Bausteine für komponentenbasierte Anwendungssysteme

Otto K. Ferstl, Elmar J. Sinz, Christoph Hammel, Michael Schlitt, Stefan Wolf

Bausteine für komponentenbasierte Anwendungssysteme

Komponentenbasierte Anwendungssysteme werden gegenwärtig als Konzept diskutiert, mit dem den wachsenden Zeit-, Kosten- und Qualitätsanforderungen in Zusammenhang mit der Entwicklung von Anwendungssystemen begegnet werden kann. Der Beitrag untersucht systemtechnische und fachliche Ansätze für Architekturen und Komponenten auf ihre Eignung für komponentenbasierte Anwendungssystementwicklung. Dabei wird deutlich, daß sich existierende Ansätze insbesondere auf systemtechnische Aspekte von Componentware konzentrieren, während die fachliche Spezifikation von Architekturen und Komponenten nur am Rande betrachtet wird. An dieser Stelle versucht die Methodik des Semantischen Objektmodells durch den Ansatz der Application Objects einen Beitrag zu leisten. Es wird ein Vorgehen skizziert, auf der Grundlage einer Unternehmensarchitektur fachliche Anwendungssystembausteine ausgehend von Geschäftsprozeßmodellen zu spezifizieren.

Inhaltsübersicht

1 Entwicklung komponentenbasierter Anwendungssysteme

2 Systemtechnische Ansätze für Componentware

2.1 Architekturen für Componentware

2.2 Komponenten-Architekturen

3 Fachlich orientierte Ansätze

3.1 Business Application Architecture

3.2 Application System Architecture der SOM-Methodik

4 Zusammenfassung

1 Entwicklung komponentenbasierter Anwendungssysteme

Die Anforderungen an betriebliche Anwendungssysteme nehmen sowohl bezüglich ihrer Leistungsmerkmale als auch bezüglich ihres Entwicklungsprozesses zu. Die Diskussion über neue

Ansätze der Entwicklung von Anwendungssystemen greift insbesondere folgende Aspekte auf ([12], [1]):

- Funktionalität und Qualität der Anwendungssysteme,
- Anpassungsfähigkeit der Anwendungssysteme an sich ändernde fachliche Anforderungen und technische Möglichkeiten,
- Wiederverwendbarkeit der Ergebnisse unterschiedlicher Entwicklungsphasen und
- Entwicklungs- und Wartungsaufwand.

Um den erhöhten Anforderungen zu begegnen, wird die Nutzung konfigurierbarer Software-Komponenten (Componentware) diskutiert, die selbst erstellt oder zugekauft werden (off-the-shelf-components [1]). Die Software-Komponenten sind für eine geplante Wiederverwendung gemäß dem Plug-and-Play-Prinzip [1] vorzusehen und sollen mit geringem Aufwand zu einer breiten Palette von Anwendungssystemen verknüpft werden können. Darüber hinaus sollen sich die Komponenten für die Verwendung in verteilten Anwendungssystemen eignen. Erst eine breite Verwendbarkeit der Software-Komponenten ermöglicht das Entstehen eines Marktes für Componentware [13].

Erforderliche Eigenschaften der Software-Komponenten hinsichtlich Interoperabilität und Integrationsfähigkeit können anhand von Integrationsmerkmalen differenziert werden, wie sie für Anwendungssysteme definiert sind. Im einzelnen sind dies das Merkmal der Plattformunabhängigkeit, die strukturorientierten Merkmale Kontrolle der Redundanz und Beherrschung der Kommunikationsstruktur, sowie die verhaltensorientierten Merkmale Gewährleistung von Konsistenz und Verfolgung der Ziele eines Anwendungssystems ([5], [6, S.197ff]).

Das Merkmal der **Plattformunabhängigkeit** wird am Konzept der programmgesteuerten Maschine deutlich [6, S.267ff]. Diese Maschine besteht aus einer Nutzermaschine sowie aus einem Programm, das die Datenobjekttypen und Operatoren der Nutzerschnittstelle auf Datenobjekttypen und Operatoren einer Basismaschine abbildet. Eine komplexe Maschine kann aus mehreren Schichten aufgebaut werden, um eine flexible Zuordnung zwischen verschiedenen Nutzer- und Basismaschinen zu erlauben. Die Nutzermaschine A in Abbildung 1 weist eine erhöhte Plattformunabhängigkeit auf, wenn mehrere austauschbare Basismaschinen mit zugehörigen Programmen verfügbar sind.

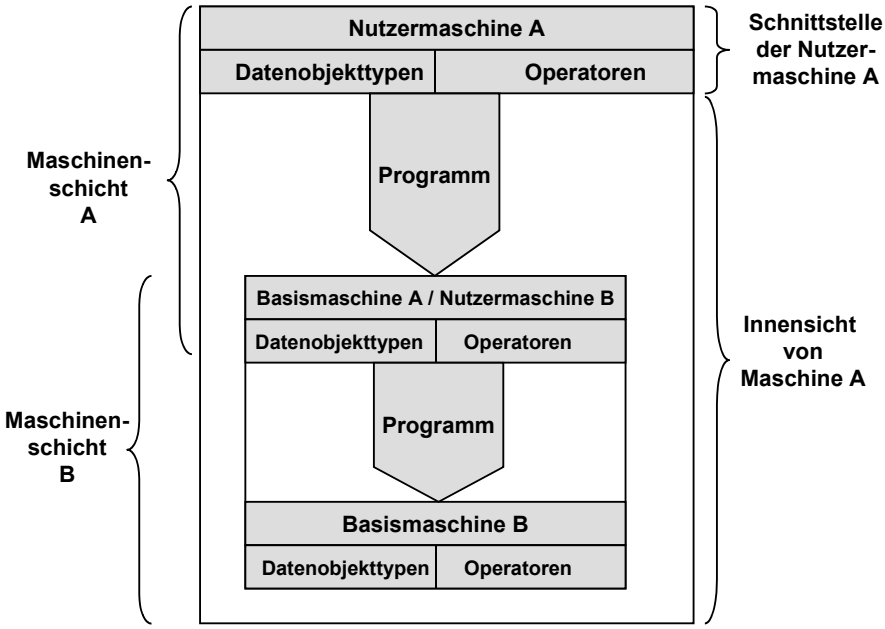

Abbildung 1: Schichtung von Nutzer- und Basismaschinen [6, S.269]

Die Schichten eines Anwendungssystems, das gemäß dem Konzept der Nutzer-/Basismaschinen strukturiert ist, können anhand semantischer Kriterien in fachliche und systemtechnische Schichten differenziert werden. Ziel bei der Entwicklung von Componentware ist die Konstruktion fachlicher Anwendungssystemschichten, die weitgehend unabhängig von systemtechnischen Schichten wie Hardware, Betriebssysteme und Datenbanksysteme, Kommunikationssysteme und Implementationssprachen sind ([10], [15], [4], [1]). In diesem Zusammenhang werden auch Altsysteme (legacy systems) als Komponenten systemtechnischer Plattformen verstanden [4].

Bezüglich des Merkmals der **Redundanz** von Systemkomponenten wird zwischen Funktionsredundanz und Datenredundanz unterschieden. Um Redundanz wirksam kontrollieren zu können, müssen für jede Komponente deren Funktionen und Daten vollständig spezifiziert sein. Problematisch kann daher die Nutzung von Altsystemen als Komponenten [4] sein, da hier selten Spezifikationen in der erforderlichen Qualität zur Verfügung stehen.

Das Merkmal der **Kommunikationsstruktur** innerhalb eines Anwendungssystems ist von der Komponentenabgrenzung auf der fachlichen Ebene abhängig. Die Schnittstellen sind semantisch zu definieren, um die Kommunikationsstruktur bereits auf fachlicher Ebene offenzulegen. Darüber hinaus ist die Verknüpfung der fachlichen mit den systemtechnischen Komponenten einfach zu gestalten [10]. Die Kommunikationsstruktur innerhalb der systemtechnischen Schichten ist ein Plattformaspekt und sollte keine Rolle für die Entwicklung und Nutzung von Komponenten spielen.

Die **Konsistenz** eines komponentenbasierten Anwendungssystems ist abhängig von den Konsistenzeigenschaften seiner Komponenten und der korrekten Konfiguration des Gesamtsystems. Voraussetzung für die Konfiguration auf der fachlichen und der systemtechnischen Ebene sind vollständige und korrekte Spezifikationen der zu integrierenden Komponenten ([10], [1]).

Einzelaspekte der Konsistenzsicherung in verteilten Systemen sind konsistente Datenhaltung, operationale Integrität nebenläufiger Zustandsübergänge sowie globaler Transaktionsschutz.

Schließlich muß die Sicherung der betrieblichen **Zielverfolgung** des Anwendungssystems gewährleistet werden. Die Interoperabilität der Komponenten allein kann nicht sicherstellen, daß das Gesamtsystem den erwarteten Beitrag zur betrieblichen Zielerreichung leistet. Die Ziele sollten in einem Unternehmensmodell, das Anwendungssysteme als maschinelle Aufgabenträger einschließt, im Rahmen der Modellierung betrieblicher Aufgaben dargestellt werden.

2 Systemtechnische Ansätze für Componentware

Ausgehend von den beschriebenen Integrationsmerkmalen werden nun existierende systemtechnische Ansätze für Software-Architekturen und darauf aufbauende Komponentenarchitekturen untersucht.

2.1 Architekturen für Componentware

Komponentenbasierte Anwendungssysteme können als Nutzermaschinen aufgefaßt werden, die geeignete Basismaschinen zur Realisierung benötigen. Lösungsvorschläge für Basismaschinen, die unter dem Begriff Middleware [15] zusammengefaßt sind, stellen Abstraktionen von systemtechnischen Plattformen dar und sind in diesem Sinne Basismaschinen eines komponentenbasierten Anwendungssystems.

2.1.1 Architekturansätze auf der Grundlage kommunizierender Objekte

Eine erste Gruppe von Architekturansätzen beruht auf der Metapher kommunizierender Objekte. Die Kommunikation der Objekte erfolgt gemäß dem Client/Server-Prinzip. Ziel ist es, eine transparente Kommunikation verteilter Client- und Server-Objekte über heterogene Plattformen zu ermöglichen.

OBJECT MANAGEMENT GROUP: CORBA

Die von der *Object Management Group* (OMG) im Rahmen der *Object Management Architecture* (OMA) standardisierte *Common Object Request Broker Architecture* (CORBA) beinhaltet als wesentlichen Baustein den *Object Request Broker* (ORB). Dieser unterstützt Interaktionen zwischen verteilten Objekten und abstrahiert dabei von zugrundeliegenden Plattformen (vgl. Abbildung 2). Die Schnittstelle dieser Basismaschine ist in der ebenfalls standardisierten *Interface Definition Language* (IDL) beschrieben ([13], [15]).

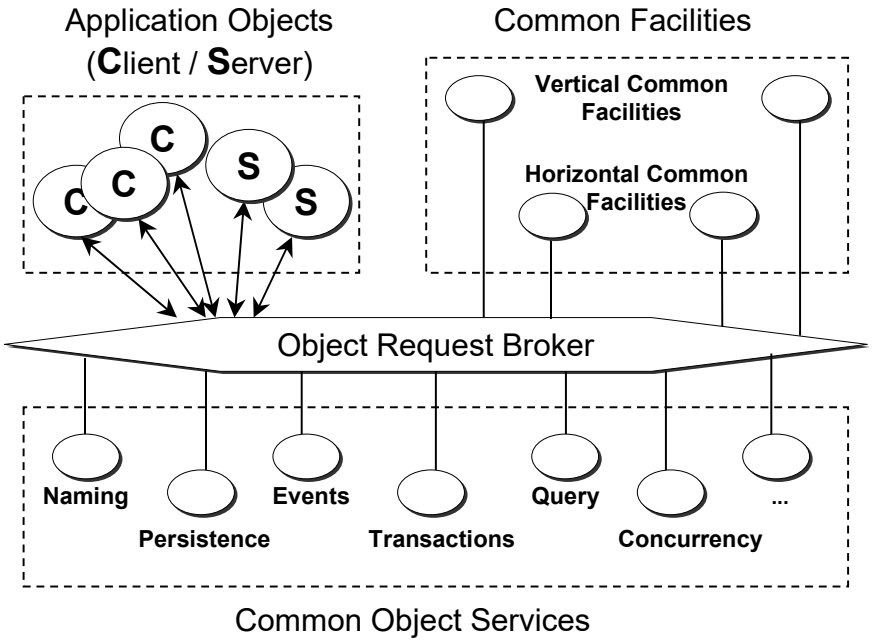

Abbildung 2: Die Object Management Architecture der OMG (in Anlehnung an [13])

Die *Object Services* spezifizieren ergänzende Dienste, um die transparente Kommunikation der Client- und Server-Objekte einfach zu gestalten. Über IDL-Interfaces werden Dienste für die Allokation von Namen, die persistente Speicherung von Objekten, die Ereignis- und Transaktionsverwaltung, Abfragemechanismen sowie die Nebenläufigkeitskontrolle für Transaktionen und Threads usw. angeboten. Diese standardisierten Dienste sind von Herstellern CORBA-konformer Produkte obligatorisch zu realisieren.

Die Implementierung der *Common Facilities* ist hingegen nicht verpflichtend. Diese auch als *CORBAfacilities* bezeichneten Interfaces stellen Services bereit, die direkt von *Application Objects* genutzt werden können. Sie sind in zwei Kategorien eingeteilt: horizontale *CORBAfacilities* sind unabhängig von der Anwendungsdomäne (z.B. E-Mail, Druckservices, grafische Benutzeroberflächen etc), während die vertikalen *CORBAfacilities* in zukünftigen Standardisierungen branchenspezifische Dienste beinhalten sollen.

Die *Application Objects* in der *Object Management Architecture* sind IDL-Schnittstellen zu anwendungsspezifischen Software-Komponenten. Vorschläge für solche Komponenten werden von der *Business Object Management Special Interest Group* (BOMSIG) erwartet, die am Konzept der *Business Objects* arbeitet. Im Gegensatz dazu stellen die in Abschnitt 3.2 vorgeschlagenen Application Objects der SOM-Methodik fachliche Anwendungssystem-Bausteine dar.

Neben CORBA existieren weitere Ansätze, die eine transparente Kommunikation verteilter Objekte unterstützen [15]. Zu nennen wäre hier das von der ISO standardisierte *Reference Model for Open Distributed Processing* (RM-ODP), das ebenfalls die Interoperabilität verteilter

Client-Server-Anwendungen spezifiziert. Mittelfristig wird eine Konvergenz dieses aus dem universitären Bereich stammenden Ansatzes mit dem Industriestandard der OMG angestrebt.

Ein weiterer Ansatz aus der Praxis ist *Portable Distributed Objects* (PDO) von NeXT. Die Architektur von PDO sichert die transparente Kommunikation von Objekten, die über LAN, WAN oder Internet realisiert sein kann. PDO basiert auf einem offenen und dynamischen Objektmodell, das die Interoperabilität mit CORBA- oder OLE-Objekten ermöglicht.

2.1.2 Der Architekturansatz des Compound Document

Eine andere Strategie für die komponentenbasierte Anwendungssystementwicklung ist der dokumentenorientierte Ansatz, der auf der Metapher des Compound Document beruht [11]. Ein Anwendungssystem stellt demnach grundsätzlich ein Compound Document dar, das andere Dokumente enthält. Zur Manipulation von Dokumenten dienen Anwendungen, die Nutzermaschinen darstellen. Die Anwendungen oder Verweise auf sie werden zusammen mit einem Dokument in das Compound Document eingebettet. Der Dokumentbegriff umfaßt dabei nicht nur textuelle Darstellungen, sondern bezieht explizit multimediale Elemente mit ein.

Der am weitesten verbreitete Ansatz ist *Object Linking and Embedding* (OLE) Version 2.0 von Microsoft. Er beruht auf der Verwendung von Containern im Sinne von Compound Documents, die Dokumente und zugeordnete Anwendungen sowie Sub-Container enthalten (vgl. Abbildung 3). Mit diesem Ansatz wird beispielsweise die Editierbarkeit typverschiedener Dokumente innerhalb eines Containers, das sogenannte In-Place-Editing, ermöglicht [13].

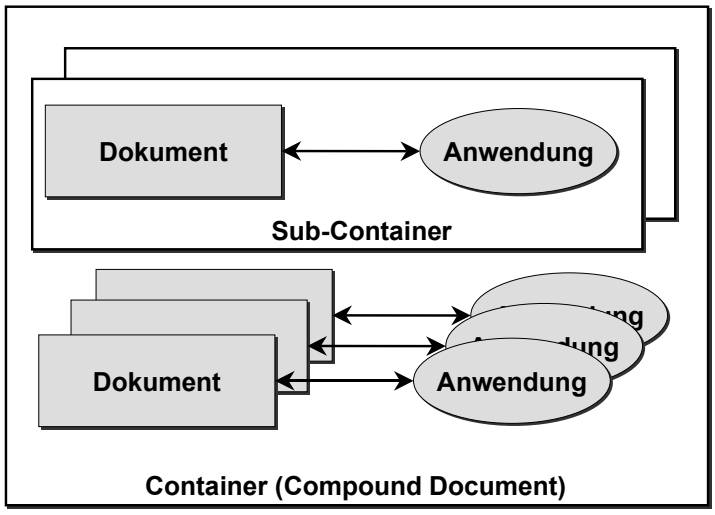

Abbildung 3: Struktur eines Compound Documents im OLE-Ansatz

Das als Grundlage dienende *Component Object Model* (COM) kennt allerdings keine Vererbung oder Polymorphie. Komponenten können aggregiert werden, wobei ihr jeweiliges Interface nach außen hin zugreifbar bleibt. Eine proprietäre Implementation von *Remote Procedure Calls* ermöglicht die Interaktion zwischen Objekten. Interoperabilität mit anderen Standards wie CORBA oder OpenDoc ist

seitens Microsoft nicht vorgesehen. Eine entsprechende Plattform (Windows, Windows NT oder UNIX/Digital) ist also Voraussetzung für die Nutzung von OLE-Komponenten.

Ein weiterer dokumentenorientierter Ansatz wird durch OpenDoc repräsentiert, eine gemeinschaftliche Entwicklung von Apple, IBM und Novell. Dieser Ansatz ist im Unterschied zu OLE stärker objektorientiert; er unterstützt Aggregation, Vererbung und Polymorphie. Grundlage ist das *System Object Model* von IBM. Seitens OpenDoc ist die Interoperabilität mit CORBA und OLE gewährleistet [13].

2.1.3 Ansätze zur Erzielung höherer Plattformunabhängigkeit

Ausgehend vom Ziel eines Komponentenmarktes stellt sich für potentielle Entwickler und Nutzer von Componentware die Frage der Plattformwahl: CORBA und OLE als die Ansätze mit der größten Verbreitung unterscheiden sich nicht nur in ihren Metaphern erheblich, sie sind nach derzeitigem Stand auch nicht interoperabel. Damit bleibt das Integrationsziel der Plattformunabhängigkeit ebenso unerreicht wie das der Kontrolle der Kommunikationsstruktur, wenn die Grenzen zwischen CORBA- und OLE-basierten Anwendungssystemen überschritten werden. Eine Nutzung beider Plattformen und damit die plattformspezifische Komponentenentwicklung könnte die Erreichung der Integrationsziele bezüglich Redundanz und Konsistenz beeinträchtigen.

Um dem entgegenzuwirken, gründete eine Gruppe von Softwareherstellern das *ComponentWare Consortium* (CWC) [4]. Ihr Ziel ist eine Architektur, die einen höheren Grad an Plattformunabhängigkeit bietet und geeignet ist, sowohl auf CORBA-kompatible *Object Request Broker* als auch auf OLE-Objekte zuzugreifen. Dazu wird eine weitere systemtechnische Abstraktionsschicht als neue Basismaschine eingeführt, die auch die Integration von bestehender Software (Legacy Systems), insbes. relationaler Datenbanksysteme (RDBMS), unterstützt. Sowohl herkömmliche Middleware als auch Legacy Systems werden als Basismaschinen niedriger Schichten eingestuft (vgl. Abbildung 4).

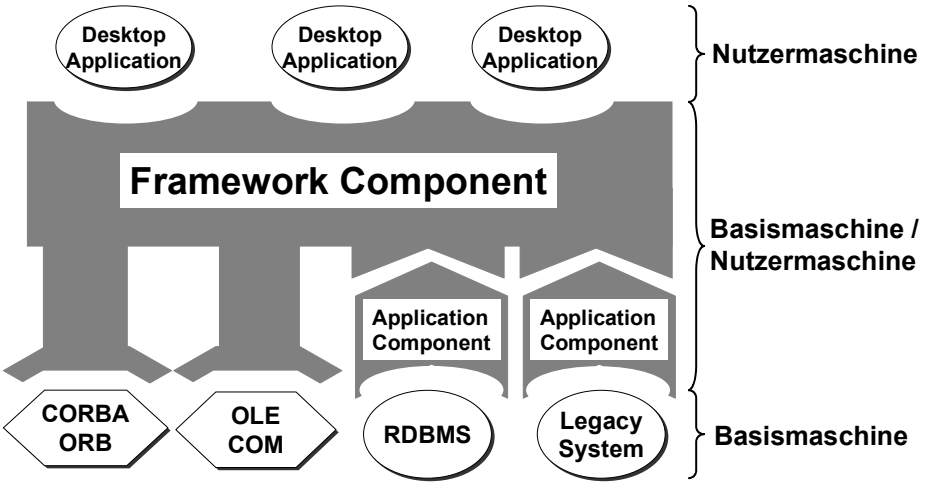

Abbildung 4: Die ComponentWare Architecture mit Abstraktionsschicht

Eine *Framework Component* bietet für sogenannte *Desktop Applications* eine einheitliche Schnittstelle, um den Zugriff auf CORBA-ORBs und auf OLE/COM transparent zu ermöglichen. Relationale Datenbanksysteme und andere objektorientierte oder nicht-objektorientierte Anwendungen können über *Application Components* eingebunden werden [4].

Abschließend ist hier der Ansatz von Java zu erwähnen, dessen Ziel ebenfalls in der Erreichung von Plattformunabhängigkeit liegt. Mit den Java Beans steht hier eine weitere Architektur für Componentware zur Verfügung, die z.B. in [9] ausführlich beschrieben wird.

2.1.4 Bewertung der Architekturansätze

Die genannten Architektur-Ansätze erfüllen die Integrationsziele in unterschiedlichem Ausmaß. Die Forderung nach voller Unabhängigkeit von Plattformaspekten wird nur durch den Ansatz des CWC erfüllt. Die bereits in der Praxis genutzten Ansätze der OMG und von Microsoft unterscheiden sich im Grad der Plattformunabhängigkeit, da OLE nur unter bestimmten Betriebssystemen nutzbar ist, während CORBA ohne Bezug auf spezielle Betriebs- und Netzwerksysteme definiert wurde.

Alle Ansätze bieten auf technischer Ebene Mittel zur Kontrolle von Redundanz und zur Erhaltung von Integritätsbedingungen an. Die Kommunikationsstruktur ist in CORBA und auch in der CWC-Architektur durch die Metapher der kommunizierenden Objekte gut zu kontrollieren, während dies in dokumentenorientierten Ansätzen wie OLE nicht explizit vorgesehen ist. Maßnahmen zur Unterstützung der Zielerreichung eines Anwendungssystems werden von keinem Ansatz berücksichtigt.

2.2 Komponenten-Architekturen

Im folgenden werden Architekturen von Komponenten beschrieben, durch deren Wiederverwendung die in Kapitel 1 beschriebenen Ziele erreicht werden sollen. Es sind Komponenten zu unterscheiden, die auf spezielle Plattformen angewiesen sind (OMG *Business Objects* (Abschnitt 2.2.1), OLE-Komponenten (Abschnitt 2.2.2)) und plattformunabhängige Komponenten. Zu letzteren zählen die Komponenten für die CWC-Architektur (Abschnitt 2.2.3) sowie die Komponenten für den Ansatz *Component-Based Software Engineering* (CBSE, Abschnitt 2.2.4). Für beide Ansätze stellen CORBA und OLE/COM die Zielplattformen dar, d.h. CWC- und CBSE-Komponenten können auf beiden Plattformen verwendet werden.

Die Beschreibung der Komponenten erfolgt, soweit möglich, anhand des ADK-Strukturmodells [6, S.270]. Danach kann ein Anwendungssystem in einen Kommunikationsteil (K), einen Anwendungsteil (A) und in einen Datenhaltungsteil (D) gegliedert werden. Innerhalb des Kommunikationsteils sind Teile für die Kommunikation mit Personen (K_p) und für die Kommunikation mit Maschinen (K_m) zu unterscheiden.

2.2.1 OMG Business Objects

Aufbauend auf der in Abschnitt 2.1 dargestellten *Object Management Architecture* stellt die von der OMG vorgeschlagene *Business Application Architecture (BAA)* [3] ein standardisiertes Framework für betriebliche Anwendungssysteme zur Verfügung. Komponenten der BAA sind *Business Objects (BO)* sowie *Presentation and Desktop* Komponenten (PD). Letztere decken den Teil Kommunikation mit Personen (K_p) ab.

Business Objects sind ausführbare SW-Bausteine; sie kapseln diejenigen Daten, Methoden, Metadaten und Business Rules, die der entsprechenden gleichnamigen fachlichen Komponente (vgl. Kapitel 3) zugeordnet sind. Einem *Business Object* können beliebig viele PD-Komponenten zugeordnet sein. Die Kommunikation zwischen einem *Business Object* und einer PD-Komponente besteht im Austausch von Daten, die die PD-Komponente dem Nutzer eines Anwendungssystems präsentieren soll, und von Metadaten, mit Hilfe derer eine PD-Komponente lernt, wie Daten darzustellen und zu verändern sind.

Ein *Business Object* kann die Teilsysteme A, D und K_p eines Anwendungssystems abdecken. Entsprechend können die Basismaschinen eines *Business Objects* ein Database Management System (DBMS als Basismaschine für D), non-object programs (insbesondere legacy systems als Basismaschine für A und D) und Object technology components wie ORB, *CORBAservices* und *CORBAfacilities* (als Basismaschine für A, D und K_m; vgl. Abschnitt 2.1) sein.

Die Architektur von BOs wird durch die *Business Object Facility (BOF)* festgelegt, die sich noch im Standardisierungsverfahren befindet. Diese ist ein "missing middle layer" zwischen den *CORBAfacilities* und BO-basierten Anwendungen [3, S.17].

In einer BO-basierten Anwendung soll es möglich sein, daß *Business Objects* ad-hoc miteinander kooperieren, ohne daß diese Kooperation vom jeweiligen Entwickler geplant war. Dies ist nur durch eine semantische Definition der Schnittstellen möglich. Auch hier werden Standards von der *Business Object Facility* erwartet.

2.2.2 OLE-Komponenten

Neben den in Abschnitt 2.1.3 genannten Anwendungen und Dokumenten, die in Container eingebettet werden können, existieren *OLE Custom Controls* (OCX) als Komponenten, mit denen geplant Wiederverwendung betrieben werden kann. Mit ihnen steht ein weiteres Instrument für die Entwicklung komponentenbasierter Anwendungssysteme zur Verfügung.

OCX können ebenso wie Dokumente und Anwendungen in Container eingebettet werden. Daneben besitzen sie die Fähigkeit, Eingaben des Nutzers in Ereignisse umzuwandeln; diese teilt ein OCX dem ihn enthaltenden Container mit. OLE stellt mit Hilfe des Konzeptes der *connectable objects* [13,

S.528] standardisierte Beziehungen nur zwischen Containern und OCX zur Verfügung, nicht jedoch zwischen mehreren OCX. Diese müssen explizit beschrieben werden.

Die Wiederverwendung von OCX wird durch die Basisklasse *OLE Control* innerhalb der *Microsoft Foundation Classes* unterstützt. Die Architektur der OCX wird bestimmt durch das *Component Object Model* (vgl. Abschnitt 2.1.2), das einheitliche Schnittstellen für alle OLE-Objekte definiert; eine eingehende Darstellung findet sich z.B. in [13].

Bei OCX handelt es sich überwiegend um visuelle Komponenten, die mit einer Entwicklungsumgebung installiert werden und vom Entwickler eines Anwendungssystems wiederverwendet werden können. Grundsätzlich kann ein OCX den A-, D- und K-Teil eines Anwendungssystems abdecken. Durch die Fähigkeit der OCX, auf Eingaben des Nutzers zu reagieren, liegt der Schwerpunkt der Anwendungsmöglichkeiten im Bereich der Gestaltung der Mensch-Maschine-Kommunikation (K_p).

2.2.3 Der Komponentenbegriff des ComponentWare Consortium

Innerhalb der *ComponentWare Architecture* (vgl. Abschnitt 2.1, insbes. Abbildung 4) können gemäß dem Client/Server Prinzip zwei Typen von Komponenten unterschieden werden. Auf der Client-Seite existieren sogenannte *Desktop Applications* [4, S.5]. Diese übernehmen die Kommunikation mit dem Nutzer (K_p); sie können daneben auch Funktionen des Anwendungsteils (A) realisieren. *Application Components* und *Framework Components* übernehmen den Teil der Maschine-Maschine-Kommunikation des Anwendungssystems, d.h. als Server stellen sie den Clients standardisierte Zugriffsmethoden auf Anwendungen zur Verfügung. Diese Anwendungen enthalten die auf der Client-Seite fehlenden Funktionen des Anwendungsteils sowie den Datenhaltungsteil.

Über die Architektur von *Desktop Applications* macht das *ComponentWare Consortium* keine Aussagen. *Desktop Application*s sind keine Komponenten, mit denen geplante Wiederverwendung betrieben werden soll, um die in Kapitel 1 beschriebenen Ziele zu erreichen.

Als wiederverwendbare Komponenten versteht das *ComponentWare Consortium* jede Art von Anwendung, insbesondere DBMS und Legacy Systems. Aufgrund der Heterogenität dieser Anwendungen werden keine Annahmen über deren Architektur gemacht. Einzige Voraussetzung für die (Wieder-)Verwendung einer Anwendung in einem Anwendungssystem, welches der *ComponentWare Architecture* folgt, ist die Existenz eines *Application Programming Interface*; dieses wird mit Hilfe einer speziell für diese Anwendung entwickelten *Application Component* auf das *ComponentWare Foundation Interface* abgebildet [4, S.12]. Aus einer plattformspezifischen *Framework Component* und der *Application Component* werden ausführbare Software-Bausteine generiert, die mit beliebigen komponentenbasierten *Desktop Applications* kooperieren.

2.2.4 Der Komponentenbegriff des Component-Based Software Engineering (CBSE)

Component-Based Software Engineering [1] hat zum Ziel, werkzeuggestützte Plug-and-Play-Software zu ermöglichen. Der Ansatz dient der Entwicklung eines Anwendungssystems aus unabhängig voneinander implementierten Komponenten. Die Unterstützung richtet sich vor allem auf die explizite Beschreibung der Beziehungen zwischen Komponenten.

Eine Komponente repräsentiert "a significant functional unit of a system" [1, S.20]; sie ist wiederverwendbar und kann von unterschiedlichen Anwendungen genutzt werden. Es bleibt offen, welche Teile eines Anwendungssytems von einer Komponente abgedeckt werden. Auch über die Granularität von Komponenten und deren Architektur wird keine Aussage gemacht. Eine Komponente wird durch ihre Schnittstellen beschrieben; deren Beschreibung erfolgt in einer *Architecture Specification Language*, einer Erweiterung von IDL.

Die Leistungen, die eine Komponente anderen Komponenten zur Verfügung stellt oder von dieser benötigt werden, werden in *provided* bzw. *required interfaces* beschrieben. Beziehungen zwischen Komponenten werden dadurch beschrieben, daß zur build-time die zu einer Leistung gehörigen Client- und Server-Komponenten explizit über ihre *required* bzw. *provided interfaces* verbunden werden. Die Überprüfung der Kompatibilität der Schnittstellen übernimmt ein spezielles Werkzeug, das bei Inkompatibilität Adapter-Komponenten generiert. Die Erreichung der Plattformunabhängigkeit wird durch ein Werkzeug unterstützt, das in Abhängigkeit der gewählten Plattform aus Komponenten, die auf diese Weise gebildet wurden, ausführbare Software-Bausteine ableitet.

2.2.5 Bewertung der Ansätze

Die bisher skizzierten Ansätze (im folgenden BO, OCX, CWC und CBSE genannt) werden nun den in Kapitel 1 beschriebenen Zielen gegenüber gestellt.

Plattformunabhängigkeit ist nur bei CBSE und CWC gegeben; alle Ansätze ermöglichen den Einsatz mehrerer Implementationssprachen, lediglich die Sprachen zur Beschreibung der Schnittstellen sind vorgegeben. Die Integration von Altsystemen ist bei BO, CWC und CBSE möglich; bei OCX sind aufgrund des überwiegenden Einsatzes bei der Gestaltung von grafischen Benutzerschnittstellen Einschränkungen zu machen.

Bei BO ist die implizite Möglichkeit der Verknüpfung von Komponenten gegeben, sofern die Forderung nach Unterstützung der ad-hoc Interoperabilität durch semantisch definierte Schnittstellen erfüllt wird. Bei den übrigen Ansätzen muß die Verknüpfung explizit angegeben werden.

Alle Ansätze bieten die Möglichkeit zur konsistenten Datenhaltung. Eine Spezifikation der Architektur wiederverwendbarer Komponenten liegt bei CWC, OCX und CBSE vor; für BO sind keine Aussagen möglich, da die OMG *Business Object Facilities* sich noch im Standardisierungsverfahren befinden.

Auffallend ist sowohl inerhalb der Ansätze als auch ansatzübergreifend die stark unterschiedliche Granularität der wiederverwendbaren Komponenten. Sie reicht von einfachen visuellen Steuerungselementen der grafischen Benutzerschnittstelle bis hin zu Legacy Systems, die beliebig komplex sein können. Einheitliche Abgrenzungskriterien für Software-Bausteine existieren in keinem der Ansätze.

3 Fachlich orientierte Ansätze

An eine komponentenbasierte Anwendungssystementwicklung werden hohe Erwartungen bezüglich Wiederverwendbarkeit, Anpaßbarkeit und Erweiterbarkeit der Entwicklungsergebnisse geknüpft (vgl. z.B. [1]). Hierzu ist es notwendig, die fachliche Architektur eines komponentenbasierten Anwendungssystems zu spezifizieren. Zum einen wird damit deutlich, in welchem Kontext die jeweiligen Komponenten wiederverwendet werden können, zum anderen ist eine Anpassung der Komponenten an veränderte fachliche Anforderungen nur dann möglich, wenn die Auswirkungen der Veränderung auf die einzelnen Komponenten nachvollziehbar sind.

In diesem Kapitel werden Ansätze beschrieben, die eine Verknüpfung zwischen der fachlichen und der systemtechnischen Ebene vornehmen, um wiederverwendbare Komponenten zu identifizieren. Zunächst wird der Ansatz der OMG *Business Application Architecture* vorgestellt. Anschließend wird der Ansatz der fachlichen Spezifikation von Komponenten auf Basis der SOM-Methodik beschrieben.

3.1 Business Application Architecture

Die *Business Application Architecture* [3] ist ein Vorschlag der OMG, eine Verbindung zwischen der fachlichen Ebene und den *Business Objects* der systemtechnischen Ebene (vgl. Abschnitt 2.2.1) herzustellen. Die fachliche Architektur umfaßt als Bausteine ebenfalls *Business Objects* und *Presentation Components*. Einem *Business Object* können beliebig viele *Presentation Components* zugeordnet sein.

Mit Hilfe von *Business Objects* und Beziehungen zwischen diesen wird ein Unternehmensmodell erstellt. Bestandteile eines *Business Objects* sind Namen, Attribute, Verhalten, Beziehungen zu anderen *Business Objects* sowie Business Rules, die das Verhalten steuern. Einem *Business Object* der fachlichen Ebene ist genau ein *Business Object* auf systemtechnischer Ebene zugeordnet.

Die Definition “A business object is a representation of a thing active in the business domain,...”[3, S.3] zeigt, daß auf der fachlichen Ebene noch keine klaren Abgrenzungskriterien für *Business Objects*

vorliegen. Die Tatsache, daß diese auf der fachlichen Ebene auch als “business-object abstraction” [2] bezeichnet werden, deutet darauf hin, daß keine strikte Trennung zwischen systemtechnischer und fachlicher Ebene vorgenommen wird. Im Vordergrund steht vielmehr der Versuch, ausgehend von systemtechnisch beschriebenen Komponenten durch Abstraktion fachliche Bausteine zu erhalten.

3.2 Application System Architecture der SOM-Methodik

3.2.1 Unternehmensarchitektur der SOM-Methodik

Das Semantische Objektmodell (SOM) ist eine umfassende Methodik zur Analyse und Gestaltung betrieblicher Systeme [7]. SOM unterscheidet zwischen Aufgaben- und Aufgabenträgerebene eines betrieblichen Systems. Auf der Aufgabenträgerebene werden Anwendungssysteme betrachtet. Die zugehörige Aufgabenebene ermöglicht eine auf fachliche Aspekte ausgerichtete komponentenbasierte Anwendungssystem-Architektur.

Modelle betrieblicher Systeme in praxisrelevanten Größenordnungen weisen meist eine sehr hohe Komplexität auf und werden daher in Modellebenen und korrespondierende Sichten strukturiert. Für jede Modellebene werden die verfügbaren Modellelemente und deren Beziehungen in Form eines Meta-Modells angegeben. Die Beziehungen zwischen benachbarten Modellebenen sind durch Beziehungs-Meta-Modelle beschrieben.

Der SOM-Ansatz unterscheidet die drei Modellebenen Unternehmensplan, Geschäftsprozeßmodelle und Anwendungssysteme (Abbildung). Sie sind Teile der Unternehmensarchitektur der SOM-Methodik und dienen der ganzheitlichen Erfassung eines Unternehmens einschließlich Unternehmensziele und Leistungserstellung. Im folgenden werden nur die Ebenen Geschäftsprozeßmodelle (2) und Anwendungssysteme (3) betrachtet.

In Ebene 2 werden die Aufgaben eines betrieblichen Systems als verteilte Geschäftsprozesse modelliert. Struktur- und verhaltensorientierte Sichten auf das System der Geschäftsprozesse erhöhen die Verständlichkeit der Modelle. Das Interaktionsschema beschreibt die strukturorientierte Sicht auf das Geschäftsprozeßmodell, ein Vorgangs-Ereignis-Schema spezifiziert das Verhalten der Geschäftsprozesse.

In Modellebene 3 werden Anwendungssysteme als Ressourcen für die Durchführung automatisierter Aufgaben beschrieben. Die fachliche Anwendungssystemarchitektur besteht aus einem konzeptuellen Objektschema (KOS) und einem Vorgangs-Objekt-Schema (VOS). Das KOS enthält die konzeptuellen Objekttypen eines Anwendungssystems und deren Beziehungen. Das VOS beschreibt in Form von Vorgangsobjekttypen das Zusammenwirken konzeptueller Objekttypen bei der Durchführung betrieblicher Aufgaben und nimmt dabei Bezug auf das Vorgangs-Ereignis-Schema der Ebene 2.

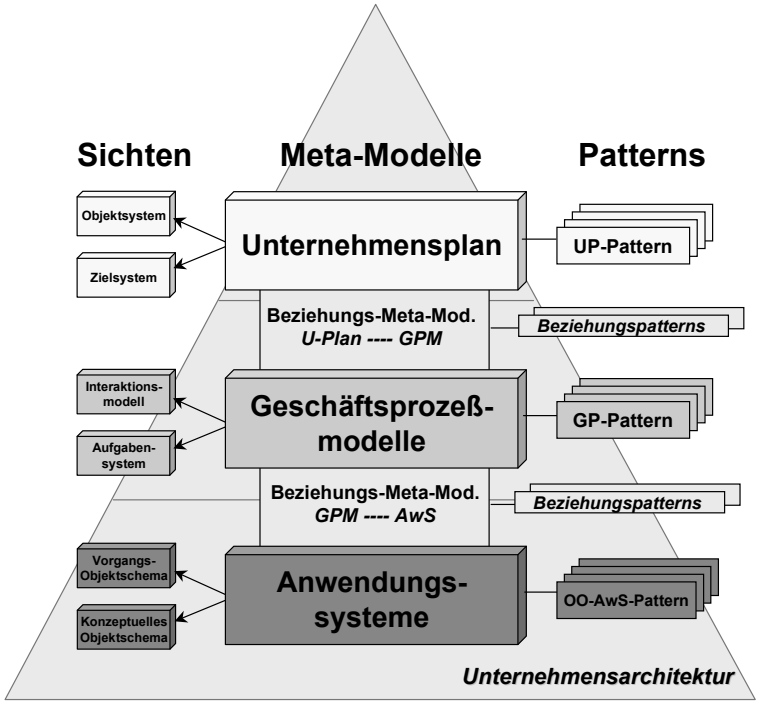

Abbildung 5: Unternehmensarchitektur des SOM-Ansatzes

3.2.2 Objektorientierte Geschäftsprozeß- und Anwendungssystemmodellierung in SOM

Die SOM-Methodik nutzt für die Geschäftsprozeß- und Anwendungssystemmodellierung das objektorientierte Paradigma mit folgenden Merkmalen [16]:

- Modellkomponenten sind Objekte, die Zustände und Operatoren kapseln.
- Objekte sind über Transaktionen lose gekoppelt.
- Objekte werden zu Aggregations- und Generalisierungshierarchien zusammengefaßt (dieser Aspekt der Modellierung wird im weiteren nicht näher betrachtet).

Die durchgängig objektorientierte Modellierung von Geschäftsprozessen und Anwendungssystemen (AwS) ermöglicht eine enge Verknüpfung der beiden Modellebenen. Durch die Objektorientierung auf der Geschäftsprozeßebene wird die Voraussetzung geschaffen, Anwendungssysteme als Systeme lose gekoppelter fachlicher Komponenten zu beschreiben.

Geschäftsprozeßmodellierung in der SOM-Methodik

Bei der objektorientierten Geschäftsprozeßmodellierung der SOM-Methodik werden eine struktur- und eine verhaltensorientierte Sicht unterschieden. Die strukturorientierte Sicht verwendet die Modellbausteine betriebliches Objekt und betriebliche Transaktion. Jede Transaktion verbindet genau zwei betriebliche Objekte und stellt einen Interaktionskanal für den Austausch von Lenkungsnachrichten und Leistungspaketen dar. Die Interaktionen folgen den kybernetischen

Prinzipien Regelung oder Verhandlung. Sie verwenden fachliche Protokolle, die Regelungsstrukturen unter Verwendung von Steuer- und Kontroll-Transaktionen (S- und K-Transaktionen) beschreiben und Verhandlungsstrukturen in Form von Anbahnungs-, Vereinbarungs- und Durchführungstransaktionen (A-, V-, D-Transaktionen) erfassen.

In der verhaltensorientierten Sicht werden die Modellbausteine Aufgabe und Ereignis verwendet. Das Verhalten eines Objekts wird durch die ihm zugeordneten Aufgaben beschrieben. Diese werden in Form von Vorgängen durchgeführt, die von Vorereignissen ausgelöst werden und Nachereignisse erzeugen. Im folgenden werden die beschriebenen Sichten auf ein Geschäftsprozeßmodell an einem Beispiel verdeutlicht.

Beispiel eines Geschäftsprozesses aus dem Bereich Finanzdienstleistung

Im nachstehenden Interaktionsschema (vgl. Abbildung 6) ist die strukturorientierte Sicht eines Geschäftsprozesses "Kapitalbeschaffung" dargestellt. Ein derartiger Geschäftsprozeß kann in vielen Unternehmen Verwendung finden (z.B. Geschäftsbanken, Leasinganbieter etc.), die sich mit der Erbringung von Finanzdienstleistungen beschäftigen. Die Kapitalbeschaffung dient der Refinanzierung der für die Kunden des Unternehmens erbrachten Finanzierungsleistungen. In Abhängigkeit von der gewählten Refinanzierungsart steht das Objekt Kapitalbeschaffung mit unterschiedlichen Partnern am Kapitalmarkt in Kontakt. Aus fachlichen Gründen wird das komplexe Objekt Kapitalbeschaffung nach dem Regelungsprinzip in die Objekte Refinanzierungsentscheidung und Refinanzierungsabwicklung zerlegt. Zwischen den beteiligten betrieblichen Objekten werden nun Protokolle unter Nutzung von Transaktionen spezifiziert. Diese Protokolle beschreiben die Koordination der Objekte im Rahmen der Erstellung und Übergabe der Refinanzierungsleistung.

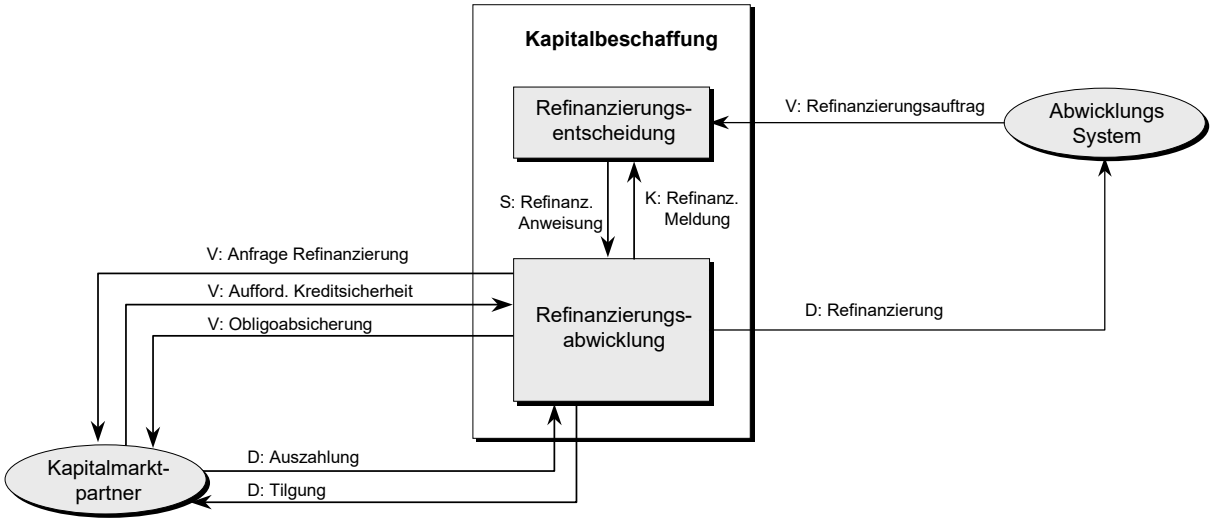

Abbildung 6: Interaktionsschema des Geschäftsprozesses "Kapitalbeschaffung"

Das Verhalten des Geschäftsprozesses wird im Vorgangs-Ereignis-Schema durch Aufgaben, Ereignisse und Transaktionen modelliert. Jede Transaktion wird durch zwei Aufgaben T> ("send T")

und >T ("receive T") der beteiligten betrieblichen Objekte durchgeführt. Mehrere Aufgaben eines Objekts sind durch (objektinterne) Ereignisse gekoppelt, welche die Reihenfolgebeziehungen zwischen den Aufgaben eines Objekts festlegen. Abbildung 7 enthält einen Ausschnitt des Vorgangs-Ereignis-Schemas des Geschäftsprozesses Kapitalbeschaffung. Die Aufgabenbezeichnungen sind aus den Transaktionsbezeichnungen abgeleitet. Die schichtenweise Anordnung der Aufgaben verdeutlicht die Zuordnung von Aufgaben zu Objekten.

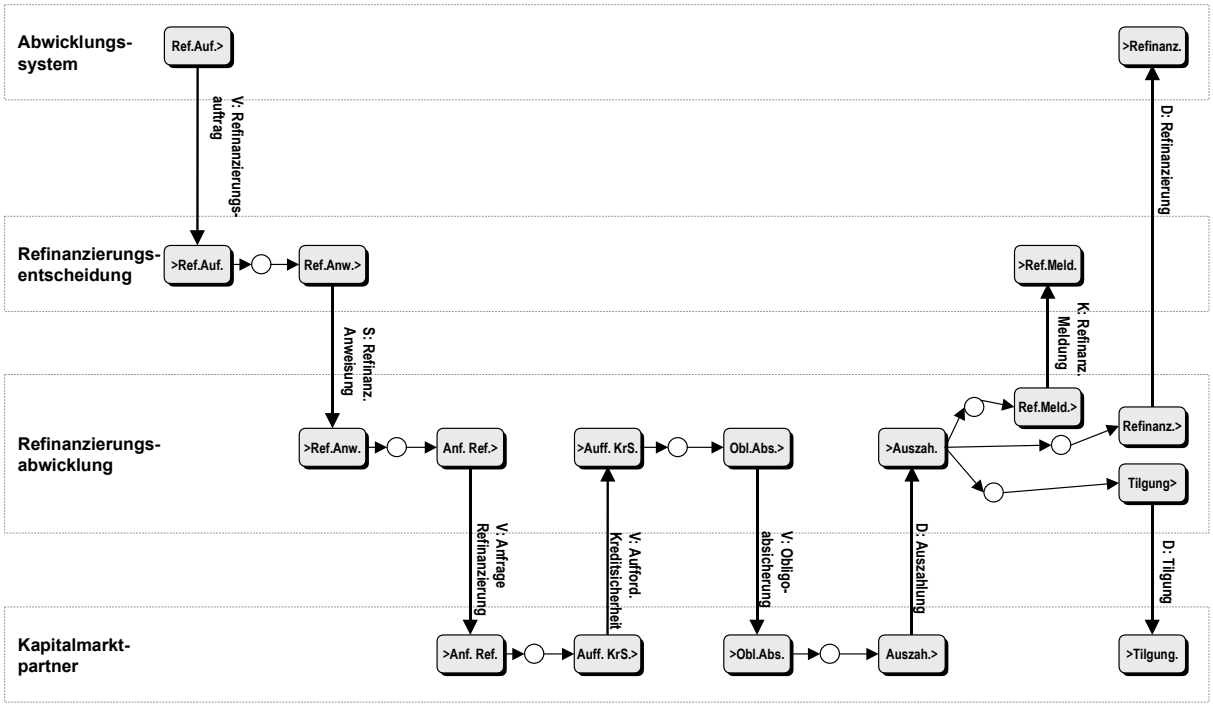

Abbildung 7: Vorgangs-Ereignis-Schema "Kapitalbeschaffung"

Objektorientierte Anwendungssystemspezifikation

Die fachliche Spezifikation der Anwendungssysteme erfolgt in der SOM-Methodik auf der Grundlage eines Geschäftsprozeßmodells. Ein betriebliches Anwendungssystem automatisiert einige oder alle informationsverarbeitenden Aufgaben eines oder mehrerer betrieblicher Objekte. Für jedes Anwendungsssystem wird ein konzeptuelles Objektschema sowie ein korrespondierendes Vorgangs-Objekt-Schema entwickelt.

Der Ableitung der AwS-Spezifikation liegen folgende Regeln zugrunde:

- Die Modellbausteine betriebliches Objekt, Transaktion, Leistung und Aufgabe werden durch Objekttypen bzw. Klassen auf der AwS-Ebene modelliert.
- Objekte, Transaktionen und Leistungen werden in konzeptuelle Objekttypen (KOT) abgebildet.
- Aufgaben werden in Form von Vorgangs-Objekttypen (VOT) spezifiziert.

Zur Ableitung einer initialen AwS-Spezifikation wird die unterste Zerlegungsebene eines hierarchischen Geschäftsprozeßmodells herangezogen. Die entstehenden Schemata haben folgende Inhalte:

Ein initiales KOS wird aus dem Interaktionsschema in Verbindung mit dem Vorgangs-Ereignis-Schema abgeleitet. Es enthält als Modellbausteine konzeptuelle Objekttypen und interacts_with-Beziehungen zwischen KOTs. Letztere stellen semantische Integritätsbedingungen dar.

Ein initiales Vorgangs-Objekt-Schema wird unmittelbar aus dem Vorgangs-Ereignis-Schema des Geschäftsprozeßmodells abgeleitet. Jede Aufgabe führt zu einem Vorgangsobjekttyp (VOT), objektinterne Ereignisse zwischen Aufgaben führen zu interacts_with-Beziehungen zwischen den korrespondierenden VOTs. Da jede Transaktion durch zwei Aufgaben durchgeführt wird, die den betrieblichen Objekten zugeordnet sind, werden die VOTs über interacts_with-Beziehungen mit dem die Transaktion abbildenden KOT verknüpft.

Einzelne Anwendungssysteme können grundsätzlich durch eine geeignete Projektion auf das Geschäftsprozeßmodell spezifiziert werden. Die daraus resultierenden Objektschemata beschreiben ein Anwendungssystem aus fachlicher Sicht mit maximaler Verteilung der Aufgaben (bzgl. der untersten Zerlegungsebene des Geschäftsprozeßmodells).

Die fachliche Spezifikation eines Anwendungssystems für den Geschäftsprozeß "Kapitalbeschaffung" besteht aus einem konzeptuellen Objektschema und einem Vorgangsobjektschema (vgl. Abbildung 8).

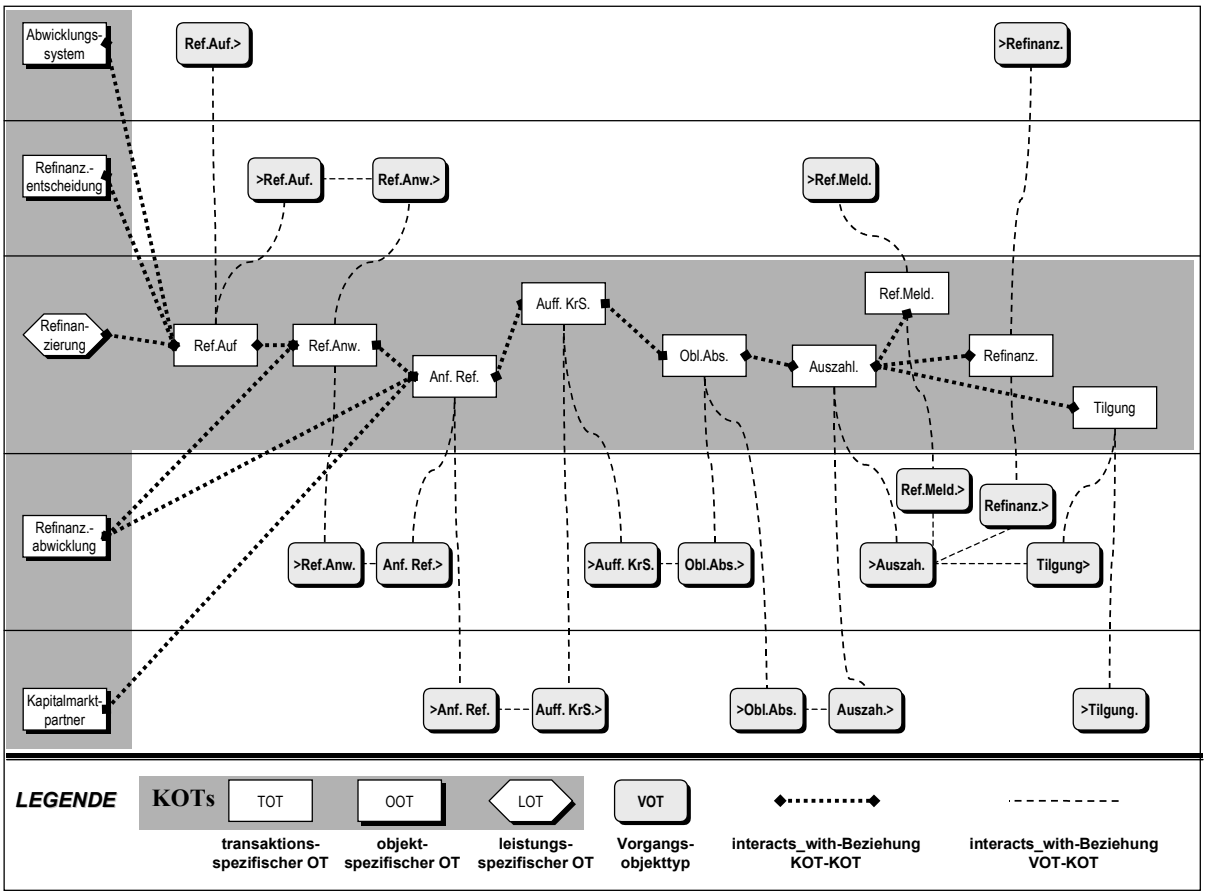

Abbildung 8: KOS/VOS der "Kapitalbeschaffung"

Im allgemeinen ist die Granularität von KOTs und VOTs zu fein, um wiederverwendbare Teil-Anwendungssysteme zu spezifizieren. Objekttyp- bzw. Klassenspezifikationen weisen nicht die für Wiederverwendbarkeit und Anpaßbarkeit erforderlichen Eigenschaften auf. Vielmehr werden Komponenten benötigt, die bezüglich der Wiederverwendungszielsetzung geeignete Klasssengruppen kapseln [14, S.40].

3.2.3 Application Objects als fachliche Komponenten objektorientierter Anwendungssysteme

Eine komponentenbasierte Anwendungssystementwicklung weist eine Reihe von zukunftsweisenden Gestaltungsmerkmalen auf (vgl. [12], [1]). Allerdings liegen kaum Vorschläge für eine geeignete Abgrenzung von Komponenten vor. Gegenwärtig bestimmen die durch neue Plattformkonzepte entstandenen Möglichkeiten (siehe Kapitel 2) die Diskussion. Im folgenden wird daher ein Ansatz zur Definition fachlicher Anwendungssystem-Bausteine skizziert, der im BMBF-Projekt 'Wiederverwendbare und erweiterbare Geschäftsprozeß- und Anwendungssystem-Architekturen' (WEGA) entwickelt wurde [17]. Dabei stehen folgende Forderungen an fachlich determinierte AwS-Bausteine im Vordergrund:

- Die Granularität der Bausteine ist allein durch fachliche Anforderungen abzuleiten. Gemäß der SOM-Methodik bestimmt der Detaillierungsgrad des Geschäftsprozeßmodells die Abgrenzung der AwS-Komponenten.
- Jede AwS-Komponente muß aus dem Geschäftsprozeßmodell ableitbar sein. Der fachliche Kontext der Komponenten ist durch das Geschäftsprozeßmodell definiert und dokumentiert.
- Die Komponenten-Architektur soll eine hinreichend überschaubare Komplexität der AwS-Spezifikation ermöglichen. Dies wird durch Verwendung genau eines Bausteintyps unterstützt.
- Die Schnittstellenspezifikation hat in der Ebene der fachlichen AwS-Spezifikation ausschließlich semantische Merkmale und beschreibt ein fachliches Protokoll zwischen AwS-Komponenten.

Der Ansatz sieht vor, auf der Grundlage des Geschäftsprozeßmodells Application Objects als fachliche Bausteine des zu spezifizierenden Anwendungssystems zu bilden. Dazu wird aus jedem betrieblichen Objekt ein Application Object abgeleitet. Die Beziehungen zwischen Application Objects sind durch die Transaktionen des Geschäftsprozeßmodells determiniert. Die Transaktionen zwischen zwei betrieblichen Objekten spezifizieren dabei ein fachliches Protokoll, durch welches die Konsistenz der Zustandsräume der (Teil-) Anwendungssysteme gewährleistet wird. Die Elemente der Application Objects sind durch Regeln der objektorientierten AwS-Spezifikation in der SOM-Methodik definiert [8]. Danach besteht die Mikro-Architektur eines Application Object aus folgenden Bestandteilen (vgl. Abbildung):

- Vorgangs-Objekttypen, die den Verhaltensanteil des im Application Object abgebildeten Geschäftsprozesses spezifizieren.
- Genau ein objektspezifischer Objekttyp (OOT), der die Spezifikation des objektinternen Speichers eines betrieblichen Objekts darstellt.
- Leistungsspezifische Objekttypen (LOT), welche die Services spezifizieren, die ein Application Object erbringen kann.
- Transaktionsspezifische Objekttypen (TOT), welche die semantische Spezifikation der Schnittstelle des Application Objects beinhalten.

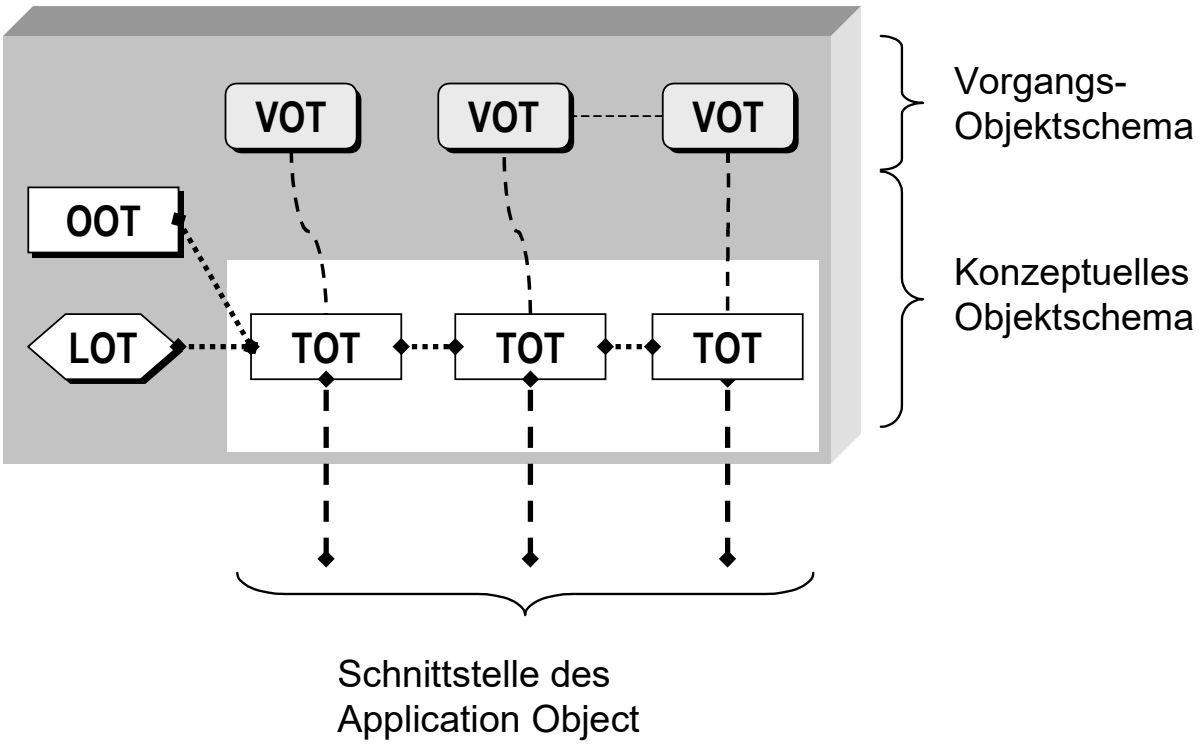

Abbildung 9: Mikro-Architektur eines Application Objects

Die Schnittstelle eines Application Objects definiert die Außensicht des Bausteins in Form transaktionsspezifischer Objekttypen. Lokale VOS-/KOS-Schemata bilden die Innensicht und sind nur über die TOT-Elemente der Schnittstelle erreichbar.

In Abbildung 10 sind die Application Objects des Geschäftsprozesses Kapitalbeschaffung dargestellt. Es wird deutlich, daß die Komponenten nur über ihre Schnittstelle lose gekoppelt sind. Die lokalen VOS-/KOS-Anteile der Application Objects sind nicht dargestellt, da sie für die Interaktion der Komponenten nur nachrangige Bedeutung haben. Um so definierte Komponenten in anderen Kontexten wiederzuverwenden, sind ausschließlich die Schnittstellen in Form der jeweiligen TOTs zu berücksichtigen.

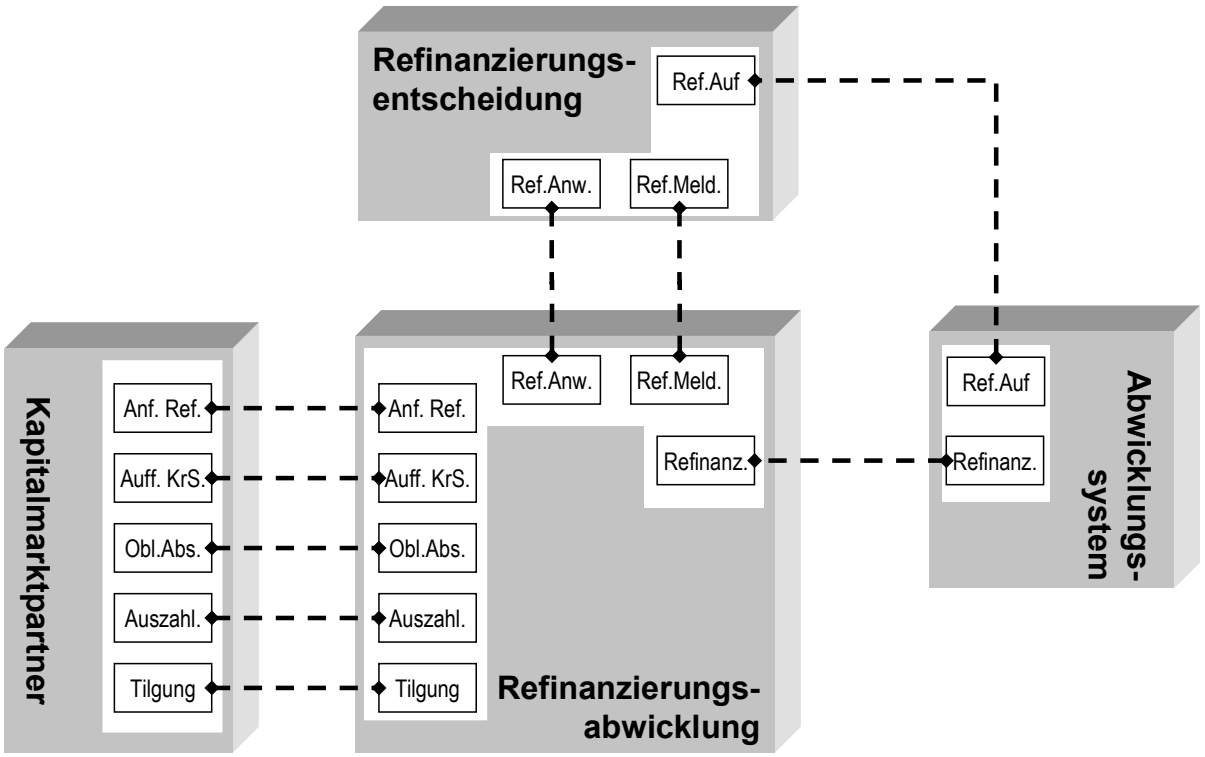

Abbildung 10: Application Objects des Geschäftsprozesses Kapitalbeschaffung

Das Konzept trägt zur Plattformunabhängigkeit der Komponenten bei, da die Außensicht der Application Objects in Form der Schnittstellenspezifikation von aktuellen Plattformdefinitionen wie z.B. CORBA unabhängig ist. Erst mit Wahl einer konkreten Plattform ist eine Realisierungsform für die Schnittstellenobjekte zu bestimmen. Bei der Integration von zwei Application Objects kann die Objektverwaltung unterschiedlich realisiert werden. Sie wird entweder nur von einem oder von beiden Application Objects übernommen. Eine weitere Möglichkeit besteht in der Einführung eines speziellen Serverobjekts für die Objektverwaltung. Die zu implementierenden Protokolle für die Objektintegration sind im Anschluß an diese Verteilungsentscheidung ableitbar.

Anhand der Spezifikationsbestandteile KOS und VOS eines Application Object können Redundanzaspekte differenziert betrachtet werden. Sie sichern die Möglichkeit einer effizienten Kontrolle der Redundanz.

Die Konsistenz einer fachlichen Anwendungssystemspezifikation wird durch zwei Aspekte unterstützt. Zum einen führt die Ableitung der Application Objects nach den oben beschriebenen Regeln zu einer Erhaltung der Strukturaspekte des Geschäftsprozeßmodells. Andererseits erfolgt die Integration der Application Objects ausschließlich über die Schnittstellendefinition in Form von transaktionsspezifischen Objekttypen.

4 Zusammenfassung

Die bisher existierenden Ansätze für die komponentenbasierte Softwareentwicklung konzentrieren sich auf die Definition systemtechnischer Architekturen und Komponenten. Fachliche Aspekte werden eher am Rande behandelt, obwohl die Notwendigkeit für eine fachlich getriebene Entwicklung und Nutzung von Componentware anerkannt wird. Hier setzt die SOM-Methodik an, indem aufbauend auf einem Geschäftsprozeßmodell eine fachliche Anwendungssystem-Architektur und Komponenten mit semantisch definierten Schnittstellen abgeleitet werden.

Die SOM-Methodik ist auf kommunizierende Objekte ausgerichtet, wie sie u. a. von CORBA unterstützt werden. Eine Nutzung CORBA-kompatibler Basismaschinen für die systemtechnische Architektur eines Anwendungssystems liegt daher nahe. Dokumentenorientierte Ansätze wie OLE sind auf die komponentenbasierte Gestaltung von Desktop-Applications ausgerichtet. Sie unterstützen die Bearbeitung komplexer Verbunddokumente.

5 Literatur

[1] Bronsard F., Bryan D., Kozaczynski W. et al.: Toward Software Plug-and-Play. In: ACM Software Engineering Notes, Vol. 22 No. 3 May 1997, P. 19-29 (Proceedings of 1997 Symposium on Software Reusability)

[2] Burt C. (Hrsg.): OMG BOMSIG survey with published definition of a business object. OMG document 95-02-04, 1995.

[3] Casanave C.: Business-Object Architectures and Standards. Miami o.J.

[4] ComponentWare Architecture. White Paper des ComponentWare Consortiums © 1995 I-Kinetics, Inc.

[5] Ferstl, O.K.: Integrationskonzepte Betrieblicher Anwendungssysteme. Fachberichte Informatik der Universität Koblenz-Landau Nr. 1/1992

[6] Ferstl O.K., Sinz E.J.: Grundlagen der Wirtschaftsinformatik – Konzepte, Modelle und Methoden. 2. Auflage, Oldenbourg, München 1994.

[7] Ferstl O.K., Sinz E.J.: Der Ansatz des Semantischen Objektmodells (SOM) zur Modellierung von Geschäftsprozessen. Wirtschaftsinformatik 37 (1995) 3, S. 209 - 220.

[8] Ferstl O.K., Sinz E.J.: Multi-Layered Development of Business Process Models and Distributed Business Application Systems. An Object-Oriented Approach. In: König W., Kurbel K., Mertens P., Preßmar D. (ed.): Distributed Information Systems in Business. Springer, Berlin 1996, p. 159 - 179

[9] Flanagan D.: JAVA in a Nutshell. Cambridge 1997

[10] Johnson R.E.: Components, Frameworks, Patterns (extended abstract). In: ACM Software Engineering Notes, Vol. 22 No. 3 May 1997, P. 10-17 (Proceedings of 1997 Symposium on Software Reusability)

[11] Malischewski C.: ComponentWare. Wirtschaftsinformatik 37 (1995) 1, S. 65-67

[12] Nierstrasz O., Meijler T.D.: Research Directions in Software Composition. In: ACM Computing Surveys, Vol. 27 No. 2, June 1995, P. 262-264

[13] Orfali R., Harkey D., Edwards J.: The Essential Distributed Objects Survival Guide. New York, 1996.

[14] Raue H.: Konzeption einer objektorientierten Methode zur Entwicklung von wiederverwendbaren betrieblichen Anwendungssystemen. Deutscher Universitäts-Verlag, Wiesbaden 1996

[15] Scharf T.: Architekturen und Technologien verteilter Objektsysteme – Eine Einführung. In: HMD 186/1995 S. 10-30

[16] Sinz E.J.: Ansätze zur fachlichen Modellierung betrieblicher Informationssysteme. Entwicklung, aktueller Stand und Trends. In: Heilmann, H., L.J. Heinrich u. F. Roithmayr (Hrsg.): Information Engineering. Oldenbourg, München 1996, S. 123-143

[17] Verbundprojekt WEGA - Wiederverwendbare und erweiterbare Geschäftsprozeß- und

Anwendungssystem-Architekturen. In: Grote U., Wolf G. (Hrsg.): Statusseminar des BMBF Softwaretechnologie. Berlin, DLR e.V., 1996