BERUFSAKADEMIE MANNHEIM
Wirtschaftsinformatik

Studienarbeit

Objektorientierte Softwareentwicklung
am Beispiel einer Offline-Suche

Oliver Theobald

Studienfach: Systementwicklung

Kurs:	WWI00H
Ausbildungsbetrieb:	BASF Aktiengesellschaft, Ludwigshafen
Betreuender Dozent:	Herr Eckardt

Abstract

Die Methode der objektorientierten Analyse (OOA) stellt eine Reihe von Notationstypen und –formen als Hilfsmittel zur Verfügung, die das Modell einer Anwendung veranschaulichen und näher beschreiben können. Darüber hinaus existiert jedoch eine Vielzahl weiterer Notationsmöglichkeiten aus anderen Methoden oder Anwendungsgebieten der Softwareentwicklung.

Im Rahmen dieser Studienarbeit werden sowohl die in der OOA typischen Notationen als auch Notationen, z.B. aus der klassischen Strukturierten Analyse, auf ihren Beitrag zu einem Softwaremodell hin untersucht.
In einem zweiten Teil werden ausgewählte Notationsformen zur Modellierung einer Beispielanwendung benutzt.

Inhaltsverzeichnis

1 Gegenstand der Arbeit

1.1 Ziel der Arbeit

Mit dieser Arbeit soll das Modell einer Anwendung entwickelt werden. Das Modell soll dabei so viele Informationen beinhalten, dass ein Programmierer dieses in einer beliebigen objektorientierten Programmiersprache implementieren könnte.
Die Anwendung wird HTML-Dokumente verarbeiten, aus diesen die wichtigsten Begriffe im Text identifizieren und getrennt vom Ursprungsdokument ablegen.
Aufgrund des Archivs, das bei der Ausführung des Programms entstehen wird, soll später in einem zweiten Schritt eine *Offline-Suche* realisiert werden. Offline-Suche bedeutet, dass der Anwender sich die Archiv-Dateien für eine oder mehrere Webpräsenz(en) und eine grafische Benutzeroberfläche auf seinen Rechner installiert. Anschließend kann er ohne Internetverbindung eine Suche durchführen, sich die Ergebnisse anzeigen lassen und, nachdem er eine Verbindung zum Internet hergestellt hat, per Link direkt an die entsprechende Stelle auf der Webseite springen.

1.2 Themenabgrenzung

Ausgehend von den verschiedenen Phasen der Softwareentwicklung, liegt der Schwerpunkt dieser Arbeit auf der Phase der Analyse, insbesondere der objektorientierten Analyse (OOA). Themenbereiche, die über die OOA hinausgehen, werden nur teilweise dargestellt.
Das Ergebnis der OOA ist ein Modell; vor der Implementierung müsste also noch die Verarbeitung des Modells zu einem Software-Entwurf erfolgen. Ziel dieser Arbeit ist, diesen Schritt und den damit verbundenen Aufwand möglichst klein zu halten.
Das Modell der Anwendung im zweiten Teil dieser Studienarbeit beschränkt sich auf die Verarbeitung der HTML-Dokumente und das Erstel-

len des Sucharchivs. Die spätere Suche nach Begriffen und die Anzeige der Suchergebnisse in einem Browser sind nicht Gegenstand dieser Arbeit.

CASE-Werkzeuge im allgemeinen, sowie deren Arbeitsweise und Nutzen werden nicht betrachtet.

Für die grafische Darstellung des Modells wird ein CASE-Werkzeug verwendet, Rational Rose. Dieses Werkzeug ist allerdings nur Hilfsmittel zur Entwicklung von Diagrammen in dieser Studienarbeit.

2 Notationsformen in der Softwaremodellierung

2.1 Unterscheidung nach Sichten

In der Systementwicklung ist es üblich, die Beschreibung eines Systementwurfs durch die Verwendung von Diagrammen zu unterstützen. Seit Einführung des UML-Standards im Jahr 1999 ist es bis heute sogar möglich, aus erstellten Diagrammen mit CASE-Werkzeugen Programm-Code zu generieren, der die Implementierung des Entwurfs erleichtert.[1]
Um einen möglichst genauen und übersichtlichen Entwurf zu entwickeln, ist es daher wichtig, diejenigen Diagrammtypen auszuwählen, die zum Verständnis des zu entwickelnden Systems beitragen und nicht die Komplexität unnötig erhöhen. Schließlich soll ein grafischer Entwurf der besseren Verständlichkeit dienen.
Balzert gliedert die verschiedenen Diagrammtypen bzw. Notationsformen in insgesamt sieben Sichten:

- Funktionale Sicht
- Datenorientierte Sicht
- Objektorientierte Sicht
- Algorithmische Sicht
- Regelbasierte Sicht
- Zustandsorientierte Sicht
- Szenariobasierte Sicht[2]

Die einzelnen Sichten umfassen dabei eine oder mehrere Notationsformen, die austauschbar sind bzw. sich gegenseitig ergänzen[3]. Für die

[1] vgl. Balzert, Helmut: Lehrbuch der Softwaretechnik – Software-Entwicklung, 2. Aufl., Heidelberg/Berlin 2000, S. 174 ff.
[2] vgl. ebenda, S. 106, Abb. 2.2-2
[3] vgl. ebenda, S. 105

objektorientierte Analyse (OOA) schlägt Balzert die folgenden Diagrammtypen vor:

Diagrammtyp	Sicht
Geschäftsprozessdiagramm	Funktionale Sicht
Entity Relationship-Modell	Datenorientierte Sicht
Klassendiagramm	Objektorientierte Sicht
Zustandsautomat	Zustandsorientierte Sicht
Sequenzdiagramm	Szenariobasierte Sicht
Kollaborationsdiagramm	Szenariobasierte Sicht

Tabelle 2.1-1: Zu modellierende Diagramme der OOA nach Balzert[4]

Im folgenden werden nun die einzelnen Sichten mit ihren verschiedenen Diagrammtypen auf ihren Nutzen bei der OOA untersucht. Dabei finden die oben genannten Notationsformen besondere Berücksichtigung.
Die algorithmische Sicht entfällt, da sie nach Balzert implizit in der OOA enthalten ist, was bedeutet, dass die Aufgaben dieser Sicht durch andere Sichten innerhalb der OOA schon abgedeckt sind[5]. Deshalb wird die algorithmische Sicht in dieser Studienarbeit auch nur am Rande betrachtet und erhält kein eigenes Kapitel.

2.2 Funktionale Sicht

In der funktionalen Sicht finden sich die Notationsformen *Funktionsbaum*, *Geschäftsprozesse* und *Datenflussdiagramm* wieder.[6]

Funktionsbaum

Ein Funktionsbaum entsteht durch die hierarchische Anordnung von Funktionen. Dabei bedeutet die Verbindung zwischen über- und

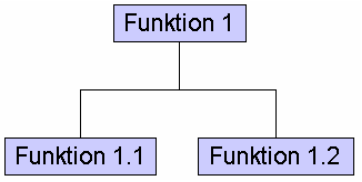

Abb. 2.2-1: Funktionsbaum

[4] vgl. Balzert, a.a.o., S. 108, Abb. 2.2-4
[5] vgl. Balzert, a.a.o., S. 108, Abb. 2.2-4
[6] vgl. Balzert, a.a.o., S. 123

untergeordneter Funktion meistens „Besteht-aus“ oder „Ruft-auf“[7]. Es findet also lediglich eine Strukturierung der einzelnen Funktionen des Systems statt, ohne dass Daten oder Arbeitsabläufe berücksichtigt werden[8].

Geschäftsprozessdiagramm

Mit Hilfe von Geschäftsprozess- oder Anwendungsfalldiagrammen können sowohl der Kontext eines Systems, als auch die Anforderungen an ein System modelliert werden[9]. Sie beschreiben die Arbeitsabläufe (Anforderungen), die ein Akteur (Kontext) im System auslösen kann. Die Beschreibung der einzelnen Geschäftsprozesse bleibt dabei auf einer sehr hohen Abstraktionsebene.[10]

Geschäftsprozessdiagramme bieten sich daher als Unterstützung der Kommunikation zwischen Entwicklern und Endanwendern an[11]. Als Ausgangsbasis der OOA sind Geschäftsprozessdiagramme notwendig für die weitere Entwicklung eines Softwaresystems[12].

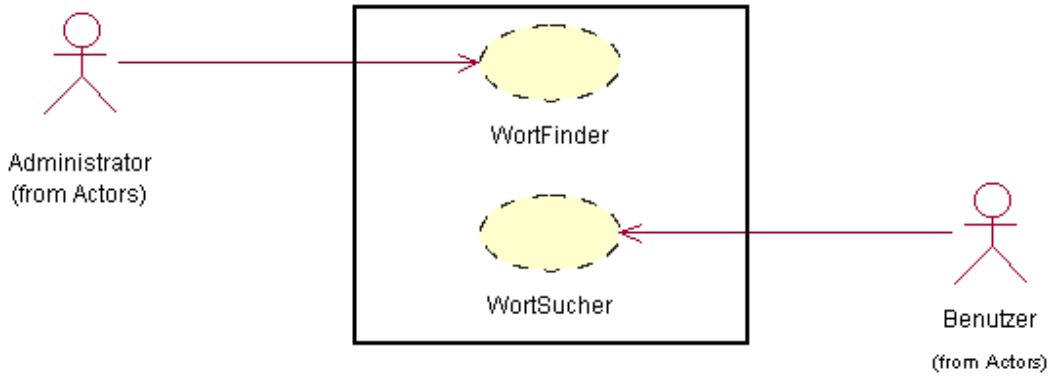

Abb. 2.2-2: Geschäftsprozessdiagramm

[7] vgl. Balzert, a.a.o., S. 124
[8] vgl. Stahlknecht, Peter/Hasenkamp, Ulrich: Einführung in die Wirtschaftsinformatik, 10., überarbeitete und aktualisierte Auflage, Berlin/Heidelberg 2002, S. 262
[9] vgl. Booch, Grady/Rumbaugh, Jim/Jacobson, Ivar: Das UML-Benutzerhandbuch, Bonn 1999, S. 266
[10] vgl. Balzert, a.a.o., S. 145
[11] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 264
[12] vgl. Stahlknecht, Hasenkamp, a.a.o., S. 283

Es gibt generelle Unterscheidungen von Geschäftsprozessen, abhängig von der Abstraktionsebene, auf der man sich befindet:

- Geschäftsprozess als Unternehmensprozess
- Geschäftsprozess als Arbeitsablauf in einem Softwaresystem

Es ist also möglich für ein und dasselbe System verschiedene Geschäftsprozessdiagramme auf unterschiedlichen Abstraktionsebenen zu entwerfen.[13]

Datenflussdiagramm

Die Beschreibung der Datenflüsse im Datenflussdiagramm ist Ausgangspunkt der Strukturierten Analyse nach De Marco[14]. Es wird wie bei Geschäftsprozessen versucht, Informationsflüsse zwischen dem Softwaresystem und der Umwelt darzustellen[15].
Obwohl das Datenflussdiagramm neben den Funktionen auch Datenströme berücksichtigt, wird dennoch die Trennung von Funktionen und Daten beibehalten, da die Modellierung der Daten nicht betrachtet wird.[16]

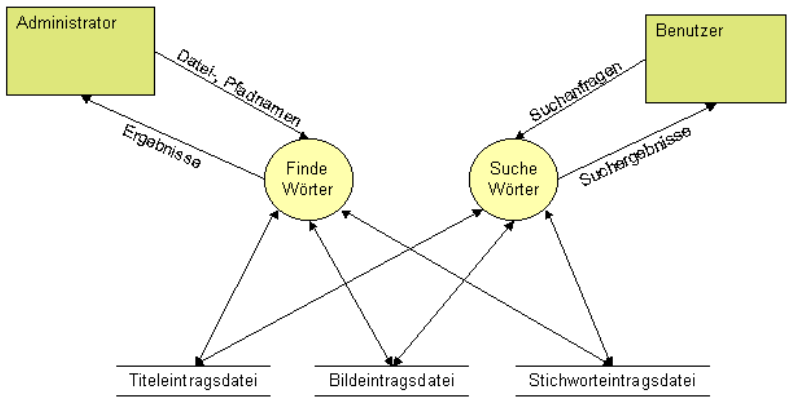

Abb. 2.2-3: Datenflussdiagramm

[13] vgl. Balzert, a.a.o., S. 126
[14] vgl. Stahlknecht/Hasenkamp, a.a.o., S. 264
[15] vgl. Balzert, a.a.o., S. 142
[16] vgl. Stahlknecht/Hasenkamp, a.a.o., S. 264 f.

2.3 Datenorientierte Sicht

Balzert fasst unter der datenorientierten Sicht zwei Notationsformen zusammen, das *Data Dictionary* und das *Entity-Relationship-Modell*, im folgenden ER-Modell genannt.[17]

Dabei ordnet er diese beiden Modelle den verschiedenen Modellierungsansätzen in der Softwareentwicklung unterschiedlich zu, wie die folgende Tabelle zeigt:

Modellierungsansatz	ER-Modell	Data Dictionary
Objektorientierte Analyse (OOA)	X	
Strukturierte Analyse (SA)		X
Echtzeitanalyse (RT)	X	X

Tabelle 2.3-1: Zuordnung der Diagrammtypen zu Modellierungsansätzen[18]

Nach dieser Zuordnung wäre in der OOA lediglich das ER-Modell einzusetzen.

Allerdings sagt Balzert auch, dass sobald von einer hohen Komplexität der Daten auszugehen ist, auf jeden Fall ein ER-Modell einzusetzen ist, egal welcher Modellierungsansatz gewählt wurde[19]. Doch „dies gilt nicht, wenn OOA eingesetzt wird, da OOA das *Entity Relationship*-Modell enthält.“[20]

Inwiefern das ER-Modell in der OOA enthalten ist, zeigen Booch, Rumbaugh und Jacobson. Sie verweisen darauf, dass Klassendiagramme, feste Bestandteile der OOA, eine Obermenge von Entity-Relationship-Diagrammen bilden.[21]

Als Methodik schlagen sie daher vor, ausgehend von einem bestehenden Klassendiagramm, diejenigen Klassen, deren Zustand länger als die Zeit, in der die Anwendung aktiv ist, festgehalten werden muss, zu identifizieren

[17] vgl. Balzert, a.a.o., S. 223
[18] vgl. Balzert, a.a.o., S. 108
[19] vgl. Balzert, a.a.o., S. 109
[20] Balzert, a.a.o., S. 109
[21] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 123

und entsprechend als *persistent* zu kennzeichnen. Anschließend werden eben diese um datenbankspezifische Eigenschaften erweitert.[22]
Bei dieser Vorgehensweise bleiben die in den ursprünglichen Klassen definierten Methoden erhalten, was eine Abkehr von der rein datenorientierten Sichtweise im ER-Modell bedeutet. Allerdings wird somit auch ein Bruch in der durchgängigen Systementwicklung vermieden, da zum einen die Begrifflichkeiten nicht geändert werden (ER-Modell: *Entitätstyp* entspricht UML: *Klasse*) [23], und zum anderen die Trennung von Daten und Funktionen aufgehoben wird.[24]

Mit Data Dictionary-Einträgen (DD-Einträge) ist es möglich, die Struktur der Daten sehr genau zu beschreiben. DD-Einträge ähneln in der Notationsform einer Programmiersprache, sind also keine grafische Notationsform und deshalb nur eingeschränkt lesbar. Es findet ebenfalls wie beim ER-Modell die unerwünschte Trennung von Daten und Funktionen statt.[25]

2.4 Objektorientierte Sicht

Mit Hilfe von Klassen- und Objektdiagrammen lassen sich sämtliche statischen Konzepte, die die Objektorientierung ausmachen, konkret anwenden:

- Datenkapselung und Objektbildung
- Klassenbildung und Vererbung

Lediglich die dynamischen Aspekte der Objektorientierung, Botschaftenkommunikation und Polymorphismus, müssen mit anderen Hilfsmitteln dargestellt werden, z.B. durch Interaktionsdiagramme. [26]
Die Darstellung von Klassen- und Objektdiagrammen entspricht einer Zusammenstellung von Kanten und Knoten[27].

[22] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 123 f.
[23] vgl. Balzert, a.a.o., S. 225
[24] vgl. Stahlknecht/Hasenkamp, a.a.o., S. 265
[25] vgl. Balzert, a.a.o., S. 247 ff.
[26] vgl. Stahlknecht/Hasenkamp, a.a.o., S. 284
[27] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 118 und 221

Dabei zeigen Klassendiagramme die statischen Elemente eines Systems, z.B. Klassen, Beziehungen und Schnittstellen[28]. Klassen beschreiben eine „Menge von Objekten mit gleichen Attributen, Operationen, Beziehungen und gleicher Bedeutung.“[29]

Objektdiagramme werden von Balzert nicht explizit in seiner Einteilung der verschiedenen Sichten aufgeführt[30]. Ein Grund hierfür könnte sein, dass er die Objektdiagramme als eine Sonderform der Klassendiagramme ansieht, was allerdings nicht aus dem Text hervorgeht. Booch, Rumbaugh und Jacobson sehen dagegen die Objektdiagramme als eigenständige Notationsform an[31].

In dieser Studienarbeit werden Objektdiagramme als eigener Diagrammtyp behandelt, da hieraus kein Widerspruch mit den beiden Werken entsteht.

Ein Objektdiagramm stellt eine gewisse Anordnung der Instanzen der verschiedenen Klassen zu einem bestimmten Zeitpunkt dar. Objektdiagramme sind also nichts anderes als eine Momentaufnahme eines laufenden Systems. Dabei besteht eine Beziehung zwischen Objekt- und Klassendiagramm: Was ein Objekt für eine Klasse darstellt, ist ein Objektdiagramm für ein Klassendiagramm. Es handelt sich also bei einem Objektdiagramm um die Instanz eines Klassendiagramms.[32]

2.5 Regelbasierte Sicht

Balzert versteht unter der regelbasierten Sicht Regeln und Entscheidungstabellen[33].

Regeln werden vorrangig in wissensbasierten Systemen bzw. dem Bereich der Künstlichen Intelligenz (KI) eingesetzt[34], können aber auch zur Formulierung von Anforderungen an ein Softwaresystem eingesetzt werden. Eine Regel kann formal in Pseudo-Code festgehalten werden. Es

[28] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 520
[29] Booch/Rumbaugh/Jacobson, a.a.o., S. 520
[30] vgl. Balzert, a.a.o., S. 106 ff.
[31] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 26
[32] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 220 f.
[33] vgl. Balzert, a.a.o., S. 106, Abb. 2.2-2
[34] vgl. Stahlknecht/Hasenkamp, a.a.o., S. 435 ff.

ist aber auch möglich, Regeln verbal in Wenn-dann-Form anzugeben. Dadurch ist die Verständlichkeit von Regeln so hoch, dass auch Endanwender leicht an der Entwicklung von Regeln für Software mitwirken können. Die Regeln werden dann während der Implementierung durch den Programmierer in Verzweigungen transformiert.[35]

Entscheidungstabellen stellen strukturierte Regelsammlungen dar, die mehrere Vorbedingungen haben können („Wenn-Teil“), und von denen abhängig von der (Nicht-)Erfüllung der Vorbedingungen mehrere Aktionen ausgehen können. Die Darstellung ist genormt (DIN 66241) und erfolgt in Tabellenform.[36]

Um die Nachvollziehbarkeit zu erhöhen, ist es aber auch möglich, Entscheidungstabellen als Entscheidungsbaum darzustellen[37].

2.6 Zustandsorientierte Sicht

Zustandsdiagramme in der UML sind die grafische Darstellungsform von Zustandsautomaten. Sie basieren auf den Automatenmodellen von Mealy und Moore sowie den *Statecharts* von Harel[38]. Sie werden eingesetzt, um den Lebenszyklus von Objekten einer Klasse oder allgemein die dynamischen Aspekte von einem ganzen System bzw. einem Teilsystem darzustellen[39].

Aktivitätsdiagramme sind ein Sonderfall der Zustandsdiagramme: hier sind alle Zustände sogenannte Aktivitätszustände. Diese zeichnen sich dadurch aus, dass mit Eintritt in einen solchen Zustand eine Art Prozedur beginnt, und dass mit dem Abschluss der Prozedur der Übergang in den nächsten Zustand bzw. in die nächste Aktivität stattfindet.[40]

Mit Aktivitätsdiagrammen werden meist Operationen oder Anwendungsfälle ausgestaltet, d.h. es wird ein Arbeitsablauf entworfen. In dieser Hinsicht können Aktivitätsdiagramme die Funktion eines sogenannten

[35] vgl. Balzert, a.a.o., S. 292
[36] vgl. Balzert, a.a.o., S. 270
[37] vgl. Balzert, a.a.o., S. 276
[38] vgl. Balzert, a.a.o., S. 320 ff.
[39] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 379
[40] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 297

Programm-Ablauf-Plans (PAP) aus der algorithmischen Sicht übernehmen.[41]

Dies stützt die Annahme, dass die algorithmische Sicht implizit in der OOA enthalten ist (siehe Abschnitt 2.1).

Neben den Zustandsautomaten zählt Balzert auch die Petri-Netze zur zustandsorientierten Sicht[42]. „Sie eignen sich zur Modellierung, Analyse und Simulation von dynamischen Systemen mit nebenläufigen und nichtdeterministischen Vorgängen.“[43] Zustandsautomaten dagegen beschreiben deterministische Vorgänge[44]. Somit gilt für Zustandsautomaten, dass auf jeden Zustand genau ein Folgezustand folgt. In Petri-Netzen dagegen können auf einen Netzknoten mehrere Knoten folgen. Ebenso können mehrere Knoten einen gemeinsamen Folgeknoten haben.

Ein Petri-Netz ist ein gerichteter Graph. Die Unterschiede zwischen den einzelnen Netztypen (siehe Abb. 2.6-1) liegen dabei vor allem in der Interpretation der beiden Knotenarten *Stelle* und *Transition*, sowie den Kanten (*Pfeil*) und *Markierungen.*[45]

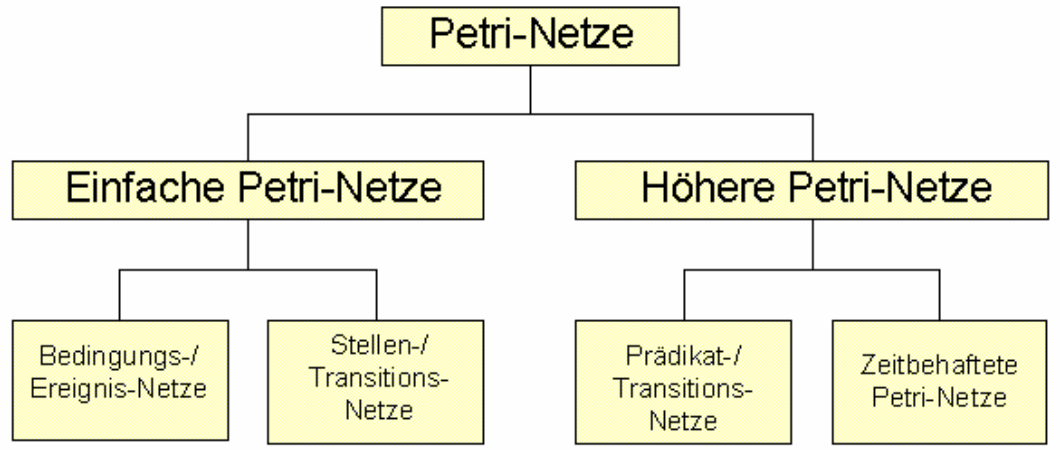

Abb. 2.6-1: Unterscheidung von Petri-Netzen[46]

[41] vgl. Balzert, a.a.o., S. 334 f.
[42] vgl. Balzert, a.a.o., S. 106, Abb. 2.2-2
[43] Balzert, a.a.o., S. 346
[44] vgl. Balzert, a.a.o. S. 341
[45] vgl. Balzert, a.a.o., S. 370
[46] vgl. Balzert, a.a.o., S. 369 f.

2.7 Szenariobasierte Sicht

Sequenz- und Kollaborationsdiagramme werden in der UML als Interaktionsdiagramme bezeichnet[47]. Balzert ordnet diese der szenariobasierten Sicht zu[48].

Interaktionsdiagramme zeigen nicht wie Klassendiagramme den Bauplan eines Systems, sondern sie präsentieren ein Geschehen, ein Szenario zur Laufzeit des Systems.[49]

Sequenz- und Kollaborationsdiagramme sind semantisch äquivalent[50]. Die Unterschiede liegen in der Darstellungsform und in der Art der repräsentierten Informationen:

- Während Kollaborationsdiagramme einem Graphen mit Kanten und Knoten entsprechen, sind Sequenzdiagramme nichts anderes als eine Tabelle[51].
- Sequenzdiagramme zeigen den zeitlichen Ablauf von Kontrollflüssen, Kollaborationsdiagramme dagegen betonen die strukturellen Beziehungen zwischen Objekten, zeigen also die Organisation des Kontrollflusses[52].

Beiden Darstellungsformen ist gemeinsam, dass sie die dynamischen Konzepte der Objektorientierung, Botschaftenkommunikation (das Aufrufen von Operationen) und damit verbunden Polymorphismus (eine Botschaft löst abhängig vom Empfängerobjekt unterschiedliche Reaktionen aus), darstellen.[53]

Daraus folgt, dass Interaktionsdiagramme für den dynamischen Teil der Anwendung ebenso notwendig sind wie Klassendiagramme für den statischen.

[47] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 275
[48] vgl. Balzert, a.a.o., S. 106, Abb. 2.2-2
[49] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 275
[50] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 282
[51] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 277
[52] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 283
[53] vgl. Stahlknecht/Hasenkamp, a.a.o., S. 285

2.8 Weitere Notationsformen

Es gibt noch viele weitere Notationsformen, von denen die wichtigsten an dieser Stelle angegeben werden:

- In der UML gibt es noch die Implementierungsdiagramme (Komponenten- und Einsatzdiagramme)[54]. Sie zeigen die physischen Aspekte eines objektorientierten Systems, wie etwa die Beziehungen zwischen Dateien in einem System oder die Architektur eines Hardwaresystems[55].
- In der Strukturierten Analyse (SA) gibt es durch Anwendung des sogenannten *Hierarchiekonzepts* auf der höchsten Abstraktionsebene (dem Gesamtsystem) eine Sonderform des Datenflussdiagramms, das Kontextdiagramm[56]. In der tiefsten Ebene der Abstraktion spricht man von MiniSpecs (Pseudo Code bzw. Entscheidungstabellen)[57].
- Durch das Hinzufügen von Kontrollflüssen bei den Datenflussdiagrammen erhält man in der Echtzeitanalyse (Real Time Analysis, RT) die Flussdiagramme[58]. MiniSpecs werden in der RT PSpecs genannt[59], diese werden abhängig von Zuständen aktiviert, die in CSpecs (Zustandsautomat oder Entscheidungstabelle) festgehalten werden[60].

2.9 Zusammenfassung der Ergebnisse

Die oben vorgestellten Notationsformen haben alle gemeinsam, dass sie die Modellierung von Softwaresystemen unterstützen.
Für die OOA sind einige Notationsformen eher von Nutzen als andere. Die nachstehende Tabelle soll dies veranschaulichen:

[54] vgl. Stahlknecht/Hasenkamp, a.a.o., S. 283
[55] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 443
[56] vgl. Balzert, a.a.o., S. 432 ff.
[57] vgl. Balzert, a.a.o., S. 443
[58] vgl. Balzert, a.a.o., S. 455
[59] vgl. Balzert, a.a.o., S. 459
[60] vgl. Balzert, a.a.o., S. 457 ff.

Diagrammtyp	hoher Nutzen	mittlerer Nutzen	geringer Nutzen	kein Nutzen
Klassendiagramm	X			
Geschäftsprozessdiagramm	X			
Automaten	X			
Interaktionsdiagramme	X			
Objektdiagramm		X		
Entscheidungstabellen			X	
Regeln			X	
Data Dictionary			X	
Datenflussdiagramm				X
Funktionsbaum				X
ER-Diagramm				X

Tabelle 2.9-1: Nutzenwert der verschiedenen Notationen zur OOA

Unbedingt notwendig in der OOA sind Geschäftsprozess- und Klassendiagramme. Geschäftsprozesse beschreiben die grundlegenden Anforderungen an ein System, das Klassendiagramm dessen statische Elemente.
Ebenso wichtig sind Interaktionsdiagramme und Automaten. Mit Automaten ist es überhaupt erst möglich, lauffähige Systeme zu entwickeln. Sie ersetzen in der UML die Programmablaufpläne (PAP) und Struktogramme. Allerdings ist zu entscheiden ob man endliche Zustandsautomaten oder Petri-Netze einsetzen will. Dies hängt davon ab, ob man es mit einer deterministischen endlichen (Zustandsautomat) oder nichtdeterministischen nebenläufigen Problemstellungen (Petri-Netz) zu tun hat.
Interaktionsdiagramme ergänzen die Klassendiagramme um die dynamischen Elemente der Objektorientierung. Hier ist zwischen dem Einsatz von Sequenz- oder Kollaborationsdiagrammen zu wählen.

Objektdiagramme sind immer dann von Nutzen, wenn das Verhalten von Objekten in bestimmten Situationen modelliert oder untersucht werden soll.

Mit Hilfe von Entscheidungstabellen und Regeln lassen sich Endanwender in den Entwicklungsprozess einbinden. Sie können mit diesen Hilfsmitteln sehr leicht Anforderungen an ein System beschreiben.
Data Dictionary Einträge sind bei sehr komplexen Datenstrukturen nützlich. Sie können die Beschreibung der Klassenattribute ergänzen bzw. Daten beschreiben, die in irgendeiner Form gespeichert werden müssen.

Funktionsbaum, Datenflussdiagramm und ER-Diagramm sind normalerweise nicht zwingend in der OOA einzusetzen.
Die Strukturierung der Funktionen bzw. Operationen wie im Funktionsbaum wird durch die Klassenstruktur im Klassendiagramm erreicht. Das ER-Diagramm kann ebenfalls durch ein Klassendiagramm, ergänzt um datenbankspezifische Details, ersetzt werden.
Die Funktionalität des Datenflussdiagramms, Schnittstellen und Speicher zu kennzeichnen, sowie die Umwelt eines Systems darzustellen, wird arbeitsteilig von Klassen- und Geschäftsprozessdiagrammen übernommen. Letztere zeigen ein System in seiner Umwelt; Schnittstellen und Speicher können durch Akteure und Klassen repräsentiert werden.

3 Entwurf der Anwendung

3.1 Auswahl der Hilfsmittel

Im folgenden wird nun unter Berücksichtigung der vorherigen Ergebnisse die in Abschnitt 1.1 kurz vorgestellte Anwendung modelliert.
Dabei werden ausgehend von den Geschäftsprozessen mit Hilfe von Klassendiagrammen zunächst die statischen Teile des Systems entworfen. Anschließend werden die persistenten Klassen in einem erweiterten Klassendiagramm zur Datenmodellierung herangezogen.
Unter welchen Kriterien eine Zeichenkette in einer HTML-Datei als Stichwort gelten soll, wird durch Regeln formuliert.
Schließlich wird das spezielle Verhalten von verschiedenen Anwendungsfällen bzw. Geschäftsprozessen und diverse Algorithmen mittels Zustandsautomaten und Interaktionsdiagrammen dargestellt.
Für alle folgenden Abschnitte gilt, dass, im Gegensatz zum Reverse-Engineering, Forward-Engineering betrieben wird. Es werden also Modelle zur Code-Generierung erstellt.[61]

3.2 Geschäftsprozesse identifizieren

Die zu entwickelnde Anwendung ist zweigeteilt: Das eine Modul findet Stichworte in bestehenden HTML-Dateien und erstellt daraus ein Archiv; mit dem zweiten Modul lassen sich HTML-Seiten über die Stichworte im Archiv suchen. Entsprechend heißen die beiden Teile WortFinder und WortSucher. Dabei wird der WortFinder vom Webseitenbetreiber eingesetzt, der WortSucher von Besuchern der Webseite. Beide Module beinhalten jedes für sich eine Reihe von weiteren Geschäftsprozessen und werden daher als Kollaborationen dargestellt:

[61] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 101

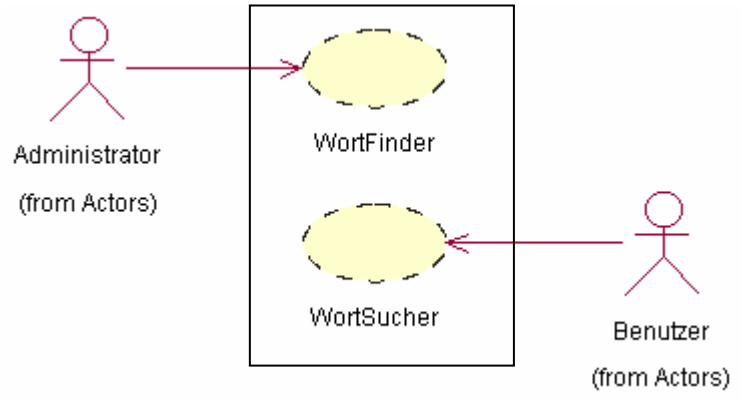

Abb. 3.2-1: Aufteilung des Systems in zwei Hauptgeschäftsprozesse

Gegenstand dieser Studienarbeit ist jedoch nur der Entwurf des Wort-Finders. Trotzdem ist die Existenz des zweiten Moduls zu berücksichtigen, da Schnittstellen zwischen den beiden Systemen geschaffen werden müssen, damit beide Module auf die gleichen Daten zugreifen können.

Das nächste Schaubild zeigt nun die Geschäftsprozesse des WortFinders. Die Systemgrenze wurde weggelassen, da sich lediglich der Administrator außerhalb des Systems befindet.
Der Geschäftsprozess „Ergebnisse sortieren“ ist dabei ein Ergebnis der Beachtung des WortSuchers: Sind die Stichworte alphabetisch sortiert, könnte der Suchvorgang beschleunigt werden.

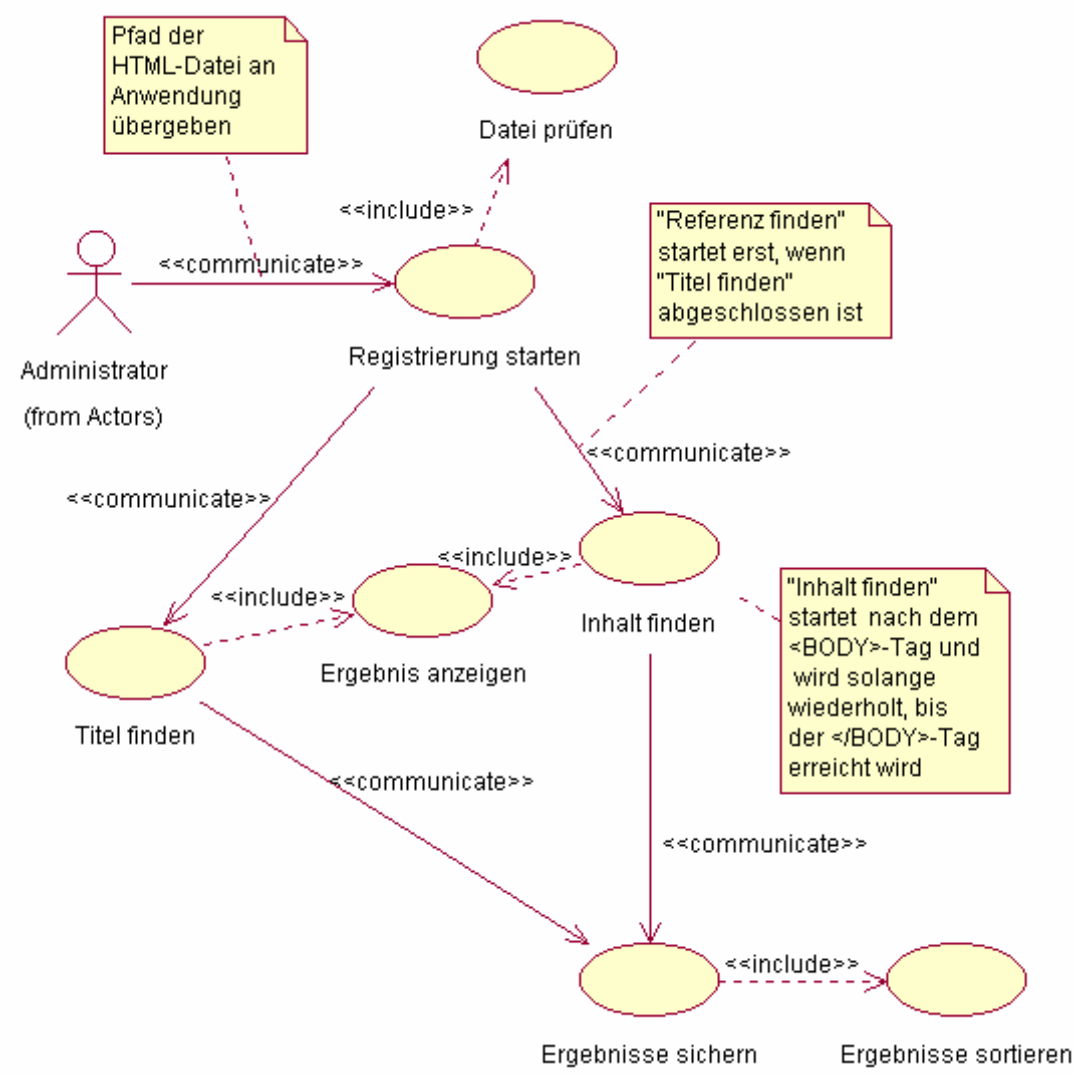

Abb. 3.2-2: Geschäftsprozessdiagramm – Wortfinder

3.3 Klassendefinition

Aus der Betrachtung der Geschäftsprozesse lässt sich schon auf verschiedene Klassen mit ihren Attributen und Operationen schließen: So wird eine *Datei* geprüft, diese Datei hat einen *Titel,* es werden *Inhalte* gefunden und irgendetwas startet den Prozess der Stichwortsuche. Beim Sichern der Ergebnisse entsteht eine Art *Datensatz*.

Daraus ergeben sich die Klassen „Datensatz“, „Datei“, „Inhalt“ und die aktive Klasse „Stichwortfinder“, die Objekte der zuvor genannten Klassen erzeugt. Zudem stellen *Stichwörter* und *Bilder* jeweils *Inhalt einer Datei* dar, womit zwei weitere Klassen und eine Beziehung gefunden wären. Die

genaue Struktur der Elemente zeigt das unten stehende Klassendiagramm.

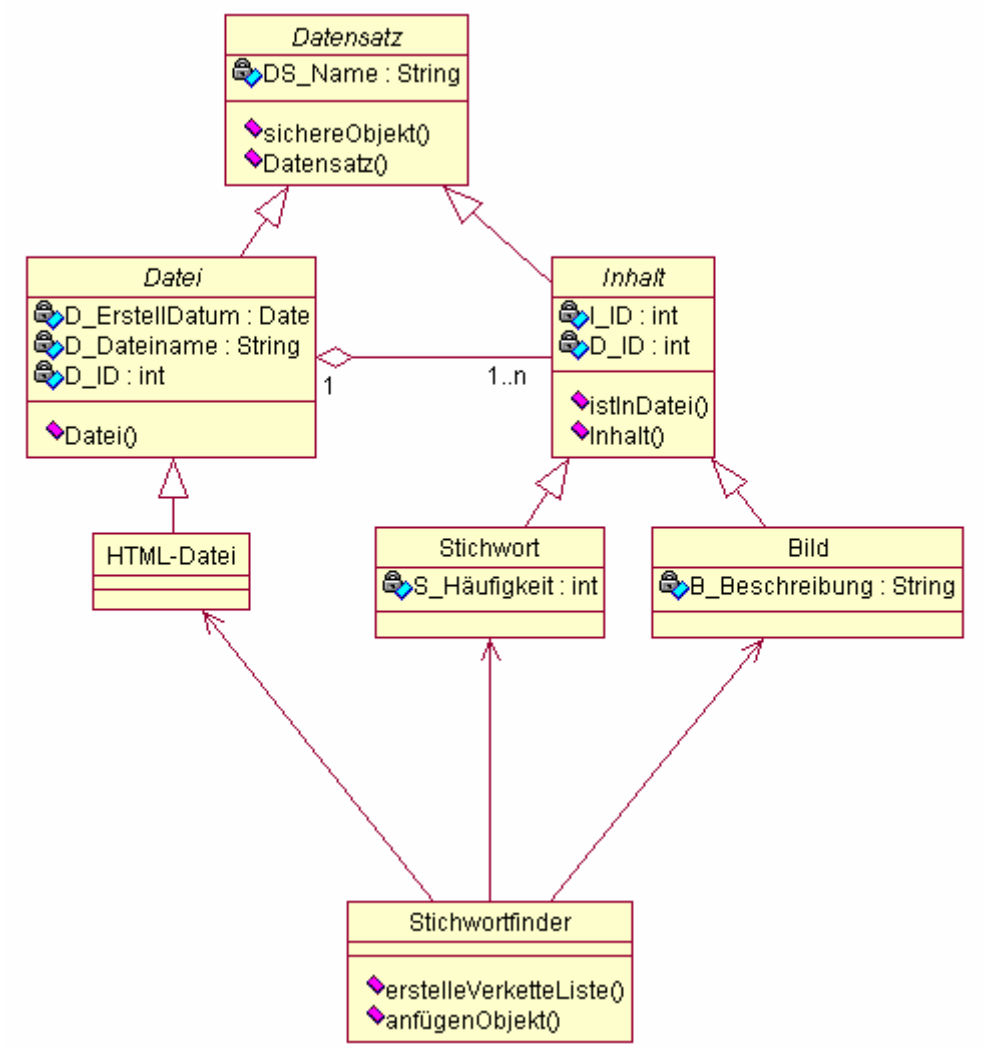

Abb. 3.3-1: Klassendiagramm – WortFinder

Die Klassen „Datensatz“, „Datei“ und „Inhalt“ sind als abstrakte Klassen gekennzeichnet, „HTML-Datei“, „Stichwort“ und „Bild“ als persistent.
Die Klasse „HTML-Datei“ ist ein Sonderfall: Sie erbt sämtliche Operationen und Attribute der Klasse „Datei“, enthält aber keine eigenen Attribute oder Operationen. Dies ist beabsichtigt, da in Zukunft eventuell auch andere Dateiformate bearbeitet werden sollen, die zusätzliche Eigenschaften aufweisen. Durch diesen Entwurf bleibt die Klasse Datei abstrakt und das Modell kann an dieser Stelle leicht erweitert werden.

Konstruktoren und Operationen zur Datenkapselung (get- und set-Operationen) sowie zur Objektverwaltung werden meist nicht im Klassendiagramm aufgeführt[62]. Im obigen Diagramm werden sie nur in besonderen Fällen zur Veranschaulichung bzw. Konsistenzbewahrung mit anderen Diagrammen aufgeführt.

3.4 Datenmodellierung

Die persistenten Klassen „HTML-Datei“, „Stichwort“ und „Bild“ dienen als Grundlage für die weitere Datenmodellierung.

Geht man nach der von Booch, Rumbaugh und Jacobson vorgeschlagenen Methode vor[63], erhält man das folgende Diagramm:

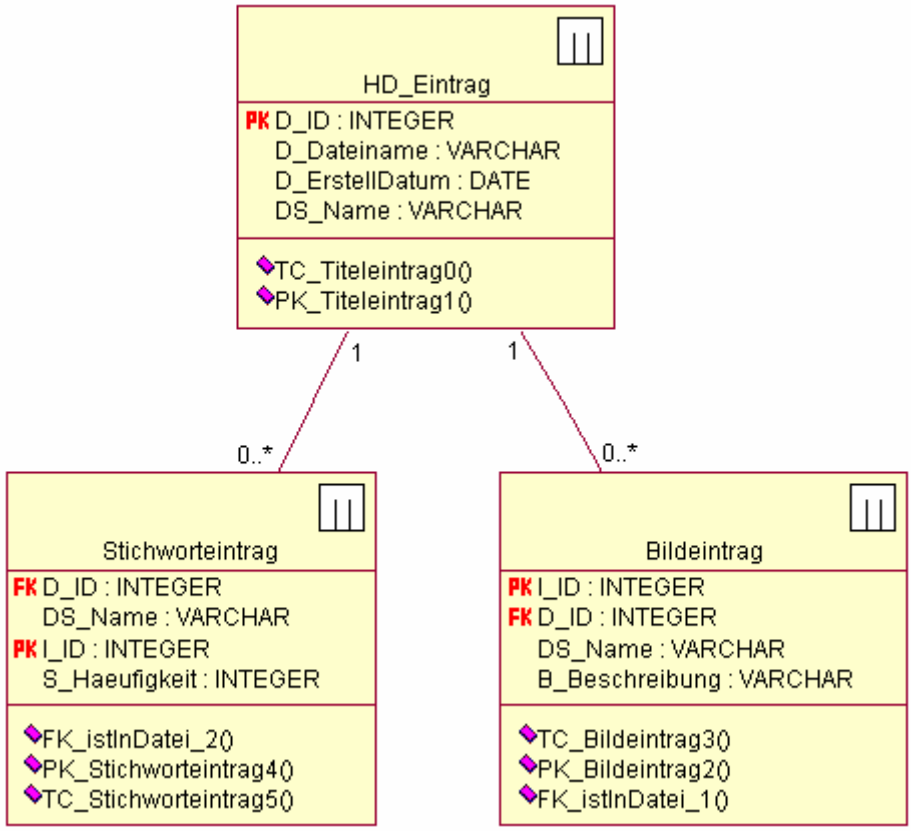

Abb. 3.4-1: Datenmodell - WortFinder

[62] vgl. Balzert, a.a.o., S. 174
[63] vgl. Booch/Rumbaugh/Jacobson, a.a.o., S. 123 f.

Die von den Klassen „Inhalt“ und „Datei“ jeweils vererbte Aggregation zwischen den Klassen bleibt als Fremdschlüsselbeziehung im Datenmodell erhalten.
Pro Klasse wird ein Speicherbereich benötigt. Dies können drei Tabellen in einer Datenbank oder aber drei Textdateien sein, in die z.B. pro Zeile ein Datensatz geschrieben wird. Für diese Anwendung sollen die Daten in Dateien gespeichert werden.

3.5 Regeln zur Texterkennung

3.5.1 Namenskonvention für Regeln

In den folgenden Abschnitten werden Regeln benutzt, um Begriffsdefinitionen und Anforderungen an die Anwendung zu formulieren. Dabei erfolgt die Formulierung der Regeln nach dem Wenn-dann-Schema.
Zur einfacheren Unterscheidung werden Regeln und Definitionen mit Namen versehen. Regeln erhalten als Namen ein „R“ + eine laufende Nummer, Definitionen ein „D“ + eine laufende Nummer und Ausnahmen den Namen der Regel, ein „A“ und eine weitere laufende Nummer.

3.5.2 Regeln zum Auffinden von Stichwörtern

Die folgenden Regeln dienen dazu, Kriterien festzulegen, wann das System eine Zeichenfolge als Stichwort deklarieren soll. Vor der Entwicklung der Regeln müssen erst einige Definitionen erstellt werden.

Definitionen:

- D1 = EOL (End Of Line): entspricht dem Einrücken in die nächste Zeile (Betätigen der Return- oder Eingabe-Taste)
- D2 = EOF (End Of File): entspricht dem Datei-Ende
- D3 = Blank (Leerzeichen): entspricht dem Betätigen der Leertaste
- D4 = Tag-Klammern: die Zeichen „<“ und „>“ schließen Tag-Anweisungen in HTML-Dokumenten ein.
- D5 = Trennzeichen sind die folgenden Zeichen:

  - Interpunktionszeichen {„!“, „?“, „:“, „.“, „,“, „;“}
  - Klammern {„(“, „)“, „[“, „]“, „{“, „}“}
  - Anführungszeichen oben und unten, sowie Hochkommata {„‘“,’}
  - Mathematische Zeichen {„+“, „=“, „-“, „/“, „*“}
  - Striche {„|“, „_“}
  - Sonderzeichen { Blank, EOL, EOF, Tag-Klammern }
- D6 = Großbuchstaben sind in der folgenden Menge enthalten: {„A“, „B“, „C“, „D“, „E“, „F“, „G“, „H“, „I“, „J“, „K“, „L“, „M“, „N“, „O“, „P“, „Q“, „R“, „S“, „T“, „U“, „V“, „W“, „X“, „Y“, „Z“, „Ä“, „Ö“, „Ü“, „1“, „2“, „3“, „4“, „5“, „6“, „7“, „8“, „9“, „0“}

Regeln:

- R1 = Wenn ein Stichwort gefunden wurde, das in der gleichen Datei, d.h. im aktuellen Prozess, schon einmal registriert wurde, wird nur der Häufigkeitsanzeiger des bestehenden Stichwort-objekts, also der Wert des Attributs S_Haeufigkeit, um eins erhöht.
- R2 = Wenn zwischen zwei Trennzeichen mehr als zwei Zeichen stehen, die keine Trennzeichen sind, und das erste Zeichen ein Großbuchstabe ist, dann handelt es sich bei der Zeichenfolge zwischen den Trennzeichen um ein Stichwort.

Ausnahmen:

- R2A1 = Wenn das Trennzeichen vor einer Zeichenfolge das Zeichen „.“ ist, dann wird die anschließende Zeichenfolge ignoriert und die Suche wird nach dem nächsten Trennzeichen fortgesetzt.
- R2A2 = Wenn auf die Tag-Klammer „<“ keine der unten angegebenen Zeichenfolgen folgt, wird die Suche nach Stichworten nach der Tag-Klammer „>“ fortgesetzt.
  - Folgt auf die Tag-Klammer „<“ die Zeichenfolge „img“, werden die Inhalte „title“, „alt“ und „src“ auf Stichworte hin untersucht.
  - Folgt auf die Tag-Klammer „<“ die Zeichenfolge „/body“, wird die Suche nach Stichworten beendet.

- Folgt auf die Tag-Klammer „<“ die Zeichenfolge „a“, wird der Inhalt „href“ ausgelesen und als Vorschlag der weiteren Registrierung an den Benutzer ausgegeben.

3.5.3 Regeln für das Auffinden eines Titels

Das Suchen nach dem Titel eines HTML-Dokuments erfolgt vor dem Suchen nach Stichwörtern. Hier gelten die folgenden Regeln:

- R3 = Wenn in einem HTML-Dokument die Zeichenfolge „<title>“ notiert ist, beginnt der Titel der Datei unmittelbar nach dieser Zeichenfolge und endet unmittelbar vor der Zeichenfolge „</title>“ oder dem nächsten „<“-Zeichen.
- R4 = Wenn die Zeichenfolge „<title>“ nicht notiert ist, wird als Titelangabe die Zeichenfolge „- Kein Titel angegeben -“ übernommen.

3.5.4 Textverarbeitung

In HTML-Dokumenten müssen bestimmte Zeichen wie Umlaute maskiert werden, damit sie im Browser korrekt dargestellt werden. Für das Archiv, das mit dem WortFinder aufgebaut werden soll, muss diese Maskierung wieder rückgängig gemacht werden. Die folgende Tabelle soll einen kurzen Eindruck geben, welcher Art die Zeichenfolgen sind, und wie sie wieder „demaskiert“ werden müssen. Eine gute Übersicht über die HTML-Zeichenreferenz findet sich in der Onlineausgabe von SelfHTML des Autors Stefan Münz[64].

maskierte Zeichen	demaskierte Zeichen
ß	ß
ä	ä
ö	ö
ü	ü
Ä	Ä
Ö	Ö
Ü	Ü
©	Copyright

Tabelle 3.5-1: Maskierte Zeichen in HTML-Dokumenten

[64] vgl. Münz, Stefan: SelfHTML, Version 8.0 vom 27.10.2001, http://selfhtml.teamone.de/html/referenz/zeichen.htm

3.6 Geschäftsprozessmodellierung

Im folgenden werden Angaben gemacht, wie die einzelnen Geschäftsprozesse (GP) auszugestalten sind. Bei manchen Geschäftsprozessen werden Beispiele für Umsetzungsmöglichkeiten in Java angegeben.

Operation „erstelleVerketteteListe“ bzw. GP „Ergebnisse sortieren“

Zur temporären Datenhaltung ist es nötig, verkettete Listen zu erstellen. Java bietet die Möglichkeit an, solche Listen zu implementieren, und stellt gleichzeitig komfortable Methoden, wie z.B. zum Anfügen und Löschen, bereit, um auf die Elemente der Liste zuzugreifen[65]. Außerdem sind sogar Sortieralgorithmen für Listen und Sammlungen als Methoden in Java verfügbar[66].

GP „Ergebnisse sichern“

Die einzelnen Listen werden in Dateien gespeichert; pro Liste existiert eine Datei. Das Ablegen kann in Zeilen (pro Zeile ein Objekt, Attributwerte durch Trennzeichen getrennt) oder in Datensätzen (pro Datensatz ein Objekt) erfolgen. In Java stehen für beide Möglichkeiten entsprechende Streams zur Verfügung[67].

GP „Titel finden“ bzw. „Inhalt finden“

Diese Geschäftsprozesse werden durch die Regeln R3 und R4 (Titel finden) sowie den Regeln R1 und R2 inklusive der zugehörigen Ausnahmen (Inhalt finden) beschrieben.

GP „Registrierung starten“

Für diesen Geschäftsprozess wurde ein Aktivitätsdiagramm erstellt, da die modellierten Zustände Aktivitätszustände sind. Aus Gründen der Über-

[65] vgl. Horstmann, Cay/Cornell, Gary: Core Java 2, Band 2 – Expertenwissen, München, 2002, S. 152 ff.
[66] vgl. ebenda, S. 206 ff.
[67] vgl. Horstmann, Cay/Cornell, Gary: Core Java 2, Band 1 – Grundlagen, München, 1999, S. 773 ff. und S. 788 ff.

sichtlichkeit wurden manche Aktivitäten zusammengefasst und vereinfacht dargestellt.

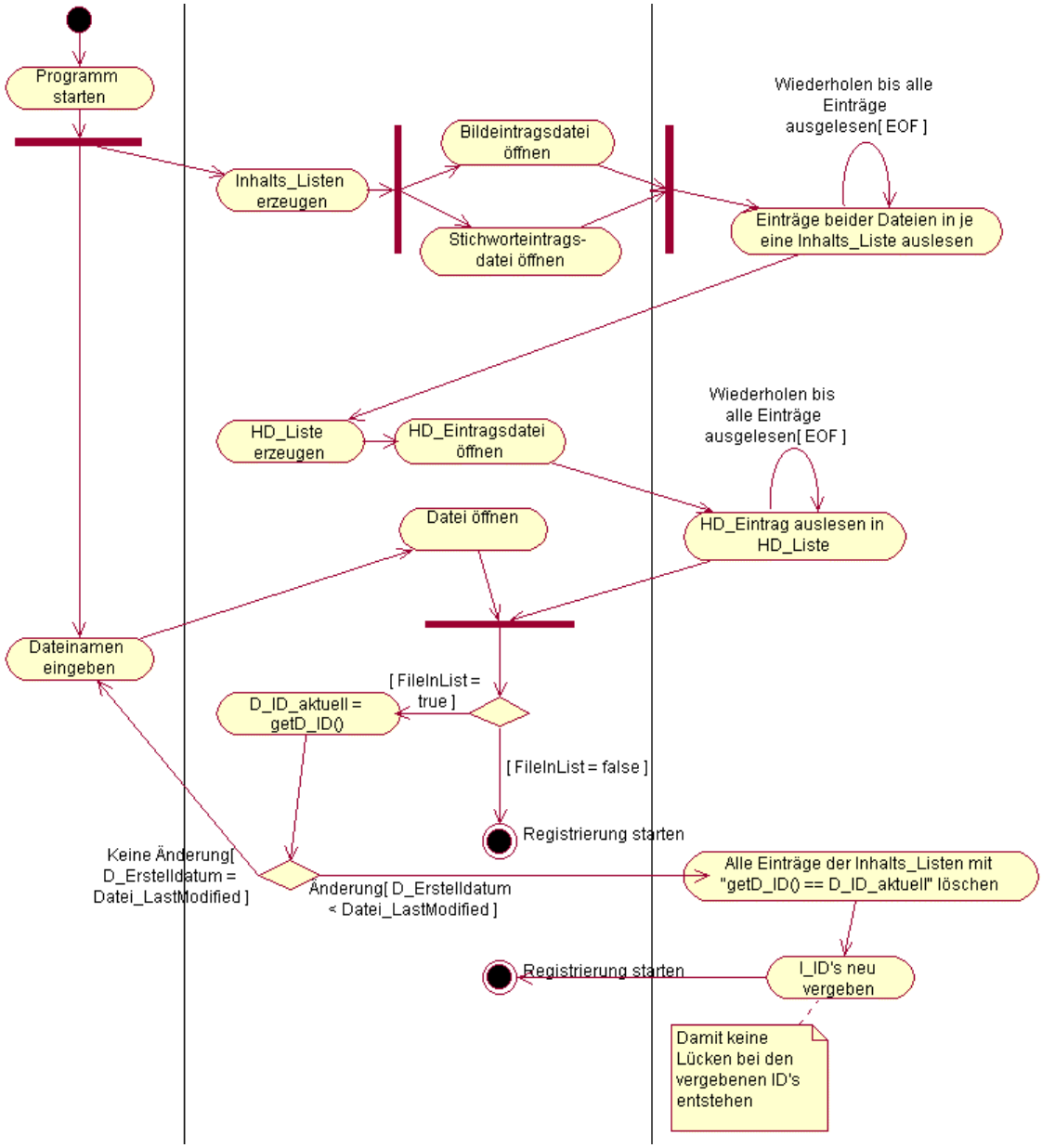

Abb. 3.6-1: Aktivitätsdiagramm zum GP "Registrierung starten"

Hinzukommt, dass der Prozess „Registrierung starten“ auch die Definitionen des Regelteils umsetzen muss. Für die einzelnen Mengen können Arrays gebildet werden. Die Definitionen D1 bis D3 müssen den Gegebenheiten der jeweiligen Programmiersprache angepasst werden, z.B.

in Java gilt „Blank“ = „ “, „EOL“ = „/n“ (Escape-Sequenz)[68], „EOF“ löst, wenn es erreicht wird, eine „IOException“ aus[69].
Zur Prüfung nach der letzten Änderung bzw. ob die vom Administrator angegebene Datei überhaupt existiert, wäre in Java die Methode „lastModified()“ der Klasse „java.io.File“ anzuwenden[70].

GP „Ergebnis anzeigen“

Die Anzeige kann für einen ersten Prototyp auf der Konsole erfolgen, wenn überhaupt. Als Erfolgskontrolle können auch die Archiv-Dateien mit einem Editor oder Tabellenkalkulationsprogramm eingesehen werden.

3.7 Interaktion der Klassen

Da die Anwendung bislang keine Nebenläufigkeit oder stark zeitbezogene Komponenten aufweist, ist der Einsatz eines Sequenzdiagramms nicht notwendig.
Allerdings lässt sich mit Hilfe eines Kollaborationendiagramms sagen, in welcher Beziehung die Objekte zur Laufzeit stehen.

Im Abb. 3.7-1 wurden die gesendeten Botschaften wegen der besseren Übersichtlichkeit nicht an den Pfeilen sondern unter den Empfängerobjekten notiert.
Zuerst einmal wird deutlich, dass nur von den persistenten Klassen und der aktiven Klasse Stichwortfinder Objekte erzeugt werden. Alle anderen Klassen sind abstrakt.
Das Objekt „wortFinder“ ist eine Instanz der aktiven Klasse „Stichwortfinder“, die mit verschiedenen Instanzen der Klasse „HTML-Datei“, „Bild“ und „Stichwort“ kommuniziert.
Die Reihenfolge der gesendeten Botschaften ist an den Nummern abzulesen:

[68] vgl. Horstmann/Cornell 1999, a.a.o., S. 73
[69] vgl. Horstmann/Cornell 1999, a.a.o., S. 777
[70] vgl. Horstmann/Cornell 1999, a.a.o., S. 821

(1) Es wird eine verkettete Liste für HTML-Dateien in der aktiven Klasse erstellt. Die HD_Eintragsdatei wird nun ausgelesen.

(2) Für einen Dateieintrag wird ein neues HTML-Datei-Objekt erstellt.

(3) Das Objekt wird an die Liste angefügt.

(4) Sind alle HTML-Datei-Objekte inklusive dem Objekt, das durch den neuen Registrierungsprozess entstanden ist, in der Liste, werden alle einzelnen Elemente wieder in die Datei gesichert.

Ähnlich ist der Ablauf für die Stichwort- und Bild-Objekte (Nummer 5 bis 10), jedoch wird hier die Liste nur für die neu registrierten Stichwörter und Bilder angelegt.

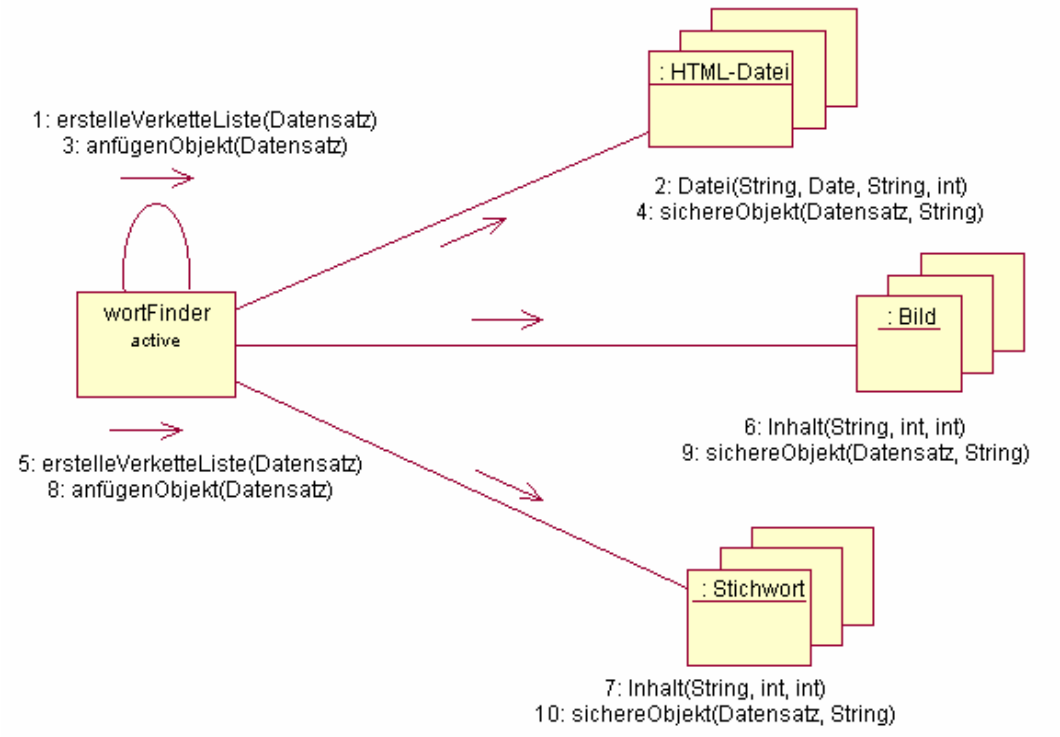

Abb. 3.7-1: Kollaborationendiagramm - WortFinder

4 Wertung der Ergebnisse

Das Ziel, ein System zu modellieren, das reif für die Implementierung ist, wurde nicht verfehlt.

Der Entwicklungsprozess, der zu einer objektorientierten, lauf- und auslieferungsfähigen Anwendung führt, wird als evolutionär und iterativ beschrieben. Das mit dieser Arbeit vorliegende Modell stellt dabei das OOA-Modell 1, die erste Iteration, dar. Nach dessen Implementierung können Schwachstellen und eventuelle Ungenauigkeiten entdeckt und das vorhandene Modell entsprechend verbessert und ergänzt werden.[71]

Es war sinnvoll, die verschiedenen Diagrammtypen und Notationsformen unabhängig von ihrer Zugehörigkeit zu einer bestimmten Modellierungsmethode (OOA, SA oder RT) zu betrachten.

Auch außerhalb der UML und außerhalb der OOA gibt es Darstellungsformen, die die Standards der UML und OOA, abhängig von der Aufgabenstellung, ergänzen können. Als Beispiele sollen Petri-Netze und die in dieser Arbeit angewandte Form der Regeln dienen:

- Petri-Netze können (nicht-deterministische) Problemstellungen beschreiben, die von Zustandsautomaten aufgrund ihrer deterministischen Eigenschaften nicht beschreibbar sind.
- Mit einer relativ simplen Technik (Wenn-dann-Form) lassen sich Anforderungen an Anwendungssysteme formulieren, die sowohl für Endanwender verständlich, als auch für Programmierer einfach in Programmcode umsetzbar sind.

[71] vgl. Balzert, a.a.o., S. 388

5 Ausblick

Um die entstehenden Archiv-Dateien für die Besucher von Webseiten nutzbar zu machen, muss im nächsten Schritt auch der WortSucher modelliert werden. Dabei könnten Entwicklungsschritte gemacht werden, die Auswirkungen auf das Modell des WortFinders haben.
Beide Modelle werden natürlich in der nächsten Phase implementiert und getestet. Fehler und Schwachstellen die hier entdeckt werden, führen direkt zu einer weiteren Iteration des objektorientierten Entwicklungsprozesses.
Außerdem sind diverse Erweiterungen der Anwendung möglich und zu diskutieren:

- Es könnten auch andere Dokumentarten als HTML-Dokumente nach Stichworten durchsucht werden (siehe Abschnitt 3.3).
- Es wäre möglich mehrere Instanzen der Klasse „Stichwortfinder“ zu erzeugen, um entsprechend viele Dokumente nebenläufig zu bearbeiten.
- Statt Webseiten, die auf dem eigenen Rechner liegen, könnte man auch Webseiten auf Servern durchsuchen. Statt einem Dateipfad müsste dann ein URL an die Anwendung übergeben werden, anschließend die entsprechende Datei per Internet abgerufen und durchsucht werden.

Seit etwa Mitte 2001 wird an einer neuen Version von UML gearbeitet. Es könnten also, wenn UML 2.0 verabschiedet und als neuer Standard etabliert sein wird, was noch eine Weile dauern kann[72], Anpassungen der bestehenden Diagramme an eine eventuell neue bzw. leicht veränderte Semantik und Syntax der UML nötig sein.[73]

[72] Alexander, Sascha: UML 2.0 wird zum Politikum, in: Computerwoche, Heft 3 vom 17.01.2003, http://www.computerwoche.de/index.cfm?pageid=255&artid=44876
[73] Object Management Group (o.V.): Introduction into OMG UML, http://www.omg.org/gettingstarted/what_is_uml.htm

I. Abbildungsverzeichnis

A. Abbildungen

B. Tabellen

II. Literaturverzeichnis

A. Selbständige Bücher und Schriften

Balzert, Helmut
Lehrbuch der Softwaretechnik – Software-Entwicklung
2. Auflage
Heidelberg/Berlin 2000

Booch, Grady/Rumbaugh, Jim/Jacobson, Ivar:
Das UML-Benutzerhandbuch
Bonn 1999

Horstmann, Cay/Cornell, Gary 1999
Core Java 2, Band 1 – Grundlagen
4., überarbeitete Fassung
München 1999

Horstmann, Cay/Cornell, Gary 2002
Core Java 2, Band 2 – Expertenwissen
5., überarbeitete Fassung
München 2002

Stahlknecht, Peter/Hasenkamp, Ulrich:
Einführung in die Wirtschaftsinformatik
10., überarbeitete und aktualisierte Auflage
Berlin/Heidelberg 2002

B. Zeitschriftenaufsätze

Alexander, Sascha:
UML 2.0 wird zum Politikum
in: Computerwoche
Heft 3 vom 17.01.2003

C. Onlinequellen

Münz, Stefan
SelfHTML – HTML-Dateien selbst erstellen
Version 8.0 vom 27.01.2001
http://selfhtml.teamone.de
Besucht am 15.02.2003

Object Management Group
UML Resource Page
Version vom 13.02.2003
http://www.omg.org/uml
Besucht am 16.02.2003

III. Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich,

1. dass ich meine Studienarbeit zum Thema „Objektorientierte Softwareentwicklung am Beispiel einer Offline-Suche“ ohne fremde Hilfe angefertigt habe;

2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe;

3. dass ich meine Studienarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Albisheim, den 17.02.2003 _____