Praktikum Software-Engineering

Systementwurf: Stilanalyse von Texten

Jan Tammen (jan.tammen@fh-konstanz.de)
Christoph Eck (christoph.eck@fh-konstanz.de)

17. November 2005

Inhaltsverzeichnis

1 Festlegung der Einflussfaktoren auf Softwarearchitektur

1.1 Einsatzbedingungen

Definiert im Pflichtenheft.

1.2 Umgebungs- und Randbedingungen

Definiert im Pflichtenheft.

1.3 Nichtfunktionale und Qualitätsanforderungen

1.3.1 Nichtfunktionale Anforderungen

Es werden keine besonderen Anforderungen bezüglich Internationalisierung, Skalierbarkeit und Wiederverwendbarkeit gestellt.

1.3.2 Qualitätsanforderungen

Es werden keine besonderen Anforderungen bezüglich Zuverlässigkeit, Änderbarkeit und Effizienz gestellt.

2 Entwurf der Benutzungsschnittstelle

Detaillierte Entwürfe der grafischen Benutzungsschnittstelle sind im Pflichtenheft enthalten. Das folgende Diagramm zeigt den schematischen Aufbau der Oberfläche.

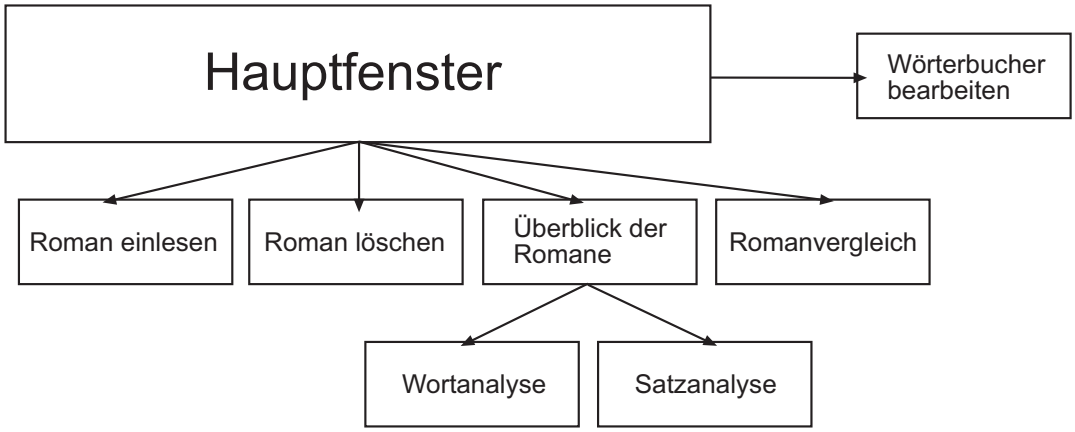

Abbildung 1: Benutzungsschnittstelle

3 Grobarchitektur

Das folgende Diagramm stellt die im System enthaltenen Komponenten und deren Abhängigkeiten untereinander dar.

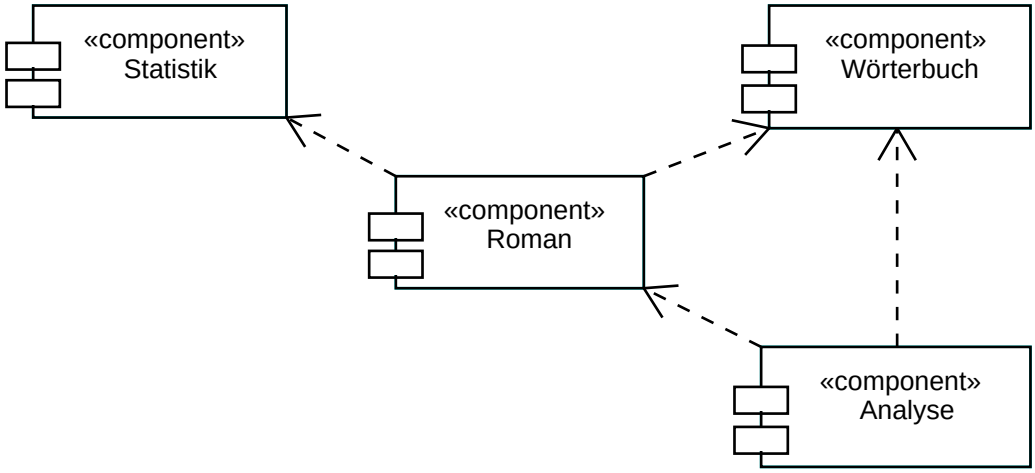

Abbildung 2: Grobarchitektur: Komponenten

3.1 Beschreibung der Komponenten

3.1.1 Komponente Roman

Die Komponente Roman speichert den Text eines Romans ab und der Text kann verwaltet werden. Die Komponente beinhaltet folgende Module:

- RomanVerwaltung
- Roman
- Text
- Textdatei
- Satz
- Wort

Die Komponente Roman benutzt die Komponente Statistik und Wörterbuch. Die Klasse für die Benutzeroberfläche wird in der Komponente nicht aufgeführt.

3.1.2 Komponente Analyse

Die Komponente Analyse analysiert den eingelesenen Roman, teilt die Analyse in Wort- und Satzanalyse auf und kann verschiedene Romane miteinander vergleichen. Die Komponente beinhaltet folgende Module:

- Wortanalyse
- Satzanalyse
- Vergleichsanalyse
- Analyse

Die Komponente Analyse benutzt die Komponente Roman und Wörterbuch. Die Klasse für die Benutzeroberfläche wird in der Komponente nicht aufgeführt.

3.1.3 Komponente Wörterbuch

Die Komponente Wörterbuch verwaltet die Wörterbücher. Die Komponente beinhaltet folgende Module:

- WörterbuchVerwaltung
- Wörterbuch

Die Komponente Wörterbuch benutzt keine Komponente. Die Klasse für die Benutzeroberfläche wird in der Komponente nicht aufgeführt.

3.1.4 Komponente Statistik

Die Komponente Statistik ist für die Verwaltung der bei der Textanalyse anfallenden statistischen Daten zuständig. Die Komponente beinhaltet folgende Module:

- Wortstatistik
- Satzstatistik

Die Komponente Statistik benutzt keine Komponente. Die Klasse für die Benutzeroberfläche wird in der Komponente nicht aufgeführt.

4 Feinarchitektur

Die folgenden Abschnitte beinhalten die detaillierten Diagramme der einzelnen Komponenten sowie die Beschreibungen der in ihnen enthaltenen Module (Klassen).

4.1 Komponente Roman

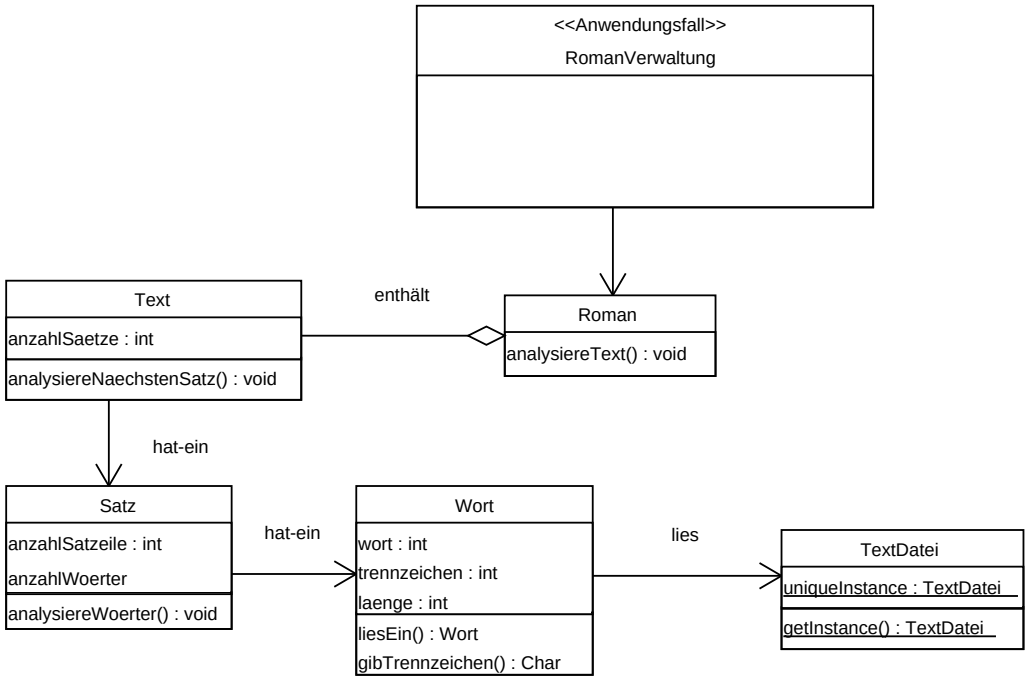

Abbildung 3: Komponente Roman

4.1.1 Klasse RomanVerwaltung

Diese Klasse dient als Schnittstelle zwischen der Benutzeroberfläche und der Klasse `Roman`. Über sie werden die Angaben zu einem Roman wie Autorname, Erscheinungsdatum usw. manipuliert. Sie enthält die folgenden Operationen:

Operation	**Aufgabe**	**Parameter**
`setzeAutor`	Setzt den Autornamen des Romans	*in*: Roman (Typ: `Roman`), Autor-Namen (Typ: `String`)
`setzeTitel`	Setzt den Titel des Romans	*in*: Roman (Typ: `Roman`), Titel (Typ: `String`)
`setzeDatum`	Setzt das Erscheinungsdatum des Romans	*in*: Roman (Typ: `Roman`), Erscheinungsdatum (Typ: `Date`)
`setzeAbsatzmenge`	Setzt die Absatzmenge des Romans	*in*: Roman (Typ: `Roman`), Absatzmenge (Typ: `Integer`)

Tabelle 1: Operationen der Klasse RomanVerwaltung

4.1.2 Klasse Roman

Diese Klasse repräsentiert einen Roman (Anwendungsweltobjekt) und enthält Informationen über diesen. Die Klasse wird benutzt von der Klasse `Analyse` und benutzt selbst die Klasse `Text` („hat-ein“-Beziehung).

Operation	Aufgabe	Parameter
`setzeAutor`	Setzt den Autornamen des Romans	*in*: Autor-Namen (Typ: `String`)
`setzeTitel`	Setzt den Titel des Romans	*in*: Titel (Typ: `String`)
`setzeDatum`	Setzt das Erscheinungsdatum des Romans	*in*: Erscheinungsdatum (Typ: `Date`)
`setzeAbsatzmenge`	Setzt die Absatzmenge des Romans	*in*: Absatzmenge (Typ: `Integer`)

Tabelle 2: Operationen der Klasse Roman

4.1.3 Klasse Text

Diese Klasse nimmt den eigentlichen Text eines Romans auf. Ein Text wird dabei als Menge von Sätzen angesehen; deshalb enthält die Klasse `Text` einen Verweis auf einen speziellen mengenbasierten Datencontainer, in welchem die einzelnen Sätze effizient abgelegt werden können. Die Klasse wird benutzt von der Klasse `Roman` und benutzt selbst die Klasse `Satz`.

Operation	Aufgabe	Parameter
`analysiereNaechstenSatz`	Wird innerhalb der Klasse `Roman` aufgerufen und untersucht den nächsten Satz des Quelltextes	keine

Tabelle 3: Operationen der Klasse Text

4.1.4 Klasse TextDatei

Diese Klasse bietet einen Zugriff auf die Quelldatei des zu analysierenden Romans. Damit es in der Anwendung stets nur einen gleichzeitigen Zugriff auf eine Datei gibt, wird diese Klasse als Singleton implementiert. Die Klasse wird benutzt von der Klassen `Wort` und `Analyse`.

Operation	Aufgabe	Parameter
`getInstance`	Wird z.B. innerhalb der Klasse `Wort` aufgerufen, um anschließend von der Datei lesen zu können	*out*: Instanz des Objekts

Tabelle 4: Operationen der Klasse TextDatei

4.1.5 Klasse Satz

Diese Klasse ist für die Speicherung von Sätzen zuständig. Ein Satz wird dabei als Menge von Wörtern angesehen; deshalb enthält die Klasse `Satz` einen Verweis auf einen speziellen mengenbasierten Datencontainer (HashSet), in welchem die einzelnen Wörter effizient abgelegt werden können. Weiterhin enthält sie Informationen über die Anzahl der Wörter sowie Anzahl der Satzteile innerhalb des Satzes. Die Klasse wird benutzt von der Klasse `Text` und benutzt selbst die Klassen `Wort` und `Satzstatistik`.

Operation	Aufgabe	Parameter
`analysiereWoerter`	Wird innerhalb der Klasse `Text` aufgerufen und extrahiert solange Wörter aus dem Quelltext, bis das Satzende erreicht ist	keine

Tabelle 5: Operationen der Klasse TextSatz

4.1.6 Klasse Wort

Diese Klasse bildet die kleinste im System bekannte Einheit eines Textes ab – das Wort. Ein Wort besteht aus einer Menge von Zeichen, welche in der Klasse in einem `String`-Datencontainer (HashSet) gespeichert werden. Ebenso wird dasjenige Zeichen (Typ: `Char`) gespeichert, mit welchem das Wort beendet wurde, also z.B. ein Leerzeichen oder Komma. Die Klasse wird benutzt von der Klasse `Satz` und benutzt selbst die Klassen `TextDatei`, `Wörterbuch` und `Wortstatistik`.

Operation	Aufgabe	Parameter
`liesEin`	Solange zeichenweise aus der Quelldatei lesen, bis definiertes Trennzeichen erreicht ist	*out*: Referenz auf das eigene Objekt
`gibTrennzeichen`	Anhand des Trennzeichens kann die Klasse `Satz` erkennen, ob es sich um ein Satzende oder z.B. den Anfang eines Satzteils handelt	*out*: Trennzeichen (Typ: `Char`)

Tabelle 6: Operationen der Klasse Wort

4.2 Komponente Analyse

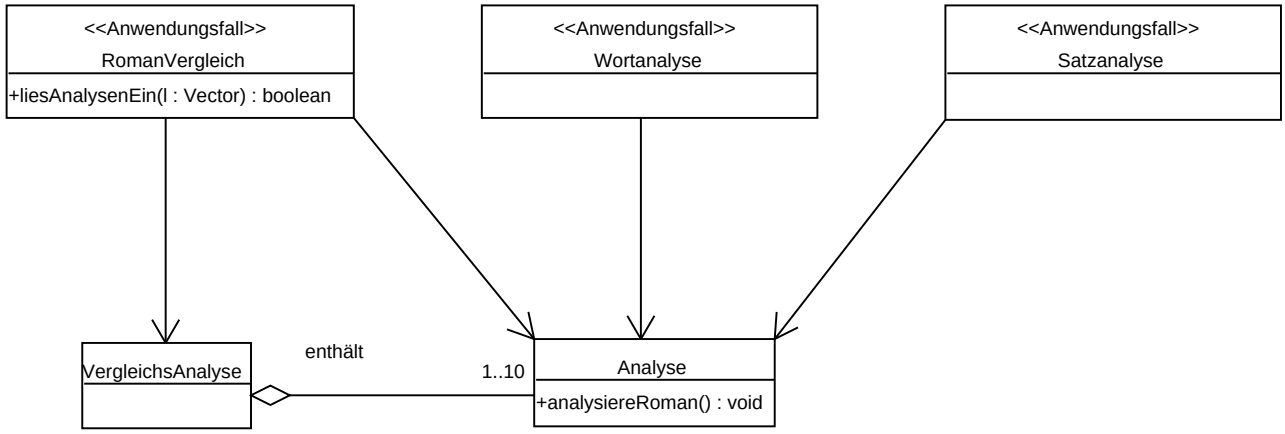

Abbildung 4: Komponente Analyse

4.2.1 Klasse RomanVergleich

Diese Klasse dient als Schnittstelle zwischen der Benutzeroberfläche und der Klasse `Vergleichsanalyse`. Sie dient dazu, mehrere ausgewählte Roman-Analysen vergleichen zu können. Die Klasse benutzt die Klassen `VergleichsAnalyse` und `Analyse`.

Operation	Aufgabe	Parameter
`liesAnalysenEin`	Liest für jeden der gewählten Romane die Analyse ein	*in*: Romane (Typ: `Vector`)

Tabelle 7: Operationen der Klasse RomanVergleich

4.2.2 Klasse Wortanalyse

Diese Klasse dient als Schnittstelle zwischen der Benutzeroberfläche und der Klasse `Analyse`. Sie dient dazu, die Wortanalyse eines Romans aufzurufen. Die Klasse benutzt die Klasse `Analyse`.

4.2.3 Klasse Satzanalyse

Diese Klasse dient als Schnittstelle zwischen der Benutzeroberfläche und der Klasse `Analyse`. Sie dient dazu, die Satzanalyse eines Romans aufzurufen. Die Klasse benutzt die Klasse `Analyse`.

4.2.4 Klasse Vergleichsanalyse

Diese Klasse dient dazu, mehrere Analysen verschiedener Romane zu vergleichen. Die Klasse benutzt daher die Klasse `Analyse`.

4.2.5 Klasse Analyse

Diese Klasse verwaltet die für die Textanalyse benötigten Elemente: den Roman sowie die Wort- und Satzstatistik. Die Klasse benutzt daher die Klassen `Roman`, `Wortstatistik` und `Satzstatistik` und wird benutzt von den Klassen `VergleichsAnalyse`, `RomanVergleich`, `Wortanalyse`, `Satzanalye` und `RomanVerwaltung`.

Operation	Aufgabe	Parameter
`analysiereRoman`	Stößt die Erstellung der Text-Analyse des Romans an	keine

Tabelle 8: Operationen der Klasse Analyse

4.3 Komponente Wörterbuch

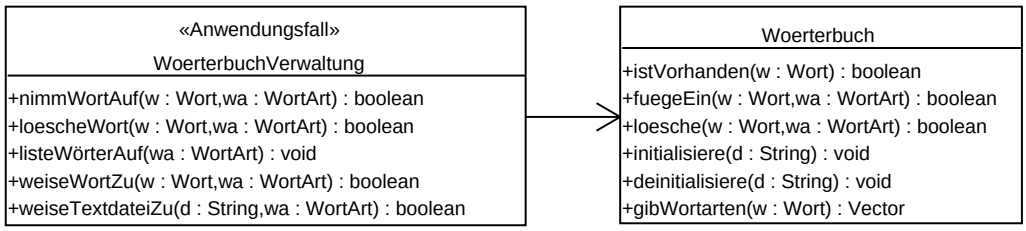

Abbildung 5: Komponente Wörterbuch

4.3.1 Klasse WoerterbuchVerwaltung

Diese Klasse verwaltet alle vorhanden Wortarten. Alle bei der Analyse nicht zugeordneten Wörter gehören der Wortart „Sonstige Wörter“ an. Mit einer Textdatei mit Wörtern gleicher Wortart können die neuen Wörter in eine bestehende oder neue Wortart eingelesen werden.

Operation	**Aufgabe**	**Parameter**
`nimmWortAuf`	Wörter, die noch nicht spezifiziert sind, werden in die Wortart „Sonstige Wörter“ gespeichert	*in*: Wort (Typ: `Wort`), Wortart (Typ: `Wortart`)
`loescheWort`	Wort löschen, in jeder Wortart	*in*: Wort (Typ: `Wort`), Wortart (Typ: `Wortart`)
`listeWoerterAuf`	listet alle Wörter der Wortart auf	*in*: Wortart (Typ: `Wortart`)
`weiseWortZu`	Das Wort wird der Wortart zugeordnet	*in*: Wort (Typ: `Wort`), Wortart (Typ: `Wortart`)
`weiseTextdateiZu`	Durch Öffnen der gewünschten Textdatei werden die Wörter einer Wortart zugewiesen. Beim Einlesen müssen doppelte Wörter erkannt werden.	*in*: Dateiname (Typ: `String`), Wortart (Typ: `Wortart`)

Tabelle 9: Operationen der Klasse WoerterbuchVerwaltung

4.3.2 Klasse Woerterbuch

Diese Klasse implementiert das Wörterbuch. Die Wörter werden in einer HashMap gespeichert; dabei wird pro Wortart ein Schlüssel in der Datenstruktur erstellt, welcher auf die Menge (HashSet) der zu dieser Wortart gehörenden Wörter zeigt. Z.B. „Substantiv“ -> {„Haus“„Auto“,...}.

Operation	Aufgabe	Parameter
`istVorhanden`	ist das Wort in einer Wortart bereits vorhanden?	*in*: Wort (Typ: `Wort`)
`fuegeEin`	Wort der entsprechenden Wortart zuweisen	*in*: Wort (Typ: `Wort`), Wortart (Typ: `Wortart`)
`loesche`	lösche ein Wort	*in*: Wort (Typ: `Wort`), Wortart (Typ: `Wortart`)
`initialisiere`	baut die HashMap auf	*in*: Dateiname (Typ: `String`)
`deinitialisiere`	löscht die HashMap im Speicher und speichert sie in der Datei ab	*in*: Dateiname (Typ: `String`)
`gibWortarten`	wird von der Klasse Wort aufgerufen, zu welchen Wortarten das Wort gehört	*in*: Wort (Typ: `Wort`), *out*: Wortarten (Typ: `Vector`)

Tabelle 10: Operationen der Klasse Woerterbuch

4.4 Komponente Statistik

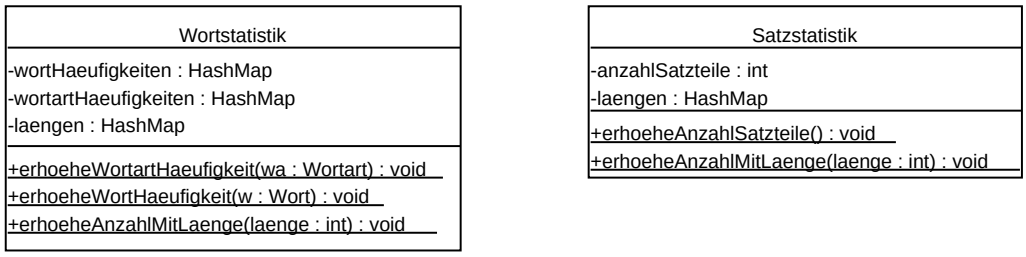

Abbildung 6: Komponente Statistik

4.4.1 Klasse Wortstatistik

Diese Klasse speichert statisch die Wort-Häufigkeiten (`wortHaeufigkeiten`), die Wortart-Häufigkeiten (`wortartHaeufigkeiten`) und die Längen (`laengen`) aller Wörter in einer HashMap.

Operation	Aufgabe	Parameter
`erhoeheWortartHaeufigkeit`	erhöhe die Anzahl der Wörter dieser Wortart um eins	*in*: Wortart (Typ: `Wortart`)
`erhoeheWortHaeufigkeit`	erhöhe die Anzahl des Wortes um eins	*in*: Wort (Typ: `Wort`)
`erhoeheAnzahlMitLaenge`	erhöhe die Anzahl des Wortes gleicher Länge	*in*: laenge (Typ: `int`)

Tabelle 11: Operationen der Klasse Wortstatistik

4.4.2 Klasse Satzstatistik

Diese Klasse speichert statisch die Anzahl der Satzteile (`anzahlSatzteile`) und die Anzahl der jeweiligen Satzlängen (`laengen`) in einer HashMap.

Operation	Aufgabe	Parameter
`erhoeheAnzahlSatzteile`	erhöhe die Anzahl der Satzteile	*in*: keine
`erhoeheAnzahlMitLaenge`	erhöhe die Anzahl des Satzes mit gleicher Länge	*in*: laenge (Typ: `int`)

Tabelle 12: Operationen der Klasse Satzstatistik