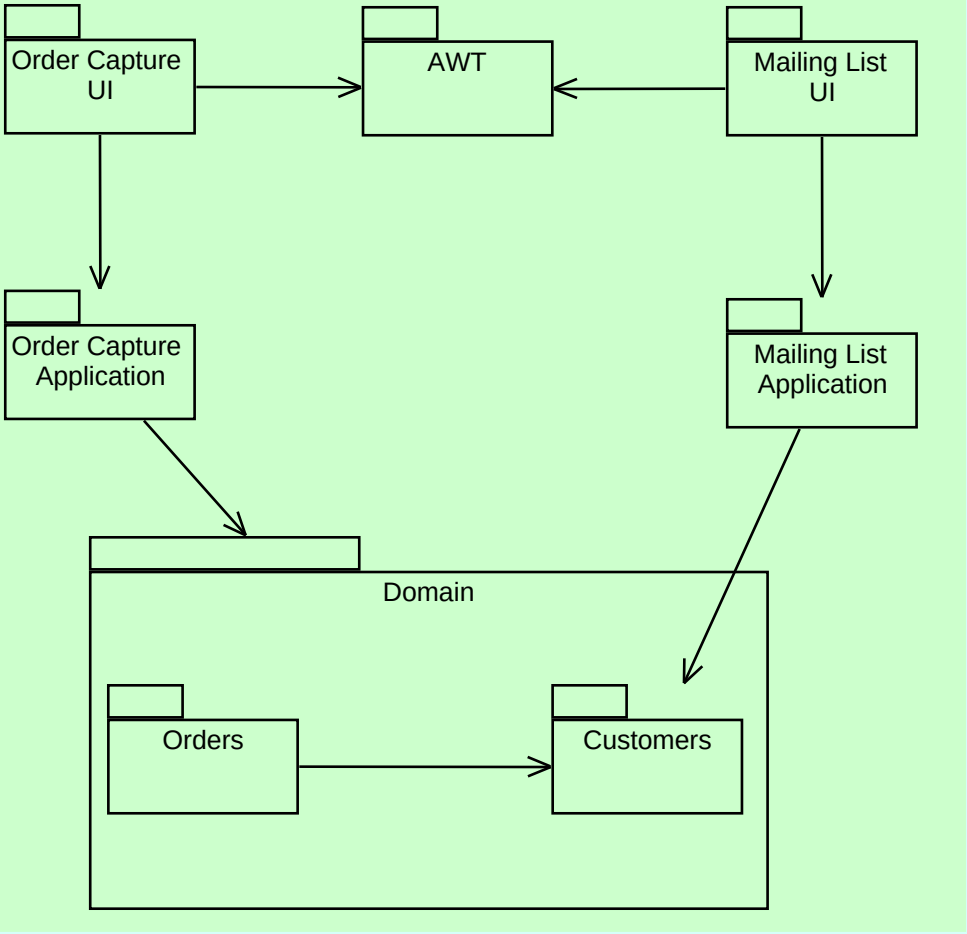
  Dependency exists between 2 elements if changes to the definition of one element may cause changes to the other
- Goal (& art) of large scale design: Minimize dependencies
  - Constraints effects of changes

Example of packages

A System to mail and manage orders



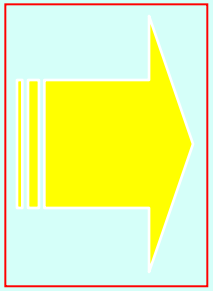
Nested packages



- 2 interpretations
  - transparent: all contents of contained packages visible
  - opaque: only classes of the container are visible

How can the complexity of a package interface be restricted?

- Give all classes in the package only package visibility
- Define a public class that provides the public behavior of the package
- Delegate the public operations to the appropriate class inside the package



Facades [Gamma et al. 1994]

Rules of thumb

- Try to avoid cycles in the dependency structure
- To many dependencies: Try to refactor the system
- Use them when the system class diagram is not legible on a single letter size sheet of paper

Packages vs. Diagrams

- Packages represent physical divisions of the development to do
  - ☑ Their goal is to ease the definition of workpackages and to develop software systems
- Diagrams represent logical assembly of information
  - ☑ Their goal is ease the understanding of the target domain

A Glimps on Design Patterns (1/2)

... you'll find recurring patterns of classes and communicating objects in many object-oriented systems. These patterns solve specific design problems and make object-oriented design more flexible, elegant, and ultimately reusable. They help designers reuse successful designs by basing new designs on prior experience. A designer who is familiar with such patterns can apply them immediately to design problems without having to rediscover them.

Gamma and his colleagues

More to follow (1/2)

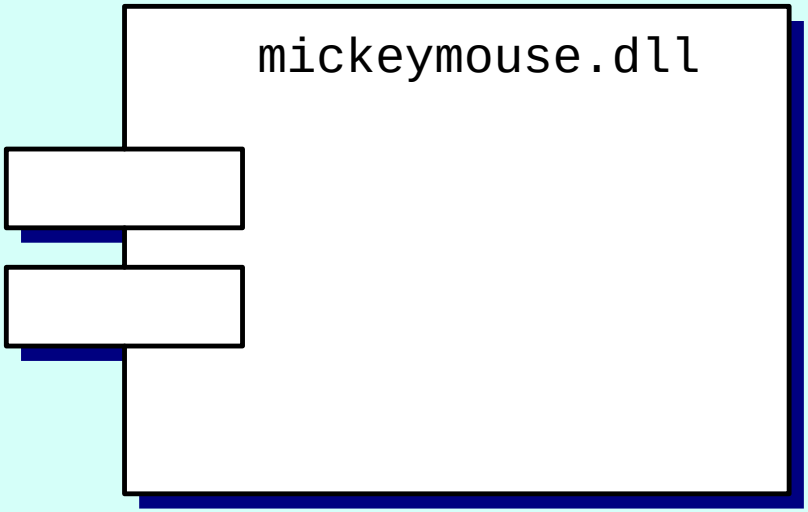
A Glimps on Design Patterns (2/2)

- The design pattern name is an abstraction that conveys significant meaning about it applicability and intent.
- The problem description indicates the environment and conditions that must exist to make the design pattern applicable.
- The pattern characteristics indicate the attributes of the design that may be adjusted to enable the pattern to accommodate into a variety of problems.
- The consequences associated with the use of a design pattern provide an indication of the ramifications of design decisions.

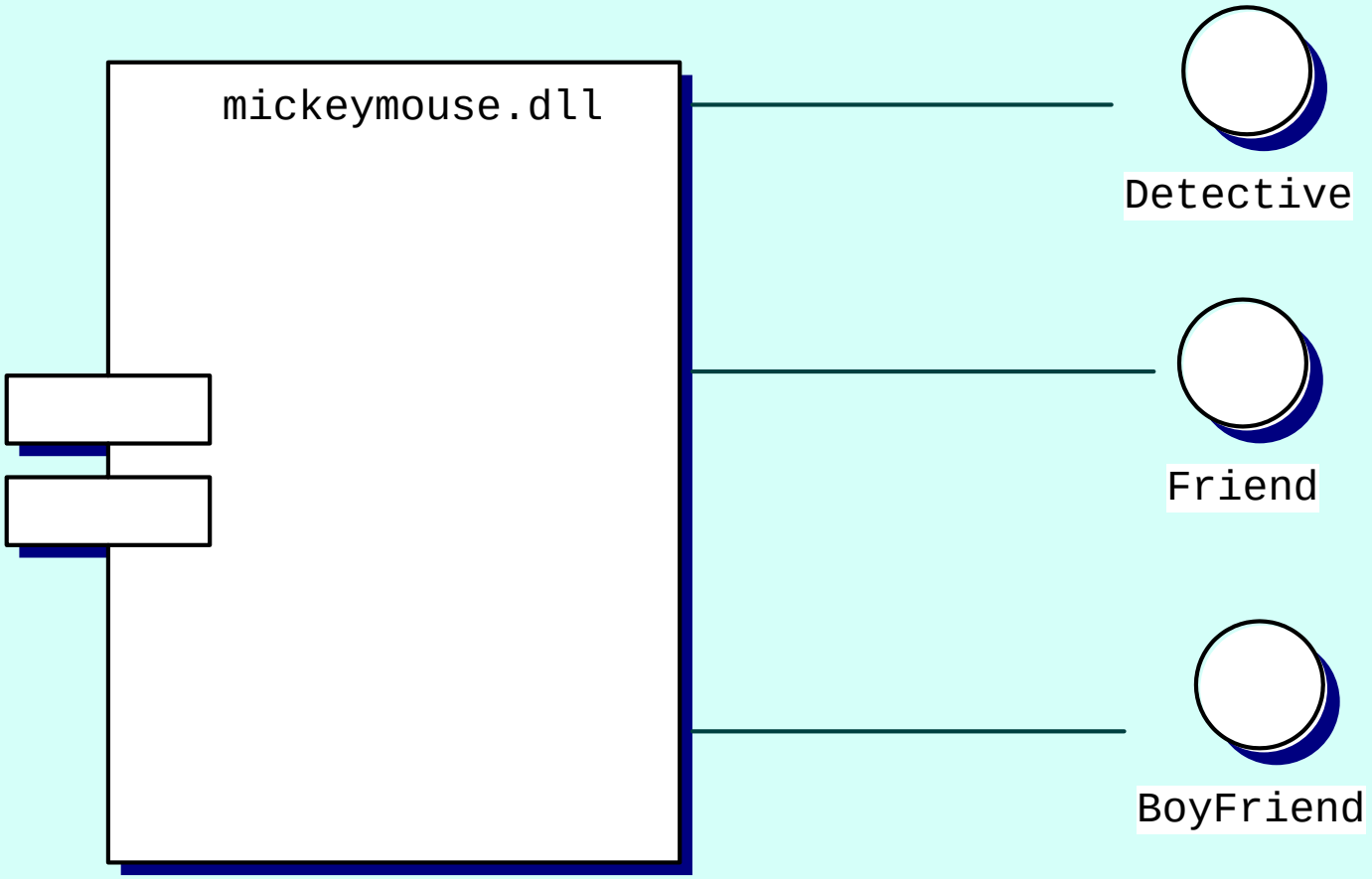
More to follow (2/2)

Components in UML

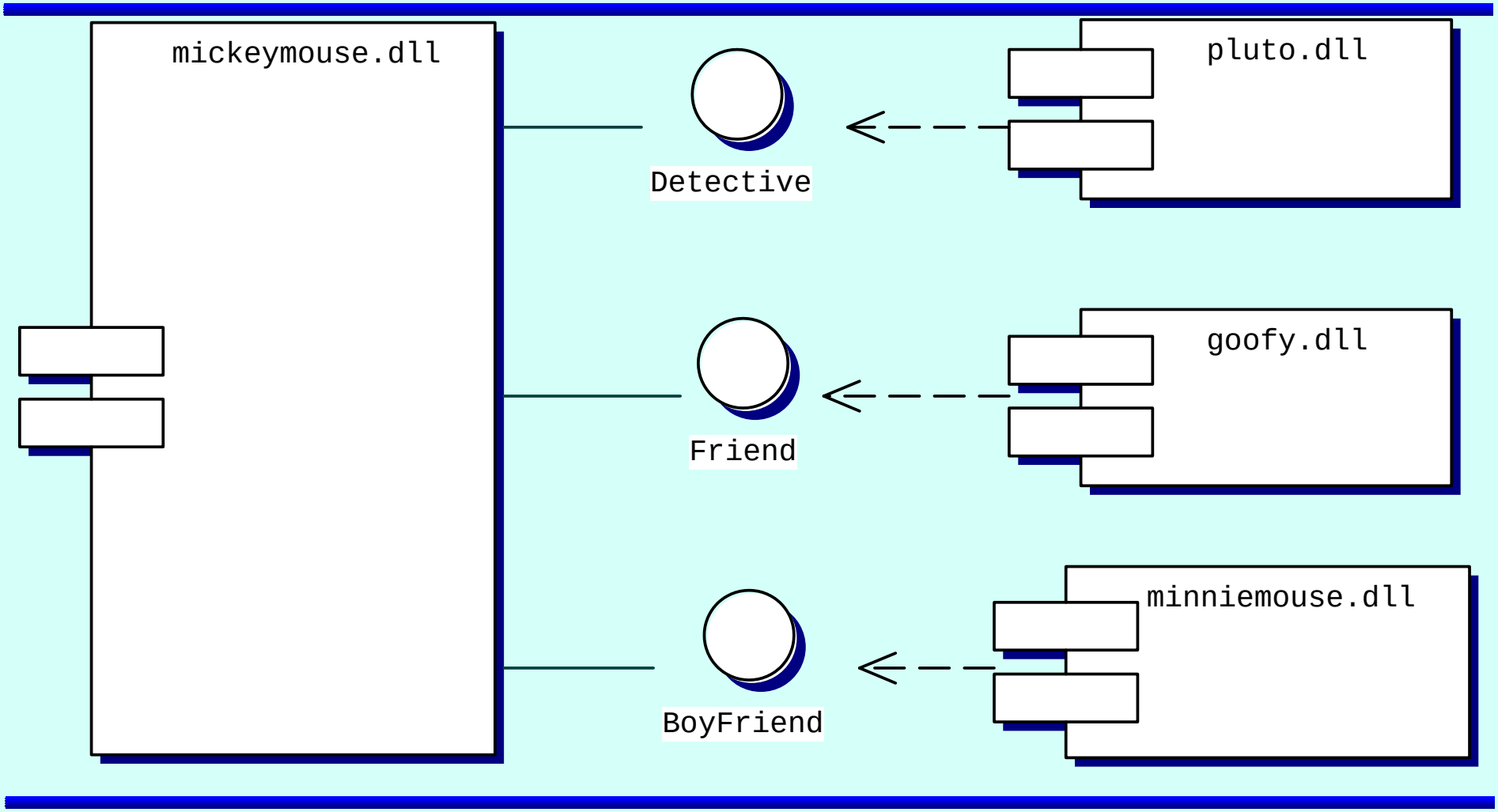
How components are represented in UML



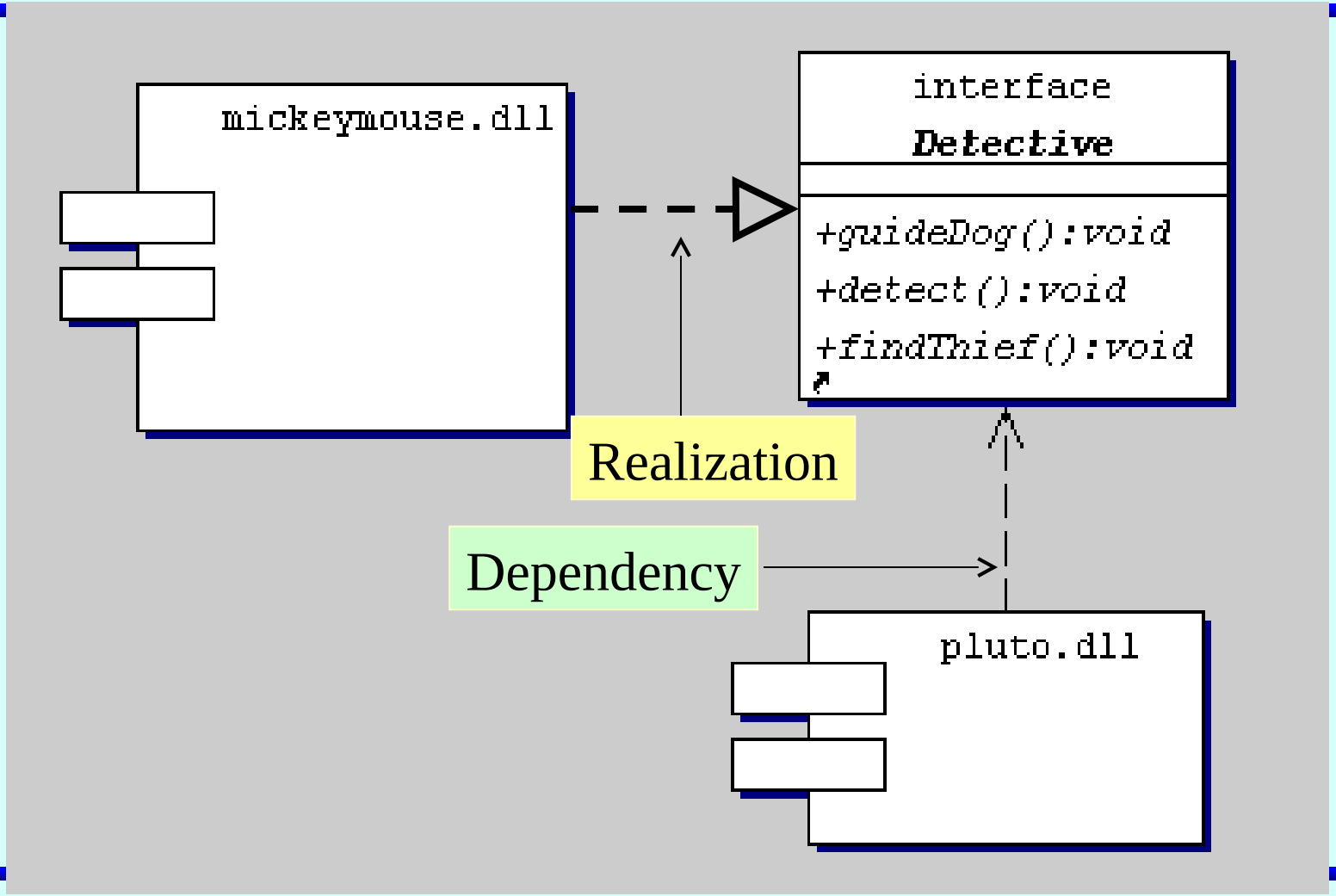
Components and interfaces



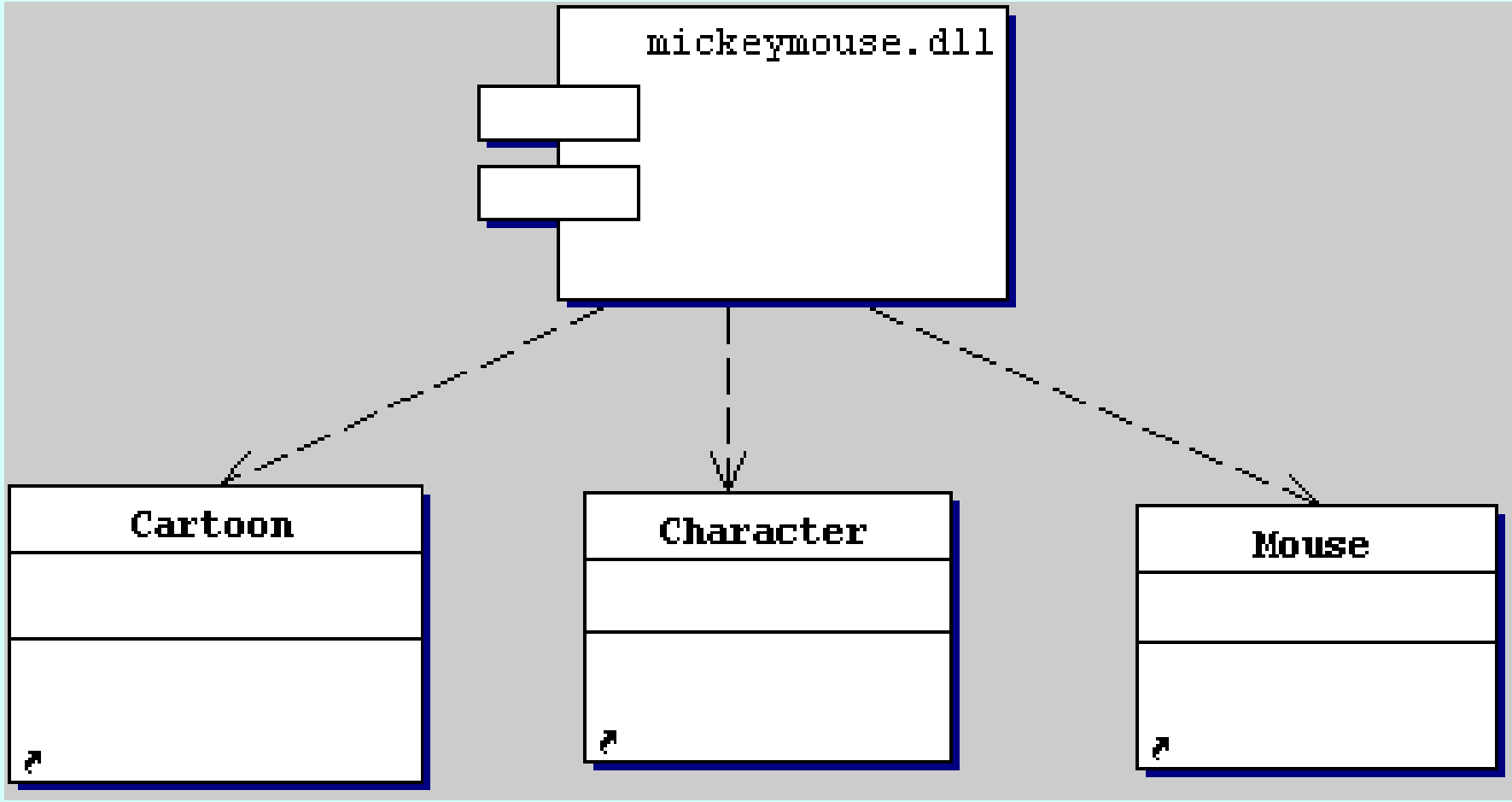
Components and Collaborations



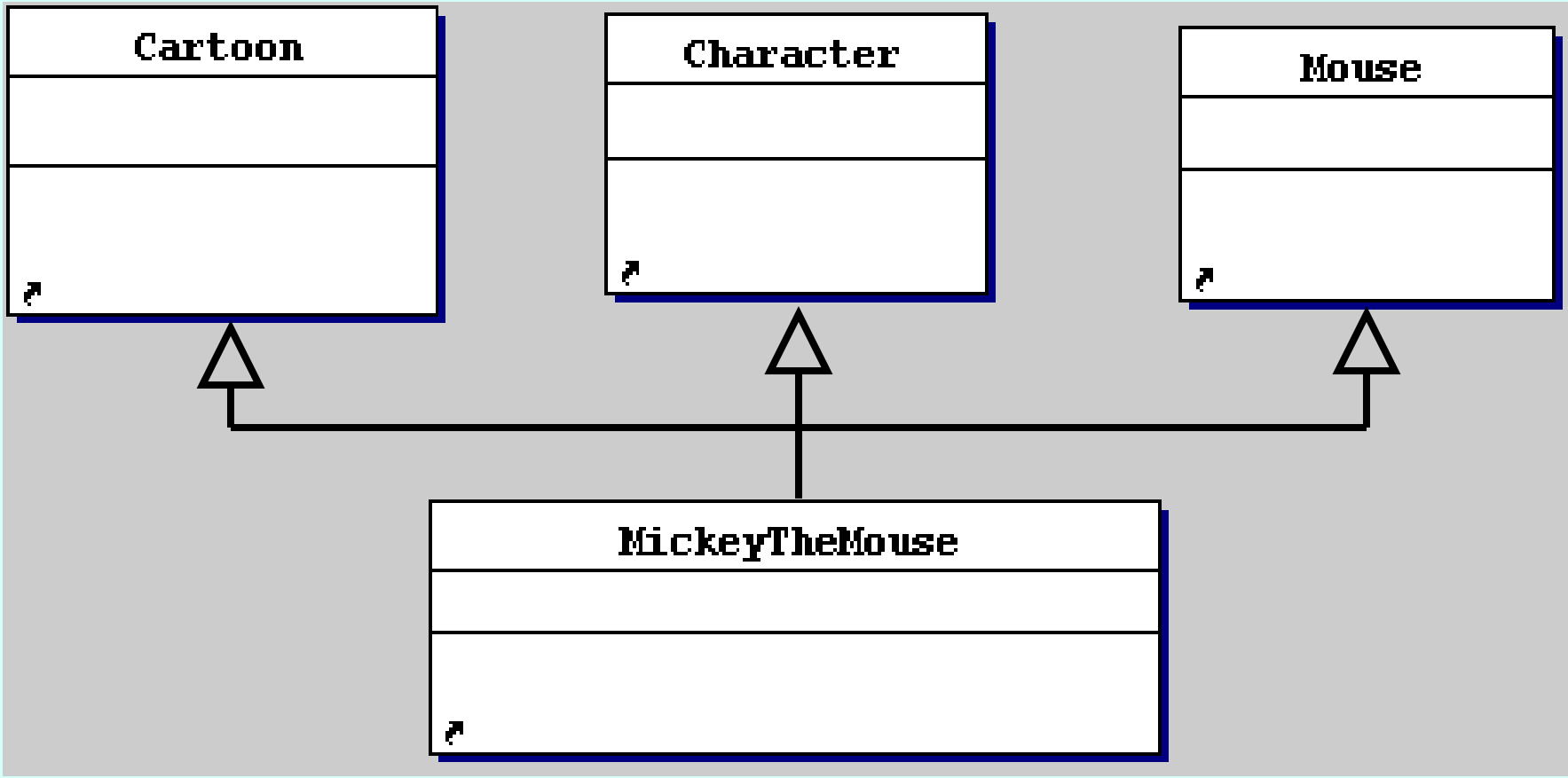
Detailed View of a Collaboration



Components and classes



Something totally different

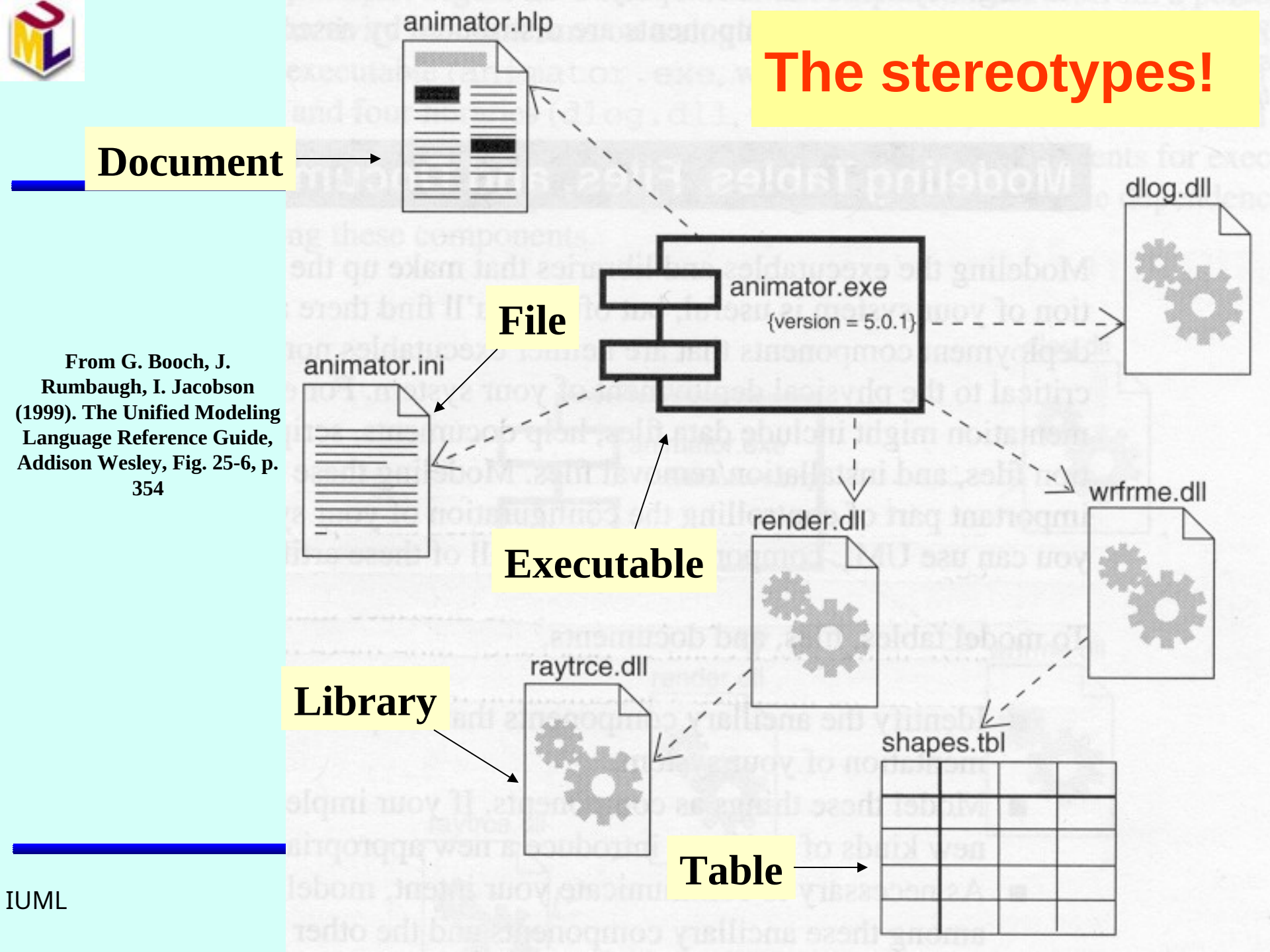


Stereotypes for components

UML defines five kinds of stereotypes that can be applied to components:

- executable
- library
- table
- file
- document

The stereotypes!



From G. Booch, J. Rumbaugh, I. Jacobson (1999). The Unified Modeling Language Reference Guide, Addison Wesley, Fig. 25-6, p. 354

We can define our own icons...

Detective

Friend

Boyfriend

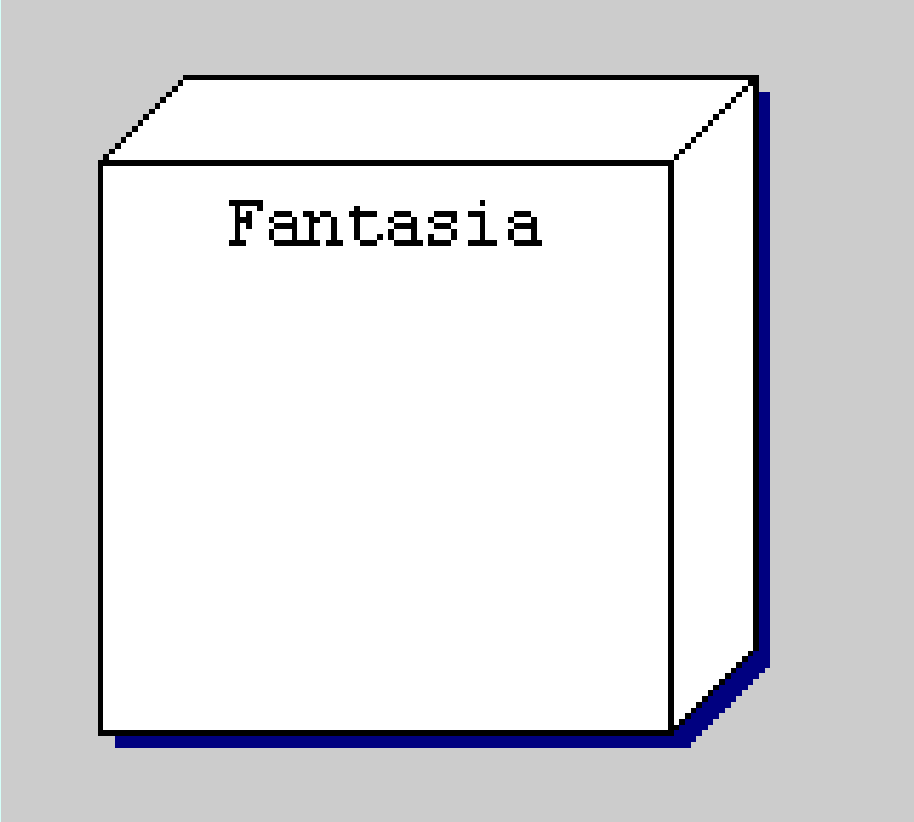
The icons are from the Walt Disney web site

Nodes

A node is a physical element that exists at run time and represents a computational resource, generally having at least some memory and, often, processing capabilities.

From G. Booch, J. Rumbaugh, I. Jacobson (1999). The Unified Modeling Language Reference Guide, Addison Wesley, p. 360

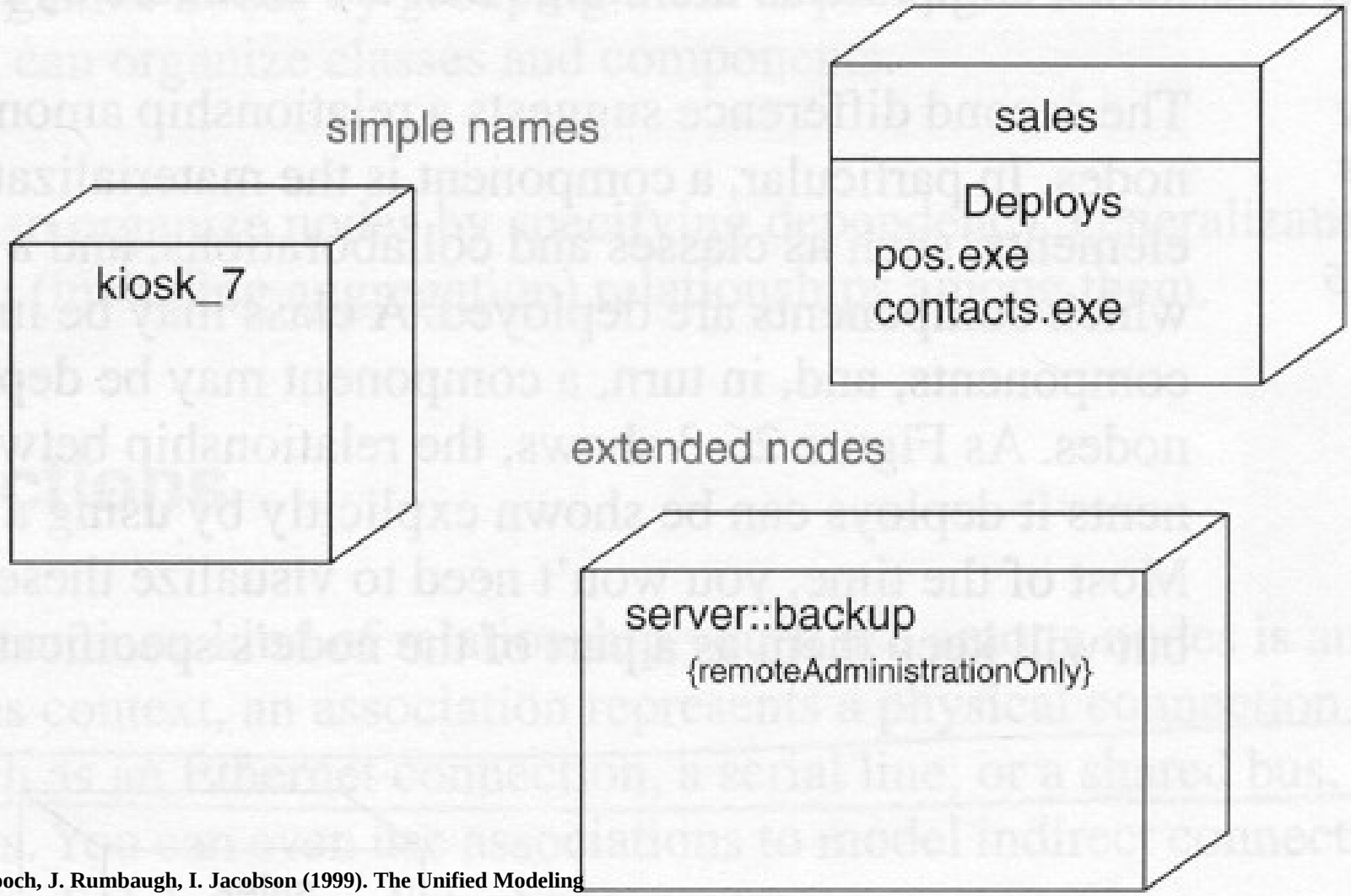
Graphical representation of a node in UML



Icons can be associated to nodes...

The icon is from the Walt Disney web site

Nodes can be identified by extended names

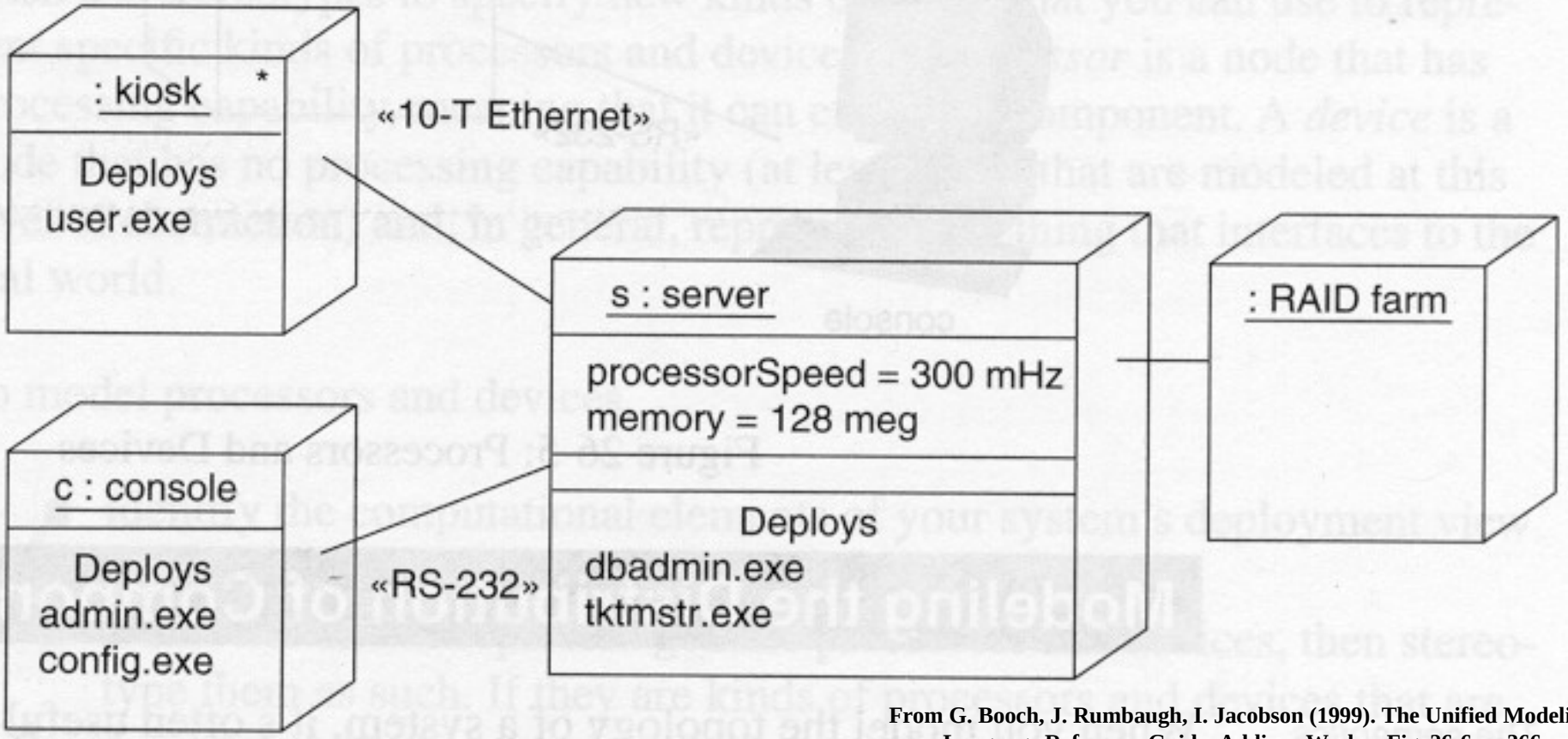


From G. Booch, J. Rumbaugh, I. Jacobson (1999). The Unified Modeling Language Reference Guide, Addison Wesley, Fig. 26-2, p. 361

Components and Nodes

- Components are like nodes (have names; participate in dependency, generalization, and association relationships; may be nested; may have instances; may participate in interactions; …).
- However, components are things that participate in the execution of a system; nodes are things that execute components.
- Components represent the physical packaging of otherwise logical elements; nodes represent the physical deployment of components .

A Deployment Diagram with Nodes containing Components…



From G. Booch, J. Rumbaugh, I. Jacobson (1999). The Unified Modeling Language Reference Guide, Addison Wesley, Fig. 26-6, p. 366

Real Time System Analysis and Design in UML

Real Time UML

Dealing with UML we have to consider 2 different aspects:

1. The features of UML that can be used to support Real Time Applications
2. The proposed UML extensions that handle Real Time Applications -the self-extending features of UML can be used for this purpose

Real Time Constructs in UML

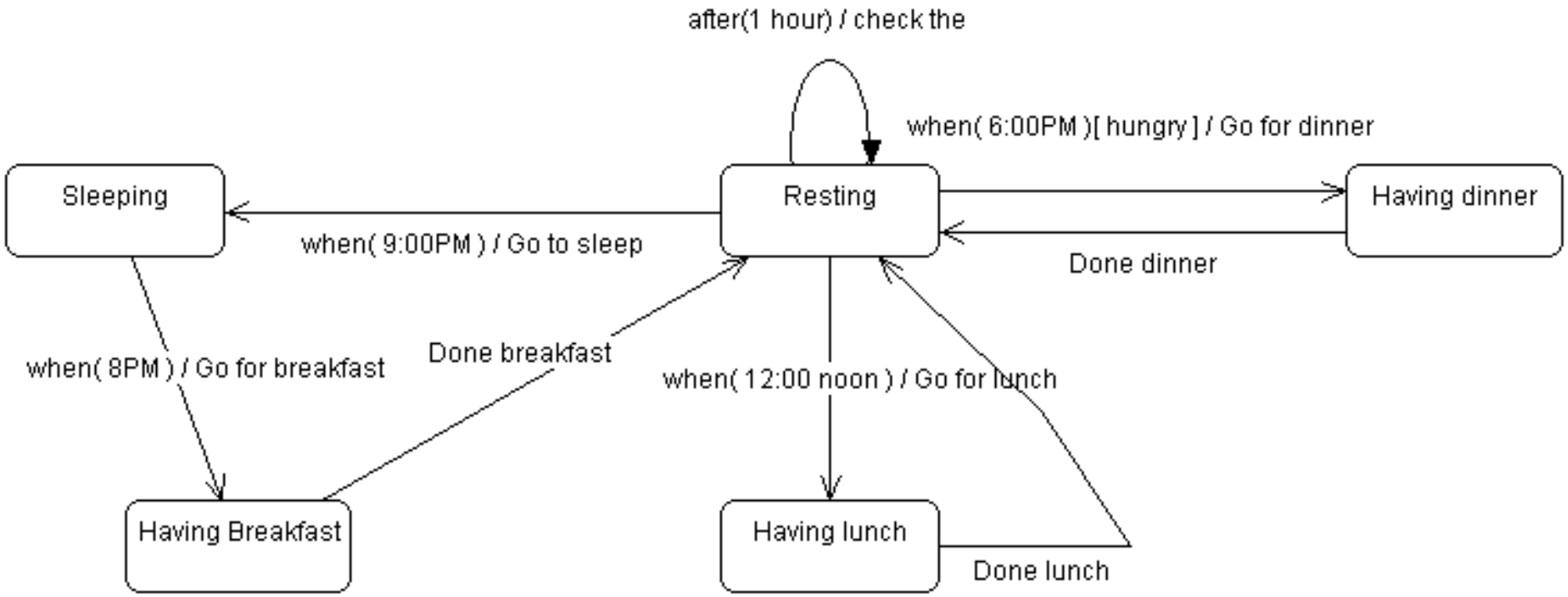
UML has features to support Real-Time applications:

- Change Event
- Time Event
- Timing constraints

Time and Change Events

- Time and change events specify when action has to be performed at a given instant of time
- “when” is a change event that specifies the date/time when the event will occur
  - when (12:00AM) / goForLunch()
- “after” is a change event that specifies after how much time a given event will be executed
  - after(5 minutes) / put the egg in the pot

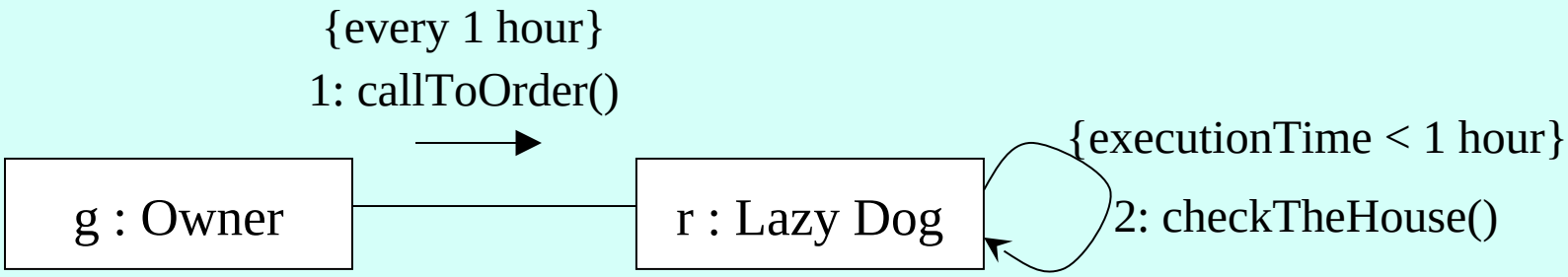
Change and Time Events - Example



Timing Constraints

- Timing constraints can be added to messages:
  - there are three “built-in” variables that can be used, startTime, stopTime, and executionTime
  - the “every” construct identifies a message that is resent at given intervals of time

Example of Timing Constraints



Unfortunately

- No existing tool or programming language enforce such constraints!!!

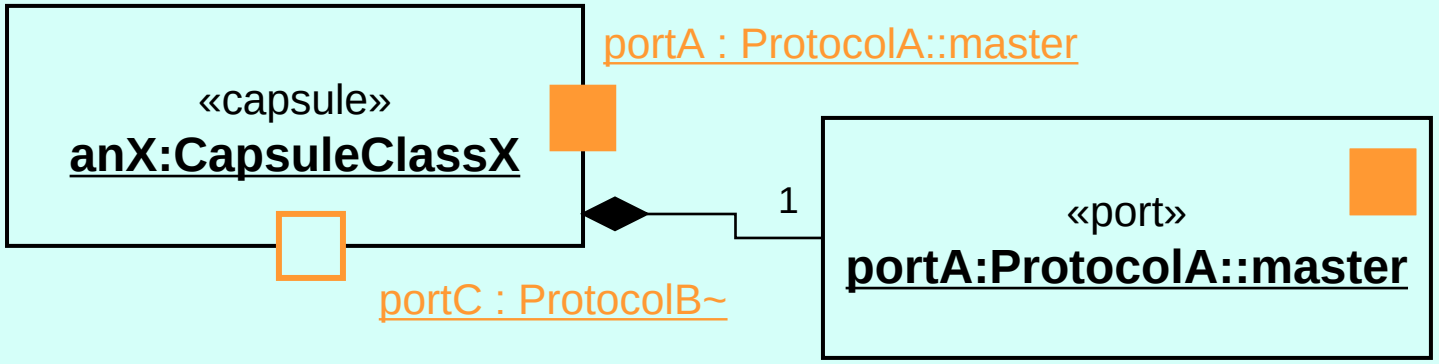
  (Neither Rose 2000 … ☺)

Proposed Exercise

- Model the problem of calling the taxi company to get a taxi to the airport in time to get the plane
- Start from the taxi class diagram of last week

ROOM: Real Time Object Oriented Modeling

From a slide of B. Selic of Object Time



- Shorthand notation for capsule instances
  - iconified form

For more information: http://www.objecttime.com/

The Object Constraint Language (OCL)

What is OCL

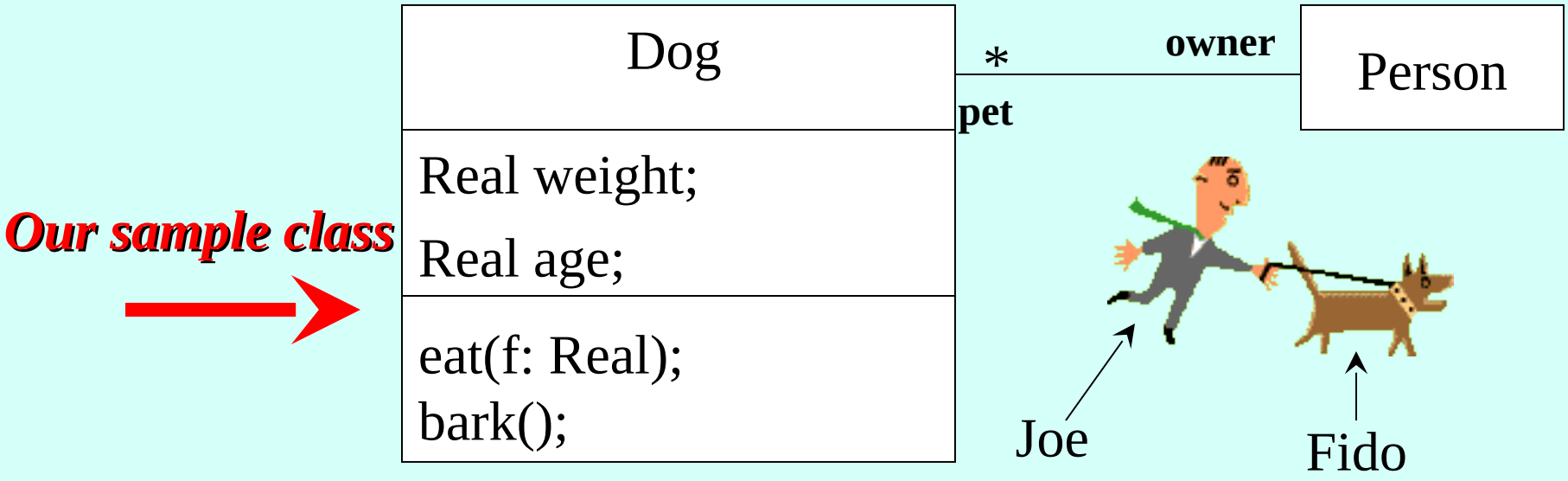
OCL is the expression language for UML

- ☑ It is a pure language, no side effects!
- ☑ It is a modeling language, it cannot be executed
- ☑ It is a formal language, it has a well defined semantics

For more information: http://www.software.ibm.com/ad/standards/ocl.html

Goal of OCL

- To define in a clean and unambiguous way the guard statements in UML



Major OCL features (1)

- Attributes have the same syntax as in Java (but “this” that becomes “self”)
  - e.g., self.weight < 100
- Expressions are strongly typed and type conformance is checked
  - e.g., self.weight < “abc”

Major OCL features (2)

- Pre and post conditions are assigned to operations

  Dog::bark()

  – pre: neighborhood is happy

  – post: neighborhood is angry

  Dog::eat(f: Real)

  – post: weight := weight@pre + f

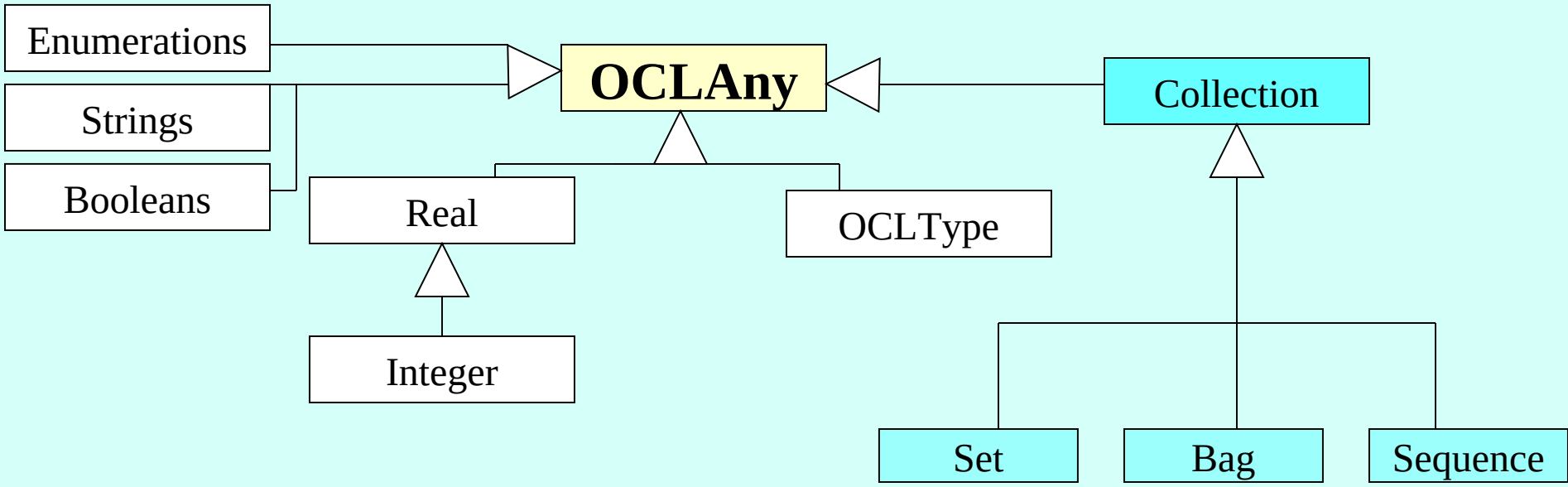
Major OCL features (3)

- Associations of cardinality 1 are treated as attributes
  - Fido.owner = Joe
- Association of cardinality not 1 are treated as sets
  - Joe->pet is a set of all the pets of Joe

and have well defined operations

“.” and “->” can be used polymorphically

Major OCL features (4)



Methods of collections in OCL

- size
- includes
- count
- includesAll
- isEmpty
- notEmpy
- sum
- exists
- forAll
- iterate
  - select
  - reject
  - collect

Introduction to Refactoring

Content

- The concept of refactoring UML diagrams
- A simple example
- Techniques used for refactoring

Source: Don Roberts Tutorial on Refactoring

Key idea

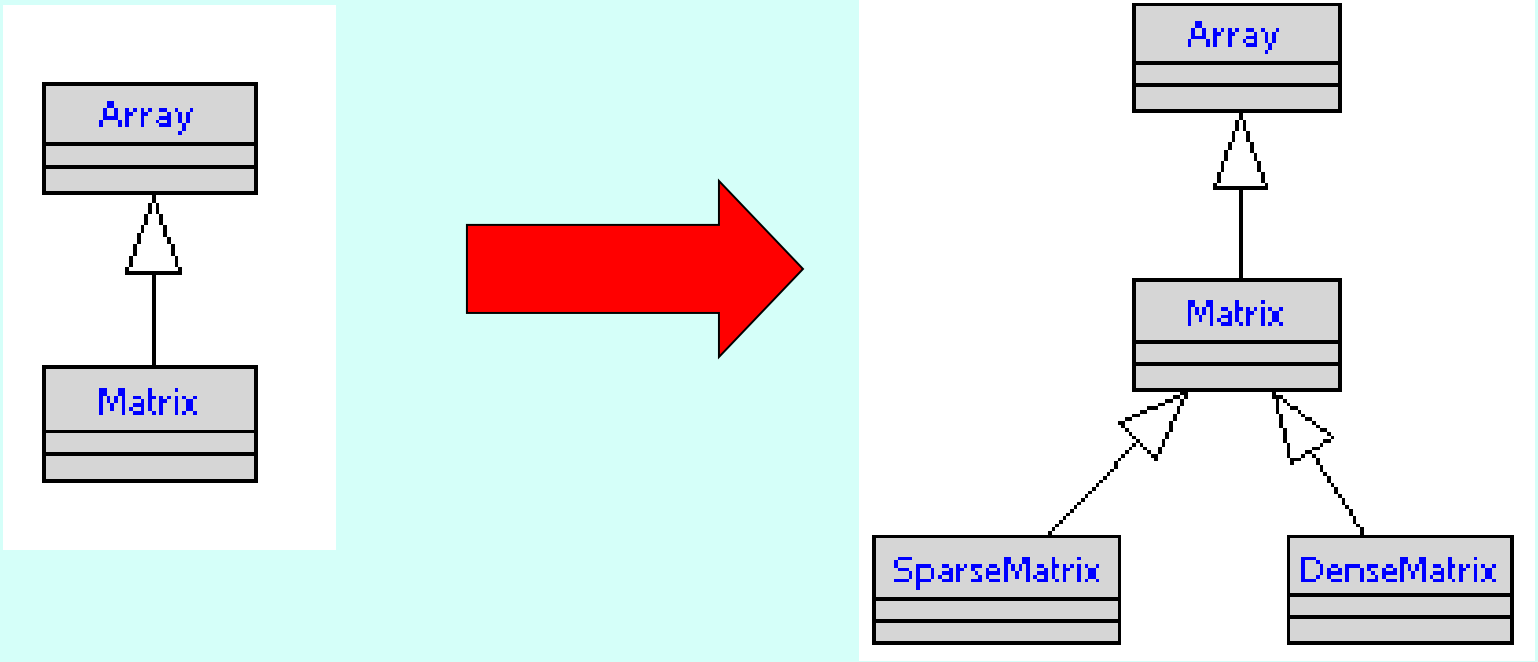
- Software systems are evolving entities that require constant upgrading and housekeeping
- We need to define ways to make upgrading and housekeeping “simple” aka “not horrendously difficult”
- Incremental development with short life-cycles and constant ***refactoring***

Refactoring UML diagrams

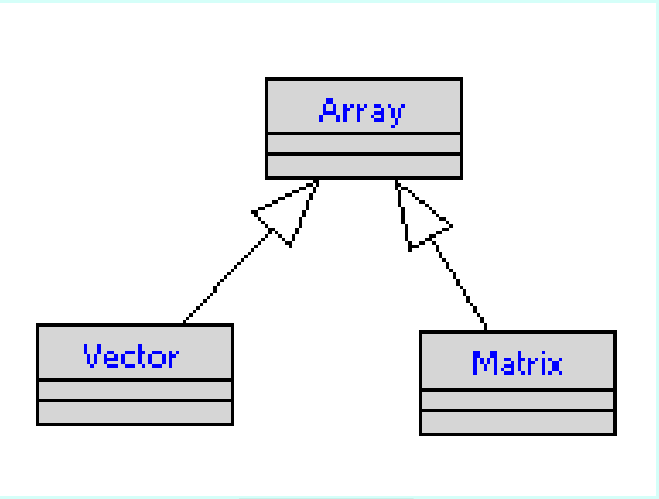
- Revising our system, restructuring its design and the design of its components striving for ***simplicity*** and higher adherence to the ***structure*** of the domain

Maintaining its behavior

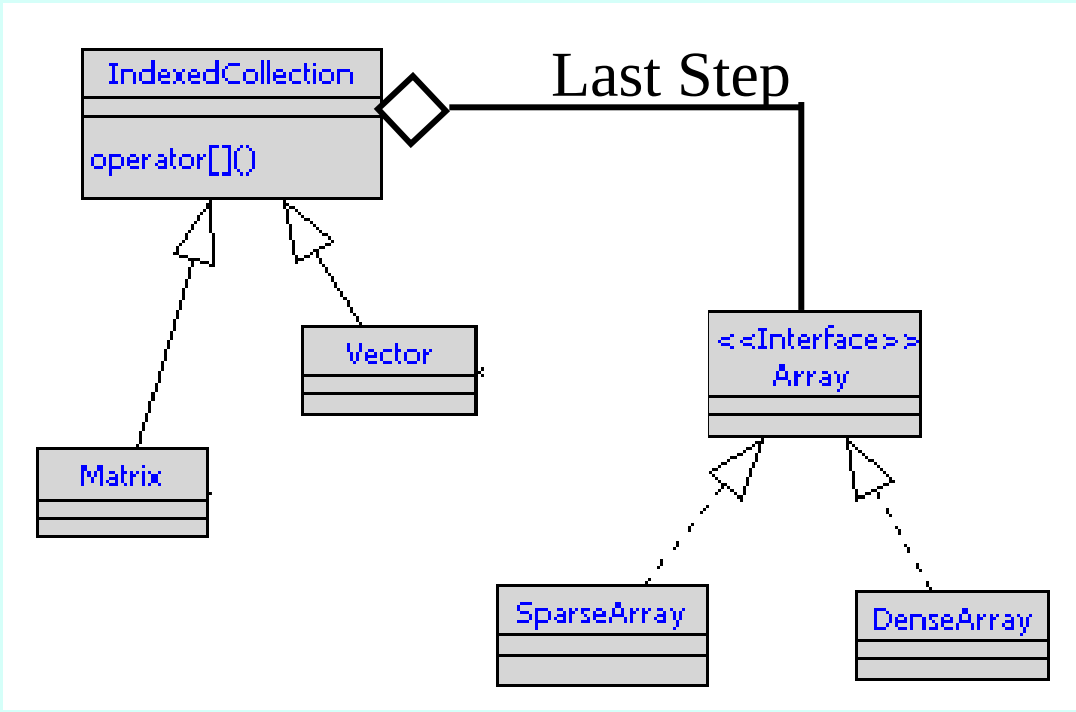
Example



Example (cont)



Refactoring



Techniques used for refactoring

	Class	(Static) (member) data	(Static) (member) function	Temp. data
Add				
Delete				
Rename				
Push Up/Down				
Break Down / Condense				
Re-qualify				

Design patterns and refactoring

- Design patterns can be instrumental for refactoring:
  1. ***They hint the inner structure of the code***
  2. ***They guide the development of new structures of objects and classes***
  3. ***They help separating the implementation strategies from the implementation goal***

Guide for refactoring with Design Patterns

- Consistent creation across classes: **Abstract Factory**
- Creation depending on environment needs: **Factory Method**
- Creation based on a prototypical entity: **Prototype**
- Variable number of features: **Decorator**
- Tree structure: **Composite**
- Double dispatching: **Visitor**
- Flexible implementation of an algorithm: **Strategy**
- Flexible implementation of a class: **Bridge**

Criteria for refactoring

- ☑ Refactor when the program shows deficiencies and not for “conceptual improvement”
- ☑ Write the code exactly once (see the <<include>> relation in Use Cases)
- ☑ Eliminate large entities (methods, classes, packages/namespaces)

Inhibitor$$

- "The system works" and
  - *No time*
  - *No money*
- Dealing with someone else's design abstractions

Continuation of the Proposed Exercise (1)

Develop the remaining OO Design Diagrams for the system supporting the reservation and scheduling for taxi drivers that was discussed in the first day of the course. Then think at how you could refactor the system developed so far.

Continuation of the Proposed Exercise (2)

Develop the remaining OO Design Diagrams for the system that manages the search, the selection, and the purchase of books at Chapters (http://www.chapters.ca) that was discussed in the first day of the course. Then think at how you could refactor the system developed so far.

Continuation of the Proposed Exercise (3) *Stocks Trading Service*

Develop the remaining OO Diagrams for…
Then refactor …

- By connecting to the service, a user can connect to different banks to acquire stock prices. The system also allows the user to perform some trend and prediction analysis of prices. If users are interested in ordering some stocks, they can choose to order them immediately or with a delay. They can also either bid at a single price or within a range of prices.
- The system should handle the situation where the connection to a bank is down, there is a conflict of bids, or if a particular stock is no longer available.

Continuation of the Proposed Exercise (4) N*etwork Printing Service*

Develop the remaining OO Diagrams for…
Then refactor …

- There is a super high-resolution color laser printer available on the network for users to print documents to. The service allows users to preview the output of their document on their screens. In addition, the user can also view the status of the printer to see whether there are other documents waiting to be printed and whether there are any problems with the printer (such as paper jams, out of paper, low on toner, etc…). In addition, users can monitor their own print jobs and delay or delete jobs as they see fit.
- To use this service, a user needs to have the proper authorization and print quota to print. A system administrator manages users and their print quotas.

Continuation of the Proposed Exercise (5) *Component Brokerage System*

Develop the remaining OO Diagrams for…
Then refactor …

- This system essentially acts as a broker for software components. When developers have completed development of their software, they can deploy them as reusable software components for others to use. By connecting to the system over the Internet, these developers can submit their components to the system. An administrator then reviews the component for its functionality and ways of connecting to other components, categorizes it, and publishes it in a publicly-viewable area.
- Customers (such as other developers) can then connect to the public system and browse/search the components. When they have found something useful, they can download it for use on their own machine.
- Later, the providers of the components can connect to the system and view the download statistics of their components. They can also add/remove components from the system.

Continuation of the Proposed Exercise (6) *Bug Tracking System*

Develop the remaining OO Diagrams for…
Then refactor …

- Developers use this system to track bugs in an on-going software project. Developers who find bugs can submit a report. Other developers can then assign the bug to a particular developer (especially the developer responsible for the software module) to fix it. In addition, users can browse/search all the bugs in the system so far.
- An administrator manages users to restrict access to the bug tracking system. In addition, the administrator should also be able to generate reports on the state of the project in the form of a set of web pages updated daily at 2am.

Proposed comprehensive example

- Define a system to support all the features (calling, billing, roaming, + the extra gadgets of your choice) of a new form of cellular service
  - While at home or within 10 meters of your home, same charges as the local phone
  - While in the city limits, only charge for airtime
  - Within Canada, roaming access from the local provider that set himself the charges + airtime
  - Outside Canada, no service in place