Verbesserung der Lokalität und Wiederverwendbarkeit von Geschäftsprozeßspezifikationen: Probleme und Lösungsansätze am Beispiel kundenorientierter Hotelgeschäftsprozesse

Diplomarbeit von Volker Ripp
Fachbereich Informatik, Universität-Hamburg

betreut von
Prof. Dr. Florian Matthes
Technische Universität Hamburg-Harburg
Arbeitsbereich Software-Systeme

Prof. Dr. Joachim W. Schmidt
Technische Universität Hamburg-Harburg
Arbeitsbereich Software-Systeme

extern betreut von
Dr. Rüdiger Buck-Emden
SAP-AG Walldorf

20 März 1998

Zusammenfassung

In der Software-Entwickung stellt die evolutionäre Weiterentwicklung betriebswirtschaftlicher Systeme ein großes Problem dar. Ein in dieser Arbeit dargestellter Lösungsansatz verfolgt die Entwicklung von kooperierenden Informationssystemen, die der evolutionären Systemweiterentwicklung Rechnung tragen, indem sie neue Abläufe in Satellitensystemen implementieren und weiterhin altbewährte in vorhandenen Systemen weiterbenutzen. Die Entwicklung eines solchen Satellitensystems wird nachfolgend beschrieben; dabei liegt der Schwerpunkt dieser Arbeit auf der Methode des Systementwurfs.
Hierbei handelt es sich um einen geschäftsprozeß-orientierten Ansatz, mit dem zielgerichtete Geschäftsprozesse, d.h. Geschäftsprozesse, in denen alle Beteiligten auf ein gemeinsames Ziel hinarbeiten, modelliert und anschließend in Form eines Informationssystems implementiert werden.
Bei der Entwicklung werden semantische Lücken, die bei der Modellierung von Geschäftsprozessen bis hin zur Implementierung des unterstützenden Systems in einer Programmiersprache bestehen, durch die Wiederverwendung von Geschäftsprozeßspezifikationen geschlossen, was zu einer bruchlosen Entwicklungsmethode führt, die gleichermaßen kundenorientierte Geschäftsprozeßmodellierung als auch die Systemmodellierung und Implementierung in allen Phasen des iterativen, inkrementellen Software-Entwicklungsprozesses hinreichend gut und lückenlos unterstützt.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Unternehmen heutiger Zeit stehen unter dem zunehmenden Druck eines dynamischen Marktwandels. Der voranschreitenden Globalisierung und zeitgleicher Neuordnung des Marktes von einem Verkäufermarkt hin zu einem Käufermarkt fühlen sich hierarchisch strukturierte Unternehmen nicht mehr gewachsen. Ihre historisch entstandenen, auf relativ konstanten Bedingungen basierenden, hierarchischen und funktionalen Organisationsformen verankern starre Informationsflüsse in den unternehmens- und abteilungsübergreifenden Geschäftsprozessen. Eine Folge solcher Organisationsformen sind lange Liegezeiten von internen und externen Aufträgen sowie Abstimmungsschwierigkeiten an den Schnittstellen.
Heutzutage vollziehen sich Marktveränderungen im allgemeinen rasch und diskontinuierlich. Unternehmen versuchen diesen zu begegnen, indem sie eine engere Markt- und Kundennähe suchen, um somit Auftragsabwicklungszeiten und Kosten zu reduzieren. Geschäftsprozeßgetriebene Organisationsstrukturen, in denen ein organisatorisches Veränderungspotential systemimmanent ist, erhalten dadurch zusehends Einzug in Unternehmen und bewirken eine Reorganisation der Unternehmensstruktur [17, 33].
Eine Verlagerung der Wertschöpfung von der materiellen Produktion hin zu Dienst- und Serviceleistungen direkt am Kunden zeichnet sich ab. Ziel der Unternehmen ist es, den Kunden in den gesamten Wertschöpfungsprozeß zu integrieren, wobei die Kundenzufriedenheit als ein Maß des Erfolges dieses Ansatzes gewertet werden kann. Zur Leistungserbringung werden daher oftmals eine Vielzahl von kundenspezifischen Informationen benötigt. Eine konsequente Ausrichtung der Geschäftsprozesse an den Kundenbedürfnissen erfordert eine hohe Anpassungsleistung, eine gleichzeitige Verkürzung der Produktzyklen sowie betriebliche Prozesse, die dem Kundennutzen höchste Priorität einräumen. Eine dynamische und evolutionäre Anpassung der organisatorischen Unternehmensstrukturen an die Marktgegebenheiten ist die Folge.

Eine daraus resultierende wichtige Frage ist, inwieweit heutige und zukünftige Informationstechnologien dem permanenten organisatorischen Wandel eines Unternehmens Rechnung tragen können, ohne dabei Gefahr zu laufen, aufgrund von Veränderungen neue ad hoc eingeführte und untereinander nicht abgestimmte Hilfswerkzeuge prozeßübergreifend einsetzen zu müssen. Ein solcher Wildwuchs von Einzelsystemen würde dem Ziel des produktivitätssteigernden Einsatzes von Informationstechnologie entgegenwirken und im schlimmsten Fall sogar blockieren. Der Fokus bei der Entwicklung von Informationssystemen muß sich zwangsweise auf die Unterstützung des ganzheitlichen und effizienten Abwickelns unterschiedlicher,

teils variabler Geschäftsprozesse auf flexiblen und offenen Anwendungsarchitekturen richten.

1.1 Motivation

Der Druck des globalen Wettbewerbs und fallende Marktschranken verändern die weltweite Unternehmenslandschaft. Unternehmen befinden sich nicht nur in einer organisatorischen Umstrukturierung, sondern sie schließen sich zusehends zu schlagkräftigen strategischen Allianzen zusammen, um im immer härter werdenden Wettbewerb bestehen zu können. Diese Allianzen erfordern eine neue Dimension der unternehmensübergreifenden Kooperation unter Ausnutzung der sich rasch fortentwickelnden Netz- und Informationstechnik. Durch offene Netzwerkinfrastrukturen können in naher Zukunft immer mehr Dienste, die weit über den reinen Informationsaustausch hinausgehen, verlagert werden. Die kommerzielle Nutzung des WWW bietet Unternehmen eine solide Basis für Zusammenarbeit und Kooperation in dieser Hinsicht. So könnten z.B. mehrere Unternehmen ihre gemeinsamen Aktivitäten über das WWW koordinieren; mit der gleichen Technologie wäre die Koordination von Mitarbeitern innerhalb einer Firma über das Intranet realisierbar, oder die Erschließung von neuen Vertriebskanälen zum Endverbraucher wäre möglich [17, 33].
Diese unter dem Schlagwort *electronic commerce* zusammengefaßten Kooperationsformen sind mit einer geeigneten Entwicklungsmethodik, basierend auf einem Koordinationsmodell für aktive Agenten, zu modellieren.
Das zugrundeliegende Koordinationsmodell ist durch den Entwicklungsverlauf kooperativer Geschäftsanwendungen begründet. Der Verlauf zeigt die anfängliche Verwendung von datenorientierten Modellen hin zu objektorientierten und agentenbasierten auf. Letzteres nimmt eine Systemsicht ein, in der das System aus einer Komposition von verschiedenen, aber verteilten, unter Umständen auch mobilen, autonomen Agenten besteht. Jeder Agent ist hierbei wohldefiniert, autonom und besitzt eine eigene Wissensbasis. So wird z.B. ein Unternehmen als Agent angesehen, das durch zeitlich langanhaltende Aktivitäten mit einem anderen Agenten, zum Beispiel einen Kunden, versucht, das gemeinsame Kooperationsziel zu erreichen. Diese sogenannten *Business Agents* stellen keine einfache Erweiterung der in betriebswirtschaftlichen Anwendungen gebräuchlichen *Business Objects* um autonome Aktivitäten oder Mobilität dar, vielmehr zeichnen sie sich durch Verhalten und die Fähigkeit, Konversationen zu führen, aus [26].
Konversationen dienen der Koordination einzelner Agenten; insbesondere regeln sie die kooperative Zusammenarbeit zwischen ihnen. Konversationsspezifikationen hingegen beschreiben die zwischen zwei Agenten möglichen Konversationen und spiegeln demzufolge einen Teil der Ablauflogik innerhalb des Systems wieder [16]. Die Komposition autonomer Agenten zu einem Gesamtsystem mittels loser, aber autonomiebewahrender Kopplung über Konversationen kann als Ansatz hin zur Entwicklung von evolutionären Informationssystemen dienen, der den raschen marktinduzierten Veränderungen organisatorischer Unternehmensstrukturen sowie variabler Geschäftsprozesse gerecht werden kann.

1.2 Zielsetzung

Betriebswirtschaftliche Softwaresysteme, wie zum Beispiel das SAP R/3-System, bilden eine Fülle von standardisierten Geschäftsprozessen ab und unterstützen deren Ablauf in komfortabler Weise.

Im Zuge der Globalisierung ist ein Wandel organisatorischer Unternehmensstrukturen zu beobachten, der eine Veränderung bestehender Geschäftsabläufe mit sich bringt. Hersteller betriebswirtschaftlicher Software müssen zügig auf diesen wachsenden Wandel reagieren. Es fällt ihnen jedoch schwer, ihre Software den evolutionären Veränderungen rechtzeitig anzupassen. Die Ursache dieser Schwierigkeiten liegt in der Systemrealisierung. So sind in vielen Systemen Geschäftsabläufe festkodiert und dadurch sehr eng an die Ablauflogik des Programms und an die modellierten Geschäftsdaten, den sogenannten *Business Objects*, gebunden.
Eine Umstrukturierung „festverdrahteter" Geschäftsabläufe birgt große Veränderungen im zugrundeliegenden Gesamtsystem und ist kaum in vertretbarem Aufwand umzusetzen.
Evolutionäre Weiterentwicklung betriebswirtschaftlicher Systeme stellt demzufolge ein großes Problem dar und kann nur mit sehr umfangreichen Einstellungsänderungen, zum Beispiel beim SAP R/3-System durch das *customizing*, oder durch Programmänderung durchgeführt werden.
Ein in dieser Arbeit dargestellter Lösungsansatz verfolgt die Entwicklung von kooperierenden Informationssystemen, die der evolutionären Systemweiterentwicklung Rechnung tragen, indem sie neue Abläufe in Satellitensystemen implementieren und weiterhin altbewährte in vorhandenen Systemen weiterbenutzen.
Kooperation zwischen autonomen Systemen setzt Kommunikation und Koordination voraus. Um bei der Entwicklung von kooperierenden Systemen weitestgehend von der Kommunikationsebene abstrahieren zu können, wird als Basisarchitektur die aus der Sprechakttheorie [6] abgeleiteten *Business Conversations* [25] in ihrer Implementierung der TBC-Agentenumgebung [38] eingesetzt. Hierbei handelt es sich um eine Systemarchitektur, in der autonome Agenten zielgerichtete Konversationen führen, um gemeinsame Aufgaben zu lösen oder Arbeitsabläufe zu unterstützen [26, 16, 30].
Die Entwicklung eines solchen Satellitensystems wird nachfolgend beschrieben; dabei liegt der Schwerpunkt dieser Arbeit auf der Methode des Systementwurfs.
Hierbei handelt es sich um einen geschäftsprozeß-orientierten Ansatz, mit dem zielgerichtete Geschäftsprozesse, d.h. Geschäftsprozesse, in denen alle Beteiligten auf ein gemeinsames Ziel hinarbeiten, modelliert und anschließend in Form eines Informationssystems implementiert werden.
Bei der Entwicklung wird versucht, semantische Lücken zu schließen, die bei der Modellierung von Geschäftsprozessen, zum Beispiel mit Ereignisgesteuerten Prozeßketten (EPK), bis hin zur Implementierung des unterstützenden Systems in einer Programmiersprache, bestehen.
Ziel ist es, eine bruchlose Entwicklungsmethode einzuführen, die gleichermaßen die Geschäftsprozeßmodellierung als auch die Systemmodellierung und Implementierung in allen Phasen des iterativen, inkrementellen Software-Entwicklungsprozesses hinreichend gut und lückenlos unterstützt.

1.3 Gliederung

Das in dieser Arbeit verwirklichte erste Inkrement eines umfangreichen *Hotel-Front-Office-Systems* ist im Rahmen des praktischen Teils dieser Arbeit bei SAP in Walldorf entstanden und deckt die Funktionalität eines Hotelzimmer-Reservierungssystems ab, welches Zimmer und deren Verfügbarkeiten verwaltet und den Geschäftsprozeß der Zimmerreservierung unterstützt.
Das System wurde in der objektorientierten Sprache Tycoon-2 [10] implementiert und setzt

auf der in [38] entwickelte Agentenumgebung auf.
Der folgende beschriebene Entwicklungsprozeß dieses ersten Inkrements lehnt sich an die Methoden der objektorientierten Entwicklung von Booch, Rumbaugh und Jacobson [1, 32, 13] an und erweitert Jacobsons [13, 14] Anwendungsfälle um zielgerichtete Geschäftsprozesse. Die entwickelten Klassenmodelle werden in der *Unified Modelling Language* (UML) notiert [2, 7].

Kapitel 2

Modellierung arbeitsteiliger Geschäftsprozesse

Ziel der Modellierung arbeitsteiliger Prozesse ist es, betriebliche Abläufe auf einem verständlichem Abstraktionsniveau zu formulieren und mit Hilfe geeigneter Notationen zu beschreiben. Ideal wäre eine Beschreibung, aus der sich wesentliche Aspekte der späteren Implementierung maschinell ableiten ließen. Dabei müssen im wesentlichen die betrieblichen Abläufe eines Geschäftsprozesses in zeitlicher Reihenfolge wiedergegeben werden, sowie die einzelnen Ereignisse innerhalb eines Prozesses und die auf ihnen folgenden Aktionen. Es entsteht eine komplexe Struktur, die chronologische sowie ablauforganisatorische Zusammenhänge eines Prozesses wiedergibt.
Ein Modell zur Beschreibung von Geschäftsprozessen muß im wesentlichen Antwort auf diese Fragen geben:

- Wann muß etwas bearbeitet werden?
- Wer bearbeitet es?
- Wo wird es bearbeitet?
- Welche weiteren Informationen werden zur Bearbeitung benötigt?

In den folgenden beiden Abschnitten werden zwei Modelle zur Beschreibung von Geschäftsprozessen vorgestellt. Das erste Modell ist die Ereignisgesteuerte Prozeßkette (EPK), mit der zum Beispiel die im SAP R/3-System modellierten Geschäftsprozesse beschrieben sind.
Das zweite ist die aus der Sprechakttheorie abgeleitete Konversationsspezifikation, die die Grundlage für die prozeß- und agentenorientierte Modellierung von Geschäftsprozessen dieser Arbeit bildet. In einem weiteren Abschnitt wird die Analogie zwischen EPK-Paaren und Konversationsspezifikation erläutert; insbesondere wird die Eignung dieses Modells für die weiterführende Systemmodellierung und Systemimplementierung untersucht.

2.1 Ereignisgesteuerte Prozeßkette (EPK)

Ereignisgesteuerte Prozeßketten sind in der Lage, Geschäftsprozesse so zu beschreiben, daß sie die im vorherigen Abschnitt gestellten Fragen unmittelbar beantworten.

Das Grundprinzip der Ereignisgesteuerten Prozeßkette beruht auf dem Eintreten eines Ereignisses und der Forderung, daß auf jedes dieser Ereignisse in Form einer Handlung reagiert werden muß. Jede Handlung löst neue Ereignisse aus, denen in entsprechender Weise begegnet wird. Die so entstandene Kette ist das Skelett einer Prozeßbeschreibung bestehend aus einer Folge von Ereignissen und Aktionen, die von einem Anfangsereignis bis zu einem oder mehreren Endereignissen läuft.
In einem Prozeß sind wie in Abbildung 2.1 angedeutet mehrere verschiedene Abschluß- oder

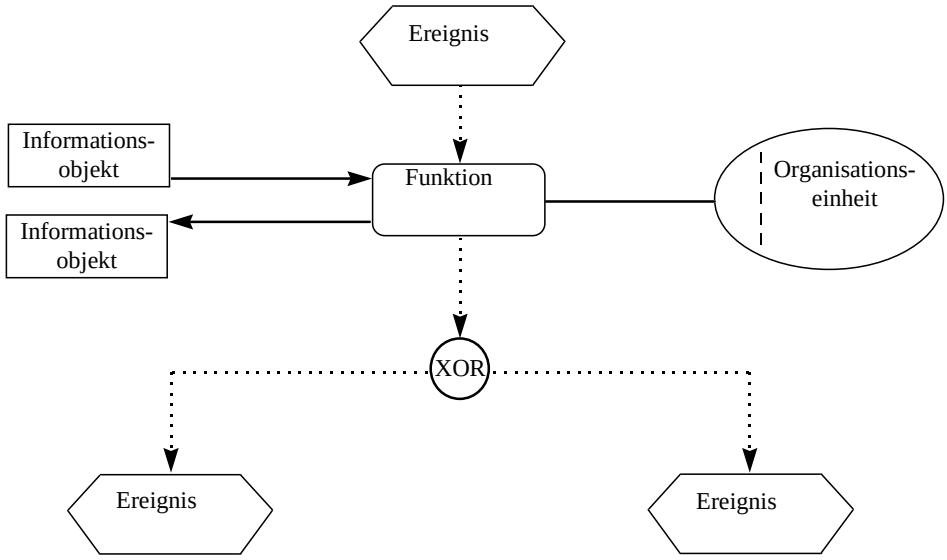

Abbildung 2.1: Elemente einer EPK

Folgeereignisse möglich: So kann zum Beispiel der Prozeß einer Auftragsbearbeitung mit dem Zahlungseingang, aber auch mit dem Eintreffen einer Stornierung enden.
Handlungen können mehrere Ereignisse nach sich ziehen, so daß beim Eintreten mehrerer Ereignisse jeder Pfad des einzelnen Ereignisses verfolgt werden muß. Hierbei handelt es sich um das nebenläufige Ausführen mehrerer Teilprozesse innerhalb des beschriebenen Gesamtprozesses. Verzweigungen sind elementare Teilstrukturen einer EPK, die durch Verknüpfungsoperatoren beschrieben werden. Bei exklusiven Verzweigungen könnten zum Beispiel mehrere einzelne Ereignisse auf eine Aktion folgen, von denen aber nur eins eintreten darf, d.h. das Eintreten eines Ereignisses schließt das Eintreten aller anderen aus. In diesem Fall darf nur der Pfad des eingetroffenen Ereignisses weiterverfolgt werden. Eine EPK spezifiziert den generellen Ablauf eines Prozesses, wobei eine Prozeßinstanz durch einen konkreten Pfad durch die EPK von einem Anfangsereignis bis zum jeweiligen aktuellen Ereignis, dem Ausführungsstand entsprechend, beschrieben ist. Handlungen werden organisatorischen Einheiten zugeordnet, die von ihnen auszuführen sind. Die dafür benötigen Informationen sind durch Informationsobjekte beschrieben. Der chronologische Ablauf eines Prozesses ist durch den Pfad innerhalb einer EPK bestimmt. Eine formale Beschreibung der Ereignisgesteuerten Prozeßketten findet sich in [17].
In Abbildung 2.1 werden die Grundelemente einer EPK dargestellt und im folgenden kurz erläutert:

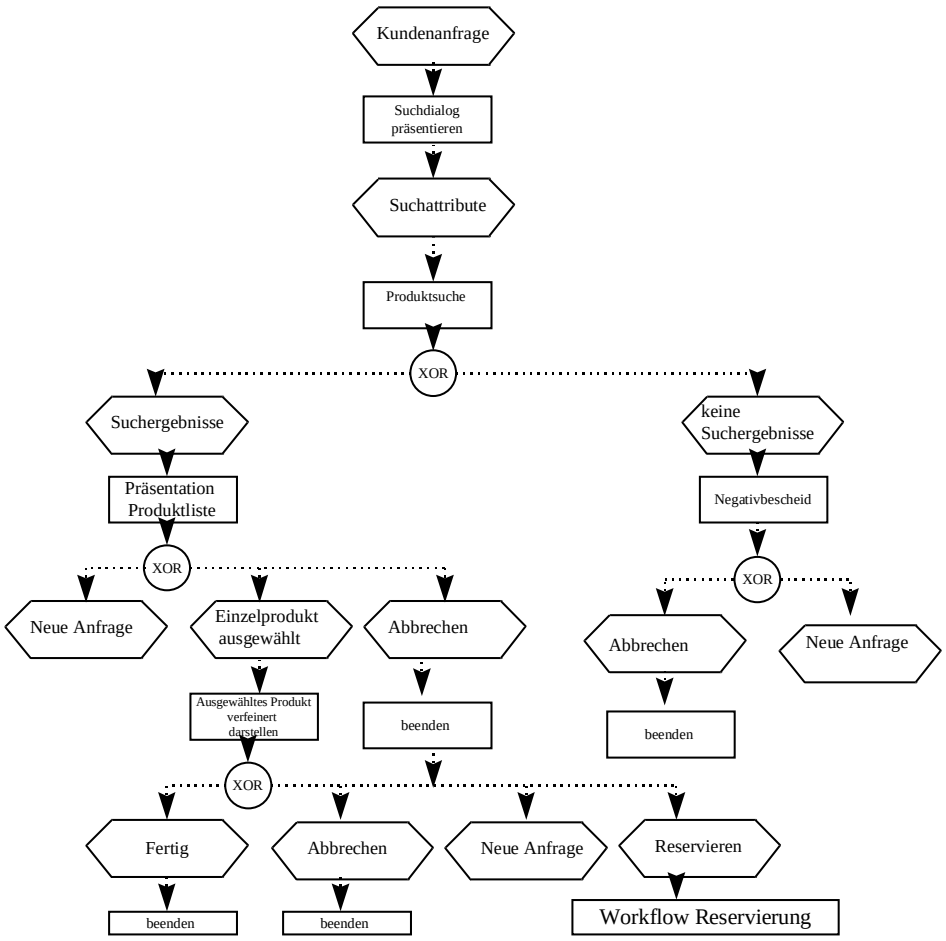

Abbildung 2.2: Angebotserstellung aus Sicht des Hotels

Ereignisse werden durch ein Sechseck beschrieben, die anzeigen, wann etwas innerhalb eines Prozesses bearbeitet wird. Wie diese Bearbeitung stattfindet, beschreibt die mit einem abgerundeten Rechteck dargestellte Aktion, Aufgabe oder Funktion. Jede auszuführende Aktion benötigt eingehende Daten, die als ein Informationsobjekt in einem Rechteck dargestellt sind. Werden während der Bearbeitung Informationen gewonnen, so werden auch diese mittels Informationsobjekte dargestellt. Ereignisse und Funktionen werden durch gepunktete Pfeile verbunden; sie geben den zeitlichen Geschäftsablauf wieder. Eingehende und ausgehende Informationen werden mit durchgezogenen Pfeilen zwischen Funktionen und Informationsobjekten dargestellt, wobei die Pfeilrichtung anzeigt, ob es sich um Eingabe- oder Ausgabedaten der Funktion handelt. Der Informationsfluß innerhalb eines Prozesses wird dadurch unmittelbar ersichtlich.
Welche Organisationseinheiten die Funktionen oder Aufgaben zu erbringen haben, beschreibt die an die Funktion durch einen Strich dargestellte statische Bindung.
Verzweigungen innerhalb eines Prozesses werden mit den Verknüpfungsoperatoren dargestellt, wobei XOR, OR oder AND innerhalb des Kreises die Art der Verzweigung wiedergibt. Sprünge

aus einem Prozeß zu einem Folgeprozeß werden durch einen Prozeßwegweiser beschrieben, der in Abbildung 2.2 und 2.3 zu sehen ist. Oftmals werden Organisationseinheiten und Informationsobjekte zu Gunsten der Übersichtlichkeit ausgeblendet, so daß Prozesse nur auf Ebene der Ereignisse und Funktionen betrachtet werden. Diese Teilansicht reicht jedoch schon aus, um dem Betrachter ein grobes Verständnis des Prozeßablaufs zu vermitteln. Exemplarisch wird in Abbildung 2.2 eine EPK gezeigt, wie sie bei einer Angebotserstellung aussehen könnte. Das auslösende Ereignis dieses Prozesses ist die Kundenanfrage. Die Durchführung des beschriebenen Prozesses kann von einer an der Rezeption arbeitenden Person ausgeführt werden, wobei die Kundenanfrage über das Telefon kommt. Es ist ebenso denkbar, daß die Anfrage aus dem Internet an ein Reservierungssystem gestellt wird, so daß die Angebotserstellung vom Reservierungssystem und nicht von einem Menschen bearbeitet werden muß. Dieser Anwendungsfall wird in Abbildung 2.2 betrachtet. Der Prozeß beschreibt, wie auf eine ankommende

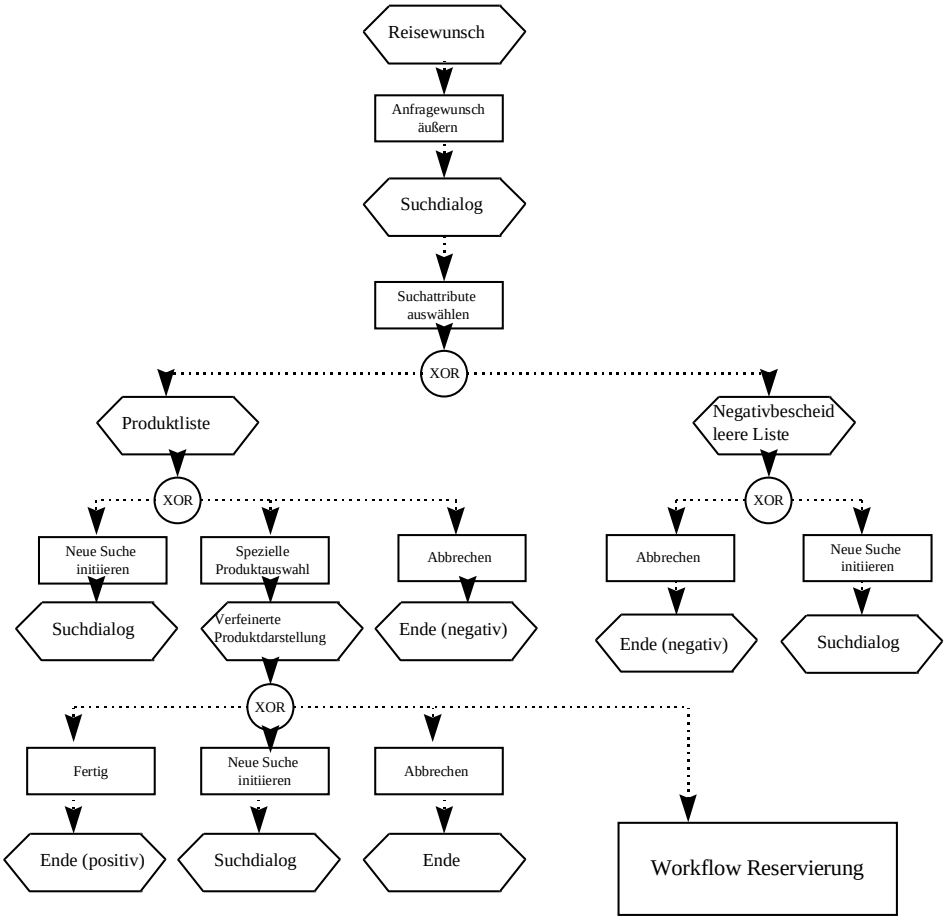

Abbildung 2.3: Kundenanfrage aus Sicht des Kunden

Kundenanfrage zu reagieren ist. In diesem Fall wird dem Kunden eine Schablone des Suchdialoges geschickt, die die für den Kunden frei wählbaren Suchattribute einer möglichen Anfrage enthalten. Durch das Ausfüllen des Suchdialogs äußert der Kunde seine Anfrage, die nachfolgend ausgewertet und mit entsprechenden Angeboten in Form einer Produktliste beantwortet

wird. Dabei ist jedes Produkt als einzelnes Angebot zu verstehen, denn die Produktliste stellt das gesamte Angebot des Hotels bezüglich der Anfrage dar. Der Kunde kann bei Gefallen eines oder mehrere Produkte wählen, d.h. er entscheidet sich für ein entsprechendes Angebot und erteilt den Auftrag, indem er in den hier nicht weiter aufgeführten Prozeß der Zimmerreservierung verzweigt.
Betrachtet man eine Organisationseinheit, die sowohl Angebote erfragt als auch Angebote erstellt, d.h. die in der Rolle eines *brokers*, auftritt, so erkennt man, daß innerhalb einer EPK nur *eine* Prozeßsicht wiedergeben ist, nämlich die des Hotels oder die des Kunden. Die in Abbildung 2.3 aufgeführte analoge EPK zur Angebotserstellung stellt die Anfrage aus Sicht des Kunden dar.
Man erkennt, daß Ereignisse aus Sicht des Kunden Aktionen innerhalb der Hotelsicht sind und umgekehrt, so daß man durch das Zusammenfassen von analogen EPK-Paaren eine redundanzfreie Prozeßbeschreibung beider Ereignisgesteuerten Prozeßketten finden sollte. Eine nicht formale, aber plausible Verschmelzung des hier vorgestellten EPK-Paares findet sich im Abschnitt 2.3.

2.2 Das Modell der „Business Conversations“

Es besteht die Notwendigkeit, neue Informationssysteme zu schaffen, die basierend auf neuen Modellen zur Unterstützung von kundenorientierten Geschäftsprozessen rapiden und kontinuierlichen Veränderungen (Kapitel 1.1) gewachsen sind. Sie eignen sich besonders, wenn für die Leistungserbringung eine Vielzahl kundenspezifischer Informationen benötigt werden. So ist es Aufgabe des Dienstleisters, den Kunden in den gesamten Wertschöpfungsprozeß zu integrieren. Der Erfolg dieser Bemühungen läßt sich nur schwer fassen und wird maßgeblich an der systematische Messung von Kundenzufriedenheit festgemacht [17].

Wertschöpfung am Kunden

Der Trend hin zu individuell zugeschnittenen Dienstleistungen verdrängt zusehends Standardleistungen, auf die man als bald verzichten wird, was zu Prozessen führt, die dem Kundennutzen höchste Priorität einräumen und weit über bisherige Organisationsbereiche und Grenzen hinweggehen.
Es bedarf also einer neuen Sicht, die Geschäftsprozesse nicht nur funktional strukturiert, sondern auch prozeß- und kundenorientiert. Das Hauptmerkmal solcher Geschäftsprozesse liegt auf der Existenz einer binären Kunden Dienstleisterbeziehung. Die sich hieraus ableitende Definition eines Geschäftsprozesses sieht wie folgt aus:

> Im allgemeinen erzeugt ein Geschäftsprozeß einen Wert für den Kunden. Bei der ganzheitlichen Betrachtung des Geschäftsprozesses, d.h. von der Auftragserteilung bis hin zur Befriedigung des Kundenwunsches werden Verantwortlichkeiten und Rollen sichtbar, die zur Erfüllung dieses Ziels beitragen.

„Business Conversations“

Das Leitmotiv des Modells der *Business Conversations* sind Sprechakte zwischen Kunde und Dienstleister, die das von Winograd [40] eingeführte Konzept der *conversation for action* berücksichtigen. Es bildet den Rahmen für eine anwendungsnahe Modellierung von Informationssystemen auf Grundlage von Sprechakten. Dabei orientiert man sich begrifflich an der Geschäftsprozeßdomäne und erfaßt neben den statischen Aspekten eines Dienstes gleichermaßen die dynamische Interaktion von Kunde und Dienstleister über die Zeit.
Basis des Modells ist die Annahme, daß die Kommunikationsmuster der Sprechakte universell sind, d.h. sie sind von Art der Akteure und von den verwendeten Kommunikationsmedien und -protokollen unabhängig. Dabei basiert das Modell auf der Theorie, daß eine Kommunikation mit dem Ziel einer Aktion zwischen einem Kunde und Dienstleister in einem Zyklus mit vier Phasen unterteilt werden kann: Anfrage, Übereinkunft, Leistung und Rückmeldephase (feedback) werden in Abbildung 2.4 verdeutlicht. Aus dem Kooperationsmodell der *Business*

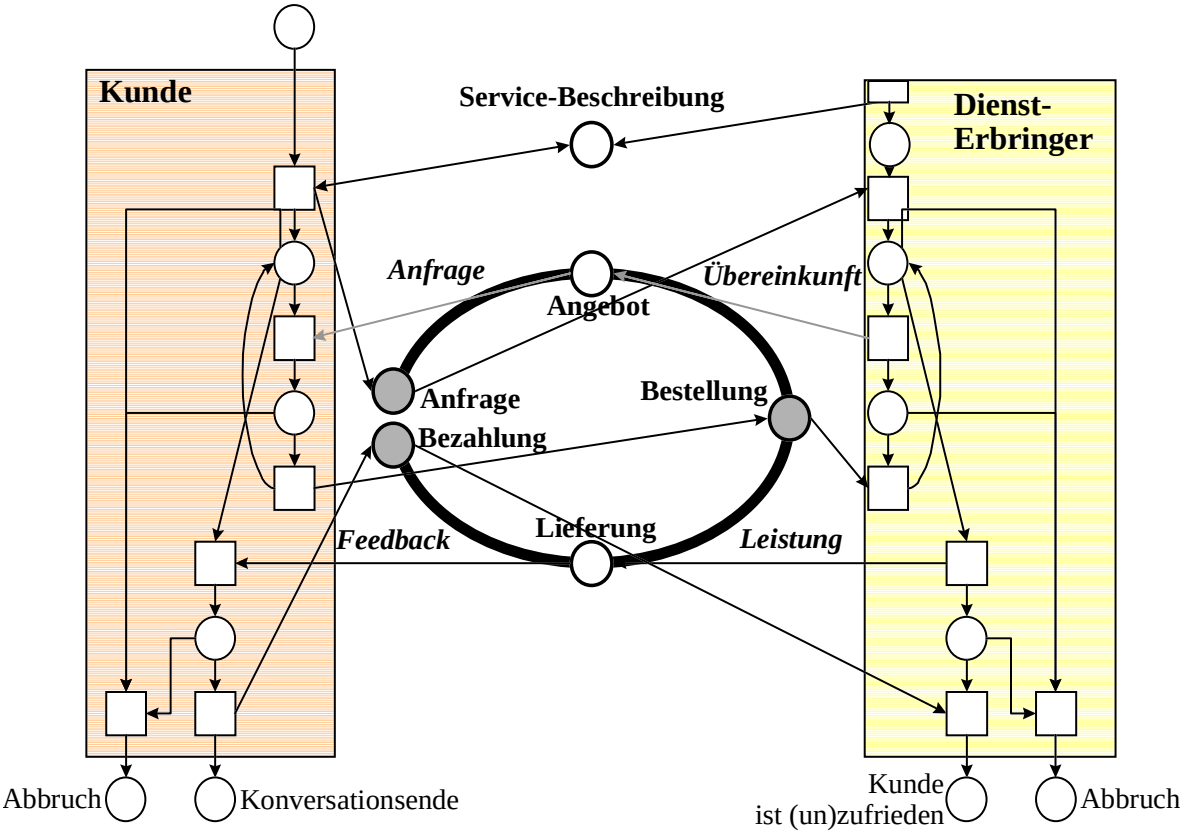

Abbildung 2.4: Vier Phasen des Kooperationsmodell der „Business Conversations“

Conversations wird unmittelbar ersichtlich, daß die wichtigste Voraussetzung beim Beginn einer Kooperation die vom Diensterbringer zur Verfügung gestellte Dienstbeschreibung in Form einer oder mehrerer Konversationsspezifikationen ist. Am Anfang einer jeden Konversation wird überprüft, ob die Konversationsspezifikation des Kunden mit der des Diensterbringers konform ist. Die Konformität zweier Spezifikationen ist gegeben, falls es in beiden gemein-

same Pfade vom initialen Zustand, der als Eintrittspunkt einer Konversation zu verstehen ist, bis zu mindestens einen Endzustand, der nicht den Abbruch einer Konversation, sondern das Erbringen der gewünschten Leistung verkörpert, existiert. In dieser Abbildung wird die Konformität zweier Spezifikationen als gemeinsames Petrinetz [15] angedeutet. Kommt eine Konversation und damit die Kooperation zwischen Kunde und Diensterbringer zustande, befinden sich beide Partner innerhalb einer der vier angesprochenen Phasen. Die Steuerung des Petrinetzes läßt ein Hin- und Herspringen in der sogenannte Übereinkunftsphase zwischen Angebot und Bestellung zu. In dieser Phase werden die Voraussetzungen, d.h. unter welchen Bedingungen eine Kooperation stattfinden soll, verhandelt. Kommt es zu einer Übereinkunft, so ist das Ergebnis dieser Verhandlung für beide vertraglich bindend, so daß danach in die Leistungsphase übergegangen werden kann. Nach Erbringen der Leistung endet der Zyklus mit der Rückmeldung des Kunden in Form einer Bezahlung oder irgendeiner anderen Rückmeldung. In dieser Phase erhält der Diensterbringer Informationen über die Kundenzufriedenheit und damit Aussagen über die Güte seiner Dienstleistung. Gesammelte Informationen dieser Art führen zu Kundenprofilen, die einen kundenorientierten Service ermöglichen, die Güte der Dienstleistung aus Sicht des Kunden verbessern oder gezieltes Werben einer ausgewählten Klientel vorbereiten.
Weitere Informationen über das Modell der *Business Conversations* finden sich in [16, 26, 30, 33, 38].

Konversationsspezifikationen

Die *Business*-Konversationen stellen den kooperativen Charakter eines Informationssystems in den Vordergrund. Ein integraler Bestandteil des Modells ist das vereinbarte Kommunikationsprotokoll, die sogenannte Konversationsspezifikation, die die Interaktionen innerhalb eines Gesprächs festlegen. Das Protokoll muß beiden Gesprächspartnern bekannt sein und strikt eingehalten werden, um einen korrekten Ablauf der Konversation zu gewährleisten. Sprechakte werden demnach als Dialoge innerhalb von Konversationsspezifikationen gefaßt und mit der Intention eingesetzt, die Lücke zwischen abstrakten Gruppenkooperationsmodellen und den herkömmlichen Programmentwicklungsmethoden zu schließen. Sprechakte zwischen Kunde und Dienstleister werden formal als strukturierte Dialoge innerhalb von Konversationsspezifikationen formuliert. Zwei aufeinanderfolgende Sprechakte einer Konversation werden als Dialogschritt zusammengefaßt. Ein Dialog wird als hierarchisch strukturiertes Formular verstanden und besitzt einen eindeutigen Namen. Der Vorteil, Konversationen mit Hilfe von Metadaten (Konversationsspezifikationen) zu beschreiben, liegt in der Analogie von Datentypen und Daten. Typen bilden die Grundlage zur Konsistenzprüfung, denn nur durch sie werden eine Klassifikation, Generalisierung oder Parametrisierung erst möglich [4]. Ähnliche Aspekte sind in den Konversationsspezifikationen zu finden, die detailliert in [12, 38] beschrieben sind.

2.3 Überführung von EPK-Paaren zu Konversationsspezifikationen

Die Geschäftsprozeßspezifikation eines Dienstes beschreibt alle möglichen Ereignisse innerhalb eines Geschäftsvorganges sowie die Aktionen, die auf Ereignisse folgen. Im vorherigen Abschnitt wurde gezeigt, daß für das Zustandekommen einer Kooperation zwischen Kunde und Diensterbringer eine gemeinsame Konversationsspezifikation vorliegen muß. Ziel ist es, eine Geschäftsprozeßbeschreibung zu finden, die gleichermaßen als Modell zum Spezifizieren der Geschäftsprozesse, als auch später in der Laufzeitumgebung zur Ablaufsteuerung der jeweiligen Prozeßinstanzen zu verwenden ist. Fände man einen solche Beschreibungsform, wäre ein bruchloser Entwicklungsprozeß (Kapitel 3.3) von der Analysephase bis hin zur Implementierung und damit eine prozeßorientierte Entwicklung des gesamten Systems möglich.

In Kapitel 2.1 wurden die Ereignisgesteuerten Prozeßketten eingeführt. Ein Nachteil der EPK liegt in der inhärent redundanten Beschreibung des Prozeßablaufs. Eine EPK kann aus Sicht des Kunden oder aus Sicht des Diensterbringers beschrieben sein. Vergleicht man die EPK beider Sichten, d.h. untersucht man das EPK-Paar eines Prozesses, so zeigt sich, daß Aktionen der einen Sicht Ereignisse in der anderen sind und umgekehrt. Ist ein System gleichzeitig Kunde als auch Dienstleister eines bestimmten Dienstes, so liegen die Prozeßspezifikationen doppelt vor. Dieses ist immer der Fall, falls das System in der Rolle eines Dienstvermittlers (*broker*) auftritt. Redundante Informationen führen generell bei der Evolution von Daten oder Systemen zu Inkonsistenzen und sind tunlichst zu vermeiden.
Abbildung 2.5 zeigt in der linken Spalte einen möglichen Pfad der EPK aus Abbildung 2.3 und

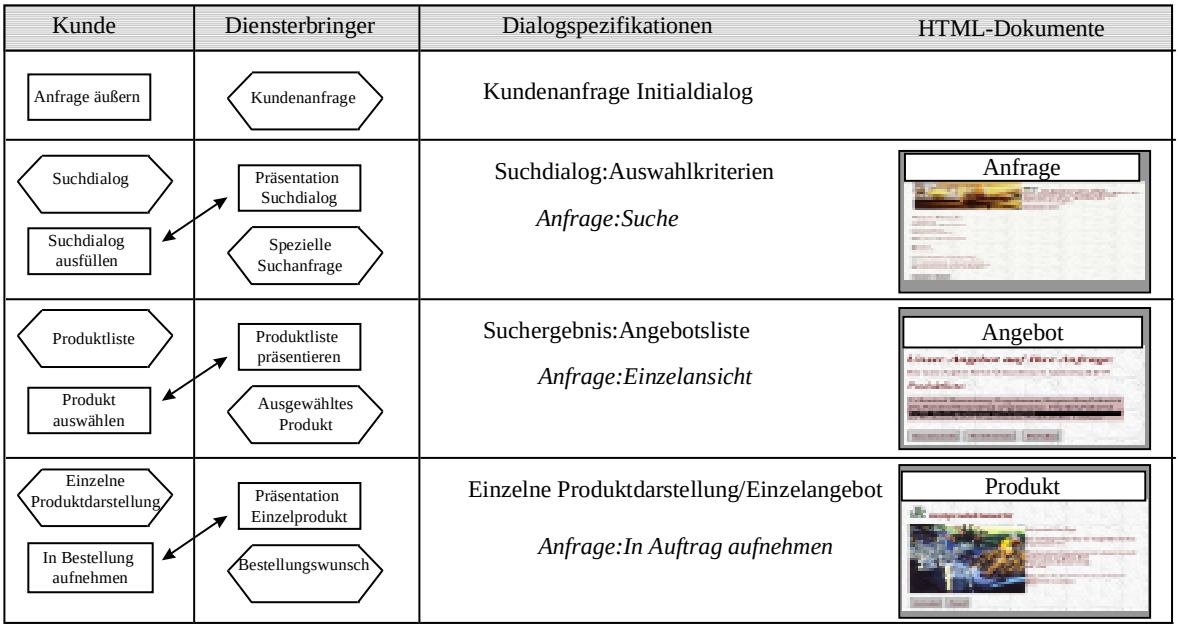

Abbildung 2.5: Überführung eines EPK-Paares zur Konversationsspezifikation

kann als Prozeßinstanz einer Zimmerreservierung aus Sicht des Kunden verstanden werden.

Die Spalte daneben zeigt die analoge Kette aus Sicht des Dienstleisters. Jeweils zwei Ereignis- und Aktionspaare lassen sich zu einer Dialogspezifikation zusammenfassen. Eine Dialogspezifikation beschreibt einzelne Dialogschritte einer Konversationsinstanz. Sie legen Inhalte sowie alle möglichen Folgedialoge fest und stehen in Analogie zu Datentypen und Daten innerhalb einer Programmiersprache.
Eine Konversationsspezifikation entspricht technisch gesehen einem endlichen Automaten [11], dessen Zustände durch Dialogspezifikationen festgelegt und dessen Zustandsübergänge durch Anfragen des Kunden (*requests*) in Verbindung mit den aktuellen Inhalten einer Dialoginstanz bestimmt sind. Der genaue Aufbau einer Konversationsspezifikationen ist in [16, 30, 38] in Form eines Entity-Relationship Diagramms wiedergegeben.
Geht man dazu über, Dialogspezifikationen in Form von HTML-Dokumenten zu modellieren, bekommt man eine prototypische statische Ablaufinstanz des betrachteten Prozesses. Diese Form der Darstellung eignet sich sehr gut, um interaktive Prozeßabläufe des zu entwickelnden Systems mit Anwendern zu besprechen, ohne daß nähere Kenntnisse über Prozeßnotationen nötig sind. Weitere Vorteile, die sich aus dieser Form der Geschäftsprozeßbeschreibung ergeben, werden im nächsten Kapitel erläutert.

Kapitel 3

Entwicklung komplexer Softwaresysteme

Bei der Entwicklung großer komplexer Softwaresysteme rücken die Qualitätsmerkmale wie Korrektheit, Robustheit, Erweiterbarkeit und Wiederverwendbarkeit eines Systems in den Vordergrund. Eine entscheidende Rolle zum Erreichen dieser Qualitätsmerkmale spielt die zur Implementierung verwendete Programmiersprache. Es zeigt sich schnell, daß Programmiersprachen, die keine Modulkonzepte besitzen, für komplexere Systemprogrammierungen ungeeignet sind [37]. Bemühungen der achtziger Jahre führten zu Programmiersprachen wie Ada, Modula und Eiffel, die mit ihren ausgereiften Modul- und Paketkonzepten die Entwicklung großer Softwaresysteme *programming in the large* gut unterstützen.
Bertrand Meyer stellte in [27] fünf Kriterien für die Bewertung von Entwurfsmethoden heraus. Sie beziehen sich auf die modulare Zerlegbarkeit, Kombinierbarkeit, Verständlichkeit, Beständigkeit und Geschütztheit. Aus diesen Kriterien folgert Meyer Prinzipien, die zum Erreichen sauberer Modularität beachtet werden müssen. Wichtig erscheinen dabei sprachliche Moduleinheiten mit schmalen, aber expliziten Schnittstellen (lose Kopplung) und das Geheimnisprinzip, das die Implementierungsdetails nach außen hin versteckt.
Eiffel ist eine objektorientierte Sprache und unterstützt neben den oben genannten Eigenschaften auch die Wiederverwendbarkeit durch Vererbung und Generizität polymorpher Funktionen. Meyer beschränkt sich bei seinem Vorgehensmodell auf eine systematische Programmentwicklung durch das Finden und Spezifizieren von Klassen sowie die Konsistenzerhaltung durch Zusicherungen, welche sich im wesentlichen auf ein Vertragskonzept abstützen.
Als weiterführende Vorgehensmodelle zur Softwareentwicklung, die sich auf das objektorientierte Paradigma von Programmiersprachen stützen, entwickelten sich Anfang der neunziger Jahre die objektorientierte Analyse [5], die Object Modelling Technique [32], und der *use-case driven approach* [13]. Diese Modelle beschreiben im wesentlichen einen iterativen und inkrementellen Entwicklungsprozeß von Softwaresystemen.

3.1 Allgemeines Vorgehensmodell

In dieser Arbeit wird ein geschäftsprozeßorientiertes Vorgehensmodell untersucht, das sich in den Grundzügen an Jacobsons *a use case driven approach* [13] anlehnt. In diesem Modell sind nicht die Anwendungsfälle Grundlage der Systementwicklung, sondern die Kernprozesse des zu entwickelnden Systems, aus denen sich dann die von Jacobson betrachteten Anwendungsfälle

herauskristallisieren. Ziel ist es, ein Modell zu finden, das den Entwicklungsprozeß gleichzeitig innerhalb der prozeßorientierten und der objektorientierten Sicht synergetisch und bruchlos über alle Entwicklungsphasen unterstützt.

3.1.1 Iterative und inkrementelle Softwareentwicklung

Charakteristisch für objektorientierte Entwicklungsprozesse ist das mehrmalige Durchlaufen einzelner Phasen und Wiederholen von gleichen oder sehr ähnlichen Tätigkeiten, die als Iteration bezeichnet wird. Dabei handelt es sich in der Regel um sehr fein granulare Tätigkeiten, die von Booch in der nach ihm benannten Methode [1] als Mikroprozesse bezeichnet werden. Der Durchlauf einer Iteration ist im Vergleich zur Projektdauer relativ kurz (ein Tag bis zu vier Wochen). Mit erneutem Durchlaufen bestimmter Phasen oder dem Rücksprung an bestimmte Stellen des Entwicklungsprozesses versucht man einzelne Ergebnisse zu verfeinern oder durch Verwerfen alter Ergebnisse neue zu entwickeln. Der Nachteil der rein iterativen Entwicklung liegt in der Gefahr, daß nach Durchlauf einer Iteration keine Gewähr dafür besteht, im Entwicklungsprozeß weiter fortgeschritten zu sein als vor dieser Iteration.
Die inkrementelle Systementwicklung versucht den obengenannten Nachteil des iterativen Ansatzes zu umgehen, indem explizit das Erreichen bestimmter Ziele, zum Beispiel die Vorgabe des ersten Inkrementes, formuliert werden. Das erste Inkrement legt den Entwicklungsumfang eines ersten Prototyps fest; er sollte schon einige wenige Kernprozesse unterstützen. Weitere Inkremente führen durch schrittweise Verfeinerung hin bis zur endgültigen Fertigstellung des geforderten Gesamtsystems.
Abbildung 3.1 zeigt die Aktivitäten des iterativen und inkrementellen Systementwurf und benennt die Phasen des objektorientierten Entwicklungsprozesses. Sie stimmen mit den von Jacobson benannten Phasen [14] überein und werden im nächsten Abschnitt genauer besprochen.

3.1.2 Objektorientierte Analyse und Entwurf

Der in Abbildung 3.1 skizzierte objektorientierte Entwicklungsprozeß entspricht weitestgehend dem Vorgehensmodell der *Unified Modelling Language* (UML) [2] [7]. Die UML ist ein werdender Standard zur objektorientierten Modellierung und eignet sich gut als Beschreibungsmittel für den Entwurf betrieblicher und kommerzieller Softwaresysteme und ist von Grady Booch, Ivar Jacobson und James Rumbaugh entwickelt worden.
Der Entwicklungsprozeß untergliedert sich in mehrere Phasen, die grob in Analyse, Entwurf und Implementierung eingeteilt werden. Innerhalb der Analysephase werden der Anforderungskatalog und das *Use-Case*-Modell erstellt, sowie die möglichen Akteure des Systems herausgearbeitet. In der Entwurfsphase werden die aus dem *Use-Case*-Modell herauskristallisierten Objekte in einem ersten Klassenmodell fixiert und ihre Beziehungen untereinander beschrieben. Dieses Klassenmodell ist idealisiert in dem Sinne, daß keine Einschränkungen bezüglich der Programmiersprache, Verteilung und Persistenz von Objekten gemacht werden. Diese Einschränkungen bestehen erst in späteren Klassenmodellen und werden erst dort mit berücksichtigt. Weiterhin werden in der Entwurfsphase Zustands-, Sequenz- und Interaktionsdiagramme von Objekten angefertigt, die das Zusammenspiel bzw. das Verhalten komplexer Objekte beschreiben. Näheres bezüglich der UML sind in den Kapiteln 5.3 und 6 zu finden. Festzuhalten gilt, daß jede Phase des Entwicklungsprozesses genau spezifizierte Ergebnisse hervorbringt, die in Form von Diagrammen, Modellen und Dokumenten beschrieben sind. Es

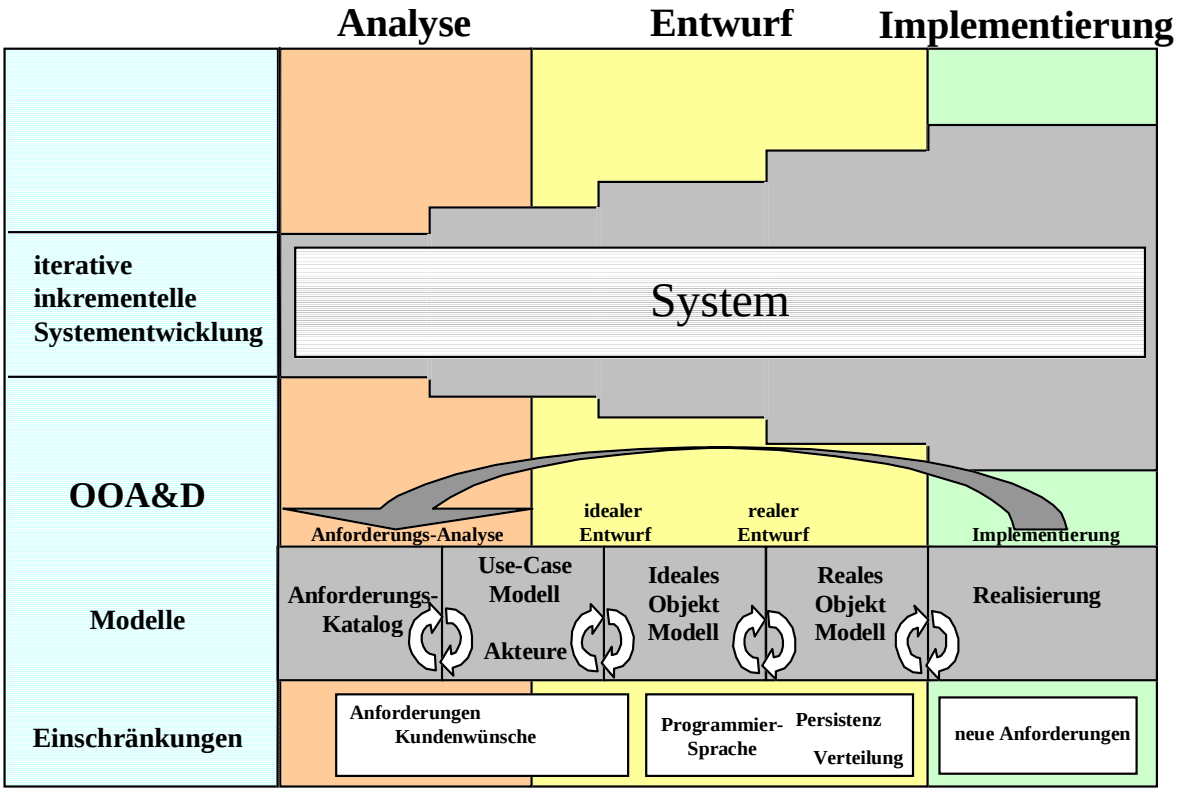

Abbildung 3.1: Iterative und inkrementelle Systementwicklung

zeigt sich, daß die UML einen wesentlichen Beitrag zur iterativen und inkrementellen Entwicklung von Softwaresystemen liefert.

3.1.3 Komponentenbildung

Wie einleitend erwähnt, ist die Entwicklung sehr großer Softwaresysteme nur zu bewältigen, falls das gesamte System in einzelne Komponenten zerlegbar ist und die eingesetzte Programmiersprache entsprechende unterstützende Mittel dazu liefert. Es gibt unterschiedliche Ansätze, Komponenten zu bilden.
Der intuitive Ansatz liegt in der funktionalen Unterteilung des Systems. Er liefert Funktionsgruppen, des weiteren auch Module genannt, die nach außen hin klar definierte Schnittstellen besitzen und von anderen Modulen benutzt werden. Dabei sind Abhängigkeiten zwischen einzelnen Modulen zu beachten. Module, die in einer binären „Benutzt-Beziehung“ stehen, d.h. Module, die sich gegenseitig verwenden, sollten besser innerhalb eines Moduls angesiedelt werden. Die UML stellt für diese Untergliederung das *Package*-Diagramm zur Verfügung. Die getrennt voneinander entwickelten Module werden nach Fertigstellung zusammengebunden und bilden das Laufzeitsystem in ihrer gesamten Funktionalität.
Ein weiterer gängiger Ansatz der Komponentenbildung ist die Schichtenarchitektur [35]. Innerhalb des Systems gibt es verschiedene Aufgaben, die meist in Schichten, wie z.B. der

Persistenzschicht, Kommunikationsschicht, Anwendungsschicht und der Darstellungsschicht, die der Benutzungsoberfläche entspricht, gekapselt werden. Einzelnen Schichten sind meist wiederum in funktionale Komponenten untergliedert.

Autonome Softwaresysteme

Aspekte des Investitionsschutzes führen zu neueren Ansätzen, Systeme zu untergliedern bzw. untereinander zu koppeln. Ansätze, in denen Wiederverwendung von bestehenden Softwaresystemen Rechnung getragen wird, führen zu neuen Kooperationsmodellen, in denen Systeme ihre Aufgaben eigenverantwortlich wahrnehmen und durch Kooperation mit anderen auf ein gemeinsames Ziel hinarbeiten. Das hier betrachte Kooperationsmodell der *Business Conversations* wurde in [16, 26] entwickelt und in [38] in der Programmiersprache Tycoon-2 implementiert und bildet die Basis des in dieser Arbeit vorgeschlagenen Architekturmodells für die Erweiterbarkeit bestehender Systeme durch die Entwicklung sogenannter Satellitensysteme. Neue Funktionalität wird dabei innerhalb des Satellitensystem angegliedert, ohne das zu erweiternde System verändern zu müssen. Dieser Ansatz eignet sich besonders gut, wenn aufgrund von Veränderungen entstehende Folgeänderungen nicht zu überblicken und deren Kosten kaum abzuschätzen sind.

Softwareagenten

Autonome Softwaresysteme führen auf den Begriff des Softwareagenten. Agenten nehmen aufgrund ihrer Wissensbasis ihnen zugeteilte Aufgaben wahr. In der Regel erbringen sie anderen Agenten einen Dienst oder fordern Dienste anderer an. Dabei ist es im Modell nicht wesentlich, ob Agenten in Form eines maschinellen Software-Systems oder in Form eines menschlichen Kooperationspartners auftreten. Gemeint ist die Kooperationsform, die durch Kommunikation untereinander gewährleistet wird. Die Kommunikation findet durch das Austauschen strukturierter Dialoge statt (Kapitel 2.2), die als Informationsträger fungieren. Eine Kooperation ist wiederum in verschiedenen Modalitäten möglich und hängt nicht vom Kommunikationsmedium ab. Agenten können mobil sein und sind dadurch nicht an ihren Ort gebunden. Diese Eigenschaft führt zu mobilen Softwareagenten, die zu ihren Kommunikationspartnern migrieren und die Kooperation vor Ort durchführen. Migrationsfähige Agenten sind Gegenstand der Forschung [16, 21] und werden zu neuen Modellen und Architekturen von Workflowsystemen führen.

3.2 Geschäftsprozesse im SAP R/3

Die vom SAP R/3-System unterstützten Geschäftsprozesse werden in Form von Ereignisgesteuerten Prozeßketten im sogenannten Referenzmodell gehalten und geben dort dem Anwender einen ersten groben Überblick von der Vielfalt der implementierten Prozesse. Das Referenzmodell dient zum Einstieg eines jeden Anwenders in die Modellwelt des R/3-Systems. Er kann von dort aus zum Organisationsmodell, Datenmodell oder zum Customizing navigieren, um kundenspezifische Einstellungen vorzunehmen.

3.2.1 Implementierung und „customizing“ von Geschäftsprozessen

Das R/3-Customizing ermöglicht dem Anwender die individuelle Anpassung der R/3-Prozesse an das eigene Unternehmen. Er kann gemäß seinen Anforderungen Zweige einer EPK auswählen, parametrisieren oder ganz ausblenden. Er muß sich jedoch im vorgedachten Lösungsraum des R/3-Systems bewegen, um die Konsistenz nach einem Versionswechsel zu wahren.
Der eigentliche Prozeßablauf erfolgt transaktional. Jede Aktion einer EPK hat eine zugeordnete Transaktionsnummer, unter der die entsprechende Aktion implementiert und nach Aufruf ausgeführt wird. Jede Transaktion wiederum besteht aus einem oder mehreren dynamischen Programmen, den sogenannten „DynPros“, die Benutzerinteraktionen unterstützen. Dabei gibt jedes „DynPro“ einen *okcode* zurück, an dem erkannt werden kann, ob die Benutzereingabe erfolgreich abgeschlossen wurde. Nähere Einzelheiten dazu finden sich in [23, 17, 41].
Ein Geschäftsprozeß ist als eine Ausführungsfolge aufeinanderfolgenden Transaktionen realisiert. Jede einzelne Transaktion greift direkt auf die Daten des Systems zu und manipuliert diese in entsprechender Weise. Transaktionen genügen den ACID-Anforderungen [35] und sind im System festkodiert, was die obengenannten Einschränkungen des *customizings* erklärt. Beim Ausblenden ausgewählter Aktionen einer EPK wird nur die Ablaufkette der korrespondierenden Transaktionen verändert und gegebenenfalls der mögliche Verzweigungsgrad von Transaktionen verkleinert, wenn bestimmte Zweige einer EPK nicht ausgeführt werden sollen.
EPK im Referenzmodell besitzen demnach bis auf die Navigationsmöglichkeiten zum R/3-Customizing keinen weiteren Einfluß auf die Implementierung der Geschäftsprozesse. Sie existieren daher als alleinstehendes Modell neben ihrer eigentlichen physikalischen Realisierung.

3.2.2 Semantische Lücken im Entwicklungsprozeß

Verfolgte man den Entwicklungsprozeß eines neu zu unterstützenden Geschäftsprozesses innerhalb des SAP R/3-Systems, so wäre der im folgenden beschriebene Ablauf denkbar: Nach dem Entschluß, einen neuen Geschäftsprozeß in das SAP R/3-System aufzunehmen, beschriebe man innerhalb der Analysephase den gesamten Prozeß als Ereignisgesteuerte Prozeßkette. Dabei würde man alle möglichen Spielarten berücksichtigen, wie sie im späteren Standard gebraucht werden könnten, und erhielte den beim *customizing* vorgegebenen und im vorherigen Abschnitt erwähnten Lösungsraum.
Ausspezifizierte EPK oder Teilketten würden nach ihrer Fertigstellung an die jeweils zuständige Entwicklungsgruppe gereicht, um dort programmiert zu werden.

Der Wiederverwendbarkeitsgrad einer solchen EPK ist eher gering. Sie dient lediglich dem Programmierer als Vorlage und später für den Anwender als Navigationshilfe beim Systemeinstieg. Dieser Nachteil zeigt sich im Entwicklungsprozeß als semantische Lücke (Abbildung 3.2) und führt zu deutlichen Mehraufwand, da ein Geschäftsprozeß zweimal mit unterschiedlichen Mitteln beschrieben werden muß.

3.3 Wiederverwendbarkeit und Lokalität von Geschäftsprozeßspezifikationen

Der im vorherigen Abschnitt aufgeführte Bruch im Entwicklungsprozeß steht im Widerspruch zur Idee der inkrementellen Systementwicklung, da nur ein sehr geringer Anteil des Prozeßwissens innerhalb einer EPK direkt in die Implementierung dieser Prozesse fließt. Die Erhöhung

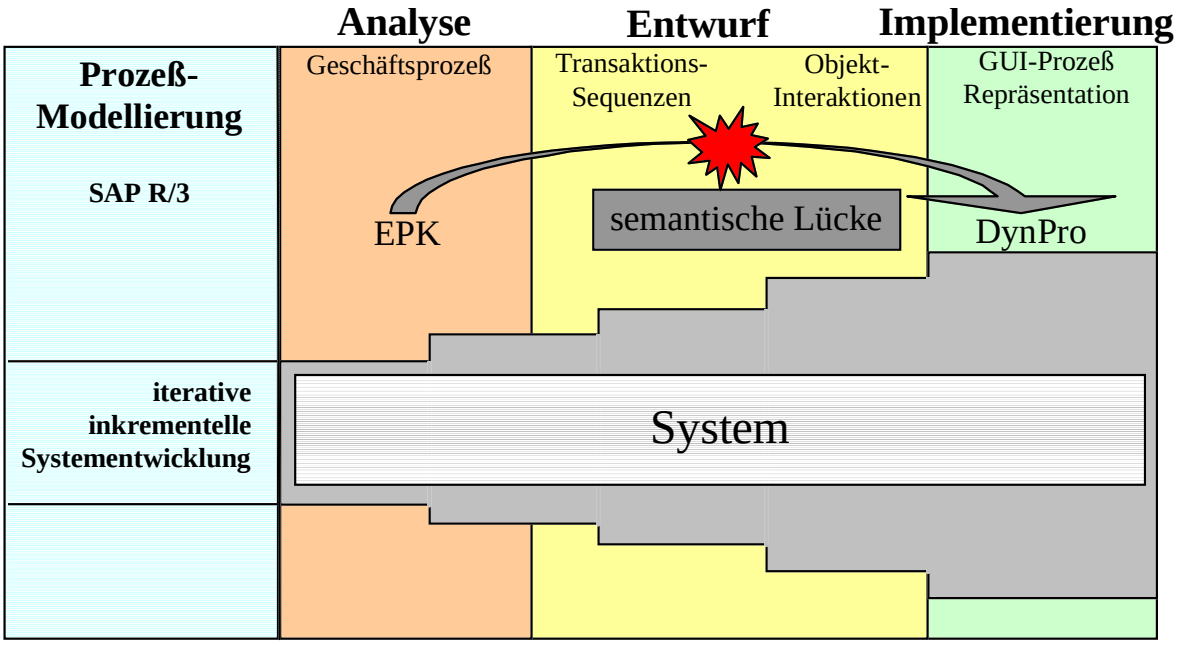

Abbildung 3.2: Semantische Lücke im Entwicklungsprozeß

der Wiederverwendbarkeit von Ereignisgesteuerten Prozeßketten könnte zu einer stringenten prozeßorientierten Entwicklungsmethode führen, die einerseits die Produktivität des Entwicklungsprozesses erhöht und andererseits ein System hervorbringt, das auf evolutionäre Veränderungen von Geschäftsprozessen sehr flexibel reagieren könnte.
Ziel ist es, eine Methode zu finden, die den Entwicklungsprozeß bei Vorlage der Geschäftsprozeßspezifikation wesentlich beschleunigt. So könnte zum Beispiel die gesamte Ablauflogik eines Prozesses samt Eingabe- und Ausgabeparametern aus der EPK gewonnen werden.

Ein bruchloser Softwareentwicklungsprozeß

Eine Methode zum Systementwurf wird nun vorgeschlagen. Hierbei soll sowohl der prozeßorientierte als auch der objektorientierte Ansatz eines Entwicklungsprozesses berücksichtigt werden, sowie das Schließen semantischer Lücken auf Prozeßebene erreicht werden. Die betrachteten Ansätze ergänzen sich synergetisch, was in Abbildung 3.3 symbolisch durch Brücken angedeutet wird. Zusätzlich wird ein bruchloser Übergang zwischen Analyse, Entwurf und Implementierung gefordert, der den gesamten Entwicklungsprozeß zu einer Einheit verschmelzen läßt. Eine daraus abgeleitete mögliche Methode des Systementwurfs wird im folgenden kurz beschrieben und weiter am Prozeß der Hotelzimmer-Reservierung und der Entwicklung eines Prototyps ausführlich in den Kapiteln 5 und 6 untersucht.

Grundlage der gesamten Entwicklung bildet der eigentliche Geschäftsprozeß. Er wird in Form von statischen HTML-Seiten beschrieben und spezifiziert jeden einzelnen Schritt des Prozesses. Die Folge von einzelnen HTML-Seiten kann mit dem Konzept der dynamischen Programmkette, die eine Prozeßinstanz im SAP R/3-System darstellt, verglichen werden.

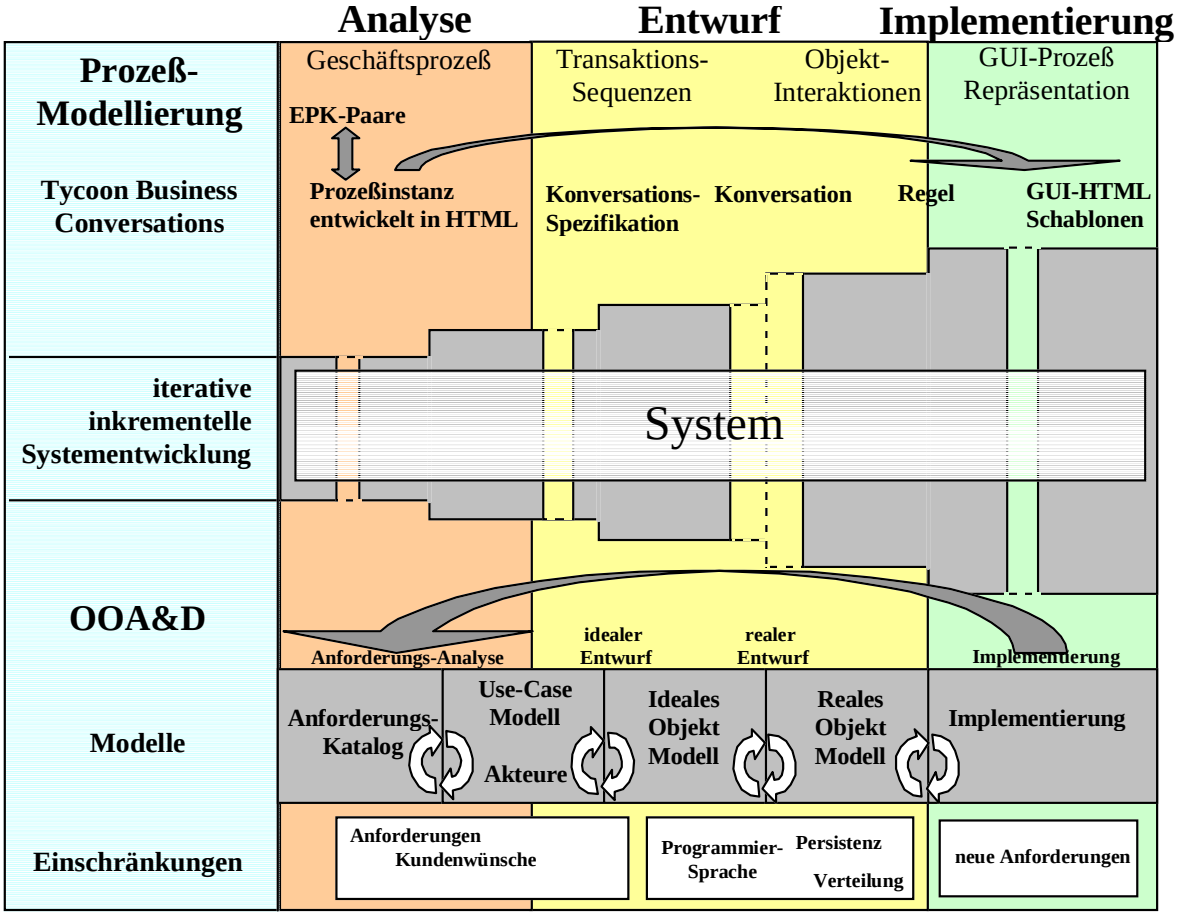

Abbildung 3.3: Ein bruchloser Entwicklungsprozeß

HTML-Seiten spezifizieren Inhalte, die die nötigen Parameter zur Prozeßausführung liefern, und Prozeßverzweigungen, aus denen sich später die Ablaufsteuerung des Gesamtprozesses generieren läßt. Auf Seite der objektorientierten Analyse können ausgehend von einer so vorliegenden Prozeßbeschreibung die Geschäftsobjekte, Anwendungsfälle und Akteure des Systems in einem Anforderungskatalog und dem *use case*-Modell fixiert werden.
Beim Übergang in die Designphase werden aus dem *use case*-Modell die Objektmodelle entwickelt. Sie beschreiben Geschäftsobjekte und deren Beziehungen unter Berücksichtigung vorgegebener Einschränkungen. Auf der Prozeßebene benötigt man innerhalb der Entwicklungsphase ein Werkzeug, das HTML-Seiten in eine dem Laufzeitsystem bekannte Ablaufstruktur, nämlich der Konversationsspezifikation, konvertiert. Außerdem werden die mit den Dialogspezifikationen korrespondierenden HTML-Schablonen in Form von Datenstrukturen im Laufzeitsystem gespeichert. In der eigentlichen Implementierungsphase werden die vom Prozeß benutzen Geschäftsobjekte realisiert. Die eigentliche Kopplung des Prozesses mit den Objekten findet über sogenannte Regeln statt, die die einzige Kopplung zwischen Prozeß und Geschäftsobjekten darstellt. Der Vorteil dieser losen Kopplung von Prozeßspezifikation und Geschäftsobjekten führt zu einer leichteren Anpassung des Systems bei evolutionären

Veränderungen von Geschäftsprozessen. Der Ablauf des eigentlichen Geschäftsprozesses und die mit ihm verbundene Anwenderinteraktion werden durch Dialoge der Konversationsinstanz mit Hilfe der vorliegenden HTML-Schablonen visualisiert. Dabei werden die jeweils aktuellen Inhalte eines Dialoges über Regeln von den Geschäftsobjekten geholt und unter Verwendung der Schablonen zu dynamisch generierten HTML-Seiten zusammengesetzt. Sie können so von jedem gängigen WWW-Browser angezeigt werden.

Erweiterbarkeit von Prozeßspezifikationen

Ein weiterer Vorteil, die Ablaufstruktur eines Prozesses in Form einer dynamisch erzeugten Datenstruktur vorliegen zu haben, liegt in der Möglichkeit, Prozesse und damit die angebotenen Dienste eines Systems flexibel zu erweitern. Wie in Abschnitt 2.2 erläutert, kann es nur zu einer Kooperation zwischen Kunde und Dienstleister kommen, falls beide über eine kompatible Konversationsspezifikation verfügen. Die Konformität kann indes leicht durch Überprüfung der Struktur beider vorliegenden Spezifikationen bestimmt werden. Die Übergabe der Konversationsspezifikation erfolgt mittels der erzeugten Datenstruktur als Objekt erster Klasse. Eine Kooperation kommt demnach zustande, falls die Konversationsspezifikation des Diensterbringers spezieller als die des Kunden ist. In Analogie zu Programmiersprachen muß die Spezifikation des Dienstleisters Subtyp von der des Kunden sein. Die Spezifikation kann überall dort verwendet werden, wo einer ihrer Supertypen erwartet wird.
Bietet demnach ein Dienstleister eine erweiterte Spezifikation seines Dienstes an, so können bisherige Kunden den alten Service beziehen, ohne sich jedesmal auf die aktuelle Dienstbeschreibung des Diensterbringers stützen zu müssen. Andererseits kann der Kunde durch Generalisierung seiner Konversationsspezifikation eine Auswahl der angebotenen Dienste treffen, was dem *customizing* von Prozessen im SAP R/3-System gleichkommt. Regeln, die festlegen, wann eine Konversationsspezifikation in einer Subtypbeziehung zu einer anderen steht, sind in [38] nachzulesen.

Kapitel 4

Anforderungskatalog Zimmer-Reservierungssystem

In den folgenden vier Kapiteln wird der in Abschnitt 3.3 skizzierte bruchlose Entwicklungsprozeß am Beispiel einer prototypischen Realisierung eines Hotelzimmer-Reservierungssystems durchlaufen. Die Entwicklung beginnt mit der Aufstelltung von Anforderungen an das System, die mit verschiedenen Domänenexperten in Fachgesprächen herausgearbeitet und während der gesamten Analysephase schrittweise verfeinert wurden. Die Komplexität eines Reservierungssystems versucht Abbildung 4.1 an Hand aller erforderlichen Komponenten aufzuführen. Diese Komponenten wurden während der Analysephase gefunden und deren Funktionalität im Anforderungskatalog fixiert. Gut zu erkennen sind die Schnittstellen des Systems. So sind hier die typischen Interaktionsmuster zwischen Komponenten eines Workflow-Systems zu erkennen, in dem menschliche als auch maschinelle Agenten auf ein gemeinsames Ziel hinarbeiten. So gibt im letzten Fall das System dem Wartungspersonal Anweisungen, welche Zimmer als nächstes zu reinigen sind. Das System wiederum bezieht seine Informationen von der Rezeption, wo sich der Gast beim *check-out* abmeldet. Als erstes Inkrement des Entwicklungsprozesses wurde die zentrale Komponente „Reservierung“ festgelegt. Der Umfang der eigentlichen Analyse erstreckte sich wie angedeutet weit über die allgemeinen Anforderungen einer Zimmerreservierung. Sie umfaßten zusätzlich das Folio-Management, das sich mit der Erfassung und Abrechnung ausgeführter Dienstleistungen am Kunden befaßt, sowie die Nuancen der Einzel- und Gruppenreservierungen, Probleme beim *check-in*, Ansätze für Überbuchungs- und Wartungsstrategien auf Ressourcen und die Anbindung von sogenannten Verkaufsstationen (POS-Systemen). Sie werden innerhalb des ersten Inkrementes nicht näher betrachtet.
Der in der Analyse entstandene Anforderungskatalog ist auszugsweise im Anhang A wiedergegeben und skizziert im wesentlichen eine Systemvision mit ihren Schlüsselabstraktionen, Verantwortlichkeiten und Beziehungen zwischen den grob abgesteckten Komponenten.
Im folgenden Abschnitt werden die Schlüsselabstraktionen im Kontext des Hotelbetriebs vorgestellt und bestehende Beziehungen zwischen ihnen erläutert.

4.1 Reservierungssystem

Mit einem Reservierungssystem können die von einem Hotel angebotenen Dienstleistungen für beliebige Zeiträume gebucht werden. Dabei werden Dienstleistungen von einem Hotel als Produkte angeboten und als solche vertrieben. Eine neue Anforderung an ein Reservierungs-

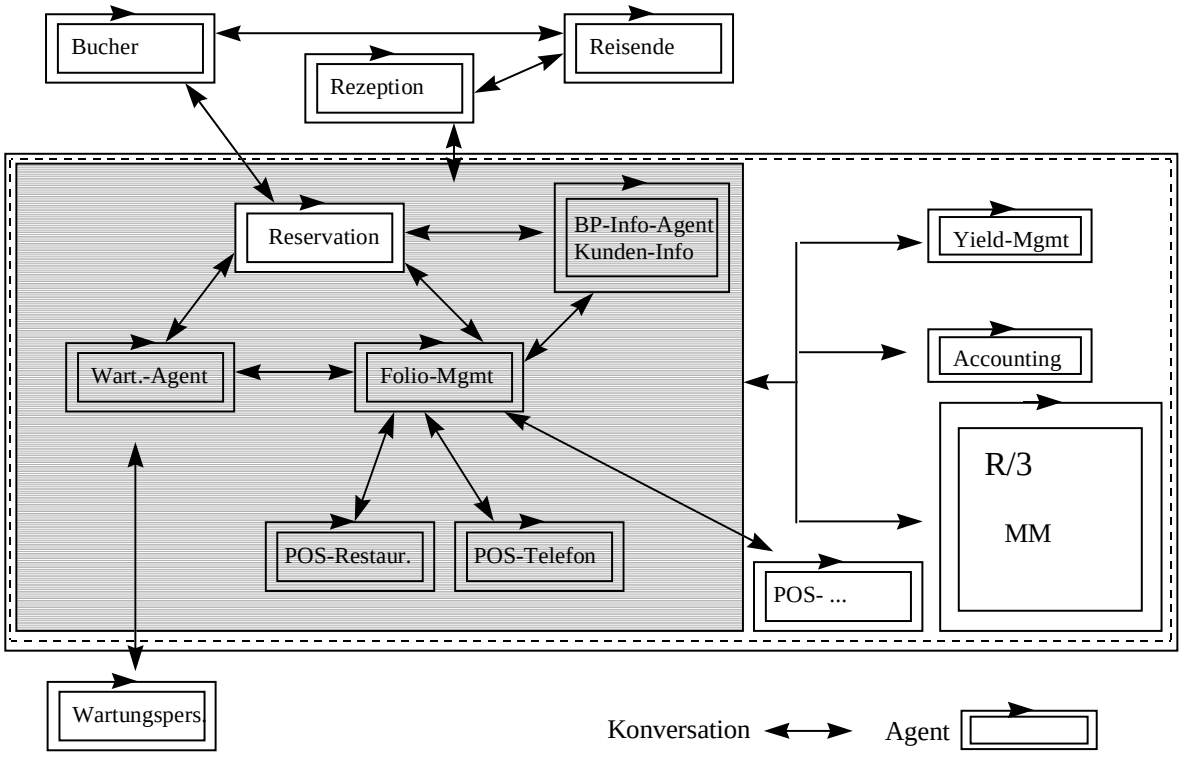

Abbildung 4.1: Komponenten eines Hotel-Softwaresystems (Property-Management-System)

system ist die Verarbeitung von Reservierungsaufträgen aus dem Internet in der Form, daß Anfragen aus dem Netz unmittelbar an das System weitergeleitet und beantwortet werden sollten. Die gesamte Reservierungsfunktionalität muß selbstverständlich auch an der Rezeption zur Verfügung stehen, wo telefonische und schriftliche Anfragen beantwortet werden.
Anfragen beziehen sich auf die Produktpalette eines Hotels und werden in Form von Wünschen, die einer Aneinanderreihung von Suchattributen auf Produkten gleichkommt, übermittelt. Kein Kunde verfügt über Kenntnisse der hotelspezifischen Produktpalette. Kundenanfragen werden vom System beantwortet, indem zutreffende Produkte angeboten werden, aus denen der Kunde seinen endgültigen Reservierungswunsch ableitet. Anschließend kommt es zu einer Produktreservierung, die mit einer Buchungsbestätigung quittiert wird.
Die Güte einer Buchung hängt von den Sicherheiten ab, die der Kunde seiner Reservierung beilegt. Ist ein Kunde dem System bekannt oder liegen Informationen über seine Kreditkartennummer vor, so kann dieser Buchung genauso viel Wert beigemessen werden, wie bei einer schriftlichen oder telefonischen Buchung mit zusätzlicher Baranzahlung.

4.2 Produkte und Pakete

Produkte geben den einzelnen Dienstleistungen eines Hotels ihren Namen und ihren Preis, der im allgemeinen vom Kunden, von der Saison und von der Buchungsdauer abhängt. Produkte

spiegeln das Angebot eines Hotels wieder und können in verschiedenen Variationen auftreten. Ein Produkt kann eine einfache Übernachtung im Doppelzimmer darstellen, aber auch ein Wochenendpaket, das zwei Übernachtungen beinhaltet, oder auch einen Opernbesuch, eine Stadtrundfahrt, das Frühstücksbuffet, Abendbrot oder Tennisstunden. Pakete setzen sich aus einzelnen Produkten zusammen und tragen den Wünschen unterschiedlicher Gäste Rechnung. Sie erscheinen dem Gast als preiswert und stellen eine echte Alternative neben der aufwendigen Einzelbuchung aller Teilleistungen dar.
Beim Verkauf von Paketen wird der Gast weitestgehend mit dem Ziel, das komplette Spektrum seiner Wünsche abdecken zu können, an das Hotel gebunden aber auch, um zusätzlich versteckte Gewinne, sogenannte Paketgewinne, einzufahren, die anfallen, falls ein Gast die in einem Paket inbegriffenen und dadurch bezahlten Leistungen nicht in Anspruch nehmen kann oder will. Beide Punkte spielen bei der Produkt - und Imagepflege eines Hotels eine wichtige Rolle und legen im wesentlichen auch seine Zielgruppe fest.
Die eigentliche Preispolitik kann auf Grund der heute im Einsatz befindlichen Programme nur mittels einer breiten Produktpalette abgebildet werden. Produkte mit identischen Leistungen und unterschiedlichen Preisen führen zu redundanten Daten und dadurch oftmals zu einer unüberschaubaren Produktvielfalt. So wird zum Beispiel unterschieden zwischen der einfachen Übernachtung eines wiederkehrenden Geschäftsreisenden, der des Urlaubers, der seine Buchung über ein Reiseunternehmen hat vornehmen lassen, oder der des Tagesgastes, der abends nach einer Unterkunft sucht. In jedem Fall wird eine Doppelzimmer-Übernachtung verkauft, dessen Preis sich meist erheblich, aber nicht notwendig von den anderen unterscheidet. Neben dem Nachteil des Mehraufwandes bei der Produktpflege entsteht der Vorteil, dem Kunden ein Produkt zu verkaufen, das ihn anspricht. Dem Geschäftsmann wird deshalb das „Business-Paket“, dem Reisenden das „Wochenend-Spezial-Paket“ und dem Tagesgast die einfache „Übernachtung im Doppelzimmer“ verkauft.
Produkte beziehen sich auf Ressourcen eines Hotels, die im folgenden Abschnitt beschrieben werden.

4.3 Ressourcen

Ressourcen sind die physikalischen Objekte eines Hotels, deren Bestand geführt wird. So stellt zum Beispiel jedes von der Ausstattung her identische Doppelzimmer eine Ressource dar, die jedem Produkt, das in seiner Leistung eine Doppelzimmer-Übernachtung enthält, zugeordnet ist. Friseurtermine, Tennisplätze, Trainerstunden, Konferenzsäle und alle anderen zeitgebundenen Dienstleistungen fallen in dieselbe Ressourcenklasse. Man kann diese Ressourcen als „erneuerbar“ ansehen, da sie nach Inanspruchnahme durch Wartungskräfte wiederhergestellt oder durch Personal erbracht werden können. Im Gegensatz dazu stehen die „verbrauchbaren“ Ressourcen, für die zusätzlich ein Lager mit logistischen und bestelltechnischen Strategien geführt werden muß. Hierunter fallen Ressourcen wie Getränke, Speisen, Waschutensilien usw.
Unverbrauchbare Ressourcen hingegen sind immer im Hotel implizit vorhanden, obwohl sich diese genaugenommen aus verbrauchbaren Ressourcen zusammensetzen. So wird zum Beispiel das Frühstück im Paket „*Bed & Breakfast*“ als eine solche geführt. Bei einer Reservierung darf die Anzahl der vorhandenen Brötchen nicht ausschlaggebend für die Verfügbarkeit eines Produktes sein. Die Verfügbarkeit dieses Produktes hängt rein von der Verfügbarkeit der Zimmer ab. Neben diesen Ressourcen können noch künstliche Ressourcen eingeführt werden, die die

Verfügbarkeit von Ressourcen gezielt beschränken. Das sogenannte Yield-Management befaßt sich mit der Gewinnmaximierung mittels optimaler Reservierungsstrategien. So könnte zum Beispiel ein Zimmer zu einem späteren Zeitpunkt bei zu erwartender hoher Nachfrage einen viel besseren Preis erzielen, als bei einer lange vorher durchgeführten Reservierung. Ziel eines Hotels ist die maximale Auslastung zu möglichst hohen Preisen.

4.4 Verfügbarkeit von Produkten

Aus dem vorherigen Abschnitt wurde ersichtlich, daß sich die Produktverfügbarkeit aus den zugeordneten Ressourcen errechnet. Der Kunde stellt seine Anfrage in Form von Wünschen, die einer Attributbeschreibung des Produktes gleichkommt. Fragt er zum Beispiel nach einem Nichtraucher-Doppelzimmer in ruhiger Lage mit Seeblick und einem überdurchschnittlich großen Bett, so kommt nur eine Teilmenge aller Zimmer des Hotels in Frage. Das System muß anhand dieser Attribute jedes zutreffende Produkt bestimmen und deren Verfügbarkeit über die zugeordneten Ressource berechnen. Produkte, deren Verfügbarkeit größer Null ist, werden in einem Angebot zusammengestellt, aus dem der Gast seinen Buchungswunsch ableitet.
Bei der Berechnung der Produktverfügbarkeit kommt es auf die zugrundeliegende Ressource an. Verbrauchbare Ressourcen besitzen eine inhärente Verfügbarkeit, nämlich die Anzahl der auf Lager liegenden Produkte, bei zeitgebundenen Ressourcen hingegen berechnet sich die Verfügbarkeit aus der gesamten Anzahl der im Hotel zur Verfügung stehenden und gleichzeitig benutzbaren Zimmern abzüglich der reservierten und belegten. Dabei ist zu beachten, daß Zimmer durch temporäre Schäden, zum Beispiel durch eine verstopfte Toilette, nicht immer durchgehend verfügbar sein könnten.
Konferenzzimmer, Tennis- und Squashplätze werden stundenweise vermietet und können wie Hotelzimmer mit zeitbezogenen Ressourcen verwaltet werden, nur setzt sich in diesem Fall der Mindestbuchungszeitraum von Tagen auf Stunden herab. Selbst Friseurtermine fielen in diese Ressourcekategorie, würde deren Verfügbarkeit von der täglichen Personalbesetzung und der maximalen Kapazität des Salons als oberste Schranke bestimmt.

4.5 Geschäftspartner

Die wichtigsten Informationen eines Hotels sind Daten über Geschäftspartner. Zu den Geschäftspartner gehören nicht nur die Hotelgäste, sondern auch Firmen, Reiseagenturen, Kontaktpersonen innerhalb von Firmen, sowie auch Lieferanten und eigene Mitarbeiter. Kundenspezifische Daten werden in der Regel dazu benutzt, dem Kunden eine individuelle Aufmachung seines Aufenthalts anzubieten. Dazu sind Gewohnheiten, Vorlieben und Wünsche des Gastes aufzunehmen. Im gezielten Werben liegt eine weitere wichtige Möglichkeit, diese Informationen vorteilhaft auszunutzen. Das Pflegen von kompletten Gasthistorien gilt als Mindestanforderung an ein System. Allgemein wird ein Geschäftspartner eindeutig durch eine Firma oder eine Person mit Namen und einer dazugehörigen Anschrift spezifiziert. Er trägt wesentlich zum Erlös des Hotels bei und kann spezielle Produktkonditionen besitzen. Ein Geschäftspartner tritt innerhalb des Systems in verschiedenen Rollen auf und nimmt in Bezug auf die jeweilige Rolle unterschiedliche Aufgaben wahr.
An den Rollen, die ein Geschäftspartner innerhalb des Systems einnehmen kann, hängen wichtige Informationen zur Erledigung rollenspezifischer Aufgaben. So werden die vertraglichen Zahlungskonditionen eines Gastes, dessen Bankverbindung, Kreditkartennummern und Bo-

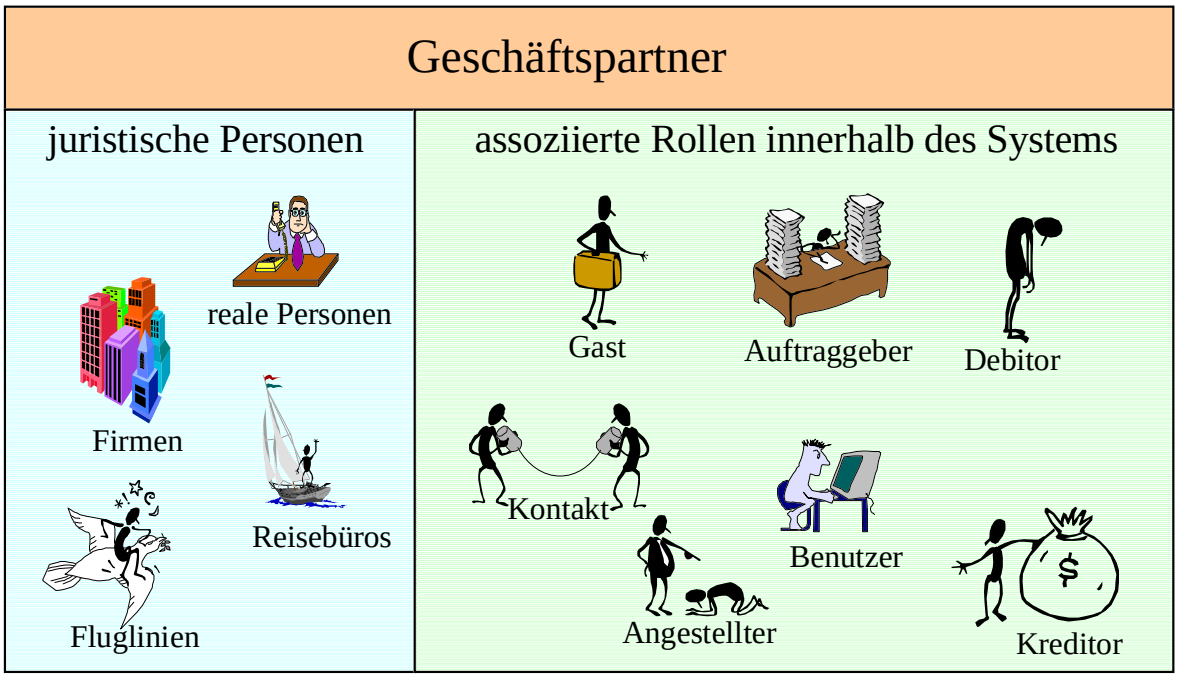

Abbildung 4.2: Geschäftspartner und assoziierte Rollen innerhalb des Reservierungssystems

nität innerhalb seiner Rolle als Debitor geführt. Lieferanten hingegen treten in der Rolle eines Kreditors auf.
Ein Geschäftspartner kann gleichzeitig mehrere Rollen einnehmen, die nicht notwendigerweise in Beziehung zueinander stehen müssen. Abbildung 4.2 zeigt eine Reihe von unterschiedlichen Rollen, die Geschäftspartner innerhalb des Systems einnehmen könnten.

Kapitel 5

Analyse

In diesem Kapitel wird die Analysephase des in Kapitel 3.3 vorgestellten bruchlosen Entwicklungsprozesses beschrieben. Sie beginnt mit der Auswertung der Kundenwünsche und Kundenanforderungen. Diese Anforderungen sind in der Regel sehr schwammig und unpräzise formuliert. Es ist daher sinnvoll, Kernprozesse des Systems mittels statischer Dialoginstanzen zu visualisieren, um einen prototypischen Prozeßablauf mit dem Kunden besprechen zu können. Anforderungen werden dabei schnell präziser und spiegeln das Prozeßwissen des Kunden wider. Als Ergebnis eines ersten Zyklus erhält man den im vorherigen Kapitel erarbeiteten und im Anhang A wiedergegebene Anforderungskatalog und die im folgenden vorgestellte statische Prozeßinstanz einer Zimmerreservierung, die vorzugsweise in HTML zu realisieren ist. In einem zweiten Schritt werden aus der Prozeßinstanz die einzelnen Akteure und *use cases* des Systems abgeleitet. Aus den *use cases* wiederum lassen sich *Business*-Objekte herauskristallisieren, die letztlich zu einem ersten Klassendiagramm zusammengefaßt werden. Mit diesem Klassendiagramm beginnt die Designphase des untersuchten Entwicklungsprozesses. Sie wird im nächsten Kapitel beschrieben.

5.1 Identifikation von Akteuren und Konversationen

Ein typischer Kernprozeß innerhalb eines Hotels ist der Besuch eines Gastes. Dieser Prozeß berührt im wesentlichen alle Teilprozesse, die im Hotel ablaufen könnten. Er beginnt mit der Anfrage an der Rezeption. Nach der Verhandlung über erstellte Angebote kommt es zu einer Übereinkunft, die der Reservierung einer Dienstleistung gleichkommt. Der eigentliche Teilprozeß der Reservierung ist an dieser Stelle schon abgeschlossen. Nach der Buchung erfolgt die Anreise des Gastes. Der Gast meldet sich an und beginnt mit der Zimmerzuteilung seinen Aufenthalt. Innerhalb des Hotels nimmt er weitere Dienstleistungen in Anspruch, die zum Beispiel von den im vorherigen Kapitel erwähnten Verkaufsstationen erbracht werden. Verbindlichkeiten werden dem Gast auf die Zimmerrechnung hinzugebucht. Vor der Abreise werden beim *check out* die monetären Angelegenheiten geklärt und Feedback gegeben, das in Form eines Trinkgeldes, einer Beschwerde oder einer Weiterempfehlung vom Gast geäußert wird. Viele andere Formen des Feedbacks sind denkbar. Das Hotel sollte an dieser Stelle die Möglichkeit nutzen, viel vom Kunden zu lernen, um bei seinem nächsten Besuch einen noch besseren Service anbieten zu können. Der hier beschriebene Prozeß wird in Abbildung 5.1 als sogenannte *action workflow loop* dargestellt [3, 24]. Sie visualisiert auf gleiche Art und Weise

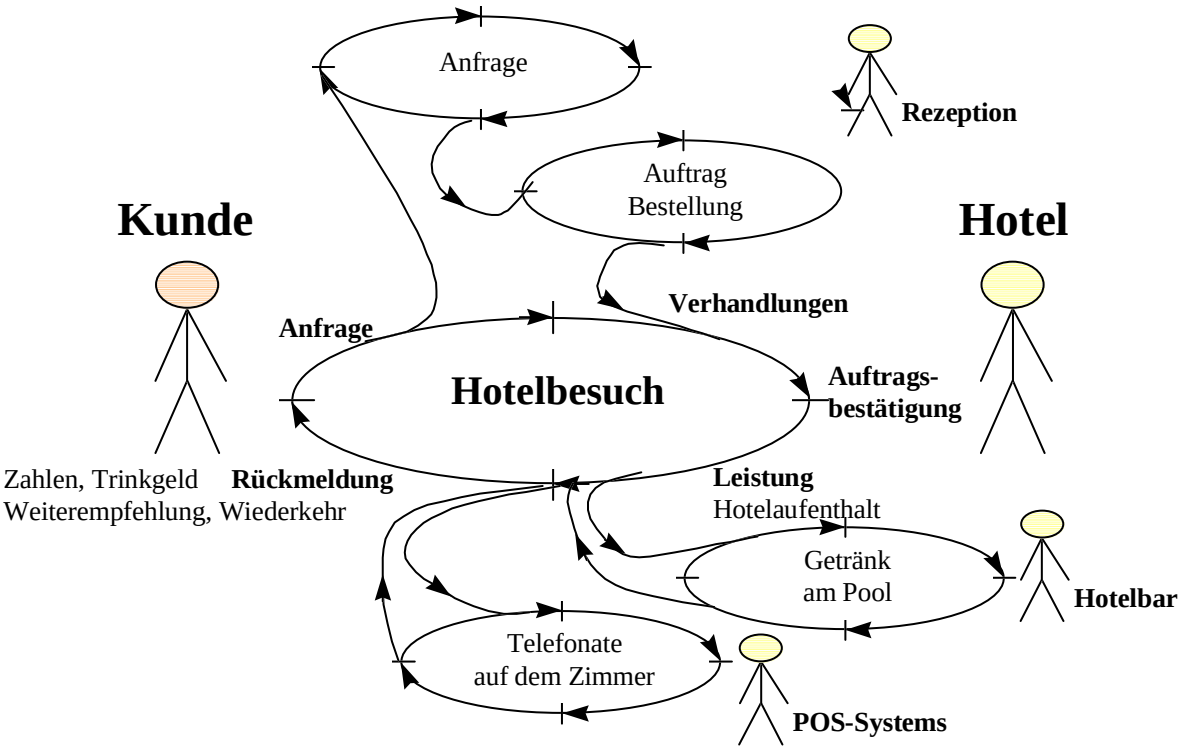

Abbildung 5.1: Action Workflow Modell: Hotelbesuch

mögliche Sekundärprozesse, die innerhalb des Gesamtprozesses auftreten. Hervorzuheben ist die ebenfalls binäre Kunde-Dienstleister-Beziehung innerhalb eines jeden Teilprozesses. Wie schon erwähnt, ist der eigentliche Reservierungsprozeß nach Beendigung der Übereinkunftsphase abgeschlossen, so daß für die Realisierung einer Zimmerreservierung in einem ersten Inkrement nur die Hälfte des im *Action-Workflow*-Modell dargestellten Prozesses zu analysieren ist. Es bleibt zu bemerken, daß die weitaus komplexeren Prozesse innerhalb eines Hotels erst nach der Reservierung auftreten. Sie werden allerdings im ersten Inkrement nicht betrachtet.

5.1.1 Konversationen

In dem Modell der *Business Conversations* wird das Hotel als Agent gesehen, der mit anderen Agenten wie zum Beispiel Gästen, Lieferanten und Mitarbeitern eine Reihe von lang anhaltenden (*long term*) Konversationen führt. Das Hauptziel ist die Koordination der ansonsten autonomen Aktivitäten aller Agenten in Richtung des gemeinsamen Ziels.
Betrachtet man in diesem Beispiel die Zimmerreservierung, so könnte zum Beispiel das gemeinsame Ziel sein, ein Zimmer über den verhandelten Zeitraum zu belegen. Dabei ziehen beide Agenten unterschiedlichen Nutzen aus dieser Belegung, nämlich das Hotel den Erlös aus der Zimmervermietung, und der Gast die Unterkunftsmöglichkeit. Bevor es zu einer Buchung des Zimmers kommt, müssen beide Beteiligte über das Prozeßwissen verfügen, wie man in diesem Falle eine Zimmerbuchung durchzuführen hat. Menschlichen Agenten ist unmittelbar klar, wie ein Zimmer zu buchen ist. Übernimmt man die einzelnen Schritte einer menschli-

chen Konversation und gliedert diese in strukturiere Sprechakte, so kommt man unmittelbar zu einer Art Prozeßbeschreibung, die in Abbildung 5.2 in Form von Dialogen wiedergegeben ist und der im Kapitel 2.2 eingeführten Konversationsspezifikation entspricht. Die einzelnen

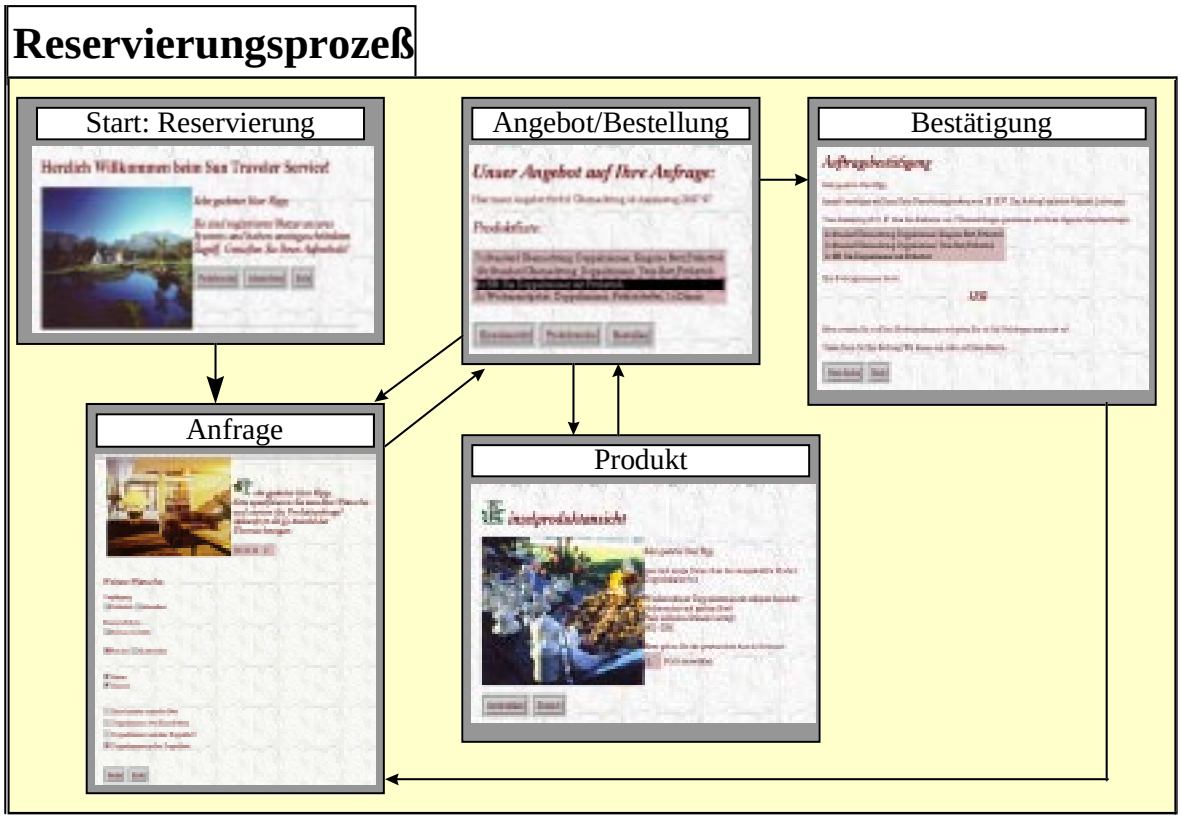

Abbildung 5.2: Konversationsspezifikation der Zimmerreservierung

Dialogschritte des Reservierungsprozesses wurden mit der Dokumentenbeschreibungssprache HTML erstellt und statisch untereinander verkettet. Diese Verkettung wird mit den Pfeilen in Abbildung 5.2 angedeutet. Sie entsprechen dem späteren Verlauf möglicher Konversationen und legen Folgedialoge einer Konversation fest. Diese durch HTML-Formulare beschriebene Prozeßinstanz der Zimmerreservierung wurde, wie einführend erwähnt, mit Domänenexperten diskutiert und abgestimmt. Anforderungen an das System lassen sich unmittelbar aus dieser Spezifikation ableiten. Im folgenden werden die einzelnen Dialogschritte dieser statischen Konversationsinstanz besprochen.
Im ersten Schritt wählt der Kunde seinen Dienstleister, was der Wahl des Hotels gleichkommt. Das Hotel legt dem Kunden die Wahl möglicher Attribute vor, die ihm die Produktsuche entscheidend vereinfacht. In dem Anfragedialog erkennt man die im vorherigen Kapitel erläuterte attributbezogene Suche auf Hotel-Produkten. Nach Übermittlung der Wünsche erhält der Gast ein Angebot, aus dem er eine beliebige Anzahl von Produkten auswählen kann. Danach legt er die von ihm benötigte Produktmenge fest und gibt seine Bestellung oder Buchung auf. Konnte eine Buchung durchgeführt werden, kommt es abschließend zu einer Buchungsbestätigung, die den Umfang der Buchung nochmals wiedergibt und eine zugeordnete Buchungsnum-

mer übermittelt, auf die sich der Gast jederzeit bei Änderungswünschen oder Stornierungen bezüglich dieser Buchung beziehen kann. Hiernach ist die Konversation beendet. Ein Abbruch ist allerdings vorher jederzeit möglich.
Aus der nun vorliegenden Konversationsspezifikation lassen sich eine Reihe von weiteren wichtigen Objekten mit entsprechenden Verantwortlichkeiten herausarbeiten. Dazu betrachtet man jeweils zwei aufeinanderfolgende Dialoginstanzen, wie in Abbildung 5.3 dargestellt, und ermittelt die auszuführenden Aktivitäten und Beteiligte. Es handelt sich dabei um die Akteure des Systems in Wahrnehmung ihrer unterschiedlichen Rollen und Aufgaben.

5.1.2 Agenten

Agenten sind autonome Teilsysteme menschlicher oder maschineller Natur, die Aufgaben eines übergeordneten Gesamtprozesses eigenverantwortlich lösen. Dabei obliegt es dem Entwickler maschineller Agenten, den Aufgabenbereich eines Agenten einzugrenzen. Das Eingrenzen ist keinesfalls trivial und nicht nach einem festen Schema zu lösen. In der Regel handelt es sich um eine Gratwanderung zwischen der Entwicklung von zu vielen Komponenten, deren Eigenverantwortlichkeiten klar umschrieben sind, aber dessen Kommunikationsaufwand zur Erreichung des gemeinsamen Ziels zu Ineffizienzen im Laufzeitsystem führen, oder von zu großen Komponenten, die monolithische Strukturen annehmen, schwer skalieren und sich nur unter sehr hohem Aufwand an evolutionäre Prozeßveränderungen anpassen lassen.
Abbildung 4.1 zeigt mögliche Komponenten und damit autonome Agenten eines Hotel-Softwaresystems. Die im ersten Inkrement zu realisierende Reservierungskomponente könnte ihrerseits wiederum in weitere Komponenten untergliedert werden. So wären zum Beispiel Agenten für die Verwaltung von Produkten, Ressourcen, Geschäftspartnern oder für die Berechnung von Produktverfügbarkeiten denkbar. Die innerhalb des Reservierungsprozesses anfallenden Aufgaben werden allerdings in dieser Arbeit mit nur einer Komponente realisiert und obliegen der Verantwortung des Reservierungsagenten. Das enge Zusammenspiel von Produkt, Ressource und Geschäftspartner sowie die kontextbezogene Preisbildung rechtfertigen diese Entscheidung.

5.1.3 Akteure und Rollen

Nach Jacobsons Ansatz [13] sind Akteure die Benutzer des Systems und werden als Hilfsmittel für die Identifikation von Anwendungsfällen benutzt. Anwendungsfälle wiederum bilden die Basis des Jacobsonschen Entwicklungsprozesses des *use case driven approach*. Akteure stehen für alles, was in irgendeiner Form mit dem System Informationen austauscht. Sie können menschliche Benutzer sein, aber auch andere Systeme darstellen, die mit dem modellierten System kooperieren, also im wesentlichen Komponenten, die von außen auf das System einwirken.
Sie unterscheiden sich wesentlich von den im vorherigen Abschnitt beschriebenen Agenten. Akteure müssen nicht notwendigerweise eigenständig sein; sie üben allenfalls eine Wirkung auf das System aus. Agenten hingegen sind eigenständig und bilden autonome Teilsysteme.
Akteure findet man, indem man Prozesse untersucht, die das System unterstützen soll. Betrachtet man beispielhaft den in Abbildung 5.3 dargestellten Prozeßschritt, so lassen sich Akteure und deren Rollen identifizieren. In diesem Beispiel initiiert der eigentliche Benutzter die Buchung eines Zimmers, indem er das Suchformular über den Weg des Internets mit Infor-

mationen füllt und abschickt, d.h. der Kontakt mit dem Hotel wird hergestellt. Kontakte sind für das Hotel sehr wichtig, da sie Aufträge an das Hotel herantragen. Sie werden vom Hotel gesondert gepflegt und stellen potentielle Ziele für Werbekampagnen dar. Der eigentliche Vertragspartner wird die Person oder Firma sein, die die Kosten des späteren Hotelaufenthalts trägt. Er muß nicht identisch mit dem späteren Hotelgast sein.
Der Produktmanager ist für die Erstellung von Angeboten verantwortlich und könnte als menschlicher Agent an der Rezeption arbeiten.
Es lassen sich demnach die Akteure „Internetbenutzer“ in den Rollen „Systembenutzer“„Kon-

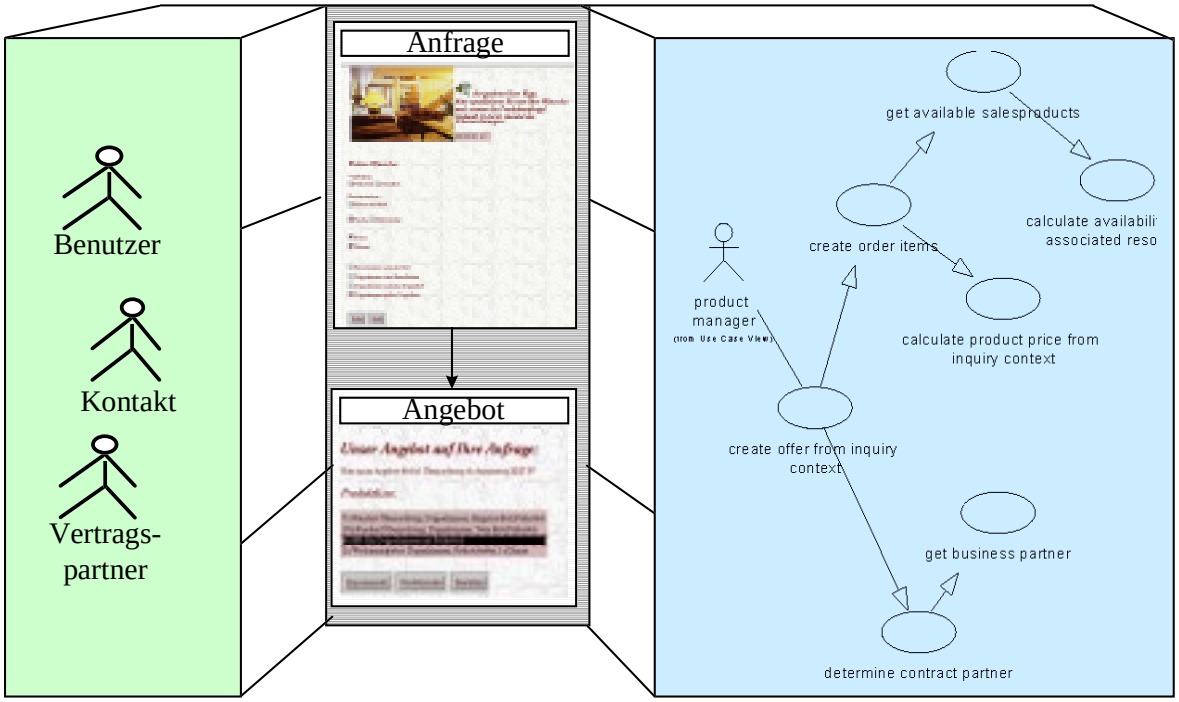

Abbildung 5.3: Identifikation von Akteuren, Rollen und Anwendungsfällen

takt“ und „Vertragspartner“ sowie den dem Reservierungsagenten inhärenten „Produktmanager“ finden. Akteure führen Anwendungsfälle aus und helfen diese zu identifizieren.

5.2 Anwendungsfälle („use cases“)

Anwendungsfälle helfen dem Entwickler, die Systemanforderungen besser zu verstehen. Sie wurden jeher schon in Form von Szenarios aufgeschrieben und waren Teil des Anforderungskataloges. Jacobson formalisierte Anwendungsfälle erstmals 1994 und machte sie zu einem fundamentalen Teil des Software-Entwicklungsprozesses. Sie flossen in visualisierter Form als *Use-Case*-Diagramm in die *Unified Modelling Language* (UML) mit ein.
Im wesentlichen beschreibt ein Anwendungsfall die Systemfunktionalität, die bei einer Benutzerinteraktion erbracht wird. Akteure liefern, wie schon beschrieben, die für den Ablauf notwendigen Informationen zum Auslösen einer Systemaktion. Jede Aktion, die von einem Ereignis ausgelöst wird, kann als potentieller Anwendungsfall angesehen werden. Betrachtet

man wiederum Abbildung 5.3 so löst die Anfrage des Benutzers beim Produktmanager die Aktion „erstelle Angebot“ aus.

5.2.1 Hierarchische Zerlegung

Der oben identifizierte Anwendungsfall hat die Aufgabe, ein Angebot aus dem Kontext der Anfrage zu erstellen. Der Vorteil, Anwendungsfälle an Hand einzelner Dialogschritte innerhalb des Prozesses in Verbindung mit den identifizierten Akteuren zu ermitteln, liegt im Finden von Anwendungsfällen, die in einer unmittelbaren hierarchischen „benutzt“-Beziehung zum eigentlich betrachteten Anwendungsfall stehen. Anwendungsfälle können demnach hierarchisch gegliedert sein und in einem *uses*- bzw. *extends*-Verhältnis zueinander stehen. Im oben betrachteten Fall erkennt man gut, daß bei der Angebotserstellung Bestellpositionen erzeugt werden, die sich auf Verkaufsprodukte beziehen, deren Verfügbarkeit überprüft und deren Preise in Abhängigkeit vom Geschäftspartner bestimmt werden müssen. Es läßt sich ein *Use-Case*-Diagramm erstellen, das die Beziehungen und Abhängigkeiten unmittelbar wiedergibt. Das Diagramm ist im rechten Kasten von Abbildung 5.3 dargestellt.

5.2.2 „Uses“-Beziehungen

Die *uses*-Beziehung wird eingesetzt, falls Anwendungsfälle von anderen in ähnlicher Form benutzt werden, d.h. wenn Anwendungsfälle an unterschiedlichen Stellen doppelt auftauchen. Um schlichtes Kopieren zu vermeiden, führt man die „benutzt“-Beziehung ein und bekommt hierarchische Strukturen von Anwendungsfällen. In unserem Beispiel ist das Auffinden eines Geschäftspartners ein typischer Anwendungsfall, der innerhalb des Systems an unterschiedlichen Stellen auftreten kann. Er steht zum Beispiel in einer Benutztbeziehung zum Anwendungsfall „Vertragspartner festlegen“ oder „Systembenutzer führt Login aus“.

5.2.3 „Extends“-Beziehungen

Die *extends*-Beziehung erweitert einen allgemeinen Anwendungsfall um spezielleres Verhalten. Sie ist immer dann zu verwenden, wenn ein allgemeiner Anwendungsfall in einer Variation auftritt. Varianten sind durch Erweiterungen zu beschreiben. Innerhalb des Hotels würde zum Beispiel der Anwendungsfall „berechne Produktpreis unter Berücksichtigung des Geschäftspartners“ als Erweiterung der einfachen Berechnung des Produktpreises gesehen werden. Ein Ausschnitt der innerhalb des Reservierungsprozesses identifizierten Anwendungsfälle ist im *Use-Case*-Katalog im Anhang A.2 wiedergegeben.

5.3 Identifikation von Business Objekten

Aus dem Anforderungskatalog mit seinen Schlüsselabstraktionen und Anwendungsfällen, den identifizierten Akteuren, den *Use-Case*-Diagrammen und der statischen Prozeßinstanz lassen sich im nächsten Schritt die wesentlichen *Business*-Objekte identifizieren. Dabei wird der Übergang von der Analysephase in die Designphase mit einem ersten Klassendiagramm begonnen. Dieses Klassendiagramm spiegelt im wesentlichen die Beziehungen der identifizierten *Business*-Objekte wider und abstrahiert vorerst von den Einschränkungen, die Verteilungs-Architektur- und Persistenzkonzepte mit sich brächten. In einem ersten Objektmodell werden wie oben angedeutet nur die rein betriebswirtschaftlichen Aspekte berücksichtigt und

dargestellt. In Abbildung 5.4 wird verdeutlicht, wie die in der Analysephase entwickelten Ergebnisse in das Objektmodell einfließen. Das dazugehörige Klassendiagramm ist im Anhang B nochmals vergrößert wiedergegeben.

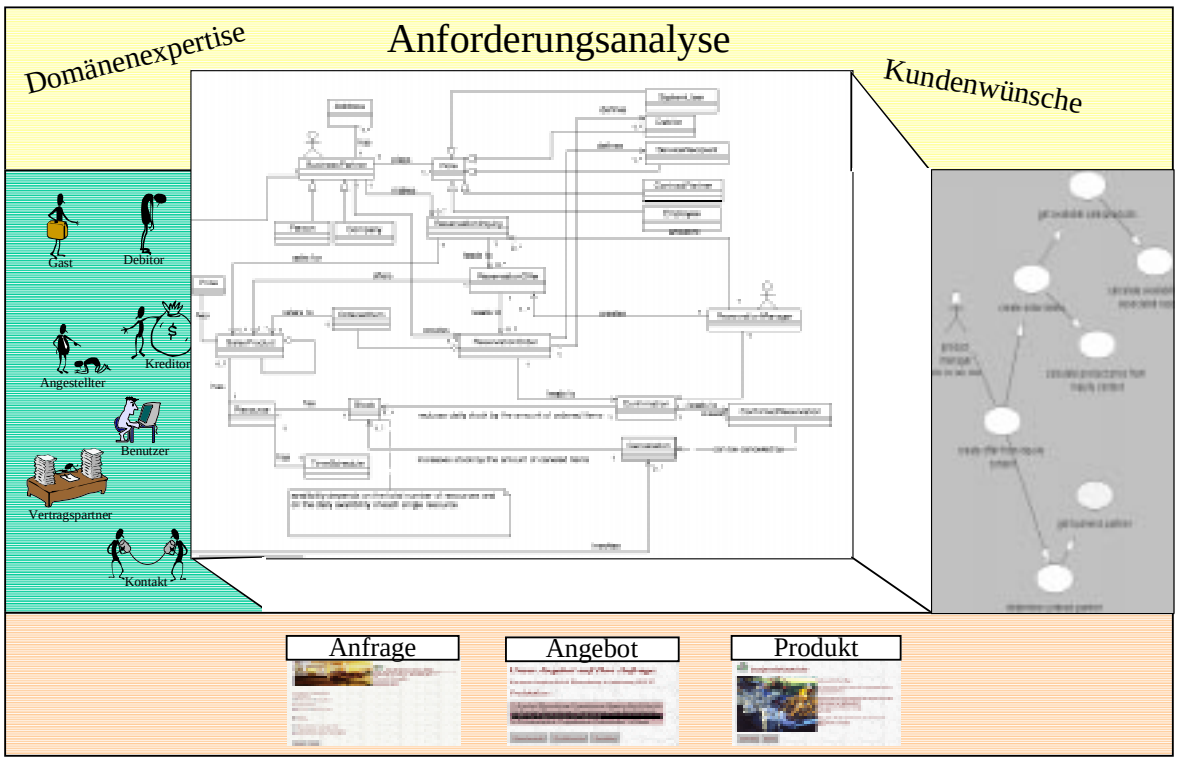

Abbildung 5.4: Erstes UML-Klassendiagramm

Im folgenden wird das Objektmodell aus Abbildung B.1 diskutiert. Es bleibt zu erwähnen, daß es mit den Mitteln der UML [7, 34] notiert wurde. Als Leitfaden durch das Diagramm wird wieder der Ablauf einer Prozeßinstanz verfolgt. Ausgehend von dem Geschäftspartner, der anhand der Spezialisierungsbeziehung eine Person oder eine Firma sein kann, wird in seiner Rolle des Systembenutzers eine Anfrage an den Reservierungsmanager gestellt, der diese mit einem Angebot beantwortet. Dazu müssen Bestellpositionen erzeugt werden, die in enger Beziehung mit den Verkaufsprodukten stehen. Ein Produkt kann wiederum Produkte enthalten. Jedes Einzelprodukt besitzt einen Preis und eine zugeordnete Ressource, die seine Verfügbarkeit bestimmt. Verfügbarkeiten zeitbezogener Ressourcen werden über Reservierungen oder Stornierungen gesteuert. Nach einem Angebot folgt die Buchung, die ihrerseits mit einer Buchungsbestätigung quittiert wird, was zu einer bestätigten Reservierung innerhalb des Systems führt und die Verfügbarkeit der betroffenen Ressource verringert. Reservierungen können ihrerseits storniert werden, was die Verfügbarkeit wieder erhöht. Als weitere Rollen eines Geschäftspartners werden beispielhaft noch der Debitor, Kreditor und der Angestellte aufgeführt.
Attribute und Methoden wurden innerhalb dieses ersten Klassendiagrammes noch nicht berücksichtigt. Sie werden erst im weiteren Verlauf der Entwurfsphase in einem zweiten Klassendiagramm schrittweise verfeinert.

5.4 Kooperative Informationssysteme

Die Entwicklung von kooperativen Informationssystemen soll einen Beitrag zur evolutionären Weiterentwicklung von Systemen und deren Anpassung an die sich schnell veränderten Marktanforderungen leisten. Dabei wird versucht, die Entwicklungskosten auf einem finanzierbaren Niveau zu halten und die Wiederverwendung von Systemen und Integration von Altsystemen voranzutreiben.
Als vielversprechender Schritt gilt der Bau von Satellitensystemen, die bestehende Systeme um neue Funktionalitäten erweitern, ohne diese verändern zu müssen. Voraussetzung für eine solche Erweiterung ist allerdings eine Kooperationsschnittstelle, über die die Handlungsabsichten innerhalb des kooperativen Prozesses transparent für alle Beteiligten vermittelt wird. Ein kooperatives Informationssystem besteht demnach aus mehreren abgeschlossenen und separat entwickelten Anwendungskomponenten, die als eine Menge von verteilten, kommunizierenden Komponenten zu einem Gesamtsystem verschmelzen und keinen gemeinsamen Adressraum voraussetzen. Systeme dieser Art agieren mit Systemen ihrer Umgebung, den sogenannten Agenten, indem sie Nachrichten austauschen und in der Rolle eines Diensterbringers oder eines Kunden auftreten. Dabei werden menschliche und maschinelle Agenten als gleichberechtigte Partner bezüglich ihrer Aufgaben innerhalb des Gesamtsystems angesehen, deren primäres Ziel das Lösen gemeinsamer Aufgaben ist. Als adäquates Kooperationsmodell für den Bau von kooperativen Informationssystemen wird das der *Business Conversations* angesehen, das bereits in Kapitel 2.2 vorgestellt wurde.

Das Modell unterstützt unterschiedliche Modalitäten der Kooperation, die in Abbildung 5.5 dargestellt sind. Dabei spielt es keine Rolle, welches Kommunikationsmedium zwischen den Kooperationspartnern gewählt wird, solange beiden Partnern dieses bekannt ist und zur Verfügung steht. Im wesentlichen zeigt Abbildung 5.5 die möglichen Kooperationsformen unterschiedlichster Agenten. Bei einer Maschine-Maschine-Kooperation spricht man von *Application Linking*, bei einer Mensch-Maschine von *GUI-Management*, bei Maschine-Mensch von *Workflow Management* und bei der in dieser Abbildung nicht aufgeführte Mensch-Mensch Kooperation von *Structured Message Handling* [26]. Da jeder Agent eigenständig innerhalb des Gesamtsystems unterschiedliche Aufgaben wahrnimmt, jedoch zur Erreichung des gemeinsamen Ziels mit anderen Agenten kooperiert, müssen Informationen zwischen ihnen ausgetauscht werden. In jedem tritt ein Agent entweder als Kunde oder als Diensterbringer auf. Eine Kooperation kommt zustande, falls sich Kunde und Dienstleister über die zu erbringende Leistung einigen. Kann ein Agent eine Leistung nicht erbringen, obwohl er sie anderen anbietet, so kann er diese von einem anderen Agenten beziehen und so als Vermittler weiterleiten. In diesem Falle nimmt ein Agent beide Rollen gleichzeitig ein. Es ist jedoch für den eigentlichen Kunden transparent, daß die erhaltene Leistung nicht vollständig vom beauftragten Dienstleister erbracht wurde.
Aufbauend auf diesem Kooperationsmodell wird im nächsten Kapitel eine Architektur zur Entwicklung kooperativer Informationssysteme vorgestellt, die sich sehr gut für den Bau von Satellitensysteme eignet.

5.5 Bewertung der Analysephase

Rückblickend hat sich das geschilderte Vorgehen in der Analysephase sehr gut bewährt. Innerhalb dieser Phase wurden viele Gespräche mit Domänenexperten geführt. Dabei konnte

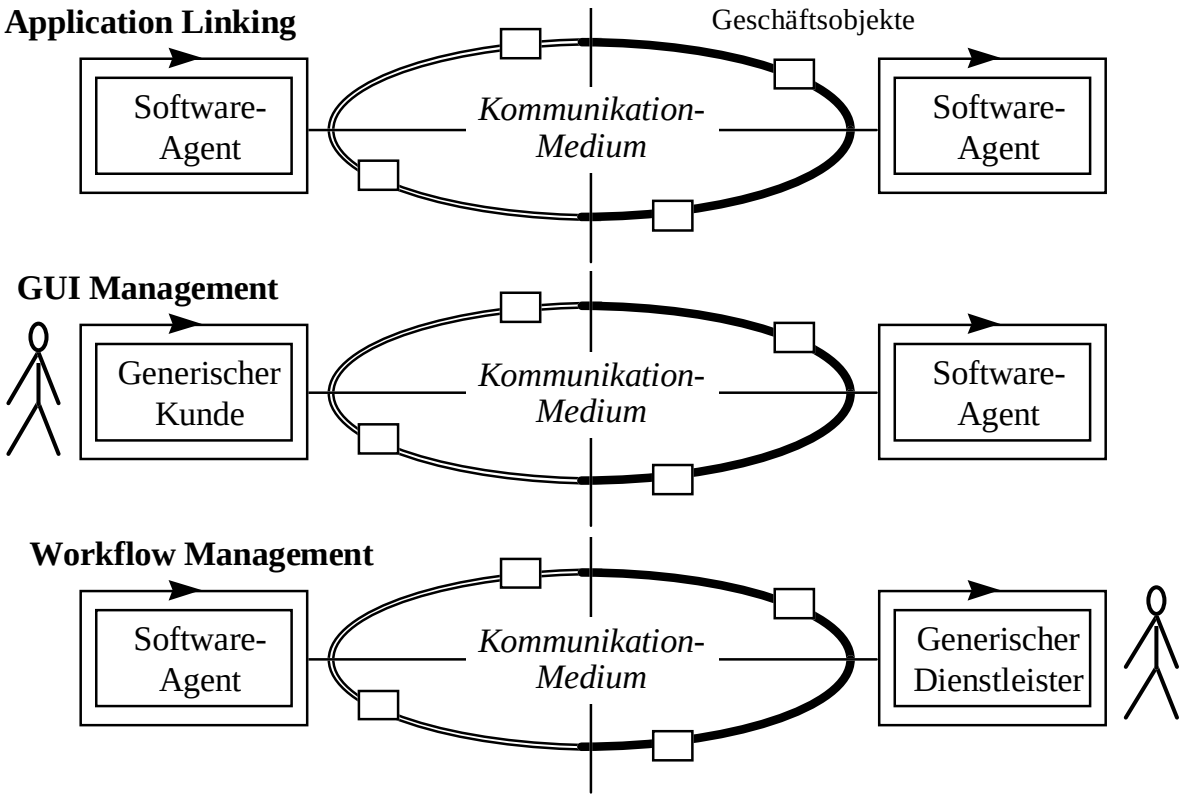

Abbildung 5.5: Unterstützung unterschiedlicher Modalitäten der Kooperation

beobachtet werden, daß oftmals Anforderungen von verschiedenen Experten unterschiedlich bewertet wurden. Gleichzeitig wurden sehr wichtige Systemanforderungen nur nebenbei oder gar nicht erwähnt, von deren Bedeutung erst Folgegespräche zeugten. Aus diesem Grund hat es sich bewährt, Gespräche an den wichtigen Kernprozessen anzulehnen. Das Spezifizieren von Kernprozessen half, eine vernünftige Abgrenzung des Systems zu finden und die Analyse in überschaubare Einheiten einzuteilen. Bei der Analyse von Kernprozessen kam man schrittweise zu den wichtigen Abweichungen und Nuancen der untersuchten Prozesse. Im Falle des Reservierungssystems hat es sich als vernünftig herausgestellt, ihn prototypisch zu visualisieren. Der Prozeß wurde dadurch anschaulich und konnte mit Experten und Kunden besser besprochen werden.
Bei der Analyse einzelner Prozeßschritte dieser visualisierten Prozeßinstanz konnten relativ zügig die Akteure und Beteiligten und oftmals auch hierarchisch strukturierten Anwendungsfälle herausgearbeitet werden. Anwendungsfälle, Akteure und Schlüsselabstraktionen des Anforderungskatalogs führten wie in Abbildung 5.4 skizziert schnell zu einem ersten Objektmodell, das den Übergang zur Entwurfsphase einleitete.

Kapitel 6

Entwurf

In diesem Kapitel wird die Entwurfsphase des in dieser Arbeit verfolgten Entwicklungsprozesses beschrieben. In ihr werden die notwendigen Entscheidungen, die vor der Realisierung eines Systems getroffen werden müssen, festgeschrieben und darauf aufbauend Entwurfmodelle entwickelt.
Wichtige solcher Eckwerte bildet die zugrundeliegende Systemarchitektur, die Antwort auf die Fragen gibt, ob es sich bei dem System um ein Einzelplatz- oder Mehrbenutzersystem handelt, ob es verteilt, zentral oder agentenbasiert ist, was für eine Benutzungsoberfläche eingesetzt wird, welche Programmiersprache die Entwurfsmodelle später implementiert und welches Persistenzkonzept für die Speicherung von Anwendungsdaten in Frage kommt. Stehen diese Eckdaten des zukünftigen Systems fest, kann dazu übergegangen werden, aus den Modellen der Analyse weiterführende Entwurfsmodelle unter Berücksichtigung der eben aufgezählten Einschränkungen zu entwickeln. Dazu gehören Klassendiagramme, die die in der Analysephase identifizierten Objekte und Klassen konkret beschreiben. Im speziellen gehen sie auf Klassenattribute, Methoden und ihren Beziehungen zu anderen Klassen ein. Darüber hinaus lassen sich mit Hilfe von Interaktions- und Zustandsdiagrammen Objektinteraktionen und interne Zustandsübergänge von Objekten beschreiben. Nachfolgend werden diese Schritte am Beispiel des Reservierungssystem erörtert.

6.1 Architekturmodell

Als erster Schritt in der Entwurfsphase gilt es, die Soft- und Hardwarearchitektur für das zu entwickelnde System festzulegen. Die Entscheidung für eine Architektur bildet die Basis des Systems und sollte im Vorfelde gründlich diskutiert und im wesentlichen auf Fragen der Verteilung, der Speicherung von Anwendungsdaten und auf die zu verwendende Programmiersprache eingehen. Die in diesem Entscheidungsprozeß getroffenen Vereinbarungen sind fundamental und wirken sich unmittelbar auf den Entwurf aus. Sie sollten gründlich überdacht und nach einer Entscheidung nicht wieder umgestoßen werden.
Sind diese grundlegenden Fragen geklärt, kann unmittelbar mit dem Umsetzen der Ergebnisse aus der Analysephase unter Berücksichtigung der gewählten Architektur und Einschränkungen der Programmiersprache begonnen werden. Bei Verwendung einer objektorientierten Sprache hat sich der Einsatz von Entwurfsmustern [8] bewährt.
Als Architektur für das Reservierungssystem wird die aus dem Modell der *Business Conversations* abgeleitete und in [38] als Agentenumgebung implementierte Architektur eingesetzt. Sie

eignet sich besonders für den Bau von kooperativen Informationssystemen und stellt den kooperativen Charakter eines Informationssystems in den Vordergrund. Die Agentenumgebung wird als sogenannte *middleware* verstanden und deckt die Basisdienste zur Kommunikation und der regelbasierten Verarbeitung von Konversationsspezifikationen ab. Auf sie kann die eigentliche betriebswirtschaftliche Anwendung, in diesem Falle das Reservierungssystem, gesetzt werden, die als autonome verteilte, unter Umständen auch mobile, Systemeinheit ihre speziellen Aufgaben wahrnimmt. Eine auf dieser Architektur basierende Systemkonfiguration könnte wie in Abbildung 6.1 aussehen. Hierbei handelt es sich um eine Systemkonfiguration

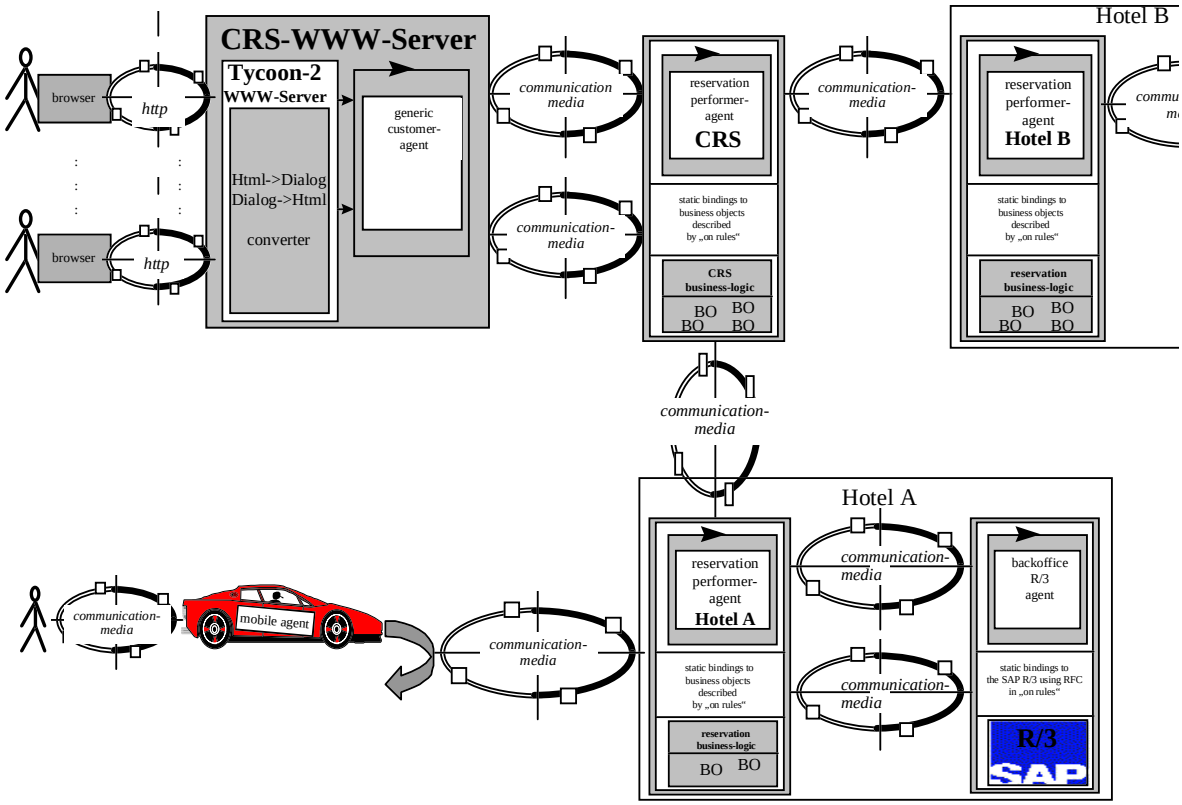

Abbildung 6.1: Architekturmodell für verteilte und mobile Agenten innerhalb kooperativer Informationssysteme

von beliebig vielen Reservierungssystemen, wovon jedes einzelne innerhalb eines Hotels oder einer Hotelkette steht und seinerseits unter Verwendung derselben Technologie als Satellitensystem zu einem *Back-Office-System*, hier dem SAP R/3-System, realisiert ist. Des weiteren gibt es ein zentrales Reservierungssystem (CRS), das gestellte Anfragen an die entsprechenden Hotels transparent weiterleitet. Anfragen können sowohl vom Benutzer über einen als Agenten implementierten WWW-Server kommen, als auch über mobile Agenten, die vom Kunden mit der Aufgabe, ein Zimmer zu reservieren, ins Leben gerufen und wie E-Mails an das Hotel verschickt werden, wo sie aufgrund ihrer Autonomie und inhärenten Prozeßwissens eine Zimmerbuchung vornehmen. Mobile Agenten [21] stellen eine asynchrone Kommunikation zwischen dem Benutzer und dem Hotelsystem her, hingegen handelt es sich bei einer

WWW-Verbindung um eine synchrone Kommunikationsform.

6.1.1 Autonome Agenten

Autonome Agenten sind wie angedeutet eigenständige Systeme oder Systemeinheiten, die nach individuellen und lokalen Gesetzen handeln und spezielle Aufgaben einer möglichen übergeordneten Gesamtaufgabe übernehmen. Dabei kapselt ein Agent Daten, Funktionen und eine Historie, die seinen momentanen Zustand bestimmt, aber auch sein zukünftiges Verhalten unmittelbar beeinflussen wird. Als charakteristisches Merkmal besitzt ein autonomer Agent einen eigenen Kontrollfluß, der Voraussetzung für Selbstbestimmung und Autonomie ist.
Beim Zustandekommen einer Kooperation zwischen einzelnen Agenten müssen eine Reihe von Voraussetzungen gegeben sein. Im speziellen sind dies eine gemeinsame Zielidentität, die Veränderbarkeit von Plänen und der Austausch von Ressourcen. Handlungsabsichten müssen für jeden Kooperationspartner im voraus erkennbar sein, so daß die Transparenz des kooperierenden Prozesses gewährleistet wird. Bei lange anhaltenden Kooperationen richtet sich die Lebensdauer von maschinellen Agenten nicht nach Betriebssystemsitzungen, sondern nach der Dauer der zu unterstützenden Geschäftsprozesse.

6.1.2 Koordination mittels Konversationen

Ein integraler Bestandteil der *Business Conversations* ist das Kooperationsmodell, das die eigentliche Interaktion innerhalb eines Gespräches zwischen zwei Kooperationspartnern festlegt. Den Gesprächspartnern ist die zugrundeliegende Konversationsspezifikation bekannt und innerhalb einer Konversation für beide bindend. So wird der zielgerichtete Fortgang einer Konversation sichergestellt.
Interaktionen werden mit strukturierten Dialogen realisiert, deren Spezifikation Bestandteil der Konversationsspezifikation ist. Mittels dieser Dialoge werden Informationen so strukturiert, daß der jeweilige Kommunikationspartner die im Sprechakt enthaltene Information seinem eigenen Ausführungskontext zuordnen kann.
Jeder Dialog beschreibt demnach den aktuellen Zustand einer Konversation. Eine Anfrage an den Dienstleister führt zum Bearbeiten einer an den Dialog gebundenen Dienstleisterregel, die wiederum einen neuen Dialog für den Kunden generiert. Auf diese Weise werden Dialogfolgen erzeugt, die aufgrund der Kunden- und Dienstleisterregeln wechselseitig bearbeitet und der Spezifikation entsprechend durchlaufen werden. Tritt ein Gesprächspartner als *broker* auf, so kann dieser nebenläufig sekundäre Konversationen führen, um seine angebotene Dienstleistung vertragsgemäß zu erbringen. Genaueres zum Ablauf von sekundären Konversationen findet sich in [16, 30, 38].

6.1.3 Benutzungsschnittstelle („GUI“)

Die graphische Benutzungsoberfläche stellt eine der wichtigsten Komponenten eines Informationssystems dar. Mit ihr werden Informationen erfragt, Datenpflege betrieben und Prozeßabläufe kontrolliert. Sie ist die Schnittstelle zum menschlichen Benutzer und sollte ihm die Arbeit so einfach wie möglich gestalten. Dazu gehören übersichtlich angeordnete Dialoge, Eingabehilfen und Schaltelemente, die der gewohnten Rechnerumgebung entsprechen.
Moderne Entwicklungsplattformen für Datenbank- und Informationssysteme koppeln die Benutzungsschnittstelle sehr eng an die mit der zugrundeliegenden Datenbankstruktur. Dieses

hat den Vorteil, daß man in kürzester Zeit relativ schnell kleinere Anwendungen entwickeln kann. Diese enge Kopplung führt später allerdings zu erheblichen Mehraufwand bei der Wartung und Pflege solcher Systeme. Zusätzlich ist man auf den Hersteller angewiesen, der für den Einsatz des Systems in heterogenen Umgebungen jeweils entsprechende Laufzeitumgebungen zur Verfügung stellen muß, was sehr schwer einzuhalten ist, besonders wenn es sich dabei um propretäre Technologie eines Herstellers handelt.
Eine Trennung von Benutzerinteraktion und betriebswirtschaftlicher Funktionalität sollte auf jeden Fall eingehalten werden. Dadurch steht der Wiederverwendung betriebswirtschaftlicher Funktionalität nichts im Wege. Außerdem ließen sich weitere Zugänge unterschiedlichster Modalität mit relativ geringem Aufwand neu einrichten, so zum Beispiel ein Zugang über das Internet mittels eines WWW-Servers.
Der hier vorgestellten Architektur entsprechend werden zwischen zwei kooperierenden Agenten strukturierte Dialoge ausgetauscht, deren Inhalte sehr gut mit HTML-Formularen darzustellen sind. Setzt man einen weiteren Agenten zur Konvertierung der Dialoge nach HTML ein, der als WWW-Server fungiert, also die Aufgabe der Schnittstelle zwischen Internet und der Kommunikationsschicht der Agentenumgebung wahrnimmt, erhält man durch den Einsatz von handelsüblichen WWW-Browsern eine günstige Benutzungsschnittstelle für heterogene Systeme.
Menschliche Agenten kommunizieren über diese Schnittstelle und werden innerhalb der Agentenumgebung als generischer Kunde angesehen, der jede Konversation führen kann, sofern diese vom WWW-Server visualisiert wird.

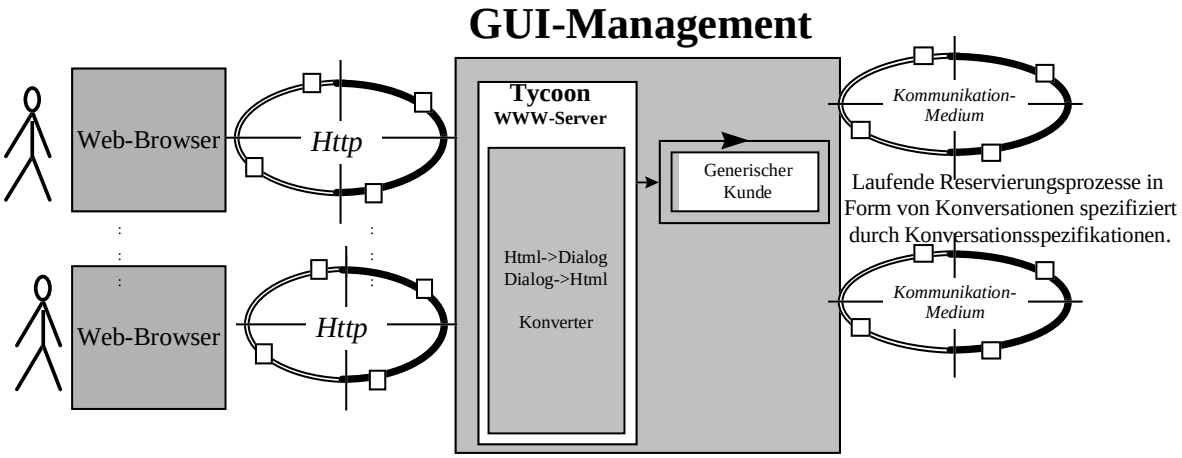

Abbildung 6.2: Der WWW-Server als HTML-Konverter strukturierter Dialoge

Führt man sich den im Kapitel 3.3 geschilderten bruchlosen Entwicklungsprozeß vor Augen, so läßt sich eine Konversationsspezifikationen und damit ihre inhärenten Dialogspezifikationen aus einem Satz von HTML-Formularen erzeugen. Sie werden mittels eines *parsers* in eine dem Laufzeitsystem bekannte Datenstruktur konvertiert, aus denen sich wiederum Dialoginstanzen im Sinne der *Business Conversations* generieren lassen.
Dialoginstanz und HTML-Formular sind bis auf die Dialoginhalte äquivalent und über ihren Namen erreichbar. Es erscheint sinnvoll, diese HTML-Formulare innerhalb des WWW-Servers als Strukturinformationen zur Darstellung von Dialoginstanzen zu speichern. Dialoginstanzen und HTML-Schablonen lassen sich dann zu neuen HTML-Dokumenten verschmelzen, die die Dialoge einer Konversation innerhalb des WWW-Browsers dem menschlichen Agenten visualisieren.
Inhalte von Dialogen sind veränderbar und können in Form von Anfragen an das System zurückgeschickt werden. Der WWW-Server übernimmt die geänderten Inhalte, verändert entsprechend die dazugehörige Dialoginstanz und schickt diese an das Reservierungssystem, das wiederum neue Dialoge in Abhängigkeit des Konversationskontextes generiert. Dieser Ablauf wiederholt sich bis zum eigentlichen Ende der Konversation. Das hier beschriebene Verfahren zur Visualisierung von Dialogen wird in dem hier vorgestellten und entwickelten Zimmer-Reservierungssystem eingesetzt, um Anfragen eines Gastes an das Reservierungssystem weiterzuleiten. Implementierungsdetails und ablaufspezifische Eigenschaften dieses Verfahrens sind im Abschnitt 7.4 nachzulesen.

6.2 „UML“-Klassendiagramme

Das Klassendiagramm der *Unified Modelling Language* stellt die statische Klassenstruktur aller identifizierten Objekte dar und beschreibt ihre Beziehungen zueinander mittels Assoziationen. Bei Assoziationen handelt es sich um Vererbungs-, Aggregat- und „benutzt“-Beziehungen zwischen Klassen. Sie können in uni- und bidirektionaler Form auftreten, was später in der Implementierung Einfluß auf die Art der Verkettung von Objekten hat und wesentlich die Navigation zwischen Objekten bestimmt. Zusätzlich besitzt jede Assoziation eine Kardinalität, die angibt, ob es sich um eine 1:1-, 1:n- oder n:m-Beziehung handelt.
Neben Assoziationen werden auch Attribute und Methoden aller aufgeführten Klassen beschrieben. Attribute und Methoden werden durch ihren Namen und Angabe ihres Typs bzw. ihrer Signatur spezifiziert. Das Klassendiagramm hat demnach einen deklarativen Charakter und bezieht sich überwiegend auf die Typebene. Es bleibt zu erwähnen, daß eine Generierung von Programmrümpfen aus dem Klassendiagramm heraus möglich ist, da alle wesentlichen Informationen für die Deklaration von Klassen vorhanden sind.
Auf eine Einführung zur Notation von Klassendiagrammen der UML muß verzichtet werden, kann aber in [2, 7, 34] nachgeholt werden.
Im folgenden werden Entwurfsmuster, die die Realisierung der im Kapitel 4.1 aufgeführten Anforderungen wesentlich vereinfachen, an Hand von Klassendiagrammen erläutert. Hierbei handelt es sich um Auszüge aus dem gesamten Klassendiagramm des Zimmer-Reservierungssystems, welches in vollständiger Form im Anhang C wiedergegeben ist.

6.2.1 Entwurfsmuster

Entwurfsmuster beschreiben einfache und elegante Muster zur Lösung von Aufgaben gleicher Problemklassen. Dabei entstammen die hier besprochenen Muster – sie gehören zum Rüstzeug

eines jeden OO-Programmierers – aus dem objektorientierten Sprachparadigma.
Im allgemeinen werden Entwurfsmuster nicht einfach „mal eben“ entwickelt, sondern sind aus dem Versuch heraus gefunden worden, Programme anpassungsfähiger, flexibler und wiederverwendbarer zu gestalten. Ein Satz von wichtigen Entwurfsmustern wird in [8] vorgestellt. Sie lassen sich mit jeder herkömmlichen objektorientierten Sprache implementieren. Nachfolgend werden die wichtigsten Muster beschrieben und ihr Einsatz innerhalb des Reservierungssystems erläutert und deren Nutzen erklärt. Sollte die generelle Funktionsweise dieser Muster nicht verstanden werden, sollte [8] zur Hilfe genommen werden.

„Composite“

Ein *composite*[1] setzt Objekte zu baumartigen Strukturen zusammen, die ganze Teile-Hierarchien darstellen. Dabei werden sowohl einzelne Objekte als auch zusammengesetzte von ihren Benutzern gleichförmig behandelt. Die Entscheidung, ob ein *composite* zusammengesetzt ist oder nur aus einem Blatt besteht, wird von der eigentlichen Objektinstanz dynamisch zur Laufzeit entschieden und obliegt nicht der Verantwortung des Benutzers.
Der Grundgedanke dieses Musters liegt in der Verwendung einer abstrakten Superklasse, die die gemeinsame Schnittstelle von Blättern und Knoten definiert und dadurch ihre gleichförmige Nutzung sicherstellt. Betrachtet man Abbildung 6.3 so besteht das *composite* aus den Klassen *Product*, *SimpleProduct* und *ComposedProduct*, die wiederum die Funktionalität der *container*-Klasse *Dictionary* ererbt. Ein Produkt kann demnach aus einem einfachen Produkt mit dazugehörigem Preis oder aus einem zusammengesetzten bestehen. Zusammengesetzte Produkte aggregieren einzelne und zusammengesetzte Produkte. Beide Produktklassen implementieren die abstrakten Funktionen ihrer gemeinsamen Superklasse, zum Beispiel die Methode *acceptVisitor*, die das Inspizieren eines *composites* mit Hilfe eines *visitor*-Musters unterstützt, das im nächsten Abschnitt eingeführt wird.

„Visitor“

Ein *visitor* realisiert eine beliebige Operation auf einer relativ stabilen Objektstruktur, die zum Beispiel durch das *composite*-Muster beschrieben ist. Dabei lassen sich mit einem *visitor* sehr einfach und schnell neue und verschiedenartige Operationen auf einer solchen Struktur definieren, ohne daß diese dafür verändert werden müßte. Einzige Voraussetzung ist, daß jedes Element der Objektstruktur die Methode *acceptVisitor* implementiert und zusätzlich jeder *visitor* eine dem Element zugeordnete Methode realisiert, die die Operation auf dem Element ausführt.
In dem Beispiel von Abbildung 6.3 wären es die Methoden *singleProduct* und *composedProduct*, die in der abstrakten Klasse *ProductVisitor* definiert und in jeder ihrer Subklassen, in diesem Fall nur *SalespriceProductVisitor*, implementiert werden. Der *SalespriceProductVisitor* berechnet den Verkaufspreis eines beliebig zusammengesetzten Produktes.
Der Vorteil, die Preisbildung mit Hilfe eines *visitors* zu implementieren, liegt in ihrer Flexibilität. Hat man zum Beispiel mehrere Preisstrategien oder muß man eine weitere einführen, so ist die Funktionalität der Preisbildung innerhalb des *visitors* versteckt, obwohl das Produkt Auskunft über seinen Preis gibt. Mit Hilfe des *factory*-Musters (eine *factory* erzeugt in

[1] Innerhalb dieses Abschnittes werden die englischen Musternamen verwendet, jedoch wird statt *pattern* das deutsche Wort „Muster“ benutzt.

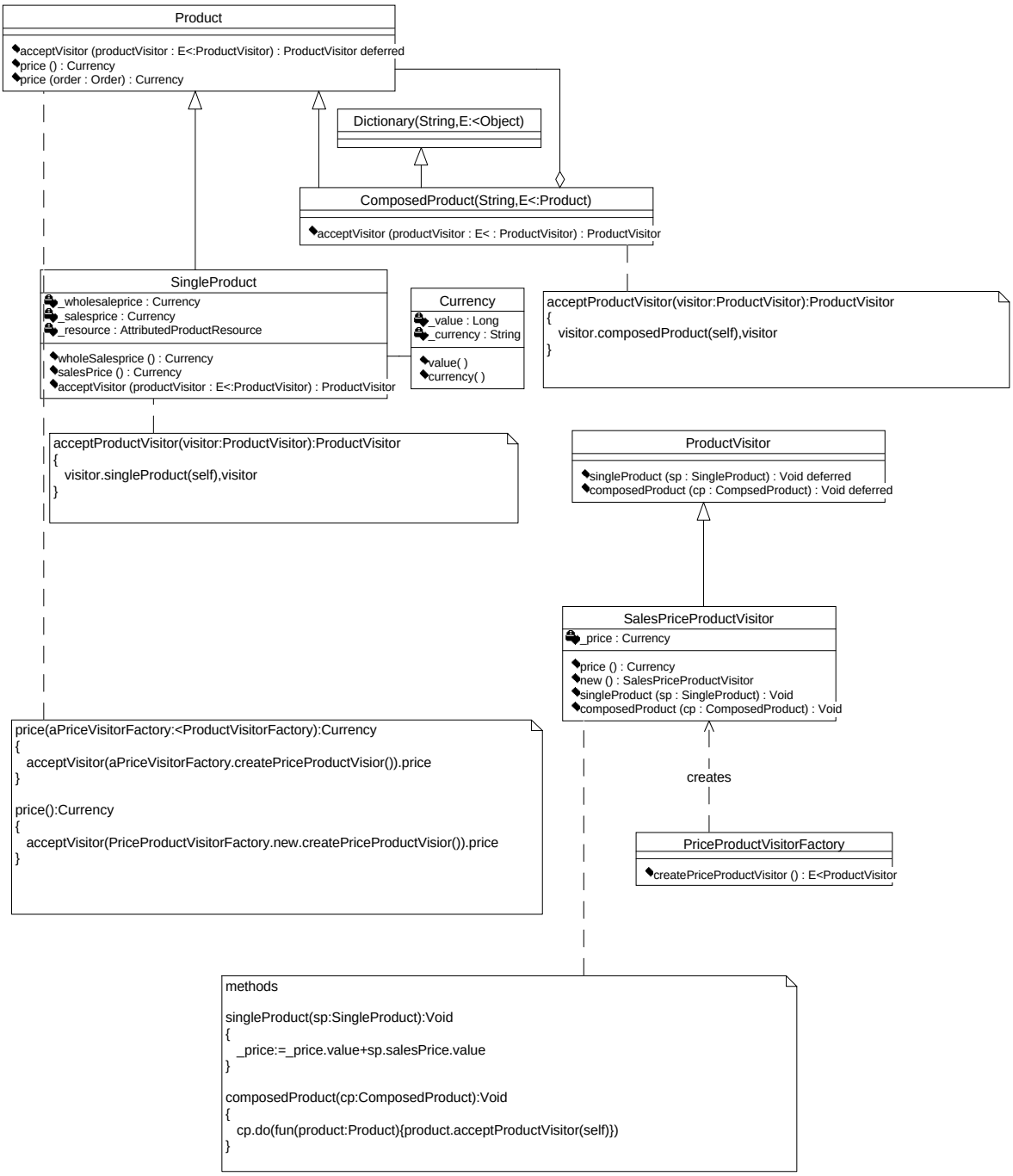

Abbildung 6.3: Das „composite“- und „visitor“-Muster

Abhängigkeit des Ausführungskontextes unterschiedliche *visitors*) kann das Produkt die Methode *price* implementieren, ohne etwas über die zur Preisbildung zugrundeliegende Strategie zu wissen. In Abbildung 6.3 läßt sich unmittelbar nachvollziehen, wie eine solche ausgeführt wird. Eine Bestellung muß demnach Subtyp von *PriceVisitorFactory* sein und sich selbst als einzigen Parameter der Methode *price* übergeben. *Price* ruft die Methode *acceptVisitor* auf, indem sie mit Hilfe der Bestellung einen neuen *PriceProductVisitor* erzeugt. Die Bestellung kennt ihren Ausführungskontext, zum Beispiel die Anzahl der Übernachtungen, Anreisetag und den Vertragspartner, d.h. daß ein neuerzeugter *PriceProductVisitor* alle relevanten Informationen, die für eine Preisberechnung benötigt werden, von ihr übernehmen kann.
Dieses Verfahren ermöglicht eine leichte Anpassung oder Umstellung der Preispolitik, ohne daß ein Hotel, wie in Abschnitt 4.2 erläutert, eine unüberschaubare Vielfalt von gleichartigen Produkten pflegen muß, und läßt unterschiedlichste Preisstrategien für Produkte zu.

„Role“

Wie aus Abschnitt 4.5 bekannt, spielen Rollen, die Geschäftspartner annehmen, eine wesentliche Rolle in Bezug auf die betriebswirtschaftliche Funktionalität eines Systems. Es ist jedoch nicht sinnvoll alle rollenbezogenen Daten innerhalb eines Objektes, genaugenommen im Geschäftspartner, zu speichern, denn konkrete Geschäftspartner werden nur einige wenige Rollen innerhalb des Systems einnehmen können .
Im sogenannten *role*-Muster in Abbildung 6.4 ist ein Muster aufgeführt, das im wesentlichen die dynamische Zuordnung von Rollen unterstützt. Als Kern dieses Musters dient das *MultipleRoleObject*, das mit der Methode *registerRole* eine beliebige Anzahl von Rollen über den Rollennamen registriert und diese nach erstmaligen Zugriff eigenständig instanziiert und aggregiert. Auf diese Weise lassen sich dynamisch alle möglichen Rollen eines Objekts instanziieren, um rollenspezifische Daten zu speichern oder zu lesen.
Ein Geschäftspartner ist ein *MultipleRoleObject* und kann die registrierten Rollen „Vertragspartner“, „Gast“, „Benutzer“ und „Angestellter“ auf diese Art und Weise annehmen. Versucht man hingegen einem *MultipleRoleObject* eine nicht registrierte Rolle zuzuordnen, wird eine Ausnahmebehandlung eingeleitet. Mit diesem Muster ist es möglich, Objekten beliebige Rollen zuzuordnen. Es unterstützt deshalb die Erweiterbarkeit des Systems in vortrefflicher Art und Weise. Soll zum Beispiel eine weitere Geschäftspartnerrolle eingeführt werden, sind die auszuführenden Erweiterungen relativ gering. Es muß lediglich eine neue Rolle implementiert und in der Klasse *BusinessPartner* registriert werden. Eine Veränderung oder Anpassung dieser Klasse ist nicht erforderlich.

„Decorator“

Das *decorator*-Muster fügt dynamisch vorhandenen Objekten Funktionalität hinzu und stellt damit eine flexiblere Alternative zur Vererbung dar. Objekte werden demnach mit zusätzlicher Funktionalität dekoriert. Der Vorteil dieses Musters liegt in der dynamischen Erweiterbarkeit, so können Objektverantwortlichkeiten zur Laufzeit zugeteilt oder auch wieder entzogen werden. Im Beispiel des Reservierungssystems wird in Abbildung 6.5 die in einem Hotel auftretende Korrespondenz durch den Einsatz des *decorator*-Musters erweitert. Unterstützt die zur Implementierung verwendete Programmiersprache Mehrfachvererbung, so kann beliebige

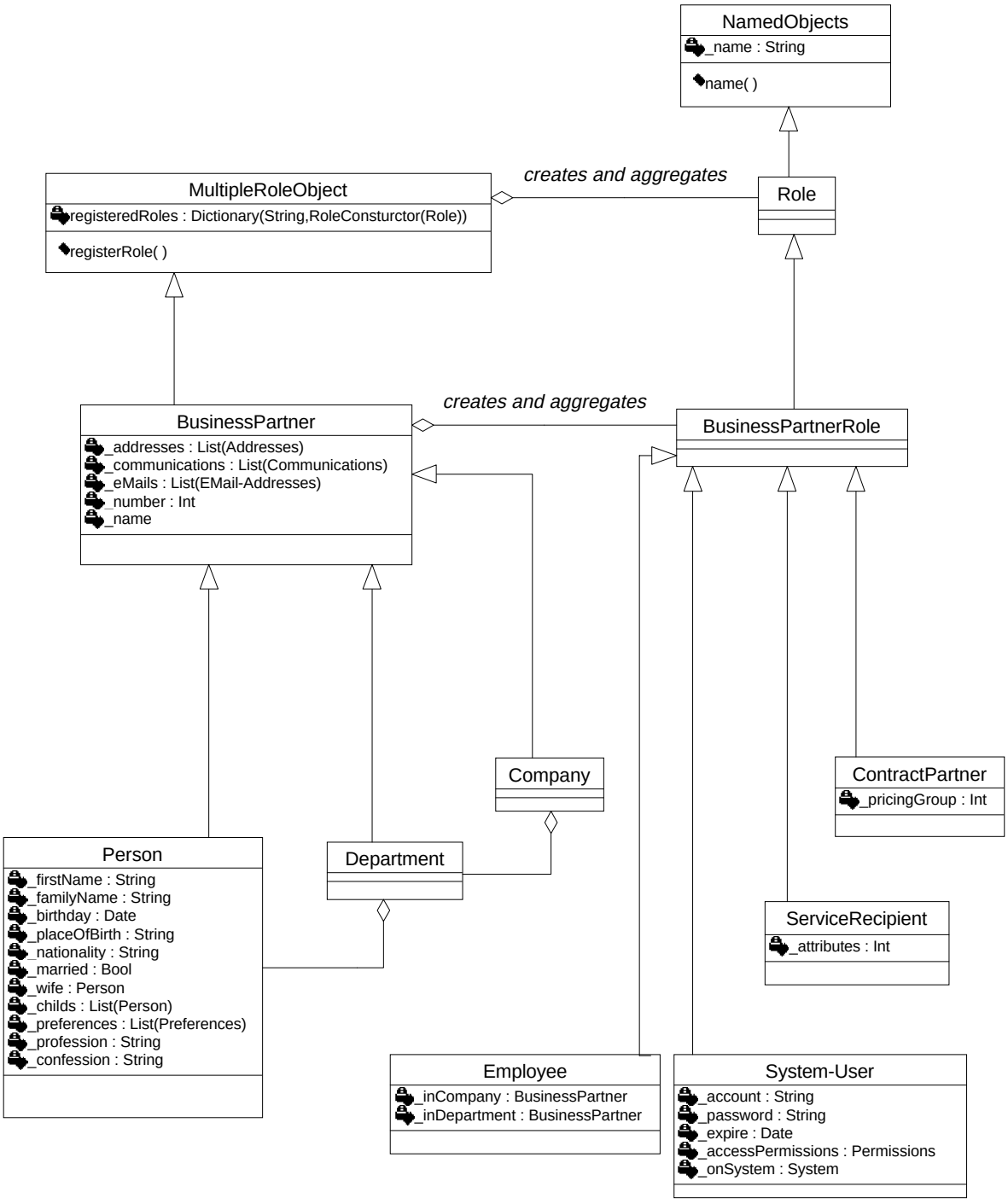

Abbildung 6.4: Das *role*-Muster

Funktionalität anderer *decorators* ererbt und mittels eines neuen *decorator* jeder Korrespondenz dynamisch zur Laufzeit hinzugefügt werden.
Geht man von einem System aus, in dem es schon Korrespondenzklassen wie Anfrage, Angebot, Bestellung und Auftragsbestätigung gibt, so können diese um hotelspezifische Funktionalität erweitert werden, ohne geändert werden zu müssen. Kernidee dieses Musters ist die Benutzung von vorhandenen Objekten durch den *decorator*, der mindestens die Schnittstelle des zu dekorierenden Objektes unterstützt, indem er Methodenaufrufe an das eigentliche Objekt delegiert. Dafür ist nur die Objektreferenz des dekorierten Objektes nötig.
Erweiterte Funktionalität wird im *decorator* implementiert. So werden in diesem Falle (Abbildung 6.5) hotelspezifische Attribute wie Anreisetag, Aufenthaltsdauer und Attribute der Produktanfrage durch den *DecoratorHotelCorrespondence* implementiert. Die Funktionalität, Bestellpositionen aggregieren zu können, stellt der *DecoratorCorrespondenceWithOrderitems* zur Verfügung, der wiederum die Funktionalität einer *PriceProductVisitorFactory* ererbt, die die oben erläuterte kontextabhängige Preisbildung ermöglicht. Durch dieses elegante Muster kann die Korrespondenz, Angebot, Bestellung und Auftragsbestätigung um oben geforderte Funktionalität erweitert werden. Es ist ersichtlich, daß sich die Wiederverwendung von vorhandenen Klassen bezahlt macht. Zudem fällt eine Codeduplizierung in Klassen mit ähnlicher Funktionalität weg, so daß sich die Wartbarkeit des Systems erhöht.

6.2.2 Weitere Klassendiagramme

Drei wichtige Ausschnitte des gesamten Klassendiagramms wurden schon im vorherigen Abschnitt bei der Vorstellung eingesetzter Entwurfsmuster besprochen. Sie verwirklichen im wesentlichen die geforderte Funktionalität von Geschäftspartnern, Hotel-Korrespondenz und Produkten, wie in Kapitel 4.1 gefordert.
Im Anhang C läßt sich das gesamte Klassendiagramm betrachten. Gut zu erkennen ist die Abgeschlossenheit dieses Diagramms, was auf die agentenbasierte Sicht innerhalb des Entwicklungsprozesses zurückzuführen ist. Im wesentlichen stellt das Klassendiagramm den Umfang des Reservierungsagenten innerhalb der im Abschnitt 6.1 vorgestellten Architektur dar.
Als letzter Auszug aus dem Klassendiagramm wird in Abbildung 6.6 die Realisierung der Hotel-Ressourcen veranschaulicht.
Ressourcen werden auch mit der Hilfe des *composite*-Musters realisiert, da sie wie Produkte eine relative stabile Objektstruktur bilden. Eine Ressource besitzt Attribute, die sie beschreiben, und eine von der Ressource selbst abhängende Verfügbarkeit. So haben verbrauchbare Ressourcen eine einfache Bestandszahl, zeitbezogene Ressourcen hingegen einen Verfügbarkeitsplan. Die Verfügbarkeitsprüfung wird mit Hilfe eines *visitors* implementiert. Außerdem wird jede Ressource mindestens einem Verkaufsprodukt zugeordnet.
Gleichartige Ressourcen, d.h. Ressourcen gleicher Attributbeschreibungen, werden in einer *AttributedComposedRessource* gesammelt, was den Vorteil hat, nach Ressourcen mit bestimmten Eigenschaften fragen zu können, ohne den gesamten Bestand aller Ressourcen daraufhin untersuchen zu müssen. Eine Ressource sollte jedoch nicht mit mehr als bis zu zehn verschiedenen Attributen beschrieben sein, da jede mögliche Attributkombination zu einer Klasseninstanz der *AttributedComposedRessource* führt und die Anzahl dieser Instanzen exponentiell mit der Anzahl aller Attribute einer Ressource steigt.

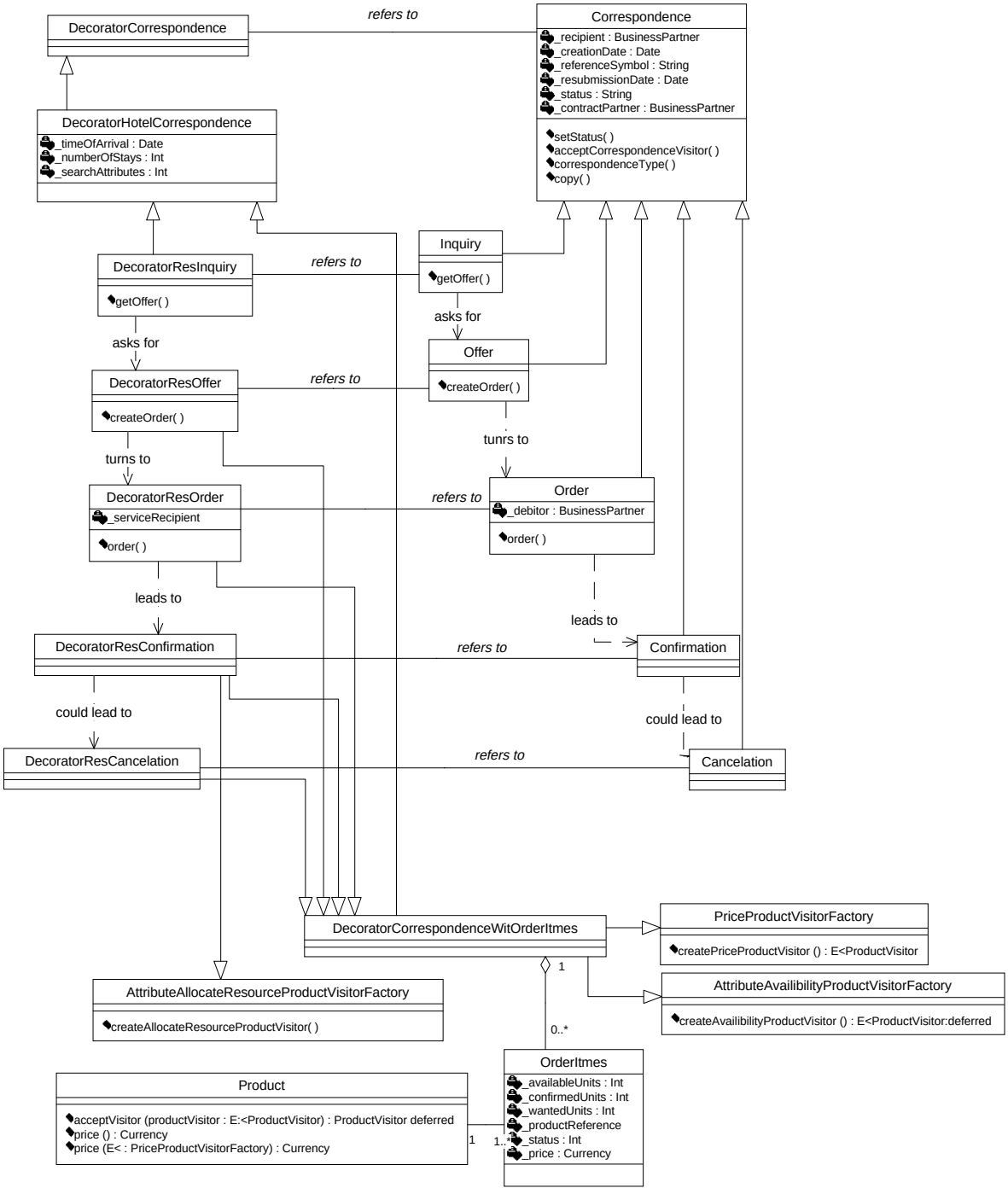

Abbildung 6.5: Das „decorator“-Muster

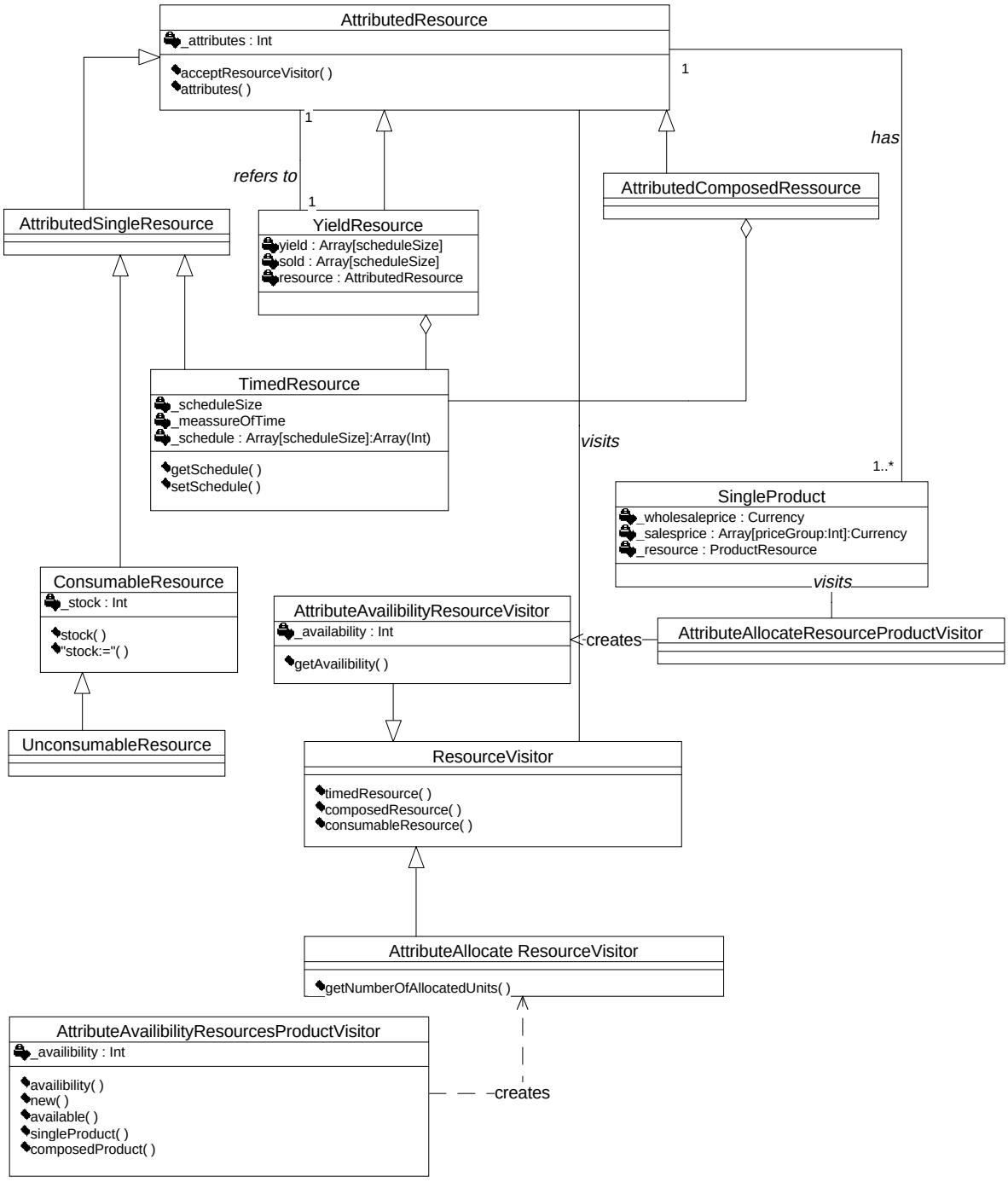

Abbildung 6.6: Das Klassendiagramm über Hotelressourcen

6.2.3 Interaktionsdiagramme

Interaktionsdiagramme sind weitere wichtige Ergebnisse innerhalb der Entwurfsphase. Sie sind anzufertigen, falls Objektinteraktionen nicht trivial sind und deshalb einer Beschreibung bedürfen. So stellt das Zusammenspiel zwischen WWW-Browser und Diensterbringer, in diesem Fall dem Reservierungssystem, während einer geführten Konversation komplizierte Objektinteraktionen dar. Das dazugehörige Interaktionsdiagramm wird allerdings erst im Abschnitt 7.3.3 dargestellt, in dem das gesamte Zusammenspiel zwischen Benutzungsschnittstelle und Agentenumgebung dargelegt wird.

6.3 Bewertung der Entwurfsphase

Die Entwurfsphase wurde geprägt durch den Entwurf von Klassendiagrammen. Grundlegende Fragen bezüglich der Architektur, der zu verwendenden Programmiersprache und der Speicherung von Objekten standen schon als Vorgabe dieser Arbeit fest. Der Einsatz des „CASE-Tools“ von Rational Rose hatte den Vorteil, Klassen-Diagramme relativ schnell in der UML notieren zu können. Innerhalb dieser Arbeit konnten allerdings nicht die gesamten Möglichkeiten von Rational genutzt werden, speziell die Funktionalität zur Programmrumpfgenerierung, da eine Implementierung in der Programmiersprache Tycoon-2 zu realisieren war. Die in UML modellierten Klassendiagramme wurden in weiteren Fachgesprächen unter Kollegen diskutiert. Dabei bewährte sich der Einsatz von Entwurfsmustern. Sie verhalfen den Entwicklern eine gemeinsame „Sprache“ zu sprechen und trugen zu eleganten Lösungen von betriebswirtschaftlichen Problemen bei. Zusätzlich hatte man sich durch den Einsatz von Entwurfsmustern eine bessere Wartbarkeit des Systems und die einfachere Wiederverwendung von Klassen versprochen. Die Wiederverwendbarkeit von Klassen durch den Einsatz von Patterns ließ sich schon beim Entwurf der Klassendiagramme zeigen und später in der Implementierungsphase verifizieren.

Kapitel 7

Implementierung

In diesem Kapitel wird die Implementierungsphase des in dieser Arbeit untersuchten Entwicklungsprozesses beschrieben. Es wird gezeigt, daß sich die Entwurfsmodelle mittels der gewählten Programmiersprache in einen evolutionären Prototypen umsetzen lassen. Wie in der Einleitung von Kapitel 6 erwähnt, wurden die Einschränkungen der gewählten Programmiersprache bei der Erstellung der Entwurfsmodelle mit berücksichtigt, so daß in dieser Phase bezüglich ihrer Umsetzung keinerlei Probleme zu erwarten sind.
Sollte es dennoch Probleme bei der Umsetzung des Entwurfsmodells geben, so liegt ein klarer Entwurfsfehler vor oder es wird versucht, eine andere Programmiersprache zu verwenden, als die, die in der Entwurfsphase vorausgesetzt wurde. Wenn ein Großteil der Klassendiagramme nicht implementierbar ist, handelt es sich um einen schwerwiegenden Fehler innerhalb des Entwicklungsprozesses.
Des weiteren wird näher auf die als „middleware“ bezeichnete Agentenumgebung eingegangen, die die Kooperation zwischen verschiedenen Agenten unter Bereitstellung einer Kommunikationsschnittstelle unterstützt. Es wird gezeigt, wie sich aus HTML-Dialogen Dialogspezifikationen im Sinne der *Business Conversations* ableiten lassen und wie Dialoginstanzen zur Laufzeit über Regeln an die *Business Objects* gebunden werden. Als letzter wichtiger Punkt wird auf sogenannte „Session“-Objekte eingegangen, die sitzungsbezogene Informationen während einer Konversation bereitstellen und den Schlüssel zur Entwicklung von kundenorientierten Informationssystemen liefern. Abschließend folgt eine Bewertung der Implementierungsphase.

Die Umsetzung des Entwurfsmodells findet an mehreren Stellen statt. Einerseits müssen Prozesse und mit ihnen die verbundenen Ein- und Ausgabeoperationen realisiert werden, zum anderen müssen Prozeßzustände innerhalb der *Business Objects*, die in Kapitel 6.2.2 in den Klassendiagrammen als Klassen spezifiziert wurden, gehalten und gegebenenfalls langlebig gespeichert werden. Diese *Business Objects* realisieren betriebswirtschaftliche Zustände eines Systems und werden im folgenden beschrieben. Prozesse hingegen liegen als statisch verkettete HTML-Dokumente vor, die maschinell in eine Konversationsspezifikation im Sinne der *Business Conversations* konvertiert werden müssen, um sie der Agentenumgebung in Form von strukturierten Datentypen zur Verfügung stellen zu können. Näheres dazu im Abschnitt 7.2.
Die Implementierung der *Business Objects* stellt eine grundlegende Aufgabe innerhalb dieser Entwicklungsphase dar. Sie lassen sich genau wie in den Klassendiagrammen spezifiziert mit der objektorientierten Programmiersprache Tycoon-2 [10] implementieren.

7.1 Die Programmiersprache Tycoon-2

Herausragende Eigenschaften von Tycoon-2 sind die Mehrfachvererbung, Funktionen höherer Ordnung und der parametrischer Polymorphismus. Tycoon-2 ist die Weiterentwicklung des objektorientierten Sprachzweigs von Tycoon (**Ty***ped* **c***ommunicating* **o***bjects in* **o***pen e***n***vironments*), die erstmals als Sprache innerhalb von Forschungen über persistente Objektsysteme am Arbeitsbereich Datenbanken- und Informationssystem des Fachbereichs Informatik der Universität Hamburg in [22] eingeführt wurde.
Tycoon-2 ist eine objektorientierte Programmiersprache und wird genauso wie Java von einer virtuellen Maschine ausgeführt. Tycoon-2 lehnt sich an die Laufzeitumgebung seines Vorgängers an. Sie besteht aus einem persistenten Objektspeicher, einer virtuellen Maschine, dem Übersetzer, der den Quellcode in eine für die virtuelle Maschine ausführbaren Bytecode übersetzt und einem Typüberprüfer. Tycoon-2 ist statisch typisiert und unterstützt Mehrfachvererbung von Programmcode, was ein wesentlicher Vorteil gegenüber Java oder andern objektorientierten Sprachen ist, die nur Einfach- und Schnittstellenvererbung unterstützen. Soll eine Klasse mehrere Schnittstellen implementieren, so kommt es bei Klassen mit gleicher Schnittstelle zu einer Kodeduplizierung, was zu einem erheblichen Mehraufwand bei der Systementwicklung und Wartung führt. Dieser Mehraufwand entfällt bei Tycoon-2.
Zudem unterstützt Tyccon-2 die Ausführung von nebenläufigen *threads*, was innerhalb der Agentenumgebung wesentlich ist, da autonome Agenten untereinander kooperieren und ihren eigenen Kontrollfluß besitzen. Dabei wird jeder einzelne Prozeß auf sogenannte *p-threads* abgebildet. Diese *threads* werden vom Betriebssystem verwaltet, was zu einer optimalen Aufteilung der CPU-Zeit führt und nicht zum *overhead* einer weiteren Zeitscheibenverteilung [39].

7.1.1 Mehrfachvererbung

Der intensive Gebrauch von Mehrfachvererbung zeichnete sich schon im Klassendiagrammen der Korrespondenz aus Abbildung 6.5 ab. Exemplarisch wird nun der Tycoon-2-Quellcode der Dekorateur-Klasse *DecoratorCorrespondenceWithOrderitmes* wiedergegeben.

Es ist beeindruckend, daß dieser Dekorateur oder jeder Subtyp von ihm der Korrespondenz Angebot, Bestellung und Auftragsbestätigung die Funktionalität der kontextabhängigen Preisbildung, der Verfügbarkeitsprüfung auf Produkten, der Fähigkeit Bestellpositionen zu aggregieren und zusätzliche hotelspezifischen Attribute, zu verleihen vermag.
Die Funktionalität wird einfach durch seine Superklassen ererbt. Es ist jedoch bei einer Instanziierung eines Objektes dieser Klasse nötig, die entsprechenden Initialisierungsmethoden aller Superklassen aufzurufen. Dieses geschieht in der eigenen Initialisierungsmethode *initDecoratorHotelCorrespondence*. Man sieht, daß zur Initialisierung die hotelspezifischen Attribute übergeben werden. Sie sind auch später für die kontextabhängige Preisbildung relevant. Bei der Initialisierung der *PriceProductVisitorFactory* wird die Preisgruppe des Vertragspartners aus der eigentlichen Korrespondenz, die über den Objektzeiger *reference* erreichbar ist, übergeben. Gut zu erkennen ist auch, wie auf die rollenspezifischen Daten eines Geschäftspartners zugegriffen werden kann. Der von der Korrespondenz referenzierte Geschäftspartner aggregiert seine Rollen, die über den Rollennamen, in diesem Fall *ContractPartnerRole*, zugreifbar sind. Innerhalb der referenzierten Rolle können Attribute oder Funktionen wie bei jedem andern Objekt aufgerufen werden.

```
class DecoratorCorresWithOrderItems
super DecoratorHotelCorrespondence,Dictionary(String,OrderItem),
      AttributeAvailibilityProductVisitorFactory(AttributeAvailibilityProductVisitor),
      BPPriceProductVisitorFactory(BPPriceProductVisitor)
Self  < : DecoratorCorresWithOrderItems
metaclass AbstractClass

public methods
print():Void
{
 referenceToCorrespondence.print,
  self.do(
    fun(orderItem:OrderItem)
      { orderItem.acceptOrderItemVisitor(PPOrderItemVisitor.new)}
   )
}

private methods
initDecoratorCorresWithOrderItems
    (reference:Correspondence,dateOfArrival:Date,numberOfStays:Int,attributes:Int):Self
{
  super.init1(0),
  super.initDecoratorHotelCorrespondence(reference,dateOfArrival,numberOfStays,attributes),
  super.initAttributeAvailibilityProductVisitorFactory(dateOfArrival,numberOfStays,1,attributes),
 super.initBPPriceProductVisitorFactory
      (reference.contractPartner.inRole
       (" ContractPartnerRole",:ContractPartnerRole).priceGroup),

  self
}
```

Abbildung 7.1: „Decorator“ für Hotelkorrespondenzen mit der Funktionalität Bestellpositionen zu aggregieren

```
class DecoratorResOffer
super DecoratorCorresWithOrderItems
Self  <:  DecoratorResOffer

metaclass DecoratorResOfferClass(DecoratorResOffer)
...
  let orderItem:OrderItem=OrderItem.new(aProduct),
  orderItem.name:=aProduct.name,
  orderItem.price:=aProduct.priceFromContext(self),
  orderItem.availableUnits:=aProduct.availibilityFromContext(self),
...
```

Abbildung 7.2: „Decorator“ für Hotelangebote

Wie erwähnt, wird hier die Preisgruppe vom Vertragspartner erfragt und damit die *PriceProductVisitorFactory* initialisiert. Hiernach wird in jedem Fall der Preis eines Produktes für den speziellen Geschäftspartner richtig bestimmt. Technisch gesehen muß das instanziierte Objekt nur der Methode *price* in der Klasse Produkt sich selbst übergeben.
Um dieses zu verdeutlichen, wird ein Ausschnitt der Initialisierung eines Dekorateurs für Angebote auszugsweise gezeigt.

Es handelt sich dabei um den Subtyp *DecoratorResOffer* von dem oben aufgeführten Dekorateur *initDecoratorHotelCorrespondence*. Es wird die Stelle gezeigt, an der von einem verfügbaren Produkt eine Bestellposition mit dem Namen des Produktes, der Anzahl der verfügbaren Produkte und dem spezifischen Produktpreis erzeugt wird. Der Konstruktor von *OrderItem* stellt die Referenz zum zugeordneten Produkt her.

7.1.2 Polymorphismus

Polymorphismus ist eine der wichtigsten Eigenschaften objektorientierter Sprachen. Er unterstützt die gleichförmige Behandlung verwandter Objekte oder Objekte mit gleicher Schnittstelle. Wie im vorherigen Abschnitt gesehen, kann eine Preisberechnung von Objekten durchgeführt werden, die mindestens so speziell wie eine *PriceProductVisitorFactory* sind. Die Methode *price* ist demnach eine polymorphe Funktion mit der Beschränkung mindestes so spezielle Parameter zu bekommen, wie der generellste Datentyp, der als Eingabeparameter dieser Methode genügt. Man spricht hierbei von *bounded polymorphism*. Er wird am Beispiel des Quelltextes der Klasse *DecoratorResOffer* und *Product* verdeutlicht. Der Ausschnitt von Abbildung 7.2 zeigt die Erzeugung einer Bestellposition, die Berechnung des Preises und die Berechnung der Produkt-Verfügbarkeit. Die fettgedruckte Zeile zeigt den Aufruf für die Preisberechnung. Sie verläuft wie oben angedeutet. Es wird die Methode *priceFromContext* aufgerufen und der Kontext des Angebotes übergeben, der aus dem Objekt selbst besteht.
In Abbildung 7.3 wird ein Quelltextauszug der abstrakten Klasse Produkt gezeigt, um die Realisierung der Methode *priceFromContext* aufzuzeigen.

Innerhalb der Methode *priceFromContext* wird nur ein *PriceProductVisitor* erzeugt und der Methode *acceptProductVisitor* übergeben. Der *visitor* durchläuft die Produktstruktur und aggregiert den kontextbezogenen Preis, der nur, wie schon geklärt, vom erzeugten *visitor*

```
class Product
super Named,Described
Self <: Product
metaclass AbstractClass

public methods
acceptProductVisitor(visitor:ProductVisitor):ProductVisitor deferred

priceFromContext(context:PriceProductVisitorFactory(PriceProductVisitor)):Currency
{
  typeCast
        (acceptProductVisitor(context.createPriceProductVisitor),
                              :PriceProductVisitor).price
}
...
```

Abbildung 7.3: Quelltextauszug der Klasse Produkt

abhängt, da dieser die Rechenvorschrift für die Berechnung des Preises in sich trägt.
Leider findet implizit mit der Methode *acceptProductVisitor* ein sogenannter *downcast* statt. Diese Methode kann nämlich nur den generellsten Typ eines *visitors* zurückgeben, was in diesem Falle ein Objekt der Klasse *ProductVisitor* wäre. Deshalb muß ein speziellerer *visitor* nach seinem Besuch wieder auf seinen eigentlichen Typ, hier *priceProductVisitor*, gehoben werden, da sonst die Typisierung statisch falsch wäre.
Es bleibt zu erwähnen, daß man das *typecasting* hätte umgehen können, wenn man die Methode *acceptProductVisitor* so realisiert hätte, daß nur ein *Void* zurückgeben worden wäre. Jedoch ließe sich der Aufruf in der Methode *priceFromContext* nicht mehr in einer Zeile schreiben und man müßte sich die Objektreferenz des erzeugten *visitors* innerhalb dieser Methode merken, um nach einem Besuch des *visitors* seinen berechneten Produktpreis erfragen zu können.
Die hier benutzte Variante veranschaulicht die Funktionsweise des *visitor*-Musters sehr viel deutlicher, erfordert jedoch für eine fehlerfreie Typüberprüfung diesen unschönen *typecast*.

7.2 Generierung von Konversationsspezifikationen

Konversationsspezifikationen werden nicht als Datenstrukturen innerhalb der Realisierung eines Agenten implementiert, sondern mit Hilfe der Prozeßdefinition aus statischen HTML-Dokumenten generiert und zur Laufzeit an entsprechende Agenten gebunden, was deren Rolle innerhalb des Informationssystems bestimmt. Im nächsten Abschnitt wird beschrieben, wie sämtliche Informationen für eine Konversationsspezifikation aus der Struktur eines HTML-Formulares gewonnen werden kann.
Ausgangspunkt für die Generierung von Konversationsspezifikationen bilden die einzelnen Dialogschritte des Prozesses (Kapitel 2.3), die in Form von HTML-Formularen vorliegen. Abbildung 7.4 zeigt einen solchen Dialogschritt. Er soll exemplarisch ein mögliches Angebot auf

eine vorher gestellte Kundenanfrage zeigen. Wie jede andere HTML-Seite läßt sich auch diese mit einem beliebigen HTML-Editor erstellen. Welche Konventionen für die spätere Generierung von Konversationsspezifikationen eingehalten werden müssen, werden am Quelltext des HTML-Formulares in Abbildung 7.5 gezeigt. Nähere Einzelheiten zu HTML finden sich zum Beispiel in [18].
Jede Dialogspezifikation einer Konversationsspezifikation besitzt einen Namen, über den der Agent während einer laufenden Konversation Dialoginstanzen identifiziert. Der Name eines Dialoges wird mit Hilfe der Markierungen $< TITLE > Name < /TITLE >$ festgelegt und steht im Titel des Dokumentes. Die eigentlichen Inhalte eines Dialoges stehen in der $< FORM > Inhalte < /FORM >$ Markierung. Alles was in dieser Markierung steht, wird in dem Dialog zugeordneten *Content* gehalten. Dieser Inhalt aggregiert beliebige Daten eines Dialoges, die wiederum aus zusammengesetzten oder atomaren Daten bestehen können. Atomare Datentypen werden mit Hilfe der Markierung $< INPUT$ Type= "..." $>$ spezifiziert, zusammengesetzte mit Hilfe der Markierungen $< SELECT >, < Option > x1, < Option > x2, ..., < /SELECT >$. Hierbei handelt es sich entweder um einfache oder mehrfache Auswahltypen. Da es innerhalb von HTML keine Typen für Daten gibt, müssen Inhalte, die später in Tycoon-2 als typisierte Daten auftreten, mit Hilfe eines Tricks typisiert werden: Dazu wird der Variablenname des Feldes verwendet, indem der Typ des Datums als Präfix dem eigentlichen Namen vorangestellt wird. Dieser Name referenziert später in der Laufzeitumgebung den eigentlichen Dialoginhalt einer Konversation.

Abbildung 7.4: Dialoginstanz: Angebot auf eine Zimmeranfrage im WWW-Browser

```
< HTML >
< HEAD >
< TITLE > productList < /TITLE >
< /HEAD >

< BODY >
< CENTER >
< IMG SRC= "pict/gif/logo.gif " WIDTH= "189 " HEIGHT= "139 " ALT= "Logo " >
< /CENTER >
< HR >
< H1 >
< EM > Unser Angebot auf Ihre Anfrage: < /EM >
< /H1 >

< FORM METHOD= "GET " ACTION= "nextDialog.stml " >
< INPUT TYPE= "HIDDEN " NAME= "cookie " VALUE= "con1" >
< INPUT TYPE= "HIDDEN " NAME= "Date_arrivalDate " VALUE= "18.07.97 " >
< INPUT TYPE= "HIDDEN " NAME= "Int_numberOfStays " VALUE= "2 " >
Hier unser Angebot für < A NAME= "Int_numberOfStays " > 14 < /A > x1
Übernachtung
ab Anreisetag < A NAME= "Date_arrivalDate " > 01.06.98 < /A > :
< BR >
< BR >
Produktliste:
< BR >
< SELECT NAME= "SingleChoise_String_products " SIZE=4 >
< OPTION > 5x1 angeboten: Doppelzimmer-zwei Betten, 130,00 DM pro Nacht. < /OPTION >
< /SELECT >
< BR >
< INPUT TYPE= "SUBMIT "NAME= "Request_product"VALUE= "Einzelansicht " >
< INPUT TYPE= "SUBMIT " NAME= "Request_searchAttributes" VALUE= "Produktsuche " >
< INPUT TYPE= "SUBMIT " NAME= "Request_confirmation " VALUE= "Bestellen " >
< HR >
< /FORM >

< /BODY >
< /HTML >
```

Abbildung 7.5: Dialogspezifikation in HTML für Dialog „productList“

Der innerhalb der $< FORM >$ Markierung enthaltene Parameter *action* verweist normalerweise auf das Folgeformular eines HTML-Formulares. Bei einer Konversation handelt es sich jedoch um einen dynamischen Ablauf, so daß Folgeseiten dynamisch generiert und nicht durch statische Verkettung referenziert werden können. Deshalb zeigt der Anker *action* immer auf den Folgedialog *nextDialog.stml*, der stellvertretend für jedes dynamisch erzeugte Folgeformular steht. STML ist eine HTML 3.2 kompatible Dokumentenbeschreibungssprache, die um eingebettete Tycoon-2-Ausdrücke erweitert wurde [31]. Der Aufruf dieser URL generiert eine dem Ablauf entsprechende HTML-Seite. Die Generierung erfolgt mit Hilfe sogenannter HTML-Schablonen, die wiederum aus den hier zur Spezifikation des Prozesses verwendeten HTML-Formularen abgeleitet werden.

Nach dem Analysieren der HTML-Formulare werden Schablonen erzeugt, die nur noch HTML-Strukturinformationen enthalten und dem WWW-Server zur Dialogdarstellung zur Verfügung gestellt werden. Beim Ablauf einer Konversation werden diese Schablonen mit den aktuellen Inhalten der Dialoge gefüllt und als vollständiges HTML-Dokument an den WWW-Browser geschickt. Diese Form der dynamischen HTML-Seiten-Generierung ist wesentlich eleganter als die sonst übliche CGI-Programmierung [9], da eine unmittelbare Verknüpfung zwischen HTML-Schablone und Dialogspezifikation besteht. Dadurch können nämlich nur Inhalte auftreten, die vorher im Dokument spezifiziert wurden sind, so daß beim Konvertieren der Dialoginstanzen in HTML-Seiten keine Probleme bezüglich unbekannter oder falsch benannter Daten auftreten können. HTML-Schablone und Dialogspezifikation werden immer als Paar angesehen. Eine aus dem HTML-Dokument in Abbildung 7.5 generierte Dialogspezifikation ist in Abbildung 7.6 auszugsweise wiedergegeben. Der strukturelle Aufbau einer Dialogspezifikation ist in [16, 38] ausführlich erläutert worden.
Zur Dialogspezifikation gehören weiterhin auch Anfragen (*requests*), die innerhalb eines Dialogs möglich sind, um Folgedialoge des Prozesses einzuleiten. Sie werden mit Hilfe der Markierung $< INPUT$ TYPE= $SUBMIT >$ spezifiziert. Der Parameter $NAME$=“*Folgedialog*“ gibt den Namen des Folgedialogs an, dem zusätzlich das Präfix *request* vorangestellt wird. Unveränderliche Inhalte eines Dialoges werden in versteckten Formularfeldern spezifiziert und mit Hilfe der Markierung $< A\ NAME =$ “$Feldname$“ $>< /A >$ innerhalb einer beliebigen Textstelle plaziert. Im Beispiel von Abbildung 7.5 werden so die Anzahl der gewünschten Übernachtungen und der Anreisetag in die Textschablone eingefügt. Mit dieser Methode lassen sich personalisierte Dokumente realisieren, obwohl die Grundstruktur eines Dokumentes als statische Schablone vorliegt.
Es bleibt zu erwähnen, daß jedes Dokument das versteckte Feld *cookie* besitzt, welches jedoch nicht innerhalb der Dialogspezifikation auftaucht. Es wird vom WWW-Server zur Unterbringung des Konversationszustandes benutzt, die er zum korrekten Ablauf einer Sitzung benötigt. Dies ist notwendig, da es sich beim HTTP-Protokoll um ein zustandsloses handelt, das keinerlei Sitzungsinformationen übermittelt, so daß diese Informationen in versteckten Feldern untergebracht werden müssen.

```
CONVSPEC reservation WITH

 DIALOG productList WITH
   REQUEST Einzelansicht TO product END
   REQUEST Bestellen TO confirmation END
   REQUEST Produktsuche TO searchAttributes END
   CONTENT
     Int_numberOfStays :Int
     SingleChoice_String_products :SINGLE CHOICE(nil :String)
     Date_arrivalDate :Date
   END – Content
 END – Dialog

 DIALOG
 ...
 END – Dialog

 DIALOG product WITH
   REQUEST Zurueck TO productList END
   REQUEST Auswaehlen TO productList END
   CONTENT
     SRC_Product :String
     String_formOfAddress :String
     String_lastName :String
     String_productDescription :String
     String_salesPrice :String
     String_productName :String
     Int_outputOfUnits :Int
   END – Content
 END – Dialog
END – CONVSPEC
```

Abbildung 7.6: Generierte Konversationsspezifikation aus HTML-Formularen

7.3 Agentenumgebung

Die Entwicklung von kooperierenden Informationssystemen wird durch die sogenannte *middleware* unterstützt, die alle Aspekte der Kooperation zwischen Agenten abdeckt. Die eigentliche Entwicklung einzelner Agenten sollte sich möglichst auf die betriebswirtschaftliche Funktionalität beschränken. Die in [38] entwickelte Agentenumgebung genügt diesen Anforderungen. Sie stellt autonome Agenten zur Verfügung, die innerhalb ihrer Umgebung Rollen einnehmen. Diese Rollen bestimmen das betriebswirtschaftliche Verhalten eines Agenten und demzufolge auch seine Aufgaben innerhalb des Informationssystems. Eine Rolle wird durch die Anzahl aller unterstützten Konversationen bestimmt, die ein Agent führen darf. Es bleibt zu bemerken, daß ein Agent beliebig viele Rollen einnehmen und innerhalb einer Rolle beliebig viele unterschiedliche Konversationen führen darf.
In den nächsten beiden Abschnitten wird gezeigt, wie mit Hilfe der Agentenumgebung autonome Agenten erzeugt, ihnen Rollen zugeordnet und Bindungen an *Business*-Objekte erstellt werden können.

7.3.1 Rollen- und Regelimplementierung

In Abbildung 7.7 wird gezeigt, wie man mit Hilfe der Agentenumgebung Agenten erzeugt und ihnen spezielle Aufgaben zuteilt. In diesem Fall nimmt der neuerzeugte Agent mit dem Namen *Hotel* die Rolle eines Dienstleisters ein, nämlich die des Reservierungssystems. Diese Rolle ist durch die Konversationsspezifikation *reservation* beschrieben, die von der Fabrik *ConvSpecFactory* nach der Analyse der HTML-Dokumente bereitgestellt wird. Konversationsspezifikationen bestimmten demnach das Rollenverhalten eines Agenten.
Damit der so erzeugte Agent seine betriebswirtschaftliche Funktionalität erbringen kann, müssen noch Regeln der Rolle hinzugefügt werden, die die *Business*-Objekte und den Prozeßablauf der Zimmerreservierung statisch miteinander verbinden. Diese Regeln werden für jede Anfrage (*request*) aller Dialoge einer Konversationsspezifikation erzeugt. Die letzte fettgedruckte Zeile in Abbildung 7.7 zeigt, wie eine Regel, hier *PerfResProductList*, an den Dialog *productList* gebunden wird. Sie wird zukünftig während einer Konversation bei der Anfrage „Einzelansicht“ ausgeführt werden. Der Quelltext dieser Regel ist in Abbildung 7.8 auszugsweise wiedergegeben.
Regeln verknüpfen Prozeßabläufe in Form von Konversationen mit *Business*-Objekte, die die Aufgabe besitzen, betriebswirtschaftliche Funktionalität zu realisieren. Im wesentlichen stellen sie Informationen bereit und verwalten diese in konsistenzerhaltender Art und Weise.
Das Prozeßwissen liegt nicht in den *Business*-Objekten, sondern es wird in den Konversationsspezifikationen gehalten. Eine Kopplung über Regeln ermöglicht eine weitestgehend getrennte Entwicklung und macht Prozesse und *Business*-Objekte unempfindlich gegenüber evolutionären Veränderungen. Veränderungen innerhalb eines Prozesses müssen nicht zwangsweise zu Veränderungen in den *Business*-Objekten führen. Die Regeln bilden demnach eine Schicht zwischen Prozeß und Geschäftslogik, die die Auswirkungen von Änderungen sehr lokal begrenzen können.
Im Beispiel von Abbildung 7.8 ist die Klassenbeschreibung der Dienstleisterregel *PerfResProductList* abgedruckt, die, wie im vorherigen Abschnitt gesehen, dem Dialog *productList* zugeordnet wurde. In dieser Regel werden alle möglichen Fälle von auftretenden Anfragen behandelt. Im folgenden wird das Eintreffen der Anfrage „Einzelansicht“ erläutert.

```
setupHotelAgent :Agent
{
   (*  create hotel-reservation agent *)
   let aAgent:Agent = Agent.new("Hotel"),
   let resConvSpec:ConversationSpec=
      ConvSpecFactory.instance.getCSpec("reservation"),

   (* create a perfomer for the conversation-specification above *)
   let aPerf:PerformerRole(PerfResContext) =
      PerformerRole.new(aAgent,"reservation",resConvSpec),

   (* Add rules to the Performer. *)

      (*  Init  *)
    aPerf.addRule("", resConvSpec.initialRequestName,PerfResInit.new),
   ...
      (*productList*)
   aPerf.addRule("productList","Produktsuche",PerfResProductList.new),
   aPerf.addRule("productList","Einzelansicht",PerfResProductList.new),
   aPerf.addRule("productList","Bestellen",PerfResProductList.new),

     (*product*)
   aPerf.addRule("product","Auswaehlen",PerfResProduct.new),
   aPerf.addRule("product","Zurueck",PerfResProduct.new),
   hotelAgent:=aAgent
}
```

Abbildung 7.7: Festlegung von Rollen autonomer Agenten mit der Erzeugung von On-Regeln für Dialogbindungen an Business-Objekte

```
class PerfResProductList
super PerformerRule(PerfResContext), Trace
metaclass SimpleConcreteClass(PerfResProductList)

public methods

transition( conv: Conversation(PerfResContext),
            dialog :Dialog,
            request: String) :Dialog

  {
   let d:Dialog=nil,
   (request =" Produktsuche" )
    ?{
        ...
    }
    :{(request =" Einzelansicht" )
     ?{
      d:= conv.newDialog(" product" ),
      d.content[" String_lastName" ].str:= conv.s.lastName,
      d.content[" String_formOfAddress" ].str:=conv.s.formOfAddress,
      let offer = conv.s.offer,
      let key:String=dialog.content[ " SingleChoice_String_products" ].sch
       [dialog.content[ " SingleChoice_String_products" ].sch.chosen].spec.name,
      d.content[" SRC_Product" ].str:=offer[key].product.pictsName,
      d.content[" String_productDescription" ].str:=offer[key].description,
      d.content[" String_salesPrice" ].str:=offer[key].price.asString,
      d.content[" String_productName" ].str:=offer[key].name,
      d.content[" Int_outputOfUnits" ].int:=offer[key].outputOfUnits,
      d
     }
    :{(request =" Bestellen" )
...
}
```

Abbildung 7.8: On-Regel zum Binden von Dialoginstanzen an Business-Objekte

Die Regel implementiert die Methode *transition*, die als Parameter die laufende Konversation erwartet und den aktuellen Dialog mit Inhalten und Anfrage. Sie erzeugt aufgrund ihres Ausführungskontextes, der Dialoginhalte und der Anfrage einen neuen Folgedialog, der nach der Erzeugung mit Inhalten gefüllt und über die Kommunikationsinfrastruktur der Agentenumgebung zum Kunden verschickt wird. Als Gegenstück zu den Dienstleisterregeln gibt es die Kundenregeln, die statt der Methode *transition* die Methode *handle* implementieren. Diese Regel verändert Inhalte des zugeschickten Dialoges und wählt eine mögliche Anfrage aus.
Innerhalb der in Abbildung 7.8 aufgeführten Regel wird das aus dem vorherigen Dialog gewählte Produkt identifiziert und dessen individuellen Daten im Dialog „Einzelansicht“ dargestellt. Dieser Dialogschritt ermöglicht es, dem Kunden eine genauere Produktbeschreibung des angebotenen Produkts zu geben. Dabei wird das jeweils aktuelle Angebot vom sogenannten *Session*-Objekt referenziert, das hier unter dem Namen *s* innerhalb der Konversation *conv* zur Verfügung steht. In ihm werden auch Daten des Kunden wie Anrede und Name gehalten. Dadurch können die im vorherigen Abschnitt erwähnten personenbezogenen Dialoge zur Laufzeit erstellt werden.

7.3.2 „Session“-Management

Innerhalb einer Konversation ist der gesamte Ablauf, d.h. alle unter den Gesprächspartnern ausgetauschten Dialoge, bekannt und in einer Historie abgelegt. Der Verlauf einer Konversation bis hin zum aktuellen Dialog wird als Spur bezeichnet, in dem der Kunde und Dienstleister Informationen über den jeweiligen Gesprächspartner erwerben. Viele dieser gewonnenen Informationen werden in späteren Folgedialogen erneut benötigt. Um diese Informationen nicht jedesmal erneut zu erfragen oder zu berechnen, werden diese im sogenannten *Session*-Objekt gespeichert.

```
class PerfResContext

super Object
metaclass SimpleConcreteClass(PerfResContext)

public

  customerID :Int,
  formOfAddress :String,
  lastName :String,
  dateOfArrival :Date,
  numberOfStays :Int,
  offer :DecoratorResOffer
```

Abbildung 7.9: Klasse eines typisierten „Session“-Objektes

Session-Objekte vereinfachen den Ablauf einer Konversation wesentlich, da benötigte Daten innerhalb der Konversation nur einmal erfragt werden müssen und danach bis zum Ende einer jeden Konversation zur Verfügung stehen. Der Gebrauch des *Session*-Objektes wurde schon innerhalb der Regel im vorherigen Abschnitt in Abbildung 7.8 verdeutlicht. Das *Session*-Objekt selbst ist wie in Abbildung 7.9 definiert. Es besteht im wesentlichen aus öffentlichen Attributen, die die wichtigsten Informationen innerhalb einer Konversation widerspiegeln. In diesem Beispiel werden Kundeninformationen gehalten, wie zum Beispiel Kundennummer, Kundenname und Anrede, des weiteren die Eckdaten einer Reservierung, die aus Anreisetag und Anzahl der Übernachtungen bestehen sowie aus dem gegebenenfalls gestellten aktuellen Angebot.

7.3.3 Benutzungsschnittstelle

Die Schnittstelle zwischen Mensch und Maschine ist eine der wichtigsten innerhalb eines jeden Informationssystems. Sie wird in der Regel mit Hilfe einer Benutzungsschnittstelle (*Graphical User Interface GUI*) realisiert. Bei der Entwicklung der Benutzungsschnittstelle ist zu beachten, daß auch hier eine Trennung zwischen der graphischen Oberfläche und der eigentlichen Funktionskomponente, die im wesentlichen die betriebswirtschaftliche Logik darstellt, vorgenommen wird, um die Wartbarkeit und Erweiterbarkeit des Systems zu gewährleisten.
Der in dieser Arbeit gewählte Ansatz liegt in der Konvertierung von Dialoginstanzen einer Konversation von und nach HTML. Die Konvertierung wird von einem Agenten ausgeführt, der innerhalb des Systems seine Aufgabe als WWW-Server wahrnimmt und die in Abschnitt 7.2 erwähnten HTML-Schablonen dazu benutzt, Dialoginstanzen nach HTML zu überführen. Im folgenden wird in Abbildung 7.10 der Ablauf einer Konversation zwischen einem Internet-Benutzer und dem Reservierungsagenten näher erläutert. Auf technische Details des WWW-Servers geht der nächste Abschnitt 7.4.2 ein.
In Abbildung 7.10 sind im wesentlichen drei Akteure zu sehen. Links der Internet-Benutzer vertreten durch seinen WWW-Browser, rechts ein Dienstleister als Reservierungssystem und in der Mitte der WWW-Server, der gleichzeitig als Agent Kunde des Reservierungssystem ist. Eine Konversation wird durch den Aufruf einer bestimmten URL initiiert. Dieser Aufruf führt dazu, daß der WWW-Server *Tycoon-Httpd* den Kunden dazu veranlaßt, eine Konversation zu starten. Der Dienstleister erhält die Nachricht, eine bestimmte Konversation zu führen. Kann er darauf eingehen, so schickt er die Nachricht *acceptConversation*. Der Kunde weiß in diesem Falle, daß beide Agenten eine konforme Konversationsspezifikation unterstützen und fragt mit einer initialen Anfrage nach dem ersten Dialog. Dieser wird vom Dienstleister verschickt und über den Kundenagenten mit Hilfe der generischen Kundenregel *genericCustomerRule* in ein HTML-Formular verwandelt.
Bei dem ersten Dialog handelt es sich in der Regel um eine Systemanmeldung mit Benutzerkennung und Paßwort. Diese Informationen werden vom Benutzer eingetragen und zurück an den WWW-Server geschickt, wo dieselbe Kundenregel den Dialoginhalt und die Anfrage vom Kunden ermittelt. Dieser verschickt beides an den Dienstleister. Dieser Zyklus (in Abbildung 7.10 diagonal grau hinterlegt) wiederholt sich nun bis zum Ende der Konversation oder bis zu ihrem expliziten Abbruch.

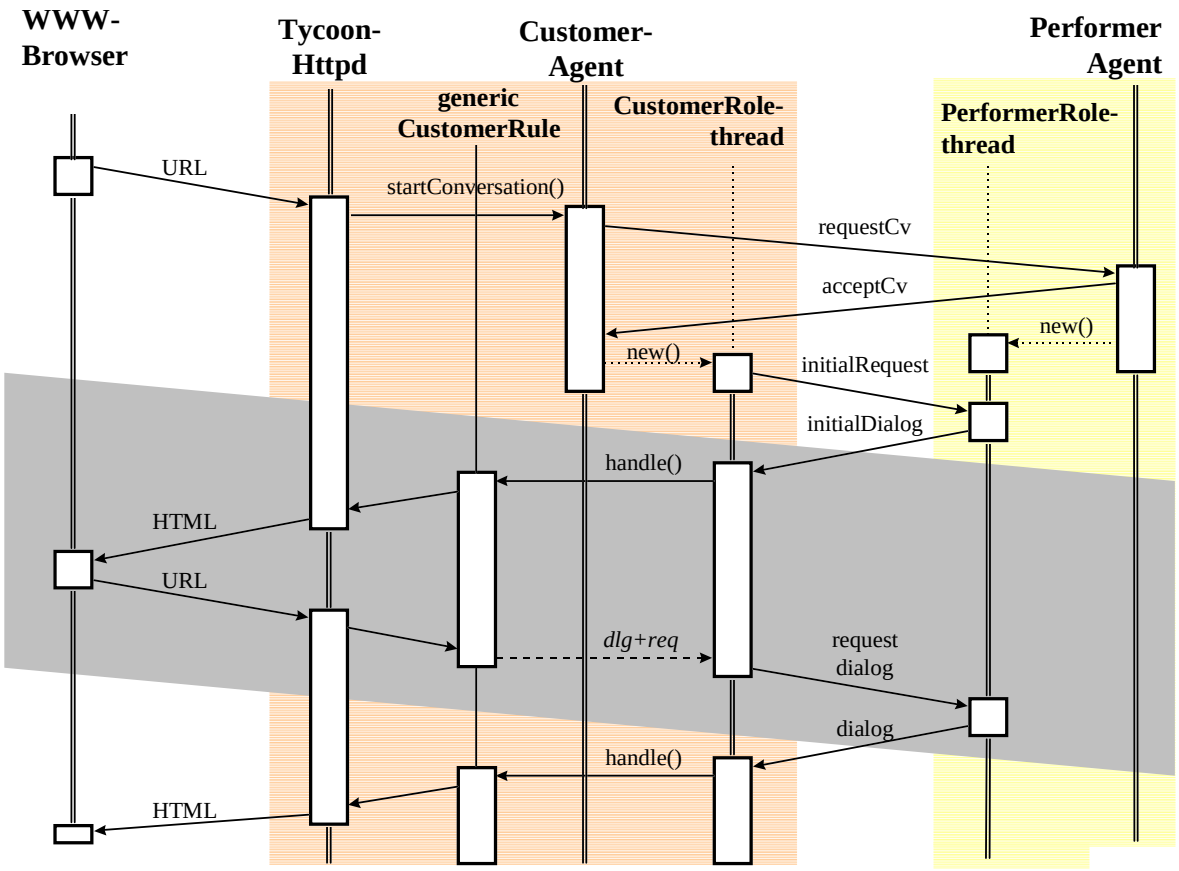

Abbildung 7.10: Ablauf einer „Tycoon Business Conversation“ über das WWW

7.4 Werkzeugunterstützung für einen bruchlosen Software-Entwicklungsprozeß

Zur Durchführung des in dieser Arbeit erläuterten Entwicklungsprozesses sind einige unterstützende Werkzeuge eingesetzt worden. Neben dem *CASE-Tool* von Rational Rose und einem HTML-Editor kam der Konversationsspezifikationsgenerator zum Einsatz, der in Zusammenarbeit mit der Agentenumgebung den beschriebenen bruchlosen Entwicklungsprozeß ermöglicht.
Dieser Generator erzeugt, wie in Kapitel 7.2 angedeutet, aus einem durch HTML-Formulare beschriebenen Geschäftsprozeß eine Konversationsspezifikation, die innerhalb der Agentenumgebung als Vorgabe für eine Ablaufsteuerung zwischen kommunizierenden Agenten eingesetzt wird. Zusätzlich werden aus dieser Prozeßbeschreibung HTML-Schablonen für eine Benutzungsoberfläche erzeugt. Mit Hilfe dieser Methode lassen sich in der Analyse entwickelte Prozeßmodelle ohne großen Aufwand über die Design- und Implementierungphase bis hin in die Laufzeitumgebung der Anwendung übernehmen. Diese Tatsache unterstreicht die Annahme, daß Systeme, die mit dieser Methode entwickelt werden, sich gut auf evolutionäre Prozeßveränderungen anpassen lassen. Die veränderten Spezifikationen müssen lediglich zu

neuen Konversationsspezifikationen übersetzt und mit Hilfe von Regeln an die vorhandene *Business*-Logik gebunden werden. Ein Dienstleister kann diese neue Konversationsspezifikationen seiner Rolle hinzufügen und dadurch sowohl nach wie vor den alten als auch den neuen Prozeß unterstützen, was ein wesentlicher Vorteil ist, da andere unveränderte Kundenagenten wie gewohnt mit dem System kooperieren können. In den folgenden Abschnitten wird auf das Zusammenspiel zwischen Agentenumgebung, Konversationsspezifikationsgenerator und dem WWW-Server als Kunde näher eingegangen.

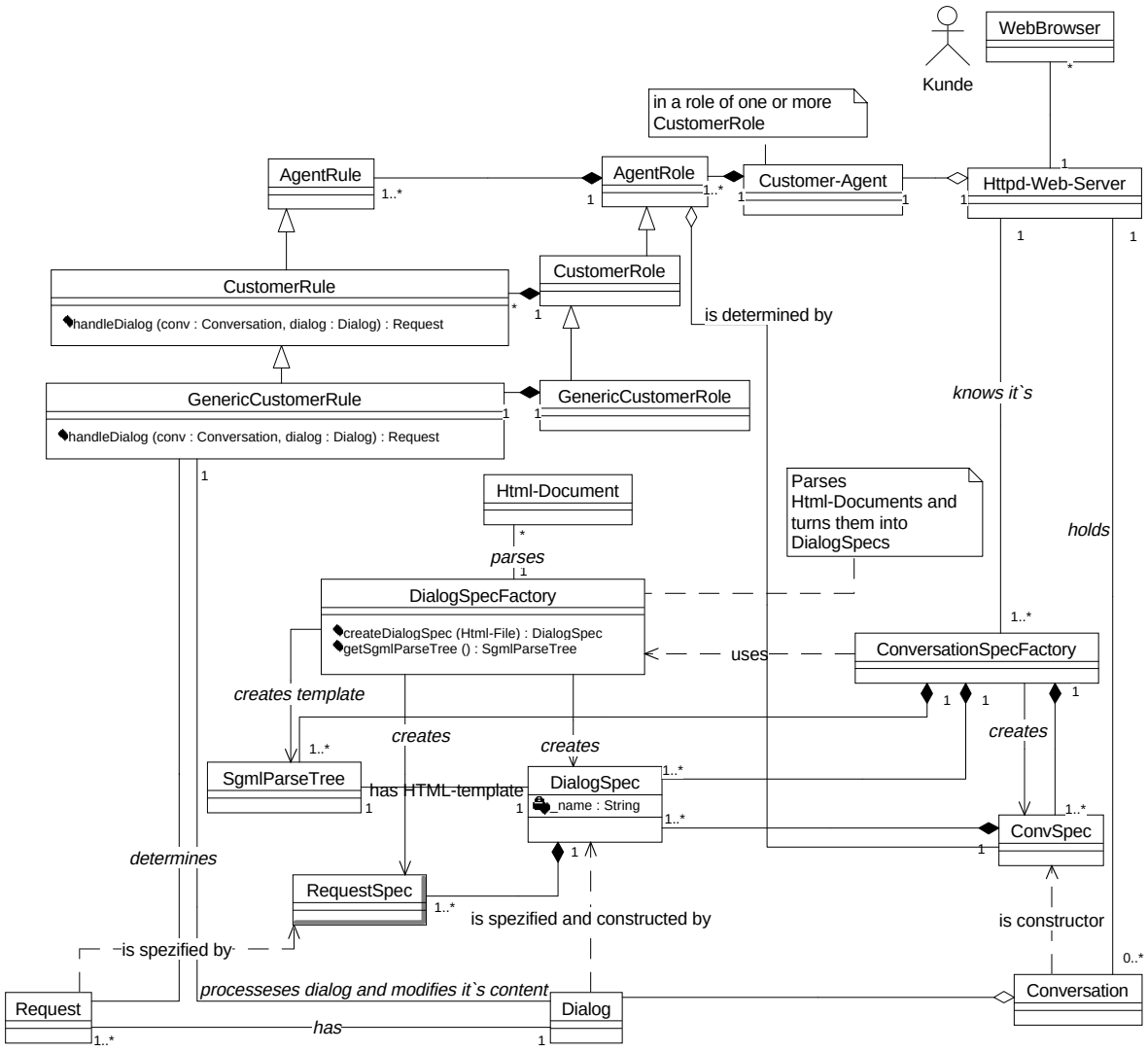

Abbildung 7.11: Zusammenspiel zwischen Konversationsspezifikationsgenerator und der Agentenumgebung

7.4.1 Konversationsspezifikationsgenerator

Um das Zusammenspiel zwischen Konversationsspezifikationsgenerator und der Agentenumgebung nochmals zu verdeutlichen, wird in Abbildung 7.11 das Klassendiagramm der beteiligen Klassen abgedruckt und im folgenden erklärt. Kernstück bildet die Klasse *DialogSpecFactory*. Sie generiert aus einem HTML-Formular ein Objektpaar bestehend aus Dialogspezifikation und HTML-Schablone. Die Klasse *ConversationSpecFactory* benutzt sie, um einen Satz von HTML-Formularen in eine Konversationsspezifikation zu wandeln. Es bleibt zu erwähnen, daß eine Dialogspezifikation Anfragespezifikationen, die auch von der *DialogSpecFactory* generiert werden, aggregiert. Konversationsspezifikationen aggregieren Dialogspezifikationen und werden selbst von Agentenrollen referenziert, die das Verhalten von Agenten festlegen. In diesem Klassendiagramm sind nur Kunden aufgeführt sowie deren speziellen Rollen und Regeln. Eine besondere Bedeutung kommen der generischen Kundenrolle und Kundenregel bei; sie werden aber erst im nächsten Abschnitt im Zusammenhang mit dem WWW-Server erläutert. Weiterhin ist ersichtlich, daß aus der Konversationsspezifikation die Konversationsinstanzen hervorgehen und daß die Dialogspezifikationen Konstruktoren der Dialoge darstellen. Anfragen, die von Kundenregeln her stammen, bilden die treibende Kraft innerhalb der Konversation und damit innerhalb des Geschäftsprozesses. Eine genauere Beschreibung der Beziehungen zwischen Agenten, Konversationen, Spezifikationen und Inhalten ist in [38] wiedergegeben und kann hier nicht näher behandelt werden.

7.4.2 Generischer Kunde und WWW-Server

Der WWW-Server tritt als Kundenagent innerhalb der Agentenumgebung auf. Er wird als generischer Kunde bezeichnet (Abbildung 7.11), weil er mit Hilfe einer generischen Kundenregel jeden Dialog einer beliebigen Konversation zu beantworten vermag, im Gegensatz zu anderen Kundenagenten, die für jeden Dialog eine spezifische Kundenregel bereitstellen und die auszuführenden Aktionen innerhalb der Methode *handleDialog* implementieren müssen.
Die generische Kundenregel hat die Aufgabe, Dialoge einer Konversation dem menschlichen Kunden mittels der Übersetzung nach HTML über einen WWW-Browser anzuzeigen und Inhaltsänderungen und Anfragen entgegenzunehmen, um diese dann wiederum der generischen Kundenrolle zukommen zu lassen. Dieses geschieht in Form eines Dialogs mit geänderten Inhalten und der entgegengenommenen Anfragen. Aktionen dieser Art sind in der überschriebenen Methode *handleDialog* innerhalb der generischen Kundenregel implementiert und sehr gut im Interaktionsdiagramm von Abbildung 7.10 erkennbar.
Damit der WWW-Server Dialoginstanzen nach HTML übersetzen kann, benötigt er HTML-Schablonen zugehöriger Dialogspezifikationen. Diese kann er über die Instanz der *conversationSpecFactory* erreichen, die die *SgmlParseTrees* aggregiert. *SgmlParseTrees* enthalten Strukturinformationen der HTML-Formulare und werden mit den aktuellen Inhalten direkt als *HTML-Stream* an den *WWW-Browser* geleitet, was zur unmittelbaren graphischen Ausgabe der Dialoge innerhalb des *WWW-Browsers* führt.
Ein noch zu entwickelndes Werkzeug wird in der Studienarbeit von Patrick Hupe [12] realisiert. In dieser Studienarbeit wird der Tycoon-2 Übersetzer dahin erweitert, daß Inhaltsspezifikationen von Dialogen bei der Übersetzung von Regeln mit einbezogen werden können, d.h. Namen, Typen und Sichtbarkeitsbereiche von Dialoginhalten sind dem Übersetzer vor der Übersetzung von Regeln bekannt, so daß referenzierte Inhalte aus dem aktuellen Dialog oder aus vorherigen Dialogen Syntax- und Typgeprüft werden können. Dadurch lassen sich

schon während der Übersetzung Syntaxfehler bei Variablennamen erkennen. Außerdem wird sichergestellt, daß referenzierte Inhalte zum Ausführungszeitpunkt dem Agenten vorliegen. Dieses kann anhand der Spur einer möglichen Konversation festgestellt werden. Liegen diesbezügliche Informationen noch nicht vor, so wird bei der Übersetzung der Regel ein Fehler ausgegeben, da solche fehlenden Inhalte zur Laufzeit zu einem Wert *nil* evaluieren, der wiederum zu Laufzeitfehlern führen könnte.
Faßt man die hier vorgestellten Werkzeuge samt der TBC-Agentenumgebung in eine gemeinsame Entwicklungsumgebung zusammen, so läßt sich eine komfortable Entwicklungsplattform erstellen, die den Ansprüchen zukünftiger, agentenbasierter und kooperativer Informationssysteme genügt.

7.5 Bewertung der Implementierungsphase

7.5.1 Lokalität und Wiederverwendbarkeit von Geschäftsprozeßspezifikationen

In der Implementierungsphase des in dieser Arbeit verfolgten Entwicklungsprozesses hat sich gezeigt, daß sich mit der hier eingesetzten Entwicklungsmethode und den unterstützenden Werkzeugen eine bruchlose und an betriebswirtschaftliche Kernprozesse sich anlehnende Software-Entwicklung durchzuführen ist. Sie wurde am Beispiel der Zimmer-Reservierung nachvollzogen.
Besonders gut bewährte sich die Steuerung von autonomen Agenten über Konversationsspezifikationen. Sie bildeten die Basis des Kooperationsmodell autonomer Agenten und führten auf die Idee, Prozesse aus einer Aneinanderreihung von Dialogschritten zu modellieren, die letztlich auf den Konversationsspezifikationsgenerator führten, der aus den HTML-Prozeßbeschreibungen Konversationsspezifikationen generierte und gleichzeitig in Verbindung mit dem Tycoon-WWW-Server die Benutzungsoberfläche realisierte. Der extrem geringe Aufwand für die im Prototypen eingesetzte Benutzungsoberfläche soll hier noch einmal hervorgehoben werden. Sie fiel sozusagen als Nebenprodukt durch Wiederverwendung der in der Analysephase erzeugten Geschäftsprozeßspezifikationen ab. Diese Tatsache belegt, daß das Modell der *Business Conversations* den Entwicklungsprozeß über alle Phasen der Entwicklung durchgängig unterstützt und eine sehr anwendungsnahe Modellierung zuläßt. Des weiteren zahlte sich die lose Kopplung zwischen Prozeß und der *Business*-Logik mit Hilfe von Regeln aus, die Änderungen innerhalb des Geschäftsprozesses auf nur sehr lokale Änderungen innerhalb des Systems begrenzen konnten. Das Einführen von weiteren Diensten fiel mit Hilfe neuer Konversationsspezifikationen sehr leicht und hatte keinerlei unmittelbare Auswirkung auf bestehende Kundenagenten, da ihre alte Spezifikationen für den korrekten Ablauf weiterhin unterstützt wurde. Es wurden lediglich neue Konversationsspezifikationen erzeugt und an Rollen eines Agenten gebunden, der dadurch einen neuen Dienst für andere Agenten anbieten konnte.

7.5.2 Agentenumgebung

Das Modell der Business Conversations und die auf ihm aufbauende Agentenumgebung trägt einen erheblichen Teil zum Erreichen oben aufgeführter Ergebnisse bei. Neben einem Kooperationsmodell für autonome Agenten ließen sich mit dem Modell kundenorientierte Geschäftsprozesse realisieren, die gleichzeitig die Einbindung von alten betriebswirtschaftlichen Systemen

und damit ihre Wiederverwendung unterstützte. Auf diesem Wege können monolithische Systeme innerhalb einer agentenbasierten Architektur mit aufgenommen werden, so fern sie eine Schnittstelle für den Zugang auf ihre betriebswirtschaftlichen Daten unterstützen, über die ein Dienstleisteragent darauf zugreift. In einer Folgearbeit zu diesem Thema wird eine Anbindung an das SAP R/3-System über Funktionsaufrufe realisiert und dessen Eignung letztlich bewertet. Innerhalb dieser Arbeit wird der Bau von Satellitensystemen als durchführbarer Ansatz gesehen, alte Systeme zu erweitern und in neue kooperative Systemarchitekturen zu integrieren. Neue Möglichkeiten zur Kooperation werden mobile Agenten mit sich bringen, die auf dem Kooperationsmodell der *Tycoon Business Conversations* aufbauen. Diese oder ähnliche Architekturen werden zukünftige Systeme im Bereich der *Computer-Supported Cooperative Work* (CSCW) revolutionieren und speziell ihren Zugang innerhalb von Workflowsystemen finden. Die Gleichbehandlung von Mensch und Maschine innerhalb solcher Architekturen wirkt sich beim Bau von kooperativen Systemen als sehr vorteilhaft aus, da keine Unterscheidung oder Sonderbehandlung im Bezug auf die Systemkonfiguration vorgenommen werden muß. Ob ein Agent eine Aufgabe erledigen kann oder nicht, hängt allein von den Konversationsspezifikationen ab, die er unterstützt. Generell ist es nicht ersichtlich, ob eine erbrachte Leistung von einem Menschen oder einer Maschine stammt, sofern sie konform mit ihrer Spezifikation ist.

7.5.3 Tycoon-2

Tycoon-2 eignete sich vorzüglich für die Umsetzung aller hier vorgestellten Designmodelle. Sie konnten unverändert implementiert werden und zeichneten sich aus durch ein gutes Laufzeitverhalten. So waren in einem Testlauf des Zimmer-Reservierungssystems mit mehr als tausend Zimmern keine wahrnehmbare Verzögerungen gegenüber bei einer kleineren Konfiguration bei der Verfügbarkeitsberechnung von Ressourcen festzustellen. Bei der Umsetzung der Modelle zahlte sich besonders die Möglichkeit von Tycoon-2 aus, mehrfach zu vererben. Sie beschleunigte die Implementierung spürbar und stellt im Vergleich zu andern objektorientierten Sprachen eine erhebliche Erleichterung für den Entwickler dar.
Tycoon-2 ist mehrprozeßfähig und unterstützt zusätzlich die orthogonale Persistenz von *threads*, Code und Objektinstanzen. Besonders die Mehrprozeßtechnik zahlte sich bei der Entwicklung der Tycoon-2-Agentenumgebung aus, in der jeder Agent und jede laufende Konversation eines Agenten durch einen eigenen laufenden *thread* ausgeführt wird. Mehr dazu findet sich ausführlich in [38]. Eine Lastverteilung auf mehrere Prozessoren ist bei Mehrprozessorsystemen ohne weiteres denkbar.

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

In dieser Arbeit wurde eine prozeßorientierte Software-Entwicklungsmethode, die sich in den Grundzügen an Jacobsons *a use case driven approach* anlehnt, vorgestellt und anhand der Entwicklung eines Zimmerreservierungssystems nachvollzogen. Dabei bildeten die zu unterstützenden Kernprozesse den Ausgangspunkt der Entwicklung, in diesem Falle der Reservierungsprozeß. Es zeigte sich, daß diese Methode sowohl die Geschäftsprozeßmodellierung als auch die Systementwicklung über die Entwicklungsphasen der Analyse, Entwurf und Implementierung sehr gut und lückenlos unterstützt, was anhand der durchgängigen Wiederverwendung von Gechäftsprozeßspezifikationen innerhalb des gesamten Entwicklungsprozesses belegt werden konnte. Als zugrundeliegendes Kooperations- und Architekturmodell diente das Modell der *Business Conversations* [26], das implementiert in der sogenannten Agentenumgebung [38] zur Verfügung stand. Hierbei handelt es sich um eine Systemarchitektur, in der autonome Agenten zielgerichtete Konversationen führen, um gemeinsame Aufgaben zu lösen oder gemeinsame Arbeitsabläufe zu unterstützen. Sie bot gleichzeitig einen Ansatz zur Erweiterung von altbewährten Software-Systemen, bei denen neue zusätzliche Funktionalität in sogenannten Satellitensystemen realisiert wird, ohne das bestehende System verändern zu müssen. Mit der Implementierung des Reservierungssystems konnte nachvollzogen werden, daß die Verfolgung dieses Ansatzes zur Wiederverwendung von Software-Systemen beiträgt. So können auf diesem Wege große verteilte Systeme durch Komposition von autonomen über Konversationen gekoppelte Agenten realisiert werden.
Die kommerzielle Nutzung des WWW als günstigen Vertriebskanal und die Ausrichtung von Geschäftsprozessen an Kundenbedürfnisse verspricht neue Absatzmärkte und eine starke Kundenbindung. Dieses Wissen wurde beim Entwurf des Zimmerreservierungssystems mit berücksichtigt. Die Entwicklung des Reservierungssystems durchlief alle wichtigen Phasen des verfolgten Entwicklungsprozesses. Ergebnisse der Analysephase waren der Anforderungs- und *Use-Case*-Katalog, die die Anforderungen an das System und die üblichen Anwendungsfälle beschreiben. Die Entwurfsphase lieferte eine Reihe von Klassen- und Interaktionsdiagrammen, die mit Hilfe eines *CASE-Tools* in der UML notiert wurden. Beim Entwurf der Klassendiagramme wurden gängige Entwurfsmuster zur Realisierung der Anforderungen eingesetzt. Die in der Entwurfsphase entstandenen Klassenmodelle konnten innerhalb der Implementierungsphase mit der Programmiersprache Tycoon-2 problemlos umgesetzt werden. Zusätzlich konnte durch Wiederverwendung der in HTML beschriebenen Geschäftsprozeßspezifikationen

eine *browser*-gestützte Benutzungsoberfläche entwickelt werden. Fazit dieser Arbeit ist, daß sich mit Hilfe des beschriebenen prozeßorientierten Entwicklungsprozesses und der Wiederverwendung von Prozeßspezifikationen sehr flexible kooperative Informationssysteme entwickeln lassen, die auf evolutionäre Veränderungen von Geschäftsprozessen reagieren und dadurch dem permanenten organisatorischen Wandel eines Unternehmens Rechnung tragen.

8.2 Ausblick

Weiterführende Arbeitsthemen lassen sich grob in drei Themenbereiche einteilen: Verfeinerung des Entwicklungsprozesses durch Einsatz spezieller Werkzeuge, Weiterentwicklung der Agentenumgebung und Anwendungsentwicklung innerhalb von agentenbasierten Architekturen.
In den ersten Bereich fällt die Studienarbeit von Patrick Hupe [12], in der der Tycoon-2-Übersetzer zur Analyse von Konversationsspezifikationen erweitert wird, um Sichtbarkeitsbereiche und Namen von typisierten Dialogvariablen herauszufinden, damit Regeln, die Konversationen zur Laufzeit an *Business*-Objekte binden, typ- und ablaufsicher übersetzt werden können. Weitere Untersuchungen bezüglich der Konformität von Konversationsspezifikationen sollten diese Arbeit ergänzen und zeigen, daß Konversationsspezifikationen in Analogie zu typisierten Objekten stehen. Grundlegende Gedanken darüber werden in der Arbeit von Holm Wegner [38] geäußert. Zusätzlich sollte in einer weiterführenden Arbeit die Erweiterbarkeit von Prozeßspezifikationen und die Auswirkungen von Erweiterungen auf laufende Systeme oder auf kooperierende Agenten untersucht werden.
Der zweite Themenbereich befaßt sich mit der Weiterentwicklung der Agentenumgebung. Die in dieser Arbeit verwendete und in Tycoon-2 implementierte Agentenumgebung unterstützt keine mobile Agenten, wie sie etwa in [16] beschrieben worden sind. Mit einer Erweiterung dieser Umgebung um mobile Agenten könnten die in dieser Arbeit skizzierten verteilten Anwendungen realisiert werden, in denen es Vermittlungsdienste gibt, die mobile Agenten zu ihren potentiellen Dienstleistern innerhalb der heterogenen offenen Anwendungsarchitektur weiterleiten, wo sie eigenständig ihre Aufgaben wahrnehmen. Eine Realisierung von neuartigen agentenbasierten *Workflow*-Systemen würde möglich werden.
Weitere offene Fragen gibt es zur konsistenten Datenhaltung in verteilten agentenbasierten Systemen und zur zuverlässigen Einbindung von Standardsystemen, wie das SAP R/3-System, und fallen in den dritten Themenbereich.
Transaktionale Konzepte, die die sogenannten ACID-Anforderungen erfüllen, d.h. die eine Folge von Änderungsoperationen gegen konkurrierenden Zugriff isolieren und im Fehlerfall entweder vollständig oder gar nicht einbringen, sind nicht mit den herkömmlichen Methoden zu realisieren. Ein möglicher Ansatz zur Erreichung transaktionaler Sicherheit könnte im Führen von kompensierenden Aktionen liegen oder in der Unterstützung von langanhaltenden Aktionen mittels *ConTracts* [36], deren verallgemeinerter Kontrollmechanismus oberhalb klassischer Transaktionen liegt und den konsistenten Ablauf von Geschäftsprozessen innerhalb von großen verteilten Anwendungen zusichert. Als nächster Schritt wird die Kopplung des SAP R/3-Systems mit der Hilfe von entfernten Funktionsaufrufen [20] an das Agentensystem erfolgen. Agenten exportieren dann in Gestalt von Satellitensystemen SAP R/3-Anwendungsfunktionalität oder erweitern diese. Es bleibt abzuwarten, wann große, kommerziell relevante agentenbasierte Informationssysteme unter Einbindung altbewährter ans Netz gehen.

Anhang A

Anforderungskatalog

A.1 Schlüsselabstraktionen

Schlüssel-abstraktion	Verantwortlichkeiten	Beziehungen
Hotel	Hat organisatorische und legale Verantwortung für den Betrieb des Hotels, tritt Gästen als Auftragnehmer gegenüber. Intern kontrolliert es seine Abteilungen, extern ist es auskunftspflichtig, (z.B. dem Finanzamt und dem Meldeamt).	• Rahmen des Gesamtsystems
Reservation-Manager	Nimmt Reservierungsanfragen von Gästen oder Firmen entgegen, bearbeitet Angebote und führt Reservierungsaufträge bzw. Änderungen dieser Aufträge aus.	• Geschäftspartner • Korrespondenz: Anfragen, Angebote, Bestellungen, Bestätigungen ...
Geschäfts-partner	Identifiziert in eindeutiger Weise eine Person oder Firma mit zugeordneten Namen und Anschrift. Nimmt innerhalb des Systems rollenspezifische Aufgaben wahr (Vertragspartner, Auftraggeber, Auftragnehmer, Gast, Kontakt ...).	• Geschäftskonditionen • Preise
Produkt	Beschreibt eine Dienstleistung oder einen verkaufbaren Artikel. Besitzt einen Preis.	• Ressourcen • Verfügbarkeit • Preis

Schlüssel-abstraktion	Verantwortlichkeiten	Beziehungen
Preis	Der Produktpreis berücksichtigt die lokalen Steuergesetze, er ist vom Geschäftspartner, von der Saison und von der Buchungsdauer abhängig.	• Produkte • Preise innerhalb des Bestellkontextes • Geschäftspartner • Anfragen
Ressource	Eine Ressource bestimmt die Verfügbarkeit in Zeit und Menge eines physikalischen Objektes innerhalb des Hotels.	• Produkte • Physikalische Objekte: Zimmer, Tennisplätze, Konferenzräume, ...
Zimmer	Können Suiten, Doppelzimmer, Einzelzimmer oder ähnliches sein. Besitzen eine Status (belegt, sauber, kaputt) und werden durch Attribute beschrieben (schöne Aussicht, Nichtraucher, großes Bett, blaue Tapeten ...).	• Ressourcen
Anfrage	Initiierendes Ereignis eines Buchungsprozesses. Führt in der Regel zu einem Angebot von Produkten. Löst eine Verfügbarkeitsprüfung aus.	• Geschäftspartner • Produkte • Angebote
Angebot	Reaktion auf eine Anfrage, bietet Liste von Produkten an. Überprüft dabei im Vorfelde deren Verfügbarkeit und bestimmt nicht bindende Preise. Bonitätsprüfung des Auftraggebers ist möglich.	• Geschäftspartner • Produkte
Bestellung	Einverständnis über ein Angebot mit der Verpflichtung, dieses wahrzunehmen.	• Auftraggeber, Auftragnehmer • Angebot • Produkte mit Preisen • Zahlungsmodalitäten

A.2 Auszug aus dem „use case“-Katalog

„use case“	Beschreibung
Berechne Verfügbarkeit einer Ressource	Bestimmt die Verfügbarkeit einer Ressource, dabei kann die Verfügbarkeit von der Anzahl der sich im Lager befindlichen Produkte abhängen, oder falls es zeitbezogene Ressourcen sind, vom Buchungszeitraum und der ihm zugeordneten Belegungsrate.
Suche verfügbare Produkte	Führt eine attributbezogene Suche auf Produkten durch. Liefert alle Produkte deren Verfügbarkeit größer Null ist und die den Suchattributen genügen.
Berechne Produktpreis	Berechnet den Produktpreis, dabei kann es Variationen in der Berechnung geben. Der Preis ist vielleicht abhängig vom Vertragspartner, von der Saison oder von der Länge des Buchungszeitraumes.
Erzeuge Bestellpositionen	Erzeugt ausgehend von einer Produktpalette die Bestellpositionen eines Angebotes.
Suche Geschäftspartner	Sucht anhand eines Namens oder einer Kundennummer den entsprechenden Geschäftspartner samt seinen assoziierten Rollen.
Lege Vertragspartner fest	Bestimmt einen Geschäftspartner in der Rolle des Vertragspartners. Dabei spielen die rollenspezifischen Daten eines Geschäftspartners eine wichtige Rolle.
Erstelle Angebot im Anfragekontext	Erstellt ein Angebot aus der Kundenanfrage. Dabei fließen Informationen aus dem Kontext der Anfrage mit ein. Zum Erzeugen des Angebots werden die oben aufgeführten Anwendungsfälle mitbenutzt.
Erzeuge Ressource	Erzeugt innerhalb des Systems eine Ressource. Eine Ressource besitzt eine Verfügbarkeit und eine Attributbeschreibung.
Erzeuge Produkt	Erzeugt innerhalb des Systems ein Produkt. Ein Produkt ist einer Ressource zugeordnet und besitzt einen Preis.
Lege Geschäftspartner an	Legt einen Geschäftspartner innerhalb des Systems an. Berücksichtigt dabei jede von ihm eingenommene Rolle.
...	...

Anhang B

Analyse-Klassendiagramm

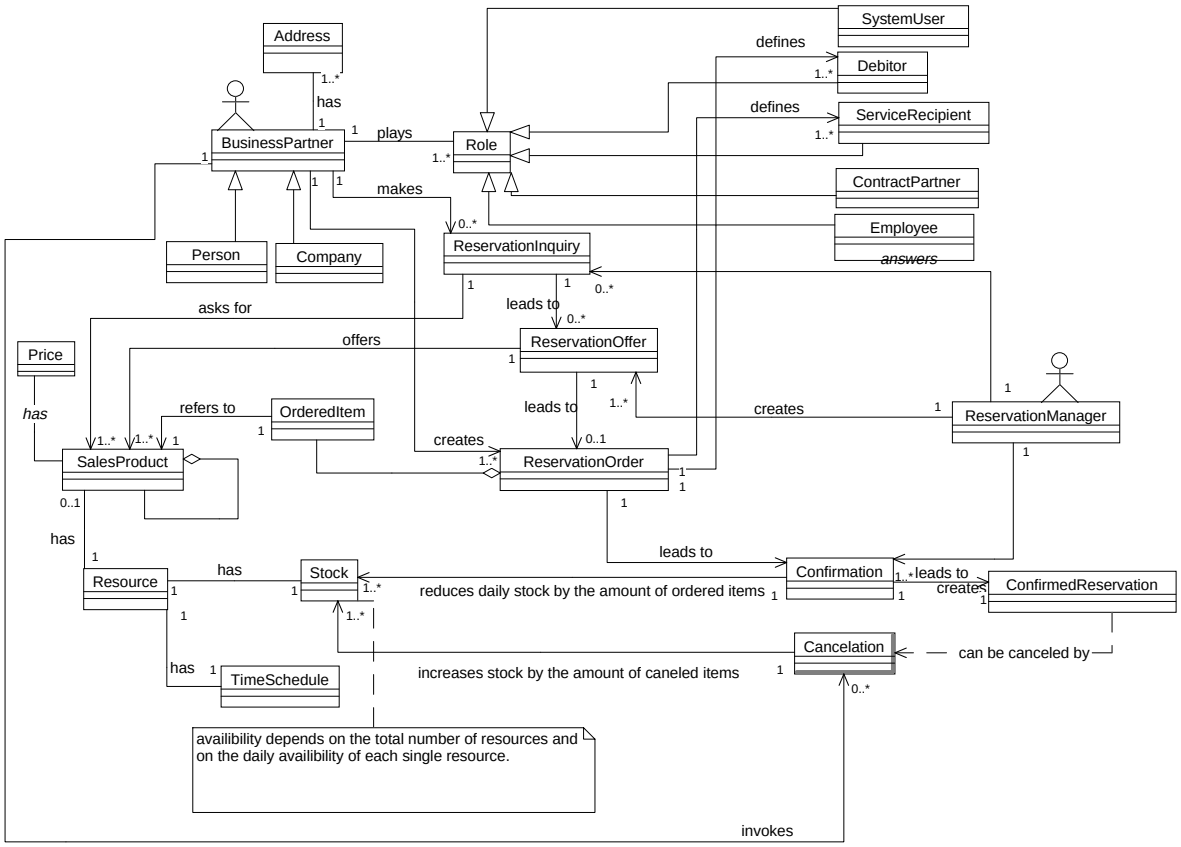

Abbildung B.1: Analyse-Klassendiagramm des Reservierungssystems

Anhang C

Design-Klassendiagramm

Literaturverzeichnis

[1] Grady Booch. *Object-Oriented Analysis and Design Second Edition.* The Benjamin/Cummings Publishing Company, Inc, Redwood City, California, 1994.

[2] Rainer Burkhardt. *UML-Unified Modeling Language.* Addison-Wesley, Bonn, 1997.

[3] Nilüfer Caliskan. Konversations- und Agentenorientierte Modellierung von Geschäftsprozessen eines Informatik-Arbeitsbereichs unterstützt durch das Intranet, 1997. Universität Hamburg, Fachbereich Informatik, Arbeitsbereich STS, TU-Harburg, Studienarbeit.

[4] Luca Cardelli. *A theory of objects.* Springer Verlag, New York, Inc., New York, 1996.

[5] Peter Coad and Edward Yourdon. *Object-Oriented Analysis.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[6] Fernando Flores, Michael Graves, Brad Hartfield, and Terry Winograd. Computer Systems and the Design of Organisational Interaction. *ACM Transaction on Office Information Systems*, 6(2):153–172, April 1988.

[7] Martin Fowler. *uml distilled.* Addison-Wesley Longman, Inc, Massachusetts, 1997.

[8] Erich Gamma. *Design Patterns: Elements of reusable object-oriented software.* Addison-Wesley Publishing Company, Inc., Massachusetts, 1990.

[9] Eric Herrmann. *CGI prgramming with perl in a week.* Sams.net Publishing, 201 West 103rd Street, Indianapolis, Indiana, 1996.

[10] Higher-Order. www.sts.tu-harburg/projects/tycoon2/entry.html. WWW-Server, Arbeitsbereich STS, TU-Harburg 1997/98.

[11] John E. Hopcroft. *Introduction to automata theory, languages and computation.* Addison-Wesley Publishing Company, Inc., Wokingham, England, 1979.

[12] Patrick Hupe. Zustandstypisierung. Universit t Hamburg, Fachbereich Informatik, Arbeitsbereich STS, TU-Harburg, Studienarbeit 1998.

[13] Ivar Jacobson. *Object-Oriented Software Engineering, A use-case driven approach.* Eddison-Wesley, Harlow, England, 1992.

[14] Ivar Jacobson. *The Object Advantage.* Addison-Wesley, Wokingham, England, 1995.

[15] E. Jessen and R.Valk. *Rechensysteme, Grundlagen und Modellbildung.* Springer, Berlin, 1987.

[16] N. Johannisson. Eine Umgebung für mobile Agenten. Universität Hamburg, Fachbereich Informatik, Arbeitsbereich DBIS, Diplomarbeit 1997.

[17] Gerhard Keller and Thomas Teufel. *SAP R/3 prozeßorientiert anwenden.* Addison-Wesley, Bonn, 1997.

[18] Laura Lemay. *Teach yourself web publishing with html 3.2 in 14 days.* Sams.net Publishing, 201 West 103rd Street, Indianapolis, Indiana, 1996.

[19] Cricket Liu. *Internet-Server, Einrichten und Verwalten.* O'Reilly / International Thomson Verlag, Bonn, 1995.

[20] Sebastian Lutz. Eine polymorph typisierte Schnittstelle der Sprache Tycoon zum System SAP R/3. Universität Hamburg, Fachbereich Informatik, Arbeitsbereich STS, TU-Harburg, Diplomarbeit 1997.

[21] Bernd Mathiske. *Mobilität in persistenten Objektsystemem.* PhD thesis, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich DBIS, Mai 1996.

[22] F. Matthes. *Persistente Objektsysteme: Integrierte Datenbankentwicklung und Programmerstellung.* Springer Verlag, 1993. (In German.).

[23] Florian Matthes. Betriebswirtschaftliche Informatiossysteme am Beispiel SAP R/3, Skript zur Vorlesung SS97. Arbeitsbereich STS, TU-Harburg.

[24] Florian Matthes. Objektorientierte Daten- und Workflowmodellierung, Vorlesung 96/97. Arbeitsbereich STS, TU-Harburg.

[25] Florian Matthes. www.sts.tu-harburg/projects/bc/entry.html. WWW-Server, Arbeitsbereich STS, TU-Harburg 1997/98.

[26] Florian Matthes. Business Conversations, A High-Level System Model for Agent Coordination, 1997. TU Harburg, Arbeitsbereich STS.

[27] Bertrand Meyer. *Objektorientierte Softwareentwicklung.* Addison-Wesley/Carl Hanser, Wokingham, England/Munich, 1990.

[28] Robert Orfali, Dan Harkey, and Jeri Edwards. *The Essential Distributed Objects Survival Guide.* John Wiley & Sons, New York, 1996.

[29] Trygve Reenskaug. *Working with Objects.* Manning Publications Co., 3 Lewis Street, Greenwich, CT 06830, 1996.

[30] Ingo. Richtsmeier. Vergleich objekt- und agentenbasierter verteilter Datenbankprogrammierung am Beispiel der Kopplung autonomer Internet WebSiteProfiler. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Datenbanken und Informationssysteme, 1996.

[31] Yuri Rubinsky. *SGML on the WEB, small steps beyond HTML.* Prentice Hall PTR, Upper Saddle River, New Jersey 07548, 1997.

[32] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[33] Thomas Schäl. *Workflow Management Systems for Process Organisations.* Springer, Berlin, 1997.

[34] J.W. Schmidt. Object-oriented analysis and design. Vorlesung Arbeitsbereich STS, TU-Harburg 1997.

[35] P. C. Lockemann und J.W. Schmidt. *Datenbank-Handbuch.* Springer, Berlin, 1987.

[36] Helmut Wächter. *Fehlertolerantes Workflow-Management, Eine Architektur für die zuverlässige Ausführung verteilter Geschäftsprozesse, Dissertation.* Verlag Dr. Kovac, Arnoldstraße 49, 22763 Hamburg, Germany, 1996.

[37] A. Watt. *Programming Language Concepts and Paradigms.* Prentice Hall c.a.r. hoare series editor, Prentice Hall Europe, 1990.

[38] Holm Wegner. Objektorientierter Entwurf und Implementierung eines Agentensystems für kooperative Internet-Informationssysteme. Universität Hamburg, Fachbereich Informatik,Diplomarbeit 1997.

[39] Mark Weikard. Entwurf und Implementierung einer portablen multiprozessorfähigen virtuellen Maschine für eine persistente, objektorientierte Programmiersprache. Universität Hamburg, Fachbereich Informatik, Arbeitsbereich STS, TU-Harburg, Diplomarbeit 1998.

[40] Terry A. Winograd. A Language/Action Perspective on the Design of Cooperative Work. Technical Report STAN-CS-87-1158, Department of Computer Science, Stanford University, Stanford, CA 94305, April 1987.

[41] Stephan Ziemer. Studienarbeit: SAP R/3: An Overview of its Concepts and Languages. Universität Hamburg, Fachbereich Informatik,Studienarbeit 1997.

Erklärung

Ich erkläre hiermit, die vorliegende Arbeit selbständig durchgeführt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Hamburg, den 20.3.98

Volker Ripp