Fallstudie
zur Spezifikation von Fachkomponenten

Jörg Ackermann

Überblick

- **Einleitung und Vorgehen**
- Ebenen der Spezifikation
- Zusammenfassung

Einleitung

- **Beitrag stellt Ergebnisse einer Fallstudie vor, in der eine Komponente beispielhaft spezifiziert wird**

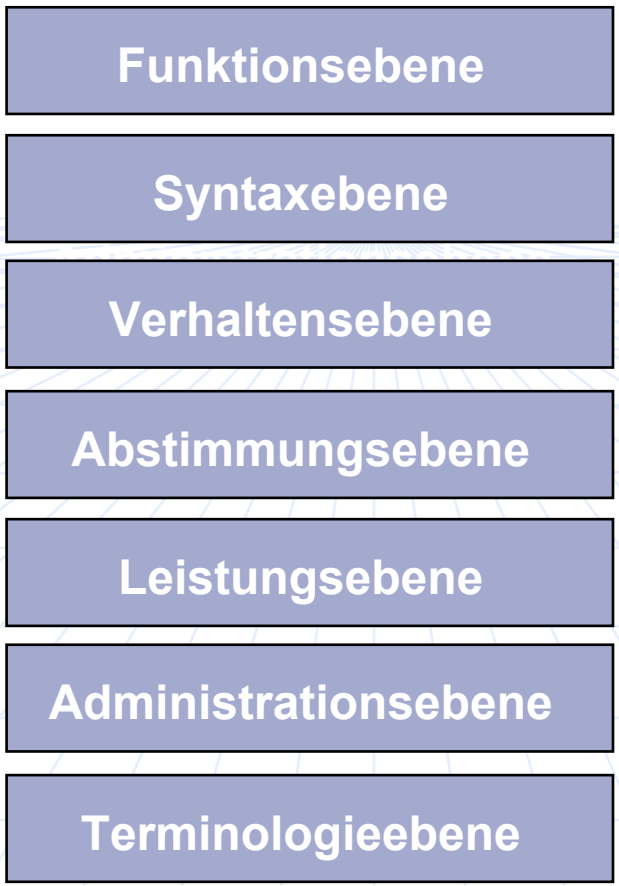
- **Grundlage der Fallstudie: Memorandum des GI-Arbeitskreises 5.10.3.**
- **Fallstudie beschreibt Komponente auf allen Ebenen**
- **Hauptfokus der Untersuchung und Ergebnisse lag auf:**
  - Syntaxebene
  - Verhaltensebene
  - Abstimmungsebene

Vorstellung der Beispielkomponente

- **(Pseudo-) Komponente „Flugticketverkauf“**
  - Funktionalität ist Teil einer SAP-Schulungsanwendung
  - Existiert seit Release 6.10 des SAP Web Application Server
  - Stellt Dienste zur Verfügung, die ein Reisebüro zum Verkauf von Flugtickets benötigt
- **Wurde nicht als Komponente implementiert, ist jedoch weitgehend gekapselt und kann daher als Fachkomponente betrachtet werden**
- **Komponente wurde unabhängig von der Fallstudie erstellt**
- **Wichtig: Aus dem Beispiel ergibt sich keine Aussage über die derzeitige und zukünftige Komponentenstrategie von SAP!**

Ebenen der Spezifikation

- **Komponente wird auf folgenden Ebenen spezifiziert (in dieser Reihenfolge):**



Einige allgemeine Punkte

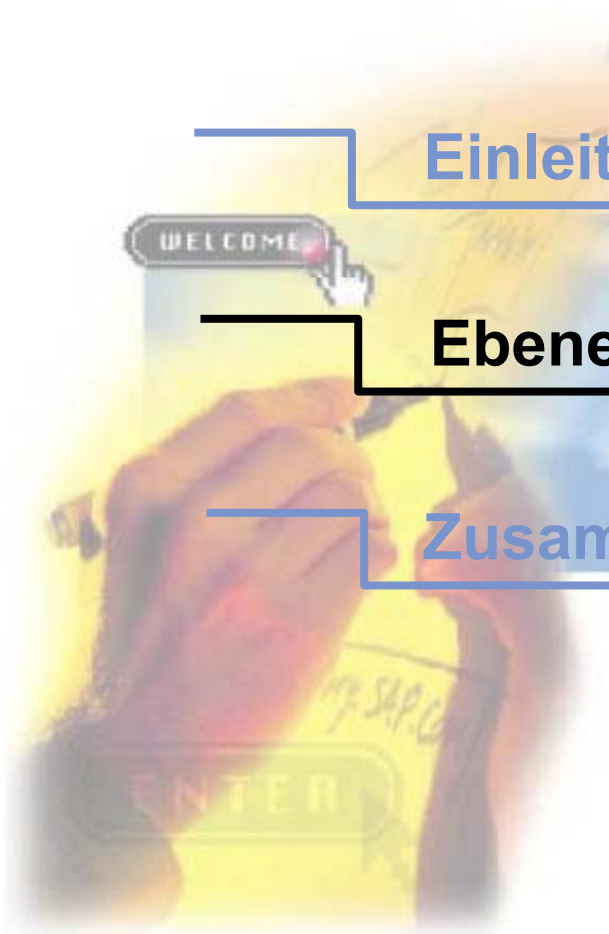
- **Spezifikation muss die Außensicht einer Komponente vollständig beschreiben**
- → **Beschreibung der angebotenen Dienste und der benötigten Dienste notwendig**
  - Relevant für Syntaxebene, Verhaltensebene und Abstimmungsebene

- **Deutliche Unterscheidung zwischen angebotenen und benötigten Diensten nötig**
- → **Folgende Notation in der Fallstudie:**
  - *NameDerFachkomponente::AngebotenerDienst()*
  - *Extern::BenötigterDienst()*
- **Beispiel:**
  - *Flugticketverkauf::Flugreise::Anlegen()*
  - *Extern::Reisebüro::PrüfeExistenz()*
- **Keine Aussage, welche andere Komponente oder Komponenten die benötigten Dienste zu erbringen hat.**

Thema Objektorientierung

- **Auswirkungen von OO auf die Spezifikation von Fachkomponenten muss noch näher untersucht werden**
- **Mindestens drei verschiedene Arten, wie OO bei Komponenten verwendet werden kann:**
- **a. Komponenteninterne objektorientierte Implementierung**
  - Keine Auswirkungen auf Spezifikation
- **b. Komponentenübergreifende objektorientierte Implementierung**
  - d.h. es können von außerhalb der Komponente Objekte instanziiert werden und/oder es werden Objekte an den Schnittstellen der Dienste übergeben
  - Erhöht Komplexität der Spezifikation und macht Semantik schwerer beherrschbar
  - Sollte in einer eigenen Fallstudie genauer untersucht werden
- **c. Modellierung und Beschreibung der Komponente mit objektorientierten Konstrukten**
  - Ermöglicht eine Reihe von Vorteilen bei der Spezifikation → wird allgemein empfohlen

Überblick



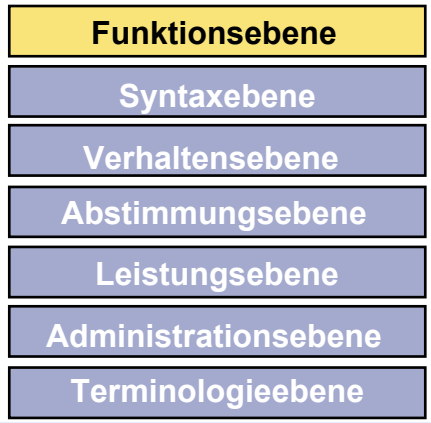
Einleitung und Vorgehen

Ebenen der Spezifikation

Zusammenfassung

Funktionsebene

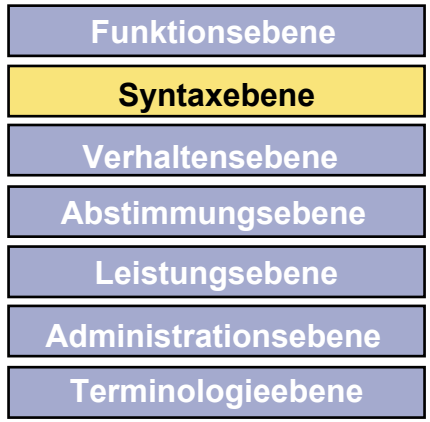
- **Inhalt und Aufbau der Funktionsebene**
  - Funktionalität der Komponente
  - Unterstützte betriebliche Aufgaben
  - Verwendungsmöglichkeiten
  - Beschreibung der wichtigsten Entitäten
  - Beschreibung der angebotenen und erwarteten Dienste
- **Es wurde eine Zuordnung zwischen den betrieblichen Aufgaben und den dafür notwendigen Diensten hergestellt:**



Betriebliche Aufgabe	Dienst der Komponente	Externer Dienst
Angebotserstellung für Flugreisen	Flugverbindung::LiefereListe Flugverbindung::LiefereDetails	Flugverfügbarkeit::Check
...		

Syntaxebene: Vorgehen

- **OMG IDL und OCL verwenden verschiedene Modularisierungskonstrukte**
- **Lösungsmöglichkeit zur Umgehung dieses Methodenbruchs:**
- **Eindeutige Zuordnung dieser Konstrukte und konsistente Verwendung**



Fachkonzept	Beispiel	OMG IDL	OCL / UML
Fachkomponente	Flugticketverkauf	module	package
Entität	Flugreise	interface	class
Dienst	Anlegen	operation	method

- **In der Fallstudie wurden definiert:**
  - Zwei Module „Flugticketverkauf“ und „Extern“
  - Jeweils mit Datentypen und Interfaces

Syntaxebene: Beispiel

```
interface Flugreise {
    enum                    FlugklasseTyp {Y, C, F};
    typedef string<8>       FlugkundeTyp;
    typedef string<8>       ReisenummerTyp;
    ...
    struct ReisedatenTyp {
            ReisebüronummerTyp          Reisebüronummer;
            FlugkundeTyp                Flugkundennummer;
            FlugverbindungsnummerTyp    Hinflugverbindung;
            DatumTyp                    Hinflugdatum;
            FlugverbindungsnummerTyp    Rückflugverbindung;
            DatumTyp                    Rückflugdatum;
            FlugklasseTyp               Flugklasse;    };
    ...
    void Anlegen(
            in  ReisedatenTyp           Reisedaten,
            in  PassagierlistenTyp      Passagierliste,
            out ReisebüronummerTyp      Reisebüronummer,
            out ReisenummerTyp          Reisenummer,
            out ReisepreisTyp           Reisepreis,
            out StatuslistenTyp         Statusmeldungen); };
};
```

Syntaxebene: Zusammenfassung

- **OMG IDL ist mit Einschränkungen geeignet**
  - Parameter können nicht optional sein
  - Semantisch reichere Datentypen können nicht definiert werden
    - Beispiel: Datum, Uhrzeit
  - OMG IDL erscheint mir zu implementierungsnah und zu spezifisch
    - Für die Nicht-CORBA-Welt eventuell zu einschränkend

- **Mögliche Alternativen:**
  - Web Service Definition Language (WSDL)
  - Komponenten-Framework spezifische Syntax

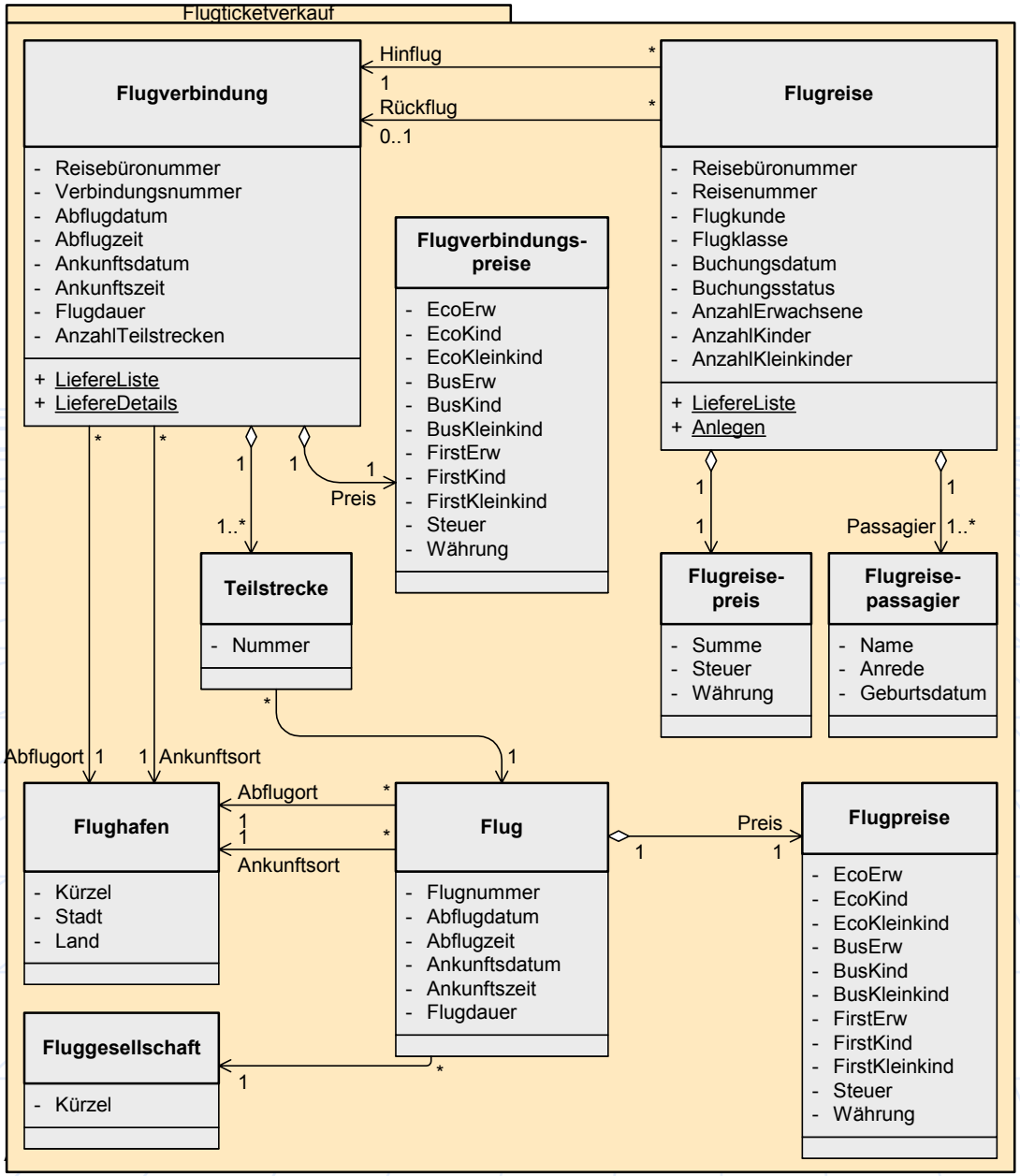
Funktionsebene
Syntaxebene
Verhaltensebene
Abstimmungsebene
Leistungsebene
Administrationsebene
Terminologieebene

Verhaltensebene: Ergebnisse

- **Es sollte unbedingt ein UML-Modell verwendet werden**
  - Stellt wichtige Entitäten und deren Beziehung dar
  - Macht Bedingungen verständlicher und erhöht die Ausdruckskraft
  - Modell ist reines Spezifikationsartefakt!
    - Enthält nur Entitäten / Daten, über die in der Spezifikation Aussagen getroffen werden
    - Muss keinen Bezug zur Implementierung haben
    - → Verletzt nicht den Black-Box-Gedanken von Komponenten
- **Handhabung von Vorbedingungen**
  - Angabe einer Vorbedingung immer mit Angabe eines Fehlers / einer Ausnahme bei nicht erfüllter Vorbedingung
- **Alle Bedingungen zusätzlich in Prosaform ausdrücken**

Funktionsebene
Syntaxebene
Verhaltensebene
Abstimmungsebene
Leistungsebene
Administrationsebene
Terminologieebene

Verhaltensebene: UML - Modell



- Funktionsebene
- Syntaxebene
- **Verhaltensebene**
- Abstimmungsebene
- Leistungsebene
- Administrationsebene
- Terminologieebene

Verhaltensebene: Beispiel 1

Entität Flugreise

Die Gesamtanzahl der Passagiere (Kardinalität der mit der Flugreise verbundenen Passagierliste) ist die Summe der Attribute *AnzahlErwachsener*, *AnzahlKinder* und *AnzahlKleinkinder*.

```
Flugticketverkauf::Flugreise
 inv: self.Passagier->size = self.AnzahlErwachsene +
                    self.AnzahlKinder + self.AnzahlKleinkinder
```

Das Attribut *AnzahlKinder* beschreibt die Anzahl der Passagiere, die am Tage des Hinflugs mindestens 2 und höchstens 11 Jahre alt sind. Dabei beschreibt *PassAlter* das Alter des Passagiers in ganzen Jahren.

```
Flugticketverkauf::Flugreise
 inv: self.AnzahlKinder = self.Passagiere->select(pass |
    let (GebJahr  = pass.Geburtsdatum.div(10000)) in
    let (GebTag   = pass.Geburtsdatum – GebJahr * 10000) in
    let (FlugJahr = self.Hinflug.Abflugdatum.div(10000)) in
    let (FlugTag  = self.Hinflug.Abflugdatum – FlugJahr * 10000) in
    let     if    GebTag <= FlugTag
            then  PassAlter = FlugJahr – GebJahr
            else  PassAlter = FlugJahr – GebJahr – 1 endif  in

            (1 < PassAlter) and (PassAlter < 12) )->size
```

Verhaltensebene: Beispiel 2

Dienst Flugverbindung::LiefereDetails

Die angegebene *Flugverbindung* (definiert durch *Reisebüronummer* und *Verbindungsnummer*) existiert. Ist die Vorbedingung nicht erfüllt, wird ein entsprechender Fehler zurückgegeben.

`Flugticketverkauf::Flugverbindung::LiefereDetails(In Rbnr, Vbnr, Fldat, Out Fvbd, ..., Status)`

```
pre FlugverbindungExistiert:
   Flugverbindung->exists(fvb | fvb.Reisebüronummer   = Rbnr
                            and fvb.Verbindungsnummer = Vbnr)

post: FlugverbindungExistiert = false implies
   Status->exists(st | st.Typ = ´E´ and st.Nummer = ´250´)
```

Verhaltensebene: Beispiel 3

Dienst Flugverbindung::LiefereDetails

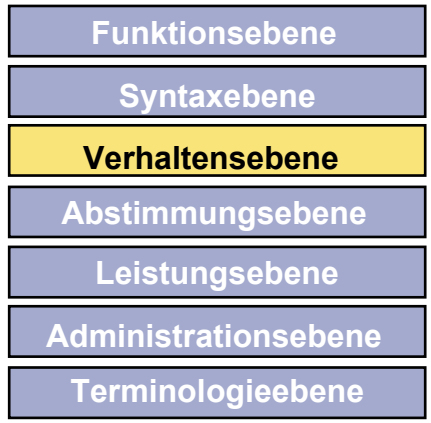
Die vom Dienst zurückgegebenen Daten beschreiben eine konkrete Flugverbindung. Das heißt, es gibt eine (in OCL als Objekt modellierte) Entität von *Flugverbindung*, deren Attribute mit den Exportdaten des Dienstes übereinstimmen. Dies auszudrücken, ist etwas länglich:

```
Flugticketverkauf::Flugverbindung::LiefereDetails(In Rbnr, Vbnr,
Fldat, Out Fvbd, ..., Status)
    post: Flugverbindung->exists (fvb |
            Fvbd.Reisebüronummer   =  fvb.Reisebüronummer
       and  Fvbd.Verbindungsnummer      =  fvb.Verbindungsnummer
       and  Fvbd.Abflugdatum      =  fvb.Abflugdatum
       and  Fvbd.Abflugzeit       =  fvb.Abflugzeit
       and  Fvbd.Startflughafen   =  fvb.Abflugort.Kürzel
       and  Fvbd.Abflugstadt      =  fvb.Abflugort.Stadt
       and  Fvbd.Ankunftsdatum    =  fvb.Ankunftsdatum
       and  Fvbd.Ankunftszeit     =  fvb.Ankunftszeit
       and  Fvbd.Zielflughafen    =  fvb.Ankunftsort.Kürzel
       and  Fvbd.Ankunftsstadt    =  fvb.Ankunftsort.Stadt
       and  Fvbd.Flugdauer        =  fvb.Flugdauer
       and  Fvbd.AnzahlTeilstrecken =  fvb.AnzahlTeilstrecken) ...
```

Bemerkung: Da es sich um eine Flugverbindung handelt, gelten alle Invarianten aus Kapitel 3.1.

Verhaltensebene: Zusammenfassung

- **OCL ist prinzipiell gut geeignet**
  - Fast alle Bedingungen konnten adäquat ausgedrückt werden
  - Nur wenige Ausnahmen (Alternative: Funktionsebene)
- **Aber:**
- **Formale Sprache führt zu großem Aufwand**
  - Verhaltensebene ist 30 Seiten lang
  - Etwa 50% der gesamten Spezifikation
- **Soll die formale Spezifikation beibehalten werden, wäre eine gute Toolunterstützung sehr wünschenswert**



Abstimmungsebene: Ergebnisse

- **Nur wenige Bedingungen auf der Abstimmungsebene**
  - Möglicher Grund: Komponente implementiert eher Dienste und weniger Prozesse
  - Siehe auch folgende Beispiele
- **Temporale Form der OCL war dafür gut geeignet**
  - Aber: Erweiterung der Syntax um die Operatoren before() und after() nötig
- **Temporale Form der OCL führt zu einigen Einschränkungen in der OCL**
  - Diese sollten explizit angegeben werden
  - Sinnvoll ist eine strikte Trennung:
    - Reine OCL auf der Verhaltensebene
    - Temporale Form der OCL auf der Abstimmungsebene

Abstimmungsebene: Einschränkungen in der OCL

- Funktionsebene
- Syntaxebene
- Verhaltensebene
- **Abstimmungsebene**
- Leistungsebene
- Administrationsebene
- Terminologieebene

- **Verzicht auf die Auswertbarkeit eines OCL-Ausdrucks zu einem bestimmten Zeitpunkt**
  - Wert eines OCL-Ausdrucks muss jederzeit (bei Vor- und Nachbedingungen bei der Methodenausführung) ermittelbar sein
  - Ist bei zukunftsgerichteten Operatoren nicht möglich
- **Zulassen von Nicht-Query-Methoden in temporalen OCL-Ausdrücken**
  - Einschränkung auf Query-Methoden (wie in OCL notwendig) ist nicht sinnvoll
- **Konsequenz: Temporale Operatoren werden in der Spezifikation nur als Modellierungskonstrukt für die Mensch-Mensch-Kommunikation verwendet**

Abstimmungsebene: Beispiel 1

Dienst Flugverbindung::LiefereListe

Während der Abarbeitung des Dienstes *Flugverbindung::LiefereListe* ruft dieser Dienst den externen Dienst *Extern::Reisebüro::PrüfeExistenz*. Soll also der Dienst *Flugverbindung::LiefereListe* verwendet werden, muss der externe Dienst von einer anderen Komponente zur Verfügung gestellt werden.

```
Flugticketverkauf::Flugverbindung::LiefereListe(Rbnr, ...)
 post: before(Extern::Reisebüro::PrüfeExistenz(Rbnr))
            sometime_since_last
       before(Flugverbindung::LiefereListe(Rbnr, ...))
  and after(Extern::Reisebüro::PrüfeExistenz(Rbnr))
            sometime_since_last
       before(Extern::Reisebüro::PrüfeExistenz(Rbnr))
```

Diese Bedingung kann ohne die zusätzlichen Operatoren before() und after() nicht ausgedrückt werden.

Abstimmungsebene: Beispiel 2

Dienst Flugverbindung::LiefereListe

Die Fachkomponente Flugticketverkauf ist so erstellt, dass Parallelverarbeitung unterstützt wird. Allerdings kann die Methode *Flugreise::Anlegen* nur unter bestimmten Bedingungen aufgerufen werden, solange noch ein anderer Aufruf derselben Methode abgearbeitet wird.

Die Bedingung ist, dass sich die jeweiligen Teilstreckenflüge voneinander unterscheiden. Ansonsten kann es bei der Buchung von Plätzen bei der Fluggesellschaft zu Sperrproblemen kommen.

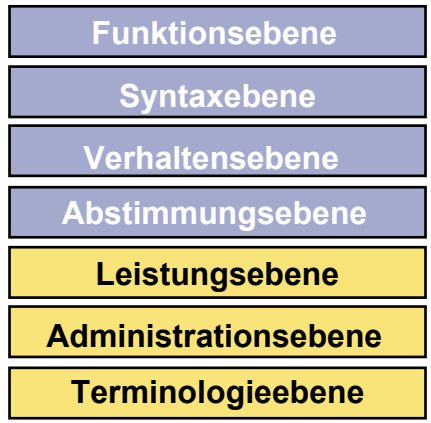
```
Flugticketverkauf::Flugreise::Anlegen(Rd2, ...)
 pre: not after(Flugreise::Anlegen(Rd1, ...))
    sometime_since_last before(Flugreise::Anlegen(Rd1, ...))
    implies

    let (flüge1 = ...) in
    let (flüge2 = ...) in

    flüge1->forAll(fl1 | flüge2->forAll(fl2 |
      fl1.Fluggesellschaft.Kürzel <> fl2.Fluggesellschaft.Kürzel
           or fl1.Flugnummer      <> fl2.Flugnummer
           or fl1.Abflugdatum     <> fl2.Abflugdatum ) )
```

Weitere Ebenen

- **Administrations- und Leistungsebene**
  - Analog zur Komponente „Bankleitzahlen“ im Memorandum
  - Auf diesen Ebenen ist weitere Detailarbeit nötig
  - Aufgetretenes Problem: Viele der Eigenschaften werden mehr durch das Komponenten-Framework und weniger durch die Komponente bestimmt
    - Ist Spezifikation solcher Eigenschaften sinnvoll?
- **Terminologieebene**
  - Spezifikation durch ein Thesaurus (insgesamt 23 Begriffe)
  - War dafür gut geeignet und stellte kein Problem dar



Überblick

Einleitung und Vorgehen

Ebenen der Spezifikation

Zusammenfassung

Zusammenfassung

- **Fachkomponente konnte anhand des Memorandums zufriedenstellend spezifiziert werden**
  - Mit Einschränkungen auf der Administrations- und Leistungsebene
- **Aber:**
- **Aufwand zur Erstellung ist relativ hoch**
  - Betrug ein mehrfaches des Aufwands für die Erstellung der Komponente (herkömmliche Spezifikation, Implementierung, Test und Dokumentation)
  - Dies enthält Einmaleffekte wie Einarbeitung in die verschiedenen Techniken und bestimmte Erfahrungen / Fehler einer ersten Anwendung
  - Ohne diese Einmaleffekte ist der Aufwand schätzungsweise gleich dem Erstellungsaufwand
- **Spezifikation ist ziemlich umfangreich (60 Seiten)**
  - Obwohl Komponente mit 4 Diensten nicht umfangreich ist

→ **Möglichkeiten zur Reduzierung des Aufwands sind nötig, z.B.**

  - Techniken und Tools zur effektiven Erstellung einer Spezifikation
  - Verwenden der Verhaltensebene für automatische Test