Dokumentation zu

UTOPP

der Gruppe

CvK Mber(t)

Version vom 04. 02. 2001

Inhaltsverzeichnis

1 Zielbestimmungen

Das Programm U-Topp stellt eine Art Case Tool dar, welches dem Entwickler bei der Softwareentwicklung mit UML hilfreich zur Seite stehen soll.Durch Eingabemasken für Objekte, Klassen, Verbindungen, Use Cases und für ein Pflichtenheft soll dem Entwickler die Planungsphase erleitert und übersichtlicher gestaltet werden. Wichtig ist die Modularität der einzelnen Programmteile, die es ermöglicht,jedes Modul einzeln einzubinden und somit die Möglichkeit bietet, die Einzelmodule zu einem Gesamtmodul(Projekt) zusammenzufügen. Weiterhin soll die Ansicht der einzelnen Module wechselbar sein. Dies bedeutet, dass dem Benutzer die Freiheit gegeben werden soll, eine grobe Übersicht zu erhalten, oder sich die gewünschten Details näher anzusehen.

1.1 Musskriterien

Das Programm soll Erfassmasken für die UML-Komponenten Klassen, Objekte, Use-Cases, Verbindungen und Pflichtenheft enthalten. Diese Komponenten sollen in unterschiedlichem Detaillierungsgrad graphisch darstellbar und auch dauerhaft speicherbar sein. Desweiteren gibt das Programm dem Benutzer die Möglichkeit die Komponenten zu editieren, d.h. die gespeicherten Daten zu verändern. Aufgelistet müsste man also folgende Punkte darstellen:

1. Erfassung der Daten für die Module Objekte, Klassen, Verbindungen, Use Cases und Pflichtenheft
2. Editiermöglichkeit dieser Daten
3. Speichermöglichkeit der Daten
4. Graphische Darstellung der Objekte(Ausgabe)
5. Wechseln des Anzeigemodus
6. Möglichkeit der Einbindung in ein späteres Gesamttool

1.2 Wunschkriterien

Das Programm könnte bestimmte Arten von Diagrammen erstellen, wie z.B. Klassen-, Use-Case- und Sequenzdiagramme. Wünschenswert wäre unter Umständen, diese Diagramme in verbreitete Formate exportieren und/oder drucken zu können. Außerdem könnte man die einzelnen Komponenten so vernetzen, dass zum Beispiel automatisch festgestellt würde, ob die eingegebenen Daten konsistent sind. Die Wunschkriterien wären somit:

1. Möglichkeit der Erstellung von Use Case Diagrammen
2. Darstellungsmöglichkeit von Klassendiagrammen
3. Erstellungsmöglichkeit von Sequenzdiagrammen
4. Export in gängige Formate, wie z.B. gif, tiff, jpg
5. Interaktion der Komponenten untereinander
6. Datenspeicherung in XML

1.3 Abgrenzungskriterien

Im Gegensatz zu anderen Systemen (wie z.B. ArgoUML) wird nicht die Möglichkeit bestehen, vorhandene Daten zur Generation von Quellcode (z.B. »Rahmen« von Javaklassen) zu verwenden. Die Darstellung der Komponenten erfolgt einzeln, d.h. nicht in einem Gesamtbild. Weiterhin wird man vorhandene Information nicht graphisch (d.h. durch direkten Zugriff auf eine Ansicht)ändern können. Eine klare Abgrenzung wäre die Tatsache, dass kein vollständiges UML-Tool, wie beispielsweise Argo-UML erstellt werden soll.

2 Produkteinsatz

Das Produkt kann im Bereich des Softwareengineerings eigesetzt werden zur Unterstützung des Entwicklers in der Planung, der Analyse und im Design mit UML. Eine weitere Möglichkeit ist die Präsentation von Entwicklungsideen.

2.1 Zielgruppe

Das Programm U-Topp ist für Benutzer einer objektorientierten Softwareentwicklung gedacht. Softwareentwicklung wird immer komplexer und anspruchsvoller. Aus diesen Gründen ist es sinnvoll nach Methoden zu suchen, welche die Komplexität beherrschbar machen und dem Anwender eine zuverlässige Übersicht über die bei der Softwareentwicklung zu berücksichtigen Komponenten zu geben, um so die Qualität und die Zuverlässigkeit der Software aufrechtzuerhalten. Das Programm U-Topp soll dem Entwickler (in Ahnlehnung an Argo-UML) ein wenig hilfreich in der Planungsphase zur Seite stehen, um den Blick für das Wesentliche nicht zu verlieren.

2.2 Betriebsbedingungen

Das Programm kann sicherlich später von folgender Webseite heruntergeladen werden: `wwwmath.uni-muenster.de/cs/u/lammers/Programmierpraktikum`. Ansprechpartner sind die für dieses Programm Verantwortlichen.

3 Produktumgebung

3.1 Software

Das Produkt benötigt eine virtuelle Java-Maschine, z.B. das Java-Runtime-Environment von Sun, das auch bei der Entwicklung verwendet wird. Zusätzlich wird zur Darstellung der erstellten HTMl-Datei des Pflichtenheftes ein funktionstüchtiger Browser benötigt (z.B. Netscape Navigator oder Internet Explorer.)

Für die Erstellung des Programmes wurde folgende Software genutzt:

1. Software: Symantic© VisualCafé wurde zur Erstellung eines Teils der Benutzerdialoge verwendet,
2. ArgoUML, um Diagramme (ZB das Klassendiagramm) zu zeichnen.
3. Für die manuell erstellten Klassen wurden JPad© und Kawa© verwendet.

3.2 Hardware

Die Software sollte im wesentlichen Plattform-unabhängig sein. Zumindest sollte sie unter folgenden zwei Hardware-Umgebungen laufen:

- Genügend ausgestatteter PC
- Sun©-Workstations im mathematischen Institut

Darüber hinaus wird keine besondere Hardware benötigt.

3.3 Orgware

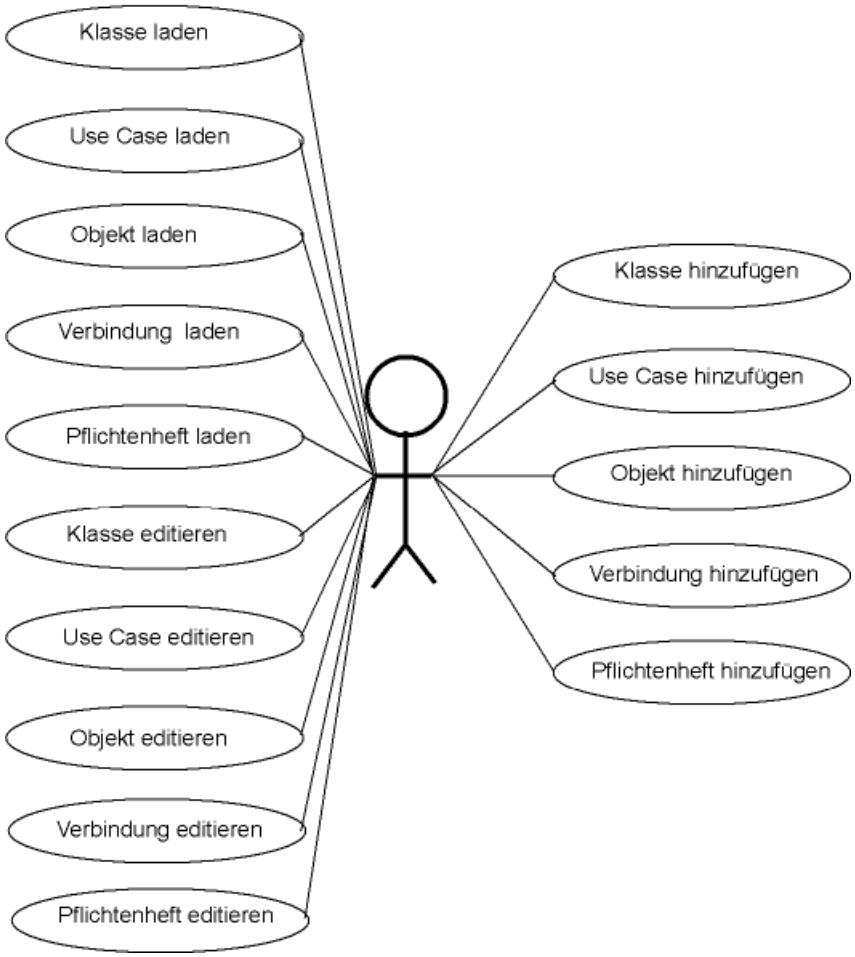
Es wird keine besondere Orgware benötigt.

3.4 Schnittstellen

Es werden keine besonderen Schnittstellen benötigt.

4 Produktdefinition

4.1 Use Case Diagramm



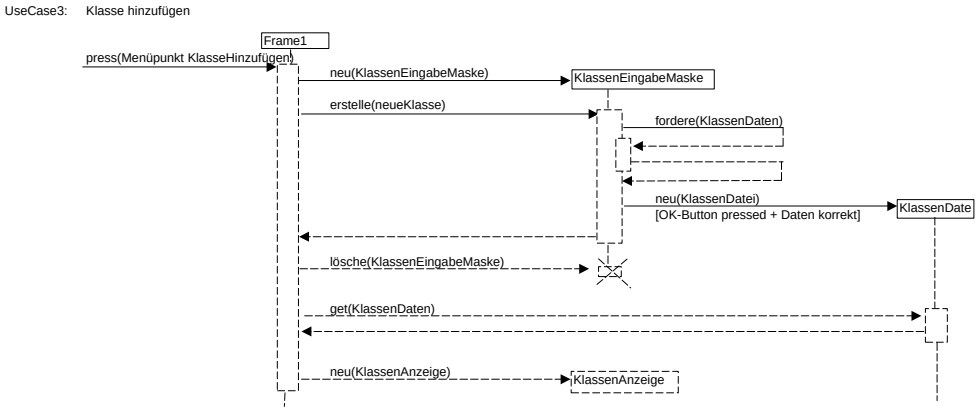
4.2 Akteure

Die alleinigen Akteure sind die Benutzer des Programms.

4.3 Anwendungsfälle

4.3.1 Klasse hinzufügen

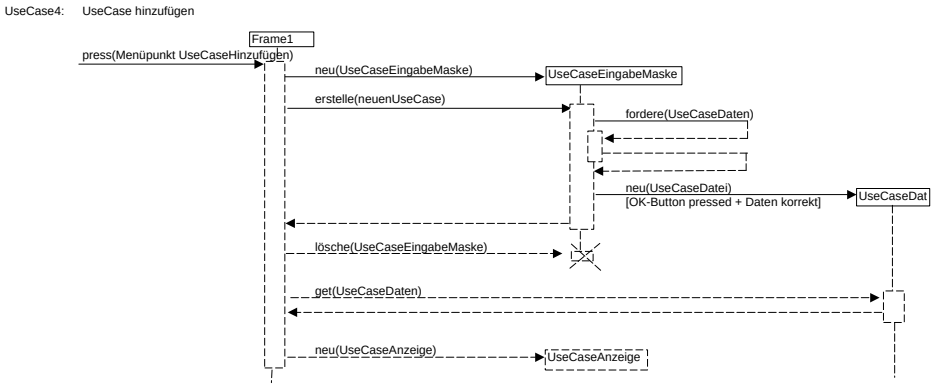
- **Akteure:** Benutzer
- **Vorbedingungen:** keine
- **Nachbedingungen:** Falls gewünscht wird die hinzugefügte Klasse abgespeichert, sonst gehen die eingegebenen Daten verloren
- **Ablaufbeschreibung:**
  1. Aufruf des Menüpunktes Klasse → Klasse hinzufügen
  2. Eine Eingabemaske für Stereotyp, Paket, Name der Klasse, Eigenschaftswerte, Attribute und Operationen erscheint. Die eingegeben Attribute bzw. Operationen können einer Attribut- bzw. Operationsliste hinzugefügt werden. Elemente dieser Listen können noch mit Typ, Intialwert und Zusicherung, bzw. Parameter und Zusicherung versehen werden. Desweitern besteht die Wahl zwischen public/private/protected und static/nonstatic
  3. Der Benutzer trägt die gewünschten Daten ein und bestätigt diese.
  4. Es erscheint eine Maske, die dem Benutzer die Möglichkeit gibt, die Klasse abzuspeichern (Typ: *.kla)



4.3.2 Use-Case hinzufügen

- **Akteure:** Benutzer
- **Vorbedingungen:** keine
- **Nachbedingungen:** Falls gewünscht wird der hinzugefügte Use-Case abgespeichert, sonst gehen die eingegebenen Daten verloren.
- **Ablaufbeschreibung:**
  1. Aufruf des Menüpunktes Use-Case → Use-Case editieren
  2. Eine Eingabemaske für Nummer, Name, Akteur, Vorbedingung, Nachbedingung, Invariante, Ablaufbeschreibung und Variation erscheint
  3. Der Benutzer kann »erweitert« wählen, dort kann er Nicht funktionale Anforderungen, Ausnahmen, Regeln, Services, Ansprechpartner, Anmerkungen und Dialogbeispiele eintragen, auf Bestätigung werden diese den Daten des Use-Cases hinzugefügt

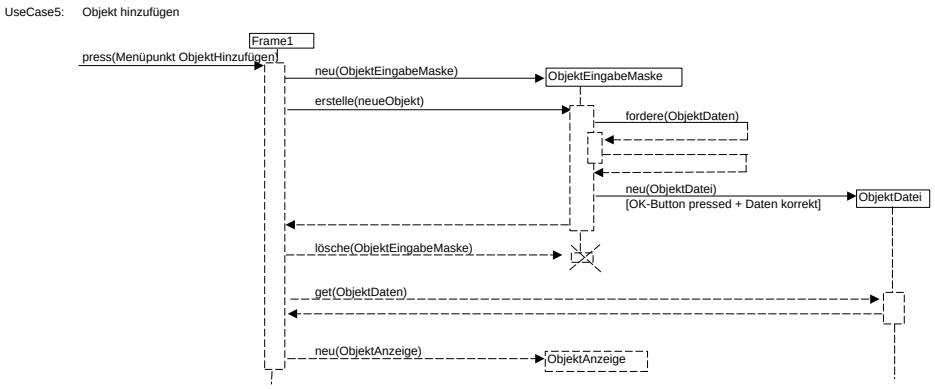
4. Der Benutzer trägt die gewünschten Daten ein und bestätigt diese.
5. Es erscheint eine Maske, die dem Benutzer die Möglichkeit gibt, den Use-Case abzuspeichern (typ: *.use)



4.3.3 Objekt hinzufügen

- **Akteure:** Benutzer
- **Vorbedingungen:** keine
- **Nachbedingungen:** Falls gewünscht wird das hinzugefügte Objket abgespeichert, sonst gehen die eingegebenen Daten verloren.
- **Ablaufbeschreibung:**
    1. Aufruf des Menüpunktes Objekt → Objekt hinzufügen

2. Eine Eingabemaske für Objektname, Objektzugehörige Klasse, und Attribut erscheint. Die eingegebenen Attribute können einer Attributliste hinzugefügt werden. Elemente dieser Liste können mit einem Initialwert versehen werden
3. Der Benutzer trägt die gewünschten Daten ein und bestätigt diese.
4. Es erscheint eine Maske, die dem Benutzer die Möglichkeit gibt, das Objekt abzuspeichern (Typ: *.obj)

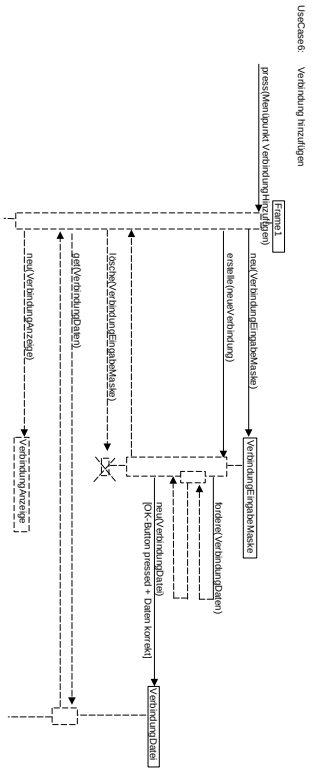


4.3.4 Verbindung hinzufügen

- **Akteure:** Benutzer
- **Vorbedingungen:** keine
- **Nachbedingungen:** Falls gewünscht wird die hinzugefügte Verbindung abgespeichert, sonst gehen die eingegebenen Daten verloren

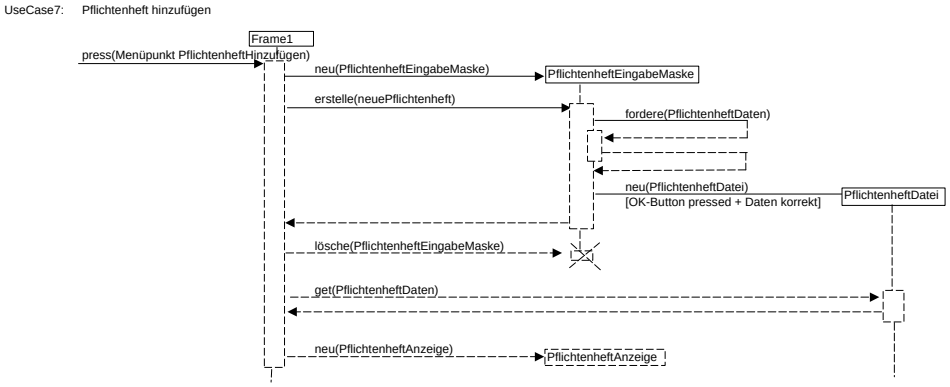
- **Ablaufbeschreibung:**

  1. Aufruf des Menüpunktes Verbindung → Verbindung hinzufügen
  2. Eine Eingabemaske für Name, Stereotyp, Beziehungsname, Zusicherung, Eigenschaftswere, Multiplizität links/rechts erscheint. Es kann eine Leserichtung gewählt werden. Unter »(Pfeil-)spitze « kann gewählt werden: Linie durchgezogen/unterbrochen, Pfeilspitze links/rechts/keine, Pfeilspitze offen/dreieckig unausgefüllt/dreieckig ausgefüllt/Raute ausgefüllt/ Raute unausgefüllt
  3. Der Benutzer trägt die gewünschten Daten ein und bestätigt diese.
  4. Es erscheint eine Maske, die dem Benutzer die Möglichkeit gibt, die Verbindung abzuspeichern (Typ: *.ver)



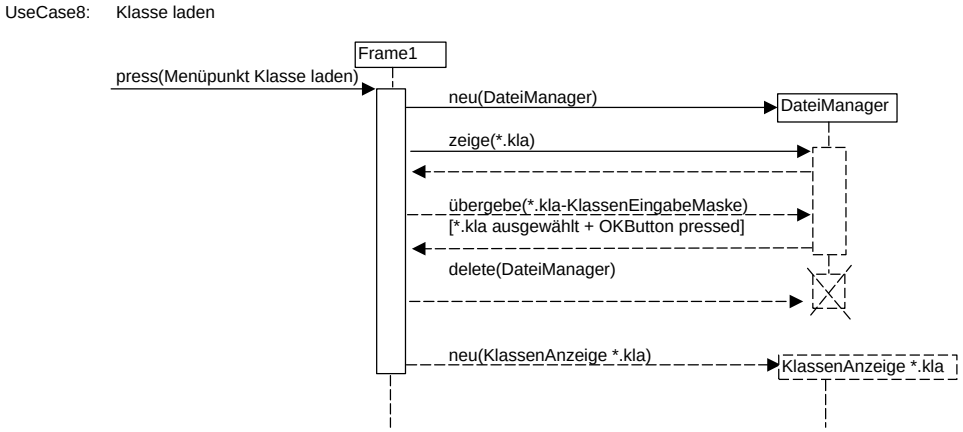
4.3.5 Pflichtenheft hinzufügen

- **Akteure:** Benutzer
- **Vorbedingungen:** keine
- **Nachbedingungen:** Falls gewünscht wird das hinzugefügte Pflichtenheft abgespeichert, sonst gehen die eingegebenen Daten verloren
- **Ablaufbeschreibung:**
  1. Aufruf des Menüpunktes Pflichtenheft → Pflichtenheft hinzufügen
  2. Eine Eingabemaske für die im Pflichtenheft zu tätigenden EInträge erscheint.
  3. Der Benutzer trägt die gewünschten Daten ein. Er kann nun ein Pflichtenheft in HTML erzeugen oder bestätigen.Auf zweiteres erscheint eine Maske, die dem Benutzer die Möglichkeit gibt, das Pflichtenheft abzuspeichern (Typ: *.pfl)



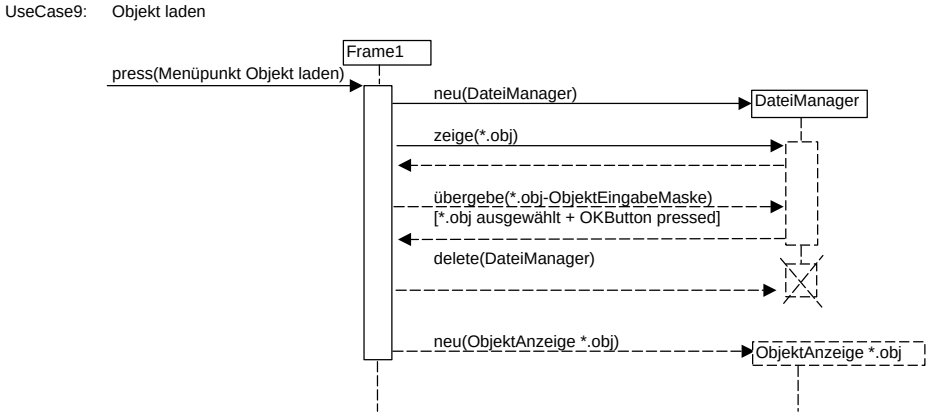
4.3.6 Klasse laden

- **Akteure:** Benutzer
- **Vorbedingungen:** keine
- **Nachbedingungen:** Es erscheint ein Rahmen in dem das gewählte Element vom Typ Klasse dargestellt wird. In diesem kann es bearbeitet werden. Zur Verfügung stehen die Menüpunkte editiern, sowie Modus → simpel/standard/detailiert
- **Ablaufbeschreibung:**
  1. Aufruf des Menüpunktes Klasse → Klasse laden
  2. Es erscheint eine Maske, die dem Benutzer die Möglichkeit gibt, ein bereits gespeichertes Element zu laden
  3. Der oben beschriebene Rahmen erscheint



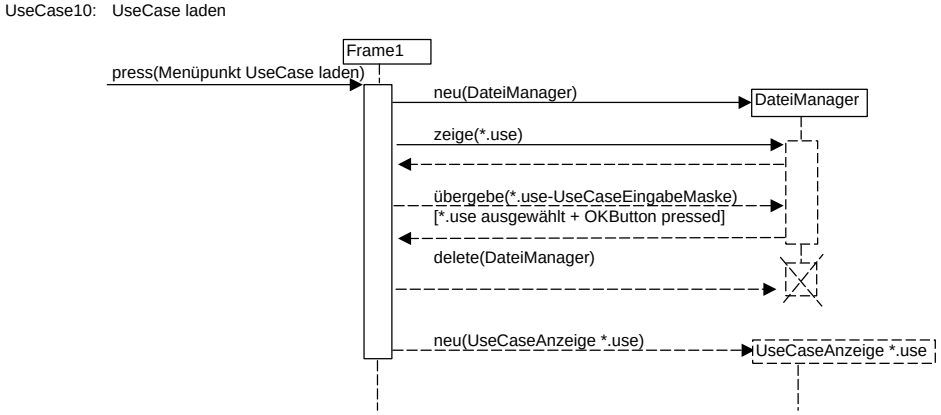
4.3.7 Objekt laden

- **Akteure:** Benutzer
- **Vorbedingungen:** keine
- **Nachbedingungen:** Es erscheint ein Rahmen in dem das gewählte Element vom Typ Objekt dargestellt wird. In diesem kann es bearbeitetwerden. Zur Verfügung stehen die Menüpunkte editiern, sowie Modus → simpel/standard/detailiert
- **Ablaufbeschreibung:**
  1. Aufruf des Menüpunktes Objekt → Objekt laden
  2. Es erscheint eine Maske, die dem Benutzer die Möglichkeit gibt, ein bereits gespeichertes Element zu laden
  3. Der oben beschriebene Rahmen erscheint



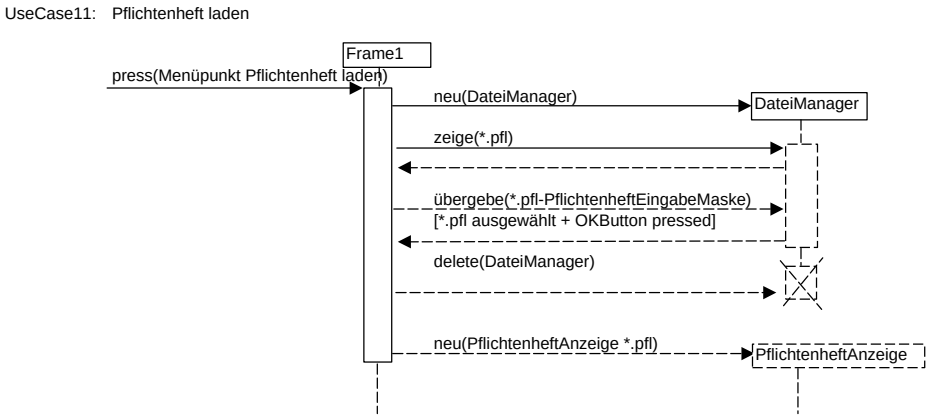
4.3.8 Use-Case laden

- **Akteure:** Benutzer
- **Vorbedingungen:** keine
- **Nachbedingungen:** Es erscheint ein Rahmen in dem das gewählte Element vom Typ Use-Case dargestellt wird. In diesem kann es bearbeitet werden. Zur Verfügung stehen die Menüpunkte editiern, sowie Modus → simpel/standard/detailiert
- **Ablaufbeschreibung:**
  1. Aufruf des Menüpunktes Use-Case → Use-Case laden
  2. Es erscheint eine Maske, die dem Benutzer die Möglichkeit gibt, ein bereits gespeichertes Element zu laden
  3. Der oben beschriebene Rahmen erscheint



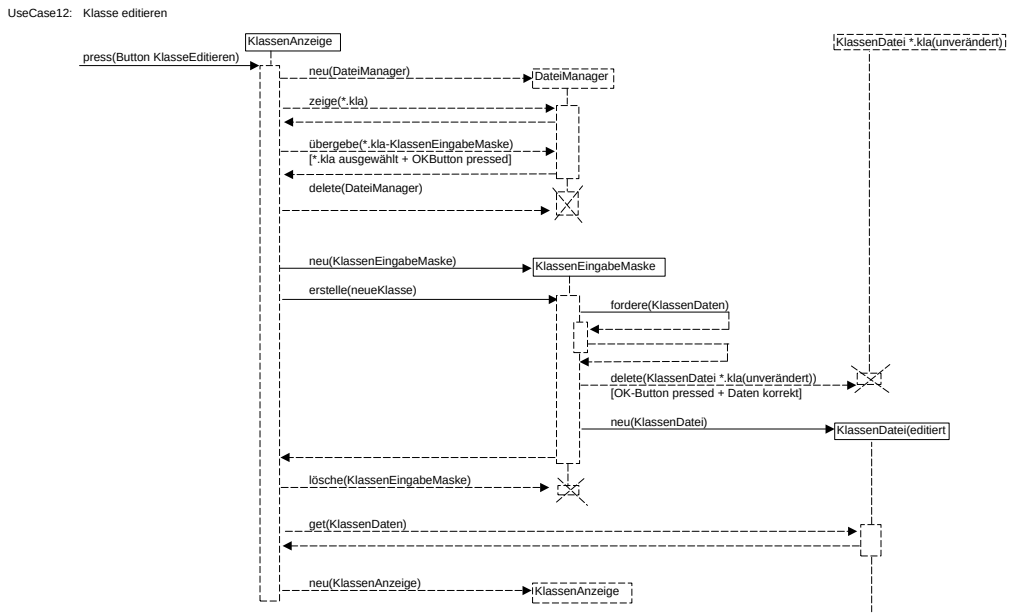
4.3.9 Pflichtenheft laden

- **Akteure:** Benutzer
- **Vorbedingungen:** keine
- **Nachbedingungen:** Es erscheint ein Rahmen in dem das gewählte Element vom Typ Pflichtenheft dargestellt wird. In diesem kann es bearbeitet werden. Zur Verfügung stehen die Menüpunkte editiern, sowie Modus → simpel/standard/detailiert
- **Ablaufbeschreibung:**
  1. Aufruf des Menüpunktes Pflichtenheft → Pflichtenheft laden
  2. Es erscheint eine Maske, die dem Benutzer die Möglichkeit gibt, ein bereits gespeichertes Element zu laden
  3. Der oben beschriebene Rahmen erscheint



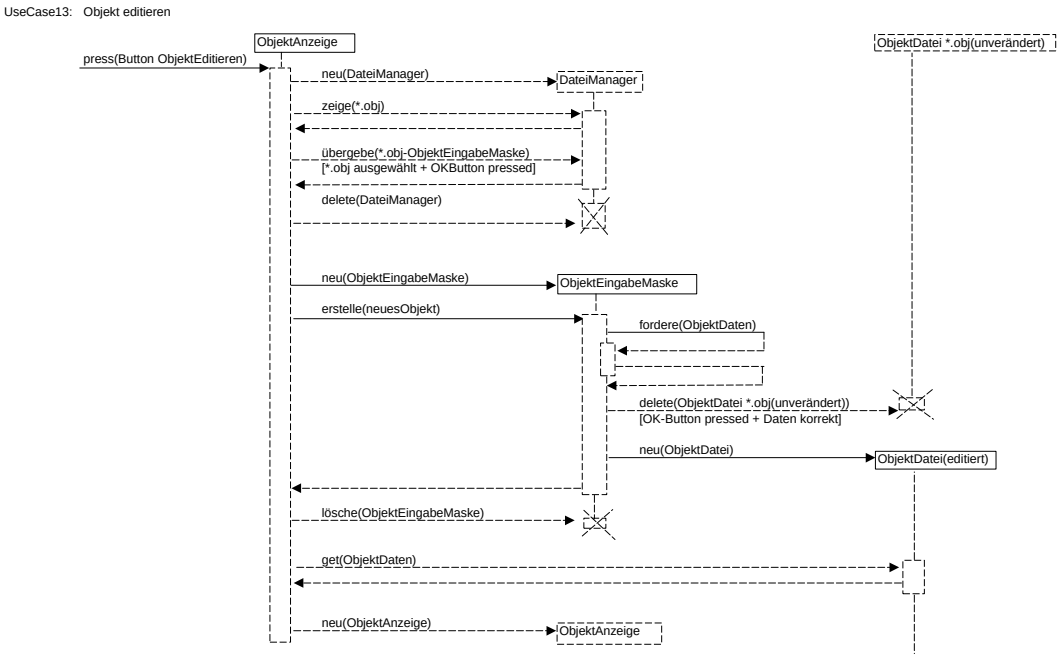
4.3.10 Klasse editieren

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein darstellender Rahmen für Klasse existiert
- **Nachbedingungen:** Eine Klasse ist editiert und die Änderungen sind abgespeichert worden. Ist letzteres nicht der Fall, so gehen die Änderungsdaten verloren
- **Ablaufbeschreibung:**
  1. Aufruf des Menüpunktes editieren
  2. Die entsprechende Eingabemaske mit den bereits gegebenen Daten erscheint. Diese können nun nach Belieben geändert werden. Auf Bestätigung kann die Änderung abgespeichert werden.



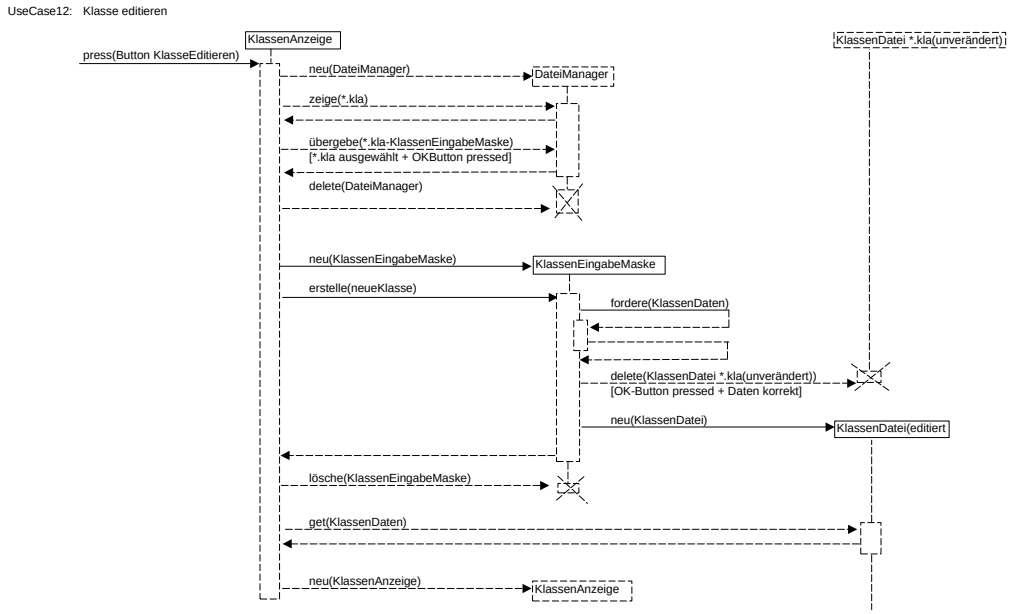
4.3.11 Objekt editieren

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein darstellender Rahmen für Objekt existiert
- **Nachbedingungen:** Ein Objekt ist editiert und die Änderungen sind abgespeichert worden. Ist letzteres nicht der Fall, so gehen die Änderungsdaten verloren
- **Ablaufbeschreibung:**
  1. Aufruf des Menüpunktes editieren
  2. Die entsprechende Eingabemaske mit den bereits gegebenen Daten erscheint. Diese können nun nach Belieben geändert werden. Auf Bestätigung kann die Änderung abgespeichert werden.



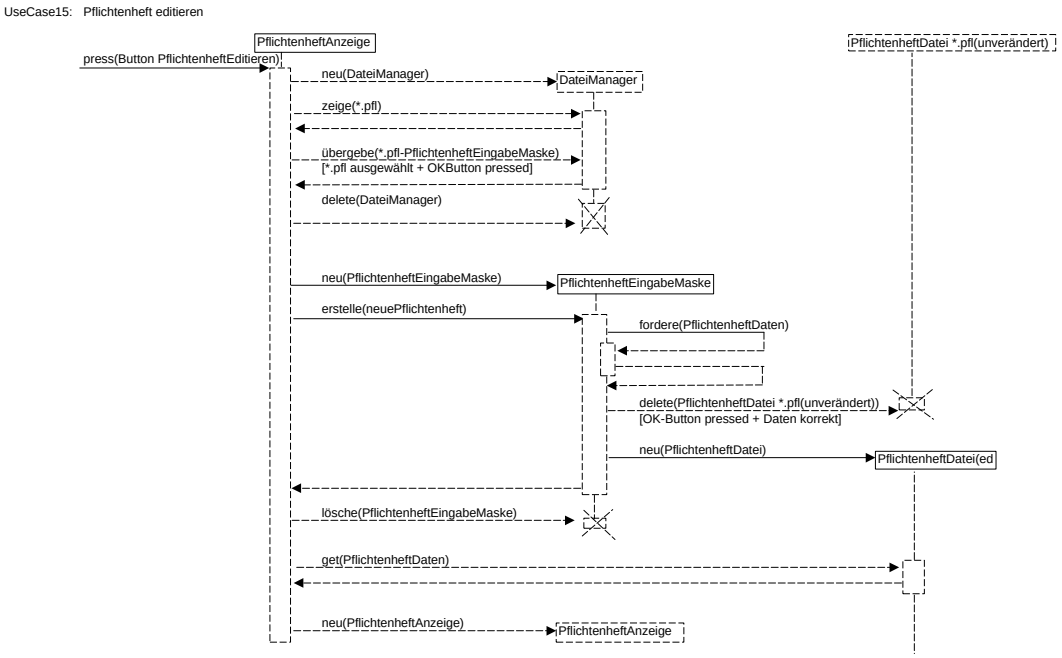
4.3.12 Use-Case editieren

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein darstellender Rahmen für Use-Case existiert
- **Nachbedingungen:** Ein Use-Case ist editiert und die Änderungen sind abgespeichert worden. Ist letzteres nicht der Fall, so gehen die Änderungsdaten verloren.
- **Ablaufbeschreibung:**
  1. Aufruf des Menüpunktes editieren
  2. Die entsprechende Eingabemaske mit den bereits gegebenen Daten erscheint. Diese können nun nach Belieben geändert werden. Auf Bestätigung kann die Änderung abgespeichert werden



4.3.13 Pflichtenheft editieren

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein darstellender Rahmen für Pflichtenheft existiert
- **Nachbedingungen:** Ein Pflichtenheft ist editiert und die Änderungen sind abgespeichert worden. Ist letzteres nicht der Fall, so gehen die Änderungsdaten verloren
- **Ablaufbeschreibung:**
  1. Aufruf des Menüpunktes editieren
  2. Die entsprechende Eingabemaske mit den bereits gegebenen Daten erscheint. Diese können nun nach Belieben geändert werden. Auf Bestätigung kann die Änderung abgespeichert werden



5 Produktleistungen

5.1 Funktionen

- Objekte vom Typ Objekt, Klasse, Verbindung, Use Case und Pflichtenheft erstellen
- HTML-Dokument des Pflichtenheftes erstellen
- Einträge in die Eingabemasken erstellen
- Einträge in den Eingabemasken bearbeite
- Einträge der Eingebemasken speichern
- Anzeige der Einträge in unterschiedlichen Modi

5.2 Leistungsanforderungen

- Anzeige der Objekte in äußerlich schöner Form innerhalb kurzer Zeit
- Erstellung von beliebig vielen Objekten

6 Benutzeroberfläche

6.1 Grundkonzeption

Nach dem Start des Programms erscheint eine Menüleiste, der Frame1, die den Ausgangspunkt zu allen anderen Funktionen, die UTopp ermöglicht, darstellt. Die darin enthaltenen Meüpunkte sind »Klasse«, »Objekt«, »Verbindungen«, »UseCases« und »Pflichtenheft«, die jeweils die Unterpunkte »hinzufügen« und »laden« enthalten.

Der Ablauf der weiteren Funktionen ist bei allen diesen Möglichkeiten, das Pflichtenheft ausgenommen, der gleiche: **»hinzufügen«** lässt eine leere Eingabemaske des jeweils angewählten Menüpunktes erscheinen, mit den Optionsbuttons »abbrechen« und »speichern«. Wird »abbrechen« angeklickt, wird die Eingabemaske geschlossen. Bei »speichern« wird die Eingabemaske durch einen Dateimanager ersetzt, worauf, falls wieder bestätigt wird, auch dieser verschwindet und durch eine Anzeige des hinzugefügten Objektes ersetzt wird.

Bei **»laden «** wird ein Dateimanager aufgerufen, um den User die gewünschte Datei auswählen zu lassen. Bei Bestätigung verschwindet der Dateimanager und die Anzeige des gewählten Objektes erscheint.

Auf dieser AnzeigenOberfläche exisitiert eine weitere Menüleiste mit den Menüpunkten »Ansicht« und »Editieren«. Ersteres ändert den Detailliertheitsgrad der Anzeige, dh verringert die Anzahl der angezeigten Informationen oder vergrössert sie. »Editieren« lässt die Anzeige verschwinden und ruft dafür die Eingabemaske mit den Daten dieses speziellen Objektes auf. Werden diese durch »Speichern« bestätigt, erscheint die modifizierte Anzeigeoberfläche.

6.2 Benutzerführung

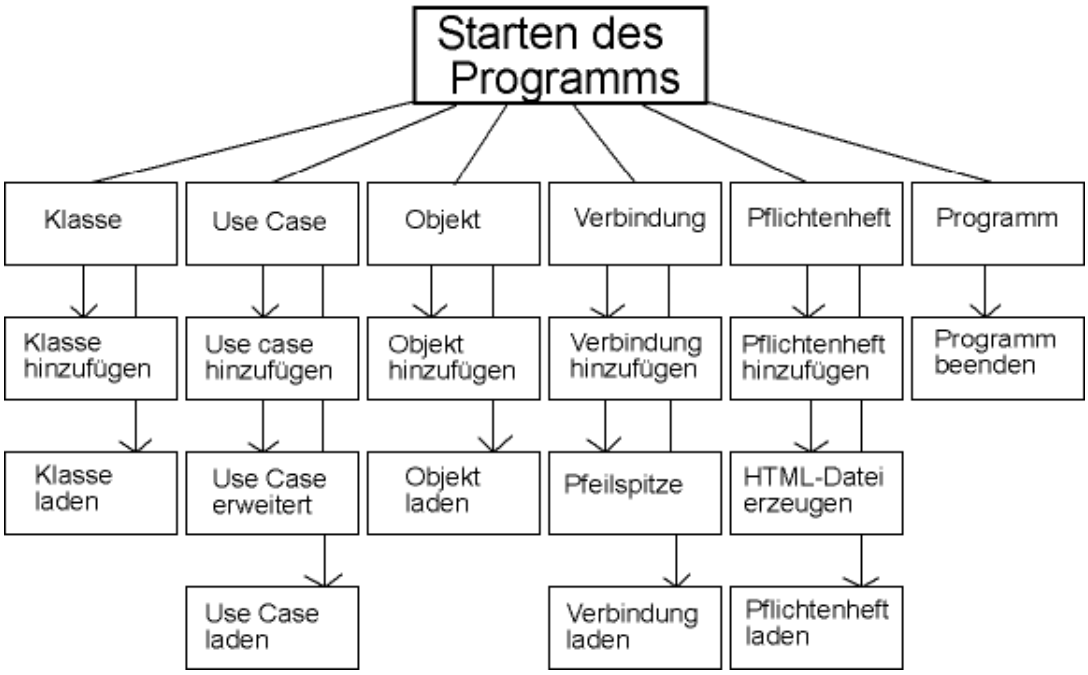


Abbildung 1: Menüführung des Hauptmenüs und der Eingabemasken

6.3 Screenshots

Abbildung 2: Hauptmenü

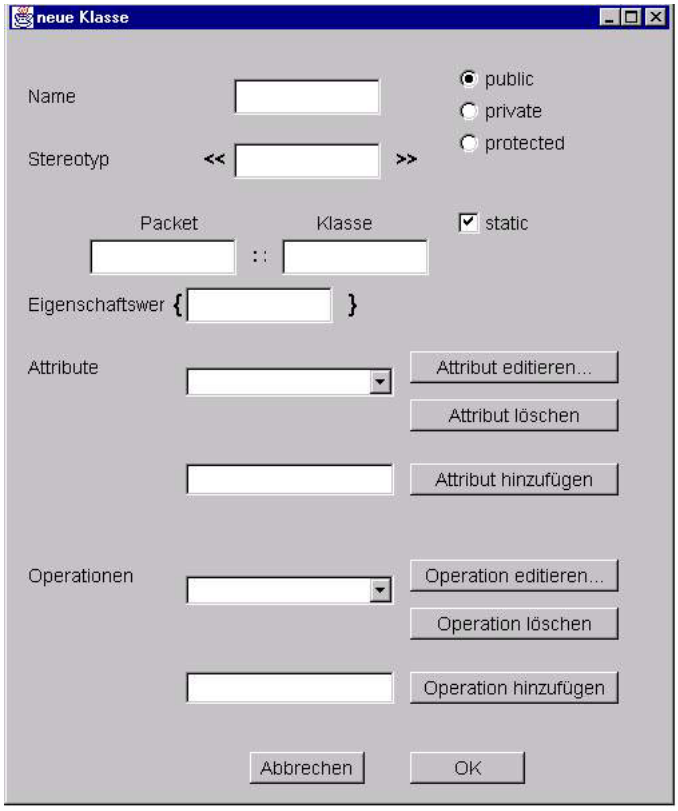


Abbildung 3: Eingabemaske für eine neue Klasse

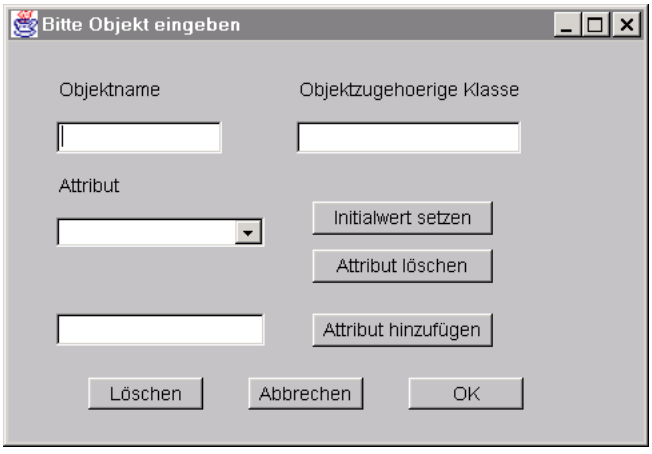


Abbildung 4: Eingabemaske für ein neues Objekt

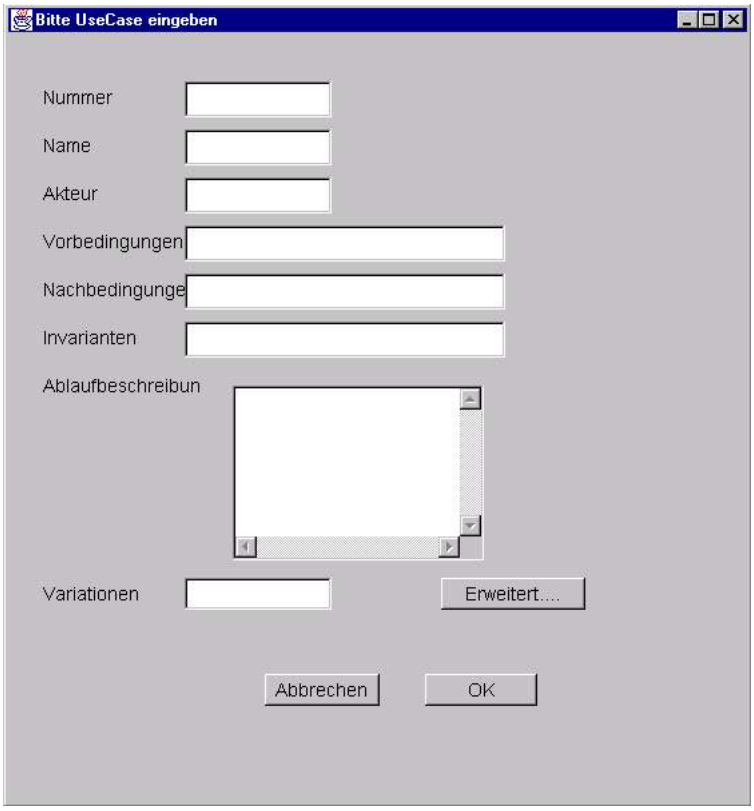


Abbildung 5: Eingabemaske für einen neuen Use-Case

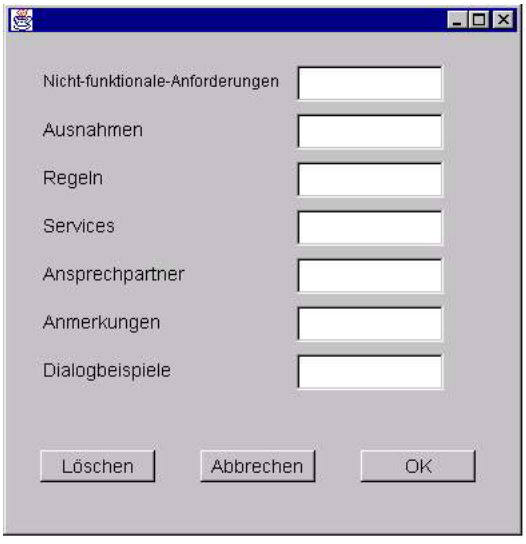


Abbildung 6: Eingabemaske für optionale Angaben zu Use-Cases

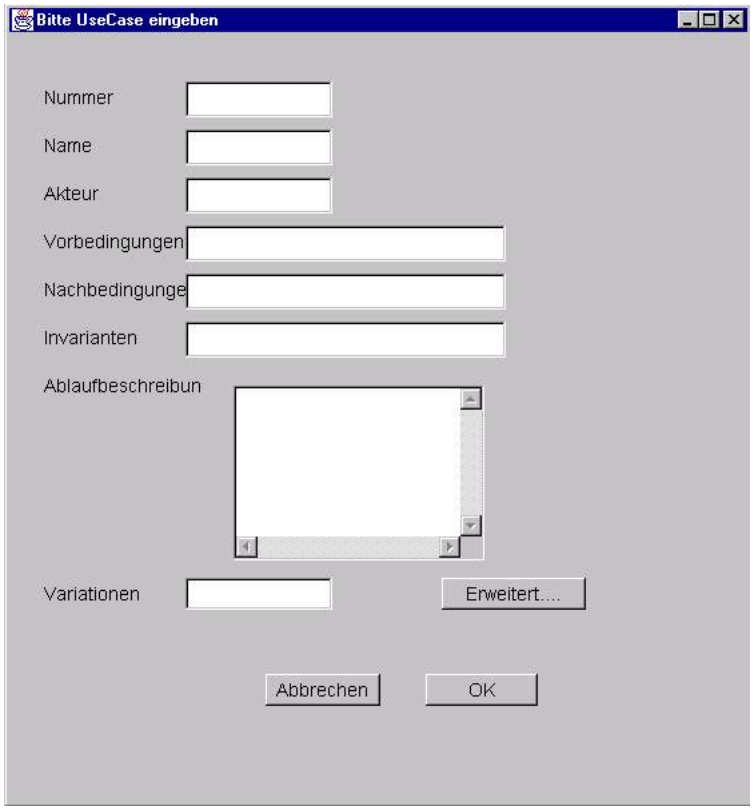


Abbildung 7: Eingabemaske für Verbindungen

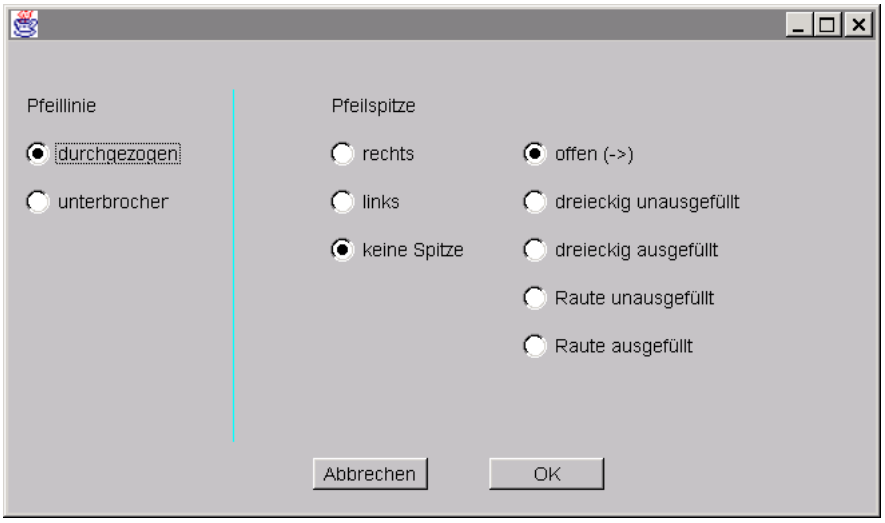


Abbildung 8: Eingabemaske für die Spitze der Verbindung

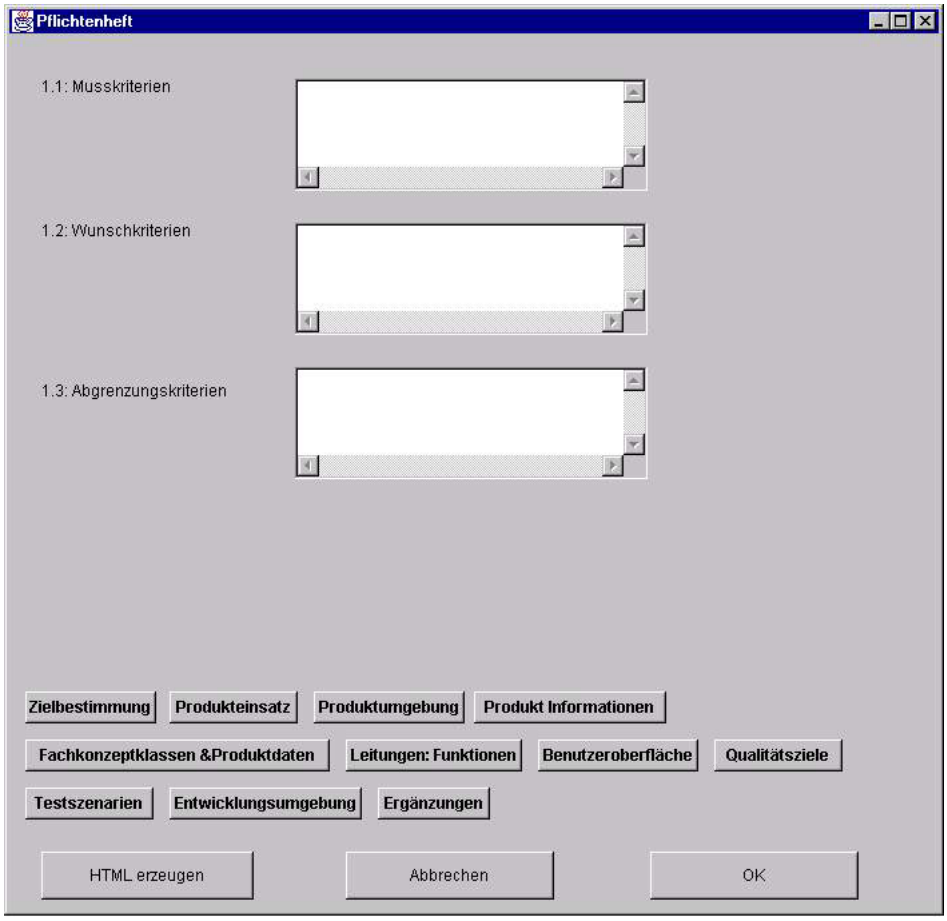


Abbildung 9: Eingabemaske für das Pflichtenheft

7 Qualitätsziele

Ziele	sehr wichtig	wichtig	weniger wichtig	unwichtig
Funktionalität			○	
Zuverlässigkeit		+		
Benutzbarkeit		+		
Effizienz		+		
Änderbarkeit	++			

8 Testfälle

8.1 Programmstart

Nach dem Aufruf des Programms in der Eingabeaufforderung oder mit einem geeigneten Tool erscheint lediglich die AWT Applikation Leiste mit einer Menüleiste, welche die Menüpunkte Klasse, Use-Case, Objekt, Verbindung, Pflichtenheft und Programm enthält. Wird das Programm unter falschem Namen aufgerufen, so erscheint natürlich keine Leiste. Groß- und Kleinschreibung sind also zu beachten.

8.2 Klasse hinzufügen

Anlegen mit vollständigen Angaben: Einträge werden bestätigt durch automatische Anzeige der neu erstellten Klasse (nach der Speicherung) in der Klassen-Anzeige

Anlegen mit unvollständigen Angaben: Einträge werden bestätigt durch automatische Anzeige der neu erstellten Klasse (nach der Speicherung) in der Klassen-Anzeige. Die Klasse wird als private, protected und public dargestellt, wenn entsprechende Angaben fehlen.

Anlegen mit Formatfehlern: Die Klasse wird dargestellt. Bei zu langen Eingaben läuft die Eingabe über den Darstellungsrahmen jedoch hinaus. Sonstige Formatfehler sind nicht bekannt.

8.3 Use-Case hinzufügen

Anlegen mit vollständigen Angaben: Einträge werden bestätigt durch automatische Anzeige der neu erstellten Klasse (nach der Speicherung) in der Use-Case-Anzeige.

Anlegen mit unvollständigen Angaben: Einträge werden bestätigt durch automatische Anzeige des neu erstellten Use-Cases (nach der Speicherung) in der Use-Case-Anzeige. Bei der normalen und der detaillierten Darstellung bleiben die entsprechenden Unterpunkte leer, können später aber durch weiteres bearbeiten gefüllt werden.

Anlegen mit Formatfehlern: Es sind keine Formatfehler bekannt. Jede erdenkliche Eingabe ist möglich.

8.4 Objekt hinzufügen

Anlegen mit vollständigen Angaben: Einträge werden bestätigt durch automatische Anzeige des neu erstellten Objektes (nach der Speicherung) in der Objekt-Anzeige.

Anlegen mit unvollständigen Angaben: Einträge werden bestätigt durch automatische Anzeige des neu erstellten Objektes (nach der Speicherung) in der Objekt-

Anzeige. Anlegen mit Formatfehlern: Bei zu langen Eingeben überschreitet die Eingabe die Grenze des Darstellungskastens. Ansonsten sind keine Formatfehler bekannt.

8.5 Verbindung hinzufügen

Anlegen mit vollständigen Angaben: Einträge werden bestätigt durch automatische Anzeige der neu erstellten Verbindung (nach der Speicherung) in der Verbindungs-Anzeige.

Anlegen mit unvollständigen Angaben: Einträge werden bestätigt durch automatische Anzeige der neu erstellten Verbindung (nach der Speicherung) in der Verbindungs-Anzeige.

Anlegen mit Formatfehlern: Es sind keine Formatfehler bekannt.

8.6 Pflichtenheft hinzufügen

Anlegen mit vollständigen Angaben: Einträge werden gespeichert und nach dem Erzeugen einer HTML-Datei kann man sich diese ansehen.

Anlegen mit unvollständigen Angaben: Einträge werden gespeichert und nach dem Erzeugen einer HTML-Datei kann man sich diese ansehen. Diejenigen Unterpunkte, zu denen keine Einträge gemacht wurden, bleiben leer.

Anlegen mit Formfehlern: Es sind keine Formfehler bekannt.

8.7 Einträge ändern

Alle Einträge können geändert werden. Je nachdem, ob die Angaben dann vollständig, unvollständig oder mit Formatfehlern behaftet sind treten dann entsprechende schon genannte Fälle auf.

8.8 Einträge speichern

Das Speichern findet automatisch nach Anlegen oder Ändern einer Instanz statt. Die Wahl eines Dateinames bleibt dem Benutzer überlassen.

8.9 Einträge löschen

Ein löschen der Einträge innerhalb des Programms ist nicht möglich. Es kann jedoch ganz normal ein löschen der Verzeichnisse innerhalb des Betriebssystems getätigt werden.

8.10 Programm beenden

Die Leiste und die Menüleiste verschwinden wie gewünscht und das Programm ist beendet.

9 Fachkonzeptklassen und Produktdaten

9.1 Klassendiagramme

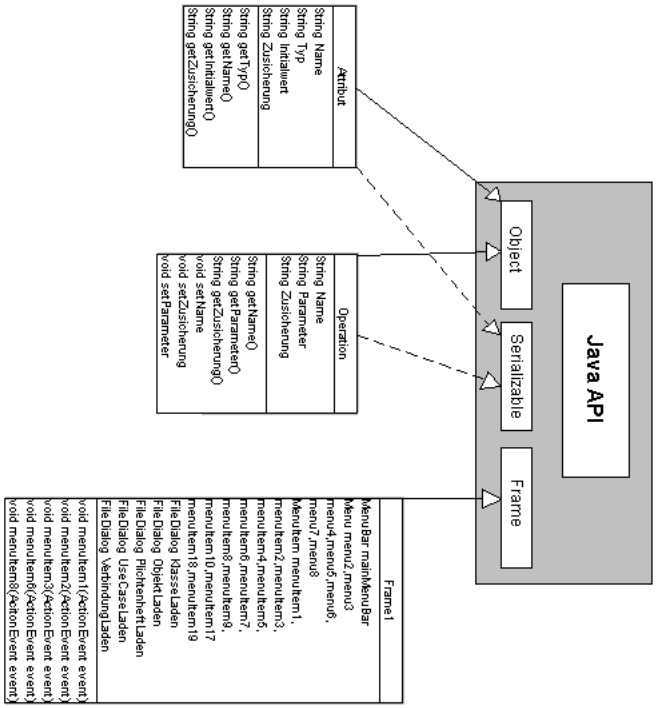


Abbildung 10: Das Klassendiagramm für das Hauptmenü

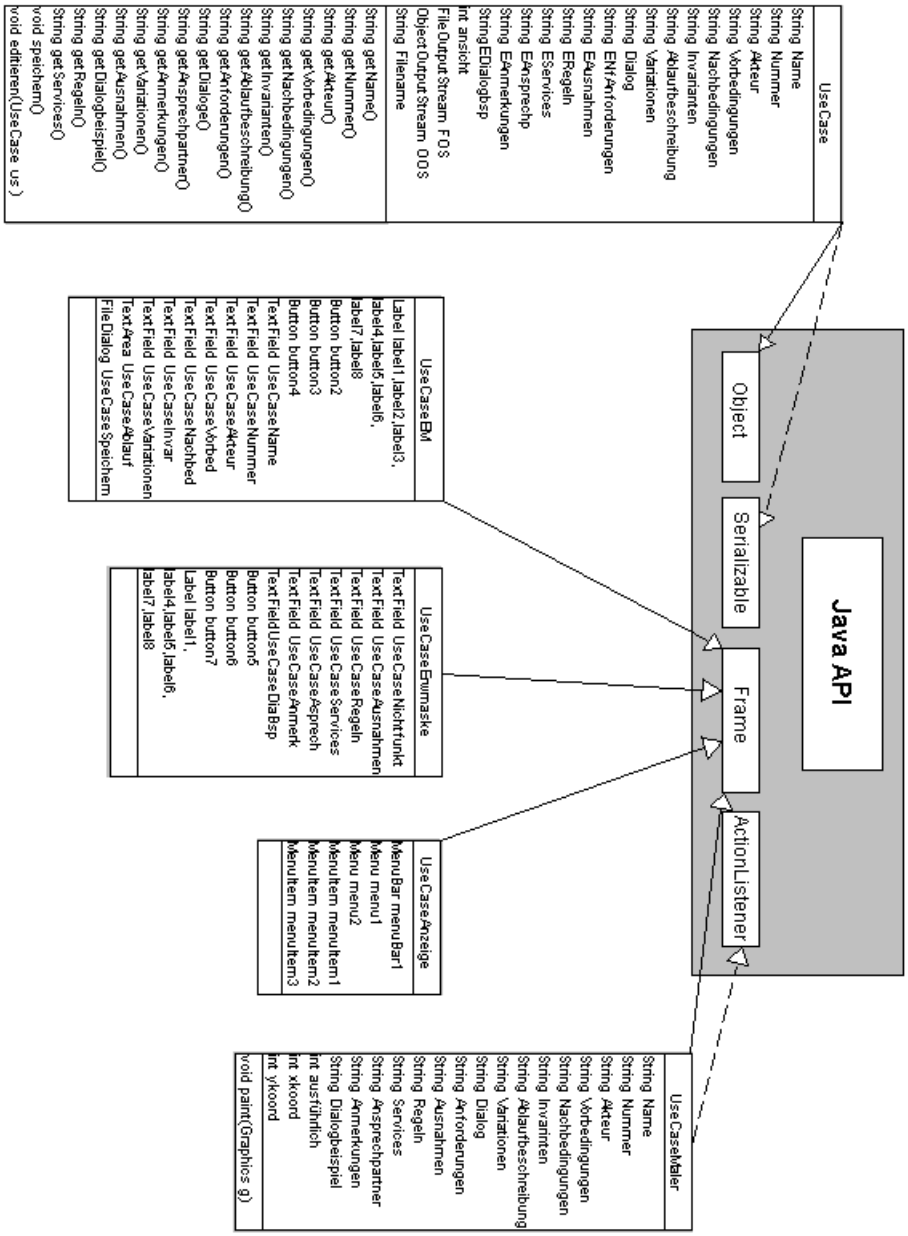


Abbildung 11: Das Klassendiagramm für den Bereich Use-Cases

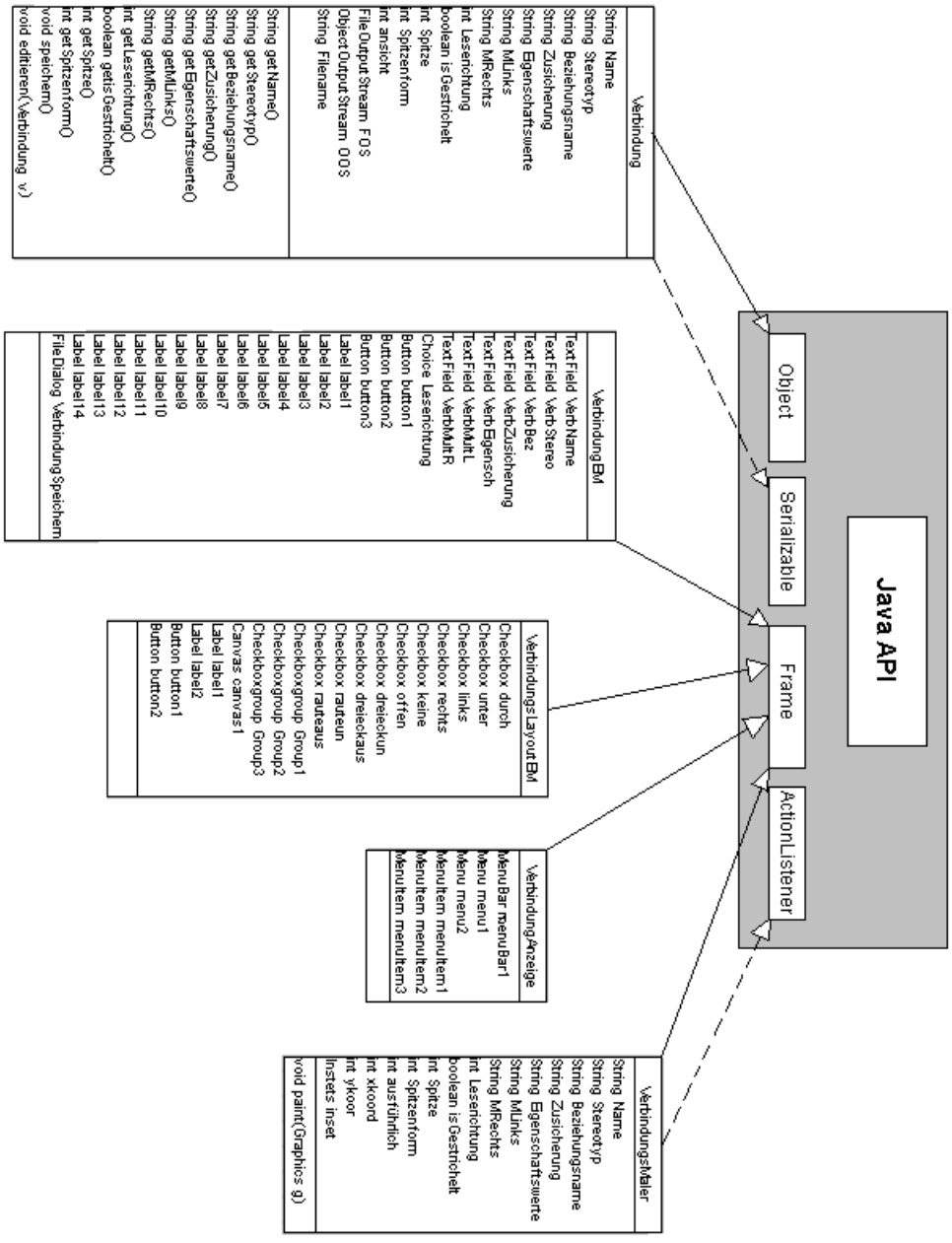


Abbildung 12: Das Klassendiagramm für den Bereich Verbindungen

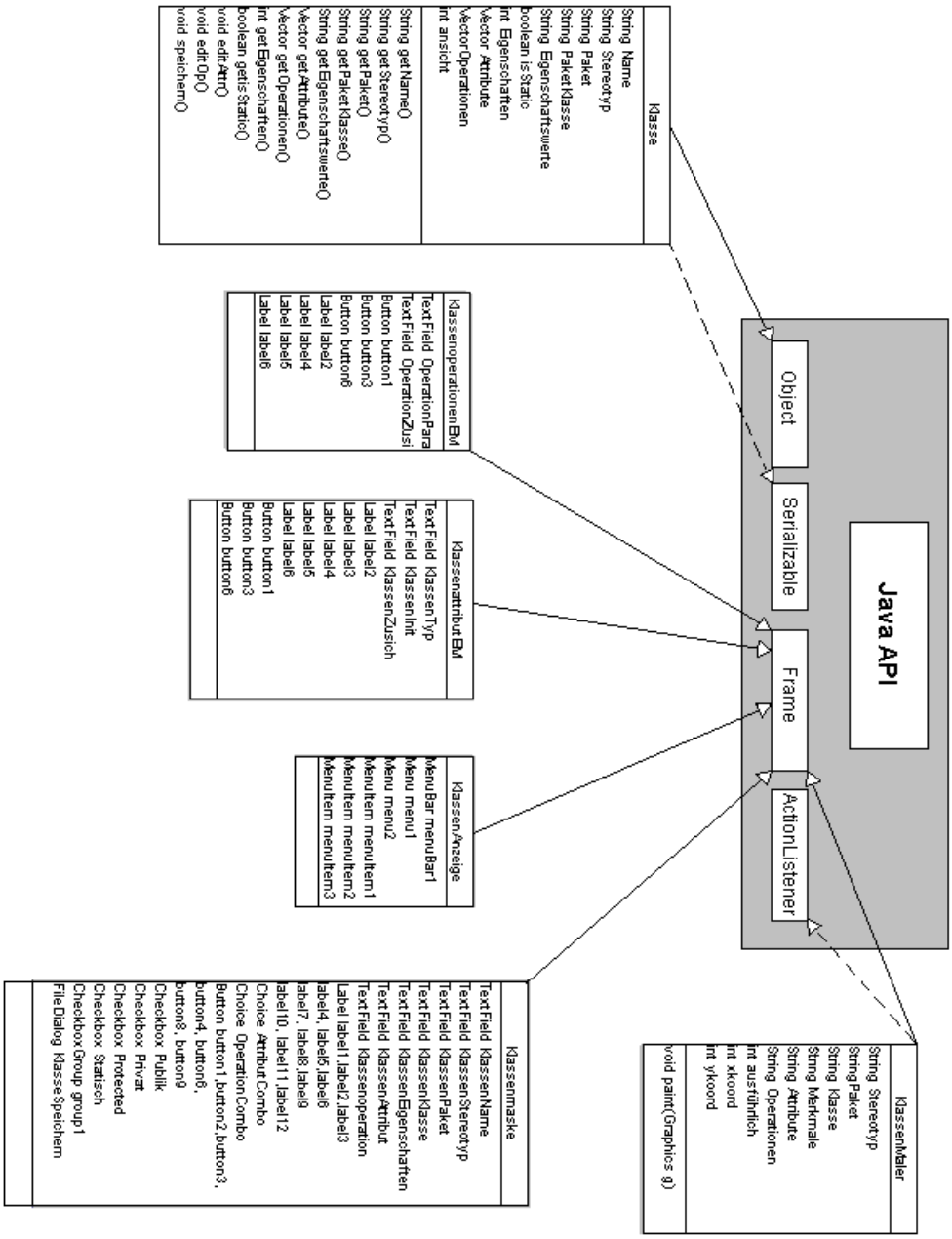


Abbildung 13: Das Klassendiagramm für den Bereich Klassen

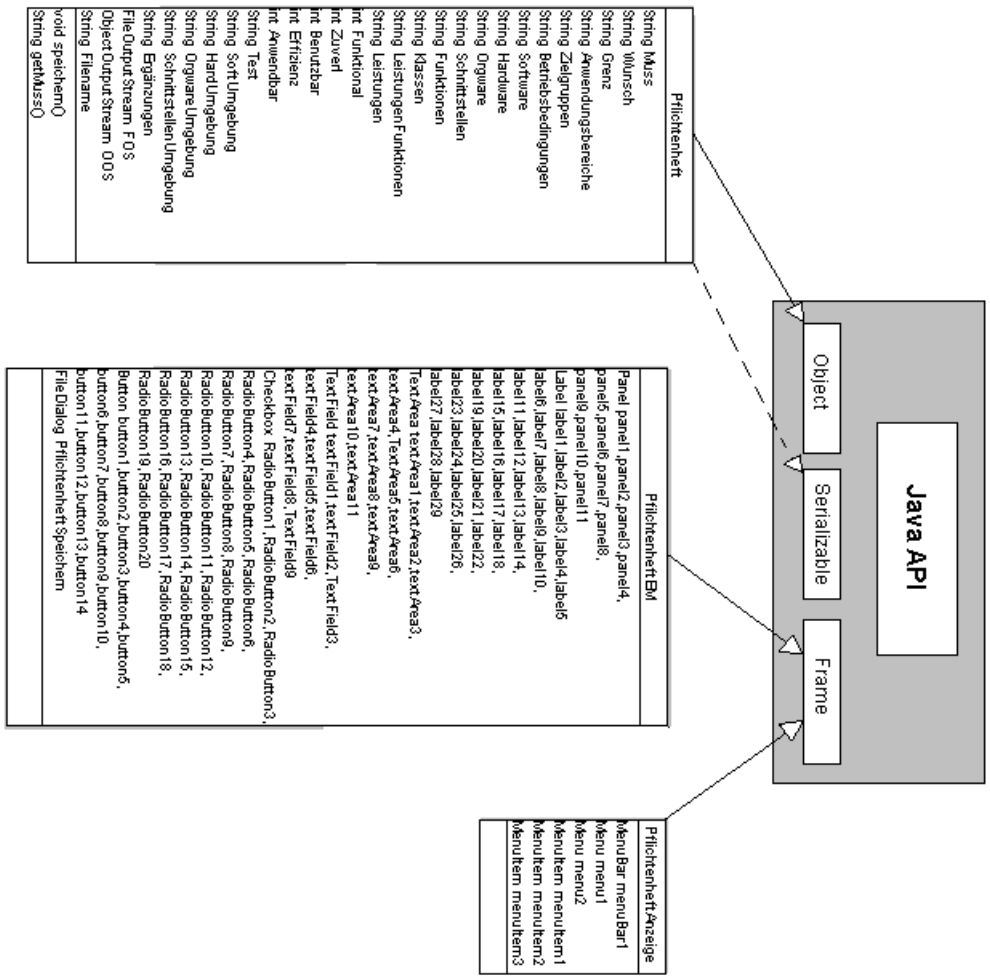


Abbildung 14: Das Klassendiagramm für den Bereich Pflichtenheft

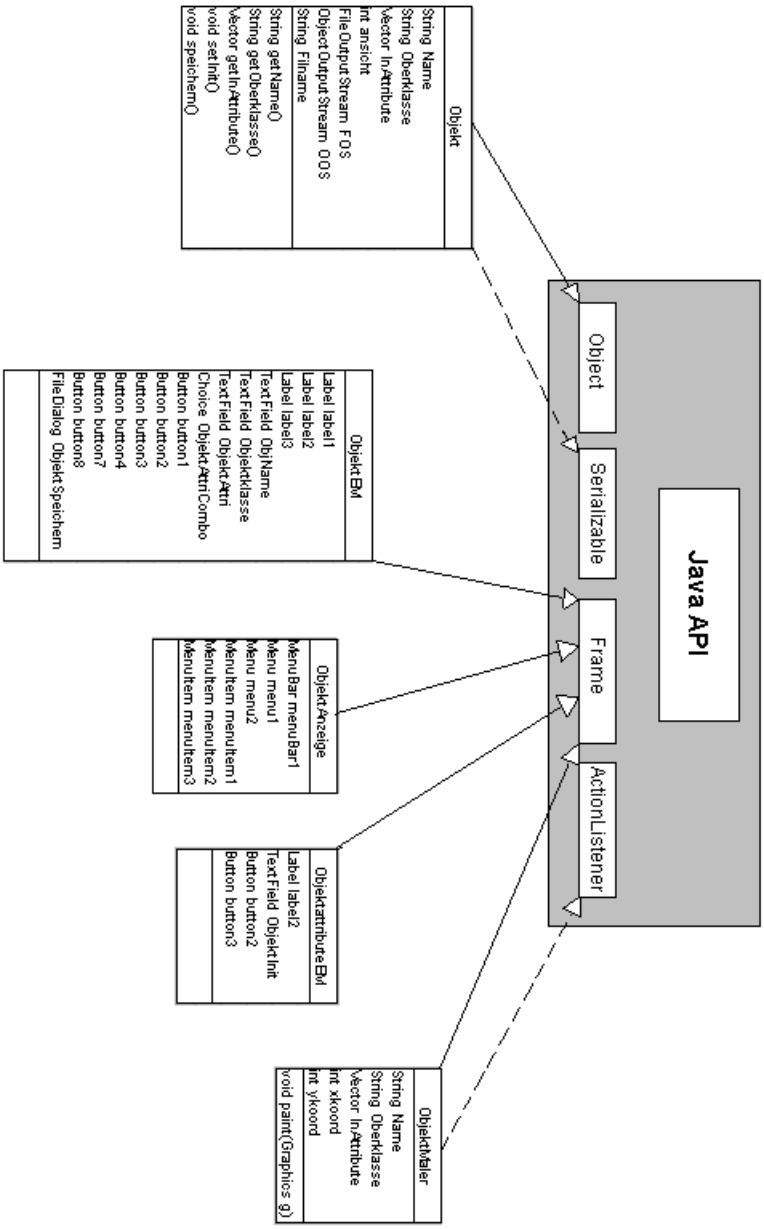


Abbildung 15: Das Klassendiagramm für den Bereich Objekte

9.2 Textuelle Beschreibung der wichtigsten Klassen

9.3 Attribut

Die class *Attribut* repräsentiert den Prototyp für ein Attributobjekt. Die Klasse erbt von der Klasse Object, um Attribut-Instanzen in dem Vector Attribute in der Klasse Klasse speichern zu können. AuSSerdem implementiert sie das Interface Serializable, um innerhalb einer Klassen-Instanz in einer Datei gespeichert werden zu können.

Sie enthält die Attribute Name, Typ, Initialwert und Zusicherung. AuSSerdem sind get- und set-Methoden vorhanden.

9.4 class Operation

Die class *Operation* repräsentiert den Prototyp für ein Operationobjekt. Die Klasse erbt von der Klasse Object, um Operation-Instanzen in dem Vector Operationen in der Klasse Klasse speichern zu können. AuSSerdem implementiert sie das Interface Serializable, um innerhalb einer Klassen-Instanz in einer Datei gespeichert werden zu können.

Sie enthält die Attribute Name, Parameter und Zusicherung. AuSSerdem sind get- und set-Methoden vorhanden.

9.4.1 Klasse

Die class *Klasse* repräsentiert den Prototyp für ein Klassenobjekt. Die Klasse implementiert das Interface Serializable, um ihre Instanzen als ganze in Dateien zu speichern. Es werden der Name der Instanz (Name), der Stereotyp, das packet, in das die Klasse eingebunden ist, und die packet-Klasse sowie die Eigenschaftswerte als String-Variablen im Objekt gespeichert.

Weiterhin wird als boolean-Wert die Information isStatic gespeichert, die festhält, ob die Klasse static ist. Die int-Variable Eigenschaften beinhaltet die Information =FCber die Eigenschaft (public, private, protected) der Klasse. Neben der int-Variable ansicht, die die aktuelle Ansichts-Art (einfach, Standard, komplex) speichert, sind noch zwei Vector-Instanzen, Attribute und Operationen, vorhanden, die die in der Klasse angelegten Attribute bzw. Operationen als Attribut- oder Operation-Instanz speichert.

Der FileOutputStream FOS, der ObjectOutputStream OOS und der String Filename sind für den Speicherungsablauf nötig, wobei der String-Wert den Namen der Datei speichert, in der das Klassenobjekt gespeichert wird.

Es gibt zwei Konstruktoren, von denen der eine eine Klasse mit allen Variablen einzeln initialisiert, der andere als Argument eine Klasse hat und diese kopiert.

Neben den üblichen get- und set-Methoden enthält die class Klasse nur noch die Methode speichern(), die das aktuelle Objekt als Datei im aktuellen Verzeichnis unter dem Namen Filename speichert.

9.4.2 Objekt

Die class *Objekt* repräsentiert den Prototyp für ein Objekt. Sie implementiert das Interface Serializable, um Instanzen als ganzes Objekt mit Hilfe eines ObjectOutputStream speichern zu können.

Es sind Variablen für den Objektnamen (Name), die Oberklasse und, analog zur class Klasse, die Ansicht vorhanden. Darüberhinaus gibt es eine Vector-Instanz InAttribute, welche die initialisierten Attribute speichert. AuSSerdem sind wie in der class Klasse ein FileOutputStream FOS, ein ObjectOutputStream OOS und ein String Filename vorhanden.

Die class Objekt hat einen Konstruktor, der ein Objekt mit allen Variablen initialisiert. Neben den get- und set-Methoden ist nur noch die speichern()-Methode vorhanden, die analog zu der in der class Klasse funktioniert.

9.4.3 Die Menüklassen

Für jede der »Malklassen« gibt es eine eigene Klasse für das zugehörige Menü. Diese erweitern die vorhandene Klasse `MenuBar`. Durch die Methoden `addNewMenuItem` und `add` wird das einfügen neuer Menüpunkte erleichtert.

Gefordert wird in jedem Fall ein `ActionListener`, der sich um die auftretenden Menüereignisse kümmert.

9.4.4 objekt- und klasseMmaler

Diese beiden Klassen funktionieren im wesentlichen gleich. Für jede der eventuell abzubildenden Informationen wird ein String angelegt und initiert, um die Rückgabe von `null` zu vermeiden. Die einzige von außen erreichbare Methode, `malen`, wird von einem Objekt des Typs Klasse bzw. Objekt aufgerufen und setzt zuerst einmal für die oben genannten Strings die entsprechenden Werte ein.

Die dann aufgerufene `paint`-Methode unterscheidet zunächst nach dem Wert von `ausfuehrlich`. Entweder werden nur einige Standardfelder angezeigt, oder auch weitere Informationen in separaten »Kästchen«. Dabei werden potenziell über den (festen) Rand des Rechteckes herausragende Strings umbrochen, indem sie zuerst in einen `AttributedString` verwandelt und dann Stück für Stück mittels eines `LineBreakMeasurer`es in `Layouts` aufgeteilt werden. Diese können dann einfach angezeigt werden. Das umgebende Rechteck wächst dynamisch mit.

9.4.5 useCaseMaler

Diese Klasse unterscheidet sich nicht prinzipiell von den vorhergehenden. Allerdings sind hier fast alle Strings potenziell »zu lang«, so dass die oben beschriebene Prozedur oft wiederholt werden muss. Darunter leidet ein wenig die Performance. Es kann umgeschaltet werden zwischen der für das Usecase-Diagramm wichtigen Ansicht (Ellipsen mit dem Namen in der Mitte) und der ausführlichen textuellen Darstellung.

9.4.6 verbindungsMaler

Der Unterschied zu den anderen Visualisierungs-Klassen besteht in der Verwendung geometrischer Objekte (z.B. Rauten, Linien und Pfeile). Hierfür werden ausschließlich Java »Bordmittel« wie `drawLine` oder `fillPolygon` benutzt. Um die Positionierung einiger Strings relativ zum Pfeil zu vereinheitlichen (rechts- bzw. linksbündige Ausrichtung), benötigt man ein Objekt des Typs `FontMetrics`, welches (die Größe der verwendeten Schrift einbeziehend) zum Beispiel in der Lage ist, die Länge eines Strings zu bestimmen.

9.5 Frame1

Die Klasse Frame1 ist die Main-Klasse unseres Programmes. Wird sie durch eine Java-VM gestartet, so öffnet sich unsere Menu-Bar mit den einzenlen Menüpunkten (siehe Use-Cases). Sie erweitert die Klasse java.awt.Frame. Desweiteren befinden sich noch alle Funktionen des Ladens von Klassen in ihr. Im C.v.K. Mbert - advanced, also im erweiterten U-TOPP Programm wird sie noch mit der Methode der »Projekt-Speicherung« versehen.

9.6 Klasse BildVonUns

Eine einfache Klasse, die die Klasse Frame erweitert und nur zum Anzeigen eines einzigen Bildes (bildvonuns.gif) dient.

9.7 UseCase

Die Klasse UseCase kann durch die Implementierung »Serializable« mit Hilfe eines »SystemOutput- bzw. SystemInput- Streams« gespeichert bzw. geladen werden.

Aufbau der Klasse
Variablen, 2 Konstruktoren (einfach und erweitert, wobei wir den einfachen Konstruktor noch nie gebraucht haben), Methoden um die »privaten« Variablen aus der

Klasse ausgeben zu können. Danach noch die Methode zur Speicherung und zum editieren. Die Speichermethode wird von der UseCaseEM aufgerufen, die Editiermethode aus dem UseCase-Anzeigeframe (UseCase-Maler).

9.8 UseCaseEM

Funktionen
Eingabedaten aufnehmen 3 Buttons für weitere Funktionen: erweiterte Daten in eine zweite Dialogmaske (UseCaseErwmaske) eingeben, Abbrechen, OK zum Speichern und Anzeigen des UseCases

Aufbau
Die Klasse UseCaseEM erweitert die Klasse Frame, d.h. dort werden verschiedene Buttons und Textfields bzw. Textareas zur Eingabe angeordnet, in die der Benutzer später seine Eingaben macht.

Danach folgen 2 Konstruktoren. Der erste Konstruktor sorgt dafür, dass eine leere Eingabemaske erscheint. Der zweite tritt beim Laden eines UseCases in Kraft und füllt die ganzen Eingabefelder mit den Informationen, die er von dem geladenen UseCase erhält. Danach kommen die ganzen notwendigen Variablen, teilweise auch Variablen, die notwendig sind um die erweiterten Eingaben aus der UseCaseErwmaske zu speichern und später in die erzeugte Instanz des UseCases zu integrieren.

Danach werden die Buttons mit Funktionen/Methoden versehen. Jede Methode zu besschreiben wäre an dieser Stelle zu aufwendig. Aufgrund von technischen Problemen mussten wir an einigen Stellen etwas tricksen, so dass teilweise die Lesbarkeit des Programmes ein wenig gelitten hat.

9.9 UseCaseErwmaske

Die Klasse UseCaseErwmaske hat im Prinzip die gleichen Aufgaben, wie die UseCaseEM, nur mit dem Unterschied, dass diese Maske nur von der UseCaseEM aufgerufen werden kann und man bei der Bestätigung der Eingaben (»OK«) nur zu der UseCaseEM wieder zurückkehren kann, wobei die Variablen in der UseCaseEM mit den erweiterten Eingaben gefüllt werden. Der Aufbau entspricht aber im wesentlichen der Klasse UseCaseEM.

9.10 Verbindung

Die Klasse Verbindung kann durch die Implementierung »Serializable« mit Hilfe eines »SystemOutput- bzw. SystemInput- Streams« gespeichert bzw. geladen wer-

den.

Aufbau
Variablen, 1 Konstruktor, Methoden um die »privaten« Variablen aus der Klasse ausgeben zu können. Danach noch die Methode zur Speicherung und zum editieren. Die Speichermethode wird von der VerbindungEM aufgerufen, die Editiermethode aus dem Verbindungs-Anzeigeframe (Verbindungs-Maler).

9.11 VerbindungEM

Funktionen
Eingabedaten aufnehmen 3 Buttons für weitere Funktionen: Layout-Daten in eine zweite Dialogmaske (VerbindungsLayoutEM) eingeben, Abbrechen, OK zum Speichern und Anzeigen der Verbindung

Aufbau
Die Klasse VerbindungEM erweitert die Klasse Frame, d.h. dort werden verschiedene Buttons und Textfields bzw. Textareas zur Eingabe angeordnet, in die der Benutzer später seine Eingaben macht. Danach folgen 2 Konstruktoren. Der erste Konstruktor sorgt dafür, dass eine leere Eingabemaske erscheint. Der zweite tritt beim Laden einer Verbindung in Kraft und füllt die ganzen Eingabefelder mit den Informationen, die er von der geladenen Verbindung erhält. Danach kommen die ganzen notwendigen Variablen, teilweise auch Variablen, die notwendig sind um die Layout-Eingaben aus der VerbindungsLayoutEM zu speichern und später in die erzeugte Instanz der Verbindung zu integrieren. Danach werden die Buttons mit Funktionen/Methoden versehen. Jede Methode zu besschreiben wäre an dieser Stelle zu aufwendig.

9.12 VerbindungsLayoutEM

Die Klasse VerbindungsLayoutEM hat im Prinzip die gleichen Aufgaben, wie die VerbindungEM nur mit dem Unterschied, dass diese Maske nur von der VerbindungEM aufgerufen werden kann und man bei der Bestätigung der Eingaben (»OK«) nur zu der VerbindungEM wieder zurückkehren kann, wobei die Variablen in der VerbindungEM mit den Layout-Eingaben gefüllt werden. Der Aufbau entspricht aber im wesentlichen der Klasse VerbindungEM.

9.13 Pflichtenheft

Die Klasse Pflichtenheft kann durch die Implementierung »Serializable« mit Hilfe eines »SystemOutput- bzw. SystemInput- Streams« gespeichert bzw. geladen werden. Die Zunächst angegebenen Variablen sind für die Instanziierung der Klasse notwendig, danach kommt der eigentliche Konstruktor und danach die Methode zur Speicherung. »get und set Methoden« wurden in dieser Klasse nicht gebraucht, weil die gesamten Variablen public sind und man ohne Einschränkungen auf sie zugreifen kann. Diese Sicherheitslücke haben wir in den Klassen: Klasse, Objekt, Use-Case und Verbindung schon behoben.

9.14 QuitDialog

Auch eine Erweiterung der Klasse Frame. Sie stellt eine Sicherheitsabfrage, bevor das Programm vom User beendet werden kann.

9.15 PflichtenheftEM

Die Klasse PflichtenheftEM erweitert die Klasse Frame, d.h. dort werden verschiedene Buttons und Textfields bzw. Textareas zur Eingabe angeordnet, in die der Benutzer später seine Eingaben macht. Die Klasse enthält insgesamt 4 wichtige Funktionen.

1. Die Buttons der einzelnen Eingaben-Bereiche können angewält werden, wodurch sich die Eingabemaske modifiziert, d.h. die Textfelder des angewählten Bereiches erscheint.

2. Der Button »HTML-erzeugen«. Wird dieser gedrückt, wird (bisher noch ohne Meldung) die Datei »Pflichtenheft.html« neu erzeugt, in welche dann alle bisherigen Eingaben als HTML-Code geschrieben wird.

3. Der »Abbrechena« Button schliesst die Eingabemaske

4. Der »OK« Button öffnet die Speicherungsmaske

Die Klasse Pflichtenheft besteht aus allen Strings und Variablen, die zum Speichen und Laden nötig sind, damit eingegebene Daten nicht verloren gehen. Die Klasse Pflichtenheft ist folgendermaSSen aufgebaut: Zunächst wird das gesamte Layout mit allen Buttons und Textfeldern des Frames festgelegt. Danach kommen 2 Konstruktoren, die notwenig waren, denn einmal soll beim UseCase: »Pflichtenheft erstellen« eine leere Pflichtenhefteingabemaske erscheinen, beim UseCase: »Pflichtenheft laden« die Pflichtenheftmaske mit den gespeicherten Daten gefüllt werden.

Danach kommen die Variablen und einige Methoden, die die GröSSe des Fensters festlegen und die zum Schliessen des Fensters notwendig sind. Danach werden die Buttons mit Funktionen/Methoden versehen (die wichtigen vier, oben genannt). Jede Methode zu besschreiben wäre an dieser Stelle zu aufwändig.

9.16 Beteiligte

C Carolin Woltermann carowo@uni-muenster.de

v Verena Kutschker veku@uni-muenster.de

K Karsten Wantia karsten.wantia@math.uni-muenster.de

M Marc Rogge mrogge@muenster.de

b Birga Schwartz birgasch@uni-muenster.de

e Emin Bahdir bahadir@math.uni-muenster.de

r Ralph Carrie carrie@uni-muenster.de

t ???

10 Benutzerhandbuch

10.1 Einführung

Das Programm U-Topp ist im Rahmen des Programmierpraktikums im Wintersemester 2000/2001 am Fachbereich Mathematik und Informatik der Westfälischen Wilhelms-Universität Münster entstanden. Ziel war es, ein Tool zu erstellen, welches in der Lage ist der Softwareentwickler in der Planungsphase zu unterstützen un dadurch die Planungsphase übersichtlicher und klarer zu gestalten. Weiterhin ist es vorteilhaft bei einem Entwicklerteam Datenstrukturen, Objekte, Speicherabläufe und Attribute für die einzelnen Personen des Entwicklerteams offen zu legen und somit eine erfolgreiche Entwicklung zu gerantieren. Die Grundfunktionalität dieses Tools besteht also darin, dem Benutzer bei der Anlegung von Objekten, Klassen, Verbindungen, Use Cases und einem Pflichtenheft Grundgerüste zur Verfügung zu stellen, die leicht zu bedienen sind und zu einer möglichst großen Effizienz führen. So kann der Benutzer dieses Tools beispielsweise durch die Nutzung der Eingabemasken auf die wichtigsten Funktionen zurückgreifen, die in der Planungsphase benötigt werden. Dieses Tool erhebt keinerlei Anspruch auf Vollständigkeit, unterstützt allerdings die Basisfunktionen.

10.2 Programmstart

Mit dem Aufruf des Programms in der Eingabeaufforderung oder in einer speziellen Startumgebung wird das Tool U-Topp aufgerufen. Der Benutzer hat nun die Möglich keit innerhalb der nun erscheinen den Menüleiste verschiedene Interaktionen durchzuführen.

10.3 Anlegen neuer Objekte

Das Anlegen neuer Objekt geschieht jeweils unter den dafür vorgesehenen Menüpunkten(Objekt, Klasse, Verbindung, Use Case, Pflichtenheft). Unter diesen Menüpunkten wählt man jeweils 'hinzufügen' aus. Es erscheint dann eine Eingabemaske, in die der Benutzer die gewünschten Daten eintragen, Radiobuttons markieren oder aus Scrolleisten auswählen kann. Klickt der Benutzer abschließend auf das 'OK' Button der jeweiligen Eingabemaske, so erscheint ein Fenster, in dem der Benutzer einen Pfad auswählen kann, unter welchem er sein Objekt zu speichern wünscht. Nach der Wahl eines geeigneten Namens kann das Objekt dann gespeichert werden.

10.4 Aufrufen vorhandener Objekte

Das Aufrufen von schon existiernden Objekten geschieht-ähnlich wie beim Anlegendurch die Wahl eines Menüpunktes(Objekte, Klassen, Verbindungen, Use Cases und Pflichtenheft). Unter diesem Menüpunkt ist jedoch dann die Option »laden« auszuwählen. In dem erscheinenden Fenster kann man dann das gewünschte Objekt auswählen, welches dann angezeigt wird.

10.5 Verlassen des Programms

Das Programm ist über 'Programm → beenden' zu beenden