OFFIS

OLDENBURGER FORSCHUNGS- UND ENTWICKLUNGSINSTITUT
FÜR INFORMATIK-WERKZEUGE UND -SYSTEME

Prozessorientierte Beschreibung von Fachkomponenten

Holger Jaekel
jaekel@offis.de

Thorsten Teschke
teschke@offis.de

Agenda

- Diskussion des Memorandums zur Vereinheitlichung der Spezifikation von Fachkomponenten
- Vorstellung des CDL-Komponentenmodells
- Vorstellung der CDL-Verhaltensbeschreibungen
- Notationsvorschläge auf der Basis von UML
- Ausblick

Diskussion des Memorandums: Syntax-Ebene

- Vorschlag: OMG IDL; Repräsentation von Komponenten durch Interfaces
- Komponten und Objekte:
  - keine Spezifikation direkter (prozeduraler) und indirekter (objektorientierter) Schnittstellen
- Kontrakte:
  - keine Spezifikation benötigter Dienste möglich
- Frameworks:
  - keine Spezifikation von Anwendungs- und System-Frameworks
  - Spezifikation von Abhängigkeiten von Frameworks unklar

Diskussion des Memorandums: Verhaltens-Ebene

- Vorschlag: Object Constraint Language
- deklarative Beschreibungen
- Beschreibungen werden schnell komplex und für Anwender schwer verständlich
- Zielgruppe: Fachanwender, Bewertung der Eignung von Fachkomponenten
- kein formaler Abgleich zum Aufdecken von Inkompatibilitäten

Diskussion des Memorandums: Abstimmungs-Ebenen

OFFIS

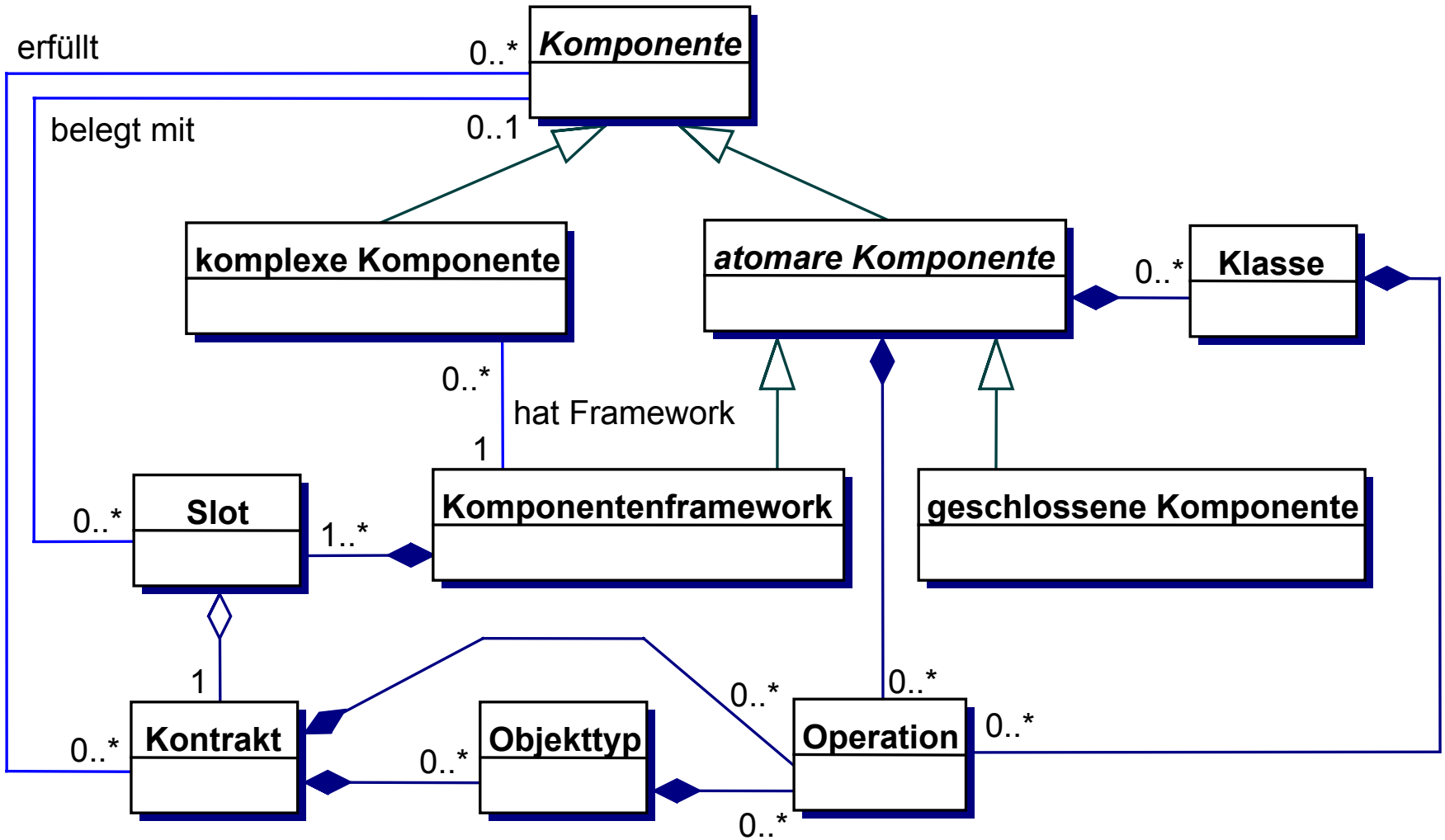
- Vorschlag: um temporale Operatoren erweiterte OCL
- deklarative Sichtweise (Vor- und Nachbedingungen)
- kein Abgleich mit den betrieblichen Abläufen, die häufig in prozeduraler Form vorliegen (→ Geschäftsprozesse)

CDL-Komponentenbeschreibungen

- Herkunft: Projekt KOSOBAR (Komponentenbasierte Softwareentwicklung auf Basis von Referenzmodellen)
- Zielsetzung:

  Entwicklung eines Makroprozesses und einer entsprechenden Rahmenarchitektur für die verteilte Entwicklung von komponentenbasierten Anwendungssystemene
- Entwicklung von CDL (Component Description Language)
  - integrierte Beschreibung von Struktur und Verhalten
  - Unterscheidung zwischen Komponenten und Kontrakten
  - Verhaltensbeschreibungen basierend auf Statecharts und Prozesstermen
  - Abstraktion von existierenden Komponentenmodellen

CDL-Komponentenmodell



CDL-Verhaltensbeschreibungen (1)

OFFIS

- Prozessorientierte Beschreibungen auf der Protokollebene
- Basierend auf Prozesstermen aus dem π-Kalkül und UML-Statecharts
- Spezifikation von aktivem und passivem Verhalten
  - Aktives Verhalten: Reihenfolge der benötigten Dienste der Komponente
  - Passives Verhalten: Die in einem Zustand von der Komponente angebotenen Dienste

CDL-Verhaltensbeschreibungen (2)

- Semantik der Verhaltensbeschreibungen von Komponenten und Klassen:

  Beschreibungen spezifizieren die *obere* Grenze der *möglichen* Interaktionssequenzen

- Semantik der Verhaltensbeschreibungen von Kontrakten und Objekttypen:

  Beschreibungen spezifizieren die *untere* Grenze der *erwarteten* Interaktionssequenzen

CDL-Verhaltensbeschreibungen (3)

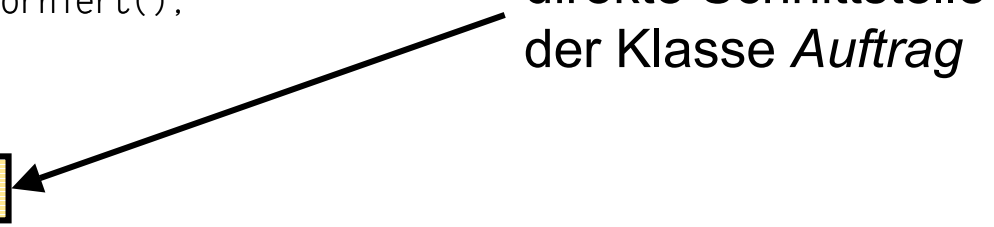
- Basis: π-Kalkül
  - send-Aktionen (!):
    - spezifizieren *aktives Verhalten*
    - Verwendung in Methoden
  - receive-Aktionen (?):
    - spezifizieren *passives Verhalten*
    - Verwendung in Prozessbeschreibungen von Klassen und Komponenten
- Basis: UML-Statecharts
  - Zustände, Trigger, Aktionen, Call Events

CDL-Verhaltensbeschreibungen (4)

```
component Auftragsabwicklung {

  class Auftrag {

    rechnungDrucken() {...};
    stornieren() {...};

    created()     = initialized();
    initialized() = rechnungDrucken?().initialized()
                    + stornieren?().storniert();
    storniert()   = ();
  };

  auftragAnnehmen(out at: Auftrag) {
    new(Auftrag at);
    return(at);
  };

  created()     = initialized();
  initialized() = auftragAnnehmen?(at).initialized();
};
```

CDL-Verhaltensbeschreibungen (4)

```
component Auftragsabwicklung {

  class Auftrag {

    rechnungDrucken() {...};
    stornieren() {...};

    created()     = initialized();
    initialized() = rechnungDrucken?().initialized()
                    + stornieren?().storniert();
    storniert()   = ();
  };

  auftragAnnehmen(out at: Auftrag) {
    new(Auftrag at);
    return(at);
  };

  created()     = initialized();
  initialized() = auftragAnnehmen?(at).initialized();
};
```



CDL-Verhaltensbeschreibungen (4)

```
component Auftragsabwicklung {

  class Auftrag {

    rechnungDrucken() {...};
    stornieren() {...};

    created()     = initialized();
    initialized() = rechnungDrucken?().initialized()
                    + stornieren?().storniert();
    storniert()   = ();
  };

  auftragAnnehmen(out at: Auftrag) {
    new(Auftrag at);
    return(at);
  };

  created()     = initialized();
  initialized() = auftragAnnehmen?(at).initialized();
};
```

(passives) Verhalten von *Auftragsabwicklung*

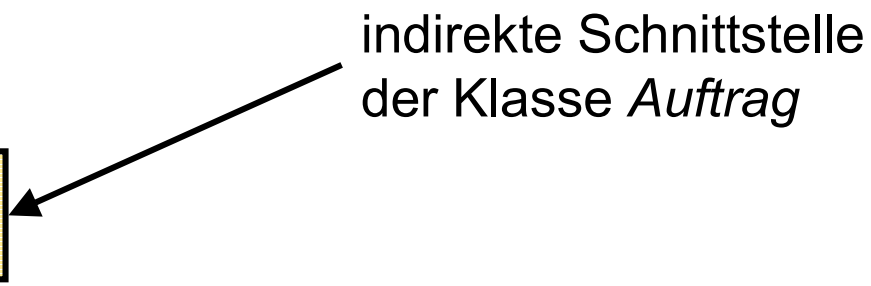
CDL-Verhaltensbeschreibungen (4)

```
component Auftragsabwicklung {

  class Auftrag {

    rechnungDrucken() {...};
    stornieren() {...};

    created()     = initialized();
    initialized() = rechnungDrucken?().initialized()
                    + stornieren?().storniert();
    storniert()   = ();
  };

  auftragAnnehmen(out at: Auftrag) {
    new(Auftrag at);
    return(at);
  };

  created()     = initialized();
  initialized() = auftragAnnehmen?(at).initialized();
};
```

(aktives) Verhalten von *auftragAnnehmen*

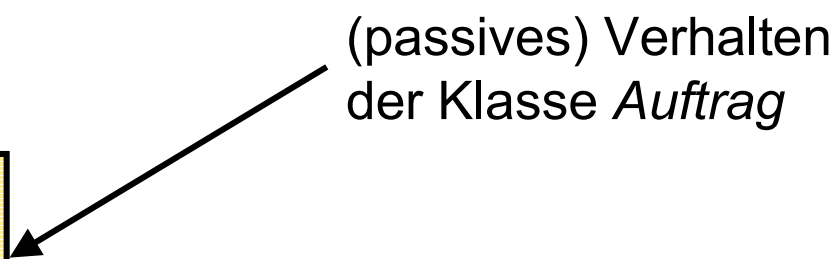
CDL-Verhaltensbeschreibungen (4)

```
component Auftragsabwicklung {

  class Auftrag {

    rechnungDrucken() {...};
    stornieren() {...};

    created()     = initialized();
    initialized() = rechnungDrucken?().initialized()
                    + stornieren?().storniert();
    storniert()   = ();
  };

  auftragAnnehmen(out at: Auftrag) {
    new(Auftrag at);
    return(at);
  };

  created()     = initialized();
  initialized() = auftragAnnehmen?(at).initialized();
};
```



CDL-Verhaltensbeschreibungen (4)

```
component Auftragsabwicklung {

  class Auftrag {

    rechnungDrucken() {...};
    stornieren() {...};

    created()     = initialized();
    initialized() = rechnungDrucken?().initialized()
                    + stornieren?().storniert();
    storniert()   = ();
  };

  auftragAnnehmen(out at: Auftrag) {
    new(Auftrag at);
    return(at);
  };

  created()     = initialized();
  initialized() = auftragAnnehmen?(at).initialized();
};
```



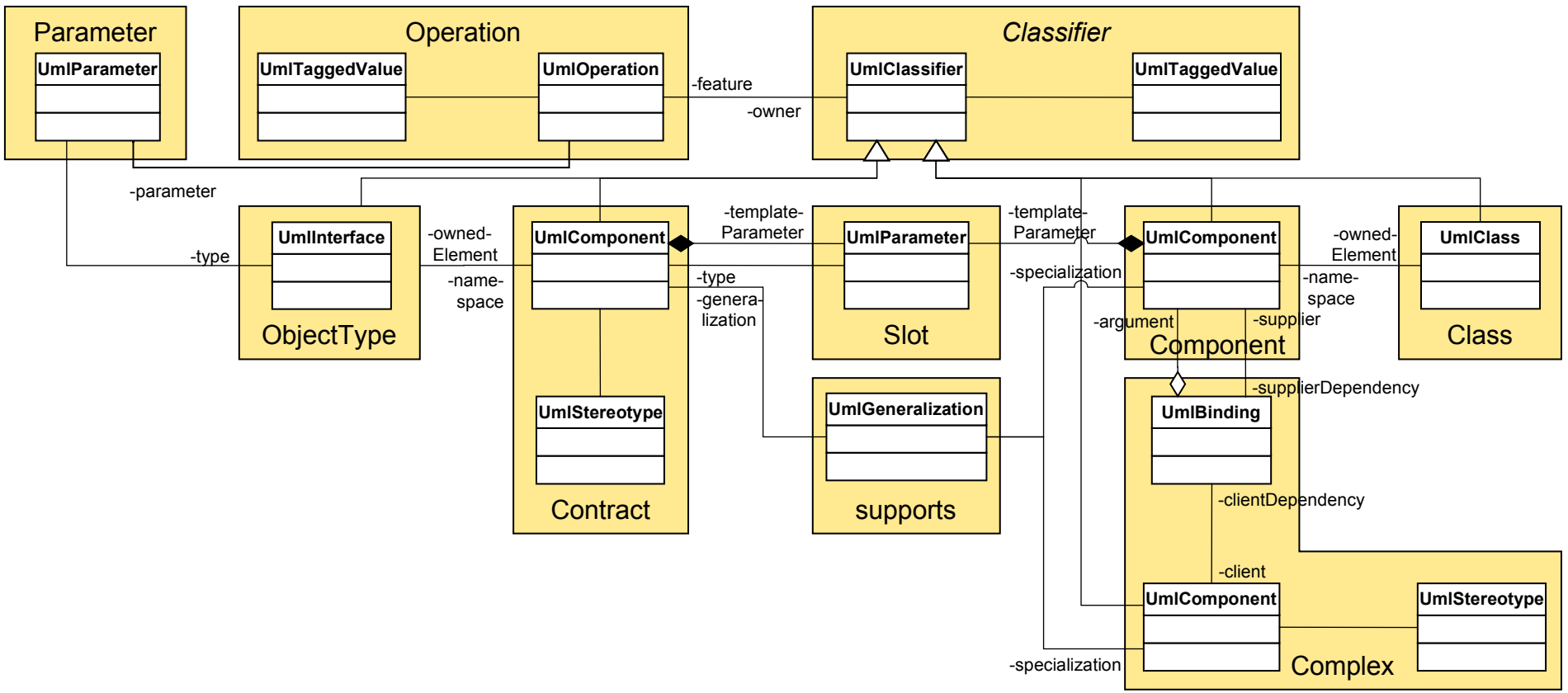
Notationsvorschlag Syntax-Ebene (1)

OFFIS

- Spezifikation von Fachkomponenten auf Basis der UML
  - Nutzung eines akzeptierten Standards
  - Ablegen von Komponentenbeschreibungen in UML-Repositories
- Abbildung des CDL-Komponentenmodells auf die UML
- Erweiterung der UML mittels *TaggedValue* und *Stereotype*

Notationsvorschlag Syntax-Ebene (2)

Abbildung des CDL-Komponentenmodells auf die UML



Notationsvorschlag Verhaltens-Ebene

- Anforderung: fachliche Ausrichtung der Beschreibungen
- Beschreibung der betriebswirtschaftlichen Dienste der Komponenten für den Abgleich mit Anforderungen
- Verbindung der Namen von Komponenten, Klassen und Operationen mit einer standardisierten Terminologie
- mögliche Grundlagen:
  - Terminologie aus KEBBA-Projekt
  - Normsprachen (vgl. Arbeiten von Ortner, Schienmann)

Intra-Komponenten-Abstimmungs-Ebene

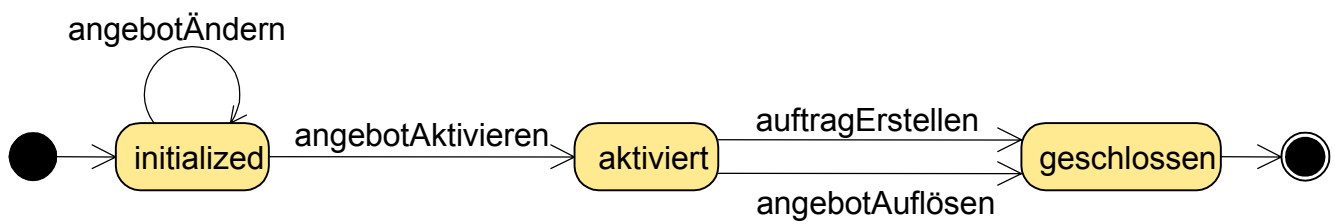
OFFIS

- Spezifikation der Reihenfolge, in der andere Komponenten die Dienste einer Komponente verwenden können
- Vorschlag: Prozessorientierte Spezifikation auf Basis der UML
  - Verständlichkeit
  - Abgleich mit Geschäftsprozessen
- Abbildung der passiven CDL-Verhaltensbeschreibungen auf UML
  - Verwendung von UML-Statecharts
  - Besonderheit in CDL: Parametrisierbare Zustände
    - Verwendung von *TaggedValues*

Intra-Komponenten-Abstimmungs-Ebene (Beispiel)

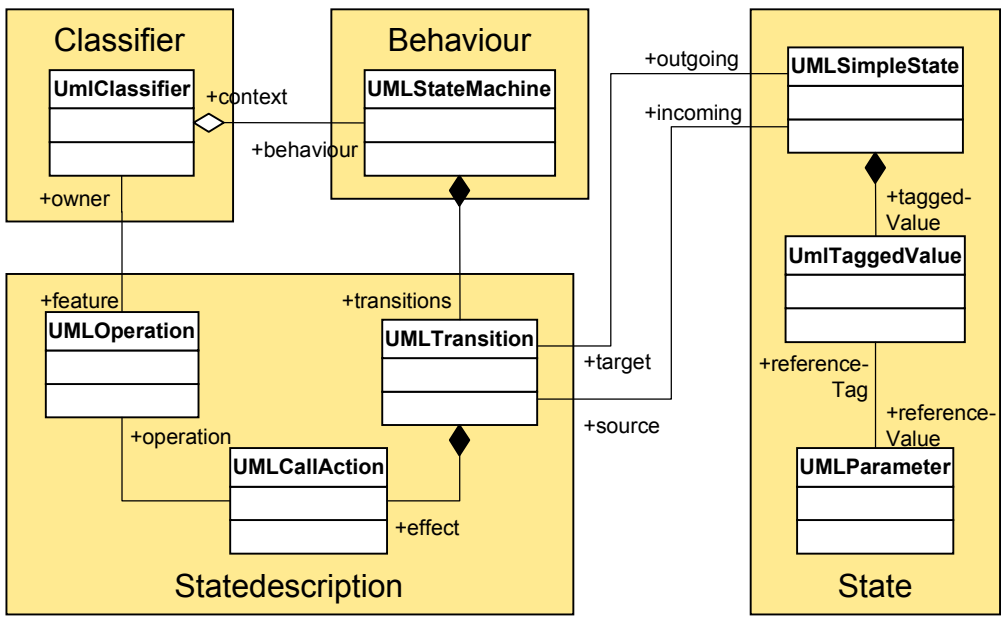
```
component Angebot {

  angebotÄndern() {...};
  angebotAktivieren() {...};
  angebotAuflösen() {...};
  auftragErstellen(out auftrag: Auftrag) {...};

  created()     = initialized();
  initialized() = angebotÄndern?().initialized()
                + angebotAktivieren?().aktiviert();
  aktiviert()   = ( auftragErstellen?(auftrag)
                  + angebotAuflösen?() ).geschlossen();
  geschlossen() = ();
};
```

Eine Komponente ist bereit, einen Dienst auszuführen, wenn der Automat sich in einem Zustand befindet, dessen Prozessterm die Kommunikation des entsprechenden Dienstes zulässt.



Notationsvorschlag Abstimmungs-Ebenen

Abbildung von CDL-Verhaltensbeschreibungen auf UML-Statecharts



Inter-Komponenten-Abstimmungs-Ebene

- Spezifikation der Reihenfolge, in der eine Komponente die Dienste anderer Komponenten verwendet
- Referenzierung der verwendeten Komponenten über Slots des zugrunde liegenden Frameworks
- Abbildung der aktiven CDL-Verhaltensbeschreibungen auf UML
  - Assoziation von Operationen mit einem Statechart
  - Abbildung der sequentiellen Methodenaufrufe durch verkettete *PseudoStates*

Zusammenfassung/Ausblick

- CDL
  - integrierte Beschreibung von Struktur und Verhalten
  - Unterscheidung zwischen Komponenten und Kontrakten
  - Verhaltensbeschreibungen basierend auf Statecharts und Prozesstermen
  - Abstraktion von existierenden Komponentenmodellen
- CDL-Beschreibungen lassen sich auf die UML abbilden
- CDL liefert Beiträge für die Beschreibung von Fachkomponenten auf der Syntax-Ebene und den Abstimmungs-Ebenen
- Definition der Verhaltens-Ebene durch standardisierte Terminologien