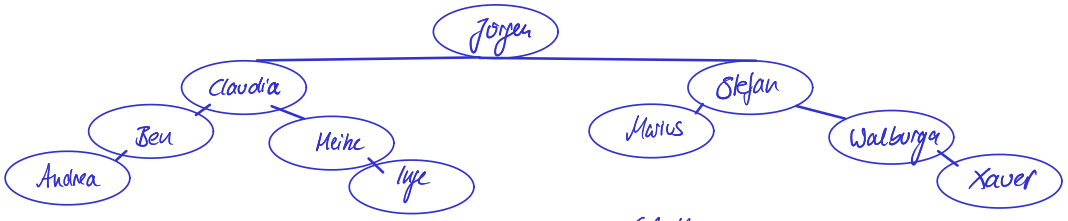
d) Im Gegensatz zur Liste haben „mittlere" Elemente des Baums nicht nur einen, sondern maximal zwei „Nachfolger". Bei der Liste wird das entnommene Element durch den Nachfolger ersetzt. Hier kann es nur durch eines der beiden ersetzt werden. Eine mögliche Strategie wäre es, das entnommene Element durch seinen linken Unterknoten zu ersetzen und alle Elemente des rechten Kindbaums neu in den Baum einzusortieren.



(falls zuerst W. dann X. einsortiert wird)

Aufgabe 3c

Im schlechtesten Fall wird eine Suche in der Liste mit 63 Kontakten

$$\Delta t_L = 63 \cdot 5\,ms = 315\,ms \approx 0{,}32\,s$$

in Anspruch nehmen. Um 63 Kontakte in einem Baum zu verwalten sind nur 6 Ebenen nötig, wenn der Baum balanciert ist. Dann dauert die Suche maximal

$$\Delta t_B = 6 \cdot 5\,ms = 30\,ms = 0{,}030\,s$$

Wenngleich auch eine Wartezeit der Suche von 0,32 s akzeptabel erscheint, ist dieser Wert deutlich oberhalb der Reaktionszeit des Menschen und daher deutlich merkbar.

Musterlösung

Aufgabe 1a

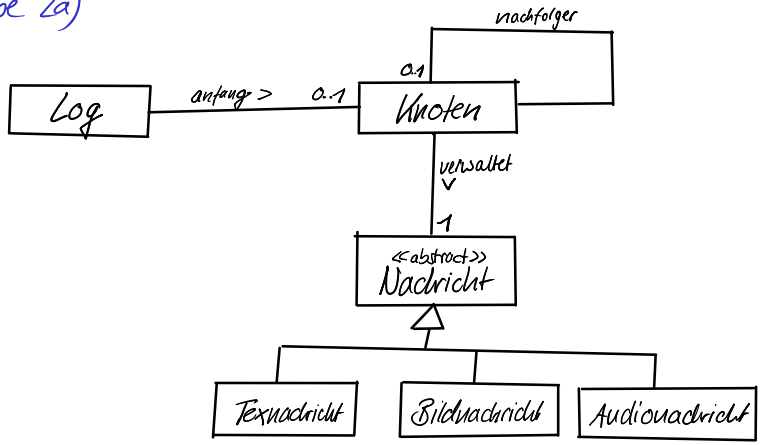
In beiden Fällen ist aufgrund der Dynamik der Daten (neue kommen hinzu, die ältesten fallen weg) eine einfach verkettete Liste die geeignetere Struktur, da sie mit deutlich einfacheren Algorithmen zum Hinzufügen auskommt.

Im Fall (ii) kommt hinzu, dass die Anzahl der Nachrichten stark variiert und nicht begrenzt ist. Ergo müsste dort entweder die Größenanpassung des Felds algorithmisch gelöst werden oder ein derart großes Feld vorinstanziert sein, dass viel Speicherplatz sinnlos blockiert wird.

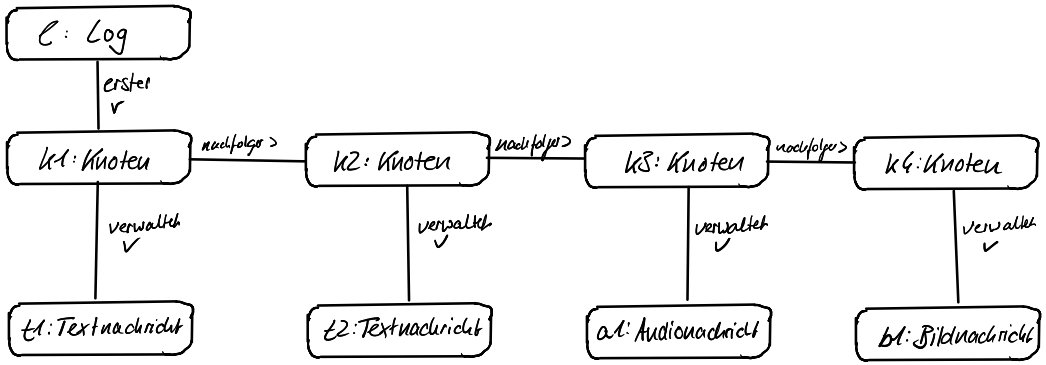
Aufgabe 1b

Es handelt sich um eine Schlange, d.h. neue Elemente werden ausschließlich an einem Ende hinzugefügt, alte am anderen Ende entnommen.

Aufgabe 2a)



Aufgabe 2b



Aufgabe 2c

In Klasse Log:

```
einfuegen (n : Nachricht)
    falls anfang existiert
        temp = anfang
        anfang = new Knoten(n)
        anfang.nachfolgerSetzen(temp)
    sonst
        anfang = new Knoten(n)
    endefalls
    anfang.restLoeschen(50)
ende Methode
```

In Klasse Knoten

```
Knoten (n: Nachricht)
    verwaltet = n;
endeKonstruktor
nachfolgerSetzen(k: Knoten)
    nachfolger = k
ende Methode
restLoeschen (i : int):
    falls (i <= 1)
        nachfolger = null
    sonst
        falls nachfolger existiert:
            nachfolger.restLoeschen(i-1)
        endefalls
    endefalls
ende Methode.
```

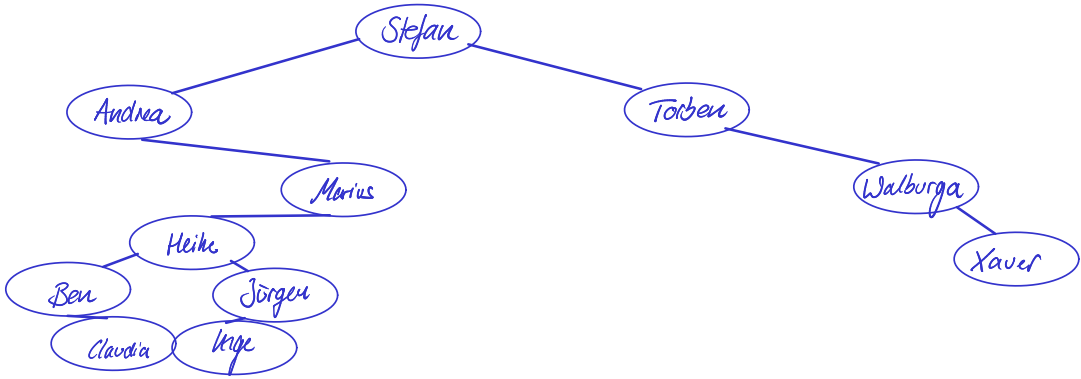
Aufgabe 2d)

Rekursive Methoden enthalten erneute Aufrufe der selben Methode. (Diese ist nicht zwingend beim selben Objekt und i.d.R nur bedingt.) In der obigen Lösung der 2c) enthaelt nur die Methode restLoeschen einen rekursiven Aufruf.

Aufgabe 3a

Es handelt sich um das Entwurfsmuster (Design Pattern) Kompositum (composite). Wird es in der Liste eingesetzt, so sind alle rekursiven Aufrufe „eleganter", da nicht mehr überprüft werden muss, ob überhaupt ein Nachfolgeelement existiert. Objekte der Klasse Abschluss sind hier das Pendant nichtexistenter Nachfolger.

Aufgabe 3b



Aufgabe 2c