

3.4 Interaction Diagrams

Subject/Topic/Focus:

- Dynamic aspects of objects

Summary:

- Sequence Diagrams
- Concurrent Processes
- Collaboration Diagrams

Literature:

- Fowler

Interaction Diagrams

Interaction diagrams describe **exemplary** how groups of objects **collaborate** in some **behavior**.

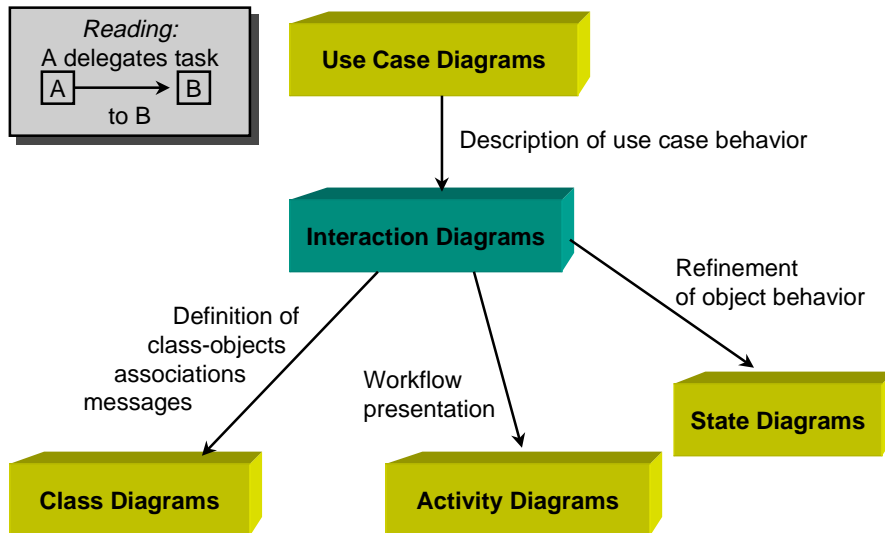
An interaction diagram typically captures the behavior of a **single use case**.

Interaction diagrams do **not** capture the **complete** behavior, **only** typical **scenarios**.

There are two types of **interaction** diagrams:

- **Sequence** diagrams emphasize the **order** or **concurrency** of the interactions.
- **Collaboration** diagrams emphasize the interacting **objects**.

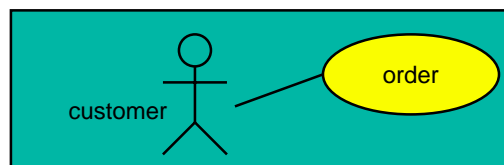
Role of Interaction Diagrams in UML



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

3.4.3

Interaction Diagrams and Use Cases



Behavior of the "order" use case:

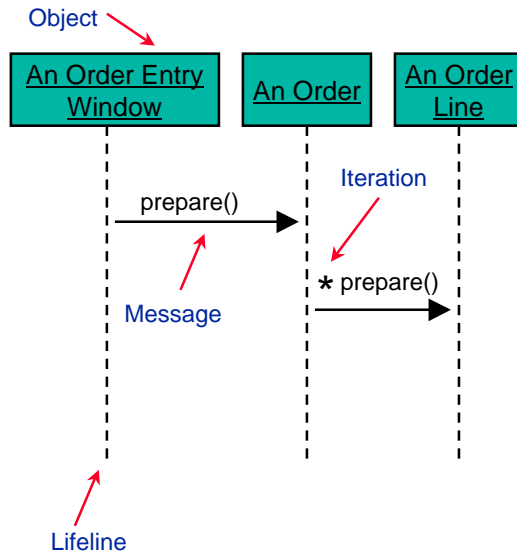
- A customer orders several products.
- The (sub-)orders ("order lines") for each product are prepared separately.
- For each product check the stock.
 - If the product is in stock, remove requested amount from stock.
 - If the product stock falls below a predefined level, reorder it.

Note: The class diagram is prepared in parallel or before the interaction diagram!

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

3.4.4

Sequence Diagrams (1)

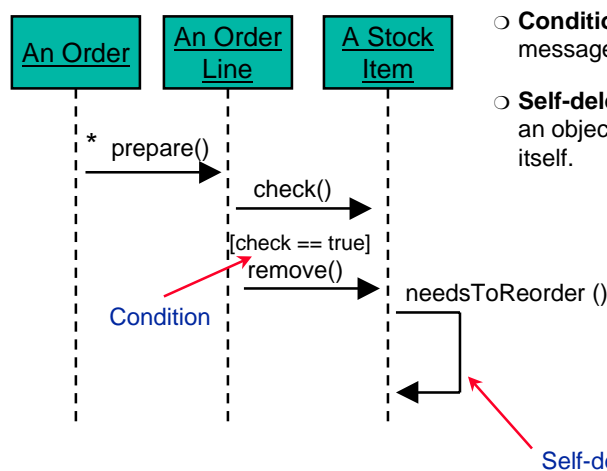


- The **objects** and **messages** are taken from the class diagram.
- The **lifeline** represents the object's life during the interaction.
- **Messages** are represented by arrows between lifelines, labeled at minimum with the message name.
- To send a message between objects, there has to be an **association** between the classes in the class diagram.
- The **sequence** in which messages occur is shown top to bottom.
- The **iteration** marker indicates that a message is sent many times to multiple receivers.

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

3.4.5

Sequence Diagrams (2)

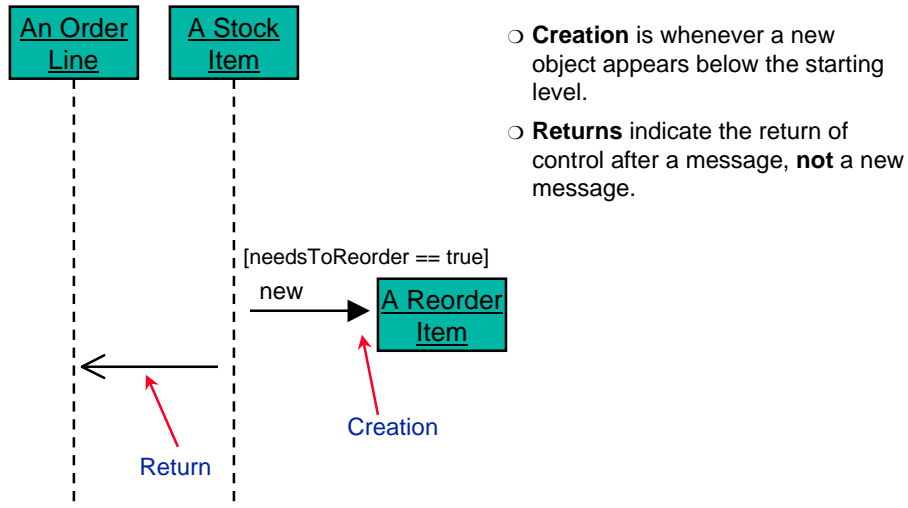


- **Conditions** report whether a message is sent or not.
- **Self-delegation** is used whenever an object sends a message to itself.

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

3.4.6

Sequence Diagrams (3)

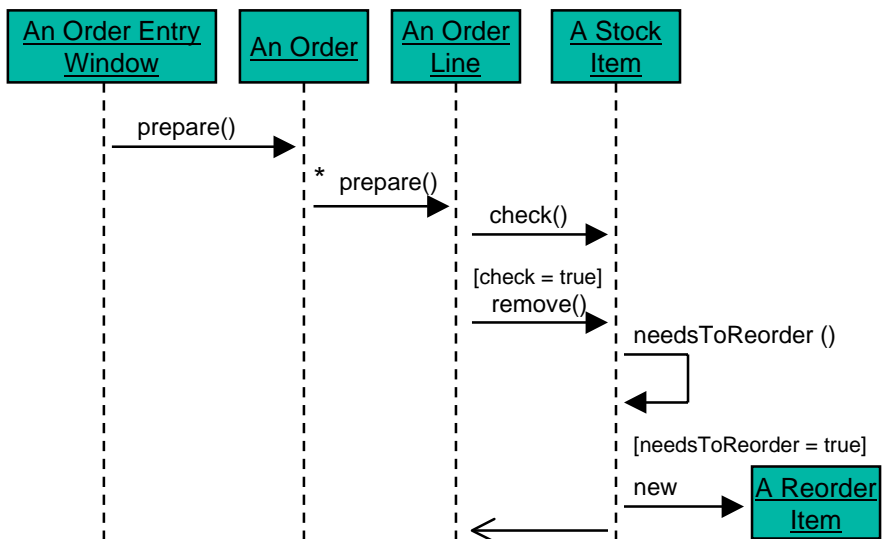


- **Creation** is whenever a new object appears below the starting level.
- **Returns** indicate the return of control after a message, **not** a new message.

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

3.4.7

Sequence Diagrams (4)



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

3.4.8

Concurrent Processes and Activations

UML offers diagram elements to show concurrent processes explicitly within sequence and collaboration diagrams.

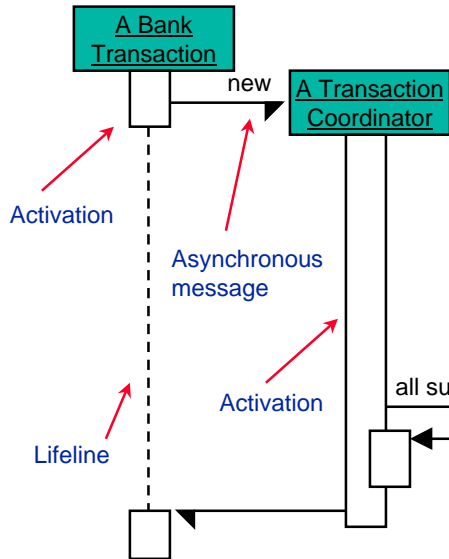
- Activation:
 - indicates whenever a method is active,
 - shows methods which are waiting for returns.
- Asynchronous message:
 - allows the creation of new threads or objects,
 - communicates with already running threads or objects.
- Deletion:
 - occurs by self destruction of threads or objects,
 - can be initiated by external messages.

Concurrency in Sequence Diagrams ⁽¹⁾

Example:

- A bank transaction transfers \$100 from bank account #123 to #456.
 - Withdraw \$100 from bank account #123.
 - Deposit \$100 at bank account #456.
- Withdraw and deposit may be executed in parallel (concurrently).
- The transaction is only successful if both sub-transactions (withdraw and deposit) are successful.
- Therefore a transaction coordinator checks the successful completion of the concurrent sub-transactions.
 - Each time a sub-transaction completes successfully it checks if all other sub-transactions have been completed successfully. If this is the case, it reports success to its client.
 - If any sub-transaction fails, it cancels the other sub-transactions and reports failure to its client.

Concurrency in Sequence Diagrams (2)

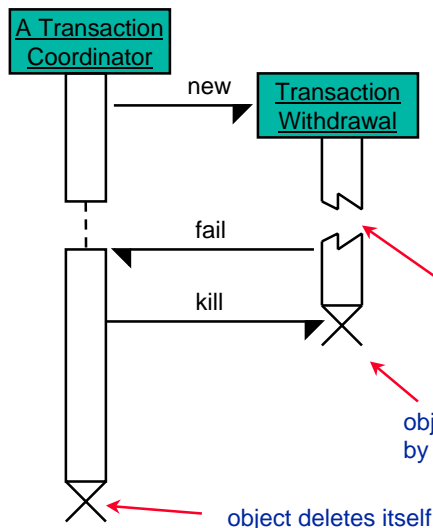


- An **activation** is a method that is either executing or waiting for a return (active method).
- Activations are also useful for describing **self-delegation**.
- An **asynchronous message** is a message that does not block the caller, i.e., caller and receiver execute concurrently.

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

3.4.11

Concurrency in Sequence Diagrams (3)



- Unnecessary **details** may be **suppressed**. Remember that sequence diagrams are only scenarios and not a complete description of the behavior.
- Objects may be **deleted explicitly** by other objects or by themselves (self-destruction).

other processing suppressed

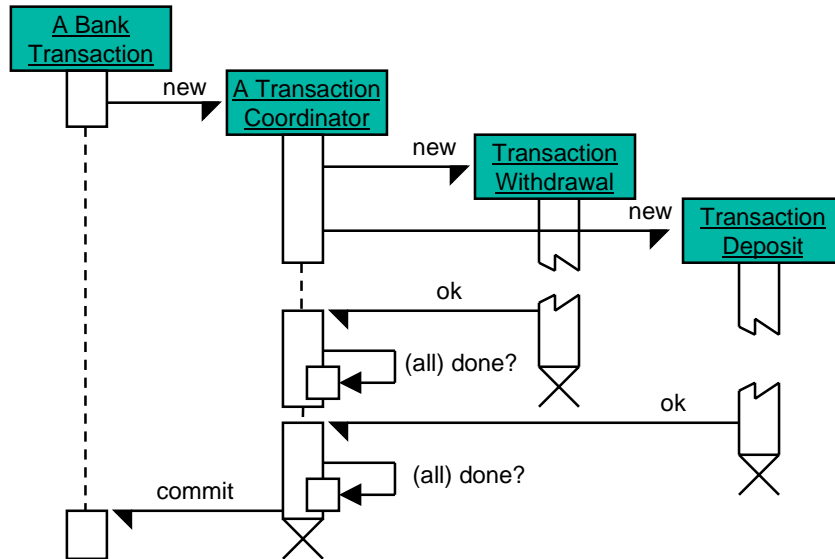
object is deleted by another object

object deletes itself

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

3.4.12

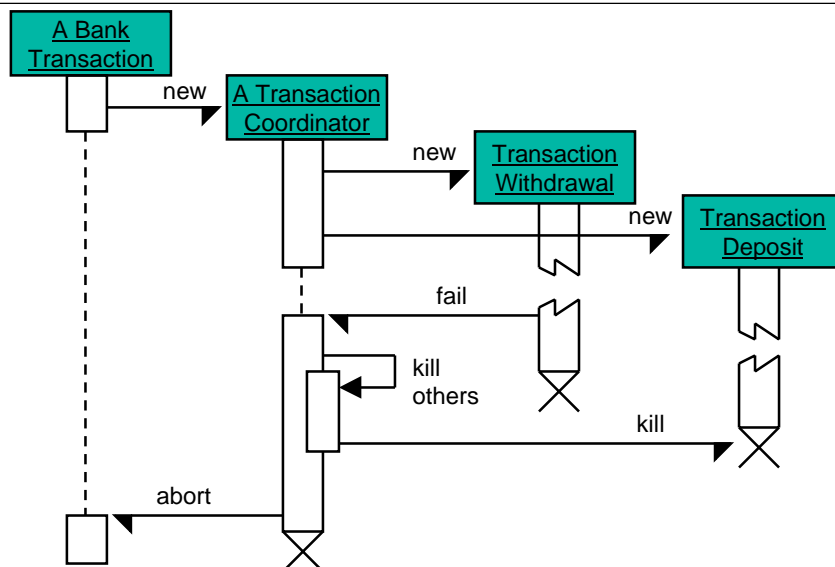
A Successful Bank Transaction



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

3.4.13

A Failing Bank Transaction



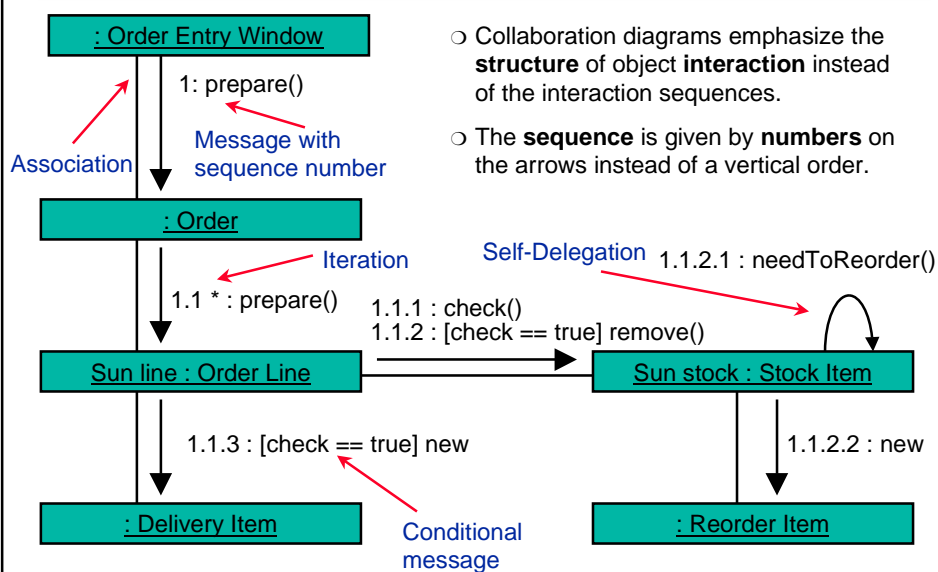
OOA&D © J.W. Schmidt, F. Matthes, TU Hamburg-Harburg

3.4.14

Collaboration Diagrams

- Dynamic behavior of objects can, in addition to sequence diagrams, also be represented by **collaboration** diagrams.
- The transformation from a sequence diagram into a collaboration diagram is a bidirectional function.
- The difference between sequence diagrams and collaboration diagrams is that collaboration diagrams emphasize more the structure than the sequence of interactions.
- Within sequence diagrams the order of interactions is established by vertical positioning whereas in collaboration diagrams the sequence is given by numbering the interactions.

Collaboration Diagrams



When to Use Interaction Diagrams

Use Interaction Diagrams

- When catching user requirements:
 - Describe the behavior of several objects within a single use case.
 - Show collaborations among objects.
- After having described the object behavior completely with state and activity diagrams:
 - Test the state and activity diagrams against the scenarios.

Do not Use Interaction Diagrams

- For precise definition of a single class behavior (use state diagrams).
- If you want to describe the behavior across many use cases or many threads (consider an activity diagram).