September 2001

Common Business Component Model (COBCOM): Generelles Modell komponentenbasierter Anwendungssysteme

Claus Rautenstrauch
Otto-von-Guericke-Universität Magdeburg, rauten@iti.cs.uni-magdeburg.de

Klaus Turowski
Universität der Bundeswehr München, turowski@informatik.unibw-muenchen.de

In: Buhl, Hans Ulrich, u.a. (Hg.) 2001. *Information Age Economy*; 5. Internationale Tagung Wirtschaftsinformatik 2001. Heidelberg: Physica-Verlag

ISBN: 3-7908-1427-X

Common Business Component Model (COBCOM): Generelles Modell komponentenbasierter Anwendungssysteme

Claus Rautenstrauch
Otto-von-Guericke-Universität Magdeburg

Klaus Turowski
Universität der Bundeswehr München

Zusammenfassung: Ausgehend von konventionellen Architekturen werden strukturelle und dynamische Aspekte komponentenbasierter betrieblicher Anwendungssysteme beleuchtet. Für die strukturelle Sicht wird auf dieser Grundlage ein generelles Architekturmodell für komponentenbasierte betriebliche Anwendungssysteme abgeleitet. Zusammen mit der den allgemeinen Lebenszyklus einer Fachkomponente darstellenden dynamischen Sicht wird das generelle Modell komponentenbasierter Anwendungssysteme COBCOM hergeleitet.

Schlüsselworte: Fachkomponente, Architektur, betriebliches Anwendungssystem

1 Von Monolithen zu Komponenten

Frühe betriebliche Anwendungssysteme waren durch ihre *monolithische* Umsetzung (Abbildung 1 (1)) geprägt. Unter einem Monolithen versteht man ein Anwendungssystem, das aus der Sicht der Softwaretechnik aus teilweise nicht klar voneinander abgrenzbaren Systemteilen besteht oder dessen Systemteile so eng miteinander vermascht sind, dass Systemteile aufwendig und in der Regel unwirtschaftlich herausgelöst oder ersetzt werden können. Ferner zeichnen sich monolithische Anwendungssysteme dadurch aus, dass sie sowohl Systemteile umfassen, die anwendungsbezogen sind, als auch solche, die anwendungsübergreifend verwendbar sind, z. B. Systemteile zur Datenverwaltung.

Mangelnde Integrationsfähigkeit und zunehmende Dezentralisierungsbestrebungen führten zur Etablierung von Client/Server-Architekturen, die zunächst aus drei Systemteilen bestehen (Abbildung 1 (2)) [StHa1997, S. 161]:

– *Verarbeitung*, in dem die eigentliche Verarbeitungslogik kodiert ist, z. B. für die Berechnung eines Nettobedarfs oder zur Erstellung eines Produktionsplans,

– *Präsentation*, der Systemteile zur Abwicklung der Benutzerinteraktion umfasst, z. B. zur Darstellung einer Eingabemaske auf dem Bildschirm, und

– *Datenhaltung*, der die Verwaltung der Daten z. B. zur Speicherung von Aufträgen, Rechnungen, Teiledaten usw. beinhaltet

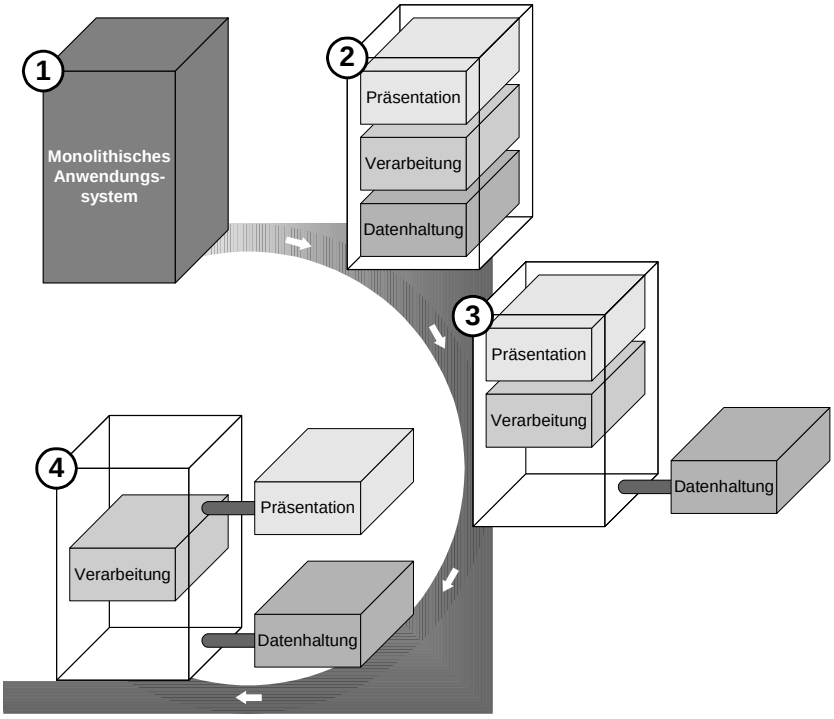

Abbildung 1: Vom Monolithen zur Client/Server-Architektur

Hiermit wird Funktionalität aus dem ursprünglich monolithischen Anwendungssystem herausgelöst (Abbildung 1 (3)) und die Verarbeitung auf verschiedene, miteinander vernetzte Betriebsmittel (Rechner) verteilbar. Die in Abbildung 1 (4) dargestellte Architektur wird auch als dreistufige Client/ Server-Architektur bezeichnet [Sera1999, S. 7-9] und findet sich in heute verfügbaren betrieblichen Standardanwendungssystemen wieder, vgl. z. B. [Buck1999, S. 115-121].

Ausgehend davon lassen sich weitere Systemteile herauslösen. So kann z. B. die Systemkomponente *Verarbeitung* weiter in eine Systemkomponenten *Funktion* und eine Systemkomponente (betriebliche) *Ablauflogik* untergliedert werden (Abbildung 2 (2)).

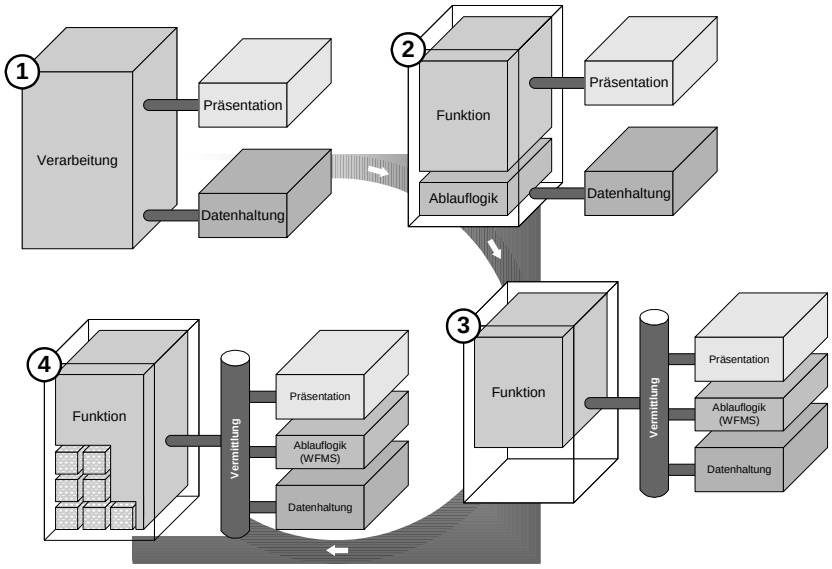

Abbildung 2: Weitergehende Spezialisierung bis hin zur Komponentenbauweise

Die Systemkomponenten *Funktion* stellt in diesem Fall Dienste bereit, um *einzelne* Tätigkeiten zu unterstützen oder zu automatisieren, die sich aus der jeweiligen betrieblichen Aufgabe ergeben, z. B. Dienste zur Bestandsbuchung, Stücklistenauflösung oder Feinterminierung. Die Systemkomponente *Ablauflogik* setzt darauf auf, indem sie die Verknüpfung einzelner Tätigkeiten zu wiederkehrenden (betrieblichen) Arbeitsabläufen, so genannten *Workflows* [Holl1995, S. 6], erlaubt und deren Ausführung überwacht und koordiniert. Programme, die eine solche automatisierte Vorgangsbearbeitung unterstützen werden als Workflowmanagementsysteme (WFMS) bezeichnet [Holl1995, S. 6]. WFMS tragen dazu bei, betriebliche Anwendungssysteme an die Ablauforganisation eines Unternehmens anzupassen und haben insbesondere im Zuge der *Prozessorientierung* Bedeutung erlangt, in deren Rahmen Unternehmen versuchen ihre Ablauforganisation zu verbessern.

Mit dem Herauslösen der Systemkomponente *Ablauflogik* ist jedoch ein zusätzlicher Kommunikationsbedarf entstanden. Die Lösung des damit verbundenen Problems ist insbesondere dann nicht trivial, wenn die jeweiligen Systemkomponenten unter Nutzung voneinander verschiedener Betriebssysteme, Hardware oder Entwicklungswerkzeuge realisiert werden, d. h., wenn ein *heterogenes* Systemumfeld gegeben ist. Um die damit einhergehenden Probleme zu überwinden, bietet es sich an, eine weitere Systemkomponente herauszulösen. Das ist die in Abbildung 2 (3) dargestellte Systemkomponente *Vermittlung*.

Der Systemkomponente *Vermittlung* obliegt es eine Verbindung zwischen den jeweiligen Dienstnehmern und Dienstgebern herzustellen und die Kommunikation

zwischen diesen (in einem eventuell heterogenen Systemumfeld) abzuwickeln. Typische Vertreter dieser Systemkomponente sind z. B. *Object Request Broker* (ORB), die der *Common Object Request Broker Architecture* (CORBA) entsprechen, die von der *Object Management Group* (OMG) als Standard für derartige Systemkomponenten vorgeschlagen wurde. Zur Vermittlung verwaltet der ORB dazu ein Liste von Dienstgebern nebst der angebotenen Dienste. Ferner stellt er auf der Basis von Konvertierungstechniken die Kommunikation in heterogenen Umgebungen sicher.

Die bisher unterschiedenen Systemkomponenten zeichnen sich dadurch aus, dass sie unabhängig von der konkreten betrieblichen Anwendung, z. B. Produktionsplanung und -steuerung, Warenwirtschaft, Finanzbuchhaltung usw., identifiziert werden können. Die in Abbildung 2 (4) angedeutete, weitergehende Differenzierung der Systemkomponente *Funktion*, deren Ergebnis letztlich *Fachkomponenten*, im Sinne der in [FeRT1999, S. 26] gegebenen Definition, wären, steht noch aus.

Gleichwohl trägt die Darstellung in Abbildung 2 (4) dazu bei, eine generelle Architektur komponentenbasierter betrieblicher Anwendungssysteme herzuleiten. So wird bestätigt, dass ein betriebliches Anwendungssystem aus verschiedenen Systembausteinen bestehen kann, die sich hinsichtlich ihrer Anwendungsnähe unterscheiden. Damit lassen sich grundsätzlich drei Gruppen von Software unterscheiden:

- Das Anwendungssystem, das aus der Systemkomponente *Funktion* sowie weiteren, als Fachkomponenten herausgelösten Systemkomponenten besteht,
- das Betriebssystem incl. aller systemnahen Dienste (Treiber, Utilities etc.) und
- Systemkomponenten, die unabhängig von der eigentlichen betrieblichen Aufgabe verwendbar sind.

2 BCArch

Auf der Basis der genannten Gruppen lassen sich unmittelbar Ebenen ableiten, denen die einzelnen Systembausteine zum Zwecke der Klassifikation zuordenbar sind und die sich auch in der *generellen Architektur komponentenbasierter Anwendungssysteme* (kurz: *BCArch* (*Business Component Architecture*)) (Abbildung 3) wiederfinden. Dieses sind die Anwendungs-, Middleware-, Betriebssystem- und Hardwareebenen. Dabei wird hier bezogen auf den Begriff der Middleware eine gegenüber der in der Literatur verbreiteten Definition eine weitere Sicht eingenommen. So umfasst die Middleware alle anwendungsunabhängigen Programme, die nicht zum Betriebssystem gehören. In der Literatur wird dem gegenüber häufig eine engere Sichtweise auf Middleware vertreten, welche die Aspekte Unterstützung der (Interprozess-) Kommunikation sowie Anwendungsintegration

in den Vordergrund stellt, z. B. [Soef1997], [Sera1999, S. 4-6].

Abbildung 3 zeigt BCArch für einen einfachen Fall. Damit Fachkomponenten verwendet werden können, bedarf es demnach zusätzlicher Systemteile: eines *Komponenten-Anwendungs-Frameworks* und eines *Komponenten-System-Frameworks*. Eine ähnliche Unterscheidung findet sich beispielsweise auch bei [FaSJ1999, S. 9f.]. Komponenten-Anwendungs-Frameworks werden dort als *Enterprise Application Framework* bezeichnet. Für Komponenten-System-Frameworks wird jedoch eine feinere Unterscheidung in *Middleware Integration Frameworks* und *System Infrastructure Frameworks* vorgenommen.

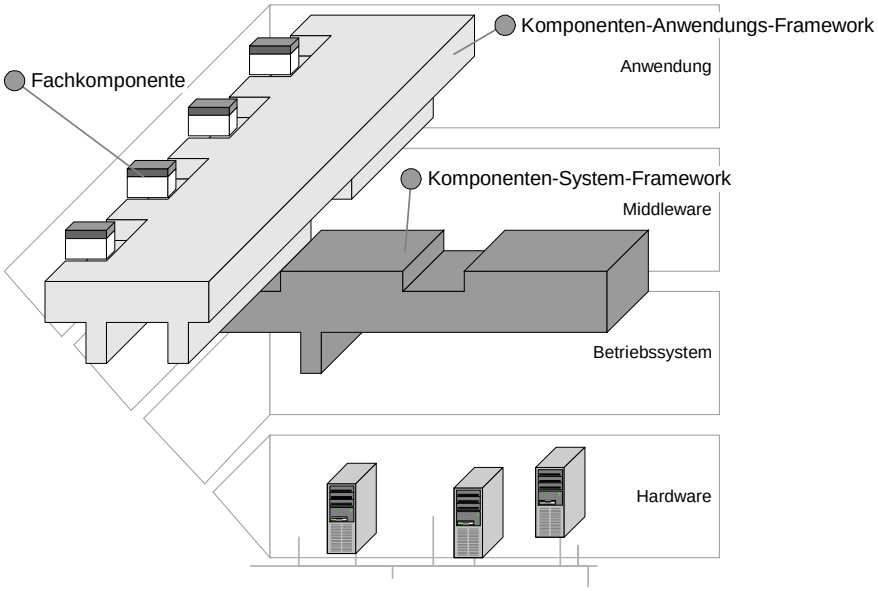

Abbildung 3: BCArch: Generelle Architektur komponentenbasierter Anwendungssysteme

Allgemein gilt, dass (Fach-)Komponenten eines ergänzenden Frameworks (oder einer Rahmenarchitektur) bedürfen, der deren Zusammenarbeit erst ermöglicht [NiLu1997, S. 19]. Unter einem *Framework* wird in der Regel ein (Code-)Rahmen verstanden, der für eine bestimmte Gruppe von Programmen, z. B. grafische Benutzeroberflächen [KrPo1988] oder Softwareentwicklungswerkzeuge [Satt1997], ein Skelett vorgibt, das um eigene Programme ergänzt werden kann [FaSJ1999, S. 3]. Frameworks treten häufig in Form von Klassenhierarchien oder Klassenbibliotheken auf, die Klassen und abstrakte Klassen enthalten, die in der Regel unter Verwendung einer einzigen Programmiersprache kodiert wurden und die Gegenstand einer White-Box-Wiederverwendung sind, vgl. [Wirf1990], [Lewi1995, S. 34], [NiLu1997, S. 19]. Jüngere Ansätze dehnen die Verwendung von Frameworks auch auf den Bereich der Black-Box-Wiederverwendung aus, indem nur bestimmte, speziell dafür vorgesehene Teile des Frameworks ausgetauscht werden

können. Dazu werden Teile des Frameworks als so genannte *Hot Spots* definiert, um anzudeuten, dass diese durch fremde Komponenten ersetzbar sind [Pree1997, S. 46, S. 63-84].

Der Begriff des Frameworks wird im Folgenden in einer allgemeinen Bedeutung im Sinne einer Systemkomponente des Anwendungssystems verwendet, für die keine Annahmen bezüglich die Verfolgung einer bestimmten Softwareentwicklungstechnik getroffen werden sollen. Besonders wünschenswert sind jedoch solche Frameworks, die, ebenso wie Fachkomponenten, Gegenstand einer Black-Box-Wiederverwendung sind. Bei diesen soll es möglich sein, bestimmte, dafür vorgesehene Teile des Frameworks als Ganzes auszutauschen oder über speziell dafür vorgesehenen Stellen zusätzliche (Fach-)Komponenten zu integrieren.

Ein Komponenten-Anwendungs-Framework wird als Systemteil definiert, der Fachkomponenten anwendungsdomänenbezogene (Standard-)Dienste bereitstellt und für diese eine Integrationsplattform darstellt. Dazu gehören beispielsweise Mechanismen zur Bewältigung fachlicher Konflikte oder von verschiedenen Fachkomponenten gemeinsam genutzte (Fabriken zur Erzeugung von) *Business Objects*, wie sie etwa als *Common Business Objects* in dem IBM San Francisco Framework zur Verfügung stehen [Wesk1999, S. 9f.]. Das Komponenten-Anwendungs-Framework kann dabei verschieden stark ausgeprägt sein. So wird in [FeRT1999] ein Ansatz beschrieben, bei welchem das Komponenten-Anwendungs-Framework vornehmlich Dienste zur fachlichen Konfliktbehandlung bereitstellt und dabei zugleich wesentliche Mechanismen verfügbar macht, um das Gesamtsystem zusammen zu halten. Der Komponenten-Anwendungs-Framework kann jedoch auch als Rahmen dienen, in den die jeweiligen Fachkomponenten eingefügt werden. Dabei ist es unerheblich, ob der Komponenten-Anwendungs-Framework selbst als Fachkomponente vorliegt oder als Klassenhierarchie (wie z. B. im IBM San Francisco Framework), die Gegenstand einer White-Box-Wiederverwendung ist. Ferner können für ein Anwendungssystem mehrere Komponenten-Anwendungs-Frameworks zugleich verwendet werden, z. B. bei einer zusätzlichen Einbindung von Altanwendungen. Hierbei kann das einzubindende Anwendungssystem selbst in die Rolle eines Komponenten-Anwendungs-Frameworks rücken, indem es entsprechende Dienste über Schnittstellen verfügbar macht. Ein Beispiel dafür ist das *Business Application Programming Interface (BAPI)* der SAP [SAP1997], das Dienste von SAP R/3 für externe Anwendungen zur Verfügung stellt. Obwohl diese Schnittstelle ursprünglich dazu gedacht war, zusätzliche Dienste in R/3 einzubinden, kann sie umgekehrt genutzt werden, um R/3 als Komponenten-Anwendungs-Framework wiederzuverwenden. Ergänzend zu diesen anwendungsnahen Diensten sind weitere middlewarenahe Dienste für den Aufbau komponentenbasierter Anwendungssysteme notwendig. Diese finden sich in einem oder mehreren Komponenten-System-Frameworks wieder.

Ein Komponenten-System-Framework wird als Systemteil definiert, der Fachkomponenten anwendungsinvariante, middlewarenahe Dienste zur Verfügung stellt.

Beispiele für solche Dienste finden sich für alle Plattformen, auf denen komponentenbasierte betriebliche Anwendungssysteme aufsetzen können, z. B. der *Object Management Architecture* (OMA) der OMG, in Suns *JavaBeans* oder Microsofts *Distributed Component Object Model* (DCOM). Zu nennen sind hierzu besonders Vermittlungs-Dienste, wie sie etwa von am Markt verfügbaren Object Request Brokern erbracht werden, die sich nach dem von der OMG standardisierten CORBA richten. Neben dem Bereitstellen einer grundlegenden technischen Infrastruktur, z. B. zur Unterstützung von entfernten Methodenaufrufen, Selbstbeobachtung, Persistenz oder Ereignissen, fallen auch spezielle Dienste wie die Unterstützung von Transaktionen, z. B. durch spezielle Dienste (*Object Transaction Service* [OMG2000]), in den Aufgabenbereich eines Komponenten-System-Frameworks. Darüber hinaus sind auch WFMS, die eine automatische Vorgangsbearbeitung in komponentenbasierten Systemen unterstützen, sowie Datenbankmanagementsysteme als spezielle Ausprägungen Komponenten-System-Frameworks zuzuordnen.

Abbildung 2 hat erste Anhaltspunkte zur Architektur komponentenbasierter Anwendungssysteme geliefert. Diese konnten im Folgenden zur Generalisierung der Teilsysteme eines komponentenbasierten Anwendungssystems genutzt werden. So findet sich in Abbildung 3 das Komponenten-Anwendungs-Framework als Systemkomponente *Funktion* wieder. Die Fachkomponenten würden, wie im obigen Beispiel zu SAP R/3 ausgeführt, an diesen gekoppelt werden und über diesen auf Dienste zur Benutzerinteraktion und Datenhaltung zugreifen. Es sei jedoch darauf hingewiesen, dass damit nur *ein* mögliches Szenario beschrieben wird. Es wäre ebenso gut denkbar, dass eine Fachkomponente nicht auf eine Systemkomponente *Datenhaltung* zugreift, die als Komponenten-System-Framework in das Anwendungssystem integriert ist, sondern *selbst* über einen solchen Systemteil verfügt.

Alternative Ansätze zur Beschreibung der Architektur von komponentenbasierten (Anwendungs-)Systemen finden sich z. B. in [Szyp1998, S. 273-279]. Hierbei wird besonders auf die Abhängigkeiten zwischen Komponenten-Frameworks und auf die Notwendigkeit diese zu hierarchisieren eingegangen, jedoch wird keine Unterscheidung zwischen Komponenten-System- und -Anwendungs-Frameworks vorgenommen. Hier soll insofern den dort getroffenen grundlegenden Aussagen gefolgt werden, als eine Hierarchisierung ausdrücklich unterstützt wird.

Nach diesen Ausführungen ist die generelle Architektur eines komponentenbasierten betrieblichen Anwendungssystems motiviert und die prinzipiellen Zusammenhänge der identifizierten Systemteile beschrieben. Ferner steht damit ein Ordnungsrahmen (siehe Abbildung 3) zur architekturbezogenen Klassifikation der Systemteile eines komponentenbasierten Anwendungssystems zur Verfügung.

3 BCLifeCycle: Lebenszyklus einer Fachkomponente

Neben der strukturellen Sicht, die komponentenbasierte Anwendungssysteme als Ganzes ins Auge fasst, unterliegen Fachkomponenten und die darauf aufbauenden Anwendungssysteme einem Lebenszyklus. Einen Überblick der einzelnen Phasen des (Produkt-)Lebenszyklus (einer Fachkomponente) gibt der in Abbildung 4 dargestellte Lebenszyklus einer Fachkomponente (kurz: *BCLifeCycle* (Business Component Life Cycle)). Der Lebenszyklus ist dabei rein qualitativ zu sehen, und zwar in dem Sinne, dass die genannten Phasen in der Regel erforderlich sind. Eine strenge Einhaltung der vorgeschlagenen Reihenfolge der einzelnen Phasen, wie etwa bei einem Vorgehensmodell, soll und kann hier jedoch nicht gefordert werden, da die einzelnen Phasen zum Teil in anderer Reihenfolge durchlaufen werden können. Im Folgenden werden die einzelnen Phasen jeweils kurz erläutert.

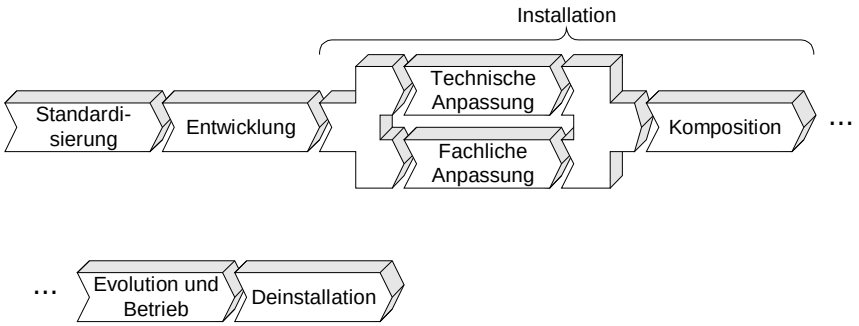

Abbildung 4: (Produkt-)Lebenszyklus einer Fachkomponente

Alternative Ansätze zu *BCLifeCycle* finden sich z. B. in [BrWa1996] und [Same1997, S. 159-193]. Während die erstgenannte Quelle vornehmlich technische Fragestellungen in den Vordergrund stellt, wird in der Zweiten detailliert auf die Phasen *Standardisierung* und *Entwicklung* eingegangen.

3.1 Standardisierung

Damit Fachkomponenten beliebiger Hersteller zum Zwecke der kooperativen Aufgabenbearbeitung zu kundenindividuellen Anwendungssystemen zusammengesetzt werden können, bedarf es der Schaffung von Standards. Unter einem *Standard* wird eine von Verwendern und Herstellern akzeptierte und verbindliche Vereinheitlichung bestimmter Tatbestände verstanden, die Fachkomponenten betreffen. Die Phase der Standardisierung dient damit der Vereinheitlichung von Fachkomponenten. Für eine Diskussion weiterer im IT-Bereich verwendeter Standardisierungsbegriffe vgl. [Jako2000, S. 9-12].

Die Standardisierung von Fachkomponenten betrifft die fachliche Ebene, z. B. die Definition einheitlicher Begriffssysteme für betriebliche Aufgaben und Daten, die Spezifikation der durch die jeweilige Fachkomponente abgedeckte Funktionalität oder die Definition klarer Schnittstellen zwischen Fachkomponenten. Weiterhin bedarf es technischer Standards, die z. B. die Parameterübergabe bei Dienstaufrufen regeln oder grundlegende Kommunikationsprotokolle festlegen.

Ohne Standardisierung kann das Ziel einer *beliebigen* Austauschbarkeit von Fachkomponenten (gleicher Funktionalität) gegeneinander nicht erreicht werden, da sonst für jede Fachkomponente angegeben werden müsste, welche konkrete andere Fachkomponente sie ersetzen kann. Gleichwohl kann nicht davon ausgegangen werden, dass eine *vollständige* (fachliche) Standardisierung der betrieblichen Anwendungsdomäne jemals gegeben sein wird. Dies kann z. B. mit einer für die Standardisierung zu hohen Änderungsgeschwindigkeit bzw. Variantenvielfalt der betrieblichen Anwendungsdomäne begründet werden. Darüber hinaus ist die Vollständigkeit eines Standards für die gesamte betriebliche Anwendungsdomäne auf Grund des damit verbundenen Erhebungsproblems nicht nachweisbar.

Dieses Dilemma kann nur aufgelöst werden, wenn die Anforderungen gegenüber dem Idealbild einer vollständigen Standardisierung der betrieblichen Anwendungsdomäne etwas zurücknimmt. So kann als Ergebnis verschiedener aktueller Standardisierungsbestrebungen von der Etablierung von Teil- oder *Kernstandards* ausgegangen werden, die nur (wichtige) Teilbereiche der betrieblichen Anwendungsdomäne betreffen (vgl. dazu und für einen Überblick zu aktuellen Standardisierungsbemühungen [Turo2000]). Bemerkenswert ist, dass die Phase der Standardisierung der Entwicklung *vorhergeht*, was nicht dem allgemeinen Procedere bei der konventionellen Systementwicklung entspricht, bei der vorhandene Systeme oder Systemteile oftmals nachträglich zu Standards erhoben werden.

3.2 Entwicklung

Die Phase der Entwicklung umfasst alle Aufgaben der klassischen Softwareentwicklung (vgl. z. B. [Balz1998, S. 97-137]), die auf die Entwicklung einer Fachkomponente angewendet werden. Die Entwicklung einer Fachkomponente beinhaltet mit Ausnahme der Standardisierung alle Aufgaben, die zur Realisierung der Fachkomponente durchzuführen sind. Unterschiede zu diesen Softwareentwicklungsmethodiken ergeben sich dabei besonders im Rahmen des Requirements Engineering (RE) und beim Systemtest. So vereinfacht sich beispielsweise das RE, da auf Vorarbeiten zur Analyse der Anwendungsdomäne aufgebaut werden kann, die im Rahmen der Standardisierung erfolgt sind. Die Entscheidung darüber, welche Fachkomponenten überhaupt zu entwickeln sind (d. h. die Festlegung des Produktionsprogramms des Softwareherstellers), muss in einer dem RE vorgelagerten Phase unter Einbeziehung betriebswirtschaftlicher Erwägungen getroffen werden. In diesem Rahmen können beispielsweise wirtschaftliches Ent-

wicklungsrisiko, erwarteter Absatz, realisierbare Verkaufspreise und Return on Investment berücksichtigt werden.

Ein weiterer Unterschied zur Entwicklung herkömmlicher Software ergibt sich im Rahmen des Systemtests. So kann eine neu entwickelte Fachkomponente zwar gegen verschiedene Komponenten-Anwendungs- und -System-Frameworks und im Zusammenspiel mit anderen Fachkomponenten getestet werden, ein vollständiger Integrationstest kann jedoch nicht erfolgen, da eine Fachkomponente in einer Vielzahl von Kombinationen mit anderen Fachkomponenten zu einem konkreten Anwendungssystem konfiguriert werden kann.

3.3 Technische Anpassung

Die technische Anpassung einer Fachkomponente dient der Überwindung implementierungsbedingter Inkompatibilitäten zur technischen Integration in ein komponentenbasiertes Anwendungssystem. Sie findet nach dem Erwerb einer Fachkomponente statt. Inkompatibilitäten können durch Verwendung unterschiedlicher Entwicklungswerkzeuge oder verschiedener Programmiersprachen entstehen und führen z. B. zu Problemen bei der Behandlung gemeinsamer Betriebsmittel wie Speicher oder Drucker, bei der Benutzerinteraktion, bei der Verwendung von Umgebungsvariablen und temporären Dateien, bei der Parameterübergabe oder durch die Erzeugung von Namenskonflikten. Verstärkt werden die genannten Probleme in *verteilten Systemen*, da dann zusätzliche Probleme durch die Heterogenität von Betriebssystemen und Hardwareplattformen hinzukommen können.

Insbesondere *Wrapper* dienen als Technik für die Überwindung technischer Konflikte zu überwinden, ist der Einsatz von anzuführen. Mittels eines Wrappers wird eine Komponente mit einer Ummantelung versehen, welche die konfliktären und unerwünschten Eigenschaften der Komponente verbirgt und eine einheitliche Schnittstelle implementieren kann. Beispiele für den Einsatz von Wrappern sind die von CORBA bekannten *Stubs* und *Skeletons* [OMG1999, S. 2.1-2.11]. Allgemein können Wrapper z. B. auf den Entwurfsmustern *Adapter*, *Bridge*, *Decorator* oder *Proxy* basieren [GHJ+1997, S. 151-188, 227-238].

3.4 Fachliche Anpassung

Die fachliche Anpassung (auch Parametrisierung oder Customizing) kann sowohl vor als auch nach der Komposition stattfinden. Sie beinhaltet die Anpassung der Eigenschaften und Dienste der jeweiligen Fachkomponente hinsichtlich deren fachlicher Funktionalität. Dazu werden beispielsweise Nummernkreise eingerichtet, Stammdaten initialisiert oder die zu verwendenden Verfahren ausgewählt (z. B. bei einer Fachkomponente zur Lagerverwaltung, welches Verfahren für welches Bestandskonto zur Bestellmengenplanung anzuwenden ist). Auch können hier bestimmte Dienste deaktiviert werden, um fachliche Konflikte zu vermeiden.

Ist dies jedoch nicht möglich, da z. B. eine Fachkomponente Materialwirtschaft, die bereits Dienste zur Lagerverwaltung umfasst, diese selbst verwendet und deren Deaktivierung darum nicht vorgesehen ist, müssen im Rahmen der *Komposition* Maßnahmen zur Konfliktbehebung ergriffen werden.

Ferner können bei der fachlichen Anpassung fachkomponenteninterne Abläufe festgelegt werden, z. B. für eine Fachkomponente zur Personalverwaltung, die Art und Reihenfolge der mit einer Einstellungsmaßnahme verbundenen Vorgänge oder in einer Fachkomponente zur Finanzbuchhaltung die Festlegung von automatisch zu erfolgenden Buchungen. Dem gegenüber können fachkomponentenübergreifende Abläufe, für deren Realisierung insbesondere Workflowmanagementsysteme anwendbar sind, erst im Anschluss an die Komposition festgelegt werden. Wird beispielsweise ein Geschäftsprozess *Auftragsabwicklung* in einem komponentenbasierten Anwendungssystem mit Hilfe eines Workflowmanagementsystems realisiert, dann können im Rahmen der Machbarkeitsprüfung Dienste verschiedener Fachkomponenten benötigt werden, z. B. dann, wenn die Machbarkeitsprüfung in eine kaufmännische und eine technische Machbarkeitsprüfung zerfällt, die jeweils über Dienste verschiedener Fachkomponenten erbracht werden.

3.5 Komposition

Die Komposition umfasst die technische und fachliche *Integration* der jeweiligen Fachkomponente in ein Anwendungssystem. Zusammen mit der technischen und der fachlichen Anpassung wird damit die Installation einer Fachkomponente vollendet.

Die technische Integration wird durch die technische Anpassung vorbereitet und beinhaltet z. B. die Anmeldung der neuen Dienste bei einem Object Request Broker oder die Änderungen der Adresse eines Dienstgebers in einem Skript, das im Sinne des *Gluing* [NiLu1997, S. 20f.] zur Komposition eingesetzt wird. Als grundlegende Techniken zur Realisierung einer technischen Integration sind z. B. Bussysteme, Ereigniskanäle, Tupelräume oder Skriptsprachen zu nennen [Grif1998, S. 182-336]. Dabei ist zwischen rein struktureller Komposition und Maßnahmen, die den fachlichen Ablauf (das softwaretechnische Abbild des Geschäftsprozesses) ändern, zu unterscheiden. So können z. B. Skriptsprachen, neben Workflowmanagementsystemen, zur Festlegung fachkomponentenübergreifender Abläufe dienen. Damit setzen die Techniken zur fachlichen Integration auf den Techniken zur technischen Integration auf.

Die fachliche Integration wurde durch die fachliche Anpassung vorbereitet und wird im Rahmen der Komposition komplettiert. Eine besondere Bedeutung kommt hierbei der Behandlung fachlicher Konflikte zu, da auch solche Fachkomponenten angepasst werden müssen, die schon im Anwendungssystem vorhanden sind (z. B. kann auch die Einbindung von nicht komponentenbasierten Altanwendungen erforderlich sein), oder die Einführung zusätzlicher Systembausteine zur Konfliktbe-

handlung erforderlich ist, z. B. die Einführung von Linkobjekten [FeRT1999]. Im allgemeinen handelt es sich dabei um spezielle Systembausteine, die zwischen Dienstgeber und -nehmer in geeigneter Weise vermitteln und damit die Grundfunktionalität eines Mediators [GHJ+1997, S. 345-355] implementieren.

Falls Dienste von alten Systembausteinen durch Dienste neuer Systembausteine ersetzt werden sollen, ist bei der Komposition zu gewährleisten, dass auch weiterhin auf bestehende Datenbestände zugegriffen werden kann. Soll z. B. eine Fachkomponente zur Kundenverwaltung durch eine neue Fachkomponente ersetzt werden, muss sichergestellt sein, dass die bereits vorhandenen Kundendaten auch weiterhin verwendet werden können. Um dies zu erreichen kann z. B. die Migration der fraglichen Daten zu der neuen Fachkomponente erforderlich sein. Eine Alternative zur Migration besteht im Falle nur lesender Zugriffe durch die Verwendung von Mediatoren nach [Wied1992], die Daten aus verschiedenen Quellen für einen übergeordneten Verwender bereitstellen. Dieser Ansatz grenzt sich von dem oben genannten Verständnis eines Mediators jedoch dadurch ab, dass hier, entgegen der obigen multi-direktionalen Kommunikation, lediglich eine uni-direktionale (nur lesende) Kommunikation unterstützt wird.

3.6 Evolution und Deinstallation

Die Evolution beinhaltet alle Aufgaben, die mit der Anpassung einer Fachkomponente nach deren erstmaligen Installation in Verbindung stehen. Je nach Umfang der Änderungen ist dazu wieder eine technische und fachliche Anpassung sowie eine Komposition durchzuführen, sodass die oben genannten Probleme und Lösungsansätze auch hier relevant sind. Die Deinstallation umfasst alle Aufgaben, die mit der Entfernung einer Fachkomponente aus einem Anwendungssystem zusammenhängen.

4 CoBCoM

Fasst man die beiden oben eingeführten Sichten zusammen, lässt sich daraus der in Abbildung 5 dargestellte Ordnungsrahmen ableiten, der sich insbesondere dazu eignet, um (Forschungs-)Ansätze zu klassifizieren, die sich mit Themenstellungen zur komponentenbasierten Gestaltung betrieblicher Anwendungssysteme befassen.

Die jeweiligen Ansätze werden dazu in eines der hervorgehobenen Felder eingeordnet (*Komponenten-Anwendungs-*, *Komponenten-System-Framework*, *Standardisierung*, *Entwicklung*, usw.). Die Zuordnung der Ansätze kann dabei jedoch nicht auf einer eindeutigen Zuordnungsvorschrift erfolgen, z. B. auf der Grundlage einer Metrik, sondern wird entsprechend der größten Übereinstimmung mit den oben angegebenen Beschreibungen der jeweiligen Felder des Ordnungsrahmens

vorgenommen. So ist z. B. der IBM San Francisco Framework primär als Komponenten-Anwendungs-Framework einzuordnen, obwohl auch einige Dienste eines Komponenten-System-Frameworks angeboten werden, z. B. eine rudimentäre Workflowmanagementfunktionalität.

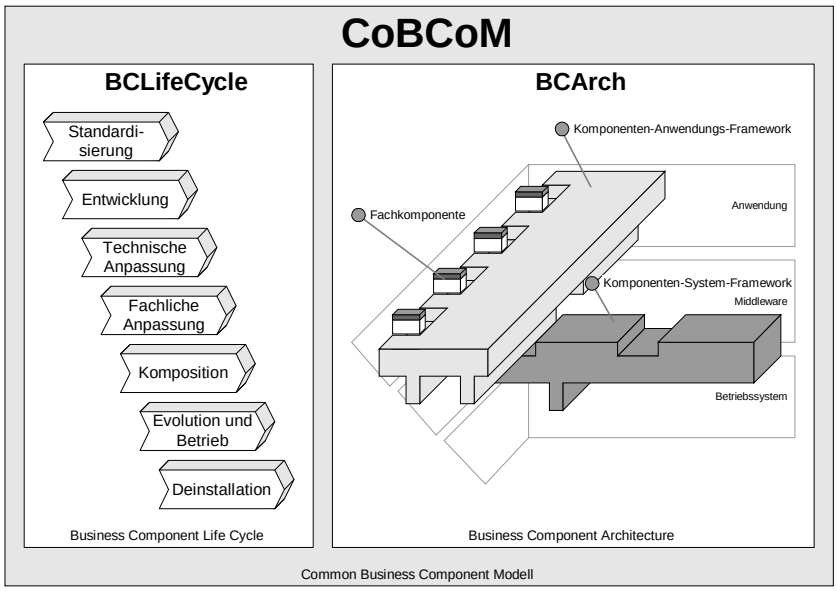

Abb. 5: Ordnungsrahmen für komponentenbasierte betriebliche Anwendungssysteme

Literatur

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

[BrWa1996] Brown, A. W.; Wallnau, K. C.: Engineering of Component-Based Systems. In: A. W. Brown (Hrsg.): Component-Based Software Engineering: Selected Papers from the Software Engineering Institute. IEEE Computer Society Press, Los Alamitos, California 1996, S. 7-15.

[Buck1999] Buck-Emden, R.: Die Technologie des SAP R/3 Systems. 4. Aufl., Addison-Wesley, Bonn 1999.

[FaSJ1999] Fayad, M. E.; Schmidt, D. C.; Johnson, R. E.: Application Frameworks. In: M. E. Fayad; D. C. Schmidt; R. E. Johnson (Hrsg.): Building Application Frameworks: Object-Oriented Foundations of Framework Design. John Wiley, New York 1999, S. 3-27.

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

[GHJ+1997] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software. Addison-Wesley, Bonn 1997.

[Grif1998] Griffel, F.: Componentware. Konzepte und Techniken eines Softwareparadigmas. dpunkt, Heidelberg 1998.

[Holl1995] Hollingsworth, D.: Workflow Management Coalition: The Workflow Reference Model. Winchester 1995.

[Jako2000] Jakobs, K.: Standardisation Processes in IT: Impact, Problems and Benefits of User Participation. Vieweg, Braunschweig 2000.

[KrPo1988] Krasner, G. E.; Pope, S. T.: A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. In: Journal of Object-Orientated Programming 1 (1988) 3, S. 26-49.

[Lewi1995] Lewis, T. G.: Object-Oriented Application Frameworks. Manning, Greenwich 1995.

[NiLu1997] Nierstrasz, O.; Lumpe, M.: Komponenten, Komponentenframeworks und Gluing. In: HMD 34 (1997) 197, S. 8-23.

[OMG1999] OMG (Hrsg.): The Common Object Request Broker: Architecture and Specification: Version 2.3.1, October 1999. OMG, Framingham 1999.

[OMG2000] OMG (Hrsg.): Transaction Service Specification: Version 1.1, May 2000. OMG, Needham 2000.

[Pree1997] Pree, W.: Komponentenbasierte Softwareentwicklung mit Frameworks. dpunkt, Heidelberg 1997.

[Same1997] Sametinger, J.: Software Engineering with Reusable Components. Springer, Berlin 1997.

[SAP1997] SAP (Hrsg.): BAPIs - Einführung und Überblick. SAP, Walldorf 1997.

[Satt1997] Sattler, K.-U.: A Framework for Component-Oriented Tool Integration. 4th International Conference on Object-Oriented Information Systems (OOIS '97). Brisbane, Australia 1997, S. 455-465.

[Sera1999] Serain, D.: Middleware. 2. Aufl., Springer, London 1999.

[Soef1997] Soeffky, M.: Middleware. In: P. Mertens (Hrsg.): Lexikon der Wirtschaftsinformatik. 3. Aufl., Springer, Berlin 1997, S. 264-267.

[StHa1997] Stahlknecht, P.; Hasenkamp, U.: Einführung in die Wirtschaftsinformatik. 8. Aufl., Springer, Berlin 1997.

[Szyp1998] Szyperski, C.: Component Software: Beyond Object-Oriented Programming. 2. Aufl., Addison-Wesley, Harlow 1998.

[Turo2000] Turowski, K.: Establishing Standards for Business Components. In: K. Jakobs (Hrsg.): Information Technology Standards and Standardisation: A Global Perspective. Idea Group Publishing, Hershey 2000, S. 131-151.

[Wesk1999] Weske, M.: Business-Objekte: Konzepte, Architekturen, Standards. In: Wirtschaftsinformatik 41 (1999) 4, S. 4-11.

[Wied1992] Wiederhold, G.: Mediators in the Architecture of Future Information Systems. In: IEEE Computer 25 (1992) 3, S. 38-49.

[Wirf1990] Wirfs-Brock, A.: Designing Reusable Designs: Experiences Designing Objet-Oriented Frameworks. Proceedings of ECOOP/OOPSLA ’90. Ottawa 1990.