

INSTITUT FÜR TECHNISCHE INFORMATIK
RECHNERGESTÜTZTE WISSENSVERARBEITUNG
UNIVERSITÄT HANNOVER

Rollenbasierter Entwurf am Beispiel eines benutzeradaptierbaren Hyperbooks

Studienarbeit

von

Bernd Horle

1. Prüfer: Prof. Dr. techn. W. Nejdil

2. Prüfer: Prof. Dr.-Ing. C. Müller-Schloer

Betreuer: Dipl.-Inform. Dr. techn. F. Steimann

Inhaltsverzeichnis

1. Einleitung.....	1
2. Die Modelliersprache UML und Modellierungswerkzeuge.....	2
2.1. UML	2
2.1.1. Die Geschichte von UML.....	2
2.1.2. Der Aufbau von UML	3
2.1.2.1. Das Metamodell.....	4
2.1.2.2. Die Notation	5
2.2. Modellierungswerkzeuge von UML.....	6
2.2.1. Modellierungswerkzeuge allgemein.....	6
2.2.2. Arbeiten mit Together J.....	7
3. Entwurf des Hyperbooks anhand des aktuellen Metamodells.....	9
3.1. Aufbau des Hyperbooks und weitere Vorgehensweise	9
3.2. Use cases.....	10
3.3. Klassendiagramme.....	11
3.3.1. Package Textbase	11
3.3.2. Package Vorlesung	12
3.3.3. Package Notizen	13
3.3.4. Package Indexsearch.....	14
3.3.5. Package Classifikation.....	14
3.3.5.1. Aufbau des Package Classifikation	14
3.3.5.2. Erklärung des Package Classifikation anhand eines Beispiels.....	16
3.4. Sequenzdiagramme.....	17
3.4.1. Vorlesung bearbeiten.....	17
3.4.2. Vorlesung hören	19
3.4.3. Notizen bearbeiten.....	20

3.4.4. Schlagwortsuche	21
3.4.5. Kategorische Suche	22
4. Rollenbasierte Modellierung und Änderungen des Metamodells	24
4.1. Der Rollenbegriff allgemein	24
4.2. Rollenbegriffe in der aktuellen UML Version	24
4.3. Interfaces in der aktuellen Version	26
4.4. Änderung des Metamodells	27
4.5. Arbeiten mit dem geänderten Metamodell	28
4.5.1. Klassendiagramme	28
4.5.2. Kollaborationsdiagramm	30
4.5.3. Bedeutung für die Praxis	30
5. Entwurf des Hyperbooks nach dem geänderten Metamodell	31
5.1. Package Textbase	31
5.2. Package Vorlesung	32
5.3. Package Notizen	33
5.4. Package Indexsearch	34
5.5. Package Classification	35
6. Vergleich der Entwürfe und Beurteilung	39
6.1. Vergleich vom konventionellen und dem rollenbasierten Entwurf des Hyperbooks	39
6.2. Beurteilung des rollenbasierten Entwurfes	47
7. Zusammenfassung und Fazit	50
8. Literaturverzeichnis	52

1. Einleitung

In der heutigen Gesellschaft nehmen Rollen eine immer größere Bedeutung ein, sie kommen in vielen Bereichen des täglichen Lebens vor. Auch in der Softwaremodellierung gewinnen die Rollen in einigen Modellierungskonzepten immer mehr an Bedeutung. So finden die Rollen in einigen Konzepten wie z.B. in der OOram Software Engineering Method von Reenskaug [Reenskaug] rege Anwendung.

In der Modellierungssprache UML (Unified Modeling Language), die sich in den letzten Jahren als Standard durchgesetzt hat, führen sie dagegen ein Schattendasein. Die in UML vorhandenen Rollenkonzepte werden von den Softwaremodellierern eher sporadisch oder gar nicht eingesetzt.

Um den Rollen mehr Beachtung zukommen zu lassen, wurde ein Rollenkonzept [Steimann, Habilitationsschrift] entwickelt, welches den Einsatz von Rollen im Entwurf vorschreibt. Im Rahmen dieser Arbeit sollen die Unterschiede gegenüber dem aktuellen Standard an einem Beispiel aufgezeigt und bewertet werden. Als Beispiel dient hierbei ein benutzeradaptierbares Hyperbook.

So wird im 2.Kapitel ein kurzer Überblick über die Entstehung und den Aufbau von UML gegeben. Des Weiteren werden einige Modellierungswerkzeuge, die UML unterstützen, vorgestellt und insbesondere das im Rahmen dieser Arbeit verwendete Tool betrachtet.

Um das Rollenkonzept mit dem aktuellen UML-Standard vergleichen zu können, wird sowohl ein Entwurf nach dem heutigen Standard als auch ein rollenbasierter Entwurf durchgeführt. Das 3.Kapitel beinhaltet daher die Darstellung der Funktionsweise des Hyperbooks und die Modellierung nach dem aktuellen UML Standard. Im 4.Kapitel werden dann die für den rollenbasierten Entwurf notwendigen Änderungen im Metamodell und der dadurch notwendige geänderte Gebrauch der UML-Notation aufgezeigt. Der rollenbasierte Entwurf des benutzeradaptierbaren Hyperbooks ist im 5.Kapitel dargestellt und erklärt. Anschließend werden die Ergebnisse der beiden Entwürfe miteinander verglichen und die Unterschiede der beiden Entwürfe aufgezeigt. Die beim rollenbasierten Entwurf entstehenden Vorteile werden gegenüber den bestehenden Nachteilen abgewogen. Ausgehend von den Ergebnissen wird eine Bewertung auf Praxistauglichkeit vorgenommen. Dieser Vergleich und die Beurteilung des rollenbasierten Entwurfes erfolgen im 6.Kapitel. Anschließend wird im 7.Kapitel die Arbeit zusammengefasst und ein Fazit gezogen.

2. Die Modelliersprache UML und Modellierungswerkzeuge

2.1. UML

Bei UML (Unified Modeling Language) handelt es sich um eine grafische Modelliersprache, die in der heutigen Softwareentwicklung weite Verbreitung findet. Sie bildet den heutigen Standard und wird von der OMG (Objekt Management Group: Gremium von den bedeutendsten Softwareentwicklungsfirmen) laufend überarbeitet und den wachsenden Bedürfnissen angepasst.

2.1.1. Die Geschichte von UML

In den 80er Jahren kam die objektorientierte Programmierung auf, die objektorientierten Sprachen Smalltalk und C++ fanden immer mehr Verwendung. Ende der 80er und am Anfang der 90er wurden verstärkt Analyse- und Entwurfsmethoden entwickelt. Es gab eine Reihe von Softwareentwicklern, die mit ihren Ansätzen und Methoden den Anfang der objektorientierten Analyse- und Entwurfsmethoden mitgeprägt haben. Hier sind für die Entstehung von UML insbesondere Booch, Rumbaugh und Jacobsen zu erwähnen. Aber auch Shlaer / Mellor, Coad / Yourdon oder die von Wirfs-Brock, um nur einige zu nennen, lieferten ihren Beitrag dazu.

Zunächst entwickelte jeder seine Methoden für sich. 1994 wechselte Rumbaugh von General Electric zu Rational Software, wo auch Booch arbeitete. Die beiden begannen ihre Methoden zusammenzufassen und brachten eine gemeinsame Notation namens Unified Method (UM) heraus. Dies war der Grundstein für das spätere UML. Später stieß, durch den Kauf von Objectory durch Rational Software, dann noch Ivar Jacobsen zu den beiden. Jacobson brachte seine Use Cases mit ein. Die drei nannten sich „die drei Amigos“ [Schmuller].

Da die Methoden von Booch, Rumbaugh und Jacobsen zu dem Zeitpunkt sehr beliebt waren und einen hohen Marktanteil hatten, bildete das von den drei Amigos als Nachfolger von UM entwickelte UML einen Quasistandard in der Softwareentwicklung. Dies stieß auf großen Widerwillen bei den anderen großen Methodenentwickler. Sie waren der Meinung, dass UML nicht das letzte Wort sein könnte.

Um die verschiedenen Methoden der einzelnen Hersteller unter einen Hut zu bekommen und einen Standard in diesem Bereich einzuführen, wurde innerhalb der OMG die OMG Task Force gegründet. Auch früher hatte die Objekt Management Group schon versucht, einen

Standard durchzusetzen, war aber damals gescheitert, da jeder seine eigenen Konzepte und Methoden im Standard berücksichtigt sehen wollte. Der Vorsitzende der Task Force, Odell, war dieses Mal bereit, seine eigenen Methoden für einen gemeinsamen Standard aufzugeben, wollte jedoch keinen reinen Rational-Software-Standard.

Die verschiedenen Firmen wurden aufgefordert, Ihre Vorschläge einzureichen. 1997 wurden von verschiedenen Firmen Vorschläge eingereicht, die sich jeweils aus einem Metamodell und einer Notation zusammensetzten. Rational reichte die bis dahin entwickelte Version 1.0 von UML ein. Aus den verschiedenen Vorschlägen wurde dann von der OMG-Gruppe der UML-Standard in der Version 1.1 erarbeitet. Seit diesem Zeitpunkt trifft sich die OMG regelmäßig um den Standard anzupassen und aufrechtzuerhalten. Zur Zeit ist UML in der Version 1.3 aktuell, an der Version 2.0 wird gearbeitet.

2.1.2. Der Aufbau von UML

UML wurde konstruiert aus einer 4 Ebenen metamodellierenden Architektur [Seite 2-4 in OMG1999], die aus den vier folgenden Ebenen besteht: Benutzerobjekte, Modell, Metamodell und dem Meta-Metamodell. Die Aufgabe des Meta-Metamodells besteht darin, eine Sprache zur Spezifizierung von Metamodellen zur Verfügung zu stellen.

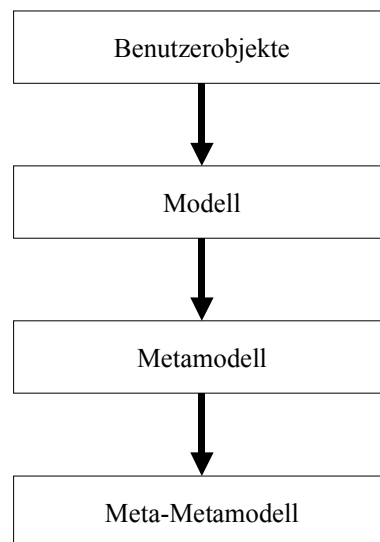


Abbildung 1: Metamodellierende Architektur von UML

Das Metamodell ist eine Instanz des Meta-Metamodells. So ist beispielsweise die Klasse im Metamodell eine Instanz der Metaklasse im Meta-Metamodell. Das Metamodell definiert wiederum eine Sprache zur Spezifizierung von Modellen. Modelle stellen wiederum die Spra-

che für Informationsdomänen bereit. Benutzerobjekte sind eine Instanz von einem Modell und spezifizieren eine Informationsdomäne. Das Metamodell bildet die Semantik von UML, während die Notation als die Syntax bezeichnet werden kann. Um UML einfach anwenden zu können, muss man nicht unbedingt das Metamodel kennen bzw. genau verstanden haben. Es reicht, wenn man die Notation und deren Anwendung begriffen hat. Dementsprechend ist das Metamodell vielen Usern nicht bekannt. Erst wenn man sich tiefer mit der Modellierung mit UML beschäftigt, wird die Kenntnis des Metamodells unerlässlich.

2.1.2.1. Das Metamodell

Das Metamodell von UML ist ein rein logisches und kein Implementierungsmodell. Dieses hat den Vorteil, dass Implementationsdetails unterdrückt werden und die Ausdruckssemantik hervorgehoben wird. Dadurch lassen sich leicht Änderungen am Metamodell vornehmen, ohne dass die entsprechende Notation geändert werden muss. Durch das Metamodell, das meistens in der Form eines Klassendiagramms abgebildet wird, werden die in der Notation verwendeten Methoden exakt beschrieben. Ein Metamodell behandelt eine Menge Modellierungsthemen, lässt aber auch viele unterschiedliche Interpretationen zu [OMG 1999].

Um die Komplexität des Metamodells überhaupt managen zu können, ist das Metamodell in logische Pakete aufgeteilt. Auf der obersten Ebene besteht das Metamodell aus drei Paketen (Abbildung 2).

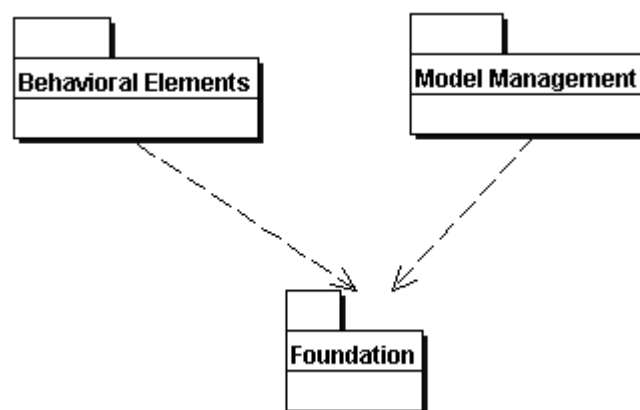


Abbildung 2: Top-Level Packages [OMG 1999]

Das Diagramm zeigt die Abhängigkeiten der drei Pakete untereinander. Die Pakete Behavior Elements und Foundation werden noch in weitere Pakete zerlegt [Seite 2-7 in OMG 1999]. Das Foundation Paket bildet das sprachliche Grundgerüst für die Modelle und unterteilt sich

in folgende drei Unterpakete: Core, Extension Mechanismus und Data Types. Das wichtigste Paket ist dabei das Core Paket, es bildet das Grundgerüst des Metamodells. In ihm sind alle Elemente definiert, die zur einfachen Modellierung benötigt werden wie Klasse, Assoziation, Classifier Interfaces und Assoziationsende. Im Extension Mechanismus wird definiert, wie die einzelnen Elemente mit einer neuen Semantik umgehen. Die Datentypen, die zum Definieren von UML benutzt werden, sind im Paket Data Types angegeben. Im Paket Behavioral Elements befinden sich die Unterpakete Anwendungsfälle (Use Cases), Kollaborationen, State Machines und Activity Grafen.

Die Abbildung 3 zeigt einen Ausschnitt aus dem aktuellen Metamodell. Es ist ein Ausschnitt aus dem Core Paket und zeigt den Zusammenhang zwischen den Assoziationen, den Klassen und Interfaces mit den entsprechenden Rollen. Dieser Ausschnitt stellt den Bereich des Metamodells dar, in dem in Kapitel 4.4 Änderung vorgenommen werden.

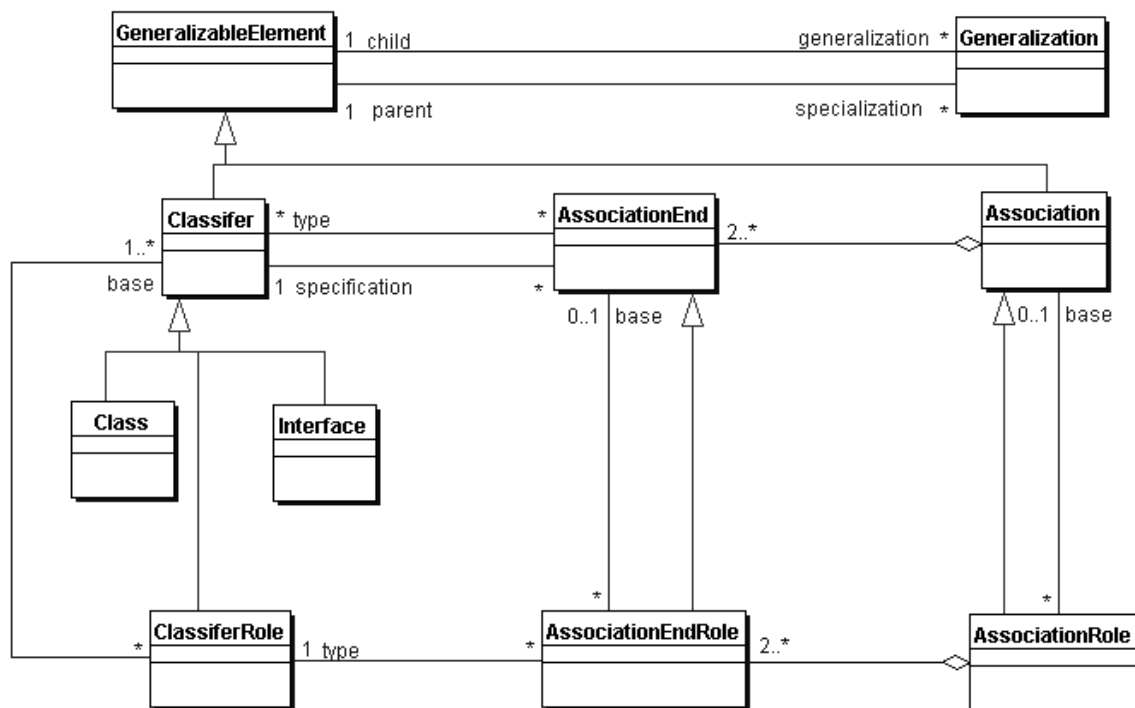


Abbildung 3: Ausschnitt aus dem aktuellen Metamodell [Steimann, Habilitationsschrift]

2.1.2.2. Die Notation

Die Notation umfasst die grafisch dargestellten Elemente in den Modellen. Sie bildet die Syntax der Modellierungssprache und besteht aus 2-dimensionalen Symbolen. Es gibt grundsätzlich vier verschiedene Konstrukte, die in der UML Notation verwendet werden:

1. Icons
2. 2-dimensionale Symbole
3. Pfade
4. Strings

Icons sind graphische Symbole von fester Größe. Sie können keine zusätzlichen Inhalte wie Strings o.ä. aufnehmen und werden zum Beispiel zur Terminierung der Pfade eingesetzt. Unter den 2-dimensionalen Symbolen versteht man graphische Zeichen ohne feste Größe, die noch mit weiteren Symbolen oder Strings gefüllt werden können. Ein Beispiel für solch ein Symbol ist das Rechteck, das eine Klasse symbolisiert. In dem Rechteck stehen somit eine Reihe von Strings, wie z.B. der Klassenname oder die verschiedenen Attribute und Operationen. Die 2-d Symbole bilden mit den Pfaden zusammen die wesentlichen grafischen Elemente der einzelnen Diagramme. Ein Pfad stellt die Verbindung zwischen den einzelnen Symbolen dar, wie die Assoziationen im Klassendiagramm. Strings beinhalten Zeichenketten. Durch sie werden weitere Informationen vermittelt. Sie können als zusätzliche Informationen an Pfaden auftreten oder die einzelnen Komponenten in den Symbolen bezeichnen. Sie können auch lediglich als Kommentare im Diagramm auftauchen.

Um einen kleinen Einblick in die Notation zu bekommen, sind nachfolgend einige Darstellungsformen aufgeführt: Assoziationen werden als eine einfache Linie zwischen den Klassen bzw. Interfaces dargestellt. Aggregationen haben gegenüber den Assoziationen eine Raute als Abschluss der Linie. Klassen und Objekte werden als Vierecke dargestellt, wobei die Namen, Attribute und Operationen ins Viereck eingetragen werden. Bei Objekten ist der Name unterstrichen. Rollen und Kardinalitäten stehen als Label an den einzelnen Assoziationen bzw. Aggregationen.

2.2. Modellierungswerkzeuge von UML

2.2.1. Modellierungswerkzeuge allgemein

Es gibt eine Reihe von Firmen, die Modellierungswerkzeuge zum Entwurf von Softwareprojekten mit UML bereitstellen. Die bekannteste und in diesem Bereich führende Firma ist Rational mit ihrem Produkt Rational Rose. Daneben gibt es auch viele weitere Firmen, die Modellierungswerkzeuge zur Verfügung stellen, wie Cayenne Software, PLATINUM Tech-

nology, Inc. [Neumann] oder TogetherSoft LLC, deren Produkte qualitativ als gleichwertig anzusehen sind. Im Rahmen dieser Arbeit wurde das Tool Together J von TogetherSoft LLC aus den nachfolgend aufgeführten Gründen (2.2.2) verwendet. Dabei wurden die Versionen 3.1, 3.2 und 4.1 benutzt. Der offensichtlichste Unterschied zwischen den Versionen 3.x und 4.1 besteht darin, dass bei der Version 4.1 ein Debugger implementiert wurde, was die Fehlersuche bei der Implementation des eigenen Codes erheblich vereinfacht. Außerdem wurde die farbliche Hervorhebung des Codes verbessert, so dass der Code besser zu lesen ist.

2.2.2. Arbeiten mit Together J

Der große Vorteil des Modellierungswerkzeuges von Together liegt darin, dass sowohl das Modellierungsmodell als auch der dazugehörige Quellcode gleichzeitig erzeugt werden. Dadurch kann man sofort erkennen, wie das Modellerte später im Code aussehen wird und gegebenenfalls im Implementierungscode Änderungen vornehmen. Diese werden dann umgehend im Diagramm berücksichtigt. Together ist nach eigenen Angaben das erste Tool, das dieses Feature der synchronen Erzeugung von Modell und Code anbietet [Together]. Demgegenüber ist der Editor, der für Änderungen des Codes zur Verfügung steht, sehr einfach gehalten und aufgrund des mangelnden Komforts nicht besonders gut für die Implementierung von längeren Codes geeignet. Für kleine Änderungen reicht er aber vollkommen aus. Der Code wird bei dem Werkzeug Together J in JAVA erzeugt. Da JAVA eine plattformunabhängige Programmiersprache ist, bietet es den Vorteil, dass die modellierten Programme auf allen Betriebssystemen laufen.

Die Benutzeroberfläche von Together ist übersichtlich aufgebaut. Neben dem Modellierungsfenster gibt es ein Propertyfenster, in dem man die Eigenschaften der einzelnen Komponenten einfach und komfortabel eintragen kann. Weiterhin steht ein Explorerfenster zur Navigation zur Verfügung. Das Arbeiten mit dem Tool ist nach einer kurzen Eingewöhnungszeit einfach und intuitiv, wobei die Menüführung und die Einstellung der Optionen sich etwas unübersichtlich und damit gewöhnungsbedürftig gestaltet. Leicht verbesserungswürdig sind die Hilfefunktion und maßgeblich das Benutzerhandbuch. Dort enthält insbesondere das Schlagwortregister zu wenig Begriffe, so dass die gesuchte Textstelle relativ schwierig zu finden ist. Die Erklärungen in einigen Bereichen sind teilweise etwas dürftig und helfen somit nicht immer weiter.

Ein hilfreiches Feature zur Präsentation und Dokumentation der modellierten Diagramme besteht in der Möglichkeit, das gesamte Modell im Html-Format oder jedes Diagramm ein-

zeln im Gif- oder Wmf-Format abzuspeichern. Bei der Abspeicherung im Html-Format wird auch gleichzeitig schon eine Baumstruktur für Präsentationen erzeugt, so dass lediglich die persönlichen Kommentare eingefügt werden müssen.

Insgesamt überwiegen bei dem System die Vorteile des direkten Anzeigens des Implementierungscode und der in ihm vorgenommenen Änderungen. Dadurch bleibt einem die Benutzung eines weiteren Programmiertools mit dessen Benutzungseigenschaften erspart. Gerade ab der Version 4.1 mit dem integrierten Debugger steht eine fast komplette Programmierumgebung zur Verfügung. Die teilweise etwas gewöhnungsbedürftige Menüführung fällt bei längerer Benutzung kaum noch ins Gewicht.

3. Entwurf des Hyperbooks anhand des aktuellen Metamodells

3.1. Aufbau des Hyperbooks und weitere Vorgehensweise

Das benutzeradaptierbare Hyperbook dient dem Bereitstellen und Halten einer Vorlesung im Internet. Der Dozent kann hierzu die Vorlesung aus einer bestehenden Sammlung von Texten zusammenstellen und eigene Texte hinzufügen. Diese Texte werden in einzelne Lernabschnitte zusammengefasst. Die Lernabschnitte bilden das dem Benutzer angezeigte Grundgerüst der Vorlesung, in denen er sich dann weiter durchklicken kann. Dieses Zugreifen auf das Hyperbook geschieht über eine sogenannte Benutzerschnittstelle.

Im Rahmen des Hyperbooks stehen auch zwei Suchfunktionen zur Verfügung, mit deren Hilfe der Hörer sich weitere Texte zum Thema heraussuchen kann. So gibt es zum einen die Indexsuche und zum anderen das kategorische Suchen. Ein Querverweis auf den gefundenen Text kann bei Bedarf in den persönlichen Notizen abgespeichert werden. Da beim Suchen keine Volltextsuche durchgeführt wird, müssen beim Hinzufügen eines Textes in die Datenbank die entsprechenden Suchbegriffe vom Dozenten mit angegeben werden.

Das Hinzufügen von Notizen bleibt den registrierten Teilnehmern der Vorlesung vorbehalten. Dazu werden alle Teilnehmer am Anfang der Vorlesung registriert. Sie erhalten einen Benutzernamen und ein Passwort, womit sie sich in das Hyperbook einloggen und entsprechend jedem Text ihre persönlichen Notizen hinzufügen können. Dabei werden die Notizen gesondert vom Text gespeichert, so dass der Text selber nicht verändert wird. Ohne Registrierung kann an der Vorlesung ebenfalls teilgenommen werden, allerdings ohne die Möglichkeit des Annotierens.

Im Rahmen des folgenden Entwurfes des Hyperbooks werden zunächst die möglichen Anwendungsfälle anhand der Use Cases aufgezeigt. Anschließend wird über die Klassendiagramme der Entwurf des Hyperbooks nach derzeitigem Stand vorgenommen. Schließlich wird über die Sequenzdiagramme der Ablauf der einzelnen Use Cases verdeutlicht.

Im Rahmen dieser Arbeit wurde eine Beschränkung auf den Kern des Hyperbooks vorgenommen. Die Peripherie wurde nicht berücksichtigt, da sie im Vergleich zur Modellierung des rollenbasierten Entwurfes keine Unterschiede hervorruft und somit keine Relevanz für den Vergleich besitzt. Bei tatsächlicher Anwendung müsste jedoch in der Klasse Filerepresentation die Umwandlung des Textes ins Mimeformat implementiert werden. Weiterhin müsste die Umwandlungsfunktion implementiert werden, mit dessen Hilfe die Benutzerschnittstelle

aus den Textnamen, die von den Suchfunktionen als ein String übergeben werden, die entsprechenden Links zu den Texten rekonstruieren kann. Damit kann dann direkt auf die Texte zugegriffen werden.

3.2. Use cases

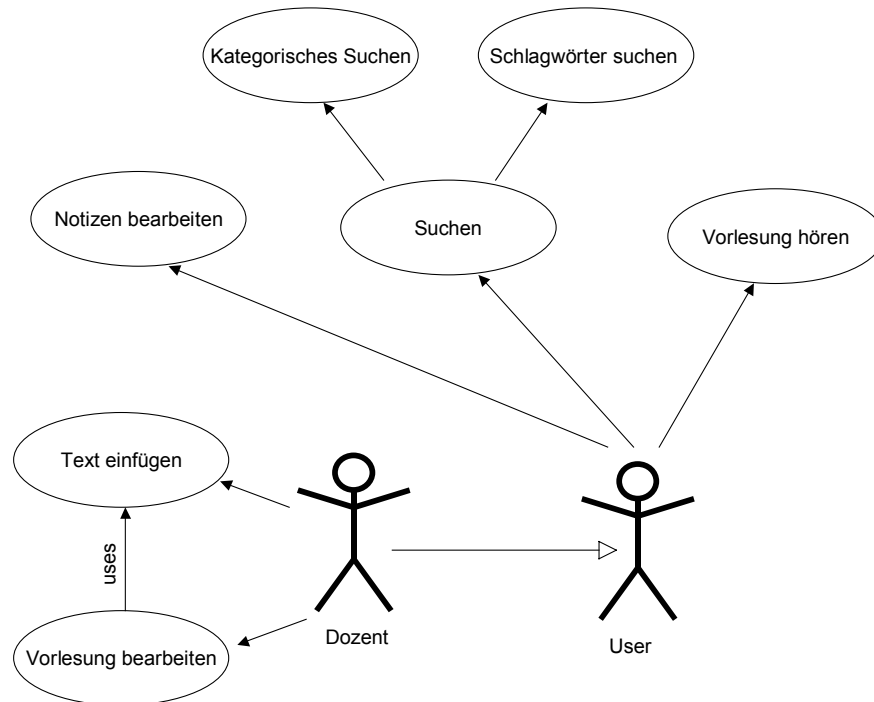


Abbildung 4: Use cases

Das Use Cases Diagramm zeigt die einzelnen Anwendungsfälle des Hyperbooks. Entsprechend des unter 3.1. beschriebenen Aufbaus ergeben sich die nachfolgend aufgeführten fünf Use Cases Notizen bearbeiten, Suchen, Vorlesung hören, Vorlesung bearbeiten und Text einfügen. Die beiden Fälle *Text einfügen* und *Vorlesung erstellen* können dabei nur vom Dozenten ausgeführt werden, die anderen drei (*Notizen bearbeiten*, *Vorlesung hören* und *Suchen*) sind allen zugänglich, also auch dem Dozenten (dargestellt durch den Pfeil zwischen Dozent und User [Schneider]). Das Use Case Suchen unterteilt sich in die Bereiche Schlagwortsuche und kategorische Suche. Das in Abbildung 4 abgebildete Use Case Diagramm zeigt das Haupt-Use Case und bildet die Gesamtübersicht der Anwendungsfälle ab. Auf eine detailliertere Darstellung der einzelnen Use Cases wird hier verzichtet, da die Nutzung des Hyperbooks in den Sequenzdiagrammen später ausreichend beschrieben wird.

3.3. Klassendiagramme

Das Klassendiagramm besteht aus den fünf Packages Textbase, Vorlesung, Indexsearch, Classification und Notizen, die nachfolgend näher dargestellt werden sollen. Dabei wird auf die in den Klassen enthalten Methoden nicht näher eingegangen, da dieses später in der Beschreibung der Sequenzdiagramme erfolgt.

3.3.1. Package Textbase

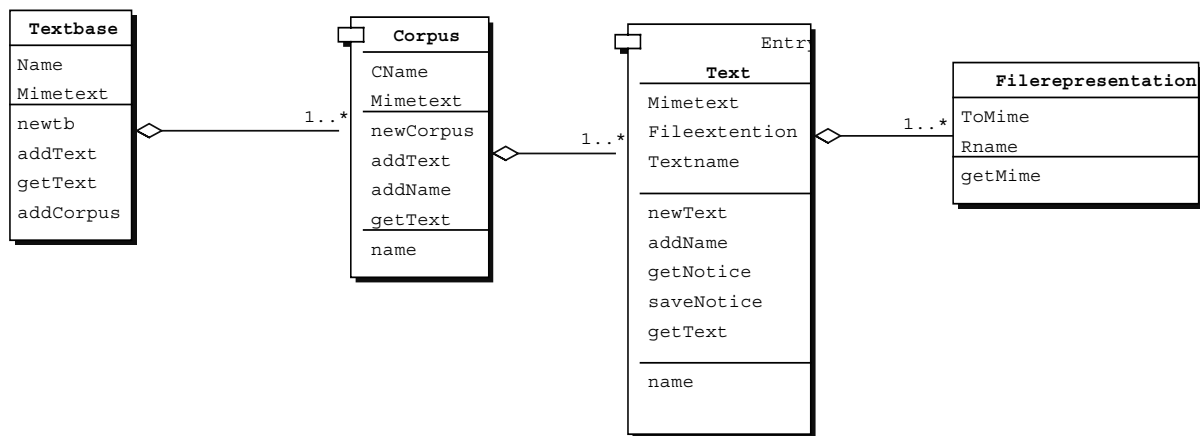


Abbildung 5: Package Textbase

In dem Package Textbase wird die Ablage der Texte im Hyperbook beschrieben. Es beinhaltet die Ansammlung der Texte in der Datenbank und besteht aus den Klassen Textbase, Corpus, Text und Filerepresentation. Eine Textbase besteht aus mehreren Corpora. Da die Corpora ein Teil der Textbase darstellen, ist die Beziehung zwischen den Klassen eine Aggregation, dargestellt durch 1..* in der Abbildung. Ein Corpus besteht wiederum aus mehreren Texten; auch hier sind die Texte Teile des Corpus und deshalb mittels einer Aggregation miteinander verbunden. Die Reihenfolge der Corpora in der Textbase bzw. der Texte im Corpus ist für die Funktion nicht entscheidend. Grundsätzlich lassen sich Aggregationen als Hashtabelle, über Vektoren, Listen oder Arrays implementieren. Da die Zugriffszeit in einer Hashtabelle im Mittel geringer ausfällt als bei einer Realisierung durch einen Vektor, Liste oder Array, wurden die Aggregationen hier als Hashtabellen implementiert. Als Key diente der jeweilige Corpusname bzw. Textname.

Um die in den verschiedenen Dateiformaten abgespeicherten Texte einheitlich für alle User darstellen zu können, gibt es die Klasse Filerepresentation. Sie ist mit der Klasse Text durch eine Aggregation verbunden. Die Texte werden dort in ein für den Transport über das Netz

brauchbares Format umgewandelt. Jedes Objekt der Klasse Text ist mit einem oder mehreren Objekten der Klasse Filerepresentation verbunden. Dargestellt wird dieses durch die“ 1..* „, an der Aggregation auf Seiten der Klasse Filerepresentation.

3.3.2. Package Vorlesung

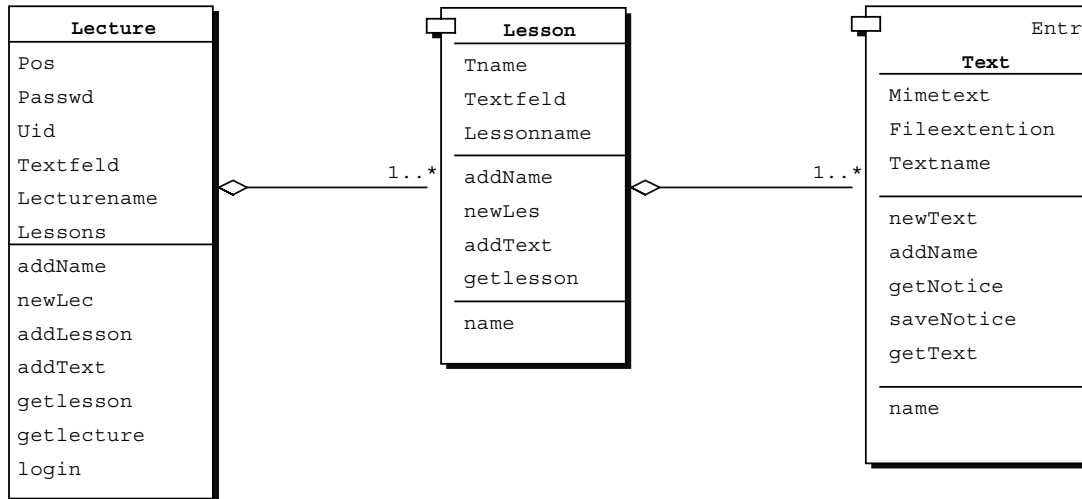


Abbildung 6: Package Vorlesung

Das Package Vorlesung zeigt die Struktur einer Vorlesung im Hyperbook auf. Es besteht aus den Klassen Lecture, Lesson und Text. Eine Lecture (Vorlesung) besteht aus mehreren Unterrichtseinheiten, die durch die Klasse Lesson repräsentiert werden. Im Gegensatz zum Package Textbase ist hier die Reihenfolge der Unterrichtseinheiten von Bedeutung. Aus diesem Grunde wurde hier die Aggregation nicht als Hashtabelle, sondern als Vektorliste realisiert. Eine Lesson setzt sich wiederum aus mehreren Vorlesungstexten zusammen, die in der Klasse Text gespeichert werden. Auch hier wird die Reihenfolge durch den Dozenten vorgegeben. Daher wurde sie wie die Aggregation in *Lecture* als Vektorliste implementiert. Das Einfügen eines Eintrages an der richtigen Stelle wird durch die Angabe der entsprechenden Position in der Liste verwirklicht. Dieses gilt sowohl für eine Vorlesungseinheit (Lesson) als auch für die in der Vorlesungseinheit verwendeten Texte.

3.3.3. Package Notizen

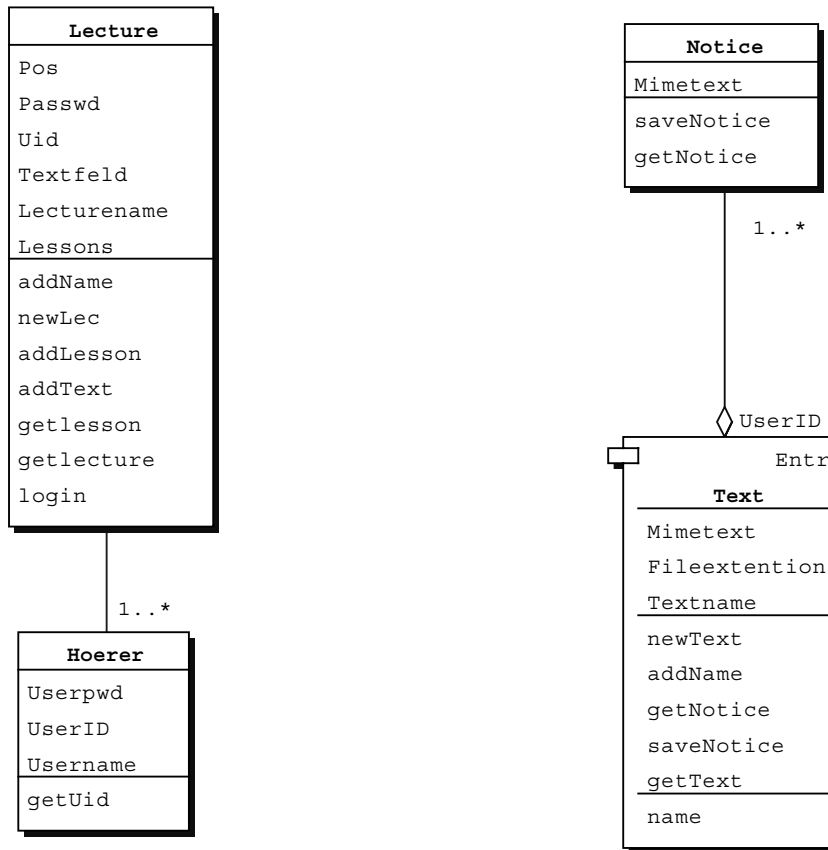


Abbildung 7: Package Notizen

Das Package Notizen besteht aus zwei Teilen: Zu Beginn der Benutzung des Hyperbooks erfolgt ein Login, der von der Klasse *Lecture* durchgeführt wird. Der eingegebene Username und das Passwort werden in der Klasse *Hoerer* mit dem dort gespeicherten Passwort verglichen. Bei Übereinstimmung wird die entsprechende UserID zurückgegeben, bei Divergenzen erhält der User die UserID 0. Die Beziehung zwischen den Klassen *Lecture* und *Hoerer* wurde ebenfalls über eine als Hashtabelle realisierte Aggregation abgebildet, da es hier wiederum nicht auf die Reihenfolge ankommt.

Der zweite Teil besteht aus den Klassen *Text* und *Notice*. Fügt ein User bei einem Text eine persönliche Notiz ein, so wird diese in der Klasse *Notice* unter seiner UserID abgespeichert. Wird später dieser Text aufgerufen, so lädt der entsprechende Text die Notiz, indem er in der Hashtabelle der Aggregation nach der UserID sucht und anschließend den entsprechende Notiztext in der Klasse *Notice* lädt. Die UserID dient als eindeutiger Key für die Hashtabelle.

3.3.4. Package Indexsearch

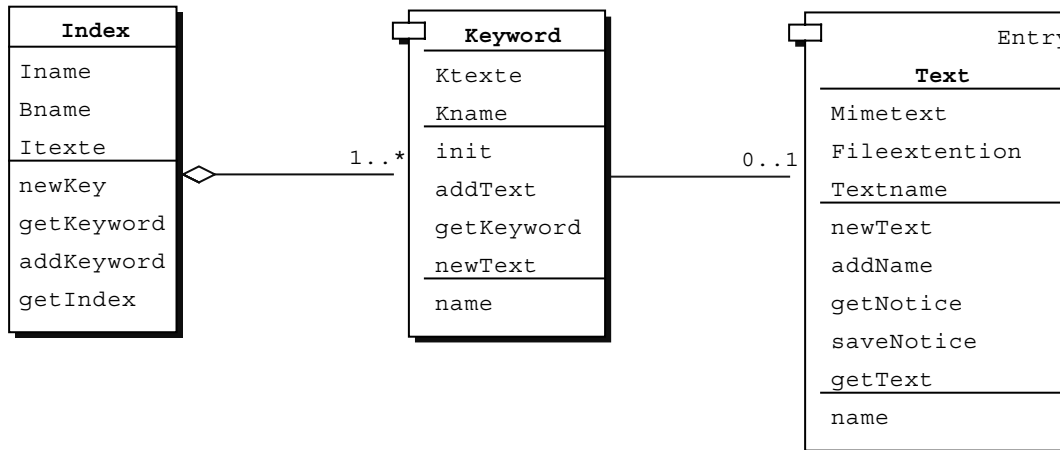


Abbildung 8: Package Indexsearch

Das Package Indexsearch beinhaltet die Suchfunktion, mit deren Hilfe man bestimmte Schlagworte innerhalb einer Vorlesung herausfinden kann. Der Aufbau gleicht dem des Packages Textbase. Hier besteht ein Index aus mehreren Schlagwörtern, die Aggregation ist ebenfalls als Hashtabelle implementiert. Ein Keyword kann in mehreren Texten enthalten sein. Da jedoch ein Text nicht Bestandteil eines Keywords ist, wird hier im Gegensatz zum Package Textbase die Klasse *Text* mittels einer Assoziation mit der Klasse *Keyword* verbunden. Die Assoziation wurde wegen der besseren Zugriffszeiten als Hashtabelle realisiert. Die Ablage der Keywords und Texte in der Hashtabelle erfolgt unsortiert. Auf Wunsch kann eine sortierte Ausgabe über die Benutzerschnittstelle erfolgen, die die Sortierung vor dem Weiterleiten an den User vornimmt.

3.3.5. Package Classifikation

3.3.5.1. Aufbau des Package Classifikation

Das Package Classifikation umfasst die kategorische Suche. Dabei bildet die Klasse *Classifikation* den Einstieg dazu. Durch sie werden die einzelnen Klassifikationen für die Suche unterschieden. Sie ist mittels einer Assoziation mit der abstrakten Klasse *Entry* verbunden. Die Klasse *Entry* besitzt die beiden Unterklassen *Text* und *Term*. In der Klasse *Term* sind die einzelnen Kategorien gespeichert. Jede Kategorie hat eine weitere Anzahl von Einträgen. Diese können entweder weitere Kategorien, also Instanzen der Klasse *Term*, oder ein Text, also eine Instanz von der Klasse *Text*, sein. Stellt ein Eintrag nun eine weitere Kategorie dar, so wird

über die abstrakte Klasse *Entry* mittels Vererbung die neue Instanz von *Term* aufgerufen. Dieser Aufruf wird mittels einer Aggregation über die abstrakte Klasse *Entry* realisiert. So werden die einzelnen Unterkategorien rekursiv aufgerufen, und dadurch sind der Verschachtelungstiefe keine Grenzen gesetzt. Eine unbegrenzte Verschachtelungstiefe birgt jedoch auch die Gefahr, dass u.U. übermäßig viele Unterkategorien eingefügt werden. Dann müsste ein User sich durch zu viele Kategorien durchhangeln, um an seine gesuchten Informationen zu kommen. Deshalb sollte eine sinnvolle Begrenzung von ca. 3 Schleifen implementiert werden. Das Design für den rekursiven Aufbau ist das des Composite patterns und entstammt dem Buch Design Patterns [Gamma].

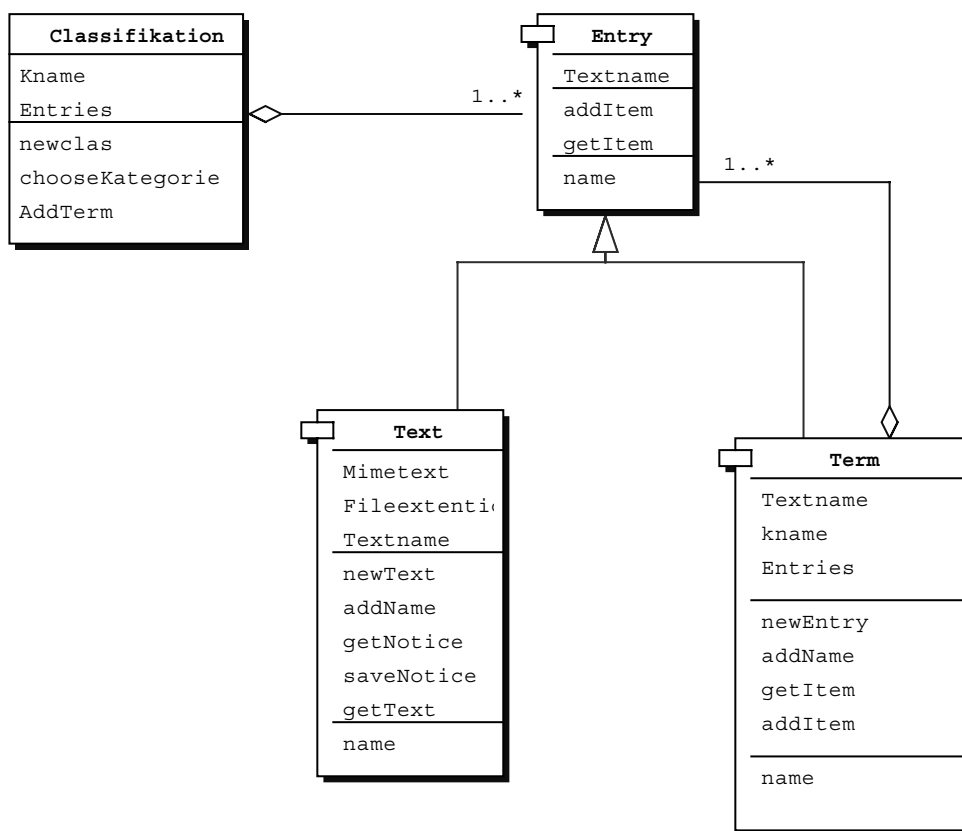


Abbildung 9: Package Klassifikation

Falls es sich bei dem Eintrag um einen Text handelt, wird dieser ebenso über *Entry* aufgerufen. Da der Text aber einen Endknoten im Klassifikationsbaum darstellt, geht keine weitere Verschachtelung von diesem aus.

3.3.5.2. Erklärung des Package Classification anhand eines Beispiels

Der Aufbau und die Funktion des Packages Classification werden im folgenden anhand eines Beispiels veranschaulicht. Hierbei können über die Suchfunktion Informationen zu Gebäuden und deren einzelnen Einrichtungen abgerufen werden. Gesucht werden in diesem Fall Informationen zu einem konkreten Bauteil (Lichtschalter). Eine mögliche Struktur der Kategorien und Texte kann der Abbildung 10 entnommen werden.

Für die Kategorische Suche eines Textes über Lichtschalter ruft der User die Hauptkategorie auf (Gebäude). Dieses geschieht durch den Aufruf der Methode `chooseKategorie` in der Klasse `Classification`. Dort werden die ersten Unterkategorien ermittelt, indem die Hashtabelle nach Einträgen durchsucht wird. Die Namen der Einträge (Fabrik, Wohnhaus) werden zurückgeliefert.

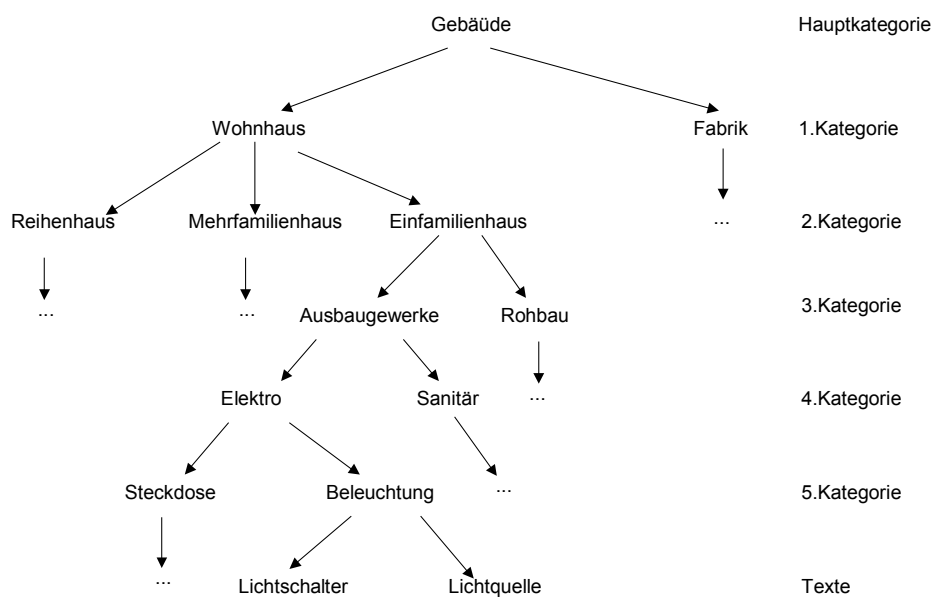


Abbildung 10: Beispielstruktur fürs kategorische Suchen

Bei der Auswahl der nächsten Kategorie (Wohnhaus) ruft die Methode `chooseKategorie`, nachdem die ausgewählten Kategorien aus der Hashtabelle herausgesucht wurden, die Methode `getItem` in der entsprechenden Instanz von `Term` auf (über die abstrakte Klasse `Entry`). Hier werden alle Einträge der Hashtabelle mit den neuen Unterkategorien (Reihenhaus, Mehrfamilienhaus, Einfamilienhaus) ausgelesen und die Namen zurückgeliefert. Bei der Auswahl der Kategorie Einfamilienhaus erfolgt wiederum der Aufruf der Methoden `chooseKategorie`

(Classifikation) und *getItem* (Term), um zu der Kategorie Wohnhaus zu gelangen. Dort werden dann durch den rekursiven Aufruf (über *Entry*) von *getItem* die Unterkategorien von Einfamilienhaus bestimmt und zurückgegeben. Dieser Ablauf erfolgt analog bei den Unterkategorien Ausbaugewerke, Elektro, Beleuchtung bis im letzten Schritt die Namen der Unterkategorien von Beleuchtung zurückgegeben werden. Damit erhält der User den gesuchten Text über Lichtschalter und einen Text über Lichtquellen. Wie oben beschrieben, kann dieser Vorgang beliebig oft weitergeführt werden. Allerdings zeigt dieses simple Beispiel, dass eine zu große Anzahl von Unterkategorien nicht unbedingt zur Übersichtlichkeit und einem schnellen Zugriff beiträgt, sondern eher das Gegenteil bewirkt. Deshalb erweist es sich als sinnvoll, sich auf maximal drei Unterkategorien zu beschränken. In diesem Beispiel sollte eine Umsetzung eher in der Struktur Gebäude, Haus, Elektro, Beleuchtung, Lichtschalter erfolgen.

Die kategorische Suche des Hyperbooks wurde aus oben angeführten Gründen auf drei Unterkategorien beschränkt. In der Technischen Informations- Bibliothek Hannover besteht die kategorische Suche sogar nur aus 2 Unterkategorien.

3.4. Sequenzdiagramme

Aus dem Hyperbook lassen sich analog den Use Cases fünf Sequenzdiagramme ableiten, wobei der Use Case Text einfügen in der Sequenz *Vorlesung bearbeiten* enthalten ist.

3.4.1. Vorlesung bearbeiten

Die Sequenz *Vorlesung bearbeiten* erfolgt in vier Schritten: Zunächst wird der Text in den entsprechenden Corpus eingefügt. Dazu wird in der Klasse *Textbase* die Methode *addText* aufgerufen. Die Methode *addText* sucht in der Hashtabelle nach dem Link des Corpus, in dem der Text eingefügt werden soll. Mit Hilfe des Links wird die Methode *addText* in der entsprechenden Instanz von *Corpus* ausgeführt. Sie fügt den Text in die Hashtabelle des Corpus ein. In der Hashtabelle sind somit alle Texte abgelegt, die im Corpus enthalten sind.

Im zweiten Schritt wird der Text an die gewünschte Stelle im Index der entsprechenden Vorlesung eingefügt. Hierzu wird die Methode *addText* der Klasse *Lecture* aufgerufen. Diese ruft in der entsprechenden Instanz der Klasse *Lesson*, die durch den in den Parametern angegebenen Lessonnamen bestimmt wird, ebenfalls die Methode *addText* auf. In *addText* der Klasse *Lesson* wird der in der Parameterliste angegebene Text an der entsprechenden Position in die Vektorliste der in der Stunde vorkommenden Texte eingefügt.

Anschließend werden die Schlagwörter des Textes in die Schlagwortsuche eingebunden. Dazu werden die Schlagwörter des Textes für die Schlagwortsuche einzeln hintereinander durch die Methode *addKeyword* im Index hinterlegt. Hierbei wird im Index überprüft, ob schon ein entsprechender Eintrag besteht oder ein neuer erzeugt werden muss. In der entweder neu erzeugten oder schon vorhandenen Instanz der Klasse *Keyword* wird dann anschließend mit der Methode *addText* der entsprechende Link zum Text in der Hashtabelle gespeichert.

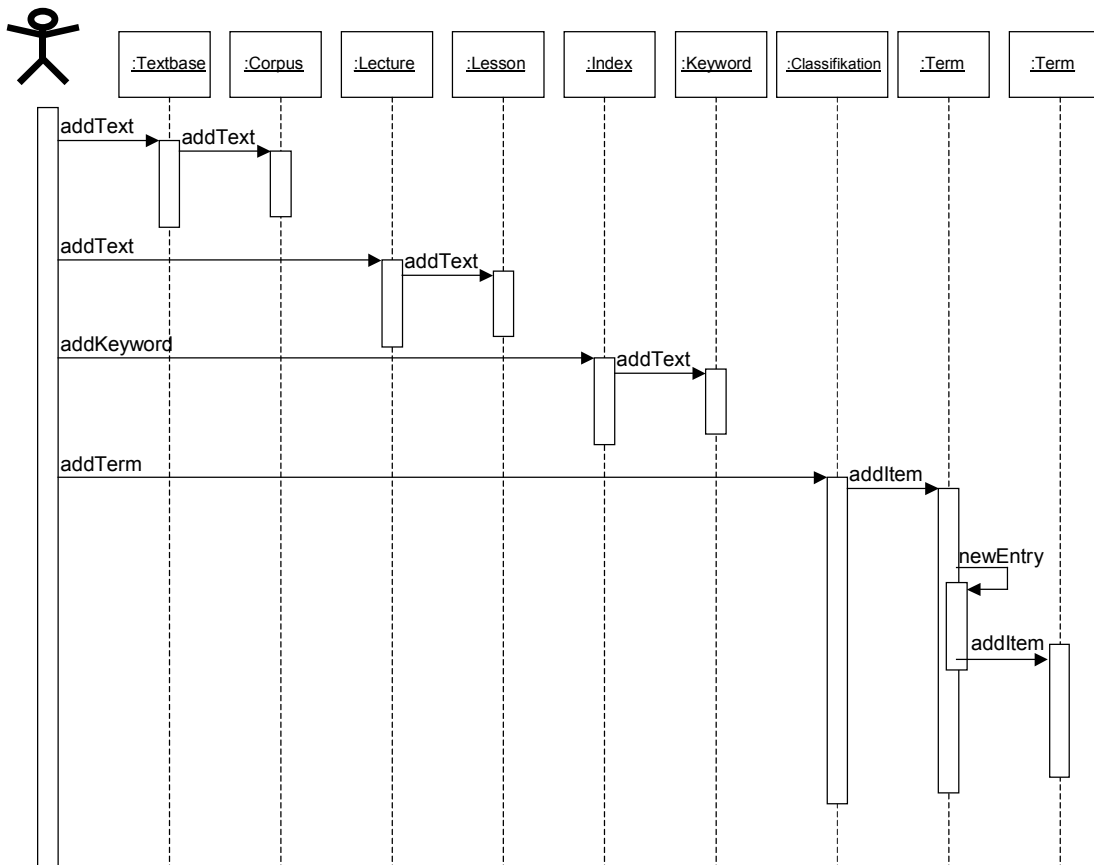


Abbildung 11: Vorlesung bearbeiten

Im vierten Schritt werden die Kategorien des Textes in die kategorische Suche eingebunden. Dafür wird bei der Klassifikationssuche mit *addTerm* ein neuer Eintrag hinzugefügt. Hierbei werden alle drei Kategorien und der einzubindende Text in die Parameterliste übergeben. In der abstrakten Klasse *Entry* wird die Methode *addItem* aufgerufen, wodurch mittels Vererbung die Methode *addItem* in der Klasse *Term* ausgeführt. Dort wird, wenn der Eintrag nicht schon existiert, ein neues Objekt mit dem entsprechenden Kategorienamen erzeugt und anschließend die nächste Kategoriestufe aufgerufen. Wenn der Eintrag bereits besteht, wird gleich zur nächsten Stufe weitergegangen. Dieses Aufrufen der nächsten Stufe geschieht re-

kursiv über die Methode *addItem* in der abstrakten Klasse *Entry*. Ein leerer String als Kategorienname wird dabei als nicht vorhandene Kategoriestufe aufgefasst, und statt dessen wird der Link zum neu eingefügten Textobjekt als nachfolgender Verweis eingefügt.

3.4.2. Vorlesung hören

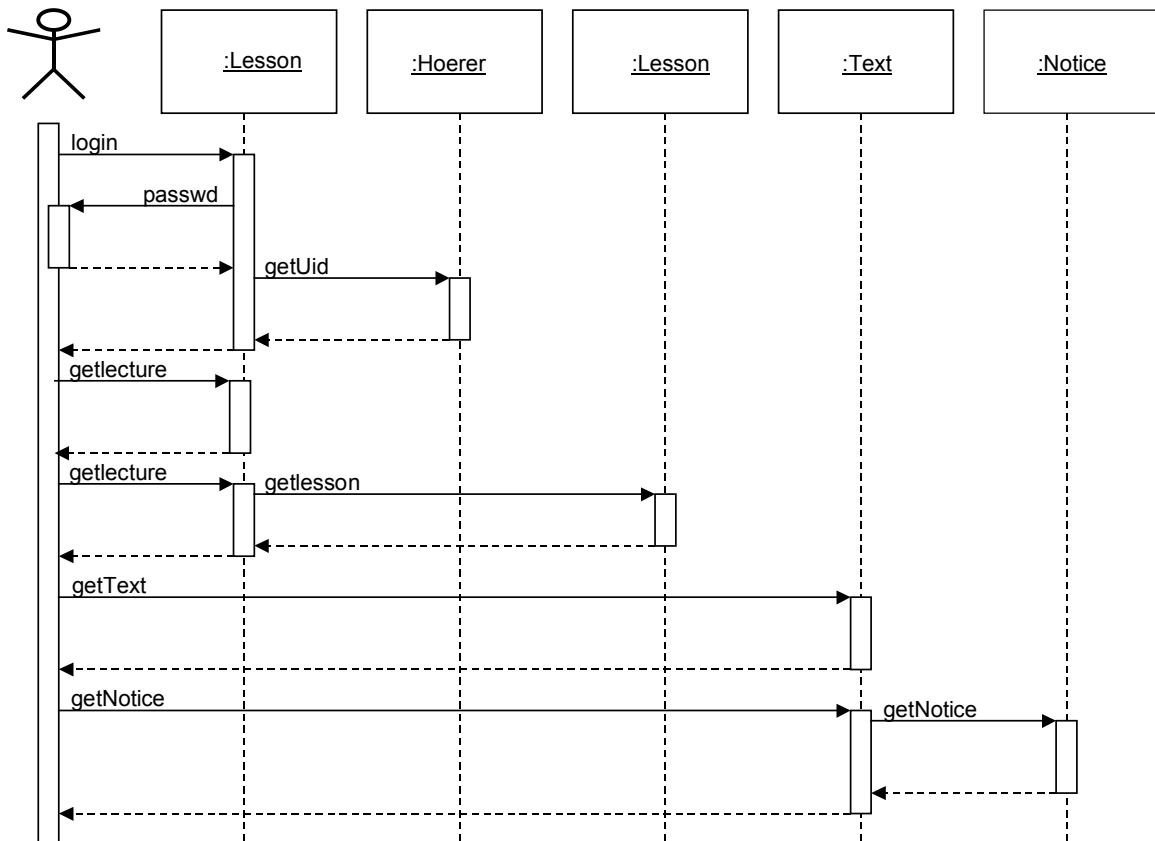


Abbildung 12: Vorlesung hören

Um eine Vorlesung hören zu können, muss sich der User zunächst einloggen. Dazu wird in der Klasse *Lecture* die Methode *login* aufgerufen. Diese fragt nach der UserID und dem Passwort des Users und ruft die Methode *getUid* in der Klasse *Hörer* auf. Dabei dient der Username zur Identifikation des entsprechenden Eintrages in der Hashtabelle von *Lecture*. In *getUid* wird das in der Parameterliste übergebene Passwort mit dem in dem Objekt gespeicherten Passwort verglichen. Bei Übereinstimmung wird die entsprechende UserID (ansonsten die UserID 0) an die Instanz der Klasse *Lecture* zurückgegeben. Diese leitet die UserID an die Benutzerschnittstelle weiter. Die UserID ist später zum Verfassen von persönlichen Notizen des Hörers bzw. deren Abruf notwendig.

Über die Benutzerschnittstelle wird durch den Aufruf der Methode *getlecture* in dem Objekt der Klasse *Lecture* die gewünschte Vorlesung folgendermaßen ausgewählt: Die in der Hash-tabelle eingetragenen Vorlesungsstunden werden durch den Aufruf der Methode *getName* in der Klasse *Lesson* in eine Liste eingetragen. Die Methode *getName* liefert den Namen der Stunde an die Instanz von *Lecture* zurück. Die Liste wird an die Benutzerschnittstelle zurückgegeben. Durch den über die Benutzerschnittstelle ausgelösten Aufruf von *getlesson* in den Klassen *Lecture* und *Lesson* werden dann die Texte der entsprechenden Vorlesungseinheit aufgelistet. Dazu wird in der Methode *getlesson* von *Lecture* die Methode *getlesson* in der ausgewählten Instanz (der Name der Vorlesungsstunde wird in der Parameterliste mitübergeben) von *Lesson* aufgerufen. In *getlesson* werden alle Namen der in der Stunde vorkommenden Texte (die Links sind in der Hashtabelle hinterlegt) durch einen entsprechenden Aufruf der Methode *getName* in der Klasse *Text* in eine Liste eingetragen, die an die Benutzerschnittstelle zurückgeliefert wird. Mittels *getText* in der Instanz der Klasse *Text* wird dann der Inhalt des Textes des vom User ausgewählten Textes (durch klicken) geladen. Anschließend wird über den Aufruf der Methode *getNotice* im entsprechenden Objekt der Klasse *Text* die persönliche Notiz zum Text aus der Klasse *Notice* im Html-Format an die Benutzerschnittstelle gesendet. Dazu wird in der Methode *getNotice* in der Klasse *Text* der Link zur gesuchten Notiz mittels der UserID aus der Hashtabelle herausgesucht. Durch den Aufruf der Methode *getNotice* in der entsprechenden Instanz von *Notice* werden die persönlichen Notizen an die Klasse *Text* übergeben. Bei einer UserID von 0, die dem eines Gasthörers entspricht, wird keine bzw. eine leere Notiz übermittelt.

3.4.3. Notizen bearbeiten

Beim Notizen bearbeiten wird über die Benutzerschnittstelle die Methode *getNotice* in der Instanz der Klasse *Text* mit den Parametern UserID und Textname aufgerufen. Hier wird mittels der UserID der Link auf die entsprechende Instanz der Klasse *Notice* aus der Hashtabelle herausgelesen. Anschließend wird im entsprechenden Objekt der Klasse *Notice* die Methode *getNotice* ausgeführt, diese liefert den Inhalt der Notiz an die Instanz von *Text* zurück. Die Methode *getNotice* gibt diesen an die Benutzerschnittstelle weiter und der Benutzer kann die entsprechende Notiz ändern. Nach den durch den User vorgenommenen Änderungen wird über die Methode *saveNotice* in der Klasse *Text* wiederum der entsprechende Link zum Objekt der Klasse *Notice* herausgesucht. In der dazugehörigen Instanz von *Notice* wird die Methode *saveNotice* ausgeführt, in ihr wird die neue Notiz in der Datenbank gespeichert. Die Sequenz verläuft analog beim Erstellen einer neuen Notiz.

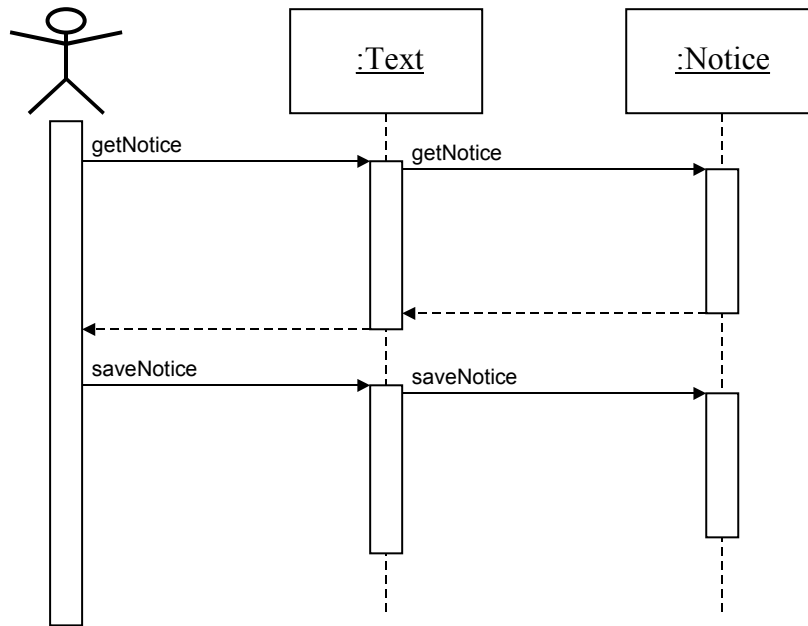


Abbildung 13: Notizen bearbeiten

3.4.4. Schlagwortsuche

Bei der Schlagwortsuche wird über die Benutzerschnittstelle die Methode *getIndex* aufgerufen. Dort werden in dem Objekt der Klasse *Index* alle dort eingetragenen Schlagwörter herausgesucht. Die Links zu den entsprechenden Instanzen der Klasse *Keyword* werden aus der Hashtabelle entnommen. Durch den anschließenden Aufruf der Methode *getName* in der Klasse *Keyword* werden die Namen der einzelnen Schlagwörter an den Index zurückgegeben, wo sie in einer Liste (als String) zusammengefasst und danach an die Benutzerschnittstelle übermittelt werden. Nach der Auswahl eines Schlagwortes durch den User wird die Methode *getKeyword* ausgeführt. Dadurch wird in der Instanz der Klasse *Index* das gewünschte Schlagwort in der Hashtabelle herausgesucht und der entsprechende Link auf das Objekt in der Klasse *Keyword* geladen. Anschließend wird die Methode *getKeyword* in der Klasse *Keyword* aufgerufen. Die Methode *getKeyword* ruft alle Einträge der Hashtabelle mit den Links zu den Objekten der Klasse *Text* ab. Durch den Aufruf der Methode *getName* in den jeweiligen Instanzen der Klasse *Text* werden die Namen der Texte in einen String eingetragen. Die dadurch entstandene Liste der gefundenen Texte wird von der Methode *getKeyword* in den Klassen *Index* und *Keyword* an die Benutzerschnittstelle übergeben. Der User kann jetzt durch Auswählen des gesuchten Textes über die Methode *getText* in der Klasse *Text* sich den gewünschten Text anzeigen lassen.

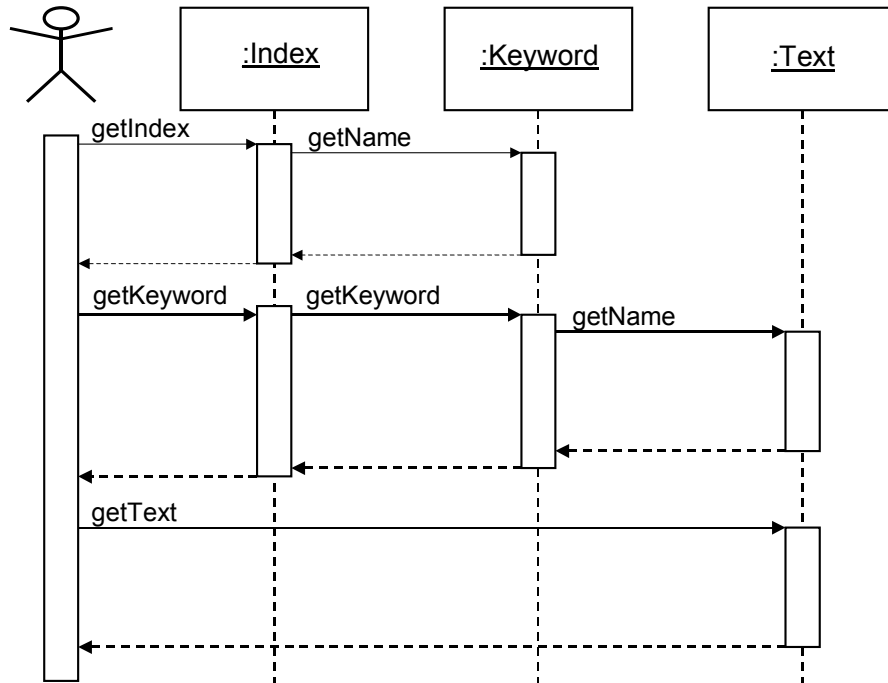


Abbildung 14: Schlagwortsuche

3.4.5. Kategorische Suche

Bei der Kategorischen Suche werden die Unterkategorien einer Kategorie über die abstrakte Klasse *Entry* und der Subklasse *Term* rekursiv aufgerufen. Da sich aber Rekursionen im Sequenzdiagramm schlecht darstellen lassen, wird in Abbildung 15 nur eine Beispielsequenz dargestellt. Die Sequenz zeigt das Auslesen der Texte der dritten Kategorie. Hierbei wird der Aufruf über die abstrakte Klasse *Entry* nicht angezeigt, da im Sequenzdiagramm nur Instanzen der Klassen angezeigt werden. *Entry* besitzt als abstrakte Klasse jedoch keine Instanzen, sondern ruft lediglich per Vererbung die Instanzen in ihren Subklassen (*Text* u. *Term*) auf.

Durch den User wird über die Benutzerschnittstelle der Methodenaufruf *chooseKategorie* in der Klasse *Classification* angestoßen, mit dessen Hilfe die Untereinträge der gewählten Kategorie herausgelesen und in eine Ausgabeliste eingetragen werden. Dies erfolgt, indem die Methode *getItem* in der abstrakten Klasse *Entry* aufgerufen wird. Dadurch wird die entsprechende Methode *getItem* in der Subklasse *Term* ausgeführt. In den aufgerufenen Instanzen der Klasse *Term* werden die in der Parameterliste übergebenen Kategorien mit den Einträgen in der Hashtabelle verglichen. Dazu wird mittels der Methode *getName* der Name der Einträge ausgelesen. Bei Übereinstimmung mit dem entsprechenden Link wird erneut die Methode

getItem in der abstrakten Klasse *Entry* aufgerufen. Dadurch erfolgt abermals ein Aufruf der entsprechenden Instanz von *Term*. Vorab wird der Name der Kategorie in die Ausgabeliste übernommen.

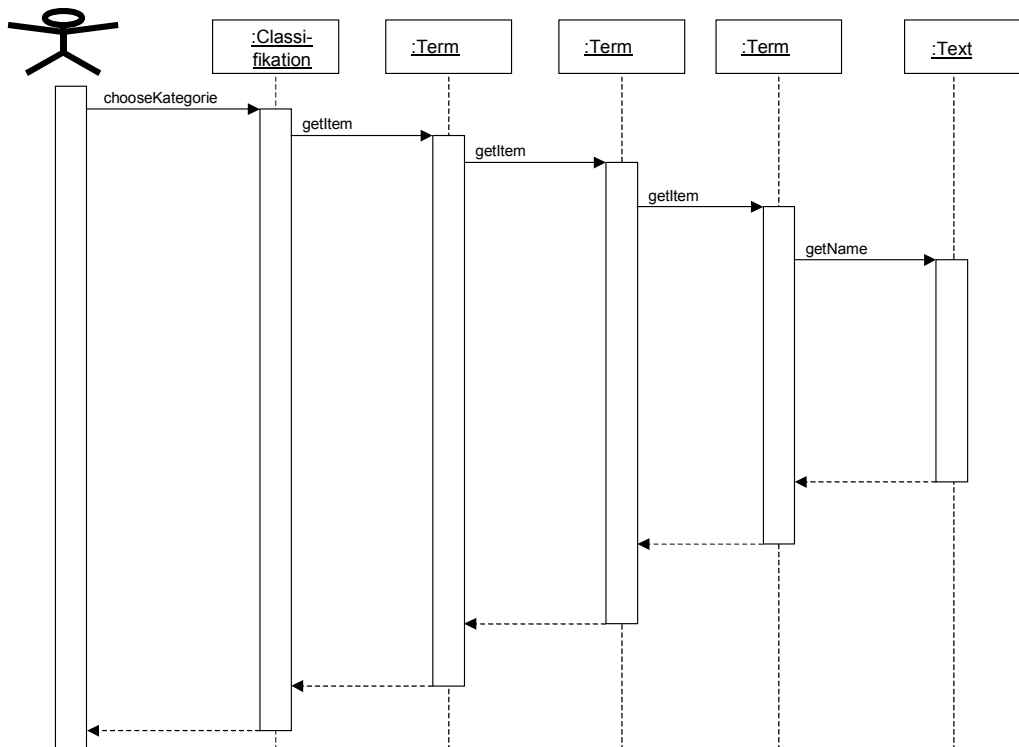


Abbildung 15: Kategorische Suche

Dieses geschieht solange, bis alle Einträge in der Hashtabelle Objekte der Klasse *Text* darstellen. Anschließend wird dort in der Methode *getName* der Textname zurückgeliefert, der in die Ausgabeliste übernommen wird. Die Ausgabeliste wurde hier als String realisiert, wobei die Einträge entsprechend der Kategoriestufe eingerückt sind. Der gesuchte Text kann nun vom User ausgewählt und durch das Aufrufen der Methode *getText* aus der Klasse *Text* bereitgestellt werden.

4. Rollenbasierte Modellierung und Änderungen des Metamodells

Rollen finden in der heutigen objektorientierten Modellierung immer mehr Beachtung, werden jedoch in dem aktuellen UML Standard nicht ihrer Bedeutung entsprechend berücksichtigt. Sie führen erstaunlicherweise immer noch ein Schattendasein und werden von den Modellierern nur sporadisch eingesetzt oder sogar ganz vernachlässigt. Im Rahmen dieser Arbeit soll durch eine Änderung des aktuellen Metamodells und der dadurch entsprechend geänderten Anwendung der aktuellen Notation den Rollen die Bedeutung zukommen, die sie in der heutigen Modellierung verdienen.

4.1. Der Rollenbegriff allgemein

Außerhalb der Softwaremodellierung werden in vielen Bereichen Rollen verwendet. Die wohl Bekannteste ist die Rolle eines Schauspielers in einem Theaterstück oder Film. Eine Rolle im Theater beschreibt, wen der Schauspieler im Stück zu spielen hat (z.B. König), vorgegeben durch den Rollentext. Des weiteren beschreibt sie die Eigenschaften des Schauspielers (z.B.: mächtig, streng) in dem Stück. Sie legt aber nicht fest, welcher Schauspieler die Rolle einnimmt. Aber auch außerhalb des Films treten Rollen auf: So kann ein Unternehmen die Rolle eines Abteilungsleiters kreieren. Auch hier werden lediglich die Eigenschaften dieser Stelle beschrieben, nicht aber, welche Person den Posten übernimmt.

In der Softwaremodellierung werden Rollen vom Verhalten her dieser Beschreibung entsprechend angewendet. Sie beschreiben die Eigenschaften, die die Instanzen einer Klasse haben müssen, um eine bestimmte Rolle spielen zu können. Sie legen jedoch nicht fest, welche Instanz die Rolle spielt. Dabei beschreibt die Rolle nur die Eigenschaften, die Umsetzung bzw. Implementation übernimmt sie nicht.

4.2. Rollenbegriffe in der aktuellen UML Version

In der aktuellen Ausgabe von UML (Version 1.3) existieren zur Zeit vier Rollenkonzepte [Steimann, Habilitationsschrift]:

1. Rollen als Enden von Assoziationen
2. Rollen als Teilnehmer einer Kollaboration.
3. Rollen von Akteuren (in einem Use Case-Diagramm)
4. (Rollen zum Zweck der dynamischen Klassifikation).

Die erste Verwendung wurde vom relationalen Modell über das Entity-Relationship-Modell auch in UML übernommen. Dieses Modell bildet die Basis der statischen Modellierung fast jeder objektorientierten Analyse und Designmethode. Das Ende einer Assoziation wird hier durch den Rollennamen eindeutig bezeichnet. Diese Verwendungsform ist dann sinnvoll, wenn eine Klasse mehrfach an einer Relation beteiligt ist und die Beteiligungen der Klasse eindeutig bestimmt werden müssen.

Wird ein Rollenname an einem Assoziationsende verwendet, so dient dieser der Instanz, die am anderen Ende der Assoziation liegt, zum Navigieren zu der Instanz an diesem Ende (dort wo der Rollenname steht). Der Rollenname stellt ein Pseudoattribut dar, da er gegenüber seiner Quelle eindeutig sein muss.

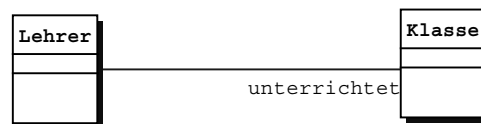


Abbildung 16: Beispiel für eine Rolle am Assoziationsende

So ist die Rolle „unterrichtet“ das Pseudoattribut für die Instanz von Lehrer, die auf eine oder mehrere Instanzen von Klasse zugreifen möchte. Das zweite Konzept bestimmt Rollen als Teilnehmer einer Kollaboration. Demnach kann derselbe Classifier oder dieselbe Assoziation in unterschiedlichen Kollaborationen, aber auch mehrfach in einer Kollaboration auftauchen, jedes Mal in einer unterschiedlichen Rolle. [OMG 1999, §2.10.1, S. 2-103]. Dabei wird bei jedem Auftauchen der Kollaborationsrolle unterschieden, welche Eigenschaften des Classifiers oder der Assoziation in diesem besonderen Fall benötigt werden. Die entsprechenden Eigenschaften werden von den Rollen festgelegt und erfüllen somit hier die Funktion von partiellen Spezifikationen der beteiligten Akteure.

Die Abbildung 17 zeigt ein Kollaborationsdiagramm auf Spezifikationsebene des aktuellen Metamodells. Die Aufteilung des Classifiers Person in die beiden Rollen Teacher und Student zeigt, dass nur Teacher zum Personal gehören kann. Diese Information geht aus dem Klassendiagramm (Abbildung 19a) nicht hervor. Bei dem Kollaborationsdiagramm auf Instanzebene würde sich das Diagramm weiter aufteilen, da dort die Rolle des Lehrers und des Studenten von konkreten Instanzen der Klasse Person (Teacher Smith, Student John) gespielt und somit auch einzeln dargestellt werden.

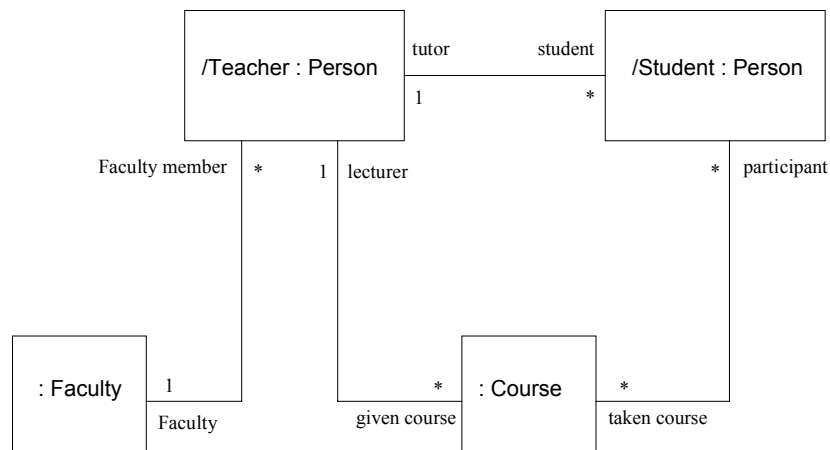


Abbildung 17: Kollaborationsdiagramm auf Spezifikationsebene [OMG 1999]

Weiterhin werden Rollen in UML in Use Case-Diagrammen benutzt. Ein User kann in verschiedenen Use Case-Diagrammen verschiedene Rollen spielen. So kann, bezogen auf das Hyperbook, ein User beispielsweise zum einen im Bereich Informatik die Rolle des Dozenten und im Bereich Linguistik die Rolle eines Hörers übernehmen. Wenn User über ihre Assoziationen mit ihren Use Cases verbunden sind und das Auftreten des Akteurs in der Assoziation als eine Rolle betrachtet wird, dann entspricht das einer Übertragung der ersten beiden Konzepte auf das Use Case-Diagramm.

Der vierte Punkt sei hier nur der Vollständigkeit halber aufgeführt. Er findet jedoch aktuell keine Anwendung und wird selbst in der aktuellen UML Notation nur mit einem Satz erwähnt [OMG 1999, S. 3-46].

Hervorzuheben sind insbesondere die ersten beiden Konzepte, da sie am häufigsten angewendet werden. Auch in dieser Arbeit werden im weiteren Verlauf nur die beiden ersten Konzepte berücksichtigt.

4.3. Interfaces in der aktuellen Version

Ein Interface dient der äußerlich sichtbaren Bereitstellung von Operationen von Klassen, Komponenten oder anderen Klassifizierern, ohne die interne Struktur darzulegen. Dabei spezifiziert ein Interface meistens nur einen bestimmten Teil des Verhaltens der aktuellen Klasse

und kann ausschließlich Operationen enthalten. Ein Interface kann dabei das Ziel einer Assoziation sein, aber es kann keine Assoziation navigieren.[OMG 1999].

4.4. Änderung des Metamodells

Um das unter Kapitel 5 beschriebene Rollenkonzept einführen zu können, müssen am Metamodell einige Änderungen vorgenommen werden, die hier kurz aufgezeigt werden sollen [Steimann, Habilitationsschrift] .

Zuerst werden die Konzepte der Rollen (Classifierrole) und Interfaces zusammengelegt. Da Rollen und Interfaces in UML ohnehin recht ähnlich verwendet werden, verliert das Modell durch diese Änderung nicht an Ausdruckstärke. Die Rollen und Interfaces werden hier in der üblichen Lollipopnotation, die in UML für die Darstellung von Interfaces benutzt wird, abgebildet. Dadurch muss kein neues Symbol eingeführt werden und eine Änderung der Notation erübrigt sich.

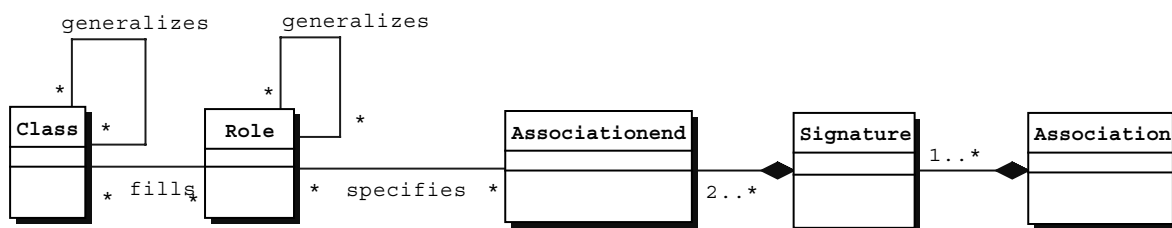


Abbildung 18: Das geänderte Metamodell

Klassen und Rollen werden voneinander unterschieden, wobei lediglich Klassen instanziiert werden können, Rollen dagegen nicht. Die gemeinsame Beschreibung durch den Classifier (Abbildung 3) verliert somit seine Bedeutung. Die Assoziation specifies geht aus den zusammengefassten Pseudoattribute type und specification vom Assoziationsende hervor. Sie ist genau einer Rolle zugeordnet. Diese beschreibt, welche Eigenschaften die Assoziation von den Instanzen der Klasse fordert, damit sie die Assoziation verwenden können. Assoziationen verbinden nicht mehr Klassen, sondern nur noch Rollen. So kann das Konzept Rollenname im aktuellen Modell entfallen, da dieses von den Rollen übernommen wird. Jede Rolle muss innerhalb der Assoziation eindeutig sein und somit auch das Assoziationsende, das von der Rolle bezeichnet wird. Eine Klasse kann mehrere Rollen füllen und eine Rolle kann von mehreren Klassen gefüllt werden.

4.5. Arbeiten mit dem geänderten Metamodell

In der Praxis erweist es sich häufig als schwierig, eine neue Notation für ein geändertes Metamodell einzuführen. Das liegt darin begründet, dass viele Modellierungswerkzeuge die aktuelle Notation unterstützen. Bei einer neuen Notation müssten somit eine Vielzahl der Anwender zum einen diese erst einmal erlernen und verinnerlichen und zum anderen wären in der Regel neue Versionen der Modellierungswerkzeuge nötig, die die neue Notation unterstützen. Der große Vorteil dieser Änderung liegt darin, dass trotz geändertem Metamodells die alte Notation beibehalten werden kann [Steimann, Rollen]. Dadurch dürfte eine schnellere Akzeptanz des rollenbasierten Modellierens in UML unter den Softwaredesignern zu erwarten sein.

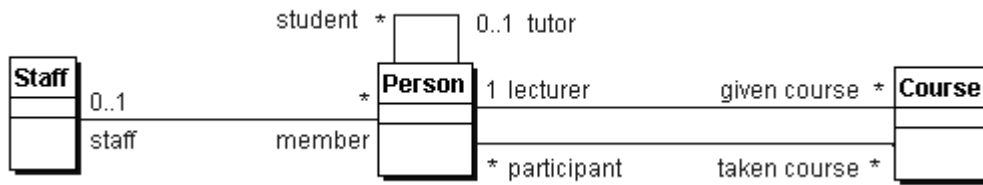
Nachfolgend sollen die Auswirkungen auf die Klassendiagramme und das Kollaborationsdiagramm erläutert werden. Auf eine Darstellung der Sequenzdiagramme wurde hier verzichtet, da die Einführung der Rollen für sie ohne Auswirkungen bleibt.

4.5.1. Klassendiagramme

Die Klassen werden auf Instanzen von *Class* und Interfaces auf Instanzen von *Role* abgebildet. Die Assoziationsenden bleiben im neuen Metamodell unverändert. Im Gegensatz zum jetzigen Metamodell sind die Classifier, die eine Klasse darstellen und vorher auf dem Pseudoattribut *type* abgebildet wurden, nicht mehr direkt mit den Assoziationsenden verbunden. Das Assoziationsende steht nunmehr mit einer Instanz von *Role* in Verbindung. Die Instanz von *Role* ist über *specifies* (Abbildung 18) mit dem Assoziationsende verbunden. Die Rolle wird wiederum von dem Classifier als Instanz der Klasse gefüllt. Die Instanz der Rolle trägt dabei den Namen (sofern vorhanden), der mit dem Assoziationsende verbunden ist, dem alten Rollennamen.

Im Klassendiagramm kann ein Assoziationsende auch mit einem oder mehreren Interface spezifiern ausgestattet sein (Diese Variante tritt jedoch selten auf). Interface specifier im neuen Metamodell entsprechen den Werten des Pseudoattributs *specification* im aktuellen Metamodell. Die verschiedenen Interface specifier des Assoziationsendes werden auf der Rolle abgebildet, die die größte gemeinsame Unterrolle der Interfaces (Rollen) darstellt. Die Rolle trägt den Namen des Assoziationsendes und entspricht derjenigen, die beim Benutzen eines Classifiers entsteht. Um Interface specifier und Classifier (bzw. die Pseudoattribute *type* und *specification* in Abbildung 3) zu vereinen, sollte einer Klasse als Classifier diese die Rolle füllen, die aus dem Interface specifier hervorgeht [Steimann, Habilitationsschrift].

a)



b)

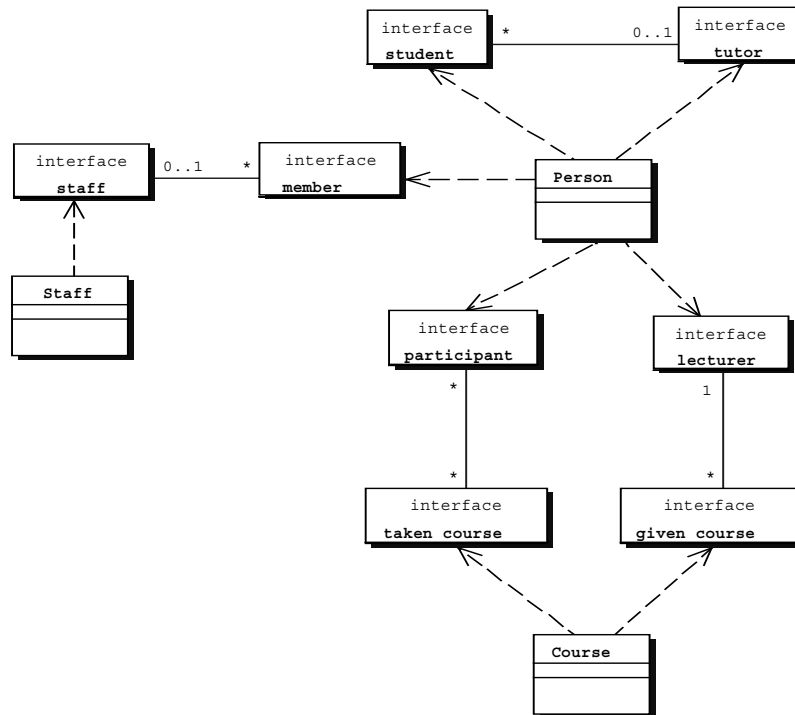


Abbildung 19: (a) herkömmliches Klassendiagramm und (b) Klassendiagramm mit expliziten Rollen [Steimann, Habilitationsschrift, Abbildung 4-7]

Die Rollen wurden hier als Interfaces dargestellt, da die Konzepte Rollen und Interfaces nach dem neuen Metamodell (siehe Abschnitt 4.4) zusammengelegt wurden. Die Darstellung der Interfaces in der Abbildung 19 entspricht aufgrund des verwendeten Modellierungswerkzeuges Together J nicht der oben genannten Lollipopnotation (Absatz 2.2.2). Bei einer Darstellung der Interfaces in den Diagrammen von Together J in der Lollipopnotation können die Operationen der Interfaces nicht mit angezeigt werden. Da diese aber im nächsten Kapitel abgebildet werden sollen, wurde diese Darstellung der Interfaces gewählt. Sie entspricht aber trotzdem der aktuellen UML-Notation [OMG 1999, §3.28.2, S 3-48].

4.5.2. Kollaborationsdiagramm

Im jetzigen UML-Kollaborationsdiagramm auf Spezifikationsebene treten Classifier-Rollen und Assoziationsrollen an die Stelle von Classifiern und Assoziationen. Da das geänderte Metamodell nur eine Art von Rollen kennt, werden die Assoziationsrollen und die Assoziationsendenrollen als ganz normale Assoziationen bzw. Assoziationsenden behandelt. Die Classifier-Rollen des UML-Kollaborationsdiagramm auf Spezifikationsebene bilden die Rollen im geänderten Metamodell.

Das UML-Kollaborationsdiagramm auf Instanzebene stellt im Prinzip eine alternative Darstellung des Sequenzdiagramms dar. Es wird hier nicht weiter aufgeführt, da die Instanzebene in diesem Metamodell keine Berücksichtigung findet.

4.5.3. Bedeutung für die Praxis

Mit dem geänderten Metamodell können Softwaredesigner weiterhin die auf dem Markt üblichen Entwicklungswerkzeuge verwenden. Darüber hinaus kann jedoch durch ein Umdenken mit wenigen Änderungen ein rollenreiches Design erstellt werden. Da im neuen Design die Assoziationen in Rollen enden und diese von Klassen gefüllt werden, können zwei Klassen nicht mehr wie bisher im Klassendiagramm mit einer Assoziation direkt verbunden werden. Die Einschränkungen der einzelnen Assoziationen wurden im ursprünglichen Klassendiagramm nicht hervorgehoben. Dies steht im Gegensatz zum jetzigen Modell, da nach dem neuen Modell nur die Instanzen einer Klasse die Assoziation benutzen können, die auch die entsprechende Rolle am Ende der Assoziation füllen, z. B: nur eine Person die auch *Lecturer* ist, kann einen *course* geben (siehe Abbildung 19 b). Dadurch werden schon im Klassendiagramm die Einschränkungen der Rollen deutlich, die sonst nur im Kollaborationsdiagramm dargestellt werden konnten. So rücken die Klassen und Kollaborationsdiagramme näher zusammen.

5. Entwurf des Hyperbooks nach dem geänderten Metamodell

Wie im vorigen Kapitel beschrieben, können Klassen nicht mehr direkt mit den Assoziationsenden verbunden werden, sondern sind über Rollen mit den Assoziationen verbunden. Dabei füllen die Klassen die entsprechenden Rollen. Die Rollen werden als Interfaces implementiert. Aufgrund dieser Vorgehensweise tritt bei der Modellierung insofern ein Problem auf, als in JAVA die Assoziationen nicht durch eine mit dem Interface deklarierte Instanzvariable implementiert werden können. In JAVA können Interfaces laut Spezifikation keine Variablen beinhalten. Die Realisierung muss somit von den entsprechenden Klassen übernommen werden, zum Beispiel in der Klasse Textbase durch die Instanzvariable Textbaseelements, oder im Package Classification durch die Instanzvariablen Entries in den Klassen Term und Classification. In den folgenden Beschreibungen der Packages wird dieser Umstand nicht explizit dargelegt: Wenn eine Relation zwischen zwei Rollen mittels einer Aggregation oder Assoziation beschrieben wird, erfolgt die eigentliche Realisierung durch die Variable der entsprechende Klasse.

Der nachfolgende Entwurf erfolgt über die Klassendiagramme, da hier die Änderungen durch die Einführung der Rollen auftreten und am besten zu sehen sind.

5.1. Package Textbase

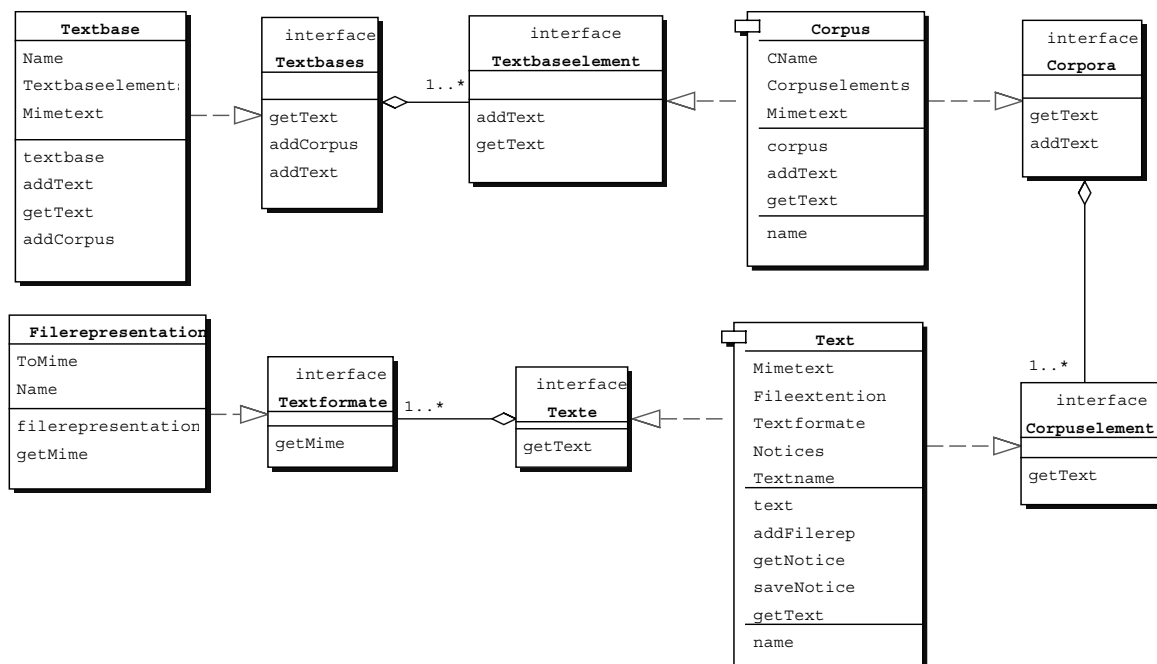


Abbildung 20: Package Textbase nach dem neuen Metamodell

Während im vorherigen Design die Klasse *Textbase* mit der Klasse *Corpus* direkt mittels einer Aggregation verbunden war, muss nun die Aggregation an zwei Rollen enden. In diesem Fall endet die Aggregation auf Seiten der Klasse *Textbase* in der Rolle *Textbases* und auf der Seite der Klasse *Corpus* in der Rolle *Textbaseelements*. Die Instanzen der Klassen *Textbase* und *Corpus* müssen die entsprechenden Rollen füllen, wenn sie sie spielen wollen. Damit eine Instanz von *Textbase* auf diejenigen Corpora, aus denen sie besteht, zugreifen kann, muss sie die Rolle *Textbases* spielen. Die jeweiligen Corpora füllen wiederum die Rolle *Textbaseelements*. Analog verhält es sich mit der Relation zwischen den Klassen *Corpus* und *Text*: Die Objekte der Klassen müssen hier die Rollen *Corpora* bzw. *Corpuselemente* füllen. Ein Instanz von *Corpus* spielt die Rolle *Corpora* und hat mehrere Texte als Elemente. Die Texte spielen jeweils die Rolle eines Corpuselementes. Um die Rolle *Corpuselement* spielen zu können, müssen die Texte, die Instanzen der Klasse *Text* darstellen, diese auch füllen. Das Speichern der Texte erfolgt in verschiedenen Textformaten, um in allen Formaten dargestellt werden zu können. Die einzelnen Textformate beschreiben Instanzen der Klasse *Filerepresentation*. Um den Text im gewünschten Textformat abzurufen, muss ein Text (Instanz der Klasse *Text*) die Rolle *Texte* übernehmen und auch füllen. Die Verbindung mit der Rolle *Textformate* erfolgt über die Relation „im Format“, die durch eine Aggregation realisiert wurde, da die unterschiedlichen Formate des Textes Teile der jeweiligen Instanz von *Text* sind.

5.2. Package Vorlesung

Wenn eine Vorlesung, dargestellt durch eine Instanz der Klasse *Lecture*, die ihr zugehörigen Unterrichtseinheiten aufrufen oder eine neue Einheit hinzufügen möchte, so muss sie die Rolle *Lectures* spielen. Die Rolle *Lectures* ist über eine als Aggregation realisierte Relation mit der Rolle *Lectureelement* verbunden. Die Unterrichtseinheiten (Instanzen der Klasse *Lesson*) müssen die Rolle *Lectureelement* füllen. Bei einem Aufruf ihrer Operationen (z.B. *getLesson*) spielt ein Objekt der Klasse *Lesson* die Rolle des Vorlesungselements (*Lectureelement*). Um anschließend auf die in der Vorlesungsstunde verwendeten Unterrichtstexte zugreifen zu können, muss es ebenfalls die Rolle *Lessons* einnehmen. Der Zugriff erfolgt über die Beziehung Unterrichtsstunden (Rolle *Lessons*) zu Unterrichtselemente (Rolle *Lessonelement*). Die einzelnen Vorlesungstexte in der Klasse *Text* füllen die Rolle *Lessonelement*. Da auch hier sowohl die Reihenfolge der Vorlesungsstunden als auch die Reihenfolge der in den Vorlesungsstunden verwendeten Texte durch den Dozenten beim Einfügen vorgegeben werden, wurden hier die Aggregation als Vektor realisiert.

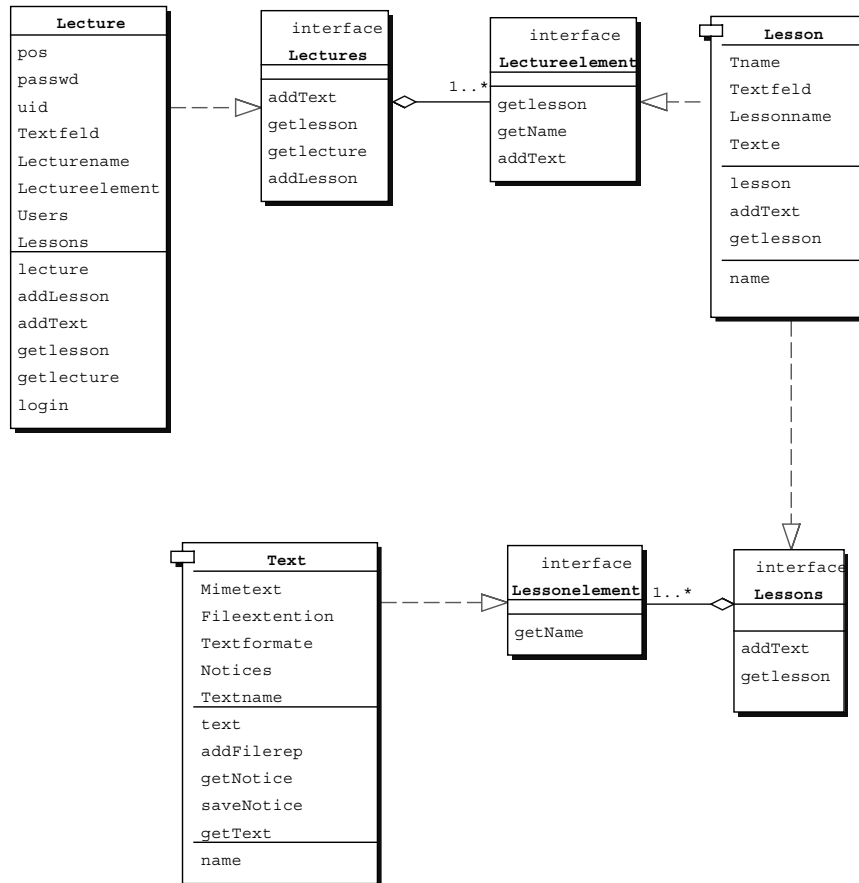


Abbildung 21: Package Vorlesung nach dem neuen Metamodell

5.3. Package Notizen

Das Package Notizen lässt sich, den Zugriff betreffend, in zwei Bereiche einteilen: Zum einen wird die Anmeldung des Hörers hier abgearbeitet, zum anderen das Abrufen und Ändern der persönlichen Notizen des Users.

Beim Anmelden eines Users wird in der Klasse *Lecture* die Methode *login* aufgerufen. Diese fragt den User nach seinem Usernamen und dem Passwort. Um die für die Nutzung der Notizen notwendige UserID zu bekommen, muss die Klasse *Lecture* die Rolle *Identifizierende* spielen. Die Relation identifizieren, hier durch ein Aggregation realisiert, verbindet die beiden Rollen *Identifizierende* und *Identifizierter*. Über sie kann die Instanz von *Lecture* die Methode *getUid* der Rolle *Identifizierter* aufrufen. Die Instanzen der Klasse *Hoerer* füllen diese Rolle. In der aufgerufen Instanz wird das Passwort des Users mit seinem dort gespeicherten Passwort verglichen. Bei Übereinstimmung liefert diese, die entsprechende UserID an die Klasse *Lecture* zurück. Die Klasse *Lecture* leitet die UserID für den weiteren Gebrauch an die Benutzerschnittstelle weiter.

Der zweite Bereich beinhaltet die Klassen und Rollen, die für den eigentlichen Gebrauch der Notizen nötig sind. Um eine Notiz bearbeiten oder abrufen zu können, muss ein Text der Klasse *Text* die Rolle des Annotierenden spielen. Über die Relation *annotieren*, die die beiden Rollen *Annotierende* und *Annotierte* verbindet, kann die Instanz der Klasse *Text* auf die Operationen der Rolle *Annotierte* zugreifen. Die Relation *annotieren* zwischen den Rollen *Annotierende* und *Annotierte* wurde über eine Aggregation realisiert. Da die Operationen in den Interfaces (Rollen) nur deklariert und nicht implementiert werden, müssen Instanzen der Klasse *Notice* die Rolle *Annotierte* füllen. Die Ausführung der aufgerufenen Operationen geschieht dann in der Klasse *Notice* (*getNotice* o. *saveNotice*).

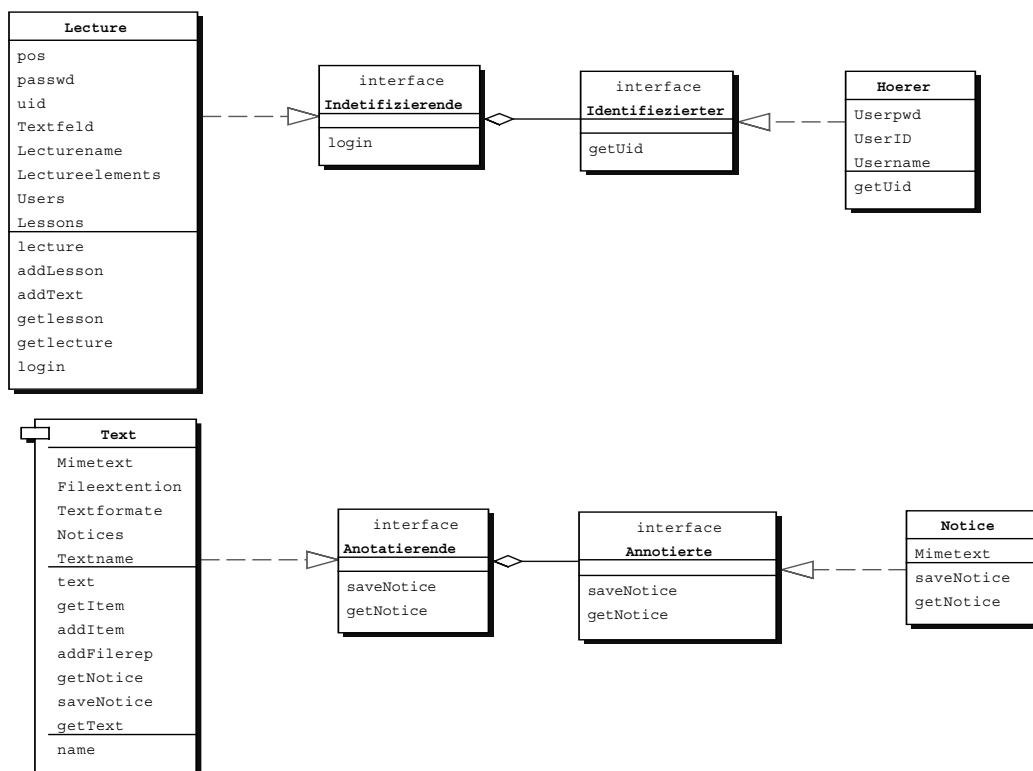


Abbildung 22: Package Notizen nach dem neuen Metamodell

5.4. Package Indexsearch

Damit ein User sich die Schlagwörter einer Vorlesung anzeigen lassen kann, muss die Instanz von *Index* die Methoden der Klasse *Keyword* aufrufen können. Dafür muss sie die Rolle *Indexe* übernehmen. Die Rolle *Indexe* ist über eine als Aggregation realisierte Relation mit der Rolle *Indexelement* von der Klasse *Keyword* verknüpft. Bei dem Aufruf der Methoden spielt die aufgerufene Instanz der Klasse *Keyword* die Rolle *Indexelement* und füllt diese auch. Sollen die unter den Schlagwörtern abgelegten Texte angezeigt bzw. ein neuer Text hinzugefügt

werden, so muss das Schlagwort als Instanz von *Keyword* die Rolle *Keywords* spielen. Diese ist über eine Assoziation mit der Rolle *Keywordelement* der Klasse *Text* verbunden. Hier wurde die Relation als Assoziation und nicht als Aggregation verwirklicht, da Texte nicht Teile eines Schlagwortes sein können, sondern nur durch diese charakterisiert werden. Die Instanzen der Klasse *Text* müssen die Rolle *Keywordelement* füllen. So können sie diese Rollen auch übernehmen und ihre Operationen können von den Instanzen der Klasse *Keyword* aufgerufen werden.

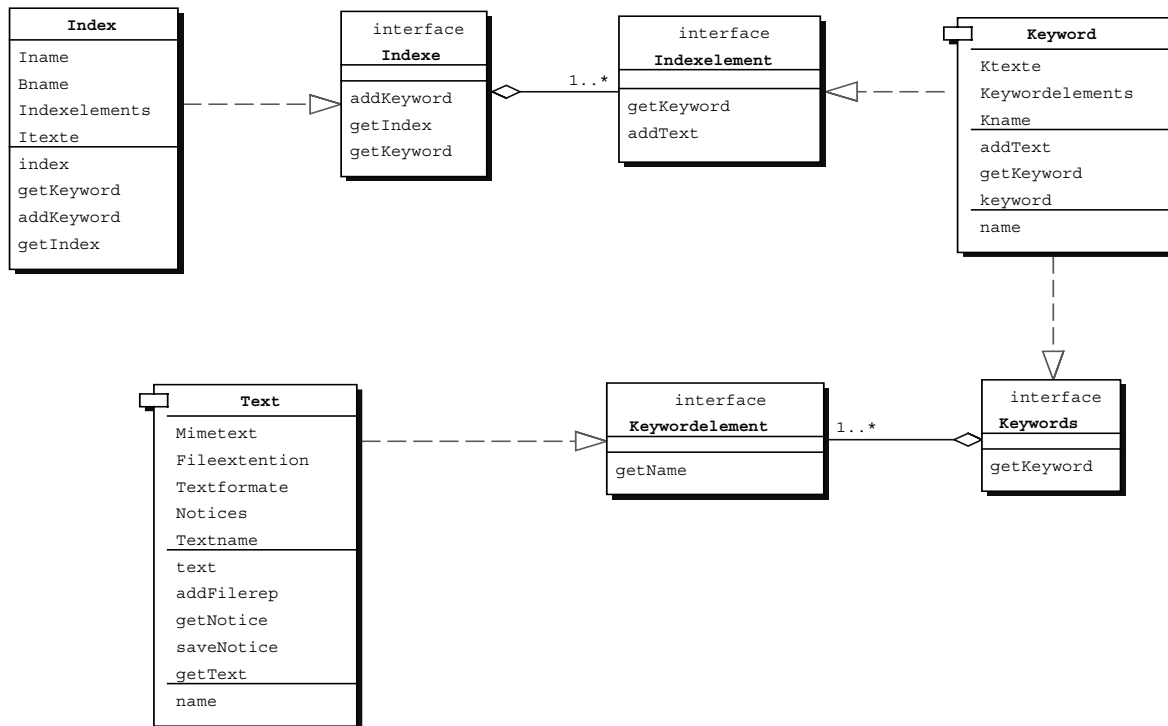


Abbildung 23: Package Indexsearch nach dem neuen Metamodell

5.5. Package Classifikation

Bei der kategorischen Suche gibt es eine Anzahl von Komponenten, die entweder Klassifikatoren oder Einträge sind. Beides sind Rollen und im Diagramm durch Classifier und Entry dargestellt. Ein Klassifikator klassifiziert einen Eintrag, diese Beziehung wird durch die Aggregation modelliert. Während ein Text nur die Rolle eines Eintrages spielen kann, kann ein Term sowohl die Rolle eines Eintrages als auch die Rolle eines Klassifikators spielen. Die Rolle eines Eintrages spielt ein Term, wenn er von seinem vorhergehenden Klassifikator aufgerufen wird. Nach seinem Aufruf spielt der Term die Rolle eines Klassifikators, indem er seine eigenen Einträge aufruft. Dadurch entsteht hier wieder eine Rekursion wodurch eine unendliche Verschachtelungstiefe möglich wäre. Die Einträge eines Terms können sowohl

wieder Terme oder Texte sein, so dass ein Term immer die Rollen Klassifikator und Eintrag spielt. Hat ein Term (Kategorie) keine Subterme (Unterkategorien) und somit nicht den Klassifikator für andere Terme spielt, so sind ihm immer noch Texte untergeordnet, die er klassifiziert. Für diese untergeordneten Texte spielt der Term wiederum die Rolle des Klassifikator.

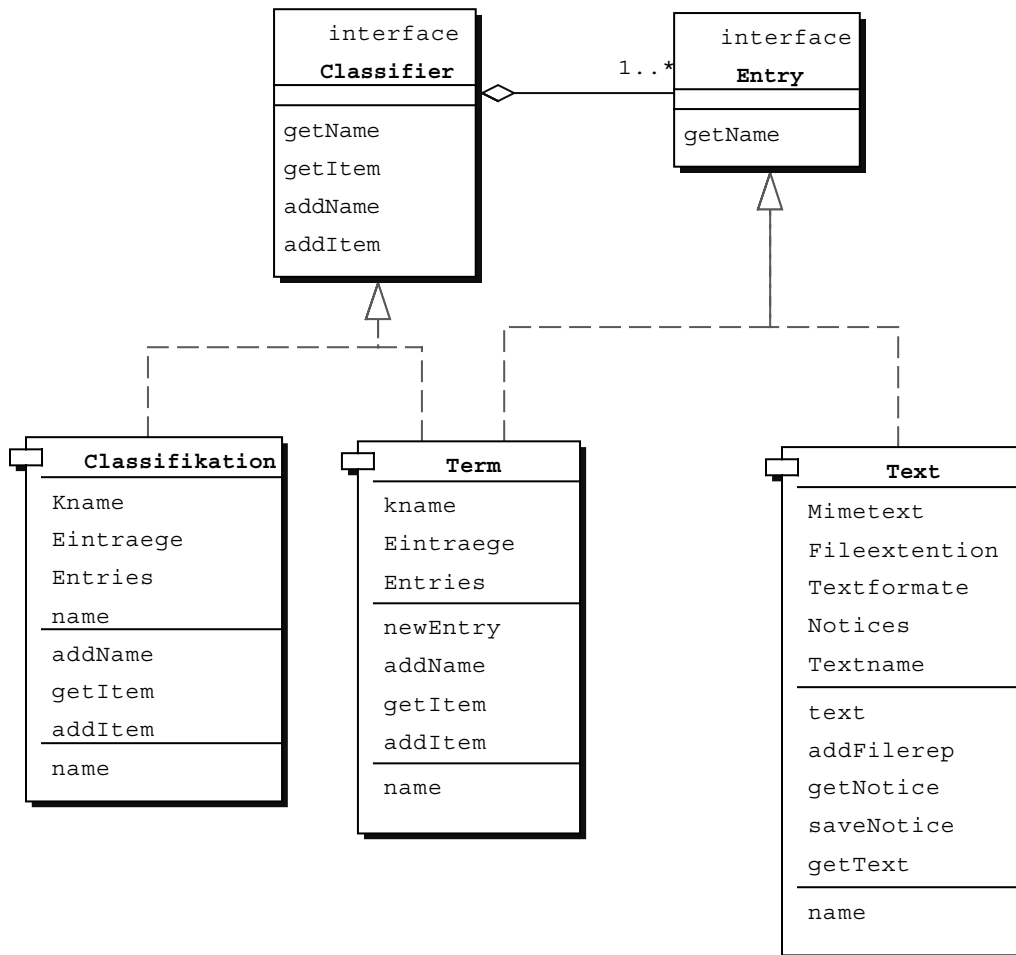


Abbildung 24 : Package Classification nach dem neuen Metamodell

Der Einstieg in die kategorische Suche wurde in dem vorherigen Entwurf über die Aggregation zwischen Klassifikation und Entry verwirklicht. Da die abstrakte Klasse Entry hier von der Rolle Entry übernommen wird, ist man im ersten Augenblick geneigt, eine neue Rolle einzuführen, die von den Instanzen der Klasse Klassifikation gefüllt wird. Diese Klasse wäre dann über eine Aggregation mit der Rolle Entry verbunden. Vergleicht man jedoch diese Rolle mit der Rolle Classifier, so stellt man fest, dass sie von der Art her gleich sind. Sie beinhalten die gleichen Methoden und greifen beide über die Rolle Entry auf ihre Untereinträge zu. Somit stellt sich die Frage, ob die Klasse *Klassifikation* nicht auch ein Term sein und somit weggelassen werden könnte. Dieses ist nicht der Fall, denn die Instanzen der Klasse *Klassifi-*

kation rufen zwar ihre Untereinträge auf und können damit die Rolle *Classifier* füllen und spielen, aber die Instanzen stellen keine Einträge eines Terms dar und spielen somit nicht die Rolle *Entry*. Da die Objekte von *Classifikation* aber keine Einträge sind, können sie auch keine Instanz der Klasse *Term* darstellen. Deshalb bleibt die Klasse *Classifikation* als eigene Klasse bestehen und füllt die Rolle *Classifier*.

Auch hier soll ein Beispiel zur näheren Erklärung der Funktionsweise des Packages aufgeführt werden: Wieder wird der Text über den Lichtschalter gesucht. Da im letzten Beispiel empfohlen wurde, die Anzahl der Unterkategorien zu begrenzen, gibt es in diesem Beispiel nur drei Unterkategorien.

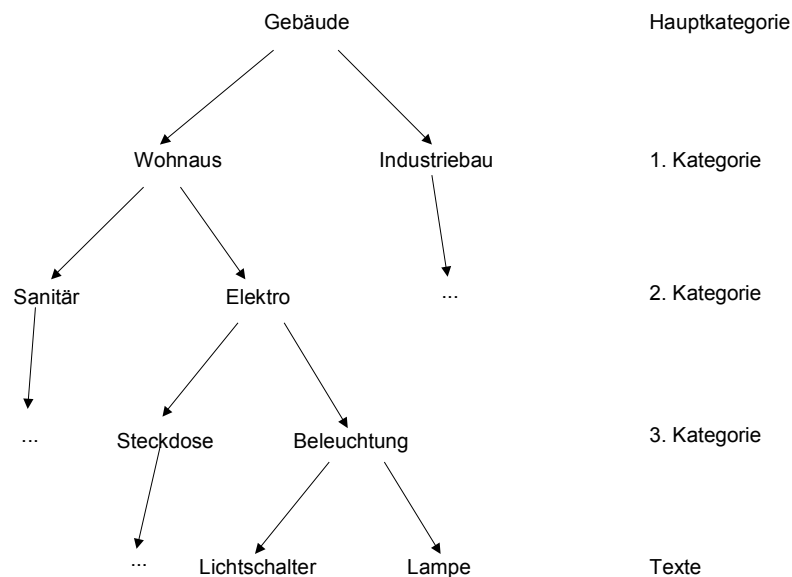


Abbildung 25: Beispielstruktur der kategorischen Suche

Um die ersten Unterkategorien der Hauptkategorie Gebäude aufzurufen, wird die Methode *getItem* in der Klasse *Classifikation* angewendet. Die Instanz Gebäude der Klasse *Classifikation* muss die Rolle *Classifier* spielen, um ihre Einträge (Haus, Fabrik) aufzurufen. Dieser Aufruf erfolgt über die Relation klassifizieren (Aggregation zwischen Rollen *Classifier* und *Entry*). Die Namen der Einträge werden an die Benutzerschnittstelle zurückgegeben. Im nächsten Schritt werden die Unterkategorien von Haus gesucht. Dazu ruft die Instanz Gebäude der Klasse *Classifikation* über die Relation klassifizieren die Methode *getItem* in der Instanz Haus der Klasse *Term* auf. Hier muss die Instanz Haus die Rolle *Entry* übernehmen. In der Methode *getItem* werden die Namen der Unterkategorien von Wohnhaus rekursiv über die

Aggregation zwischen *Classifier* und *Entry* ausgelesen. Dazu spielt die Instanz Wohnhaus der Klasse *Term* die Rolle *Classifier* und die Instanzen Sanitär und Elektro die Rolle *Entry*. Die Namen werden wiederum an die Benutzerschnittstelle zurückgeliefert. Um die Unterkategorien von Einrichtung zu bestimmen, wird wieder die Methode *getItem* in den schon vorher durchlaufenen Instanzen rekursiv aufgerufen, bis sie in der Instanz Elektro aufgerufen wird. Dort werden die Namen der Unterkategorien (Steckdose, Beleuchtung) bestimmt. Dieses geschieht auf die gleiche Weise wie vorher in der Instanz Wohnhaus. Die Kategorie Elektro spielt die Rolle des Klassifikators und die Kategorien Beleuchtung und Steckdose spielen jeweils die Rolle *Entry*. Das Auslesen der Einträge der Kategorie Beleuchtung erfolgt analog. Allerdings handelt es sich hier bei den Einträgen um Texte. Somit spielt die Instanz Beleuchtung der Klasse *Term* zwar die Rolle des Klassifikators, aber die Rolle *Entry* wird dieses Mal von Instanzen der Klasse *Text* (Lampe, Lichtschalter) übernommen. Die Instanzen der Klasse *Text* können nicht die Rolle *Classifier* einnehmen, so dass dort auch keine weiteren Unterkategorien möglich sind. Die Textnamen werden an die Benutzerschnittstelle zurückgegeben und der gewünschte Text über den Lichtschalter kann von dieser aufgerufen werden.

6. Vergleich der Entwürfe und Beurteilung

Im Nachfolgenden sollen die beiden Entwürfe (konventioneller Entwurf vs. Rollenbasierter Entwurf) miteinander verglichen werden, um darauf aufbauend den rollenbasierten Entwurf beurteilen zu können. Dabei werden insbesondere die Klassendiagramme eingehend begutachtet.

6.1. Vergleich vom konventionellen und dem rollenbasierten Entwurf des Hyperbooks

Das Package Textbase weist aufgrund seines einfachen Aufbaus eine relativ ähnliche Struktur auf (Abbildung 26).

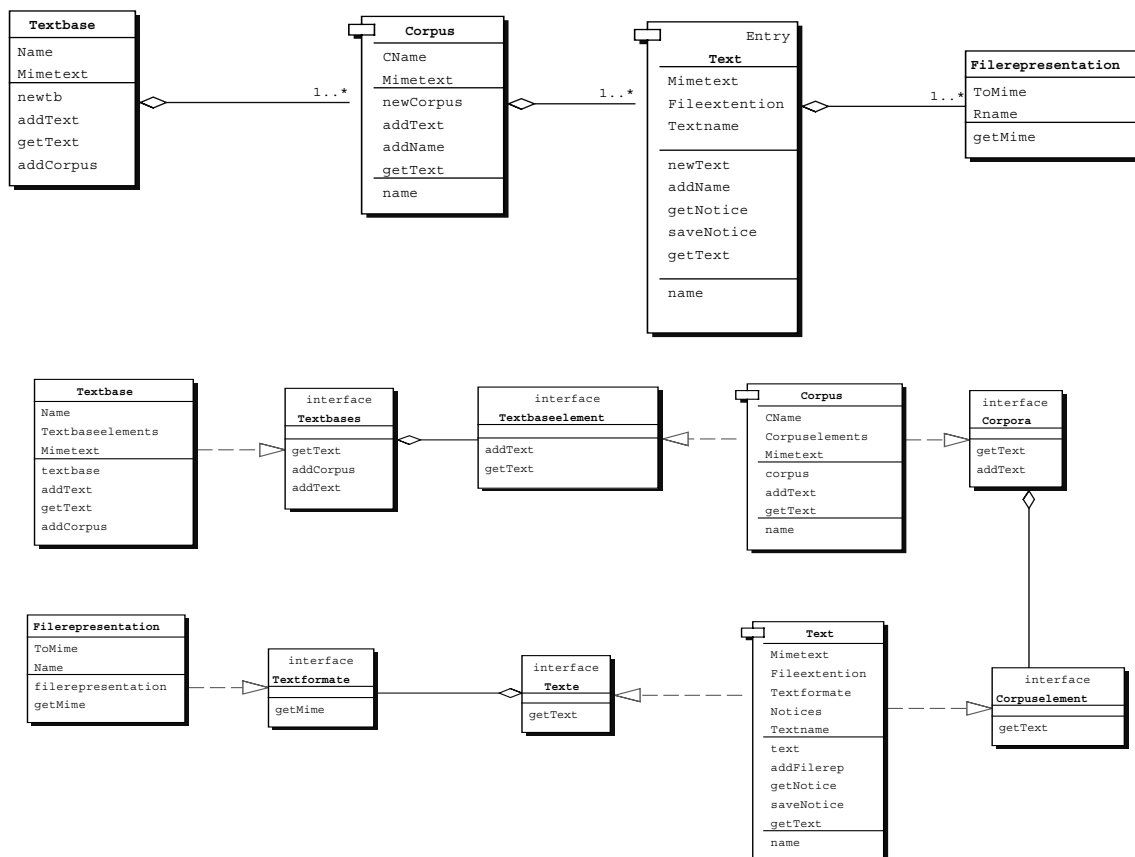


Abbildung 26: Gegenüberstellung der Packages Textbase von Abb. 5 und Abb. 20

Durch das Einführen der Rollen *Textbases* und *Textbaseelement* wird die Relation zwischen den Klassen *Textbase* und *Corpus* näher beschrieben. Alle *Textbases* müssen die gleiche Rolle einnehmen, um auf ihre *Corpora* zugreifen zu können und ebenso müssen die *Corpora* die gleiche Rolle (*Textbaseelement*) füllen, damit ihre Methoden von den Instanzen der Klasse

Textbase aufgerufen werden können. Ähnlich verhält es sich zwischen der Klasse *Corpus* und *Text*: Von Seiten der Klasse *Corpus* entspricht die Rolle *Corpora* in der Relation zwischen *Corpora* und *Corpuselements* der Rolle *Textbases* in der Beziehung zwischen *Textbases* und *Textbaseelement*. Die Rolle *Corpuselements* entspricht von der Funktion her der Rolle *Textbaseelement*, wobei hier die Einschränkungen durch die deklarierten Methoden für die Instanzen der Klasse *Text* wesentlich größer ausfallen. Während bei der Rolle *Textbaseelements* quasi alle Methoden der Klasse *Corpus* deklariert werden, ist durch die Rolle *Corpuselement* nur der Zugriff auf die Methode *getText* möglich. Die anderen Methoden wie z.B. *getNotice* stehen nicht zur Verfügung. Diese gesamten Informationen über die Art des Zugriffes und deren Beschränkungen werden im konventionellen Klassendiagramm nicht deutlich. Bei der Relation zwischen den Klassen *Text* und *Filerepresentation* wird im konventionellen Klassendiagramm ebenfalls nicht ersichtlich, welche Methoden der Klasse *Text* auf die Instanzen der Klasse *Filerepresentation* zugreifen. Im rollenbasierten Entwurf dagegen kann aus dem Klassendiagramm nachvollzogen werden, dass lediglich die Methode *getText* auf die Klasse *Filerepresentation* zugreift. Die Art der Zugriffe werden bei dem konventionellen Modell erst im Sequenzdiagramm deutlich. In beiden Entwürfen erkennt man, dass zwischen den einzelnen Klassen jeweils nur eine Relation besteht. Da auch nach dem Einführen von Rollen nur eine Relation mit jeweils einer Rolle an jedem Ende zwischen den Klassen besteht, fällt der Vorteil der besseren Lesbarkeit des Klassendiagramms nicht besonders stark ins Gewicht.

Im Package Vorlesung sind die Klassen *Lecture* und *Lesson* durch eine Aggregation verbunden, im rollenbasierten Entwurf stehen am Ende der Relation die Rollen *Lectures* und *Lectureelement*. Während im rollenbasierten Entwurf auf Seiten der Klasse *Lecture* durch die Rolle *Lectures* deutlich wird, dass die Methode *login* zum Aufrufen der Methoden in der Klasse *Lesson* nicht benötigt wird, gehen aus der Rolle *Lectureelement* keine Einschränkungen für den Zugriff hervor, da alle Methoden der Klasse *Lesson* in der Rolle deklariert sind. Zwischen den Klassen *Lesson* und *Text* besteht auch nach dem Einführen der Rollen nur eine Relation, die an den Rollen *Lessons* und *Lessonelement* endet. Durch die Rolle *Lessonelement* wird deutlich, dass der Zugriff von der Klasse *Lesson* auf die Methode *getName* der Klasse *Text* reduziert ist. Im konventionellen Klassendiagramm ist diese Einschränkung nicht ersichtlich, sie wird erst im Kollaborations- bzw. Sequenzdiagramm deutlich.

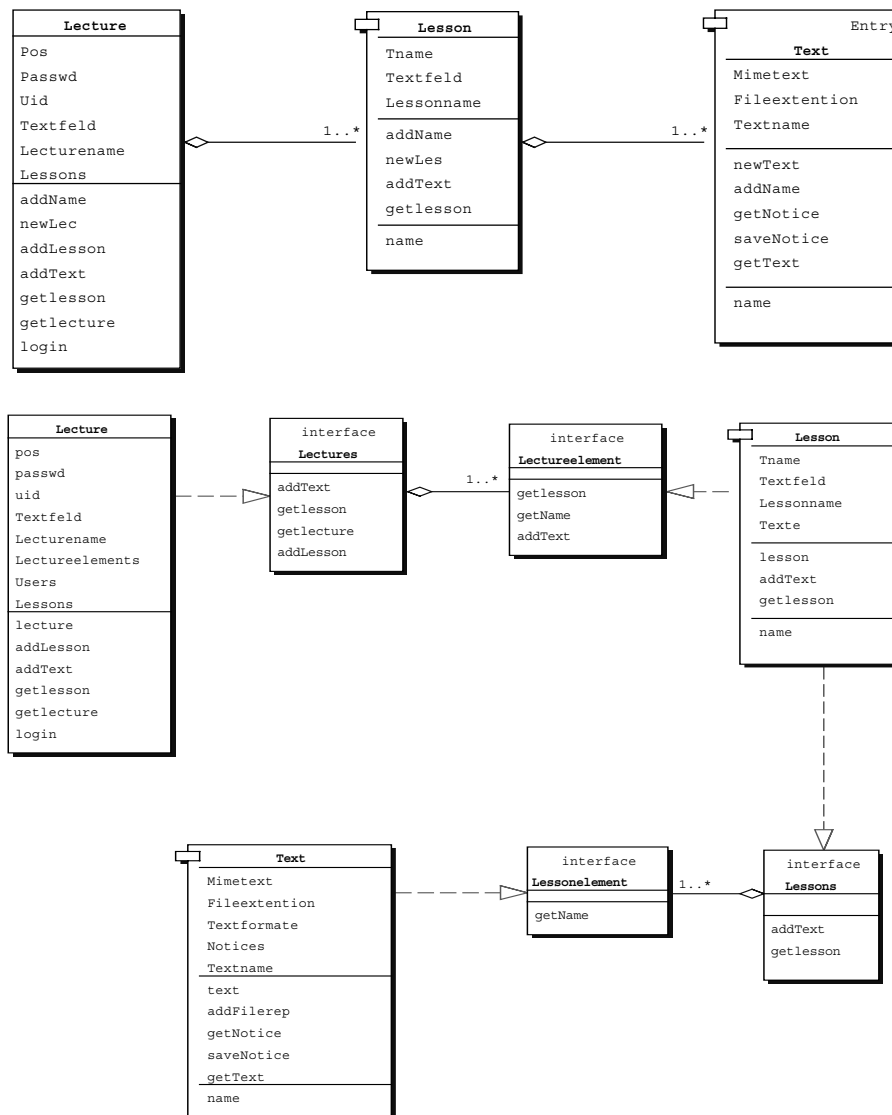


Abbildung 27: Gegenüberstellung der Packages Vorlesung von Abb. 6 und Abb. 21

Im Package Notizen ruft die Klasse *Lecture* die *UserID* ab, die in der Klasse *Hoerer* gespeichert wird. Im konventionellen Entwurf besteht eine Assoziation zwischen den beiden Klassen (vgl. Abbildung 7), diese endet im rollenbasierten Entwurf in den Rollen *Identifizierende* und *Identifizierter* (vgl. Abbildung 22). Während in der Relation ohne Rolle nicht ersichtlich ist, welche Funktion der Assoziation zukommt, geht aus dem rollenbasierten Entwurf deutlich hervor, dass die Methode *login* der Klasse *Lecture* die *UserID* über ihre Rolle *Identifizierende* in der Klasse *Hoerer* abrufen. Die entsprechende Instanz der Klasse nimmt die Rolle *Identifizierter* ein und stellt somit die Methode *getUid* zur Verfügung. Die restlichen Methoden der Klasse *Lecture* werden für die Abfrage nicht benötigt und somit auch nicht in der Rolle *Identifizierende* berücksichtigt. Im zweiten Teil des Packages stehen die Klassen *Text* und *Notice*

durch eine Aggregation in Verbindung, die beim rollenbasierten Klassendiagramm in den Rollen *Annotierende* und *Annotierte* endet. Die Rolle *Annotierende* verdeutlicht, dass nur die Methoden *getNotice* und *saveNotice* in der Klasse *Text* über die Relation auf die Methoden der Klasse *Notice* zugreifen. Im konventionellen Klassendiagramm ist nicht ersichtlich, welche Methoden in *Text* die Methoden der Klasse *Notice* aufrufen können.

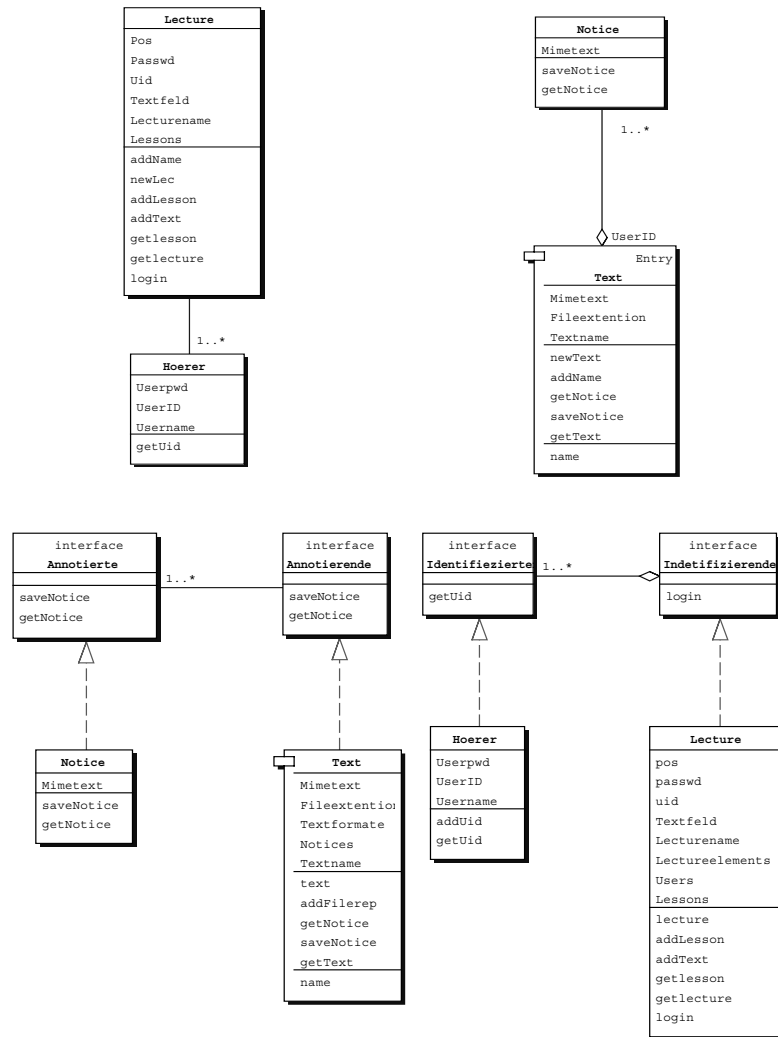


Abbildung 28: Gegenüberstellung der Packages Notizen von Abb. 7 und Abb. 22

Die drei bisher betrachteten Packages bringen die Vorteile eines rollenbasierten Entwurfes nicht besonders deutlich hervor, da die Instanzen der Klassen in der Regel jeweils nur eine Rolle für den Zugriff auf andere Klassen einnehmen und in ihr meistens alle Methoden der Klasse aufgeführt werden. So verhält es sich beispielsweise mit den Klassen *Hoerer* und *Notice*. Ähnliche Strukturen weisen die Packages *Textbase* und *Vorlesung* auf: Die Klassen *Textbase*, *Corpus* und *Lesson* spielen maximal zwei Rollen (*Corpus*, *Lesson*). Durch die Rol-

len der Klassen lässt sich dadurch auch kein gravierender Vorteil gegenüber der konventionellen Modellierung feststellen. Es wird lediglich die Richtung der Zugriffe (z.B. *Textbase-Corpus-Text*) besser dargestellt und dadurch die Lesbarkeit des Entwurfes verbessert.

In den beiden Packages Indexsuche und kategorische Suche werden die Vorteile des rollenbasierten Entwurfes wesentlich deutlicher.

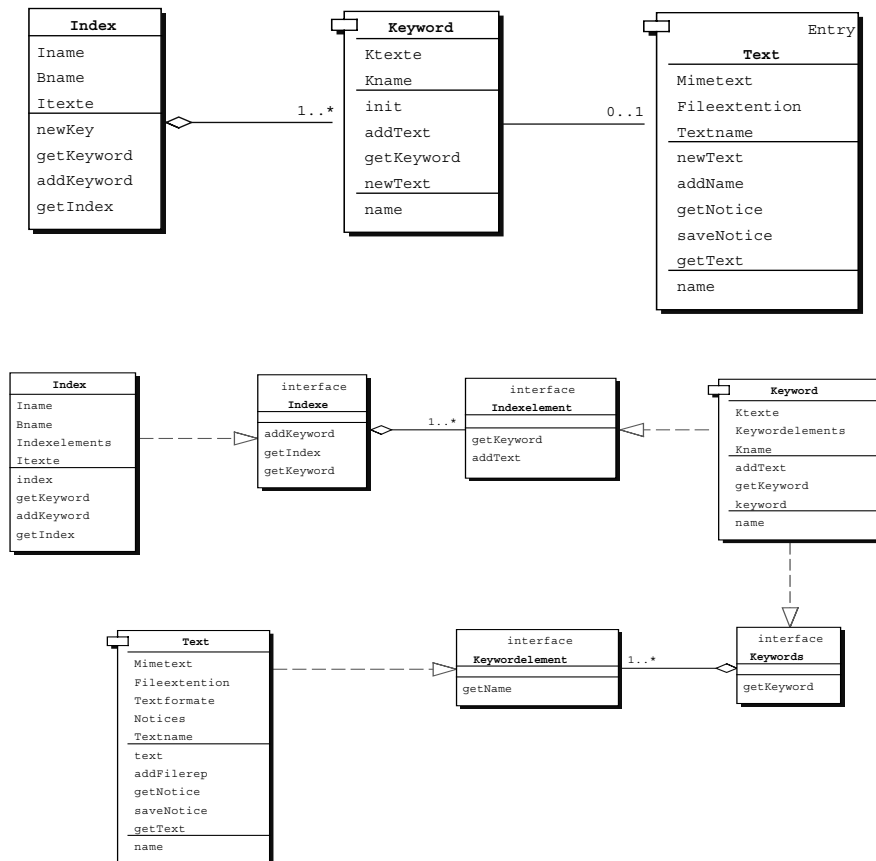


Abbildung 29: Gegenüberstellung der Packages Indexsearch von Abb. 8 und Abb. 23

Im Package Indexsearch sind die Klassen *Index* und *Keyword* sowie die Klassen *Keyword* und *Text* durch jeweils eine Relation miteinander verbunden (vgl. Abbildung 8). Im rollenbasierten Design stehen an den Enden der Relationen die Rollen *Indexe* und *Indezelement* sowie *Keywords* und *Keywordelement* (vgl. Abbildung 23). Da die Rolle *Indexe* alle Methoden der Klasse *Index* und die Rolle *Indezelement* alle Methoden der Klasse *Keyword* deklariert und auch weiterhin nur eine Beziehung zwischen den Klassen über die beiden Rollen besteht, gewinnt das Klassendiagramm durch den rollenbasierten Entwurf nur unwesentlich an Lesbarkeit. Bei der Relation zwischen den Klassen *Keyword* und *Text* entsteht die gleiche Situation wie in dem Package Vorlesung. Durch das Einführen der Rollen *Keywords* und *Keywordele-*

ment wird im rollenbasierten Entwurf deutlich, dass die Methoden der Klasse *Keyword* (und hier eigentlich nur die Methode *getKeyword*, wie aus der Rolle *Keywords* ersichtlich wird) nur auf die Methode *getName* der Klasse *Text* zugreifen können, da dies die einzige Methode ist, die in der Rolle *Keywordelement* deklariert ist. Diese Begrenzung auf eine Methode beim Zugriff auf die Klasse geht aus dem konventionellen Klassendiagramm nicht hervor. Es kommt dort erst im Sequenz- oder Kollaborationsdiagramm zum Vorschein.

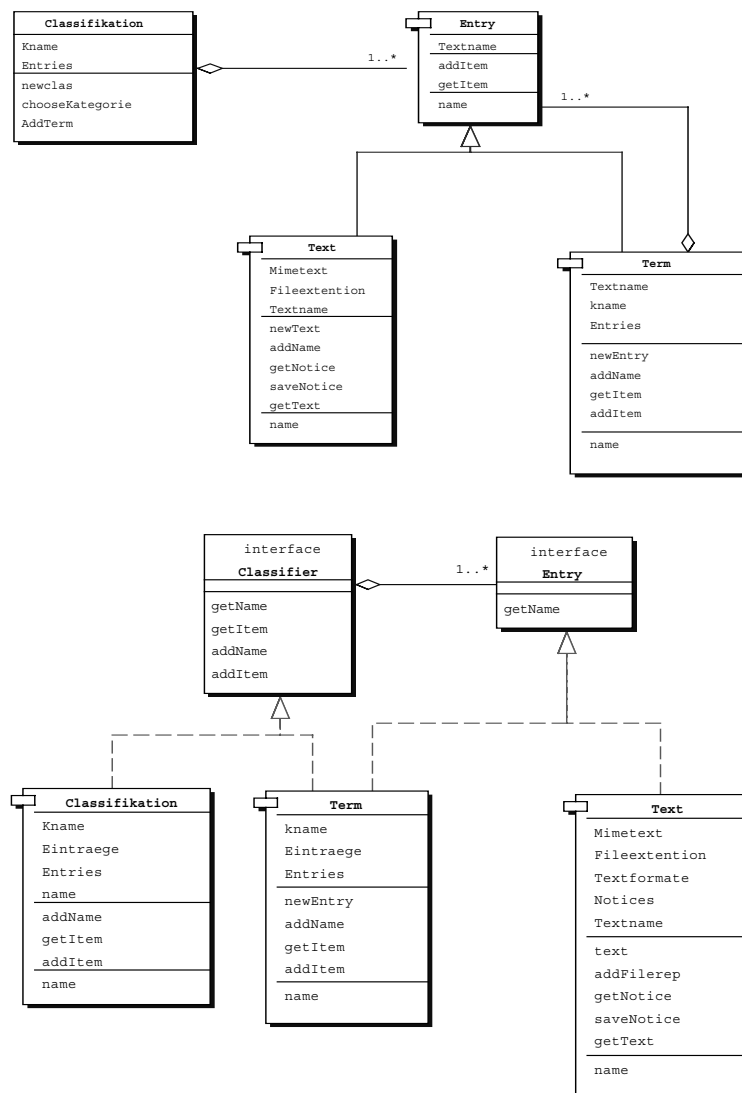


Abbildung 30: Gegenüberstellung der Packages Classifikation von Abb. 9 und Abb. 24

Beim Vergleich der Entwürfe bei der kategorischen Suche zeigen die beiden Klassendiagramme doch einige Unterschiede auf, die für den rollenbasierten Entwurf sprechen. Dabei sticht besonders hervor, dass die abstrakte Klasse aufgelöst und ihre Funktion von der Rolle *Entry* übernommen wurde. Diese Tatsache allein reicht sicherlich nicht aus, um den rollenba-

sierten Entwurf zu favorisieren. Bei einer näheren Betrachtung der Klassen *Term* und *Text* im klassischen Entwurf stellt sich jedoch heraus, dass die Klassen eigentlich von ihrer Art her gar keine Einträge sind, sondern eher von der Vererbung ihrer Oberklasse *Entry* leben. Auch wird die Beziehung der Terme zu ihren Untereinträgen nicht besonders präzise dargestellt, da in dem Package *Classification* des konventionellen Entwurfes (Abbildung 9) die abstrakte Klasse *Entry* im Mittelpunkt steht. Im rollenbasierten Entwurf stehen dagegen die Rollen mit ihrer Relation im Vordergrund, so dass die Beziehung zwischen den Termen und ihren Untereinträgen aus den Klassendiagrammen wesentlich deutlicher hervorgeht. Durch die Rolle *Entry* wird ebenfalls sichergestellt, dass die Klassen *Term* und *Text*, wenn sie diese Rolle spielen, auch wirklich Einträge darstellen. Dieses ist aus der abstrakten Klasse *Entry* nicht eindeutig zu ersehen.

Wenn man die beiden Packages *Indexsearch* und *Classification* zusammen betrachtet, stellt man einen weiteren Unterschied zwischen den beiden Entwurfstilen fest, denn durch das Einführen von Rollen lassen sich einige Vereinfachungen durchführen: Betrachtet man die Eigenschaften von den Klassen *Term* und *Keyword*, so stellt sich heraus, dass die Klasse *Term* auch die Eigenschaften der Klasse *Keyword* übernehmen kann. Ein Term hat genau dann die Eigenschaften eines Keywords, wenn es die letzte Unterkategorie ist und nur noch Texte als Einträge besitzt. Ein Keyword hat ebenso wie der Term eine Reihe von Texten als Untereinträge. Durch Hinzufügen einer weiteren Assoziation zwischen *Term* und *Text* lässt sich die Klasse *Keyword* durch die Klasse *Term* ersetzen. Um dann auf die Schlagwörter zugreifen zu können, wird die Rolle *Indexelements* nicht mehr von der Klasse *Keyword* sondern von der Klasse *Term* gefüllt. Somit können durch den rollenbasierten Entwurf Klassen, deren Instanzen die gleichen Funktionen in unterschiedlichen Situationen ausüben, zusammengelegt werden. Die Unterscheidung zwischen den einzelnen Einsatzgebieten wird durch das Übernehmen der dafür zuständigen Rolle durch die Instanzen vorgenommen. So müssen Methoden, die in beiden Klassen vorkommen und die gleiche Funktion erfüllen, nur einmal implementiert werden. Dieses birgt noch einen weiteren Vorteil in der rollenbasierten Modellierung derart, dass durch das Zusammenlegen der Klassen *Term* und *Keyword* die Suche zu einem übersichtlichen Paket zusammengefasst werden kann. Auch weitere Erweiterungen wie das Hinzufügen eines Glossars lassen sich dadurch leicht verwirklichen. Dazu müsste keine neue Klasse implementiert, sondern lediglich neue Rollen für die benötigten Klassen definiert werden. Die Einführung der Rollen führt hier neben der detaillierteren Abbildung und damit besseren Lesbarkeit zu einer deutlichen Arbeitserleichterung.

Insgesamt kommt der Vorteil der besseren Lesbarkeit und damit besseren Veranschaulichung eines Entwurfes durch die Rollen besonders dann zum Vorschein, wenn eine Klasse mehrere Rollen spielt. Im Hyperbook geschieht dies bei den Klassen *Text* und *Lecture*. Bei einer Betrachtung des kompletten Entwurfes stellt man fest, dass die Klasse *Text* an insgesamt sieben Relationen beteiligt ist und somit auch sieben verschiedene Rollen spielen kann. Allein in den drei Packages *Textbase*, *Vorlesung* und *Notizen* spielt die Klasse *Text* vier verschiedene Rollen. So können beispielsweise bei dem Aufruf der Klasse *Text* durch eine Instanz von *Lesson* nicht gleichzeitig die Notizen mit aufgerufen werden, da innerhalb der Rolle *Lessonelement* die Methode *getNotice* nicht deklariert ist und somit auch der Instanz der Klasse *Lesson* für einen Aufruf nicht zur Verfügung steht. Diese Einschränkungen, die durch die Rollen für die Klasse hervorgerufen werden, werden im konventionellen Klassendiagramm nicht deutlich.

Noch besser kommt der Vorteil der besseren Lesbarkeit im folgenden Beispiel hervor: Eine Person kann mit anderen Personen in unterschiedlichen Beziehungen stehen. So kann ein Frau Mutter einer Tochter oder eines Sohnes sein oder ein Mann der dazugehörige Vater. Ebenso kann dieser Mann der Ehemann seiner Ehefrau und Onkel für seine Nichte sein. Dies sind nur einige wenige Beispiele für die unterschiedlichen Beziehungen zwischen zwei Personen. Im konventionellen Kassendiagramm werden diese Relationen, wie in Abbildung 31 zu sehen, durch eine einfache Assoziation dargestellt.

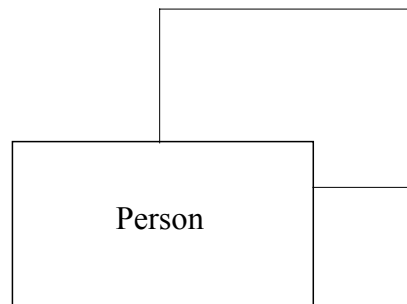


Abbildung 31: Beziehung zwischen Personen im konventionellen Entwurf

In der rollenbasierten Modellierung sind dagegen die einzelnen Beziehungen zwischen den Personen durch die Rollen eindeutig beschrieben (Abbildung 32). So kann eine männliche Person Vater seines Sohnes sein, wie auch eine Frau Mutter ihrer Tochter sein kann. Ebenso ist ein Mann nur dann mit einer Frau verheiratet, wenn er der Ehemann von ihr ist und sie somit seine Ehefrau. Diese Einschränkungen sind im konventionellen Entwurf nicht zu erken-

nen, sie kommen erst im Sequenz- oder Kollaborationsdiagramm hervor. In der rollenbasierten Modellierung werden die Rahmenbedingungen, die eine Person erfüllen muss um eine Rolle einer Beziehung einnehmen zu können, schon im Klassendiagramm deutlich, so dass ein weiteres Kollaborationsdiagramm zur Erklärung nicht nötig ist.

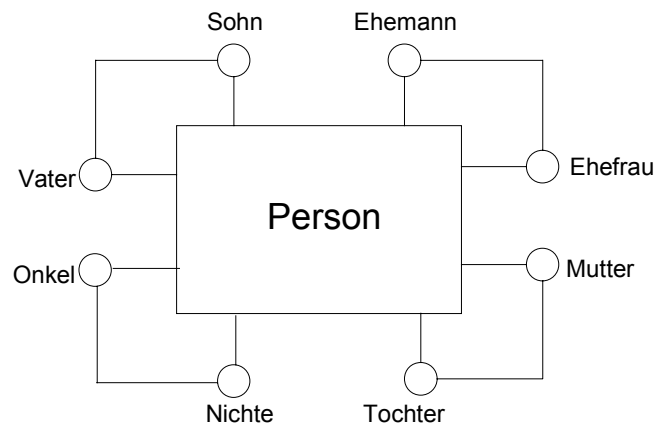


Abbildung 32: Beziehung zwischen Personen im rollenbasierten Entwurf

Bei den Kollaborations- und Sequenzdiagrammen auf Instanzebene treten durch die neue Modellierung keine grundlegenden Änderungen auf, da Instanzen nur jeweils eine Rolle während eines Methodenaufrufs einnehmen und sich der Methodenaufruf durch die Einführung der Rollen nicht geändert hat. Allerdings wird deutlich, dass viele Informationen aus den Kollaborations- bzw. Sequenzdiagrammen durch den neuen Entwurf bereits im Klassendiagramm enthalten sind. Somit kann die Erstellung eines zusätzlichen Kollaborationsdiagramms u.U. entfallen. Dies würde zu einer entsprechenden Arbeitserleichterung in der Modellierung führen.

6.2. Beurteilung des rollenbasierten Entwurfes

Der rollenbasierte Entwurf weist gegenüber dem konventionellen Entwurf einige Unterschiede auf: Dabei sticht zunächst ins Auge, dass am Ende einer Assoziation eine Rolle als Interface steht. So wird einerseits die Funktion der Assoziationen zwischen den Klassen im Klassendiagramm wesentlich besser hervorgehoben, die sonst erst in den Kollaborations- oder Sequenzdiagrammen deutlich wurde. Andererseits bedeutet die Einführung der Rollen aber auch einen Mehraufwand beim Design. Dieser muss durch die Vorteile des rollenbasierten

Entwurfes kompensiert werden, damit sich die Einführung des Rollenkonzeptes rechtfertigt. Dabei muss hier ein weiterer Aspekt berücksichtigt werden, der durch das Einführen von Rollen auftritt: das einfache Hinzufügen von ähnlichen Komponenten. Während im konventionellen Entwurf beim Hinzufügen von neuen Komponenten, die eine ähnliche Struktur wie bereits implementierte Komponenten aufweisen, neue (Sub-)Klassen implementiert werden müssen, reicht es bei dem rollenbasierten Entwurf teilweise aus, lediglich neue zusätzliche Rollen für die vorhandenen Klassen zu deklarieren. Dadurch lässt sich viel Implementierungsarbeit sparen, da die Klassen bereits vorhanden sind und ihre Methoden mitbenutzt werden können. Durch das Benutzen der Rollen werden auch unerwünschte Nebeneffekte vermieden, wie sie beim Einführen von Subklassen durch die Vererbung auftreten können.

Betrachtet man nun die beiden Entwürfe des Hyperbooks, so sind doch einige Vorteile des rollenbasierten Entwurfes erkennbar: In den Packages Textbase, Vorlesung, Notizen sowie Indexsearch liegt eine relativ einfache Struktur des Klassendiagramms vor, so dass durch die Einführung der Rollen keine großen Änderungen ersichtlich werden. Aber selbst in diesen Diagrammen lässt sich schon eine Verbesserung der Lesbarkeit feststellen. Dies gilt insbesondere bei den Rollen der Klasse *Text*, da hier die Einschränkungen, die durch das Einnehmen einer Rolle durch eine Instanz der Klasse *Text* für diese hervorgerufen werden, in dem konventionellen Klassendiagramm nicht zu erkennen sind. Bei dem konventionellen Klassendiagramm kann aus den Relationen zwischen den einzelnen Klassen nicht nachvollzogen werden, ob es unterschiedliche Beziehungen zwischen den einzelnen Instanzen der Klassen gibt.

Das Package Classification wird durch das Einführen der Rollen wesentlich übersichtlicher und damit auch lesbarer. Während bei dem Entwurf ohne Rollen die Beziehung zwischen den Termen und ihren Untereinträgen nicht deutlich hervorkommt, steht gerade diese beim rollenbasierten Entwurf im Vordergrund. Berücksichtigt man darüber hinaus auch die Möglichkeit der einfachen Zusammenlegung der Packages Indexsearch und Classification, so ist der Vorteil des rollenbasierten Entwurfes gegenüber dem konventionellen Entwurf doch so erheblich, dass der Aspekt des Mehraufwandes in den Hintergrund rückt.

Besonders gut kommen die Vorteile des rollenbasierten Entwurf hervor, wenn eine Instanz einer Klasse gegenüber einer anderen Klasse mehrere unterschiedliche Rollen spielt. Hier werden die einzelnen Einschränkungen durch die einzelnen Rollen der Klassen am Anfang der Relation hervorgehoben, da nur die Instanz die Rolle einnehmen kann, die sie auch füllen kann. Dies bedeutet, dass sie alle Methoden, die durch die Rolle deklariert sind, ausführen

kann. Dieser Aspekt tritt im Entwurf des Hyperbooks nicht auf. Betrachtet man jedoch das Beispiel in Abbildung 32, so wird schnell deutlich, dass hier der rollenbasierte Entwurf einen enormen Vorteil gegenüber dem konventionellen aufweist. Hier sind die unterschiedlichen Zugriffe und die damit verbundenen Einschränkungen der Klassen untereinander schon im Klassendiagramm deutlich erkennbar, während im konventionellen Entwurf dieses erst in den Kollaborations- bzw. den einzelnen Sequenzdiagrammen deutlich wird.

Insgesamt betrachtet treten die Vorteile des rollenbasierten Entwurfes immer deutlicher hervor, je komplexer die Beziehungen zwischen den Klassen sind. Dort gewinnt der Vorteil der besseren Lesbarkeit sehr stark an Bedeutung und der gegebene Mehraufwand wird durch die Einsparungen der zusätzlichen Diagramme zum besseren Verständnis mehr als kompensiert. Im Falle des Hyperbooks fällt der Vorteil der besseren Lesbarkeit nicht in allen Bereichen stark ins Gewicht, da hier teilweise keine komplexeren Beziehungen zwischen den Klassen vorliegen. Aber insbesondere im Package Classification erweist sich die Einführung der Rollen in bezug auf die Übersichtlichkeit als enormer Vorteil. Gleichzeitig lässt sich hier ein Zusammenlegen der Packages Classification und Indexsearch realisieren, was allein aus strukturellen Gründen zu begrüßen ist. Dies führt zu einer entsprechenden Arbeitserleichterung, so dass sich insgesamt durch die rollenbasierte Modellierung kein erheblicher Mehraufwand ergibt. Damit überwiegen selbst bei einem Entwurf mit relativ geringer Komplexität wie dem Hyperbook insgesamt gesehen die Vorteile, so dass der Einsatz eines rollenbasierten Entwurfes hier empfohlen wird.

7. Zusammenfassung und Fazit

Rollen nehmen eine immer größere Bedeutung in der Softwareentwicklung ein. In der Modellierungssprache UML werden sie bisher jedoch mehr oder weniger vernachlässigt. Um die Rollen aus ihrem Schattendasein herauszuholen, wurden Änderungen am aktuellen Metamodell von UML vorgeschlagen, die die Rollen mehr in den Vordergrund rücken. Die Änderung des Metamodells bedingt eine geänderte Anwendung der bestehenden UML-Notation. Die Notation selbst wird dabei nicht abgewandelt, so dass keine Anpassung der Modellierungswerkzeuge nötig wird.

Um die Unterschiede des rollenbasierten Entwurfes gegenüber dem Entwurf nach dem aktuellen UML-Standard aufzeigen zu können, wurden im Rahmen dieser Arbeit die beiden Entwürfe am Beispiel eines benutzeradaptierbaren Hyperbooks durchgeführt.

Bei der Beschreibung des Entwurfes des Hyperbooks nach dem aktuellen Metamodell wurden gleichzeitig die Funktionsweise und deren Anwendungsfälle des Hyperbooks beschrieben. Die nähere Ausführung erfolgte über eine Darstellung der einzelnen Anwendungsfälle in den Use Case- und Sequenzdiagrammen.

Beim rollenbasierten Entwurf stehen in den Klassendiagrammen an den Enden einer Assoziation jeweils Rollen, die als Interfaces implementiert wurden. Hierbei werden schon im Klassendiagramm die Einschränkungen der Assoziationen deutlich, die im konventionellen Entwurf erst im Kollaborationsdiagramm hervorgehoben wurden. Dadurch erspart man sich das Modellieren von zusätzlichen Kollaborationsdiagrammen im rollenbasierten Entwurf.

Die durch das Einführen der Rollen entstandenen Änderungen wurden im erneuten Entwurf des Hyperbooks deutlich.

Beim Vergleich der beiden Entwürfe fällt als erstes der Mehraufwand auf, der durch das zusätzliche Einfügen der Rollen entsteht. Bei einer näheren Betrachtung der beiden Entwürfe wird aber schnell deutlich, dass die Rollen in gewissen Bereichen des Entwurfes Vorteile in der Lesbarkeit bergen und dadurch zusätzliche Diagramme zur Erklärung überflüssig machen. Der entstandene Mehraufwand wird somit relativiert. Besonders prägnant tritt der Vorteil hervor, wenn Klassen über mehrere Rollen miteinander kommunizieren. Dieses ist im Hyperbook für die Klasse Text der Fall, da sie über mehrere Rollen mit anderen Klassen in Verbindung steht. Da der Umstand der besseren Lesbarkeit jedoch nicht überall in der Modellierung des Hyperbooks stark ins Gewicht fällt, wurde der Vorteil der Rollen an einem kleinen Beispiel nochmals hervorgehoben.

Ein weiterer Aspekt, der für den rollenbasierten Entwurf spricht, kommt beim Vergleich der kategorischen Suche hervor: So wird durch das Einführen der Rollen das Klassendiagramm nicht nur wesentlich deutlicher, sondern es lässt sich weiterhin feststellen, dass durch die Rollen u.U. das Implementieren von neuen Klassen reduziert werden kann. Diese Möglichkeit besteht dann, wenn beim Hinzufügen von neuen Komponenten Klassen vorhanden sind, die von der Funktion her einer bereits implementierten Klasse ähneln. In diesem Fall kann auf das Einfügen von neuen Klassen verzichtet werden, indem man den vorhandenen Klassen neue Rollen zufügt, die die Aufgaben übernehmen. So ist im rollenbasierten Entwurf des Hyperbooks festzustellen, dass sich die Indexsuche und die kategorische Suche im Modell zusammenlegen lassen. Es müssen dazu nur neue Rollen eingeführt werden, während die Funktionen der Klasse Keyword von der Klasse Term übernommen werden. Ebenso lässt sich ein zusätzliches Glossar ohne das Einfügen von neuen Klassen in das Hyperbook einfügen.

Insgesamt betrachtet, stellt der rollenbasierte Entwurf zwar einen gewissen Mehraufwand in der Modellierung dar. Dieser wird jedoch durch die bessere Lesbarkeit der Anwendungssituationen in den Klassendiagrammen und den Einsparungen bei der Erweiterung bzw. Zusammenlegung von Packages wieder aufgehoben. Je komplexer das Design wird, desto stärker kommt der Vorteil der besseren Lesbarkeit des Klassendiagramms hervor und spart dadurch Arbeit, die in weitere Erklärungen in Form von weiteren Diagrammen gesteckt werden muss.

Für eine allgemeingültige Bewertung wäre es empfehlenswert, ein größeres Projekt konsequent im rollenbasierten Design umzusetzen und an den dort entstehenden Unterschieden gegenüber dem konventionellen Design eine Beurteilung der Vorteile gegenüber dem Mehraufwand vorzunehmen. Hierbei ist zu erwarten, dass die Vorteile des rollenbasierten Entwurfes gegenüber dem konventionellen wesentlich deutlicher hervorkommen, als sie schon beim Entwurf des Hyperbooks zu sehen sind.

Vorab wäre eine Anpassung eines Modellierungswerkzeuges wie z.B. Together J an das neue Rollenkonzept wünschenswert, so dass der Einsatz der Rollen verlangt wird. Dadurch würden die entstehenden Unterschiede für die Modellierung und Benutzung in der Praxis besser deutlich und könnten zur allgemeingültigen Bewertung mit herangezogen werden.

8. Literaturverzeichnis

- [Booch] G. Booch, I. Jacobson, J. Rumbaugh, *UML Distilled: Applying the Standard Object Modeling Language* (Addison-Wesley 1997)
- [Gamma] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley 1995)
- [Neumann] H. A. Neumann, *Objektorientierte Softwareentwicklung mit der Unified Modeling Language (UML)* (Carl Hanser Verlag 1998)
- [OMG 1999] OMG, *Unified Modelling Language Specification Version 1.3* (<http://www.omg.org>)
- [Reenskaug] T. Reenskaug, P. Wold, O. A. Lehne, *Working with Objects-The OOram Software Engineering Method* (Manning Greenwich 1996)
- [J. Rumbaugh] J. Rumbaugh, I. Jacobson, G. Booch, *The Unified Modeling Language Reference Manual* (Addison-Wesley 1999)
- [Schmuller] J. Schmuller, *Jetzt lerne ich UML* (Markt + Technik Verlag 2000)
- [Schneider] G. Schneider, J.P. Winters, *Applying Use Cases – A practical guide* (Addison-Wesley 1998)
- [Steimann, Habilitationsschrift] F. Steimann, *Formale Modellierung mit Rollen*, Habilitationsschrift(2000)
- [Steimann, Rollen] F. Steimann, *Rollen – Jetzt oder nie*, interner Bericht
- [Together] Togethersoft LLC, *User Manual Set Version 3.1* (<http://www.togethersoft.com>)