

# Funktionale Architektur für BigSpenders

Mike Sperber

Created: 2023-10-20 Fri 12:26



# Funktionale Architektur für BigSpenders

**Mike Sperber**

Active Group

@sperbsen@discuss.systems



# Aufgabenstellung



# Nomen aus dem Aufgabentext

- Rechnung
- Beleg
- Spesen
- Genehmigung
- Rückfrage
- Auszahlung
- Spesenritter
- Projekt
- Postkorb
- Gehaltsabrechnung
- User Directory
- Dokumentenarchiv
- Kunde





# UML-Diagramm

- Begriffe "Vorgang", "Consultant", "Kostenstelle" sind neu



# Beschreibung -> Code

"Spesenritter: Mitarbeiter, der Spesen verursacht hat"

```
data Spesenritter = Spesenritter {  
    spesenritterName :: String  
}
```



# Beschreibung -> Code

“Vorgang: Anlass, dem mehrere Spesenbelege zuzuordnen sind, z.B. eine Reise zum Projekteinsatz, ein Messebesuch oder eine Vertriebsaktion”

```
data Vorgang = Vorgang {  
  vorgangAnlass :: Anlass, -- Vorgang == Anlass?  
  vorgangSpesenritter :: Spesenritter -- UML-Diagramm  
}
```

```
data Anlass  
= Reise {  
  reiseZiel :: String,  
  reiseStart :: Calendar.Day,  
  reiseEnde :: Calendar.Day  
}  
| Messebesuch {  
  messeBesuchMesse :: String,  
  messeBesuchStart :: Calendar.Day,  
  messeBesuchEnde :: Calendar.Day  
}  
| Vertriebsaktion
```



# Belege

”Beleg: Eine Quittung über ausgegebenes Geld, z.B. ein Flugticket, ein Bahnticket, eine Taxiquittung, eine Hotelrechnung, ein Restaurantbeleg”

**data BelegInfo**

- = Flugticket
- | Bahnticket
- | Taxiquittung
- | Hotelrechnung
- | Restaurantbeleg





# Belege

“Vorgang: Anlass, dem mehrere Spesenbelege zuzuordnen sind, z.B. eine Reise zum Projekteinsatz, ein Messebesuch oder eine Vertriebsaktion”

```
data Beleg = Beleg {  
    belegInfo :: BelegInfo,  
    belegVorgang :: Vorgang,  
    belegDatum :: Calendar.Day, -- erfunden  
    belegGeld :: Geld -- erfunden  
}
```



# Belege / UML-Diagramm

```
data Beleg = Beleg {  
    belegInfo :: BelegInfo,  
    belegVorgang :: Vorgang,  
    belegDatum :: Calendar.Day, -- erfunden  
    belegGeld :: Geld, -- erfunden  
    belegKostenstelle :: Kostenstelle -- neu  
}
```



# Rechnung

”Rechnung: Eine Zusammenstellung der Leistungen, die im Rahmen eines Beratungsprojektes erbracht wurden, zusammen mit Angaben über das Honorar und die Spesen.”

```
data Rechnung = Rechnung {  
    rechnungProjekt :: Projekt,  
    rechnungLeistungen :: [Leistung],  
    rechnungHonorar :: Honorar,  
    rechnungSpesen :: Spesen  
}
```

```
-- ???
```

```
data Leistung = Leistung  
data Honorar = Honorar Geld  
newtype Spesen = Spesen Geld
```



# CFO Manfred Schneider über Spesen

Spesen haben:

- Steueranteil
- geldwerter Vorteil
- nicht erstattbare Anteile bei den Verpflegungskosten
- Erstattungsbetrag





# CFO Manfred Schneider, mehr Begriffe

- Währungen
- Grenze
- Abteilung
- "Regeln"



# CFO Manfred Schneider, Interaktionen

- Beleg-Scanner
- Spesenkonto
- "E-Mails"



# "Regeln"

- "Wenn Spesen für Projekt X anfallen, die am Ende der Woche insgesamt kleiner als Y Euro sind, dann genehmige sofort und frage nicht nach."
- "Wenn Spesen entstehen, die mehr als 20% vom Wochendurchschnitt des Projektes Z abweichen, dann löse eine Frage an den Genehmiger aus."



# Noch neu aus den User Stories

- Steuervorschriften
- Fragen
- Rückfragen





# Genehmigung

```
data Genehmigung = Genehmigung ??? GenehmigungsErgebnis
```

```
data GenehmigungsErgebnis  
  = GenehmigungErteilt  
  | GenehmigungZurueckgewiesen  
  | ???
```

"Als Spesenritter möchte ich in meinem Spesenkonto erkennen können, welche meiner **Belege** schon erfasst wurden, welche genehmigt wurden und welche schon ausgezahlt sind."

"Als Genehmiger möchte ich eine Rückfrage zu einem **Vorgang** an den Spesenritter stellen können."



# Genehmigungsregel

```
data Genehmigung = Genehmigung Beleg GenehmigungsErgebnis
--                ^^^^^^ vs Vorgang

data Genehmigungsregel =
  GenehmigungsRegel {
    genehmigungsregelName :: String,
    genehmigungsregelProzess ::
      Beleg -> GenehmigungsProzess (Maybe GenehmigungsErgebnis)
  }

data GenehmigungsErgebnis
  = GenehmigungErteilt
  | GenehmigungZurueckgewiesen
  | GenehmigungFrageGenehmiger Frage
```



# 1. Genehmigungsregel

```
genehmigungsregel1 :: Geld -> Genehmigungsregel
genehmigungsregel1 grenze =
  let prozess beleg =
    do let projekt = belegProjekt beleg
       belege <- belegeDerLetztenWoche projekt
       if belegeSumme belege <= grenze
       then return (Just GenehmigungErteilt)
       else return Nothing
  in GenehmigungsRegel {
    genehmigungsregelName = "Wenn Spesen für Projekt X anfallen
    ++ "die am Ende der Woche insgesamt kleiner als Y Euro sind,
    "dann genehmige sofort und frage nicht nach.",
    genehmigungsregelProzess = prozess
  }
```



## 2. Genehmigungsregel

```
genehmigungsregel2 :: Genehmigungsregel
genehmigungsregel2 =
  let prozess beleg =
    do let projekt = belegProjekt beleg
       durchschnitt <- wochenDurchschnitt projekt
       belege <- belegeDerLetztenWoche projekt
       if belegeSumme belege >= skalriereGeld 1.2 durchschnitt
       then return (Just (GenehmigungFrageGenehmiger
                           (Frage "Ganz schön teuer.")))
       else return Nothing
  in GenehmigungsRegel {
    genehmigungsregelName = "Wenn Spesen entstehen, "
    ++ "die mehr als 20% vom Wochendurchschnitt "
    ++ "des Projektes Z abweichen, "
    ++ "dann löse eine Frage an den Genehmiger aus.",
    genehmigungsregelProzess = prozess
  }
```





# Hilfsfunktion im Genehmigungsprozeß

```
belegeDerLetztenWoche :: Projekt -> GenehmigungsProzess [Beleg]
belegeDerLetztenWoche projekt =
  do projektBelege <- holeProjektBelege projekt
     stichtag <- frageStichtag
     let letzterTag = Calendar.addDays (-1) (montagVon stichtag)
         woche = Calendar.weekAllDays Calendar.Monday letzterTag
         belege = filter (\beleg -> belegDatum beleg `elem` woche)
                        projektBelege
  return belege
```



# Genehmigungsmonade

```
data GenehmigungsProzess a =  
    HoleProjektBelege Projekt ([Beleg] -> GenehmigungsProzess a)  
  | FrageStichtag (Calendar.Day -> GenehmigungsProzess a)  
  | GenehmigungFertig a
```



# Genehmigung durchführen

```
data GenehmigungsKontext =  
  GenehmigungsKontext {  
    genehmigungsKontextStichtag :: Calendar.Day,  
    genehmigungsKontextBelege  :: [Beleg] -- es gibt mindestens ein  
  }  
  
runGenehmigungsProzess :: GenehmigungsKontext -> GenehmigungsProzess
```



# Fallout

```
data Beleg = Beleg {  
    belegNummer :: Int,  
    belegInfo  :: BelegInfo,  
    belegVorgang :: Vorgang,  
    belegDatum :: Calendar.Day, -- erfunden  
    belegGeld  :: Geld, -- erfunden  
    belegKostenstelle :: Kostenstelle,  
    belegProjekt :: Projekt -- neu  
}
```





# Zurück zu den User Stories

```
data Spesenkonto a =
```

```
-- "Als Beleg-Scanner möchte ich Spesenbelege einscannen,  
-- im System ablegen und einer Kostenstelle zuordnen."
```

```
  LegeBelegAb
```

```
    Spesenritter Calendar.Day BelegInfo Projekt Vorgang Geld Kos  
    (Beleg -> Spesenkonto a)
```

```
-- "Als Spesenritter möchte ich in meinem Spesenkonto erkennen  
-- können, welche meiner Belege schon erfasst wurden,  
-- welche genehmigt wurden und welche schon ausgezahlt sind."
```

```
| SpesenritterBelegStatus
```

```
  Spesenritter
```

```
    ([ (Beleg, BelegStatus) ] -> Spesenkonto a)
```

```
-- "Als Genehmiger möchte ich per E-Mail über einen zu  
-- genehmigenden Vorgang informiert werden, aber nur wenn  
-- dessen Spesenbetrag ein eingestelltes Limit überschreitet."
```

```
| GenehmigerGenehmigungsStatus
```

```
    ([ (Beleg, GenehmigungsStatus) ] -> Spesenkonto a)
```



# Aus den User Stories

```
data Spesenkonto a =
```

```
-- ...
```

```
| WendeGenehmigungsRegelAn Genehmigungsregel Beleg (()) -> Spesen
```

```
-- "Als Genehmiger möchte ich eine Rückfrage
```

```
-- zu einem Vorgang an den Spesenritter stellen können."
```

```
| StelleRueckfrage Beleg Frage (()) -> Spesenkonto a)
```

```
| Genehmige Beleg (()) -> Spesenkonto a)
```

```
| LehneAb Beleg (()) -> Spesenkonto a)
```

```
| BeantworteRueckfrage Beleg Antwort (()) -> Spesenkonto a)
```

```
| StelleRechnung Rechnung (()) -> Spesenkonto a)
```

```
| ZahleSpesen Beleg Auszahlung (()) -> Spesenkonto a)
```

```
-- Steuern zahlen?
```

```
| SpesenkontoFertig a
```



# Kontext und Zustand

```
data SpesenkontoState = SpesenkontoState {  
    spesenkontoStateBelegStatus :: Map Beleg BelegStatus,  
    spesenkontoStateRechnungen :: [Rechnung],  
    spesenkontoStateAuszahlungen :: [(Beleg, Auszahlung)]  
}
```

```
runSpesenkonto :: Calendar.Day -> Antworten -> SpesenkontoState ->  
                Spesenkonto a -> (a, SpesenkontoState)
```



# Funktionalitätsbereiche

LegeBelegAb  
SpesenritterBelegStatus  
GenehmigerGenehmigungsStatus

WendeGenehmigungsRegelAn  
StelleRueckfrage  
Genehmige  
LehneAb  
BeantworteRueckfrage

StelleRechnung  
ZahleSpesen





# Willkür bei der Genehmigung

```
data Genehmigungsergebnis
```

```
  = Genehmigungerteilt  
  | Genehmigungzurueckgewiesen  
  | Genehmigungfragegenehmiger Frage
```

```
instance Semigroup Genehmigungsergebnis where
```

```
  Genehmigungerteilt <> _ = Genehmigungerteilt  
  _ <> Genehmigungerteilt = Genehmigungerteilt  
  Genehmigungzurueckgewiesen <> _ = Genehmigungzurueckgewiesen  
  _ <> Genehmigungzurueckgewiesen = Genehmigungzurueckgewiesen  
  Genehmigungfragegenehmiger frage1 <> Genehmigungfragegenehmiger  
    ???
```



# Halbgruppe der Genehmigung

```
data Genehmigungsergebnis
```

```
  = GenehmigungErteilt
```

```
  | GenehmigungZurueckgewiesen
```

```
  | GenehmigungFrageGenehmiger [Frage]
```

```
-- Halbgruppe
```

```
-- (a <> b) <> c == a <> (b <> c)
```

```
instance Semigroup Genehmigungsergebnis where
```

```
  GenehmigungErteilt <> _ = GenehmigungErteilt
```

```
  _ <> GenehmigungErteilt = GenehmigungErteilt
```

```
  GenehmigungZurueckgewiesen <> _ = GenehmigungZurueckgewiesen
```

```
  _ <> GenehmigungZurueckgewiesen = GenehmigungZurueckgewiesen
```

```
  GenehmigungFrageGenehmiger fragen1 <> GenehmigungFrageGenehmiger
```

```
    GenehmigungFrageGenehmiger (fragen1 ++ fragen2)
```



# Was haben wir getan?

- ausführbare Spezifikation
- "dependency injection"
- Probleme im Lastenheft gefunden
- Monade für Genehmigungsprozess
- Algebra für Genehmigungsergebnisse
- => Flexibilität bei den Abläufen
- fehlt noch: Algebra für Geld



# Was können wir jetzt tun?

- Unit-Tests
- property-based tests
- Unterhaltung mit Fachbereich über Abläufe
- DSL für Regeln
- schneiden





# Parnas on Architecture

“We have tried to demonstrate by these examples that it is almost always incorrect to begin the decomposition of a system into modules on the basis of a flowchart. We propose instead that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others.”

*D. Parnas: On the Criteria To Be Used in Decomposing Systems into Modules. CACM, 15:12.*



