Dr. Michael Eichberg
Software Engineering
Department of Computer Science
Technische Universität Darmstadt

Introduction to Software Engineering

Modeling Dynamic Behavior

The following slides use material from:
Craig Larman; Applying UML and Patterns, 3rd Edition; Prentice Hall
Two types of diagrams can be distinguished:

- UML Sequence Diagrams
- UML Communication Diagrams
Interaction diagrams are used to **visualize the interaction via messages between objects**; they are used for **dynamic object modeling**.
Modeling the **dynamic** behavior is often more rewarding w.r.t. understanding the domain than modeling the **static** structure.
Four types of interaction diagrams are available.

- **Sequence diagrams** (which use a fence format.)
- Communication diagrams (which use a graph or network format)
- Timing diagrams (not discussed)
- Interaction overview diagrams (not further discussed)
Four types of interaction diagrams are available.

- Sequence diagrams (which use a fence format.)
- Communication diagrams (which use a graph or network format)
- Timing diagrams (not further discussed)
- Interaction overview diagrams (not further discussed)
Java Code for Interaction Diagrams

```java
public class A {
    private B myB = ...;

    public void do1() {
        myB.do2();
        myB.do3();
    }
}
```
Java Code for Interaction Diagrams

public class A {
    private B myB = ...;

    public void do1() {
        myB.do2();
        myB.do3();
    }
}

Example Communication Diagram
Java Code for Interaction Diagrams

```java
public class A {
    private B myB = ...

    public void do1() {
        myB.do2();
        myB.do3();
    }
}
```

Example
Common Notations for UML Interaction Diagrams

Lifeline box representing an unnamed instance of class Sale.
Common Notations for UML Interaction Diagrams

Lifeline box representing a named instance (s1) of Sale.

Java Code:
Sale s1 = ...;
Common Notations for UML Interaction Diagrams

Lifeline box representing the class Font, or more precisely, that Font is an instance of class Class - an instance of a metaclass.

Java Code:
```java
Class<Font> fontClass = Font.class;
```
Common Notations for UML Interaction Diagrams

Java Code:
```
ArrayList<Sale> sales = ...;
```

Lifeline box representing an instance of an ArrayList class, parameterized to hold Sale objects.
Common Notations for UML Interaction Diagrams

Lifeline box representing one instance of class Sale, selected from the sales ArrayList<Sale> collection.

Java Code:
```java
ArrayList<Sale> sales = ...;
Sale sale = sales.get(i);
```

sales[i]:Sale
Common Notations for UML Interaction Diagrams

:sale

`s1:Sale`

«metaclass»

Font

sales:ArrayList<Sale>

sales[i]:Sale

Overview
Common Notations for UML Interaction Diagrams - Format for Interaction Messages

“Commonly” Used Grammar:

\[
\text{return} = \text{message}(\text{parameter}:\text{parameterType}):\text{returnType}
\]

Parentheses are usually excluded if there are no parameters. Type information may be excluded if unimportant.

```
initialize(code)
initialize
d = getProductDescription (id)
d = getProductDescription (id : ItemId)
d = getProductDescription (id : ItemId) : ProductDescription
```

The same syntax is used by Scala.
UML
Sequence Diagrams
Modeling (Synchronous) Messages

- A found message whose sender will not be specified

- Execution specification bar indicates focus of control (optional)
  dt. Ausführungssequenz

- Typical synchronous message shown with a filled-arrow line
Modeling (Synchronous) Messages

If the Message represents a CallAction, there will normally be a reply message from the called Lifeline back to the calling lifeline before the calling lifeline will proceed.

UML Superstructure

a found message whose sender will not be specified.

execution specification bar indicates focus of control (optional).

dt. Ausführungssequenz
Self messages can be modeled using nested execution specification bars.
To show the return value of a message you can either **use the message syntax (A)** or **use a message line at the end of an execution specification bar (B)**.

**Variant A**

- :Register
- theReport = report
- :Sale

**Variant B**

- :Register
- report
- :Sale
- theReport
Object Instance Creation

The name `create` is an **UML idiom**; it is not required.
Object Instance Destruction

The object destruction notation is also used to mark objects that are no longer usable.
Invoking Static Methods (Class Methods)

Beware, other notations are also used (e.g. underlined method names).
public class Register {
    public void report() {
        Locale[] locales = Calendar.getAvailableLocales();
    }
}
Diagram frames in UML sequence diagrams are used to support - among others - conditional and looping constructs. Frames have an operator and a guard.
How to model the iteration over a collection?

Modeling task: Calculate the total of a sale by summing up the sub totals for each sales line item.
Use a **UML loop frame** to iterate over a collection.

**Modeling task:** Calculate the total of a sale by summing up the sub totals for each sales line item.
Use a **UML loop frame** to iterate over a collection.

**Modeling task:** Calculate the total of a sale by summing up the sub totals for each sales line item.

```
:Sale

[for each sales line item]

lineItems[i] : SalesLineItem

st = getSubTotal

t = getTotal
```

Diagram:
Java code corresponding to a UML loop frame.

```java
public class Sale {

    private List<SalesLineItem> lineItems = new ArrayList<SalesLineItem>();

    public Money getTotal() {
        Money t = new Money();
        Money st = null;
        for (SalesLineItem lineItem : lineItems) {
            st = lineItem.getSubtotal();
            t.add(st);
        }
        return t;
    }

}
```

Modeling task: Calculate the total of a sale by summing up the sub totals for each sales line item.
How to model the sending of a message only if a guard condition matches?

Modeling task: Get the sum of all sales that happened today after 18:00 o’clock.
Use a **UML opt frame** to model the sending of a message if the guard condition matches.

Modeling task: Get the sum of all sales that happened today after 18:00 o’clock.
How to model mutually exclusive alternatives?

Modeling task: A register should be able to handle credit card payments and cash payments.
Use the **UML alt frame** to model between 2 and n mutually exclusive alternatives.

Modeling task: A register should be able to handle credit card payments and cash payments.

```
:Register

makePayment(type,sale) →

alt
[type = CreditCardPayment]
create :CreditCardPayment

[type = CashPayment]
create :CashPayment
```
Diagram frames in UML sequence diagrams are used to support - among others - conditional and looping constructs. Frames have an operator and a guard.

In general, sequence diagrams are not really well suited to show looping and conditional behavior. Activity diagrams and code may be alternatives.
An interaction occurrence (interaction use) is a reference to an interaction within another interaction.

References are used to simplify a diagram and factor out a portion into another diagram or to enable reuse.

Modeling task: We want to calculate the store’s overall total.
An interaction occurrence (interaction use) is a reference to an interaction within another interaction.

Given:

\[ r = \text{getTotal}(\text{startDate}) \]

\[ sales[i] : \text{Sale} \]

\[ t = \text{getTotal} \]

\[ \text{date} = \text{getDate} \]

\[ \text{loop} \quad [\text{for each sale}] \]

\[ \text{opt} \quad [\text{startDate} < \text{date}] \]
An interaction occurrence (interaction use) is a reference to an interaction within another interaction.

```
given: Register
r = getTotal(startDate)
t = getTotal(sales[i] : Sale)

for each sale
   date = getDate
   opt [startDate < date]
   t = getTotal
```

sd = sequence diagram
sd:CalculatePerRegisterTotal
An interaction occurrence (interaction use) is a reference to an interaction within another interaction.

```
\text{CalculatePerRegisterTotal}
```

```
\text{Register} \\
\text{sales[i]} : \text{Sale} \\
```

```
\text{r = getTotal(startDate)} \\
\text{t = getTotal} \\
\text{sales[i]} : \text{Sale} \\
```

```
\text{for each sale} \\
\text{opt} \quad [\text{startDate < date}] \\
\text{date = getDate} \\
\text{t = getTotal} \\
\text{loop} \\
```

```
\text{sd} \\
```

```
```

UML Sequence Diagrams | 39
An interaction occurrence (interaction use) is a reference to an interaction within another interaction.
How to model the sending of asynchronous messages?
How to model objects that have their own thread of execution?

Modeling task: The log information should automatically be collected and processed in the background.
Asynchronous messages are messages that don’t block.

An active object is an object where each instance runs on and controls its own thread of execution.

Asynchronous messages are shown using stick arrows.

Active objects are modeled using double vertical lines on the left and right side of the lifeline boxes.

Modeling task: The log information should automatically be collected and processed in the background.
UML Communication Diagrams
• A **link** is a connection path between two objects (it is an instance of an association)
  A link indicates that some form of navigation and visibility between the objects is possible.

• Each **message** between objects is represented with a message expression and a small arrow indicating the direction of the message
  Sequence numbers are added to show the sequential order of messages in the current thread of control; the starting message is often not numbered.

```
makePayment ↓

1: makePayment(cashTendered) →
2: getTaxes →
2.1. getDate ←
```

:Register → :Sale
• Modeling self messages
Alternative Notations for Modeling Instance Creation

Create message, with optional initializing parameters. This will normally be interpreted as a constructor call.

If an unobvious creation message name is used, the message may be stereotyped for clarity.
Message Number Sequencing

The initial message is not numbered to make the numbering easier to comprehend.
Modeling Conditional Messages

The message is only sent if the condition evaluates to true. The condition is written in square brackets. In case of modeling mutually exclusive message conditional path letters are prepended.

Small letters are sometimes used to mark methods that are executed in parallel.
Messages to Class Objects

makePayment ↓

:Sale

1: locs = getAvailableLocales →

«metaclass» Calendar

e.g. to format the date as either 3/2/2009 or 2.3.2009
UML Communication vs. UML Sequence Diagrams
# Strengths and Weaknesses Interaction Diagrams

<table>
<thead>
<tr>
<th>Type</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sequence Diagram</strong></td>
<td>✓ clearly shows sequence or time ordering of messages &lt;br&gt; ✓ large set of detailed notation options</td>
<td>– forced to extend to the right when adding new objects; consumes horizontal space</td>
</tr>
<tr>
<td><strong>Communication Diagram</strong></td>
<td>✓ space economical - flexibility to add new objects in two dimensions</td>
<td>– more difficult to see sequence of messages  &lt;br&gt; – fewer notational options</td>
</tr>
</tbody>
</table>
## Strengths and Weaknesses Interaction Diagrams

<table>
<thead>
<tr>
<th>Type</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Communication Diagram</td>
<td>✓ clearly shows sequence or time</td>
<td>– forced to extend to the right when adding new objects; consumes horizontal space</td>
</tr>
<tr>
<td></td>
<td>✓ space economical - flexibility to add new objects in two dimensions</td>
<td>– more difficult to see sequence of messages; – fewer notational options</td>
</tr>
</tbody>
</table>

UML tools often emphasize sequence diagrams, because of their greater notational power.
The goal of this lecture is to enable you to systematically carry out small(er) software projects that produce quality software.

- Modeling the dynamic behavior is often more rewarding than modeling the static structure w.r.t. understanding a domain
- Modeling the dynamic behavior is often particularly useful if the control-flow is more involved; but only draw the part that is relevant to understand the problem at hand
- The UML is often used informally - this is OK if everyone interprets the diagrams in the same way
The goal of this lecture is to enable you to systematically carry out small(er) commercial or open-source projects.