Eine Infrastruktur für den Austausch von Fachkomponenten

Holger Jaekel, Thorsten Teschke

OFFIS, Escherweg 2, 26121 Oldenburg, Deutschland, Tel.: (04 41) 97 22 - 1 25, Fax: - 1 02, E-Mail: {holger.jaekel | thorsten.teschke}@offis.de. URL: http://www.offis.de

Zusammenfassung. Neben der fachlichen und technischen Standardisierung ist für den Erfolg der komponentenbasierten Softwareentwicklung auch eine Unterstützung des Entwicklungsprozesses durch entsprechende Werkzeuge notwendig. In diesem Beitrag wird die im Projekt KOSOBAR entwickelte Infrastruktur zur Unterstützung der komponentenbasierten Softwareentwicklung vorgestellt. Mittels dieser Infrastruktur interagieren die an der Softwareentwicklung beteiligten Akteure durch Austausch standardisierter Komponentenbeschreibungen, deren Semantik durch einfache normsprachliche Sätze über einer standardisierten Terminologie definiert wird. Diese Beschreibungen werden von Suchdiensten verwendet, die auf einem Komponentenmarkt geeignete Komponenten auf Basis von Geschäftsprozessmodellen finden.

Schlüsselworte: Fachkomponente, Vorgehensmodell, Terminologie, Normsprache, Komponentensuche, Geschäftsprozessmodell, Behavioural Subtyping

1 Einleitung

Eine Voraussetzung für den Erfolg der komponentenbasierten Softwareentwicklung ist neben der technischen Standardisierung durch Komponentenmodelle wie z. B. Component Object Model (COM) und dessen Nachfolger DCOM und COM+, Enterprise JavaBeans (EJB) oder CORBA Component Model (CCM) und der fachlichen Standardisierung durch Interoperabilitätsbestrebungen wie z. B. ebXML oder OAGIS die Unterstützung des gesamten Entwicklungsprozesses durch geeignete Werkzeuge. Dazu gehören einerseits Werkzeuge zur Veröffentlichung standardisierter Komponentenbeschreibungen, andererseits sind Dienste für die Komponentensuche notwendig, die geeignete Komponenten auf einem Komponentenmarkt auf Basis fachlicher Anforderungen finden.

Ausgehend von dem Gedanken, Komponenten ähnlich wie große, betriebliche Standardsoftwareprodukte wie SAP R/3 durch strukturierte, semi-formale Beschreibungen (sogenannte *Softwarereferenzmodelle*) zu beschreiben [Rit98], ist in den Jahren 1999 bis 2002 das Projekt *KOSOBAR (Komponentenbasierte Softwareentwicklung auf Basis von Referenzmodellen)* in OFFIS durchgeführt worden. Ziel dieses Projekts ist die Entwicklung einer Infrastruktur für den Austausch von Fachkomponenten auf internetbasierten Komponentenmärkten gewesen. Dazu sind die an der Erstellung komponentenbasierter Software beteiligten Akteure und deren Interaktionen wie z. B. der Austausch von Komponenten oder Anforderungsdefinitionen in einem Vorgehensmodell für die komponentenbasierte Softwareentwicklung definiert worden. Ausgehend von diesem Vorgehensmodell wurde eine Infrastruktur zur Unterstützung komponentenbasierter Softwareentwicklungsprozesse entwickelt, die die Kommunikation der beteiligten Akteure auf der Grundlage standardisierter Komponentenbeschreibungen unterstützt. Diese Kommunikation betrifft insbesondere das Anbieten und die Suche von Fachkomponenten.

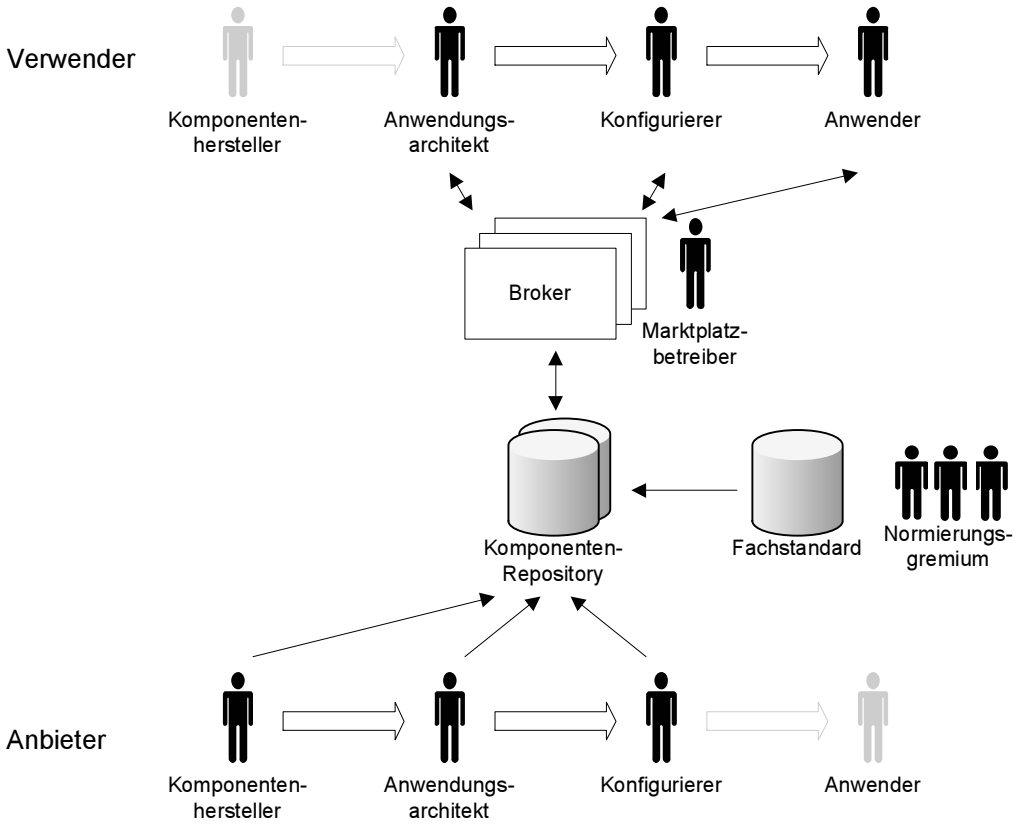

Abbildung 1: KOSOBAR-Vorgehensmodell

2 Vorgehensmodell und Architektur

Traditionelle (z. B. sequentielle, iterative oder evolutionäre) Vorgehensmodelle beschreiben in der Regel nur produktbezogene Aktivitäten wie Analyse, Entwurf, Implementierung, Test und Wartung, deren Ergebnisse unmittelbar in das Softwareprodukt eingehen. Demgegenüber werden prozessbezogene Aktivitäten wie z. B. die Koordination eines Softwareprojektes, die Kooperation zwischen den beteiligten Akteuren oder die Verwaltung des zu entwickelnden Produktes oft nicht in hinreichendem Maße berücksichtigt. Durch den Ansatz der komponentenbasierten Anwendungsentwicklung entsteht die Notwendigkeit, traditionelle Vorgehensweisen für die Entwicklung von Software zu überdenken, da dort Probleme wie die Vermarktung und der Erwerb von Komponenten nicht ausreichend berücksichtigt werden. Das im Projekt KOSOBAR entwickelte Vorgehensmodell [STR00] basiert auf der Annahme, dass sich künftig ein Komponentenmarkt etablieren wird, auf dem Anbieter umfassende Beschreibungen ihrer Komponenten in speziellen Repositories kostenlos zur Verfügung stellen. Broker ermöglichen Interessenten die Suche nach Komponenten auf Basis der in diesen Anbieter-Repositories abgelegten Beschreibungen. Abbildung 1 stellt das Vorgehensmodell anhand der beteiligten Akteure, ihrer Interaktionen sowie der zugrunde liegenden Architektur graphisch dar.

Während in anderen Vorgehensmodellen häufig nur die Rollen des Softwareherstellers und des Anwenders unterschieden werden, sehen wir die sechs Rollen Komponentenhersteller, Anwendungsarchitekt, Konfigurierer und Anwender sowie Marktplatzbetreiber und Normierungsgremium vor. Sie sind durch ihre jeweiligen charakteristischen Tätigkeiten (Entwicklung, Konstruktion, Konfiguration, Einsatz sowie Vermittlung und Normierung) gekennzeichnet. Die Rol-

le des *Komponentenherstellers* beinhaltet die Aufgabe der Entwicklung möglichst domänenunabhängiger Komponenten mit den Mitteln des klassischen Software Engineerings. Die Aufgabe des *Anwendungsarchitekten* besteht darin, aus verschiedenen Komponenten ein konfigurierbares Anwendungssoftwareprodukt zusammenzustellen. Die Grundlage eines solchen Anwendungssoftwareprodukts bilden in der Regel Frameworks, die durch Komposition mit geeigneten Komponenten erweitert werden müssen. Anwendungssoftwareprodukte können einerseits wieder als Komponente von einem Anwendungsarchitekten zur Entwicklung komplexerer Anwendungssoftwareprodukte weiterverwendet werden, andererseits können sie von einem *Konfigurierer* zu domänen-, branchen- oder unternehmenspezifischen Lösungen (vor-)konfiguriert werden. Indem der Konfigurierer den *Anwender* bei der Auswahl geeigneter Anwendungssoftwareprodukte und der Bewertung von Konfigurationsalternativen unterstützt, erbringt er zudem Beratungsleistungen gegenüber dem Anwender. Für die Auswahl von geeigneten Komponenten oder (vorkonfigurierten) Anwendungssoftwareprodukten nutzen die *Verwender* (Anwendungsarchitekten, Konfigurierer und Anwender) die Dienste eines *Brokers*, der von einem *Marktplatzbetreiber* unterhalten wird. Im Rahmen der Komponentensuche vergleicht ein Broker die von den *Anbietern* (Komponentenhersteller, Anwendungsarchitekten und Konfigurierer) in Komponenten-Repositories angebotenen Komponentenbeschreibungen mit den spezifizierten Anforderungen eines Verwenders. Zur Unterstützung der Interoperabilität von Komponenten sowie Effektivitätsverbesserungen bei der Komponentensuche beschreiben Anbieter die fachliche Semantik der von einer Komponente angebotenen Dienste unter Berücksichtigung von (domänenspezifischen) Fachstandards, die von *Normierungsgremien* spezifiziert werden. Die Rolle des Normierungsgremiums könnte einerseits von Organisationen wie z. B. die Open Applications Group (OAG) und die Organization for the Advancement of Structured Information Standards (OASIS) oder Unternehmen mit einer besonderen Kompetenz in einem Anwendungsbereich wie z. B. die DATEV eG eingenommen werden.

Ein Ziel des Projektes KOSOBAR ist es, die oben dargestellten Interaktionen zwischen den an der Erstellung von komponentenbasierten Anwendungssystemen beteiligten Akteuren informationstechnisch zu unterstützen. Dazu sind verschiedene Werkzeugen und Schnittstellen entwickelt worden, die im Folgenden überblicksartig vorgestellt werden sollen (siehe auch Abbildung 2). Die Aufgaben des Normierungsgremiums werden durch einen *Terminologie-Manager* zur Pflege domänenspezifischer Terminologien unterstützt, auf deren Grundlage Anbieter bzw. Verwender die fachliche Semantik von Komponenten bzw. Anforderungen beschreiben. Abschnitt 3 beschreibt den in KOSOBAR verfolgten Ansatz zum Aufbau solcher Terminologien und geht kurz auf den *Terminologie-Manager* ein. Zur Verwaltung von Komponentenbeschreibungen durch Anbieter dient der *CDL Component Manager*, der in der Beschreibungssprache *Component Description Language (CDL)* verfasste Komponentenbeschreibungen in einem UML-Repository ablegt. CDL und der *CDL Component Manager* werden in Abschnitt 4 vorgestellt. Die Suche nach Komponenten auf Komponentenmärkten wird durch Broker unterstützt, die auf in Form von Geschäftsprozessmodellen spezifizierte fachliche Anforderungen zurückgreifen. Der für die Komponentensuche verfolgte Ansatz wird in Abschnitt 5 beschrieben. Abschließend folgt eine Zusammenfassung in Abschnitt 6.

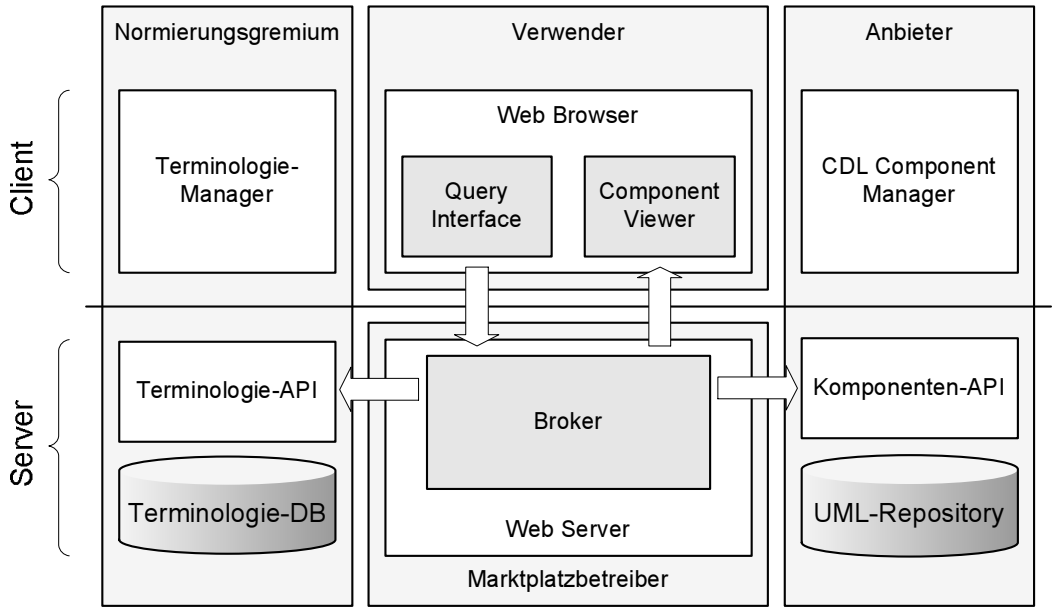

Abbildung 2: Architektur zur Unterstützung der komponentenbasierten Softwareentwicklung

3 Terminologische Semantik von Komponenten

3.1 Sprachliche Spezifikation der Semantik von Komponenten

In der Informatik wird die Semantik syntaktischer Konstrukte verbreitet formal (z. B. mit Hilfe der Prädikatenlogik) oder semi-formal (z. B. auf Basis der UML) definiert. So wird beispielsweise vorgeschlagen, die Semantik von Methoden durch prädikatenlogische Zusicherungen (Vor- und Nachbedingungen sowie Invarianten) zu spezifizieren (vgl. [Hoa69, Mey92, LW94]). Zwar sind derartige formale Spezifikationen präzise, sie weisen jedoch den Nachteil auf, nicht oder nur in eingeschränktem Maße für Fachexperten verständlich zu sein.

Alternativ zur formalen Spezifikation der Semantik von Methoden lässt sich die Bedeutung einer Methode im objektorientierten Paradigma auch als natürlichsprachliche Aussage verstehen, die bei jedem Aufruf der Methode geäußert wird. Der Bezeichner der Methode entspricht dabei einer *Aktion*, die durch ein *Prädikat* und ein *direktes Objekt* spezifiziert wird. Ihre Eingabe- und Ausgabeparameter werden durch ein entsprechende Anzahl *indirekter Objekte* in der Aussage repräsentiert [Tes02]. Als Akteur einer Methodenausführung betrachten wir die Bezeichnung der Benutzerrolle, die für die Ausführung der Methode erforderlich ist, und ordnen ihr die Stellung des *Subjekts* in der Aussage zu [JT02]. Dieses einfache syntaktische Muster lässt sich auf beliebige objektorientierte Methodendeklarationen anwenden. Das folgende Beispiel zeigt eine Methodendeklaration in OMG IDL-Notation:

```
interface SalesDepartment {
      Offer createOffer(in Request request);
}
```

Dabei wird in diesem Beispiel der Begriff „Offer“ (bzw. das deutsche Pendant „Angebot“) zweifach verwendet: zum einen im Methodennamen (dem Prädikat) und zum anderen als Bezeichner des Typs eines erzeugten Objekts (dem direkten Objekt). Damit wird explizit zwischen einer intensionalen Sicht (der Methodenname bezieht sich auf das abstrakte Konzept, das durch

den Begriff bezeichnet wird) und einer extensionalen Sicht (das Objekt bezieht sich auf eine Instanz des Konzepts) auf das „Angebot“ unterschieden. Wenn wir von der intensionalen Verwendung absehen, können wir den Aufruf der Methode `createOffer()` als die Äußerung „Ein Vertriebsmitarbeiter erzeugt ein Angebot mit einer Anfrage“ verstehen. Dabei geht die geforderte Benutzerrolle des Vertriebsmitarbeiters allerdings nicht aus der IDL-Spezifikation hervor. Außerdem haben wir die Lesbarkeit durch unbestimmte Artikel und die Präposition „mit“ erhöht.

3.2 Normsprachliche Spezifikation fachlicher Semantik

Natürliche Sprachen haben gegenüber formalen Sprachen den Vorteil, auch für Fachexperten verständlich zu sein. Sie weisen allerdings Schwächen wie Mehrdeutigkeiten, unzureichende Präzision und Missverständlichkeit auf, die auch als *Sprachdefekte* bezeichnet werden. *Kontrollierte Sprachen* wie z. B. *Attempto Controlled English (ACE)* [FS96] stellen einen Ansatz zur Behebung dieser Defekte natürlicher Sprachen dar. Sie bestehen aus einer vereinfachten Grammatik, die durch spezielle Muster für die Konstruktion von Sätzen (sogenannte *Satzbaupläne*) gegeben ist, und einem kontrollierten, d. h. vordefinierten und überwachten Vokabular, das für die Konstruktion von Sätzen zur Verfügung steht.

Normsprachen können als eine besondere Form kontrollierter Sprachen angesehen werden. Sie haben ihren Ursprung in der konstruktiven Wissenschaftstheorie und werden durch methodische Rekonstruktion der in einem Anwendungsbereich gesprochenen natürlichen (Fach-)Sprache entwickelt [Ort97]. Dabei bezeichnet der Begriff der Rekonstruktion die Klärung und eindeutige Definition der Bedeutung von Wörtern, ihrer Beziehungen untereinander (z. B. mereologische Strukturen, Abstraktionsbeziehungen) sowie von Regeln für ihren Gebrauch. In diesem Zusammenhang ist auch der Grundstock der Logik (Bindewörter „und“, „oder“) zu errichten ebenso wie die z. B. die Bedeutung der Abstraktion zu klären [Büt95].

Abhängig von der gewählten Einteilung in Wortarten gestatten Normsprachen grundsätzlich die Definition verschiedenster Satzbaupläne für die Konstruktion von Sätzen. So definiert *TAOS (Terminologiebasierter Ansatz für die objektorientierte Spezifikation)* [Sch97], ein Rahmenwerk für die Entwicklung objektorientierter Spezifikationen, eine umfassende Menge von Satzbauplänen für die Spezifikation objektorientierter Softwaresysteme. In unserer Arbeit zur fachlich ausgerichteten Beschreibung von Komponenten nutzen wir jedoch lediglich den Satzbauplan

$$[\ \mathtt{N}\ |\ \pi\ |\ \mathtt{P}\ |\ \mathtt{O}_1\ |\ ...|\ \mathtt{O}_n] \tag{1}$$

zur Beschreibung der Semantik von Diensten. Dabei bezeichnet `N` das Subjekt des Satzes (den sogenannten *Nominator*), π die Kopula „tut“, `P` das Prädikat, $\mathtt{O}_1$ das direkte Objekt und $\mathtt{O}_\mathtt{i}(\mathtt{i} > 1)$ evtl. vorhandene indirekte Objekte. Das Prädikat `P` wird immer in seiner Infinitivform gebraucht. Die Verwendung von Adverbien und Adjektiven ist in diesem einfachen Satzbauplan nicht vorgesehen. Ein Beispiel für einen Satz, der sich mit diesem Satzbauplan konstruieren lässt, ist `Angestellter` π `hinzufügen Auftragsposition zu-Auftrag` (lies „[ein] Angestellter tut hinzufügen [eine] Auftragsposition zu [einem] Auftrag“). In diesem Beispiel wird das indirekte Objekt „Auftrag“ um die Präposition „zu“ ergänzt.

Die Fähigkeit, Sätze zu bilden, erfordert ein gewisses Vokabular (*Material*), das genutzt werden kann, um einem Satzbauplan (dem formalen Rahmen) „Leben einzuhauchen“. Normsprachen greifen auf ein kontrolliertes Vokabular zurück, dessen Worte wie oben beschrieben methodisch rekonstruiert wurden. Die Bedeutung einzelner Wörter ist dabei mittels expliziter

Definitionen wie z. B. „ein Auftrag ist eine rechtliche Vereinbarung zwischen einem Abnehmer und einem Anbieter einer Leistung . . . “ festgelegt, während semantische Beziehungen zwischen Wörtern auf normsprachliche Aussagen reduziert werden können. So gestattet der Satzbauplan

$$[\ N_1\ |\ \epsilon\ |\ N_2\ ] \tag{2}$$

auszudrücken, dass ein Kunde ein Geschäftspartner ist („`Kunde` ϵ `Geschäftspartner`“) oder dass Müller ein Kunde ist („`Müller` ϵ `Kunde`“). Während die erste Aussage als *allgemeine Aussage* bezeichnet wird, die für die Entwicklung eines Schemas herangezogen werden kann, stellt die zweite Aussage eine *singuläre Aussage* dar, die als Beispiel einer zulässigen Ausprägung eines Schemas dienen kann [Ort97]. ϵ repräsentiert dabei die Kopula „ist ein“, die Generalisierungsbeziehungen zwischen Konzepten beschreibt. Ganzes-Teile-Beziehungen werden durch den Satzbauplan

$$[\ N_1\ |\ \nu\ |\ N_2\ ] \tag{3}$$

formuliert, wobei das Symbol ν für die Kopula „hat“ steht. Als Beispiel mag die Aussage „`Auftrag` ν `Auftragsposition`“ dienen.

Die Satzbaupläne 2 und 3 ermöglichen die Rekonstruktion der statischen Struktur eines Anwendungsbereichs, d. h. die Definition der relevanten Konzepte eines Anwendungsbereiches sowie deren semantische Beziehungen untereinander. Satzbauplan 1 gestattet die Beschreibung des Verhaltens eines Systems innerhalb dieses Anwendungsbereichs, d. h. die beobachtbaren Interaktionen zwischen Instanzen der Konzepte [Ort97].

3.3 Eine Terminologie für die fachliche Beschreibung von Komponenten

In den vergangenen Jahren wird zur Bezeichnung semantischer Modelle von Themenbereichen vielfach der Begriff der *Ontologie* herangezogen. Ursprünglich aus der Philosophie stammend bezeichnet der Begriff „Ontologie“ die *Lehre vom Sein*. In der Informatik wird er jedoch uneinheitlich unter zwei unterschiedlichen Bedeutungen verwendet. Zum einen bezeichnet er verbreitet ein Vokabular für die Repräsentation von Wissen und die dieser Repräsentation zugrunde liegende Konzeptualisierung, zum anderen bezieht er sich auf Wissen über einen Anwendungsbereich selbst [CJB99]. In unserer Arbeit zur Terminologieverwaltung berücksichtigen wir beide Bedeutungen:

- Auf der *Konzeptualisierungsebene* spezifizieren wir ein allgemeines, domänenunabhängiges Vokabular für die Repräsentation von (normsprachlichen) Aussagen.
- Basierend auf der Konzeptualisierungsebene definiert die *Wissensebene* Wissen über einen betrachteten Anwendungsbereich.

3.4 Konzeptualisierungsebene

Die Konzeptualisierungsebene leitet sich aus den drei in Abschnitt 3.2 vorgestellten Satzbauplänen ab. Abbildung 3 zeigt ein UML Klassendiagramm, das die konzeptuelle Struktur unserer Terminologie spezifiziert. Bezüglich der betrachteten Wortarten unterscheiden wir ausgehend von einer generellen Klasse abstrakter *Wörter* zunächst *Partikel* (Strukturwörter) und *Prädikatoren* (Fachwörter, die einem Gegenstand zu- oder abgesprochen werden). Bei den Partikeln

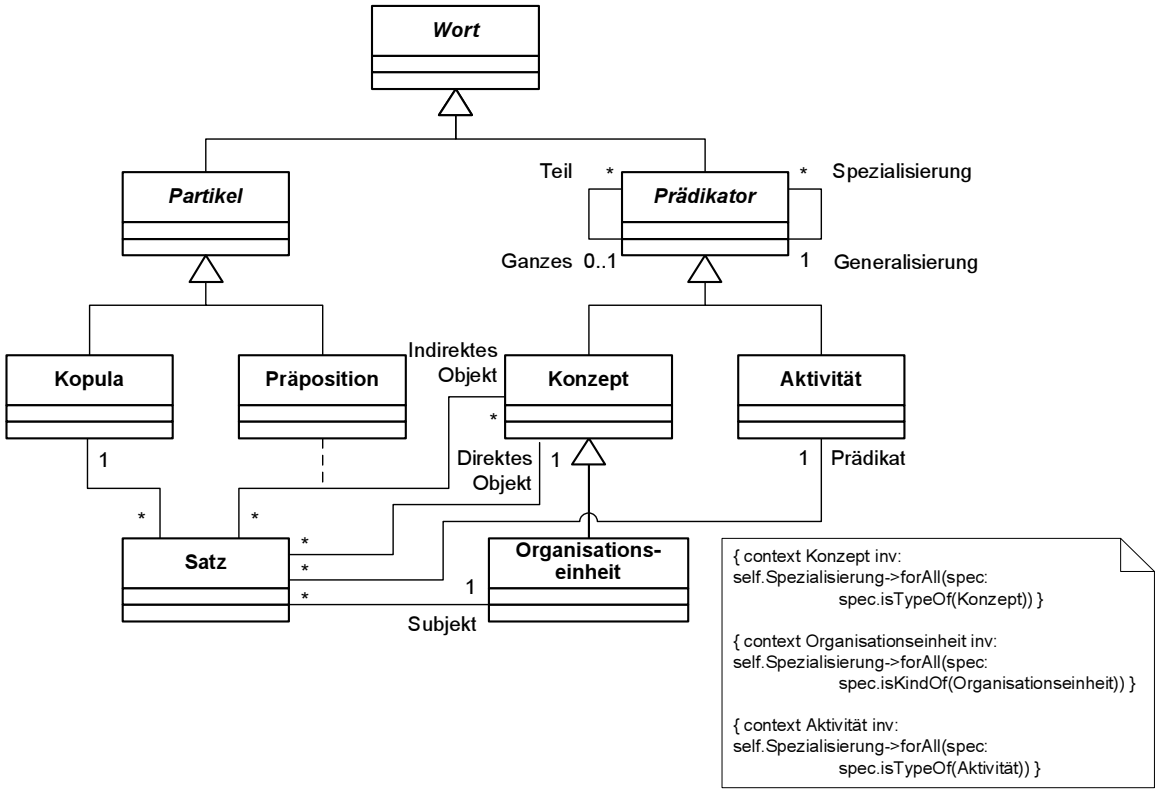

Abbildung 3: Konzeptualisierung der Terminologie

betrachten wir *Kopulae* wie z. B. π („tut“) und *Präpositionen* wie z. B. „zu“ oder „mit“. Auf Seiten der abstrakten Klasse der Prädikatoren unterscheiden wir zwischen *Aktivitäten* wie z. B. „erzeugen“ oder „modifizieren“ und *Konzepten* wie z. B. „Auftrag“, auf die Aktivitäten einwirken können. *Organisationseinheiten* wie z. B. „Angestellter“ sind spezielle Arten von Konzepten; sie können nicht nur Gegenstand der Ausführung von Aktivitäten sein, sondern diese auch selbst ausführen. Prädikatoren können in einer Generalisierungshierarchie angeordnet werden, wobei allerdings eine Instanz eines Prädikators nur durch Instanzen derselben Klasse oder einer Unterklasse spezialisiert werden kann (vgl. hierzu auch die OCL-Ausdrücke in Abbildung 3). Dies bedeutet insbesondere, dass eine Organisationseinheit nicht durch ein Konzept spezialisiert werden darf. Neben Generalisierungsbeziehungen gestattet die Konzeptualisierungsebene auch die Definition von Ganzes-Teile-Beziehungen zwischen Prädikatoren.

Sätze sind syntaktische Strukturen für die Repräsentation von Aussagen. Sie entsprechen den syntaktischen Konventionen, die durch den in Abschnitt 3.2 genannten Satzbauplan 1 definiert werden. Folglich besteht ein Satz aus einer Organisationseinheit in der Subjektstellung, einer Kopula und einer Aktivität in der Stellung des Prädikats sowie Konzepten als direkte und indirekte Objekte. Indirekte Objekte sind mit dem Prädikat über Präpositionen verbunden.

3.5 Wissensebene

Die Wissensebene definiert Taxonomien von Konzepten und Aktivitäten, die für die Konstruktion von Sätzen innerhalb eines Anwendungsbereichs genutzt werden können. Diese Taxonomien definieren z. B., dass „Auftrag“ eine „Transaktion“ ist und dass „Produktionsauftrag“ und „Kundenauftrag“ Spezialisierungen ($\leq$) von „Auftrag“ sind. Auf ähnliche Weise können „er-

zeugen“, „ändern“ und „löschen“ als Spezialisierungen der Aktivität „modifizieren“ definiert werden. Neben diesen Taxonomien definiert die Wissensebene auch semantische Restriktionen bzgl. der Kombination von Konzepten und Aktivitäten in Sätzen. Diese Restriktionen verbieten z. B. die Konstruktion von Sätzen wie „Kunde (tut) erzeugen Angestellter“. Sie sind durch sogenannte *Kernsätze*, die Beispiele „korrekter“ Sätze geben, sowie eine Regel für die Ableitung neuer Sätze aus diesen Kernsätzen definiert: Ein neuer Satz darf nur dann abgeleitet werden, wenn er einen Kernsatz spezialisiert und gleichzeitig nicht gegen die durch speziellere Kernsätze spezifizierten Restriktionen verstößt. Die allgemeine Idee, die der Spezialisierungsbeziehung zwischen Sätzen zugrunde liegt, ist zu verlangen, dass alle Satzbestandteile (Subjekt, Prädikat sowie direkte und indirekte Objekte) des generelleren Satzes durch entsprechende Satzbestandteile des spezielleren Satzes spezialisiert werden. In der praktischen Anwendung dieses Ansatzes schränken wir die Spezialisierungsbeziehung jedoch auf die Betrachtung von Subjekt, Prädikat und direktem Objekt ein:

Definition 1 (Spezialisierungsbeziehung) *Seien $S_i = (S_{S_i}, P_{S_i}, O_{S_i})$, $i \in \{1, 2\}$ Sätze, wobei S_{S_i} das Subjekt, P_{S_i} das Prädikat und O_{S_i} das direkte Objekt des Satzes S_i repräsentieren. Dann gilt:*

$$S_1 \text{ ist eine Spezialisierung von } S_2 \ (S_1 \leq S_2)$$
$$\iff \quad S_{S_1} \leq S_{S_2} \wedge P_{S_1} \leq P_{S_2} \wedge O_{S_1} \leq O_{S_2},$$

wobei sich die Spezialisierungsbeziehungen zwischen Subjekten, Prädikaten und direkten Objekten aus der Aktivitäts- bzw. der Konzepttaxonomie ergeben.

Damit ein abgeleiteter Satz gültig ist, ist es nicht ausreichend, einen Kernsatz zu spezialisieren. Wir verlangen zusätzlich, dass der abgeleitete Satz auch die Resktriktionen speziellerer Kernsätze respektiert:

Definition 2 (Gültige Spezialisierung) *Sei CS die Menge der Kernsätze und $S = (S_S, P_S, O_S)$ ein Satz. Dann gilt:*

$$S \text{ ist ein gültiger Satz bzgl. } CS$$
$$\begin{aligned} \iff \quad & \exists S_C \in CS : \ S \leq S_C \\ \wedge \quad & \forall S'_C \in CS, \ S'_C \leq S_C, S'_C \neq S : \\ & \neg(S_S \leq S_{S'_C} \vee P_S \leq P_{S'_C} \vee O_S \leq O_{S'_C}), \end{aligned}$$

d. h. der abgeleitete Satz ist nicht teilweise spezieller als ein Kernsatz S'_C, der spezieller als der Kernsatz S_C ist.

Ein neuer Satz stellt eine gültige Spezialisierung eines Kernsatzes dar, genau dann wenn es keinen spezielleren Kernsatz gibt, der die Benutzung des Subjekts, Prädikats oder direkten Objekts weiter einschränkt. Ein Beispiel soll diese Gültigkeitsregel erläutern. Nehmen wir an, dass der Satz „Organisationseinheit (tut) tun etwas“ der einzige Kernsatz ist. Vorausgesetzt, dass die Konzept- und Aktivitätstaxonomien entsprechend definiert sind, ließen sich dann die Sätze „Angestellter (tut) planen Produktionsauftrag“ und „Angestellter (tut) planen Rechnung“ ableiten. Um auszudrücken, dass wir nur die Planung von Aufträgen, nicht aber von Rechnungen gestattet wollen, können wir den obigen Kernsatz durch einen weiteren Kernsatz „Organisationseinheit (tut) planen Auftrag“ spezialisieren. Nun stellt „Angestellter (tut) planen Rechnung“ keine gültige Spezialisierung des ersten Kernsatzes dar, da $\neg((Organisationseinheit \leq$

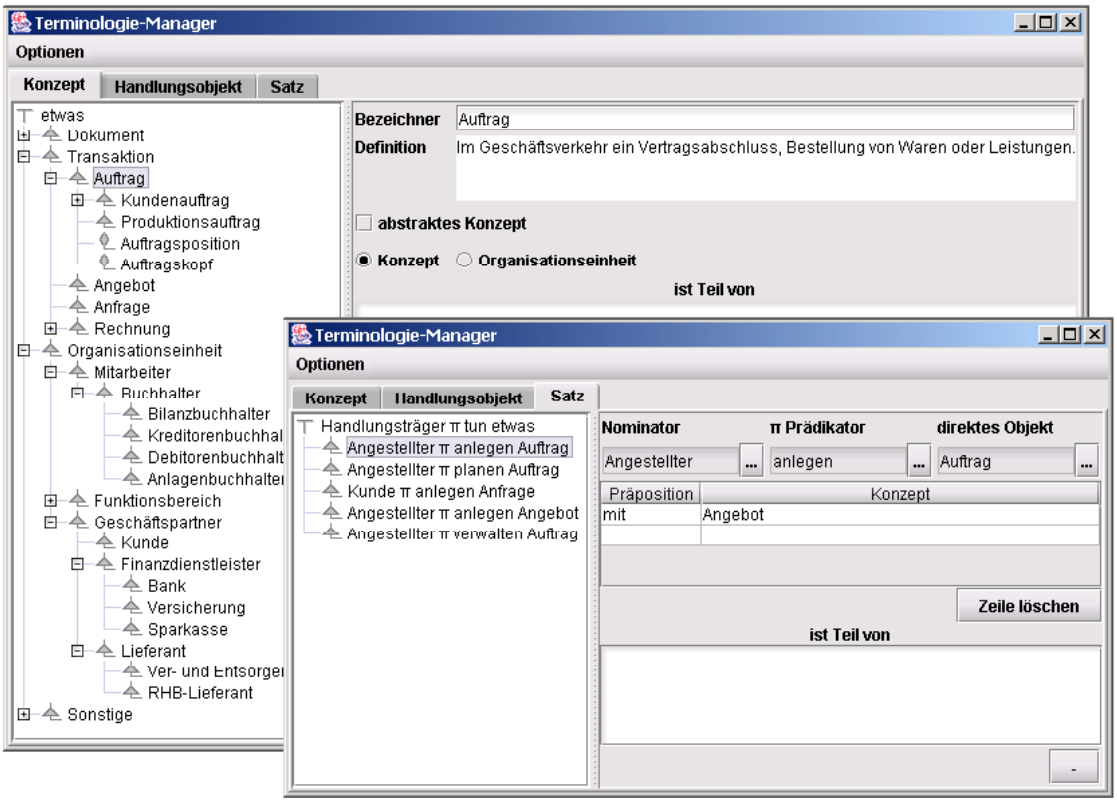

Abbildung 4: *Terminologie-Manager* zur Verwaltung von Fachterminologien

$Organisationseinheit) \vee (planen \leq planen) \vee (Rechnung \leq Auftrag)) = false$ (vgl. den zweiten Teil von Definition 2). Es ist offensichtlich, dass dieser Satz ebensowenig eine gültige Spezialisierung des zweiten Kernsatzes darstellt.

3.6 Werkzeugunterstützung

Zur Unterstützung der Aufgaben eines Normierungsgremiums haben wir mit dem *Terminologie-Manager* ein Werkzeug für die Verwaltung von Prädikatoren einerseits sowie von Kernsätzen andererseits prototypisch entwickelt (siehe Abbildung 4). Daneben ist ein Algorithmus zur Berechnung gültiger Spezialisierungen von Kernsätzen gemäß Definition 2 entwickelt und in Form eines Dialogs zur Ableitung von Sätzen implementiert worden. Dieser Dialog unterstützt die Aufgaben von Anbietern bzw. Verwendern, die in ihren Komponentenbeschreibungen bzw. Geschäftsprozessmodellen Bezug auf eine Fachterminologie nehmen.

4 Komponentenbeschreibungen mit CDL

4.1 Das CDL-Komponentenmodell

Voraussetzung für die Definition einer Komponentenbeschreibungssprache ist die Existenz eines (abstrakten) Komponentenmodells. Das Komponentenmodell der CDL stellt eine einheitliche Abstraktion von den Komponentenmodellen der Komponententechnologien COM, EJB und

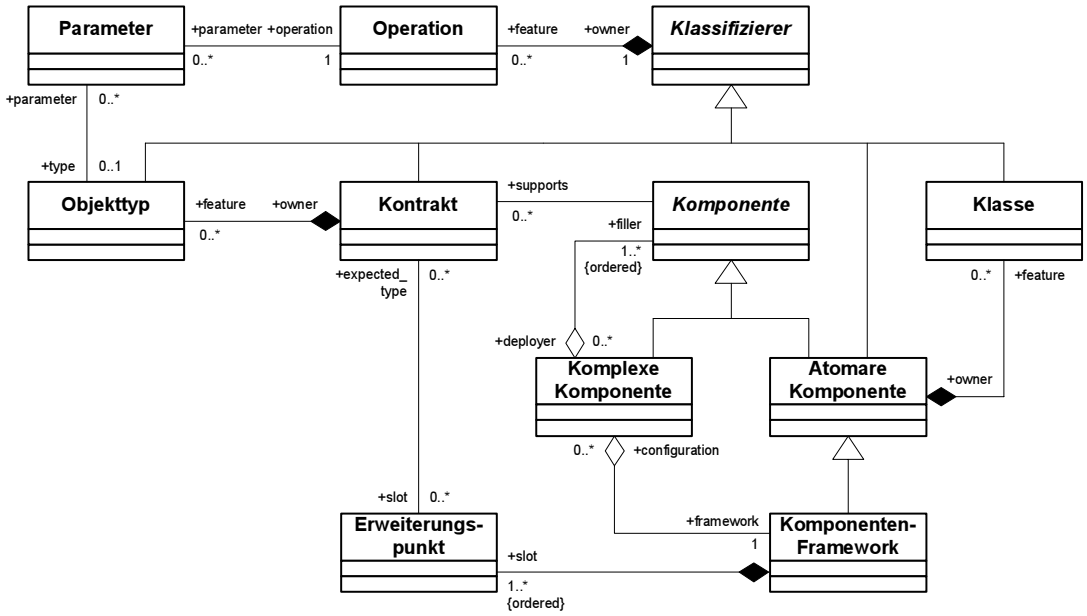

Abbildung 5: CDL-Komponentenmodell

CCM dar (vgl. auch [TR01]). Es ist in Abbildung 5 in Form eines Klassendiagramms graphisch dargestellt.

Zentrales Element des CDL-Komponentenmodells ist der abstrakte Begriff der Komponente, von der sich die spezielleren Varianten atomare Komponente und komplexe Komponente ableiten. Atomare Komponenten können eine *direkte Schnittstelle* und – über die möglicherweise von ihr realisierten Klassen von Geschäftsobjekten – *indirekte Schnittstellen* anbieten, da sowohl atomare Komponenten als auch Klassen Klassifizierer darstellen und somit eine Menge von Operationen als Schnittstelle exportieren. Instanzen einer Klasse (die Geschäftsobjekte) werden über die direkte Schnittstelle einer atomaren Komponente erzeugt und lassen sich über ihre eigene Schnittstelle (die Teil der indirekten Schnittstelle der atomaren Komponente ist) ansprechen. Komponenten-Frameworks sind spezielle atomare Komponenten, die sich dadurch auszeichnen, dass sie Erweiterungspunkte („hot spots“) definieren, über die sie im Sinne einer Komposition mit anderen Komponenten verbunden werden können (atomare Komponenten, die keine Komponenten-Frameworks sind, werden – falls eine explizite Unterscheidung erforderlich ist – auch als *geschlossene atomare Komponenten* bezeichnet). Komponenten-Frameworks stellen damit die Basis für die Erstellung komplexer Komponenten, d. h. durch Komposition mehrerer Komponenten hervorgegangene Komponenten, dar. Eine komplexe Komponente besteht aus einem Komponenten-Framework und einer Menge von Komponenten, mit denen die Erweiterungspunkte des Komponenten-Frameworks belegt werden. Anders als eine atomare Komponente definiert eine komplexe Komponente keine eigene (direkte und/oder indirekte) Schnittstelle, sondern exportiert die Schnittstellen des verwendeten Komponenten-Frameworks.

Die Belegung der Erweiterungspunkte eines Komponenten-Frameworks wird durch Typisierung mit Kontrakten kontrolliert. Kontrakte repräsentieren ein zu den Komponenten auf Seiten der Implementierung analoges Konzept auf der Vertragsebene: Ähnlich wie atomare Komponenten (und indirekt auch komplexe Komponenten) eine direkte Schnittstelle sowie (über Klassen) indirekte Schnittstellen definieren, spezifizieren Kontrakte neben einer direkten Schnittstelle mittels sogenannter Objekttypen indirekte Schnittstellen. Objekttypen repräsentieren da-

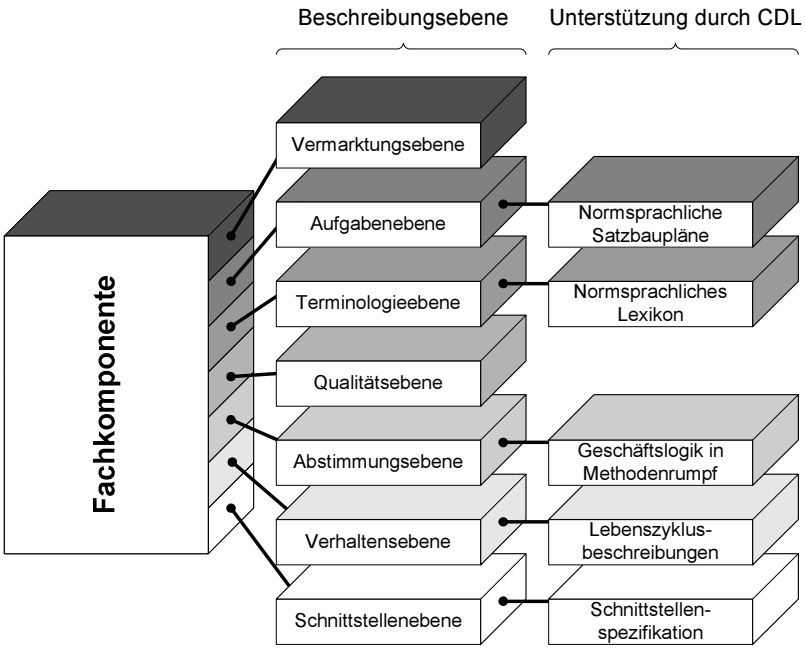

Abbildung 6: Berücksichtigung der Beschreibungsebenen nach [ABC+02] durch CDL

bei Typen von Geschäftsobjekten, die durch Klassen einer atomaren Komponente realisiert und neben Basisdatentypen wie `int`, `float` und `String` als Typen der Parameter von Operationen verwendet werden können.[1] Eine Komponente stellt eine gültige Belegung eines Erweiterungspunktes dar, wenn sie den vom Erweiterungspunkt vorgegebenen Kontrakt unterstützt, d. h. wenn sie mindestens die im Kontrakt geforderte direkte Schnittstelle anbietet und darüber hinaus noch Klassen realisiert, die die Objekttypen des Kontrakts implementieren. Dabei kann eine Klasse mehrere Objekttypen gleichzeitig implementieren.

4.2 CDL – Component Description Language

Die Entwicklung der CDL wurde durch die Anforderung motiviert, fachlich orientierte Beschreibungen bei hinreichend formaler Fundierung zu ermöglichen. Eine zentrale Annahme unserer Arbeit zur Beschreibung und Suche von Komponenten besteht darin, dass Komponenten bzw. ihre Geschäftsobjekte ihre Dienste nicht gleichförmig zur Verfügung stellen (*non-uniform service availability* [Nie95]), sondern oft eine spezifische Geschäftslogik implementieren. So muss ein Auftrag z. B. freigegeben werden, bevor er ausgeführt werden kann. Aus diesem Grund bestand ein Anspruch an die CDL darin, neben strukturellen Merkmalen von Komponenten auch deren Verhalten beschreiben zu können. Im Memorandum zur vereinheitlichen Spezifikation von Fachkomponenten [ABC+02] wurden sieben Ebenen für die Beschreibung von Fachkomponenten identifiziert. CDL berücksichtigt die fünf in Abbildung 6 dargestellten Beschreibungsebenen. Weitere, durchaus relevante Informationen wie z. B. zu Hersteller, erforderlicher Plattform, Lizenzierung, Quality of Service oder Performance, die auf der Vermarktungsebene bzw. Qualitätsebene einzuordnen sind, werden durch CDL nicht näher betrachtet.

[1] Aus praktischen Erwägungen lassen wir in CDL neben Objekttypen auch den mengenwertigen Typ `Set<·>`, der in Anlehnung an C++-Templates mit einem Elementtyp parametrisiert werden kann, und Klassen für die Typisierung von Parametern bei der Spezifikation von Komponenten zu.

Wie bereits aus dem CDL-Komponentenmodell hervorgeht, unterscheidet CDL zwischen Komponenten- und Kontraktbeschreibungen, die wie folgt charakterisiert werden können. Komponentenbeschreibungen definieren eine obere Grenze der Menge von Interaktionssequenzen, an der die Komponente (evtl. mittels ihrer Geschäftsobjekte) teilnehmen kann. Diese Semantik ist aus Sicherheitsbetrachtungen heraus interessant: Eine Komponente geht nicht mehr Interaktionen ein, als durch ihre Beschreibung zugesichert wird. Demgegenüber spezifizieren Kontraktbeschreibungen eine untere Grenze der Menge erwarteter Interaktionssequenzen und damit die erwartete Lebendigkeit einer Komponente. Dieses Semantik stellt sicher, dass eine Komponente, die einen bestimmten Kontrakt unterstützt, auch mindestens die erforderlichen Dienste anbietet. Da CDL jedoch explizit Unterspezifikationen von Komponenten und Kontrakten unterstützt (z. B. durch die Möglichkeit, nicht-deterministische Entscheidungen zu spezifizieren), darf die vorangegangene Charakterisierung von Komponenten- und Kontraktbeschreibungen nicht strikt (z. B. im Sinne einer Garantie des Komponentenherstellers) interpretiert werden. Sie stellt stattdessen lediglich eine Art Richtschnur dar, an der sich „gute“ Beschreibungen ausrichten.

Komponentenbeschreibungen werden durch das Schlüsselwort `component` eingeleitet. Sie können Beschreibungen einer beliebigen Anzahl von Klassen enthalten, die die durch die Komponente implementierten Typen von Geschäftsobjekten spezifizieren und durch das Schlüsselwort `class` gekennzeichnet sind (siehe Listing 1). Die direkte Schnittstelle einer Komponente besteht aus den Operationen, die die Komponente direkt, d. h. nicht über ihre Geschäftsobjekte, anbietet, während die indirekte Schnittstelle einer Komponente aus ihren Klassen und den von ihnen angebotenen Operationen besteht. Parameter von Operationen sind gerichtet und mit Objekttypen, Klassen, Basisdatentypen wie `int`, `float` und `String` oder dem mengenwertigen Typ `Set<·>` typisiert. Die Richtung eines Parameters wird durch die Schlüsselworte `in` für Eingabeparameter, `out` für Ausgabeparameter und `inout` für Ein- und Ausgabeparameter gekennzeichnet. Die Beschreibung von Komponenten-Frameworks und komplexen Komponenten wird durch spezielle Schlüsselworte für Erweiterungspunkte und die Komposition von Komponenten ermöglicht. Die CDL-Syntax für die Beschreibung von Kontrakten unterscheidet sich kaum von der für die Beschreibung geschlossener atomarer Komponenten: lediglich die für die Beschreibung von Komponenten verwendeten Schlüsselwörter `component` und `class` müssen bei der Beschreibung von Kontrakten durch die Schlüsselwörter `contract` und `objecttype` ersetzt werden.

Neben diesen syntaktischen Elementen zur Beschreibung struktureller Eigenschaften von Komponenten (und Kontrakten) bietet CDL auch Beschreibungselemente für die Spezifikation des Verhaltens einer Komponente und ihrer Klassen. Zu diesem Zweck greifen wir auf den Grundgedanken zurück, der der Kommunikation im π-Kalkül zugrunde liegt. Das Senden eines Operationsaufrufs wird durch den Bezeichner der Operation, gefolgt von einem „!“ gekennzeichnet, die Bereitschaft zum Empfang eines solchen Aufrufs durch ein dem Bezeichner folgendes „?“. Eine Operation wird ausgeführt, wenn das Senden eines Operationsaufrufs durch eine Komponente mit der Empfangsbereitschaft einer entsprechenden Operation einer Zielkomponente zusammenfällt. Dieser dualen Betrachtung des Aufrufs von Operationen folgend basiert die Beschreibung des Verhaltens von Komponenten und Klassen mit der CDL auf der Idee, die Spezifikation des *aktiven Verhaltens*, d. h. die ausgelösten Operationsaufrufe, von der Spezifikation des *passiven Verhaltens*, d. h. der Bereitschaft zum Empfang solcher Aufrufe, zu trennen. Dazu spezifizieren wir das aktive Verhalten in Operationsrümpfen, während das passive Verhalten in speziellen Verhaltensbeschreibungen dokumentiert wird.

Listing 1: Komponente `Vertrieb`

```
component Vertrieb {
   class Auftrag {
      plan() [Vertriebsangestellter terminplanen Auftrag] {
         return; };
      execute() [Vertriebsangestellter ausführen Auftrag] {
         return; };
      updateTerms() [Vertriebsangestellter ändern Auftrag] {
         return; };
      cancel() [Vertriebsangestellter stornieren Auftrag] {
         return; };
       ... /* Beschreibung weiterer Operationen der Klasse Auftrag */
      created()     =   initialized();
      initialized() =   plan?().planned()
                      + cancel?().cancelled();
      planned()     =   execute?().executed()
                      + updateTerms?(terms).initialized()
                      + cancel?().cancelled();
      executed()    =   ();
      cancelled     =   ();
   };
   class Angebot {
      ... /* Beschreibung der Klasse Angebot */
   };
   createOrder(in offer: Vertrieb.Angebot, out order: Vertrieb.Auftrag)
      [Vertriebsangestellter anlegen Auftrag mit-Angebot] {
         new (Vertrieb.Auftrag order);
         return(order);
   };
   createOffer(out offer: Vertrieb.Angebot)
      [Vertriebsangestellter anlegen Angebot] {
         new (Vertrieb.Angebot order);
         return(offer);
   };
   created()     =    initialized();
   initialized() =    createOrder?(offer, order).initialized()
                    + createOffer?(offer).initialized();
};
```

Für die Beschreibung des aktiven Verhaltens im Rumpf einer Operation stellt CDL Konstrukte für den Aufruf von Operationen, die Erzeugung von Referenzen auf neue bzw. bereits existierende Geschäftsobjekte, (nicht-deterministische) Entscheidungen, Wiederholungsblöcke und die Rückgabe von Parameterwerten zur Verfügung. Die Beschreibung des aktiven Verhaltens einer Komponente in Operationsrümpfen hat primär illustrierenden Charakter. Sie wird im praktischen Einsatz oftmals als starke Unterspezifikation des tatsächlichen Verhaltens angegeben werden und eignet sich insbesondere dazu, um verwendete Algorithmen zu dokumentieren und Abhängigkeiten von anderen Komponenten explizit darzustellen.

Die Verhaltensbeschreibungen zur Spezifizierung des passiven Verhaltens einer Komponente bzw. einer Klasse basieren auf *Protokollmaschinen* (*Protocol State Machines*), einer Variante der UML State Machines, die für die Spezifikation der Lebenszyklen von Objekten eingesetzt wird [OMG01]. Zustände einer solchen Protokollmaschine beschreiben dabei sinnvolle Ab-

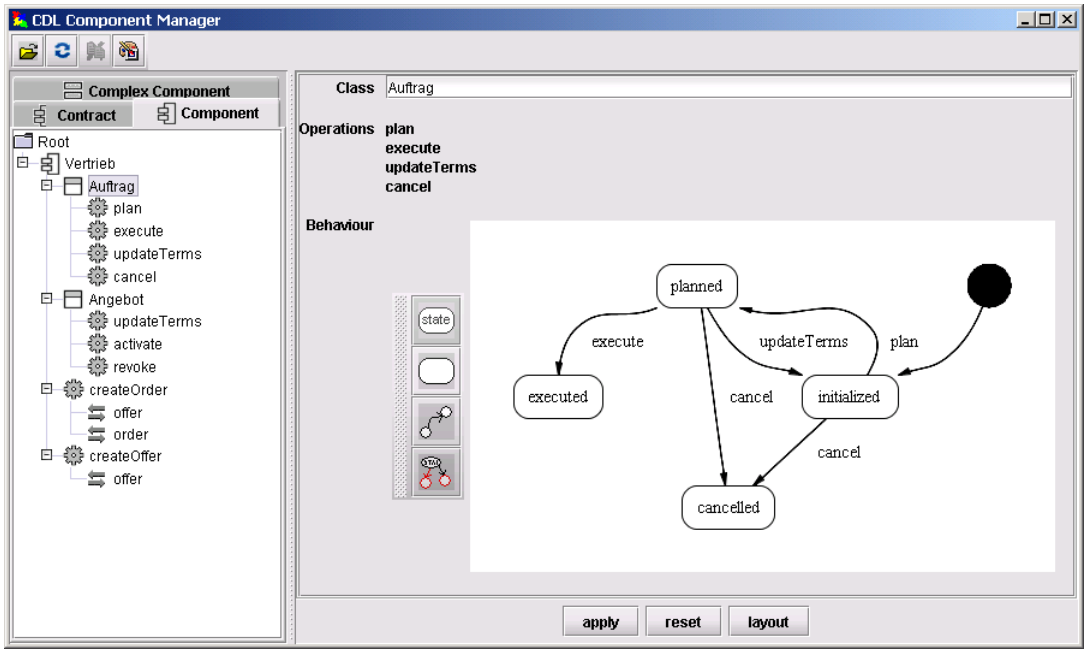

Abbildung 7: *CDL Component Manager* zur Verwaltung von Komponentenbeschreibungen

straktionen der tatsächlichen Zustände, in denen sich eine Komponente oder ein Geschäftsobjekt befinden kann, während Transitionen die Operationsaufrufe darstellen, die eine Komponente oder Geschäftsobjekt von einem Zustand in einen Folgezustand überführen. Beschreibungen des Verhaltens, das in einem Zustand möglich ist, dürfen nur die Bereitschaft zum Empfang von Operationsaufrufen, d. h. das passive Verhalten der Komponente oder des Geschäftsobjekts spezifizieren.

Als illustrierendes Beispiel für Komponentenbeschreibungen in CDL zeigt Listing 1 eine Komponente namens `Vertrieb`, die einfache Dienste zur Verwaltung von Angeboten und Aufträgen anbietet. Die fachliche Semantik dieser Dienste ist gemäß Abschnitt 3 mittels normsprachlicher Sätze definiert (siehe eckige Klammern im Listing). Die Komponente bietet über die beiden Klassen `Auftrag` und `Angebot` zwei indirekte Schnittstellen an. Die Klasse `Auftrag` bietet in ihrer Schnittstelle Dienste zum Einplanen, Ausführen, Ändern und Stornieren des Auftrages an. Die anschließend spezifizierte Verhaltensbeschreibung ist als Protokollmaschine in Abbildung 7 im rechten Teil dargestellt. Die direkte Schnittstelle der Komponente `Vertrieb` beschränkt sich auf das Erzeugen der Geschäftsobjekte. Ihre Verhaltensbeschreibung erlaubt die beliebige Erzeugung von Aufträgen und Angeboten.

4.3 Werkzeugunterstützung

Zur Unterstützung der Aufgaben eines Marktplatzbetreibers oder Komponentenherstellers haben wir mit dem *CDL Component Manager* ein Werkzeug für die Verwaltung von Komponentenbeschreibungen prototypisch entwickelt (siehe Abbildung 7). Der *CDL Component Manager* gestattet die CDL-konforme Beschreibung von Komponenten und Kontrakten und stellt Verhaltensbescheibungen graphisch als editierbare UML State Machines dar.

Auf technischer Ebene setzen wir für die Verwaltung von CDL-Komponentenbeschreibungen ein MOF-basiertes Metadaten-Repository [HT01] ein, auf das wir über ein speziel-

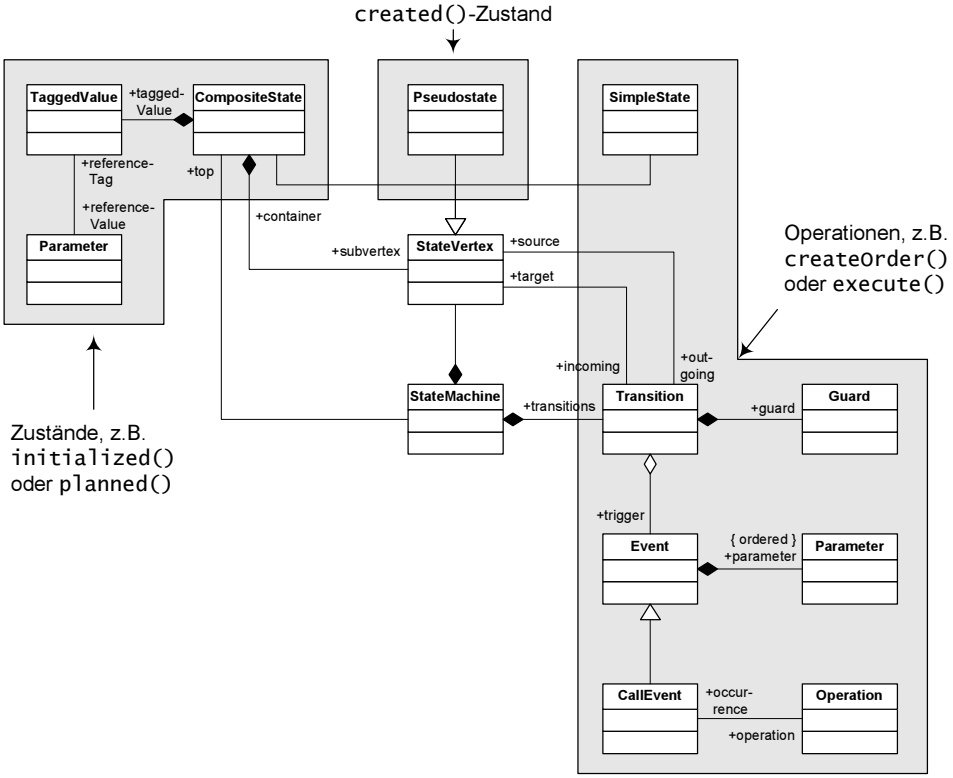

Abbildung 8: Abbildung der CDL-Verhaltensbeschreibungen auf die UML

les Komponenten-API zugreifen (vgl. auch Abbildung 2). Zu diesem Zweck haben wir die Sprachelemente der CDL auf das Metamodell der UML abgebildet, das die konzeptionelle Struktur unseres Komponenten-Repositories definiert. Während die Abbildung der strukturellen CDL-Konstrukte wie z. B. Komponenten, Klassen und Operationen durch Einsatz der UML-Erweiterungsmechanismen *Tagged Values* und *Stereotypen* relativ direkt vorgenommen werden kann, gestaltet sich die Abbildung der verhaltensbeschreibenden Elemente aufwändiger. Wir legen CDL-Verhaltensbeschreibungen als *StateMachines* in dem UML-Repository ab und repräsentieren die Zustände einer CDL-Verhaltensbeschreibung als *CompositeStates* einer solchen *StateMachine*. Die Verknüpfung von CDL-Zuständen wird dann über *Transitionen* zwischen *CompositeStates* modelliert. Sequenzen von Operationen, Alternativen und bedingte Ausführungen können durch Transitionen zwischen internen Zuständen innerhalb eines *CompositeState* abgebildet werden. Für die Repräsentation solcher interner Zustände verwenden wir *SimpleStates*, die einem *CompositeState* untergeordnet werden können. Die in Abbildung 8 skizzierte Abbildung von CDL-Verhaltensbeschreibungen auf das UML-Metamodell kann automatisiert durch einen Parser für CDL-Beschreibungen vorgenommen werden.

5 Geschäftsprozessorientierte Komponentensuche

5.1 Komponentensuche auf Basis von Geschäftsprozessmodellen

Seit den ersten Erfolgen beim Handel mit Komponenten für graphische Benutzungsoberflächen (z. B. *ActiveX-Controls* oder *JavaBeans*) über das Internet sind eine Reihe elektronischer

Marktplätze entstanden, über die Komponentenhersteller die von ihnen entwickelten System- und Fachkomponenten vertreiben können.[2] Ein wichtiger Erfolgsfaktor solcher Komponentenmärkte im Speziellen und der komponentenbasierten Softwareentwicklung durch Wiederverwendung im Allgemeinen ist die Fähigkeit, das Problem der gezielten Suche nach Komponenten zu lösen. Die Suche nach geeigneten Komponenten wird auf aktuellen Komponentenmärkten lediglich durch relativ simple Suchverfahren wie Stichwortsuche in Volltextbeschreibungen von Komponenten, hierarchische Klassifikation von Komponenten oder Feldsuche unterstützt, die kaum entscheidende Charakeristika von Software berücksichtigen (vgl. z. B. [Com02, Fla02, Sun02, Sof02, SAP02, Nom02]). Als Vorteil dieser Suchverfahren darf gelten, dass sie für einen Interessenten einfach zu verstehen und benutzen sind. Ihr Nachteil allerdings ist, dass ein Interessent nur sehr eingeschränkte Möglichkeiten hat, seine (fachlichen) Anforderungen an eine Komponente in den Suchvorgang einzubringen. Dem Interessenten fällt daher die Aufgabe zu, aus einer u. U. umfangreichen Menge gefundener Komponenten diejenige auszuwählen, die seine Anforderungen am besten erfüllt. Da Beschreibungen verfügbarer Komponenten in der Realität aktueller Komponentenmärkte jedoch insbesondere im Hinblick auf die angebotene Funktionalität zumeist uneinheitlich, unvollständig und missverständlich sind, wird der „manuelle“ Abgleich der Anforderungen mit den Leistungen gefundener Komponenten bzw. der direkte Vergleich gefundener Komponenten erheblich erschwert, wenn nicht gar faktisch verhindert. In Anbetracht des erwarteten Zuwachses an angebotenen Komponenten lässt sich zusammenfassend feststellen, dass die Kombination unpräziser Suchverfahren mit nur eingeschränkt aussagekräftigen Komponentenbeschreibungen (zukünftig) zu erheblichen Problemen führen bzw. die Entwicklung von Komponentenmärkten behindern wird.

Bei der Betrachtung der Spezifikation fachlicher Anforderungen einerseits und der Charakteristika von (Fach-)Komponenten andererseits lässt sich folgendes feststellen:

1. Seit den 90er Jahren hat sich bei der Realisierung von Softwareprojekten ein Wandel von einer bis dato primär daten- und funktionsorientierten Sicht hin zu einer verstärkt prozessorientierten Ausrichtung vollzogen. Anforderungen werden daher vielfach in Form von Geschäftsprozessmodellen spezifiziert (z. B. mittels ereignisgesteuerter Prozessketten (EPKs) [KNS92] oder Aktivitätsdiagrammen [OMG01]). Die in diesen Modellen erfassten Informationen finden derzeit keine ausreichende Berücksichtigung bei der Komponentensuche und damit bei der komponentenbasierten Softwareentwicklung.

2. Komponenten implementieren häufig eine bestimmte Geschäftslogik, die die Verfügbarkeit von Diensten und damit die Einsetzbarkeit der Komponente einschränkt (vgl. auch Abschnitt 4.2). Ähnlich wie bei der Einführung von Standardsoftware sind infolgedessen die Geschäftsprozesse eines Wiederverwenders ggf. an die eingesetzten Komponenten anzupassen.

Die Notwendigkeit der Anpassung von Geschäftsprozessen an die eingesetzte Software erscheint inakzeptabel, da gerade die Komponentensoftware den Anspruch erhebt, durch ihre Flexibilität und die Vielfalt verfügbarer Komponenten die Anpassung der Software an spezifische Anforderungen zu ermöglichen [Has02]. Eine stärkere Berücksichtigung der in Form von Geschäftsprozessmodellen spezifizierten fachlichen Anforderungen bei der Komponentensuche kann dazu beitragen, dieses Problem zu entschärfen.

[2] Eine Auswahl aktueller Komponentenmärkte findet sich in [Tec02].

In unserer Arbeit zur geschäftsprozessorientierten Komponentensuche konzentrieren wir uns daher auf fachliche Anforderungen, die in Geschäftsprozessmodellen spezifiziert sind. Dieser Ansatz zur Komponentensuche kann aufgrund seiner starken Differenzierung als Ergänzung einfacherer Suchverfahren wie der Klassifikation von Komponenten (z. B. zur Eingrenzung der Anwendungsdomäne) oder der Feldsuche (z. B. zur Wahl der Komponententechnologie) verstanden werden.

5.2 Komponentensuche als Subtyping-Problem

Komponentensuche setzt sich mit der Frage auseinander, ob eine Komponente in einem spezifischem Kontext (wieder-)verwendet werden kann, der zur Zeit ihrer Entwicklung nicht in allen Details bekannt gewesen ist. Im objektorientierten Paradigma ist der Begriff der (Wieder-)Verwendbarkeit eng mit der Frage nach der *Substituierbarkeit* von Klassen bzw. ihren Objekten verknüpft. In beiden Fällen ist für eine gegebene Schnittstelle eine geeignete Implementierung zu finden. Während bei Wiederverwendungsproblemen im Allgemeinen noch keine solche Implementierung vorhanden ist, geht es bei der Frage der Substitutierbarkeit darum, eine vorhandene Implementierung durch eine andere zu ersetzen. Dabei wird verlangt, dass die Ersetzung dieser Implementierung für einen Nutzer der Schnittstelle nicht feststellbar ist. Der Begriff der Substituierbarkeit wird gewöhnlich durch das Konzept des *Subtypings* formal gefasst, d. h. durch die Feststellung von Subtyp-Beziehungen zwischen Klassen bzw. den durch sie implementierten Typ. WEGNER und ZDONIK beschreiben das *Prinzip der Substituierbarkeit* wie folgt:

> »*An instance of a subtype can always be used in any context in which an instance of a supertype was expected.*« [WZ88]

Ein *Typ* kann dabei als Sammlung von Objekten betrachtet werden, denen gewisse gemeinsame, extern beobachtbare Eigenschaften innewohnen [Ame91]. Beim Subtyping wird allgemein die Fragestellung betrachtet, ob ein Typ die Eigenschaften eines anderen Typs aufweist (die er durch weitere, nicht konkurrierende Eigenschaften ergänzen darf). Ist dies der Fall, so wird der erstgenannte Typ als *Subtyp* des zweiten Typs bezeichnet, der dann einen *Supertyp* des ersten Typs darstellt. Die Eigenschaften eines Supertyps können also als Anforderungen betrachtet werden, die der Subtyp durch entsprechende Leistungen erfüllen muss. Eine solche Anforderung besteht z. B. darin, dass ein Subtyp alle Methoden des Supertyps anbietet (bzw. geeignete, z. B. durch ko- und kontravariante Veränderungen der Eingabe- und Ausgabeparametertypen erzeugte Varianten davon [Szy98]). Allerdings berücksichtigt dieser auf die Betrachtung von Signaturen ausgerichtete Begriff des Subtypings nicht das Verhalten eines Typs. So kann ein Typ eine Geschäftslogik definieren, die die zulässigen Sequenzen von Methodenaufrufen und damit die Wiederverwendbarkeit des Typs einschränkt. Daneben können zwei Dienste trotz gleicher Signatur ein abweichendes, im Extremfall sogar konträres Verhalten aufweisen. Da diese Einflüsse dazu führen können, dass eine bei bloßer Betrachtung von Signaturen bestehende Subtypeigenschaft bei zusätzlicher Berücksichtigung des Verhaltens verletzt wird, ergänzen das *verhaltensorientierte* bzw. das *zustandsbasierte Behavioural Subtyping* [Weh02] den Vergleich von Signaturen um die zusätzliche Betrachtung des Verhaltens von Typen.

Übertragen auf das Problem der geschäftsprozessorientierten Komponentensuche kann ein Geschäftsprozess, der in einem Unternehmen ausgeführt werden soll, vergleichbar mit einem Supertyp als eine Spezifikation verhaltensmäßiger Anforderungen betrachtet werden. Ziel der

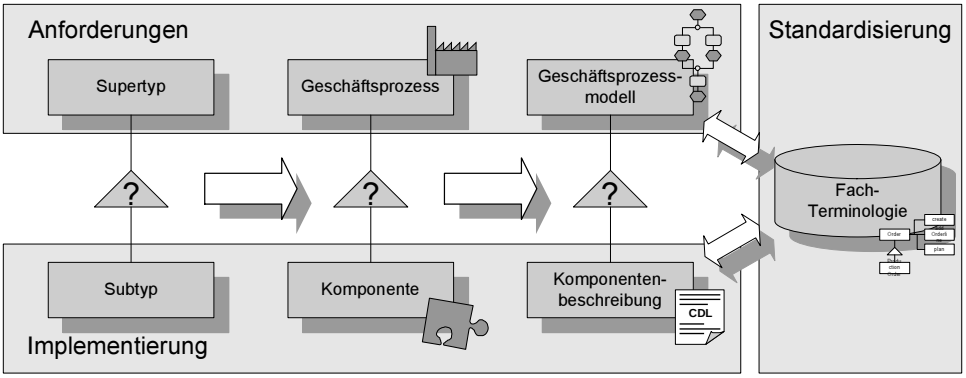

Abbildung 9: Geschäftsprozessorientierte Komponentensuche als Subtyping-Problem

geschäftsprozessorientierten Komponentensuche ist es nun, Komponenten zu finden, die vergleichbar mit einem Subtyp zumindest Teile dieser Anforderungen durch ihre Leistungen erfüllen. Grob gesagt erfüllt eine Komponente die Anforderungen eines Geschäftsprozesses, wenn sie

1. Dienste anbietet, die die Ausführung der einzelnen Aktivitäten des Geschäftsprozesses geeignet unterstützen,
2. den Aufruf dieser Dienste in der geforderten Reihenfolge ermöglicht und
3. die Verzweigungsstruktur des Geschäftsprozesses respektiert.

Auf einer höheren Abstraktionsebene schließlich beschreiben Geschäftsprozessmodelle Ablaufvarianten der in einem Unternehmen auszuführenden Geschäftsprozesse, während Komponentenbeschreibungen die von Komponenten angebotene Funktionalität spezifizieren, die für die Ausführung von (Teilen von) Geschäftsprozessen zur Verfügung steht. Um die fachliche Zuordnung von Dienste einer Komponente zu betrieblichen Aktivitäten eines Geschäftsprozesses zu ermöglichen, sehen wir die Verwendung einer standardisierten Fachterminologie gemäß Abschnitt 3 für die Spezifikation der fachlichen Semantik von Aktivitäten und Methoden vor. Abbildung 9 fasst unsere Interpretation der geschäftsprozessorientierten Komponentensuche als Subtyping-Problem graphisch zusammen.

5.3 Phasenmodell der geschäftsprozessorientierten Komponentensuche

Der Ansatz der semantischen Komponentensuche auf Basis von Geschäftsprozessmodellen soll im Folgenden anhand eines einfachen Phasenmodells erläutert werden. Dabei betrachten wir ein einfaches Beispiel zur Veranschaulichung des Ansatzes. Abbildung 10 zeigt Auszüge eines Geschäftsprozessmodells, der bereits vorgestellten CDL-Komponentenbeschreibung der Komponente `Vertrieb` sowie einer Terminologie.

1. *Angebotsphase*: *Anbieter* veröffentlichen CDL-Beschreibungen von Komponenten, die auf dem Marktplatz angeboten werden sollen, in einem Komponenten-Repository. Dabei ordnen Sie den einzelnen Diensten der Komponenten Sätze aus einer Terminologie nach Abschnitt 3 zu.

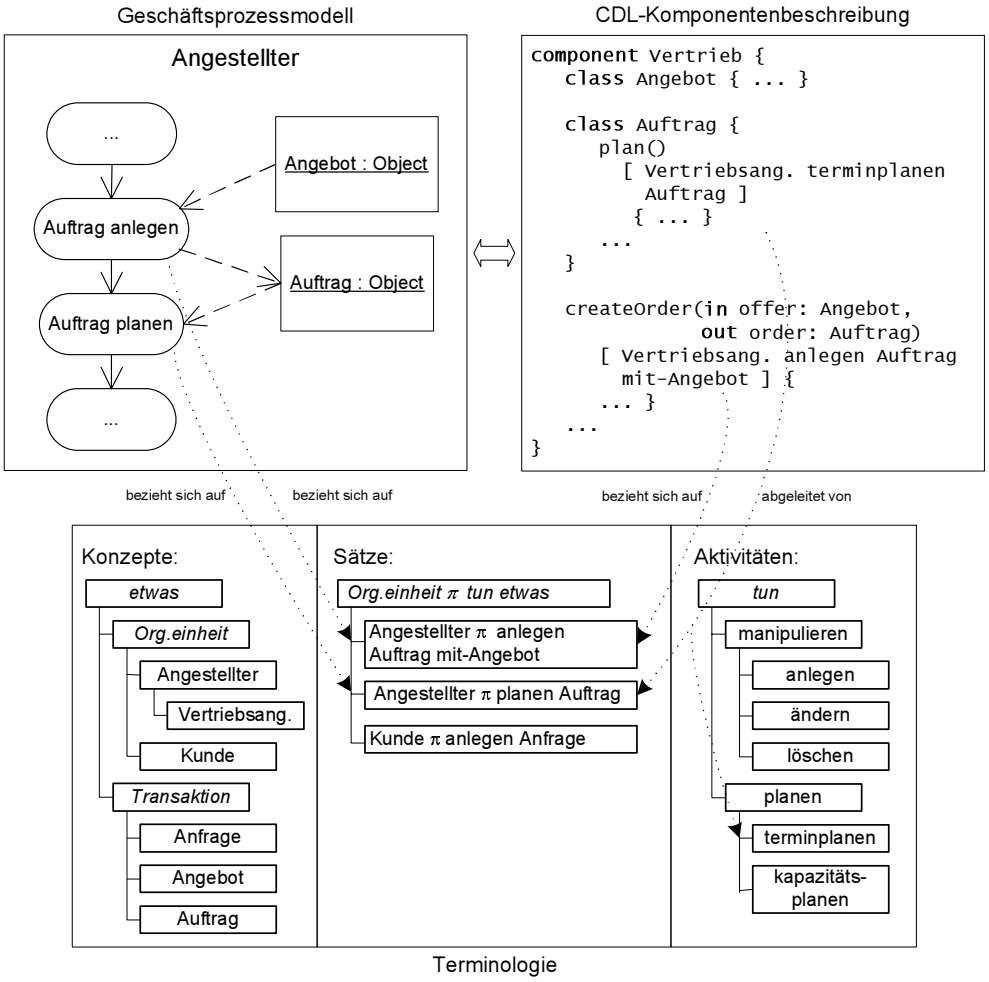

Abbildung 10: Komponentensuche auf Basis von Geschäftsprozessmodellen

In Abbildung 10 wird der Operation `createOrder` der Komponente `Vertrieb` die Semantik „Vertriebsangestellter (tut) anlegen Auftrag mit-Angebot“ zugeordnet (in eckigen Klammern). Dieser Satz stellt eine gültige Spezialisierung des in der Beispiel-Terminologie definierten Kernsatzes „Angestellter (tut) anlegen Auftrag mit-Angebot“ dar (vgl. Definition 2). Analog ist der Operation `plan` der enthaltenen Klasse `Auftrag` die Semantik „Vertriebsangestellter (tut) terminplanen Auftrag“ zugeordnet. Dieser Satz repräsentiert eine Spezialisierung des Satzes „Angestellter (tut) planen Auftrag“.

2. *Anforderungsphase*: *Verwender*, die am Erwerb von Komponenten interessiert sind, spezifizieren ihre Anforderungen gegenüber einem Broker in Form von Geschäftsprozessmodellen. In unserer Arbeit zur Komponentensuche konzentrieren wir uns auf ablauforganisatorische Aspekte, während Anforderungen an die Aufbauorganisation oder den Datenfluss etc. unberücksichtigt bleiben. Die Semantik der einzelnen Prozessschritte definieren wir analog zur Semantik von Diensten über Sätze aus einer Terminologie. Dies lässt sich damit begründen, dass sich die Schritte eines Geschäftsprozessmodells ebenfalls als sprachliche Handlungen mit Subjekt, Prädikat und direktem sowie indirekten Objekten verstehen lassen (vgl. [Tes02]).

   Die Aktivitäten des in Form eines UML Aktivitätsdiagramms spezifizierten Geschäftpro-

zessmodells aus Abbildung 10 sind direkt über Kernsätze aus der Terminologie definiert.

3. *Suchphase*: Für den Vergleich von Geschäftsprozessmodellen mit Komponentenbeschreibungen auf der *Protokollebene* (partielle Ordnung zwischen Diensten) greifen wir direkt auf den Ansatz des verhaltensorientierten Behavioural Subtypings zurück und untersuchen eine Subtyping-Relation zwischen Protokollautomaten. Wir verwenden dabei eine abgewandelte Form der Verfeinerungsrelation *Weak Simulation* [Mil80]. Eine genauere Einführung in den verfolgten Ansatz bietet [Tes01].

   Der Vergleich auf der *Semantikebene* (Bedeutung der Dienste) basiert auf der Idee, die fachliche Semantik von Schritten eines Geschäftsprozessmodells einerseits und Operationen von Komponenten andererseits miteinander zu vergleichen. In Anlehnung an das zustandsbasierte Behavioural Subtyping, bei dem die Semantik von Operationen über Vor- und Nachbedingungen beschrieben wird, verlangen wir für eine Übereinstimmung von Prozessschritten und Operationen auf der Semantikebene, dass der Satz, der die fachliche Semantik der Operation beschreibt, spezieller ist als der Satz, der die fachliche Semantik des Prozessschrittes spezifiziert. Dabei gehen jedoch abweichend von Definition 1 und 2 lediglich das Prädikat und das direkte Objekt in den Vergleich ein.

   In unserem Beispiel entspricht die Operation `createOrder` der Komponente dem initialen Prozessschritt des Geschäftsprozessmodells, da ihre Semantik – gemäß Definition 2 unter Berücksichtigung der erwähnten Einschränkung auf Prädikate und direkte Objekte – eine gültige Spezialisierung des Satzes „Angestellter (tut) anlegen Auftrag mit-Angebot“ (der fachlichen Semantik des Prozessschrittes) darstellt. Auf ähnliche Weise ist die Operation `plan` der Klasse `Auftrag` eine gültige Spezialisierung des Prozessschrittes „Angestellter (tut) planen Auftrag“, da die Aktivität „terminplanen“ als Spezialisierung der geforderten Aktivität „planen“ definiert ist.

4. *Präsentations- und Evaluierungsphase*: Für die Präsentation gefundener Komponenten bieten sich grundsätzlich zwei Ausrichtungen an. Zum einen können aufgrund des verfolgten normsprachlichen Ansatzes für die Beschreibung der fachlichen Semantik von Komponenten auf einfache Weise textuelle Repräsentationen der gefundenen Komponenten hergestellt werden, die auch von einem Fachexperten bewertet werden können. Zum anderen lässt sich das Verhalten von Komponenten und ihren Klassen in Form von Automatenmodellen graphisch veranschaulichen. Hierbei ist es möglich, die Übereinstimmung von angefordertem Geschäftsprozessmodell und Komponentenverhalten geeignet hervorzuheben.

5.4 Werkzeugunterstützung

Teile des im Projekt KOSOBAR verfolgten Ansatzes zur geschäftsprozessorientierten Komponentensuche sind im Rahmen einer Diplomarbeit in Form eines internetbasierten Brokers umgesetzt worden (siehe [Kei01]). Dabei beschränkte sich die Komponentensuche allerdings auf die Untersuchung von Behavioural-Subtyping-Beziehungen auf der Protokollebene. Aktuell wird diese Umsetzung im Rahmen eines Dissertationsvorhabens überarbeitet und um die Berücksichtigung der Semantikebene beim Vergleich von Geschäftsprozessmodellen und Komponentenbeschreibungen ergänzt.

6 Zusammenfassung

Neben der fachlichen und technischen Standardisierung ist für den Erfolg der komponentenbasierten Softwareentwicklung auch eine Unterstützung des Entwicklungsprozesses durch entsprechende Werkzeuge erforderlich. Im OFFIS-Projekt KOSOBAR ist auf der Grundlage eines Vorgehensmodells für die komponentenbasierte Softwareentwicklung eine Infrastruktur für die Suche und den Austausch von Fachkomponenten entwickelt worden. Das Vorgehensmodell berücksichtigt dabei insb. Entwicklungsprozesse, bei denen Akteure in verschiedenen Rollen über internetbasierte Komponentenmärkte interagieren. Die Infrastruktur unterstützt die beteiligten Akteure durch Werkzeuge zur Verwaltung von (domänenspezifischen) Terminologien, zur Beschreibung und Verwaltung von Komponentenbeschreibungen, zur Veröffentlichung dieser Beschreibungen in Komponenten-Repositories sowie durch Broker zur geschäftsprozessorientierten Komponentensuche. Die Beschreibung und Suche nach Fachkomponenten basiert auf dem Gedanken, die fachliche Semantik von Komponenten und Suchanfragen in Form normsprachlicher Sätze über einer standardisierten Terminologie zu definieren. Die im Projekt *KOSOBAR* verfolgten Ansätze im Bereich der Beschreibung von Fachkomponenten sind in Form von Workshop-Beiträgen [JT01, JT02] in die Arbeit des Arbeitskreises 5.10.3 *Komponentenorientierte betriebliche Anwendungssysteme* zur Vereinheitlichung der Spezifikation von Fachkomponenten eingeflossen.

Eine Evaluierung der vorgestellten Konzepte wird im Rahmen des in Abschnitt 5.4 angesprochenen Dissertationsvorhabens durchgeführt. Dazu werden Fachkomponenten (EJBs) für die kommunale Verwaltung im Kfz-Zulassungswesen mittels CDL unter Benutzung einer geeigneten Fachterminologie beschrieben und auf der Grundlage von Modellen kommunaler Verwaltungsprozesse gesucht.

Literatur

[ABC+02] J. Ackermann, F. Brinkop, S. Conrad, P. Fettke, A. Frick, E. Glistau, H. Jaekel, O. Kotlar, P. Loos, H. Mrech, E. Ortner, U. Raape, S. Overhage, S. Sahm, A. Schmietendorf, T. Teschke und K. Turowski. Vereinheitlichte Spezifikation von Fachkomponenten, Februar 2002. Memorandum des Arbeitskreises 5.10.3 Komponentenorientierte betriebliche Anwendungssysteme.

[Ame91] P. America. Designing an Object-Oriented Programming Language with Behavioural Subtyping. In J. W. de Bakker, W. P. de Roever und G. Rozenberg, Hrsg., *Foundations of Object-Oriented Languages, Proceedings of REX School/Workshop, Noordwijkerhout, Niederlande, 28. Mai - 1. Juni 1990*, Jgg. 489 von *Lecture Notes in Computer Science*, Seiten 60–90. Springer-Verlag, 1991.

[Büt95] W. Büttemeyer. *Wissenschaftstheorie für Informatiker*. Spektrum Akademischer Verlag, 1995.

[CJB99] B. Chandrasekaran, J. R. Josephson und V. R. Benjamins. What Are Ontologies, and Why Do We Need Them? *IEEE Intelligent Systems*, 14(1):20–26, 1999.

[Com02] ComponentSource, ComponentSource: The Definitive Source of Software Components. Elektronische Quelle: *http://www.componentsource.com/*, zuletzt besucht im Juni 2002.

[Fla02] Flashline.com, Flashline – Transforming Software Development. Elektronische Quelle: *http://www.flashline.com/*, zuletzt besucht im Juni 2002.

[FS96] N. E. Fuchs und R. Schwitter. Attempto Controlled English (ACE). In *The First International Workshop on Controlled Language Applications (CLAW 96)*, Katholieke Universiteit Leuven, 1996.

[Has02] W. Hasselbring. Component-Based Software Engineering. In S.K. Chang, Hrsg., *Handbook of Software Engineering and Knowledge Engineering*, Jgg. 2. World Scientific Publishing, 2002.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–583, 1969.

[HT01] A. Harren und H. Tapken. TODAY Open Repository: An Extensible, MOF-Based Metadata Repository System. Technischer Bericht, OFFIS, Oldenburg, 2001.

[JT01] H. Jaekel und T. Teschke. Prozessorientierte Beschreibung von Fachkomponenten. In K. Turowski, Hrsg., *Modellierung und Spezifikation von Fachkomponenten: 2. Workshop*, Seiten 105–112, Bamberg, 2001.

[JT02] H. Jaekel und T. Teschke. Berücksichtigung von Berechtigungskonzepten bei der Spezifikation von Fachkomponenten. In K. Turowski, Hrsg., *Modellierung und Spezifikation von Fachkomponenten: 3. Workshop*, Seiten 69–85, Nürnberg, 2002.

[Kei01] M. Keilers. Ein internetbasierter Komponentensuchdienst auf Basis von Verhaltensbeschreibungen. Diplomarbeit, Carl von Ossietzky Universität Oldenburg, Fachbereich Informatik, 2001.

[KNS92] G. Keller, M. Nüttgens und A.-W. Scheer. Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)“. Technischer Bericht 89, Institut für Wirtschaftsinformatik, Universität des Saarlandes, 1992.

[LW94] B. Liskov und J. Wing. A Behavioral Notion of Subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6):1811–1841, 1994.

[Mey92] B. Meyer. Applying “Design By Contract”. *IEEE Computer*, 25(10):40–51, 1992.

[Mil80] R. Milner. A calculus of communicating systems. In *Lecture Notes in Computer Science 92*. Springer-Verlag, 1980.

[Nie95] O. Nierstrasz. Regular Types for Active Objects. In O. Nierstrasz und D. Tsichritzis, Hrsg., *Object-Oriented Software Composition*, Seiten 99–121. Prentice Hall, 1995.

[Nom02] Nomina, Software-Marktplatz: Die ISIS Software- und Firmendatenbanken. Elektronische Quelle: *http://www.software-marktplatz.de/*, zuletzt besucht im Juni 2002.

[OMG01] Object Management Group. *Unified Modeling Language Specification, Version 1.4*, 2001.

[Ort97] E. Ortner. *Methodenneutraler Fachentwurf*. Wirtschaftsinformatik. B. G. Teubner Verlag, Stuttgart, Leipzig, 1997.

[Rit98] J. Ritter. PROSECCO – Eine Methode zur modellbasierten Anwendungssystemgestaltung. In *Proceedings of Business Information Systems '98*, Posen, Polen, 1998.

[SAP02] SAP, SAP – Software Partner Directory. Elektronische Quelle: *http://www.mysap.com/partners/software/directory/*, zuletzt besucht im Juni 2002.

[Sch97] B. Schienmann. *Objektorientierter Fachentwurf (in German)*, Jgg. 20 von *Teubner-Texte zur Informatik*. B.G. Teubner Verlag, Stuttgart, Leipzig, 1997.

[Sof02] SoftSelect, SoftSelect Matching Plattform. Elektronische Quelle: *http://www.softselect.de/*, zuletzt besucht im Juni 2002.

[STR00] C. Sandmann, T. Teschke und J. Ritter. Ein Vorgehensmodell für die komponentenbasierte Anwendungsentwicklung. In B. Britzelmaier und S. Geberl, Hrsg., *Information als Erfolgsfaktor*, Seiten 49–58, Stuttgart, 2000. Teubner.

[Sun02] Sun Microsystems, SUN Solutions Marketplace. Elektronische Quelle: *http://industry.java.sun.com/solutions/*, zuletzt besucht im Juni 2002.

[Szy98] C. Szyperski. *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley, 1998.

[Tec02] Technische Universität Chemnitz, Informationssysteme & Management, Component Markets – An Overview. Elektronische Quelle: *http://www.tu-chemnitz.de/wirtschaft/wi2/projects/components/*, zuletzt besucht im Mai 2002.

[Tes01] T. Teschke. Using Business Process Knowledge for Software Component Retrieval. In *Proceedings of 11th Annual BIT Conference*, Manchester, 2001.

[Tes02] T. Teschke. Ontological Intermediation between Business Process Models and Software Components. In H.-M. Haav und A. Kalja, Hrsg., *Databases and Information Systems, Proceedings of Fifth International Baltic Conference Baltic DB&IS'2002*, Jgg. 1, Tallinn / Estland, 2002.

[TR01] T. Teschke und J. Ritter. Towards a Foundation of Component-Oriented Software Reference Models. In G. Butler und S. Jarzabek, Hrsg., *Generative and Component-Based Software Engineering*, Lecture Notes in Computer Science (LNCS) 2177, Seiten 70–84, Berlin, 2001. Springer.

[Weh02] H. Wehrheim. *Behavioural Subtyping in Object-Oriented Specification Formalisms*. Dissertation, Carl von Ossietzky Universität Oldenburg, Fachbereich Informatik, 2002.

[WZ88] P. Wegner und S. B. Zdonik. Inheritance as an Incremental Modification Mechanism or What Like Is and Isn't Like. In S. Gjessing und K. Nygaard, Hrsg., *Proceedings ECOOP '88*, Lecture Notes in Computer Science 322, Seiten 55–77, Oslo, Norwegen, 1988. Springer-Verlag.