UNIVERSITÄT
BERN

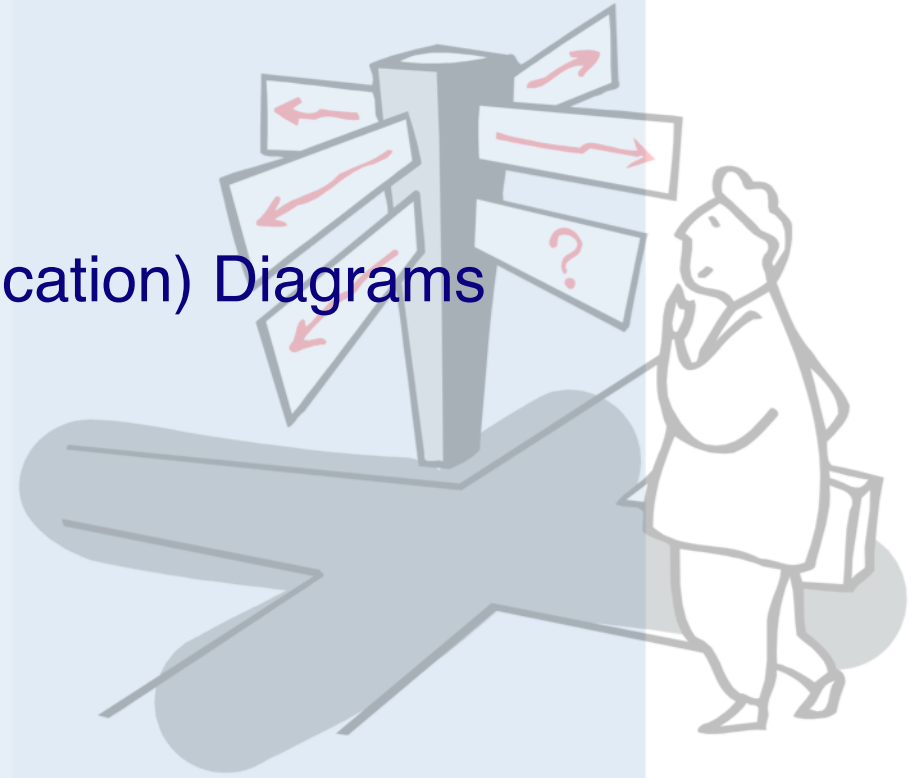
ESE
Einführung in Software Engineering

7. Modeling Behaviour

Prof. O. Nierstrasz

Roadmap

- Use Case Diagrams
- Sequence Diagrams
- Collaboration (Communication) Diagrams
- Statechart Diagrams
  - Nested statecharts
  - Concurrent substates
- Using UML

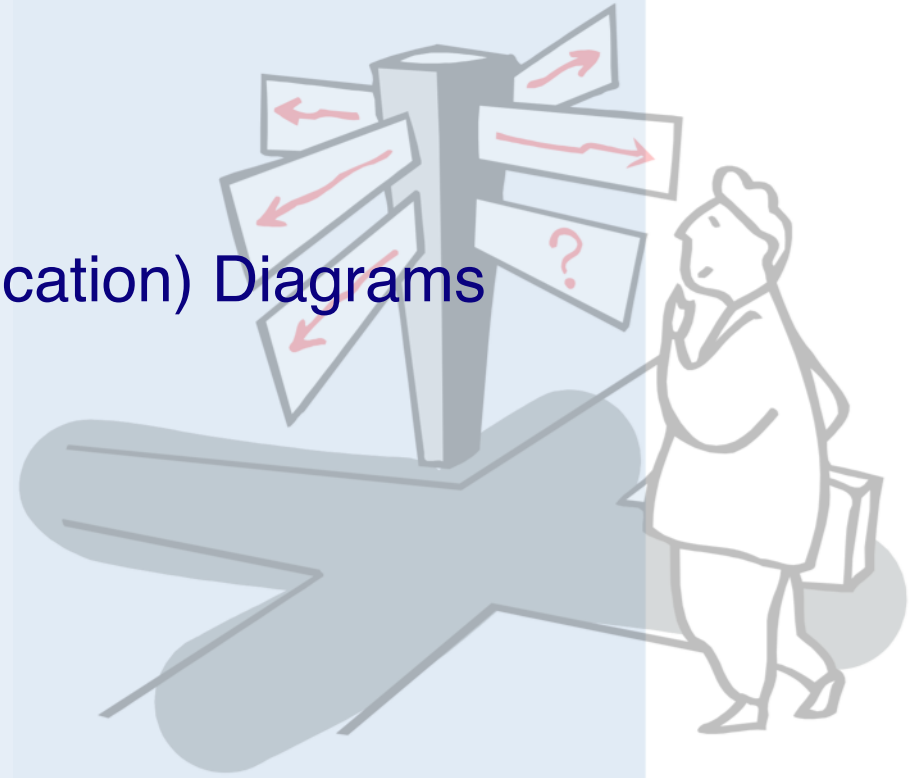


Source

- *The Unified Modeling Language Reference Manual*, James Rumbaugh, Ivar Jacobson and Grady Booch, Addison Wesley, 1999.

Roadmap

- **Use Case Diagrams**
- Sequence Diagrams
- Collaboration (Communication) Diagrams
- Statechart Diagrams
  - Nested statecharts
  - Concurrent substates
- Using UML



Use Case Diagrams

A use case is a *generic description of an entire transaction* involving several actors.

A use case diagram presents a *set of use cases* (ellipses) and the external actors that interact with the system.

Dependencies and *associations* between use cases may be indicated.

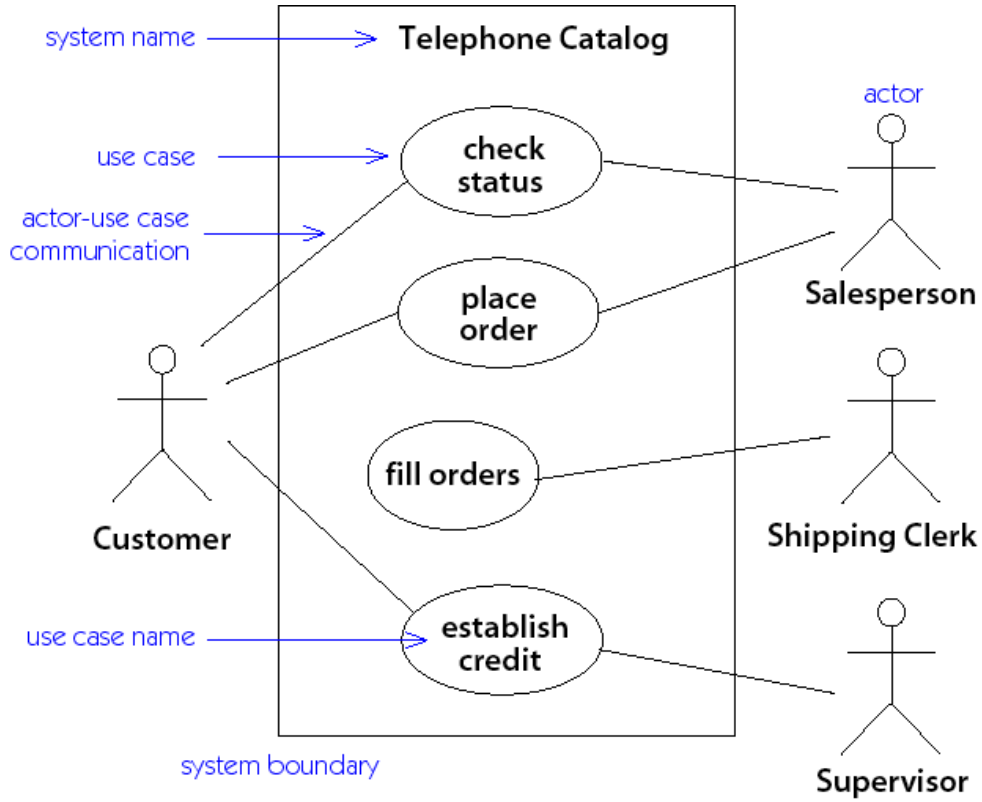


Figure 5-1. *Use case diagram*

Using Use Case Diagrams

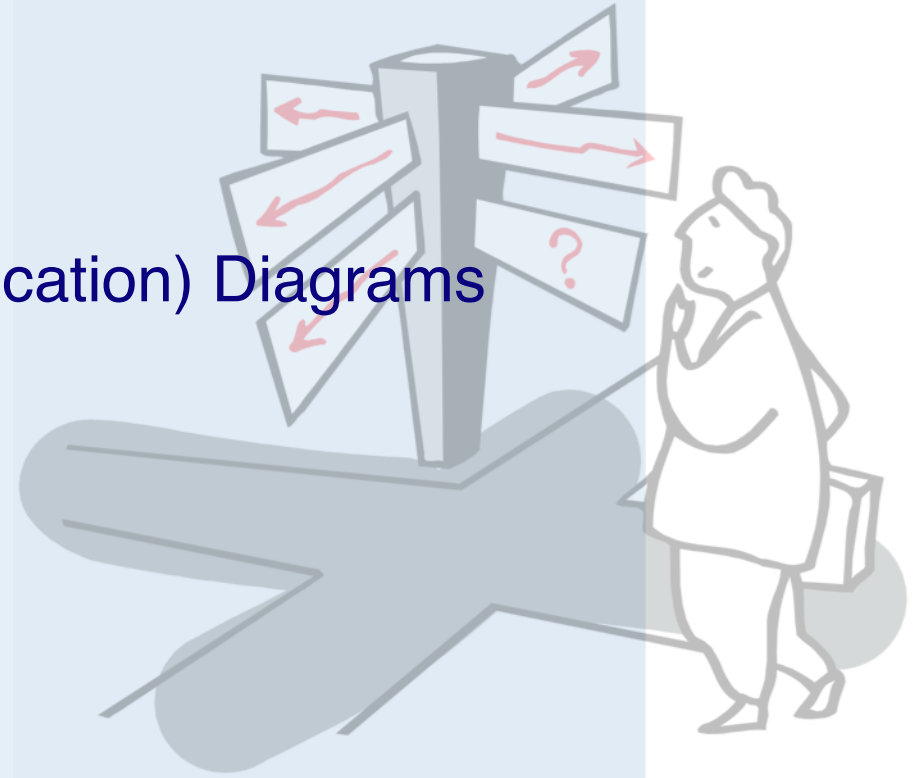
- "A use case is a *snapshot of one aspect* of your system. The sum of all use cases is *the external picture* of your system …"

  — *UML Distilled*

- "As use cases appear, assess their impact on the domain model."
  - Use cases can *drive domain modeling* by highlighting the important concepts.

Roadmap

- Use Case Diagrams
- **Sequence Diagrams**
- Collaboration (Communication) Diagrams
- Statechart Diagrams
  - Nested statecharts
  - Concurrent substates
- Using UML



Scenarios

A scenario is an *instance* of a use case showing a *typical example* of its execution.

Scenarios can be presented in UML using either *sequence diagrams* or *collaboration diagrams*.

Note that a scenario only describes an example of a use case, so conditionality cannot be expressed!

Sequence Diagrams

A sequence diagram depicts a scenario by showing the interactions among a set of objects in *temporal order*.

Objects (not classes!) are shown as *vertical bars*. *Events* or message dispatches are shown as horizontal (or slanted) *arrows* from the sender to the receiver.

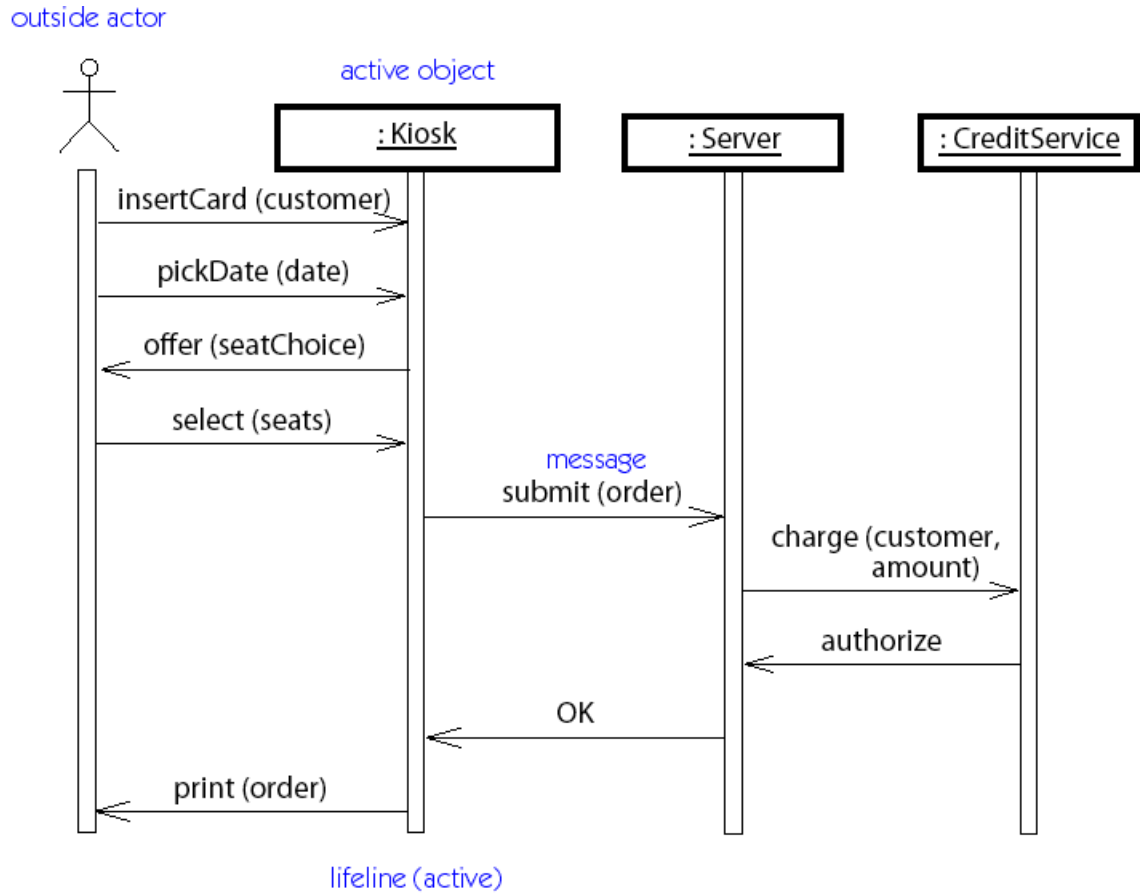
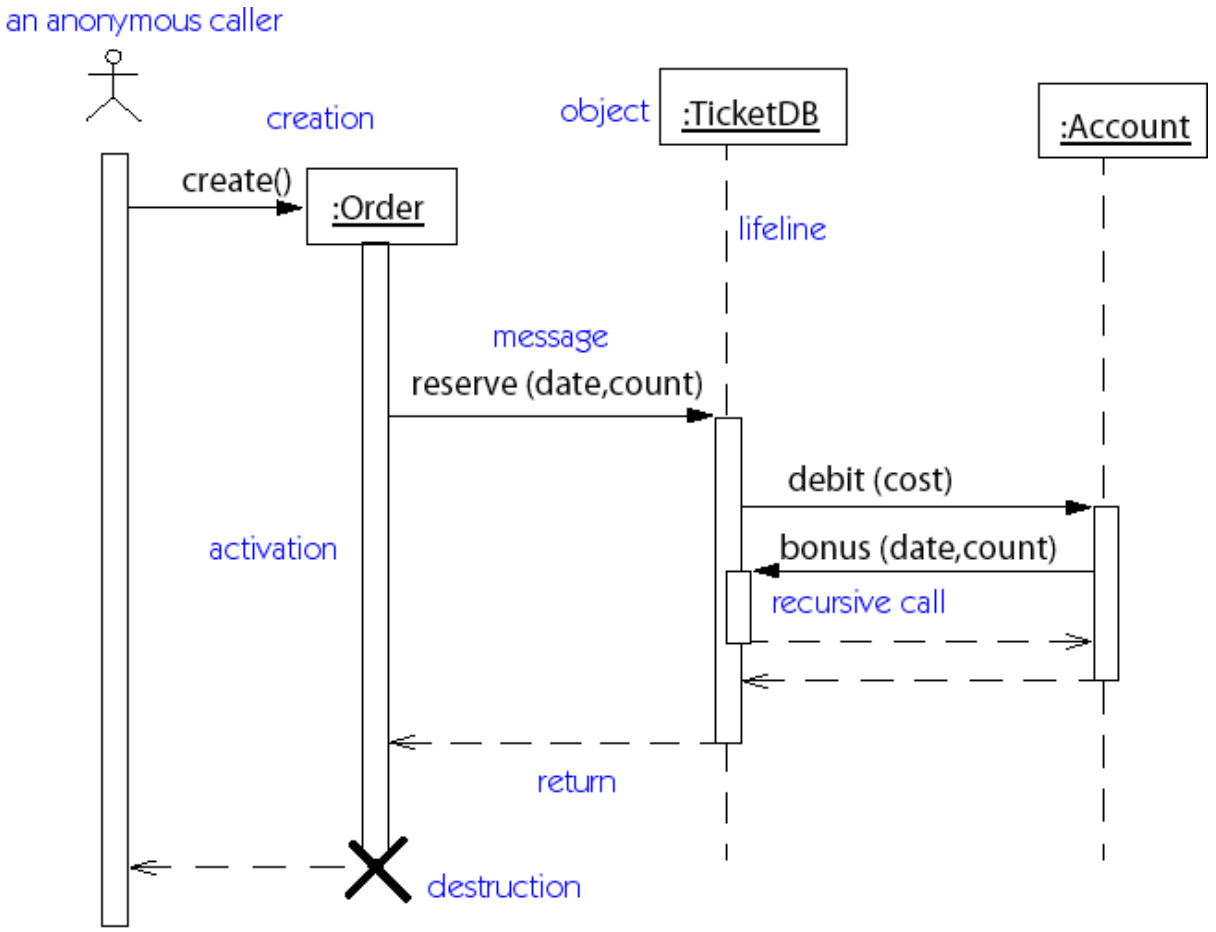


Figure 8-1. *Sequence diagram*

Activations



Avoid returns in sequence diagrams unless they add clarity.

Figure 8-2. *Sequence diagram with activations*

Asynchrony and Constraints

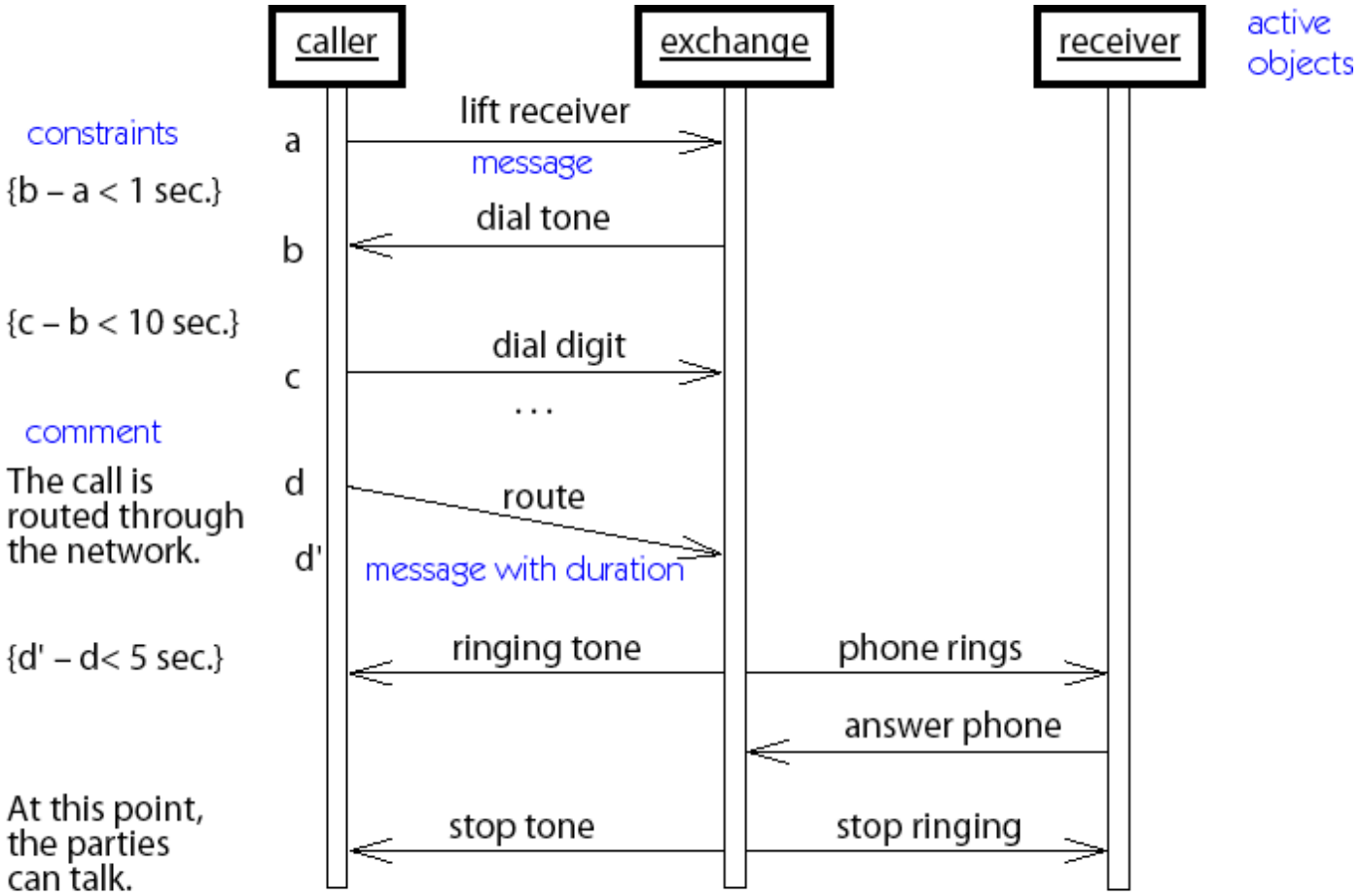
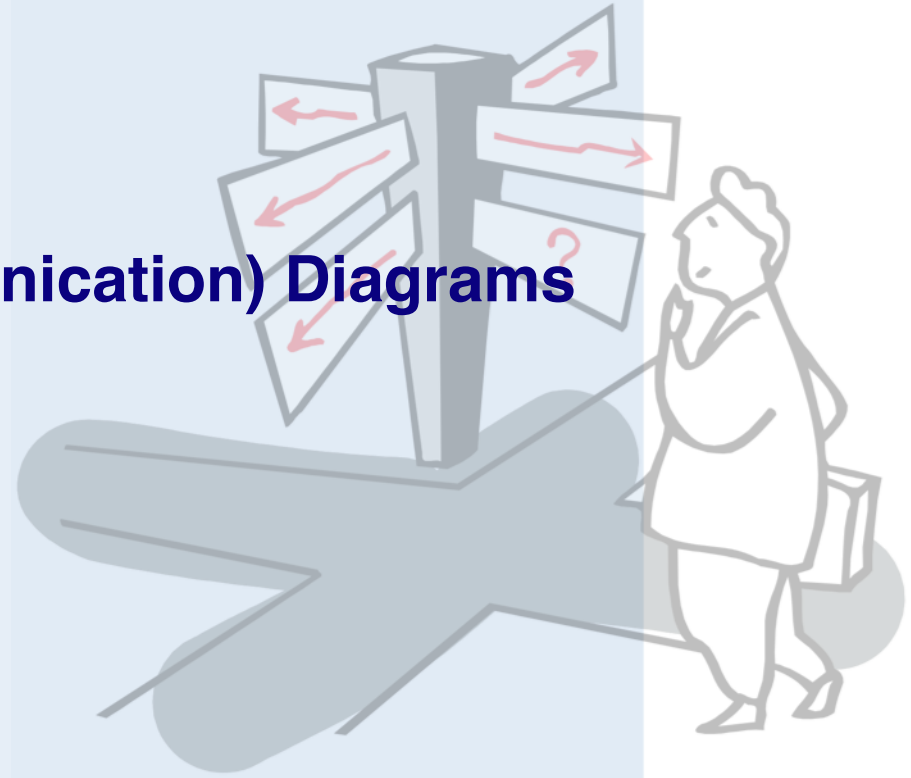


Figure 13-161. *Sequence diagram with asynchronous control*

Roadmap

- Use Case Diagrams
- Sequence Diagrams
- **Collaboration (Communication) Diagrams**
- Statechart Diagrams
  - Nested statecharts
  - Concurrent substates
- Using UML



Collaboration Diagrams

Collaboration diagrams (called *Communication* diagrams in UML 2.0) depict scenarios as *flows of messages* between objects:

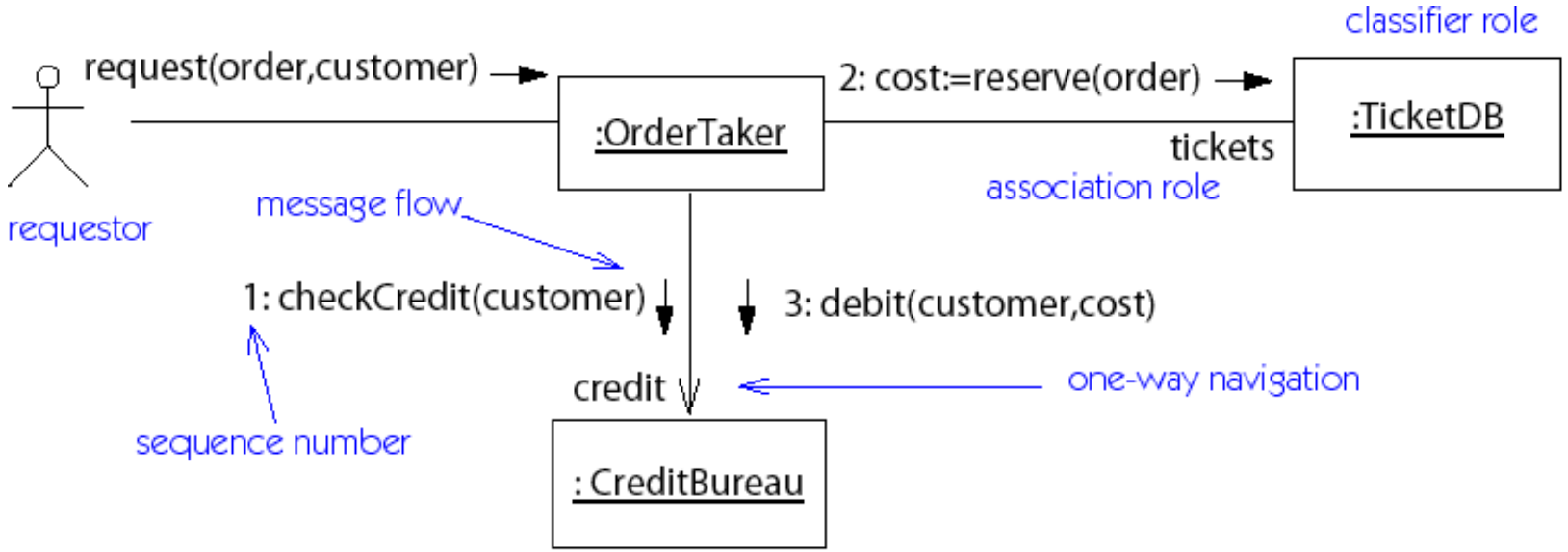


Figure 8-3. *Collaboration diagram*

Message Labels

Messages from one object to another are labelled with text strings showing the *direction* of message flow and information indicating the message *sequence*.

1. *Prior messages* from other threads (e.g. “[A1.3, B6.7.1]”)
   - *only needed with* concurrent *flow of control*
2. Dot-separated list of sequencing elements
   - sequencing *integer (e.g., “3.1.2” is invoked by “3.1” and follows “3.1.1”)*
   - *letter indicating* concurrent *threads (e.g., “1.2a” and “1.2b”)*
   - iteration *indicator (e.g., “1.1*[i=1..n]”)*
   - conditional *indicator (e.g., “2.3 [#items = 0]”)*
3. *Return value* binding (e.g., “status :=”)
4. Message name
   - *event or operation name*
5. Argument list

Nested Message Flows

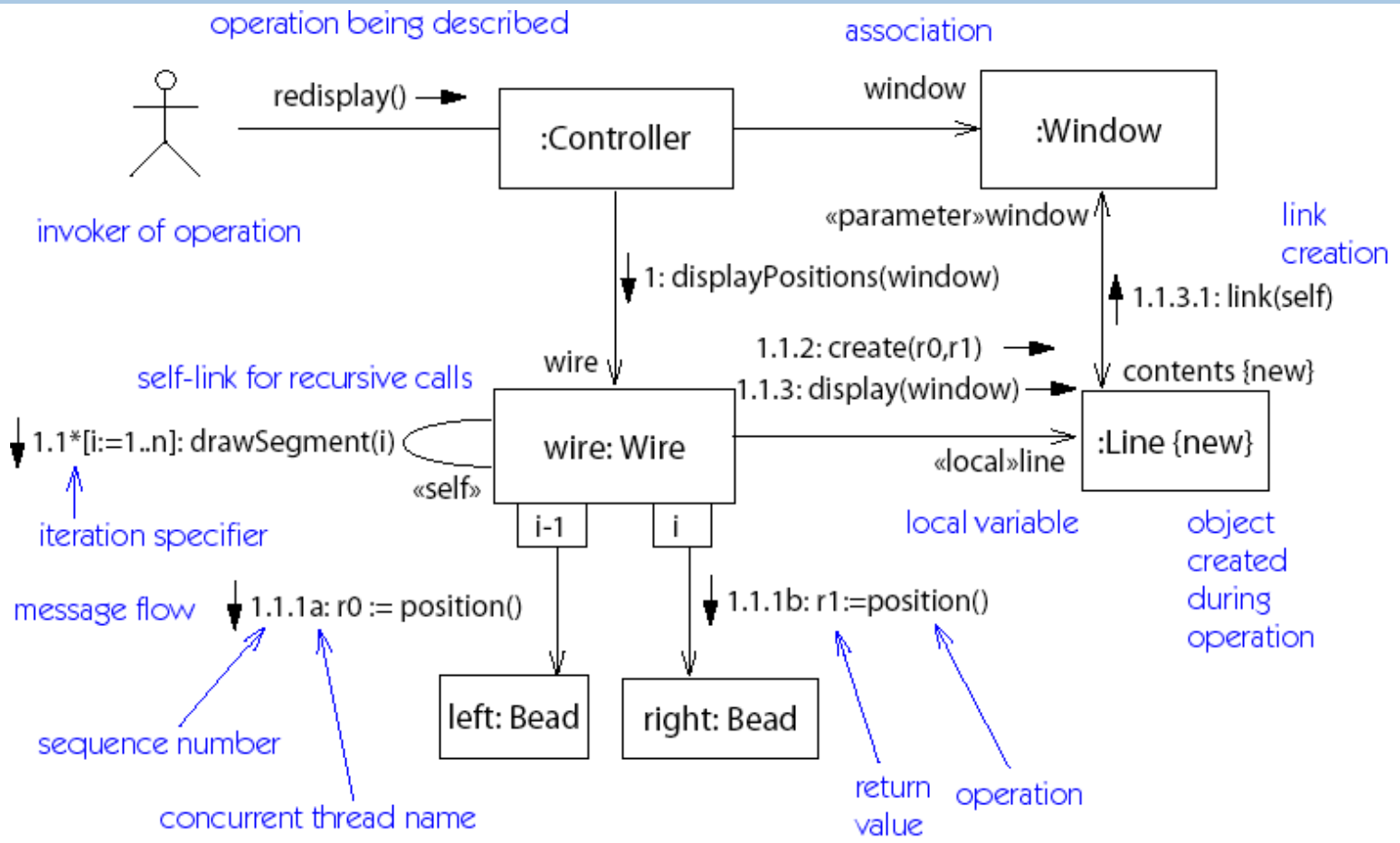
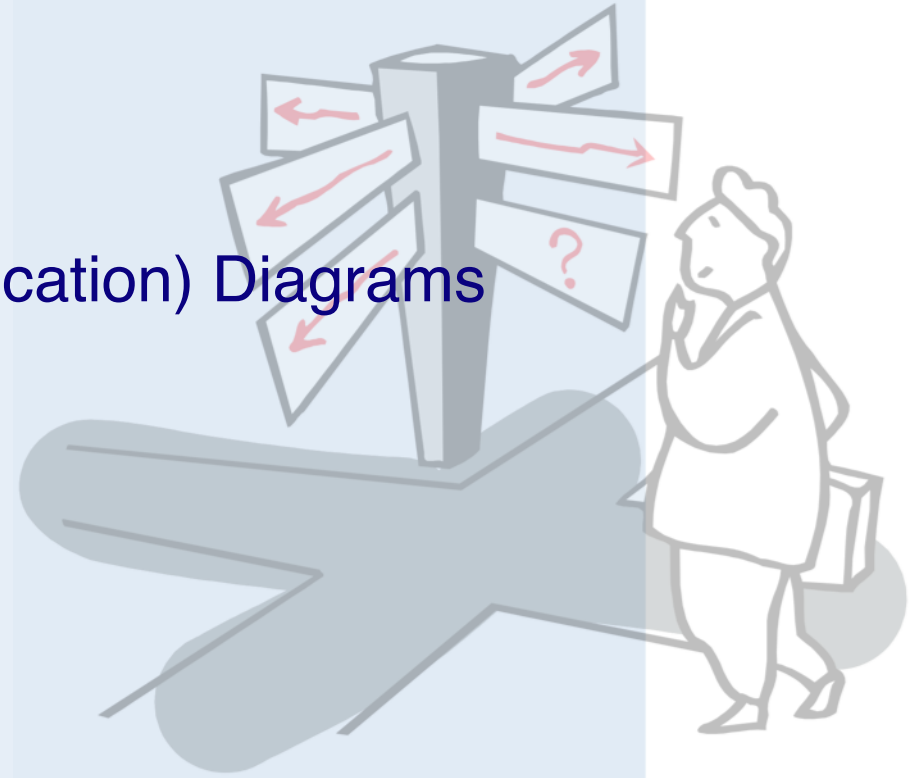


Figure 13-51. *Collaboration diagram with message flows*

Roadmap

- Use Case Diagrams
- Sequence Diagrams
- Collaboration (Communication) Diagrams
- **Statechart Diagrams**
  - Nested statecharts
  - Concurrent substates
- Using UML



Statechart Diagrams

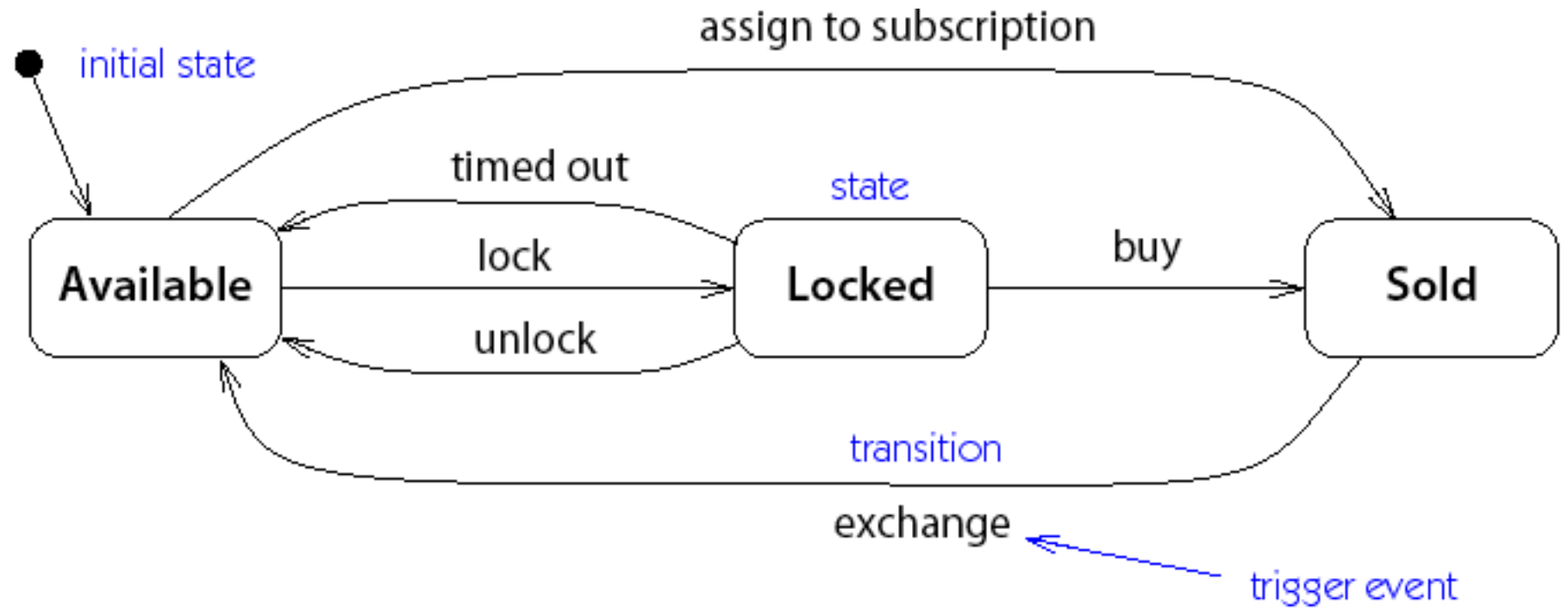


Figure 3-5. *Statechart diagram*

Statechart Diagram Notation

A Statechart Diagram describes the *temporal evolution* of an object of a given class in response to *interactions* with other objects inside or outside the system.

An event is a one-way (asynchronous) communication from one object to another:

- *atomic* (non-interruptible)
- includes events from *hardware* and real-world objects e.g., message receipt, input event, elapsed time, ...
- notation: ***eventName(parameter: type, ...)***
- may cause object to make a *transition* between states

Statechart Diagram Notation ...

A state is a period of time during which an object is *waiting* for an event to occur:

- depicted as *rounded box* with (up to) three sections:
  - name — *optional*
  - state *variables* — *name: type = value (valid only for that state)*
  - triggered operations — *internal transitions and ongoing operations*
- may be *nested*

State Box with Regions

The *entry event* occurs whenever a transition is made into this state, and the *exit operation* is triggered when a transition is made out of this state.

The *help* and *character* events cause internal transitions with no change of state, so the entry and exit operations are not performed.

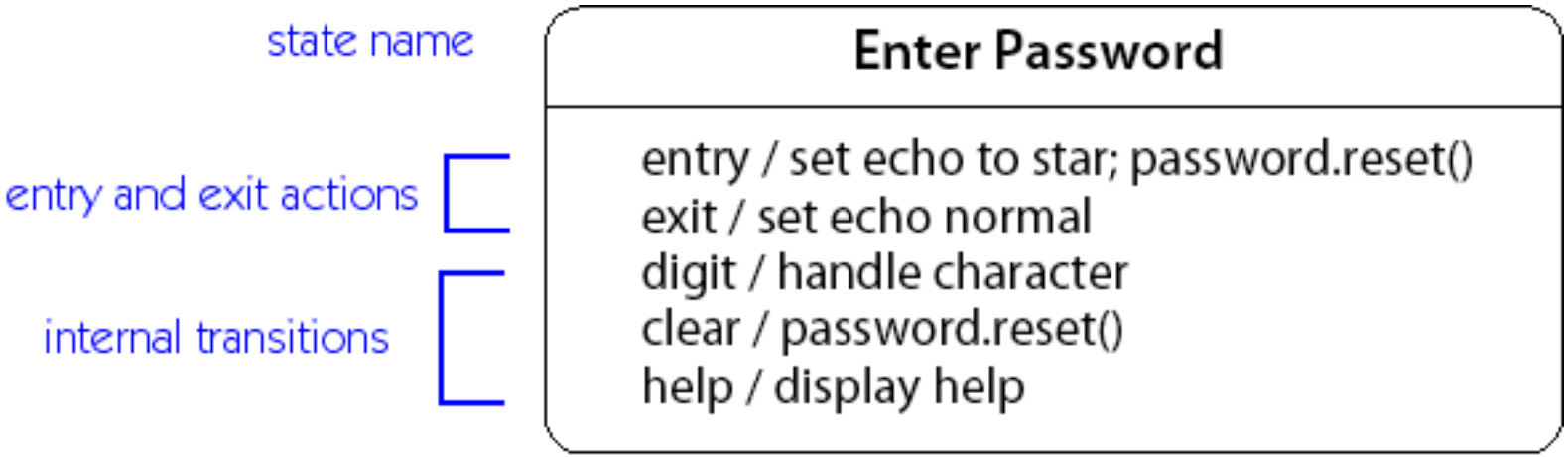


Figure 6-4. *Internal transitions, and entry and exit actions*

Transitions

A transition is an *response to an external event* received by an object in a *given state*

- — May *invoke* an operation, and cause the object to change state
- — May *send* an event to an external object
- — Transition syntax (each part is optional):

  event(arguments) [condition]
  / ^target.sendEvent operation(arguments)

- — *External transitions* label arcs between states
- — *Internal transitions* are part of the triggered operations of a state

Operations and Activities

An operation is an *atomic action* invoked by a transition

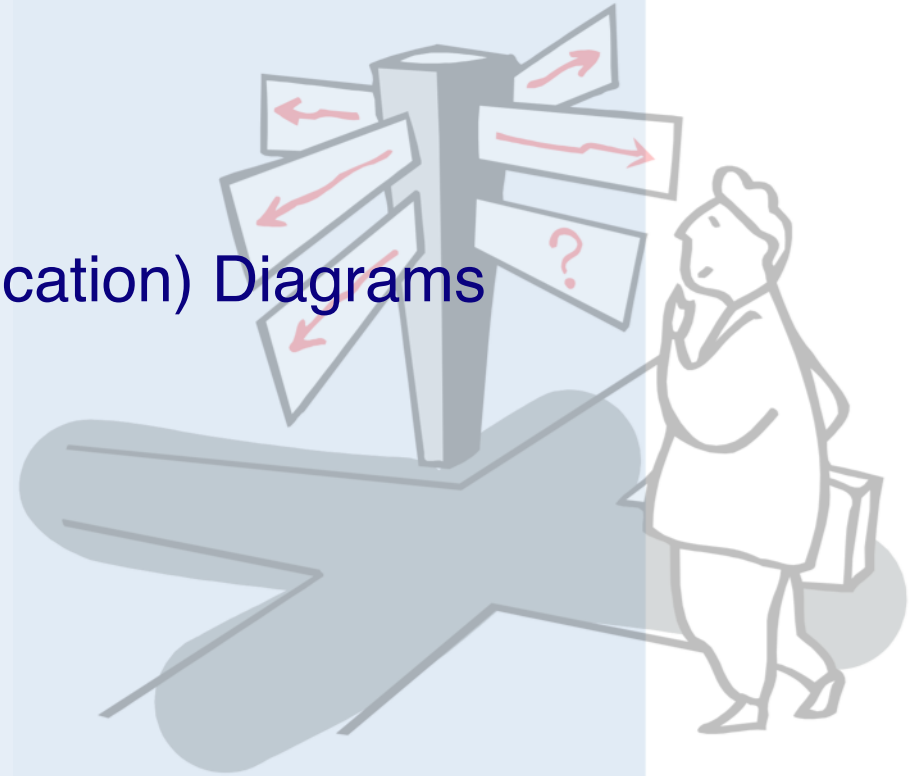
— *Entry and exit operations* can be associated with states

An activity is an *ongoing operation* that takes place while object is in a given state

— Modelled as “internal transitions” labelled with the pseudo-event ***do***

Roadmap

- Use Case Diagrams
- Sequence Diagrams
- Collaboration (Communication) Diagrams
- Statechart Diagrams
  - **Nested statecharts**
  - Concurrent substates
- Using UML



Nested Statecharts

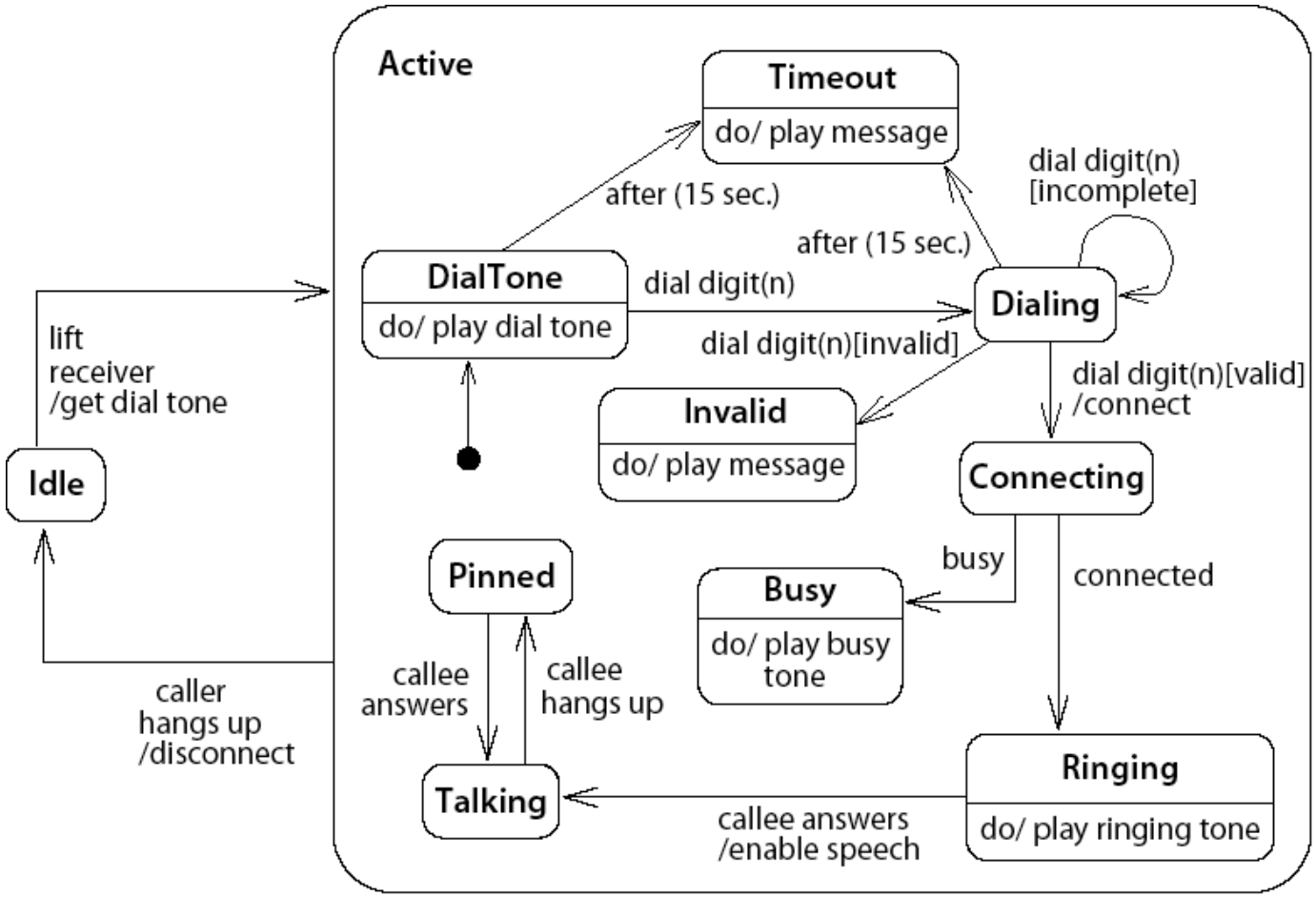


Figure 13-169. *State diagram*

Composite States

Composite states may depicted either as high-level or low-level views.

"Stubbed transitions" indicate the presence of internal states:

Initial and terminal substates are shown as black spots and "bulls-eyes"

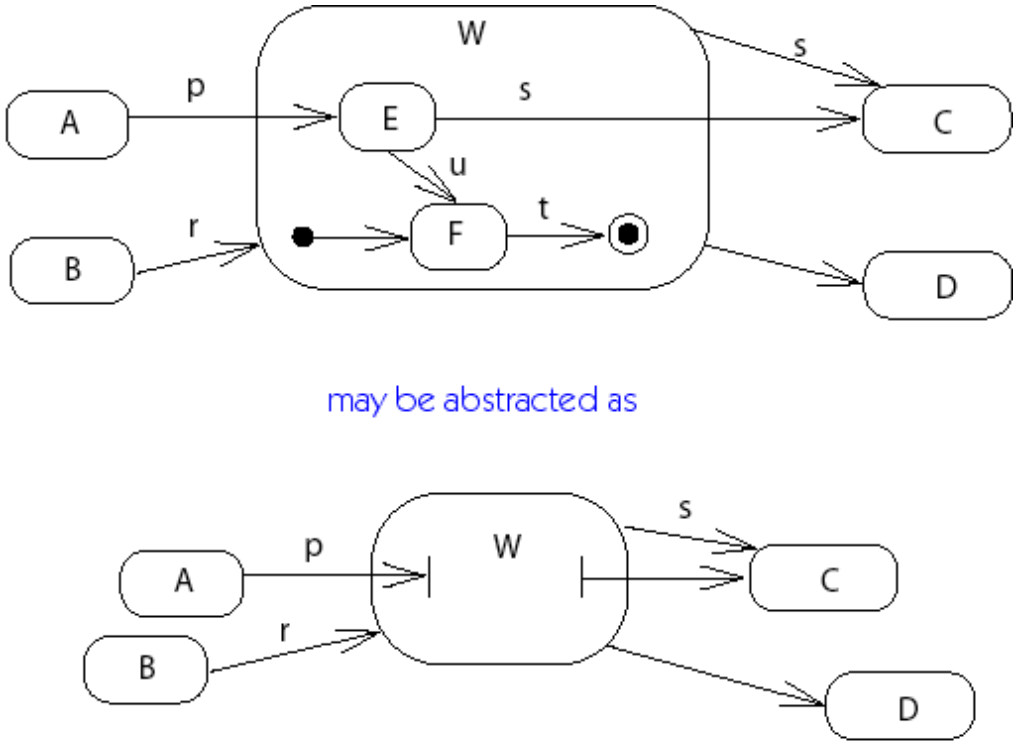


Figure 13-172. *Stubbed transition*

Sending Events between Objects

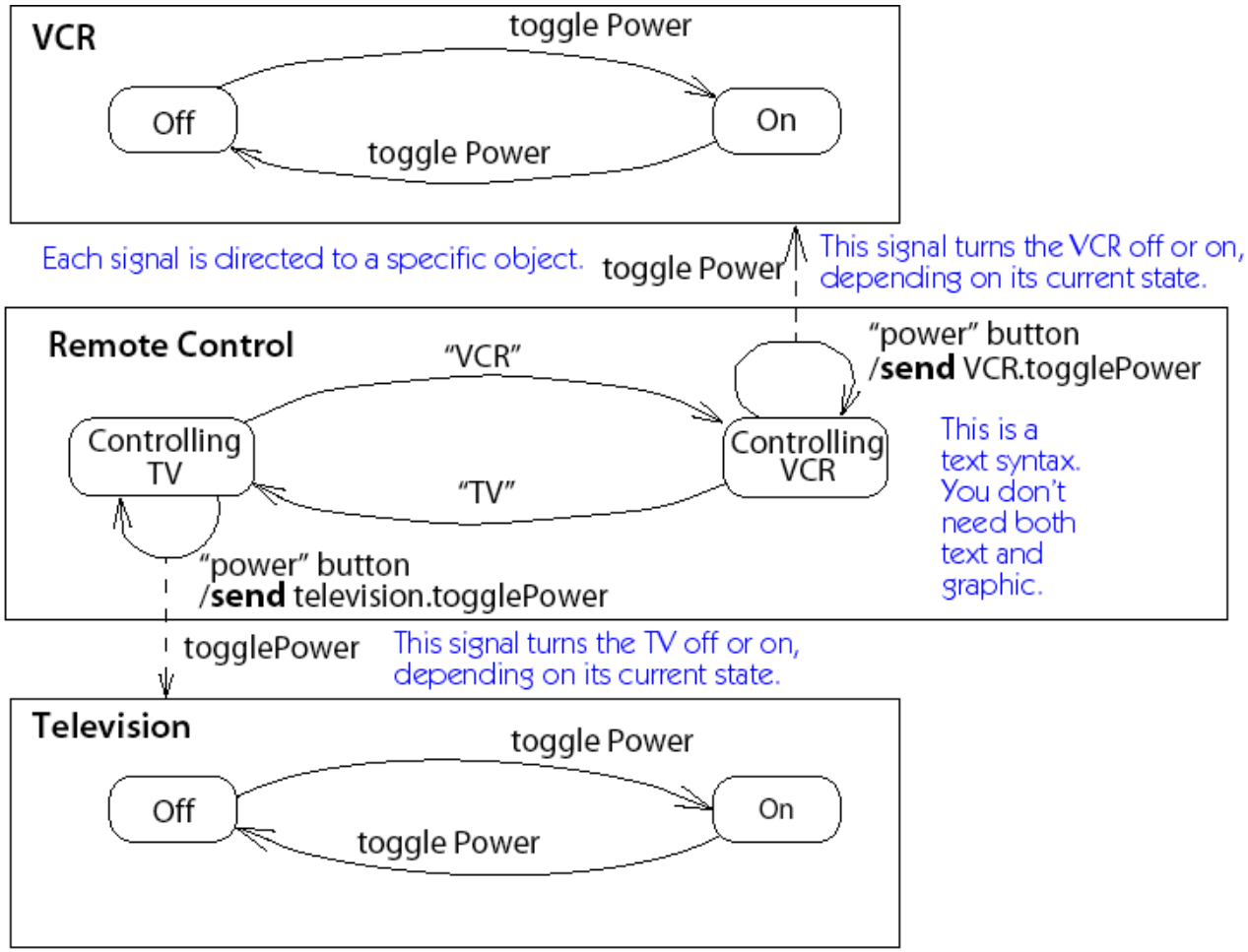
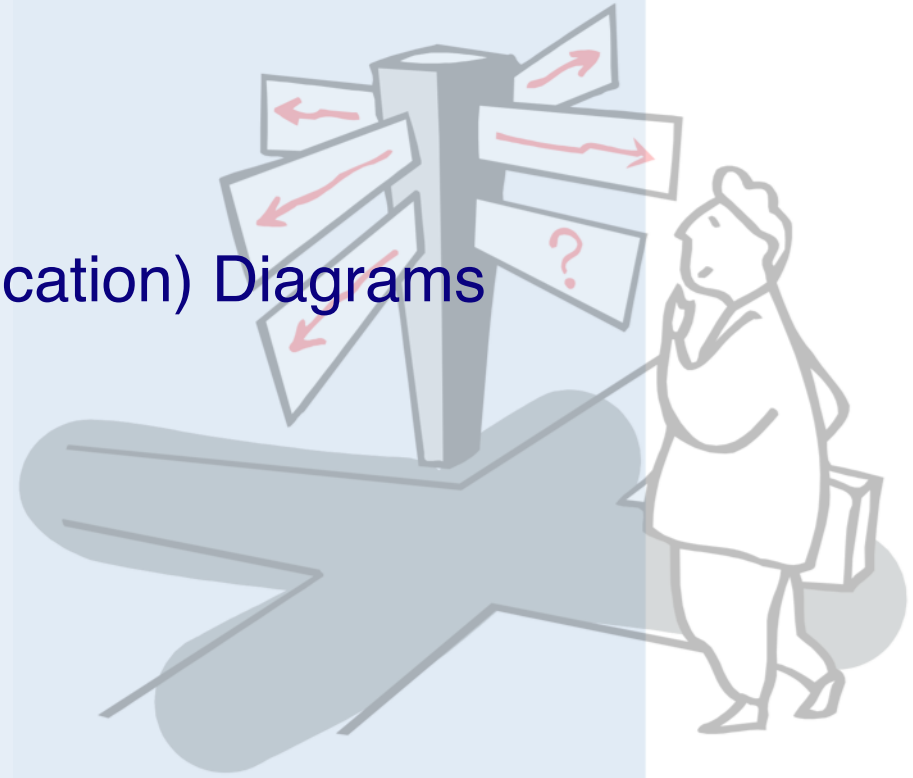


Figure 13-160. *Sending signals between objects*

Roadmap

- Use Case Diagrams
- Sequence Diagrams
- Collaboration (Communication) Diagrams
- Statechart Diagrams
  - Nested statecharts
  - **Concurrent substates**
- Using UML



Concurrent Substates

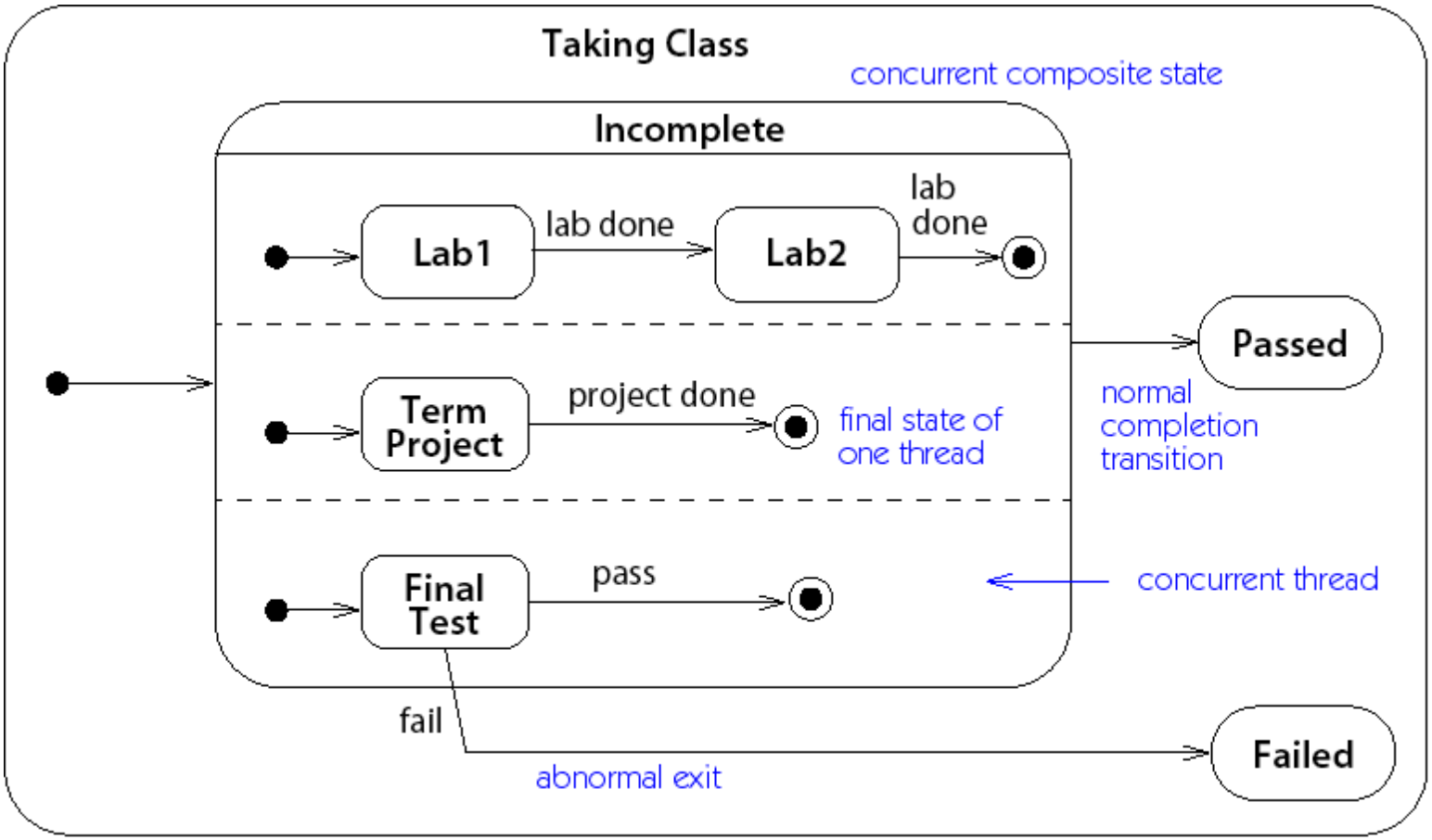


Figure 6-6. *State machine with concurrent composite state*

Branching and Merging

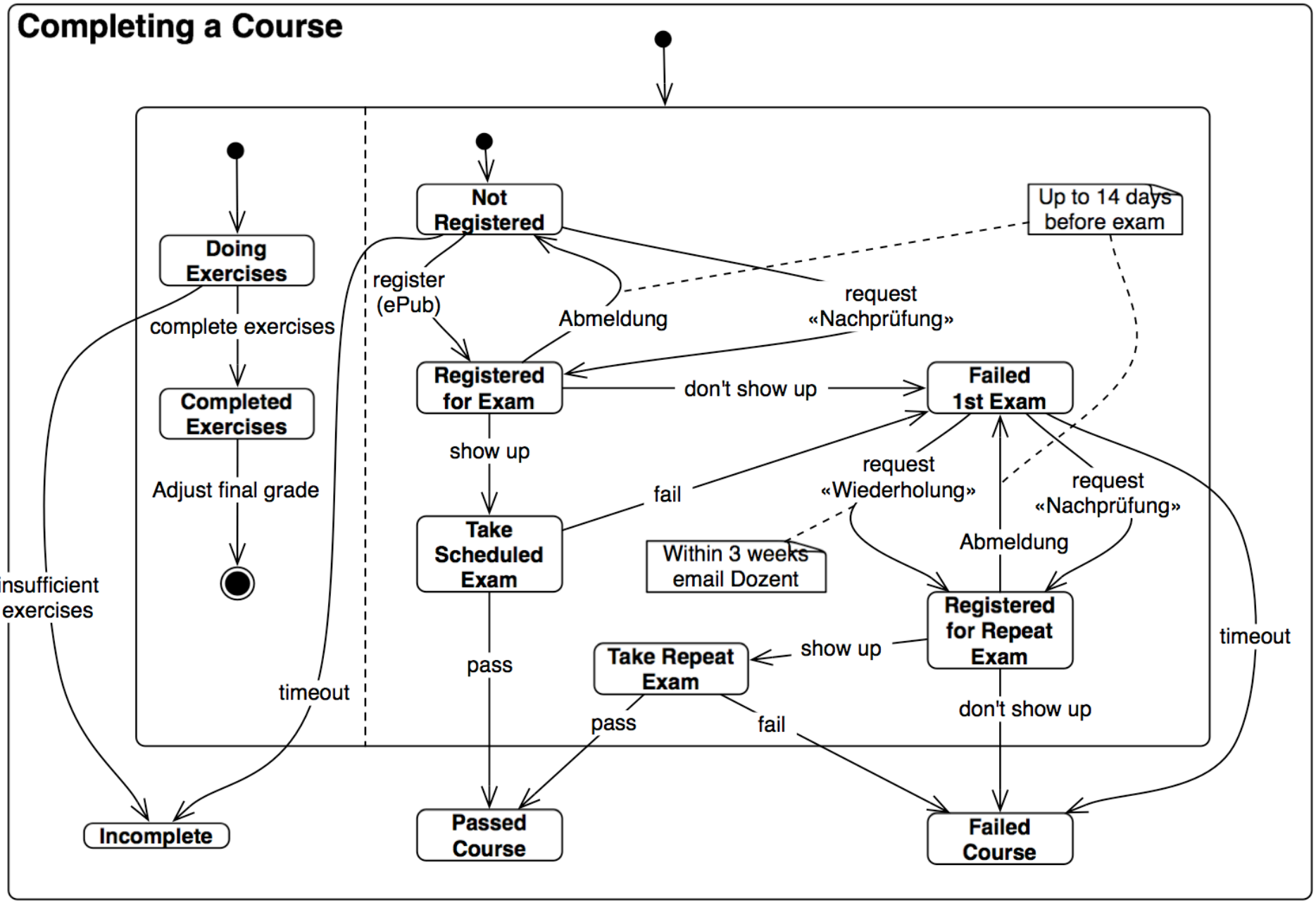
Entering concurrent states:

Entering a state with concurrent substates means that *each of the substates is entered concurrently* (one logical thread per substate).

Leaving concurrent states:

A *labelled transition* out of any of the substates *terminates all of the substates.*

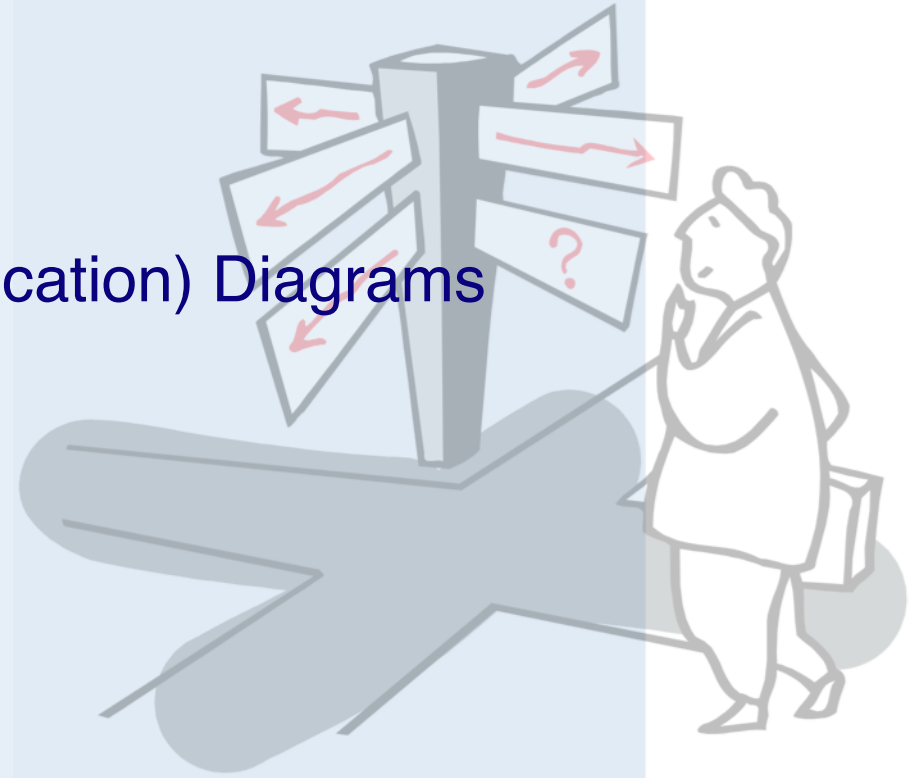
An *unlabelled transition* out of the overall state *waits* for all substates to terminate.



Is it correct?

Roadmap

- Use Case Diagrams
- Sequence Diagrams
- Collaboration (Communication) Diagrams
- Statechart Diagrams
  - Nested statecharts
  - Concurrent substates
- **Using UML**



Perspectives

Three perspectives in drawing UML diagrams:

1. ***Conceptual***
   - — Represent domain concepts
     - – *Ignore software issues*
2. ***Specification***
   - — Focus on visible interfaces and behaviour
     - – *Ignore internal implementation*
3. ***Implementation***
   - — Document implementation choices
     - – *Most common, but least useful perspective(!)*

— *UML Distilled*

Using the Notations

The diagrams introduced here complement class and object diagrams.

During Analysis:

— Use case, sequence and collaboration diagrams document *use cases and their scenarios* during requirements specification

During Design:

— Sequence and collaboration diagrams can be used to document *implementation scenarios* or refine use case scenarios

— State diagrams document *internal behaviour* of classes and must be *validated* against the specified use cases

What you should know!

- What is the purpose of a use case diagram?
- Why do scenarios depict objects but not classes?
- How can timing constraints be expressed in scenarios?
- How do you specify and interpret message labels in a scenario?
- How do you use nested state diagrams to model object behaviour?
- What is the difference between “external” and “internal” transitions?
- How can you model interaction between state diagrams for several classes?

Can you answer the following questions?

- Can a sequence diagram always be translated to an collaboration diagram?
- Or vice versa?
- Why are arrows depicted with the message labels rather than with links?
- When should you use concurrent substates?

License

> http://creativecommons.org/licenses/by-sa/3.0/