Westfälische Wilhelms-Universität Münster
Fachbereich Mathematik und Informatik
Programmierpraktikum WS 2000/2001
Dozent: Dr. Dietmar Lammers

Pflichtenheft

zum UML-Tool des Programmierpraktikums

Projektgruppe **SynergieSoft**

Baumann, Nadine	nbauman@uni-muenster.de
Debald, Tobias	debald@uni-muenster.de
Hartmann, Uta	hartmau@math.uni-muenster.de
Lohmeyer, Henning	lohmeyh@uni-muenster.de
Sahlmann, Arnd	sahlmann@uni-muenster.de
Starke, Alexander	alstar@uni-muenster.de
Wennmacher, Stefanie	wennmac@muenster.de

Inhaltsverzeichnis

1. Zielbestimmung

U-Topp ist ein Tool, das Anwendungsentwickler bei der Entwicklung von Software unterstützen will. U-Topp hilft beim Aufbau des entsprechenden Modells und geht hierbei von einer Modellierung mit der Unified Modelling Language (UML) aus.

In einer ersten Version des Programms soll es möglich sein, einzelne Komponenten von UML zu **erfassen**. Man gibt die Angaben zu diesen Komponenten über Erfassungsmasken ein und erhält hierbei von U-Topp kontextbezogene Vorschläge oder Auswahlmöglichkeiten. Als erfassbare UML-Einzelkomponenten sind Klassen, Use-Cases und Verbindungen vorgesehen. Der Anwendungsentwickler soll außerdem bei der Erstellung eines Pflichtenheftes unterstützt werden.

Die so erfassten UML-Komponenten können in dieser Version auch einzeln **graphisch dargestellt** werden. In weiteren Versionen soll das Programm aber auch in der Lage sein, die aus der UML bekannten Diagramme (Klassen-, Use-Case-, Sequenzdiagramme usw.) darstellen zu können.

Dem Benutzer von U-Topp soll es dann zudem ermöglicht werden, **interaktiv** mit der graphischen Darstellung einer UML-Komponente arbeiten zu können. Das heißt z. B., dass sich bei Doppelklick auf eine solche Graphik ein entsprechendes Dialogfenster mit genaueren Angaben zu diesem Objekt öffnet und ein Editieren ermöglicht wird.

Anmerkung: Die im Pflichtenheft vorkommenden Bezeichnungen „U-Topp“, „UTopp“ und „UTopp2001“ meinen jeweils das gleiche Programm.

1.1 Musskriterien

Die Komponenten müssen so realisiert werden, dass sie einfach in ein späteres Gesamttool eingebunden werden können. Komponentendaten, die später über ziehen mit der Maus eingetragen werden sollen (z. B. Start- und Zielklasse einer Verbindung), werden hier nicht erfasst. Die Speichermöglichkeit muss dafür aber vorgesehen werden. Folgende Komponenten müssen realisiert werden:

1. Use-Case
2. Klasse/ Objekt
3. Verbindung (auch Vererbung)
4. Pflichtenheft (ohne Use-Cases und Klassendiagramme etc.)

1.2 Wunschkriterien

1. Speicherung der Komponenten in einer durch XML definierten Sprache.
2. Fast alle Komponenten der UML sind nicht voll ausdefiniert, deswegen sollten immer frei wählbare Datenzusätze eingegeben werden können.
3. Es gibt bei den UML-Komponenten immer zentrale Daten (Use-Case: Name, Akteur, Ablauf). Schön wäre es, wenn zur besseren Übersichtlichkeit erst diese Daten abgefragt würden, und nur auf Wahl eine sehr umfangreiche Eingabemaske erscheint.
4. Die graphische Darstellung von Klassen und Objekten kann ja mehr oder weniger ausführlich sein. Es wäre nett, wenn es dementsprechend auch mehrere Darstellungsformen geben würde.

1.3 Abgrenzungskriterien

Zu diesem Zeitpunkt sollte kein vollständiges UML-Tool erstellt werden. Der Präsentations- und Bearbeitungsrahmen sollte so simpel wie möglich ausfallen.

2. Produkteinsatz

2.1 Anwendungsbereiche

- Softwareentwicklung

2.2 Zielgruppen

- Softwareentwickler

2.3 Betriebsbedingungen

- Es ist davon auszugehen, dass die Benutzer über einige Erfahrung in Einsatz und Betrieb von Software verfügen, und ggf. auch über einige weitere Zusatzsoftware verfügen bzw. bereit sind, diese zu beschaffen/ zu installieren.

3. Produktumgebung

3.1 Software

- Eine Java-VM sollte bereitstehen.

3.2 Hardware

- Hinreichend leistungsfähig.

3.3 Orgware

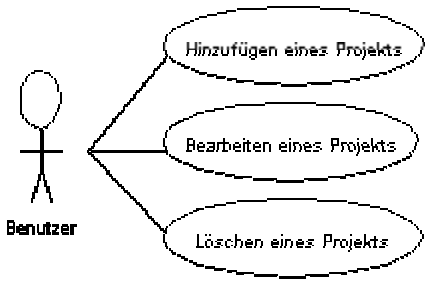
- -

3.4 Schnittstellen

- -

4. Produktinformationen

Projekt



4.1 Hinzufügen

Akteur	Benutzer
Vorbedingung	-
Nachbedingung	Neues Projekt hinzugefügt.
Ablaufbeschreibung	Die Erfassungsmaske erscheint. Es werden die Projekt-Daten eingegeben.
Ausnahmen, Fehlersituationen	1. Falls der Projektname nicht eingegeben wurde, fordert das Programm den Benutzer zur Eingabe der fehlenden Daten auf. 2. Falls ein Projektname bereits besteht, fragt das Programm den User, ob bestehende Daten überschrieben werden sollen.
Variationen	Bei durch den User erzeugtem Abbruch werden die bereits erfassten Daten verworfen.
Dialogbeispiel	U-Topp-Hauptfenster und Projekt-Erfassungsmaske

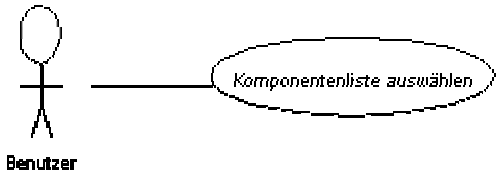
4.2 Bearbeiten

Akteur	Benutzer
Vorbedingung	Es ist ein Projekt vorhanden.
Nachbedingung	Projektdaten editiert.
Invarianten	Projektname vorhanden.
Ablaufbeschreibung	Der Editier-Dialog erscheint. Die Daten können nun verändert werden.
Ausnahmen, Fehlersituationen	1. Falls der Projektname nach dem Editieren nicht mehr vorhanden ist, wird der Benutzer aufgefordert einen Projektnamen einzugeben. 2. Wenn der Projektname gleich dem eines bereits vorhandenen Projektes ist oder der Projektname nicht verändert wurde fragt das Programm den User, ob die bestehenden Daten verändert werden sollen.
Variation	Bei durch den User erzeugten Abbruch werden die bereits erfassten Daten verworfen.
Dialogbeispiel	U-Topp-Hauptfenster und Projekt-Erfassungsmaske

4.3 Löschen

Akteur	Benutzer
Vorbedingung	Es ist ein Projekt vorhanden.
Nachbedingung	Projekt gelöscht.
Ablaufbeschreibung	Der Anwender wählt ein zu löschendes Projekt aus.
Ausnahmen, Fehlersituationen	Kein Projekt ausgewählt. Es erscheint eine Fehlermeldung.
Variation	Abbruch der Operation.
Dialogbeispiel	U-Topp-Hauptfenster und Projekt-Erfassungsmaske

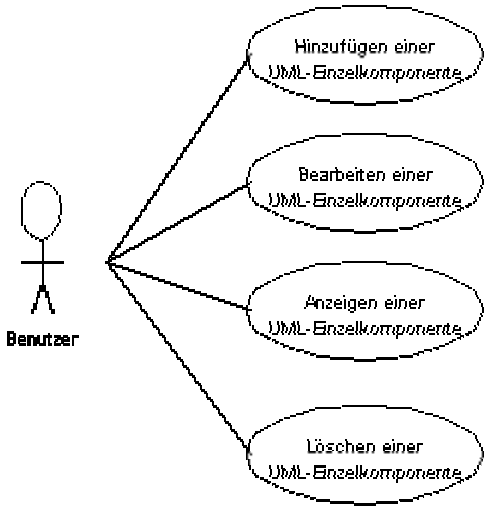
Komponentenliste



4.4 Auswählen

Akteur	Benutzer
Vorbedingung	Projekt vorhanden.
Nachbedingung	Komponenten aufgelistet.
Ablaufbeschreibung	Die anzuzeigende Komponente wird vom Benutzer ausgewählt.
Ausnahmen, Fehlersituationen	-
Dialogbeispiel	Projektfenster mit Komponentenauswahlleiste.

Komponenten



4.5 Hinzufügen

Akteur	Benutzer
Vorbedingung	Es ist ein Projekt vorhanden, dies kann auch ein neues, leeres sein.
Nachbedingung	Neue UML-Komponente ist dem Projekt hinzugefügt.
Ablaufbeschreibung	Eine UML-Komponente wird ausgewählt. In einer UML-Komponenten spezifischen Erfassungsmaske werden die Komponenten-Daten eingegeben.
Ausnahmen, Fehlersituationen	Falls benötigte Daten nicht eingegeben wurde, fordert das Programm den Benutzer zur Eingabe der fehlenden Daten auf. Falls kritische Daten der UML-Komponente schon vorhanden sind (bspw. doppelte Klassen- oder Pflichtenheft-Namen) fragt das Programm den User, ob die bestehende Komponente ersetzt werden soll.
Variationen	Bei durch den User erzeugtem Abbruch werden die bereits erfassten Daten verworfen.
Dialogbeispiel	Projektfenster zur Auswahl eines UML-Komponententyps und zum Hinzufügen dieser Komponente. Dialoge zu den UML-Komponenten Klasse, Objekt, Use-Case, Verbindungen, Pflichtenheft

4.6 Bearbeiten

Akteur	Benutzer
Vorbedingung	Eine UML-Komponente ist vorhanden.
Nachbedingung	UML-Komponenten-Daten wurden geändert.
Ablaufbeschreibung	Die zu bearbeitende UML-Komponente wird ausgewählt. Ein UML-Komponenten entsprechender Editier-Dialog erscheint. Die Daten können nun verändert werden.
Ausnahmen, Fehlersituationen	1. Falls für die UML-Komponente zwingend benötigte Daten (Komponentenname etc.) fehlen, wird der Benutzer aufgefordert, die fehlenden Daten zu ergänzen.

	2. Falls der Komponentenname bereits besteht, wird vor Änderung der Daten eine Bestätigung des Benutzers verlangt. 3. Es ist keine UML-Komponente ausgewählt. Es erscheint eine Fehlermeldung.
Variationen	Bei durch den User erzeugtem Abbruch werden die neu erfassten Daten verworfen.
Dialogbeispiel	Projektfenster zur Auswahl eines UML-Komponententyps und zum Hinzufügen dieser Komponente. Dialoge zu den UML-Komponenten Klasse, Objekt, Use-Case, Verbindungen, Pflichtenheft

4.7 Anzeigen

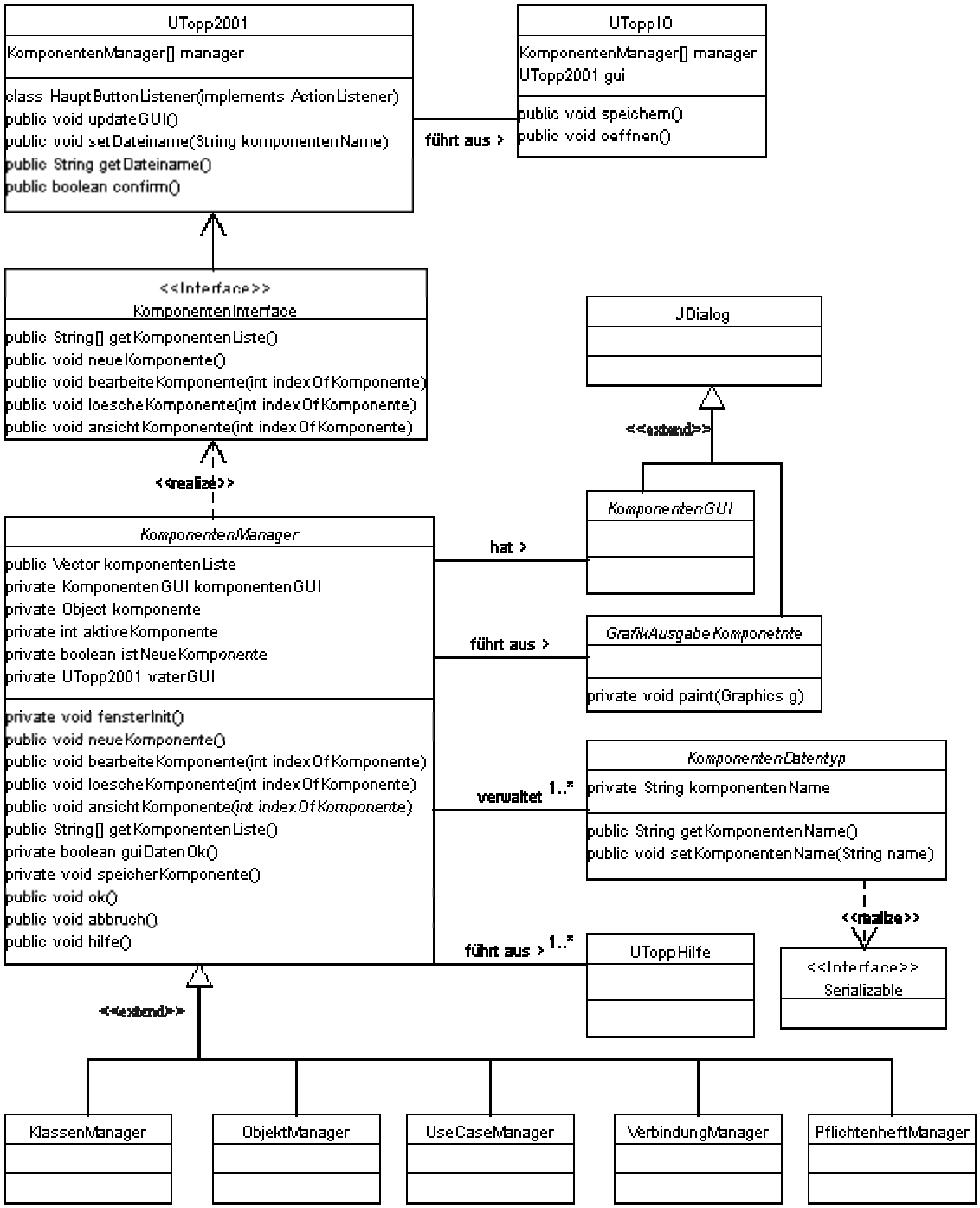
Akteur	Benutzer
Vorbedingung	Eine UML-Komponente ist vorhanden.
Nachbedingung	Die Komponente wird graphisch dargestellt.
Ablaufbeschreibung	Die anzuzeigende Komponente wird vom Benutzer ausgewählt.
Ausnahmen, Fehlersituationen	Es ist keine Komponente ausgewählt. Es erscheint eine Fehlermeldung
Dialogbeispiel	Projektfenster zur Auswahl eines UML-Komponententyps und zum Hinzufügen dieser Komponente. Dialoge zu den UML-Komponenten Klasse, Objekt, Use-Case, Verbindungen, Pflichtenheft

4.8 Löschen

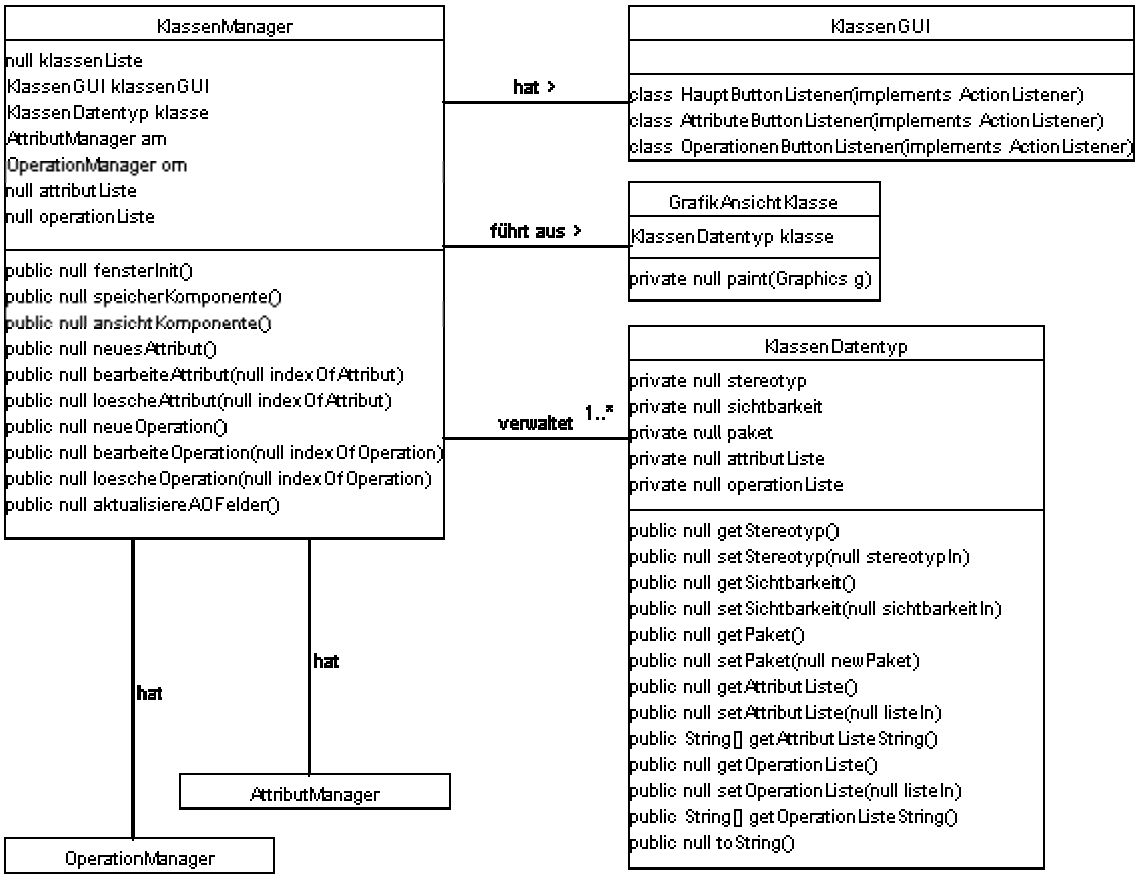
Akteur	Benutzer
Vorbedingung	Es ist eine UML-Komponente vorhanden.
Nachbedingung	UML-Komponente ist aus dem Projekt entfernt.
Ablaufbeschreibung	Der Anwender wählt eine zu löschende Komponente aus.
Ausnahmen, Fehlersituationen	Keine Komponente ist ausgewählt. Es erscheint eine Fehlermeldung.
Services	Bestätigungsdialog bevor Komponente gelöscht wird.
Dialogbeispiel	Projektfenster zur Auswahl eines UML-Komponententyps und zum Hinzufügen dieser Komponente. Dialoge zu den UML-Komponenten Klasse, Objekt, Use-Case, Verbindungen, Pflichtenheft

5. Fachkonzeptklassen und Produktdaten

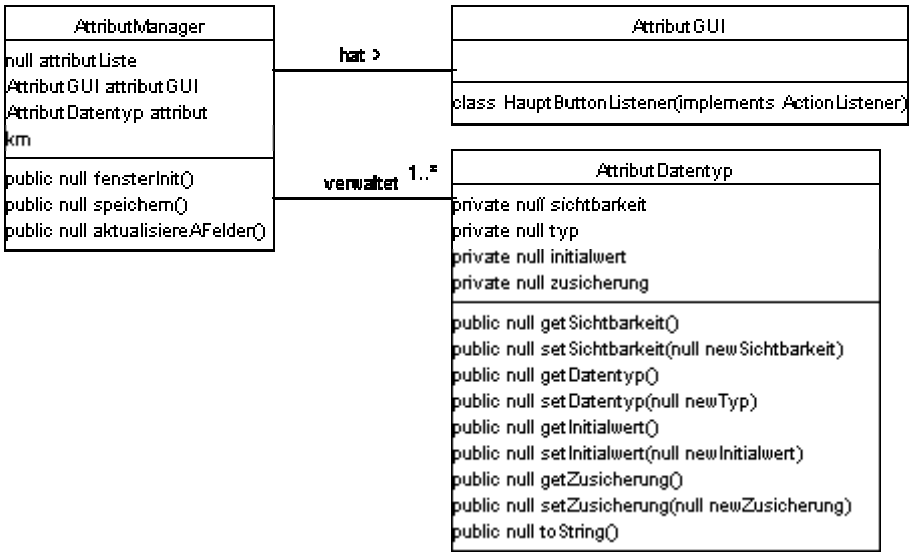
5.1 UTopp



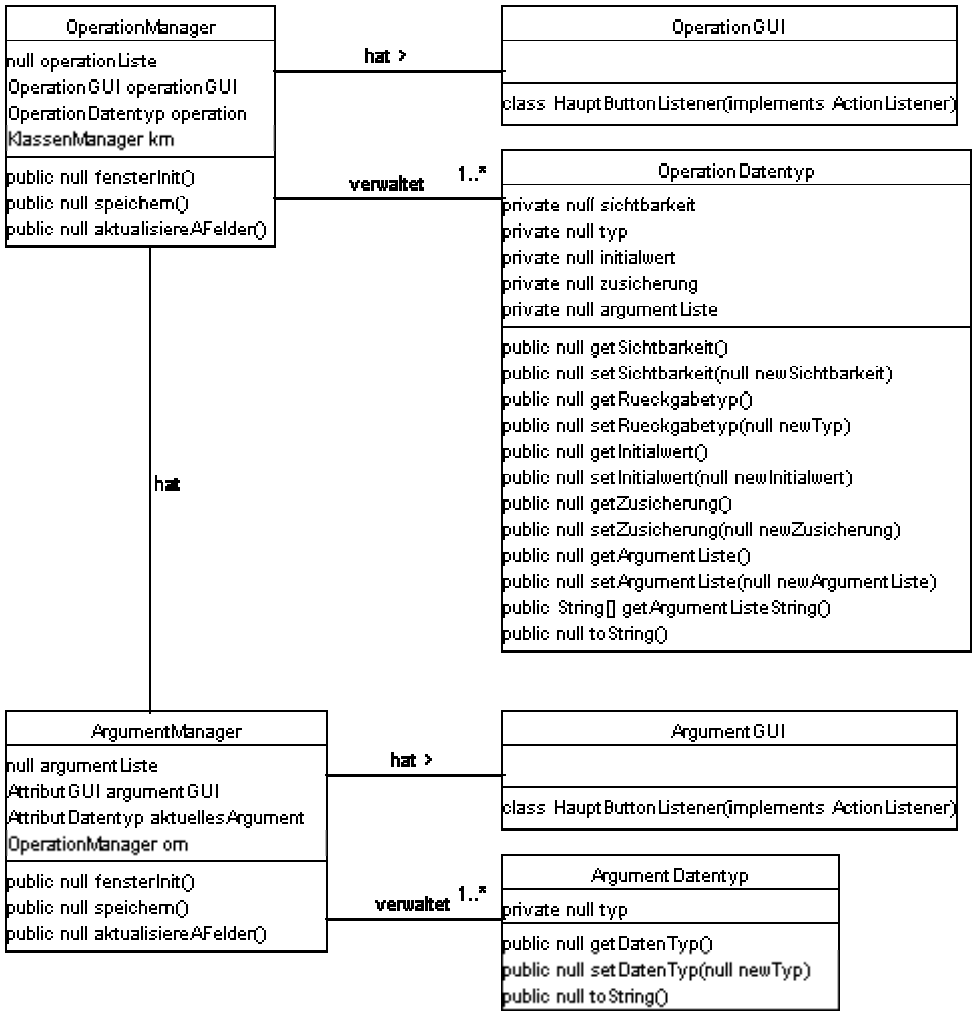
5.2 Klasse



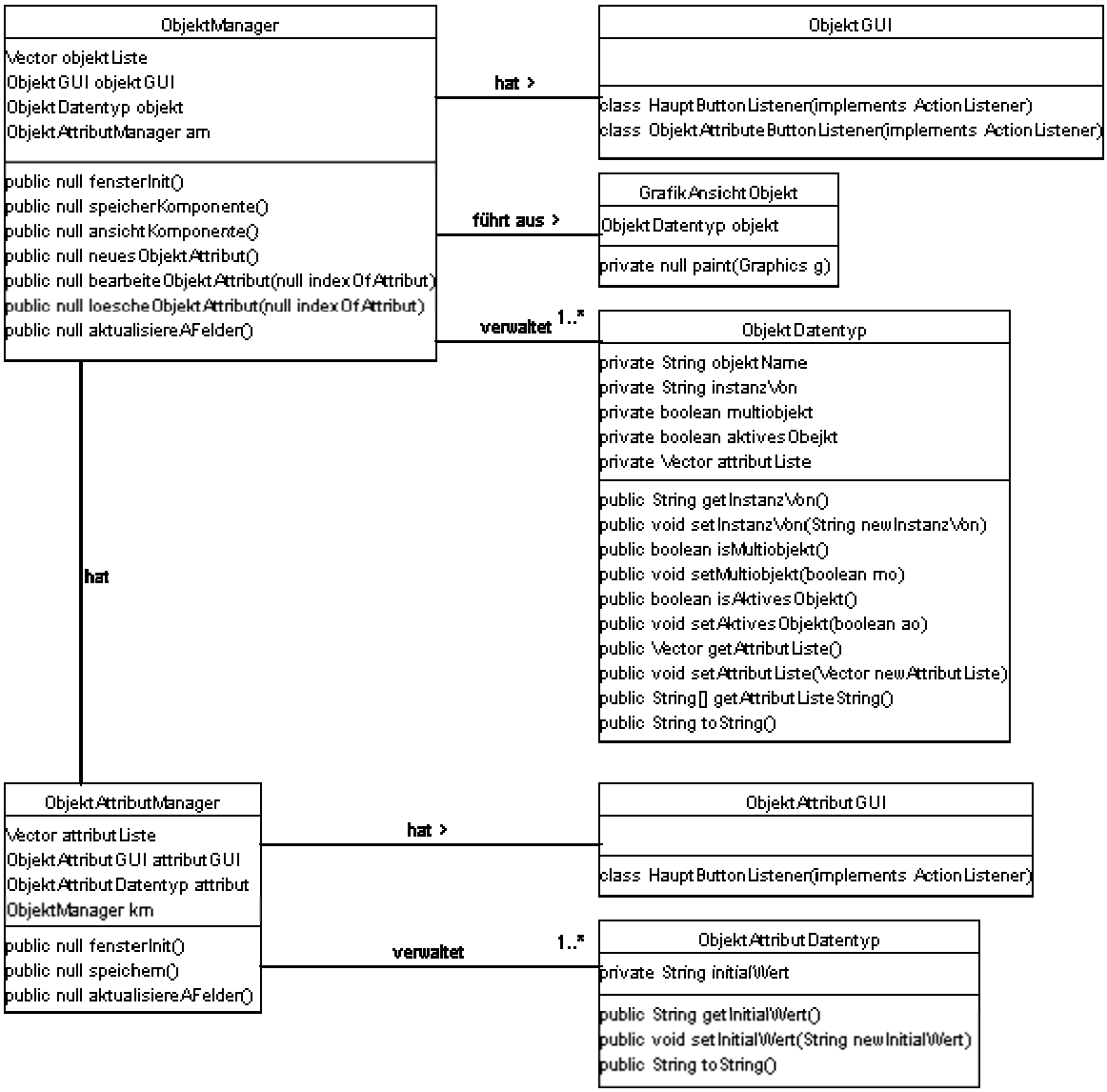
5.2.1 Klassenattribute



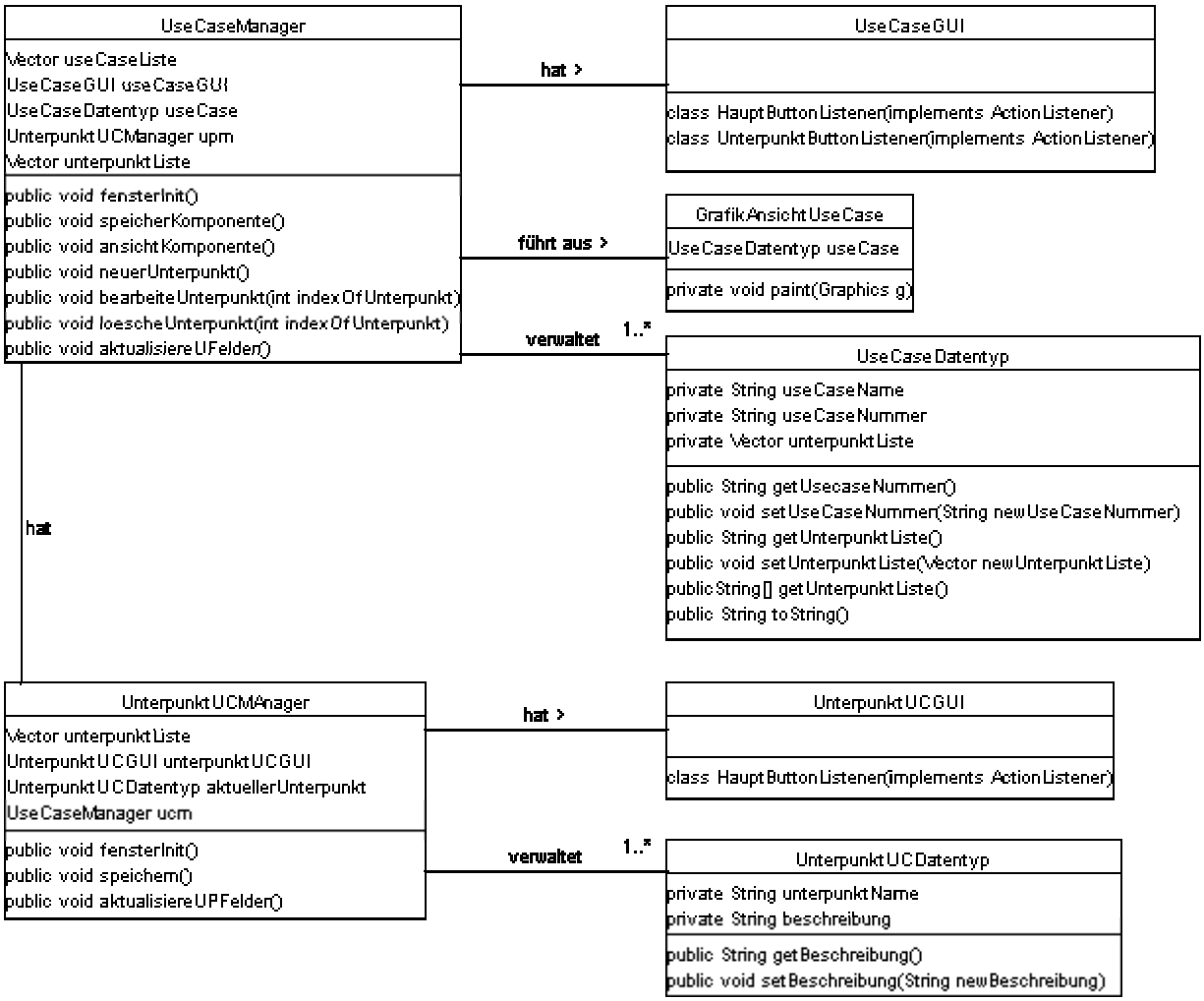
5.2.2 Klassenoperationen



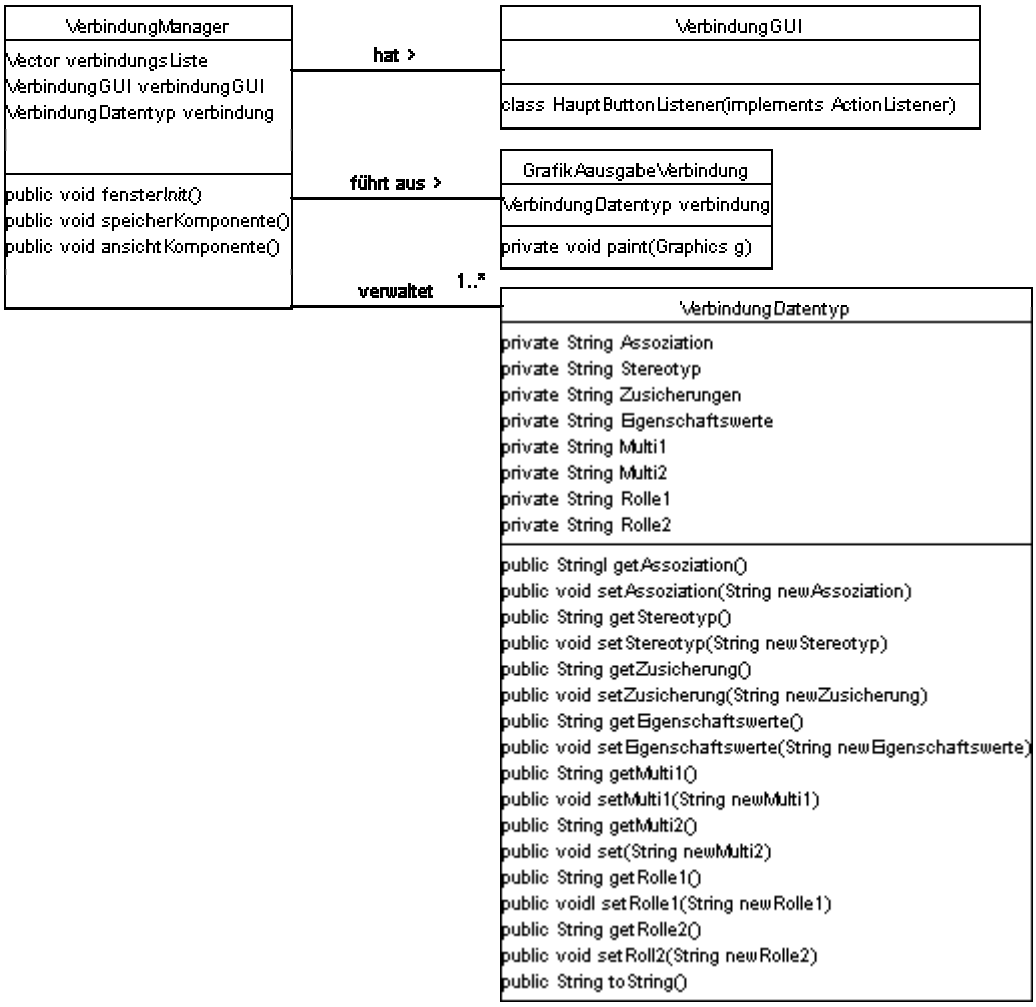
5.3 Objekt


ObjektManager
Vector objektListe
ObjektGUI objektGUI
ObjektDatentyp objekt
ObjektAttributManager am
public null fensterInit()
public null speicherKomponente()
public null ansichtKomponente()
public null neuesObjektAttribut()
public null bearbeiteObjektAttribut(null indexOfAttribut)
public null loescheObjektAttribut(null indexOfAttribut)
public null aktualisiereAFelder()
hat >
ObjektGUI
class HauptButtonListener(implements ActionListener)
class ObjektAttributeButtonListener(implements ActionListener)
führt aus >
GrafikAnsichtObjekt
ObjektDatentyp objekt
private null paint(Graphics g)
verwaltet
1..*
ObjektDatentyp
private String objektName
private String instanzVon
private boolean multiobjekt
private boolean aktivesObejkt
private Vector attributListe
public String getInstanzVon()
public void setInstanzVon(String newInstanzVon)
public boolean isMultiobjekt()
public void setMultiobjekt(boolean mo)
public boolean isAktivesObjekt()
public void setAktivesObjekt(boolean ao)
public Vector getAttributListe()
public void setAttributListe(Vector newAttributListe)
public String[] getAttributListeString()
public String toString()
hat
ObjektAttributManager
Vector attributListe
ObjektAttributGUI attributGUI
ObjektAttributDatentyp attribut
ObjektManager km
public null fensterInit()
public null speichern()
public null aktualisiereAFelder()
hat >
ObjektAttributGUI
class HauptButtonListener(implements ActionListener)
verwaltet
1..*
ObjektAttributDatentyp
private String initialWert
public String getInitialWert()
public void setInitialWert(String newInitialWert)
public String toString()

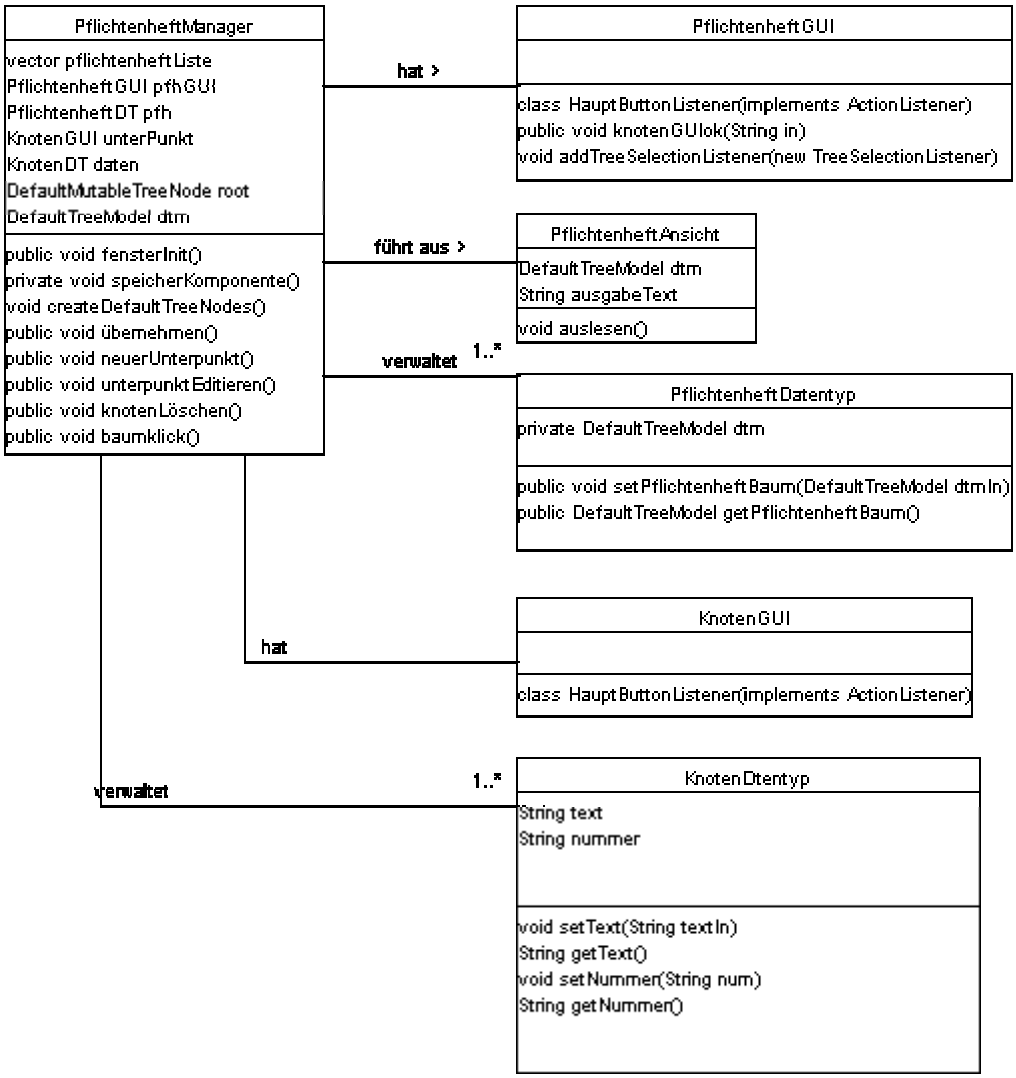
5.4 Use-Case


UseCaseManager
Vector useCaseListe
UseCaseGUI useCaseGUI
UseCaseDatentyp useCase
UnterpunktUCManager upm
Vector unterpunktListe
public void fensterInit()
public void speicherKomponente()
public void ansichtKomponente()
public void neuerUnterpunkt()
public void bearbeiteUnterpunkt(int indexOfUnterpunkt)
public void loescheUnterpunkt(int indexOfUnterpunkt)
public void aktualisiereUFelder()
hat >
UseCaseGUI
class HauptButtonListener(implements ActionListener)
class UnterpunktButtonListener(implements ActionListener)
führt aus >
GrafikAnsichtUseCase
UseCaseDatentyp useCase
private void paint(Graphics g)
verwaltet
1..*
UseCaseDatentyp
private String useCaseName
private String useCaseNummer
private Vector unterpunktListe
public String getUsecaseNummer()
public void setUseCaseNummer(String newUseCaseNummer)
public String getUnterpunktListe()
public void setUnterpunktListe(Vector newUnterpunktListe)
public String[] getUnterpunktListe()
public String toString()
hat
UnterpunktUCManager
Vector unterpunktListe
UnterpunktUCGUI unterpunktUCGUI
UnterpunktUCDatentyp aktuellerUnterpunkt
UseCaseManager ucm
public void fensterInit()
public void speichern()
public void aktualisiereUPFelder()
hat >
UnterpunktUCGUI
class HauptButtonListener(implements ActionListener)
verwaltet
1..*
UnterpunktUCDatentyp
private String unterpunktName
private String beschreibung
public String getBeschreibung()
public void setBeschreibung(String newBeschreibung)

5.5 Verbindung



5.6 Pflichtenheft



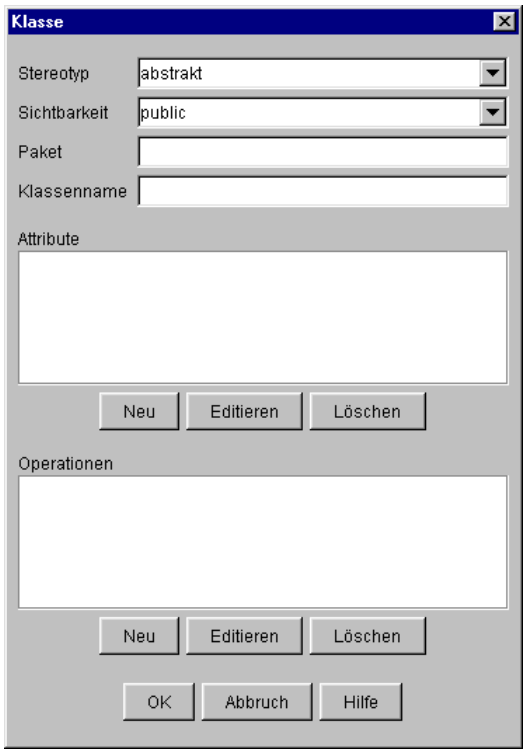
6. Produktleistungen

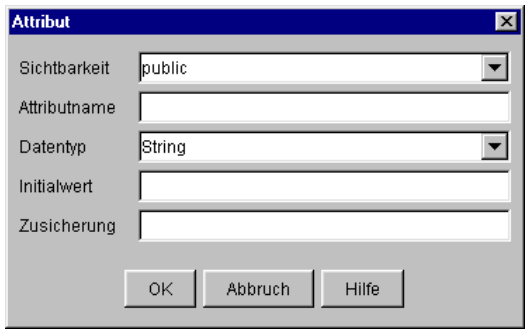
Die Produktleistungen sind den Produktinformationen unter Punkt 4 zu entnehmen.

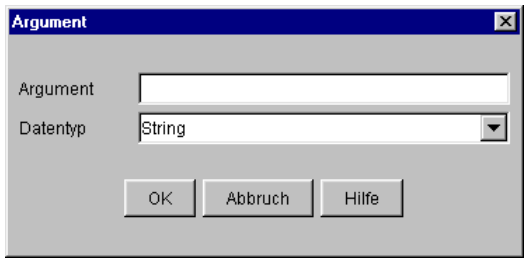
7. Benutzeroberfläche

7.1 Projekt

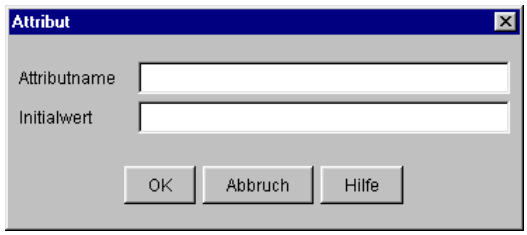
7.2 Klasse



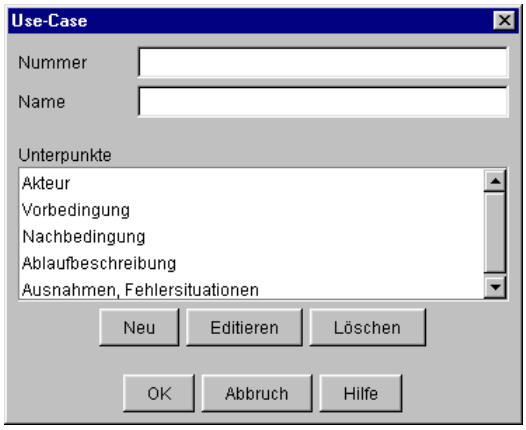


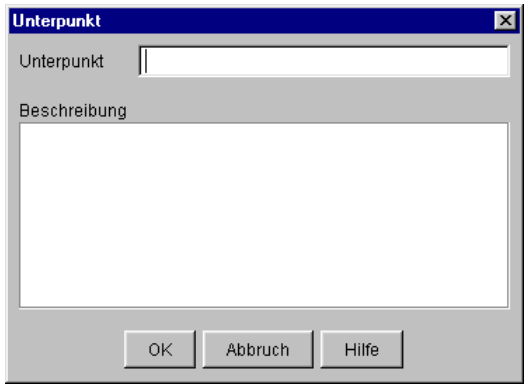


7.3 Objekt

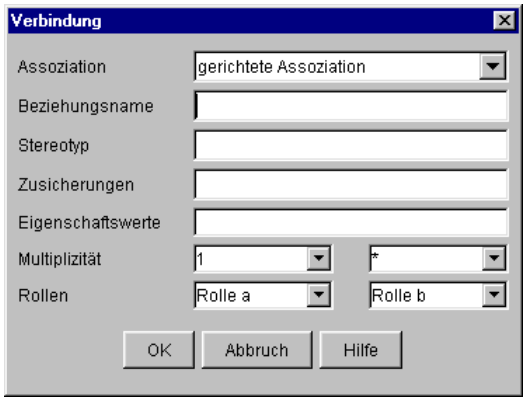


7.4 Use-Cases

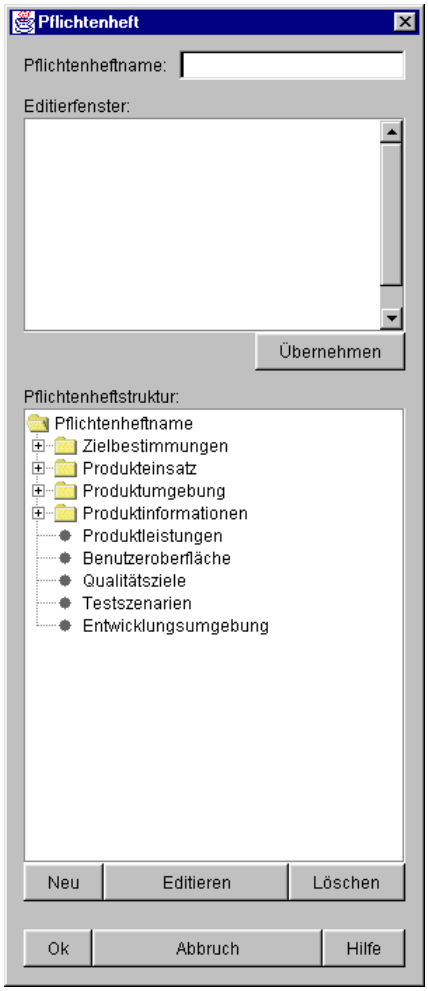


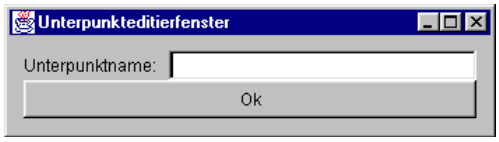


7.5 Verbindungen



7.6 Pflichtenheft





8. Qualitätsziele

Ziel	++	+	+-	-
Funktionalität		X		
Zuverlässigkeit			X	
Benutzbarkeit		X		
Effizienz			X	
Änderbarkeit	X			

9. Testszenarien/Testfälle

/TF110/ Speicherung einer neu angelegten Datei
Es wird geprüft, ob sich ein neu angelegtes Projekt abspeichern lässt.

/TF120/ Öffnen einer vorhandenen Datei
1 - Es wird überprüft, ob sich ein abgespeichertes Projekt öffnen lässt.
2 - Daraufhin wird geprüft, ob die erwarteten, zuvor abgespeicherten Daten auch alle vorhanden sind.

/TF130/ Schließen der vorhandenen Datei
1 - Es wird geprüft, ob sich das Programm schließen lässt und
2 - ob gegebenenfalls eine Abfrage zur Speicherung ungesicherter Änderungen erfolgt.

/TF140/ Aufruf der Hilfe-Datei
Getestet wird, ob sich das kontextbezogene HTML-Hilfe-Dokument öffnet.

/TF210/ Anlegen eines neuen Objekts
1 - Es wird eine neue UML-Komponente "Objekt" angelegt und überprüft, ob das Objekt mit den eingegebenen Daten korrekt angelegt wird.
2 - Es wird außerdem überprüft, ob ein Objekt mindestens einen Namen haben muss, um angelegt werden zu können.
3 - Zudem wird nachgeprüft, ob auch wirklich nur einmal vorkommende Namen akzeptiert werden.
4 - Geprüft wird, ob sich die vorangehenden Testfallunterpunkte 1 bis 3 auf Objektattribute übertragen lassen.

/TF220/ Bearbeiten eines Objekts
Es wird ein Objekt ausgewählt und überprüft, ob das Objekt nach der Änderung korrekt abgespeichert ist.

/TF230/ Löschen eines Objekts
Es wird ein Objekt gelöscht und überprüft, ob das Objekt vollständig gelöscht wurde.

/TF240/ Ansicht eines Objekts
Es wird ein Objekt ausgewählt und überprüft, ob die in der Ansicht wiedergegebene Darstellung den erfassten Daten des Objekts entspricht.
Ergebnis hierbei: Attribute eines Objektes werden in der grafischen Darstellung nicht wiedergegeben.

/TF310/ Anlegen eines neuen Pflichtenheftes

1 - Bei der Dateneingabe wird überprüft, ob voreingestellte Ordner bzw. Unterpunkte auch gelöscht und erweitert werden können und ob der jeweils im Editierfenster eingegebene Text auch in den gewählten Ordnern bzw. Unterpunkten aufgenommen wird.

2 - Außerdem wird überprüft, ob ein Pflichtenheft einen Namen haben muss, um angelegt werden zu können, und ob auch wirklich nur einmal vorkommende Namen akzeptiert werden.

Ergebnis hierbei: Wird U-Topp2001 neu gestartet und ein abgespeichertes Projekt geöffnet, so wird ein im geöffneten Projekt vorhandenes Pflichtenheft durch ein mit gleichem Namen neu angelegtes überschrieben.

3 - Es wird überprüft, ob beim Anlegen eines neuen Pflichtenheftes auch immer ein leeres Eingabefenster erscheint.

Ergebnis hierbei: Dies war dann nicht der Fall, wenn zuvor schon ein Pflichtenheft eingegeben wurde, dessen Name im Editierfenster für die Pflichtenheftwurzel eingegeben wurde. Dann erscheint dieser Pflichtenheftname nämlich auch im Eingabefenster für das neu anzulegende Pflichtenheft.

/TF320/ Bearbeiten eines Pflichtenheftes

Es wird ein Pflichtenheft ausgewählt und überprüft, ob das Pflichtenheft nach der Änderung korrekt abgespeichert ist.

/TF330/ Löschen eines Pflichtenheftes

Es wird ein Pflichtenheft gelöscht und überprüft, ob das Pflichtenheft vollständig gelöscht wurde.

/TF340/ Ansicht eines Pflichtenheftes

Es wird ein Pflichtenheft ausgewählt und überprüft, ob die in der Ansicht wiedergegebene Darstellung den erfassten Daten des Pflichtenheftes entspricht.

Ergebnis hierbei: Bei Pflichtenheften von abgespeicherten Projekten, die mit U-Topp2001 wieder geöffnet werden, funktioniert die Ansicht noch nicht.

/TF410/ Plattformunabhängigkeit

Programm wird unter den zugänglichen Betriebssystemen kompiliert und getestet (nach TF1xx-TF3xx).

Ergebnis hierbei: Generell läuft es unter Windows sowie unter Linux/Unix auf Einzelplatzrechnern mit lokaler virtueller Maschine (bei X-Servern übers Netzwerk usw. gibt es Probleme). Detailabweichungen bestehen in der GUI-Darstellung, die aber nicht zu Einschränkungen führen. Unter Linux/Unix werden Schriften nicht korrekt gefunden, was allerdings ebenfalls nicht zu Einschränkungen führt, lediglich zu unschönen Fehlermeldungen beim Start des Programms. Allerdings sind in der Entwicklungsphase immer wieder gravierende Fehlfunktionen unter einem Betriebssystem aufgetreten, welche unter einem anderen nicht vorhanden waren. Die Kontrolle dieser Unregelmäßigkeiten erscheint schwierig und bleibt zunächst späteren Bug-Fixes vorbehalten. Letztendlich läuft UTopp2001 unter Windows-Systemen am stabilsten.

10. Entwicklungsumgebung

1. Software
   - Java 2.0
   - JBuilder
   - Texteditoren
2. Hardware
   - PC
3. Orgware
   - -
4. Schnittstellen
   - -