Fraunhofer Institut
Software- und
Systemtechnik

Projekt
„Kontinuierliches Engineering für Evolutionäre Infrastrukturen"[1]

Das Vorgehensmodell nach Cheesman und Daniels für komponentenbasierte Softwaresysteme

Alexander Bilke, Jens Fleischer, Olaf Klischat

Bericht (Version 0.1.1)
Juli 2001

1. Gefördert vom Bundesministerium für Bildung und Forschung (BMBF) unter dem Förderkennzeichen 01 IS 901

1 Einleitung

Das Ziel dieser Arbeit besteht darin, einen zusammenfassenden Überblick über das *Vorgehensmodell* von Cheesman und Daniels ([Chees00]) zu geben. Das Vorgehensmodell liefert ein pragmatisches Verfahren (»A Simple Process«), um komponentenbasierte Software-Systeme zu entwerfen. Ausgehend von einem komponenten-orientierten Ansatz werden Abstraktionen und Techniken vorgestellt, die server-seitig die Entwicklung einer kohärenten Komponentenarchitektur unterstützen. Das dabei verwendete Beispiel stammt ebenfalls aus [Chees00].

Das Vorgehensmodell beschränkt sich auf den konstruktiven Kern, dem *Entwicklungsprozeß* von Komponenten-Systemen. Die administrative Begleitung von Software-Projekten wird dagegen ausgeklammert.

Der Entwicklungsprozeß ist umfassend und wird durch verschiedenen Entwicklungsstadien (sog. *Workflows*) beschrieben. Den Schwerpunkt bildet die *Komponentenspezifikation*, die in Form der drei Phasen: *Komponentenidentifikation* (Kapitel 5) *Komponenteninteraktion* (Kapitel 6) und *Komponentenspezifikation* (Kapitel 7) genauer betrachtet wird. Angrenzende Phasen wie die Anforderungsanalyse zu Beginn oder die Auslieferung am Ende werden nur gestreift.

Obwohl die Modellierungstechnik UML ([Booch99]) im Vorgehensmodell von Cheesman und Daniels eine besondere Stellung einnimmt, wird auf die Darstellung von UML und deren spezifischen Erweiterungen durch Cheesman und Daniels weitgehend verzichtet. Auch die begleitenden Beispiele werden, bis auf wenige Ausnahmen, nicht übernommen.

2 Komponentensystem

In diesem Kapitel werden die wesentlichen Grundzüge eines komponentenbasierten Software-Systems vorgestellt und die Konzepte und Terminologie von Cheesman und Daniels eingeführt.

2.1 Komponenten

Komponenten folgen dem Prinzip der Komplexitätsbewältigung durch Aufteilung des Ganzen in überschaubare Teilprobleme (»divide and conquer«). Cheesman und Daniels betonen, daß das Ziel ihrer Arbeit die *Veränderbarkeit* eines Softwaresystems ist, nicht jedoch die *Wiederverwendbarkeit* von Teilen des Systems. Zwar ist die Wiederverwendung von Komponenten anzustreben, muß aber mangels allgemein akzeptierter Konzepte zur Zeit als „schwierig" angesehen werden.

Die wichtigsten Eigenschaften einer Komponenten (aus technischer Sicht) stammen bereits aus dem objekt-orientierten Paradigma: i) Die *Kollokation von Daten und Funktionen* (gemeinsame Schnittstelle), ii) das *Kapseln* der Implementation und iii) der *Identität* einer Komponente.

Cheesman und Daniels umgehen die direkte Begriffsdefinition. Stattdessen charakterisieren Sie eine Komponenten durch die verschiedenen Erscheinungsformen, die sie einnehmen können:

- *Komponentenstandard*: Jede Komponente muß einen gewissen (Umgebungs-) Standard befolgen.
- *Komponentenspezifikation*: Zu jeder Komponente gehört die Beschreibung von dem was sie leistet.
- *Komponentenschnittstelle*: Zu jeder Komponente gehört die Beschreibung ihrer Schnittstelle.
- *Komponentenimplementation*: Die strikte Trennung von Schnittstelle, Spezifikation und Implementation führt zur separaten Beschreibung der Implementation einer Komponenten.
- *Installierte Komponente*: Die Beschreibung einer installierten Komponente umfaßt die Anbindung an das Zielsystem.
- *Komponentenobjekt*: Ein Komponentenobjekt ist eine installierte Komponente zur Ausführungszeit, die sich in ihrem Zustand von anderen installierten Komponenten (gleichen Typs) unterscheidet.

2.2 Komponenten- und Systemarchitekturen

Zur Klärung der Begriffe definieren Cheesman und Daniels die Systemarchitektur als:

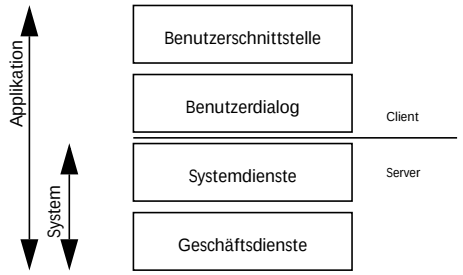
> »die Struktur der Teile, die eine Software-Installation bilden, zuzüglich ihrer gegenseitigen Beziehungen und Verpflichtungen, möglicherweise auch der zugrundeliegenden Technologien«,

und die Komponentenarchitektur als:

> »eine Menge von Software-Komponenten auf Applikationsebene, ihren strukturellen Beziehungen und ihre Abhängigkeiten bzgl. ihres Verhaltens«.

Komponenten können prinzipiell auf allen Ebenen einer Applikation angesiedelt werden. Aus pragmatischer Relevanz wird die Betrachtung auf Software-Systeme eingeschränkt, die wie in Bild 1 aus vier Schichten bestehen.

Bild 1 Die vier Architekturebenen der von Cheesman und Daniels betrachteten Applikationen)



Die *Benutzerschnittstelle* ist die Schnittstelle für den Nutzer der Applikation (menschlicher Nutzer oder Software). Darunter folgt der *Benutzerdialog* zur Durchführung einer Nutzersitzung. Auf Systemebene werden die Ebenen der System- und Geschäftsdienste unterschieden. Die *Systemdienste* repräsentieren das System nach außen hin und bieten die Funktionalität des Anwendungskontextes (Problemdomäne). Die Ebene der *Geschäftsdienste* ist der Kern der Applikation, der das Geschäftsmodell, die Geschäftsregeln und die Transformationen des Systems implementiert.

Komponentenarchitekturen

Cheesman und Daniels betrachten nur die Komponentenbildung auf der Ebene der System- und Geschäftsdienste. Die Zuordnung der Komponenten bilden eine Komponentenarchitektur, die von verschiedenen Standpunkten aus betrachtet werden kann:

- Die *Komponentenspezifikation-Architektur* beschreibt die Schnittstellen und die Abhängigkeiten jeder einzelne Komponente vollständig und explizit.
- Die *Komponentenimplementation-Architektur* beschreibt die tatsächlichen Abhängigkeiten der verschiedenen Komponentenimplementationen. Diese Sicht ist die Vereinigung der Komponentenspezifikation und den zusätzlichen Abhängigkeiten der Implementation.
- Die *Komponentenobjekt-Architektur* beschreibt die Abhängigkeiten der verschiedenen Instanzen der Komponentenobjekte untereinander, die eine lauffähige Applikation bilden.

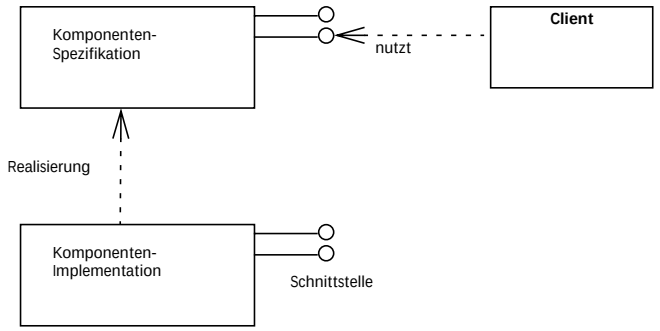
2.3 »Design by Contract«

Komponenten müssen sie ihre Eigenschaften klar definieren; sie gehen einen Vertrag (contract) mit ihren Anwendern ein. Zwei verschiedene Arten von Verträgen werden unterschieden (s. Bild 2):

- dem *Nutzungsvertrag*, der den Vertrag zwischen der Schnittstelle einer Komponentespezifikation und dem Nutzer festlegt, und
- dem *Realisierungsvertrag*, der den Vertrag zwischen der Komponentenspezifikation und der Realisierung festlegt.

Durch diese Unterscheidung wird die Verwendung einer Komponente von ihrer Implementierung getrennt.

Bild 2 Die verschiedenen Vertragstypen)



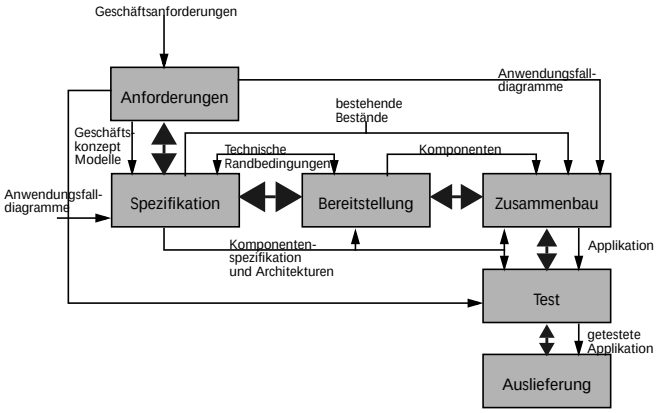
3 Der Entwicklungsprozeß

Jedes Vorgehensmodell läßt sich in zwei disjunkte Prozesse aufteilen, dem *Managementprozeß* und dem *Entwicklungsprozeß*. Der Managementprozeß (der nicht teil der Arbeit von [Chees00] ist) teilt Arbeitspakete auf, legt Zeitrahmen fest und begleitet den Entwicklungsprozeß der Softwareprodukte. Der Entwicklungsprozeß ist dagegen Schwerpunkt dieser Arbeit und wird im folgenden vorgestellt.

3.1 Workflows

Der Entwicklungsprozeß von Cheesman und Daniels besteht aus mehreren sogenannten *Workflows*[1]. Workflows werden hier als größere Arbeitspakete verstanden. Bild 3 skizziert die verschiedenen Workflows des Entwicklungsprozeß. Die großen Pfeile kennzeichnen den Hauptstrang der aufeinanderfolgenden Workflows. Die kleinen Pfeile markieren die erforderlichen/gewonnenen *Artefakte* (Dokumente, Schemata) der Workflows.

Bild 3 Die Workflows des vollständigen Entwicklungsprozeß



Beginnend mit der *Anforderungsanalyse* (Kapitel 4) werden die Geschäftsmodellkonzepte entworfen. Daran schließen sich der wichtige *Spezifikation*-Workflow an, der ausführlich in den Kapiteln 5, 6, und 7 behandelt wird. Zum

1 Diese begriffliche Vorstellung von 'Workflow' stammt von Kruchten und wird innerhalb von RUP definiert [Kruc99].

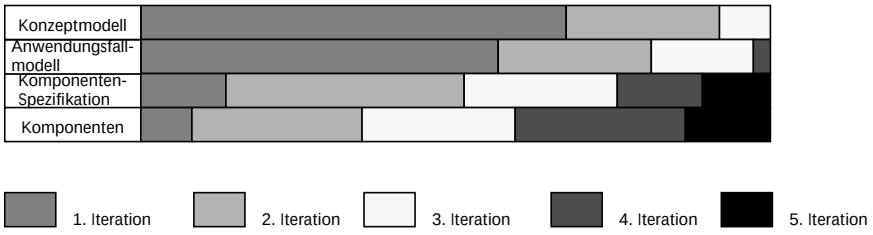
Schluß dieser Arbeit wird noch auf die *Bereitstellung* und dem *Zusammenbau* in Kapitel 8 eingegangen. Die abschließenden Workflows *Test* und *Auslieferung* sind bei Cheesman und Daniels untergeordnet und bleiben in dieser Arbeit unberücksichtigt.

3.2 Evolutionäre Entwicklung

Der Entwicklungsprozeß in Bild 3 suggeriert ein lineares Aufeinanderfolgen der Workflows, lediglich die Rückwärtskanten zwischen den Workflows legen ein einfachen Rücksprung nahe. Da Cheesman und Daniels in ihrer Arbeit den Managementprozeß unberücksichtigt lassen, weisen sie darauf hin, daß ihr Entwicklungsprozeß hinsichtlich des Managementprozeß neutral ist, d.h. in jeder Art von Managementprozeß eingebettet werden kann. Sie schlagen aber einen evolutionären Managementprozeß vor, der durch sein iteratives und inkrementelles Vorgehen in jedem Zyklus eine Verfeinerung der Artefakte der verschiedenen Workflows erzeugt.

Der evolutionäre Ansatz schließt in jedem Zyklus Aktivitäten in jedem Workflow ein. Die Entwicklung der gewonnenen Artefakte hinsichtlich der verschiedenen Phasen skizziert Bild 4. Auf die Artefakte wird an dieser Stelle nicht besonders eingegangen, sondern auf die entsprechenden Kapitel des Spezifikation-Workflow verwiesen.

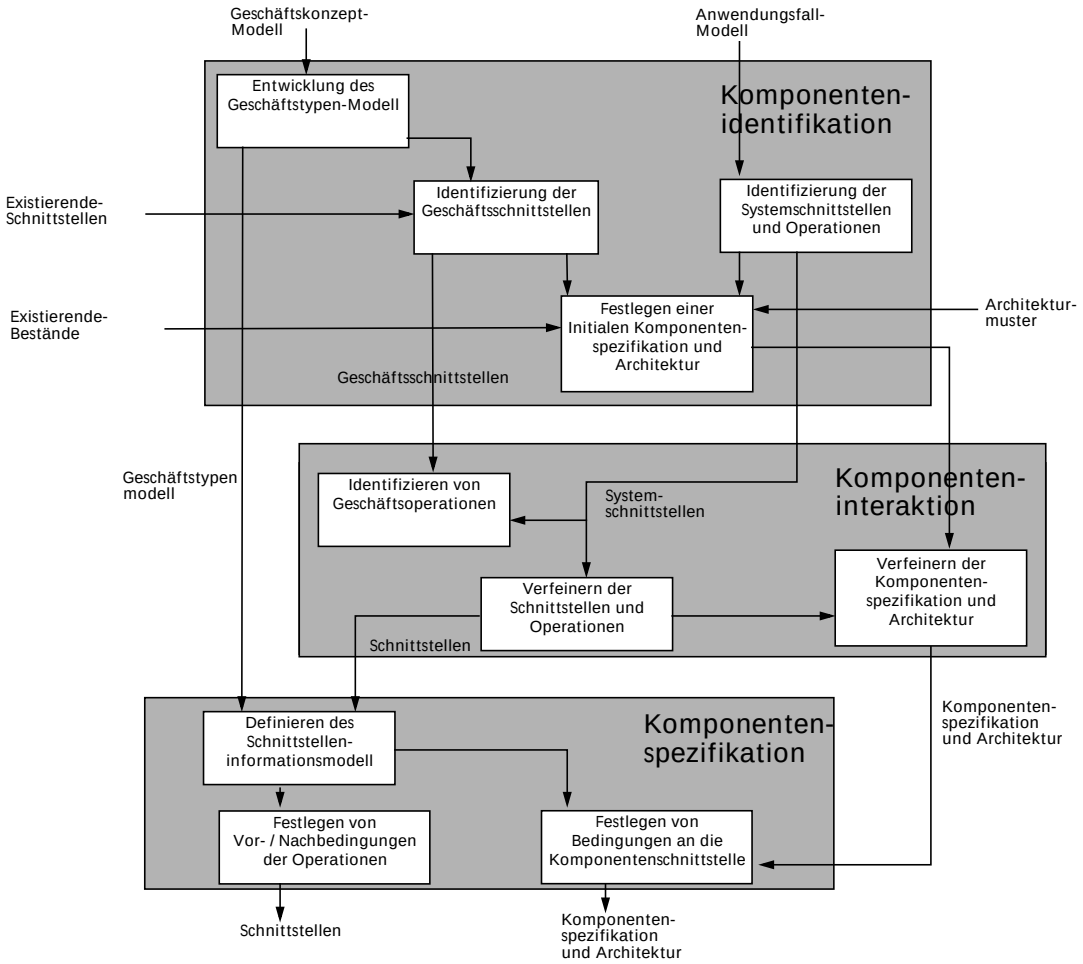
Bild 4 Evolution der Artefakte



3.3 Spezifikation-Workflow

Der für Cheesman und Daniels wichtigste Teil des Entwicklungsprozesse ist der Spezifikation-Workflow, der sich in frei Phasen aufteilt (s. Bild 5).

Bild 5 Die drei Phasen des Spezifikation-Workflow



Komponentenidentifikation

Die *Komponentenidentifikation* (s. Kapitel 5) erhält als Vorgabe das Geschäftskonzeptmodell und das Anwendungsfalldiagramm aus dem Workflow der Anforderungsanalyse. Es werden initiale Geschäfts- und Systemschnittstellen, sowie eine initiale Komponentenarchitektur identifiziert. Als Nebenprodukt dieser Phase wird das Geschäftstypenmodell später zur Komponentenspezifikation verwendet.

Komponenteninteraktion

In der Phase der *Komponenteninteraktion* (s. Kapitel 6) werden mit Hilfe von Interaktionsdiagrammen die Systemschnittstellen verfeinert und die Operationen der Geschäftsschicht ermittelt. Zusammenfügen und Ausklammern von

Schnittstellen lassen am Ende dieser Phase das gesamte System entstehen, mit einer klaren Vorstellung der Beziehungen der Komponenten untereinander, bis hinunter auf Operationenebene.

Komponentenspezifikation

In der *Komponentenspezifikation* (s. Kapitel 7), der letzten Phase des Spezifikation-Workflow, werden Details der Modellierung geklärt. Die Architektur wird dabei als feststehend angesehen. Aus der Analyse der Zustände von Komponentenobjekten wird das Schnittstelleninformationsmodell gewonnen. Ferner werden Vor- und Nachbedingungen der Operationen festgelegt und Randbedingungen des Geschäftsmodell berücksichtigt.

4 Anforderungsanalyse

Gleich am Anfang eine Warnung: der vorliegende Text soll keine Anleitung zur Anforderungsanalyse sein. Dieses Thema ist zu umfangreich, um in diesem Rahmen erläutert zu werden. Das hier beschriebene Vorgehen liefert die Modelle, die für den weiteren Verlauf zwingend sind, und stellen somit einen minimalen Aufwand dar. Ist das für das jeweilige Projekt nicht ausreichend, können weitere Schritte eingeführt werden.

4.1 Der Geschäftsprozeß

Der Geschäftsablauf sollte bekannt und von allen verstanden sein. Wie der Prozeß analysiert wurde, ist für dieses Vorgehensmodell irrelevant und deshalb nicht näher beschrieben. Als Notation bietet sich das UML-Aktivitätsdiagramm an, es sind aber auch andere Diagrammtypen möglich.

4.2 Das Geschäftskonzeptmodell

Die Beschreibung des zu unterstützenden Geschäftsprozesses führt neue Begriffe ein, die verstanden werden müssen. Zu diesem Zweck sollte eine Art Mind Map, das sogenannte Geschäftskonzeptmodell, angefertigt werden. Dieses Modell kann durch ein UML-Klassendiagramm niedergeschrieben werden, obwohl es sich natürlich nicht um Klassen handelt.

Beispiele für in einem Geschäftsprozeß auftretende Konzepte wären Kunde, Adresse, Reservierung usw. Zu diesem Zeitpunkt muß man sich noch keine Gedanken darüber machen, ob diese Konzepte später eine Rolle spielen - sie sollten erst einmal notiert werden. Details, wie z.B. Attribute, sollten auch noch nicht betrachtet werden. Wichtig sind allerdings die Beziehungen zwischen den »Klassen« und deren Kardinalitäten. So hat bspw. eine Kunde eine Postadresse und mehrere Reservierungen.

4.3 Definition der Systemgrenzen

Eine wichtige Aufgabe der Anforderungsanalyse ist die Bestimmung der Systemgrenzen. Dabei wird festgestellt, für welche Funktionen das System zuständig ist. Aus Sicht des Nutzers sollte man sich fragen, wie das System arbeitet. Eine natürlichsprachliche Darstellung ist zu diesem Zweck ausreichend.

4.4 Anwendungsfälle

In einem vorbereitenden Schritt sollten die Verantwortlichkeiten des Geschäftsprozesses geklärt werden. Das bedeutet, daß Geschäftsabläufe bestimmten Personengruppen oder dem Softwaresystem zugeordnet werden. Wird ein Arbeitsschritt dem System zugeordnet bedeutet dies, daß es diesem Vorgang unabhängig und ohne Hilfe von außen abarbeitet.

Alle anderen Schritte liegen im Verantwortungsbereich von *Akteuren*, also Personen oder externen Systemen. Das (interne) System nimmt natürlich auch an diesen Schritten teil, der Akteur initiiert und kontrolliert aber den Vorgang.

Anwendungsfälle (engl. use cases) spielen eine wichtige Rolle bei der Definition der Systemgrenzen. Sie erlauben es uns außerdem zu zeigen, wie die Software die an sie gestellten Anforderungen erfüllen will. Sie sind im wesentlichen eine Art funktionelle Spezifikation des Systems. Dabei wird das System als eine Art »black box« in Bezug auf ihre innere Struktur behandelt.

Die identifizierten Anwendungsfälle werden in einem oder mehreren Anwendungsfalldiagrammen zusammengefaßt. Zusätzlich muß jeder Anwendungsfall näher beschrieben werden. Zu diesem Zweck sollte man sich darüber Gedanken machen, wie ein Akteur mit dem System interagiert. Es können mehrere Szenarien entwickelt werden, wobei mindestens ein erfolgreicher Durchlauf dokumentiert sein muß. Szenarien mit auftretenden Fehler sollten auch betrachtet werden. Die Notation von Alistair Cockburn [CockburnWeb] hat sich in diesem Zusammenhang bewährt. Das Hauptszenario wird in mehrere Anwendungsfallschritte (engl. use case steps) unterteilt. Kann in einem dieser Schritte ein Fehler auftreten, wird zu einer Erweiterung (engl extension) verzweigt. Dadurch können sehr übersichtlich mehrere Szenarien und ihre Zusammenhänge dargestellt werden.

4.5 Zusammenfassung

Eine Anforderungsanalyse kann niemals vollständig sein. Das sollte man immer im Hinterkopf behalten. Anforderungen ändern sich im Verlauf des Projektes, und darauf sollte man vorbereitet sein.

Wichtig für das weitere Vorgehen ist die Erstellung des Geschäftskonzeptmodells und Anwendungsfallmodells, welche die Grundlage für die Definition des Geschäftstypenmodells bzw. der Systemschnittstellen in der »Komponentenidentifizierung« sind.

5 Komponentenidentifikation

Ziel der Komponentenidentifikation ist die Erstellung einer Grundmenge von Schnittstellen- und Komponentenspezifikationen. Die Basis dafür sind die in der Anforderungsanalyse gewonnenen Geschäftskonzeptmodelle und Anwendungsfälle. Diese werden in den folgenden Schritten benutzt, um System- und Geschäftsschnittstellen zu erkennen. Außerdem wird das für die dritte Phase »Komponentenspezifikation« wichtige Geschäftstypenmodell erzeugt. Der Schwerpunkt liegt in der Entdeckung der verwalteten Informationen, welche Schnittstellen dafür benötigt werden, welche Komponenten diese Funktionalität bereitsstellen und wie sie zusammenarbeiten. Dabei wird, wie oben beschrieben, zwischen der Systemschicht und der Geschäftsschicht unterschieden, was sich auch im Arbeitsablauf widerspiegelt.

5.1 Identifizierung der Systemschnittstellen

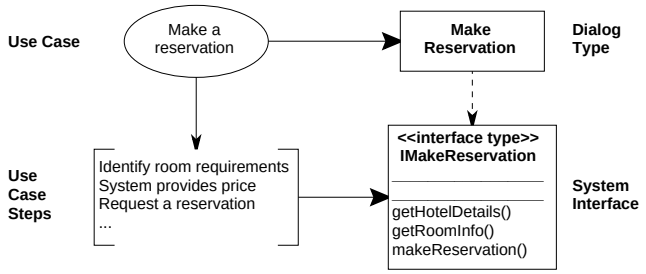
Mit Hilfe des Use Case - Modells können erste Systemschnittstellen erzeugt werden. Für jeden Anwendungsfall wird eine Systemschnittstelle und ein Dialogtyp erzeugt. Bei der Betrachtung der einzelnen Schritte innerhalb eines Anwendungsfalls muß erkannt werden, wann eine Aktion des Systems ausgeführt wird. Diese Schritte müssen durch je eine oder mehrere Methoden in der Schnittstelle dargestellt werden, auf deren Basis das Modell weiterbearbeitet werden kann. Wird in mehreren aufeinanderfolgenden Anwendungsschritten das System angesprochen, können die Aktionen durch eine einzige Methode realisiert werden.

Die in dieser Phase gewonnenen Schnittstellen sind nicht als fix zu betrachten. In der nächsten Phase »Komponenteninteraktion« wird festgelegt, welche Schnittstellen für welche Operationen verantwortlich sind, was zu einer Verschiebung der Schnittstellenmethoden führen kann.

5.1.1 Umsetzung

Der Anwendungsfall »Make a reservation« wurde in der Anforderungsanalyse in mehrere Schritte unterteilt. Für eine bestimmte Menge dieser Schritte ist das System zuständig. So muß z.B. für den Schritt »Identify room requirements« das System Informationen über Hotels und deren Räume zur Verfügung stellen. Dies wird durch Aufruf der Operation getHotelDetails() geschehen.

Bild 6 Übersetzung in Systemschnittstellen (Quelle: [Chees00])



Zur Unterscheidung zwischen den hier erstellten konzeptionellen Schnittstellentypen (interface type) und Schnittstellen in UML-Klassenmodellen wurde der Stereotyp <<interface type>> eingeführt. Schnittstellentypen sind durch Schnittstelleninformationsmodelle (interface information models) gekennzeichnet, die den Zustand der Schnittstelle modellieren. Diese Informationsmodelle werden in späteren Phasen des Vorgehensmodells erzeugt.

Die Operationen der Schnittstellentypen werden in dieser Phase nur benannt. Durch welche Parameter und Rückgabewerte sie gekennzeichnet sind, entscheidet sich nach Betrachtung der Komponenteninteraktionen in der nächsten Phase.

Die Wiederverwendbarkeit der hier erstellten Systemschnittstellen ist nicht hoch, da sie sehr systemspezifisch sind. Anders sieht das bei den Geschäftsschnittstellen aus, die möglichst systemunabhängig sein sollen.

5.2 Identifizierung der Geschäftsschnittstellen

Geschäftsschnittstellen stellen Abstraktionen über die vom System verwalteten Informationen dar. Sie werden in folgenden Schritten identifiziert:

1 Erzeugung des Geschäftstypenmodells (business type model),

2 Verfeinerung des Geschäftstypenmodells und Spezifizierung von Regeln,

3 Identifizierung der Kerntypen (core types),

4 Erschaffung von Geschäftsschnittstellen für diese Kerntypen,

5 Verfeinerung des Modells.

5.2.1 Erzeugung des Geschäftstypenmodells

Der erste Schritt zur Identifizierung der Geschäftsschnittstellen ist die Überführung des Geschäftskonzeptmodells in das Geschäftstypenmodell. Während das Konzeptmodell nur eine Übersicht aller interessanten Informationen darstellt, beinhaltet das Geschäftstypenmodell die spezifischen Informationen, die vom System verwaltet werden müssen.

Ein Geschäftstyp definiert ein Datum oder Zustand, welcher vom System beibehalten und überwacht wird und von den Benutzern als ein Geschäftsobjekt oder -prozeß verstanden werden muß. Es kann sich dabei sowohl um physische Artefakte (Kunde, Produkt, usw.) als auch nichtphysische Aspekte des Geschäfts (z.B. Bestellung) handeln.

Das Geschäftstypenmodell wird gebildet, indem Typen aus dem Geschäftskonzeptmodell entfernt (z.B. weil sie für die Anwendung irrelevant sind) und neue Typen hinzugefügt werden.

5.2.2 Verfeinerung des Modells

Sind alle geschäftsrelevanten Typen identifiziert worden, können sie mit Attributen angereichert werden. Diese Attribute haben einen festgelegten Typ, der entweder allgemeiner Natur ist (z.B. String) oder expliziert definiert wurde, was in folgenden Diagrammen durch den Stereotyp <<datatype>> kenntlich gemacht wird. Es kann sich auch um parametrisierte Attribute handeln. Das sind Attribute, deren Wert vom Parameterwert abhängig ist.

Des weiteren müssen Beziehungen zwischen Geschäftstypen identifiziert und als Assoziationen im Modell dargestellt werden. Zu diesem Zeitpunkt sollte Navigierbarkeit noch keine Rolle spielen, d.h. alle Assoziationen sind symmetrisch. Auch Randbedingungen, z.B. die Kardinalität der Assoziationen, sollten in diesem Schritt festgelegt werden.

Basierend auf dem so erstellten Modell werden Geschäftregeln in Form von Constraints aufgestellt. Zu diesem frühen Abschnitt des Spezifikationsprozesses genügt die natürlich-sprachliche Formulierung der Regeln. In der letzten Phase, wenn das Modell relativ stabil ist, können diese Regeln auch formal (z.B. in der Object Constraint Language OCL) niedergeschrieben werden.

So können bspw. gewisse Assoziationen aus anderen Beziehungen abgeleitet werden. Dies sollte entsprechend gekennzeichnet werden. Weitere Regeln können identifiziert und auch durch zusätzliche, möglicherweise parametrisierte Attribute dargestellt werden.

5.2.3 Identifizierung von Kerntypen

Jetzt müssen die Kerntypen des Modells identifiziert werden. Dabei muß bedacht werden, welche Informationen von anderen Informationen abhängen, und welche Informationen losgelöst existieren könnten. Das ist ein nützlicher Schritt in Hinblick auf die Zuweisung von Verantwortlichkeiten an Schnittstellen.

Kerntypen sind charakterisiert durch

1 eine Geschäftsbezeichnung, die normalerweise unabhängig von anderen Bezeichnungen ist, und

2 die unabhängige Existenz, d.h. es gibt keine zwingend notwendigen Beziehungen, außer zu einem kategorisierenden Typ.

Kategorisierende Typen dienen der Klassifizierung von Kerntypen. So könnte z.B. der Typ »Raum« eine Assoziation zum kategorisierenden Typ »Raumtyp« haben. Hat »Raum« keine weiteren verbindlichen Assoziationen (z.B. zu einem Typ »Hotel«), kann er als Kerntyp betrachtet werden.

Werden im Spezifikationsprozeß UML-Klassenmodelle benutzt, sollten Kerntypen durch den Stereotyp <<core>> gekennzeichnet werden.

5.2.4 Erschaffung von Geschäftsschnittstellen für diese Kerntypen

Generell gilt die Regel: eine Geschäftsschnittstelle für jeden Kerntyp. Jede dieser Schnittstellen kontrolliert die Daten, die durch den Kerntyp und seine assoziierten Typen dargestellt werden. Wenn diese Schnittstellen eine Menge von Instanzen des Kerntyps verwalten, werden sie auch als Verwaltungsschnittstellen (manager interface) bezeichnet. Zu diesem Zeitpunkt sollte allerdings noch nicht entschieden werden, ob eine Schnittstelleninstanz nur eine Instanz des Kerntyps oder eine Menge von Instanzen bearbeiten soll. Bis diese Entscheidung getroffen wird, kann man mit Verwaltungsschnittstellen weiterarbeiten.

Nachdem die Schnittstellen dem Modell hinzugefügt worden sind, kann man von einem Schnittstellenverantwortlichkeitsdiagramm (interface responsibility diagram) sprechen. Der Zweck dieses Diagramms ist zu verdeutlichen, welche Schnittstellen für welche Daten verantwortlich sind, und über Abhängigkeiten nachzudenken. Dafür werden assoziierte Typen den dazugehörigen Schnittstellen zugeordnet, wobei jeder dieser Typen nur zu einer Schnittstelle gehören kann. Das wird schwierig, wenn ein Typ mit anderen Typen assoziiert ist, die verschiedenen Schnittstellen angehören. Abhängig vom konkreten Fall muß man sich für eine Schnittstelle entscheiden, der dieser Typ angehören soll.

Beziehungen des betrachteten Typs zu anderen Typen sollten in asymmetrische, vom betrachteten Typ ausgehende Assoziationen umgewandelt werden.

5.2.5 Verfeinerung des Modells

Neben den Zuständigkeiten muß geklärt werden, welche Schnittstellen welche Daten speichern. Ein Problem sind dabei Abhängigkeiten zwischen verschiedenen Schnittstellen. Wenn Assoziationen zwischen Typen unterschiedlicher Schnittstellen (Inter-Interface Associations) existieren, treten solche Abhängigkeiten auf. Ein Beispiel wäre die Beziehung zwischen Bestellung und Kunde: eine Bestellung enthält auch immer eine Kundennummer.

Ein wichtiges Ziel ist die Reduzierung solcher Abhängigkeiten. Solche Beziehungen zwischen Assoziation bzw. deren Typen sollten asymmetrisch sein. Dadurch erkennt man sofort, wer für die Speicherung der Referenz zu sorgen hat.

Das in diesen 5 Schritten erstellte Geschäftstypenmodell wird in der Phase »Komponentenspezifikation« verwendet. Die gewonnenen Geschäftsschnittstellen werden separat weiterbearbeitet.

5.3 Einbinden existierender Schnittstellen

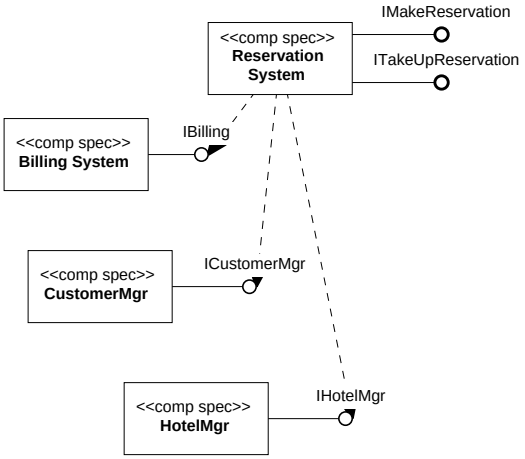
Die in Abschnitt 5.1 entwickelten Systemschnittstellen und die Geschäftsschnittstellen aus Abschnitt 5.2 bilden die Basis der Schnittstellenspezifikation. Dies muß aber auch Schnittstellen bereits existierender und weiterverwendeter Komponenten enthalten. Deren Spezifikationen sind als fix anzusehen und können weiter benutzt werden.

5.4 Erste Komponentenspezifikation

Eine Komponente ist eine Funktionseinheit, die als Ganzes eingesetzt, ersetzt oder wiederwendet werden kann. Sie sollten dementsprechend strukturiert sein. In vielen Fällen wird für jede Schnittstelle eine separate Komponenten spezifikation erstellt. Allerdings können Komponenten auch mehrere Schnittstellen unterstützen. Gründe hierfür wären gleiche oder ähnliche Lebenszyklen der Schnittstellen oder komplizierte Interaktionen zwischen den Schnittstellen. Wenn bestimmte Schnittstellen nur zusammen ausgetauscht werden, oder für eine einzelne Komponente zu feingranular sind, sollten sie in einer Komponente vereinigt werden.

Wie in Bild 7 zu sehen ist, werden auch Abhängigkeitsbeziehungen eingezeichnet. Ob diese Beziehungen so tatsächlich existieren, wird sich in der nächsten Phase herausstellen.

Bild 7 Erste Komponentenspezifikation (Quelle: [Chees00])



6 Komponenteninteraktion

Beim Spezifizieren der Komponenteninteraktion wird festgelegt, wie die während der Komponentenspezifizierung gefundenen Komponenten interagieren sollen, um die gewünschte Funktionalität des Systems zu erreichen. Dabei wird unter Zuhilfenahme der Use Cases festgelegt, wie und in welcher Reihenfolge die Operationen der Systemschnittstellen Operationen anderer Schnittstellen (System- oder Geschäftsschnittstellen) aufrufen müssen, um die im jeweiligen Use Case geforderte Funktonalität zu realisieren. Dabei werden sozusagen »nebenbei« auch die Operationen der Geschäftsschnittstellen (Geschäftsoperationen) sowie die genauen Signaturen aller Operationen festgelegt. Die Darstellung der so ermittelten Komponenteninteraktionen in UML erfolgt zweckmäßigerweise in Form von Kollaborationsdiagrammen.

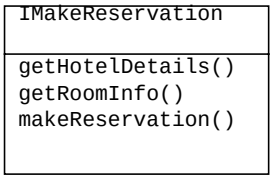
6.1 Ermittlung der Geschäftsoperationen

Im diesem Schritt werden die genauen Signaturen der Operationen der System- und Geschäftsschnittstellen sowie die Interaktionen zwischen den Komponenten beim Ausführen der einzelnen Systemoperationen ermittelt. Dabei wird einfach für jeden Anwendungsfall festgelegt, in welcher Reihenfolge welche Operationen auf der zum Anwendungsfall gehörenden Systemschnittstelle ausgeführt werden müssen. Für jede dieser Operationen wird darüberhinaus spezifiziert, welche anderen Geschäfts- und/oder Systemoperationen diese Operation in welcher Reihenfolge auszuführen hat und welche Daten dabei jeweils ausgetauscht werden. Letzteres führt direkt zu einer genauen Spezifikation der Signaturen den Operationen.

Zur Veranschaulichung des Vorgangs soll das Beispiel aus Kapitel 5 fortgeführt werden. Es sollen nun die Komponenteninteraktionen für die Systemschnittstelle »IMakeReservation« ermittelt werden.

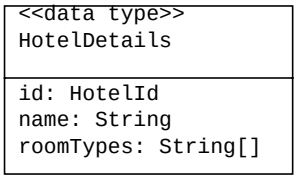
Der zugehörige Use-Case sehe vor, daß der Akteur (in diesem Fall ein Kunde, der eine Reservierung tätigen will) zunächst Informationen über zur Verfügung stehende Hotels und Zimmertypen einholt und anschließend seine Reservierung für einen bestimmten Zimmertyp in einem bestimmten Hotel ausführt. Das System vermerkt daraufhin die Reservierung und benachrichtigt den Kunden per E-Mail.

Aus der Analyse dieses Ablaufs sei während der Komponentenidentifizierung bereits folgendes Schema für die Operationen auf IMakeReservation gewonnen worden:



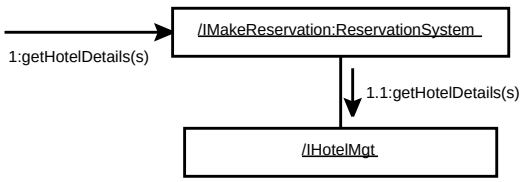
Der Akteur findet zunächst mit Hilfe der Operationen getHotelDetails() und getRoomInfo() Informationen über zur Auswahl stehende Hotels und deren Räume. Im Fall von getHotelDetails() sieht die hier gewählte Lösung vor, daß der Aufrufer einen Suchstring für den Hotelnamen übergibt und eine Liste von Hotelinformationen in Form von »HotelDetails«-Datentypen erhält:

```
IMakeReservation::getHotelDetails(in match: String): HotelDetails[]
```



Das Komponentenobjekt vom Typ ReservationSystem, das IMakeReservation bereitstellt und auf dem getHotelDetails() aufgerufen wird, enthält selber keine Hoteldaten. Es leitet die Operation deshalb an die bereits identifizierte Komponente HotelMgr in der Geschäftsschicht weiter. Die entsprechende Schnittstelle IHotelMgt bekommt deshalb ebenfalls eine Operation getHotelDetails() mit exakt derselben Signatur. Die komplette Interaktion ist in Bild 8 zu sehen.

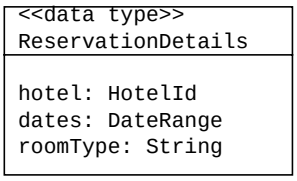
Bild 8 Interaktionen der Systemoperatioin »getHotelDetails()«



Auf ähnliche Weise kann man die Interaktionen für die Operation IMakeReservation::getRoomInfo() identifizieren: Der Aufrufer übergibt Details seiner Reser-

vierung (Hotel, Raumtyp, Zeitraum) und bekommt Verfügbarkeit und Preis mitgeteilt.

```
IMakeReservation::getRoomInfo()
         in res: ReservationDetails,
         out availability: Boolean,
         out price: Currency)
```



Auch diese Operation wird identisch an die Geschäftsschnittstelle IHotelMgt der Komponente HotelMgr weitergeleitet.

Den Abschluß der Reservierung bildet die Operation IMakeReservation::makeReservation(). Sie bekommt Informationen über die zu tätigende Reservierung und den Kunden, führt die Reservierung durch (IHotelMgt::makeReservation()), benachrichtigt den Kunden (ICustomerMgt::notifyCustomer()) und liefert eine von IHotelMgt::makeReservation() gelieferte eindeutige ID für die Reservierung zurück. Die identifizierten Operationen und Interaktionen sehen wie folgt aus:

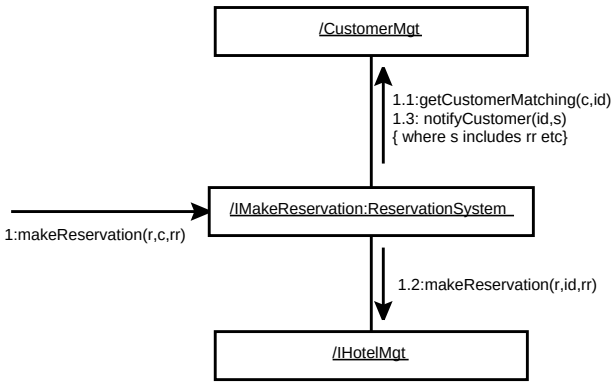
```
IMakeReservation::makeReservation
        (in res: ReservationDetails,
        in cus: CustomerDetails,
        out ResRef: String):String

ICustomerMgt::getCustomerMatching
        (in cus: CustomerDetails,
         out cusId: CustId): Integer

ICustomerMgt::notifyCustomer(in cusId: CustId, in mail: String)

IHotelMgt::makeReservation
        (in res: ReservationDetails.
         in cus: CustId,
         out resRef: String): Boolean
```

Bild 9 Interaktionen der Systemoperatioin »makeReservation()«



Hervorzuheben ist, daß der von außen übergebene CustomerDetails - Parameter zunächst mit Hilfe der Operation ICustomerMgt::getCustomerMatching() in eine Kunden-ID (CustId) umgewandelt wird. Diese (und nicht der CustomerDetails - Parameter) wird dann an IHotelMgr::makeReservation() übergeben. Dadurch wird erreicht, daß die Komponenten HotelMgr und CustomerMgr sich nicht gegenseitig referenzieren müssen und unabhängig voneinander verwendbar werden.

Bild 9 umfaßt nicht alle möglichen Interaktionen innerhalb von makeReservation(). Es ist bspw. möglich, daß der übergebene Kunde noch nicht existiert -- in diesem Fall muß er neu angelegt werden, was eine weitere Operation in ICustomerMgt erfordert. Dieser Aspekt wird hier nicht näher betrachtet.

6.2 Referentielle Integrität verwalten

Ein wichtiger Aspekt bei der Spezifikation der Komponenteninteraktion besteht in der korrekten und konsistenten Verwaltung von Referenzen zwischen Kom-

ponentenobjekten (Inter-Komponent-Referenzen). Es muß festgelegt werden, welche Kardinalitäten die Referenzen jeweils haben und ob und wann sie geändert werden. Ein Assoziationspfeil kann in UML mit dem Modifizierer {frozen} annotiert werden, um auszudrücken, daß die entsprechende Assoziation während der gesamten Laufzeit des Systems konstant und unveränderlich bleibt.

Weiterhin ist es wichtig zu klären, wie bei Inter-Komponent-Referenzen die Konsistenz sichergestellt werden kann. Diese Frage stellt sich insbesondere immer dann, wenn eine Komponente gelöscht werden soll, auf die noch von anderen Komponenten aus verwiesen wird. So verwaltet im Beispiel oben die Komponente CustomerMgr eine Liste von Kunden, und die Komponente HotelMgr verwaltet Assoziationen zwischen Kunden und Reservierungen (die Kunden werden hier durch eindeutige IDs referenziert). Wenn man in einer CustomerMgr-Instanz einen Kunden löscht, muß die HotelMgr-Instanz dies irgendwie »mitbekommen«. Ein pragmatischer Lösungsansatz sieht vor, daß eine geeignete übergeordnete Komponente die Konsistenz sicherstellt (im Beispiel müßte die Systemkomponente »Reservation System« dafür sorgen, daß immer, wenn ein Kunde aus der Komponente CustomerMgr gelöscht wird, auch alle ihn betreffenden Reservierungen in HotelMgr entfernt werden). Es kann Fälle geben, in denen eine solche übergeordnete Komponente nicht identifizierbar ist. Dann muß die Komponente, welche die referenzierten Entitäten besitzt und verwaltet (im Beispiel wäre das CustomerMgr) bei jedem Löschvorgang über einen Benachrichtigungsmechnismus alle Komponenten benachrichtigen, die Referenzen auf die Entitäten halten.

6.3 Schnittstellen und Operationen faktorisieren

Beim Entwurf der Schnittstellen und Operationen sollte darauf geachtet werden, dass man ein möglichst sauberes, übersichtliches und redundanzfreies Design erhält. Dazu kann es ggf. nützlich sein, eine zu komplexes und »multifunktionale« Schnittstelle in mehrere kleinere aufzuteilen (Schnittstellen-Faktorisierung). Gleichfalls sollte man eine Operation, deren Semantik unnötig verwirrend und kompliziert ist, vernünftig »normalisieren« und ihre Funktionalität auf mehrere Operationen verteilen. Dies gilt insbesondere dann, wenn man gemeinsam genutzte Funktionalität identifizieren kann -- diese sollte aus den entsprechenden Operationen hinausgezogen und in eine eigene Operation verschoben werden

7 Komponentenspezifikation

7.1 Überblick

In der dritten Phase des Spezifikation-Workflows, der *Komponentenspezifikation*, werden die Nutzungs- und Realisierungsverträge im einzelnen präzisiert und abschließend spezifiziert. Für den Nutzungsvertrag umfaßt dies die Schnittstellenspezifikationen der Komponenten aus Anwendersicht. Für den Realisierungsvertrag wird die Funktionalität der Komponenten festgelegt, auf dessen Grundlage die Implementierung erfolgen kann.

Als Eingabe für die Phase der Komponentenspezifikation dienen (s. Bild 5):
- das *Geschäftstypenmodell (Business Type Model)* als Ergebnis der Verfeinerung des Geschäftskonzeptmodell (Business Concept Model) aus der ersten Phase des Spezifikation-Workflows,
- die bis dahin ermittelten *Schnittstellen* der System- und Geschäftsschicht, die in der zweiten Phase gewonnen wurden und
- die bis dahin ermittelte *Komponentenspezifikation* und ihre *Architektur* aus der zweiten Phase.

Als Ergebnis dieser Phase erhält man:
- die stabilen *Schnittstellen* der System- und Geschäftsschicht als Erfüllung des Nutzungsvertrages und
- die vollständige und präzise *Komponentenspezifikationen* und ihrer *Architektur* als Erfüllung des Realisierungsvertrages.

7.2 Spezifikation der Schnittstellen

Schnittstellen sind die geeigneten Modellierungseinheiten, um Abhängigkeiten zwischen Komponenten aufzuzeigen. Im Vergleich zu Operationen zeichnen sich Schnittstellen durch mehrere Vorteile aus: i) sie modellieren den »richtigen« Abstraktionsgrad, um die Abhängigkeiten der Komponenten untereinander zu beschreiben, ii) sie gruppieren »semantisch« zusammengehörige Operationen, iii) durch ihre weiter gefaßte Sicht auf Komponenten können sie Zustände von Komponenten beschreiben. Solitäre Operationen sind dafür zu isoliert.

Spezifikation der Operationen

Operationen werden spezifiziert durch die Angaben ihrer: i) Eingabeparameter, ii) Ausgabeparameter, iii) der Statusänderung bzgl. der Komponente auf die sie wirken und iv) der Randbedingungen (constraints), denen sie obliegen.

Das Schnittstelleninformationsmodell

Für die Spezifikation einer Schnittstelle wird der Komponentenzustand mit Hilfe des *Schnittstelleninformationsmodells* beschrieben. Jeder Schnittstelle wird ihr eigenes Schnittstelleninformationsmodell zugeordnet, das keine weiteren Beschreibungen von anderen Schnittstellen verwenden darf. Lediglich bei Schnittstellenvererbung darf auf das Modell der geerbten Schnittstelle referenziert werden.

Das Schnittstelleninformationsmodell besteht aus speziellen *Informationstypen* (und deren Attributen und Assoziationen). Diese Informationstypen sind abstrakt und stellen Projektionen der, bereits in der Phase der Komponentenidentifikation eingeführten, Geschäftstypen dar. Sie modellieren das innere Datenmodell der Komponente aus Sicht der Schnittstellen. Zusätzlich existieren noch »einfache« *Datentypen*, die die Typen der Operationsparameter kennzeichnen. Wie das Schnittstelleninformationsmodell bleiben auch die Informationstypen auf diejenige Schnittstelle beschränkt, die sie beschreiben. Lediglich die Verwendung der Datentypen ist global, d.h. jede Schnittstellenbeschreibung darf auf Datentypen verweisen.

Das Schnittstelleninformationsmodell wird zusammen mit der Schnittstellenspezifikation dokumentiert.

Vor- und Nachbedingungen

Um Operationen einer Schnittstelle zu spezifizieren, sind auch ihre *Vor*- und *Nachbedingungen* (Pre- and Postconditions) zu benennen. Zu jeder Operation existiert sowohl eine Vorbedingung als auch eine Nachbedingung. Die Nachbedingung beschreibt das Ergebnis der Operation, unter der Voraussetzung, daß die Vorbedingung erfüllt ist[1]. Vor- und Nachbedingungen werden innerhalb von UML durch OCL formal ausgedrückt[2]. Vor- und Nachbedingungen dürfen

1 Bemerkungen: i) Eine Operation wird nicht automatisch aufgerufen, wenn die Vorbedingung erfüllt ist. ii) Wir eine Operation aufgerufen, deren Vorbedingung nicht erfüllt ist, ist das Ergebnis der Operation nicht definiert. iii) Genau genommen ist ein Ergebnis, das nicht in einer Nachbedingung spezifiziert ist, undefiniert. Erst dadurch können Spezifikationen partiell bzw. inkrementell definiert werden. Um aber nicht in die Verlegenheit zu kommen, alle Nicht-Veränderungen beschreiben zu müssen, gilt die Übereinkunft, daß Eigenschaften, die in der Nachbedingung nicht beschrieben werden, als unverändert angesehen werden.

2 Da OCL recht umfangreich ist, wird an dieser Stelle auf eine weiter Beschreibung von OCL verzichtet, siehe dazu ([Booch99]).

sich nur auf Elemente des (lokalen) Schnittstelleninformationsmodell beziehen, externe Referenzen sind nicht erlaubt.

7.3 Ableitung des Schnittstelleninformationsmodells

Für einfache Komponenten kann das Schnittstelleninformationsmodell als Nebenprodukt der Operationsspezifikation gewonnen werden. Für komplexe Komponenten ist dieses Vorgehen aber nicht effektiv.

Dennoch kann das Schnittstelleninformationsmodell geradlinig von dem Geschäftstypenmodell der ersten Phase (Komponentenidentifizierung) abgeleitet werden. Bereits in der ersten Phase wurde auf Grundlage des Geschäftstypenmodells das Schnittstellenabhängigkeitsdiagramm entworfen, das aufzeigt, welche Schnittstelle für welche Daten zuständig ist. Das Schnittstellenabhängigkeitsdiagramm ist nun seinerseits die Grundlage, um das Schnittstelleninformationsmodell zu entwerfen. Das Vorgehen ist einfach:

> Alle Typen, die zu einer Schnittstelle gehören (Wissen aus dem Schnittstellenabhängigkeitsdiagramm) werden in das Schnittstelleninformationsmodell übernommen. Referenziert ein Typ der untersuchten Schnittstelle auf einen Typ einer anderen Schnittstelle, so wird auch dieser referenzierte Typ in das Schnittstelleninformationsmodell kopiert und von »überflüssigen« Attributen bereinigt.

Dabei entspricht der kopierte Typ nicht mehr dem ursprünglichen referenzierten Typ des Schnittstellenabhängigkeitsdiagramms, sondern ist ein eigener Informationstyp, der nur bzgl. seines Schnittstelleninformationsmodells gültig ist.

Ein weiteres Konzept der Spezifikation des Schnittstelleninformationsmodells besteht darin, in dieser Phase Invarianten der Typen zu benennen. Beispielsweise werden an dieser Stelle Kardinalitäten zwischen den Instanzen der Typen festgelegt. Wichtig ist, daß Invarianten unabhängig von der Implementierung beschrieben werden. Dazu sind graphische Notationen oder die präzisere OCL verwendbar. Aber auch natürlichsprachliche Formulierungen sind geeignet. »Redundanz in der Spezifikation ist gut, solange Klarheit den Aufwand zur Konsistenzeinhaltung aufwiegt« ([Chees00]).

Eine weitere nützliche Technik zur Beschreibung von Vor- und Nachbedingungen sind sogenannte *Momentaufnahmen* (»*snapshots*«)[1] von Objektzuständen, die die Zustandsänderungen bzgl. eines Operationsaufrufs notieren. Zwei Momentaufnahmen werden angefertigt: Jeweils eine Momentaufnahme *vor*

1 Der Begriff »snapshot« stammt aus [DSou98].

dem eigentlichen Funktionsaufruf und eine Momentaufnahme *danach*. Momentaufnahmen sind graphische Notationen in UML-Objektdiagrammen, die helfen sollen die Vor- und Nachbedingungen der Informationstypen zu identifizieren.

7.4 Spezifikation der Systemschnittstellen

In den vorherigen Schritten wurde das Schnittstelleninformationsmodell der Geschäftsschicht abgeleitet. Aber auch die Systemschicht benötigt zur Spezifikation ein Informationsmodell, ein *lokales Schnittstelleninformationsmodell*, das eine Teilmenge des Geschäftstypenmodell sein wird.

Wie zuvor beginnt man damit, eine Kopie des Geschäftstypenmodells zu erstellen. Im Gegensatz zur Geschäftsschicht, bei der das Schnittstellenabhängigkeitsdiagramm ein klare Zuordnung von Typen zu Schnittstellen vornimmt, ist die Zuordnung von Informationstypen und Schnittstellen für die Systemschicht nicht so einfach. Das Vorgehen entspricht aber dem der Geschäftsschicht: jede Schnittstelle erhält ihr eigenes lokales Schnittstelleninformationsmodell, das nur relevante Informationstypen beinhaltet. Sind die Informationsmodelle ähnlich, kann mittels einer gemeinsamen Basisschnittstelle durch Vererbung eine Generalisierung definiert werden.

Geschäftsregeln (Business Rules) wurden bereits im Geschäftstypenmodell angewendet. Diese Regeln pflanzen sich auch auf die Schnittstellen der Systemschicht fort, so daß sie Teil der Spezifikation werden. Geschäftsregeln treten sowohl in der Geschäftsschicht als auch in der Systemschicht auf. Da die Verwendung dieser Schichten separat erfolgen kann (beispielsweise kann die Geschäftsschicht mit einer anderen Systemschicht ausgeliefert werden), sollten immer wiederkehrende Geschäftregeln der Geschäftsschicht zugeordnet werden.

7.5 Spezifikation der Komponenten

Bis jetzt ging es ausschließlich darum, den Nutzungsvertrag zwischen dem Nutzer und der Komponente zu bestimmen. Die Spezifikation der Komponenten legt hingegen den Realisierungsvertrag einer Komponente fest. Besonders wichtig ist dabei die Abhängigkeit einer Komponente von anderen Schnittstellen.

Veröffentlichte und genutzte Schnittstellen

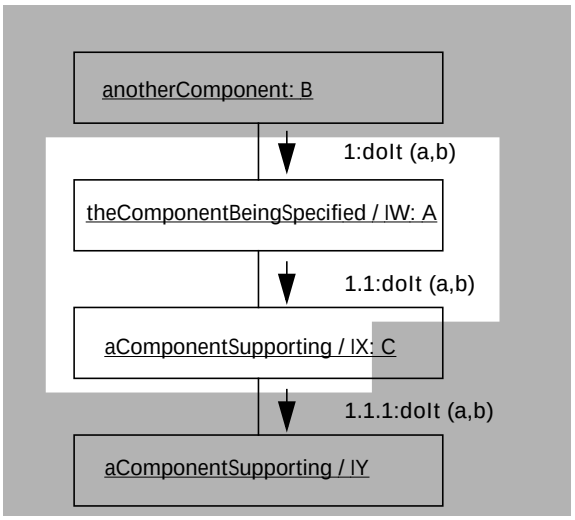
Die Benennung der Komponentenschnittstellen wurde bereits in dem Komponentenarchitekturdiagramm innerhalb der ersten Phase, der Komponentenidentifikation, vorgenommen. Jetzt wird das Diagramm zerteilt, um für jede

Komponente eine eigenständige Spezifikation zu erhalten. Auch werden jegliche Bedingungen, die andere Schnittstellen betreffen und für die Realisierung relevant sind, aufgeführt. Zur Spezifikation gehört auch die Komponentenobjektarchitektur, wie sie bereits in der Phase der Komponenteninteraktion modelliert wurde.

Interaktionsbedingungen zwischen Komponenten

Bedingungen (constraints), wie spezifische Operationen implementiert werden, werden in sog. Interaktionen (interactions) definiert. Anstatt wie „gewöhnliche" Interaktionen auf Programmebene, werden hier Bedingungen auf Spezifikationsebene angegeben. Realisierungen von Komponenten müssen diesen Bedingungen folgen, da sonst die angestrebte flexiblen Zusammenstellung (assembly) der Komponenten zu einer vollständigen Applikation nicht ermöglicht werden kann. Interaktionen sind typischerweise Fragmente der vollständigen Interaktionen, die während der Untersuchung der Operationen innerhalb der zweiten Phase aufgestellt wurden. Die hier behandelten Interaktionen (-Fragmente) beinhalten lediglich i) das Komponentenobjekt, für das die Interaktion definiert wird, ii) die Mitteilung (Operationsaufruf) und iii) die direkten Interaktionen der Komponente. Ein Beispiel für eine Interaktion skizziert Bild 10.

Bild 10 Interaktion einer Komponente



Die Interaktion interessiert nur im Kontext der Komponente A. Die Komponente B gehört nicht dazu. Auch ist lediglich die Schnittstelle IX interessant, nicht hingegen die Komponente C, von der sie angeboten wird. Die Schnittstelle IY wird nicht von A direkt angesprochen und gehört daher auch nicht zur Interaktion von A, die durch den hellen Bereich gekennzeichnet ist.

Ein Problem bei der Formulierung der Interaktionen tritt im Zusammenhang mit den meisten UML-Werkzeugen auf. Diese erlauben es nicht, die Kollaborationsdiagramme in die gewünschte Einzelteile aufzuspalten.

Bedingungen zwischen Schnittsellen

Zwei weitere Punkte gehören noch zur Komponentenspezifikation, die beide mit dem Schnittstelleninformationsmodell zusammenhängen:

- Wie hängen die veröffentlichten Schnittstellen (offered interfaces) zusammen?
- Wie hängen die veröffentlichten Schnittstellen mit den benötigten Schnittstellen (used interfaces) zusammen?

Der Zusammenhang zwischen veröffentlichten Schnittstellen wird durch das folgende Beispiel offensichtlich: zwei unabhängig voneinander spezifizierte Schnittstellen bieten einen namensgleichen Informationstypen an. Man kann daraus weder schließen, daß es sich bei den zurückgelieferten Objekten diesen Typs um dasselbe Objekt handelt, noch daß es sich überhaupt um dasselbe Konzept handelt. Dennoch liefern die verschiedenen Schnittstellen im Beispiel Referenzen auf ein und dasselbe Objekt. Solche logischen Gleichheiten müssen explizit formuliert werden.

Bereits zu Beginn wurde darauf hingewiesen, daß Fragen der Datenhaltung das Schnittstelleninformationsmodell der Systemschnittstelle nicht betreffen. Vielmehr werden die benötigten Informationen durch die Geschäftsschicht zur Verfügung gestellt. Anstatt nun diese Abhängigkeit des Informationsmodells der veröffentlichten Schnittstelle und der genutzten Schnittstelle durch Nachrichten oder Operationsaufrufe zu beschreiben, werden Bedingungen in OCL formuliert, die die Gleichheit der Informationstypen der verschiedenen Schnittstellen definieren.

7.6 Ausklammern und zusammenführen von Schnittstellen

Zum Schluß sei noch erwähnt, daß ähnliche Schnittstellen oftmals durch eine gemeinsame (abstrakte) Basisschnittstelle mittels Vererbung zusammengeführt werden können (»Factoring Interfaces«). Die Schnittstellen werden gleichsam »ausgeklammert«. Eine weitere einfache Generalisierung von Schnittstellen ist oftmals durch Verschmelzung (merging) möglich. Dies gilt besonders dann, wenn die Schnittstellen derselben agierenden Person zugeordnet werden können. Falls unterschiedliche agierende Personen existieren ist es wichtig, die Schnittstellen separat zu halten. Da Vorgänge meist an Rollen gebunden sind, können auch bei modifizierten Vorgängen die betreffenden Systemschnittstellen angepaßt werden.

8 Bereitstellung und Zusammenbau

In diesem Abschnitt soll kurz beschrieben werden, wie ein komponentenbasiertes Softwaresystem, dessen Spezifikation vollständig vorliegt, in einer vorgegebenen Komponentenumgebung installiert werden kann. »Komponentenumgebung« (component environment) bezeichnet hierbei eine Laufzeitumgebung, in der die Komponenten ablaufen können. Eine Komponentenumgebung besteht aus einer Menge von Implementationsregeln, welche von den installierten Komponenten eingehalten werden müssen (Komponentenstandard) sowie einer Menge von Infrastrukturdiensten, die installierten Komponenten zur Verfügung stehen (z.B. Transaktionsdienste, Sicherheit usw.). Beispiele für solche Umgebungen sind etwa COM+ und EJB.

8.1 Technische Umsetzungen

Die Wahl der Komponentenumgebung, in der das System laufen soll (Zielumgebung) hat Einfluß auf die Art und Weise, wie bestimmte Aspekte der Spezifikation realisiert werden. Fast nie werden alle Designmerkmale, die in der Spezifikation eine Rolle spielen, in der Zielumgebung direkt unterstützt. Gegebenenfalls muß deshalb eine *technische Umsetzung* (technology mapping) erfolgen, die den jeweiligen Teilaspekt der Spezifikation mit Hilfe der in der Zielumgebung zur Verfügung stehenden Techniken realisiert. Einige Bereiche, in denen solche Umsetzungen nötig werden können, werden im folgenden kurz behandelt.

8.1.1 Parameter für Operationen

Die in der Spezifikation gewonnenen Ein- und Ausgabeparameter für die verschiedenen Operationen müssen auf passende, in der Zielumgebung verfügbare Datentypen (Basistypen oder strukturierte Typen) abgebildet werden. Manche Umgebungen (z.B. EJB) unterstützen Ausgabeparameter nicht direkt; hier müssen entsprechende Anpassungen vorgenommen werden (indem z.B. der Rückgabetyp der Operation entsprechend angepaßt wird).

8.1.2 Fehlerbehandlung, Exceptions

Ein »Fehler« bei der Ausführung einer Operation bedeutet in der Regel, daß die in der Spezifikation angegebene Vorbedingung für die Operation nicht erfüllt ist. Es gibt verschiedene Ansätze, dieser Situation Herr zu werden. Beispiels-

weise kann man zusätzliche, alternative Vorbedingungen vorsehen, welche die entsprechenden Fehlerzustände spezifizieren. Ist eine solche Vorbedingung bei der Ausführung der entsprechenden Operation erfüllt, kann dies dem Aufrufer über zusätzliche Rückgabewerte oder eine Exception (falls von der Zielumgebung unterstützt) mitgeteilt werden.

8.1.3 Schnittstellen

Die in der Spezifikation gewonnenen Komponenten können jeweils mehrere Schnittstellen unterstützen. Bestimmte Zielumgebungen (namentlich EJB) unterstützen jedoch nur eine Schnittstelle pro Komponente. Hier müssen geeignete Anpassungen vorgenommen werden. Sollen EJBs benutzt werden, kann man z.B. alle benötigten Schnittstellen mittels Vererbung zu einer gemeinsamen Schnittstelle zusammenfassen und letztere dann als Schnittstelle der Komponente (»Remote Interface« bei EJBs) deklarieren.

8.1.4 Implementation der Geschäftskomponenten

Geschäftskomponenten modellieren in der Spezifikation oft *Manager-Komponenten*, welche dann wiederum eine größere Menge von Geschäftstypen verwalten (Beispiel: eine Geschäftskomponente »CustomerMgr« verwaltet eine Menge von »Customer«-Geschäftstypen). Bei der Implementierung eines solchen Systems stellt sich die Frage, ob neben den Manager-Komponenten auch die Geschäftstypen als vollwertige Komponenten im Sinne der Zielumgebung implementiert werden sollen. Pragmatische Anforderungen (Effizienz, »Grobkörnigkeit« der Komponenten) sprechen häufig gegen letzteren Ansatz, so daß die Geschäftstypen oft als einfache Objekte lokal beim Klienten realisiert werden. Sie kommunizieren dabei intern mit ihrer Manager-Komponente, was beim Klienten die »Illusion« einer echten Komponente erzeugt und damit ein saubereres Programmiermodell ermöglicht.

9 Glossar

Anwendungsfallmodell (Use Case Modell): Das Anwendungsfallmodell aus der Anforderungsanalyse wird zur Komponentenidentifikation verwendet.

Applikationsarchitektur: Die von [Chees00] betrachteten Applikationen lassen sich in vier Schichten unterteilen: i) Nutzerschnittstelle, ii) Nutzerdialogschicht, iii) Systemschicht und iv) Geschäftsschicht.

Geschäftkonzeptmodell (Business Concept Model): Das Geschäftskonzeptmodell dient als Vorläufer für das Geschäftstypenmodell.

Geschäftstypenmodell (Business Type Model): Das Geschäftstypenmodell wird in der Phase der Komponntenidentifikation gewonnen.

Geschäftsschicht (Business Layer): Die Geschäftsschicht beinhaltet die Geschäftsdienste und ist die vierte Schicht der Applikationsarchitektur nach [Chees00].

Informationstyp: Der Informationstyp ist ein abstrakter Typ des Schnittstelleninformationsmodells und stellt eine Projektion der konkreten Geschäftstypen (des Geschäftstypenmodells) dar.

Kerntypen: Die Kerntypen sind die wesentliche Geschäftsentitäten und werden in der Phase der Komponentenidentifizierung identifiziert.

Komponentenobjekt: Das Komponentenobjekt ist die lauffähige, zustandsbehaftete Instanz einer installierten Komponente.

Schnittstelleninformationsmodell Interface Information Model: Separates Datenmodell für jede Schnittstelle.

Schnittstellentyp: Der Schnittstellentyp ist die konzeptionelle Ausprägung einer implementierten Schnittstelle.

Spezifikation-Workflow: Der Spezifikation-Workflow ist der Hauptbestandteil des Entwicklungsprozeß und besteht aus drei Phasen:

- **Komponentenidentifizierung**,
- **Komponenteninteraktion** und
- **Komponentenspezifikation**.

Systemschicht (System Layer): Die Systemschicht beinhaltet die Systemdienste und und ist die dritte Schicht der Applikationsarchitektur nach [Chees00].

Literatur

[Booch99] Booch, Grady; Rumbaugh, James; Jacobson, Ivar: "The Unified Modeling Language, User Guide", Addison-Wesley, 1999.

[Chees00] Cheesmann, John; Daniels, John: "UML-Componentents", Addison-Wesley, 2000.

[CockburnWeb] http://members.aol.com/acockburn/

[DSou98] D'Souza, F. Desmond; Wills, Alan Cameron: "Objects, Components, and Frameworks with UML, The Catalysis Approach", Addison Wesley, 1998.

[Kruc99] Philippe Kruchten (with G. Booch and R.Reitman), Citation: Rational Unified Process - An Introduction, Addison-Wesley-Longman, 1999.

Änderungshistorie

Autor	Datum	Version	Änderung	Freigegeben
JF	25.06.2001	0.1	Ersterstellung	05.07.2001
ABi	26.07.2001	0.1.1	Referenzen angelegt	