

3.5 State Diagrams

Subject/Topic/Focus:

- Intraclass behavior

Summary:

- States and transitions
- Events, guards, actions and activities
- Abstraction, nesting and concurrency

Literature:

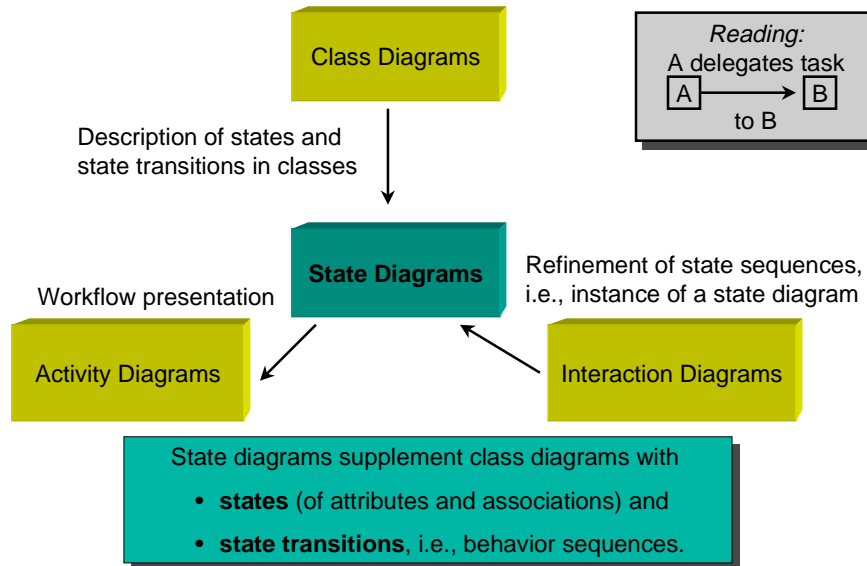
- Fowler
- Rumbaugh

- Developed as "Statecharts" by David Harel.
- Successor to Deterministic Finite Automata.

Motivation

- Object-orientation = Structure + Behavior.
- Class diagrams capture
 - structure (attributes and associations) and
 - behavior interfaces (method signatures).
- How do we catch the **dynamic behavior** and **life cycle** of an object?
 - Creation and deletion.
 - Attribute and association changes.
- How does the object **interact** with other objects?
 - **Reacting to events** and to messages received by the object.
 - **Triggering actions** and sending messages to other objects.
 - **Handling** of sequences of events accepted and actions triggered.

Role of State Diagrams in UML



OOA&D © J.W. Schmidt, F. Matthes, TU Hamburg-Harburg

3.5.3

State Diagrams

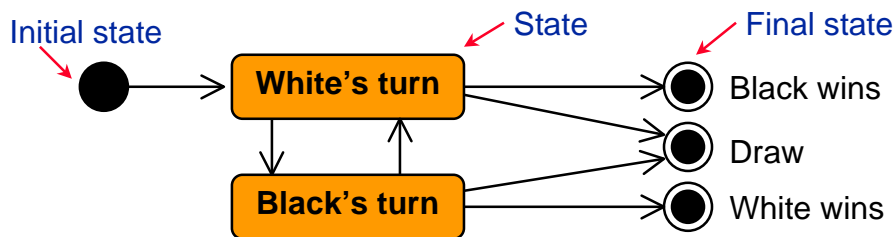
State diagrams are a technique to describe the **behavior**, i.e., **state changes** of a **single class** according to **events** and **messages** which the class **sends** and **receives**.

OOA&D © J.W. Schmidt, F. Matthes, TU Hamburg-Harburg

3.5.4

States

- A state
 - abstracts from attribute values and associations of an object;
 - represents the internal condition/state of an object **for a certain period of time**;
 - corresponds to an interval of time between two events.
- The response to events may depend on the state of an object.
- Object **creation** comes together with **an initial object state**.
- Object **deletion** may be related with (one of many) **final states**.

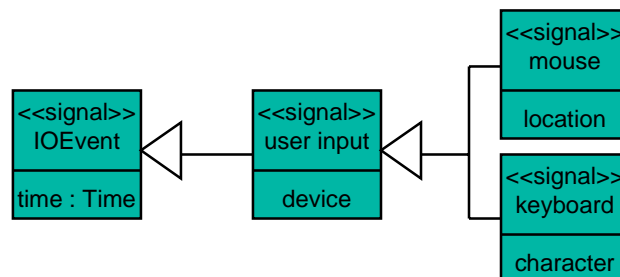


OOA&D © J.W. Schmidt, F. Matthes, TU Hamburg-Harburg

3.5.5

Events

- An **event** is something worth noticing at a **point of time**;
 - a signal from one object to another, e.g., “delivered”,
 - a message received by an object, e.g., “check item”,
 - a certain date/time, e.g., “after 10 seconds” (being in a certain state) or “at 31-dec-2000, 00:00”.
- Events may take **arguments**, e.g., “deliver to (receiver : Customer)”.
- Events may be declared in a class diagram with arguments shown as attributes.

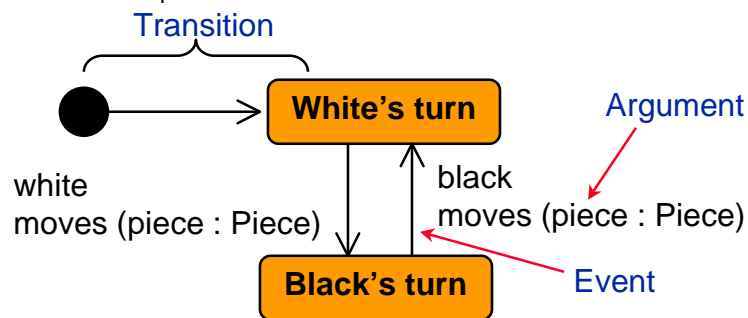


OOA&D © J.W. Schmidt, F. Matthes, TU Hamburg-Harburg

3.5.6

Transitions

- A **transition** represents a change of the internal condition/state of an object.
- A transition is usually **triggered** (“fired”) by an event. Transitions without event label (“lambda transitions”) fire immediately.
- Transitions fire instantly
 - from exactly one state to another state or to itself (self-transition) and
 - are not interruptible.

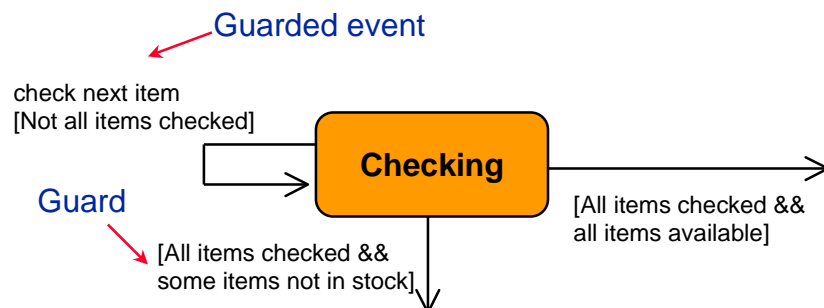


OOA&D © J.W. Schmidt, F. Matthes, TU Hamburg-Harburg

3.5.7

Guards

- A **guard** is a logical condition (value “true” or “false”).
- A guarded transition fires only if the guard resolves to “true”.
- Since only one transition can be fired in a given state, guards are intended to be mutually exclusive for any event.
- Events may be guarded.

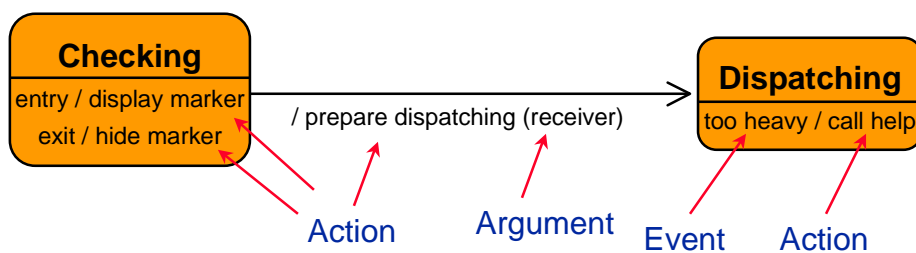


OOA&D © J.W. Schmidt, F. Matthes, TU Hamburg-Harburg

3.5.8

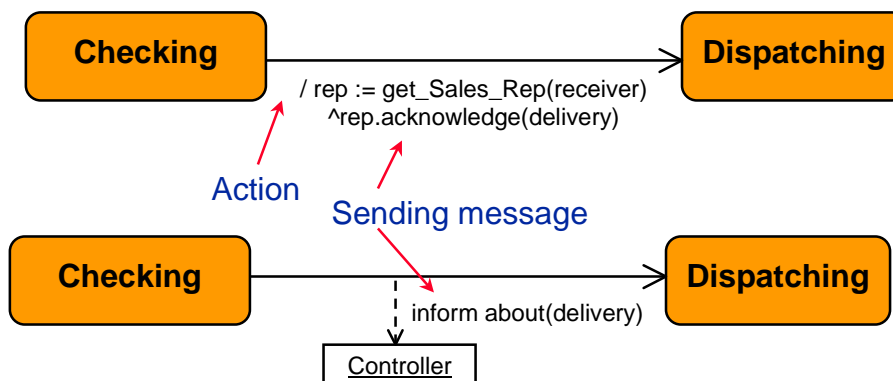
Actions

- An **action** is a short software process that executes immediately.
- A **transition** may **trigger** an action.
- Actions may be triggered on **entry** or **exit** of states (instead of labeling each incoming (entry) and outgoing (exit) transition with these actions).
- An **event** may trigger an action without leaving the state, i.e., without triggering exit and entry actions as a self-transition would do.
- An action may trigger events, usually in other objects.
- Actions may take **arguments**.



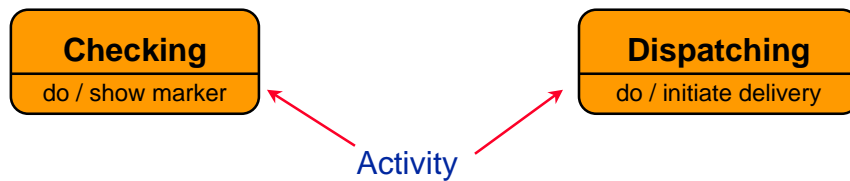
Sending Messages

- Sending messages are special actions.
- Messages address an object or a set of objects.
- Messages occur after other actions.
- Messages trigger an event/transition in the receiving object.

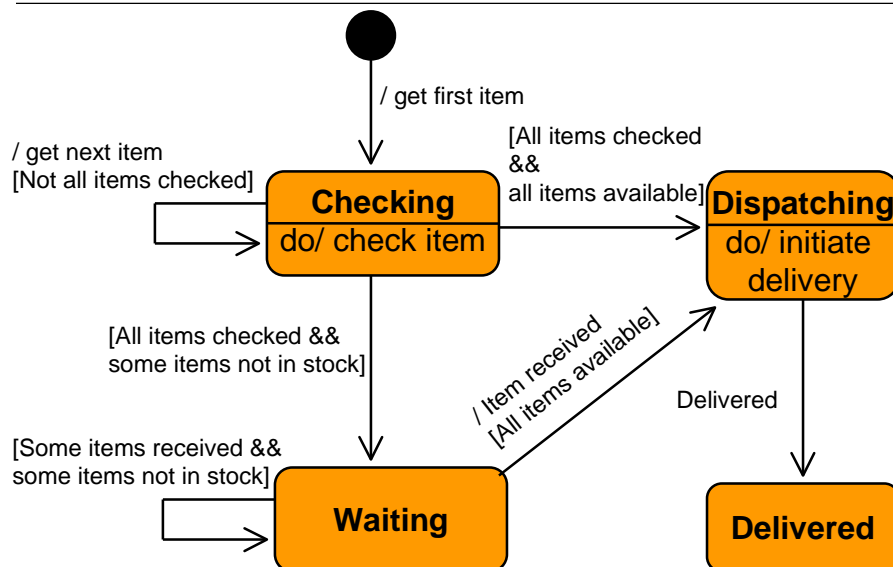


Activities

- Activities can take “longer”, i.e., they are processes which last as long as an object is in a certain **state**.
- Activities are **interruptible**, i.e., an event causing a state transition may abort an activity.
- Activities may be constructed from a start and a final action.

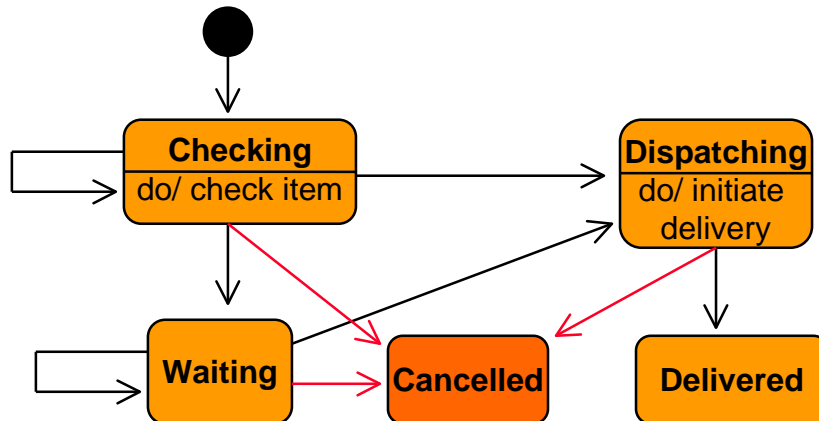


State Diagram: Example



Nesting (1)

Example: A state *Cancelled* is added to which transitions from all existing states exist.



OOA&D © J.W. Schmidt, F. Matthes, TU Hamburg-Harburg

3.5.13

Nesting (2)

Instead of drawing all the transitions from each state to the *Cancelled* state, it is easier to “wrap” the states *Checking*, *Waiting* and *Dispatching* in one **superstate** from which only one *cancelled* transition to the *Cancelled* state is necessary.

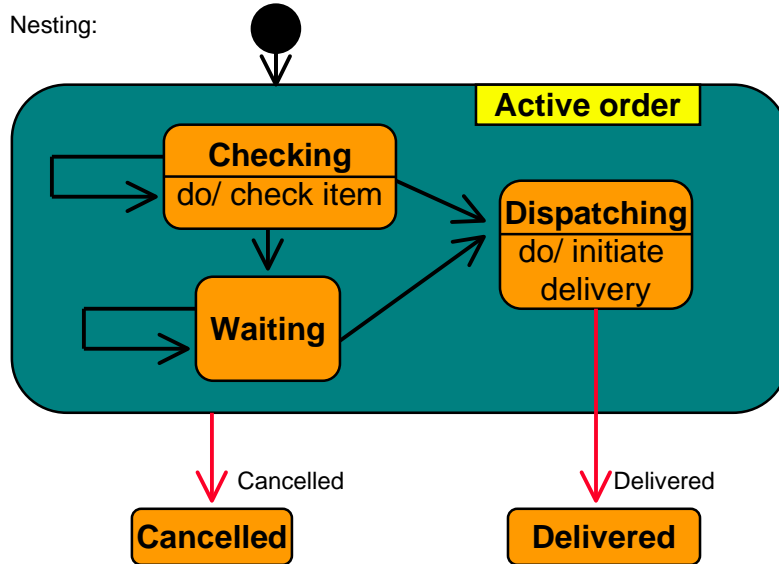
- Superstates contain state diagrams or other superstates.
- Superstates allow to simplify multiple transitions from probably many source states to a single target state by
 - introducing a (superstate) name for a (nested) state diagram and
 - substituting each of the transitions between source states and the target state by a single transition between superstate and target state.

OOA&D © J.W. Schmidt, F. Matthes, TU Hamburg-Harburg

3.5.14

Superstates: Nesting

Nesting:

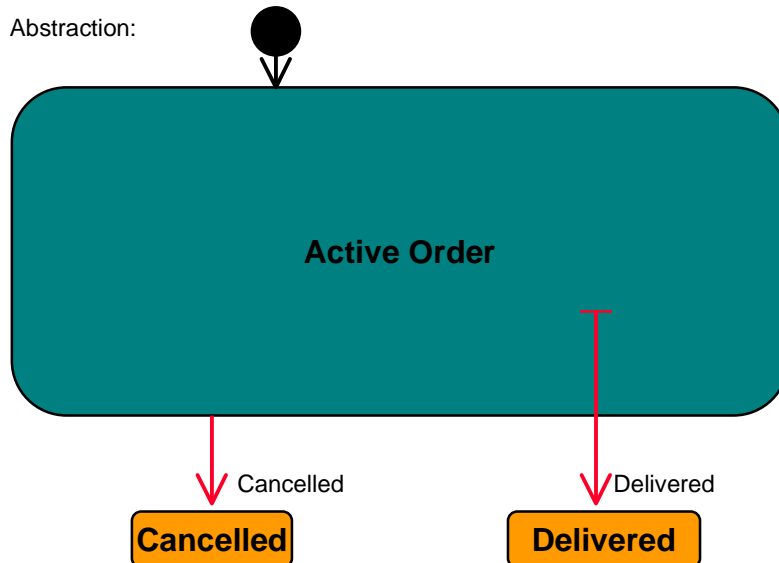


OOA&D © J.W. Schmidt, F. Matthes, TU Hamburg-Harburg

3.5.15

Superstates: Abstraction

Abstraction:



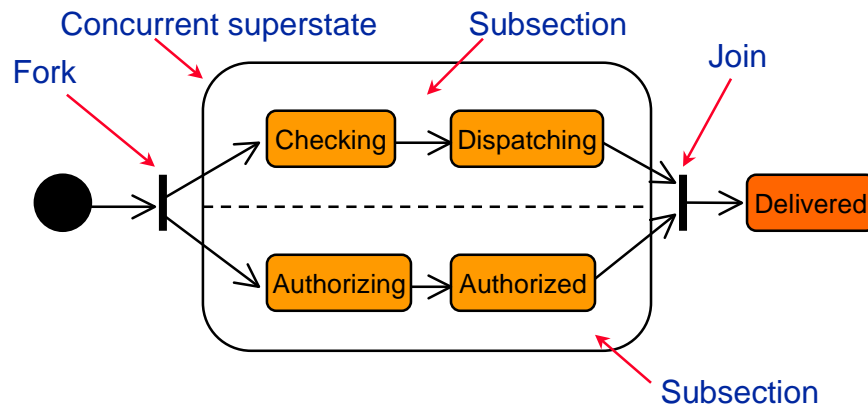
OOA&D © J.W. Schmidt, F. Matthes, TU Hamburg-Harburg

3.5.16

Concurrency in State Diagrams

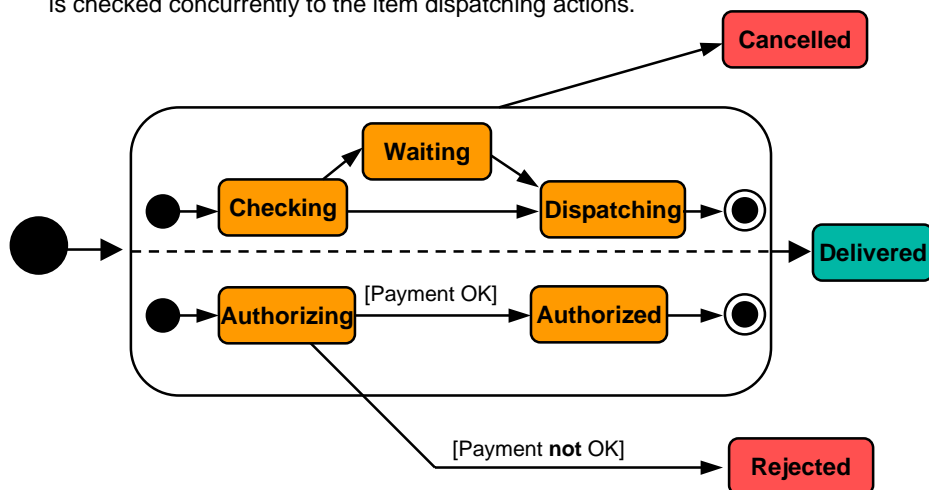
Concurrent state diagrams are useful when a given object has sets of **independent behaviors**.

The concurrent sections of a state diagram are places in which, at any time, the given object is in a composite state defined by the given subsections.



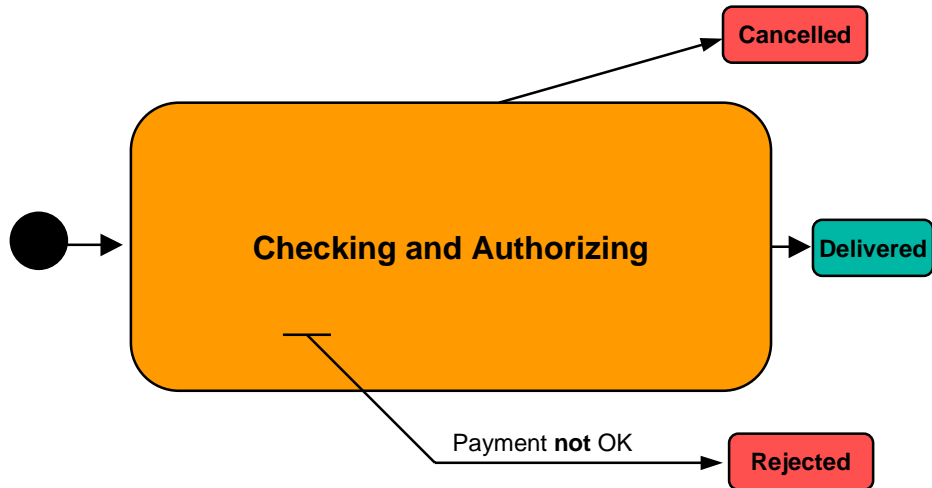
Concurrency: Alternative Notation

Example: The authorization of a customer for a certain purchase is checked concurrently to the item dispatching actions.



Concurrency and Superstate Abstraction

Abstraction of previous example at a higher level.



OOA&D © J.W. Schmidt, F. Matthes, TU Hamburg-Harburg

3.5.19

When to Use State Diagrams

- State diagrams are good at describing the **behavior** of an object **across several use cases**.
- Draw state diagrams especially for classes, which are not well understood and which need detailed description.

- If you have to describe **several objects**, which are involved in a **single use case**, use **interaction diagrams**.
- To show the general sequence for **multiple use cases** and **multiple objects**, use **activity diagrams**.
- State diagrams are not very good at describing behavior that involves a number of objects collaborating together.

OOA&D © J.W. Schmidt, F. Matthes, TU Hamburg-Harburg

3.5.20