Klaus Turowski
(Hrsg.)

Jörg Ackermann
Frank Brinkop
Stefan Conrad
Peter Fettke
Andreas Frick
Elke Glistau
Holger Jaekel
Otto Kotlar
Peter Loos
Heike Mrech
Erich Ortner
Ulrich Raape
Sven Overhage
Stephan Sahm
Andreas Schmietendorf
Thorsten Teschke
Klaus Turowski

Vereinheitlichte Spezifikation von Fachkomponenten

Memorandum des Arbeitskreises 5.10.3
Komponentenorientierte betriebliche Anwendungssysteme

Februar 2002

Gesellschaft für Informatik
Arbeitskreis 5.10.3
Komponentenorientierte betriebliche Anwendungssysteme

Vorwort

Traditionelle Ingenieurdisziplinen zeichnen sich insbesondere dadurch aus, dass in der Regel verbindliche Standards für Notation, Benennung, Bemaßung usw. zur Spezifikation der jeweiligen Konstruktionsergebnisse vorgegeben sind, die deren (Wieder-)Verwendung vereinfachen. So ist beispielsweise ein Meister oder Facharbeiter in einem Industriebetrieb in der Lage, aus einer Konstruktionszeichnung die nächsten für ihn relevanten Arbeitsschritte abzuleiten und ein Entwicklungsingenieur kann anhand der Spezifikation eines (Fremd-)Teils erkennen, ob er dieses für seine jeweilige Aufgabenstellung wieder verwenden kann.

Für den Bereich des Software Engineering und insbesondere für die Entwicklung betrieblicher Anwendungssysteme stellt sich die Situation jedoch völlig anders dar. Dort sind Komponentenstrategien bestenfalls im Bereich anwendungsübergreifender Software (z. B. Middleware) erfolgreich. Die oben genannten, verbindlichen methodischen Standards fehlen und verhindern so den Wechsel zu einer Softwareentwicklung, die auf breiter Front auf die Wiederverwendung bestehender Lösungen setzt und damit das Wirtschaftlichkeitspotential eines Produktentstehungsprozesses nach industriellem Muster ausschöpft. So ist umstritten wie Softwarekomponenten zu spezifizieren sind, welche Notationen dazu einzusetzen sind und welches die konkreten zu spezifizierenden Objekte sind. Es versteht sich von selbst, dass vor diesem Hintergrund weder in der universitären Lehre noch in einschlägigen Ausbildungsberufen eine einheitliche Konstruktionslehre vermittelt werden kann, noch die praktischen Vorteile einer solchen für den Softwareentwicklungsprozess genutzt werden können.

In Hinblick auf die dargestellte Problematik wurde auf dem ersten Workshop *Modellierung und Spezifikation von Fachkomponenten*, der am 12. Oktober 2000 vom Arbeitskreis 5.10.3 *Komponentenorientierte betriebliche Anwendungssysteme* der Gesellschaft für Informatik (GI) im Rahmen der Fachtagung *Modellierung betrieblicher Informationssysteme (MobIS) 2000* an der Universität Siegen veranstaltet wurde, vereinbart, einen Vorschlag zur Vereinheitlichung der Spezifikation von Fachkomponenten zu verfassen. Die Teilnehmer des Workshops waren aufgefordert, sich an der Erstellung eines gleichnamigen Memorandums zu beteiligen, das nun in einer ersten konsolidierten Fassung vorliegt.

Bevor diese entstehen konnte, wurden insgesamt elf Zwischenstände verfasst, innerhalb der Gruppe der Autoren diskutiert und der interessierten Öffentlichkeit vorgestellt – zuletzt im Rahmen des zweiten Workshops *Modellierung und Spezifikation von Fachkomponenten*, der, veranstaltet vom Arbeitskreis 5.10.3, in Bamberg, am 5. Oktober 2001, als Teil der Fachtagung *Verteilte Informationssysteme auf der Grundlage von Objekten, Komponenten und Agenten (vertVIS 2001)* stattfand.

Schließlich wurden die letzten Änderungen auf dem Workshop *Vereinheitlichung der Spezifikation von Fachkomponenten I* abgestimmt, der vom Arbeitskreis 5.10.3 in Kooperation mit der Universität Augsburg und mit freundlicher Unterstützung der SAP AG am 03. und 04. Dezember 2001 in Walldorf ausgerichtet wurde.

Eine Fassung dieser Schrift wird ergänzend über das WWW unter `www.fachkomponenten.de` verfügbar gemacht. Darüber hinaus finden sich dort Beispiele und Fallstudien zur memorandumskonformen Spezifikation von Fachkomponenten, Hinweise auf Projekte, Verweise auf

Lehrveranstaltungen und Lehrmaterialien zur Spezifikation von Fachkomponenten, Tagungsbände sowie Hinweise zu weiteren Veranstaltungen des Arbeitskreises 5.10.3.

Abschließend sei den Autoren und Diskutanten für ihr Engagement, die eingebrachte Arbeitszeit und ihre Beiträge gedankt, die wesentlich zur Entstehung dieses Memorandums beitragen haben. Dank gebührt ferner all jenen, die durch organisatorische Unterstützung ihren Anteil an der Fertigstellung dieser Schrift beigesteuert haben.

Augsburg, im Februar 2002

Klaus Turowski

Inhalt

Abkürzungen

CORBA	Common Object Request Broker Architecture
COTS	Commercial off-the-shelf
CPU	Central Processing Unit
DBMS	Datenbankmanagementsystem
DTD	Document Type Definition
EJB	Enterprise JavaBean
FCM	Factor Criteria Metrics Model
GQM	Goal/Question/Metric
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
OCL	Object Constraint Language
OMG	Object Management Group
RDF	Resource Description Framework
SQL	Structured Query Language
UML	Unified Modeling Language
WSDL	Web Service Description Language
XML	Extensible Markup Language

Abbildungen

1 Kompositorische Wiederverwendung und grundlegende Begriffe

Das Ziel, Softwarekomponenten (im Folgenden kurz Komponenten) verschiedener Anbieter *kundenindividuell* zu einem Anwendungssystem zu kombinieren, sodass sich die Vorteile der Verwendung von Standard- und Individualsoftware verbinden, wird seit langem verfolgt [McIl1968]. Diesem Ziel liegt als *Leitbild*, im Sinne eines idealen zukünftigen Zustands, die Idee einer kompositorischen, plug-and-play-artigen Wiederverwendung von Black-Box-Komponenten zu Grunde, deren Realisierung einem Verwender vorborgen bleibt und die auf einem Softwaremarkt gehandelt werden.

Beispielsweise könnte dann ein Unternehmen, das eine neue Software zur Unterstützung der Lagerverwaltung sucht, eine geeignete (fachliche oder Fach-) Komponente am Markt erwerben und in ihr betriebliches Anwendungssystem mit *geringem* Aufwand integrieren. Bei dem *Softwaremarkt* kann es sich dabei sowohl um einen offenen Markt (man spricht dann auch von commercial off-the-shelf (COTS) Komponenten) als auch um ein unternehmensinternes Koordinationsinstrument handeln.

In Anlehnung an [FeRT1999, S. 34] werden die Begriffe *Komponente* und *Fachkomponente* wie folgt definiert:

> Eine *Komponente* besteht aus verschiedenartigen (Software-)Artefakten. Sie ist wiederverwendbar, abgeschlossen und vermarktbar, stellt Dienste über wohldefinierte Schnittstellen zur Verfügung, verbirgt ihre Realisierung und kann in Kombination mit anderen Komponenten eingesetzt werden, die zur Zeit der Entwicklung nicht unbedingt vorhersehbar ist.

Zu den *(Software-)Artefakten* zählen der ausführbare Code, darin verankerte Grafiken, Textkonstanten usw., die einen initialen Zustand der Komponente beschreibenden Daten, z. B. Voreinstellungen oder Parameter, sowie Spezifikation, (Anwender-)Dokumentation und (automatisierte) Tests.

Das Kriterium der *Wiederverwendbarkeit* erfüllt eine Komponente dann, wenn sie, mit Ausnahme einer vom Entwickler vorgesehenen Parametrisierung, ohne Modifikation der zugehörigen (Software-)Artefakte für verschiedene Softwaresysteme mit geringem Integrationsaufwand einsetzbar ist.

Das Kriterium der *Abgeschlossenheit* erfüllt eine Komponente dann, wenn ihr ihre Bestandteile (die (Software-)Artefakte) eindeutig zuordenbar sind, sodass sie als Ganzes klar von anderen Systemteilen abgegrenzt werden kann. Damit ist die Abgeschlossenheit einer Komponente eine wesentliche Voraussetzung für ihre Vermarktbarkeit.

Vermarktbar bedeutet, dass Komponenten prinzipiell über die Eigenschaft verfügen, als für sich selbst stehendes Gut identifizierbar zu sein, sodass sie sowohl auf einem offenen als auch auf einem unternehmensinternen Markt gehandelt werden können.

> Eine *Fachkomponente* ist eine Komponente, die eine bestimmte Menge von Diensten einer *betrieblichen* Anwendungsdomäne anbietet.

Eine Voraussetzung für die komponentenorientierte Entwicklung von Anwendungssystemen aus (Fach-)Komponenten, die einheitlich beschrieben zur Verfügung stehen, ist ein tragfähiges *Komponentenmodell*. Daneben erscheint es zweckmäßig, wenn auch verschiedene Komponentenarten unterschieden werden können. Bild 1 und 2 veranschaulichen in Form einer Einteilung

von Komponentenarten und eines Metaschemas für das Komponentenmodell die Erfüllung beider Bedingungen.

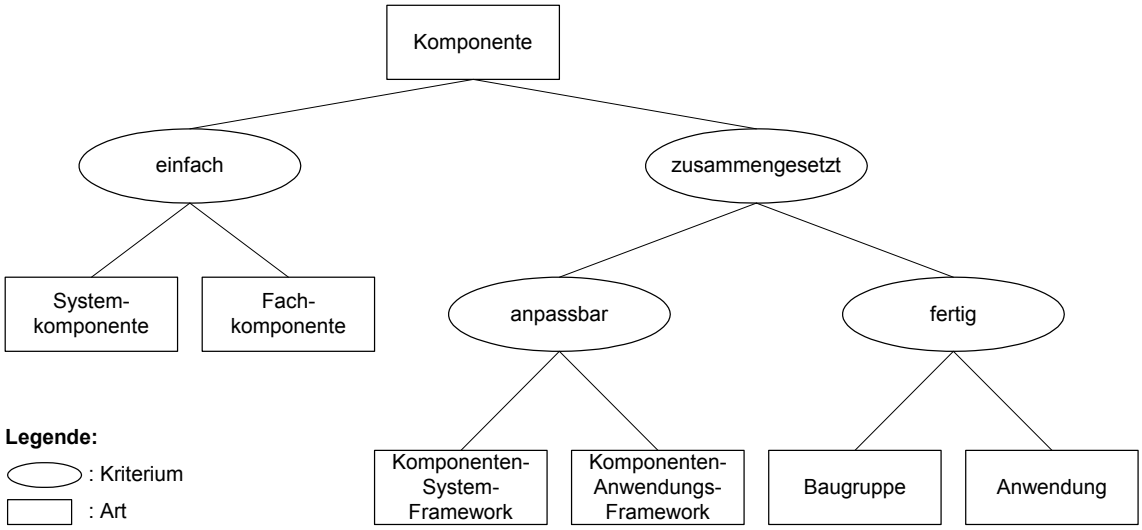

Bild 1: Einteilung der Komponentenarten

Die Komponenten werden zunächst (Bild 1) in einfache (aber abgeschlossene) und zusammengesetzte Komponenten eingeteilt. Einfache Komponenten können weiter in Systemkomponenten (sie realisieren generische Funktionen) und Fachkomponenten (mit ihnen werden anwendungsspezifische Funktionen erfüllt) eingeteilt werden. Bei den zusammengesetzten Komponenten (Bild 1) werden zunächst Komponenten, die für den Einsatz in einem Anwendungsbereich modifiziert (angepasst) werden können und zusammengesetzte Komponenten, die so wie sie – möglicherweise in einer hohen Variantenzahl – vorliegen eingesetzt werden, unterschieden.

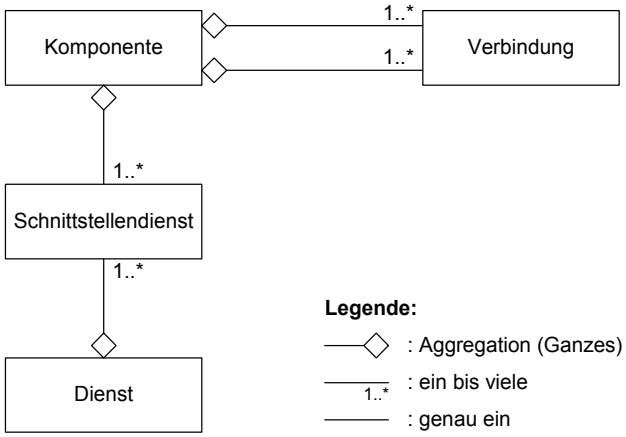

Bild 2: Metaschema des Komponentenmodells

Anpassbare, zusammengesetzte Komponenten werden bei der Realisierung generischer Funktionen *Komponenten-System-Frameworks* und bei einer Realisierung anwendungsspezifischer Funktionen *Komponenten-Anwendungs-Frameworks* genannt. Eine Form der Anpassung von Frameworks ist ihre Verbindung mit einfachen oder zusammengesetzten System- und/oder Fachkomponenten. Zusammengesetzte System- oder *Fachkomponenten* werden in Bild 1 Baugruppen (eine alternative Bezeichnung wäre *Konfigurationen*) genannt. Neben Baugruppen wer-

den noch Anwendungen, die als fertige, zusammengesetzte Komponenten-Konfigurationen (Bild 1) aufzufassen sind, unterschieden.

Das Metaschema (Bild 2), welches das zu Grunde liegende (stark vereinfachte) Komponentenmodell widerspiegelt, besteht zentral aus dem Objekttyp KOMPONENTE. Die stücklistenartige Struktur nach außen wird mit dem Objekttyp VERBINDUNG dokumentiert. Nach innen werden im Metaschema über die Objekttypen (standardisierter) DIENST und SCHNITTSTELLENDIENST die Funktionalität bzw. die Aufgaben, die Komponenten erfüllen, beschrieben.

Neben einer Standardisierung der Dienste lässt sich, wie in [OrRS1990] beschrieben, auch eine Standardisierung der Attribute einer Komponente über Datenelemente im Metaschema (Bild 2) separat hervorheben. Entscheidend ist dabei, dass sowohl die Datenelemente als auch die Dienste für eine Anwendungsdomäne „verwendungsneutral“ [OrRS1990] normiert werden können.

2 Spezifikation und Spezifikationsebenen

Um Fachkomponenten (verschiedener Hersteller) mit *geringem* Aufwand zu einem kundenindividuellen Anwendungssystem zu integrieren, bedarf es der Etablierung *inhaltlich-funktionaler* Standards. Darüber hinaus ist die in diesem Memorandum im Vordergrund stehende Schaffung eines *methodischen* Standards anzustreben, der die im Rahmen der Spezifikation von Fachkomponenten einzusetzenden Notationen festschreibt und deren unternehmens- und entwicklerübergreifende Wiederverwendung, durch ein den Akteuren bekanntes und standardisiertes Notationsmix, vereinfacht.

> Unter der *Spezifikation* einer Fachkomponente wird die vollständige, widerspruchsfreie und eindeutige Beschreibung ihrer Außensicht verstanden, d. h. die Spezifikation zeigt auf, welche Dienste eine Fachkomponente in welchem Bedingungsrahmen bereitstellt.

Als Verwender oder Verfasser der Spezifikation einer Fachkomponente kommen eine Vielzahl verschiedener Aufgabenträger in Betracht, z. B. Mitarbeiter der betrieblichen Fachabteilungen, Softwarearchitekten, Berater, Einkäufer, Vertriebsmitarbeiter, Projektleiter oder Entwickler, die im Rahmen ihrer Tätigkeit unterschiedliche *Rollen* wahrnehmen [Sahm2000]. Beispiele für Rollen mit besonderen komponentenspezifischen Aufgaben sind:

- *Konfigurator:* Anpassung der zugekauften Komponenten.
- *Assemblierer:* Installation und Verteilung von selbst entwickelten und angepassten Komponenten in einer Zielumgebung.
- *Qualitätssicherer:* Test der Komponenten gegen deren Spezifikation, formale und inhaltliche Prüfung der Spezifikation.
- *Komponentenverwalter:* Ablage der Komponenten im Repository, Verwalten verschiedener Komponentenversionen, Aufbau von Klassifikationssystemen.

In Anlehnung an [Turo1999] und [Turo2001] wird postuliert, dass die Spezifikation von Fachkomponenten auf verschiedenen Beschreibungsebenen (Bild 3) erfolgen muss und für jede dieser Beschreibungsebenen speziell für diese geeignete Notationen angegeben werden können. Als „Notationssprachen“ (Entwurfs- und Beschreibungssprachen) kommen dabei in Frage:

- mathematische oder formale Sprachen (z. B. Prädikatenlogik, Mengenalgebra,...),
- Programmiersprachen (imperative, funktionale, prädikative und objektorientierte Programmiersprachen),

- graphische Sprachen oder Diagrammsprachen (z. B. Klassendiagramme, Sequenzdiagramme, Zustandsdiagramme),
- Normal- oder Gebrauchssprachen (z. B. Alltagssprachen, Fachsprachen, Normsprachen).

Im Folgenden werden die dargestellten Beschreibungsebenen (jeweils in separaten Abschnitten) charakterisiert und um einen von den Verfassern des Memorandums getragenen Vorschlag für eine standardisierte Spezifikationstechnik ergänzt. Änderungsvorschläge sowie alternative Empfehlungen werden in Unterabschnitten berücksichtigt. Eine Sonderstellung nehmen dabei ergänzende (grafische) Notationen ein, die dazu gedacht sind, *ausgewählte* Sachverhalte *zusätzlich* in einer für eine bestimmte Adressatengruppen besonders geeigneten Form aufzubereiten.

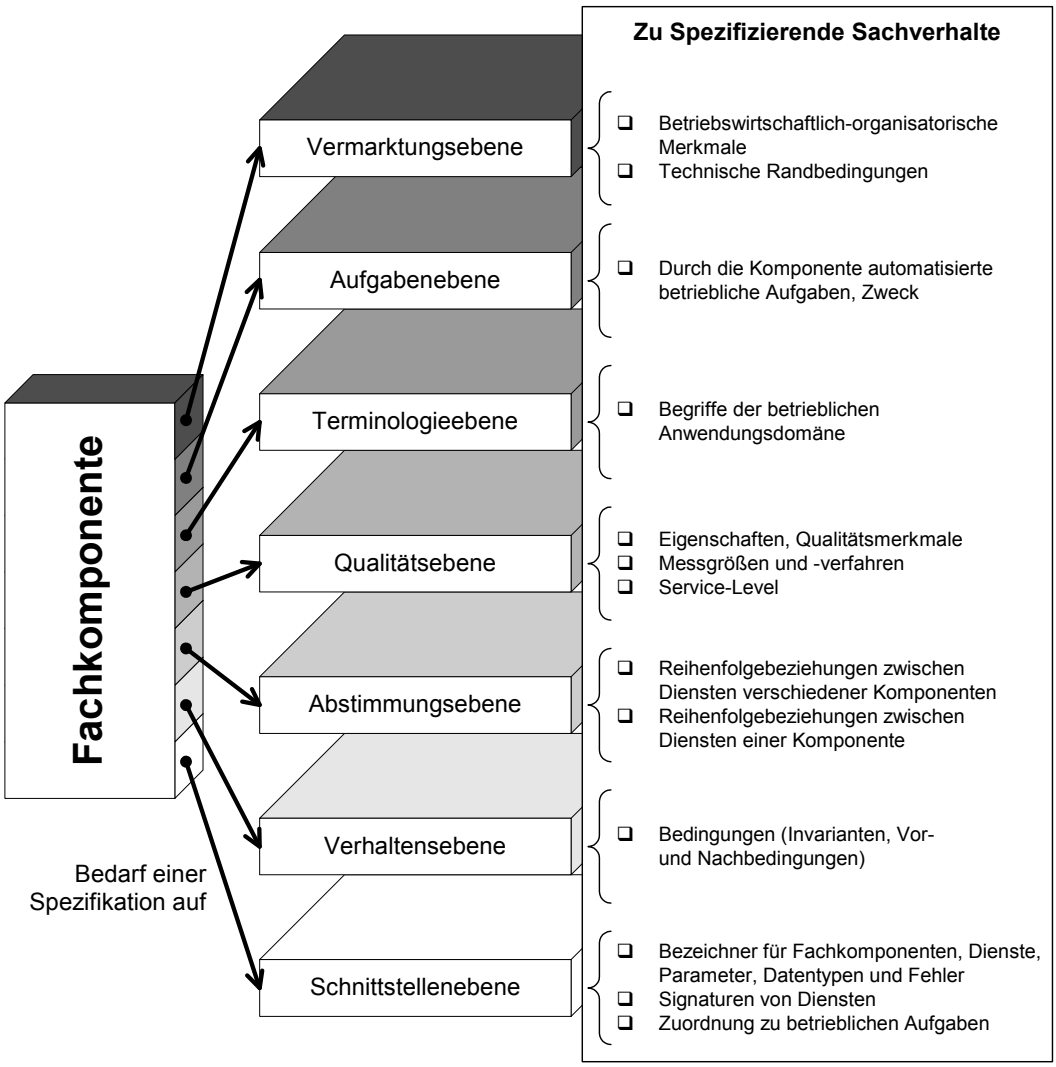

Bild 3: Beschreibungsebenen und zu spezifizierende Sachverhalte

Als *primäre* standardisierte Spezifikationstechnik soll formalen Notationen Vorrang gewährt werden, da diese die notwendige entwickler- und unternehmensübergreifende inter-subjektive Nachvollziehbarkeit der Spezifikationsergebnisse in besonderer Weise unterstützen. Eine Notation wird als *formal* charakterisiert, wenn Syntax und Semantik der Notation eindeutig und widerspruchsfrei definiert sind. Für die ergänzende (*sekundäre*) Spezifikationstechnik wird die Forderung nach einer bevorzugt formalen Notation fallen gelassen.

Insgesamt soll im Sinne einer breiten Lehr-, Lern- und Beherrschbarkeit der zur Spezifikation notwendigen Techniken erreicht werden, ein möglichst *kompaktes* Notationsmix zu finden.

Abschließend sei darauf hingewiesen, dass einige Autoren der formalen Spezifikation von betrieblichen Anwendungssystem(teil)en kritisch gegenüberstehen, da der damit verbundene Aufwand als zu groß und die allgemeine Verständlichkeit als zu gering eingestuft wird. Als Beispiel wird dazu häufig auf die algebraische Spezifikation abstrakter Datentypen als Sonderfall der formalen Spezifikation eingegangen, um die genannten Schwächen zu identifizieren, vgl. z. B. [BiMR1991, S. 288-291] und die dort genannten Quellen. Als mögliche praktikablere Alternativen zur algebraischen Spezifikation werden die *operationelle* oder die *verbale* Spezifikation angeführt. In diesem Kontext soll jedoch auch nicht unerwähnt bleiben, dass andererseits häufig die geringe Qualität der Spezifikationsergebnisse, insbesondere hinsichtlich Eindeutigkeit und inter-subjektiver Nachvollziehbarkeit, moniert wird.

3 Schnittstellenebene

3.1 Zweck

Auf der Schnittstellenebene werden grundlegende Vereinbarungen getroffen. Dazu gehören die Benennung der Dienste, die eine Fachkomponente öffentlich anbietet, die Benennung öffentlich verfügbarer Attribute, Variablen oder Konstanten, die Vereinbarung spezieller (Daten-)Typen (in der Regel auf der Grundlage standardisierter (Daten-)Typen), das Festlegen der Signaturen von Diensten sowie die Deklaration von Fehlermeldungen oder Ausnahmezuständen. Neben den angebotenen Diensten müssen auch die eventuell von der Komponente benötigten Dienste anderer Komponenten spezifiziert werden. Als Techniken werden z. B. Programmiersprachen oder Schnittstellendefinitionssprachen (*Interface Definition Language* (IDL)) verwendet. Die daraus resultierenden Vereinbarungen garantieren, dass Dienstnehmer und Dienstgeber miteinander kommunizieren können. Dabei steht die technische Herstellung einer Kommunikationsfähigkeit im Vordergrund, bei der semantische Aspekte weitgehend unberücksichtigt bleiben.

Daneben wird auf Schnittstellenebene dargelegt, welche (Fach-)Begriffe, die auf der Terminologieebene eingeführt werden, mit welchen (Daten-)Typen und welche auf der Aufgabenebene erläuterten Aufgaben mit welchen Diensten der Fachkomponente korrespondieren.

3.2 Notationsvorschlag (primäre Notation)

Zur Notation von Sachverhalten auf der Schnittstellenebene wird die Verwendung der Object Management Group (OMG) IDL vorgeschlagen [OMG2001a, S. 3.1-3.58]. Die OMG IDL ist dazu geeignet, um die auf der Schnittstellenebene zu berücksichtigenden Sachverhalte zu spezifizieren und konstituiert einen offenen, von Forschung und Industrie in der Breite getragenen Standard. Die operationale Semantik der IDL, hier insbesondere die Frage wie eine konkrete programmiersprachliche Schnittstellenspezifikation, die z. B. die Übergabe von Objektreferenzen oder Ausprägungen vorsieht, in die OMG IDL zu übertragen ist (und umgekehrt), ist in den jeweiligen OMG *Language Mappings* geregelt.

Um die angebotenen von den eventuell benötigten Diensten abzugrenzen, werden alle benötigten Dienste unter `interface extern` zusammengefasst.

Zur Notationen der korrespondierenden (Fach-)Begriffe und (Daten-)Typen bzw. Aufgaben und Dienste werden zwei jeweils zweispaltige Tabellen vorgeschlagen.

3.3 Beispiel (primäre Notation)

Bild 4 zeigt einen vereinfachten Auszug der Schnittstellenbeschreibung einer möglichen Fachkomponente *Bestandskontenverwaltung* auf Schnittstellenebene. Die Fachkomponente soll dazu dienen, verschiedene Bestandskonten zu verwalten. Die Bestandskontenverwaltung wurde als Beispiel gewählt, da sie eine gut verstandene Aufgabe innerhalb der betrieblichen Anwendungsdomäne darstellt (vgl. z. B. [Kern1993, S. 63-74, 141-154]), sodass die Ergebnisse der Spezifikation aus fachlicher Sicht nachvollziehbar und überprüfbar sind.

```
interface Bestandskontenverwaltung {
  typedef string KontoNr;
  typedef double Bestand;

  struct Zeitpunkt { ... };

  struct Konto {
    KontoNr     n;
    Bestand     Sicherheitsbestand;
    Bestand     Meldebestand;
    ...
  };

  struct Buchungssatz {
    KontoNr     n;
    Zeitpunkt   Termin;
    string      Auftragsnummer;
    double      Menge;
  };

  exception ZuGeringerBestand {};

  void    Buche(in Buchungssatz b);
  void    Reserviere(in Buchungssatz b) raises (ZuGeringerBestand);
  Bestand BerechneBestandZum(in KontoNr n, in Zeitpunkt z);
  Bestand BerechneBedarf(in KontoNr n, in Zeitpunkt z);
  void    SetzeSicherheitsbestand(in KontoNr n, in double Menge);
  void    SetzeMeldebestand(in KontoNr n, in double Menge);
};

interface extern {
  typedef long PeriodeInTagen;

  struct Zeitpunkt {
    unsigned short Tag;
    unsigned short Monat;
    unsigned short Jahr;
  };

  PeriodeInTagen BerechnePeriodeInTagen(in Zeitpunkt b, in Zeitpunkt e);
};
```

Bild 4: Beispiel für die Verwendung der OMG IDL auf Schnittstellenebene

Mit Hilfe des Schlüsselworts `interface` wird der Name der Fachkomponente festgelegt. Untergeordnete Teile der Fachkomponente werden damit eindeutig identifizierbar. So weist `Bestandskontenverwaltung::Buche` im Beispiel darauf hin, dass eine Funktion *Buche*, die zu einer Fachkomponente *Bestandskontenverwaltung* gehört, gemeint ist. Als nächstes werden einfache und strukturierte Typen auf der Basis *vordefinierter* Typen vereinbart, die für die Spezifikation der Signaturen der angebotenen Dienste benötigt werden.

Nachdem die besonderen Typen des Dienstgebers vereinbart sind, können noch spezielle Ausnahmesignale (Schlüsselwort `exception`) deklariert werden, die besondere Fehlerzustände des Dienstgebers anzeigen. Beispielsweise wird im Beispiel ein Signal vereinbart, das anzeigt, dass

auf Grund eines zu geringen (Plan-)Bestands der notwendige Bedarf nicht reserviert werden kann. Abschließend werden noch die Dienste, die der Dienstgeber anbietet, und (unter `interface extern`) die Dienste, die er benötigt, sowie deren Aufruf- und Ergebnisparameter deklariert.

(Fach-)Begriff	(Daten-)Typ
Konto	`Konto`
Bestand	`Bestand`
Buchungssatz	`Buchungssatz`

Bild 5: Beispiel für die Notation korrespondierender Begriffe und Typen

Bild 5 und Bild 6 zeigen ein Beispiel für die Notation korrespondierender (Fach-)Begriffe und (Daten-)Typen bzw. Aufgaben und Dienste auf der Schnittstellenebene.

Aufgabe	Dienst
Buche Buchungssatz *b*	`void Buche(in Buchungssatz b)`
Reserviere Buchungssatz *b*	`void Reserviere(in Buchungssatz b) raises (ZuGeringerBestand)`
Berechne Bestand des Kontos mit der Kontonummer *n* zum Zeitpunkt *z*	`Bestand BerechneBestandZum(in KontoNr n, in Zeitpunkt z)`

Bild 6: Beispiel für die Notation korrespondierender Aufgaben und Dienste

3.4 Ausblick

Als mögliche zukünftige Alternative zur OMG IDL wird die Verwendung der *Web Service Description Language* (WSDL) [CCM+2001] untersucht.

4 Verhaltensebene

4.1 Zweck

Die Vereinbarungen auf der Verhaltensebene dienen der näheren Beschreibung des Verhaltens einer Fachkomponente. Sie ergänzen die grundlegenden Vereinbarungen der Schnittstellenebene, die vornehmlich die Syntax der Schnittstelle beschreibt, jedoch offen lässt, wie sich eine Fachkomponente im Allgemeinen und insbesondere bei Grenzfällen verhält. So kann auf der Verhaltensebene für eine Fachkomponente *Lagerverwaltung* als Invariante festgelegt werden, dass der Meldebestand für jedes Bestandskonto immer größer oder gleich dem Sicherheitsbestand sein muss. Neben Invarianten werden auf der Verhaltensebene noch auf einzelne Dienste bezogene Vor- und Nachbedingungen spezifiziert.

Wichtig ist, dass das Verhalten der Fachkomponente vollständig beschrieben werden muss. Das bedeutet insbesondere, dass neben den angebotenen Diensten auch die benötigten Dienste be-

schrieben werden. Dabei ist zu spezifizieren, welches Verhalten die Fachkomponente von den benötigten Diensten anderer Fachkomponenten erwartet.

4.2 Notationsvorschlag (primäre Notation)

Zur Notation von Sachverhalten, die der Verhaltensebene zuordenbar sind, wird die *Object Constraint Language* (OCL) vorgeschlagen. Diese wurde als Bestandteil der *Unified Modeling Language* (UML) [OMG2001b] von der OMG als Notationsstandard übernommen und ist somit die empfohlene Ergänzung der OMG IDL zur Spezifikation von Sachverhalten auf der Verhaltensebene.

Für die Beschreibung der von der Fachkomponente benötigten Dienste wird, analog zur Schnittstellenebene, eine gedachte Komponente „Extern“ definiert, die alle diese Dienste enthält.

4.3 Ergänzende (sekundäre) Notation

Alle mit OCL formulierten Bedingungen sollen zusätzlich in natürlichsprachlicher Form angegeben werden.

4.4 Beispiel

In Bild 7 sind Teile der Spezifikation der oben eingeführten Fachkomponente *Bestandskontenverwaltung* auf Verhaltensebene dargestellt. Jede Bedingung wird zusätzlich natürlichsprachlich ausgedrückt (Bild 8).

```
Bestandskontenverwaltung
  self.Konto->forAll(k:Konto | k.Sicherheitsbestand >= 0)
  self.Konto->forAll(k:Konto | k.Meldebestand >= k.Sicherheitsbestand)

Bestandskontenverwaltung::BerechneBestandZum(n:KontoNr,z:Zeitpunkt):Bestand
  pre : self.Konto->exists(k:Konto | k.KontoNr = n)
  post: result = self.Buchungssatz->iterate(b:Buchungssatz; r:Bestand = 0 |
                   if b.KontoNr    = n and
                      b.Zeitpunkt <= z
                   then
                      r + b.Menge
                   endif
                )
```

Bild 7: Beispiel für die Verwendung der OCL auf Verhaltensebene

Für weitere Beispiele sei auf die Fallstudie in [Acke2001] verwiesen. Im Folgenden finden sich noch einige Erläuterungen zu den Beispielen in Bild 7.

Im Rahmen der Spezifikation wird zunächst der Kontext festgelegt, auf den sich die Zusicherungen beziehen. Der Kontext wird jeweils durch Unterstreichen deutlich gemacht. So beziehen sich beispielsweise die ersten beiden Zusicherungen auf die gesamte Fachkomponente *Bestandskontenverwaltung*. Die anderen Zusicherungen beziehen sich auf den Dienst *BerechneBestandZum*, welche die Fachkomponente anbietet. Dies wird durch die Einschränkung des Kontextes mit `::` verdeutlicht. Werden Parameter eines Dienstes benötigt, um dessen Verhalten zu beschreiben, so können diese zusätzlich angegeben werden. Beispielsweise sind zur Spezifikation des Verhaltens des Dienstes *BerechneBestandZum* zwei typbehaftete Parameter und der Typ des Ergebnisses angegeben. Zusicherungen treten entweder als Vorbedingungen (eingeleitet durch das Schlüsselwort `pre`), als Nachbedingungen (erkennbar am Schlüsselwort `post`) oder als Invarianten (kein Schlüsselwort) auf.

Im Beispiel wurden zwei Invarianten vereinbart. Die erste Invariante sagt aus, dass für jedes Konto (Schlüsselwort `forAll`), das von der Fachkomponente *Bestandskontenverwaltung* verwaltet wird, ein Sicherheitsbestand definiert sein muss (`k.Sicherheitsbestand`), der größer oder gleich null ist. Das Schlüsselwort `self` signalisiert hierbei, dass der gewählte Kontext (hier die gesamte Fachkomponente) gemeint ist. Der Kontext wird durch `.Konto->` jedoch auf die Betrachtung der Sammlung (*Collection*) von Konten eingeschränkt, die zu der Fachkomponente *Bestandskontenverwaltung* gehört. Der Begriff *Sammlung* sagt in diesem Zusammenhang lediglich aus, dass die Fachkomponente eine gewisse Anzahl von Konten verwaltet, auf die über einen Bezeichner *Konto* zugegriffen werden kann, nicht jedoch wie deren Verwaltung implementiert ist. Beispielsweise können die Konten über relationale Tabellen, verkettete Listen, Bäume, usw. verwaltet werden. Mit Hilfe von Sammlungen können auf diese Weise komplexe fachliche Sachverhalte spezifiziert werden, ohne Aussagen über die Realisierung zu treffen.

Die zweite Invariante legt fest, dass für alle verwalteten Konten ein Meldebestand definiert sein muss, der größer oder gleich dem Sicherheitsbestand ist. Da diese Invarianten für den Kontext `Bestandskontenverwaltung`, also für die gesamte Fachkomponente, vereinbart sind, gelten sie darüber hinaus für alle Dienste der Fachkomponente gleichermaßen.

Für den Dienst *BerechneBestandZum* ist eine Vorbedingung und eine Nachbedingung vereinbart. Die Vorbedingung sagt aus, dass nur für solche Konten Bestände ermittelt werden können, die existieren (Schlüsselwort `exists`), d. h., die zuvor angelegt wurden. Die Nachbedingung beschreibt das Zustandekommen des Ergebnisses des Dienstes (Schlüsselwort `result`). Dazu wird die Sammlung aller Buchungssätze durchsucht (Schlüsselwort `iterate`), welche die Fachkomponente über den Bezeichner *Buchungssatz* referenziert (`self.Buchungssatz`). Das Ergebnis dieser Schleife wird zu Beginn auf null gesetzt (`r:Bestand = 0`). Während jeder Iteration wird geprüft, ob sich der gerade betrachtete Buchungssatz auf das korrekte Bestandskonto bezieht (`b.KontoNr = n`) und ob die Buchung nicht erst nach dem als Parameter übergebenen Stichtag wirkt (`b.Zeitpunkt <= z`). Ist die Bedingung erfüllt, dann wird der Zu- oder Abgang zu dem Ergebnis addiert (`r + b.Menge`).

Der Sicherheitsbestand jedes Kontos muss größer oder gleich Null sein.
Der Meldebestand jedes Kontos muss größer oder gleich dem Sicherheitsbestand sein.
Der Dienst *BerechneBestandZum* kann nur für ein Konto ausgeführt werden, das der Fachkomponente bekannt ist.
Der (vom Dienst *BerechneBestandZum* zurückgegebene) Bestand eines Konto zu einem bestimmten Zeitpunkt ist gerade die Summe der Mengen, die bis zum betrachteten Zeitpunkt auf dieses Konto gebucht wurden. (Dabei werden Abgänge als negative Mengen betrachtet und damit vom Bestand abgezogen.)

Bild 8: Beispiel für die Verwendung der sekundären Notation auf Verhaltensebene

4.5 Alternative Notationstechniken und Ausblick

Um einen Einsatz der Fachkomponente nur auf Basis der Spezifikation zu ermöglichen, muss die Beschreibung auf der Verhaltensebene präzise sein. Deshalb wurde eine formale Notation vorgeschlagen. Da formal notierte Bedingungen nicht für alle Rezipienten gleichermaßen zu verstehen sind, sollen alle formalen Aussagen durch natürlichsprachliche Aussagen ergänzt werden. Dadurch wird, insbesondere bei der Kommunikation mit Mitarbeitern aus den betrieblichen Funktionsbereichen, eine breitere Verständlichkeit der Aussagen erreicht.

Zur Erhöhung der Qualität der natürlichsprachlichen Aussagen können diese durch die Verwendung von Satzbauplänen präzisiert werden (vgl. Abschnitt 7).

5 Abstimmungsebene

5.1 Zweck

Vereinbarungen auf der Abstimmungsebene regeln die Reihenfolge der Verwendung von Diensten und die Synchronisationserfordernisse zwischen Diensten. Beispielsweise kann hier festgelegt werden, dass, bevor auf ein Bestandskonto das erste Mal gebucht wird, ein Sicherheitsbestand festgelegt werden muss oder dass nicht zur gleichen Zeit mehrere Buchungen auf *dasselbe* Bestandskonto vorgenommen werden dürfen. Bedingungen auf der Abstimmungsebene können sich sowohl auf von der Komponente angebotene als auch von ihr benötigte Dienste beziehen. Soll nicht angegeben werden, welche konkrete Komponente nachgefragte Dienste anbietet oder ist diese nicht bekannt, dann wird der jeweilige Dienst einer gedachten Komponente „Extern“ zugeordnet.

Zweck der Abstimmungsebene ist die Bereitstellung notwendiger Informationen, wie die Komponente in eine komponentenbasierte Softwarelösung aus Prozesssicht integriert werden kann. Hierbei werden vornehmlich Bedingungen dargestellt, die sich aus betriebswirtschatlichen, sachlich-logischen Beziehungen innerhalb einer Fachkomponente und zu anderen Fachkomponenten ergeben.

5.2 Notationsvorschlag (primäre Notation)

Zur Notation von Sachverhalten, die der Abstimmungsebene zuordenbar sind, wird die Verwendung der um temporale Operatoren erweiterten OCL [CoTu2000] vorgeschlagen. Durch diese Wahl lässt sich der sonst notwendige Methodenbruch zwischen Verhaltens- und Abstimmungsebene vermeiden.

Es können die folgenden temporalen Operatoren verwendet werden:

- *sometime_past* φ, *always_past* φ, φ *sometime_since_last* ψ, φ *always_since_last* ψ
- *sometime* φ, *always* φ, φ *until* ψ, φ *before* ψ
- *initially* φ

Bei φ und ψ handelt es sich um Aussagen, denen ein zustandsabhängiger boolescher Wert zugewiesen werden kann (d. h. der Wert kann sich im Laufe der Zeit ändern). So könnte z. B. φ die Tatsache beschreiben, dass im betrachteten Zustand der Bestand größer oder gleich dem Sicherheitsbestand ist. Die temporale Formel „*always_past* φ“ drückt dann aus, dass (vom aktuellen Zustand aus) in der Vergangenheit immer φ gegolten hat. Das heißt „*always_past* φ“ ist genau dann wahr, wenn in der Vergangenheit der Bestand immer größer oder gleich dem Sicherheitsbestand war. Für weitere Erklärungen und die genaue Semantik der temporalen Operatoren vgl. [CoTu2000].

Ergänzend dazu werden die Operatoren *before* und *after* eingeführt. Das Argument beider Operatoren ist jeweils ein konkreter Aufruf eines Dienstes inklusive der Aufrufparameter.

- Der Ausdruck *before(dienst1(par1, par2))* ist zu genau dem Zeitpunkt wahr, in welchem der Dienst *dienst1* mit den Parametern *par1* und *par2* aufgerufen wird.
- Der Ausdruck *after(dienst1(par1, par2))* ist zu genau dem Zeitpunkt wahr, in welchem die Abarbeitung des Dienstes *dienst1* mit den Parametern *par1* und *par2* gerade erfolgreich beendet wurde.

Ausdrücke *before(...)* und *after(...)* können als Argumente für die temporalen Operatoren verwendet werden. Mit dieser Erweiterung kann der Zeitpunkt von Methodenaufrufen syntaktisch eindeutiger und passender in OCL-Ausdrücken verwendet werden. Außerdem können Bedingungen spezifiziert werden, die sich auf die Zustände während der Abarbeitung eines Dienstes beziehen. Für weitere Erläuterungen und Beispiele siehe [Acke2001].

Ergänzend ist zu beachten, dass durch Verwendung der temporalen Operatoren folgende Einschränkungen bei der OCL entstehen:

- Ein OCL-Ausdruck ist nicht unbedingt zu einem bestimmten Zeitpunkt auswertbar.
- In temporalen OCL-Ausdrücken sind Nicht-Query-Methoden zugelassen.

Damit ergibt sich als Konsequenz, dass temporale Operatoren in der Spezifikation nur als Modellierungskonstrukt für die Mensch-Mensch-Kommunikation verwendet werden. Die Verwendung der temporalen Operatoren sollte auf die Abstimmungsebene beschränkt bleiben, um nicht auch die Verhaltensebene obigen Einschränkungen zu unterwerfen.

5.3 Ergänzende (sekundäre) Notation

Alle mit der um temporale Operatoren erweiterten OCL formulierten Bedingungen sollen zusätzlich in natürlichsprachlicher Form angegeben werden.

5.4 Beispiel

Bild 9 zeigt zwei Beispiele zur Verwendung der um temporale Operatoren erweiterten OCL. Jede Bedingung wird zusätzlich natürlichsprachlich ausgedrückt (Bild 10).

```
Auftragsabwicklung::RechnungDrucken(at:Auftrag)
  pre : sometime_past( after( KundenauftragAnnehmen(at) ) ) and
        not( after( AuftragStornieren(at) )
        sometime_since_last ( after( KundenauftragAnnehmen(at) ) )

Auftragsabwicklung::KundenauftragAnnehmen(at:Auftrag)
  post: sometime( after( RechnungDrucken(at) ) ) or sometime( after( AuftragStornieren(at) ) )
```

Bild 9: Beispiel für die Verwendung der um temporale Operatoren erweiterten OCL

Das erste Beispiel in Bild 9 enthält eine Vorbedingung des Dienstes *RechnungDrucken* der Fachkomponente *Auftragsabwicklung*. Es wird verlangt, dass vor der Ausführung dieses Dienstes der Dienst *KundenauftragAnnehmen* für genau denselben Auftrag ausgeführt wurde und dass seit der Auftragsannahme keine Stornierung dieses Auftrags durch Ausführung des Dienstes *AuftragStornieren* stattgefunden hat.

Die zweite Bedingung ist eine Nachbedingung für den Dienst *KundenauftragAnnehmen*. Hier wird ergänzend zur vorherigen Eigenschaft verlangt, dass es nach Annahme eines Kundenauftrags irgendwann später zur Erstellung einer Rechnung für genau diesen Auftrag durch Ausführung des Dienstes *RechnungDrucken* kommen muss oder dass dieser Auftrag irgendwann durch Ausführung des Dienstes *AuftragStornieren* storniert werden muss.

Die beiden Eigenschaften drücken unterschiedliche Sachverhalte aus. Die Vorbedingung des Dienstes *RechnungDrucken* verhindert nicht, dass der Dienst *KundenauftragAnnehmen* ausgeführt wird, ohne dass es jemals zu einer Rechnungserstellung oder Stornierung für diesen Auftrag kommt. Die Nachbedingung des Dienstes *KundenauftragAnnehmen* ihrerseits verbietet nicht, dass es zur Ausführung des Dienstes *RechnungDrucken* für einen Auftrag kommen kann, auch wenn dieser Auftrag vorher nicht mittels des Dienstes *KundenauftragAnnehmen* angenommen wurde.

Die Rechnung für einen Kundenauftrag kann nur dann gedruckt werden, wenn der Kundenauftrag angenommen wurde und seit der letzten Annahme nicht mehr storniert wurde.
Für einen angenommenen Kundenauftrag muss in Zukunft entweder eine Rechnung erstellt werden oder der Kundenauftrag muss storniert werden.

Bild 10: Verwendung der sekundären Notation auf Abstimmungsebene

5.5 Alternative Notationstechniken und Ausblick

Auch auf der Abstimmungsebene müssen alle Bedingungen präzise formuliert werden, weshalb wiederum eine formale Notation vorgeschlagen wird. Um eine gute Verständlichkeit der Aussagen zu unterstützen, sollen alle formalen Aussagen durch natürlichsprachliche Aussagen ergänzt werden, die sich durch die Verwendung von Satzbauplänen weiter präzisieren lassen.

6 Qualitätsebene

6.1 Zweck

Ergänzend zu den bisher genannten Ebenen, die vornehmlich auf die Spezifikation funktionaler Eigenschaften fokussieren, sind darüber hinaus noch die nicht funktionalen Eigenschaften einer Fachkomponente zu beschreiben. Diese werden auf der Qualitätsebene spezifiziert. Beispiele hierfür sind die Verfügbarkeit, die Wartbarkeit oder auch das Performanceverhalten eines durch die Komponente angebotenen Dienstes. Eine Spezifikation auf dieser Ebene muss die Qualitätskriterien, verwendbare Messgrößen und Methoden zu deren Quantifizierung sowie ggf. durch die Komponenten einzuhaltende Service Level (Qualitätsgrenzen zur Laufzeit) berücksichtigen. Darüber hinaus ist festzulegen, in welcher Form diese Informationen dem Benutzer einer Komponente zur Verfügung gestellt werden.

Die Erfassung der Qualitätseigenschaften einer Komponente kann entweder zum Zeitpunkt der Ausführung über die Aufnahme dynamischer Größen wie z. B. des Durchsatzes und des Antwortzeitverhaltens konkreter Funktionen oder aber anhand statischer Eigenschaften erfolgen. Statische Eigenschaften wie z. B. der Umfang, Anzahl der angebotenen Funktionen, Kopplungsbeziehungen oder Testabdeckung lassen auf der Grundlage eines entsprechenden empirischen Erfahrungshintergrunds auf Qualitätseigenschaften der Komponente schließen.

Insbesondere die dynamischen Größen sind prinzipiell von den entsprechenden Randbedingungen (Hauptspeicher, Prozessor, Datenbankmanagementsystem (DBMS) etc.) abhängig, unter denen die Fachkomponente zur Ausführung gebracht wird. Diese Randbedingungen müssen konkretisiert werden, um zu objektiven Aussagen bezüglich der Qualitätseigenschaften einer

Komponente zu gelangen. Dabei ist zu beachten, dass bei verschiedenen Komponenten eventuell verschiedene Randbedingungen von Relevanz sind. Beispielsweise können bestimmte Fachkomponenten sehr stark von der Leistungsfähigkeit des DBMS abhängig sein, andere wiederum können stark abhängig von der Netzwerkkapazität oder von der Prozessorleistung sein. Daher erscheint es notwendig, die zu spezifizierenden Randbedingungen offen zu lassen.

Zur Bestimmung der Qualitätseigenschaften bei konkreter Definition von Randbedingungen sind weiterhin zwei verschiedene Möglichkeiten denkbar:

- Definition einer Referenzumgebung: Bei dieser Variante wird eine Referenzumgebung definiert, auf der prinzipiell die Qualitätseigenschaften bestimmt werden können. Dieses Vorgehen ist jedoch sehr aufwendig und Referenzumgebungen können schnell veralten und somit die Relevanz (in Bezug auf den speziellen Einsatz der Komponente) der gewonnenen Messwerte in Frage stellen.
- Sammlung von Messwerten bei konkreten Komponenten-Installationen: Bei dieser Variante wird nicht eine ausschließliche und explizite Referenzumgebung definiert. Statt dessen werden viele Messungen der Qualitätseigenschaften unter verschiedenen Randbedingungen vorgenommen. Alle diese Messfälle werden innerhalb der Spezifikation gesammelt. Problematisch hierbei ist die Form der Datenerhebung.

Durch die Berücksichtigung der vorhergehend dargestellten Randbedingungen erscheint eine Festlegung konkreter Qualitätskriterien darum nicht zielführend, da so die Universalität der Spezifikation stark eingeschränkt wird. Aus diesem Grund soll ein grober Rahmen zur Vorgehensweise der Qualitätsspezifikation festgelegt werden, der durch den jeweiligen Komponentenhersteller entsprechend konkretisiert wird.

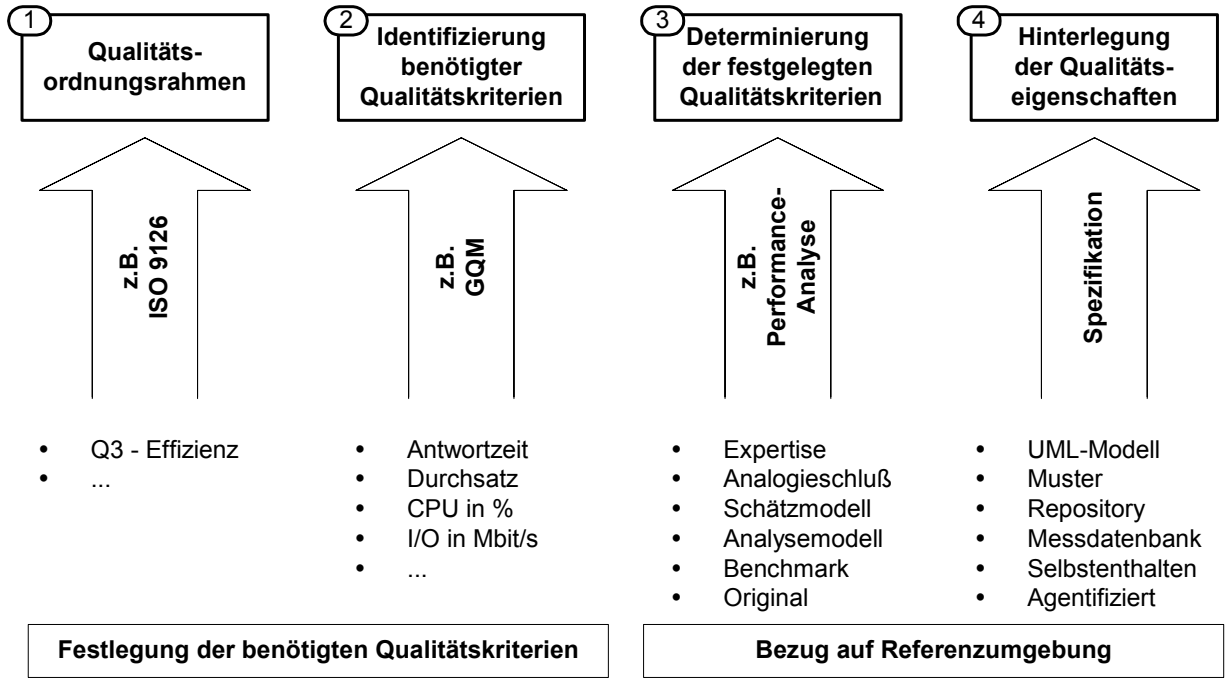

Bild 11: Schritte zur Qualitätsspezifikation [ScDu2001]

6.2 Schritte zur Qualitätsspezifikation

Bild 11 zeigt die notwendigen Schritte zur Qualitätsspezifikation einer Fachkomponente, wobei die Schritte 1 und 2 von einer im vorhergehenden Abschnitt motivierten Referenzumgebung weitgehend unabhängig sind und die Schritte 3 und 4 nur im Kontext mit einer entsprechenden Referenzumgebung ausgeführt werden können. Innerhalb von Bild 11 wird die Vorgehensweise zur Spezifikation des Qualitätskriterium der Effizienz beispielhaft skizziert. Die einzelnen Schritte werden in den folgenden Abschnitten näher erläutert.

6.3 Qualitätsklassen als Ordnungsrahmen (Schritt 1)

Der Begriff der Qualität eines Softwareproduktes im Allgemeinen als auch in Bezug auf Komponenten führt zu vielfältigen Interpretationsmöglichkeiten. Dementsprechend ist es notwendig, ein konkretes Qualitätsmodell festzulegen, um den Qualitätsbegriff zu operationalisieren. Für diese Aufgabenstellung haben sich so genannte *Factor Criteria Metrics Model* (FCM)-Ansätze herausgebildet (vgl. z. B. [Balz1998]), welche ausgehend von Qualitätsfaktoren entsprechende Qualitätskriterien festlegen und für deren Quantifizierung entsprechende Metriken vorschlagen.

Mit der ISO 9126 findet sich ein Standard für ein Qualitätsmodell entsprechend dem FCM-Ansatz. Im Folgenden soll dieser als Orientierungshilfe herangezogen werden, um Qualitätsklassen zur Bewertung von Komponenten zu bilden. Mit der Einführung der folgenden Qualitätsklassen [ScSc2000] im Rahmen der Spezifikation von Softwarekomponenten ergibt sich die Möglichkeit einer granularen Berücksichtigung von Qualitätseigenschaften konkreter Komponenten.

- *Übertragbarkeit Q1:* Anpassbarkeit, Installierbarkeit;
- *Anwendbarkeit Q2:* Erlernbarkeit, Beherrschbarkeit, Verständlichkeit;
- *Effizienz Q3:* Raumbezogen, Zeitbezogen, Ressourcenbezogen;
- *Funktionalität Q4:* Angemessenheit, Interoperabilität, Genauigkeit;
- *Zuverlässigkeit Q5:* Fehlertoleranz, Fehlerhäufigkeit, Fehlerverfolgbarkeit;
- *Wartbarkeit Q6:* Analysierbarkeit, Änderbarkeit, Stabilität, Testbarkeit.

Eine konkrete Komponente kann so z. B. die Qualitätseigenschaften Q1 und Q3 aufweisen, d. h. innerhalb der Spezifikation befinden sich Aussagen zu genau diesen Qualitätseigenschaften, nicht aber zu allen weiteren. Wie letztendlich die konkrete Ausprägung der Spezifikation in Bezug auf genau eine Qualitätsklasse erfolgt, soll in einem weiteren Schritt mit Hilfe der *Goal-Question-Metric* (GQM)-Methode geklärt werden.

6.4 Komponentenspezifische Qualitätsmodelle (Schritt 2)

Für die eigentliche Identifizierung der für eine konkrete Komponente zu spezifizierenden Qualitätseigenschaften kann z. B. das GQM-Pardigma verwendet werden [SoBe1999]. Dieses bietet eine methodische Vorgehensweise zur Erstellung eines entwicklungsspezifischen Qualitätsmodells und kann so auch für die Komponententechnologie herangezogen werden. Dafür führt die GQM-Methode, ausgehend von zu formulierenden Qualitätszielen (im speziellen Fall der Qualität können diese der ISO 9126-1 entnommen werden) und den Fragen, wie diese erreicht werden können, zu den für die Quantifizierung bzw. Beantwortung der Fragen notwendigen Messgrößen. Für die Beantwortung der identifizierten Fragestellungen werden typischerweise die folgenden Erfolgskriterien in Anlehnung an [Dumk2001] herangezogen:

- Sichtweise: Komponentenentwickler, Komponentenanwender, Auftraggeber,...
- Anwendungsbereich: spezielles Projekt bzw. ausgewählte Produktklasse
- Zweck: Analyse, Verständnis,...
- Kontext: z.B. im Rahmen eines ausgewählten Entwicklungsteams.

Vorschläge für externe Messgrößen finden sich in der ISO 9126-2, für interne Messgrößen in der ISO 9126-3. Darüber hinaus sind im Rahmen des GQM-Ansatzes Aufgabenstellungen der effizienten Messwerterfassung, der Ergebnisinterpretation und der Validation durchzuführen, vgl. z. B. [DuFS2000].

Für die Qualitätsklasse Q3 (Effizienz) ergibt sich so die folgende Ausprägung:

Goal (Ziel)

Das determinierte Performanceverhalten der durch eine Komponente angebotenen Funktionen ist für deren erfolgreichen Einsatz von entscheidender Bedeutung. (z.B. bei Komponenten im Bereich der Telekommunikation, Banken, Militär).

Question (Frage)

Welche Kenngrößen werden zur Erfassung des zeit- und raumbezogenen Performanceverhaltens benötigt?

Metric (Messgröße)

Für die Erfassung des nach außen sichtbaren Performanceverhaltens werden die Größen Antwortzeit und Durchsatz einer konkreten Funktion der Komponente herangezogen. (vgl. ISO 14756)

Darüber hinaus besteht die Notwendigkeit, den für die Erbringung dieser Größen benötigten Ressourcenverbrauch (CPU, Bandbreite, Softwareservices) festzulegen, das entsprechende Lastprofil (Auftragsarten sind im zeitlichen Verlauf zu berücksichtigen), die verwendete Hard- und Softwarearchitektur und die Leistungseigenschaften der betrachteten Gesamtarchitektur (Netzwerke, Rechner, Services usw.) zu erfassen, was der bereits angesprochenen Referenzumgebung entspricht.

Für die Qualitätsklasse Q3 wurde die Vorgehensweise zur Ermittlung der vorgenannten Messgrößen in [ScSc2000] bereits dargestellt.

6.5 Bestimmung der festgelegten Qualitätskriterien (Schritt 3)

Die Quantifizierung der Qualitätskriterien einer Komponente kann nur unter Anwendung eines konkreten Verfahrens bzw. Methode erfolgen, wobei aus Gründen der Effektivität in jedem Fall eine werkzeuggestützte Vorgehensweise anzustreben ist. Im Folgenden seien allgemeine Beispiele solcher Methoden in Bezug auf die verwendeten Qualitätsklassen genannt:

Übertragbarkeit Q1: Im speziellen Fall eines Enterprise JavaBean (EJB) könnten dafür z. B. Werkzeuge (Parser) zur Analyse der Konformität zur EJB-Spezifikation eingesetzt werden.

Anwendbarkeit Q2: Evaluierung mittels z. B. eines Fragenkatalogs zur Handhabbarkeit der Komponente. Dieser wird potentiellen Benutzern zur Verfügung gestellt.

Effizienz Q3: Verwendung von Methoden der Performanceschätzung, Performancemodelle, Performancebenchmark und entsprechender Profiler-Werkzeuge.

Funktionalität Q4: Durchführung entsprechender Funktionstests im Rahmen der Komponentenentwicklung.

Zuverlässigkeit Q5: Gewinnung von Erfahrungswerten anhand eingesetzter Komponenten innerhalb des Wirkbetriebs.

Wartbarkeit Q6: Software-Messung zur Gewinnung statischer Quellcodemetriken.

In Anlehnung an den von der IEEE vorgegebenen Standard können die grundlegenden Messstrategien der Evaluierung (z. B. Fragebogen), Merkmalsschätzung (z. B. formelbasierte Darstellung potentieller Zusammenhänge), modellbezogene Messung sowie die direkte Messung zum Einsatz kommen.

6.6 Spezifikation der Qualitätseigenschaften (Schritt 4)

Aufgrund der vielfältigen Notationen kann keine primäre Notation vorgeschlagen werden. Dem entsprechend können nur grobe Vorschläge unterbreitet werden wie die gewonnenen Qualitätsmerkmale als Spezifikation der Komponente verwendet werden können.

- Verwendung von Elementen bzw. Diagrammen der UML-Notation
- Verwendung einer formelbasierten Schreibweise
- Hinterlegung der Qualitätseigenschaften innerhalb der Komponente selbst
- Hinterlegung der Qualitätseigenschaften innerhalb eines Repository

7 Terminologieebene

7.1 Zweck

In verschiedenen Ebenen werden bei der Spezifikation von Fachkomponenten Begriffe benutzt, die eine fachliche Bedeutung (Semantik) besitzen. Diese Begriffe haben im Allgemeinen keine einheitliche Intension (Bedeutung bzw. Definition) und sind daher für die eindeutige Verwendung im Rahmen einer Spezifikation zu erklären.

Die Terminologieebene dient als zentrales Begriffsverzeichnis einer Fachkomponente und speichert in einem Lexikon alle Begriffe, die für die Spezifikation von Nutzen sind, sowie deren Definition. Die im Rahmen dieser Ebene dokumentierten Begriffe werden bei der Spezifikation der anderen Ebenen verwendet (so z. B. während der Spezifikation der Aufgaben- oder Vermarktungsebene). Beim Aufbau des Lexikons einer Komponente werden alle (wesentlichen) Fachbegriffe, die Verwendung finden, mit einer Definition aufgeführt. Dies dient der Abgeschlossenheit einer Fachkomponente, die eine Voraussetzung für ihre Vermarktbarkeit ist. Daneben soll ggf. zusätzlich auf einen Standard verwiesen werden können, wenn für einige Begriffe auf einen solchen Bezug genommen wird. Somit wird es auch möglich, einzelne Fachbegriffe bezüglich eines Standards zu redefinieren.

Für die Hersteller von Fachkomponenten sowie die Anwender, die sie in ihren Systemen einsetzen, ist die komponentenübergreifende Verwaltung von Fachtermini notwendig, um ggf. Konflikte und unterschiedliche Definitionen ausfindig zu machen. Diese Aufgabe ist z. B. von Komponentenrepositories, die zur Administration von Fachkomponenten herangezogen werden, zu leisten.

(Fach-)Begriffe und ihre Definitionen lassen sich auf sehr systemnahe Weise, z. B. durch die Verwendung des *Resource Description Framework* (RDF) speichern, einer XML-basierten

Sprache zur Speicherung von Terminologien. Für die Spezifikation der Terminologieebene soll jedoch eine Beschreibung möglichst in einer natürlichen oder für den Anwender leicht lesbaren Sprache vorgeschlagen werden. Oftmals sind nämlich die Experten eines Fachgebiets gefragt, wenn es um die Auswahl einer Komponente auf Grund der von ihr verwendeten Terminologie geht.

Neben dem Lexikon der verwendeten Fachbegriffe lässt sich im Rahmen der Terminologieebene optional auch ein integriertes Attribute- und Dienstschema spezifizieren, das eine Übersicht über die Attribute und Dienste gibt, die eine Komponente verwendet.

Obwohl man zur Spezifikation dieses integrierten Schemas die fachlich-konzeptionelle Ebene verlässt und sich eher einer systemnahen Beschreibung bedient, macht es Sinn, das Schema innerhalb dieser Ebene zu spezifizieren. Zum einen ist es in verschiedenen Ebenen sinnvoll einsetzbar (z. B. in der Abstimmungs-, Verhaltens- und Schnittstellenebene) und ist daher an dieser Stelle, die den Charakter eines zentralen Nachschlagewerkes besitzt, sinnvoll unterzubringen. Zum anderen hängt die Benennung der Attribute und Dienste (im Idealfall) eng mit der verwendeten Fachterminologie zusammen.

Darüber hinaus ist festzuhalten, dass ein solches integriertes Attribute- und Dienstschema keinesfalls die Spezifikation der Innensicht einer Komponente vorwegnehmen, sondern lediglich eine Hilfestellung für die Spezifikation der anderen Ebenen darstellen soll.

7.2 Notationsvorschlag (primäre Notation)

Auf Grund der zuvor genannten Anforderungen wird für die Terminologieebene die Verwendung einer Fachnormsprache als primäre Notation vorgeschlagen (vgl. [Ortn1997] und [Lehm1998]). Sie zeichnet sich dadurch aus, dass sie über ein Lexikon mit geklärten (d. h. definierten) Fachbegriffen verfügt und zur Bildung von Aussagen eine rationale Grammatik (rekonstruierte Grammatik der Umgangssprache) verwendet, die eine Reihe von Satzbauplänen als Schablonen vorgibt.

Bei der Spezifikation der Terminologieebene steht der Aufbau des Lexikons einer Komponente im Vordergrund, das die Fachterminologie speichert. Dies geschieht durch den Einsatz von Methoden zur Definition und Anordnung (Organisation) von Begriffen in einem Lexikon.

Methoden für die Definition sind:

- explizite Definition von Begriffen (unter Vermeidung von Zirkeldefinitionen)
- Prädikatorenregeln, die es erlauben, Begriffe in Beziehung zu einander zu setzen
- Einführung neuer Wörter mit Beispielen und Gegenbeispielen (um das Anfangsproblem für Definitionsverfahren zu lösen)

Methoden für die Organisation von Begriffen sind:

- alphabetische Anordnung
- Kurz- und Langdefinitionen
- Einsatzbeispiele

Zur grafischen Veranschaulichung *konzeptueller* Beziehungen wird die ergänzende Erstellung eines integrierten Attribute- und Dienstschemas als *UML-Klassendiagramm* empfohlen. Durch dessen Verwendung soll dabei jedoch keinesfalls eine objektorientierte Implementierung oder eine spezielle Innensicht der Komponente festgeschrieben werden.

7.3 Beispiel (primäre Notation)

Ein Lexikon, das durch den Einsatz obiger Methoden spezifiziert wird, hat für die Terminologie der Finanzbuchhaltung beispielsweise das in Bild 12 dargestellte Aussehen.

AKTIVA ▪ …
BILANZ ▪ Kurzdefinition: BILANZ $=_{DF}$ die Gegenüberstellung der AKTIVA und PASSIVA einer Unternehmung zu einem bestimmten Zeitpunkt (STICHTAG). ▪ Langdefinition: BILANZ $=_{DF}$ BILANZ ist zusammen mit der GEWINN- UND VERLUSTRECHNUNG Teil von JAHRESABSCHLUSS. BILANZ ist eine Gegenüberstellung der AKTIVA und PASSIVA zu einem bestimmten Zeitpunkt (STICHTAG). BILANZ ist nach den GRUNDSÄTZE ORDNUNGSGEMÄßER BUCHFÜHRUNG zu erstellen. […] ▪ Einsatzbeispiele: HANDELSBILANZ, STEUERBILANZ, ORDENTLICHE BILANZ, SUMMEN- UND SALDENBILANZ ▪ Prädikatoren: $x \in \text{BILANZ} \Rightarrow x \in \text{JAHRESABSCHLUSS}$ $x \in \text{BILANZ} \Rightarrow x \notin \text{GEWINN- UND VERLUSTRECHNUNG}$
GEWINN- UND VERLUSTRECHNUNG: ▪ …

Bild 12: Beispiel für die Spezifikation der Terminologie

Auf die Angabe eines integrierten Attribute- und Dienstschemas soll an dieser Stelle verzichtet werden, da die Modellierung von Komponenten mittels UML-Klassen hinreichend bekannt ist.

7.4 Ergänzende (sekundäre) Notation

Man kann bei der Erstellung eines Lexikons auf die Verwendung normsprachlicher Methoden verzichten und gelangt dann zu einer umgangssprachlichen Fassung eines Lexikons.

Darüber hinaus wäre als sekundäre Notation auch die Verwendung systemnaher Sprachen zur Erfassung von Begriffssystemen denkbar. Als Beispiel hierfür sei das XML-basierte RDF (vgl. [Roth2001]) genannt, das insbesondere für die Bestrebungen unter dem Schlagwort *Semantic Web* Bedeutung erlangt hat.

Im Rahmen dieses Spezifikationsvorschlages soll auf die Bevorzugung einer bestimmten sekundären Notation für das Lexikon verzichtet werden. Ähnliches gilt für das integrierte Attribute- und Dienstschema.

7.5 Beispiel (sekundäre Notation)

Beispiele für eine umgangssprachliche Fassung von Lexika findet man in der entsprechenden Fachliteratur. In Bezug auf die Finanzbuchhaltung finden sich in solchen Nachschlagewerken auch die Termini, die im Rahmen des obigen Beispiels verwendet wurden.

Ein mit RDF spezifiziertes Begriffssystem findet sich z. B. bei [Roth2001].

8 Aufgabenebene

8.1 Zweck

Im Rahmen der komponentenorientierten Entwicklung betrieblicher Anwendungssysteme werden Fachkomponenten zur Erfüllung bzw. Unterstützung betrieblicher Aufgaben spezifiziert. Die Dokumentation dieser von einer Fachkomponente unterstützten betrieblichen Aufgabe sowie ggf. deren Zerlegung in mehrere Teilaufgaben auf fachlichem (konzeptionellem) Niveau bietet eine Reihe von Vorteilen.

Zum einen ergibt sich auf diese Weise in Zusammenhang mit der (in der Vermarktungsebene beschriebenen) Anwendungsdomäne der Einsatzzweck für eine spezielle Fachkomponente. Durch die Spezifikation auf fachlicher Ebene wird es für die entsprechenden Fachleute möglich, die Einsatzmöglichkeiten für eine Komponente in einem konkreten Anwendungsfall zu beurteilen.

Darüber hinaus ist die Spezifikation der betrieblichen Aufgabe (und ihre funktionale Zerlegung in Teilaufgaben) die Basis für eine nachvollziehbare Zuordnung zu den Diensten, die eine Fachkomponente an ihrer Schnittstelle anbietet. Eine solche Zuordnung erfolgt unter Bezugnahme auf diese Ebene im Rahmen der Spezifikation der Schnittstelle, vgl. Abschnitt 3.

Beim Aufbau des Aufgabenverzeichnisses einer Komponente sollten alle betrieblichen Aufgaben, die unterstützt werden, explizit aufgelistet und ggf. in Teilaufgaben untergliedert sein. Dies dient der Abgeschlossenheit einer Fachkomponente, die eine Voraussetzung für ihre Vermarktbarkeit ist. Daneben soll zusätzlich auf einen Standard verwiesen werden können, wenn für einige Aufgaben und deren Untergliederung auf einen solchen Bezug genommen wird. Somit wird es auch möglich, abweichend von Standards einzelne Aufgaben zu redefinieren.

8.2 Notationsvorschlag (primäre Notation)

Die Spezifikation der von einer Fachkomponente unterstützten (betrieblichen) Aufgaben soll auf fachlich-konzeptionellem Niveau erfolgen. Daher ist eine Erfassung in einer für den Anwender leicht lesbaren Sprache zu präferieren. Gleichzeitig sollte die Erfassung in einer klaren Sprache erfolgen, die Missverständnisse in der Dokumentation möglichst ausschließt.

Als primäre Notation wird daher die Spezifikation in einer rekonstruierten Fach-, d. h. in einer der natürlichen ähnlichen, Sprache vorgeschlagen.

Eine solche Fachnormsprache zeichnet sich dadurch aus, dass sie über ein Lexikon mit geklärten (d. h. definierten) Fachbegriffen verfügt und zur Bildung von Aussagen eine rekonstruierte Grammatik verwendet, die eine Reihe von Satzbauplänen als Schablonen vorgibt. Während das Lexikon im Rahmen der Spezifikation der Terminologieebene aufzubauen ist, sind im Rahmen der Aufgabenebene Aussagen über die betriebliche Aufgabe, welche die Komponente unterstützt, sowie deren funktionale Zerlegung zu treffen. Dabei werden alle (wesentlichen) an dieser Stelle verwendeten Fachbegriffe in der Terminologieebene definiert.

Der Aufbau von Fachnormsprachen wird unter anderem in [Ortn1997] und [Lehm1998] ausführlich beschrieben. An dieser Stelle sollen einige Satzbaupläne als Schablonen für die Spezifikation von Aussagen genannt werden:

- Abstraktive Beziehungen (Gleichheit, Vererbung):
  Ein A ist ein B (oder ein C).

- Kompositive Beziehungen (Abhängigkeit):
  Ein A besteht aus / ist Teil von einem B.
- Verteilungszusammenhang:
  Ein A besitzt / verfügt über ein B.
- Operationszusammenhang:
  Ein A tut / erleidet ein B.
- Wandlungszusammenhang:
  Ein A wird zu einem B.
- Reihenfolgezusammenhang:
  Ein A findet vor / nebenläufig zu / nach einem B statt.
- Historienzusammenhang:
  Ein A war vor / muss abgeschlossen sein vor einem B.

Der unbestimmte Artikel „ein“, „einer“, „eines“, … weist hierbei darauf hin, dass es sich um Satzbaupläne zur Bildung allgemeiner Aussagen, aus denen man Schemata ableiten kann, handelt.

8.3 Beispiel (primäre Notation)

Aufbauend auf den zuvor genannten Satzbauplänen können Aussagen über die betriebliche Aufgabe, die von einer Fachkomponente unterstützt wird, spezifiziert werden. Für das Beispiel einer Fachkomponente „Bilanzierung“ (vgl. Bild 13), die im Rahmen einer Finanzbuchhaltung einsetzbar ist, soll davon ausgegangen werden, dass sämtliche Fachbegriffe (Bezeichner in Großbuchstaben) in der Terminologieebene definiert (und damit geklärt) sind.

Eine BILANZIERUNG ist Teil von einem JAHRESABSCHLUSS.
Eine BILANZIERUNG ist nach einer GESETZLICHEN GRUNDLAGE erstellt.
Eine GEWINN- UND VERLUSTRECHNUNG findet vor einer BILANZIERUNG statt.
Eine BILANZIERUNG besteht aus einem EINBUCHEN, einer GEWINNERMITTLUNG und einer DOKUMENTATION.
...

Bild 13: Beispiel für die Spezifikation betrieblicher Aufgaben

8.4 Ergänzende (sekundäre) Notation

Als Alternative zur Spezifikation in einer normierten Fachsprache bleibt die Verwendung der natürlichen Sprache (Prosaform). Entscheidet sich ein Entwickler für diese Notation, greift er implizit auf die Satzbaupläne, die für die jeweilige natürliche Sprache gültig sind, zurück. Um einen gewissen Grad der Einfachheit zu wahren, sollte er dabei jedoch auf Schachtelsätze möglichst verzichten. Natürlich ist es darüber hinaus sinnvoll, verwendete Fachbegriffe (im Rahmen der Terminologieebene) auch weiterhin zu definieren und sich nicht auf ein allgemeines Verständnis zurück zu ziehen.

Auf Grund der Nachteile für die computergestützte Verwaltung solcher Aussagen sollte auf diese Notationsform jedoch möglichst verzichtet werden.

8.5 Beispiel (sekundäre Notation)

Das Beispiel aus Abschnitt 8.3 lässt sich in sekundärer Notation ähnlich darstellen (Bild 14). Allerdings gibt es für einzelne Aussagen keine klaren Schablonen mehr, so dass ein (Computer-)System den Typ einer Aussage nicht mehr automatisch erkennen (und ggf. in eine Diagrammsprache o. ä. umsetzen) kann.

Die BILANZIERUNG ist Teil des Jahresabschlusses.
Sie ist nach den Bestimmungen einer GESETZLICHEN GRUNDLAGE zu erstellen.
Zuvor ist die GEWINN- UND VERLUSTRECHNUNG abzuschließen.
Eine Bilanzierung besteht aus den Teilaufgaben EINBUCHEN, GEWINNERMITTLUNG und DOKUMENTATION.
...

Bild 14: Beispiel für die Spezifikation betrieblicher Aufgaben in sekundärer Notation

9 Vermarktungsebene

9.1 Zweck

Der Zweck der Vermarktungsebene ist es, die wesentlichen Merkmale einer Fachkomponente zu spezifizieren, die prinzipiell benötigt werden, um eine Fachkomponente *betriebswirtschaftlich-organisatorisch* handhabbar zu machen. Diese Ebene umfasst die Merkmale, die Arbeitsgrundlage für den Verkäufer und Einkäufer sowie Assemblierer und Qualitätssicherer von Fachkomponenten sind. Weitere Merkmale von Fachkomponenten, die durchaus für andere Rollen von Relevanz sind, z. B. bestimmte Verhaltenseigenschaften oder Qualitätsmerkmale, werden auf dieser Ebene nicht behandelt. Die Merkmale auf dieser Ebene umfassen drei Bereiche [Kauf2000]:

- Betriebswirtschaftlich-semantische Merkmale: Merkmale, die betriebswirtschaftlich-semantische Eigenschaften der Anwendungsdomäne einer Fachkomponente beschreiben.
- Technische Merkmale: Merkmale, die technische Randbedingungen des Betriebs und der Nutzung der Fachkomponenten beschreiben.
- Sonstige Merkmale: Merkmale, die weder betriebswirtschaftlich-semantischen noch technischen Merkmalen sinnvoll zugeordnet werden können.

9.2 Notationsvorschlag (primäre Notation)

Als primäre Notation wird die Tabellenform vorgeschlagen. Die einzelnen zu spezifizierenden Attribute werden im Folgenden näher erläutert. Es gelten die Konventionen:

- Jeder Tabelleneintrag nennt den Name des Attributes. Dieser ist **fett** gesetzt.
- Falls das Attribut nur optional anzugeben ist, wird es in eckigen Klammern gesetzt. Beispiel: [optionales Attribut]

- Falls das Attribut wiederholt angegeben werden kann, wird es in geschweiften Klammern gesetzt. Beispiel {wiederholbares Attribut}

Name Ein alphanumerischer Name, unter dem die Komponente vermarktet wird.
[Kennung] Eine eindeutige Kennung zur Identifizierung der Komponente.
Version Dieses Merkmal definiert Version sowie Release-Stand einer Komponente.
Wirtschaftszweig Die Angabe des Wirtschaftszweigs gibt ergänzende Informationen, in welchen volkswirtschaftlichen Bereichen eine Fachkomponente zum Einsatz kommen kann. Grundlage zur Definition der Wirtschaftszweige ist die Klassifikation des Statistischen Bundesamtes [Stat1993]. Es sind folgende Ausprägungen möglich: A Land- und Fortwirtschaft; B Fischerei und Fischzucht; C Bergbau und Gewinnung von Steinen und Erden; D Verarbeitendes Gewerbe; E Energie- und Wasserversorgung; F Baugewerbe; G Handel, Instandhaltung und Reparatur von Kraftfahrzeugen und Gebrauchsgütern; H Gastgewerbe; I Verkehr und Nachrichtenübermittlung; J Kredit- und Versicherungsgewerbe; K Grundstücks- und Wohnugswesen, Vermietung beweglicher Sachen, Erbringung von Dienstleistungen überwiegend für Unternehmen; L Öffentliche Verwaltung, Verteidigung, Sozialversicherung; M Erziehung und Unterricht; N Gesundheits-, Veterinär- und Sozialwesen; O Erbringung von sonstigen öffentlichen und persönlichen Dienstleistungen; P Private Haushalte; Q Exterritoriale Organisationen und Körperschaften. Es ist möglich, mehrere oder alle Wirtschaftszweige zu selektieren sowie die Auswahl „unabhängig von Wirtschaftszweigen“ zu selektieren. Alternativ kann die International Standard Industrial Classification of All Economic Activities [UNSD1989] zum Einsatz kommen.
Domäne Ferner wird zur groben fachlichen Einordnung der Dienste, die durch eine Komponente erbracht werden, ein Funktionalbereich definiert. Die Abgrenzung des Funktionalbereichs wird in [Mert2000] definiert. Hierbei wird sich auf die oberste Ebene des Funktionenmodells beschränkt. Es können folgende Ausprägungen gewählt werden: Forschung sowie Produkt- und Prozessentwicklung; Vertriebssektor; Beschaffungssektor; Lagerhaltungssektor; Produktionssektor; Versandsektor; Kundendienstsektor; Finanzsektor; Sektor Rechnungswesen; Personalsektor; Gebäudemanagement; funktionsbereich- und prozessübergreifende Integrationskomplexe. Es ist möglich, mehrere oder alle Funktionalbereiche zu selektieren.
Lieferumfang Per Definition besteht eine Komponente aus verschiedenartigen (Software-)Artefakten. Zweck des Lieferumfangs ist es zu spezifizieren, aus welchen (Software-)Artefakten eine Komponente besteht. Der Lieferumfang ist für mehrere Rollen im Software-Entwicklungsprozess eine wichtige Grundlage, um die ihnen jeweils zugeteilten Aufgaben fachgerecht und vollständig ausführen zu können. Im Rahmen der Komponentenproduktion erlaubt der Lieferumfang dem Qualitätssicherer überprüfen zu können, ob ein von einer Komponente erstelltes Release gemäß der in dem Lieferumfang definierten (Software-)Artefakte vollständig ist, ob die angegebenen (Software-)Artefakte im Release vorhanden sind oder ob (Software-)Artefakte fälschlicherweise dem Release zugefügt worden sind. Im Rahmen der Komponentenkonsumtion ist der Lieferumfang Grundlage für die Installation der Komponente. Zunächst kann ein Assemblierer überprüfen, ob alle in dem Lieferumfang angegebenen (Software-)Artefakte vorhanden sind. Darüber hinaus erläutert der Lieferumfang dem Assemblierer den Sinn und Zweck der verschiedenen (Software-)Artefakte, die zu einer Komponente gehören.

Komponententechnologie Die Beschreibung der verwendeten Komponententechnologie sowie des entsprechende Versionsstands, die einer Komponente zugrunde liegen.
{Systemanforderung} Über dieses Merkmal wird die Systemanforderung der Komponente für eine Zielplattform bzw. Systemkonfiguration beschrieben. Eine Systemkonfiguration umfasst im einzelnen: Prozessorarchitektur, Hauptspeichergröße, Plattenspeichergröße, Betriebssystem, Betriebssystemversion, Komponenten-Anwendungs-Framework und Komponenten-System-Framework (Middleware, Datenbanksystem etc.).
[Hersteller] Dieses Merkmal erlaubt, den Hersteller der Komponente eindeutig zu identifizieren. Dies ist sowohl wichtig, um mit dem Hersteller in Kontakt zu treten, als auch entsprechende weitere Informationen über den Hersteller einzuholen. Derartige weiterführende Informationen, die von freien Auskunfteien vorgehalten werden können, sind: Börsenkurse, Gewinne, Umsätze und deren Entwicklung, wirtschaftliche Entwicklungen, Reputation, Zahl der Installationen, Referenzkunden sowie Erfahrungsberichte von Kunden. [Kauf2000]
[Ansprechpartner] Dieses Merkmal nennt einen ersten Ansprechpartner, falls ein potentieller Käufer der Komponente diese vom Hersteller erwerben möchte.
[Vertragliche Konditionen] Über dieses Merkmal kann definiert werden, zu welchen Modalitäten die Komponente beim Hersteller erworben werden kann. Diese werden definiert durch Preise (Kosten pro Lizenz, pro Benutzer, etc.), Leistungsumfänge, Allgemeine Geschäftsbedingungen, Preise für Service-Verträge, eventuell mögliche Beratungen sowie Schulungen, Zahlungsbedingungen, Gerichtsstand etc.
[Freitext] Über dieses Merkmal können weitere Eigenschaften der Komponente spezifiziert werden, die von Interesse sind. Dies könnten z. B. Merkmale wie Mandantenfähigkeit, Mehrbenutzerfähigkeit o. ä. sein.

9.3 Beispiele (Primäre Notation)

Im Folgenden wird eine exemplarische Spezifikation für eine Fachkomponente „Bankleitzahlen“ angegeben.

Name	
Bankleitzahlen	
Version	
V 1.0	
Wirtschaftszweig	
Unabhängig von Wirtschaftszweigen	
Domäne	
Finanzsektor	
Lieferumfang	
bankcodes.jar:	Java-Class-Dateien der Implementierung
bankcodes.tws:	Komponenten-IDL für Middlware Twister von Brokat
create_db.sql:	SQL-Skript zur Erstellung der Oracle-Datenbank zum Speichern der Daten-

	basis
blz0010pc.txt:	Originaldatenquelle der Bankleitzahlen der Deutschen Bundesbank (Quelle: http://www.bundesbank.de/)
SATZ188.doc:	Beschreibung der Originaldatenquelle blz0010pc.txt (Quelle: http://www.bundesbank.de/)
script_sampledata.pl:	Perl-Skript zum Konvertieren des Datenfiles der Deutschen Bundesbank in entsprechende SQL-Anweisungen
bankcodestests.jar:	Beispiel-Client-Implementierungen zur Überprüfung der Komponentenfunktionalität

Komponententechnologie

Brokat Twister 2.3.5

Systemanforderung

Prozessorarchitektur:	x86
Hauptspeichergröße:	512 MB
Festplattengröße:	10 MB
Betriebssystem:	Windows NT 4.0, SP 3
Datenbanksystem:	Oracle 8i
Komponenten-System-Framework:	Brokat Twister 2.3.5

Hersteller

Technische Universität Chemnitz, Professur Wirtschaftsinformatik II, D-09107 Chemnitz

Ansprechpartner

Peter Fettke, +49/371/531-4362, peter.fettke@isym.tu-chemnitz.de

Vertragliche Konditionen

Allgemeine Lizenzbedingungen liegen nicht vor, sondern müssen jeweils im Einzelfall ausgehandelt werden. Prinzipiell werden Forschungseinrichtungen oder forschungsinteressierte Industrieunternehmen angesprochen.

9.4 Alternative Notationen, Entscheidungsdeterminanten, Ausblick

Aus Gründen der guten Lesbarkeit und leichten Erstellbarkeit wurde die Tabellenform als primäre Notation der Vorrang gegenüber einer formaleren Notation eingeräumt. Zur Realisierung der Spezifikationselemente dieser Ebene wird ergänzend von [FeLo2001] ein Ansatz vorgeschlagen, der auf der *Extensible Markup Language* (XML) und einer entsprechend spezifizierten *Document Type Definition* (DTD) beruht. Weitere Merkmale, die für die Vermarktungsebene von Fachkomponenten von Relevanz sind, werden in [Kauf2000, S. 110-115], [Ohle1998, S. 135-137] und [W3C1997] beschrieben.

Literatur

[Acke2001] *Ackermann, J.*: Fallstudie zur Spezifikation von Fachkomponenten. In: *K. Turowski (Hrsg.)*: Modellierung und Spezifikation von Fachkomponenten: 2. Workshop. Bamberg 2001, S. 1-66.

[Balz1998] *Balzert, H.*: Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung. Spektrum Akademischer Verlag, Heidelberg 1998.

[BiMR1991] *Biethahn, J.; Mucksch, H.; Ruf, W.*: Ganzheitliches Informationsmanagement: Daten- und Entwicklungsmanagement. Bd. 2, Oldenbourg, München 1991.

[CCM+2001] *Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S.* Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, Abruf am 2002-02-06.

[CoTu2000] *Conrad, S.; Turowski, K.*: Vereinheitlichung der Spezifikation von Fachkomponenten auf der Basis eines Notationsstandards. In: *J. Ebert; U. Frank (Hrsg.)*: Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik: Beiträge des Workshops "Modellierung 2000" St Goar, 5.-7. April 2000. St. Goar 2000, S. 179-194.

[Dumk2001] *Dumke, R.*: Software Engineering - Eine Einführung für Informatiker und Ingenieure: Systeme, Erfahrungen, Methoden, Tools. Vieweg-Verlag, Braunschweig 2001.

[DuFS2000] Dumke, R.; Foltin, E.; Schmietendorf, A. (2000): Metriken-Datenbanken in der Informationsverarbeitung. Preprint der Fakultät für Informatik Nr. 8, Otto-von-Guericke-Universität Magdeburg. Magdeburg.

[FeRT1999] *Fellner, K.; Rautenstrauch, C.; Turowski, K.*: Fachkomponenten zur Gestaltung betrieblicher Anwendungssysteme. In: IM Information Management & Consulting 14 (1999) 2, S. 25-34.

[FeLo2001] *Fettke, P.; Loos, P.*: Ein Vorschlag zur Spezifikation von Fachkomponenten auf der Administrations-Ebene. In: *K. Turowski (Hrsg.)*: Modellierung und Spezifikation von Fachkomponenten: 2. Workshop. Bamberg 2001, S. 95-104.

[Kauf2000] *Kaufmann, T.*: Entwurf eines Marktplatzes für heterogene Komponenten betrieblicher Anwendungssysteme. Berlin 2000.

[Kern1993] *Kernler, H.*: PPS der 3. Generation: Grundlagen, Methoden, Anregungen. Hüthig Buch Verlag, Heidelberg 1993.

[Lehm1998] *Lehmann, F. R.*: Normsprache. Das aktuelle Schlagwort. In: Informatik-Spektrum 21 (1998) 5, S. 360-367.

[McIl1968] *Mcllroy, M. D.*: Mass Produced Software Components. In: *P. Naur; B. Randell (Hrsg.)*: Software Engineering: Report on a Conference by the NATO Science Comittee. NATO Scientific Affairs Division, Brussels 1968, S. 138-150.

[Mert2000] *Mertens, P.*: Integrierte Informationsverarbeitung 1: Administrations- und Dispositionssysteme in der Industrie. 12. Aufl., Gabler, Wiesbaden 2000.

[Ohle1998] *Ohlendorf, T.*: Architektur betrieblicher Referenzmodellsysteme - Konzept und Spezifikation zur Gestaltung wiederverwendbarer Norm-Software-Bausteine für die Entwicklung betrieblicher Anwendungssysteme. Aachen 1998.

[OMG2001a] *OMG (Hrsg.)*: The Common Object Request Broker: Architecture and Specification: Version 2.5, September 2001. OMG, Framingham 2001a.

[OMG2001b] *OMG (Hrsg.)*: Unified Modeling Language Specification: Version 1.4, September 2001. OMG, Needham 2001b.

[Ortn1997] *Ortner, E.*: Methodenneutraler Fachentwurf: Zu den Grundlagen einer anwendungsorientierten Informatik. Teubner, Stuttgart 1997.

[OrRS1990] *Ortner, E.; Rößner, J.; Söllner, B.*: Entwicklung und Verwaltung standardisierter Datenelemente. In: Informatik-Spektrum 13 (1990) 1, S. 17-30.

[Roth2001] *Rothfuss, G.*: Content Management mit XML. Springer, Berlin 2001.

[Sahm2000] *Sahm, S.*: Organisationsmodelle zur komponentenorientierten Anwendungsentwicklung / terminologiebasierte Spezifikation von Fachkomponenten. In: *K. Turowski (Hrsg.)*: Modellierung und Spezifikation von Fachkomponenten: Workshop im Rahmen der MobIS 2000 Modellierung betrieblicher Informationssysteme, Siegen, Deutschland, 12. Oktober 2000, Tagungsband. Siegen 2000, S. 17-40.

[ScDu2001] *Schmietendorf, A.; Dumke, R.*: Spezifikation von Softwarekomponenten auf Qualitätsebene. In: *K. Turowski (Hrsg.)*: Modellierung und Spezifikation von Fachkomponenten: 2. Workshop. Bamberg 2001, S. 113-123.

[ScSc2000] *Schmietendorf, A.; Scholz, A.*: Spezifikation der Performance - Eigenschaften von Softwarekomponenten. In: *K. Turowski (Hrsg.)*: Modellierung und Spezifikation von Fachkomponenten: Workshop im Rahmen der MobIS 2000 Modellierung betrieblicher Informationssysteme, Siegen, Deutschland, 12. Oktober 2000, Tagungsband. Siegen 2000, S. 41-49.

[SoBe1999] *Solingen, v. R.; Berghout, E.*: The Goal/Question/Metric Method. McGraw Hill, 1999.

[Stat1993] *Statistisches Bundesamt (Hrsg.)*: Klassifikation der Wirtschaftszweige, Ausgabe 1993 (WZ 93). http://www.destatis.de/allg/d/klassif/wz93.htm, Abruf am 2002-02-15.

[Turo1999] *Turowski, K.*: Standardisierung von Fachkomponenten: Spezifikation und Objekte der Standardisierung. In: *A. Heinzl (Hrsg.)*: 3. Meistersingertreffen. Schloss Thurnau 1999.

[Turo2001] *Turowski, K.*: Spezifikation und Standardisierung von Fachkomponenten. In: Wirtschaftsinformatik 42 (2001) 3.

[UNSD1989] *United Nations Statistics Division (Hrsg.)*: International Standard Industrial Classification of All Economic Activities, Third Revision, (ISIC, Rev.3). http://esa.un.org/unsd/cr/family2.asp?Cl=2, Abruf am 2002-02-15.

[W3C1997] *World Wide Web Consortium (Hrsg.)*: The Open Software Description Format (OSD). http://www.w3.org/TR/NOTE-OSD.html, Abruf am 2001-04-25.

Verfasser :

Jörg Ackermann

Von-der-Tann-Str. 42, 69126 Heidelberg
E-Mail: joerg.ackermann.hd@t-online.de

Dr. Frank Brinkop

iteratec Gesellschaft für iterative Softwaretechnologien mbH
Inselkammerstr. 4, 82008 München-Unterhaching
Phone: +49(89)614551-0, Fax: -10
E-Mail: frank.brinkop@iteratec.de

Prof. Dr. Stefan Conrad

Ludwig-Maximilians-Universität München
Institut für Informatik
Oettingenstr. 67, 80538 München
E-Mail: conrad@informatik.uni-muenchen.de

Peter Fettke

Technische Universität Chemnitz
Fakultät für Wirtschaftswissenschaften
Information Systems & Management
09107 Chemnitz
Phone: +49(371)531-4375, Fax: -4376
E-Mail: peter.fettke@isym.tu-chemnitz.de

Andreas Frick

ExperTeam AG
Niederlassung Dortmund
Emil-Figge-Straße 85, 44227 Dortmund
Phone: +49(231)9704-292/(-200), Fax: -299
E-Mail: andreas.frick@experteam.de

Dr. Elke Glistau

Otto-von-Guericke-Universität Magdeburg
Fakultät Maschinenbau
Universitätsplatz 2, 39106 Magdeburg
Phone: +49(391)671-2660
E-Mail: elke.glistau@mb.uni-magdeburg.de

Holger Jaekel

Oldenburger Forschungsinstitut für Informatik-Werkzeuge und -Systeme (OFFIS)
Escherweg 2, 26121 Oldenburg
Phone: +49(441)9722-125
E-Mail: holger.jaekel@offis.de

Otto Kotlar

Rohmer Str. 24, 60486 Frankfurt
Phone: +49(69)0795602
E-Mail: otto_kotlar@yahoo.de

Prof. Dr. Peter Loos

Technische Universität Chemnitz
Fakultät für Wirtschaftswissenschaften
Information Systems & Management
09107 Chemnitz
Phone: +49(371)531-4375, Fax: -4376
E-Mail: loos@isym.tu-chemnitz.de

Prof. Dr. Heike Mrech

Fachhochschule Merseburg
Fachbereich Maschinenbau
Geusaer Str., 06217 Merseburg
Phone: +49(3461)46-3027
E-Mail: heike.mrech@mb.fh-merseburg.de

Prof. Dr. Erich Ortner

Technische Universität Darmstadt
Fachbereich Rechts- und Wirtschaftswissenschaften
Wirtschaftsinformatik I
Hochschulstr. 1, 64289 Darmstadt
Phone: +49(6151)16-4309; Fax: -4301
E-Mail: ortner@bwl.tu-darmstadt.de

Dr. Ulrich Raape

Fraunhofer Institut für Fabrikbetrieb und -automatisierung IFF
Sandtorstraße 22, 39106 Magdeburg
Phone: +49(391)4090-359; Fax: -93359
E-Mail: ulrich.raape@iff.fhg.de

Sven Overhage

Technische Universität Darmstadt
Fachbereich Rechts- und Wirtschaftswissenschaften
Wirtschaftsinformatik I
Hochschulstr. 1, 64289 Darmstadt
Phone: +49(6151)16-4309; Fax: -4301
E-Mail: overhage@bwl.tu-darmstadt.de

Stephan Sahm

itelligence AG
Westendstraße 16-22, 60325 Frankfurt am Main
E-Mail: stephan.sahm@itelligence.de

Dr. Andreas Schmietendorf

System- und Technologieentwicklung Bereich EP2
T-Systems Nova GmbH
Entwicklungszentrum Berlin
Wittestraße 30 N, 13509 Berlin
Phone: +49(30)43577-7531; Fax: -7507
E-Mail: andreas.schmietendorf@t-systems.com

Thorsten Teschke

Oldenburger Forschungsinstitut für Informatik-Werkzeuge und -Systeme (OFFIS)
Escherweg 2, 26121 Oldenburg
Phone: +49(441)9722-216
E-Mail: thorsten.teschke@offis.de

Prof. Dr. Klaus Turowski

Lehrstuhl für Betriebswirtschaftslehre, insbesondere Wirtschaftsinformatik II
Universität Augsburg
Universitätsstraße 16, 86135 Augsburg
Phone: +49(821)598-4431; Fax : -4432
E-Mail: klaus.turowski@wiwi.uni-augsburg.de
URL: http://wi2.wiwi.uni-augsburg.de