Ein generisches Online-Verkaufssystem: Anforderungsanalyse, Objektorientierter Entwurf, Realisierung mit Lotus Notes und SAP R/3

Diplomarbeit

Universität Hamburg
Fachbereich Informatik

Andreas Carlsen
Am Kuchenberg 49
21079 Hamburg
Mat.-Nr.: 4340144

Betreuung:

Prof. Dr. Florian Matthes
Technische Universität Hamburg-Harburg
Arbeitsbereich Softwaresysteme

Prof. Dr. Rüdiger Valk
Universität Hamburg
Fachbereich Informatik
Arbeitsbereich Theoretische Grundlagen der Informatik

3. Juni 1999

Zusammenfassung

In dieser Arbeit wird ein generisches Online-Verkaufssystem entworfen und realisiert. Um die Anforderungen an das Online-Verkaufssystem zu klären, erfolgt eine Analyse kommerzieller Online-Verkaufssysteme. Diese klärt die Kernanforderungen an das zu erstellende System. Der objektorientierte Entwurf des Systems unter Berücksichtigung der extrahierten Kernanforderungen geschieht unter Verwendung der *Unified Modelling Language*. Die Implementierung des Systems findet mit *Lotus Notes*, einem Produkt für gruppenunterstützende Arbeit, statt. In diesem Zusammenhang werden die Konzepte von *Lotus Notes* beschrieben. Da *Lotus Notes* kein objektorientiertes System ist, wird ein Vorschlag für die Implementierung der objektorientierten Konzepte der *Unified Modelling Language* vorgestellt. Die Kommunikation und Kooperation des Kunden mit dem System findet unter Verwendung des Modells der *Business Conversations* statt, das langandauernde Interaktionen der Akteure mittels spezifizierter Konversationen beschreibt und die Basis für die generischen und dynamischen Eigenschaften des Systems darstellt. Die Problematik der Integration externer Systeme in einer heterogenen Systemumgebung wird am Beispiel der betriebswirtschaftlichen Standard-Software *SAP R/3* behandelt, das betriebswirtschaftliche Standardfunktionalität für das System zur Verfügung stellt.

Um die Lesbarkeit der Arbeit zu vereinfachen, werden für englische Fachausdrücke, soweit es sinnvoll möglich ist, deutsche Synoyme benutzt. In den Fällen, in denen sich kein sinnvolles deutsches Synonym finden läßt, werden die englischen Begriffe durch *kursive* Schriftart gekennzeichnet. Analog dazu werden Firmen-, Produktnamen und wichtige deutsche Begriffe durch *kursive* Schriftart abgehoben. Weiterhin werden Klassennamen und Programmfragmente sowie Schlüsselwörter von Programmiersprachen durchgängig in `Schreibmaschinenschrift` gesetzt.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Ziele der Arbeit 3

1.2 Aufbau der Arbeit 4

2 Analyse kommerzieller Online-Verkaufssysteme **7**

2.1 Generelle Architektur der Standard-Software 8

2.2 Produktbeschreibungen 10

2.2.1 *Intershop 3.01* 10

2.2.1.1 Programmierung der Geschäftslogik 13

2.2.1.2 Integration externer Systeme 15

2.2.2 *Microsoft Site Server 3.0, Commerce Edition* 16

2.2.3 *Lotus Domino.Merchant Serverpack 2.0* 18

2.2.4 *iCat Electronic Commerce Suite, Professional Edition 3.01* 20

2.2.5 Zusammenfassung der Funktionen der Produkte 22

2.3 Architekturanforderungen und Kriterienauswahl 23

2.4 Anforderungsdefinition und Architektur des IPOS 24

3 Lotus Notes **29**

3.1 Historie 30

3.2 Die Architektur von Notes 31

3.3 Strukturelle Konzepte 32

3.3.1 Masken und Feld-Definitionen 35

3.3.2 Dokumente 38

3.3.3 Ansichten 38

3.3.4 Navigatoren 40
3.3.5 Agenten 40
3.3.6 Script-Bibliotheken 41
3.3.7 Sicherheitskonzept 42
3.3.8 Gegenüberstellung von *Notes* und dem relationalen Modell 44
3.4 Programmierung 44
3.4.1 *Formelsprache* 47
3.4.2 *LotusScript* 48
3.5 Integrationsmöglichkeiten für das Internet 49
3.5.1 *Domino-URL-Syntax* 50
3.5.2 Datenbanken 52
3.5.3 Masken und Feld-Definitionen 52
3.5.4 Agenten 53
3.6 Integration des *SAP R/3*-Systems 54
3.6.1 *LotusScript*-Extension 54
3.6.2 *LotusScript*-Extension für das *SAP R/3*-System 54

4 Systementwicklungsprozeß: Anforderungen und Entscheidungen 57
4.1 Das Modell der *Business Conversations* 58
4.2 Vorgehensmodell 59
4.3 Implementierung der Dokumente des Vorgehensmodells in *Notes* 60
4.4 Systemanforderungen an die *Business Conversations* 63

5 Realisierung der *Notes Business Conversations* 65
5.1 Architektur 65
5.2 *Notes Business Conversations*-Datenbank 67
5.2.1 Konversationsspezifikationen und -instanzen 68
5.2.2 Kommunikationsschnittstelle 72
5.2.3 Das Agentenskelett und die Anwendungslogikschnittstelle 72
5.2.4 Administrationsschnittstelle 78
5.2.4.1 Konversationsspezifikationen 79
5.2.4.2 Konversationsinstanzen 80

5.2.5 Subkonversationen 81
5.3 Generischer Kunde 84

6 Realisierung des Online-Verkaufssystems: Internet Point of Sale 89
6.1 Analyse 89
6.1.1 Akteure 90
6.1.2 Anwendungsfälle 90
6.2 Entwurf 91
6.2.1 Zustandsdiagramme, Konversationsspezifikationen und Maskenentwurf 91
6.2.2 Entwurfs-Klassendiagramm 92
6.3 Implementierung 92
6.3.1 Geschäftslogik und -schnittstelle 93
6.3.2 Administrationslogik und -schnittstelle 94
6.4 Erweiterbarkeit und Integration externer Systeme 95
6.5 Implementierung der Integration des *SAP R*/3-Systems 95

7 Zusammenfassung, Bewertung und Ausblick 97
7.1 Zusammenfassung 97
7.2 Bewertung 98
7.3 Ausblick 100

A Abkürzungsverzeichnis 103

B Diagrammtypen der *Unified Modeling Language* 105

C Modell der *Business Conversations* 107
C.1 Das Modell der *Business Conversations* 107
C.1.1 Leitbild 107
C.1.2 Grundlagen des Modells 108
C.1.3 Modalitäten 110
C.1.4 Abstraktionen und Verfeinerungen 111
C.1.5 Konversationsspezifikationen 113
C.1.6 Modellierung durch Petrinetze 117

D Glossar und Kataloge der Anwendungsfälle **119**

E *LotusScript*-Klassenbibliothek **123**

F Ansichtengruppen der NBC-DB **125**

Literaturverzeichnis **127**

Abbildungsverzeichnis

2.1 Generelle Architektur von Online-Verkaufssystemen 9
2.2 Architektur des Intershop 11
2.3 Programmierung im *Intershop* 14
2.4 Integration externer Systeme im *Intershop* 15
2.5 Architektur des *Microsoft Site Server 3.0, Commerce Edition* 17
2.6 Architektur des *Domino.Merchant* 19
2.7 Architektur der *iCat Electronic Commerce Suite, Professional Edition 3.01* . 21
2.8 Architektur des IPOS 25

3.1 *Notes* Architektur 31
3.2 Struktur von *Notes* 33
3.3 Datentypen der Feld-Definitionen 37
3.4 Ansichten 39
3.5 Sicherheitskonzept für die Anwendungsentwicklung 43
3.6 Programmierung in *Notes* 45
3.7 Erweiterung des Datenbankservers für die Integration des Internets 50
3.8 *Domino-URL-Syntax* und Anweisungen 51
3.9 Ausprägungen von Masken mit spezieller Semantik für das Internet 53
3.10 Ausprägungen von Feld-Definitionen mit spezieller Semantik für das Internet 53
3.11 Erweiterte *LotusScript*-Fähigkeiten durch Einführung der LSX 55
3.12 LSX-Klassen für das System *SAP R/3* 56

4.1 Die vier Interaktionsphasen 58
4.2 Vorgehensmodell zur Entwicklung eines Internet-Informationssystems 59

5.1 Architektur der *Notes Business Conversations* 66
5.2 Aufbau der NBC-Datenbank 68
5.3 Klassendiagramm der Konversationsspezifikationen 70
5.4 Klassendiagramm der Konversationsinstanzen 71
5.5 Klassendiagramm des Agentenskeletts 73
5.6 Interaktionsdiagramm für eine Konversation 74
5.7 Subkonversation als abstrakter Dialog 82
5.8 Interaktionsdiagramm des generischen Kunden 84

6.1 Anwendungsfall „Warenkorb ansehen“ des Kunden 90
6.2 Kunden-Zustandsdiagramm der Datenbank „Verwaltung Warenkörbe“ 91
6.3 Entwurfs-Klassendiagramm des IPOS 92
6.4 Paketdiagramm des IPOS 93
6.5 Paketdiagramm mit Klassen 94

C.1 Unterteilung der Interaktionsphasen 109
C.2 Delegation von Konversationen durch einen *Broker* 111
C.3 Koordinierung von Konversationen durch einen Koordinator 112
C.4 Ablauf einer Konversation 114
C.5 Konversationsspezifikation als nichtdeterministischer endlicher Automat . . . 115
C.6 Petrinetz zur Darstellung der *language/action*-Perspektive 117

E.1 *LotusScript*-Klassenbibliothek 123

Tabellenverzeichnis

2.1 Zusammenstellung der Merkmale der Verkaufssysteme 27

3.1 Zugriffsebenen der Zugriffskontrolliste 44
3.2 Vergleich von *Notes* und dem relationalen Modell 44

4.1 Suffixe 62
4.2 Implementierung der Dokumenttyp-Elemente in *Notes* 63

5.1 Dienstleistungen der Administrationsschnittstelle 79
5.2 Erweiterungen für die Masken der Inhalte 85
5.3 Erweiterungen für die Dialog- und Anfragespezifikationen 86

6.1 Anzahl der Anfrage- und Dialogspezifikationen 93
6.2 Anzahl der Agenten, Ansichten, Masken und Navigatoren 95

7.1 Aufwand für die *Notes Business Conversations* 99
7.2 Gegenüberstellung des Aufwandes 100

D.1 Glossar 119
D.2 Katalog der Anwendungsfälle der Kunden 120
D.3 Katalog der Anwendungsfälle der Verkaufsleiter 120
D.4 Katalog der Anwendungsfälle der Systemadministratoren 121

F.1 Ansichtengruppe „*ConversationSpecs by*“ 125
F.2 Ansichtengruppe „*DialogSpecs by*“ 125
F.3 Ansichtengruppe „*RequestSpecs by*“ 126
F.4 Ansichtengruppe „*DialogSpecs+RequestSpecs by*“ 126
F.5 Ansichtengruppe „*ConversationInst by*“ 126

F.6 Ansichtengruppe „*ErrorDialogContentInst by*“ 126

Kapitel 1

Einleitung

Aufgrund des immer weiter expandierenden Internets und der Tatsache, daß das Internet und seine Dienste immer mehr an Popularität und Marktpotential gewinnen, haben die Unternehmen die Möglichkeiten zur kommerziellen Nutzung erkannt. Diese Möglichkeiten werden unter dem Begriff *Electronic Commerce* zusammengefaßt, der folgendermaßen definiert werden kann [ADGY99; WB98]:

> „*Electronic Commerce* stellt die Gesamtheit aller Prozesse dar, die kommerzielle Aktivitäten in einem Netzwerk unterstützen und die Informationsanalyse einbeziehen."

Dabei werden drei generelle Typen von *Electronic Commerce*-Anwendungen unterschieden [KW97]:

1. Interorganisationelle Typen beziehen sich auf Anwendungen zwischen Unternehmen.
2. Intraorganisationelle Typen betreffen Anwendungen innerhalb eines Unternehmens.
3. „Kunde zu Unternehmen"-Typen sind Anwendungen zwischen Kunde und Unternehmen.

Thema dieser Arbeit ist die Realisierung eines Online-Verkaufssystems, das dem dritten Anwendungstyp „Kunde zu Unternehmen" zugeordnet wird [ADGY99]. Online-Verkaufssysteme bieten dem Unternehmen die Möglichkeit, ohne Verkaufspersonalkosten und Ausstellungsräume Waren anzubieten und zu verkaufen. Auf der anderen Seite wird den Kunden eine Einkaufsmöglichkeit ohne zeitliche Beschränkung geboten. Um die Attraktivität der Systeme für Kunden zu erhöhen, wird versucht, den gleichen Komfort zu erreichen, wie ihn die Kaufhäuser in der realen Welt anbieten. Dafür haben Anbieter von Software-Systemen Standardlösungen für Online-Verkaufssysteme entwickelt, die von Unternehmen für ihre Zwecke genutzt werden können. Damit die Unternehmen die Nutzung optimieren können, muß das Online-Verkaufssystem mit Informationen parametrisiert werden. Ein wichtiges Kriterium für die Unternehmen sollte bei der Auswahl eines geeigneten Systems die Funktionalität sein, die bei verschiedenen Anbietern differieren kann. Dafür muß eine Analyse erstellt und anhand der notwendigen Kriterien das passende System ausgewählt werden.

Andererseits wirft die Realisierung dieser Systeme vielfältige Probleme auf. Um die Probleme zu lösen, sollten die Systeme mit einer geeigneten Notation modelliert werden. Dafür bietet sich die verbreitete *Unified Modelling Language* (UML) an, da diese für sehr viele Problemstellungen angewendet werden kann [Wah98]. Vier der Problemstellungen für Online-Verkaufssysteme, die in dieser Arbeit behandelt werden, sind exemplarisch aufgelistet:

1. Kooperation und Kommunikation mit dem Kunden
2. Implementierungsplattform des Systems
3. Parametrisierbarkeit des Systems
4. Integration externer Systeme

Für die Problemstellung der Kooperation und der dafür notwendigen Kommunikation mit dem Kunden sind folgende Anforderungen vorgegeben:

- Der Kunde benutzt einen HTTP-Client, mit dem er Anfragen an den Dienstleister das Online-Verkaufssystem schickt.
- Die Antworten des Online-Verkaufssystems sind HTML-Seiten, die zum HTTP-Client des Kunden geschickt werden und auf denen die Ergebnisse der Anfragen dargestellt sind.

Die Formalisierung der Kommunikation zwischen dem Kunden und dem Online-Verkaufssystem geschieht in dieser Arbeit anhand des Modells der *Business Conversations*. Das Modell eignet sich für die Beschreibung langandauernder Interaktionen zwischen Akteuren. Angelehnt ist das Modell an die Theorie der Sprechakte [Mat97a; Mat97b]. Konversationsspezifikationen ermöglichen die Formalisierung der Kommunikation. Problematisch für die Verwendung des Modells ist die Seite der Kunden. Den Kunden sind in der Regel die verwendeten Konversationsspezifikationen unbekannt. Um dieses Problem zu lösen, kann ein generischer Kunde verwendet werden [Weg98; Rip98], der dynamisch vom Dienstleister aus den Konversationsspezifikationen generiert wird. Der menschliche Kunde verhält sich unter Verwendung des generischen Kunden gemäß der Konversationsspezifikationen und kann vom Modell abstrahieren.

Die zweite Problemstellung betrifft die Implementierungsplattform, die für ein Online-Verkaufssystem gewählt wird. Diese muß für die Entwicklung von Internet-Informationssystemen geeignet sein. Das Produkt, das dafür benutzt wird, ist *Lotus Notes*. Ab der Version 4.0 bietet es gute Möglichkeiten für die Entwicklung von Internet-Informationssystemen. Die weitere Entwicklung bis zur Version 5.0 baut diese noch weiter aus. Außerdem unterstützt *Lotus Notes* die Integration externer Systeme durch offene Schnittstellen.

Die dritte Problemstellung betrifft die Parametrisierbarkeit des Systems. Diese ist notwendig für die Anpassung des Systems an die Gegebenheiten des Unternehmens. Eine Möglichkeit für ein parametrisierbares System auf der Basis des oben genannten Modells der *Business Conversations* bietet ein Agent, wie er in [Weg98] beschrieben wird. Ein Agent bietet eine Anwendungslogik-Schnittstelle für die Parametrisierung an.

Die letzte Problemstellung ist die Integration externer Systeme. Da die meisten Unternehmen bereits vor ihrem Entschluß, ein Online-Verkaufssystem anzubieten, Waren verkauft haben, sind Informationen darüber meistens in einem vorhandenen System gespeichert. Damit die entstehenden heterogenen Systemlandschaften eines Unternehmens integriert werden können, müssen Online-Verkaufssysteme Schnittstellen für externe Systeme anbieten. Ein Beispiel für ein externes System ist das weit verbreitete *SAP R/3*-System, das eine betriebswirtschaftliche Standard-Software ist. Eine Integration des *SAP R/3*-Systems kann z.B. über das *Business Framework* geschehen [JR99], das die betriebswirtschaftliche Funktionalität des Systems von anderen Systemen aus nutzbar macht.

1.1 Ziele der Arbeit

Das Hauptziel der Arbeit ist die prototypische Erstellung eines generischen Online-Verkaufssystems für das Internet. Das zu realisierende Online-Verkaufssystem wird mit **I***nternet* **P***oint* **o***f* **S***ale* (IPOS) benannt. Vorgegebene Anforderungen an die Arbeit sind:

1. Eine Anforderungsanalyse soll durchgeführt werden.
2. Ein objektorientierter Entwurf soll erfolgen.
3. Die Implementierungsplattform ist das Produkt *Lotus Notes*.
4. Das Modell der *Business Conversations* soll für die Kommunikation mit dem Kunden verwendet werden.
5. Eine Integration des *SAP R/3*-Systems soll vorgenommen werden.
6. Das System soll einen hohen Grad an Generik aufweisen.

Die Teilziele der Arbeit leiten sich aus dem Hauptziel und den Anforderungen ab:

Die Anforderungsanalyse soll existierende Online-Verkaufssysteme untersuchen. Da Online-Verkaufssysteme eine Vielzahl an Funktionen bieten, deren Realisierung den Rahmen der Arbeit sprengen würde, muß die Kernfunktionalität identifiziert werden. Die Ergebnisse der Analyse legen die Architektur und Funktionen des IPOS fest.

Der objektorientierte Entwurf soll mit Hilfe der *Unified Modelling Language* (UML) erfolgen. Die Verwendung von UML ist motiviert durch die universellen Einsatzmöglichkeiten dieser Notation [Wah98; Sof97a; Sof97b; FS97].

Als Implementierungsplattform soll das Produkt *Lotus Notes*, im weiteren als *Notes* bezeichnet, verwendet werden. *Notes* ist ein Produkt für gruppenunterstützende Tätigkeiten und eignet sich für die Implementierung aufgrund guter Möglichkeiten für Internet-Informationssysteme und der Integration externer Systeme. Als Basis für die Implementierung sollen die für die Arbeit relevanten Konzepte von *Notes* untersucht und vorgestellt werden.

Das Modell der *Business Conversations* soll für die Kommunikation objektorientiert entworfen, implementiert und bewertet werden. Die Implementierungsplattform ist durch *Notes* vorgegeben. Das resultierende System wird im folgenden *Notes Business Conversations* genannt.

Motiviert wird die Verwendung des Modells aufgrund der guten Eignung für Interaktionen zwischen einem Paar aus Kunde und Dienstleister. Beschrieben wird das Modell der *Business Conversations* in [Mat97a; Mat97b]. Das Modell wurde bereits am Arbeitsbereich DBIS der Universität Hamburg in der funktionalen Programmierumgebung *Tycoon*[1] und dem objektorientierten *Tycoon-2*-System implementiert [Joh97; Ric97; Weg98].

Um die Möglichkeiten der Integration externer Systeme aufzuzeigen, soll eine Integration der betriebswirtschaftlichen Standard-Software *SAP R/3* erfolgen. Die Integration soll durch die NBC erfolgen und durch Subkonversationen realisiert werden, so daß die Anbindung anderer externer Systeme durch Austauschen der Subkonversationen unterstützt wird. Weiterhin soll für die Integration das *Business Framework* des *SAP R/3*-Systems verwendet werden [JR99].

Einen hohen Grad an Generik soll die Implementierung eines generischen Kunden für die NBC ermöglichen, der die Grundlage für die Kommunikation mit dem Kunden ist. Der generische Kunde soll dynamisch aus den Konversationsspezifikationen der NBC generiert werden, so daß dem Kunden korrekte Konversationsspezifikationen zur Verfügung gestellt werden. Weiterhin soll eine dynamische Modifikation der Ablauflogik des Systems möglich sein.

Weitere Anforderungen an diese Arbeit sind:

- Der Entwurf und die Implementierung der NBC soll weitgehend an den der bereits in *Tycoon* und *Tycoon-2* implementierten Systeme angelehnt werden.
- Der Einsatz der UML-Notation für die Entwicklung von *Notes*-Systemen wird bewertet.

1.2 Aufbau der Arbeit

Die Arbeit gliedert sich folgendermaßen:

Im nachfolgenden Kapitel 2 wird eine Analyse existierender Online-Verkaufssysteme vorgenommen. Der generelle Aufbau der Systeme wird aufgezeigt und es werden konkrete Produkte untersucht. Detalliert wird auf das Produkt *Intershop 3.01* eingegangen. Abschließend werden die extrahierten Kernfunktionen und die resultierende Architektur des IPOS vorgestellt, die die Grundlage für das Kapitel 6 sind.

Die Konzepte des Produktes *Notes* werden in Kapitel 3 vorgestellt. Dabei wird speziell auf die Möglichkeiten der Entwicklung von Internet-Informationssystemen und der Integration des *SAP R/3*-Systems eingegangen. Das Kapitel stellt die Basis für die Implementierung des IPOS und der NBC dar.

Kapitel 4 behandelt Anforderungen und Entscheidungen des Systementwicklungsprozesses. Beschrieben werden das verwendete Vorgehensmodell, Implementierungsmöglichkeiten von Dokumenttypen des Vorgehensmodells in *Notes* und Systemanforderungen an das Modell der *Business Conversations*.

Das Kapitel 5 stellt den Entwurf und die Implementierung der *Notes Business Conversations* vor. Besonders wird auf die Realisierung der *Notes Business Conversation*-Datenbank und den generischen Kunden eingegangen.

[1] Literatur zu Tycoon sind [Mat93; MMM93; MMS94].

Das realisierte prototypische Online-Verkaufssystem IPOS wird in Kapitel 6 betrachtet. Die Beschreibung der Analyse-, Entwurfs- und Implementierungsphase wird auf der Grundlage der in Kapitel 2 vorgestellten Architektur und Anforderungsdefinition vorgenommen.

Abschließend faßt Kapitel 7 die Ergebnisse der Arbeit zusammen und bewertet sie. Außerdem ist ein Ausblick auf weitergehende Möglichkeiten und offene Fragen enthalten.

In dieser Arbeit werden alle Abbildungen mit Ausnahme der Architekturabbildungen einheitlich mit der *Unified Modeling Language* (UML) beschrieben. UML stellt eine Notation für die objektorientierte Analyse, den Entwurf und die Implementierung dar. Eine kurze Beschreibung von UML findet sich im Anhang B und kann [Sof97b; Sof97a; Hru98; FS97] entnommen werden.

Kapitel 2

Analyse kommerzieller Online-Verkaufssysteme

Die Analyse von Online-Verkaufssystemen im Internet ist Thema dieses Kapitels. Diese Analyse ist die Basis für die Auswahl der Kernanforderungen an das zu realisierende System. Die Kernanforderungen werden aus den Ergebnissen der Analyse extrahiert.

Damit die Analyse vorgenommen werden kann, muß eine Auswahl konkreter Online-Verkaufssysteme stattfinden. Generell bieten sich für den Betrieb eines Online-Verkaufssystems drei Alternativen [ADGY99; BK98] an:

- Individualentwicklung
- Einsatz von Standard-Software
- Betrieb einer Verkaufsanwendung auf Mietbasis (*Mall*-Konzept)

Die Analyse betrachtet den Einsatz von Standard-Software, da diese den Anspruch erhebt, das erforderliche Spektrum von Funktionen für den Betrieb eines Online-Verkaufssystems vollständig abzudecken. Individualentwicklung hingegen befriedigt die Anforderungen eines speziellen Kunden und eine Verkaufsanwendung auf Mietbasis, in der Literatur auch *Mall*-Konzept genannt, basiert entweder auf einer Standard-Software oder Individualentwicklung.

Eine weitere Klassifizierung von Online-Verkaufssystemen läßt sich nach der Leistungsfähigkeit der Systeme in drei Gruppen vornehmen [Kra98]:

- große Systeme für mehrere tausend gleichzeitige Transaktionen
- mittlere Systeme für mehrere tausend Transaktionen an einem Tag
- einfache Systeme mit geringer Flexibilität und wenigen Transaktionen am Tag

In dieser Analyse werden Systeme mittlerer Größenordnung betrachtet, da in diesem Bereich die Vielfalt an Systemen besonders groß ist. Außerdem unterscheiden sich die Kernfunktionen nicht gravierend von denen der großen Systeme. Aufgrund der großen Anzahl von Systemen

am Markt fiel die Auswahl auf vier der bekanntesten und verbreitesten Produkte in diesem Bereich [BK98; Kra98].

Die Gliederung des Kapitels unterteilt sich in drei Bereiche. Im ersten Bereich in Abschnitt 2.1 wird die generelle Architektur der Online-Verkaufssysteme vorgestellt. Die generelle Architektur mit ihren Komponenten dient als Leitfaden für die Produktbeschreibungen.

Der zweite Bereich in Abschnitt 2.2 beschäftigt sich mit der Beschreibung der Konzepte und Funktionen der konkreten Online-Verkaufssysteme. Für die Produkte werden die Architektur und die sich daraus ergebenden Komponenten anhand der Betrachtungen der generellen Architektur in Abschnitt 2.1 beschrieben. In Abschnitt 2.2.1 wird detalliert auf *Intershop 3.01* eingegangen, welches einer der Marktführer ist. Abschließend faßt Abschnitt 2.2.5 die Funktionen der vorgestellten Produkte tabellarisch zusammen.

Der dritte Bereich in Abschnitt 2.3 enthält die Kriterienauswahl, die die Basis für das zu realisierende Online-Verkaufssystem bildet. Die Kriterienauswahl schränkt das Spektrum der analysierten Funktionalität auf die Kernfunktionen ein, um eine realistische Grundlage[1] für die Realisierung des Online-Verkaufssystems zu bilden. Die weiteren Funktionen können später hinzugefügt werden. In Abschnitt 2.4 werden die Anforderungsdefinition und die Architektur des zu realisierenden Systems IPOS definiert, die sich aus den extrahierten Kernfunktionen der Analyse ergeben.

2.1 Generelle Architektur der Standard-Software

Die generelle Architektur der Produkte für Online-Verkaufssysteme ist nahezu identisch. Das ist bedingt durch die vorgegebenen technischen Randbedingungen [Was97b]. Die wichtigste Randbedingung ist das Internet als Medium für die Kommunikation mit dem Kunden. Die Architektur beruht auf einer dreistufigen Client/Server-Architektur. Die Präsentations-Schicht wird durch den HTTP-Client und -Server repräsentiert. Die Anwendungslogik-Schicht enthält die Geschäftslogik- und Administrations-Funktionalität. Die dritte Schicht ist die der Datenbanken. In den konkreten Systemen unterscheiden sich die Komponenten der Schichten durch die verwendeten Produkte und Techniken[2] der verschiedenen Hersteller.

Die heterogenen Komponenten und Schnittstellen der Architektur sind in Abbildung 2.1 dargestellt. Online-Verkaufssysteme bestehen aus folgenden Komponenten und Schnittstellen:

- HTTP-Server / *Gateway*-Schnittstelle
- Datenbank-Server / Datenbank-Schnittstelle
- Administrations-*Framework* / Administrations-Schnittstelle
- Geschäftslogik-*Framework* / Geschäftslogik-Schnittstelle
- Externe-Systeme / Externe-Schnittstelle

[1] Realistisch bezüglich des Zeitrahmens der Arbeit.
[2] Die eingesetzten Techniken sind meistens abhängig von den verwendeten Produkten.

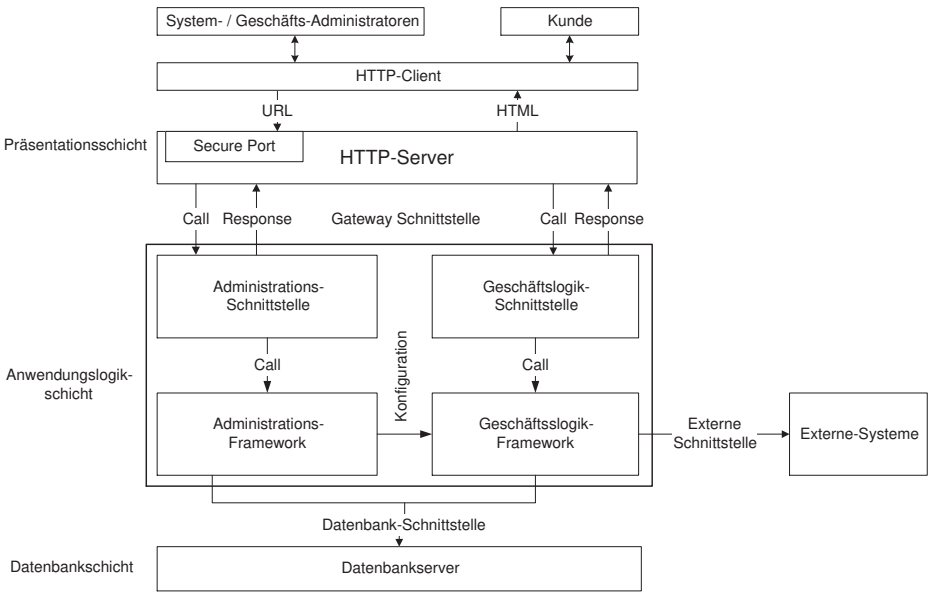

Abbildung 2.1: Generelle Architektur von Online-Verkaufssystemen

Die ersten vier Komponenten sind Standardkomponenten, welche die notwendige Funktionalität für den Betrieb bereitstellen und mit für diesen Zweck spezialisierten und bewährten Fremdprodukten besetzt werden können. Der HTTP-Server realisiert die Kommunikationsschnittstelle für die HTTP-Clients, über die die Benutzer mit dem System kommunizieren. Die *Gateway*-Schnittstelle reicht die Anfragen und Ergebnisse an die Anwendungslogik-Schicht weiter. Der Datenbank-Server verwaltet alle Informationen des Systems und wird über die Datenbank-Schnittstelle angebunden.

Interessantere Komponenten sind die *Frameworks* und Schnittstellen für die Administration und Geschäftslogik, da sie das Online-Verkaufssystem darstellen und die Systeme sich hauptsächlich in diesen Komponenten unterscheiden. Das Administrations-*Framework* stellt Funktionen bereit, um das System mittels der Administrations-Schnittstelle anzupassen und zu verwalten. Meistens werden die Bereiche der System- und Geschäfts-Administration unterschieden. Die System-Administration ermöglicht die Administration des generellen Aufbaus des Systems, wogegen die Geschäfts-Administration die Inhalte des Systems administriert. Als Schnittstelle für die Administration dienen meistens HTML-Schablonen oder teilweise externe Anwendungen. Das Geschäftslogik-*Framework* enthält die angebotene Funktionalität des Systems, auf die der Kunde über die Geschäftslogik-Schnittstelle zugreift. Die Schnittstelle ist durch das Internet mit HTML-Schablonen vorgegeben.

Die letzte Komponente sind externe Systeme. Externe Systeme repräsentieren z.B. *legacy*-Systeme, die bei bestehender heterogener System-Umgebung integriert werden müssen und ergänzende Systeme wie Systeme für digitales Geld [LL96]. Die Anbindung externer Systeme geschieht über externe Schnittstellen, die standardisierte oder proprietäre Schnittstellen sein können.

2.2 Produktbeschreibungen

In diesem Abschnitt werden am Markt vorhandene Produkte betrachtet. Wie bereits eingangs gesagt, wurden vier der bekanntesten und verbreitesten Lösungen ausgesucht [BK98; Kra98]:

- *Intershop 3.01*
- *Microsoft Site Server 3.0, Commerce Edition*
- *Lotus Domino.Merchant Server 2.0*
- *iCat Electronic Commerce Suite, Professional Edition 3.01*

Im folgenden werden die Produkte in der oben genannten Reihenfolge vorgestellt und beschrieben. Ausführlich wird auf *Intershop 3.01* eingegangen, da dieses momentan eines der modernsten Produkte ist. Die Beschreibungen betrachten die Architektur und eingesetzten Techniken, die anhand des generellen Aufbaus in Abschnitt 2.1 erläutert werden. Zusätzlich werden spezielle Funktionen der einzelnen Produkte erläutert. In Abschnitt 2.2.5 werden abschließend weitere Funktionen und Merkmale der Produkte tabellarisch zusammengefaßt und gegenübergestellt. Weitergehende Beschreibungen der einzelnen Funktionen der Produkte können den Literaturquellen, die in den Produktbeschreibungen angegeben sind, entnommen werden.

2.2.1 *Intershop 3.01*

Das Produkt *Intershop 3.01* (*Intershop*) ist eine der modernsten Lösungen für Online-Verkaufssysteme und nach eigenen Firmenangaben von *Intershop* der Marktführer in diesem Bereich. Dem Markt wurde die aktuellste Version des *Intershop* erstmals im Frühjahr 1998 vorgestellt. Die Quellen für die Produktbeschreibung waren die Dokumentation, eine *Intershop*-Installation, Demo-Systeme im Internet, Artikel und Bücher [Com98f; Com98a; Com98g; Com98j; Com98i; Com98h; Com98c; Com98e; Com98d; Com98b; BK98; Kra98; Was97b].

Der *Intershop* besteht aus einer skalierbaren dreistufigen Client/Server-Architektur, die in Abbildung 2.2 dargestellt ist. Die Skalierbarkeit des Systems drückt sich in den Optionen der Verteilung der Stufen aus. Prinzipiell können alle Stufen beliebig zusammengefaßt oder verteilt werden. Dabei können nur ein HTTP- und Datenbank-Server verwendet werden. Die Stufe der Applikations-Server, die aus dem Administrations- und Anwendungslogik-Bereich besteht, kann auf mehrere Rechner verteilt werden[3]. Nachfolgend werden die drei Stufen beschrieben:

HTTP-Server: Als HTTP-Server kann ein beliebiges Produkt eingesetzt werden. Damit der HTTP-Server mit dem *Intershop* zusammenarbeiten kann, werden als *Gateway*-Schnittstelle drei Adapter und der *Request Router* installiert:

[3] Zusätzlich können die Rechner unterschiedliche Betriebssysteme benutzen.

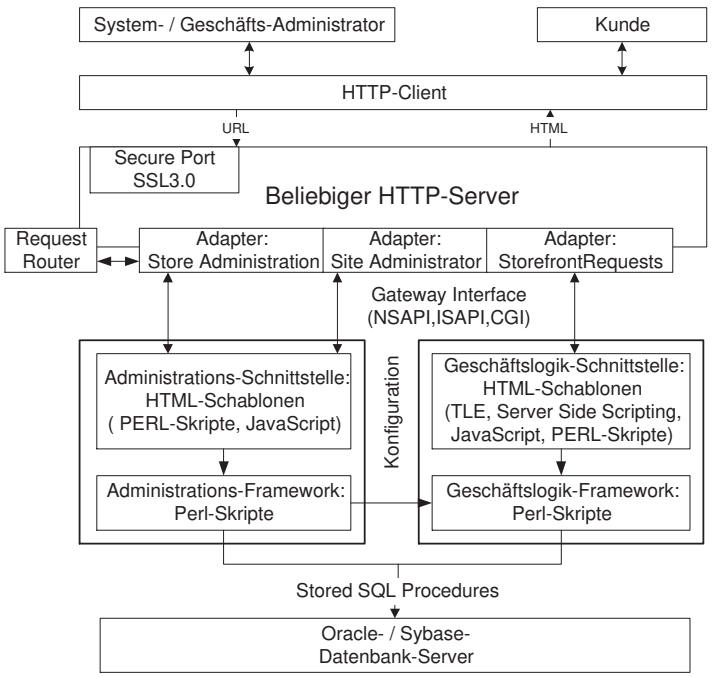

Abbildung 2.2: Architektur des Intershop

- Adapter: Die Adapter erhalten alle Anfragen, die an den HTTP-Server geschickt werden. Diese werden abhängig vom jeweiligen Kontext der Anfrage vom Adapter für die *Site* Administration, *Store* Administration oder *Storefront requests* bearbeitet. Der Adapter *Site* Administration erhält Anfragen, die von der Systemadministration stammen und der Adapter *Store* Administration Anfragen bezüglich der Geschäftsadministration. Die Anfragen der Kunden erhält der Adapter *Storefront requests*. Die Adapter leiten die Anfragen an die Applikations-Server weiter.

- *Request Router*: Der *Request Router* soll für die ausgewogene Belastung der Applikations-Server sorgen. Dafür fragen die Adapter für jede Anfrage an, zu welchem zugehörigen Applikations-Server die Anfrage geschickt werden soll. Die Entscheidung trifft der *Request Router* anhand von Informationen über die Auslastung und gibt den Adaptern die Adresse des am wenigsten belasteten Applikations-Servers zurück.

Die *Gateway*-Schnittstelle zum HTTP-Server, die die Applikations-Server mit dem HTTP-Server verbindet, kann mit drei Techniken realisiert werden. Für die HTTP-Server *Microsoft Internet Information Server* und die *Netscape Web Server* werden die entsprechenden Server-APIs (ISAPI und NSAPI) benutzt. Jeder andere HTTP-Server wird mit Hilfe der standardisierten CGI-Schnittstelle angebunden.

Applikations-Server: Die Applikations-Server kapseln die *Intershop*-Anwendungslogik. Zwei Typen von Applikations-Servern werden unterschieden, welches Vorteile für die Geschwindigkeit und Skalierbarkeit des Systems bietet.

- Die **Administrations**-Server erhalten alle Anfragen, die von der System- und Geschäfts-Administration stammen. Die Systemadministration umfaßt alle Tätigkeiten, die den Aufbau und die Wartung des Systems betreffen. Die Tätigkeiten fallen in die Bereiche der allgemeinen Einstellungen, Kundenprofile, Bezahlungs-Methoden, Zugriffskontrolle, Datenbank-Wartung, Datenimport und Testsystem. Dagegen umfaßt die Geschäftsadministration alle Tätigkeiten, die die Inhalte und täglichen Arbeiten im System betreffen. Für die Ausführung dieser Tätigkeiten existieren der Katalog-, Produkt-, Inventar-, Einkauf- und Kundenmanager. Zusätzlich können Statistiken, Vorgaben und Kundenprofile erstellt und modifiziert werden.
- Die **Geschäftslogik**-Server bearbeiten alle Kundenanfragen, die an das System gestellt werden. Die Anfragen werden dann gemäß der Ablauflogik und den zugehörigen Einstellungen der Administration abgearbeitet.

Gemeinsam haben die Typen der Applikations-Server die Schnittstellen, die durch HTML-Schablonen (*Templates*) realisiert sind. Die *Templates* werden dynamisch aus der Datenbank mit den geeigneten Inhalten gefüllt und dienen dem Benutzer als Kommunikations-Schnittstelle im HTTP-Client.

Bezüglich der Anwendungslogik, die mit der *PERL*-Skriptsprache implementiert ist, existieren Unterschiede. Das Administrations-*Framework* und die zugehörigen *Templates* sind fest vorgegeben und nicht mit integrierten Mitteln modifizierbar. Eine Modifizierung ist nur über sogenannte *Hooks* möglich, die die Registrierung von externen *PERL*-Skripten für Ereignisse im *Intershop* ermöglichen.

Im Gegensatz dazu können die *Templates* und das *Framework* für die Geschäftslogik, wofür Vorgaben existieren, mittels der System-Administration modifiziert werden und den Geschäftsgegebenheiten angepaßt werden. Die Möglichkeiten zur Anpassung der *Templates* und der Geschäftslogik sind nachfolgend stichwortartig aufgelistet und werden in den Abschnitten 2.2.1.1 und 2.2.1.2 genauer erläutert:

- *Template*-Editor
- *Template Language Extension* (TLE)
- *Server Side Scripting* (SSS)
- *JavaScript*
- *INTERSHOP Developer Cartridge* (IDC)
- Extensionen

Datenbank-Server: Der Datenbank-Server vom *Intershop* kann entweder ein *Sybase Adaptive Database 11*- oder *Oracle 8*-Server sein, die beide *multi-threading* unterstützen. Auf beide wird als Datenbankschnittstelle mittels der ODBC-Schnittstelle mit *Stored SQL Procedures* zugegriffen.

Wichtige und nützliche Funktionen vom *Intershop* sind unter anderem der *StoreDesignWizard*, das *Staging*-Konzept und Hybrid-HTML (HHTML).

- *StoreDesignWizard*: Der *StoreDesignWizard* ist ein grafisches Werkzeug, um den erstmaligen Aufbau eines Systems zu unterstützen. Der Benutzer kann ohne HTML-Kenntnisse anhand der angebotenen Stilrichtungen für die *Templates* das Aussehen des Ladens bestimmen.

- *Staging*-Konzept: Das *Staging*-Konzept bedeutet die Möglichkeit zur Verwendung eines Test-Systems parallel zum Produktiv-System, in dem Änderungen vorgenommen und getestet werden. Diese werden zu einem bestimmten Zeitpunkt selektiv in das Produktiv-System übernommen.

- Hybrid-HTML: Die Bezeichnung Hybrid-HTML bezeichnet die Möglichkeit, für bestimmte *Templates*, die keine dynamischen Informationen enthalten, generierte statische HTML-Seiten auf dem HTTP-Server zu speichern. Dieses führt zu einer Steigerung der Geschwindigkeit, da diese *Templates* bei Anfragen nicht dynamisch generiert werden. Das Aktualisierungsproblem bei Änderungen der Informationen der statischen HTML-Seite, kann über automatische oder manuelle Aktualisierungen gelöst werden.

In den folgenden Abschnitten werden die Möglichkeiten des *Intershop* für die Programmierung und Integration externer Systeme erläutert.

2.2.1.1 Programmierung der Geschäftslogik

Eine wichtige Anforderung ist die Möglichkeit zur Anpassung des *Intershops*, damit es optimal auf die zu repräsentierende Firma zugeschnitten ist. Die Anpassung kann einerseits über die Einstellungen der Administration geschehen, wobei dort nur kosmetische Einstellungen modifiziert werden können. Die Erweiterung und Modifikation der Prozesse des Systems muß mittels Modifikation der Programmierung der Geschäftslogik geschehen. Die Programmierung der Geschäftslogik erfolgt in den *Templates* der Geschäftslogik und wird auf die Erstellung und Modifizierung der *Templates* begrenzt. Da *Templates* HTML-Schablonen sind, können sie entweder im integrierten *Template*-Editor, der die HTML-Schablone textuell anzeigt oder mit einem beliebigen anderen Werkzeug bearbeitet werden. In Abbildung 2.3 sind die Bestandteile der *Templates* dargestellt.

***Template Language Extension* (TLE):** Die *Template Language Extension* ist eine proprietäre Entwicklung von *Intershop*, um die Nachteile von HTML auszugleichen[4]. Sie enthält zwei Typen von Ausdrücken: die TLE-Variablen und TLE-Statements.

TLE-Variablen sind Schlüsselwörter, die in ein Template eingefügt werden können. Im Prinzip stellt eine TLE-Variable einen Funktionsaufruf dar und wird beim Aufruf automatisch und kontextspezifisch durch einen Wert ersetzt. Zwei Typen von TLE-Variablen werden unterschieden:

- vorgegebene TLE-Variablen
- benutzerdefinierte TLE-Variablen

[4]HTML kennt keine Variablen und dynamische Datenelemente.

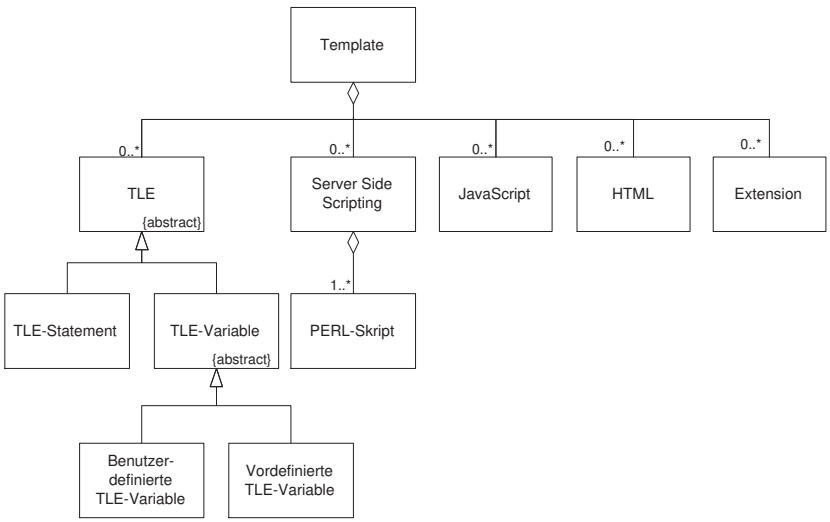

Abbildung 2.3: Programmierung im *Intershop*

Die vorgegebenen TLE-Variablen, die für alle relevanten Geschäftsbereiche existieren, repäsentieren meistens Datenbankabfragen, die den Wert der Variable ermitteln. Die benutzerdefinierten TLE-Variablen können mit einem textuellen Wert (z.B. oft gebrauchter HTML-Text) gefüllt werden.

TLE-Statements enthalten die zwei Kontrollanweisungen `IF` und `LOOP`. Konditionale Anweisungen ermöglicht die `IF`-Anweisung, wogegen die `LOOP`-Anweisung für die Ausgabe von Listen vorgesehen ist.

***Server Side Scripting* (SSS):** Das *Server Side Scripting* ermöglicht die Verwendung der *PERL*-Skriptsprache innerhalb der *Templates*. Dieses wird durch einen integrierten Interpreter für die *PERL*-Skriptsprache erreicht. Die Programmierung der Skripte wird durch fünf *Intershop*-Funktionsgruppen, die in den benutzerdefinierten *PERL*-Skripten verfügbar sind, unterstützt:

- TLE-Behandlungsfunktionen
- Datenbank-Zugriffsfunktionen
- Layout- und Formatierungsfunktionen
- Eingabe- und Ausgabefunktionen
- Warenkorbfunktionen
- Bestellungsfunktionen

Die TLE-Behandlungsfunktionen ermöglichen den lesenden und schreibenden Zugriff auf TLE-Variablen. Auf die Datenbank des Systems kann über die Datenbank-Zugriffsfunktionen zugegriffen werden, die SQL-Abfragen ermöglichen. Die Layout- und Formatierungsfunktionen ermöglichen die Formatierung von Zahlenwerten und HTML-Konvertierungs- und HTML-Decodierungsfunktionen für Zeichenketten. Die Gruppe der Eingabe- und Ausgabefunktionen erlaubt das Auslesen von Informationen und die Ausgabe von Text im HTML-Formular. Abschließend erlauben die Funktionsgruppen für

den Warenkorb und die Bestellung, die Manipulation der Inhalte des Warenkorbs und das Erzeugen und Löschen von Bestellungen.

JavaScript: In einem *Template* kann *JavaScript* verwendet werden. Für die Integration mit der *Intershop*-Anwendungslogik werden die TLE-Variablen als Parameter für die *JavaScript*-Funktionen übergeben.

HTML: In einem *Template* können beliebige HTML-Befehle verwendet werden.

Extensionen: Extensionen sind *PERL*-Skripte, die die *Intershop*-Funktionalität erweitern. Der Einbindung der Extensionen geschieht über URL-Referenzen, die auf die Extensionen verweisen. In der URL-Referenz können dem *PERL*-Skript der Extension Parameter übergeben werden. Das *PERL*-Skript der Extension kann die Funktionalität der *Intershop*-Applikation mittels der Einbindung von *PERL*-Moduln nutzen. Als Rückgabewert schickt das *PERL*-Skript ein *Template* an den HTTP-Client.

2.2.1.2 Integration externer Systeme

Um den heterogenen Systemlandschaften der Kunden gerecht zu werden, bietet der *Intershop* zwei Möglichkeiten, externe Systeme zu integrieren, die in Abbildung 2.4 dargestellt sind. Der wesentliche Unterschied zwischen den Möglichkeiten besteht darin, daß das *Server Side Scripting* (SSS) in die Ablauflogik des *Intershops 3* integriert ist. Dagegen ermöglicht das *Intershop Development Kit* (IDK) nur den Zugriff auf die *Intershop*-Datenbank und kann keinen Einfluß auf die Ablauflogik nehmen. Die erste Möglichkeit das SSS wurde bereits im

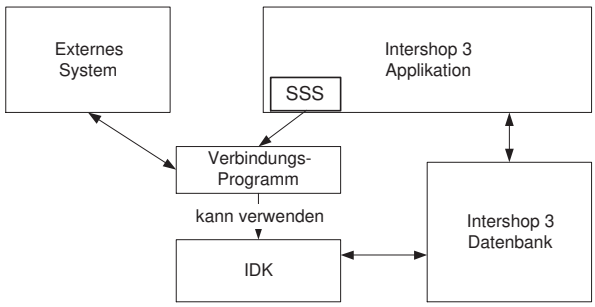

Abbildung 2.4: Integration externer Systeme im *Intershop*

vorangegangenen Abschnitt vorgestellt. Die Integration über SSS kann mit *PERL*-Skripten erfolgen, da über Erweiterungen der *PERL*-Skriptsprache auf externe Systeme zugegriffen werden kann. Die Integration würde dann über ein Verbindungsprogramm realisiert, das den Zugriff auf das externe System ermöglicht.

Die zweite Möglichkeit besteht in der Verwendung des IDK's, das den *Intershop* in ein offenes System verwandelt. Das IDK besteht aus einer *Java*-Klassenbibliothek, die in thematische `Packages` unterteilt ist. Die `Packages` sind eine Sammlung von Werkzeugen zur Vereinfachung des Datenaustauschs zwischen externen Systemen und dem *Intershop*. Mit Hilfe der `Packages` des IDK kann eine Verbindung zu der Datenbank des *Intershops* aufgebaut werden und auf die Daten schreibend und lesend zugegriffen werden. Die Integration eines externen Systems

kann unter Verwendung eines Verbindungs-Programms, das beiden Systemen Zugriff aufeinander gewährt, realisiert werden. Die mit dem IDK erstellten Integrationen werden *Intershop Developer Cartridges* (IDC) genannt. Als Beispiel gibt es ein IDC für das *SAP R/3*-System.

Als sinnvolle Ergänzung zu den beiden Möglichkeiten ist eine kombinierte Nutzung möglich. Das Verbindungsprogramm, das die *Java*-Klassenbibliothek benutzt, kann durch SSS verwendet werden, so daß die beiden Integrationsformen kombiniert werden. Eine Möglichkeit dafür ist die Verwendung von *CORBA*[5], mit der ein Zugriff von einem *PERL*-Client auf einen *Java*-Server möglich ist.

2.2.2 *Microsoft Site Server 3.0*, *Commerce Edition*

Der *Site Server 3.0, Commerce Edition*, im folgenden *Commerce* Server genannt, ist ein Produkt der Firma *Microsoft*. Das Produkt ist Bestandteil des *Microsoft Commercial Internet System 2.0*, das vier Hauptdienste mit folgenden Themen unterstützt:

- Mitgliedschaft (Benutzerauthentifizierung, etc.)
- Gemeinschaft (e-Mail, *News*, *Chat*, etc.)
- Gastgeber (Verkaufssystem, etc.)
- Zugang (Entfernter Zugriff, etc.)

Der *Commerce Server* gehört zu den Bestandteilen des Gastgeber-Hauptdienstes. Eigenständig kann der *Commerce Server* nicht genutzt werden, da er eine Erweiterung des *Microsoft Site Server 3.0* ist. Die Nutzung des *Commerce Servers* ist nur unter dem *Microsoft* Betriebssystem *Windows NT 4.0* möglich und die verwendeten Techniken sind nur mit *Microsoft* Produkten einsetzbar. Die folgende Beschreibung der Architektur und Techniken des *Commerce Servers* basieren auf einer Installation und folgender Literatur [MC98d; MC98e; MC98c; MC98f; MC98b; MC98g; MC98a; MC97].

Die Architektur eines Systems, das mit dem *Commerce Server* realisiert wird, beruht wie in Abschnitt 2.1 auf einer dreistufigen Client/Server-Architektur, die in Abbildung 2.5 dargestellt ist. Die Skalierbarkeit des Systems ist nicht so umfassend wie die des *Intershops*, da der HTTP-Server und *Commerce Server* auf dem gleichen Rechner installiert sein müssen. Ansonsten können beliebig viele dieser Rechner eingesetzt werden. Ein *Commerce Server* kann mit beliebig vielen Datenbankservern kombiniert werden. Die Verwaltung der Rechner übernimmt der *DNS Round-Robin Manager*, der die Anfragen nach Belastung verteilt. Die drei Stufen der Architektur werden nachfolgend mit ihren Techniken und Funktionen beschrieben:

HTTP-Client/HTTP-Server und MMC/Pipeline-Editor: Als HTTP-Server kann nur der *Microsoft Internet Information Server 4.0* (IIS) benutzt werden, der kostenlos von *Microsoft* erhältlich ist. Diese Einschränkung begründet sich in den *Active Server Pages* (ASP), die für die Implementierung der Systeme verwendet werden. Der IIS ist der einzige HTTP-Server, der ASP-Seiten unterstützt. Als *Gateway*-Schnittstellen zum *Commerce Server* dienen somit die ISAPI und die ASP's.

[5] CORBA = *Common Object Request Broker Architecture*

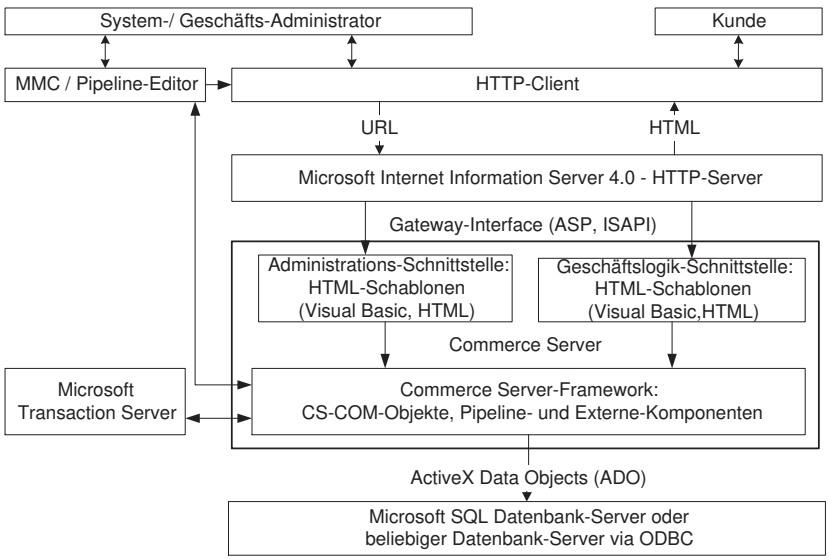

Abbildung 2.5: Architektur des *Microsoft Site Server 3.0, Commerce Edition*

Kunden und die System-/Geschäfts-Administratoren können mit einem HTTP-Client die jeweiligen Funktionen des Systems verwenden. Als weitere Clients für die Administratoren können die *Microsoft Management* Konsole und der Pipeline-Editor verwendet werden, die Anpassungen des *Commerce Servers* erlauben.

Commerce Server: Der *Commerce Server* besteht aus den HTML-Schablonen und dem *Commerce Server-Framework*:

- Die HTML-Schablonen werden in administrative und kundenbezogene unterschieden. Beide Arten sind mit der ASP-Technik implementiert. Die ASP-Technik erlaubt die Verwendung einer Skriptsprache in den HTML-Schablonen[6]. Die Skriptsprache ist frei wählbar, soweit ein entsprechendes Skriptmodul im IIS installiert ist. Für die Programmierung des *Commerce Servers* wird die Skriptsprache *Visual Basic* (VB) verwendet, mit der der Zugriff auf das *Framework* des *Commerce Servers* ermöglicht wird.
- Das *Framework* des *Commerce Servers* besteht aus mehreren Komponenten, mit denen die Funktionen des Systems realisiert werden:

  CS-COM-Objekte: Die *Commerce Server Component Object Model*-Objekte (CS-COM-Objekte) stellen die Laufzeitumgebung für die Verkaufssysteme des *Commerce Servers* dar. Die Objekte beruhen auf *Microsoft ActiveX Server*-Komponenten, deren Methoden und Attribute aus VB aufgerufen und gesetzt werden können. Somit können diese Objekte in den ASP-Seiten benutzt werden.

  Pipeline-Komponenten: Die Pipeline-Komponenten sind Bestandteile der Pipelines des *Commerce Servers*. Für jedes Verkaufssystem können Pipelines erstellt werden, die eine beliebige Anzahl von Komponenten enthalten können. Die Komponenten werden bei Aufruf der Pipeline sequentiell abgearbeitet.

[6] Ähnlich dem SSS des *Intershops 3.01*

Der *Commerce Server* enthält zwei Pipeline Modelle: die *Order Processing Pipeline* (OOP) und *Commerce Interchange Pipeline* (CIP). OOP-Pipelines enthalten Schritte für den Bestellprozeß und CIP-Pipelines für den Austausch von elektronischen Daten zwischen Applikationen. Ein wichtiger Aspekt für die Pipelines ist die Integration des *Microsoft Transaction Servers*, mit dem z.B. eine transaktionelle Ausführung des Bezahlungsvorgangs gesichert wird. Die Pipelines können aus den ASP-Seiten mit VB ausgeführt werden.

Externe Komponenten: Externe Komponenten sind entweder selbstentwickelte oder von Drittfirmen erhältliche Komponenten. Diese Komponenten lassen sich in die Pipelines integrieren. Damit kann die Funktionalität erweitert oder externe Systeme angebunden werden.

Die Funktionen des Verkaufssystems sind vielfältig, da der Entwickler die oben genannten Mittel einsetzen kann, um ein individuelles System zu implementieren. In Abschnitt 2.2.5 werden die möglichen Funktionen tabellarisch dargestellt.

Datenbankserver: Als Datenbankserver kann jeder beliebige Datenbankserver benutzt werden, der ODBC unterstützt. Dieses wird durch die Verwendung der *ActiveX Data Objects* (ADO) erreicht, die ODBC für eine Verbindung benutzen. Damit aber die Beispielsysteme genutzt werden können, muß der *Microsoft SQL Server* benutzt werden, da diese mit einem anderen Datenbankserver nicht lauffähig sind. Die in der Datenbank enthaltenen Daten sind von der Implementation des jeweiligen Systems abhängig.

Die herausragende Eigenschaft des *Commerce Servers* ist seine Programmierbarkeit, da die gesamte Funktionalität freigelegt ist und dem Entwickler alle Optionen offen lassen. Unterstützt wird die Entwicklung durch das mitgelieferte Entwicklungssystem *Microsoft Visual InterDev*. Als weiteres bietet der *Commerce Server* ein grafisches Werkzeug für die Bearbeitung der Pipelines an.

Nachteilig ist die fehlende Plattformunabhängigkeit, die das Produkt nur für Benutzer des Betriebssystems *Windows NT 4.0* interessant macht und die fehlende interaktive Konfigurierbarkeit der Verkaufssysteme, das zusätzliche Programmierung nötig macht.

2.2.3 *Lotus Domino.Merchant Serverpack 2.0*

Das *Lotus Domino.Merchant Serverpack 2.0*, im folgenden nur noch *Domino.Merchant* genannt, ist das Online-Verkaufssystem der Firma *Lotus* auf der Basis von *Notes*. Grundlagen der Produktbeschreibung waren eine Installation des *Domino.Merchant* und folgende Literatur [Cor97a; Cor97i; Cor97j; Cor97c; Cor97e; Kra98]. Die Beschreibung der Konzepte und die Programmierung von *Notes* werden kurz gehalten, da sie Bestandteil des Kapitels 3 sind.

Die Architektur des *Domino.Merchant* basiert auf einem *Notes*-System, das zwei Schichten der generellen Architektur aus Abschnitt 2.1 zusammenfaßt. Die Daten- und Applikationsschicht werden in den *Notes*-Datenbanken zusammengefaßt, die die Anwendungslogik und Daten enthalten (Siehe 3.2) [FPJ98]. Die resultierende zweistufige Client/Server-Architektur, die in Abbildung 2.6 dargestellt ist, enthält die zwei Schichten: Präsentations- und *Notes*-Datenbanken-Schicht.

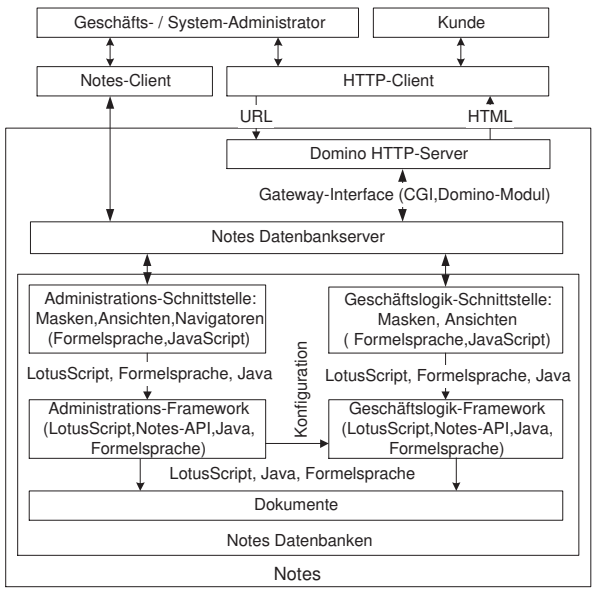

Abbildung 2.6: Architektur des *Domino.Merchant*

Präsentationsschicht: Der Kunde verwendet einen HTTP-Client, um auf das Verkaufssystem zuzugreifen. Die System- oder Geschäfts-Administratoren können dagegen entweder einen HTTP-Client oder *Notes*-Client verwenden. Der Zugriff mittels des HTTP-Clients wird durch den *Domino HTTP-Server* ermöglicht, der Bestandteil des *Notes*-Datenbankservers ist. Der *Notes*-Client greift direkt auf die *Notes*-Datenbanken über den Datenbankserver zu. Die *Gateway*-Schnittstelle zum Datenbankserver ist bedingt durch die zwei Clients unterschiedlich:

- Die *Gateway*-Schnittstelle vom HTTP-Server zum Datenbankserver wird durch das *Domino-Modul* des Datenbankservers und das *Common Gateway Interface* (CGI) realisiert. Das *Domino-Modul* fungiert als Konverter zwischen dem HTTP-Server und dem Datenbankserver, das alle Nachrichten in das jeweils korrekte Format konvertiert[7].
- Der *Notes*-Client greift direkt über das Netzwerk mit *Notes*-Anfragen auf den Datenbankserver zu und bekommt die Antworten auf dem gleichen Wege zurück.

Notes-Datenbanken: Die Datenbanken von *Notes* enthalten die notwendige Funktionalität für die Geschäfts- und Administrationslogik. Die Administrationslogik wird grob unterschieden in Geschäfts- und Systemadministration, wobei die Geschäftsadministration durch Zugriffsbeschränkungen weiter verfeinert werden kann.

Der Datenbankserver stellt die Datenbanken für den Mehrbenutzerbetrieb zur Verfügung. Die Unterscheidung der Administrations- und Geschäftslogik ist fließend, da beide in den gleichen Datenbanken gespeichert wird. Die verwendeten Techniken für die Administrations- und Geschäftslogik sind gleich.

[7] Siehe 3.5

Die Schnittstellen werden mit Masken und Ansichten realisiert, die eine Anzeige der Daten der jeweiligen Datenbank erlauben. Zusätzlich werden für die Administration Navigatoren benutzt, die eine bessere grafische Gestaltung erlauben. Innerhalb dieser Elemente wird die *Formelsprache* von *Notes* und *JavaScript* eingesetzt.

Die *Frameworks* der Administrations- und Geschäftslogik sind mit unterschiedlichen Techiken implementiert:

- *Formelsprache*
- *LotusScript*
- *Java*

Die *Formelsprache* und die Programmiersprache *LotusScript* sind proprietäre Technologien von *Notes*, wogegen *Java* eine standardisierte Programmiersprache ist.

Die Daten werden in den Datenbanken in Dokumenten gespeichert, auf die über die oben genannten Techniken leicht zugegriffen werden kann. Eine Datenbank-Schnittstelle entfällt also, da diese implizit durch die Datenbanken vorhanden ist.

Erwähnt werden soll noch der *Site Creator*, eine Datenbank für den Systemadministrator, mit der die gesamte Anwendung ohne Programmierungskenntnisse erstellt und modifiziert werden kann. Ein Nachteil des *Domino.Merchant* sind die fehlenden Anpassungsmöglichkeiten des Systems, das nur mit guten Kenntnissen von *Notes* modifiziert werden kann.

2.2.4 *iCat Electronic Commerce Suite, Professional Edition 3.01*

Die Version 3.01 des Produkts *iCat Electronic Commerce Suite, Professional Edition 3.01* (*iCat ECS*) der zur Firma *Intel* zugehörigen Firma *iCat* ist der letzte Schritt einer Entwicklung zu einem ausgereiften Online-Verkaufssystem, die bereits Anfang des Jahres 1996 begann. Als Quellen für die Beschreibung wurden Demo-Systeme im Internet, Dokumentation und Literatur benutzt [iC97e; iC97d; iC97b; iC97a; iC97c; iC97f; Kra98; Was97a; BK98]. Eine Evaluationsversion für dieses Produkt ist nicht erhältlich.

In Abbildung 2.7 ist die Architektur der *iCat ECS* dargestellt. Analog zu der generellen Architektur basiert sie auf einer dreistufigen Client/Server-Architektur. Im Unterschied zu den anderen Produkten sind zwei Datenbanken und HTTP-Server vorhanden, das in der historischen Entwicklung von *iCat ECS* begründet ist und weiter unten erklärt wird. Ähnlich wie bei der Lösung von *Microsoft* werden auch nicht HTTP- und HTML- basierte Werkzeuge benutzt, die die Arbeit mit dem System erleichtern. Die Architektur wird nachfolgend beschrieben:

HTTP-Client/-Server, Data Entry Manager/*iCat* Web Server und CARBO-Editor:
Die erste Komponente mit der Kunden und Administratoren auf den *iCat* ECS-Server zugegreifen können, ist der HTTP-Client. Der HTTP-Server kann ein beliebiger Server sein, da die *Gateway*-Schnittstelle zum *iCat* ECS-Server entweder mit ISAPI für den IIS, NSAPI für die *Netscape Web-Server* oder der standardisierten CGI-Schnittstelle realisiert wird.

Der *Data Entry Manager* (DEM) dient den Administratoren zum Import und der Eingabe von Massen-Daten für das System. Damit der DEM arbeiten kann, muß der *iCat*

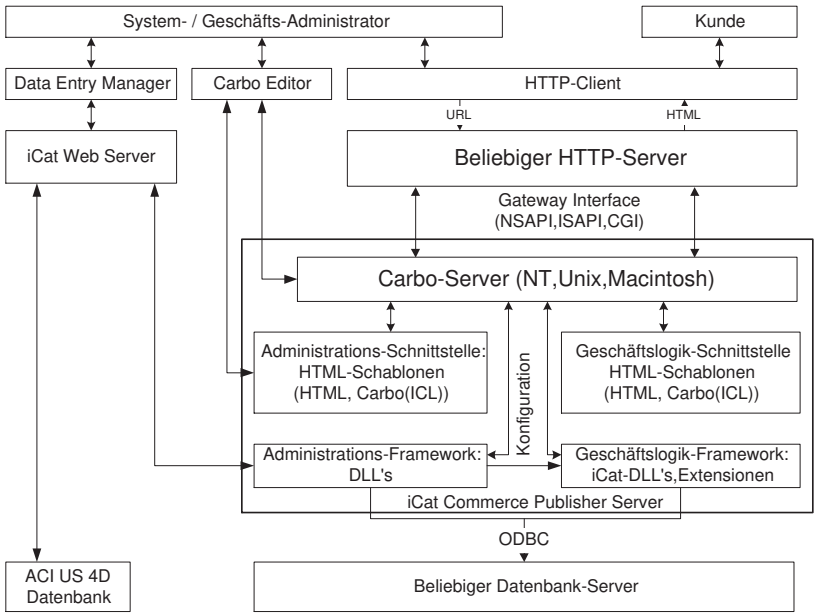

Abbildung 2.7: Architektur der *iCat Electronic Commerce Suite, Professional Edition 3.01*

Web Server, der Bestandteil des DEM ist, gestartet sein. Der Zugriff auf den *iCat Commerce Publisher Server* und die *Aci US 4D*-Datenbank ist nur mit dem *iCat Web Server* möglich.

Als Benutzerschnittstelle für den Systementwickler dient der CARBO-Editor, der eine farbliche Unterschiedung und Syntaxprüfung der in den erstellten HTML-Schablonen eingefügten *iCat Carbo Command Language* (ICL) in Zusammenarbeit mit dem CARBO-Server erlaubt.

iCat Commerce Publisher Server: Der *iCat Commerce Publisher Server* besteht aus drei unterschiedlichen Komponenten:

- Der **CARBO-Server** stellt die für die Interpretation und Ausführung der Elemente der Skriptsprache ICL notwendige Funktionalität zur Verfügung. Er setzt die ICL-Kommandos in Aufrufe der Anwendungslogik um. Für Erweiterungen von ICL enthält der CARBO-Server ein API, mit der die Funktionalität erweitert werden kann. Zusätzlich werden vorgefertigte Extensionen verwendet, die den Zugriff auf ODBC-Datenbanken, HTTP-Server, Bezahlungssoftware (z.B. *CyberCash*) und die Möglichkeit für Erweiterungen bieten.
- Die **HTML-Schablonen** für die Schnittstelle der Administration und Geschäftslogik können zusätzlich zu den HTML-Befehlen Elemente der proprietären Skriptsprache ICL enthalten. Der Sprachumfang von ICL ist dabei groß genug, um externe Systeme anzubinden. Beispielweise kann auf Datenbanken mit ODBC zugegriffen werden oder Funktionen aus Extensionen mit dem Befehl `Execute` ausgeführt werden.
- Die Anwendungslogik, die in Administrations- und Geschäftslogik aufgeteilt ist, wird mit *Dynamic Link Library*'s (DLL) realisiert. Ergänzt werden kann die An-

wendungslogik, wie bereits beim CARBO-Server erwähnt, durch weitere DLLs, die Extensionen darstellen.

Aci US 4D-Datenbank und Datenbankserver: Die *Aci US 4D*-Datenbank ist notwendig, da der DEM nur mit dieser Datenbank zusammenarbeitet. Im Lieferungsumfang ist nur der Einzelbenutzerbetrieb eines DEM vorgesehen. Wenn ein Mehrbenutzerbetrieb stattfinden soll, muß der Datenbankserver für die *Aci US 4D*-Datenbank angeschafft werden.

Als Datenbankserver des Verkaufssystems kann jeder Server benutzt werden, für die die *iCat ECS* einen ODBC-Treiber mitliefert.

Als bemerkenswerte Punkte soll die komfortable und umfangreiche Konfigurierbarkeit des Systems mittels der Administration im HTTP-Client erwähnt werden. Nachteilig und unübersichtlich sind die vielen unterschiedlichen Komponenten, die für den Betrieb der *iCat ECS* erforderlich sind. Ein weiterer wichtiger Aspekt ist die gute Unterstützung des *Staging*-Konzept, da von Beginn an zwei Datenbanken installiert werden, so daß einer Entwicklung im Testsystem nichts im Wege steht. Die Integration der Datenbanken mit dem DEM mittels des *iCat Commerce Publisher Server* ermöglicht überdies eine einfache Eingabe von Massendaten.

2.2.5 Zusammenfassung der Funktionen der Produkte

In diesem Abschnitt werden die weiteren Funktionen und Merkmale der vorgestellten Produkte tabellarisch zusammengefaßt. Die Funktionen und Merkmale werden dabei nach Bereichen geordnet dargestellt. Die Architekturmerkmale der Systeme werden nicht berücksichtigt, da diese in den vorigen Abschnitten vorgestellt wurden. Die Bereiche und Funktionen sind großteils aus [Was97a] und [Kra98] übernommen, wobei die Ergebnisse mit den aktuellen Versionen der Produkte aktualisiert wurden. Die Bereiche sind folgende:

- Produktinformationen
- Systemvoraussetzungen
- Konfiguration des Kundenservices
- Geschäftsadministration
- Systemadministration
- Gestaltungsmöglichkeiten

In Tabelle 2.1 ist die Zusammenfassung dargestellt. Die Ergebnisse für den *Microsoft Site Server 3.0, Commerce Edition* beruhen auf den Möglichkeiten der Programmierung des Produkts, da keine Standardlösung für dieses Produkt existiert. Die angegebenen Hardware-Anforderungen gelten für *Intel*-Hardware mit dem Betriebssystem *Windows NT 4.0*.

2.3 Architekturanforderungen und Kriterienauswahl für das IPOS

Damit die Anforderungdefinition an das zu realisierende IPOS erstellt werden kann, werden in diesem Abschnitt die relevanten Kriterien ausgewählt. Dieses geschieht anhand der Erkenntnisse aus den Beschreibungen der betrachteten Produkte.

Die Architektur des IPOS muß der generellen Architektur in Abschnitt 2.1 folgen, das heißt, es muß äquivalent zu den vorgestellten Produkten für jede Komponente eine entsprechende Komponente besitzen. Daraus ergeben sich die Komponenten des IPOS:

- HTTP-Server
- Schnittstellen für die Administration und Geschäftslogik
- Anwendungslogik für die Administration und Geschäftslogik
- Datenbankserver

Weitere wichtige Aspekte sind

- die Erweiterbarkeit des Systems mit einer klaren Schnittstelle zur Anwendungslogik,
- Bereitstellung einer Programmiersprache für die Anwendungslogik und
- Integrationsmöglichkeiten für externe Systeme.

Außerdem müssen Funktionen für die Geschäfts- und Administrationslogik vorhanden sein, die für ein Online-Verkaufssystem notwendig sind. In Abschnitt 2.2.5 wurden die Funktionen der betrachteten Produkte aufgelistet. Damit der Anspruch der Arbeit auf eine Implementierung eines Systems im Zeitrahmen der Arbeit realisierbar bleibt, ist eine Reduktion der möglichen auf die notwendigen Funktionen notwendig.

Die Identifizierung der notwendigen Komponenten der Geschäftslogik erfolgt anhand des Standards *Open Buying on the Internet* (OBI) Version 1.1 [Con98], der ein Modell für interorganisationelle Anwendungen zwischen Unternehmen spezifiziert. Die Konzepte lassen sich in Grenzen auf Anwendungen des Typs „Kunde zu Unternehmen“ wie das IPOS anwenden[8]. Als notwendige Funktionen lassen sich nach dem OBI-Standard und [Kra98; ADGY99] folgende identifizieren:

1. Die **Benutzerregistrierung** muß für die notwendige Identifizierung und Authentifizierung des Kunden sorgen.
2. Der **Produktkatalog** bietet die Möglichkeit, Produkte auszusuchen und dem Warenkorb hinzuzufügen.
3. Der **Warenkorb** ist ein Behälter für Produkte, die bestellt werden sollen.

[8] Vergleiche Kapitel 1

4. Die **Bestellungen**, die aus dem Warenkorb erzeugt werden, enthalten Informationen über die Bezahlung und ermöglichen die Weiterverarbeitung bis zur Lieferung und Bezahlung.

Die notwendigen Komponenten der Administrationslogik lassen sich in fünf Bereiche unterteilen, die in allen beschriebenen Produkten vorhanden sind:

1. Die Parametrisierbarkeit des Systems muß durch eine **Systemadministration** möglich sein, so daß die Konfiguration ohne Programmierung verändert werden kann.
2. Die **Benutzerverwaltung** erlaubt die Ansicht, Eingabe und Modifizierung von Benutzern des Systems.
3. Die **Produktverwaltung** erlaubt die Ansicht, Eingabe und Modifizierung von Produkten.
4. Die **Verwaltung der Bestellungen** muß die Weiterverarbeitung der Bestellungen ermöglichen.
5. **Auswertungen** sollten möglich sein, anhand derer Aussagen über die Verwendung des Systems getroffen werden können.

2.4 Anforderungsdefinition und Architektur des IPOS

Nachdem im vorigen Abschnitt die Anforderungen an die Architektur und die zu realisierenden Kriterien für ein Online-Verkaufssystem geklärt worden sind, werden in diesem Abschnitt die Ziele und Vorraussetzungen der Arbeit berücksichtigt. Die Ziele und Vorraussetzungen sind folgende:

- Das Produkt *Lotus Notes* ist die Plattform für die Implementierung.
- Die Kommunikation und Kooperation mit dem Kunden erfolgt gemäß dem Modell der *Business Conversations*.
- Die betriebswirtschaftliche Standard-Software *SAP R/3* soll integriert werden.

Aus beidem resultiert die Anforderungsdefinition und Architektur des IPOS. Die Architektur des IPOS ist in Abbildung 2.8 dargestellt. Analog zum *Domino.Merchant*, der auf der Plattform *Lotus Notes* beruht, hat das IPOS eine zweistufige Client/Server-Architektur. Die Daten- und Applikationsschicht werden in den *Notes*-Datenbanken zusammengefaßt, die die Anwendungslogik und Daten enthalten. Aufgrund der fast gleichen Architektur des *Domino.Merchants* werden in diesem Abschnitt nur die Punkte betrachtet, in denen sich die beiden Architekturen unterscheiden. Die weiteren Komponenten und Schnittstellen können in Abschnitt 2.2.3 und Kapitel 3 nachgelesen werden.

Die Unterschiede in der Architektur der beiden Systeme sind:

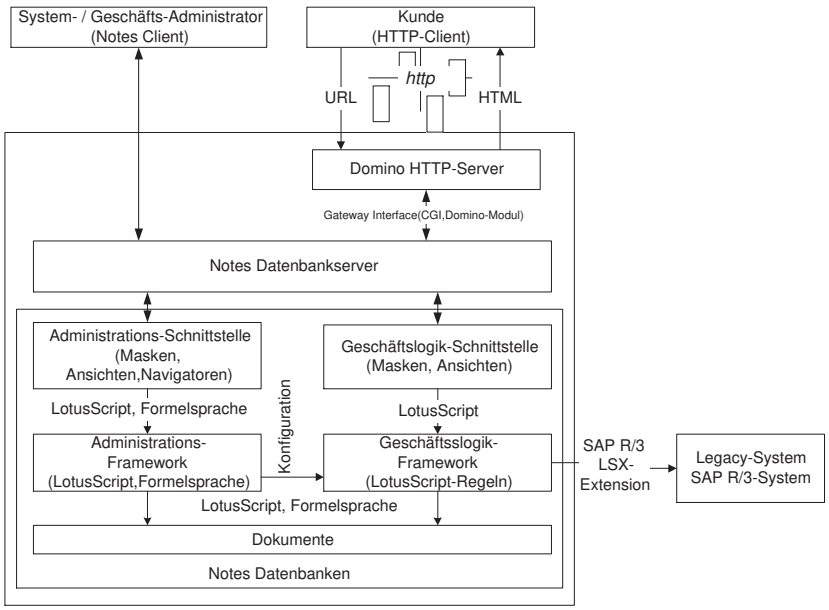

Abbildung 2.8: Architektur des IPOS

- Die System- und Geschäftsadministratoren greifen nur über den *Notes*-Client auf das System zu, da dieser mehr Sicherheit und Möglichkeiten bietet.
- Die Kommunikation mit dem Kunden erfolgt gemäß dem Modell der *Business Conversations*.
- Die Geschäftslogik wird homogen mit *LotusScript* implementiert. Die Administrationslogik verwendet *LotusScript* und die *Formelsprache*.
- Das *SAP R/3*-System ist für die Geschäfts- und Administrationslogik über die *SAP R/3 LSX*-Extension zu integrieren, damit die Funktionalität des *SAP R/3*-Systems genutzt werden kann (Siehe Kapitel 3).

Das IPOS muß die im vorigen Abschnitt vorgestellte Funktionalität der Geschäfts- und Administrationslogik implementieren. Die geforderte Integration des *SAP R/3*-Systems soll folgende Funktionalität ermöglichen:

1. Die Kunden sollen automatisch in das *SAP R/3*-System eingetragen werden. Außerdem soll ein Import von Kunden in das IPOS möglich sein.
2. Die Produkte sollen aus dem *SAP R/3*-System importiert werden.
3. Die Verfügbarkeit von Produkten kann von Kunden im IPOS geprüft werden. Die Daten werden während der Laufzeit aus dem *SAP R/3*-System ermittelt.
4. Die Bestellungen von Kunden werden während der Laufzeit automatisch im *SAP R/3*-System angelegt. Außerdem können die Kunden den jeweiligen Status ihrer Bestellungen einsehen.

5. Die Weiterverarbeitung und Bezahlung erfolgt im *SAP R/3*-System, in dem die generierten Bestellungen abgearbeitet werden.

Außerdem muß das IPOS gemäß des vorigen Abschnittes folgende Merkmale besitzen:

- Erweiterbarkeit des Systems mit einer klaren Schnittstelle zur Anwendungslogik
- Bereitstellung einer Programmiersprache für die Programmierung
- Integrationsmöglichkeiten für externe Systeme

	Intershop 3.01	*iCat Commerce Suite, Prof.*	*Microsoft Site Server 3.0, Commerce Edition*	*Lotus Domino Merchant 2.0*
Produkt-informationen				
Anbieter	*Intershop GmbH*	*iCat Europe Ltd.*	*Microsoft GmbH*	*Lotus GmbH*
Preis	11049 $	9995 $	4609 $	4052 $
Evaluationsversion	X	-	X	X
Demo-Shops im Internet	X	X	-	-
Internetadresse	www.intershop.com	www.icatcorp.com	www.microsoft.com	www.lotus.com
Systemvorraussetzungen				
Hardware-Anforderungen				
CPU	Pentium 200	Pentium	Pentium 100	Pentium 133
Arbeits-/Festplattenspeicher	128 /300 MB	32 /150 MB	64 MB /1 GB	64 MB /1 GB
Konfiguration des Kundenservice				
Mehrsprachigkeit	X	-	-	-
Persistenter Einkaufswagen	X	X	X	X
Produkte vormerken	X	-	X	-
Volltextsuche	X	X	X	X
Suche einschränkbar	-	-	X	X
Abfragen des Auftragsstatus	X	-	X	X
Abfragen der zurückliegender Einkäufe	X	-	X	X
Geschäftsadministration				
Kundenverwaltung	X	X	X	X
Rechnung/Lieferschein/Lager	X	-	-	-
Konfiguration der Kundenregistrierung				
Erkennung registrierter Kunden	X	X	X	-
anonyme Kunden	X	X	X	X
Kundenprofile allgemein	X	X	X	X
Kundenprofile nach Kaufverhalten	X	-	-	-
Verwaltung von Sonderangeboten				
Produktspezifische Sonderangebote	X	X	X	X
Cross-Selling(Ergänzungsprodukt)	X	X	X	-
Up-Selling(höherwertiges Produkt)	X	X	X	-
Checkout Sale(Sonderaktionen an der Kasse)	-	X	-	-
Ersatzangebot für vergriffene Produkte	-	X	-	-
Produktspezifische Rabatte	X	X	X	-
Rabbat nach der Abnahmemenge	-	X	X	-
Rabbat nach dem Umsatz	X	X	X	-
Kundenspezifische Rabbatsysteme	X	X	X	-
Auswertungen				
Treffer auf Produkte	X	X	X	X
Treffer auf Produktgruppen	X	X	X	-
Übersicht der beliebtesten Produkte	X	X	X	X
Zahl der benutzten Einkaufswagen	X	-	X	X
Durchschnittsbetrag eines Einkaufs	X	-	X	X
Umsatz	X	-	X	X
Gewinn	X	-	-	-
Steuerberechnungen				
Mehrwertsteuer für Warengruppen setzbar	X	bis zu 5	-	-
Regionale Steuersätze möglich	X	X	X	-
Lösung von Dritthersteller	Taxware	-	-	Taxware
Versandarten				
Definition zusätzlicher Versandarten	X	X	X	X
Berechnung nach Preis	X		X	X
Berechnung nach Gewicht	X		X	X
Berechnung nach Volumen	-		X	-
Bezahlung				
Kreditlinie festlegen und überwachen	X	-	X	-
Methodenpalette kundenbezogen einschränken	X	X	X	-
gegen Rechnung	X	X	X	X
per Nachnahme	X	X	X	-
Lastschrift	X	X	X	-
manuelle Kreditkartentransaktion via SSL	X	X	X	X
First Virtual	-	X	-	-
CyberCash	X	X	X	X
CheckFree	-	X	-	-
OpenMarket OM-SecureLink	-	X	-	-
ICVerify	X	-	-	-
Systemadministration				
mit einem HTTP-Client im LAN	X	X	X	X
mit einem HTTP-Client im Internet+SSL	X	-	X	X
Mehrbenutzerfähigkeit	X	X	X	X
Staging-Konzept	X	X	-	-
Importfunktionen	X	X	-	X
Dokumentation der Schnittstellen	X	X	X	-
Gestaltungsmöglichkeiten				
Organisation des Produktkatalogs				
hierarchisch organisiert	X	X	X	X
Anzahl der Hierarchieebenen	unbegrenzt	4	unbegrenzt	unbegrenzt
Produktes in mehreren Kategorien	X	X	-	max. 5
Verwaltung von Produktvarianten	X	-	X	-
Produktattribute beliebig hinzufügen	X	max. 25	-	max. 10
Bearbeitung von HTML-Templates				
Template-Editor	X	X	X	X
Kennzeichnung der Elemente	-	X	X	X
Syntaxüberprüfung der Elemente	-	X	X	X
Verwaltung produktspezifischer Hyperlinks	X	X	X	X
Diskussionsforen	-	-	-	X
sofortiger Download von Software usw.	-	-	X	X

Tabelle 2.1: Zusammenstellung der Merkmale der Verkaufssysteme

Kapitel 3

Lotus Notes

In diesem Kapitel werden die Konzepte des kommerziellen Systems *Lotus Notes*[1] vorgestellt. *Notes* wurde für die Implementierung der *Notes Business Conversations* und des *Internet Point of Sale* gemäß der Anforderungen der Arbeit verwendet.

Eingeordnet wird *Notes* in das Fachgebiet der *Computer Supported Collaborative Work* (CSCW). CSCW soll die flexible Zusammenarbeit mehrerer Personen durch den Einsatz von vernetzten Computern unterstützen. Der Oberbegriff für alle Systeme, die dem Gebiet der CSCW zugeordnet werden, ist *Groupware*. Für den Begriff *Groupware* existieren in der Literatur mehrere Definitionen, die sich inhaltlich unterscheiden. Als gemeinsamer Kernpunkt der Definitionen wird einheitlich die Teamarbeit genannt. Eine Definition für *Groupware* aus [FPJ98], die eine gute Charakterisierung von *Notes* bildet, ist folgende:

Groupware sind Software-Systeme, die Arbeitsgruppen bei der Lösung von wenig strukturierten Aufgabenstellungen unterstützen. Die Basis für diese Unterstützung sind drei Hauptfunktionen:

- Kommunikation: Die Übermittlung von Nachrichten in elektronischen Formaten ist möglich.
- Kooperation: Für die Arbeitsgruppe steht ein gemeinsam genutzter, virtueller Arbeitsplatz zur Verfügung.
- Koordination: Es existieren Möglichkeiten, die realen Abläufe auf den virtuellen Arbeitsplatz abzubilden.

Das Kapitel gliedert sich in folgende Abschnitte. Abschnitt 3.1 beschreibt die historische Entwicklung von *Notes*. Die Architektur wird in Abschnitt 3.2 vorgestellt. In Abschnitt 3.3 werden die strukturellen Konzepte von *Notes* erläutert. Darauf werden die Möglichkeiten der Programmierung insbesondere *LotusScript* und die *Formelsprache* in Abschnitt 3.4 beschrieben. Die letzten beiden Abschnitte 3.5 und 3.6 beschäftigen sich abschließend mit den Integrationsmöglichkeiten für das Internet und dem *SAP R/3*-System. Die Integrationsmöglichkeiten für das Internet werden über Erweiterungen der *Notes* Konzepte erreicht, die einen einfachen

[1] Im weiteren Verlauf nur noch mit *Notes* bezeichnet.

Zugriff auf *Notes* aus dem Internet ermöglichen. Integrationen externer Systeme erfolgen über Erweiterungen der Programmiersprache *LotusScript*, die *LotusScript*-Extensionen (LSX) genannt werden. Für die in dieser Arbeit geforderte Integration des *SAP R/3*-Systems existiert eine LSX, die für die Integration benutzt wurde.

3.1 Historie

Die historische Entstehung der ersten Version von *Notes*, die im Jahr 1989 von *Lotus* dem Markt vorgestellt wurde, hatte ihren Ursprung in dem Produkt *Plato Notes*, das im *Computer-based Education Research Laboratory* der Universität von Illinois entwickelt wurde und im Jahr 1973 Marktreife erlangte [Lot97]. Die wesentlichen Merkmale der Version 1.0 von *Notes*, das die Plattformen *DOS 3.1/4.0* und *OS/2* unterstützte, waren:

- Zweistufige Client/Server-Architektur
- Konzept der Replikation
- *Formelsprache* für eigene Entwicklungen
- Kryptographie durch *RSA-public key* Technologie
- Zugriffskontrolliste für Datenbanken
- Interaktiver entfernter Zugriff auf den Server über Telefonverbindungen
- *e-Mail*-Funktionalität
- Import/Export von Daten

Im Jahr 1990 folgte die Version 1.1, die die zusätzlichen Plattformen *OS/2 1.2 Extended Edition*, *Novell Netware Requester* für *OS/2 1.2*, *Novell Netware/386* und *Windows 3.0* unterstützte.

Die Version 2.0 wurde im Jahre 1991 freigegeben und enthielt unter anderem Erweiterungen für die Entwicklung von Anwendungen und das *e-Mail*-System. Außerdem wurde eine C-Schnittstelle erstellt, die den Zugriff auf *Notes* von externen Systemen und Erweiterungen durch Fremdfirmen ermöglichte.

Mit der Freigabe der Version 3.0 im Jahr 1993 wurde erstmals die *Apple*-Plattform unterstützt. Als zusätzliche Funktionen wurden z.B. die Volltextsuche in Datenbanken, selektive Replikation, uniforme C-Schnittstellen für die unterschiedlichen Plattformen und administrative Erweiterungen hinzugefügt.

Die Freigabe der Version 4.0 erfolgte im Januar 1996. Diese Version enthielt verschiedene Erweiterungen, wovon zwei wichtige hier beschrieben werden. Die erste Erweiterung betrifft die Integrationsmöglichkeiten für das Internet, die den Zugriff aus *Notes* auf das Internet und umgekehrt erlauben. Die zweite Erweiterung ist der Einbau der objektorientierten Programmiersprache *LotusScript*, die bis dahin nur in Zusatzprodukten erhältlich war. Die Realisierung

von komplexen Anwendungen wurde dadurch wesentlich vereinfacht, da die Möglichkeiten der bisher eingesetzten *Formelsprache* sehr eingeschränkt sind.

Im weiteren Verlauf folgte die Version 4.5 im Dezember 1996, die z.B. die Erweiterung des *Notes* Datenbankservers um einen HTTP-Server enthielt. Im September 1997 wurde die Version 4.6 freigegeben, die unter anderem die Verwendung der Programmiersprache *Java* in Agenten ermöglicht. Generell kann gesagt werden, daß *Lotus* seit der Version 4.0 versucht, Internet-Standards wie HTTP, POP3, SMTP, SSL, usw. zu verwenden, um *Notes* von einem proprietären System in ein offenes System zu wandeln.

3.2 Die Architektur von Notes

Die Architektur von *Notes* beruht auf einem zweistufigen Client/Server-Modell. Abweichend vom dreistufigen Modell, das aus einer Präsentations-, Applikations- und Datenschicht besteht, wird die Applikations- und Datenschicht in *Notes* Datenbanken zusammengefaßt [FPJ98][2]. Die Präsentationsschicht, die die Benutzerschnittstelle darstellt, wird entweder über einen HTTP- oder *Notes*-Client zur Verfügung gestellt. Der *Notes* Datenbankserver ermöglicht den parallelen Zugriff unterschiedlicher Benutzer auf Datenbanken. Überdies unterstützt die Kommunikation und Interaktion zwischen Datenbankservern dezentrale Organisationsformen und die Schaffung von virtuellen Arbeitsplätzen. Die Abbildung 3.1 zeigt die Architektur von *Notes* unter Berücksichtigung der oben genannten Schichten.

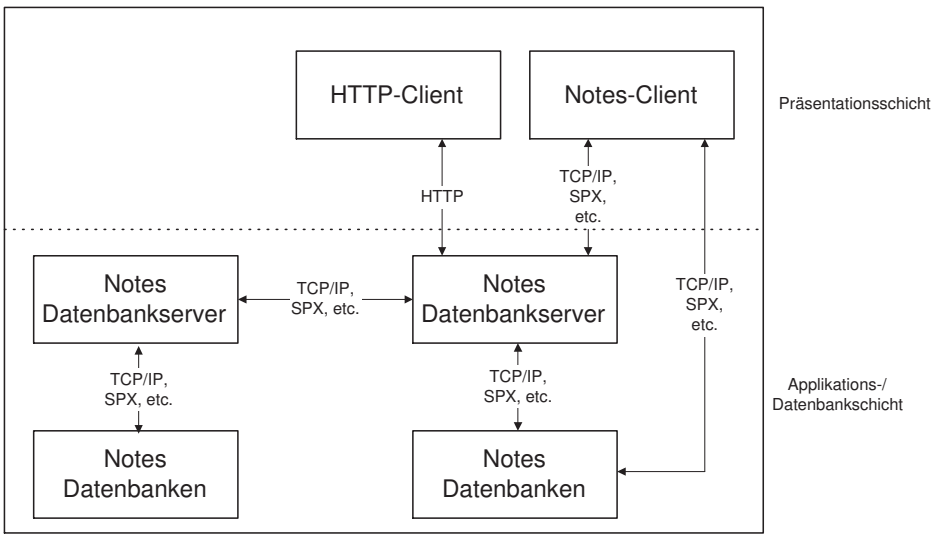

Abbildung 3.1: *Notes* Architektur

HTTP-Client: Der HTTP-Client ist eine von *Notes* unabhängige Anwendung, die die Benutzerschnittstelle der Internetbenutzer ist. Die Kommunikation mit dem Datenbankserver erfolgt über das HTTP-Protokoll. Die Anfragen werden von dem HTTP-Server

[2] Detaillierte Beschreibungen der Client/Server-Architektur in der Literatur sind z.B. [Pou97; Dad96; Nie95].

des Datenbankservers angenommen und zur Weiterverarbeitung in *Notes* konforme Anfragen konvertiert und weitergeleitet. Analog wird das Resultat der Anfrage wieder an den HTTP-Client übermittelt. Bislang existieren noch Einschränkungen bezüglich der möglichen Funktionen im HTTP-Client. Diese sind hauptsächlich bedingt durch die Beschränkungen des HTTP-Protokolls und der HTML-Sprache. Zum Beispiel kann die Gestaltung von Datenbanken nicht modifiziert werden (Siehe Abschnitt 3.5).

Notes-Client: Die klassische Benutzerschnittstelle ist der *Notes*-Client, der eine eigenständige Anwendung darstellt. Der *Notes*-Client greift über ein Netzwerk, das mit einem von *Notes* unterstützten Netzwerkprotokoll (TCP/IP, SPX, etc.) betrieben werden muß, auf die Datenbanken des Datenbankservers zu. Als Benutzeroberfläche dient ein sogenannter Arbeitsplatz, auf dem Datenbanken in Form von Ikonen abgelegt werden können. Entwickler, Administratoren und Benutzer können den Arbeitsplatz gleichermaßen nutzen, um auf Datenbanken in ihrem jeweiligen Kontext zuzugreifen. Im Gegensatz zum HTTP-Client ist ein *Notes*-Client nicht vom Datenbankserver abhängig, da er ohne Verwendung eines Datenbankservers mit Datenbanken arbeiten kann, die dann exklusiv für den Benutzer geöffnet werden. Der gravierende Nachteil der exklusiven Nutzung ist der fehlende Mehrbenutzerbetrieb, so daß diese Art der Nutzung nur für lokale Datenbanken angewendet werden sollte.

Notes Datenbanken: Der wichigste Bestandteil von *Notes* sind die Datenbanken, wobei dieser Ausdruck nicht präzise ist. Präziser wäre die Formulierung „Datenbank mit Anwendungslogik“, da Datenbanken Daten sowie Anwendungslogik enthalten. *Notes*-Anwendungen bestehen aus einer oder mehreren Datenbanken. Die Datenbanken aggregieren Dokumente und Gestaltungselemente. Dokumente enthalten die Daten und Gestaltungselemente die Anwendungslogik und grafische Darstellung der Dokumente.

Notes Datenbankserver: Der *Notes* Datenbankserver ermöglicht den parallelen Mehrbenutzerbetrieb auf Datenbanken. Zusätzlich zu dem Datenbankserver existieren noch weitere Server-Module wie z.B. der *Replicator*, *Router*, *Agent manager*, *Scheduler*, *Calendar* und POP3. Die Verwendung der Server-Module ist optional. Davon ermöglicht z.B. das Modul des *Replicators* die Datenbankreplikation mit anderen möglicherweise geographisch verteilten Datenbankservern und das *Router*-Modul die Weiterleitung von *e-Mails*. Der Verbindungsaufbau zwischen den Datenbankservern erfolgt entweder über ein lokales Netzwerk, WAN oder Telefonverbindungen.

3.3 Strukturelle Konzepte

Die Erläuterung der strukturellen Konzepte von *Notes* geschieht anhand der enthaltenen Komponenten. Die Beziehungen zwischen den unterschiedlichen Komponenten verdeutlichen die Konzepte. Als Basis für die Erläuterung dient das Klassendiagramm in Abbildung 3.2. Die hervorgehobenen Klassen in der Abbildung zeigen die hierarchische Struktur der Komponenten auf. Aufgrund der Komplexität von *Notes* konnten nicht alle Zusammenhänge und Komponenten dargestellt werden, so daß für weitergehende Recherchen auf die vorhandene Literatur verwiesen wird [Cor97b; SF98; Cor97l; FPJ98].

Um einen Vergleich mit anderen Konzepten herzustellen, werden im weiteren Verlauf die

Komponenten, die die Datenstrukturen betreffen, mit dem relationalen Modell verglichen. Das relationale Modell wurde aufgrund seiner weiten Verbreitung und Bekanntheit ausgewählt. Eine Beschreibung des relationalen Modells ist in [LS87] zu finden.

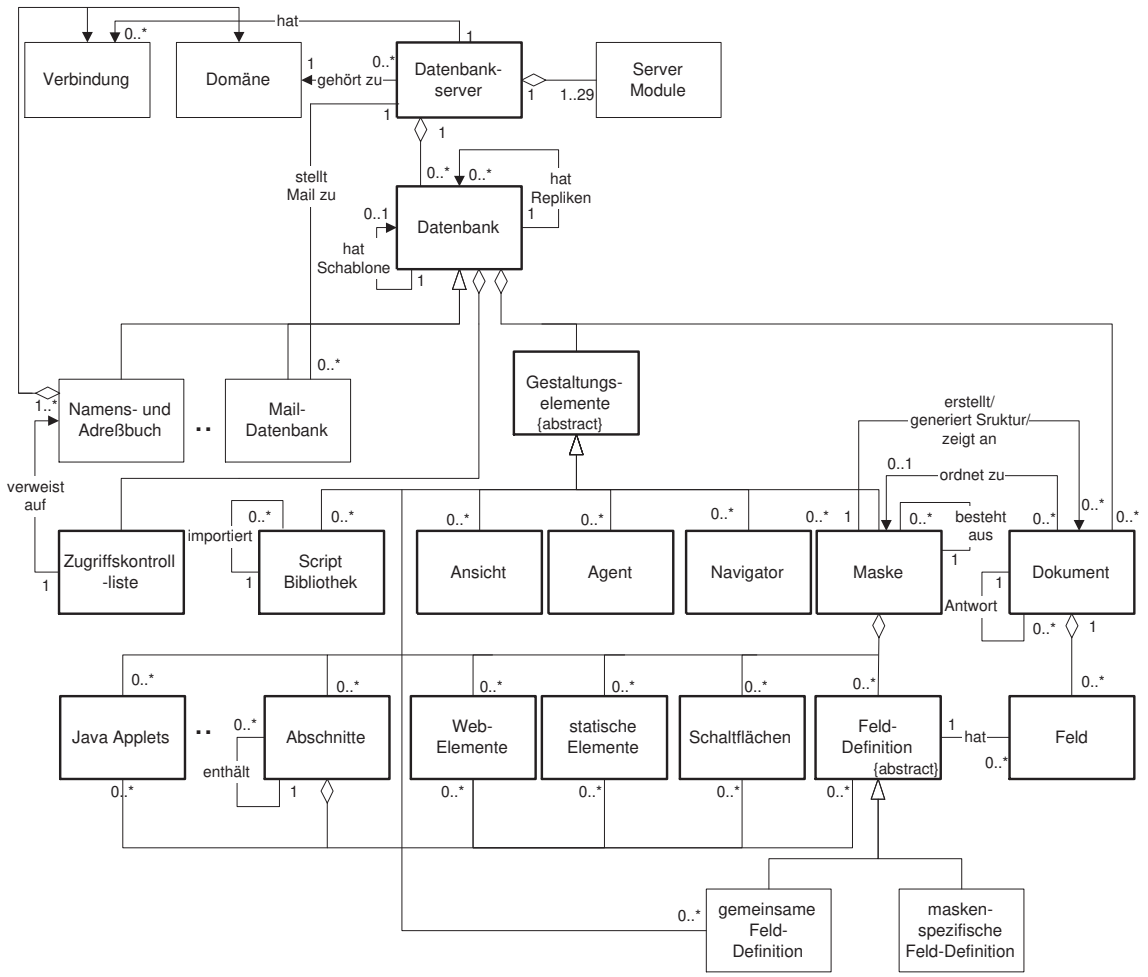

Abbildung 3.2: Struktur von *Notes*

Die Wurzel für ein *Notes*-System ist ein **Datenbankserver**, der eine beliebige Anzahl von Datenbanken aggregieren kann. Ein Datenbankserver hat einen Namen und aggregiert maximal 29 Server-Module, wobei das Modul des Datenbankservers aggregiert werden muß[3]. Eines der wichtigsten Server-Module ist der *Replicator*, der den Abgleich von Datenbanken ermöglicht. Als Basis für das Konzept der Replikation, die die dezentralen Organisationsformen und virtuellen Arbeitsplätze ermöglicht, kann ein Datenbankserver beliebig viele Verbindungen zu anderen Datenbankservern definieren. Verbindungen sind für die Kommunikation mit Datenbankservern anderer *Notes-Named-Networks* (NNN)[4] oder Domänen[5] erforderlich. Die Domänen und NNN untergliedern *Notes*-Netzwerke, die aus Datenbankservern bestehen.

[3] Aufgrund des Moduls des Datenbankservers wird der Server Datenbankserver genannt.
[4] Ein NNN bezeichnet ein Netzwerk von physikalisch verbundenen Datenbankservern.
[5] Domänen bestehen aus Datenbankservern, die dasselbe Namens- und Adreßbuch benutzen.

Verbindungen, Domänen und NNN's werden von der Datenbank *Namens- und Adreßbuch* aggregiert, das Metadaten z.B. über die Netzwerke von Datenbankservern und die Konfigurationseinstellungen der Server-Module beinhaltet.

Die nächste Hierarchieebene stellt die **Datenbank** dar. Wie bereits in Abschnitt 3.2 erwähnt, bestehen *Notes*-Anwendungen aus einer oder mehreren Datenbanken. Als Klassifizierung des Datenbank-Modells von *Notes* ist das hierarchische Modell geeignet, da Datenbanken die Bildung von hierarchischen Datenstrukturen unterstützen. Es können aber auch andere Modelle wie das relationale Modell in einer *Notes*-Datenbank nachgebildet werden.

Eine Datenbank hat eine Reihe von Eigenschaften, wovon einige wichtige erklärt werden sollen:

- Name: Jede Datenbank hat einen Namen.
- Server: Der Rechner auf dem sich die Datenbank befindet.
- Replik-ID: Die Replik-ID besteht aus einer Kombination von 16 Buchstaben und Nummern.
- Replikationseinstellungen: Ermöglicht selektive Replikation der Komponenten der Datenbank.
- Beruht auf Schablone: Gibt den Namen der Schablone-Datenbank an.

Das bereits angesprochene Konzept der Replikation soll anhand der Eigenschaften „Replik-ID“ und „Replikationseinstellungen“ erklärt werden. Anhand der Replik-ID werden Repliken, die Datenbanken gleicher Struktur sind, identifiziert. Wenn mehrere Datenbanken eine wertgleiche Replik-ID besitzen, werden sie von dem *Replicator* Server-Modul auf dem gleichen oder zwischen unterschiedlichen Datenbankservern repliziert. Die Replikation geschieht anhand der Replikationseinstellungen, die eine selektive Replikation der Datenbanken ermöglichen.

Die Eigenschaft „Beruht auf Schablone“ der Datenbanken wird für die Entwicklung benutzt. Schablonen sind Datenbanken. Eine Datenbank, die auf einer Schablone beruht, erhält alle Gestaltungselemente der Schablone. Für Datenbanken existieren eine Anzahl von vorgefertigten Schablonen. Dieses sind z.B. Schablonen für Diskussions-, Namens- und Adreßbuch- und *Mail*-Datenbanken. Die Datenbank *Namens- und Adreßbuch*, die Metadaten enthält, wurde bereits angesprochen. Die *Mail*-Datenbanken werden für die *e-Mail*-Funktionalität von *Notes* benutzt. Den adressierten *Mail*-Datenbanken wird vom Datenbankserver die *e-Mail* mittels des *Router* Server-Moduls zugestellt. Diskussionsdatenbanken sind nur eine Ausprägung von mehreren vorgefertigten Anwendungen, die benutzt werden können.

Weiterhin aggregieren Datenbanken folgende Komponenten:

- Zugriffskontrolliste: Bestimmt die Zugriffsrechte von Benutzern für die Datenbank.
- Gestaltungselemente: Enthalten Strukturinformationen und Programme der Datenbank[6].

[6]Die Programmierung von *Notes* wird in Abschnitt 3.4 beschrieben

- Dokumente: Stellen die Informationen dar, die in der Datenbank gespeichert sind.

Ein wesentlicher Unterschied der Datenbanken zu dem relationalen Modell ist, daß die Anwendungslogik, Strukturinformationen und Dokumente zusammen gespeichert werden. Die Anwendungslogik wird im relationalen Modell nicht betrachtet. Das relationale Modell beinhaltet Schemainformationen und Datensätze. Die Strukturinformationen der Gestaltungselemente entsprechen den Schemainformationen und die Dokumente den Datensätzen des relationalen Modells.

Die Gestaltungselemente, die die Struktur der Datenbank festlegen, untergliedern sich in

- Maske,
- Ansicht,
- gemeinsame Feld-Definition,
- Navigator,
- Agent und
- Script-Bibliothek.

Erstellt werden Gestaltungselemente in der Gestaltungssicht der Datenbank, die die Gestaltungselemente in Kategorien geordnet anzeigt. Für die Modifizierung werden im Gestaltungsmodus adäquate Editoren zur Verfügung gestellt. Eine gemeinsame Eigenschaft der Gestaltungselemente ist ein Name, über den sie identifiziert werden.

In den folgenden Abschnitten wird auf die Komponenten der Datenbanken die Gestaltungselemente, Dokumente und das Sicherheitskonzept von *Notes* eingegangen. Masken und Feld-Definitionen werden zusammen beschrieben, da diese Konzepte miteinander verbunden sind. Ein eigener Abschnitt widmet sich dem Sicherheitskonzept, in dem die Rolle der Zugriffskontrolliste offensichtlich wird. Der Vergleich mit dem relationalen Modell wird nur für Masken, Ansichten und Feld-Definitionen vorgenommen, da die übrigen Komponenten für die Anwendungslogik benutzt werden. Abschließend werden die Vergleiche der Komponenten mit dem relationalen Modell tabellarisch zusammengefaßt.

3.3.1 Masken und Feld-Definitionen

Masken sind elektronische Formulare, deren Zweck die Anzeige und Erstellung von Dokumenten ist. Wichtige Eigenschaften von Masken sind der Maskentyp und die Einstellung bezüglich der Konsolidierung von Replikationskonflikten.

Der **Maskentyp** gibt an, ob die mit der Maske erzeugten Dokumente Haupt- oder Antwortdokumente sind. Damit werden beliebige hierarchische Strukturen unterstützt. Der Maskentyp Hauptdokument ermöglicht die Erstellung von alleinstehenden Dokumenten. Dagegen beschränkt der Maskentyp Antwortdokument die Erstellung auf Dokumente, die im Zusammenhang mit einem beliebigen Vaterdokument[7] erstellt werden müssen und automatisch eine

[7] Ein Vaterdokument kann eine Haupt- oder Antwortdokument sein.

Referenz auf das Vaterdokument beinhalten. Diese sind dann Antworten zu dem Vaterdokument. Der Mechanismus ist mit der Funktionsweise der *Newsgroups* zu vergleichen.

Die Angabe bezüglich der **Konsolidierung von Replikationskonflikten** gibt an, ob die Replikation für Dokumente dieser Maske auf Feldebene ausgeführt werden soll. Das Konzept der Replikation wird durch die Verwendung auf Feldebene mächtiger, da Replikationskonflikte nur auftreten, wenn in zwei Datenbank-Repliken gleichzeitig[8] der Wert desselben Feldes eines Dokuments modifiziert wird. Im anderen Fall reicht die gleichzeitige Speicherung mit der Modifizierung unterschiedlicher Felder des Dokuments in zwei Datenbank-Repliken, um einen Replikationskonflikt zu erzeugen.

Masken aggregieren

- Feld-Definitionen,
- Abschnitte,
- Schaltflächen,
- statische Elemente,
- *Java-Applets*,
- *Web*-Elemente,
- Dateianhänge,
- Objekte und
- *Layout*-Bereiche.

Zusätzlich können Masken wiederum aus Masken bestehen. So können wiederverwendbare Teilmasken erstellt werden, die in beliebig vielen Masken benutzt werden können.

Die **Felddefinitionen** einer Maske legen die Struktur eines Dokumenttyps in der Datenbank fest. Sie untergliedern sich in gemeinsame oder maskenspezifische. Gemeinsame Feld-Definitionen können von mehreren Masken aggregiert werden und ermöglichen eine Wiederverwendung. Maskenspezifische Feld-Definitionen werden in einer Maske erstellt und können nur in derselben Maske verwendet werden. Feld-Definitionen haben einen Namen und eine Typangabe. Die Typangabe besteht aus der Angabe des Verhaltens und Datentyps. Eine Feld-Definition kann sich auf vier verschiedene Weisen verhalten:

- Bearbeitbar: Werte können in das Feld eingetragen werden.
- Berechnet: Der Feldwert wird anhand einer Formel[9] berechnet.
- Berechnet zur Anzeige: Der Feldwert wird bei der Anzeige berechnet und nicht gespeichert.
- Berechnet beim Anlegen: Der Feldwert wird einmalig beim Anlegen berechnet.

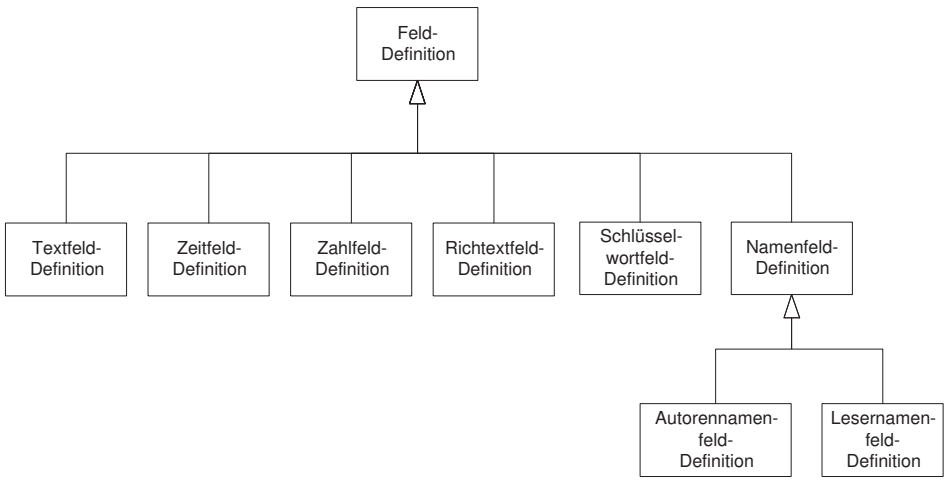

Abbildung 3.3: Datentypen der Feld-Definitionen

Die für Feld-Definitionen existierenden Datentypen sind:

- Text: alphanumerische Texte
- Zeit: Zeit- und Datumsangaben
- Zahl: Zahlen der Typen Integer, Real und Währung
- *Richtext*: Inhalte beliebiger Art und Größe z.B. Objekte, Grafiken und Dateien
- Schlüsselwort: Listen für eine vordefinierte Auswahl
- Namen, Autoren- und Lesernamen: Namen sind die *Notes*-Namen der Benutzer[10]. Autoren erhalten Autorenrechte auf das zugehörige Dokument. Analog erhalten die Leser Leserrechte auf das zugehörige Dokument.

Eine weitere Eigenschaft ist, ob Mehrfachwerte erlaubt sind. Mehrfachwerte stellen Listen von Werten des spezifizierten Datentyps dar.

Thematische Zusammenfassungen der Komponenten einer Maske werden durch **Abschnitte** ermöglicht. **Schaltflächen** können alternativ Aktionsschaltflächen oder grafische Schaltflächen sein. Beide Arten führen bei einer Betätigung eine Aktion aus. Die **statischen Elemente** untergliedern sich in Text-, Grafik- und Tabellenelemente.

Weitergehende Beschreibungen der Eigenschaften und Elemente einer Maske können der Literatur [FPJ98; Cor97l; CHT+97] entnommen werden.

Für den Vergleich mit dem relationalen Modell werden nur die Elemente betrachtet, die Einfluß auf die Datenstruktur haben. Eine Maske ist vergleichbar mit der Schemadefinition einer

[8] „Gleichzeitig“ bedeutet in diesem Fall zwischen zwei Replikationen der Datenbanken.
[9] Die *Formelsprache* wird in Abschnitt 3.4 vorgestellt.
[10] *Notes*-Namen sind hierarchische Namen, die z.B. den Datenbankserver und die Domäne enthalten.

Relation, da beide eine Datenstruktur festlegen. Die Feld-Definitionen entsprechen den Definitionen der Relationenelementen. Ein wichtiger Unterschied ist die Möglichkeit zur Definition von Mehrfachwerten in Feld-Definitionen, die erweiterte Möglichkeiten bei der Darstellung von Beziehungen bietet. Die im relationalen Modell vorhandenen Domänen können nicht definiert, aber durch die Anwendungslogik[11] nachgebaut werden.

3.3.2 Dokumente

Dokumente sind die kleinste diskrete Einheit, in der Informationen persistent gespeichert werden. Alle Informationen in einer Datenbank werden in Dokumenten gespeichert. Selbst die Gestaltungselemente werden intern als Dokumente repräsentiert, welches das Kopieren und Einfügen der Gestaltungselemente ermöglicht. Dokumente haben unter anderem Eigenschaften wie die

- *Universal*-ID und
- *Parent*-UNID.

Die ***Universal*-ID** ist eine 32-stellige Hexadezimalzahl und identifiziert ein Dokument eindeutig über alle Repliken einer Datenbank. Bei wertgleicher *Universal*-ID sind zwei Dokumente Repliken voneinander. Die ***Parent*-UNID** enthält die Universal-ID von dem Dokument, zu dem das aktuelle Dokument eine Antwort darstellt.

Informationen werden in Feldern gespeichert, die in einem Dokument aggregiert werden. Die enthaltenen Felder ergeben sich in der Regel aus den Feld-Definitionen der zugehörigen Maske, können aber auch über die Anwendungslogik eingefügt werden. Felder haben als wichtigste Eigenschaften einen Namen, Datentyp und Wert. Der Name und Datentyp des Feldes wird durch die Feld-Definitionen oder über die Anwendungslogik bestimmt. Zum Beispiel wird die Information über die zugehörige Maske in dem Feld „*Form*“ gespeichert und kann jederzeit modifiziert werden. Die Modifikation der Information der zugehörigen Maske ermöglicht die Verwendung von verschiedenen Anzeigemasken für Dokumente.

Im relationalen Modell ist ein Datensatz gleichzusetzen mit einem Dokument. Die Struktur von Datensätzen kann aber nicht modifiziert werden, ohne daß Schemainformationen modifiziert werden. Im Gegensatz dazu kann die flexible Datenstruktur von Dokumenten beliebig verändert werden, so daß Dokumente, die mit der gleichen Maske erstellt wurden, eine unterschiedliche Struktur haben können. Der aus der flexiblen Struktur resultierende Nachteil ist, daß der Einsatz von Anfragesprachen wie SQL entfällt, da nicht von einer gesicherten Dokumentstruktur ausgegangen werden kann. Die flexible Datenstruktur hat aber auch Vorteile, da sie eine Modifizierung der Dokumente entsprechend wechselnder Anforderungen ermöglicht.

3.3.3 Ansichten

Ansichten sind ein wichtiger Bestandteil von Datenbanken. In einer Datenbank gespeicherte Dokumente werden in einer Ansicht tabellarisch angezeigt und können dort geöffnet werden.

[11] Einschränkungen einer Feld-Definition können über eine Eingabevalidierung definiert werden. Diese gelten aber nur für diese eine Feld-Definition.

Geöffnet bedeutet, daß sie mit ihrer zugeordneten Maske angezeigt werden. In Abbildung 3.4 wird der generelle Aufbau von Ansichten dargestellt.

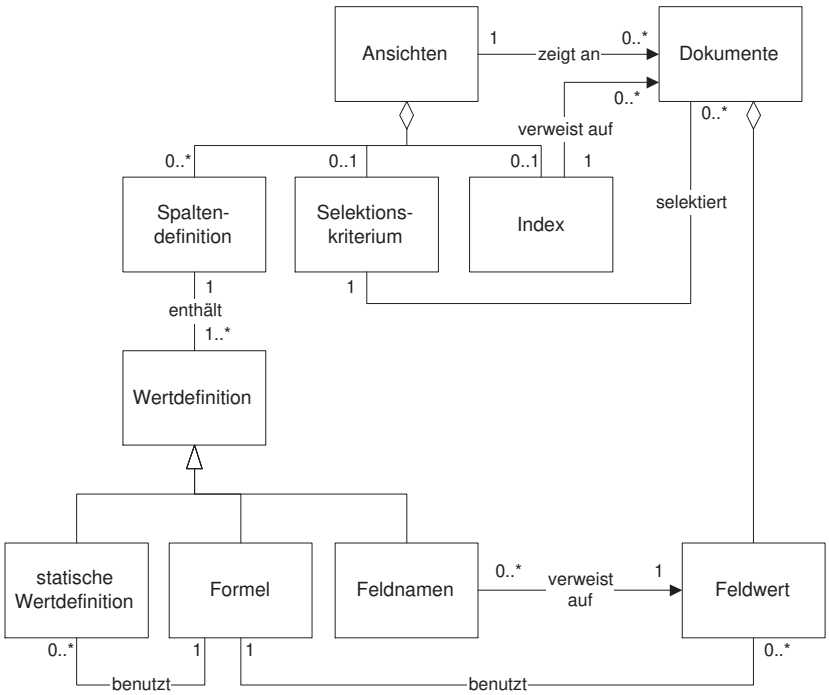

Abbildung 3.4: Ansichten

Ansichten aggregieren

- ein Selektionskriterium,
- Spaltendefinitionen und
- einen Index.

Das **Selektionskriterium** enthält eine Formel, die einen Wahrheitswert für jedes Dokument der Datenbank ergeben muß. Jedes Dokument für das die Auswertung des Selektionskriteriums „wahr“ ergibt, wird in der Ansicht angezeigt.

Spaltendefinitionen enthalten eine Wertdefinition, anhand derer der Spaltenwert ermittelt wird. Wertdefinitionen können

- statische Wertdefinitionen, die z.B. Texte oder Systemwerte enthalten,
- Feldnamen, die auf Feldwerte in Dokumenten verweisen oder
- Formeln, die einen Wert berechnen, sein.

Zusätzlich kann in den Eigenschaften angegeben werden, ob die Werte der Spalte sortiert und als Kategorie angezeigt werden sollen. Kategorien werden als zusätzliche Zeilen in der Ansicht dargestellt, die eine Klassifizierung der Dokumente ermöglichen.

Der **Index** der Ansicht enthält die Liste von Dokumenten, die in der Ansicht angezeigt werden. Die Art der Aktualisierung des Index wird in den Eigenschaften der Ansicht festgelegt. Die Aktualisierung des Index wird automatisch vom Laufzeitsystem von *Notes* anhand der Einstellungen vorgenommen.

Der Spezialfall einer Ansicht ohne Selektionskriterium wird Ordner genannt. Ordner werden als gesondertes Gestaltungselement einer Datenbank zur Verfügung gestellt. Das fehlende Selektionskriterium wird durch die Möglichkeit ersetzt, daß der Benutzer beliebige Dokumente in Ordner verschieben kann. Alle in den Ordner verschobenen Dokumente werden angezeigt.

Das in Grenzen vergleichbare Element im relationalen Modell ist eine Sicht in der relationalen Datenbanksprache SQL, die eine Auswahl von Relationenelementen anzeigt. Ein Vorteil der Sicht ist die Zusammenführung von Relationenelementen von verschiedenen Relationen zu einem Datensatz. Diese Möglichkeit besteht in einer Ansicht nicht. Dafür können verschiedene Dokumente gleichzeitig in einer Ansicht angezeigt werden, was in einer Sicht nicht möglich ist.

3.3.4 Navigatoren

Die Navigatoren repräsentieren grafische Hilfsmittel, die für die Navigation in Ansichten und Dokumenten einer Datenbank benutzt werden können. Der Navigator soll als Steuerzentrale in der Datenbank dienen. Ein Navigator aggregiert

- grafische Elemente und
- Schaltflächen.

Eine Eigenschaft der Schaltflächen und grafischen Elemente unter Exklusion des grafischen Hintergrunds ist, daß jedem Element eine Aktion zugeordnet werden kann, die bei einer Aktivierung ausgeführt wird.

Die **grafischen Elemente**, die im Navigator pixelgenau angeordnet werden können, umfassen Rechtecke, ovale Rechtecke, Polygone, Ellipsen, Text und einen grafischen Hintergrund.

Schaltflächen können in ihrer Form (Kreise, Rechtecke, Polygone und Grafiken) variieren.

3.3.5 Agenten

Der Begriff des Agenten in der Literatur unterscheidet sich von der Definition in *Notes*. Die in [Joh97; Weg98] benutzte Definition des Begriffes ist:

> „Ein Agent kann als eine einfache, heterogene, autonome und kommunizierende Software-Komponente definiert werden; viele dieser Aufgaben existieren gleichzeitig und arbeiten zusammen, um eine Aufgabe für einen Benutzer zu erfüllen [BW94].“

Eine Definition des Begriffes des Agenten, der sinngemäß [Cor97l] entnommen wurde, in *Notes* lautet:

> „Agenten sind eigenständige Programme, die in einer Datenbank gespeichert werden und eine spezielle Aufgabe in der Datenbank erfüllen. Agenten erlauben die flexibelste Art von Automation, da sie manuell oder automatisch im Hintergrund ausgeführt werden können und in keiner Weise von anderen Komponenten der Datenbank abhängen."

Der offensichtlichste Unterschied der Definitionen liegt in den grundlegenden Aspekten der Kooperation und Kommunikation von Agenten. *Notes*-Agenten sind nicht für Kommunikation und Kooperation untereinander entworfen worden, so daß hierfür keine Sprachmechanismen wie z.B. beim *JAFMAS*-System für *Java* [Cha97] existieren. Dieses kann aber mit vertretbaren Aufwand nachgebildet werden, so daß *Notes*-Agenten mittels *e-Mail* kommunizieren und kooperieren können. Ein weiterer Unterschied ist die Autonomie von Agenten, die für *Notes*-Agenten nicht zutrifft. *Notes*-Agenten agieren niemals selbstständig, sondern werden durch festgelegte Auslöser gestartet.

Notes-Agenten besitzen folgende Eigenschaften:

- den Ausführungszeitpunkt und Periodizität der Ausführung des Agenten,
- das Selektionskriterium für die Dokumente, die der Agent bearbeiten soll und
- ein Programm, das die Dokumente bearbeitet.

Der Eigenschaft, die den **Ausführungszeitpunkt und die Periodizität der Ausführung von Agenten** festlegt, kann ein Ausführungszeitpunkt aus mehreren Alternativen zugeordnet werden. Für periodische und asynchrone Ausführungszeitpunkte können zusätzlich Parameter für die Periodizität angegeben werden. Zwei Typen von Ausführungszeitpunkten werden unterschieden. Dieses sind die manuelle und synchrone Ausführung durch den Benutzer und die asynchrone Ausführung im Hintergrund durch das Server-Modul *AgentManager*. Ein Ausführungszeitpunkt, der erwähnt werden soll, ist „Wenn neue *e-Mail* ankommt". Dieser Ausführungszeitpunkt ermöglicht für angekommende *e-Mail* die periodische Ausführung eines Agenten durch den *AgentManager*. Unter Verwendung eines einheitlichen *e-Mail*-Formates könnte auf diesem Weg die Kommunikation und Kooperation zwischen Agenten realisiert werden.

Analog zur Ansicht enthält das **Selektionskriterium** eine Formel, die einen Wahrheitswert für jedes Dokument der Datenbank ergeben kann. Jedes Dokument, für das die Auswertung des Selektionskriteriums „wahr" ergibt, wird von dem Agenten bearbeitet.

Das **Programm**, das die Dokumente letztlich bearbeitet, wird im Agenten mit einer der verfügbaren Programmiersprachen realisiert.

3.3.6 Script-Bibliotheken

Script-Bibliotheken unterstützen die Wiederverwendung, zentrale Pflege und Verwaltung von *LotusScript*-Programmfragmenten[12]. Dafür können Script-Bibliotheken beliebige *LotusScript*-Programmfragmente enthalten. Script-Bibliotheken haben einen Namen, der sie eindeutig

[12] Die Programmiersprache *LotusScript* wird in Abschnitt 3.4.2 vorgestellt.

identifiziert. Unter Verwendung der Namen können andere Script-Bibliotheken importiert werden. Dieses ermöglicht eine Aufsplittung in mehrere Script-Bibliotheken. Der Import erfolgt über eine `include`-Anweisung, die eine Textersetzung durch die importierte Script-Bibliothek vornimmt.

Die Verwendung von Script-Bibliotheken ist in Abbildung 3.6 dargestellt. Alle Gestaltungselemente können Script-Bibliotheken importieren, womit die Wiederverwendbarkeit, zentrale Pflege und Verwaltung der enthaltenen *LotusScript*-Programmfragmente gewährleistet ist.

3.3.7 Sicherheitskonzept

Sicherheit ist ein wichtiger Aspekt, da *Notes* als dezentrales System mit Integrationsmöglichkeiten für das Internet zahlreiche potentielle Angriffspunkte bietet. Das Sicherheitskonzept von *Notes* bietet viele Funktionen, die benutzt werden können, um ein System zu sichern. Die Funktionen untergliedern sich in die Sicherheitbereiche für das Internet, Netzwerk, *Notes*-Client, HTTP-Client, *e-Mail*, Server, Datenbanken und Datenbankkomponenten.

In diesem Abschnitt werden nur die Sicherheitsbereiche erläutert, die die Anwendungsentwicklung betreffen, da alle anderen Bereiche in der Arbeit nicht verwendet wurden. Für alle anderen Bereiche wird auf die Literatur [Cor97b; FPJ98; SF98] verwiesen. Die Bereiche der Anwendungsentwicklung können in drei Sicherheitsebenen strukturiert werden:

1. Datenbankserver
2. Datenbanken
3. Datenbankkomponenten

In Abbildung 3.5 ist das Konzept der Sicherheit für die Anwendungsentwicklung als Diagramm abgebildet. In dem Diagramm ist die Datenbank *Namens- und Adreßbuch* (NAB) hervorgehoben, die die Basis des Sicherheitskonzeptes darstellt. Im NAB werden Dokumente für Benutzer, Gruppen und Datenbankserver des *Notes*-Systems gespeichert. Gruppen enthalten Namen von Benutzern, Datenbankservern oder Gruppen. Konkret können nur die Benutzer und Datenbankserver auf das *Notes*-System zugreifen, die im NAB gespeichert sind und die nötigen Zugriffsrechte besitzen. Die einzige Ausnahme stellen die Benutzer eines HTTP-Clients dar, für die ein anonymer Zugriff eingerichtet werden kann.

Auf der **Datenbankserver**-Ebene werden Zugriffslisten für den Datenbankserver erstellt. Die Zugriffslisten, die im Datenbankserver-Dokument definiert werden, enthalten Benutzer, Gruppen und Datenbankserver. Außerdem können Sicherheits-Einstellungen z.B. für die Ausführung von Agenten, Erzeugung von Datenbanken oder Repliken vorgenommen werden.

Die Ebene der **Datenbanken** verwendet als Basis für die Sicherheit die Zugriffskontrolliste. Dort werden sieben unterschiedliche Zugriffsebenen definiert, die verschiedene Möglichkeiten für die Arbeit mit der Datenbank bieten. Die Zugriffsebenen sind in Tabelle 3.1 aufgelistet und beschrieben. Darüber hinaus können Optionen für die einzelnen Zugriffsebenen aktiviert werden, die den Zugriff detaillierter spezifizieren.

Aggregiert werden von der Zugriffskontrolliste:

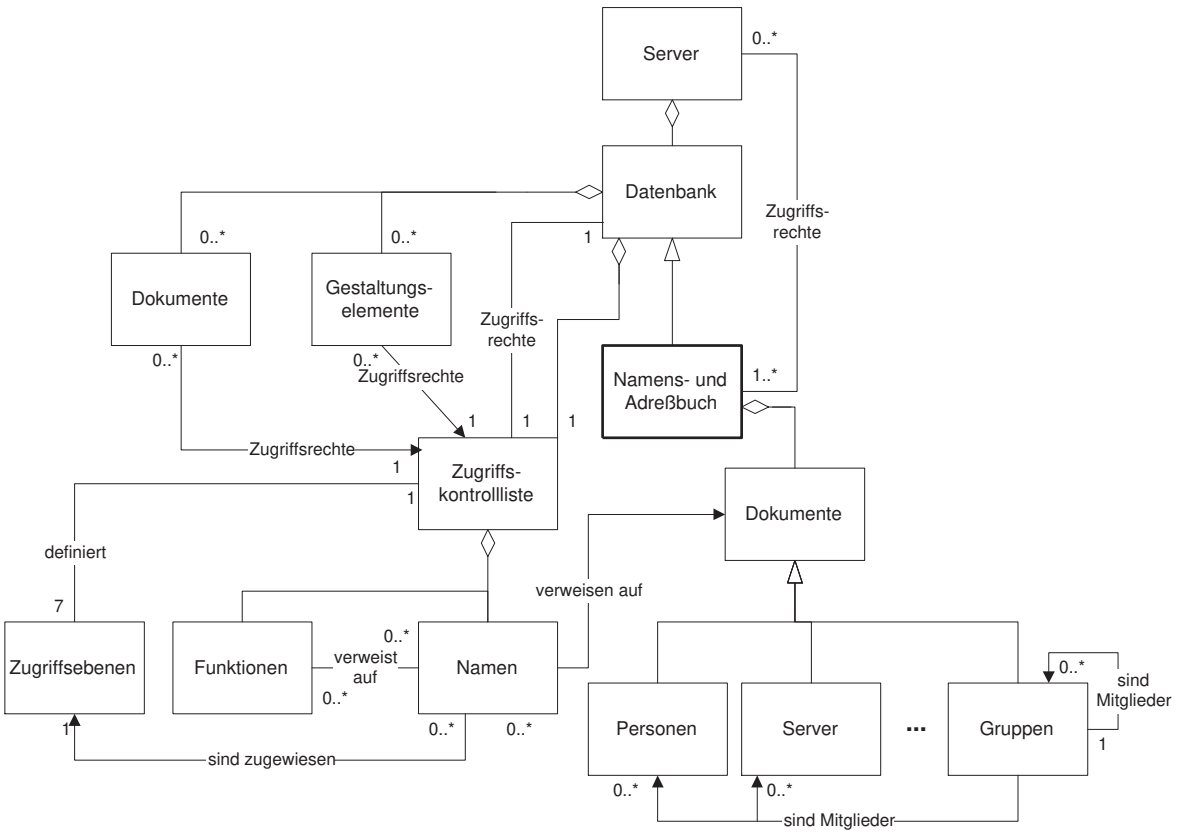

Abbildung 3.5: Sicherheitskonzept für die Anwendungsentwicklung

- Namen und
- Funktionen.

Die **Namen** bestimmen den Zugriff auf die Datenbank. Die Namen verweisen auf Dokumente der Benutzer, Gruppen oder Datenbankserver im NAB. Nur wenn die eingetragenen Namen im NAB vorhanden sind und der jeweilige Benutzer sich mit seinem Paßwort authentifiziert hat, wird der Zugriff entsprechend der zugewiesenen Zugriffsebene gewährt. In Gruppen eingetragene Benutzer werden über eine Gruppenauflösung ermittelt. Die Gruppenauflösung ermittelt für den Benutzer, ob er in einer der eingetragenden Gruppen eingetragen ist[13].

Funktionen können erstellt und benannt werden. Ihnen können Teilmengen der eingetragenen Namen zugewiesen werden. Funktionen können auch als Rollen angesehen werden. Jeder zugewiesene Name agiert in der Rolle der Funktion. Der Vorteil der Funktionen ist, daß sie wie Namen in Zugriffslisten beliebiger Datenbankkomponenten der Datenbank eingetragen werden können. Im Gegensatz zu Namen müssen Funktionen bei einer Änderung nicht modifiziert werden, da sich nur die Zuordnung der Namen zu den Funktionen ändert.

[13]Die Gruppenauflösung ist komplex, da eine Gruppe beliebig viele Gruppen beinhalten kann, in denen der Benutzer Mitglied sein kann.

Zugriffsebene	Beschreibung
Kein Zugriff	Erlaubt keinen Zugriff auf die Datenbank.
Archivar	Erlaubt nur die Erstellung von Dokumenten.
Leser	Erlaubt das Lesen von Dokumenten.
Autor	Erlaubt das Erstellen und das Lesen eigener Dokumente.
Editor	Erlaubt das Lesen und Schreiben auf Dokumente.
Entwickler	Erlaubt alle vorangegangenen Rechte und die Änderung der Gestaltung.
Manager	Erlaubt alle vorangegangenen Rechte und Änderungen der Zugriffskontrolliste, sowie Löschung der Datenbank.

Tabelle 3.1: Zugriffsebenen der Zugriffskontrolliste

Die letzte Ebene der **Datenbankkomponenten** betrifft die Gestaltungskomponenten und Dokumente. Für die Gestaltungskomponenten Masken und Ansichten können Zugriffslisten definiert werden, die den Zugriff auf die Komponenten einschränken. Zusätzlich kann in Masken die Verwendung der Feld-Definition „Abschnitt“ durch eine Zugriffsliste eingeschränkt werden. Die Zugriffsbeschränkungen für Dokumente werden über die Zugriffslisten in den Feldwerten des Datentyps Lesernamen und Autorennamen definiert.

3.3.8 Gegenüberstellung von *Notes* und dem relationalen Modell

Dieser Abschnitt faßt die Vergleiche des relationalen Modells und *Notes* in Tabelle 3.2 zusammen. Die detaillierteren Erläuterungen können den jeweiligen Abschnitten entnommen werden, die in der dritten Spalte angegeben sind.

Notes-Konzept	Relationales Modell	Abschnitt
Datenbank	Datenbankschema	3.3
Maske	Schemadefinition einer Relation	3.3.1
Feld-Definition	Schemadefinition eines Relationenelements	3.3.1
Ansicht	Definition einer Sicht in SQL	3.3.3
Dokument	Datensatz	3.3.2

Tabelle 3.2: Vergleich von *Notes* und dem relationalen Modell

3.4 Programmierung

Die Programmierung in *Notes* kann auf vielfältige Weise geschehen. Die integrierte Entwicklungsumgebung der *Notes Designer*[14] ermöglicht eine schnelle Entwicklung von Prototypen [15]. Unterstützt wird die Programmierung durch die Basisdienste der Laufzeitumgebung von *Notes* wie Speicherverwaltung, Ereignissteuerung und das Arbeiten mit den Komponenten von

[14] Der *Notes Designer* ist ein *Rapid Application Development*-Werkzeug.

[15] Informationen zum Thema „Prototyping“ können der Literatur wie [BKKZ92; BKKZ92] entnommen werden.

Notes. Die Basisdienste stellen ihre Dienste transparent zur Verfügung, schränken aber die Möglichkeiten eigener Entwicklungen ein, da ihre Verwendung zwingend ist [Cor97l; Tea97; FPJ98]. In Abbildung 3.6 sind die Konzepte der Programmierung dargestellt.

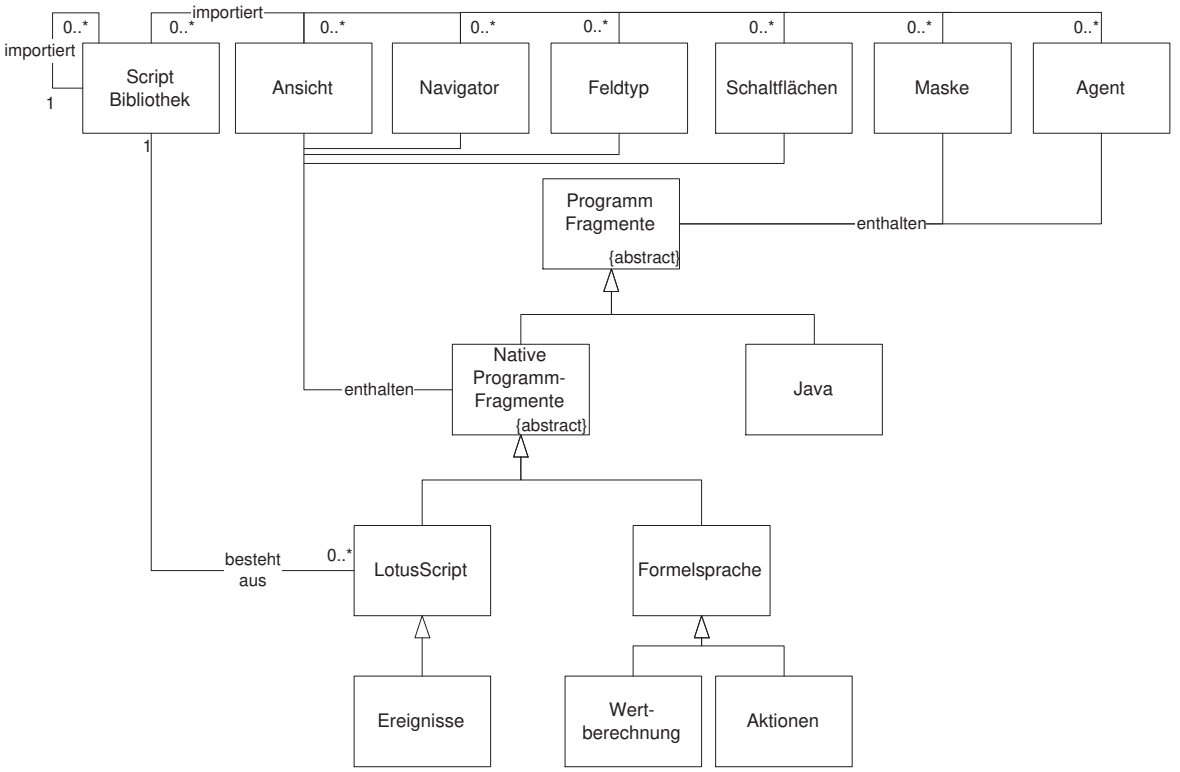

Abbildung 3.6: Programmierung in *Notes*

Programmfragmente stellen ausführbare Teilprogramme einer Anwendung dar, die eine bestimmte Tätigkeit im Kontext der Anwendung ausführen. Sie können spezialisiert werden in

- native und
- *Java*-Programmfragmente.

Native Programmfragmente sind Teilprogramme, die mit den in *Notes* integrierten proprietären Programmiersprachen *LotusScript* und der *Formelsprache* erstellt wurden, wobei die *Formelsprache* keine Programmiersprache ist. Im Gegensatz dazu müssen **Java**-Programmfragmente außerhalb von *Notes* entwickelt werden, da eine Entwicklungsumgebung[16] für die Programmiersprache nicht integriert ist[17]. *Java*-Programmfragmente sind nicht proprietär, da *Java* eine standardisierte objektorientierte Programmiersprache ist. Nachteilig für die Programmiersprachen ist, daß sie nicht orthogonal verwendet werden können.

[16] z.B. das *Java Development Kit* von Sun

[17] In Version 5.0 von *Notes* ist eine Entwicklungsumgebung im *Notes Designer* für *Java* vorhanden.

Alternative Lösungswege für die Programmierung von Anwendungen werden durch verschiedene Möglichkeiten für die Programmierung ermöglicht. Die Gestaltungselemente können verschiedene Programmfragmente enthalten. Grundsätzlich können die Möglichkeiten für die Programmierung in drei Arten von Programmfragmenten unterschieden werden:

- Ereignisse
- Aktionen
- Wertberechnungen

Ereignisse betreffen die existierenden Komponenten: Datenbanken, Masken, Ansichten, Agenten, Schaltflächen, Navigatoren und Felder. Jede dieser Komponenten erzeugt und reagiert auf vordefinierte Ereignisse. So gibt es in jeder Komponente z.B. das Ereignis `Initialize`, das bei der Erzeugung ausgelöst wird. Für die Programmierung von Ereignissen kann nur *LotusScript* verwendet werden.

Aktionen werden ausgeführt, wenn Schaltflächen und Agenten aktiviert werden. Die Differenzierung von Aktionen und Ereignissen ist eigentlich nicht existent, da Aktionen auch Ereignisse sind. Der entscheidende Unterschied ist, daß für die Programmierung von Aktionen die *Formelsprache* verwendet werden kann. Der Anwendungsentwickler muß entscheiden, ob er eine Aktion mit der *Formelsprache* oder mittels eines Ereignisses mit *LotusScript* programmiert.

Wertberechnungen kommen an vielen Orten vor. Beispiele sind Selektionsformel, berechnete Feldformel, Eingabe-Übersetzungsformel, Eingabe-Validierungsformel, Spaltenformel, usw.. Sie haben als Ergebnis einen Wert, der dem erwarteten Datentyp entsprechen muß, so daß die Laufzeitumgebung von *Notes* damit arbeiten kann. Ein Beispiel ist die Eingabe-Validierungsformel, die als Datentyp einen Wahrheitswert erwartet. Wie bereits aus den Namen der verschiedenen Wertberechnungen ableitbar, kann an dieser Stelle nur die *Formelsprache* verwendet werden.

Einen Sonderfall stellt die Verwendung von *Java* dar [Cor97d]. *Java* kann in Agenten und in Masken benutzt werden. Die Verwendung von *Java* in Masken ist auf die Verwendung von *Java-Applets*[18] beschränkt.

Für die Wiederverwendung und Wartung von *LotusScript*-Programmfragmenten können Script-Bibliotheken benutzt werden. Sie verwalten zentral *LotusScript*-Programmfragmente, die aber erst durch den Import in ein Gestaltungselement verwendet werden können.

In den folgenden Abschnitten werden die proprietäre *Formelsprache* und die Programmiersprache *LotusScript* vorgestellt. Auf eine Erläuterung der Programmiersprache *Java* wird verzichtet, da diese in der Arbeit nicht verwendet wurde und hinreichend in der Literatur [LP97; Mor97] beschrieben wird.

[18] *Java-Applets* sind von den Möglichkeiten eingeschränkte Programme, die im HTTP- und *Notes*-Client ausgeführt werden können.

3.4.1 *Formelsprache*

Die *Formelsprache* ist keine Programmiersprache, sondern besteht aus Ausdrücken, die *Notes* interpretiert. Die Auswertung der Formeln, die streng sequentiell ohne Sprünge und Schleifen erfolgt, ergibt entweder einen Wert oder führt zu der Ausführung eines bestimmten Befehls. Das Fehlen der Schleifen und Sprünge impliziert, daß die *Formelsprache* algorithmisch nicht vollständig ist. Eine Formel besteht aus einen oder mehreren der folgenden Ausdrücke [Cor97l; Tea97; FPJ98]:

- Variablen
- Konstanten
- Operatoren
- Schlüsselwörter
- *@Funktionen*

Das folgende Beispiel einer Aktion illustriert die Verwendung der genannten Ausdrücke. In dem Beispiel wird dem Benutzer, der die Aktion auslöst, zuerst in einer Dialogbox eine Auswahl von Städten angeboten, aus der er eine selektiert. Diese wird dann in das Feld `Stadt` eingetragen. Abschließend wird das Dokument aktualisiert.

```
REM "Auswahl einer Stadt";
result := @Prompt([OKCANCELLIST];"Stadt";"Bitte geben Sie die Stadt an.";
"Hamburg";Städte);
@SetField("Stadt";result);
@Command([ViewRefreshFields])
```

Variablen können entweder Felder aus Dokumenten oder temporäre Variablen sein. Im Beispiel ist `result` eine temporäre Variable und `Städte` ein Feld, das eine Liste von Städten enthält. **Konstanten** sind entweder Text-, Zahlen- oder Zeitkonstanten wie im Beispiel `Hamburg`. **Operatoren** verknüpfen Ausdrücke, wobei es Zuweisungsoperatoren, modifizierende Operatoren, etc. gibt. Im Beispiel ist `:=` ein Zuweisungsoperator. Für spezielle Funktionen enthält die *Formelsprache* **Schlüsselwörter** wie z.B. `REM`, der einen Kommentar erstellt.

Weiterhin werden **@Funktionen** zur Verfügung gestellt, mit denen z.B. Eingabe-/Ausgabe-, Steuerungs-, Listen- und mathematische Operationen durchgeführt werden können. Unterschieden werden die *@Funktionen* in *@Funktionen* und in *@Commands*. Der Unterschied besteht darin, daß *@Commands* die Ausführung von Funktionen des *Notes*-Client[19] ermöglichen, mit denen Arbeitsabläufe automatisiert werden können. Im Beispiel sind zwei *@Funktionen* abgebildet, nämlich `@Prompt` und `@SetField`, wovon die erste eine Dialogbox zur Selektion eines Wertes erzeugt und die zweite ein Feld im Dokument mit dem vorher selektierten Wert füllt. Abschließend ist im Beispiel ein *@Command* `@Command([ViewRefreshFields])` enthalten, das das Dokument aktualisiert. Die Aktualisierung hat zur Folge, daß Wertberechnungen (Siehe vorigen Abschnitt) in einem Dokument neu durchgeführt werden.

[19] Betrifft die Menüpunkte des *Notes*-Clients.

Auf den Einsatz der *Formelsprache* kann für die Programmierung nicht verzichtet werden, da Wertberechnungen nur mit der *Formelsprache* erstellt werden können. Außerdem können durch Kombinationen der Ausdrücke mächtige Formeln implementiert werden, die z.B. Operationen auf beliebigen Listen und komplexe mathematische Operationen erlauben. Ein weiterer Vorteil ist die einfache Handhabung von Feldern, die als Variablen zur Verfügung stehen und die oftmals effizientere Ausführung gegenüber gleichwertigen *LotusScript*-Programmen.

3.4.2 *LotusScript*

Die Programmiersprache *LotusScript* ist ein objektorientiertes *Basic*-Derivat, das aus dem Zusatzprodukt *Notes VIP* entstand und später in *Notes* integriert wurde. *LotusScript* ist eine statisch typisierte Sprache, die das objektorientierte und imperative Paradigma unterstützt[20]. Eine Beschränkung auf ein oder die Vermischung der beiden Paradigmen ist dem Programmierer freigestellt. Im weiteren wird kurz auf die Unterstützung des objektorientierten Paradigmas und auf Merkmale eingegangen, die nicht im normalen *Basic*-Umfang vorhanden sind. Ansonsten wird für die Beschreibung von *LotusScript* auf die Literatur verwiesen [Cor97l; Man98; Tea97; FPJ98].

Das objektorientierte Paradigma wird durch Klassen und Vererbung unterstützt. Klassen definieren Funktionen, Prozeduren und Variablen, die entweder öffentlich oder privat sind. Die Vererbung unterstützt die einfache Vererbung, so daß Klassen nur von einer Superklasse erben können. Weiterhin wird die dynamische Bindung für Methodenaufrufe unterstützt.

Für den Zugriff auf *Notes* wird eine vordefinierte Klassenbibliothek als Schnittstelle angeboten. Die Klassenbibliothek enthält Klassen für den Zugriff auf die Benutzerschnittstelle *(Frontend)* und die *Notes*-Ressourcen *(Backend)*. Eine Abbildung der Klassenbibliothek ist in Anhang E dargestellt. Am nachfolgenden Beispielprogramm wird die Verwendung der Klassenbibliothek kurz erklärt.

```
Sub Click(Source As Button)
     Dim workspace As New NotesUIWorkspace
     Dim uidoc As NotesUIDocument
     Dim doc As Notesdocument

     Set uidoc = workspace.CurrentDocument
     Set doc = uidoc.document
     Print doc.Form(0)
End Sub
```

Das Beispielprogramm erzeugt drei Objekte von Klassen der Klassenbibliothek. Zuerst wird das *Frontend*-Objekt `workspace` erzeugt, das den Arbeitsplatz repräsentiert und den Zugriff auf das *Frontend*-Objekt `uidoc` erlaubt, das das aktuell geöffnete Dokument repräsentiert. Abschließend wird das Dokument vom *Frontend*-Objekt `uidoc` dem *Backend*-Objekt `doc` zugewiesen und der Name der zugeordneten Maske des Dokuments ausgegeben.

Externe Applikationen können von *LotusScript* aus mit der DDE- und OLE-Funktionalität

[20] Weitere Informationen über die Paradigmen sind in [Wat90] zu finden.

gesteuert werden. Weiterhin können Funktionen aus einer DLL aufgerufen werden, wodurch die Verwendung von externen Applikationen ermöglicht wird. Zusätzlich können *LotusScript*-Extensionen (LSX) implementiert werden, die Anbindungen an externe Applikationen ermöglichen. Eine LSX stellt zusätzliche *LotusScript*-Klassen zur Verfügung, die in Programmfragmenten genutzt werden können.

Um die Integration der *Formelsprache* zu ermöglichen, enthält der Sprachschatz von *LotusScript* das Schlüsselwort `Evaluate`. Mit Hilfe dieses Schlüsselwortes kann ein begrenzter Teil der *Formelsprache* innerhalb von *LotusScript*-Programmen ausgeführt werden.

Vorteilhaft für die Implementierung von *LotusScript*-Programmen ist der integrierte *Debugger*, mit dem alle Ereignisse während der Programmausführung Schritt für Schritt verfolgt werden können. Wiederverwendung und zentrale Pflege von Programmen wird durch die Verwendung von Script-Bibliotheken unterstützt.

Nachteilig ist, daß die Klassenbibliothek keine äquivalenten Funktionen wie die *Formelsprache* zur Verfügung stellt, so daß diese häufig verwendeten Funktionen nachgebildet werden müssen. Ein weiterer Nachteil von *LotusScript*, der die Verwendung der *Formelsprache* notwendig macht, ist die fehlende Eingabe-Schnittstelle für Daten.

3.5 Integrationsmöglichkeiten für das Internet

Die Möglichkeiten zur Integration des Internets sind seit der Version 4.0 eine der wichtigsten Funktionen von *Notes*. Die Integration ermöglichte die Öffnung gegenüber den Benutzern des Internets. Seitdem wird versucht, die proprietären Techniken durch Internet-Standards zu ersetzen. Ein Beispiel dafür ist die Verwendung der Programmiersprache *Java*, die in der nächsten Version 5.0 vollständig integriert sein wird. Ein weiterer Vorteil der nächsten Version ist die Symbiose des *Notes*- und HTTP-Clients, so daß kein gesonderter HTTP-Client erforderlich ist.

Für die Integration wurden einige Erweiterungen vorgenommen. Die wichtigste Erweiterung, die die Grundlage der Integration darstellt, wurde für den Datenbankserver erstellt. In Abbildung 3.7 ist die Erweiterung dargestellt. Sie besteht aus zwei Server-Modulen:

HTTP-Server: Der HTTP-Server ermöglicht den Zugriff auf Dateien und Datenbanken des Datenbankservers. Die HTTP-Anforderungen einer Ressource werden überprüft und unterschieden, ob die Anforderung ein Objekt in einer Datenbank oder eine statische HTML-Datei im Dateisystem betrifft. Wenn die Anforderung ein Objekt einer Datenbank betrifft, wird die Anforderung an das *Domino-Modul* weitergeleitet. Falls die Anforderung eine statische HTML-Datei betrifft, wird diese sofort an den HTTP-Client weitergeleitet. Weiterhin wird das *Common Gateway Interface* (CGI) unterstützt.

Domino-Modul: Das *Domino-Modul* konvertiert automatisch alle HTTP-Anforderungen in eine *Notes* konforme Form, so daß der Datenbankserver die angeforderte Komponente der *Notes*-Datenbank ermitteln kann. Die Komponente wird dann automatisch vom *Domino-Modul* in das HTML-Format konvertiert und an den HTTP-Server übergeben, der die konvertierte Komponente an den HTTP-Client weiterleitet.

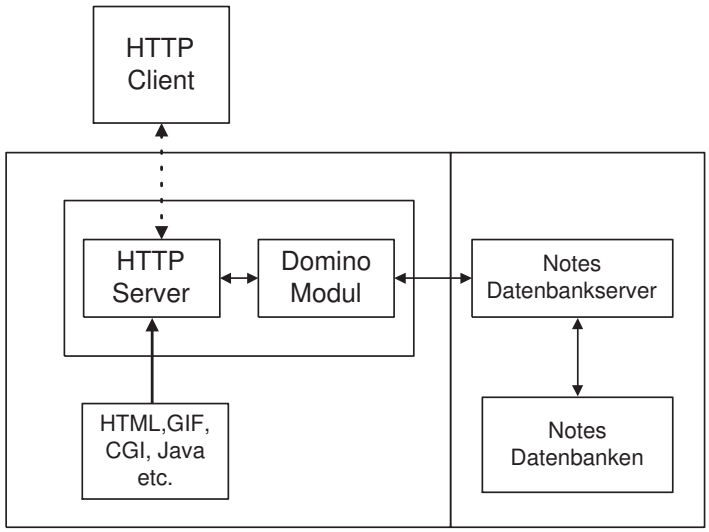

Abbildung 3.7: Erweiterung des Datenbankservers für die Integration des Internets

Die Erweiterung des Datenbankservers ist ausreichend, um Datenbanken im Internet zur Verfügung zu stellen, gewährt aber keinen Einfluß auf die Darstellung und veränderten Gegebenheiten im HTTP-Client. Um diesen Einfluß zu ermöglichen, wurden die Gestaltungselemente der Datenbanken erweitert. Andererseits existieren Einschränkungen für den Einsatz im Internet, die einen speziellen Entwurf von Anwendungen für das Internet erforderlich machen.

Ein gravierender Nachteil des HTTP-Client gegenüber dem klassischen *Notes*-Client ist die Zustandslosigkeit des HTTP-Protokolls, das z.B. Probleme bei dem Sicherheitskonzept verursacht. Für das Sicherheitskonzept wird eine zentrale Benutzerverwaltung in der Datenbank *Namens- und Adreßbuch* aufgebaut. Ein Benutzer, der über einen HTTP-Client auf geschützte Bereiche zugreift, muß sich mit seinem Benutzernamen und Paßwort authentifizieren. Die Information über den Benutzer wird dann in den CGI-Variablen gehalten.

Im weiteren werden in den nächsten Abschnitten die wichtigsten Erweiterungen und Einschränkungen für den Einsatz im Internet erläutert. Dafür wird in Abschnitt 3.5.1 die *Domino-URL-Syntax* vorgestellt, die im Internet die Referenzierung der Datenbank-Komponenten ermöglicht. Weiterhin werden in den Abschnitten 3.5.2 - 3.5.4 Erweiterungen und Einschränkungen der Datenbanken, Masken, Feld-Definitionen und Agenten beschrieben. Da die Erweiterungen und Einschränkungen sehr umfangreich sind, werden nur die Konzepte und Beispiele beschrieben und für weitergehende Informationen auf die Literatur verwiesen [SF98; CHT+97; FPJ98; Cor97f; Tea97].

3.5.1 *Domino-URL-Syntax*

Der Zugriff aus dem Internet auf Komponenten einer Datenbank geschieht über den *Uniform Ressource Locator* (URL) der Komponente. Eine Domino-URL-Referenz, die ihren Namen vom *Domino-Modul* hat, ist hierarchisch aufgebaut und folgt den syntaktischen Regeln der

Domino-URL-Syntax. In Abbildung 3.8 ist der hierarchische Aufbau zu erkennen und um Anweisungen ergänzt, die einem Datenbankserver in der URL übermittelt werden können.

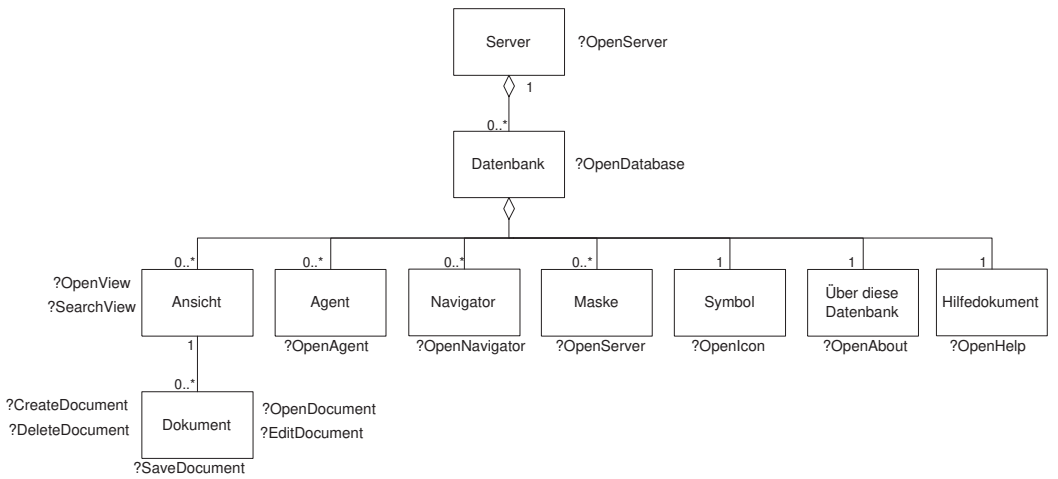

Abbildung 3.8: *Domino-URL-Syntax* und Anweisungen

Für die Bildung einer URL-Referenz werden die absolute und relative Form der Referenzierung unterschieden:

Absolute Referenzierung: http://Datenbankserver/Datenbank/Komponente/Komponente

Relative Referenzierung: /Datenbank/Komponente/Komponente

Die notwendige eindeutige Identifizierung der Komponenten kann mittels der *Universal*-ID, *Note*-ID, des Namens oder eines reservierten Namens erfolgen[21]. Ein reservierter Name identifiziert Komponenten, die im Rahmen der Datenbank eine Sonderstellung haben. Ein Beispiel dafür ist das „*AboutDatabase*“-Dokument einer Datenbank, das den Sinn und Zweck der Datenbank angibt.

Zusätzlich zu der URL-Referenzierung können einem Datenbankserver in der URL Anweisungen übermittelt werden. Anweisungen werden an die URL-Referenz mit einem „?“ angehängt[22]. Mehrere Anweisungen und Anweisungs-Parameter werden mit einem „&“ separiert.

Ein Beispiel für eine URL-Referenzierung kombiniert mit einer Anweisung, die eine Datenbank öffnet, ist:

```
http://www.gascorp.com/domino/dbwww/bconvers.nsf?OpenDatabase
```

[21] Univeral-ID, Note-ID und Namen wurden in Abschnitt 3.3 beschrieben.
[22] Entspricht dem „*QueryString*“ der CGI-Variablen.

3.5.2 Datenbanken

Die Erweiterungen der Datenbanken betreffen die Repräsentation und Verbindungsart mit einer Datenbank.

Die wichtigste Erweiterung für die Repräsentation ist die Eigenschaft „*JavaScript* verwenden". Sie bestimmt, ob *JavaScript* für die Generierung der HTML-Seiten benutzt wird. Dieses beinhaltet die Entscheidung, ob die Datenbank mit allen oder nur mit *JavaScript* fähigen HTTP-Clients benutzt werden kann. Der Vorteil ist die Möglichkeit der Verwendung von mehreren Schaltflächen in Masken, die Ausdrücke der *Formelsprache* enthalten können. Die Schaltflächen werden dann in den Dokumenten im Internet angezeigt. Ansonsten werden Schaltflächen ignoriert und automatisch eine *Submit*-Schaltfläche in die Dokumente eingefügt.

Um sichere Verbindungen mit einer Datenbank zu ermöglichen, kann als notwendige Verbindungsart eine *Secure Socket Layer*-Verbindung eingestellt werden.

3.5.3 Masken und Feld-Definitionen

Die Erweiterungen und Einschränkungen der Masken und Feld-Definitionen im Internet werden analog zu Abschnitt 3.3.1 gemeinsam erläutert. Dabei werden zuerst die Einschränkungen für den Einsatz im HTTP-Client und danach die Erweiterungen betrachtet. Darüber hinaus werden Ausprägungen von Masken und Feld-Definitionen mit spezieller Semantik für das Internet vorgestellt.

Die Einschränkungen sind genereller Natur und gelten für Masken und alle aggregierten Komponenten. *LotusScript*-Programmfragmente können im HTTP-Client nicht benutzt werden, da sie nicht konvertiert werden. So haben Ereignisse und Schaltflächen mit *LotusScript*-Programmfragmenten keine Funktion im HTTP-Client. Weitere Einschränkungen ergeben sich, wenn kein *JavaScript* für die Generierung benutzt wird[23]. Außerdem werden einige Komponenten nicht äquivalent zum *Notes*-Client repräsentiert oder können nicht verwendet werden.

Die wichtigsten Erweiterungen für die Masken, die im HTTP-Client durch HTML-Formulare repräsentiert werden und Feld-Definitionen, die im HTTP-Client durch HTML-*Input* Felder repräsentiert werden, sind folgende:

Für Masken können zwei Agenten spezifiziert werden, die entweder beim Öffnen oder Speichern von der Maske zugeordneten Dokumenten aktiviert werden:

- *WebQueryOpen*-Agent: Bestimmt den Agenten, der ausgeführt werden soll, bevor das Dokument an den HTTP-Client geschickt wird.
- *WebQuerySave*-Agent: Bestimmt den Agenten, der ausgeführt werden soll, bevor das Dokument aus dem HTTP-Client in der Datenbank gespeichert wird.

Erwähnenswert für die Feld-Definitionen ist die Möglichkeit, mit Sichtbarkeitsregeln einzustellen, unter welchen Bedingungen Felder im HTTP-Client angezeigt werden sollen.

[23] Siehe vorigen Abschnitt 3.5.2

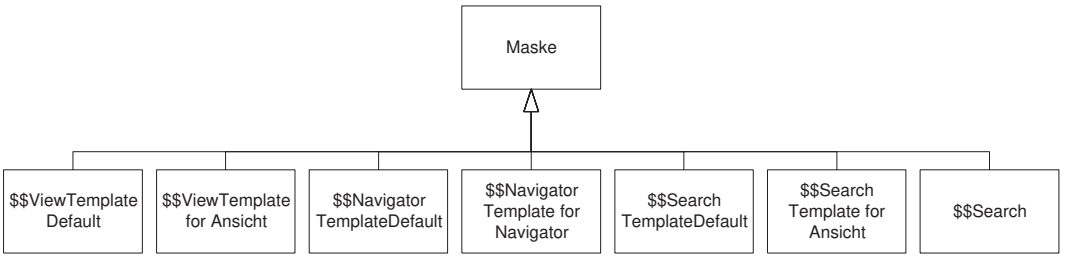

Abbildung 3.9: Ausprägungen von Masken mit spezieller Semantik für das Internet

In den Abbildungen 3.9 und 3.10 sind die speziellen Ausprägungen der Masken und Feld-Definitionen dargestellt. Die Ausprägungen sind Masken und Feld-Definitionen, die anhand ihrer Namen identifiziert werden. Ein Beispiel dafür ist die Maske „*$$ViewTemplateDefault*“. Wenn diese Maske in einer Datenbank existiert, wird eine Ansicht der Datenbank bei einer Anforderung aus einem HTTP-Client mit dieser Maske angezeigt. Damit die Maske die Ansicht anzeigen kann, enthält sie die Feld-Definition „*$$ViewBody*“, in der eine Ansicht angezeigt wird.

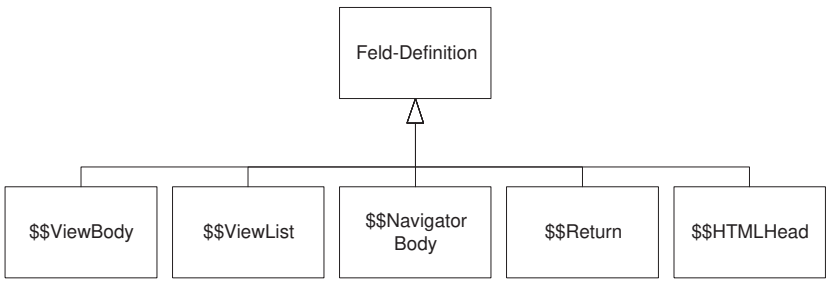

Abbildung 3.10: Ausprägungen von Feld-Definitionen mit spezieller Semantik für das Internet

Beschreibungen der übrigen Masken und Feld-Definitionen mit spezieller Semantik für das Internet können der Literatur entnommen werden [SF98; CHT+97; FPJ98; Cor97f; Tea97].

3.5.4 Agenten

Agenten besitzen Erweiterungen, die sie für den Einsatz im Internet sehr interessant machen. Zuerst sollen aber wieder die Einschränkungen betrachtet werden. Generell können nur Agenten mit manueller Startart, die synchron abgearbeitet werden, aus einem HTTP-Client gestartet werden. Dieses stellt aber keine große Einschränkung dar, da die periodischen, also asynchronen Agenten automatisch gestartet werden.

Die Erweiterungen betreffen eine Eigenschaft und die Programmierung der Agenten. Die Eigenschaft „Agent als *Web*-Benutzer ausführen“ bietet die Möglichkeit, dem Agenten während der Ausführung die Zugriffsrechte des Benutzers zuzuordnen. Der Agent kann keine Aktionen ausführen, die die Zugriffsrechte des Benutzers übersteigen. Der Punkt ist von Relevanz, da Agenten ansonsten durch den Datenbankserver ausgeführt werden, der im Regelfall alle

Zugriffsrechte hat.

Für die Unterstützung der Programmierung von Agenten existiert eine Erweiterung für das Kontextdokument[24] eines Agenten in der *Notes*-Klassenbibliothek von *LotusScript*. Das Kontextdokument repräsentiert das Dokument, aus dem der Agent gestartet wurde. Zusätzlich zu dem Dokument enthält das Kontextdokument aus dem HTTP-Client alle CGI-Variablen, die *Notes* unterstützt. Mittels der CGI-Variablen können z.B. Parameter im „*QueryString*" an den Agenten übergeben werden.

Als zusätzliche Erweiterung werden alle Ausgaben eines Agenten, der aus einem HTTP-Client gestartet wurde, direkt in den HTTP-Client geschrieben. Mit Hilfe dieser Eigenschaft können URLs oder HTML-Befehle einfach ausgegeben werden.

3.6 Integration des *SAP R/3*-Systems

Die Integration des *SAP R/3*-Systems basiert auf einer *LotusScript*-Extension (LSX). Mit Hilfe der Schnittstelle „*LotusScript* Extension für *SAP R/3*", die von der *SAP* kostenlos erhältlich ist, kann die Funktionalität des *SAP R/3*-Systems genutzt werden.

Im folgenden wird im Abschnitt 3.6.1 auf die zugrundeliegende Technik der LSX eingegangen. Auf dieser Basis wird in Abschnitt 3.6.2 die konkrete LSX für das *SAP R/3*-System erläutert.

3.6.1 *LotusScript*-Extension

LotusScript-Extensionen sind dynamische Bibliotheken von *LotusScript*-Klassen, die in der Programmiersprache C++ geschrieben sind und zur Anbindung von externen Anwendungen dienen. Die Erstellung einer LSX erfolgt mit Hilfe eines Entwicklungswerkzeuges[25]. Für eine LSX wird eine *Dynamic Link Library* (DLL) erzeugt, die in *LotusScript*-Programmfragmente importiert werden kann. Die zusätzlichen LSX-Klassen können wie jede andere *LotusScript*-Klasse genutzt werden [Cor97h; CCK+97].

In Abbildung 3.12 ist die Erweiterung der vorhandenen Fähigkeiten dargestellt. Ohne eine LSX stehen die Klassen der *Notes* Klassenbibliothek und die selbst entwickelten *LotusScript*-Klassen zur Verfügung, welche die *Notes*-API und *LotusScript*-API benutzen[26]. Mit einer LSX können zusätzlich die LSX-Klassen genutzt werden, die die Betriebssystem-API, API's externer Applikationen und die *LotusScript*-API benutzen. Über die LSX-Klassen kann auf die in ihnen gekapselten externen Applikationen oder auf das Betriebssystem zugegriffen werden, die in Form von *LotusScript*-Klassen angeboten werden.

3.6.2 *LotusScript*-Extension für das *SAP R/3*-System

Die *LotusScript*-Extension für das *SAP R/3*-System stellt auf der Basis des vorangegangenen Abschnitts LSX-Klassen zur Verfügung. Abbildung 3.12 stellt die verfügbaren LSX-

[24] Das Kontextdokument ist eine Eigenschaft der Klasse `NotesSession`
[25] Das Entwicklungswerkzeug ist eine spezielle *Notes*-Datenbank.
[26] Weitere Informationen über die API von *Notes* sind in [Cor97k; Cor98a; Cor98b; Man98] zu finden.

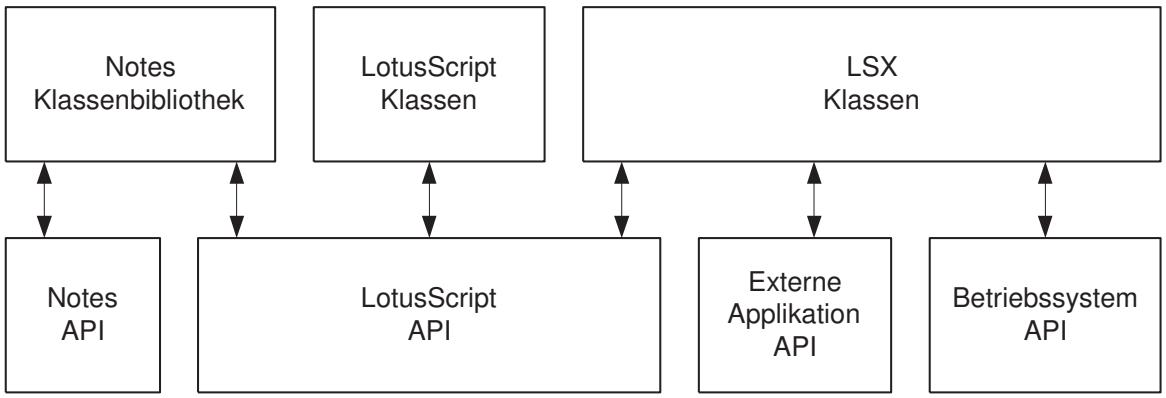

Abbildung 3.11: Erweiterte *LotusScript*-Fähigkeiten durch Einführung der LSX

Klassen dar [Cor97g; CCK+97; Cor97m]. Die zentrale Klasse `RFCServer` ermöglicht es eine TCP/IP-Verbindung zu einem *SAP R/3*-System aufzubauen. Dafür enthalten die Objekte der Klasse die Anmeldeinformationen für das konkrete System. Mit den Objekten der Klassen `RFCFunction` und `RFCTransaction` kann jede Funktion[27] und Transaktion benutzt werden. Die Eigenschaften der Klassen sind dabei meistens Objekte der Klasse `RFCCollection`, die mit dem Typ des jeweiligen aggregierten Objekts parametrisiert werden müssen, um Objekte dieses Typs zu enthalten. Im folgenden wird ein Beispiel für die Nutzung der LSX beschrieben[28].

```
%---------------------------- A B S A T Z ----------------------------
Set objServer = New RFCServer

objServer.Destination = SAPLogon.destination(0)
objServer.HostName    = SAPLogon.hostname(0)
objServer.System      = SAPLogon.system(0)
objServer.Client      = SAPLogon.client(0)
objServer.User        = SAPLogon.user(0)
objServer.Language    = SAPLogon.language(0)
objServer.Password    = SAPLogon.password(0)

Return = objServer.Logon()

Set BAPI  = New RFCFunction (objServer, "BapiCustomerOrderGetStatus")
BAPI.Exports("SALESDOCUMENT").Value = ordernmbr
Set Table = BAPI.Tables("STATUSINFO")
intresult = BAPI.Call()

Call objServer.Logoff()
```

[27] Die Funktion muß ein *Remote Function Call* sein.
[28] Die Fehlerbehandlung wurde weggelassen, um das Programm übersichtlich zu halten.

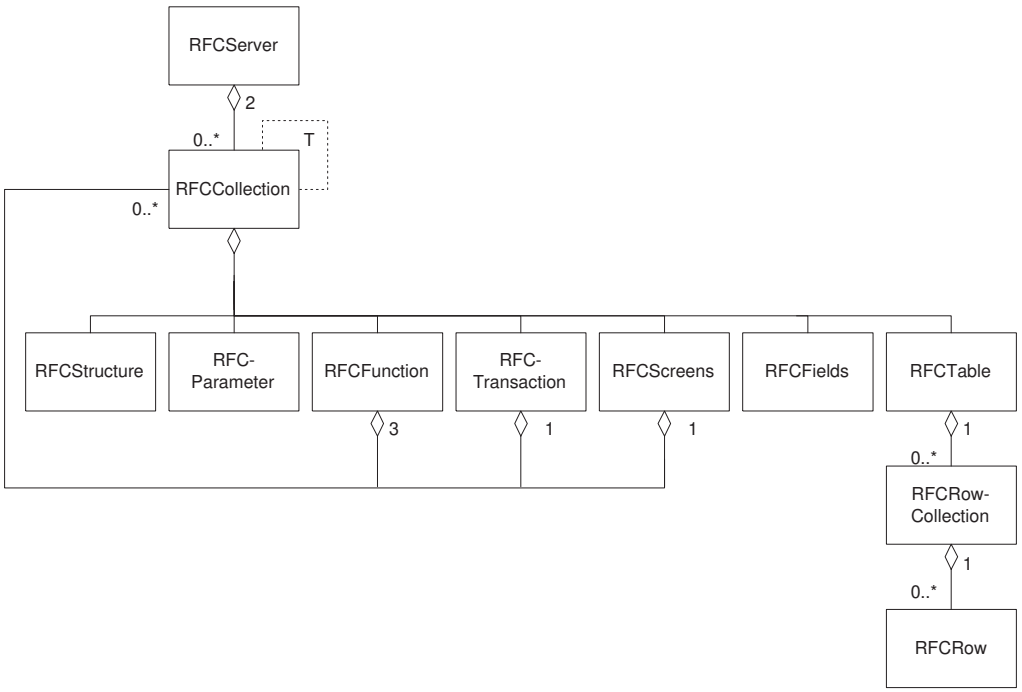

Abbildung 3.12: LSX-Klassen für das System *SAP R/3*

Im Beispiel wird ein Objekt der Klasse `RFCServer` erzeugt, das die Anmeldeinformationen des Objekts `SAPLogon` zugewiesen bekommt. Die Verbindung zum *SAP R/3*-System wird mit dem Methodenaufruf `objServer.Logon` hergestellt. Das Objekt der `RFCFunction`, die eine Statusabfrage für eine Bestellung ausführen soll, wird mit dem Serverobjekt und dem Namen der Funktion `BapiCustomerOrderGetStatus` erzeugt. Danach wird dem `Export`-Parameter die Bestellnummer zugewiesen. Dem Objekt `Table` wird der `Table`-Parameter der `RFCFunction` zugewiesen, um einen einfachen Zugriff auf die in der Tabelle enthaltenen Rückgabewerte zu ermöglichen. Die Parameter der `RFCFunction` `Exports` und `Tables` sind Objekte der Klasse `RFCCollection`, wobei der `Export`-Parameter mit Objekten der Klasse `RFCStructure` oder `RFCParameter` und der `Tables`-Parameter mit Objekten der Klasse `RFCTable` gefüllt sein kann. Danach wird der konkrete Funktionsaufruf mit `BAPI.Call` vorgenommen und die Verbindung des Serverobjekts mit `objServer.Logoff` getrennt. Die folgende Auswertung der Rückgabewerte des `Table`-Parameters ist im Beispiel nicht dargestellt.

Kapitel 4

Systementwicklungsprozeß: Anforderungen und Entscheidungen

Nachdem in Kapitel 2 aus betriebswirtschaftlicher Sicht die Architektur und die Anforderungsdefinition des zu entwickelnden Systems geklärt worden sind, behandelt dieses Kapitel die wichtigsten Anforderungen technischer Natur und Entscheidungen, die für den Systementwicklungsprozeß getroffen wurden.

Eine wichtige Entscheidung betrifft die Wahl der Notation, die für die Systemrealisierung benutzt wird. Die Wahl fiel auf die *Unified Modelling Language*, die eine Notation für die objektorientierte Analyse, den Entwurf und die Realisierung darstellt.

Zusätzlich zu der Wahl von UML wird ein Vorgehensmodell für die Realisierung des Systems benötigt. Verwendet wird ein Vorgehensmodell für die Entwicklung von Internet-Informationssystemen, das angelehnt an die Darstellungen in [FS97; Rip98; Weg98] ist und in Abschnitt 4.2 vorgestellt wird. Ein wichtiger Aspekt des verwendeten Vorgehensmodells ist der bruchlose Übergang zwischen Analyse, Entwurf und Implementierung unter Verwendung des Modells der *Business Conversations*, das im folgenden kurz vorgestellt wird.

Aufgrund der nicht objektorientierten Implementierungsplattform *Notes* bestand die Notwendigkeit, Implementierungsmöglichkeiten für die im Laufe des Vorgehensmodells erstellten Dokumenttypen zu finden. Diese werden in Abschnitt 4.3 erläutert.

Weitere Anforderungen für den Systementwurf ergeben sich aus der Kommunikation zwischen dem Kunden und dem Online-Verkaufssystem. Das Online-Verkaufssystem als kooperatives Informationssystem kommuniziert mit dem Kunden über langandauernde Aktivitäten zur Erreichung ihrer gemeinsamen Ziele [Mat97b; Weg98]. Damit dieses möglich ist, wird das vorgegebene Modell der *Business Conversations* verwendet. Zusätzlich müssen noch die Anforderungen der Anforderungsdefinition wie Erweiterbarkeit, Integration externer Systeme, etc. ermöglicht werden, so daß der Entwurf des Modells der *Business Conversations* diese berücksichtigen muß. In Abschnitt 4.1 wird das Modell der *Business Conversations* kurz vorgestellt und in Abschnitt 4.4 die notwendigen Systemanforderungen an den Entwurf und die Implementierung erläutert, die die Grundlage für Kapitel 5 bilden.

4.1 Das Modell der *Business Conversations*

In diesem Abschnitt wird kurz auf das Modell der *Business Conversations* eingegangen, wobei für detalliertere Informationen zu dem Modell auf die Literatur [Mat97a; Mat97b; Joh97; Ric97; Rip98; Weg98] verwiesen wird. Im Anhang D sind die theoretischen Grundlagen des Modells zu finden, die aus [Weg98] übernommen wurden.

Das Modell der *Business Conversations* basiert auf dem Leitmotiv der Sprechakte zwischen Kunde und Dienstleister. Die Sprechakte beruhen auf dem Konzept der *Conversations for Actions* [Win87]. Für die Sprechakte wird angenommen, daß sie universell einsetzbar sind. Die Kommunikation zwischen dem Kunden und dem Dienstleister wird in vier Phasen unterteilt: Anfrage, Übereinkunft, Leistung und Rückmeldung, die in Abbildung 4.1 dargestellt sind. Abhängig von den Aktionen zwischen Kunde und Dienstleister kann auf einzelne Phasen verzichtet werden.

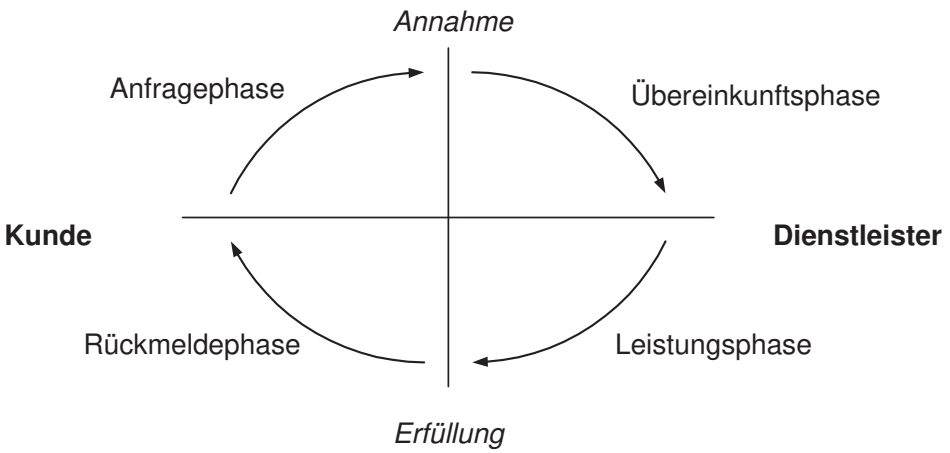

Abbildung 4.1: Die vier Interaktionsphasen

Die Ausführung von Sprechakten wird als Konversation bezeichnet. Dabei folgt die Ausführung der Sprechakte zwischen dem Kunden und Dienstleister festen Konversationsspezifikationen, die die Phasen des Kooperationsmodells ganz oder teilweise berücksichtigen. Kunde und Dienstleister müssen konforme Konversationsspezifikationen verwenden, damit sie miteinander kommunizieren können.

Die Konversationsspezifikationen bestehen aus Dialog- und Anfragespezifikationen. Die konkreten Konversationen werden nach folgendem Muster abgewickelt: Der Kunde erhält nach einer initialen Anfrage vom Dienstleister einen Dialog und eine Kollektion von möglichen Anfragen. Der Kunde modifiziert den Inhalt des Dialogs, wählt eine Anfrage aus und schickt ihn zurück. Der Dienstleister erhält den Dialog und die gewählte Anfrage, bearbeitet diese und schickt den nächsten Dialog zum Kunden. Dieser Vorgang wiederholt sich bis zum Ende der Konversation.

4.2 Vorgehensmodell

In diesem Abschnitt wird das Vorgehensmodell erläutert, das für die Realisierung des IPOS benutzt wurde. Angelehnt ist das Vorgehensmodell an die Darstellungen in [FS97; Rip98]. Das Vorgehensmodell wurde aber an die *Notes*-Terminologie angepaßt. In Abbildung 4.2 ist das Vorgehensmodell grafisch dargestellt.

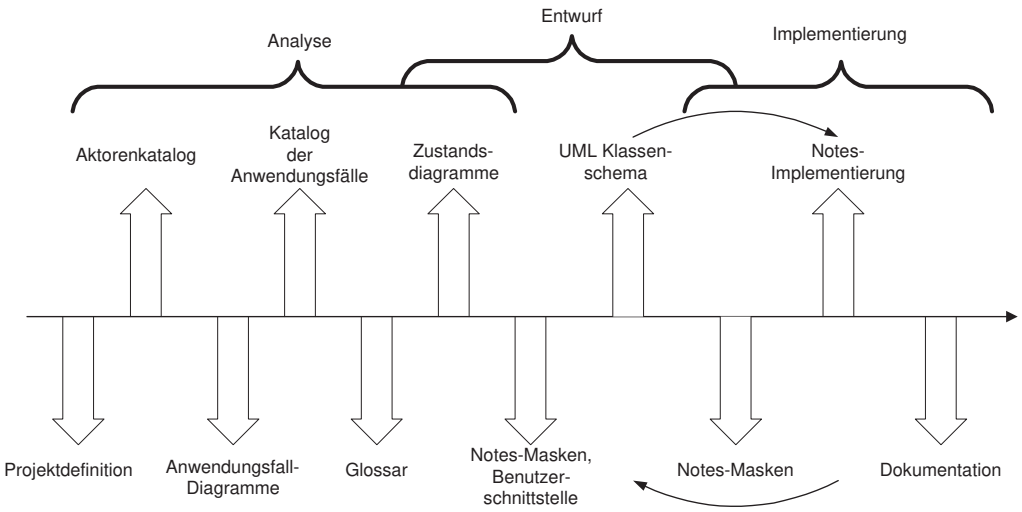

Abbildung 4.2: Vorgehensmodell zur Entwicklung eines Internet-Informationssystems

Der Entwicklungsprozeß des Vorgehensmodells untergliedert sich grob in drei Phasen:

Analyse: Innerhalb der Phase der Analyse werden die Projektdefinition, der Aktorenkatalog, der Anwendungsfall-Katalog und das Glossar erstellt. Mittels dieser Dokumente sollen die Anforderungen an das System geklärt und festgelegt werden. Dabei stellt die Projektdefinition eine kurze textuelle Beschreibung des Projekts dar. Der Aktorenkatalog enthält alle Personengruppen, die mit dem System interagieren. Der Anwendungsfall-Katalog enthält die Anwendungsfälle der Aktoren mit dem System. Das Glossar enthält Definitionen der verwendeten Terminologie und dient zum Verständnis der Dokumente.

Entwurf: Die Entwurfsphase beinhaltet die Erstellung der Benutzerschnittstelle und der zugehörigen Zustandsdiagrammen, die die dynamische Ablauflogik darstellen. Die Benutzerschnittstelle wird mit Hilfe von *Notes*-Masken erstellt, wovon jeweils eine *Notes*-Maske einen Zustand aus den Zustandsdiagrammen abbildet. Als zentrales Diagramm der Entwurfsphase wird ein Klassendiagramm aus den gefundenen Anwendungsfällen erstellt. Außerdem können zur genaueren Untersuchung und Erläuterung Interaktions- und Kollaborationsdiagramme erstellt werden, die auch die dynamischen Abläufe darstellen.

Implementierung: In der Implementierungsphase werden das Klassendiagramm und die

Zustandsdiagramme in *Notes* implementiert und eine Dokumentation dazu erstellt, wobei die in den früheren Phasen erstellten Dokumente bereits einen Teil der Dokumentation darstellen. Aus dem Klassendiagramm der Entwurfsphase entsteht ein Implementierungs-Klassendiagramm, das Einschränkungen bezüglich der Implementierungsplattform aufweisen kann. Die Zustandsdiagramme werden direkt in Konversationsspezifikationen implementiert und die in der Entwurfsphase erstellten HTML-Seiten als Benutzerschnittstelle weiterverwendet.

Ein wichtiger Aspekt des Vorgehensmodells sind die möglichen Rückkopplungen zwischen den Phasen, wodurch evolutionäre Systementwicklung ermöglicht wird. Die evolutionäre Systementwicklung ermöglicht die kontinuierliche Weiterentwicklung eines Prototypen bis zum Zielsystem. Notwendige Modifikationen, die sich im Laufe des Projekts herauskristallisieren, können auf diesem Wege leicht berücksichtigt werden.

Der bereits in der Einleitung des Kapitels angesprochene bruchlose Entwicklungsprozeß wird durch die Verwendung der Zustandsdiagramme und *Notes*-Masken erreicht [Rip98]. Die Zustandsdiagramme dienen als Konversationsspezifikationen und ermöglichen so den bruchlosen Übergang vom Entwurf zur Implementierung. Zusätzlich werden die in der Entwurfsphase erstellten *Notes*-Masken als Benutzerschnittstelle der Konversationen in der Implementierungsphase weiterverwendet. Die *Notes*-Masken können mit den in [Rip98] verwendeten HTML-Seiten verglichen werden, da *Notes*-Masken im Internet als HTML-Seiten dargestellt werden[1]. Die *Notes*-Masken können im Implementierungsprozeß problemlos weiterverwendet werden und haben so einen hohen Wiederverwendungsgrad.

4.3 Implementierung der Dokumente des Vorgehensmodells in *Notes*

In diesem Abschnitt wird die Implementierung der Dokumenttypen aus dem Vorgehensmodell in *Notes* beschrieben. Die Projektdefinition und das Glossar werden nicht behandelt, da diese textuelle Erläuterungen des Projekts sind. Interaktions- und Kollaborationsdiagramme werden nicht explizit beschrieben, da sie sich aus den Betrachtungen des Klassendiagramms implizit ergeben. Zusätzlich muß beim Zustandsdiagramm zwischen dem *Notes*- und HTTP-Client unterschieden werden, da die Modalitäten des Entwurfs verschieden sind.

Die Beschreibung der Implementierung wird für die Elemente der Diagramm- und Dokumenttypen vorgenommen, denen die in *Notes* gefundenen äquivalenten Konstrukte gegenübergestellt werden. Zusammengehörige Diagramm- und Dokumenttypen werden gemeinsam beschrieben, um Wiederholungen zu vermeiden. Abschließend faßt die Tabelle 4.2 die Ergebnisse zusammen.

Aktorenkatalog / Anwendungsfälle

Die Aktoren, die im Aktorenkatalog aufgelistet sind, werden in den Zugriffskontrollisten der Datenbanken implementiert. Für jeden Aktor wird eine Funktion angelegt, die den Zugriff des Aktors definiert. Die Personen und Gruppen, die zu diesem Aktor gehören, werden der

[1] Siehe 3.5

Funktion zugeordnet[2].

Die Anwendungsfälle der Aktoren sind die Szenarien, wie die Aktoren mit dem System interagieren. Mittels der Szenarien können die Zugriffsrechte der Aktoren auf die Datenbanken und ihre Komponenten festgelegt werden.

Zustandsdiagramm / *Notes*-Masken

Die Elemente des Zustandsdiagramms sind Zustände und Transitionen. Wie in Abschnitt 4.2 beschrieben, sind die *Notes*-Masken die Benutzeroberfläche der Zustände. Es besteht eine 1:1-Beziehung zwischen den *Notes*-Masken und Zuständen. In *Notes* wird ein Zustand durch eine Maske implementiert.

Für die Implementierung der Transitionen muß die Verwendung im HTTP-Client und *Notes*-Client unterschieden werden. Die Transitionen werden durch die Ablauflogik implementiert und diese müssen für den HTTP- und *Notes*-Client unterschiedlich realisiert werden. Eine Transition im HTTP-Client wird mit der Domino-URL-Syntax implementiert, mit der entweder der nächste Zustand direkt geöffnet oder ein Agent ausgeführt werden kann. Der Agent kann dann anhand einer Regel entscheiden, welcher Zustand als nächster geöffnet wird. Im *Notes*-Client hingegen muß die *Formelsprache* oder *LotusScript* benutzt werden, um den nächsten Dialog zu öffnen oder einen Agenten mit einer Regel aufzurufen[3].

Klassendiagramm

Klassendiagramme können aus den statischen Elementen Klasse, Objekt und Paket bestehen, die untereinander in Beziehungen stehen können. Im weiteren wird nur die Implementierung der Klassen betrachtet, da die Pakete gesondert beschrieben werden und Objekte als Instanzen von Klassen, die gleichen Eigenschaften und Beziehungen wie die Klassen aufweisen.

Die Implementierung der Klassen kann auf zwei verschiedene Arten geschehen, entweder als *Notes*-Komponente oder als *LotusScript*-Klasse. Notwendig für die Implementierung ist die Unterscheidung von drei verschiedenen Typen von Klassendiagrammen. Der erste Typ enthält die Datenbanken mit ihren Komponenten, die persistent sind. Der zweite Typ enthält *LotusScript*-Klassen, die nicht persistent sind. Im dritten Typ kommen Datenbanken mit ihren Komponenten und *LotusScript*-Klassen gleichermaßen vor.

Für den ersten Typ der Klassendiagramme, der nur *Notes*-Komponenten enthält, müssen Einschränkungen beachtet werden. Die Beziehungen von Gestaltungselementen untereinander sind durch ihre Definition fest vorgegeben. Dagegen können Dokumente mit beliebigen Dokumenten Beziehungen haben. `1:N`-Beziehungen zwischen Dokumenten werden direkt durch Haupt- und Antwortdokumente ermöglicht. `N:M`-Beziehungen können nur durch assoziativen Zugriff, also mit eindeutigen Bezeichnern in Feldwerten der Dokumente realisiert werden. Beziehungen zwischen Dokumenten und Gestaltungselementen sind durch die Gestaltungselemente vorgegeben, können aber durch Modifizierung der Dokumente geändert werden. Aggregation und Vererbung werden nicht direkt unterstützt. Dieser Klassentyp ist gut für die Erstellung von Klassendiagrammen für Datenstrukturen geeignet, da er nur persistente Komponenten enthält.

Der zweite Typ der Klassendiagramme, in denen nur *LotusScript*-Klassen enthalten sind, ist

[2]Siehe 3.3.7
[3]Siehe 3.5

problemlos, da *LotusScript* objektorientiert ist. Der einzige Nachteil ist, daß die Objekte von *LotusScript*-Klassen nicht persistent sind. Dieser Klassentyp ist unrelevant, da der Zugriff auf die Datenbanken und somit auf die Daten fehlt.

Der dritte Typ der Klassendiagramme besteht aus der Symbiose der beiden vorherigen Typen. Zusätzlich zu den *LotusScript*-Klassen stellt eine Klassenbibliothek die *Notes*-Komponenten als *Notes*-Klassen zur Verfügung[4]. Beziehungen zwischen *LotusScript*-Klassen und *Notes*-Klassen können nur einseitig realisiert werden. *LotusScript*-Klassen können nur Beziehungen zu *Notes*-Klassen haben. Für die Ablauflogik in *LotusScript*-Programmfragmenten ist dieser Typ wichtig, da er Zugriff auf die *Notes*-Ressourcen hat.

Nachteilig für die Modellierung ist, daß alle Objekte einer *Notes*-Komponente den gleichen Typ haben wie z.B. Dokumente den Klassentyp `NotesDocument` und Masken `NotesForm`. Um nun eine Unterscheidbarkeit z.B. der verschiedenen Dokumente zu erreichen, wird an dieser Stelle eine Namenskonvention getroffen, die eine Modellierung der Objekte einer Komponente, die einen speziellen Typ haben, als eigene Klasse erlaubt.

Für Objekte einer *Notes*-Komponente, die einen eigenen Klassentyp bilden, wird der Klassenname folgendermaßen gebildet. Der Klassenname setzt sich aus einem Präfix, der den speziellen Typ kennzeichnet und einem Suffix, der den Typ der Komponente kennzeichnet, zusammen. Der Präfix ist der Name der Komponente, mit Ausnahme der Dokumente, die keinen Namen haben, bei denen der Präfix der zugeordnete Maskenname ist. Beispiele hierfür sind Dokumente der Maske „Person“, die den Klassennamen `Person-Document` zugeordnet bekommen oder der Klassenname `Person-Form` für die Maske. Nachfolgend sind alle Suffixe in 4.1 aufgelistet.

Notes-Komponente	Suffix
Server	-Server
Datenbank	-Database
Maske	-Form
Ansicht	-View
Navigator	-Navigator
Agent	-Agent
Dokument	-Document

Tabelle 4.1: Suffixe

Paketdiagramm

Das Paketdiagrammm besteht aus Klassen und Paketen. Die Implementierung der Klassen wurde bereits erläutert. Das Element Paket wird in *Notes* durch eine Datenbank repräsentiert, da eine Datenbank ihre Komponenten und definierten *LotusScript*-Klassen kapselt. Das Paketdiagramm stellt daraus resultierend die Datenbanken und ihre Beziehungen untereinander dar.

[4]Die Klassenbibliothek ist in Anhang E dargestellt.

Dokumenttyp	Element	*Notes*-Implementierung
Aktorenkatalog/	Aktor	Funktion in der Zugriffskontrolliste
Anwendungsfälle	Anwendungsfälle	Zugriffsrechte auf Datenbank-Komponenten
Zustandsdiagramm/	Zustand/*Notes*-Maske	Maske
Notes-Masken	Transition	Internet: URL-Domino-Syntax
		Notes: Formelsprache oder *LotusScript*
Klassendiagramm	Klasse	*Notes*-Komponente / *LotusScript*-Klasse
	Assoziationen	Referenzen oder assoziativer Zugriff
Paketdiagramm	Paket	Datenbank

Tabelle 4.2: Implementierung der Dokumenttyp-Elemente in *Notes*

4.4 Systemanforderungen an die *Business Conversations*

In diesem Abschnitt werden die Systemanforderungen an den Entwurf und die Implementierung des Modells der *Business Conversations* erläutert, die berücksichtigt werden müssen, damit die Anforderungen an das Online-Verkaufssystem aus Kapitel 2 ermöglicht werden. Im folgenden werden die Anforderungen einzelnd erläutert und der Bezug zu den Anforderungen aus Kapitel 2 hergestellt:

Sitzungs-Management: Alle Konversationen, die die Kunden mit dem System führen, müssen protokolliert und gespeichert werden. Der Verlauf einer Konversation bis hin zum aktuellen Dialog wird als Spur bezeichnet [Hup98], in dem der Dienstleister das Online-Verkaufssystem Informationen über den Kunden erwerben kann. Diese Informationen ermöglichen die Statistik, die über die Spuren der Konversationen einen gläsernen Kunden schafft und für gezielte Maßnahmen wie Rabatte, Sonderangebote, etc. genutzt werden kann.

Administrierbarkeit: Die Administrierbarkeit der Konversationsspezifikationen muß ohne Programmierungskenntnisse möglich sein. Die Modifikation des Layouts und der Gestaltung der Dialoge und Anfragen muß dynamisch möglich sein. Analog dazu muß die Ablaufsteuerung, soweit es ohne Modifikation der Anwendungslogik möglich ist, dynamisch modifizierbar sein. Diese Punkte ermöglichen die Systemadministration und erlauben eine hohe Flexibilität des Online-Verkaufssystems.

Erweiterbarkeit und Modifikation der Anwendungslogik: Die Anwendungslogik muß über eine definierte Schnittstelle verfügen, mit der Erweiterungen und Modifizierungen des Online-Verkaufssystems möglich sind.

Integrationsmöglichkeiten: Die Integration externer Systeme muß möglich sein, die über eine Schnittstelle eingebunden werden.

Generische Kunden: Ein generischer Kunde für HTTP-Clients muß die Kommunikation mit dem System ermöglichen. Der generische Kunde muß dynamisch aus den Konversationsspezifikationen generiert werden und darf nur legale Interaktionen mit dem System erlauben. Die Benutzerschnittstelle für die Kunden des Online-Verkaufssystems wird über den generischen Kunden realisiert.

Kapitel 5

Realisierung der *Notes Business Conversations*

In diesem Kapitel wird der objektorientierte Entwurf und die Implementierung der *Notes Business Conversations* (NBC) in *Notes* beschrieben. Dieses stellte einen großen Teil der praktischen Arbeit dar. Die NBC implementieren das Modell der *Business Conversations*, auf dessen Grundlage die Kommunikation mit dem Kunden erfolgen soll. Weiterhin müssen die Anforderungen aus Abschnitt 4.4 berücksichtigt und erfüllt werden. Die theoretischen Grundlagen des Modells der *Business Conversations* sind in Anhang C enthalten.

Der Entwurf der NBC ist an die bereits implementierten *Tycoon Business Conversations* (TBC) angelehnt, die in [Ric97; Joh97; Weg98] beschrieben werden. Damit soll ein Vergleich zwischen den Implementierungen ermöglicht werden.

Das Kapitel gliedert sich in drei Abschnitte. In Abschnitt 5.1 wird die Architektur der NBC vorgestellt. Der Abschnitt 5.2 beschreibt den Entwurf und die Implementierung der NBC-Datenbank. Abschließend wird in Abschnitt 5.3 der generische Kunde vorgestellt.

5.1 Architektur

Die Architektur der NBC besteht aus vier Schichten, die in Abbildung 5.1 dargestellt sind.

Die Aufgabe der einzelnen Schichten ist, daß die jeweilige übergeordnete Schicht die Dienste der darunterliegenden Schicht transparent nutzen kann. Den Kommunikationspartnern der obersten Schicht wird mittels der Schichten eine virtuelle Verbindung zur Verfügung gestellt. Die Rollen des Kunden und Dienstleisters sind dabei fest vorgegeben. Der generische Kunde tritt als Kunde und die NBC-Datenbank in Verbindung mit der Anwendungslogik als Dienstleister auf.

Die notwendigen Anforderungen an die einzelnen Schichten werden nachfolgend beschrieben:

Kommunikationsinfrastruktur: Die Kommunikationsinfrastruktur, die durch das Netzwerk und den *Notes* Datenbankserver zur Verfügung gestellt wird, muß die Nutzung

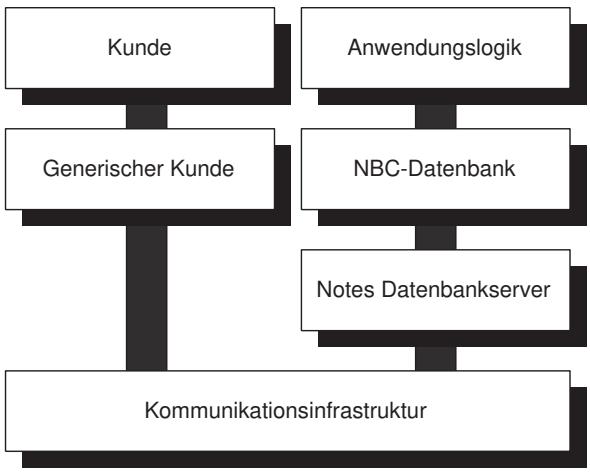

Abbildung 5.1: Architektur der *Notes Business Conversations*

von Netzwerkdiensten und niederen Kommunikationsprotokollen zur Verfügung stellen. Über diese können die Kommunikationspartner eine Verbindung herstellen. Die möglichen Protokolle sind bei der Verwendung eines HTTP-Clients für den Kunden auf das TCP/IP-Protokoll beschränkt.

Notes Datenbankserver: Der *Notes* Datenbankserver stellt Dienste zur Verfügung, um Nachrichten zwischen den Kommunikationspartnern auszutauschen[1]. Der Nachrichtenaustausch wird durch das *Domino-Modul* des Datenbankservers unterstützt. Das *Domino-Modul* fungiert als Konverter zwischen dem HTTP-Server und dem Datenbankserver. Alle Nachrichten werden in das jeweilige korrekte Format konvertiert und weitergeleitet[2].

NBC-Datenbank: Die NBC-Datenbank (NBC-DB) stellt die wichtigste Komponente der NBC dar und muß folgende Architekturmerkmale aufweisen, um die vielfältigen Anforderungen dieser Schicht zu erfüllen:

- Die NBC-DB muß das HTTP-Protokoll für die Kommunikation mit dem Kunden benutzen, das eine bindungsarme und synchrone Kommunikation erlaubt.
- Die Kommunikation mit dem generischen Kunden verläuft auf der Basis des Modells der *Business Conversations*, das eine anwendungsnahe Kooperation und Interaktion erlaubt. Dafür muß es eine Kommunikationsschnittstelle der NBC-DB zum Kunden geben.
- Die NBC-DB muß über eine Administrationsschnittstelle verfügen, die die Erstellung, Verwaltung, Auswertung und dynamische Modifizierung der Inhalte der NBC-DB erlaubt. Um dieses zu ermöglichen, muß die NBC-DB Informationen über die Konversationsspezifikationen und -instanzen enthalten und Operationen auf ihnen ausführen können.

[1] Vergleichbar mit dem Nachrichtensubsystem der TBC
[2] Siehe 3.5

- Die Anwendungslogik muß über eine Schnittstelle anbindbar sein, damit eine Parametrisierung der NBC-DB mit der Anwendungslogik ermöglicht wird.

Anwendungslogik: Die Entwicklung der Anwendungslogik, die erst eine sinnvolle Kommunikation mit der NBC-DB erlaubt, muß unterstützt werden. Dafür muß die Schnittstelle für die Anwendungslogik der NBC-DB einfach zu benutzen sein. Dieses soll durch zwei Punkte unterstützt werden:

- Durch die Verwendung eines *Frameworks* soll eine einfache Parametrisierung der NBC-DB ermöglicht werden.
- Die Definition von deterministischen Übergängen und inhaltlichen Überprüfungen der Konversationen soll bereits in den Konversationsspezifikationen möglich sein, damit unnötige Programmierung möglichst vermieden wird.

Generischer Kunde: Der generische Kunde soll im Sinne der NBC Konversationen mit dem Kunden führen. Der Kunde benutzt einen HTTP-Client für die Kommunikation gemäß der Anforderungen an das IPOS. Damit der Kunde sich entsprechend der Konversationsspezifikationen verhält, muß der generische Kunde automatisch anhand der Konversationsspezifikationen in der NBC-DB generiert werden. Damit wird eine korrekte Konversation garantiert, da der Kunde anhand der generierten Konversationsspezifikation des generischen Kunden entscheidet, was zu tun ist. Außerdem sollen Funktionen bereitgestellt werden, die die Konfiguration des Layouts der Konversationsspezifikationen ermöglichen, um Einfluß auf die Darstellung des generischen Kunden zu nehmen.

5.2 *Notes Business Conversations*-Datenbank

In diesem Abschnitt wird die NBC-DB beschrieben, die die Dienste für die Kommunikation anhand des Modells der *Business Conversations* zur Verfügung stellt. Zusätzlich werden durch die NBC-DB spezialisierte Funktionen für Internet-Informationssysteme wie das IPOS zur Verfügung gestellt, mit denen eine dynamische Konfiguration des Layouts des generischen Kunden möglich ist.

Zusätzlich zu den Anforderungen und Schnittstellen, die bereits in Abschnitt 5.1 und 4.4 erwähnt wurden, muß die NBC-DB folgende Punkte unterstützen:

- Die Konversationsspezifikationen und -instanzen sind in der NBC-DB persistent gespeichert, so daß ein einfacher Zugriff möglich ist.
- Die NBC-DB tritt im Sinne der *Business Conversations* immer in der Rolle des Dienstleisters auf.
- Parallele Ausführungen von mehreren Konversationen mit einem Kunden müssen möglich sein.
- Die Spezifizierung von Subkonversationen soll möglich sein, um austauschbare Spezifikationen zu ermöglichen.

Damit die Anforderungen an die NBC-DB erfüllt werden können, müssen die Komponenten der NBC-DB über eine Kommunikations-, Administrations- und Anwendungslogikschnittstelle verfügen. Eine weitere wichtige und notwendige Komponente sind die Konversationsspezifikationen und -instanzen, die von fast allen anderen Komponenten und Schnittstellen benutzt werden. In Abbildung 5.2 ist der Aufbau der NBC-DB mit seinen Komponenten und Schnittstellen dargestellt.

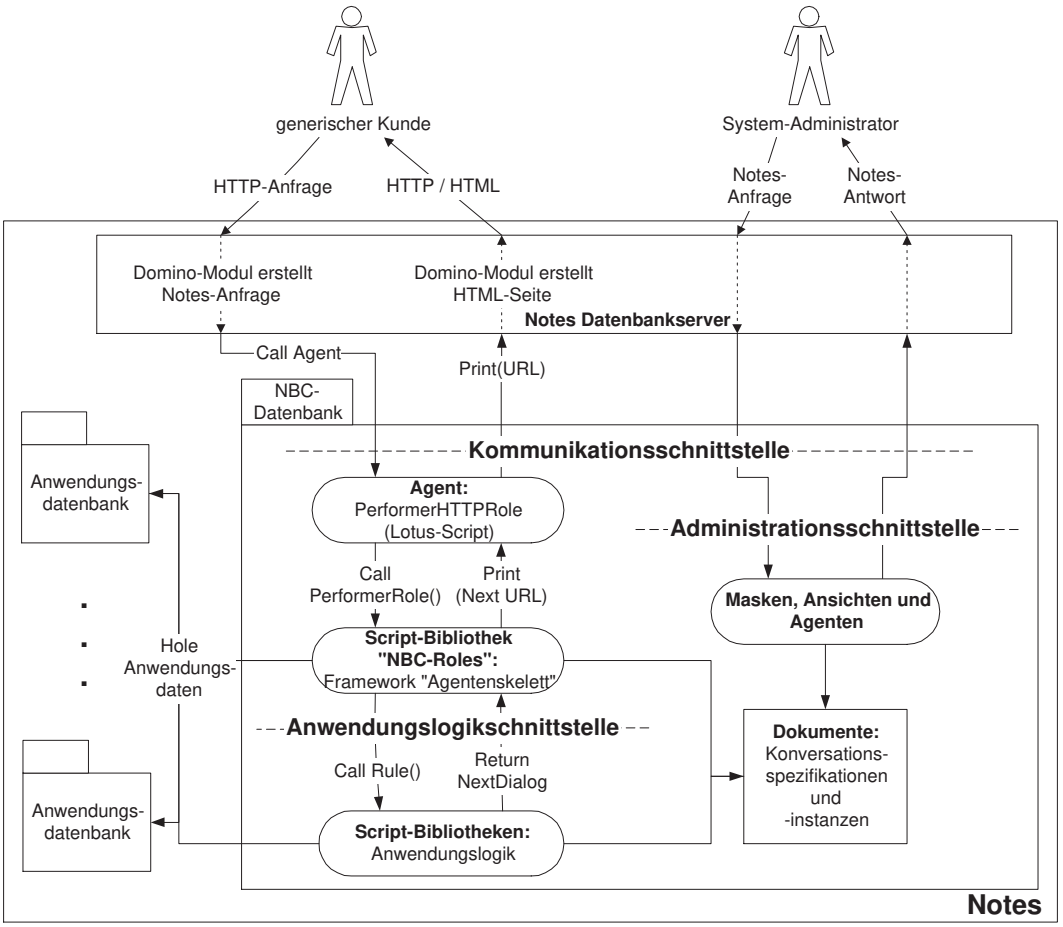

Abbildung 5.2: Aufbau der NBC-Datenbank

Im folgenden wird in Abschnitt 5.2.1 der Entwurf und die Implementierung der notwendigen Konversationsspezifikationen und -instanzen beschrieben. Die Vorstellung der Schnittstellen mit den zugehörigen Komponenten erfolgt in Abschnitt 5.2.2 für die Kommunikations-, in Abschnitt 5.2.3 für die Anwendungslogik- und in Abschnitt 5.2.4 für die Administrationschnittstelle.

5.2.1 Konversationsspezifikationen und -instanzen

Dieser Abschnitt beschäftigt sich mit dem Entwurf und der Implementierung der Konversationsspezifikationen und -instanzen. Dafür wird zuerst der Sinn und Zweck der Konversati-

onsspezifikationen und -instanzen beschrieben und danach einzelnd auf die Spezifikationen und Instanzen eingegangen. Abschließend werden dann Implementierungsentscheidungen beschrieben. Die Verwendung und Erzeugung der Konversationsspezifikationen und -instanzen wird im Rahmen der Beschreibungen der Schnittstellen und Komponenten der NBC-DB beschrieben. Gemäß der Betrachtungen aus Abschnitt 4.3 werden die Klassennamen in den verwendeten Klassendiagrammen gebildet.

Anhand der **Konversationsspezifikationen** werden die zur Verfügung stehenden Konversationen beschrieben. Diese Ebene realisiert das Typsystem für die Konversationsinstanzen. Konversationsspezifikationen müssen eine Laufzeitrepräsentation besitzen und Objekte erster Klasse sein. Dadurch werden Konformitätsprüfungen, generische Dienste, Metadienste und die dynamische Modifikation und Untersuchung der Spezifikationen ermöglicht.

Die **Konversationsinstanzen** stellen die konkreten Konversationen zwischen Kunde und Dienstleister dar. Anhand der zugehörigen Konversationspezifikationen werden die zur Spezifikation konformen Instanzen der Konversation erzeugt, die die Werte der Konversation darstellen.

Konversationsspezifikationen

Der Entwurf und die Implementierung der Konversationsspezifikationen spiegelt sich in dem Klassendiagramm in Abbildung 5.3 wieder. Die Konversationsspezifikationen sind entweder als persistente Dokumente oder Masken implementiert, die untereinander in Beziehungen stehen. Die Masken implementieren notwendigerweise die Spezifikationen der Dokumente, anhand derer die Dokumente oder Instanzen der Konversationsspezifikationen erzeugt werden[3]. Alle Klassen mit Ausnahme der Inhaltsspezifikationen (`ContentSpec-Form`) sind in der NBC-DB gespeichert.

Konversationsspezifikationen sind Dokumente der Klasse `ConversationSpec-Document`. Sie sind benannt und stehen mit beliebig vielen Dialogspezifikationen (`DialogSpec-Document`) in Beziehung. Eine davon ist die initiale Dialogspezifikation. Wiederum mehrere können finale Dialogspezifikationen sein. Dialogspezifikationen besitzen einen Namen und stehen mit beliebig vielen Anfragespezifikationen (`RequestSpec-Document`) und einer Inhaltsspezifikation (`ContentSpec-Form`) in Beziehung. Die `N:M`-Beziehung zwischen den Anfrage- und Dialogspezifikationen erlaubt die Wiederverwendung der Anfragen in beliebig vielen Dialogspezifikationen. Weiterhin stellt die *replies*-Beziehung die möglichen Folgedialoge der Anfragespezifikationen dar. Die Masken der Inhaltsspezifikationen, die in den Datenbanken der Anwendung gespeichert sind[4], sind unabhängig von der Dialogspezifikation. Die Typen der Felder in den Masken sind die Typen der Inhalte der Dialoge. Analog dazu sind die Spezifikationen der Konversations-, Dialog- und Anfragespezifikationen als Masken (`ConvSpec-Form`, `DialogSpec-Form` und `RequestSpec-Form`) implementiert.

Konversationsinstanzen

Ergänzend zu den Konversationsspezifikationen sind im Klassendiagramm in Abbildung 5.4 die Konversationsinstanzen dargestellt. Für die Implementierung der Konversationsspezifikationen wurden Dokumente, Masken und eine *LotusScript*-Klasse benutzt. Konkrete Konversationen sind Dokumente der Klasse `Conversation-Document`, die eine Identifikationsnummer

[3] Siehe Kapitel 3

[4] Das *Paket* in der Abbildung stellt eine beliebige Datenbank dar.

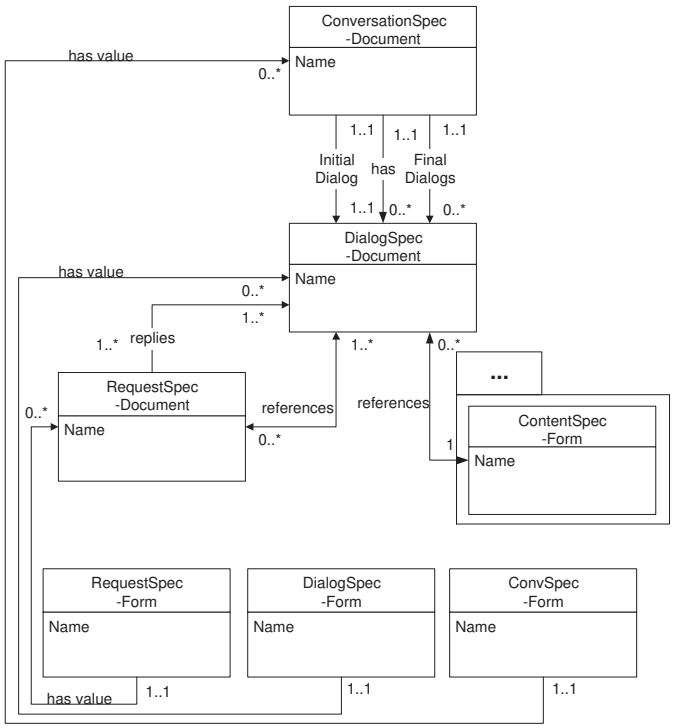

Abbildung 5.3: Klassendiagramm der Konversationsspezifikationen

(`ConversationID`) und einen Verweis auf den aktuellen Dialog besitzen. Außerdem kennt die Konversation alle Dialoge, die im Verlauf der Konversation verwendet wurden. Die Instanzen der *LotusScript*-Klasse `Dialog` kennen ihre zugehörige Dialogspezifikation und verweisen auf das Dokument des Inhalts (`Content-Document`). Die Instanzen der Dialoge sind nicht persistent und abhängig vom *Framework* „Agentenskelett“. Der Inhalt kennt die zugehörige Dialog- und Inhaltsspezifikation. Die Spezifikation der Konversationen (`Conversation-Form`) ist als Maske implementiert.

Die Beziehungen `Super Conversation` und `Abstract Dialog` ermöglichen die Verwendung von Subkonversationen. Die reflexive Beziehung `Super Conversation` ermöglicht den Verweis auf die Superkonversation, aus der die aktuelle Konversation aufgerufen wurde. Außerdem verweist die Beziehung `Abstract Dialog` auf die Dialogspezifikation der Superkonversation, die beim Erreichen des finalen Dialogs der Subkonversation als Übergang benutzt wird. Eine genauere Beschreibung der Subkonversationen findet sich in Abschnitt 5.2.5.

Implementierungsentscheidungen

Die Implementierungsentscheidungen betreffen Konversationsspezifikationen sowie -instanzen und werden nachfolgend beschrieben:

1. Die Beziehung zwischen den Spezifikationen der Dialoge und Anfragen wurde während der Implementierungsphase von einer `1:N` auf eine `N:M`-Beziehung geändert. Das hat den Vorteil, daß Anfragespezifikationen, die in mehreren Dialogspezifikationen verwendet werden, nur einmal erstellt werden müssen. Das System ist durch diese Änderung leichter

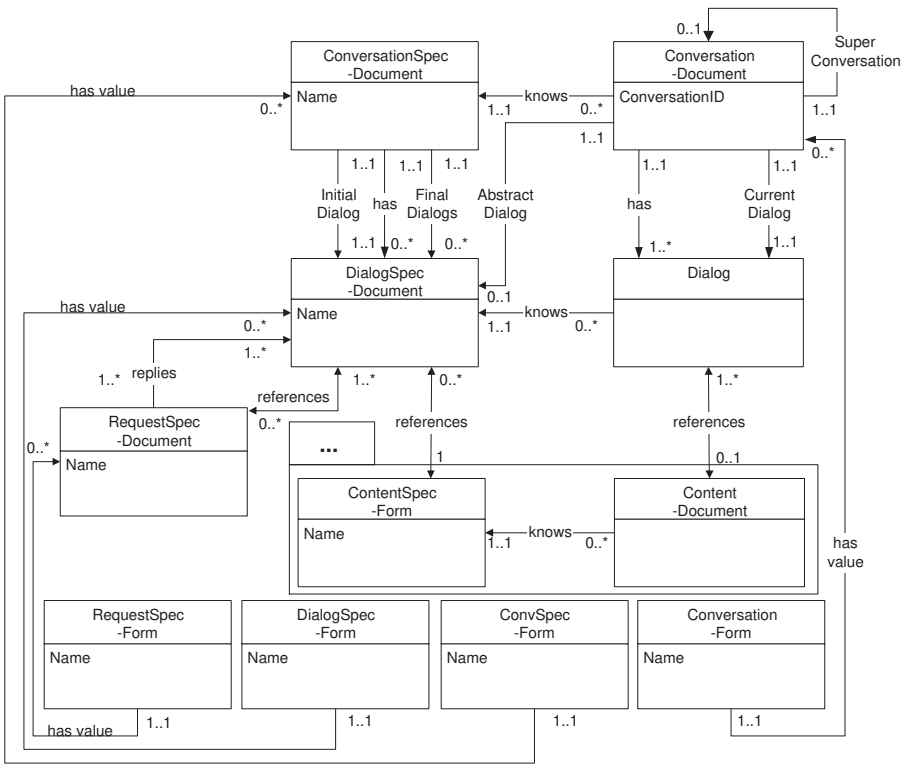

Abbildung 5.4: Klassendiagramm der Konversationsinstanzen

erweiterbar, pflegbar und administrierbar.

2. Die Implementierung der Inhaltsspezifikation als Maske wurde gewählt, da dieses zwei Vorteile hat: Erstens können beliebige vorhandene Masken benutzt werden und müssen nicht erstellt werden. Der Aufwand für die Realisierung des Systems wird dadurch geringer. Zweitens können beliebig viele Dokumente mit Hilfe der Masken erzeugt werden, die als konkrete Inhalte benutzt werden. Die Inhalte müssen somit nicht, während der Laufzeit generiert werden.

   Als Alternative für die Verwendung der Masken hätten die Dialogspezifikationen erweitert werden können. Das würde aber Nachteile mit sich bringen und alle konkreten Inhalte müßten generiert werden.

3. Die Entscheidung, den Dialog als *LotusScript*-Klasse und nicht als Dokument zu implementieren, soll hier näher erläutert werden. Da in einer Konversation mit dem generischen Kunden im HTTP-Client Dialoge, die Ansichten in *Notes* darstellen, nur eine temporäre Präsentation als Dokument haben, stellte es sich als notwendig heraus, eine Unterscheidung des Inhalts und des Dialogs vorzunehmen. Außerdem erlaubt die Unterscheidung den uneingeschränkten Einsatz der Inhalte außerhalb der NBC, da sie erst im Kontext der NBC zu Inhalten von Dialogen werden.

5.2.2 Kommunikationsschnittstelle

Die Kommunikationsschnittstelle dient dem generischen Kunden und dem Systemadministrator zur Kommunikation mit der NBC-DB. Dabei werden zwei verschiedene Arten der Kommunikation verwendet:

1. Der Systemadministrator greift über den *Notes*-Client auf die NBC-DB zu. So kann er alle Funktionen von *Notes* ausnutzen. Über den *Notes*-Client wird ihm der Zugriff auf die Masken, Ansichten und Agenten der NBC-DB gewährt, die er im jeweiligen Kontext nutzen kann.

2. Der generische Kunde kommuniziert mit der NBC-DB über das HTTP-Protokoll. Er stellt die HTTP-Anfragen, die vom menschlichen Kunden ausgewählt wurden, immer an den Agent `PerformerHTTPRole`, der als Antwort die URL des nächsten Dialogs zurückgibt. Ein Agent wird für die Kommunikation benutzt, da nur Agenten die Möglichkeit bieten, daß in *LotusScript* erstellte *Framework* `Agentenskelett` zu verwenden. Weitere notwendige Eigenschaften der Agenten sind:

   - Für jede Anfrage wird ein neuer Agenten-Thread gestartet, so daß einer parallelen Bearbeitung von mehreren Konversationen nichts im Wege steht.
   - Parameter können mittels CGI-Variablen oder dem Kontextdokument übergeben werden[5].
   - Die Programmiersprache *LotusScript* kann benutzt werden, die ansonsten in keiner anderen Komponente einer Datenbank für eine Anwendung in einem HTTP-Client verwendet werden kann.
   - Als Ergebnis des Programms des Agenten kann eine URL ausgegeben werden, die auf den nächsten Dialog verweist.

5.2.3 Das Agentenskelett und die Anwendungslogikschnittstelle

In diesem Abschnitt wird der Entwurf, die Implementierung und die Verwendung des *Frameworks* `Agentenskelett` beschrieben. Außerdem wird auf die Schnittstelle des `Agentenskeletts` zur Anwendungslogik eingegangen. Der Entwurf orientiert sich an dem der TBC-Agenten in [Weg98,Joha97]. Das Klassendiagramm des `Agentenskelett` ist in Abbildung 5.5 dargestellt. Implementiert wurde das Agentenskelett in *LotusScript* und ist auf mehrere inhaltlich unterschiedliche Script-Bibliotheken verteilt.

Agenten und Rollen

Die NBC-DB wird durch die Klasse `NBC_Agent` repräsentiert. Diese aggregiert die Konversationsspezifikationen, -instanzen und Rollen. Die aggregierten Rollen sind Instanzen der Klassen `PerformerRole` und `CustomerRole`, die Subklassen der Superklasse `Role` sind. Die Existenz der Klasse `CustomerRole` begründet sich in ihrer Relevanz für die Realisierung von Subkonversationen. Alle Instanzen der Rollen kennen die Konversationsspezifikationen, in deren Kontext sie agieren.

[5] Siehe Abschnitt 3.5.4

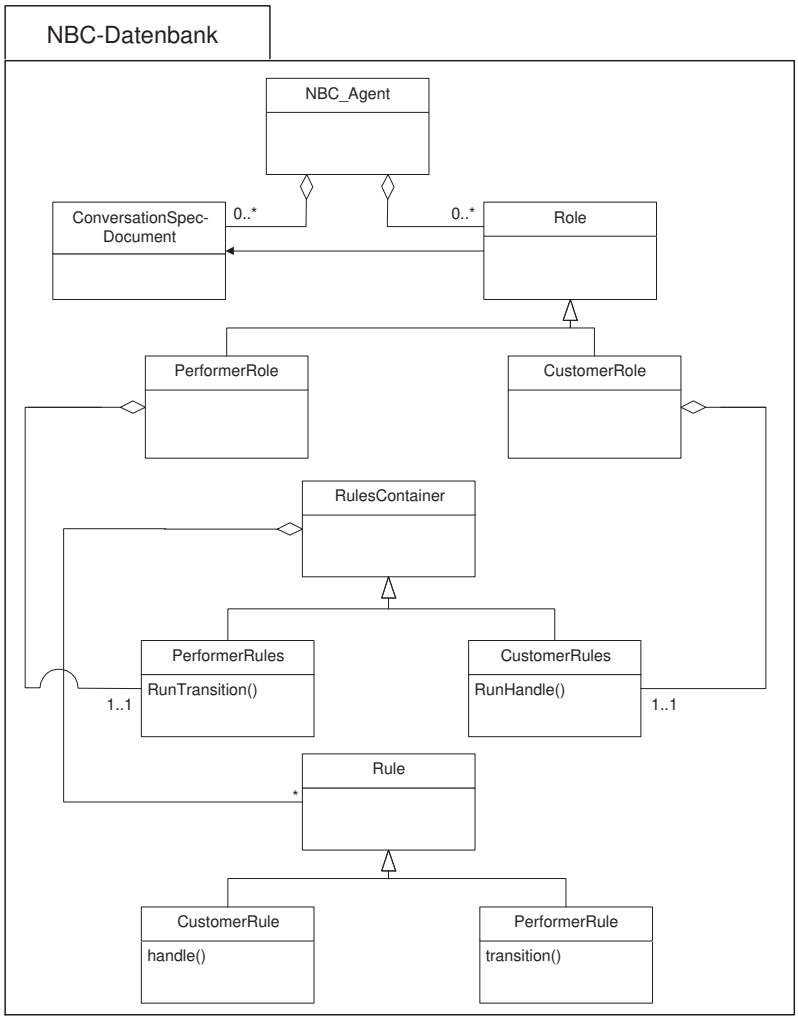

Abbildung 5.5: Klassendiagramm des Agentenskeletts

Anwendungslogikschnittstelle und Regeln

Analog zu der Darstellung in [Weg98] können Konversationsspezifikationen als nichtdeterministische Automaten verstanden werden, in denen die Übergangsfunktionen deklarativ als Regeln implementiert werden. Es existieren zwei Mengen von Übergangsfunktionen: die der Kunden- und die der Dienstleisterregeln. Die Kundenregeln modifizieren eingehende Dialoge und die Dienstleisterregeln generieren aus eingehenden Anfragen Folgedialoge. Die Anwendungslogikschnittstelle wird durch die Regeln realisiert, in denen die Anwendungslogik vom Benutzer implementiert werden muß.

Die konkreten Regeln sind Subklassen von `CustomerRule` und `PerformerRule`, deren Methoden `handle` und `transition` die Anwendungslogik enthalten. Aggregiert werden die Regeln von der Klasse `RulesContainer`, deren Subklassen `PerformerRules` und `CustomerRules` entweder Instanzen von `CustomerRule` oder `PerformerRule` aggregieren können. Die Methoden `RunTransition` und `RunHandle` ermöglichen den Aufruf der Regeln. Dabei wird den Metho-

den der Name des Regelobjekts übergeben, anhand dessen die korrekte Regel ausgeführt wird. Der Name des Regelobjekts ist in der zugehörigen Anfragespezifikation enthalten. Die Rolle `PerformerRole` aggregiert die `PerformerRules` und die `CustomerRole` die `CustomerRules`. Die Rollenobjekte sind somit mit den Konversationspezifikationen und den Regeln parametrisiert. Ergänzend zu den benutzerdefinierten Kundenregeln wurde eine vordefinierte und nicht modifizierbare Kundenregel der `PerformerRole` implementiert, die automatisch anhand eines in den Anfragespezifikationen angebenen eindeutigen Bezeichners, den zugehörigen Folgedialog ermittelt und zurückgibt. Dies ermöglicht es, Verweise auf Dialoge ohne Programmierung zu realisieren. Für die Entwicklung des IPOS war die automatische Regel eine große Vereinfachung, da oft Verweise ohne Anwendungslogik benutzt werden.

Die Verwendung der `RulesContainer` begründet sich in den Sichtbarkeitsregeln der verwendeten Script-Bibliotheken und in dem Anspruch eine möglichst einfache Anwendungslogikschnittstelle zu implementieren. Damit läßt es sich erreichen, daß der Benutzer in zwei Script-Bibliotheken `PerformerRules` und `CustomerRules` die Regeln implementieren kann und mit dem übrigen Agentenskelett nicht belastet wird. Ein Beispiel für die Verwendung der Regeln findet sich weiter unten in diesem Abschnitt.

Interaktion

In den vorigen Betrachtungen wurden bisher nur die statischen Aspekte beschrieben. Deswegen soll jetzt der dynamische Ablauf einer Konversation zwischen dem Agentenskelett und dem generischen Kunden im HTTP-Client, der in Abbildung 5.6 dargestellt ist, beschrieben werden.

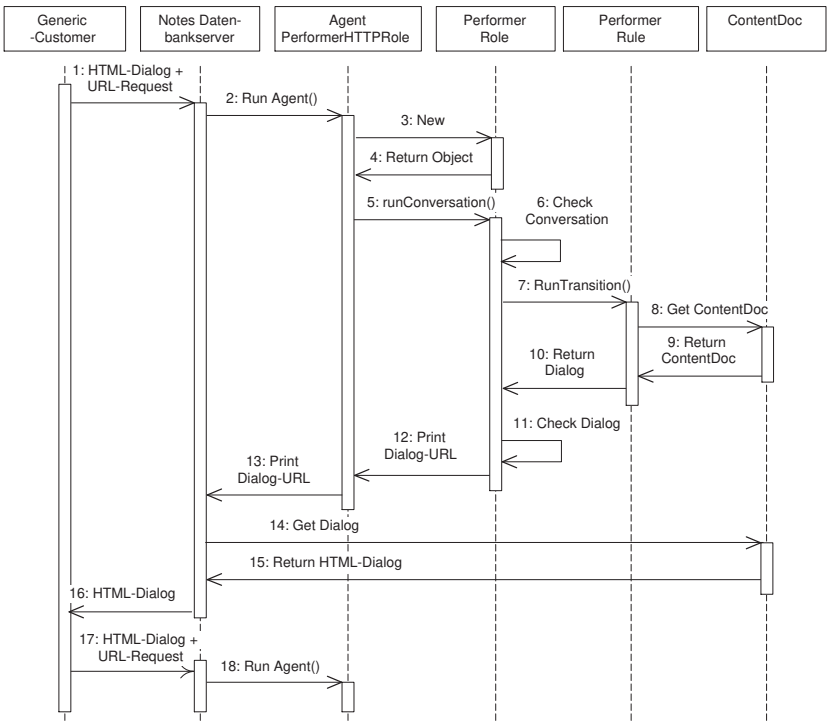

Abbildung 5.6: Interaktionsdiagramm für eine Konversation

Eine Konversation wird vom generischen Kunden (*Generic Customer*) durch eine initiale HTTP-Anfrage begonnen. Die Anfrage wird vom *Notes* Datenbankserver in einen Aufruf für den Agenten `PerformerHTTPRole` umgesetzt. Für jede Anfrage des generischen Kunden wird ein neuer Agent mit einem eigenen Thread gestartet. Der Agent erzeugt den Dienstleister ein Objekt der Klasse `PerformerRole` und startet die Konversation. Danach überprüft der Dienstleister die Konversation, ob es eine neue oder bereits vorhandene Konversation ist. Eine bereits vorhandene Konversation wird wiederhergestellt und eine neue erzeugt. Anhand der Spezifikation der Konversation wird die Anfrage ausgewertet und die jeweilige Regel aus den Regelbehälter `PerformerRules` mit `RunTransition` ausgeführt. Die Regel führt die Anwendungslogik aus, wofür der Inhalt des übergebenen Dialogs benutzt werden kann und gibt den Folgedialog zurück. Anschließend überprüft der Dienstleister den Folgedialog, ob er gemäß der Spezifikation korrekt ist. Falls er korrekt ist, wird die URL des Folgedialogs ausgegeben. Die URL wird vom *Notes* Datenbankserver ausgewertet und der Inhalt des Dialogs an den generischen Kunden geschickt. Daraufhin beginnt der Zyklus von vorn, bis ein finaler Dialog erreicht wird.

Ausführungskontext

Der Ausführungskontext ist aufgrund der zustandslosen Verbindung mit dem generischen Kunden im HTTP-Client ein Problem. Alle Objekte, die während einer Verbindung von dem `Agentenskelett` erzeugt werden, sind mit Ausnahme von Dokumenten nicht persistent und werden nach Terminierung des ausführenden Agenten gelöscht. Der Ausführungskontext muß also bei jeder erneuten Anfrage wiederhergestellt werden. Die Lösung hierfür ist, daß für jede konkrete Konversation ein Dokument generiert wird, welches eine eindeutige Identifikationsnummer (`ConversationID`) hat[6]. Damit der Ausführungskontext wiederhergestellt werden kann, muß die `ConversationID` immer vom Kunden und Dienstleister übergeben werden. Anhand der `ConversationID` kann dann das Konversationsdokument ermittelt werden[7]. Das Konversationsdokument wird bei der Ausführung einer Regel übergeben, so daß die Anwendungslogik darauf zugreifen kann. Da die Struktur von Dokumenten beliebig erweitert werden kann, besteht für die Anwendungslogik die Möglichkeit, Konversationsdokumente zu nutzen, um persistente Informationen zu speichern, auf die sie im Laufe der Konversation zugreifen kann.

Fehlerbehandlung

Die wichtige Frage der Fehlerbehandlung bei der Kommunikation wird im `Agentenskelett` detailliert betrachtet. Dabei werden zwei Arten von Fehlerquellen unterschieden:

1. Die erste sind mögliche Fehlerquellen in Konversationen. Als Fehlerquellen können folgende identifiziert werden:
   - Regelbindungen in den Anfragespezifikationen sind fehlerhaft. Es können z.B. keine Regeln angegeben oder anhand von Tippfehlern fehlerhaft eingegegeben sein.
   - Regeln können fehlerhaft implementiert werden, so daß es zu einem Programmabbruch kommt.
   - Generierte Folgedialoge von Anfragen können aufgrund fehlerhafter Regeln inkorrekt sein.

[6] Siehe Abschnitt 5.2.1

[7] Im dynamischen Ablauf durch `Check Conversation` gekennzeichnet.

2. Die zweite behandelt inhaltliche Fehler in den Konversationen, die durch den Kunden erzeugt werden können. Dieses sind z.B. fehlerhafte oder fehlende Eingaben in Dialogen.

Die Fehlerbehandlung der beiden Arten von Fehlerquellen erfolgt auf unterschiedlichen Wegen:

1. Fehler der ersten Art werden sofort abgefangen und ein Fehlerdialog wird automatisch generiert, der dem Kommunikationspartner übermittelt wird. Der Fehlerdialog wird in den Konversationsspezifikationen durch eine Dialogspezifikation repräsentiert. Das bietet die Möglichkeit dem Fehlerdialog Anfragespezifikationen zuzuweisen, die in diesem Fall eine Weiterführung der Konversation erlauben. Außerdem kann der Fehlerdialog ein finaler Dialog sein, wenn ihm keine Anfragespezifikationen zugewiesen werden. Generell ist die Fehlerdialogspezifikation implizit Folgedialog aller möglichen Anfragen.

2. Inhaltliche Fehler der zweiten Art werden abgefangen, sofern in den Anfragespezifikationen Felder des Dialogs spezifiziert wurden, für die eine inhaltliche Prüfung erfolgen soll. Die Prüfung stellt fest, ob die spezifizierten Felder gefüllt sind. Wenn die Prüfung ein Feld ohne Inhalt feststellt, wird der Dialog erneut übermittelt, wobei die Felder, die gefüllt werden sollen, gekennzeichnet sind. Diese Fehlerbehandlung war für die Implementierung des IPOS sehr nützlich, da erforderliche Felder damit leicht implementiert werden konnten.

Speziell die Fehlerbehandlung der zweiten Art könnte weiter ausgebaut werden, so daß Typ- oder Wertprüfungen möglich sind, die eine genauere inhaltliche Fehlerbehandlung ermöglichen.

Implementierungsentscheidungen

Die Implementierungsentscheidung bezüglich der Regelanbindung soll hier beschrieben werden. Generell bestehen für die namentliche Bindung von Regeln mehrere Möglichkeiten in *LotusScript*. Die gewählte und vorgestellte Möglichkeit ist eine rein objektorientierte Anbindung der Regeln.

Eine alternative Lösung würde das Schlüsselwort `Execute` verwenden, das die Kompilierung des Inhalts einer Variable vom Typ `String` zur Laufzeit erlaubt und als temporäres Modul ausführt. Die namentliche Bindung der Regeln würde mit Hilfe eines erst zur Laufzeit kompilierten Textes realisiert werden. Der Nachteil dieser Lösung ist, daß nur Funktionen oder Subroutinen aber keine Methoden von Klassen ausgeführt werden können. Ein zusätzliches Problem ist die Notwendigkeit von globalen Variablen, da keine Parameter übergeben werden können. Die Eigenschaften dieser Lösung verletzen das objektorientierte Paradigma, das letztlich zur vorgestellten Lösung geführt hat. Ein Vorteil dieser Lösung ist, daß keine Regelobjekte erzeugt werden müssen, was zur Folge hat, daß Anfragen performanter abgearbeitet werden können.

Verwendung

Die Verwendung des `Agentenskeletts` geschieht auf zwei Arten:

1. Der Agent `PerformerHTTPRole` benutzt das `Agentenskelett`, um mit dem generischen Kunden zu kommunizieren.

2. Der Anwendungsentwickler implementiert die Anwendungslogik in den Regelklassen der Schnittstelle zur Anwendungslogik.

Damit der Agent das `Agentenskelett` benutzen kann, muß er folgende Punkte erfüllen:

- Die ScriptBibliothek `NBC Roles`, die alle Script-Bibliotheken des Agentenskeletts enthält, muß importiert werden.
- Das *LotusScript*-Programm des Agenten muß bei einer Aktivierung die Dienstleisterrolle instantiieren und starten.

Nachfolgend ist das *LotusScript*-Programm des Agenten abgebildet:

```
Sub Initialize
    Dim PerformerRole As New PerformerRole()
    Call Performer.RunConversation()
End Sub
```

Die Implementierung der Regelklassen durch den Anwendungsentwickler erfolgt in den Script-Bibliothek `PerformerRules` und `CustomerRules`, in der die Regelklassen und der Regelbehälter implementiert werden müssen. Als Beispiel soll hier die Entwicklung der Regelklassen für den Dienstleister betrachtet werden. Der erste Schritt ist die Implementierung der Regelklassen, wovon die Regelklasse `OpenProduct`, die einen Produktdialog zurückgibt, nachfolgend zur Erläuterung dargestellt ist:

```
Class OpenProduct As PerformerRule
   Sub Transition(NBC_Agent As NBC_Agent, CurrentDialog As NBC_Dialog,
                  Request As String,  ConversationInstance As Conversation)
    Dim Product_Db As New NBC_Database(
      NBC_Agent.GetDatabaseSpecificationDocument( "Productcatalog" ))
    Dim Product_Doc As NotesDocument
    Set Product_Doc = Product_Db.GetDocumentbyID( CurrentDialog.Value )
    Set NextDialog = New NBC_Dialog( Product_Doc )
   End Sub

   'Konstruktor
   Sub New(RuleName As String)
   End Sub
End Class
```

Als zweiten Schritt müssen alle Regeln, die verwendet werden sollen, in dem Konstruktor der Klasse `PerformerRules` erzeugt werden. Zur späteren Identifikation wird dem Konstruktor der Regeln ein Name übergeben, der sie eindeutig identifiziert. Mit der Methode `AddRule` werden sie in das Array von Regeln eingetragen, das die Klasse `PerformerRules` von ihrer Superklasse `RulesContainer` erbt. Ein Beispiel der Klasse `PerformerRules` ist nachfolgend zu sehen.

```
Class PerformerRules As RulesContainer
   'Konstruktor
```

```
   Sub New()
     Dim Rule As PerformerRule
     Call Me.ReDimRule(6)
     Set Rule = New CheckAvailability( "CheckAvailability" )
     Call Me.AddRule( Rule )
     ...
     Set Rule = New OpenProduct( "OpenProduct" )
     Call Me.AddRule( Rule )
   End Sub
End Class
```

Der letzte Schritt ist die Bindung der Regeln an die Anfragespezifikationen. Dieses geschieht mittels der Administrationsschnittstelle über das Eintragen des Namens der Regeln in die Anfragespezifikation. Damit wird die einfache dynamische Modifikation der Regelbasis ermöglicht, da neue Regeln mit ihrem Namen in die Anfragespezifikation eingetragen werden.

5.2.4 Administrationsschnittstelle

Nachdem im vorherigen Abschnitt das `Agentenskelett` und die Anwendungslogikschnittstelle der NBC-DB erläutert wurde, ist die Administrationsschnittstelle Gegenstand dieses Abschnitts. Die Administrationsschnittstelle stellt Dienstleistungen zur Verfügung, die entsprechend der Anforderungen eine leichte Administration der NBC-DB ermöglicht. Administriert werden die in der NBC-DB gespeicherten

- **Konversationsspezifikationen** und
- **Konversationsinstanzen**.

Mit Hilfe der Administrationsschnittstelle kann der Administrator die Spezifikationen der Konversationen erstellen und modifizieren oder laufende Instanzen von Konversationen inspizieren. Dieses ermöglicht dynamische Änderungen der Konversationsspezifikationen und Auswertungen der konkreten Instanzen der Konversationen. Davon ausgenommen sind die Spezifikationen und Instanzen der Inhalte[8], da sie nicht in der NBC-DB gespeichert werden. Dieses ist in ihrer Unabhängigkeit von der NBC-DB begründet. Aufgrund der unterschiedlichen Semantik der Konversationsspezifikationen und -instanzen sind unterschiedliche Dienstleistungen für die Komponenten notwendig[9], die in der Tabelle 5.1 dargestellt sind.

Die Einschränkung der Dienstleistungen für die Konversationsinstanzen ist darin begründet, daß die Modifikation und Erzeugung automatisiert durch das `Agentenskelett` vorgenommen wird. Die Modifikation der Konversationsinstanzen wäre theoretisch denkbar, um z.B. auf Fehler in einer konkreten Konversation zu reagieren und diese zu beheben. Die Modifikation der Konversationsinstanzen könnte aber dafür genutzt werden, eine konkrete Konversation zu verändern, so daß sie sich nicht mehr entsprechend der Konversationsspezifikationen verhält. Aus diesem Grund sind sie nicht modifizierbar.

[8] Eine Ausnahme sind die Instanzen der Inhalte der Fehlerdialoge, die in der NBC-DB gespeichert werden.
[9] Siehe 5.2.1

Dienstleistung	Konversationsspezifikationen	Konversationsinstanzen
Anzeige	X	X
Erzeugung	X	-
Modifikation	X	-
Löschen	X	X

Tabelle 5.1: Dienstleistungen der Administrationsschnittstelle

Die Administration der persistenten Konversationsspezifikationen und -instanzen, die aus Masken und Dokumenten bestehen, muß auf zwei Arten betrachtet werden, da eine uniforme Behandlung von Masken und Dokumenten nicht möglich ist:

- Masken können erstellt, angezeigt, modifiziert und gelöscht werden. Dieses geschieht in der Gestaltungssicht der NBC-DB, in der Masken erstellt und gelöscht werden können. Die Modifizierung und die Anzeige der Masken geschieht über den *Notes* internen Maskeneditor, der aus der Gestaltungssicht geöffnet wird.

- Dokumente werden mit Hilfe von Masken und Ansichten administriert. Mit Masken können Dokumente erstellt, modifiziert und angezeigt werden. Ansichten bieten verschiedene Sichten und Auswertungen auf Dokumente. Dokumente können tabellarisch nach bestimmten Kriterien selektiert, sortiert und kategorisiert werden. Dokumente können nur in Ansichten gelöscht werden.

Die Administration der Masken, die z.B. für die Erstellung der Masken der Konversationsspezifikationen notwendig ist, wird nicht weiter betrachtet. Ein Implementierungsaufwand bestand hierfür nicht, da diese mit der *Notes* internen Gestaltungssicht und dem Maskeneditor erfolgt[10].

Im folgenden werden die Implementierungen der Administrationsschnittstelle für die Konversationsspezifikationen in Abschnitt 5.2.4.1 und Konversationsinstanzen in 5.2.4.2 erläutert. Dafür werden die erstellten Masken und Ansichtengruppen für die einzelnen Dokumenttypen betrachtet.

Die Ansichtengruppen, die vorgestellt werden, stellen nur eine Teilmenge der möglichen Ansichten dar. Für den Anwendungszweck in dieser Arbeit war dieser Umfang aber völlig ausreichend. Auf sinnvolle Erweiterungen der Ansichtengruppen wird im Text hingewiesen. In Anhang F sind die erstellten Ansichten tabellarisch geordnet nach Ansichtengruppen dargestellt.

5.2.4.1 Konversationsspezifikationen

Die Administrationsschnittstelle für die Konversationsspezifikationen stellt die im vorigen Abschnitt beschriebenen Dienstleistungen für die Dokumenttypen der Konversations-, Dialog- und Anfragespezifikationen zur Verfügung. Für diesen Zweck wurden Masken und Ansichten

[10] Die Möglichkeiten zur Administration der Masken können Kapitel 3 entnommen werden.

erstellt, die den Anforderungen gerecht werden. Nachfolgend werden die erstellten Masken und Ansichten beschrieben:

Masken: Für die Erzeugung, Modifikation und Anzeige der Dokumenttypen der Dialog-, Anfrage- und Konversationsspezifikationen wurden folgende Masken erstellt:

- *ConversationSpec*: Erzeugt, modifiziert und zeigt Konversationsspezifikationsdokumente an.
- *DialogSpec*: Erzeugt, modifiziert und zeigt Dialogspezifikationsdokumente an.
- *RequestSpec*: Erzeugt, modifiziert und zeigt Anfragespezifikationsdokumente an.

Damit die in Abschnitt 5.2.1 beschriebenen Beziehungen der Spezifikationen untereinander ermöglicht werden, beinhalten die Masken Schaltflächen und Ereignisse, die Beziehungen mittels assoziativen Zugriffs realisieren und die referentielle Integrität dieser Beziehungen sicherstellen. Als Beispiel soll das Ereignis `QueryClose` genannt werden, das beim Schließen eines Dokuments ausgelöst wird. Wenn eine Anfrage einem Dialog zugewiesen wurde, wird der Dialog automatisch in die Menge der Dialoge in der entsprechenden Anfrage eingetragen. Dieses gewährleistet die referentielle Integrität.

Ansichten: Für jeden Dokumenttyp der Konversationsspezifikationen wurde eine Ansichtengruppe erstellt:

1. Die Ansichtengruppe „*ConversationSpecs by*“ zeigt die Konversationsspezifikationen an.
2. Die Ansichtengruppe „*DialogSpecs by*“ zeigt die Dialogspezifikationen an.
3. Die Ansichtengruppe „*RequestSpecs by*“ zeigt die Anfragespezifikationen an.

Äquivalent zu den Ansichtengruppen für die einzelnen Dokumenttypen, wurde eine gemeinsame Ansichtengruppe „*DialogSpecs + RequestSpecs by*“ für die Dialog- und Anfragespezifikationen erstellt, welche die `N:M`-Beziehungen zwischen den Dokumenttypen in einer hierarchischen Struktur visualisiert. Dafür werden die Einträge in der Ansicht so oft dupliziert wie nötig, wobei die Duplikate nur auf ein Dokument verweisen.

Für die Erweiterung der Ansichten wäre es sinnvoll gemeinsame Ansichtengruppen für die Dokumenttypen der Konversations-/Dialogspezifikationen und Konversations-/Dialog-/Anfragespezifikationen zu erstellen. Diese würden die Zusammenhänge zwischen den Dokumenttypen aufzeigen und könnten für Auswertungen benutzt werden.

5.2.4.2 Konversationsinstanzen

Analog zum vorigen Abschnitt werden die Masken und Ansichten für die Dokumenttypen der Konversationsinstanzen erläutert. Betrachtet werden Konversationsinstanzen und Fehlerdialoginhalte. Fehlerdialoginhalte werden berücksichtigt, da diese als einzige Inhalte in der NBC-DB gespeichert werden. Nachfolgend werden die erstellten Masken und Ansichten beschrieben:

Masken: Für die Anzeige der Dokumenttypen der Instanzen der Konversationen und Fehlerdialoginhalte wurden folgende Masken erstellt:

- *ConversationInst*: Zeigt die Instanzen der Konversationen an.
- *ErrorContentInst*: Zeigt die Fehlerdialoginhalte von konkreten Konversationen an.

Die Fehlerdialoginhalte, die automatisch bei einem Fehler von dem `Agentenskelett` erzeugt werden[11], sollen Informationen liefern, anhand derer aufgetretene Fehler analysiert und behoben werden können.

Ansichten: Für die Dokumenttypen der Konversationen und der Fehlerdialoginhalte wurden zwei Ansichtengruppen erstellt:

1. Die Ansichtengruppe „*ConversationInst by*“ zeigt die Instanzen der Konversationen an.
2. Die Ansichtengruppe „*ErrorDialogContentInst by*“ zeigt die Instanzen der Fehlerdialoginhalte an.

Für die Erweiterung der Ansichten der Fehlerdialoginhalte wäre es sinnvoll z.B. nach der Anfrage oder dem Dialog zu sortieren, indem der Fehler auftrat. Für die Instanzen der Konversationen könnten Ansichten erstellt werden, die nach dem aktuellen Dialog sortiert sind.

5.2.5 Subkonversationen

In diesem Abschnitt soll das implementierte Konzept für Subkonversationen erläutert werden. Das Konzept wurde auf der Grundlage der theoretischen Betrachtungen in [Weg98; Joh97; Ric97] erstellt. Subkonversationen sollen die Wiederverwendung von definierten Konversationsspezifikationen in anderen Konversationsspezifikationen erlauben. Im weiteren werden die Betrachtungen aus [Joh97,Ric97] und [Weg98] kurz beschrieben und anschließend das in der NBC-DB implementierte Konzept erläutert.

- In [Joh97; Ric97] bestehen Konversationsspezifikationen aus mehreren abstrakten Dialogspezifikationen. Die abstrakten Dialogspezifikationen bestehen entweder aus Spezifikationen von Subkonversationen oder Dialogen. Somit können Subkonversationsspezifikationen beliebig in Konversationsspezifikationen benutzt werden. Darüber hinaus soll ein *Modifier* die Modifizierung der Subkonversationsspezifikation erlauben, was aber nicht näher erläutert wird. Weitere Betrachtungen über den Ablauf einer Subkonversation wurden in den Arbeiten nicht vorgenommen.
- In [Weg98] werden die kritischen Punkte für Subkonversationen betrachtet. Zusammengefaßt kommt die Betrachtung zu folgenden Ergebnissen: Der Einstieg in eine Subkonversation ist unkritisch und erfolgt beim initialen Dialog der Subkonversationsspezifikation. Im Gegensatz dazu ist der Ausstieg problematisch, da die Subkonversation mit Erreichen eines finalen Dialogs wieder zu der übergeordneten Konversation zurückkehren muß. Die Lösung für dieses Problem ist die Bindung einer Regel auf der Kundenseite, die in diesem Fall über die zu stellende Anfrage entscheidet. Analog muß der Dienstleister für jede Anfrage, die aus der Subkonversation folgt, eine Regel binden. Damit die

[11] Siehe Abschnitt 5.2.3

Wiederverwendbarkeit nicht eingeschränkt ist, soll für jede Subkonversation eine eigene Rolle gebunden werden.

Das Konzept für Subkonversationen, das in dieser Arbeit implementiert wurde, beruht auf einem Ansatz, der sich aus den oberen Betrachtungen teilweise ergibt. Dafür werden zuerst die notwendigen Erweiterungen für die Ebene der Konversationsspezifikationen erläutert und dann auf die Abwicklung einer Subkonversation eingegangen, in der die Erweiterungen der Konversationsinstanzen erläutert werden. In beiden Abschnitten wird die Abbildung 5.7 benutzt, die eine Konversationsspezifikation mit Subkonversationsspezifikation darstellt. Im Text werden die Äquivalente der Abbildung in Klammern angegeben.

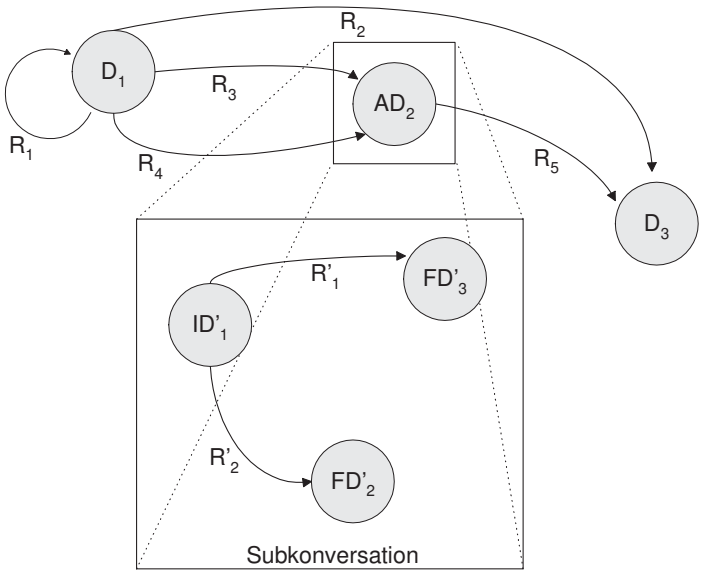

Abbildung 5.7: Subkonversation als abstrakter Dialog

Die Subkonversationsspezifikation wird in der Superkonversationsspezifikation[12] durch eine abstrakte Dialogspezifikation repräsentiert. Die abstrakte Dialogspezifikation ist eine uneingeschränkte Dialogspezifikation, der Anfragespezifikationen zugeordnet werden können. In Abbildung 5.7 ist der abstrakte Dialog durch $\mathtt{AD}_2$ dargestellt. Für eine Subkonversationsspezifikation, die die Verfeinerung der abstrakten Dialogspezifikation ist, müssen die finalen Dialogspezifikationen, dargestellt durch $\mathtt{FD}'_2$ und $\mathtt{FD}'_3$, angegeben werden[13], um den Ausstieg aus einer Subkonversation erkennen zu können.

Um den Übergang zu einer Subkonversationsspezifikation in der Superkonversationsspezifikation einfach zu erkennen, wurde die Spezifikation der Anfrage erweitert. Innerhalb einer

[12] Die Superkonversation ist die Konversationsspezifikation, in die die Subkonversationsspezifikation eingefügt wird.

[13] Auf die Angabe der finalen Dialogspezifikationen könnte eigentlich verzichtet werden, da die finalen Dialogspezifikationen ermittelt werden können. Die Implementierung wurde dadurch aber vereinfacht.

Anfragespezifikation muß für diesen Übergang die aufzurufende Subkonversationsspezifikation und die zugehörige abstrakte Dialogspezifikation angegeben werden. Zusätzlich muß eine Regel an die Anfragespezifikation gebunden werden, die den initialen Dialog der Subkonversationsspezifikation zurückgibt.

Die Abwicklung einer Subkonversation erfordert die Klärung von mehreren Punkten. Dieses sind der

- Einstieg,
- die Ausführung und
- Ausstieg aus der Subkonversation.

Einstieg in die Subkonversation: Der Einstieg in die Subkonversation ist durch die Erweiterung der Anfragespezifikation leicht durch das `Agentenskelett` zu lösen. Wenn eine Anfrage($\mathtt{R}_3/\mathtt{R}_4$) gestellt wird, wird die Subkonversationsspezifikation ermittelt und eine neue Konversation gestartet. Um den Kontext zur Superkonversation zu behalten, müssen die Attribute der Konversationsinstanzen erweitert werden. Konversationsinstanzen benötigen ein Attribut für die Superkonversation und für den abstrakten Dialog[14]. In die Instanz der Subkonversation wird bei der Erzeugung ein Verweis auf die Superkonversation und die abstrakte Dialogspezifikation eingetragen. Der Verweis auf die abstrakte Dialogspezifikation ist für den Ausstieg aus der Subkonversation notwendig. Danach wird die durch die Anfrage gebundene Regel ausgeführt und der initiale Dialog (ID'_1) der Subkonversation wird zurückgegeben. Dieser wird dann an den Kunden übergeben.

Ausführung der Subkonversation: Die Ausführung der Subkonversation ist unkritisch und folgt dem normalen Ablauf einer Konversation.

Ausstieg aus der Subkonversation: Der Ausstieg aus einer Subkonversation ist problematischer, kann aber durch die Erweiterung der Konversationsinstanzen einheitlich erfolgen. Sobald eine Subkonversation durch das Erreichen eines finalen Dialogs ($\mathtt{FD}'_3/\mathtt{FD}'_2$) beendet wird, muß der Übergang zu der aufgeschobenen Superkonversation erfolgen. Um dieses zu ermöglichen, wird für jeden Dialog der Subkonversation geprüft, ob es ein finaler Dialog ist. Wenn dieses der Fall ist, werden die Anfragen($\mathtt{R}_5$), die dem abstrakten Dialog($\mathtt{AD}_2$) zugeordnet sind, in den finalen Dialog eingetragen. Damit wird der Übergang zu der Superkonversation ermöglicht, ohne daß in der Subkonversation Informationen über die Superkonversation erforderlich sind. Der zugehörige abstrakte Dialog wird aus der Subkonversationsinstanz ermittelt. Danach wird die Superkonversation anhand der Referenz in der Subkonversationsinstanz wiederhergestellt. Abschließend wird der finale Dialog im Kontext der Superkonversation an den Kunden übermittelt.

Unter Verwendung dieses Konzepts können Konversationsspezifikationen aus beliebigen Subkonversationsspezifikationen zusammengebaut werden.

[14] Diese Beziehungen werden im Klassendiagramm der Konversationsinstanzen durch die Beziehung „SuperConversation“ und „AbstractDialog“ abgebildet

5.3 Generischer Kunde

In diesem Abschnitt soll nun abschließend der Entwurf, die Implementierung und Verwendung des generischen Kunden vorgestellt werden, der für die NBC erstellt wurde. Dieses geschieht auf der Basis der Anforderungen aus Abschnitt 5.1.

Entwurf und Implementierung

Der Entwurf und die Implementierung des generischen Kunden muß aufgrund der Anforderungen für einen HTTP-Client erfolgen. Dies erlaubt die Nutzung der Integrationsmöglichkeiten von *Notes* für das Internet, die in Abschnitt 3.5 beschrieben wurden. Im HTTP-Client werden Dokumente von *Notes* angezeigt, die in das HTML-Format konvertiert wurden. Für jedes Dokument muß die im Sinne der Konversation korrekte Spezifikation generiert werden. Dieses sichert das Verhalten des menschlichen Kunden entsprechend der Konversationsspezifikation. Der daraus resultierende Entwurf und die Implementierung für den generischen Kunden kann am besten anhand des Interaktionsdiagramms in Abbildung 5.8 erklärt werden. In diesem wird der dynamische Ablauf der Generierung der Spezifikation für ein Dokument dargestellt.

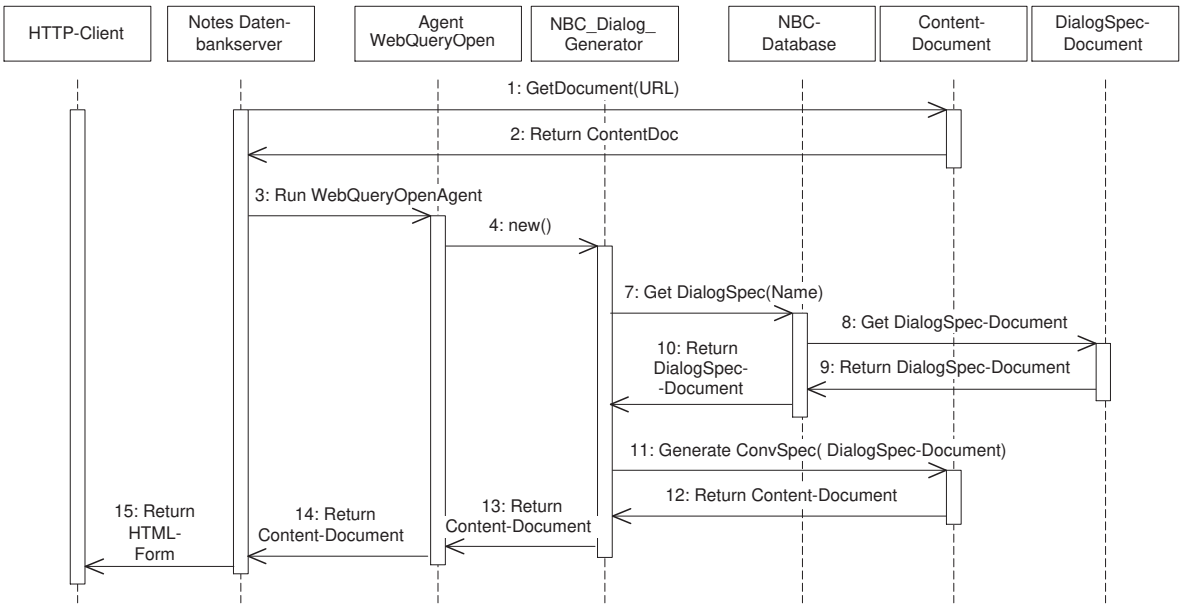

Abbildung 5.8: Interaktionsdiagramm des generischen Kunden

Der Beginn der Generierung des generischen Kunden erfolgt, nachdem der Agent `Performer-HTTPRole` die URL des nächsten Dialogs ausgegeben hat. Der Ablauf bis zu diesem Punkt ist in Abbildung 5.6 dargestellt. Der Datenbankserver ermittelt anhand der angegebenen URL den zugehörige Dialoginhalt (`Content-Document`). Daraufhin startet der Datenbankserver den im Dialoginhalt für das Ereignis `WebQueryOpen` angegebenen Agenten. Der Agent erzeugt ein Objekt der Klasse `NBC_Dialog_Generator`. Der `NBC_Dialog_Generator` holt dann die zu dem Inhalts-Dokument zugehörige Dialogspezifikation aus der NBC-Datenbank (`NBC-Database`). Die Information über die zugehörige Dialogspezifikation ist im Dialoginhalt vorhanden[15].

[15] Siehe Abschnitt 5.2.1

Anhand der Informationen in der Dialogspezifikation generiert er die korrekte Spezifikation und das Layout in dafür vorgesehene Felder des Dialoginhalts. Der Dialoginhalt wird nach der Terminierung des Agenten an den Datenbankserver übergeben, der es mit dem *Domino-Modul* in ein HTML-Formular konvertiert und an den HTTP-Client schickt.

Die für den dynamischen Ablauf notwendigen Punkte sind folgende:

1. Der erstellte Agent `WebOpen` erzeugt ein Objekt der Klasse `NBC_Dialog_Generator`. Damit der Agent für jedes Inhalts-Dokument beim Ereignis `WebQueryOpen` aufgerufen wird, muß er in die Dokumente eingetragen werden. Das geschieht über die Modifikation der zugehörigen Masken der Dialoginhalte, in der der Agent eingetragen wird, so daß er automatisch in jedes erzeugte Dokument eingetragen wird. Eine erwähnenswerte Eigenschaft der Agenten für das Ereignis `WebQueryOpen` ist, daß die Informationen nur temporär in das Dokument eingetragen werden und somit keinen Speicherplatz beanspruchen.

2. Die Masken der Dialoginhalte sind um Felder und eine Teilmaske erweitert worden, die die generierte Konversationsspezifikation und das Layout aufnehmen. Die erforderlichen Felder und die Teilmaske werden in der nachfolgenden Tabelle 5.2 beschrieben.

Erweiterung	Beschreibung
HTML_Background	Enthält die URL des Hintergrundbildes.
HTML_Body	Enthält Text mit HTML-Befehlen.
Linkleiste	Enthält Anfragen.
Aktionsleiste	Enthält Anfragen.
Failure	Enthält Fehlermeldungen.
Teilmaske NBC Fields	Enthält wichtige Felder für die NBC wie ConversationID, Benutzername und der Name der Dialogspezifikation

Tabelle 5.2: Erweiterungen für die Masken der Inhalte

3. Die *LotusScript*-Klasse `NBC_Dialog_Generator` stellt Methoden bereit, die korrekte Spezifikation und das Layout anhand der Informationen im zugehörigen Dialogspezifikationsdokument in den Dialoginhalt zu generieren.

4. Die Masken der Dialog- und Anfragespezifikationen in der NBC-DB sind erweitert worden, um die Informationen für den generischen Kunden zur Verfügung zu stellen. Damit eine Anordnung und Konfiguration der Anfragen in verschiedenen Leisten möglich ist, wurden eine Linkleiste und eine Aktionsleiste eingeführt, denen Anfragen zugeordnet werden können. Außerdem kann das Layout der Anfragen und der Dialoginhalte konfiguriert werden. Die Erweiterungen der Felder der Spezifikationen sind in Tabelle 5.3 dargestellt.

   Eine zusätzliche Erweiterung der Masken ist die Generierung der HTML-Befehle für die Leisten der Anfragen, die in den Dialogspezifikationen gespeichert werden. Damit wird erreicht, daß diese nicht während der Laufzeit erstellt werden müssen und so das System unnötig belasten. Die Generierung der Anfragen erfolgt bei der Speicherung

der Dialog- oder Anfragespezifikationen, wenn relevante Modifikationen an Feldwerten stattgefunden haben.

Dialogspezifikation	Beschreibung
HTML_Background	Ermöglicht die Eingabe der URL des Hintergrundbildes.
HTML_Body	Ermöglicht die Eingabe von Text mit HTML-Befehlen, die im Dialog angezeigt werden.
Ausrichtung der Linkleiste	Legt fest, ob die Ausrichtung der Linkleiste vertikal oder horizontal sein soll.
Linkleiste	Enthält alle Anfragen, die der Linkleiste zugeordnet sind.
Ausrichtung der Aktionsleiste	Legt fest, ob die Ausrichtung der Aktionsleiste vertikal oder horizontal sein soll.
Aktionsleiste	Enthält alle Anfragen, die der Aktionsleiste zugeordnet sind.
Anfragespezifikation	**Beschreibung**
Leiste	Legt fest, in welcher Leiste die Anfrage angezeigt werden soll.
Anzeigenummer	Die Anzeigenummer legt die Reihenfolge der angezeigten Anfragen fest.
Anzeigename	Ermöglicht die Eingabe eines Textes, der für die Anfrage im Dialog angezeigt wird.
Anzeigename vor dem Namen	Ermöglicht die Eingabe der URL eines Bildes, das vor dem Anzeigenamen angezeigt wird.
Anzeigebild	Ermöglicht die Eingabe der URL eines Bildes, das für die Anfrage im Dialog angezeigt wird.
Ausrichtung des Seperators	Bestimmt die Ausrichtung des Seperators entweder vertikal oder horizontal.
Vertikaler Seperator	Ermöglicht die Eingabe der URL eines Bildes, das für den vertikalen Seperator angezeigt wird.
Horizontaler Seperator	Ermöglicht die Eingabe der URL eines Bildes, das für den horizontalen Seperator angezeigt wird.

Tabelle 5.3: Erweiterungen für die Dialog- und Anfragespezifikationen

Implementierungsentscheidungen

Eine weitere Möglichkeit, mit der der generische Kunde erzeugt werden kann, könnte durch eine Erweiterung des Agentenskeletts erfolgen. Dieses würde die Anfragen für jeden Dialoginhalt generieren. Der Nachteil dieser Lösung ist, daß die Anfragen in den Dokumenten der Dialoginhalte gespeichert werden, welches erstens zu einem Anstieg des Speicheraufwandes führt und zweitens Probleme bei der Mehrbenutzernutzung verursacht. Im Mehrbenutzerbetrieb würde das Agentenskelett parallel verwendet werden und beim Zugriff auf die gleichen Dialoginhalte, könnten gleichzeitige Speicherungen auftreten. Dieses würde zu Replikationskonflikten führen, die das System belasten und zu Fehlern führen können.

Verwendung

Die wichtigste Verwendung des generischen Kunden ist für den Kunden, der mit Hilfe des generischen Kunden legale Konversationen mit der NBC-DB führt.

Weiterhin wird der generische Kunde über die Administrationsschnittstelle der NBC-DB verwendet, mit der der generische Kunde konfiguriert werden kann. Dafür können in den Dialog- und Anfragespezifikationen die in Tabelle 5.3 genannten Felder gefüllt werden, mit denen das Layout des generischen Kunden konfiguriert werden kann.

Kapitel 6

Realisierung des Online-Verkaufssystems: Internet Point of Sale

In diesem Kapitel werden die objektorientierte Analyse, Entwurf und die Implementierung des **I***nternet* **P***oint* **o***f* **S***ale* (IPOS) vorgestellt. Diese erfolgen unter Verwendung des in Abschnitt 4.2 vorgestellten Vorgehensmodells.

Zuerst wird im Abschnitt 6.1 die objektorientierte Analyse beschrieben, die auf der Grundlage der Analyse der kommerziellen Produkte in Kapitel 2 erstellt wurde. Der objektorientierte Entwurf ist Bestandteil des Abschnittes 6.2. Die Implementierung wird dann in Abschnitt 6.3 vorgestellt.

Die Möglichkeiten des IPOS zur Erweiterung und Integration externer Systeme werden in Abschnitt 6.4 dargestellt. Danach wird die Implementierung der Integration des *SAP R/3*-Systems beschrieben.

6.1 Analyse

Die Analysephase des IPOS beruht auf der Anforderungsdefinition aus Abschnitt 2.4. Auf dieser Basis werden die Akteure des IPOS in Abschnitt 6.1.1 identifiziert. Daraufhin werden die typischen Interaktionen der Akteure mit dem IPOS in 6.1.2 ermittelt. Dieses sind die Anwendungsfälle der Benutzer, die im IPOS berücksichtigt werden müssen. Die Akteure und Anwendungsfälle sind Grundlage für die Entwurfsphase.

Zusätzlich wurde, um die Konformität der Begrifflichkeit zu sichern, ein Glossar der Begriffe des IPOS erstellt, das in Anhang D teilweise enthalten ist.

6.1.1 Akteure

Die Akteure des Systems sind die Benutzer des Systems und stehen für alles, was in irgendeiner Form mit dem System Informationen austauscht. Anhand der Analyse in Kapitel 2 und der Anforderungsdefinition können vier Akteure identifiziert werden:

1. anonyme Kunden
2. registrierte Kunden
3. Systemadministratoren
4. Verkaufsleiter

Die **anonymen Kunden** benutzen die öffentlichen Bereiche des IPOS, um sich über die Inhalte zu informieren. Falls sie sich entschließen etwas zu kaufen, werden sie zu **registrierten Kunden**, denen alle Bereiche des IPOS zugänglich sind. Die **Systemadministratoren** betreuen die Benutzerverwaltung und den generellen Aufbau des IPOS. Die **Verkaufsleiter** sind für die Inhalte des IPOS verantwortlich. Dieses bedeutet, daß sie den Produktkatalog, die Inhalte der HTML-Seiten, das Diskussionsforum usw. pflegen müssen.

6.1.2 Anwendungsfälle

Die Anwendungsfälle der Akteure mit dem IPOS lassen sich direkt aus der Anforderungsdefinition und ergänzend durch die Funktionen der kommerziellen Produkte ableiten. Auszüge aus den Katalogen der Anwendungsfälle der Kunden, Systemadministratoren und Verkaufsleiter finden sich im Anhang D. Die Anwendungsfälle der anonymen und registrierten Kunden sind zusammengefaßt dargestellt, wobei gekennzeichnet ist, ob der Anwendungsfall auch für anonyme Kunden verfügbar ist.

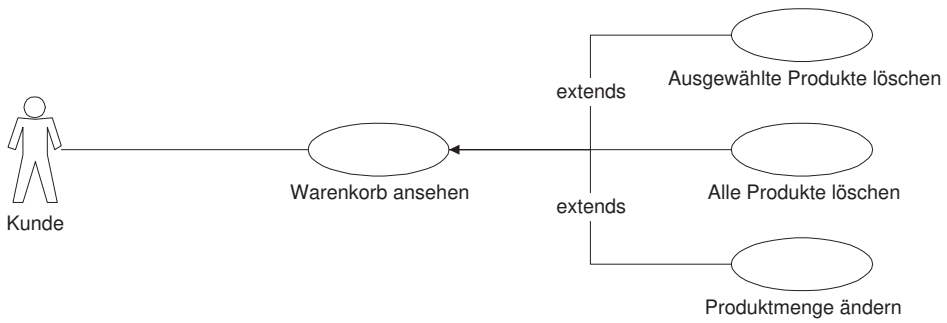

Abbildung 6.1: Anwendungsfall „Warenkorb ansehen“ des Kunden

Als Beispiel ist in Abbildung 6.1 ein Diagramm für den Anwendungsfall „Warenkorb ansehen“ dargestellt. Der registrierte Kunde kann sich seinen persönlichen Warenkorb ansehen. Außerdem werden ihm spezialisierte Funktionen angeboten, im Diagramm durch die „extends“-Beziehung gekennzeichnet, mit denen er Produktmengen ändern und einzelne oder alle Produkte löschen kann.

6.2 Entwurf

Die Entwurfsphase des IPOS beinhaltete die Festlegung der Architektur des IPOS, sowie die Erstellung von Zustandsdiagrammen, die als Spezifikationen der Konversationen des IPOS mit dem Kunden dienen und der Entwicklung des Entwurfs-Klassendiagramms. Ergänzend zu den Zustandsdiagrammen wurden die Zustände im Sinne der Dialoge des Modells der *Business Conversations* als prototypische Masken in *Notes* entworfen.

Die Architektur des IPOS wird hier nicht betrachtet, da sie bereits in Abschnitt 2.4 beschrieben und in Abbildung 2.8 dargestellt wurde.

6.2.1 Zustandsdiagramme, Konversationsspezifikationen und Maskenentwurf

Die erstellten Zustandsdiagramme beschreiben die Interaktionen, in denen die Akteure mit dem IPOS interagieren. Die Grundlage für die Zustandsdiagramme stellten die identifizierten Anwendungsfälle dar, für die Zustandsdiagramme erstellt wurden. Um den bruchlosen Übergang von der Prozeßmodellierung zur Implementierung zu unterstützen, dienten die Zustandsdiagramme als Grundlage für die Spezifikationen der möglichen Konversationen zwischen dem IPOS und den Akteuren. Die Zustände repräsentieren die Dialoge und die Transitionen Anfragen der Konversationsspezifikationen. Weiterhin wurden für die Zustände Masken in *Notes* entworfen, die einen ersten Prototypen für die Benutzerschnittstelle ermöglichten. Der Vorteil in der Verwendung der Masken besteht in ihrer Wiederverwendbarkeit in der Implementierungsphase.

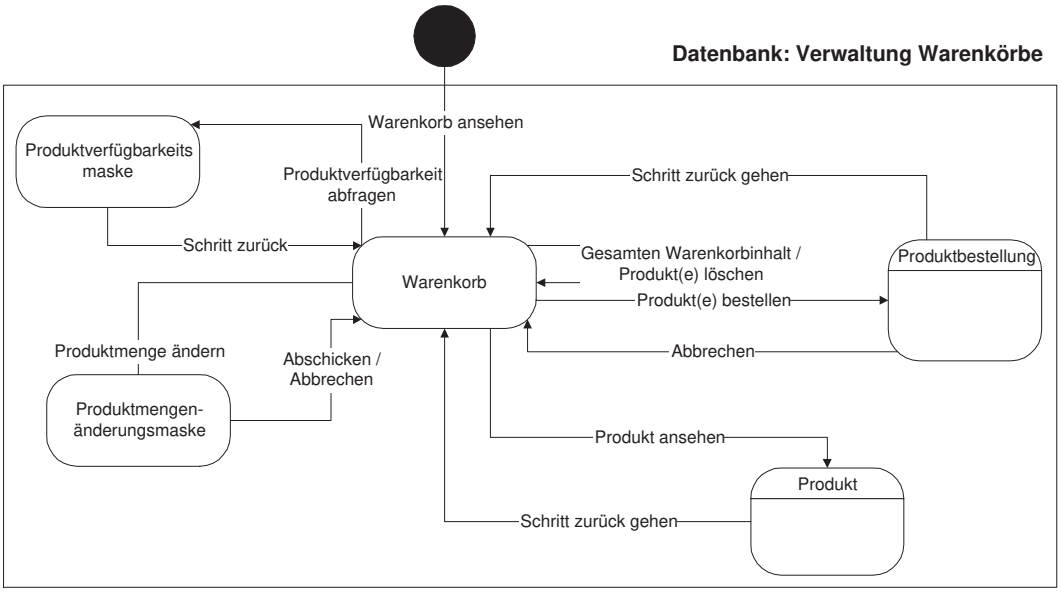

Abbildung 6.2: Kunden-Zustandsdiagramm der Datenbank „Verwaltung Warenkörbe“

In Abbildung 6.2 ist als Beispiel das Zustandsdiagramm für die Interaktion des Kunden mit

dem Warenkorb des IPOS dargestellt. Dort sind alle möglichen Transitionen dargestellt, die der Kunde im Warenkorb betätigen kann, um in Folgezustände zu gelangen. Folgezustände können Vergröberungen von Zustandsdiagrammen sein wie z.B. der Zustand „Produkt“.

6.2.2 Entwurfs-Klassendiagramm

In Abbildung 6.3 ist das in der Entwurfsphase erstellte Klassendiagramm dargestellt, daß anhand der Ergebnisse der Analysephase erstellt wurde. In diesem finden sich die Akteure und die in den Anwendungsfällen benutzten Objekte wieder. Zum Beispiel erben die drei Akteure, wobei die anonymen und registrierten Kunden wieder zusammengefaßt sind, von der Klasse `Akteur`. Die Subklasse `Customer` hat mehrere Beziehungen zu anderen Klassen wie `Shopping Cart` und `Favorites`, die die Zusammenhänge im IPOS wiederspiegeln.

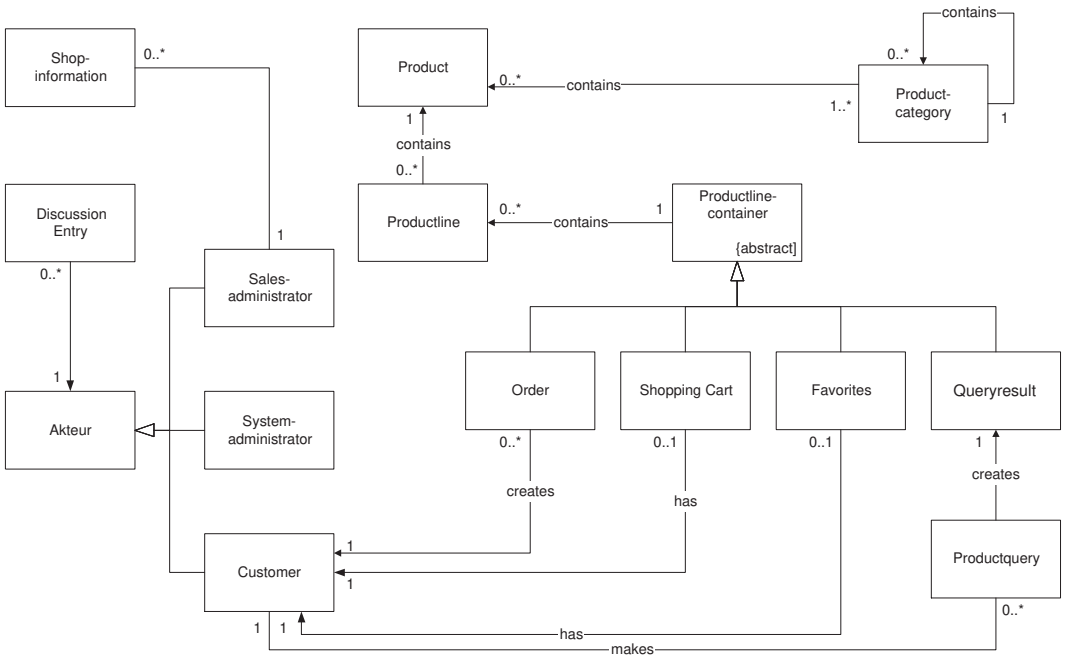

Abbildung 6.3: Entwurfs-Klassendiagramm des IPOS

6.3 Implementierung

In diesem Abschnitt wird die Implementierungsphase beschrieben. In der Implementierungsphase mußten die Einschränkungen und Möglichkeiten von *Notes* berücksichtigt werden. Aufgrund der Betrachtungen in Kapitel 3 wurde das IPOS mit Datenbanken implementiert. Die erstellten Datenbanken sind im Paketdiagramm in Abbildung 6.4 dargestellt.

Die Beziehungen der Dokumente der Datenbanken untereinander sind im Paketdiagramm

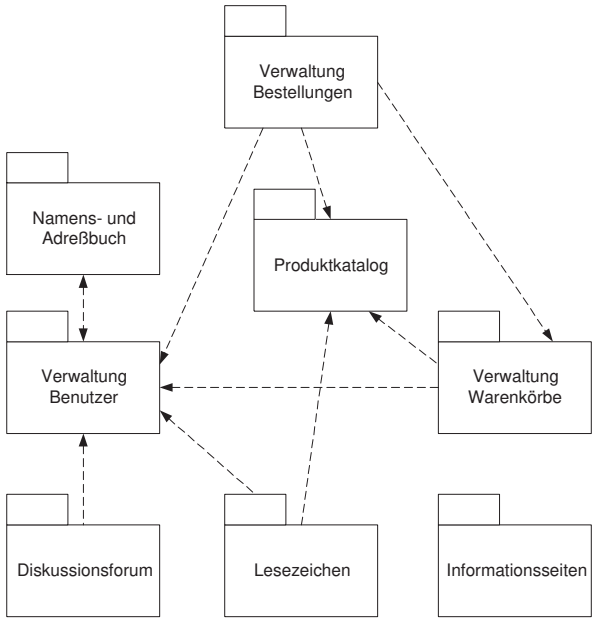

Abbildung 6.4: Paketdiagramm des IPOS

mit Klassen in Abbildung 6.5 dargestellt[1]. Dort sind die Einschränkungen von *Notes* gut zu erkennen, so daß z.B. keine Vererbung verwendet werden kann. In den folgenden Abschnitten wird zuerst die Implementierung der Geschäftslogik mit Hilfe der NBC aus Kapitel 5 und anschließend die Administrationslogik vorgestellt.

6.3.1 Geschäftslogik und -schnittstelle

Die Geschäftslogik und ihre Schnittstelle wurden unter Verwendung der NBC implementiert. Für die Geschäftslogik wurden Anfrage- und Dialogspezifikationen in der NBC-DB erstellt. Außerdem wurden an Anfragen gebundene Regeln erstellt, die die Anforderungen an die Geschäftslogik implementieren. In der Tabelle 6.1 ist die Anzahl der erstellten Anfrage- und Dialogspezifikationen dargestellt.

Die Schnittstelle zur Geschäftslogik wird durch den generischen Kunden der NBC realisiert, der dem Kunden die Kommunikation mit dem IPOS anhand generierter korrekter Konversationsspezifikation erlaubt.

	Anzahl
Anfragen	61
Dialoge	30

Tabelle 6.1: Anzahl der Anfrage- und Dialogspezifikationen

[1] Die Erstellung des Diagramms erfolgte anhand der Betrachtungen aus Abschniit 4.3.

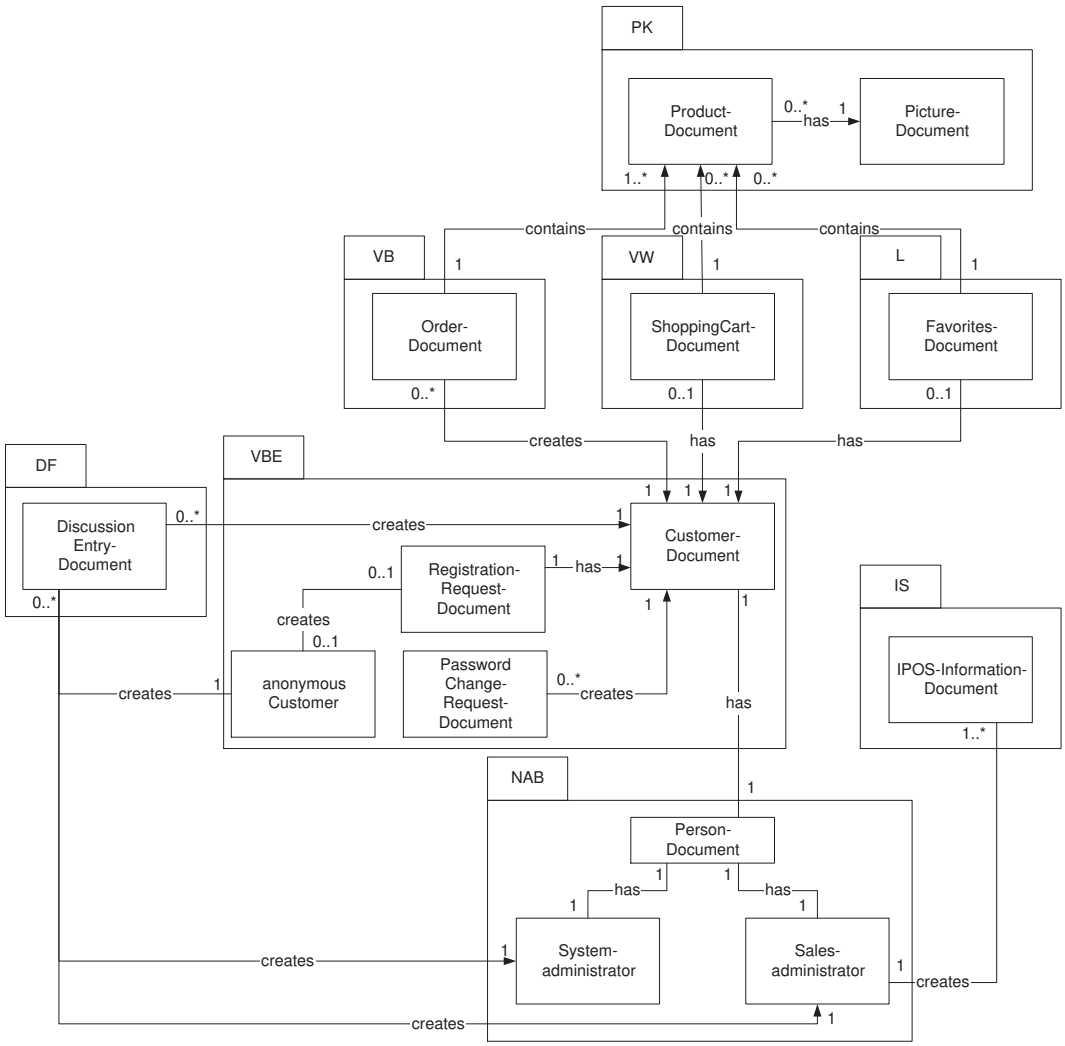

Abbildung 6.5: Paketdiagramm mit Klassen

6.3.2 Administrationslogik und -schnittstelle

Die Administrationslogik und -schnittstelle wurde durch die Erweiterung der Datenbanken des IPOS realisiert. Navigatoren, Masken und Ansichten wurden implementiert, die mit dem *Notes*-Client verwendet werden. Die konkrete Administrationslogik ist in den Agenten und Masken der Datenbanken implementiert, die durch die Schnittstelle aufgerufen werden können. Die Datenbanken für die Administrationsmöglichkeiten erstellt wurden, sind folgende:

- Verwaltung der Benutzer
- Verwaltung der Bestellungen
- Verwaltung der Warenkörbe

- Diskussionsforum
- Produktkatalog

Als weitere Administrationsschnittstelle für die Administration der Geschäftslogik wird die Administrationsschnittstelle der NBC-DB benutzt, die bereits in Abschnitt 5.2.4 beschrieben wurde.

In Tabelle 6.2 ist die Anzahl der erstellten Agenten, Ansichten, Masken und Navigatoren des IPOS dargestellt.

	Anzahl
Agenten	8
Ansichten	22
Masken	10
Navigatoren	3

Tabelle 6.2: Anzahl der Agenten, Ansichten, Masken und Navigatoren

6.4 Erweiterbarkeit und Integration externer Systeme

Die Möglichkeiten der Geschäftslogik zur Erweiterung und Integration externer Systeme werden durch die Verwendung der NBC unterstützt.

Die Erweiterung des Systems kann einfach durch das Einfügen von Dialogen und Anfragen in die Konversationsspezifikationen der Geschäftslogik geschehen. Für die Dialoge müssen Masken und Dokumente angelegt werden, die die Inhaltsspezifikationen und -instanzen darstellen. Diese können aber aufgrund ihrer Unabhängigkeit von den NBC aus existierenden Anwendungen stammen.

Die Integration externer Systeme wird durch die Verwendung von Subkonversationen unterstützt, die in vorhandene Konversationsspezifikationen eingehängt werden können. Die konkrete Anbindung an das Fremdsystem geschieht aber immer in den implementierten Regeln, so daß Subkonversationen nicht benutzt werden müssen. Sie sichern aber eine bessere Übersicht über das Gesamtsystem. Eine notwendige Vorraussetzung für eine Integration ist eine existierende Schnittstelle zu dem externen System, das integriert werden soll.

6.5 Implementierung der Integration des *SAP R/3*-Systems

Das *R/3*-System der Firma *SAP* ist, wie bereits in der Einleitung gesagt, eine betriebswirtschaftliche Standard-Software. Die Integration des *R/3*-System, das in der Architektur des IPOS ein Beispiel für ein *legacy*-System ist, ist aufgrund der angebotenen betriebswirtschaftlichen Funktionalität interessant. Die Anforderungen an die Integration aus Abschnitt 2.4 müssen nun durch eine geeignete Implementierung erreicht werden. Für weitere Informationen über das Thema der Integration mit dem *R/3*-System oder allgemeine Informationen sei

auf die Literatur wie [RF98; SAP98] und die Diplomarbeit [JR99] verwiesen. Die Diplomarbeit beschreibt die Realisierung einer objektorientierten Schnittstelle zu dem *R/3*-System.

Die Implementation der Integration des IPOS mit dem *R/3*-System ist mit folgenden Komponenten realisiert worden:

1. *LotusScript* Extension für das *R/3*-System
2. *Business Framework* des *R/3*-Systems
3. *LotusScript*-Klasse `SAP_ECOMMERCE`
4. Subkonversationen
5. Agenten

Die Vorraussetzung für die Integration ist die existierende Schnittstelle die *LotusScript* Extension für das *R/3*-System, die in Abschnitt 3.6 beschrieben ist. Für die Benutzung der Schnittstelle wurde eine Script-Bibliothek „SAPAccess“ mit der Klasse `SAP_ECOMMERCE` erstellt, die Methoden für den Zugriff auf das *R/3*-System zur Verfügung stellt. Die Implementierung der Methoden fand, soweit es möglich war, unter Verwendung des *Business Framework* des *R/3*-Systems statt.

Mit dem *Business Framework* definiert die *SAP* ihre *R/3*-Produktarchitektur. Speziell war für die Arbeit die Schnittstellentechnologie *Business Application Programs Interface* (BAPI) wichtig. Über die BAPI-Schnittstelle kann von außen auf die im *Business Framework* definierten *Business*-Objekte zugegriffen werden [RF98]. Dafür werden die BAPI-Funktionen verwendet, die den Zugriff auf die *Business*-Objekte erlauben.

Die Integration in die Geschäfts- und Administrationslogik wurde mit Subkonversationen implementiert, in deren Regeln und Agenten die Klasse `SAP_ECOMMERCE` für den Zugriff auf das *R/3*-System benutzt wurde. Subkonversationen wurden benutzt, da Teile der Konversationsspezifikationen einfach ausgetauscht und so andere Integrationslösungen verwendet und eingefügt werden können.

Kapitel 7

Zusammenfassung, Bewertung und Ausblick

Abschließend werden in diesem Kapitel in Abschnitt 7.1 die Ergebnisse der Arbeit zusammengefaßt und in Abschnitt 7.2 bewertet. Abschnitt 7.3 gibt danach einen Ausblick auf weitergehende Themengebiete.

7.1 Zusammenfassung

Diese Arbeit entwickelt das generische Online-Verkaufssystem *Internet Point of Sale* unter Verwendung des Modells der *Business Conversations*. Dafür wurde eine Anforderungsanalyse, ein objektorientierter Entwurf mit UML und die Realisierung mit *Lotus Notes* und dem *SAP R/3*-System vorgenommen.

Die erfolgte Analyse der vier kommerziellen Produkte *Intershop 3.01*, *Microsoft Site Server 3.0, Commerce Edition*, *Lotus Domino.Merchant Serverpack 2.0* und *iCat Electronic Commerce Suite, Professional Edition 3.01* zeigte die Architektur und Funktionsweise von Online-Verkaufssystemen auf. Alle Produkte beruhen mit geringen Abweichungen auf der gleichen Architektur. Anhand der Ergebnisse und Ziele der Arbeit wurden die Anforderungsdefinition und Architektur des IPOS definiert.

Als Basis für die Realisierung des IPOS wurden die Konzepte von *Notes* vorgestellt. Der Fokus lag dabei auf der Anwendungsentwicklung. Dafür wurden die Datenbanken und ihre Komponenten, die Programmierung und das Sicherheitskonzept erläutert, da diese die Grundlage für die Anwendungsentwicklung sind. Um die Einordnung in existierende Konzepte zu ermöglichen, wurden die Datenbanken und ihre Komponenten mit dem relationalen Modell verglichen. Außerdem wurden die für das IPOS wichtigen Aspekte der Integrationsmöglichkeiten für das Internet und das *SAP R/3*-System erläutert.

Ergänzend zur Anforderungsdefinition wurden noch weitere Anforderungen und Entscheidungen für den Systementwicklungsprozeß beschrieben wie das verwendete Vorgehensmodell für die Systementwicklung, die Umsetzung des objektorientierten Entwurfs in *Notes* und Anforderungen an die Umsetzung der *Business Conversations*.

Der objektorientierte Entwurf und die Implementierung der *Notes Business Conversations* wurden vorgenommen, um der Anforderung der Verwendung des Modells der *Business Conversations* gerecht zu werden. Die Hauptbestandteile der NBC sind die NBC-Datenbank, in der die Konversationsspezifikationen, -instanzen und das *Framework* für die Kommunikation implementiert sind und der generische Kunde, der für den menschlichen Kunden eine Konversationsspezifikation erzeugt, anhand derer legale Konversationen geführt werden können.

Auf der Grundlage der oben genannten Punkte wurde der objektorientierte Entwurf und die Implementierung des IPOS vorgenommen. Diese berücksichtigten die Anforderungen, die an das System gestellt wurden.

7.2 Bewertung

Die Bewertung der Ergebnisse der Arbeit untergliedert sich in drei Bereiche:

Eignung von Notes für die Realisierung von Internet-Informationssystemen:

Die Eignung von *Notes* als Plattform für Internet-Informationssysteme ist aufgrund der Erweiterungen und Öffnung bezüglich des Internets als gut zu bezeichnen. Systeme sind ohne Aufwand schnell im *Notes*- und HTTP-Client verfügbar, was eine parallele Entwicklung unnötig macht. Besonderheiten und Unterschiede zwischen dem HTML- und *Notes*-Client existieren, können aber leicht in einem System vereinigt werden, wie das IPOS zeigt. Weiterhin müssen keine HTML-Seiten erstellt werden, da die Masken von *Notes* in HTML-Seiten konvertiert werden. Zusätzlich wird die neue Version 5.0 von *Notes* viele der Standards des Internets nativ unterstützen. So kann z.B. *Java* und *JavaScript* verwendet werden.

Nachteilig ist die Einschränkung auf die Möglichkeiten von *Notes*, da diese nur mit guten Kenntnissen von *Notes* erweiterbar sind. Zum Beispiel wäre es wünschenswert, das HTML-Formular bevor es zum HTTP-Client geschickt wird, modifizieren zu können.

Einsatz von UML für den Entwurf von Systemen für Notes:

Die Diagramme der UML-Notation sind ein guter Ansatz um *Notes*-Anwendungen zu entwickeln, wobei aber aufgrund der nicht durchgängigen Objektorientierung Einschränkungen gemäß der Umsetzung der Diagramme bestehen. Als positive Punkte sollten hierbei die gute Abbildbarkeit der Anwendungsfälle auf die Benutzerrechte und Benutzerverwaltung, sowie die Umsetzung der Zustände der Zustandsdiagramme als Masken und die Verwendung der Pakete für die Datenbanken genannt werden. Eingeschränkt nutzbar ist das Klassendiagramm, da z.B. N:M-Beziehungen zwischen Dokumenten nur über Feldinhalte realisiert werden können und Vererbung nur indirekt nachbildbar ist. Abschließend kann die Verwendung der UML-Notation für *Notes* empfohlen werden, da die UML-Diagramme die Analyse, den Entwurf und die Implementierung gut unterstützt haben und gleichzeitig eine Dokumentation für das System darstellten.

Notes Business Conversations:

Die Verwendung der NBC für Internet-Informationssysteme hat Vor- und Nachteile, die nachfolgend beschrieben werden. Vorteile sind:

1. Die NBC sind unabhängig und können für jedes beliebige System wiederverwendet werden.
2. Die Administrationsschnittstelle bietet eine gute Übersicht über das System und erlaubt die dynamische Konfiguration des Systems. Außerdem wird die Pflege und Wartung des Systems gut unterstützt, da die relevanten Informationen zentral in der NBC-DB gespeichert sind.
3. Ein großer Vorteil ist die resultierende Kodereduktion, da das `Agentenskelett` bereits viele Funktionen zur Verfügung stellt.
4. Durch den Einsatz des generischen Kunde entfällt die Implementierung einer Benutzerschnittstelle für die Benutzer des Systems.

Nachteile sind:

1. Die Performanz der Systeme leidet unter der Verwendung der NBC, da für jede Anfrage das *Framework* `Agentenskelett` benutzt wird. Zusätzlich vermindert die dynamische Generierung des generischen Kunden die Performanz. Dieser Nachteil kann aber durch den Einsatz von dementsprechend dimensionierter *Hardware* ausgeglichen werden.
2. Die Verwendung der NBC erfordert zusätzlichen Aufwand für die Administration der NBC-DB.
3. Aufgrund der Einschränkungen des Modells der *Business Conversations* kann das System nie zwei Dialoge parallel öffnen.

Der Vorteil der Kodereduktion soll anhand der Aufwände der Realisierung der NBC und des IPOS weiter erläutert werden. Betrachtet werden die Masken und Programmzeilen der *Formelsprache*, HTML-Befehle und *LotusScript* (LS), die für die Realisierung notwendig waren[1]. Die HTML-Befehle implementieren die Verweise aus dem Internet auf die nächsten Dialoge, die zur Nachbildung des generischen Kunden notwendig sind. Der Aufwand für die Implementierung der NBC ist in Tabelle 7.1 dargestellt. In Tabelle 7.2 sind die Aufwände für das IPOS mit und ohne Verwendung der NBC dargestellt, woran zu erkennen ist, daß der Mehraufwand der NBC durch die Reduktion des Kodes des IPOS ausgeglichen wird.

Masken	Formelsprache	LS
10	50	1350

Tabelle 7.1: Aufwand für die *Notes Business Conversations*

Die Eignung der NBC für Internet-Informationssysteme kann als gut bewertet werden, da die für die Systeme daraus resultierende Generik und Dynamik sehr hilfreich ist. Ein weiterer Vorteil ist die Administration der NBC-DB, die eine gute Übersicht für die Systeme bietet. Die Einbußen der Geschwindigkeit durch die NBC wird die Entwicklung der *Hardware* und Netzwerktechnik ausgleichen. Der einzige Wermutstropfen ist die fehlende Berücksichtigung von parallel geöffneten Dialogen.

[1] Eine Betrachtung aller verwendeten Komponenten würde unübersichtlich werden.

IPOS	Masken	HTML	Formelsprache	LS-Agenten	LS-Regeln	LS-SAP
Mit NBC	23	0	40	0	1150	650
Ohne NBC	23	450	40	2000	0	650

Tabelle 7.2: Gegenüberstellung des Aufwandes

Internet Point of Sale:

Das realisierte Online-Verkaufssystem IPOS erfüllt die gestellten Anforderungen. Generisch ist das IPOS aufgrund der Verwendung der NBC, indem die Benutzeroberfläche im HTTP-Client vollständig generiert wird und dynamisch über die NBC-DB änderbar ist. Vorteilhaft ist auch die Möglichkeit der Verwaltung der Konversationsspezifikationen in der NBC-DB, die eine dynamische Modifizierung der Ablauflogik erlaubt. Außerdem erlaubt die Integration des *SAP R/3*-Systems, die Vorteile der heterogenen Systemumgebung zu nutzen.

Nachteilig ist die bisher fehlende Administration auf der Basis der NBC, so daß diese schwieriger zu modifizieren ist. Die Verwendung der NBC für die Administration bedeutet aber vorraussichtlich keinen großen Aufwand und würde dem System eine hohe Flexibilität geben.

7.3 Ausblick

Weitergehende Themengebiete können in drei Teilgebiete unterschieden werden:

- *Notes*
- *Notes Business Conversations*
- Online-Verkaufssysteme

Die Möglichkeiten für das erste Teilgebiet **Notes** begründen sich in der neuen Version 5.0. Mit der Version 5.0 eröffnen sich neue Möglichkeiten in der Verwendung von *Notes*. So wird die standardisierte Programmiersprache *Java* dort nativ unterstützt, wo das proprietäre *LotusScript* verwendet werden kann. Hinzu kommt die Umstellung auf die Standards im Internet wie HTTP, POP3, SMTP, SSL, usw. . Ein weiterer Vorteil ist die größere Skalierbarkeit, da andere HTTP-Server außer dem *Notes* HTTP-Server benutzt werden können. Alleine oder in Verbindung mit den NBC können weitere Untersuchungen Möglichkeiten und Grenzen von *Notes* aufzeigen.

Der Schwerpunkt für weitergehende Themengebiete sind die **Notes Business Conversations**. Für die Weiterentwicklung der NBC sind folgende Punkte möglich:

- Mit der Version 5.0 von *Notes* können die in der proprietären Programmiersprache *LotusScript* implementierten Programmfragmente durch die standardisierte Programmiersprache *Java* ersetzt werden, die neue Möglichkeiten wie Multithreading erlaubt.
- Da der generische Kunde bisher auf die Verwendung eines HTTP-Clients beschränkt ist, können im *Notes*-Client Systeme auf der Basis der NBC nicht verwendet werden. Die Erweiterung der NBC ist aber voraussichtlich ohne großen Aufwand möglich.

- Bisher ist die NBC-Datenbank auf die Rolle des Dienstleisters beschränkt und kann nur Konversationen mit dem generischen Kunden führen. Die Implementierung der Kommunikation über *e-Mail* mit anderen NBC-Datenbanken in der Rolle des Kunden oder Dienstleisters ist zwar angedacht, aber nur prototypisch realisiert. Die endgültige Realisierung der Kommunikation zwischen NBC-Datenbank und NBC-Datenbank würde weitergehende Aspekte ermöglichen wie Sekundärkonversationen.

- Ein weiterer Aspekt ist die Verwendung von Agenten für die Regelklassen. Dieses würde zu einer besseren Übersicht und zu einer dezentralen Speicherung der Regeln führen. Notwendig dafür wäre eine Realisierung über das *Notes*-API, da die notwendige Parameterübergabe an den Agenten mit den Mitteln von *Notes* nicht zu realisieren ist.

- Der generische Kunde bietet viel Raum für Verbesserungen. Die Optionen für die Visualisierung können weiter verfeinert werden, so daß der Entwickler detailliert bestimmen kann, wie der generische Kunde generiert wird.

- Analog zum generischen Kunden könnte die Funktionalität des Agentenskeletts evolutionär weiterentwickelt werden. Sinnvolle Verbesserungen sind unter anderem erweiterte inhaltliche Fehlerbehandlung (Typ- und Wertprüfung) und Sekundärkonversationen.

- Die Administrationsschnittstelle der NBC-Datenbank könnte weiter ausgebaut werden, so daß gezielte Auswertungen der historischen Konversationen möglich sind. Weiterhin wäre ein grafischer Editor hilfreich, der Dialog- und Anfragespezifikationen in Netzform darstellt. Anhand dessen können Zusammenhänge einfach erkannt werden. Diesbezüglich müßte eine sinnvolle Darstellungsform gefunden werden, damit die Darstellung nicht unübersichtlich wird. Der Editor müßte mit einer externen Anwendung realisiert werden, die über *Java* oder das *Notes*-API auf die NBC-Datenbank zugreift.

- Ein Vergleich der Implementierungen der NBC und TBC könnte vorgenommen werden, um Vor- und Nachteile der Lösungen zu evaluieren und dann entweder eine oder beide Lösungen zu verbessern.

- Abschließend wäre eine Kommunikation über *e-Mail* zwischem einen TBC-Agenten und einer NBC-Datenbank denkbar, soweit das Format der auszutauschenden *e-Mail* eindeutig geklärt ist.

Das abschließende Teilgebiet der **Online-Verkaufssysteme** ermöglicht folgende weitergehende Themen:

- Die Teilgebiete der Online-Verkaufssysteme Sicherheit und elektronischer Zahlungsverkehr bieten Raum für weitere Analysen und Untersuchungen. Speziell im Bereich des elektronischen Zahlungsverkehrs könnte eine Analyse über die zahlreichen kommerziellen Produkte vorgenommen werden.

- Da in dieser Arbeit nur die mittleren Systeme betrachtet wurden, könnten weitergehende Analysen die großen Systeme dahingehend untersuchen, mit welchen Techniken die Belastungen dieser Systeme bewältigt werden. Speziell kann die Skalierbarkeit und die ausgeglichene Belastung betrachtet werden.

Anhang A

Abkürzungsverzeichnis

API	*Application Programming Interface*
ASP	*Active Server Pages*
CGI	*Common Gateway Interface*
CORBA	*Common Object Request Broker Architecture*
CSCW	*Computer-Supported-Collaborative-Work*
DDE	*Dynamic Data Exchange*
DEM	*Data Entry Manager*
DLL	*Dynamic Link Library*
GIF	*Graphics Interchange Format*
HTML	*Hyper Text Markup Language*
HTTP	*Hyper Text Transfer Protocol*
iCat ECS	*iCat Electronic Commerce Suite, Professional Edition 3.01*
ICL	*iCat Carbo Command Language*
IDK	*Intershop Development Kit*
IMAP	*Internet Message Support Protocol*
IPOS	*Internet Point of Sale*
ISAPI	*Information Server API*
JAFMAS	*Java-based Agent Framework for MultiAgent Systems*
LAN	*Local Area Network*
LSX	*LotusScript* Extension
NBC-DB	*Notes Business Conversations*-Datenbank
NBC	*Notes Business Conversations*
NNN	*Notes-Named-Networks*
NSAPI	*Netscape Server API*
LNDS	*Lotus Notes Domino* Server
OBI	*Open Buying on the Internet*
ODBC	*Open Database Connectivity*
OLE	*Object Linking and Embedding*
OMT	*Object Modeling Technology*
OOSE	*Object-Oriented Software Engineering*
POP3	*Post Office Protocol* 3
SMTP	*Simple Mail Transfer Protocol*

SSS	*Server Side Scripting*
TBC	*Tycoon Business Conversations*
TLE	*Template Language Extension*
UML	*Unified Modelling Language*
URL	*Uniform Ressource Locator*
WAN	*Wide Area Network*
z.B.	zum Beispiel

Anhang B

Diagrammtypen der *Unified Modeling Language*

Die *Unified Modelling Language* stellt eine Notation für die objektorientierte Analyse, den Entwurf und die Implementierung dar. Entstanden ist UML aus den bis dahin dominanten objektorientierten Methoden „Booch“ von Booch, „OMT“ von Rumbaugh und „OOSE“ von Jacobson [Boo94; RBP+91; JCJÖ92]. Die Intention von UML ist, eine einheitliche Notation für viele Einsatzgebiete zu haben, wodurch alle Softwaresysteme darstellbar sein sollen [Wah98].

UML umfaßt sieben Diagrammtypen, über die nachfolgend ein Überblick gegeben wird. Dabei wird nicht auf die Details und die grafische Darstellung der einzelnen Diagramme eingegangen, da diese in der zur UML-Notation vorhandenen Literatur nachgelesen werden können [Sof97b; Sof97a; Hru98; FS97].

Anwendungsfälle: Ein Anwendungsfall beschreibt das Zusammenwirken von Aktoren mit einem System. Aktoren können Personen oder wiederum Systeme sein. Die typischen Interaktionen zwischen den Aktoren und dem System sind die Anwendungsfälle.

Statische Struktur-Diagramme: Die statischen Struktur-Diagramme sind die zentralen Bestandteile von UML, da dort die statische Struktur des Modells abgebildet wird. Hierfür gibt es zwei Arten von statischen Struktur-Diagrammen: Klassendiagramme und Objektdiagramme.

Das Klassendiagramm beschreibt die statische Struktur des Modells, das aus einer Ansammlung von deklarativen Modellkomponenten wie Klassen, Beziehungen, Objekten und Paketen besteht. Das Klassendiagramm, in dem die Komponenten als ein Graph verbunden sind, zeigt die statischen Beziehungen zwischen ihnen.

Das statische Objektdiagramm zeigt den detaillierten Zustand der Instanzen eines Klassendiagramms zu einem bestimmten Zeitpunkt. Dynamische Objektdiagramme, die den detaillierten Zustand der Instanzen eines Klassendiagramms über einem bestimmten Zeitraum zeigen, sind gleichzusetzen mit Kollaborationsdiagrammen.

Interaktionsdiagramme: Interaktionsdiagramme beschreiben das Verhalten und die zeitlichen Abläufe von Interaktionen zwischen Objekten. Hierfür stehen zwei Diagrammtypen zur Verfügung: das Sequenzdiagramm und das Kollaborationsdiagramm.

Das Sequenzdiagramm zeigt anhand einer vertikalen Zeitachse und einer horizontalen Achse, die verschiedene Objekte repräsentiert, das Verhalten und die Lebenslinie der Objekte bei einer bestimmten Interaktion. Dafür werden die Nachrichten, die im zeitlichen Verlauf von den involvierten Objekten verschickt werden und die Ereignisse, welche die Lebenslinie betreffen wie Erzeugung, Zerstörung und Aktivierung angezeigt.

Das Kollaborationsdiagramm zeigt die statische Struktur zwischen den Objekten, anhand eines Objektdiagramms und den Nachrichtenfluß zwischen den Objekten. Da das Kollaborationsdiagramm keine Zeitachse hat, werden die Nachrichten mit Sequenznummern versehen, um den zeitlichen Ablauf nachvollziehen zu können.

Paket-Diagramme: Das Paket-Diagramm erlaubt die Aufsplittung eines großen Systems. Hierzu können mehrere Klassen zu einem Paket zusammengefaßt werden, um die Übersicht über das System zu verbessern. Im Prinzip ist eine Paket eine Vergröberung der Klassendiagramme und kann Beziehungen zu anderen Paketen oder Klassen haben. Ein Paket kann Klassen und Pakete enthalten.

Zustandsdiagramme: Ein Zustandsdiagramm beschreibt die Sequenz der Zustände, die ein Objekt oder eine Interaktion aufgrund von Ereignissen während ihres Lebens durchläuft. Zustände werden hierfür mit Transitionen verbunden, welche die Ereignisse oder Anworten auf Ereignisse darstellen und zu neuen Zuständen führen.

Aktivitätsdiagramme: Das Aktivitätsdiagramm ist ein spezielles Zustandsdiagramm, in dem die Transitionen durch die Abarbeitung aller vorherigen Quellzustände ausgelöst werden. Diese Quellzustände sind Aktionszustände, die parallel abgearbeitet werden und dann die nächste Transition auslösen. Dieses Diagramm ist besonders für interne Abläufe wie Klassenmethoden gedacht, die synchron und automatisch ablaufen. Bei asynchronen Abläufen sollte das Zustandsdiagramm genommen werden.

Implementierungsdiagramm: Implementierungsdiagramme zeigen die Implementierungsaspekte bezüglich der Programmkomponenten und der Gerätekomponenten. Dabei werden die Programmkomponenten und ihre Beziehungen im Komponentendiagramm beschrieben. Das Deploymentdiagramm zeigt dann die Zusammenhänge mit den Gerätekomponenten. Dieses ist nützlich, um die Verteilung der Programmkomponenten auf verschiedene Gerätekomponenten aufzuzeigen und zu erläutern.

Anhang C

Modell der *Business Conversations*

C.1 Das Modell der *Business Conversations*

In den nun folgenden Abschnitten, die aus [Weg98] übernommen wurden, soll das Modell der *Business Conversations* ausführlich erklärt werden. In Abschnitt C.1.1 werden die Entstehung des Modells und seine Wurzeln dargestellt. Die Grundlagen des Modells werden in Abschnitt C.1.2 dargelegt; in den Abschnitten C.1.3 und C.1.4 werden die wesentlichen Aspekte der verschiedenen möglichen Kooperationsformen vorgestellt. In Abschnitt C.1.5 wird dann eine formale Definition der zu spezifizierenden Kommunikation gegeben und deren Semantik weitgehend informell beschrieben; ferner werden die dynamischen Aspekte der Kommunikation geschildert. Schließlich wird in Abschnitt C.1.6 eine Modellierung durch Petrinetze vorgestellt.

Die Darstellung lehnt sich dabei weitgehend an [Mat97a; Mat97b] sowie [Joh97; Ric97] an, der Abschnitt C.1.1 orientiert sich auch an [GHS91].

C.1.1 Leitbild

Die Sprache dient nicht nur dazu, die Welt zu beschreiben. Diese Aussage enthält eine jener Wahrheiten, von denen WITTGENSTEIN sagt, daß sie „dem Bemerktwerden nur entgehen, weil sie ständig vor unseren Augen sind“ [Wit60]. Dennoch hat erst der englische Philosoph J.L. AUSTIN diese Erkenntnis in eine Theorie umgesetzt [Aus62].

Diesem „deskriptiven Fehlschluß“ setzt AUSTIN die Erkenntnis entgegen, daß es nicht nur Äußerungen gibt, die Tatsachen *feststellen*, sondern auch solche, die Tatsache *schaffen*. Dieser Aspekt bildet die Grundlage der von ihm entwickelten und von J.R. SEARLE fortgeführten [Sea69] *Sprechakttheorie*.

AUSTIN unterscheidet dazu im ersten Teil seiner Theorie zwischen *performativen* und *konstativen* Äußerungen. Zwar revidiert und verfeinert er im folgenden diese Distinktion, da sich die Unterscheidungen zwischen wahren und falschen konstativen Aussagen einerseits und dem Glücken bzw. Nicht-Glücken performativer Aussagen nicht aufrecht erhalten ließen (zudem ließen sich keine grammatischen oder lexikographischen Kriterien für diese Klassifizierung fin-

den), als stark vereinfachte Grundlage für die Modellierung von Äußerungen innerhalb von Geschäftsprozessen ist sie aber brauchbar.

Die später von AUSTIN vorgenommene Differenzierung in *lokutionäre*, *illokutionäre* und *perlokutionäre* Sprechakte kritisiert SEARLE und schlägt eine Klassifikation in die folgenden fünf Arten von Sprechakten vor: *Assertive* Äußerungen (Behauptungen, Feststellungen, Beschreibungen), *direktive* Äußerungen (Befehle, Aufforderungen, Erlauben, Raten), *kommissive* Äußerungen (Versprechen, Ankündigungen, Drohungen), *expressive* Äußerungen (Dank, Gratulation, Entschuldigungen) sowie *deklarative* Äußerungen (Kriegserklärungen, Heiraten, Kündigungen). Nach SEARLE ist jede Äußerung dabei unmittelbar an ihre Bedeutung gekoppelt. Sprechakte werden daher als Mitteilung einer legalen Zustandsänderung ihres Sprechers innerhalb einer Konversation betrachtet.

Auf diesen Theorien aufbauend versuchten Arbeiten im Umfeld der computerunterstützten Gruppenarbeit (*computer supported cooperative work*, CSCW), Sprechakte, die im Rahmen von Geschäftsprozessen auftreten, zu identifizieren und zu klassifizieren. In [Win87] und [FGHW88] werden in dem sogenannten *language/action*-Ansatz die zwei dafür wesentlichen Arten von Sprechakten genannt: Zum einen *Conversations for Possibilities* (CfP) und zum anderen *Conversations for Actions* (CfA). Der zentraler Gedanke dabei ist, daß zwei Partner zum Zwecke der Erledigung ihres gemeinsamen Anliegens eine Reihe von Sprechakten vollführen; mit anderen Worten: eine *Konversation führen*. Eine CfA ist demnach eine Folge von Aufforderungen und Versprechen, die direkt auf die gemeinsamen Aktivitäten und Ziele der Partner gerichtet ist. Jeder Sprechakt kann im Sinne einer *performativen* Äußerung eine Aktion der Partner repräsentieren. Dabei ist innerhalb einer CfA die Reihenfolge der einzelnen Akte, also die Menge der erlaubten Gesprächstransitionen, unter den Partnern genau festgelegt. Werden hingegen in einer Konversation keine Verpflichtungen eingegangen, sondern wird z.B. die Funktion eines Managers wahrgenommen, so wird von einer CfP gesprochen. In diesem Fall werden in einer Konversation lediglich Feststellungen, Meinungen oder Erklärungen ausgetauscht und bei keinem der Partner wird dadurch eine Aktion impliziert. Eine CfP oder CfA strukturiert und koordiniert einzelne Sprechakte. Durch die Ausführung dieser Sprechakte bilden die beteiligten Kommunikationspartner eine gemeinsame Historie, vor deren Hintergrund neue Sprechakte erzeugt und interpretiert werden.

Dem Kooperationsmodell der *Business Conversations* liegt die CfA zugrunde. Dabei vereinbaren die Partner wechselseitig künftige Aktionen, die direkt an die geäußerten Sprechakte der jeweiligen Partner gekoppelt sind.

C.1.2 Grundlagen des Modells

Das eben geschilderte Leitbild bildet die Basis für folgende Annahmen:

- Das Interaktionsmuster der Sprechakte ist *universell* in dem Sinne, daß es unabhängig von der Art der Akteure (Software oder Menschen) und den von ihnen verwendeten Kommunikationsmedien und -protokollen ist.
- Die Interaktion zwischen zwei menschlichen Akteuren, zwischen einem Menschen und einem maschinellen Akteur und auch zwischen zwei maschinellen Akteuren ist innerhalb des Modells transparent.

Das bedeutet, daß die schon zuvor geforderte *Medien- und Aktorenunabhängigkeit* gegeben ist.

Das Modell der *Business Conversations* geht von der Interaktion zwischen zwei Partnern aus, von denen der eine in der Rolle des *Kunden* und der andere in der des *Dienstleisters* handelt. Die Interaktion zwischen beiden Akteuren kann in einen Zyklus mit vier Phasen unterteilt werden (siehe Abbildung C.1) und folgt stets diesem Verlauf:

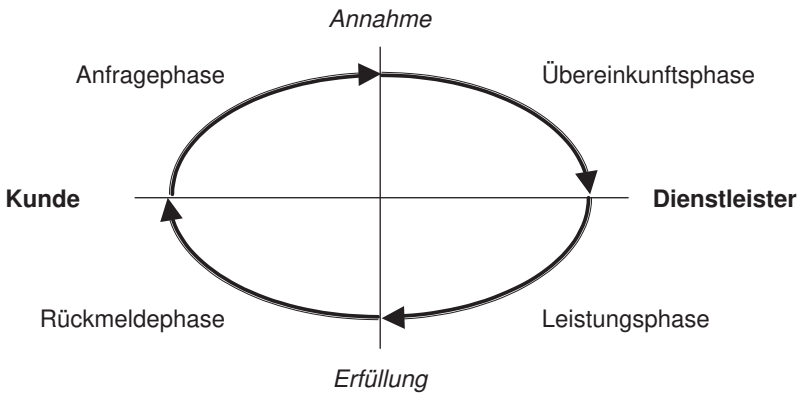

Abbildung C.1: Unterteilung der Interaktionsphasen

1. Im ersten Schritt, der *Anfragephase*, fragt der Kunde unverbindlich beim Dienstleister nach einer Dienstleistung an. Die Phase endet mit der Annahme der Anfrage. Die Initiative geht dabei immer vom Kunden aus.

2. Mit Annahme der Anfrage beginnt die zweite, die *Übereinkunftsphase*. In ihr erfolgt die Verhandlung zwischen Kunde und Dienstleister über die genauen Modalitäten des zu erbringenden Dienstes. Am Ende steht die Übereinkunft.

3. Nach der Übereinkunft beginnt die *Leistungsphase*, in der der Dienstleister die vereinbarte Leistung erbringen muß. Sie endet aus Sicht des Dienstleisters mit der Erfüllung der Leistung.

4. Die vierte Phase, die *Rückmeldephase*, dient dem Kunden dazu, dem Dienstleister mitzuteilen, ob die zuvor erbrachte Leistung seinen Anforderungen und Erwartungen entspricht.

Nur wenn alle vier Phasen erfolgreich abgeschlossen wurden, haben Kunde und Dienstleister das gemeinsame Ziel erreicht. Die Ausführung von Sprechakten wird als *Konversation* bezeichnet und bezieht sich immer auf einen konkreten Gegenstand, nämlich die Dienstleistung oder Aktion.

Der Zyklus kann natürlich durch den Verzicht auf einzelne Phasen vereinfacht werden, wenn die zugrundeliegende Beziehung zwischen Kunde und Dienstleister dies erlaubt. Es können so z.B. die Phasen der Anfrage und der Übereinkunft zusammengefaßt werden, oder die Leistungsphase kann (implizit) durch das Zustandekommen der Übereinkunft bereits erbracht

worden sein, oder es kann aus Gründen der Einfachheit auf die Rückmeldephase verzichtet werden. In allen Fällen bleibt jedoch sowohl die Reihenfolge der Phasen als auch der Zyklus als solcher erhalten.

Weiterhin steht jederzeit beiden Partnern die Möglichkeit offen, die begonnene Konversation abzubrechen, sei es durch Nichtannahme der Anfrage, Scheitern der Übereinkunft, Nichterbringen der Leistung oder weitere Gründe.

C.1.3 Modalitäten

In diesem Modell kann jedes Unternehmen, jeder Geschäftsbereich oder jeder geeignete Teil daraus als ein Akteur betrachtet werden, der gleichzeitig eine ganze Reihe von simultanen Konversationen mit anderen Akteuren wie Kunden, Lieferanten, Behörden oder anderen Teilen des Unternehmens führt. Innerhalb jeder einzelnen Konversation hat der Akteur die fest zugewiesene, nicht veränderliche Rolle entweder des Kunden oder des Dienstleisters.

Sofern eine formale Spezifikation der zu führenden Konversationen vorliegt, gestatten es die Medien- und inbesondere die Aktorenunabhängigkeit des Modells, Kooperationen in verschiedenen Modalitäten zu unterstützen.

Anwendungskoppelung: Wenn sowohl der Kunde als auch der Dienstleister als voneinander unabhängige Anwendungen (d.h. maschinelle Akteure) realisiert sind, ist die Kooperation z.B. über den Austausch von synchronen oder asynchronen Nachrichten möglich (*application linking*).

Benutzungsschnittstellen: Ein menschlicher Benutzer kommuniziert mit einem Softwaresystem, beispielsweise mit einem Informationssystem. Wenn die Interaktion formal als Konversation spezifiziert ist, kann ein generischer Dienst diese Beschreibung interpretieren und zum Aufbau einer evtl. graphischen Benutzungsschnittstelle nutzen. Durch Verwendung dieses *generischen Kunden* (*generic customer*) sind nur legale Interaktionen des Benutzers als Kunden mit dem System als Dienstleister möglich. Für den Dienstleister ist aber weder unmittelbar erkennbar noch nötig zu wissen, ob ein menschlicher oder ein maschineller Akteur sein Partner ist.

Workflow-Management: Der komplementäre Fall zum eben genannten ist, daß ein Softwaresystem in der Rolle eines Kunden Anfragen etc. an einen menschlichen Akteur sendet, der dann durch die Verwendung eines softwarebasierten *generischen Dienstleisters* die Anfragen interpretiert und ausführt. Ein Beispiel dafür wäre der Einsatz von Workflowsystemen.

Arbeitsstrukturierung: Die Zusammenarbeit von Menschen kann mit Hilfe von Softwarewerkzeugen, die die formalen Spezifikationen der Zusammenarbeit oder Kooperation kennen, durch Überprüfung der Einhaltung eben dieser Regeln unterstützt werden.

Basierend auf diesen Teilszenarien lassen sich komplexe Szenarien der Integration verschiedener vorhandener oder neu zu erstellender Dienste organisatorische Struktur von Unternehmensprozessen etc. entwerfen. Wesentliche Vorteile der relativ losen, autonomieerhaltenden Koppelung der Komponenten über *Business Conversations* sind, daß

- Teilsysteme zunächst unabhängig von der Umgebung nur auf Basis der Interaktionsmuster entworfen und realisiert werden können; wenn sie später in verteilten, kooperierenden Umgebungen eingesetzt werden sollen, ist es nicht nötig, die Anwendungslogik zu modifizieren,

- die Integration vorhandener Altsysteme durch Kapselung in sogenannte *wrapper*, die die Konversationen im obigen Sinne auf die Schnittstellen des Altsystems abbilden, leicht zu bewerkstelligen ist; dazu muß der Quellcode der Anwendungen oft nicht einmal vorhanden sein und

- die Modalität einzelner Interaktionsbeziehungen ohne Verlust entweder statisch durch Umorganisation der Systeme oder Prozesse oder sogar dynamisch während der Konversation geändert werden kann.

C.1.4 Abstraktionen und Verfeinerungen

Gemäß den schon angesprochenen Strukturierungsprinzipien kann ein durch einen Akteur angebotener einzelner Dienst von außen als einheitlicher Dienst erscheinen, während der Akteur tatsächlich nur interne Konversationen zwischen weiteren Akteuren anstößt, die als Gemeinschaft den Dienst erbringen. Beispiele wären das Zusammenwirken von teilweise unabhängigen Geschäftsbereichen eines Unternehmens, oder die Gliederung des firmeninternen Teils eines Geschäftsprozesses nach den beteiligten Personen oder Rollen.

Die erste Konversation wird dabei als die *primäre*, alle weiteren, von dem ursprünglich angesprochenen Akteur initiierten, als die *sekundären* Konversationen bezeichnet. Sekundäre Konversationen folgen dabei genau demselben Schema wie primäre. Dadurch ist es möglich, daß ein Akteur zur selben Zeit in der Rolle des Dienstleisters und des Kunden agiert, allerdings in unterschiedlichen Konversationen.

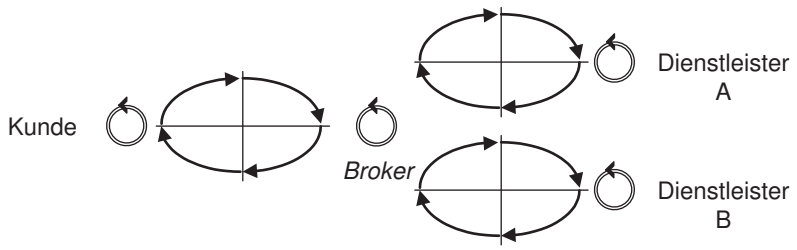

Abbildung C.2: Delegation von Konversationen durch einen *Broker*

Wie in Abbildung C.2 dargestellt, kann ein Akteur als *Broker* eine (primäre) Konversation in der Rolle des Dienstleisters mit einem Kunden führen und dazu gleichzeitig zur Erledigung seines Auftrages in der Rolle des Kunden mehrere sekundäre Konversationen mit weiteren Akteuren führen. In diesem Fall kann man auch von *Delegation* sprechen, denn der *Broker* muß zur Erfüllung seiner Aufgabe die primäre Konversation zunächst aufschieben, um untergeordneten Stellen Anfragen stellen zu können, deren erfolgreiche Ausführung erst die Ausführung der primären ermöglicht.

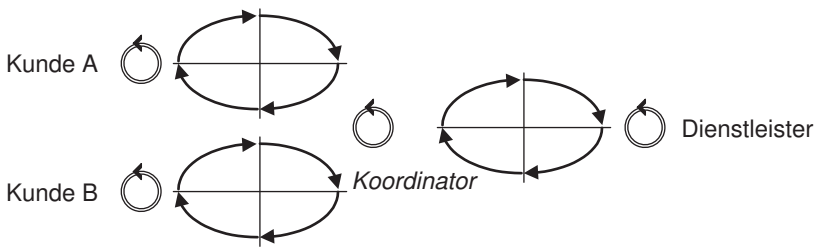

Abbildung C.3: Koordinierung von Konversationen durch einen Koordinator

Andererseits kann ebenso ein Akteur als Dienstleister für mehrere Kunden fungieren (also mehrere primäre Konversationen führen) und gleichzeitig Kunde in einer sekundären Konversation sein (siehe Abbildung C.3). Dies ist der Fall der *Koordinierung* der Arbeit mehrerer Kunden in Bezug auf einen Dienstleister.

Zu unterscheiden sind synchrone von asynchronen sekundären Konversationen. Synchrone sekundäre Konversationen unterbrechen die initiierende primäre Konversation, während asynchrone parallel zur primären ablaufen.

Zusätzlich sind natürlich noch komplexere Verfeinerungen denkbar, in denen ein Akteur gleichzeitig mehrere primäre und mehrere sekundäre Konversationen führt und synchrone und asynchrone gemischt auftreten. Dabei kann die erfolgreiche Beendigung einer neuen asynchronen sekundären Konversation die Vorbedingung für einen Schritt der parallel ablaufenden primären sein, so daß sich insgesamt Abhängigkeiten auch zwischen asynchronen Konversationen bilden. Eine Modellierung mit Hilfe von Petrinetzen bietet sich dafür an, ist aber nicht Teil des Modells der *Business Conversations*.

In den Darstellungen von [Joh97; Ric97] wird außerdem die *Subkonversation* als Mittel zur Strukturierung von Konversationen eingeführt: Eine Konversationsspezifikation darf anstelle eines einzelnen Gesprächsschrittes eine weitere, ganze Konversationsspezifikation als *Subkonversation* referenzieren, die an dieser Stelle den weiteren Ablauf festlegt. Dadurch soll die Wiederverwendung von Spezifikationen erleichtert werden. Die operationale Semantik dieses Konzepts wird allerdings nicht klar dargestellt, zumal es auch in der begleitenden Implementierung nicht realisiert wurde. Die Technik der Subkonversation ähnelt der der synchronen sekundären Konversationen stark. Der Unterschied liegt in der expliziten Modellierung der Subkonversation gegenüber der transparent für den Partner ablaufenden sekundären Konversation.

Die Nutzung von Subkonversationen verspricht die inkrementelle und auf der Wiederverwendung von bereits bestehenden Diensten basierende Entwicklung von neuen und die Weiterentwicklung von alten Diensten zu verbessern. Damit entspricht sie der Forderung nach Unterstützung der Evolution von kooperativen Informationssystemen.

Die genannten technischen Prinzipien lassen sich auch zur semantischen Strukturierung und Modellierung eines nach außen hin geschlossen erscheinenden Systems nutzen: Tatsächlich können mehrere Akteure in gleichberechtigten Positionen ihre Arbeit aufteilen, oder sie können eine hierarchische Struktur in der Art von Vorgesetzten und Untergebenen unter Verwendung von Befehlsketten bilden.

In diesem Sinne kann man die Fähigkeit, an einer solchen Konversation teilzunehmen, als die charakteristische Fähigkeit eines *Agenten* sehen [Mat97a]. Wenn man dies tut, kann man weiterhin sowohl ein ganzes Unternehmen, einzelne Geschäftsbereiche daraus, Softwaresysteme oder einzelne Personen bzw. Rollen, die sie innehaben, als Spezialisierung des generellen Konzepts des Akteurs betrachten. So gesehen, liefert die wiederholte Dekomposition der Kunde-Dienstleister-Beziehungen eines großen Systems oder eines Unternehmens zuletzt eine Ansammlung von autonomen Agenten, die menschliche oder maschinelle Akteure repräsentieren und nur noch durch ein Geflecht von Konversationsbeziehungen verbunden sind. Die sich damit eröffnenden Möglichkeiten der Systemanalyse und des Entwurfs insbesondere von Geschäftsprozessen werden näher in [Rip98] untersucht.

C.1.5 Konversationsspezifikationen

Um eine Brücke zwischen diesen abstrakten Modellen einerseits und den softwaretechnischen Realisierungsmöglichkeiten andererseits schlagen zu können, ist eine Formalisierung des eigentlichen Inhalts der Konversationen und deren genauen Ablaufs nötig.

Wir verwenden im folgenden zwei getrennte Beschreibungsebenen für Konversationen:

- Die Ebene der *Konversationsspezifikationen*: Auf dieser Ebene wird die Struktur von möglichen Konversationen beschrieben. Sie entspricht der Typebene einer Programmiersprache. Diese Spezifikationen müssen eine Laufzeitrepräsentation besitzen.

- Die Ebene der *Konversationsinstanzen*: Dies ist die der konkreten Konversationen, die zwischen zwei konkreten Akteuren in den Rollen von Kunden und Dienstleistern stattfinden. Analog zu Programmiersprachen ist jede Konversationsinstanz Ausprägung ihres Typs, der Konversationsspezifikation.

Das Modell der *Business Conversations* sieht die Gliederung jeder Konversation in einzelne *Dialogschritte* vor, wobei nach der Eröffnung der Konversation durch eine initiale Anfrage des Kunden in jedem Schritt ein formal spezifizierter *Dialog*, den man als Abstraktion eines Formulars sehen kann, vom Dienstleister an den Kunden geschickt wird, der Kunde diesen inspiziert und den Inhalt ggf. verändert und ihn schließlich unter Angabe einer aus einer Menge von für diesen Dialog angegebenen *Anfragen* zurückschickt. Dies geschieht solange, bis beide Partner übereinkommen, daß die Konversation beendet ist. Das Prinzip wird in Abbildung C.4 deutlich; der grau hinterlegte Bereich wird dabei evtl. vielfach wiederholt.

Formal kann man diese Transitionen folgendermaßen als Funktionen notieren:

$$\begin{array}{rrcl} \text{Dienstleister:} & (d_n, r_n) & \mapsto & d_{n+1} \in r_n \\ \text{Kunde:} & d_n & \mapsto & (d'_n, r_n) \end{array}$$

Die d_i stellen dabei die Dialoge und die r_i die Anfragen dar, wobei i der Index des Dialogschritts ist. Eine Anfrage r besteht formal aus der Menge der möglichen Folgedialoge.

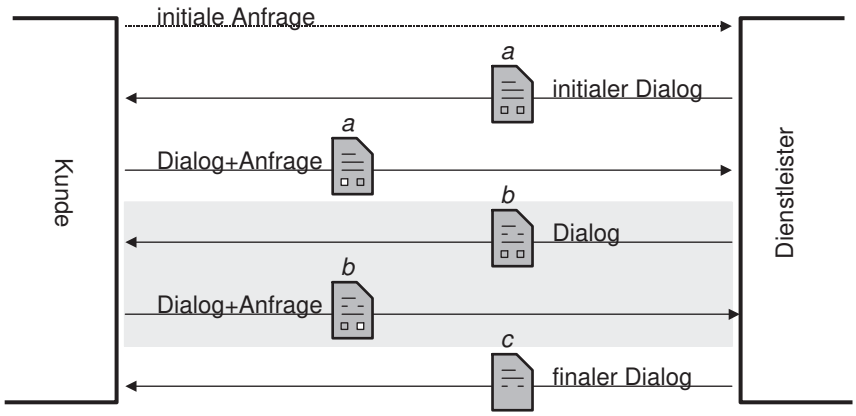

Abbildung C.4: Ablauf einer Konversation

Damit eine aktorenunabhängige Kommunikation möglich ist, bedarf es zu jeder konkreten Konversation einer abstrakten *Konversationsspezifikation*. Sie fungiert – in Analogie zu Programmiersprachen – als Typ einer konkreten Instanz einer Konversation. Diese genaue Spezifikation erst ermöglicht es den Partnern, den Kontext der Konversation zu erfassen und geeignet auf Nachrichten zu reagieren. Insbesondere sind erst dadurch maschinelle Akteure möglich.

Eine Konversationsspezifikation definiert im einzelnen folgendes:

- ihren Namen;
- alle in einer Konversation möglichen Dialoge, dazu erhält jeder Dialog einen innerhalb der Konversationsspezifikation eindeutigen Namen sowie eine formale, strukturierte Beschreibung des Inhalts durch ein simples Typsystem;
- eine Menge von Anfragen für jeden Dialog, aus der eine als Antwort des Kunden auf den Dialog (neben seinem Inhalt selbst) gefordert wird; jede Anfrage hat dazu einen innerhalb des Dialoges eindeutigen Namen;
- alle direkt auf einen Dialog als Folgedialog möglichen Dialoge, dazu wird zu jeder Anfrage, die ein Dialog enthält, die Menge der erlaubten Folgedialoge (durch Angabe der Namen) angegeben;
- den initialen Dialog, den der Dienstleister als ersten Dialog direkt auf die initiale Anfrage eines Kunden auf Eröffnung einer Konversation schickt.

Abgesehen von dem Typsystem für die Dialoginhalte läßt sich eine Konversationsspezifikation abstrakt als Spezifikation eines nichtdeterministischen endlichen Automaten (NFA) und der Ablauf einer Konversation als Interpretation oder Simulation desselben betrachten. Dabei bilden die Dialoge die Zustände und die Anfragen des Kunden die Eingaben des Automaten, wobei der Nichtdeterminismus des Automaten durch die Möglichkeit mehrerer Folgedialoge eines Dialoges auf eine Anfrage zum Ausdruck kommt (siehe Abbildung C.5). Die

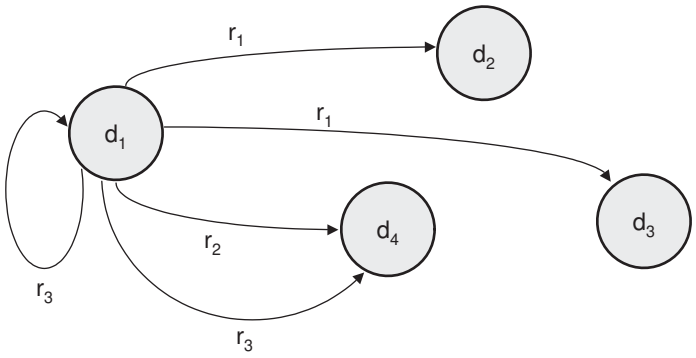

Abbildung C.5: Konversationsspezifikation als nichtdeterministischer endlicher Automat

Zustandsübergänge werden also im Rahmen der Spezifikation durch die Auswahl der Anfrage durch den Kunden und die Auswahl des Folgedialoges durch den Dienstleister festgelegt.

Überdies existieren in jeder Konversationsspezifikation zwei weitere Anfragen, die nicht Teil der expliziten Definition sind:

1. Die *initiale Anfrage*, die von einem Kunden zwecks Eröffnung einer Konversation an einen Dienstleister gerichtet wird. Sie ist nicht Bestandteil der Spezifikation, da die Konversation, zu der die Anfrage führen soll, zum Zeitpunkt der Anfrage noch nicht existiert. Die initiale Anfrage muß eine geeignete Repräsentation der Spezifikation der zu eröffnenden Konversation enthalten, damit der Dienstleister überprüfen kann, ob er in der Lage ist, die Konversation zu führen. Die Art der Repräsentation bleibt der Implementierung überlassen.

2. Die spezielle Anfrage nach dem *Abbruch der Konversation*. Sie ist implizit Bestandteil jedes Dialogs und kann deshalb jederzeit vom Kunden gestellt werden. Mit ihrer Hilfe lassen sich nicht behebbare Fehler sowohl in der Anwendung als auch solche des Systems signalisieren. Das System könnte beispielsweise auch nach Zeitüberschreitungen (*timeout*) bei der Antwort eines Partner die Konversation abbrechen und dies damit signalisieren. Auch Kunde oder Dienstleister können damit den Abbruch der Konversation erzwingen. Der Implementierung des Systems bleibt die Spezifikation eines speziellen *Fehlerdialoges* vorbehalten, der die Gründe für das Scheitern der Konversation beschreibt und im Falle des Absendens der Anfrage nach Abbruch statt des eigentlich erwarteten Dialoges vom Kunden mitgesendet bzw. vom Dienstleister als Zeichen des Abbruchs der Konversation verschickt wird.

Eine besondere Rolle spielen weiterhin die *finalen Dialoge*. Definitionsgemäß sind dies die Dialoge, die eine leere Menge von Anfragen (also keine Anfragen) enthalten. Das Fehlen von Anfragen bedeutet, daß es keine Folgedialoge gibt, also die Konversation beendet ist, wenn ein solcher Dialog versendet wird. Jede Konversationsspezifikation kann keinen, einen oder mehrere finale Dialoge enthalten. Wenn es keinen gibt, wird dadurch eine unendliche Konversation modelliert. Mehrere finale Dialoge stellen verschiedene Endzustände der Konversation dar. Der oben genannte Fehlerdialog stellt einen speziellen finalen Dialog dar.

Der finale Ablauf ist folgender: Sendet der Dienstleister einen finalen Dialog an den Kunden, so ist für den Dienstleister in genau diesem Moment die Konversation beendet. Nach Erhalt des finalen Dialoges durch den Kunden und die entsprechende Auswertung seinerseits ist die Konversation auch für ihn beendet.

Der *initiale Dialog* als ein weiterer ausgezeichneter Dialog ist von der Anforderung an die Struktur her nicht anders als andere normale Dialoge, kann also auch final sein (womit die Konversation sofort nach Beginn wieder beendet wäre).

Die Spezifikation des Inhalts eines Dialoges wird durch ein einfaches Typsystem geleistet. Es sieht neben Basisdatentypen wie `boolean`, `integer`, `real`, `string`, `date` und `currency` zusammengesetzte Typen wie `records`, `variants`, `single-` und `multiple-choice` sowie `sequences` vor. Dabei werden alle Komponenten des Inhalts sowie alle `record-` und `variant-`Komponenten durch Namen gekennzeichnet, um den Zugriff zu ermöglichen. Die `record-` und `variant-`Typen dienen zum Aufbau von komplexeren, strukturierten Typen. Dabei sind alle Typen orthogonal kombinierbar. Die `choice-`Typen, deren Instanzen sich wertmäßig nicht von den entsprechenden Typen unterscheiden, dienen dazu, die *mögliche* Wertemenge einzuschränken. Die Wertemengen sind nicht Teil der Spezifikation, sondern werden erst zur Laufzeit den Instanzen zugewiesen. Der Gedanke ist dabei, mit der Bereitstellung solcher Typen auch die Visualisierung von Datenstrukturen zu unterstützen, etwa (je nach Kardinalität) durch *checkboxes, radiobuttons* oder geeignete Auswahllisten. Der `sequence-`Typ dient zur Definition und Benutzung von Massendaten.

Der Aufbau einer vollständigen Konversationsspezifikation läßt sich durch Angabe einer Grammatik formalisieren. Wir geben dazu hier eine solche in EBNF an:

Conversation	:=	**CONVERSATION** Name **WITH** Dialog {Dialog};
Dialog	:=	**DIALOG** DialogName **WITH** {Request} [Content] **END**;
Request	:=	**REQUEST** RequestName **TO** DialogName {DialogName} **END**;
Content	:=	**CONTENT** NamedType {NamedType} **END**;
NamedType	:=	Name **OF** Type;
Type	:=	AtomicType | CompoundType | SpecType;
SpecType	:=	**CONVSPEC** | **DIALOGSPEC** | **CONTENTSPEC**;
AtomicType	:=	**BOOL** | **INT** | **REAL** | **STRING** | **DATE** | **CURRENCY**;
CompoundType	:=	**RECORD** {NamedType} **END** |
		VARIANT {NamedType} **END** |
		MULTIPLE_CHOICE OF Type |
		SINGLE_CHOICE OF Type |
		SEQUENCE OF Type;

Konkrete Konversationsspezifikationen müssen selbstverständlich nicht in einer der obigen EBNF entsprechenden textuellen Form angegeben werden; diese EBNF dient nur dazu, die Struktur als solche zweifelsfrei festzulegen. Die Art der Konstruktion von Spezifikationen und deren Instanziierung für konkrete Konversationen bleiben den konkreten Mechanismen der Implementierung überlassen.

Die zuvor bereits genannte Forderung nach Vorhandensein einer Laufzeitrepräsentation der

Konversationsspezifikation ist nicht nur für die initiale Anfrage nötig, damit die Partner die Spezifikation inspizieren können, sondern kann auch zur Entwicklung von Meta-Diensten verwendet werden, die auf Spezifikationen arbeiten: Denkbar sind graphische Editoren, generische Visualisierungswerkzeuge, Modellverifikatoren, Spezifikationsmakler etc. Ein wichtiger Nutzen ist auch die Überprüfbarkeit von Konversationen zur Ausführungszeit, um eine gesicherte maschinelle Abwicklung der Konversation zu gewährleisten.

Durch die Einführung der Typen `CONVSPEC`, `DIALOGSPEC` und `CONTENTSPEC` sind alle Spezifikationen selbst Objekte erster Klasse in ihrem eigenen Typsystem, nicht nur im Typsystem der Implementierungssprache. Dadurch können Spezifikationen wiederum Inhalte sein, die innerhalb von Konversationen verwendet werden können. So wird es besonders einfach, die oben angesprochenen Meta-Dienste zu realisieren, wenn diese selbst *Business Conversations* zur Interaktion verwenden.

C.1.6 Modellierung durch Petrinetze

Abschließend soll hier kurz eine weitere interessante Möglichkeit für die Darstellung des Ablaufes der Interaktion nach der *language/action*-Perspektive gegeben werden, die [Rip98] entnommen ist.

Dabei werden, wie Abbildung C.6 zu entnehmen ist, der Kunde, der Dienstleister und der Konversationszyklus in einem gemeinsamen Petrinetz beschrieben, das den Ablauf durch geeignete Vorbedingungen einschränkt. Der stark durchgezogene, graue Kreis symbolisiert dabei lediglich den Konversationszyklus; er hat keine Bedeutung für das Netz.

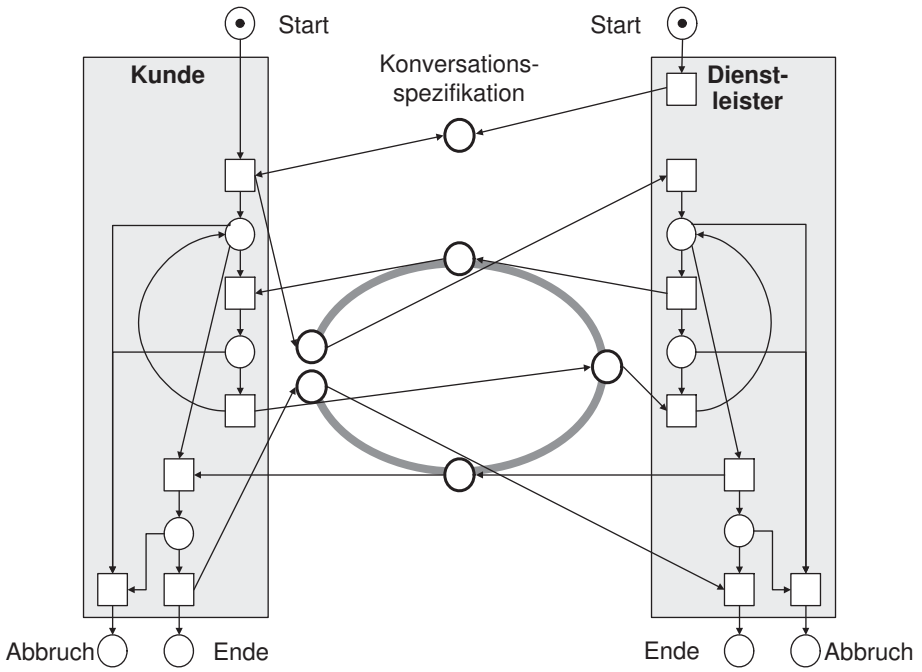

Abbildung C.6: Petrinetz zur Darstellung der *language/action*-Perspektive

Die markierten Stellen stellen den Ausgangszustand dar. Die rechte modelliert die Bereitstellung der Konversationsspezifikation durch den Dienstleister. Wenn dies geschehen ist, kann ein Kunde den Dienst in Anspruch nehmen. Dabei können auch mehrere Kunden gleichzeitig agieren, dies wäre durch mehrere Instanzen der linken Seite zu modellieren. Die Steuerung der Zyklus geschieht durch die in der Mitte angeordneten Stellen. Sie sind Ein- und Ausgangsstellen der entsprechenden Transitionen, die die Schritte von Kunde und Dienstleister darstellen. Gut zu erkennen ist auch die Möglichkeit, eine begonnene Konversation jederzeit abzubrechen. Dazu dienen die Transitionen, die zu den mit „Abbruch“bezeichneten Stellen führen.

Dieses Netz kann als Grundlage für die Modellierung der *Business Conversation* in Workflow-Systemen dienen, die Gebrauch von Petrinetzen machen.

Anhang D

Glossar und Kataloge der Anwendungsfälle des IPOS

Begriff	Beschreibung
Anfrage	Eine Anfrage bezeichnet eine Schaltfläche oder Verweis, den der Benutzer in einem Dialog betätigen kann.
Anmeldung	Um im IPOS Bestellungen oder Warenkörbe benutzen zu können, muß der Benutzer angemeldet sein. Dafür gibt er seinen *Login*-Namen und sein Paßwort ein, daß er im Registrierungsformular festgelegt hat.
Auswertung	Mit den Auswertungen werden interne Informationen nach bestimmten Kriterien präsentiert.
Bestellung	Die Bestellung enthält eine zu bestellende Produktliste mit Preisen, Benutzerdaten, Versandadresse, Rabatte und der Endsumme.
Dialog	Ein Dialog bezeichnet eine HTML-Seite, die ein Kunde benutzt.
Produkt	Ein Produkt ist eine beliebige Sache, die zum Verkauf angeboten wird. Es hat einen Preis, Namen, Gewicht, Größe, Bild, und eine Beschreibung. In den Produktdialogen für den Kunden werden nur Eigenschaften mit Inhalt angezeigt.
Produktverwaltung	Die Produktverwaltung zeigt alle Produkte an und erlaubt Operationen auf den Produkten. Es können Produkte gelöscht, Bilder zugeordnet, aus dem *SAP R/3* geladen und mit dem *SAP R/3* abgeglichen werden.
SAP R/3	Das *SAP R/3* stellt ein bereits vorhandenes System des Betreibers des IPOS dar. Aus dem *SAP R/3* werden Produktinformationen, Produktverfügbarkeitsinformationen und der Status der Bestellungen genutzt. Im IPOS erstellte Bestellungen und Kunden werden automatisch in das *SAP R/3*-System eingetragen.
...	...

Tabelle D.1: Glossar

Anwendungsfall	Beschreibung
Produkt suchen	Führt eine Suche über das Produktangebot aus, welche die Produkte als Ergebnis zurückgibt, die mit den angegebenen Suchkriterien übereinstimmen. Dabei kann entweder nach Suchbegriffen gesucht oder einzelnd die Kategorien durchsucht werden.
Warenkorb ansehen	Zeigt die Produkte mit ihrer Anzahl im Warenkorb an. Die Anzahl der Produkte kann im Warenkorb beliebig verändert werden. Außerdem können selektierte oder alle Produkte aus dem Warenkorb gelöscht werden.
Produkt ansehen	Zeigt ein Produkt mit allen Eigenschaften. Außerdem kann das Produkt in den Warenkorb gelegt oder vorgemerkt werden.
Produkt bestellen	Erstellt eine Bestellung für das betreffende Produkt, die die Liefer- und Rechnungsadresse enthält.
Produktverfügbarkeit prüfen	Prüft die Verfügbarkeit des Produktes im *SAP R/3*-System bezüglich der Menge und des Ortes.
Bestellungen ansehen	Zeigt alle aktuellen noch nicht abgeschlossene Bestellungen an. Außerdem kann der aktuelle Status der Bestellungen abgefragt werden.
...	...

Tabelle D.2: Katalog der Anwendungsfälle der Kunden

Anwendungsfall	Beschreibung
Produktangebot pflegen	Ermöglicht die Pflege des Produktangebots. Produkte können angezeigt, gelöscht und geändert werden. Neue Produkte werden aus dem *SAP R/3*-System geladen. Vorhandene Produkte können mit dem *SAP R/3*-System abgeglichen werden.
Anmeldung ausfüllen	Die Anmeldung für das Login mit dem Benutzernamen und dem Paßwort wird ausgefüllt.
Auswertungen erstellen	Ermöglicht die Erstellung von Auswertungen über das IPOS (Produkttreffer, Bestellungen, usw.) und über die Benutzer (bevorzugte Produkte, Historische Bestellungen, Verhalten, usw.).
System konfigurieren	Ermöglicht die Konfiguration des IPOS bezüglich der Inhalte der Dialoge/Anfragen und der Ablauflogik.
...	...

Tabelle D.3: Katalog der Anwendungsfälle der Verkaufsleiter

Anwendungsfall	Beschreibung
Hilfsmittel benutzen	Bezeichnet Hilfsmittel des Systemadministrators zur Administration des IPOS. Dieses sind z.B. *Backup*, *Recovery*, usw. .
System konfigurieren	Ermöglicht die Konfiguration des IPOS bezüglich der Inhalte der Dialoge, Anfragen und der Ablauflogik.
Anmeldung ausfüllen	Die Anmeldung für das *Login* mit dem Benutzernamen und dem Paßwort wird ausgefüllt.
Benutzer verwalten	In der Benutzerverwaltung können Benutzer erstellt, geändert und gelöscht werden. Zudem kann hier das Benutzerprofil des jeweiligen Benutzers eingesehen werden. Die Benutzer werden bei der Erfassung oder Änderung direkt im *SAP R/3* erstellt oder geändert.
Auswertungen erstellen	Ermöglicht die Erstellung von Auswertungen über das IPOS (Produkttreffer, Bestellungen, usw.) und über die Benutzer (bevorzugte Produkte, Historische Bestellungen, Verhalten, usw.).
...	...

Tabelle D.4: Katalog der Anwendungsfälle der Systemadministratoren

Anhang E

LotusScript-Klassenbibliothek für den Zugriff auf *Notes*-Komponenten

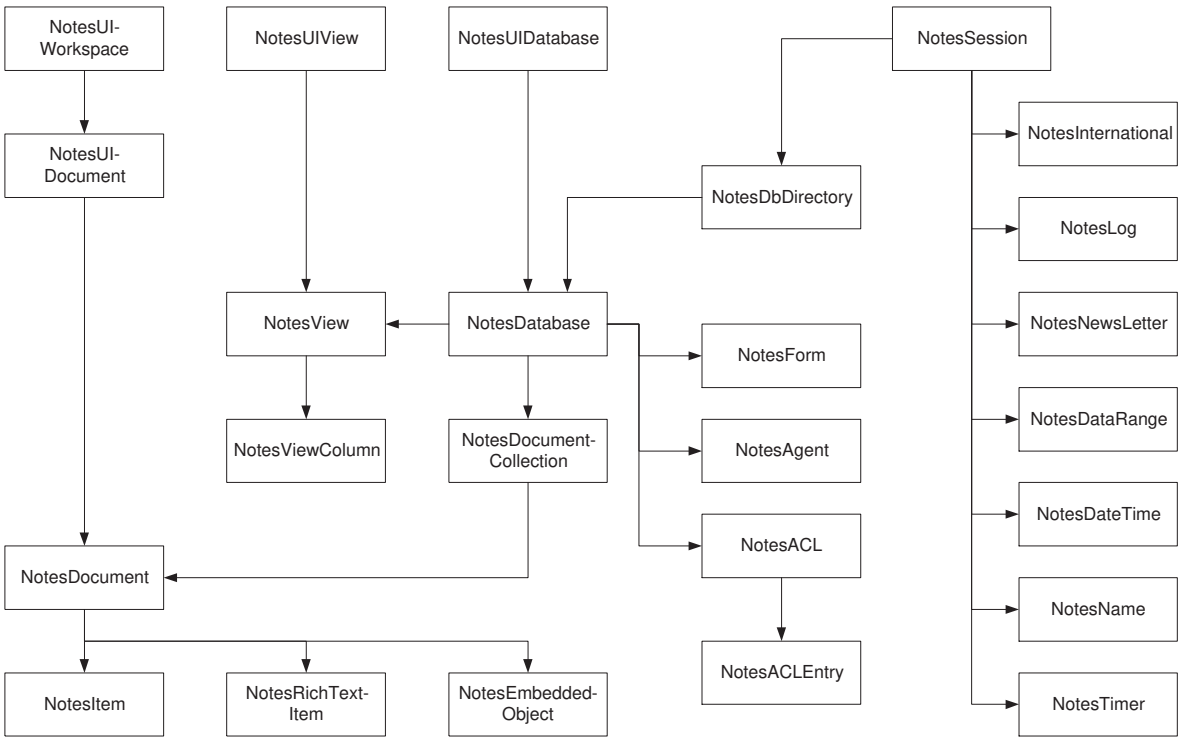

Abbildung E.1: *LotusScript*-Klassenbibliothek

Anhang F

Beschreibungen der Ansichtengruppen der Administrationsschnittstelle der NBC-DB

Ansichtenname	Selektion	Sortierung	Beschreibung
Name	*ConversationSpecs*	*Name*	Sortiert alle Konversationsspez. nach ihrem Namen.

Tabelle F.1: Ansichtengruppe „*ConversationSpecs by*“

Ansichtenname	Selektion	Sortierung	Beschreibung
Name	*DialogSpec*	*Name*	Sortiert alle Dialogspez. nach ihrem Namen.
Database	*DialogSpec*	*Database*	Sortiert alle Dialogspez. nach der Datenbank, die die Inhaltsspez. enthält.
Server and Database	*DialogSpec*	*Server and Database*	Sortiert alle Dialogspez. nach dem Server und der Datenbank, die die Inhaltsspez. enthält.

Tabelle F.2: Ansichtengruppe „*DialogSpecs by*“

Ansichtenname	Selektion	Sortierung	Beschreibung
Name	*RequestSpec*	*Name*	Sortiert alle Anfragespez. nach ihrem Namen.
Visible	*RequestSpec*	*Visible*	Sortiert alle Anfragespez. nach ihrer Sichtbarkeit (Aktion oder Link).
DialogSpecs	*RequestSpec*	*Name*	Sortiert alle Anfragespez. nach ihren zugewiesenen Dialogspez..
Security	*RequestSpec*	*Security*	Sortiert alle Anfragespez. nach ihren Sicherheitsbestimmungen.

Tabelle F.3: Ansichtengruppe „*RequestSpecs by*“

Ansichtenname	Selektion	Sortierung	Beschreibung
DialogSpecName	*RequestSpec*, *DialogSpec*	*Name*	Sortiert alle Dialogspez. nach ihrem Namen und ordnet die zugehörigen Anfragespez. zu.
RequestSpecName	*RequestSpec*, *DialogSpec*	*Name*	Sortiert alle Anfragespez. nach ihrem Namen und ordnet die zugehörigen Dialogspez. zu.

Tabelle F.4: Ansichtengruppe „*DialogSpecs+RequestSpecs by*“

Ansichtenname	Selektionskrit.	Sortierungskrit.	Beschreibung
Name	*ConversationInst*	*Name*	Sortiert alle Konversationsinst. nach dem Namen der Konversationsspez..
Username	*ConversationInst*	*Username*	Sortiert alle Konversationsinst. nach dem Namen des Kunden.
Date	*ConversationInst*	*Date*	Sortiert alle Konversationsinst. nach dem Datum der Konversation.

Tabelle F.5: Ansichtengruppe „*ConversationInst by*“

Ansichtenname	Selektionskrit.	Sortierungskrit.	Beschreibung
Message	*ErrorDialogContentInst*	*Message*	Sortiert alle Fehlerdialoginhalte nach ihrer Fehlermeldung.

Tabelle F.6: Ansichtengruppe „*ErrorDialogContentInst by*“

Literaturverzeichnis

ADGY99 N. Adam, O. Dogramaci, A. Gangopadhyay, and Y. Yesha. *Electronic Commerce - Technical, Business and Legal Issues.* Prentice Hall, 1999.

Aus62 J. Austin. How to do things with words. Technical report, Oxford University Press, Oxford, 1962.

BK98 R. Best and T. Köhler. *Electronic Commerce - Elektronischer Handel in der Praxis.* Addison-Wesley Publishing Company, 1998.

BKKZ92 R. Budde, K. Kautz, K. Kuhlenkamp, and H. Züllighoven. *Prototyping - an Approach to Evolutionary System Development.* Springer, 1992.

Boo94 G. Booch. *Object-Oriented Design with Applications.* Benjamin/Cummings Publishing Company, Inc., second edition, 1994.

BW94 Russel Beale and Andrew Wood. Agent-based interaction. *People and Computers*, IX:239–245, 1994.

CCK+97 F. Collins, Y. Cho, P. Kapur, W. Lee, D. Stump, and M. West. *Enterprise Integration with Domino.Connect.* IBM and Lotus, 5 1997.

Cha97 Deepika Chauhan. JAFMAS: A java-based agent framework for multiagent systems development and implementation. Master's thesis, ECECS Department, University of Cincinnati, 1997.

CHT+97 F. Collins, F. Hansen, T. Tulisalo, F. Vitiello, and Min X. Li. *Developing Web Applications Using Lotus Notes Designer for Domino 4.6.* IBM and Lotus, 1997.

Com98a Intershop Communications. Intershop 3: Abstract White Paper. Technical report, Intershop Communications, 1998. http://www.intershop.com/.

Com98b Intershop Communications. Intershop 3 Enterprise Edition: Extension Interface. Technical report, Intershop Communications, 1998. http://www.intershop.com/.

Com98c Intershop Communications. Intershop 3 Enterprise Edition: Hooks. Technical report, Intershop Communications, 1998. http://www.intershop.com/.

Com98d Intershop Communications. Intershop 3 Enterprise Edition: Server Side Scripting. Technical report, Intershop Communications, 1998. http://www.intershop.com/.

Com98e Intershop Communications. Intershop 3 Enterprise Edition: Shared Lib. Technical report, Intershop Communications, 1998. http://www.intershop.com/.

Com98f Intershop Communications. Intershop 3 Enterprise Edition: Solution Summary. Technical report, Intershop Communications, 1998. http://www.intershop.com/.

Com98g Intershop Communications. Intershop 3: Security White Paper. Technical report, Intershop Communications, 1998. http://www.intershop.com/.

Com98h Intershop Communications. Intershop 3: Users Guide Volume I. Technical report, Intershop Communications, 1998. http://www.intershop.com/.

Com98i Intershop Communications. Intershop 3: Users Guide Volume II. Technical report, Intershop Communications, 1998. http://www.intershop.com/.

Com98j Intershop Communications. Intershop Developer Kit 1.15. Technical report, Intershop Communications, 12 1998. http://www.intershop.com/.

Con98 OBI Consortium. Open Buying on the Internet: Technical Specifications. Technical report, OBI Consortium, June 1998. http://www.openbuy.org/.

Cor97a Lotus Development Corporation. Dokumentation des SiteCreator R2.0. Technical report, Lotus, 1997.

Cor97b Lotus Development Corporation. *Domino Administration Help.* Lotus, 1997.

Cor97c Lotus Development Corporation. Evaluation Guide. Technical report, Lotus, 1997.

Cor97d Lotus Development Corporation. *Java Programmer's Guide.* Lotus, 1997.

Cor97e Lotus Development Corporation. Lotus Domino.Merchant Specifications. Technical report, Lotus, 1997.

Cor97f Lotus Development Corporation. *Lotus Notes and the Internet.* Lotus, 1997.

Cor97g Lotus Development Corporation. *LotusScript Extension für SAP R/3.* Lotus, 1997.

Cor97h Lotus Development Corporation. *LSX Toolkit Guide.* Lotus, 1997.

Cor97i Lotus Development Corporation. Merchant Administrator's Guide. Technical report, Lotus, 1997.

Cor97j Lotus Development Corporation. Merchant Installation Guide. Technical report, Lotus, 1997.

Cor97k Lotus Development Corporation. *Notes C++ API Release 4.12 Guide.* Lotus, 1997.

Cor97l Lotus Development Corporation. *Notes Help.* Lotus, 1997.

Cor97m Lotus Development Corporation. *SAP Script Examples.* Lotus, 1997.

Cor98a Lotus Development Corporation. *Notes C API 4.6.2 Reference.* Lotus, 1998.

Cor98b Lotus Development Corporation. *Notes C API 4.6.2 User Guide*. Lotus, 1998.

Dad96 P. Dadam. *Verteilte Datenbanken und Client/Server-Systeme : Grundlagen, Konzepte und Realisierungsformen*. Springer, 1996.

FGHW88 F. Flores, M. Graves, B. Hartfield, and T. Winograd. Computer systems and the design of organizational interaction. *ACM Transactions on Office Information Systems*, 6(2):153–172, 1988.

FPJ98 K. Fochler, P. Perc, and Ungermann J. *Lotus Domino 4.6: Internet- und Intranetlösungen mit dem Lotus Domino Server*. Addison-Wesley Publishing Company, 1998.

FS97 Martin Fowler and Kendall Scott. *UML Distilled: Applying the Standard Object Modelling Language*. Addison-Wesley Publishing Company, 1997.

GHS91 Günther Grewendorf, Fritz Hamm, and Wolfgang Sternefeld. *Sprachliches Wissen: Eine Einführung in moderne Theorien der grammatischen Beschreibung*. Suhrkamp, Frankfurt/Main, 5. auflage edition, 1991.

Hru98 P. Hruschka. Ein pragmatisches Vorgehensmodell für die UML. *OBJEKTspektrum*, February 1998.

Hup98 Patrick Hupe. Zustandstypisierung. Studienarbeit, Fachbereich Informatik, Universität Hamburg, 1998.

iC97a iCat Corporation. iCat Carbo Editor User Guide. Technical report, iCat-Corporation, 1997. http://www.icat.com/.

iC97b iCat Corporation. iCat Carbo Reference. Technical report, iCat-Corporation, 1997. http://www.icat.com/.

iC97c iCat Corporation. iCat Commerce Publisher User Guide. Technical report, iCat-Corporation, 1997. http://www.icat.com/.

iC97d iCat Corporation. iCat Data Entry Manager User Guide. Technical report, iCat-Corporation, 1997. http://www.icat.com/.

iC97e iCat Corporation. iCat Electronic Commerce Solution 3.0: Facts and Features. Technical report, iCat-Corporation, 1997. http://www.icat.com/.

iC97f iCat Corporation. iCat Getting Started Guide. Technical report, iCat-Corporation, 1997. http://www.icat.com/.

JCJÖ92 I. Jacobson, M. Christerson, P. Jonson, and G. Övergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Publishing Company, 1992.

Joh97 Nico Johannisson. Eine Umgebung für mobile Agenten: Agentenbasierte verteilte Datenbanken am Beispiel der Kopplung autonomer Internet Web Site Profiler. Diplomarbeit, Fachbereich Informatik, Universität Hamburg, April 1997.

JR99 Latza J. and Lühr R. Eine generische und objektorientierte Schnittstelle von TYCOON zum System SAP R/3. Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 1999.

Kra98 J. Krause. *Electronic commerce - Geschäftsfelder der Zukunft heute nutzen.* Hanser, 1998.

KW97 R. Kalakota and A. Whinston. *Electronic Commerce: A Manager's Guide.* Addison-Wesley Publishing Company, 1997.

LL96 D. Lynch and L. Lundquist. *digital money.* John Wiley & Sons, 1996.

Lot97 Lotus. *The History of Notes.* Lotus, 1997. http://www.notes.net/history.nsf/.

LP97 Laura Lemay and Charles Perkins. *Java 1.1 in 21 Tagen.* SAMS, 1997.

LS87 P.C. Lockemann and J.W. Schmidt, editors. *Datenbank-Handbuch.* Springer, 1987.

Man98 Raimund Mann. *Lotus Script Programmieren für Notes.* Computer $ Literatur, 1998.

Mat93 F. Matthes. *Persistente Objektsysteme: Integrierte Datenbankentwicklung und Programmerstellung.* Springer-Verlag, 1993.

Mat97a Florian Matthes. Business Conversations: A high-level system model for agent coordination. In *Proceedings of the Sixth International Workshop on Database Programming Languages, Estes Park, Colorado.* Springer-Verlag, August 1997.

Mat97b Florian Matthes. Mobile processes in cooperative information systems. In *STJA'97, Smalltalk und Java in Industrie und Ausbildung.* Springer-Verlag, September 1997.

MC97 Microsoft-Corporation. MS Internet Commerce Strategy: A Foundation for Doing Business on the Internet. Technical report, Microsoft-Corporation, 1997. http://www.micrsoft.com/technet/.

MC98a Microsoft-Corporation. MCIS 2.0 Features and Benefits with Architectural Diagrams. Technical report, Microsoft-Corporation, 1998. http://www.micrsoft.com/technet/.

MC98b Microsoft-Corporation. MCIS 2.0 Overview. Technical report, Microsoft-Corporation, 1998. http://www.micrsoft.com/technet/.

MC98c Microsoft-Corporation. Microsoft Site Server, Commerce Edition Data Sheet. Technical report, Microsoft-Corporation, 1998. http://www.microsoft.com/siteserver/commerce/.

MC98d Microsoft-Corporation. Microsoft Site Server, Commerce Edition Documentation. Technical report, Microsoft-Corporation, 1998. Bestandteil der Evualationsversion.

MC98e Microsoft-Corporation. Microsoft Site Server, Commerce Edition Evaluation Guide. Technical report, Microsoft-Corporation, 1998. http://www.microsoft.com/siteserver/commerce/.

MC98f Microsoft-Corporation. MS Commercial Internet System Data Sheet. Technical report, Microsoft-Corporation, 1998. http://www.micrsoft.com/technet/.

MC98g Microsoft-Corporation. MS Site Server 3.0, Commerce Edition Whitepaper. Technical report, Microsoft-Corporation, 1998. http://www.micrsoft.com/technet/.

MMM93 B. Mathiske, F. Matthes, and S. Müßig. The Tycoon system and library manual. DBIS Tycoon Report 212-93, Fachbereich Informatik, Universität Hamburg, December 1993.

MMS94 F. Matthes, S. Müßig, and J.W. Schmidt. Persistent polymorphic programming in Tycoon: An introduction. FIDE Technical Report FIDE/94/106, Fachbereich Informatik, Universität Hamburg, August 1994.

Mor97 Michael Morrison. *Java 1.11 für Insider.* SAMS, 1997.

Nie95 K.D. Niemann. *Client/Server-Architektur, Organisation und Methodik der Anwendungsentwicklung.* Vieweg, 1995.

Pou97 N. V. Pour. Entwicklung von Client/Server-Anwendungen mit 4GL-Technologie: Tycoon, NATURAL und ABAP/4 im Vergleich. Diplomarbeit, Fachbereich Informatik, Universität Hamburg, September 1997.

RBP+91 J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

RF98 Möhrlen R. and Kokot F. *SAP R/3 Kompendium.* Markt und Technik, Buch- und Software-Verlag, 1998.

Ric97 Ingo Richtsmeier. Kommunizierende Informationssysteme am Beispiel autonomer Internet WebSiteProfiler: Vergleich objekt- und agentenbasierter Ansätze. Diplomarbeit, Fachbereich Informatik, Universität Hamburg, January 1997.

Rip98 Volker Ripp. Objektorientierter Entwurf und Realisierung eines Agentensystems für kooperative Internet-Informationssysteme. Diplomarbeit, Fachbereich Informatik, Universität Hamburg, May 1998.

SAP98 SAP. *SAP Online Hilfe 4.0b.* SAP, 1998.

Sea69 J. Searle. Speech acts. Technical report, Cambridge University Press, Cambridge, 1969.

SF98 U. Schröter and S. Fügner. *Das Domino Prinzip: Dynamische Generierung interaktiver HTML-Dokumente mit Lotus Notes/Domino.* dpunkt-Verl. für digitale Technologie, 1998.

Sof97a Rational Software. UML Notation Guide Version 1.1. Technical report, Rational Software and Microsoft and Hewlett-Packard and Oracle Sterling Software and MCI Systemhouse and Unisys and ICON Computing IntelliCorp and i-Logix and IBM and ObjecTime and Platinum Technology and Ptech Taskon and Reich Technologies and Softeam, September 1997. http://www.rational.com/uml.

Sof97b Rational Software. UML Summary Version 1.1. Technical report, Rational Software and Microsoft and Hewlett-Packard and Oracle Sterling Software and MCI Systemhouse and Unisys and ICON Computing IntelliCorp and i-Logix and IBM and ObjecTime and Platinum Technology and Ptech Taskon and Reich Technologies and Softeam, September 1997. http://www.rational.com/uml.

Tea97 ISV Development Team. Lotus Notes/Domino 4.6: Application Development Best Practices Guide. Technical report, Lotus, 1997.

Wah98 G. Wahl. UML kompakt. *OBJEKTspektrum*, 2 1998.

Was97a Michael Wasmeier. Shop in the box: Funktionsweise von online-shop-komplettlösungen. *ct magazin für Computertechnik*, July 1997.

Was97b Michael Wasmeier. Shop in the box: Komplettlösungen für online-shops. *ct magazin für Computertechnik*, November 1997.

Wat90 D.A. Watt. *Programming language concepts and paradigms.* Prentice Hall international series in computer science. Prentice Hall, Englewood Cliffs, New Jersey, 1990.

WB98 Donald Weyer and Jo Bager. Am Cyber-Markt sind noch Läden frei. *ct magazin für Computertechnik*, 23, 12 1998.

Weg98 Holm Wegner. Objektorientierter Entwurf und Realisierung eines Agentensystems für kooperative Internet-Informationssysteme. Diplomarbeit, Fachbereich Informatik, Universität Hamburg, May 1998.

Win87 T.A. Winograd. A language/action perspective on the design of cooperative work. Technical Report No. STAN-CS-87-1158, Stanford University, May 1987.

Wit60 L. Wittgenstein. *Philosophische Untersuchungen.* Frankfurt/Main, 1960.

Eidesstattliche Erklärung

„Ich versichere, daß ich die vorstehende Arbeit selbständig und ohne fremde Hilfe angefertigt und mich nicht anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.“

Hamburg, den 3. Juni 1999

(Andreas Carlsen)