Introduction to Software Engineering

The Specification Phase

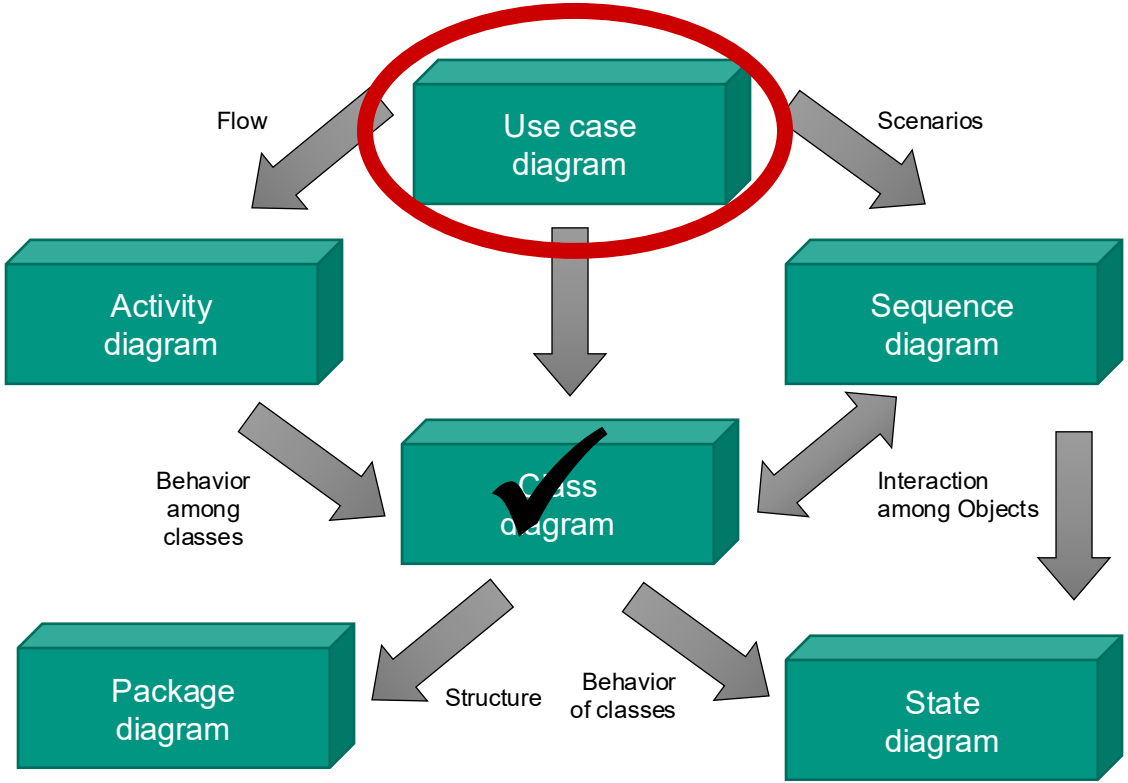
2.2 Additional UML diagrams

Prof. Walter F. Tichy

ქუთაისის
საერთაშორისო
უნივერსიტეტი

Contents

UML-diagrams



Use case diagram

- For requirements elicitation– what does the user want?
- Modelling typical interactions of users with the (future) system
- Derived from
  - Dialogue/interview with (future) user
  - Dialogue/interview with experts
  - Own experience or introspection

  } perhaps ≠ customer!
- Enables checks, whether the system will do what the user wants

Use case diagram

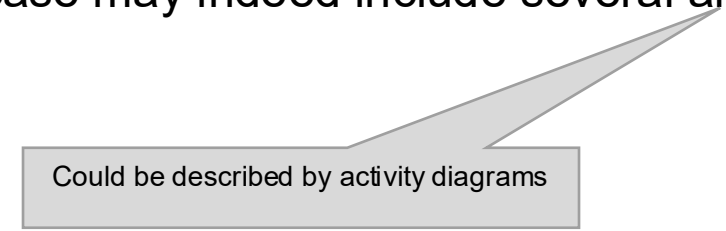
- For requirements elicitation– what does the user want?
- Modelling typical interactions with the (future) system
- Derived from
  - Dialogue/in
  - Dialogue/in
- Enables che
  will do what

Important: Use case diagrams are an aid for requirements elicitation and documentation. They ony show relationships between model elements.

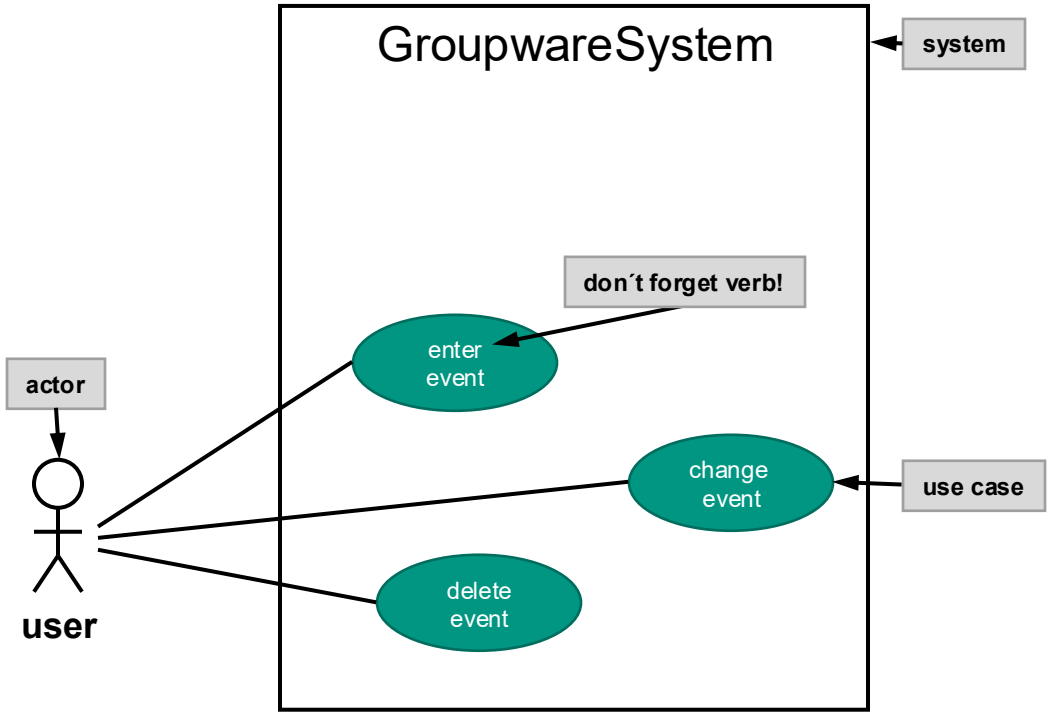
Use case diagrams do not show dynamic behavior or processes.

Definition use case

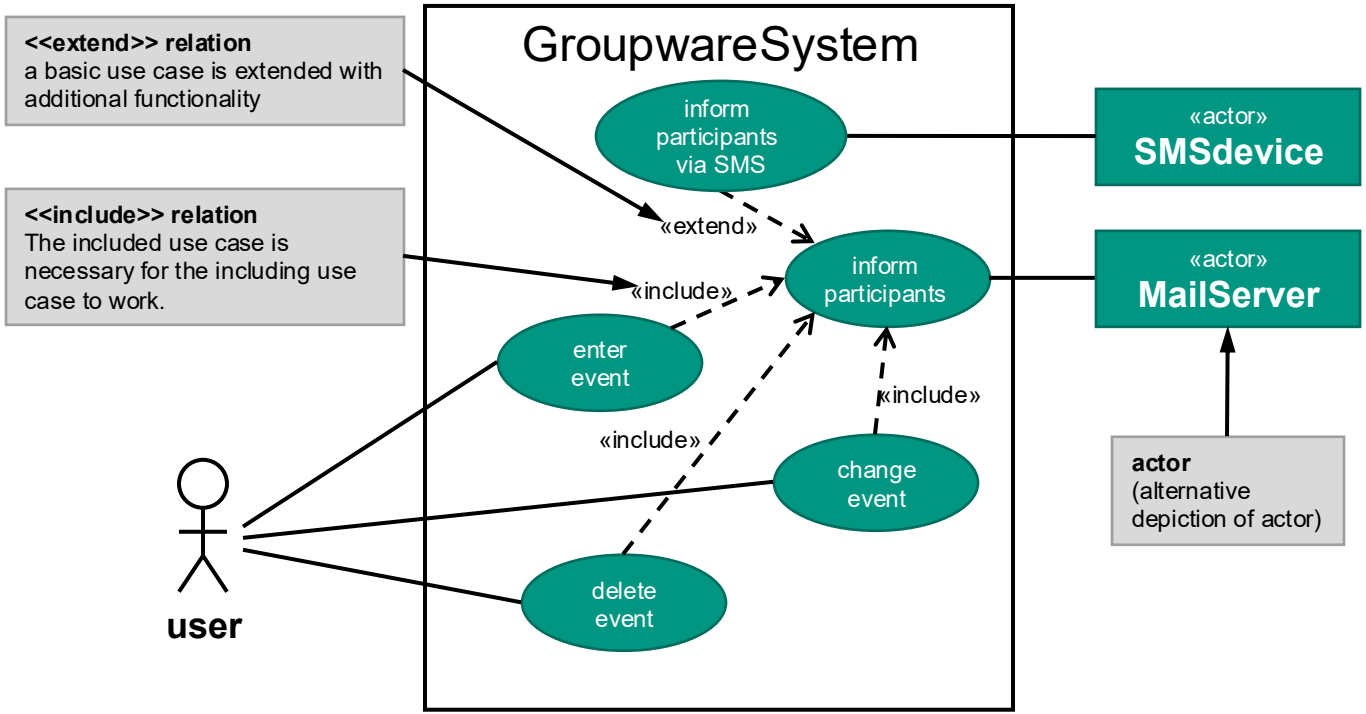
- Def. **use case**: a typical, desired interaction of one or more actors with a (technical or business) system.
  - A use case is always initiated by an actor and usually leads to results visible to an actor.
  - A use case describes ***what*** a system is to accomplish, not ***how*** to accomplish it. A single use case may indeed include several alternative processes.



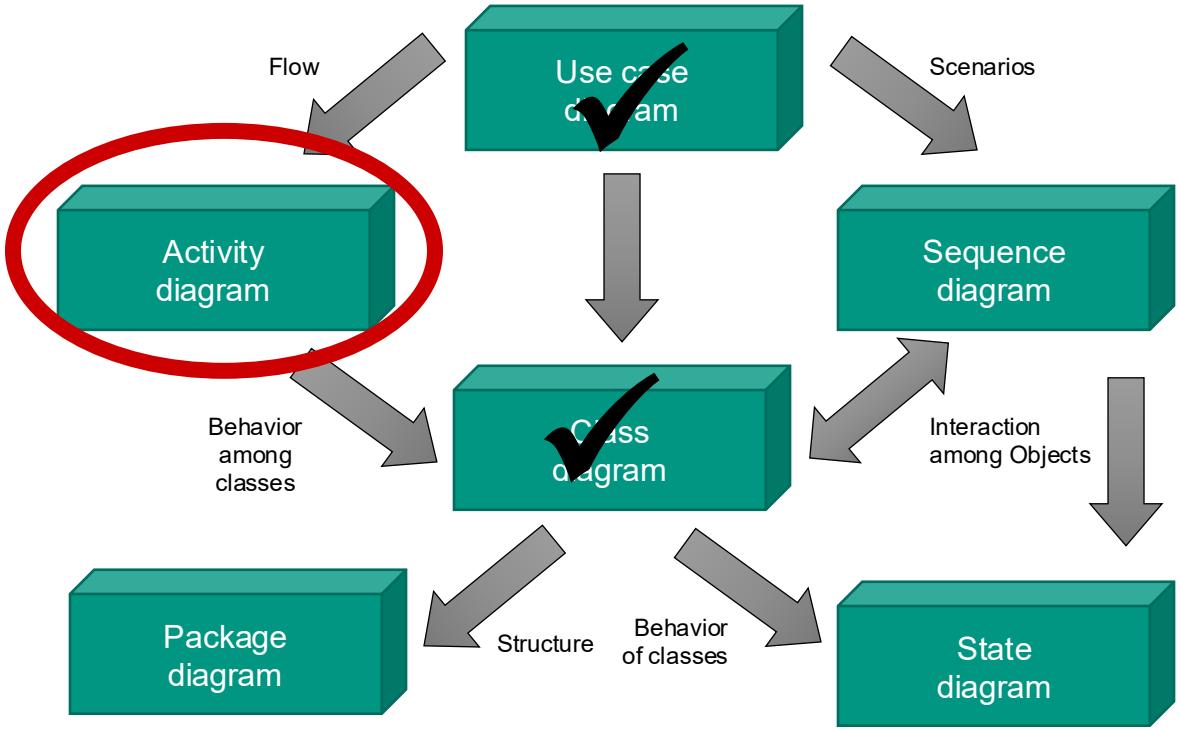
Example „Groupware System“



Additional diagram elements



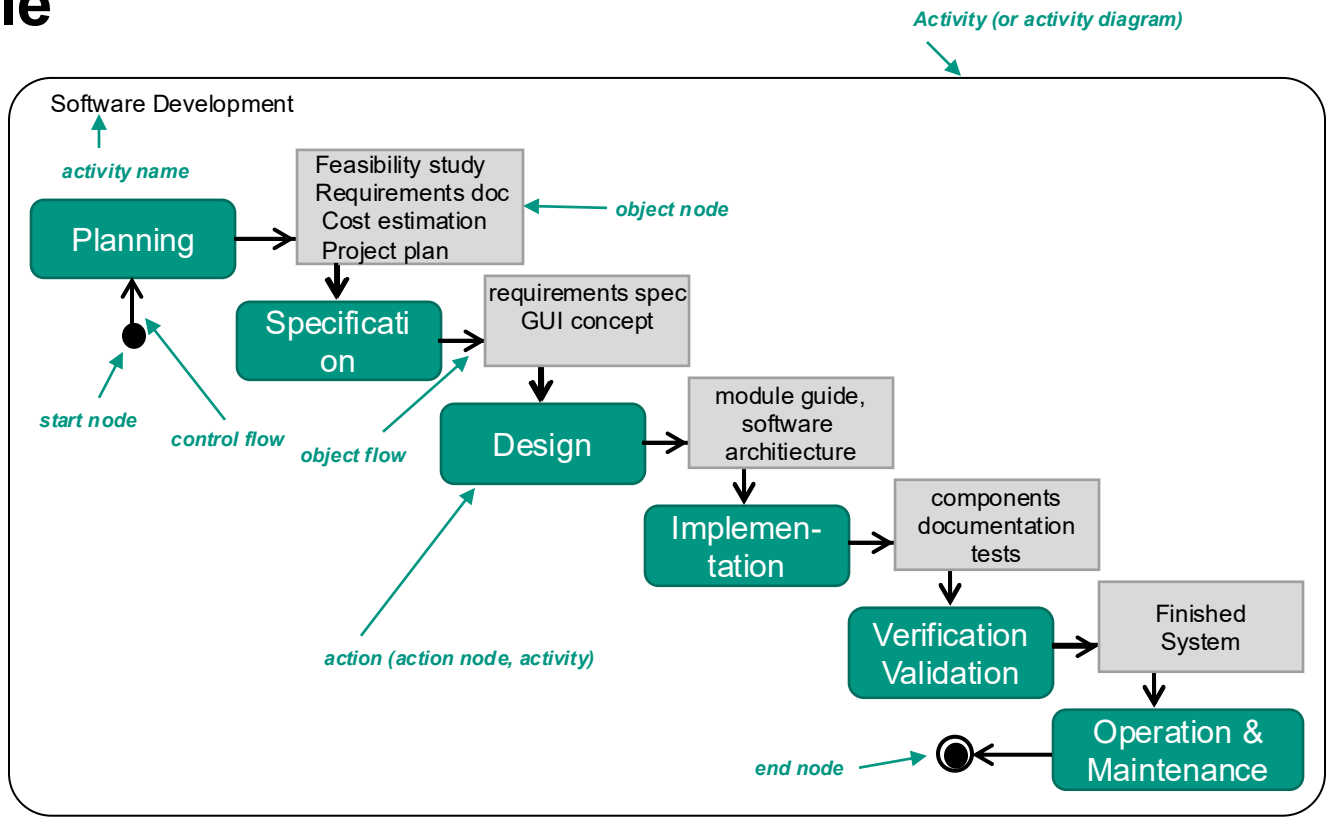
UML-diagrams



Purpose of Activity Diagrams

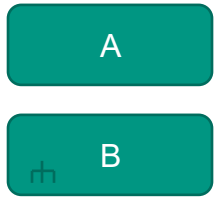
- Activity diagrams model the flow of control and objects from one activity to another
- They are used to model
  - business or managerial processes
  - use cases
  - algorithms
- Activity diagrams are dynamic models of what should happen in the modeled system
- They can model sequential as well as parallel processes
- Elements of activity diagrams:
  - action nodes, object nodes, control nodes
  - Object flows and control flows

Example


Activity (or activity diagram)
Software Development
activity name
Feasibility study
Requirements doc
Cost estimation
Project plan
object node
Planning
Specificati
on
requirements spec
GUI concept
start node
control flow
object flow
Design
module guide,
software
architiecture
Implemen-
tation
components
documentation
tests
action (action node, activity)
Verification
Validation
Finished
System
Operation &
Maintenance
end node

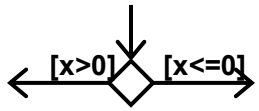
Elements of an activity diagram

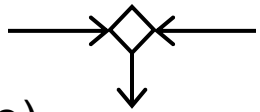
- Actions or activities
  - elementary actions
  - nested actions
- Nodes
  - start node
    - Starting point of an activity
  - end node
    - terminates all actions and control flows
  - control flow end
    - ends a single control or object flow

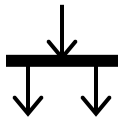


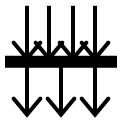
Elements of an activity diagram (2)

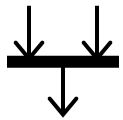
- Decision
  conditional branch
  (more than 2 branches possible)
- Join
  „or“-node
  (more than 2 incident arrows possible)
- Parallel launch
  two (or more) parallel control flows
- Synchronization
  „and“-node or barrier
  (more than 2 entries possible)

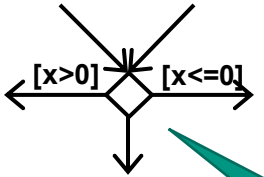










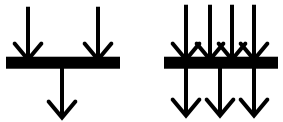


combinations possible

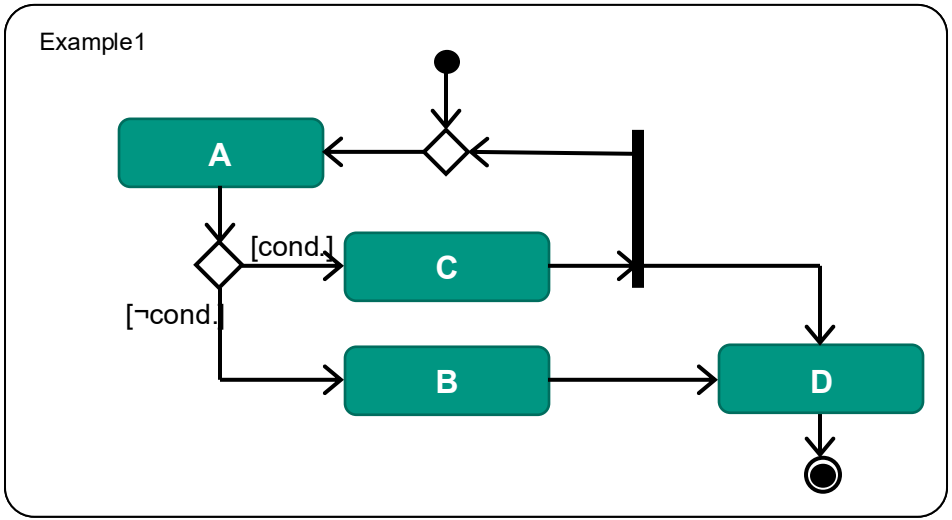
Execution semantics

- The progression of actions is determined by control flow edges (CFE) and object flow edges (OFE)
  - control tokens flow on CFEs, and objects on OFEs

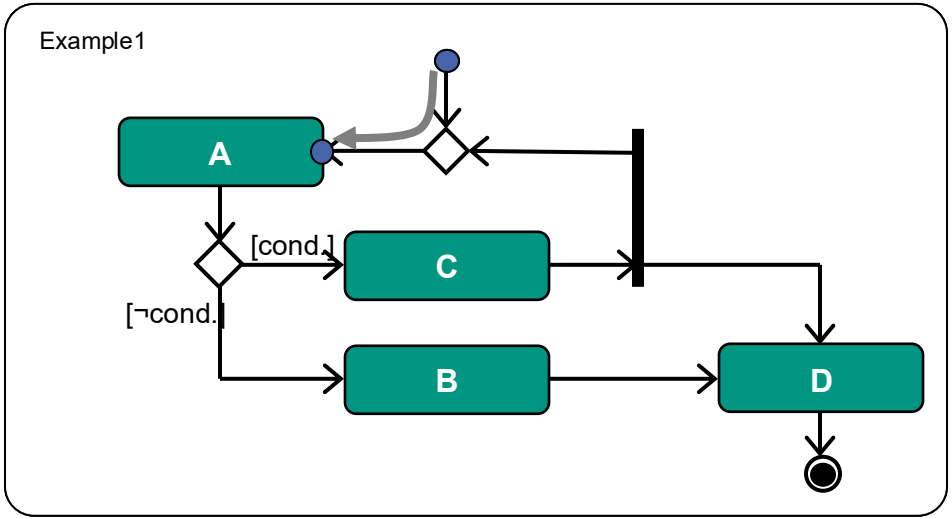
- An action can be executed as soon as all incident CFEs and OFEs carry a token or object, resp.
- When an action begins, tokens or objects are removed from all incident edges
  - **all** tokens present at CFEs
  - **one** object from each OFE (for multiplicity 1)
- After the end of an action, tokens or objects are placed on **all** outgoing edges. If there are several outgoing edges, all following actions will receive a token/object, which models parallelism.
- The same applies to barriers



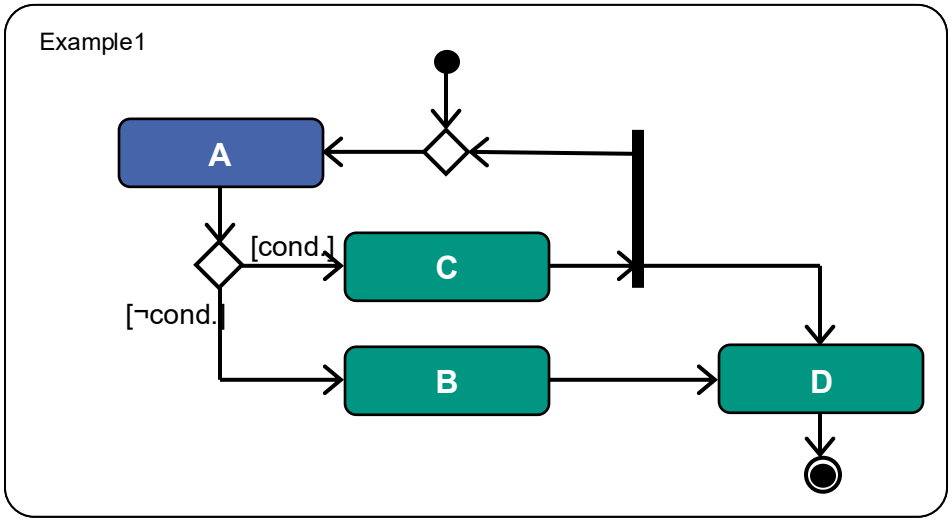
Execution semantics – example, step by step



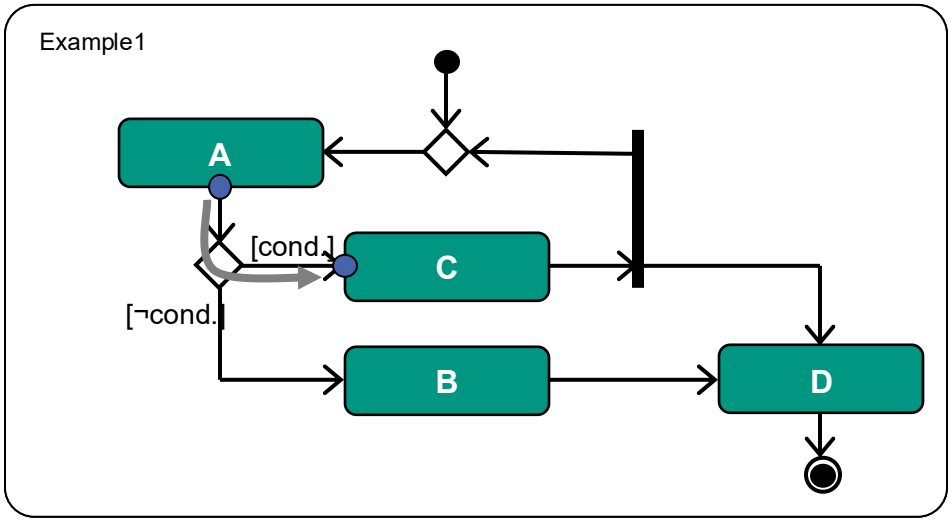
Execution semantics – example, step by step



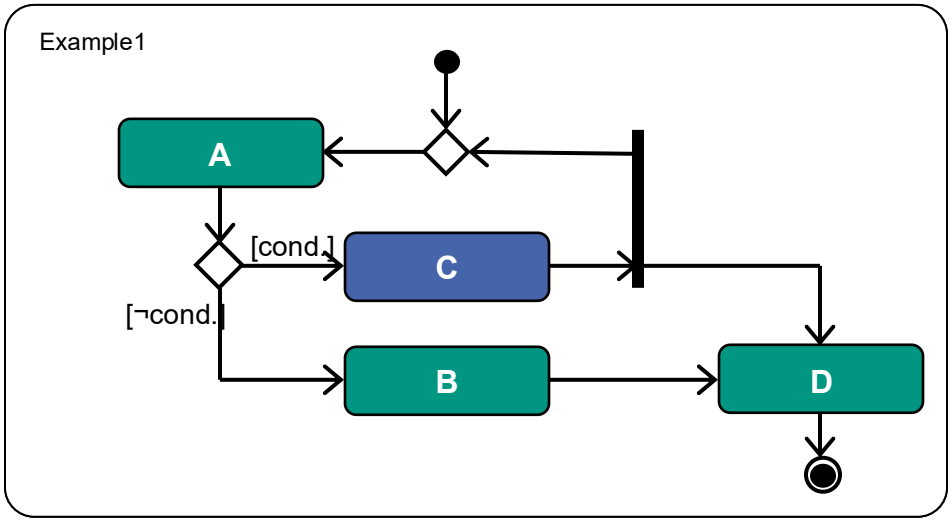
Execution semantics -- example, step by step



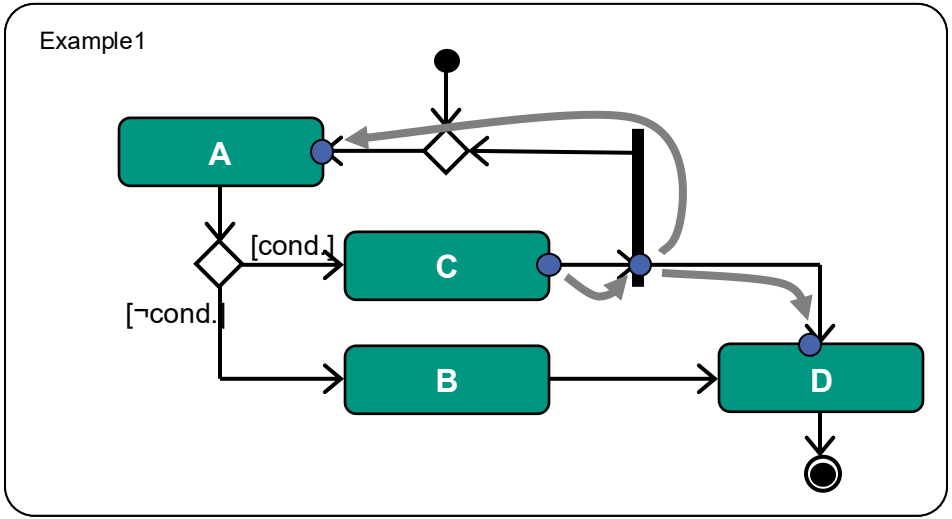
Execution semantics -- example, step by step



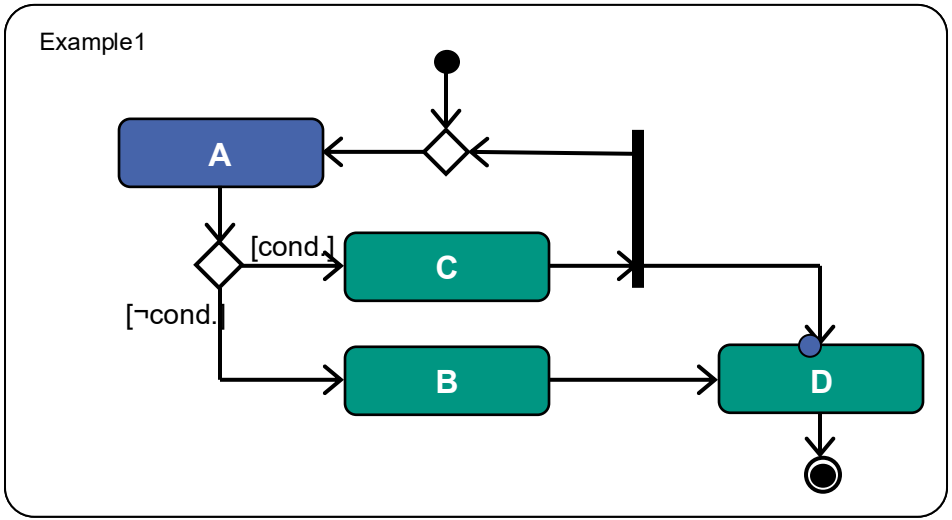
Execution semantics -- example, step by step



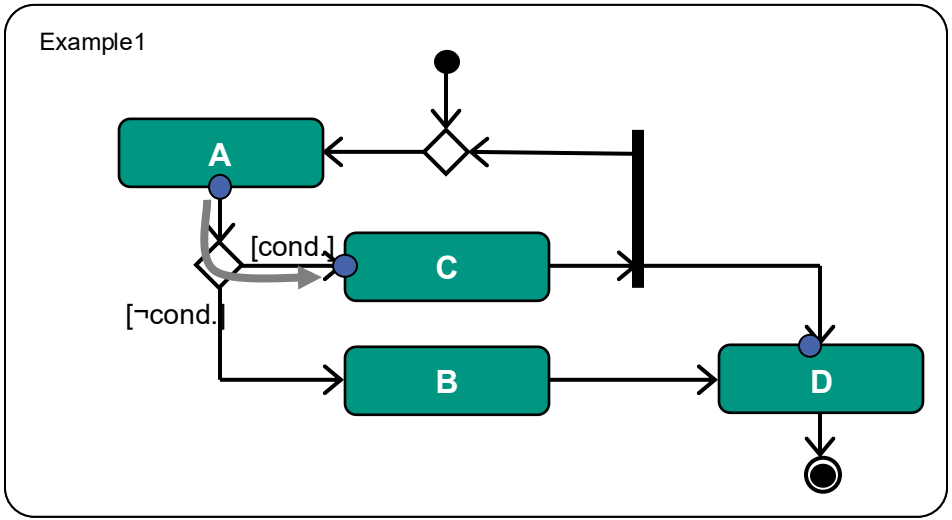
Execution semantics -- example, step by step



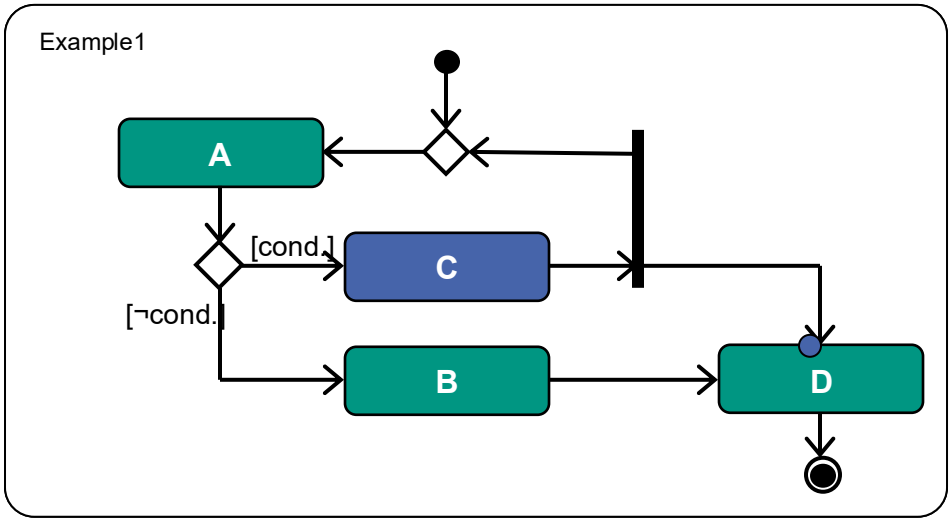
Execution semantics -- example, step by step



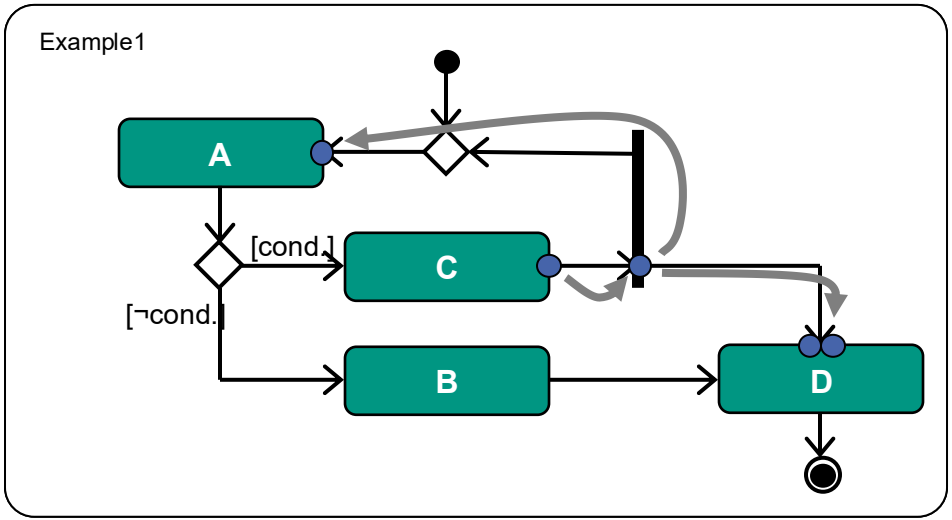
Execution semantics -- example, step by step



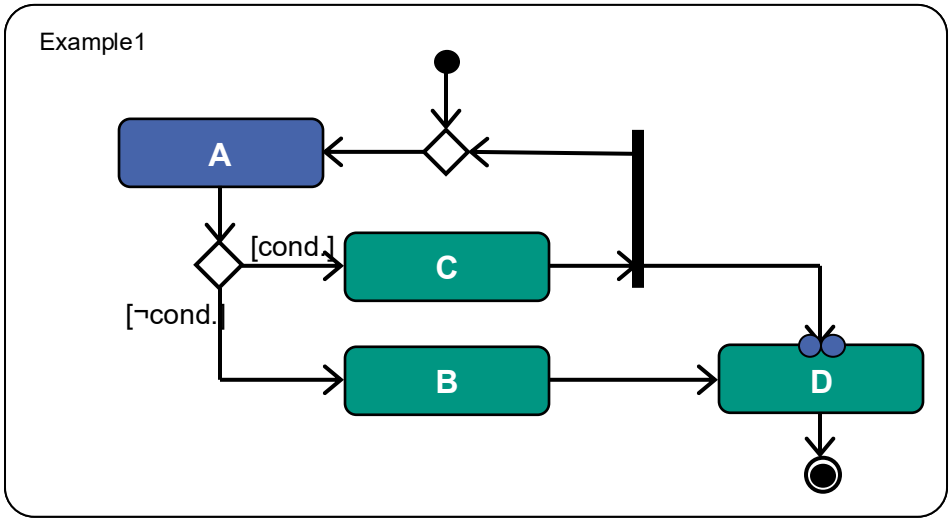
Execution semantics -- example, step by step



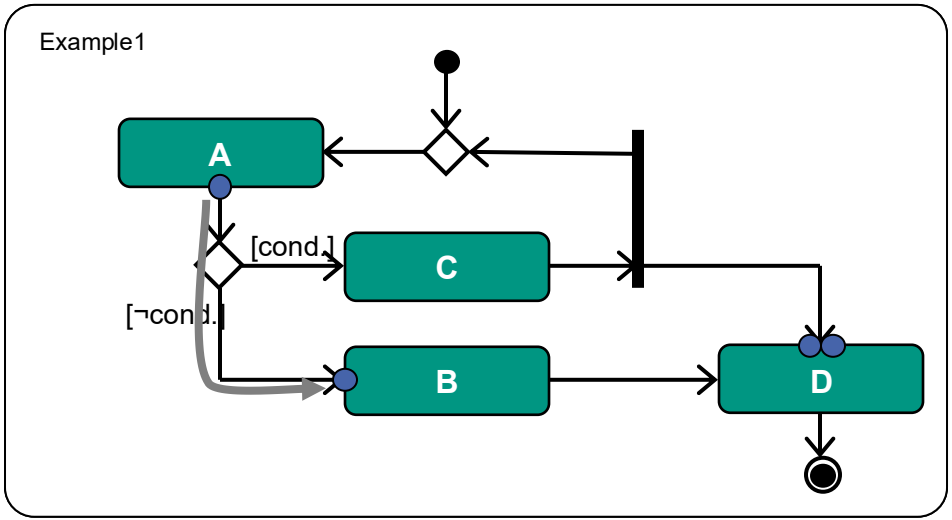
Execution semantics -- Example, step by step



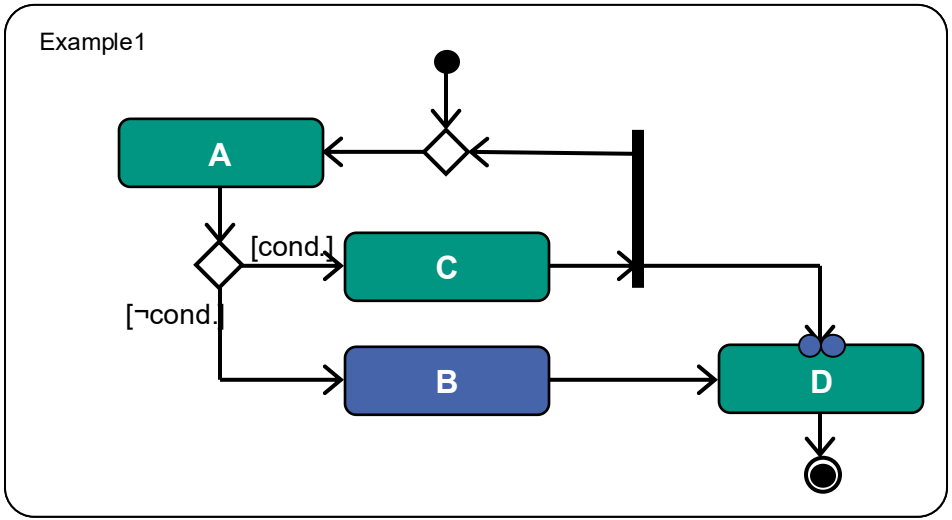
Execution semantics -- example, step by step



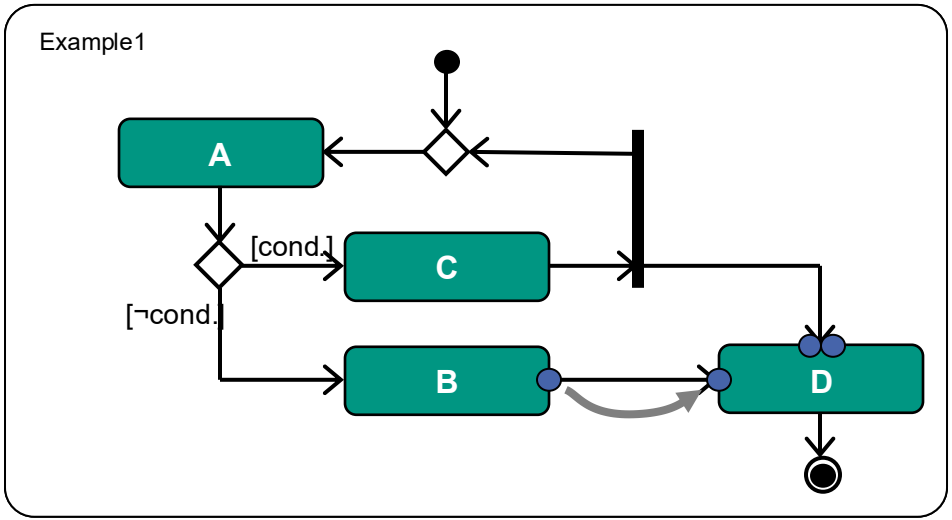
Execution semantics -- example, step by step



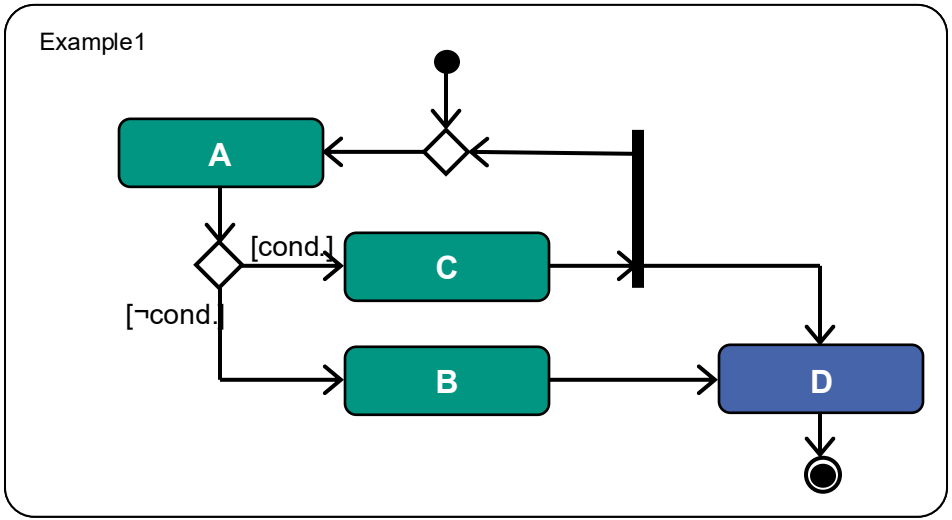
Execution semantics -- example, step by step



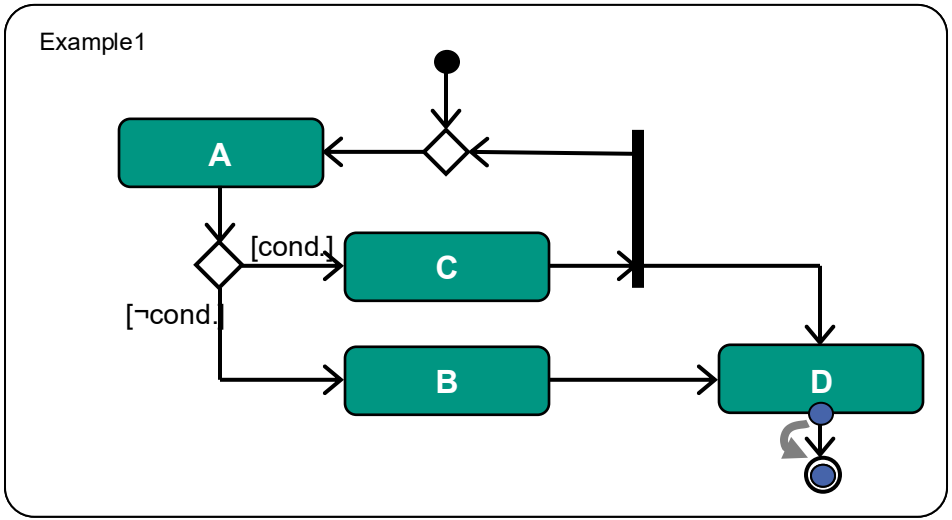
Execution semantics -- example, step by step



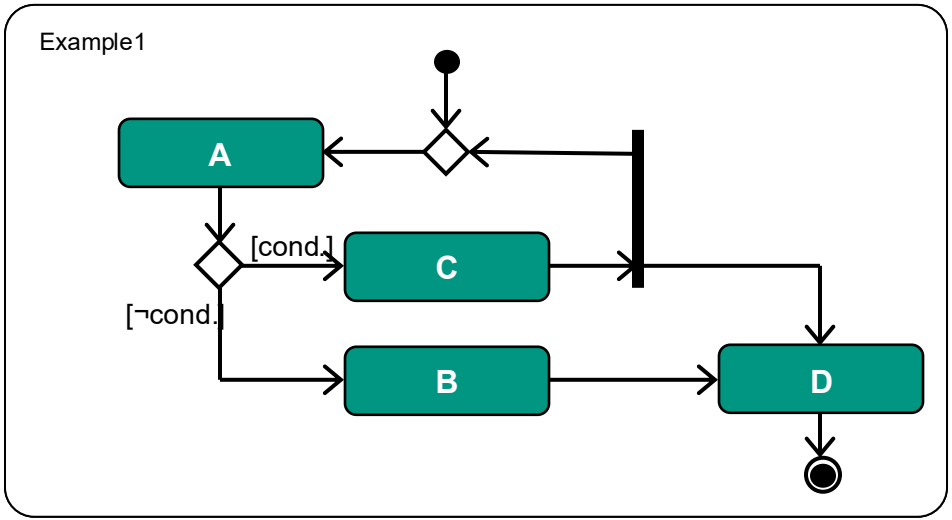
Execution semantics -- example, step by step



Execution semantics -- example, step by step

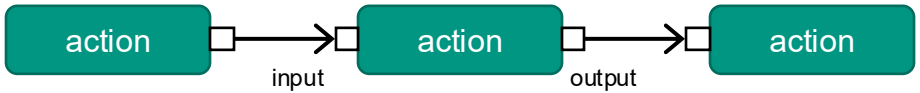


Execution semantics -- example, step by step

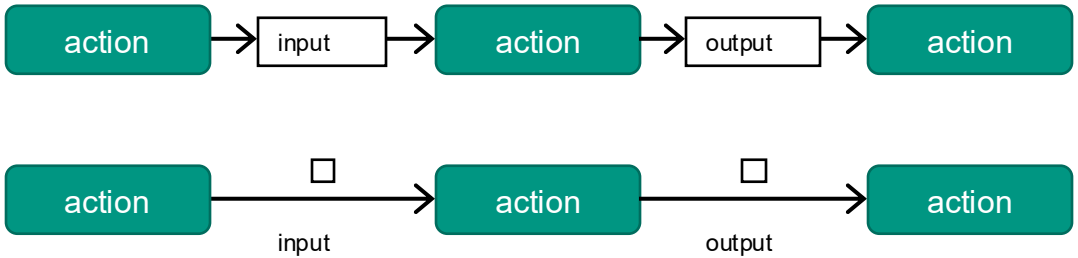


Elements of activity diagrams (3)

- Objectnodes
  - Input or output of an action
  - Shown as “pins”

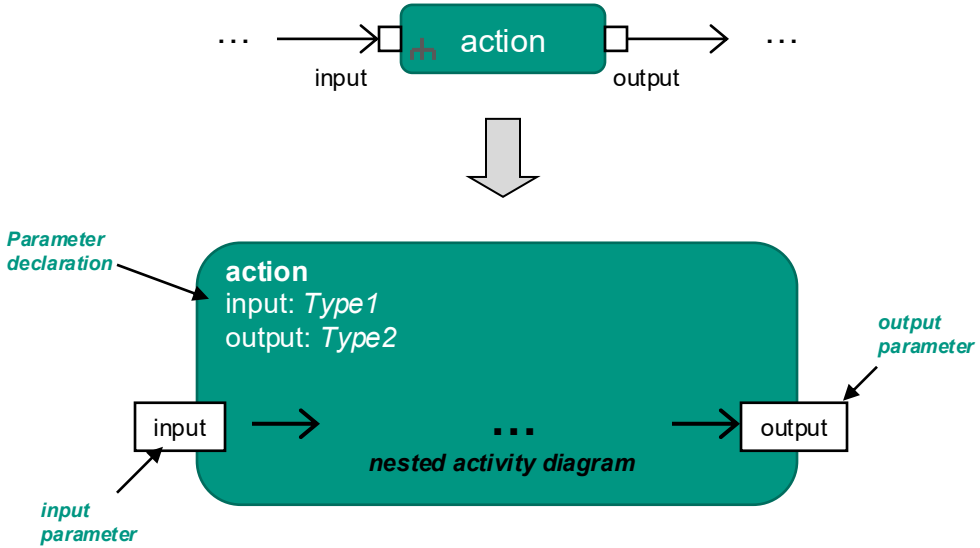


  - Alternative representation



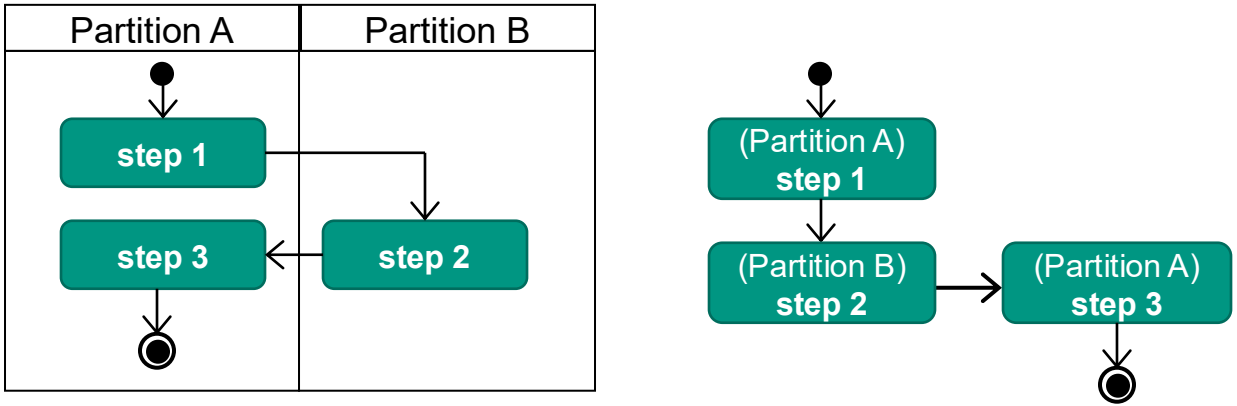
Elements of activity diagrams (4)

- Nesting of actions, parameters of actions



Elements of activity diagram (5)

- Partitions
  - Partitions describe who or what is responsible for an action, or if actions have common properties.
  - Partitions could be different computers, for example client and server, or system and user



Example from UML Specification 2.5.1

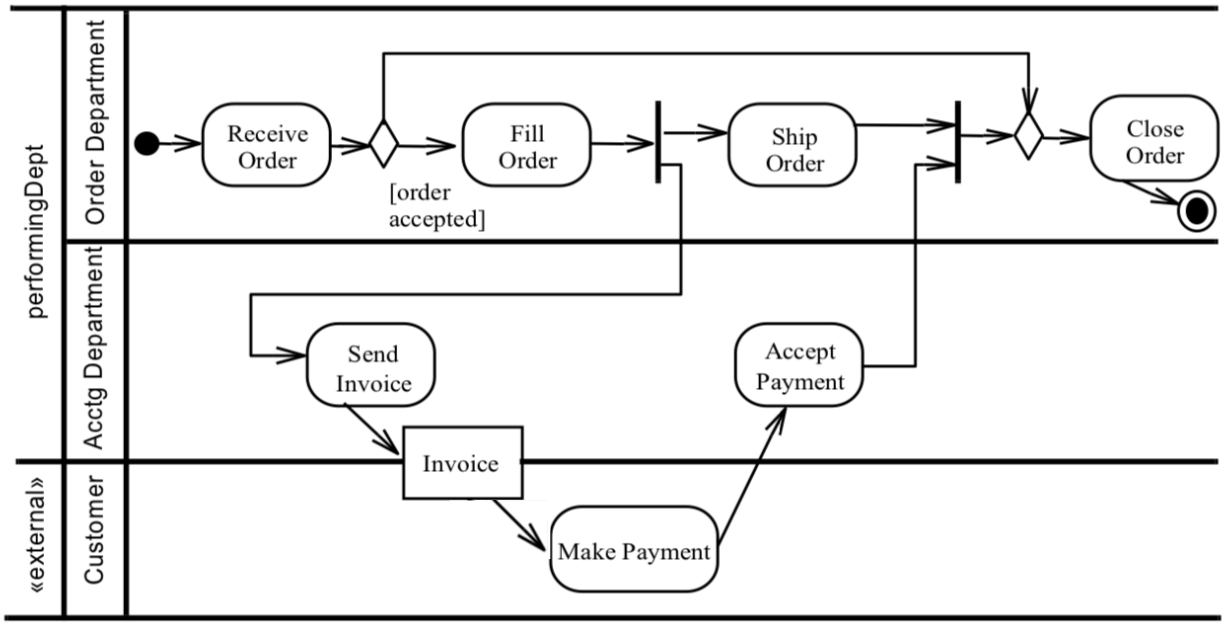
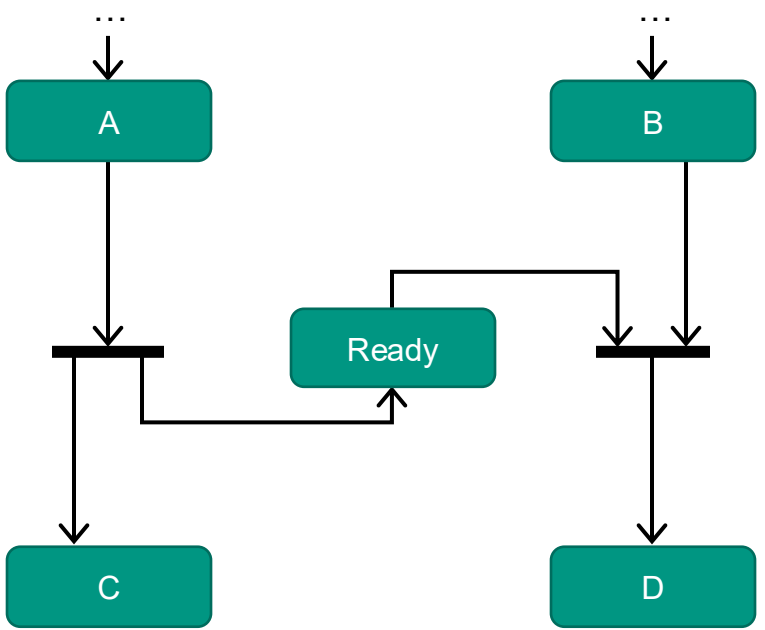


Figure 15.70 ActivityPartitions using swimlane notation

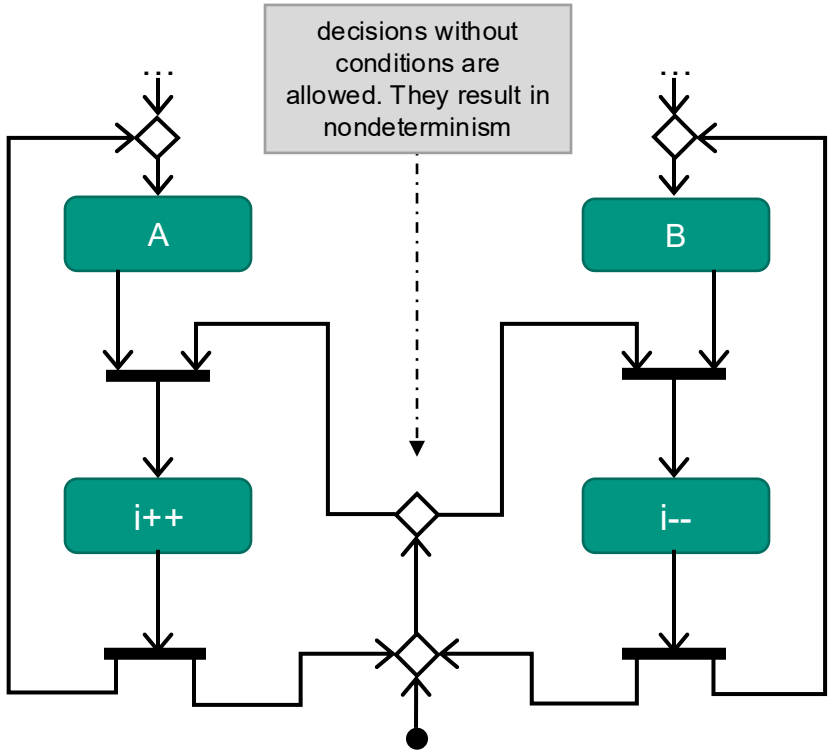
Synchronization with activity diagrams

Waiting for result or event
(D waits for A to complete)

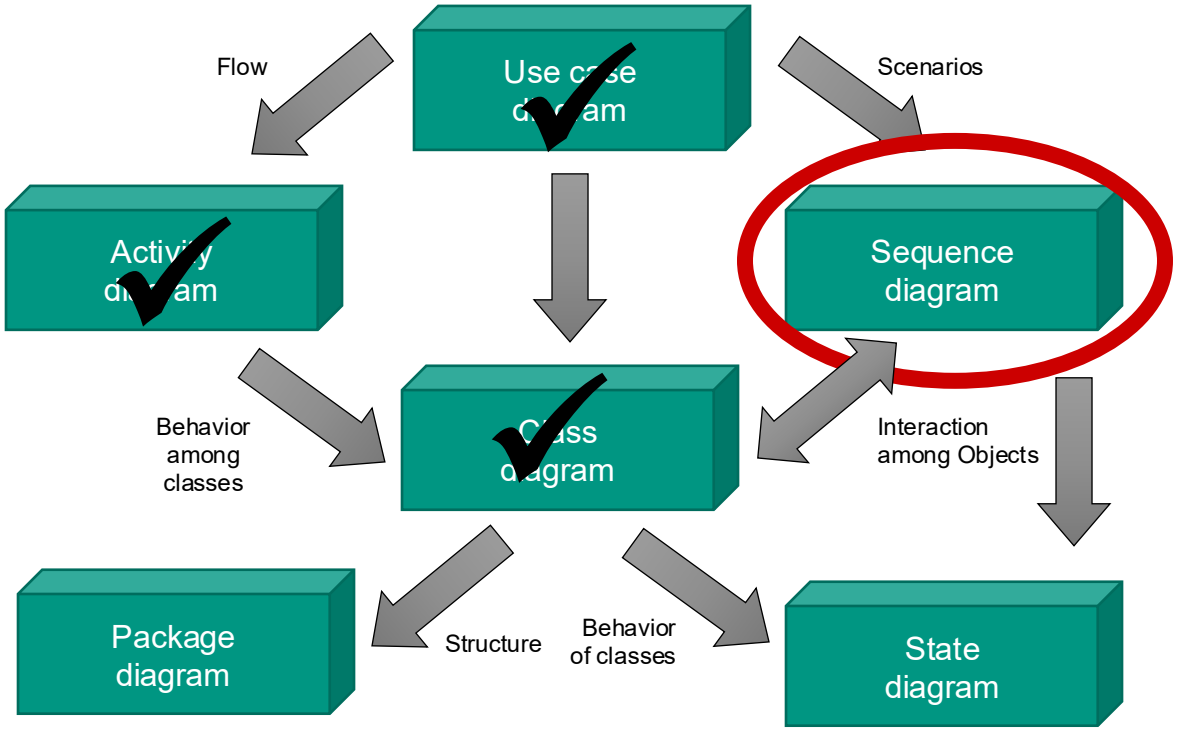


Synchronization with activity diagrams

Mutual exclusion
(i++ and i- - cannot happen simultaneously)



UML-diagrams

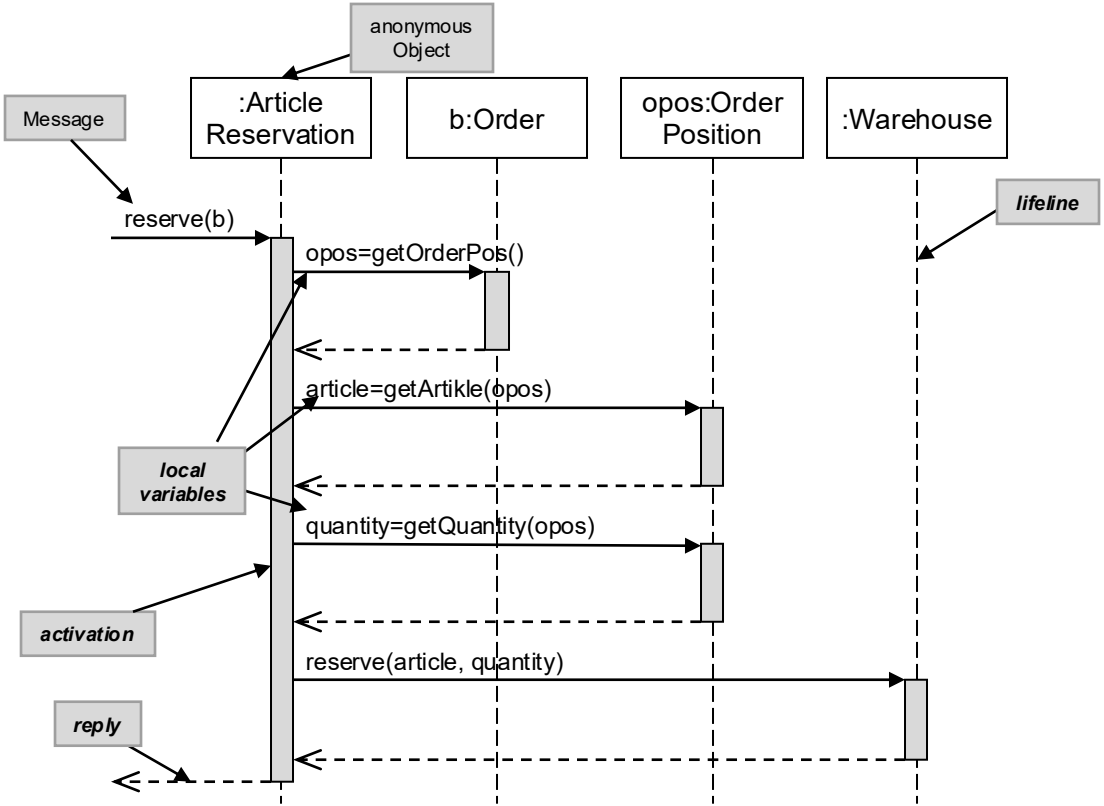


Sequence diagrams

Purpose of sequence diagram

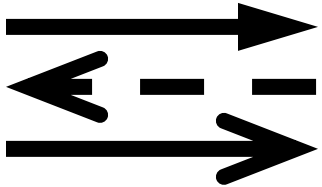
- Depiction of the sequence of method calls of a use case.
  - decisions and loops are possible („alt“, „loop“), but those should be used rarely
- Shows the temporal progression of messages
  - time runs top to bottom
  - roles are vertical dashed lines
  - messages are shown as horizontal arrows.
- Intuitive, but needs lots of space

Example


anonymous Object
Message
:Article Reservation
b:Order
opos:Order Position
:Warehouse
lifeline
reserve(b)
opos=getOrderPos()
article=getArtikle(opos)
local variables
quantity=getQuantity(opos)
activation
reserve(article, quantity)
reply

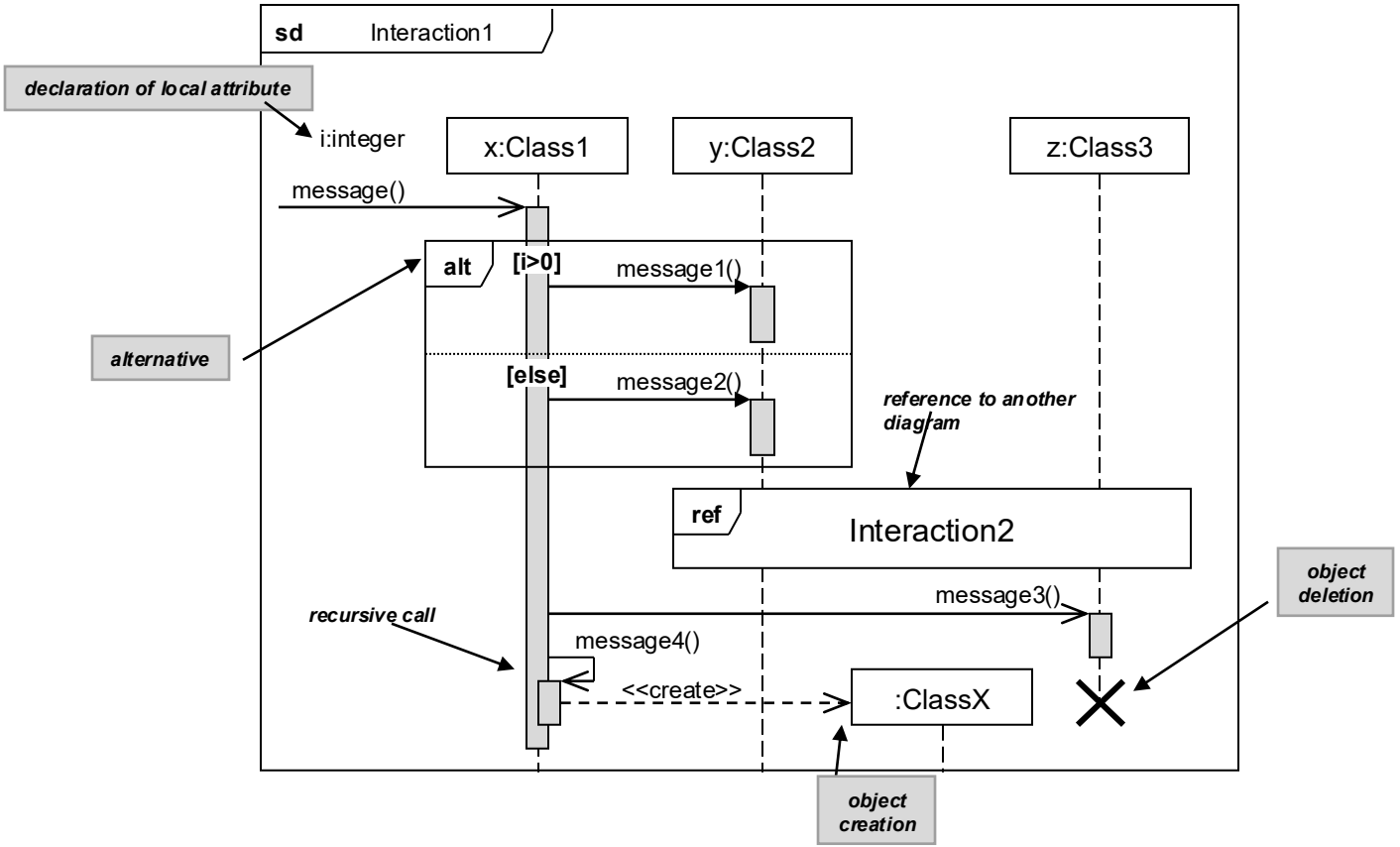
Elements of a sequence diagram

- message types
  - synchronous message
  - reply (optional)
  - asynchronous message
    (caller returns immediately, call runs in parallel)



- Activation
  - vertical bar overlaid on lifeline
  - Shows where the control is (from message/call to return/reply)

Additional Elements

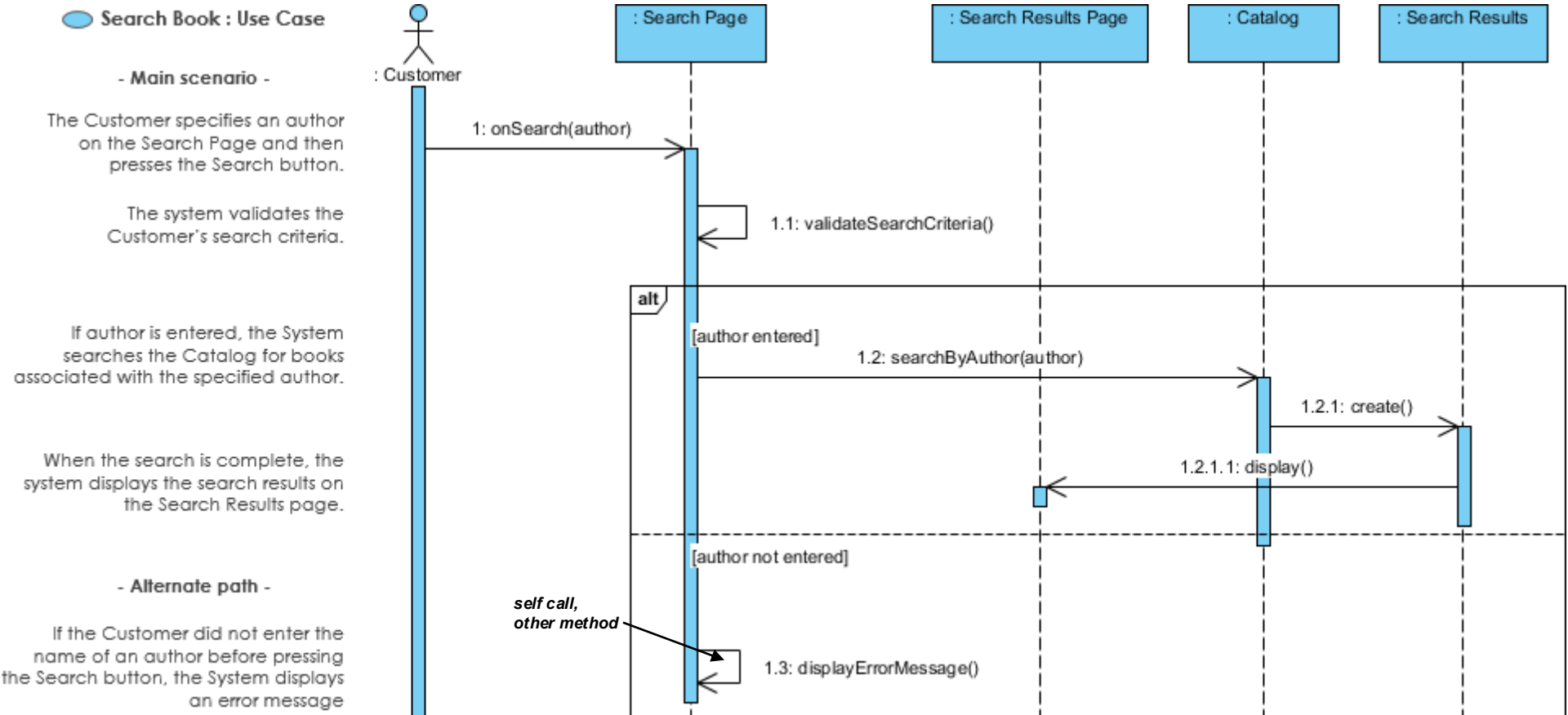


Operators

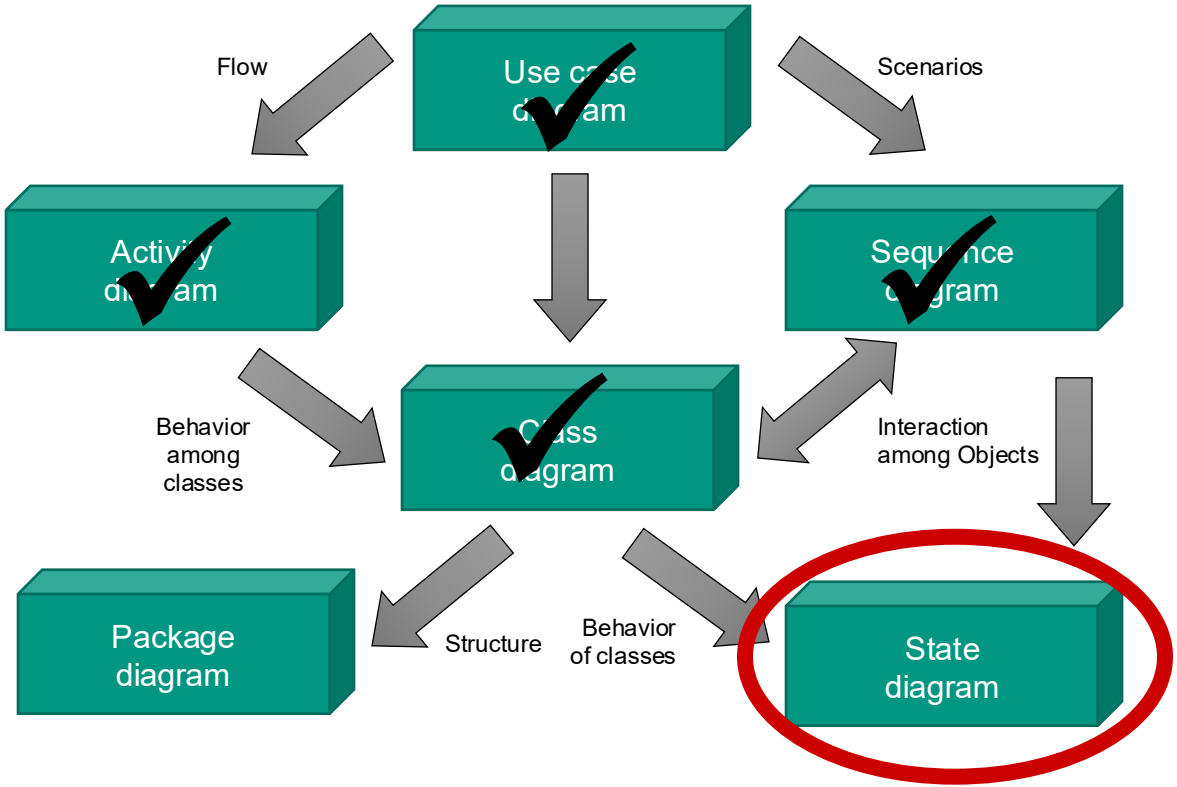
- Operators express alternatives and loops
  - Use sparingly, because they can clutter the diagram
  - If there are many alternatives, an activity diagram might be better

Operator	cond./Parameter	Semantic
alt	[cond.1], [cond.1], … [else]	Only one alternative is executed.
break	[condition]	If the condition is true, the block is executed and then the scenario ends.
opt	[condition]	Optional sequence, executed only if the condition is true
par		Contained sequences are executed in parallel.
loop	[condition]	Block is executed as long as the condition is true (”while-loop”)

Another use case (source: visual-paradigm.com)



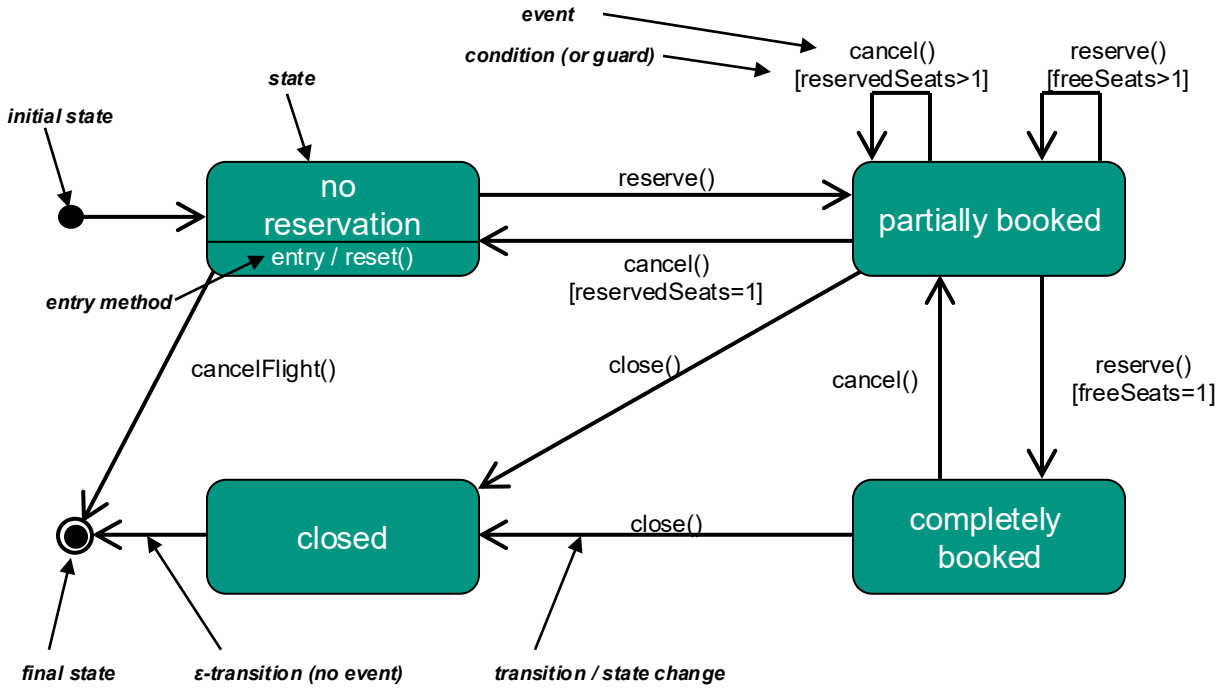
UML-diagrams



State diagram, statechart, state machine

- Purpose: Describe possible **states** of an object plus allowed **state changes** (finite state automaton)
- For classes that show different behavior depending on their states, for example
  - Real objects that are considered to run automatically (garage opener, washer, robot vacuum cleaner)
  - communications protocols
  - reactive systems, interactive devices, interactive applications (money dispenser, music player)
- Applicability
  - model the entire life cycle of an object
  - model the state-dependent behavior of a method

Example „flight reservation“



State diagrams

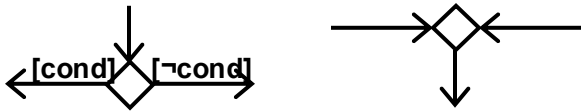
- A transition is triggered by an event
  - the event, the condition, and the operation performed during the transition are added to the transition arrow as follows:

```
event(arguments)
[condition]
/operation(arguments)
```

- A transition only takes place, when at the moment when the event happens the condition is true („guarded transition“)
- special case: ε-transition (or spontaneous transition) may happen anytime when the automaton is in the state where the ε-transition starts and the condition (if present) is true (there is no event associated with the ε-transition.)

State diagrams

- Special events
  - `at(expression)`
    The expression provides an exact and absolute point in time. The transition takes place as soon as that point in time is reached.
  - `after(expression)`
    The expression provides a relative point in time (counting from the current time). The transition takes place when the time given by „expression“ has passed.
- also allowed from activity diagrams:



- points to the initial state.

- final state

Actions

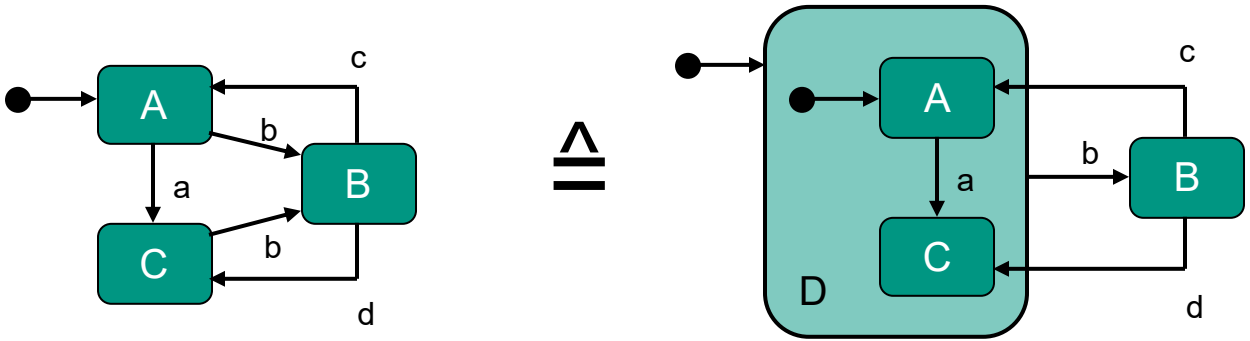
An action can be associated with a transition

- entry action: executed when a transition enters the state with the entry action
  Notation: `entry / action()`
- exit action: executed when a transition leave the state with the the exit action.
  Schreibweise: `exit / action()`
- Continuous action: executed as long as the state machine remains in the state with the continuous action
  Notation: `do / action()`

- An action is executed immediately when the transition takes place and requires no time.

Additional features

Hierarchical state machine



No Hierarchy

With Hierarchy

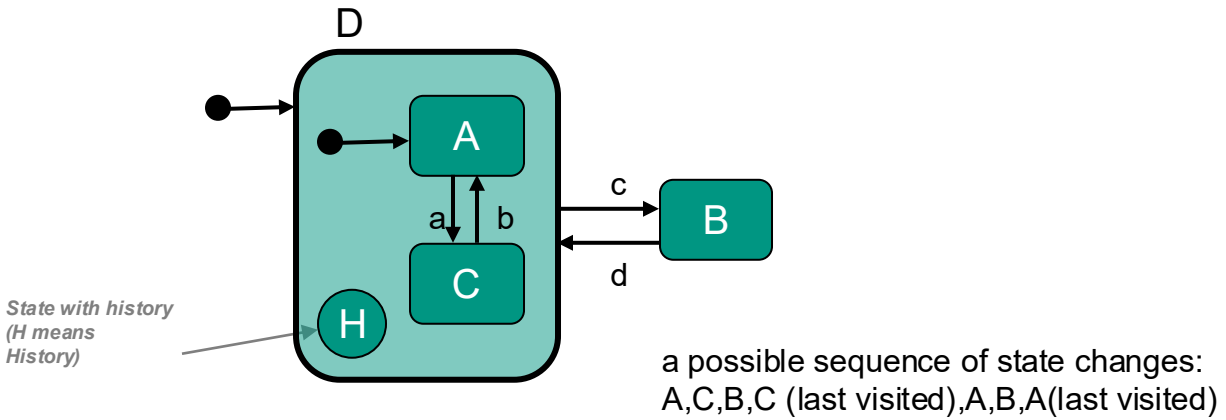
A hierarchical state machine can reduce the number of edges.

- For instance, one could define an error exit for all states this way

Additional features

States with history

- When transitioning in a state with sub-states in a hierarchical state machine, the transition ends in the last visited sub-state.

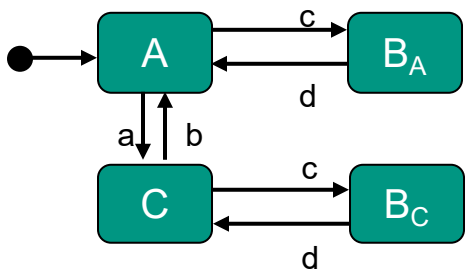


a possible sequence of state changes:
A,C,B,C (last visited),A,B,A(last visited)

Additional features

Is an automaton with history more powerful than one without?

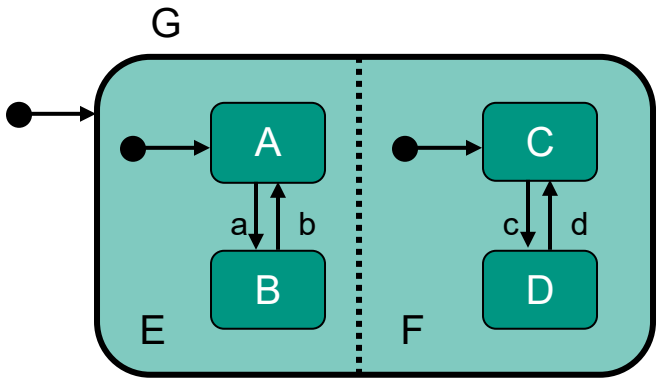
- No, because it is possible to simulate the history attribute with a finite number of additional states.
- Simulation of previous example without history:



Additional features

Concurrency

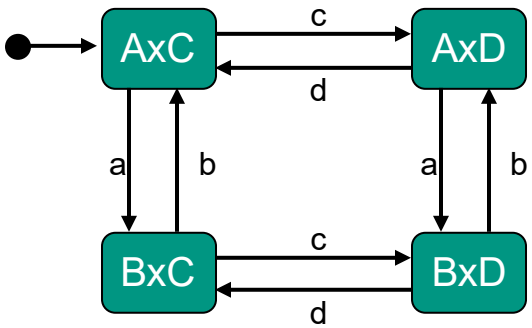
- While the automaton is in state G, it can assume any state in the cross product of E×F



Additional features

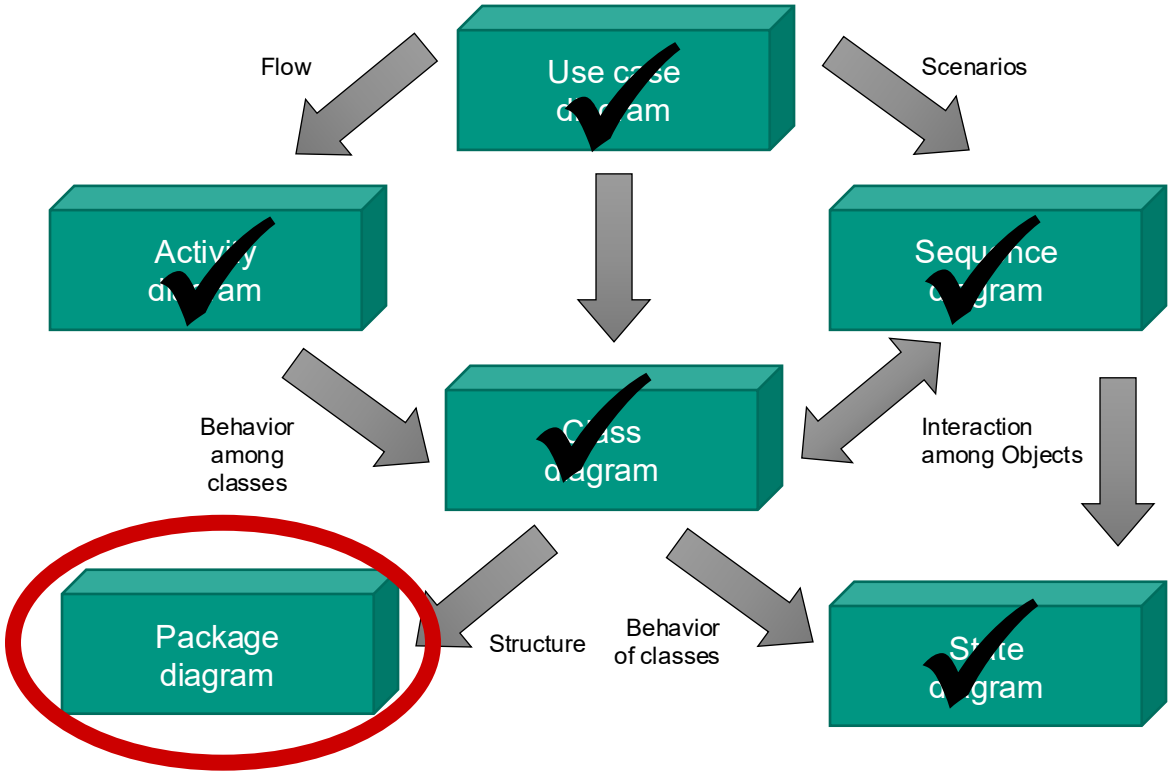
Is an automaton with concurrency more powerful than one without?

- No, because one can simulate concurrency with a finite number of additional states.
- Simulation of the previous example without concurrency:



- In this example, the number of states does not increase. In general, the number of states needed for simulation with two-fold concurrency is |E|x|F|

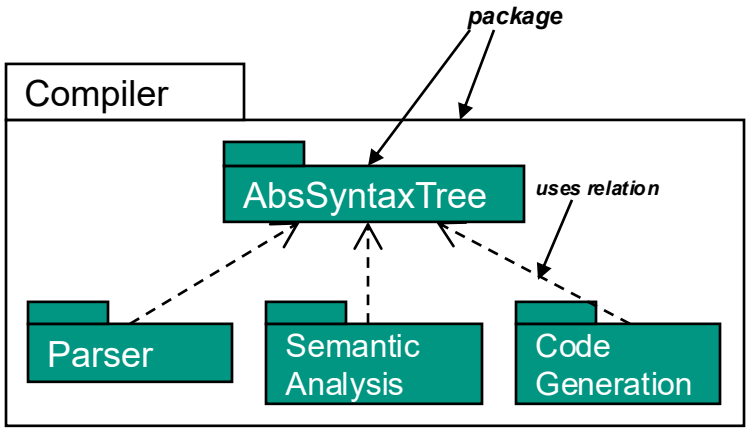
UML-diagrams



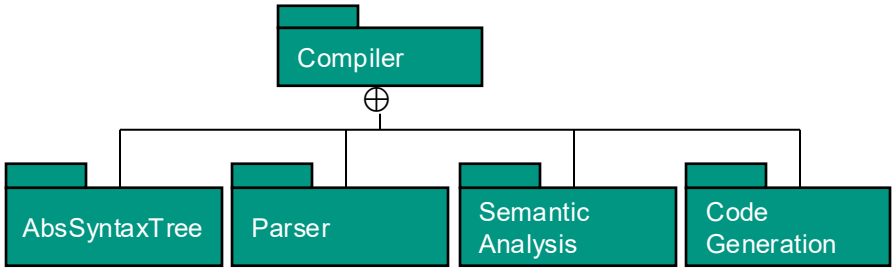
Package diagram

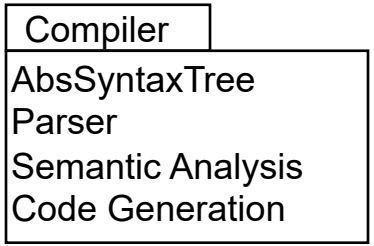
- Packages are sets of UML model elements of arbitrary types (class diagrams, use case diagrams, state charts, etc.)
- The package diagram helps structure a large model in manageable junks.
- Each model element has a name that is unique in the package.
- Each model element may have a visibility
  - standard visibility is `public`
- A model element can be identified with a qualified name:
  - `Packagename::ModelElementName`
- Dependencies among packages are indicated with a dashed arrow.

Example



Alternatives (without uses relation)





Literature

- UML-Specification http://www.omg.org/spec/UML/2.5.1/
  - „Superstructure Specification“
  - „Infrastructure Specification“
- Wikipedia