Spezifikation von Fachkomponenten mit der UML 2.0

Jörg Ackermann

Universität Augsburg, Universitätsstraße 16, 86135 Augsburg, Deutschland, E-Mail: joerg.ackermann.hd@t-online.de

Zusammenfassung. Die UML 2.0 bietet (im Vergleich zur UML 1.4) bessere Möglichkeiten zur Modellierung von Komponenten. Dieser Beitrag stellt zunächst die relevanten UML 2.0 Konzepte vor. Danach wird diskutiert, welche Auswirkungen sich daraus auf das Memorandum „Vereinheitlichte Spezifikation von Fachkomponenten“ ergeben und wie das Memorandum entsprechend weiterentwickelt werden könnte.

Schlüsselworte: Fachkomponente; (Software-)Komponente; Formale Spezifikation; UML 2.0

Die präzise und geeignete Spezifikation von Fachkomponenten ist eine wesentliche Voraussetzung, damit sich der Bau von Anwendungssystemen aus am Markt gehandelten Softwarekomponenten etablieren kann. Wichtig ist dabei vor allem die Standardisierung der Komponentenspezifikationen, damit diese von allen beteiligten Parteien richtig verstanden werden. Die Arbeitsgruppe 5.10.3. der Gesellschaft für Informatik hat einen Vorschlag erstellt, wie Aufbau und Inhalt einer Spezifikation aussehen sollten. Für weitere Details siehe das Memorandum „Vereinheitlichte Spezifikation von Fachkomponenten“ [Turo2002].

Das Memorandum verwendet auf der Verhaltens- und der Abstimmungsebene Elemente der Unified Modeling Language (UML), insbesondere die Object Constraint Language (OCL). Grundlage für das Memorandum war die damals verfügbare Version UML 1.4 [OMG2001]. Mittlerweile wurde die Version UML 2.0 [OMG2003] entwickelt, die kurz vor der Verabschiedung steht. UML 2.0 bietet deutlich bessere Möglichkeiten zur Modellierung und Spezifikation von Komponenten. Diese werden in diesem Beitrag vorgestellt und es wird diskutiert, welche Möglichkeiten sich daraus für das Memorandum ergeben.

Im Kapitel 1 werden die Komponentenbegriffe in der UML und im Memorandum vorgestellt und miteinander verglichen. Im Kapitel 2 werden die für Komponenten relevanten Konzepte von UML 2.0 vorgestellt. Im Kapitel 3 wird diskutiert, wie die UML 2.0 in einer weiterentwickelten Fassung des Memorandums berücksichtigt werden könnte. Abschließend stellt das Kapitel 4 noch zwei Neuerungen in der OCL vor und zeigt, wie diese die Spezifikation von Fachkomponenten vereinfachen.

1 Komponentenbegriffe in der UML und im Memorandum

In diesem Abschnitt wollen wir zunächst die Komponentenbegriffe in der UML und im Memorandum vorstellen und diskutieren.

In der UML 2.0 wird eine Komponente folgendermaßen definiert [OMG2003, S. G-574]:

> A component is a “modular part of a system that encapsulates its contents and whose

> manifestation is replaceable within its environment. A component defines its behavior in terms of provided and required interfaces. …"

Im Vergleich dazu betrachten wir die Definition aus dem Memorandum [Turo2002, S. 1]:

> „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."

Wir stellen dabei fest, dass beide Definitionen in wesentlichen Punkten übereinstimmen:

- Abgeschlossenheit (in UML bezeichnet als „modular part"),
- Kapselung (im Memorandum bezeichnet als „verbirgt ihre Realisierung"),
- Verwendung wohldefinierter Schnittstellen.

Die Unterschiede in den Definitionen ergeben sich (nach Meinung des Autors) vor allem aus der unterschiedlichen Sichtweise:

- UML betrachtet Komponenten als Teile von Systemen und definiert daher Komponenten in Bezug auf Systeme (Teil eines Systems, austauschbar).
- Das Memorandum fokussiert mehr auf die Komponenten an sich (wiederverwendbar, kombinierbar) und bringt zusätzlich betriebswirtschaftliche Aspekte (vermarktbar) ein.

Da beide Definitionen auf technischer Ebene weitgehend übereinstimmen, können nach Meinung des Autors UML-Komponenten zur Modellierung und Spezifikation von Komponenten gemäß des Memorandums eingesetzt werden.

Bei der Modellierung von Komponenten ist es sinnvoll, semantisch zwischen drei verschiedenen Arten von Modellen zu unterscheiden: (software-unabhängige) konzeptionelle Modelle, Spezifikationsmodelle und Implementierungsmodelle (vergleiche [Fowl1999]). Diese Modellarten fokussieren auf unterschiedlichen Aspekten bei der Beschreibung einer Komponente und ihres Umfelds.

Die UML unterscheidet zwischen logischen Komponenten (z.B. Businesskomponenten, Prozesskomponenten) und physischen Komponenten (z.B. EJB-Komponente, CORBA-Komponente, COM+-Komponente, .NET-Komponente). Leider wird diese Unterscheidung nicht näher erläutert. Aber anhand der angegebenen Beispiele kann man davon ausgehen, dass in konzeptionellen Modellen und in Spezifikationsmodellen logische Komponenten verwendet werden, während physische Komponenten in Implementierungsmodellen zu finden sind.

Im Memorandum erfolgt explizit keine weitere Unterscheidung von Komponenten wie in UML. Bei der Spezifikation von Fachkomponenten steht allerdings die Außensicht der Komponente und nicht ihre Implementierung im Vordergrund. Beim Einsatz von Modellen im Memorandum sollte es sich daher um Spezifikationsmodelle handeln und es sollten logische Komponenten verwendet werden.

2 Komponenten in der UML 2.0

Schon die UML 1.4 enthielt das Modellierungskonstrukt *Komponente*. Dieses war jedoch nur

für die Modellierung von physischen Komponenten vorgesehen. Für die Modellierung von logischen Komponenten enthielt die UML 1.4 keine explizite Unterstützung, weshalb verschiedene Autoren unterschiedliche Modellierungskonstrukte zur Beschreibung logischer Komponenten heranzogen. So wurden z.B. Subsysteme [Andr2003, S. 27] oder Klassen mit dem Stereotyp «comp spec» [ChDa2001] verwendet.

Die bedeutendste Änderung in der UML 2.0 besteht darin, dass auch logische Komponenten mit dem Konstrukt *Komponente* modelliert werden können. Dadurch ergeben sich deutlich bessere Möglichkeiten zur verständlichen und konsistenten Spezifikation von Komponenten.

Die wichtigsten Konstrukte und deren Verwendung werden nachfolgend dargestellt:

- Eine Komponente wird als ein Rechteck mit dem Schlüsselwort «component» dargestellt.
- Eine Komponente ist im UML Metamodell ein Subtyp von Klasse. Eine Komponente kann daher Attribute und Methoden haben. Außerdem kann eine Komponente optional eine interne Struktur haben und eine Menge von Ports zur Verfügung stellen.

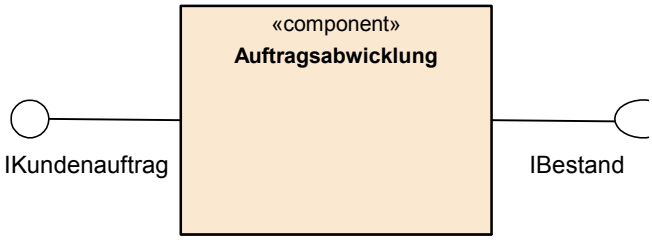

Bild 1: Komponente mit Schnittstellen

- Die externe Sicht einer Komponente (Black-Box-Sicht) besteht aus der Komponente und ihren Schnittstellen. Im Bild 1 sieht man die Darstellung einer Komponente *Auftragsabwicklung* mit den Schnittstellen *IKundenauftrag* und *IBestand*.
- Eine Komponente kann mehrere Schnittstellen haben. Dabei wird zwischen bereitgestellten und benötigten Schnittstellen unterschieden. Im Bild 1 werden exemplarisch eine bereitgestellte (*IKundenauftrag*) und eine benötigte Schnittstelle (*IBestand*) dargestellt.

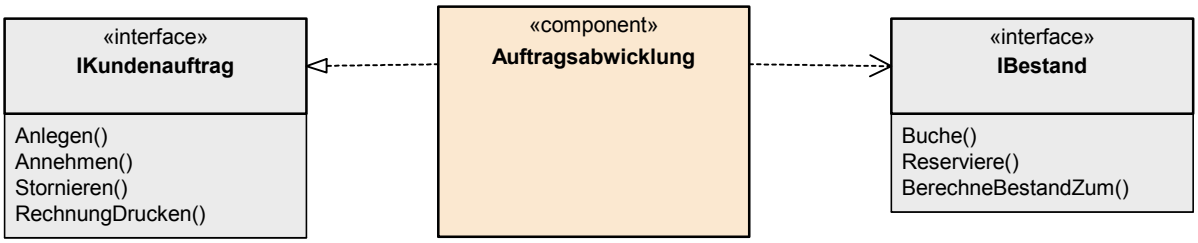

Bild 2: Detaillierte Darstellung der Komponenten-Schnittstellen

- Sollen die Schnittstellen detaillierter dargestellt werden, kann man dafür das Konstrukt

Schnittstelle («interface») verwenden. Die Schnittstelle kann inklusive ihrer Operationen (mit Parametern) und eventuell vorhandener Attribute modelliert werden. So sieht man im Bild 2, dass die Schnittstelle *IKundenauftrag* die Operationen *Anlegen*, *Annehmen*, *Stornieren* und *RechnungDrucken* zur Verfügung stellt (auf die Angabe der Parameter wurde an dieser Stelle verzichtet).

- Optional kann eine Komponente mit Ports versehen werden. Ein Port ist eine Schnittstelle mit eigenem Bezeichner. Mit Ports kann man z.B. verschiedene Zugangspunkte zu einer Komponente unterscheiden, die technisch dieselbe Schnittstelle verwenden.
- Optional kann ein Zustandsdiagramm mit einer Schnittstelle (oder einem Port) verbunden werden, welches dynamische Bedingungen (Aufrufreihenfolge von Operationen) expliziert.
- Von der internen Sicht einer Komponente (White-Box-Sicht) wird gesprochen, wenn die interne Struktur und die Realisierung einer Komponente dargestellt werden.

3 Erweiterungsvorschläge für das Memorandum

Um die Spezifikation von Fachkomponenten anhand des Memorandums einem möglichst breiten Personenkreis zugänglich zu machen, werden im Memorandum bewusst allgemein bekannte Notationstechniken vorgeschlagen. Deshalb werden im Memorandum auf der Verhaltens- und der Abstimmungsebene Elemente der UML verwendet.

Die Spezifikation und Modellierung von Komponenten ist derzeit ein wichtiges Forschungsthema und wird daher von vielen Autoren untersucht. Entsprechend gab es in den letzten 2-3 Jahren einiges an neuen Entwicklungen und Erkenntnissen. Eines der Standardwerke zur Modellierung von Komponenten mit UML ist [ChDa2001]. Viele Ansätze daraus sind in die Version UML 2.0 eingeflossen.

Das Memorandum sollte die UML-Version 2.0 berücksichtigen und Gebrauch von ihren Erweiterungen machen. Der Hauptgrund für diesen Vorschlag liegt darin, dass das Memorandum nur so weiterhin seinem Anspruch gerecht werden kann, weit verbreitete und bekannte Notationstechniken zu verwenden. Daneben ergeben sich auch einige Verbesserungen und Vereinfachungen bei der Spezifikation von Fachkomponenten.

Auf dieser Zielsetzung aufbauend unterbreiten wir die folgenden Vorschläge zur Weiterentwicklung des Memorandum. Alle beziehen sich jeweils auf die primäre Notationstechnik.

1. Allgemein:
   - Eine Komponenten-Spezifikation sollte ein Diagramm enthalten, welches die externe Sicht der Komponente darstellt. Dieses zeigt neben der Komponente auch die bereitgestellten und benötigten Schnittstellen. Vergleiche dazu Bild 1. Dieses Diagramm gilt für die gesamte Spezifikation und wird keiner Ebene zugeordnet.
   - Dienste, welche die Komponente von anderen Komponenten benötigt, werden als benötigte Schnittstellen modelliert. Die Bezeichnung und die Spezifikation solcher benötigter Schnittstellen erfolgt aus Sicht der Komponente selbst. (Man beachte, dass verschiedene andere Komponenten die benötigten Dienste zur Verfügung stellen können. Außerdem wird zum Zeitpunkt der Spezifikation im allgemeinen nicht bekannt

sein, welche konkrete Komponente dies überhaupt sein wird. Damit ist es nicht sinnvoll, bei den benötigten Diensten auf konkrete Schnittstellen konkreter Komponenten zu verweisen.) Anhand des Diagramms kann man erkennen, welche Interfaces eine Komponente zur Verfügung stellt und welche von ihr benötigt werden. Daher kann auf die Namenskonvention *Extern* verzichtet werden, die im Memorandum bisher als Workaround verwendet wurde.

- Eine Komponente kann in UML mehrere Interfaces haben. Daher sollte sich nicht, wie im Memorandum als Beispiel zur Schnittstellenebene angegeben, der Name der Fachkomponente aus dem Namen eines Interfaces ergeben.

2. Schnittstellenebene:
   - Hier sollte ebenfalls die UML zum Einsatz kommen. In einem speziellen Diagramm (vergleiche Bild 2 für eine verkürzte Darstellung) werden die Schnittstellen der Komponente detailliert beschrieben. Das Diagramm enthält für alle Schnittstellen die zugehörigen Dienste (Operationen) mit ihren Parametern und Ausnahmen.
   - Bisher gab es durch die Verwendung der OMG IDL auf der Schnittstellenebene und der UML OCL auf der Verhaltensebene einen Methodenbruch [Acke2001], der u.a. dazu führte, dass die Spezifikationsobjekte der verschiedenen Ebenen nicht eindeutig einander zugeordnet werden konnten. Dieses Problem wurde bisher durch Namenskonventionen umgangen. Durch die Verwendung der UML auf der Schnittstellenebene würde dieses Problem behoben. Außerdem könnte auf eine Notationstechnik verzichtet werden.
   - Falls weiterhin eine Schnittstellenbeschreibung mit der OMG IDL benötigt wird, sollte untersucht werden, ob diese aus der UML-Schnittstellenbeschreibung abgeleitet werden kann. Dann könnte man, wenn benötigt, die Beschreibung in OMG IDL automatisch generieren.
   - Durch diesen Vorschlag würde auf der Schnittstellen-, der Verhaltens- und der Abstimmungsebene einheitlich die UML (zusammen mit der OCL) verwendet. Dies passt auch sehr gut zum Vorschlag von Overhage, einen thematischen Bereich *Technik* mit diesen drei Ebenen zu bilden. [Over2002]

3. Verhaltensebene
   - Manche Komponenten verwalten Geschäftsdaten, die im Zusammenhang mit der Funktionalität und den Diensten der Komponente stehen. So könnte z.B. eine Komponente *Auftragsabwicklung* alle bisher erfassten Kundenaufträge speichern und verwalten. Die Gesamtheit der verwalteten Daten zu einem Zeitpunkt wird oft auch als interner State der Komponente bezeichnet. Siehe dazu z.B. [ChDa2001, S. 123ff.]. Verwaltet die Komponente Geschäftsdaten, so muss auf der Verhaltensebene die Abhängigkeit der Dienste vom internen State (und umgekehrt) spezifiziert werden. So kann ein Dienst *IKundenauftrag.Stornieren* nur Aufträge stornieren, die der Komponente bekannt sind. Oder ein Dienst *IKundenauftrag.Anlegen* wird, wenn erfolgreich ausgeführt, einen neuen Auftrag in der Komponente anlegen.
   - In der Fallstudie [Acke2001] wurde vorgeschlagen, ein UML-Klassendiagramm zur konzeptionellen Beschreibung des internen States zu verwenden. Dies wurde im Memorandum aufgegriffen, in dem es optional ein UML-Klassendiagramm als konzeptionelles Schema vorsieht, welches u.a. zur Abbildung des internen States der

Komponente verwendet werden kann.

- Mit der UML 2.0 kann das konzeptionelle Schema passender als interne Sicht der Komponente modelliert werden. Die zur Beschreibung des States notwendigen Entitäten werden als Klassen mit dem Standard-Stereotyp «type» dargestellt. Dieses Modell dient als eine Art Interface Information Model (vergleiche z.B. [ChDa2001]) und dient als Grundlage, um Zusicherungen mit der OCL zu formulieren. Im Bild 3 ist ein konzeptionelles Schema für die Komponente *Auftragsabwicklung* abgebildet. Dieses enthält die Entitäten *Kundenauftrag* und *Kunde*, deren Eigenschaften (Attribute und Assoziationen) in OCL-Ausdrücken verwendet werden können.

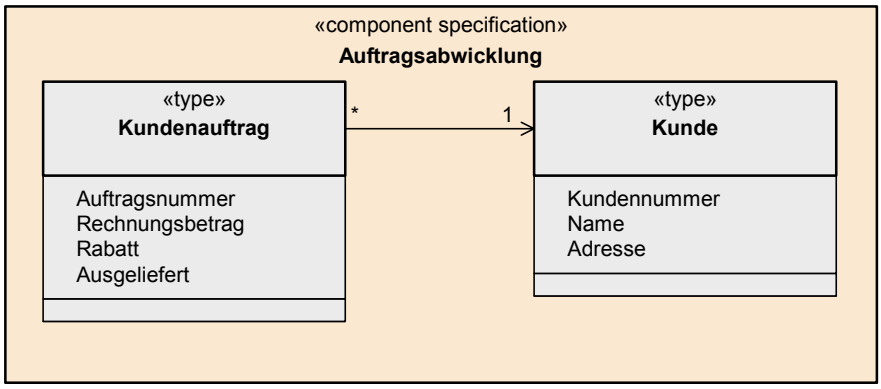

Bild 3: Komponente mit spezifikationsrelevanten internen Typen

4. Abstimmungsebene

- Die UML schlägt vor, Zustandsmaschinen als Standard zu verwenden, um Sachverhalte auf der Abstimmungsebene zu beschreiben. Demgegenüber stehen die vom Memorandum vorgeschlagenen temporalen Operatoren.
- Es wird vorgeschlagen, beide Varianten als Alternativen in das Memorandum aufzunehmen, da beide ihre spezifischen Vorteile haben. Dadurch wird die Konformität zur UML gewahrt, ohne die Ausdrucksmächtigkeit zu verringern.

Durch die unterbreiteten Vorschläge ergeben sich die folgenden Vorteile:

- Schnittstellen-, Verhaltens- und Abstimmungsebene verwenden mit der UML eine einheitliche Notationstechnik.
- Die Spezifikation auf diesen Ebenen profitiert von den erweiterten Ausdrucksmöglichkeiten von UML 2.0.
- Die Spezifikation bleibt konform zum Modellierungsstandard UML.
- Auf einige der bisher notwendigen Workarounds (gedachte Komponente Extern, Namenskonventionen zur Verbindung von Schnittstellen und Verhaltensebene) kann verzichtet werden.

Mit diesen Vorschlägen wird also dafür plädiert, die UML konsequent auf den Ebenen einzusetzen, wo sie die Spezifikation verbessern kann und wo sie international als Modellierungsstandard anerkannt ist. Das sind derzeit die Schnittstellen-, Verhaltens- und

(mit Einschränkungen) die Abstimmungsebene. Das Memorandum leistet über diese Ebenen hinaus einen wichtigen Beitrag, in dem es alle spezifikationsrelevanten Objekte identifiziert und den Spezifikationsebenen zuordnet. Das Memorandum betrachtet dabei auch Objekte (z.B. Terminologie und betriebliche Aufgaben), die von der UML nicht abgedeckt werden.

4 Verbesserungen bei der OCL 2.0

In den Fallstudien [Acke2001] und [Acke2003] sind in einigen Details Probleme bei der Verwendung der Object Constraint Language (OCL) aufgetreten. Die Version UML 2.0 enthält einige Erweiterungen, die für die Spezifikation von Fachkomponenten relevant sind und zwei der identifizierten Probleme beheben.

Es soll spezifiziert werden, dass während der Abarbeitung eines Dienstes ein anderer Dienst aufgerufen wird, bei dem es sich um eine Nicht-Query-Methode handelt. Dies war bisher mit OCL nicht möglich. Im Memorandum wurden solche Bedingungen auf der Abstimmungsebene formuliert und für die Abstimmungsebene wurde (genau wegen dieses Problems) postuliert, dass (im Gegensatz zum OCL-Standard) auch Nicht-Query-Methoden in OCL-Ausdrücken verwendet werden können. Vergleiche dazu [Acke2001, S. 55]. Die OCL 2.0 bietet nun einen Operator *hasSent* ('^'), mit welchem genau dieser Sachverhalt ausgedrückt werden kann.

Außerdem ist es mit OCL 2.0 möglich, auf die Exportparameter eines Dienstes auch im Kontext eines anderen Dienstes zuzugreifen. Dazu wurde der übliche Zugriff auf Eigenschaften (durch Verwendung von '.') auch auf die Exportparameter einer Methode erweitert. Da ein solcher Zugriff in den Fallstudien notwendig war, wurde diese Möglichkeit dort einfach vorweggenommen (allerdings durch Verwendung von '::'). Vergleiche dazu [Acke2001, S. 39].

Die neuen Möglichkeiten mit der OCL sollen nun an einem Beispiel demonstriert werden. Betrachten wir eine Beispielkomponente *Auftragsverwaltung*. Diese hat zwei Interfaces: das bereitgestellte Interface *IKundenauftrag* (u.a. mit der Methode *Annehmen*) und das benötigte Interface *IBestand* (u.a. mit der Methode *Buche*). Vergleiche dazu auch Bild 2. Der Dienst *Annehmen* soll folgende Aufgaben ausführen:

- Überprüfen, ob die Auftragsdaten konsistent sind und ob der Auftrag ausgeführt werden kann,
- Ermitteln, ob die erforderlichen Materialien verfügbar sind (bei einer anderen Komponente über die Methode *IBestand.BerechneBestandZum*),
- Buchen der erforderlichen Materialien (bei einer anderen Komponente über die Methode *IBestand.Buche*),
- Zurückmelden, ob der Auftrag angenommen werden kann.

Der Kundenauftrag kann nicht angenommen werden, wenn die erforderlichen Materialien nicht verfügbar sind. Dies wird in der Nachbedingung in Bild 4 ausgedrückt. Der verfügbare Bestand wird über die Methode *IBestand.BerechneBestandZum()* ermittelt und von dieser Methode im Exportparameter *Bestand* zurückgegeben. In der dritten Zeile von Bild 4 sieht man, wie mit Hilfe des OCL-Ausdrucks `BerechneBestandZum().Bestand` auf den Exportparameter *Bestand* zugegriffen werden kann, um die Werte danach mit dem benötigten Bestand vergleichen zu können.

```
Auftragsabwicklung::IKundenauftrag::Annehmen(in at: Auftrag): boolean
 post: let benötigterBestand = ... in
       let verfügbarerBestand = IBestand.BerechneBestandZum(...).Bestand in
       if benötigtesMaterial > verfügbaresMaterial
         then result = false endif
```

Bild 4: OCL-Bedingung mit Zugriff auf einen Exportparameter

Ist das benötigte Material vorhanden, wird von der Methode *Annehmen* die Methode *Buche* von *IBestand* gerufen. Dieser Sachverhalt wird in der Nachbedingung im Bild 5 mit Hilfe des OCL-Ausdrucks `IBestand^Buche(...)` ausgedrückt.

```
Auftragsabwicklung::IKundenauftrag::Annehmen(in at: Auftrag): boolean
 post: if benötigtesMaterial <= verfügbaresMaterial
         then IBestand^Buche(...) endif
```

Bild 5: OCL-Bedingung mit dem Operator hasSent (‘^’)

Literatur

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

[Acke2003] *Ackermann, J.*: Zur Spezifikation der Parameter von Fachkomponenten. In: *K. Turowski (Hrsg.)*: 5. Workshop Komponentenorientierte betriebliche Anwendungssysteme (WKBA 5). Augsburg 2003, S. 47 – 154.

[Andr2003] *Andresen, A.:* Komponentenbasierte Softwareentwicklung mit MDA, UML und XML. Hanser Verlag, München 2003.

[ChDa2001] *Cheesman, J.; Daniels, J.:* UML Components. Addison-Wesley, Boston 2001.

[Fowl2000] *Fowler, M.:* UML Distilled – A Brief Guide to the Standard Object Modeling Language. Addison-Wesley, 2000.

[OMG2001] *OMG (Hrsg.):* OMG Unified Modeling Language Specification, Version 1.4, September 2001. Needham 2001.

[OMG2003] *OMG (Hrsg.):* Unified Modeling Language: Superstructure, Version 2.0, Stand 10.04.2003. URL: http://www.omg.org/uml, Abruf am 2003-06-10.

[Over2002] *Overhage, S.:* Die Spezifikation – kritischer Erfolgsfaktor der Komponentenorientierung. In: *K. Turowski (Hrsg.)*: 4. Workshop Komponentenorientierte betriebliche Anwendungssysteme (WKBA 4). Augsburg 2002, S. 1 – 17.

[Turo2002] *Turowski, K. (Hrsg.).:* Vereinheitlichte Spezifikation von Fachkomponenten. Memorandum des Arbeitskreises 5.10.3 der Gesellschaft für Informatik. Augsburg 2002.