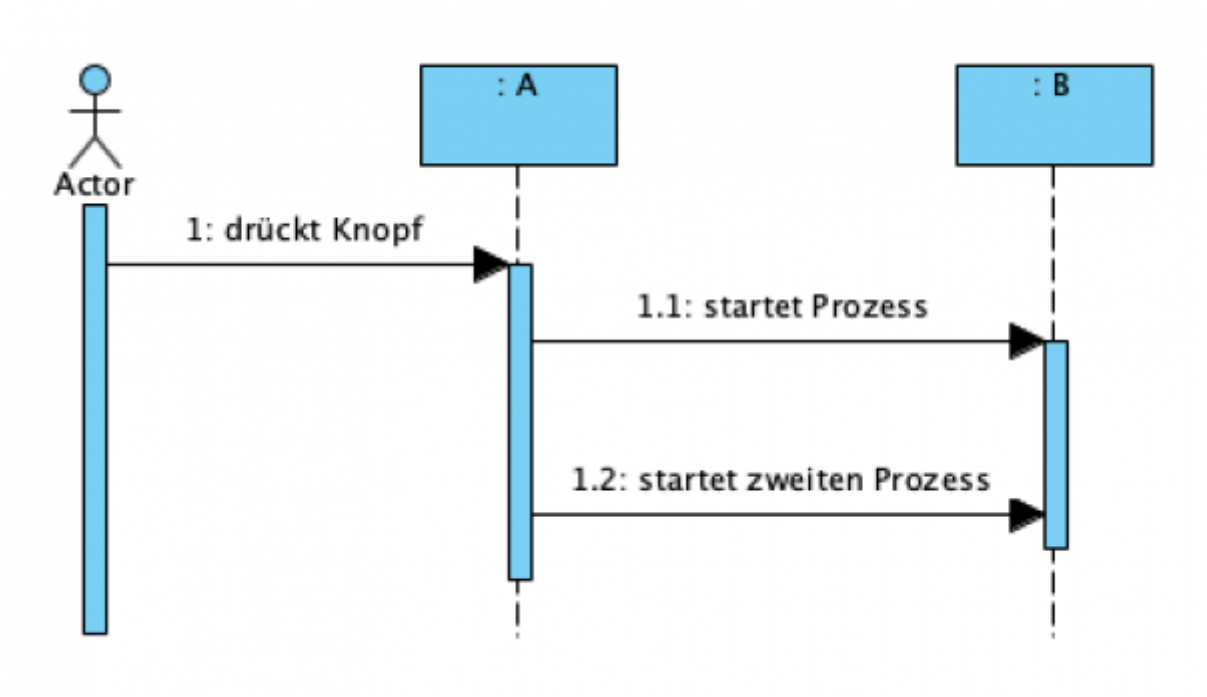
UML Interaktionsdiagramme

Es gibt zwei Arten von UML-Diagrammen:

- Das UML-Sequenzdiagramm und
- Das UML-Kommunikationsdiagramm

UML-Interaktionsdiagramme werden für die dynamische Objektmodellierung verwendet, um die Interaktion über Meldungen zwischen Objekten zu visualisieren. Darum ist das Modellieren des dynamischen Verhaltens ist in Bezug auf das Verstehen der Domäne oft lohnender als das Modellieren der statischen Struktur.

Es gibt vier Arten von Interaktionsdiagrammen, das Sequenzdiagramm, welches ein „Zaunformat“ hat, das Kommunikationsdiagramm, welches in Zusammenahng mit einer Grafik oder einem Netzwerk verwendet wird. Dann gibt es das Zeitdiagramm und das Interaktionsübersicht-Diagramm, auf beide werde ich hier nicht weiter eingehen.

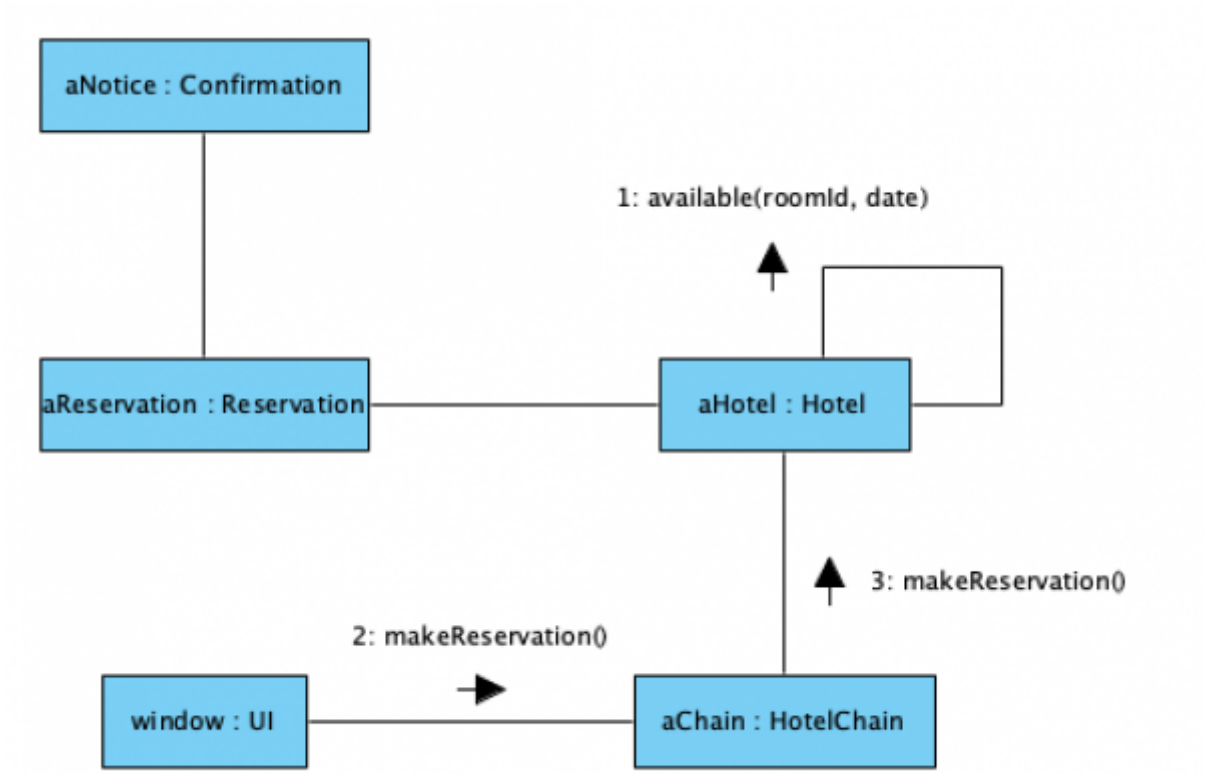


Schauen wir uns nun das Sequenzdiagramm von oben genauer an und versuchen daraus einen (Java-) Code zu generieren. Wir wissen, dass der Akteur einen Knopf zum starten einer Aktion gedrückt hat.

```
public class A {
private B testB = ...;

JToggleButton tb = new JToggleButton("drueck mich");
tb.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        JToggleButton btn =  (JToggleButton) e.getSource();
        testB.starteProzess1();
        testB.starteProzess2();
```

```
    }
});
```



Analog geht dies natürlich auch mit dem Kommunikationsdiagramm einer Hotelreservierung:

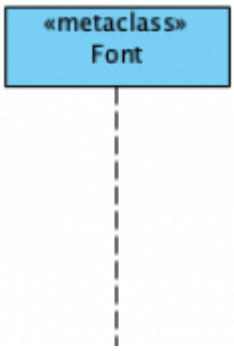
```
public class UI {
private HotelChain aChain = ...;
public void makeReservation() {
 aHotel.makeReservation();
 }
}
```

Dies ist dank der Standardisierung der UML Sprachen möglich. Darum möchte ich hier häufig verwendete Notationen in UML-Interaktionsdiagrammen ansprechen.

Die linke UML-Notation zeigt eine Box mit einer Lebenslinie oder auch Lifeline-Box. Die Box steht dabei für eine Klassen, in diesem Beispiel für eine nicht benannten Instanz der Klasse Foo. Unter der Box sehen Sie eine gestrichelte Lebenslinie. Da diese Klasse noch nicht aufgerufen und somit nicht instanziiert wurde, hat diese Klasse bis jetzt noch nicht gelebt.

Die rechte stehende UML-Notation zeigt wie oben eine Box mit einer Lebenslinie. Auch hier steht die Box für die Klasse Foo, nur dass hier die Klasse instanziiert wurde. Hier ist f1 eine Instanz der klasse foo. In Javacode wäre die rechte Notation äquivalent mit folgender Codezeile:

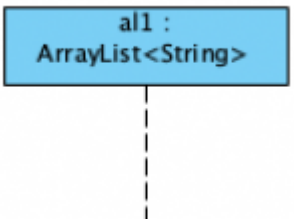
```
Sale s1 = …;
```



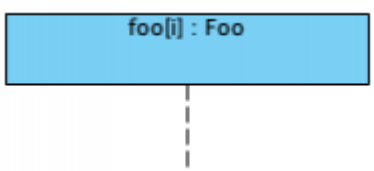
Dann gibt es noch Metaklassen, diese werden wie die oben stehenden Beispiele mit einer Box und einer Lebenslinie angegeben, jedoch ist dies dann eine Instanz einer Metaklasse. In meinem Beispiel ist das die Klasse Font, oder genauer gesagt, Font ist eine Instanz der Klasse Class und somit eine Instanz dieser Metaklasse. Leichter ist dies anhand folgendem äquivalenten (Java-) Code zu verstehen:

```
Class<Font> fontClass = Font.class;
```

Mit der rechts stehenden Notation können aber auch Objekte einer Klasse dargestellt werden, so wie beispielsweise hier rechts. Dabei stellt die Box mit der zugehörigen Lebenslinie eine Instanz al1 einer ArrayList-Klasse dar, die für das Speichern von String-Objekten parametrisiert ist. Auch hier wieder ein (Java-) Code-Fragment zur leichteren Verständlichkeit:




```
ArrayList<String> al1 = …;
```

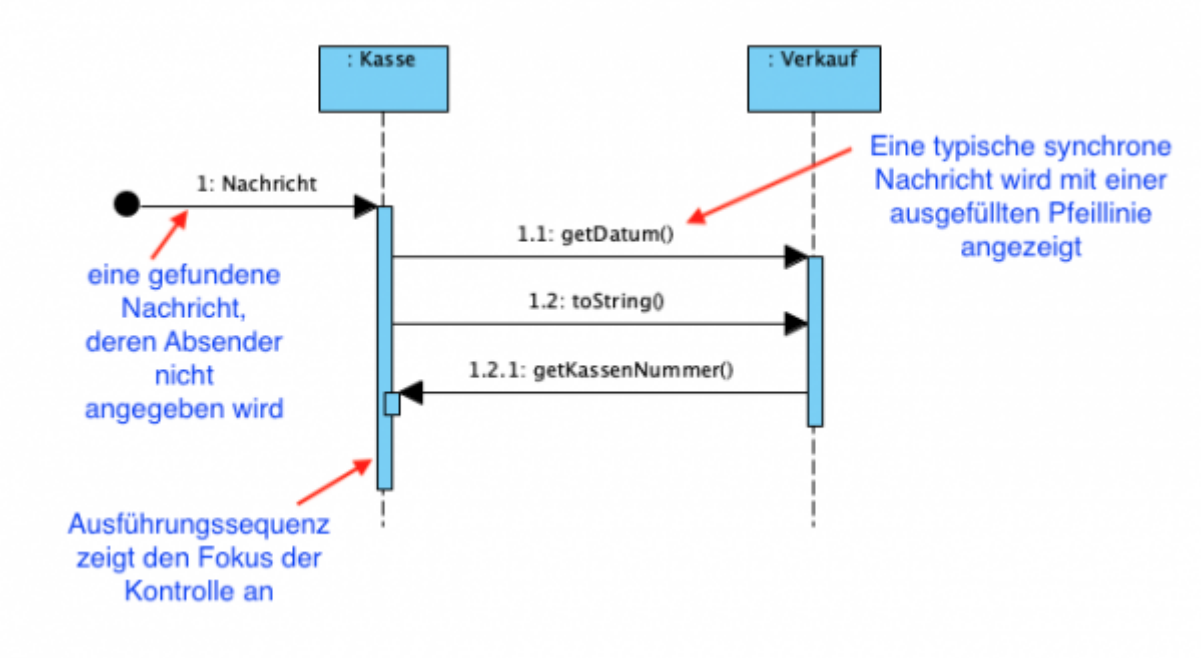


Links wird eine Instanz der Klasse Foo darstellt, bei der das i-te Objekt aus der Sammlung der Liste foo ausgewählt ist. Auch hier für das leichtere Verständnis ein (Java-) Code-Fragment:

```
ArrayList<Foo> foo = …;
Foo foo = foo.get(i);
```

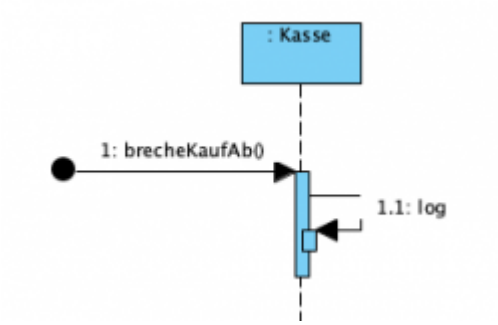
- return = Nachricht (Parameter: parameterType): returnType
- Klammern werden normalerweise nicht verwendet, wenn keine Parameter vorhanden sind.
- von Typeninformationen kann abgesehen werden, wenn diese unwichtig sind.

Modellieren synchroner Nachrichten



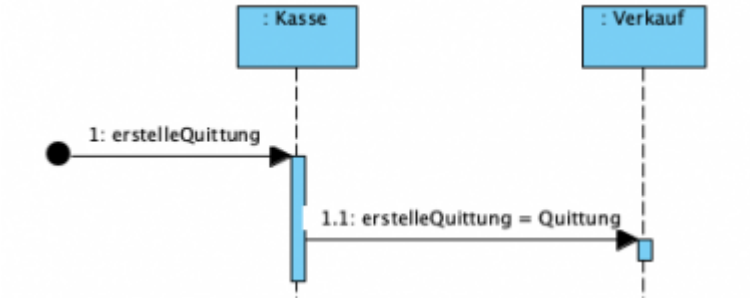
UML-Überstruktur oder UML-Superstructure

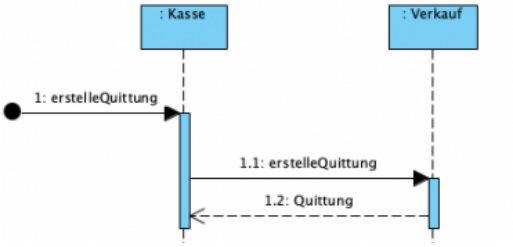
Wenn die Nachricht eine Aufruferaktion angegeben ist, gibt es normalerweise auch eine Antwortnachricht von der angerufenen Instanz, welche zurück gegeben wird, bevor die aufrufende Instanz fortgesetzt wird.



Eigene Aufrufe können wie links dargestellt auch als verschachtelte Ausführungsspezifikationsleisten (execution specification bars) dargestellt werden. In diesem Beispiel wird der Kauf an der Kasse abgebrochen. jedoch soll der Vorgang nicht verworfen, sondern protokoliert werden. Dazu findet dann ein Selbstaufruf statt, der in der Kasse in das Log-File schreiben soll, dass dieser Kauf abgebrochen wurde.

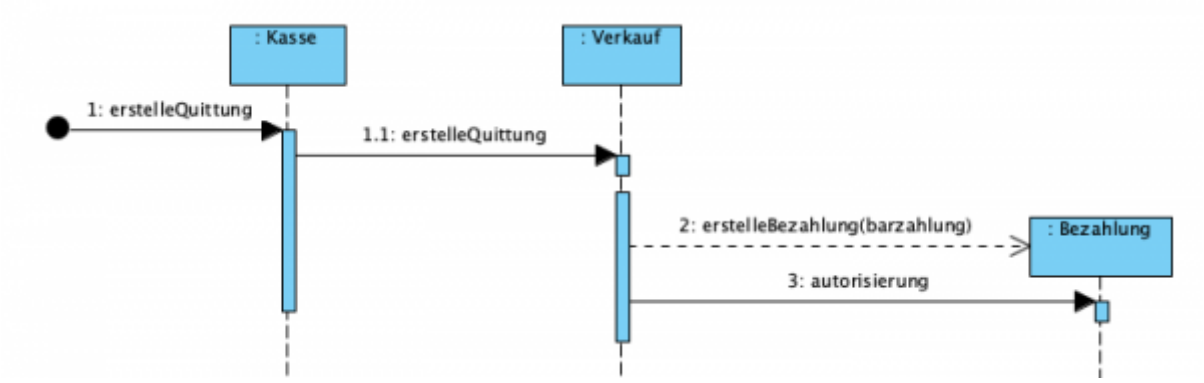
Rückgabebehandlung in UML





Um den Rückgabewert einer Nachricht anzuzeigen, können Sie entweder die Nachrichtensyntax, wie in der Darstellung links oder eine Nachrichtenzeile am Ende einer Ausführungs- spezifikationsleiste, wie in der Darstellung darunter verwenden.

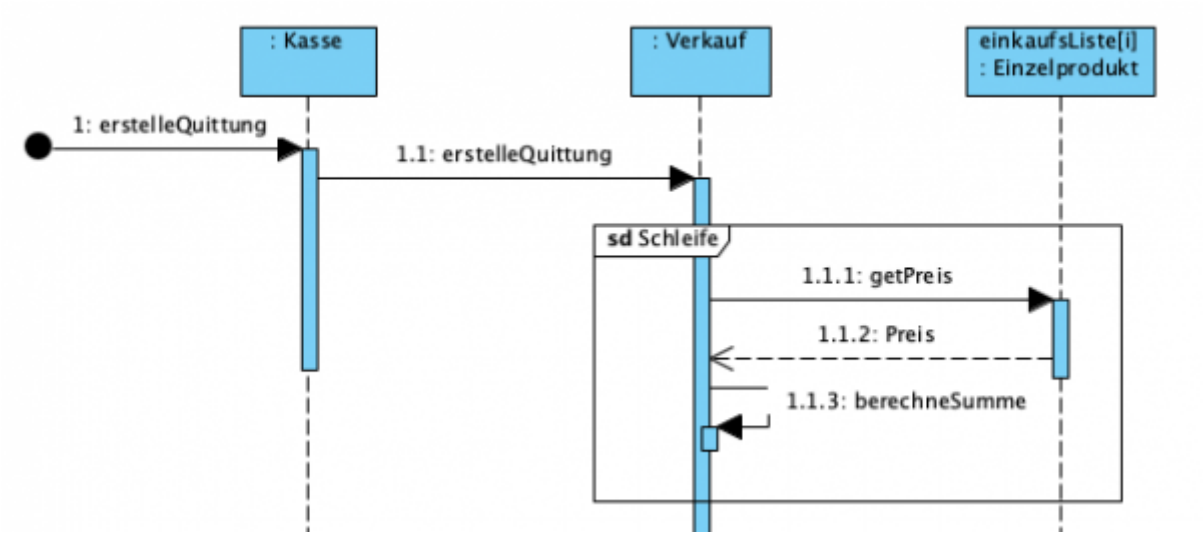
Es können aber auch neue Objekte durch einen Instanzaufruf erstellt werden. So wie Beispielsweise für die Bezahlung in bar, bei der eine „andere“ Instanz der Bezahlungsmodalität zur Autorisierung verwendet werden muss als bei Kartenzahlungen oder Guthabenzahlungen. Hier Ist der Kassierer, für die Korrektheit der Autorisierung zuständig, also ob das Geld des Kunden echt und ausreichend für den Kauf ist.



Hingegen ist eine Zerstörung des Objektes oder der Klasse nicht notwendig, dies übernimmt beispielsweise der Garbage Collector in Java, sobald ein Objekt oder eine Klasse nicht mehr benötigt wird.

Schleifen in UML

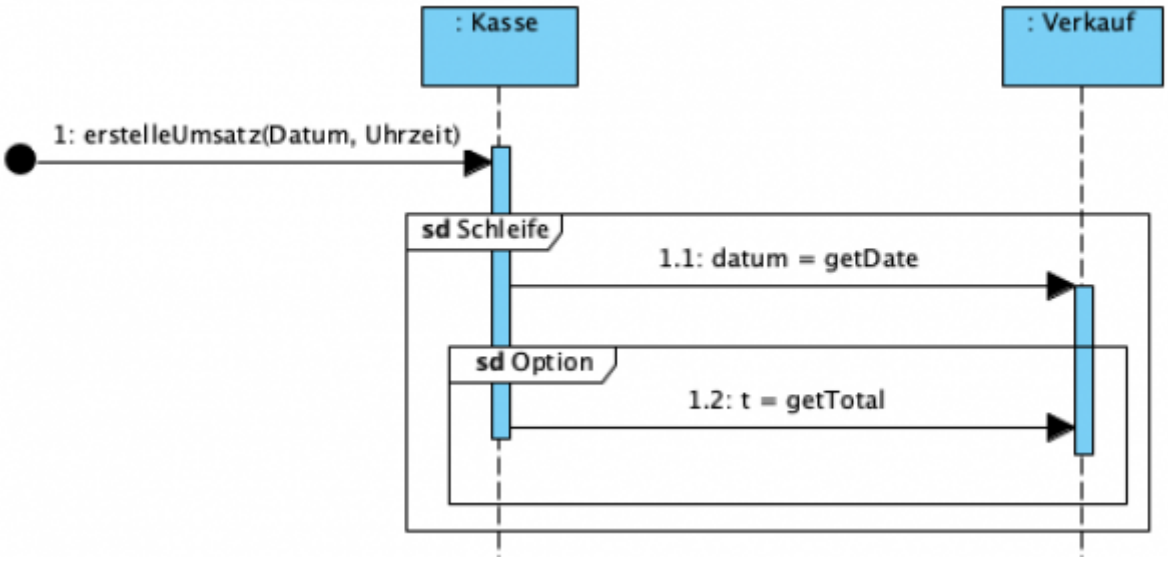
Modellierungsaufgabe: Berechnung der Summe eines Kaufs, indem Sie die Zwischensummen für jede Verkaufsposition aufsummieren.



Hier soll das Programm über alle Produkte des Kunden itterieren und diese dann zusammenrechnen, damit eine Gesamtsumme für die Quittung erstellt werden kann. Bestenfalls schreiben Sie dann noch den Operator in unserem Fall wäre das die Schleife an sich und den Guard, hier einkaufsListe.size. Der Guard beschreibt die Abbruchbedingung für den Operator. Hier der äquivalente Code:

```
public class Verkauf {
private List<Einzelprodukt> einkaufsListe= new ArrayList<>();
public Betrag erstelleQuittung() {
 Betrag t = new Betrag();
 Betrag st = null;
 for (Einzelprodukt Einkauf : einkaufsListe) {
 st = Einkauf.getSubtotal();
 t.add(st);
 }
 return t;
 }
}
```

Erweiterung: Sie möchten die Summe aller Verkäufe, die heute 18:00 Uhr getätigt wurden haben.



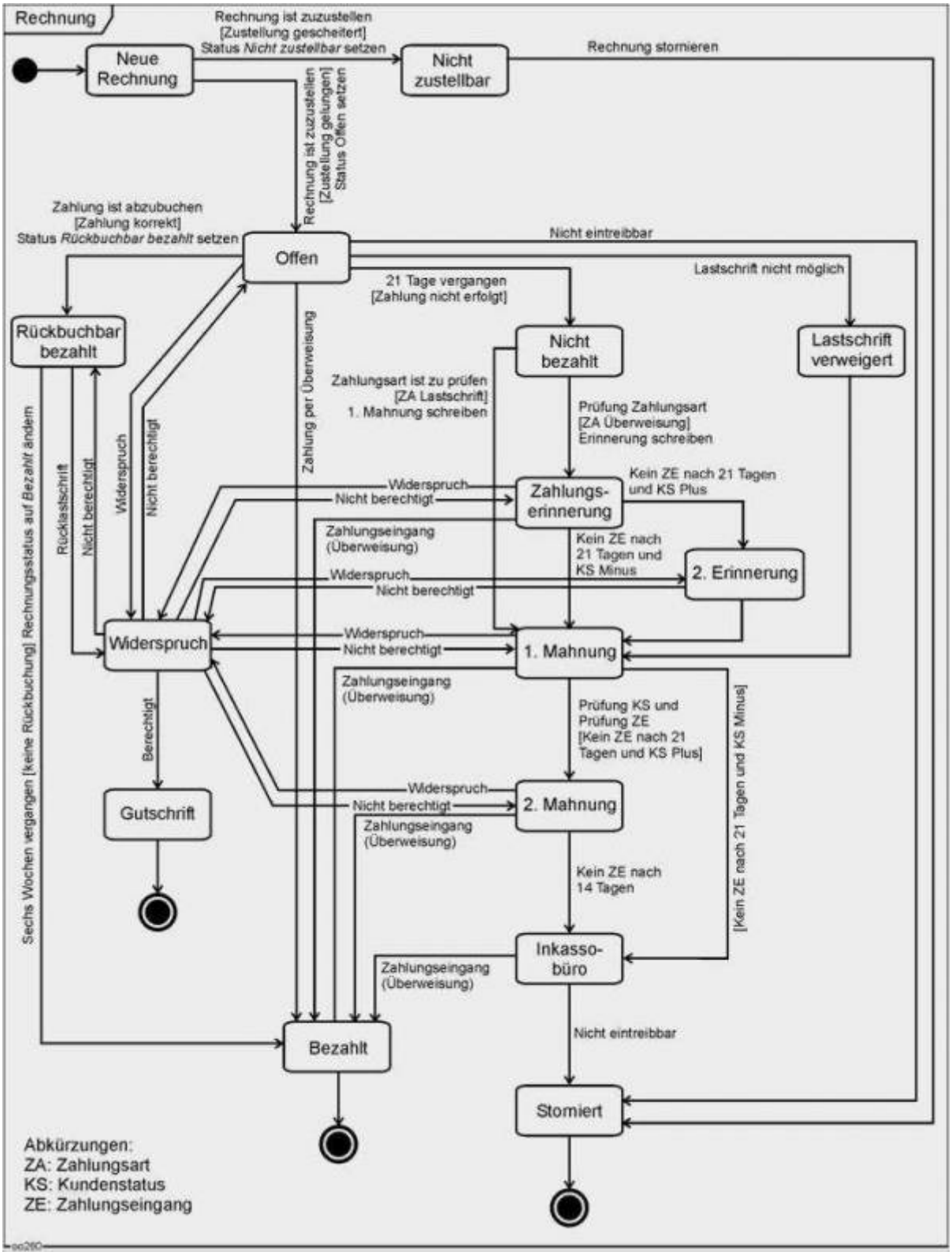
Hierbei wird die Schleife um eine Option erweitert, dabei müssen Sie dann auch diese genau

definieren, z.B. day == currentDay && time ⇐ currentTime, wobei bei unserem Beispiel dann time = 18.00 Uhr wäre.

Sequenzdiagramme sind im Allgemeinen nur gut geeignet, um einfaches Schleifen- und Bedingungsverhalten zu zeigen. Aktivitätsdiagramme und der finale Code können und werden natürlich Abweichungen haben.

Beim Kommunikationsdiagramm ändert sich die Notation geringfügig, diese ist nahezu selbsterklärend, da es weniger Notationsmöglichkeiten gibt.

Kommunikationsdiagramm



Quelle: ccfischer.de

Typ	Vorteile	Schwächen
Sequenzendiagramm	zeigt deutlich den Programmablauf und auch die Reihenfolge der Nachrichten. Bietet eine große Auswahl an detaillierten Notationsoptionen	muss sich beim Hinzufügen neuer Objekte nach rechts erstrecken dadurch verbraucht es horizontal viel Raum

Typ	Vorteile	Schwächen
Kommunikationsdiagramm	Platz sparend und damit flexibel beim Hinzufügen neuer Objekte	schwieriger zu erfassen des Programmablauf und Abfolge der Nachrichten und zudem weniger Notationsmöglichkeiten

Ich bevorzuge Sequenzdiagramme, da diese über eine größere Detailgenauigkeit verfügen und schon Abläufe klar deutlich machen.

- dynamischen Modellierung eines Domänen-Verhaltens ist oft lohnender als das Modellieren der statischen Domänen-Struktur
- Die Modellierung des dynamischen Verhaltens ist besonders nützlich, wenn der Steuerungsfluss / control-flow stärker involviert ist. Man Zeichnet in der Regel nur den relevanten Teil, um ein Problem zu verstehen
- UML wird häufig informell verwendet