U-Topp Pflichtenheft, Version 2

OOPS!

05. Februar 2001

1 Zielbestimmung

U-Topp ist ein Computerprogramm, das in der Planungs-, Analyse- und Designphase eines objektorientierten Software-Systems eingesetzt werden kann. Man kann mit U-TOPP ein Pflichtenheft, sowie die UML-Komponenten Klassen, Use-Cases und Verbindungen erstellen und graphisch anzeigen lassen, **und in einem Projekt verwalten**. Die Erfassung der Daten erfolgt durch Eingabemasken, die graphische Darstellung kann in verschiedenen 'Feinheitsgraden' erfolgen. **Es besteht die Möglichkeit zur Erstellung von UML-Diagrammen**. Die erstellten Projekte können gespeichert und später wieder geladen werden.

1.1 Mußkriterien

Folgende Dinge müssen in U-Topp realisiert sein:

- Erfassmasken für die verschiedenen Komponenten (Klassen, Objekte, Verbindungen, UseCases, Pflichtenheft),
- Graphische Darstellung der Komponenten (in unterschiedlichen Detaillierungsgraden),
- Veränderbarkeit der erstellten Objekte.

Folgende Dinge müssen zusätzlich realisiert werden:

- Verwalten der Komponenten
- Erstellmöglichkeit für Diagramme
- Man muss das Programm später noch erweitern können

1.2 Wunschkriterien

Wie schon geschrieben, sollen Klassen- und Use-Case Diagramme in einer späteren Version dargestellt werden können.

1.3 Entwicklungsumgebung

Wir verwenden unter anderem die folgenden Programme:

- Borland JBuilder 4.0,
- Argo-UML,UTOPP 1.1
- Frontpage,
- JDK 1.3 von SUN, Jikes von IBM

2 Produkteinsatz

2.1 Anwendungsbereiche

U-TOPP unterstützt die Planungs-, Analyse- und Designphase eines Software-Systems.

2.2 Zielgruppe

Das Programm richtet sich an Softwareentwickler.

3 Produktumgebung

3.1 Software

U-Topp läuft auf jedem Computer, auf dem eine Java virtuelle Maschine installiert ist.

3.2 Hardware

Das Programm soll auf jedem Standard-Rechner laufen.

3.3 Orgware

keine Bedingungen

3.4 Schnittstellen

keine Bedingungen

3.5 Betriebsbedingungen

Das Programm ist eine Einzelplatzanwendung.

4 Produktinformationen (Funktionen/UseCases)

4.1 Projekt neu

Akteur:

- User

Vorbedingung:

- keine

Nachbedingung:

- neues Projekt angelegt. Komponenten können nun generiert und editiert werden.

Invarianten:

- nur ein Projekt aktiv

Ablauf:

1. „Projekt neu" Button wird gedrückt
2. Wird gefragt ob altes Projekt gespeichert werden soll
3. neues Projekt wird geladen

4.2 Projekt laden:

Akteur:

- User

Vorbedingung:

- keine

Nachbedingung:

- Projekt geladen

Invarianten:

- nur ein Projekt aktiv

Ablauf:

1. „Projekt laden" Button wird gedrückt
2. Wird gefragt ob altes Projekt gespeichert werden soll
3. Projekt geladen

Ausnahmen, Fehlersituationen:

- Falscher Pfad, ungültiger Dateityp oder Dateiname wird mit entsprechenden Fehlermeldungen abgefangen

4.3 Projekt speichern

Akteur:

- User

Vorbedingung:

- Projekt erzeugt und vorhanden

Nachbedingung:

- Projekt gespeichert

Invarianten:

- keine

Ablauf:

1. Projekt speichern Button wird gedrückt
2. Name und Speicherpfad wird gefragt
3. Projekt gespeichert

Ausnahmen, Fehlersituationen:

- Falscher Pfad, schon existierender Name, oder kein Speicherplatz werden mit entsprechenden Fehlermeldungen abgefangen

4.4. Komponente hinzufügen:

Akteur:

- User

Vorbedingungen:

- Die zur Komponente passende Diagrammansicht ist aktiviert. Klassen kann man z.B. nur in der Klassendiagramm-Ansicht hinzufügen, Use-Cases nur in der Use-Case Diagrammansicht.

Invarianten:

- Die entsprechende Diagramm-Ansicht bleibt aktiviert

Ablauf:

- Benutzer wählt <Komponente> hinzufügen im Menü
- Neue Komponente wird mit default-Werten initialisiert und erscheint im Diagramm

Nachbedingungen:

- Neue Komponente steht zum Positionieren und mit Daten füllen zur Verfügung

4.5. Komponente löschen:

Akteur:

- User

Vorbedingungen:

- Die zur Komponente passende Diagrammansicht ist aktiviert. (vgl. 4.) Das Diagramm enthält mindestens eine Komponente

Invarianten:

- Die entsprechende Diagramm-Ansicht bleibt aktiviert

Ablauf:

- Benutzer wählt „Löschen“ aus dem Kontextmenü der Komponente
- Falls Komponente bereits editiert wurde, wird eine Sicherheitsnachfrage ausgegeben
- Die Komponente wird aus dem Diagramm entfernt

Nachbedingungen:

- Aus dem Diagramm wurde eine Komponente entfernt

4.6. Komponente verschieben:

Akteur:

- User

Vorbedingungen:

- Die zur Komponente passende Diagrammansicht ist aktiviert. (vgl. 4.)
- Das Diagramm enthält mindestens eine Komponente

Invarianten:

- Die entsprechende Diagramm-Ansicht bleibt aktiviert.

Ablauf:

- Benutzer wählt mit dem Mauspfeil auf die Komponente und hält die linke Maustaste gedrückt und verschiebt die Komponente innerhalb des Diagramms
- Benutzer legt durch Loslassen der Maustaste die Komponente ab

Nachbedingungen:

- Eine Komponente wurde verschoben

Variationen:

Falls die Komponente eine Verbindung ist:

- Benutzer kann beide Seiten der Verbindung getrennt verschieben
- Beim Ablegen werden nicht die genauen Koordinaten verwendet, sondern das jeweilige Ende mit der dort liegenden Klasse bzw. dem dort liegenden Use-Case/Akteur verbunden.
- Liegt keine Klasse bzw. kein Use-Case oder Akteur an der Stelle, wo man das Verbindungsende ablegen will, wird die Positionierung nicht durchgeführt

4.7. Komponente editieren:

Akteur:

- User

Vorbedingungen:

- Die zur Komponente passende Diagrammansicht ist aktiviert. (vgl. 4.)
  Das Diagramm enthält mindestens eine Komponente

Invarianten:

- Die entsprechende Diagramm-Ansicht bleibt aktiviert

Ablauf:

- Benutzer wählt „editieren" aus dem Kontextmenü der Komponente oder durch einen Doppelklick auf die Komponente
- Der Benutzer erhält im jeweiligen Änderungs-Dialog die Möglichkeit, die Daten der Komponente einzugeben oder zu verändern

Nachbedingungen:

- Daten einer Komponente wurden eingegeben oder verändert

4.8. Objekt/Klassen-Detail-Einstellungen:

Akteur:

- User

Vorbedingungen:

- Die Klassendiagramm-Ansicht ist aktiviert.
- Das Klassendiagramm enthält mindestens ein Objekt oder eine Klasse

Invarianten:

- Die Klassen-Diagramm-Ansicht bleibt aktiviert

Ablauf:

- Benutzer aktiviert das Kontextmenü der Klasse/des Objektes
- Das Kontextmenu enthält einen Punkt für „Attribute" und bei Klassen noch einen Punkt „Methoden". Falls das Einblenden der Attribute bzw. der Methoden aktiviert ist, ist der entsprechende Punkt jeweils mit einem Haken versehen
- Benutzer wählt einen der beiden Einträge und wechselt damit jeweils zwischen Einblenden/Ausblenden der Attribute/Methoden.

Nachbedingungen:

- Die Detail-Einstellung der Darstellung der jeweiligen Klasse bzw. des jeweiligen Objekts wurden geändert

4.9. Ansichtswechsel:

Akteur:

- User

Vorbedingungen:

- Keine

Invarianten:

- Keine

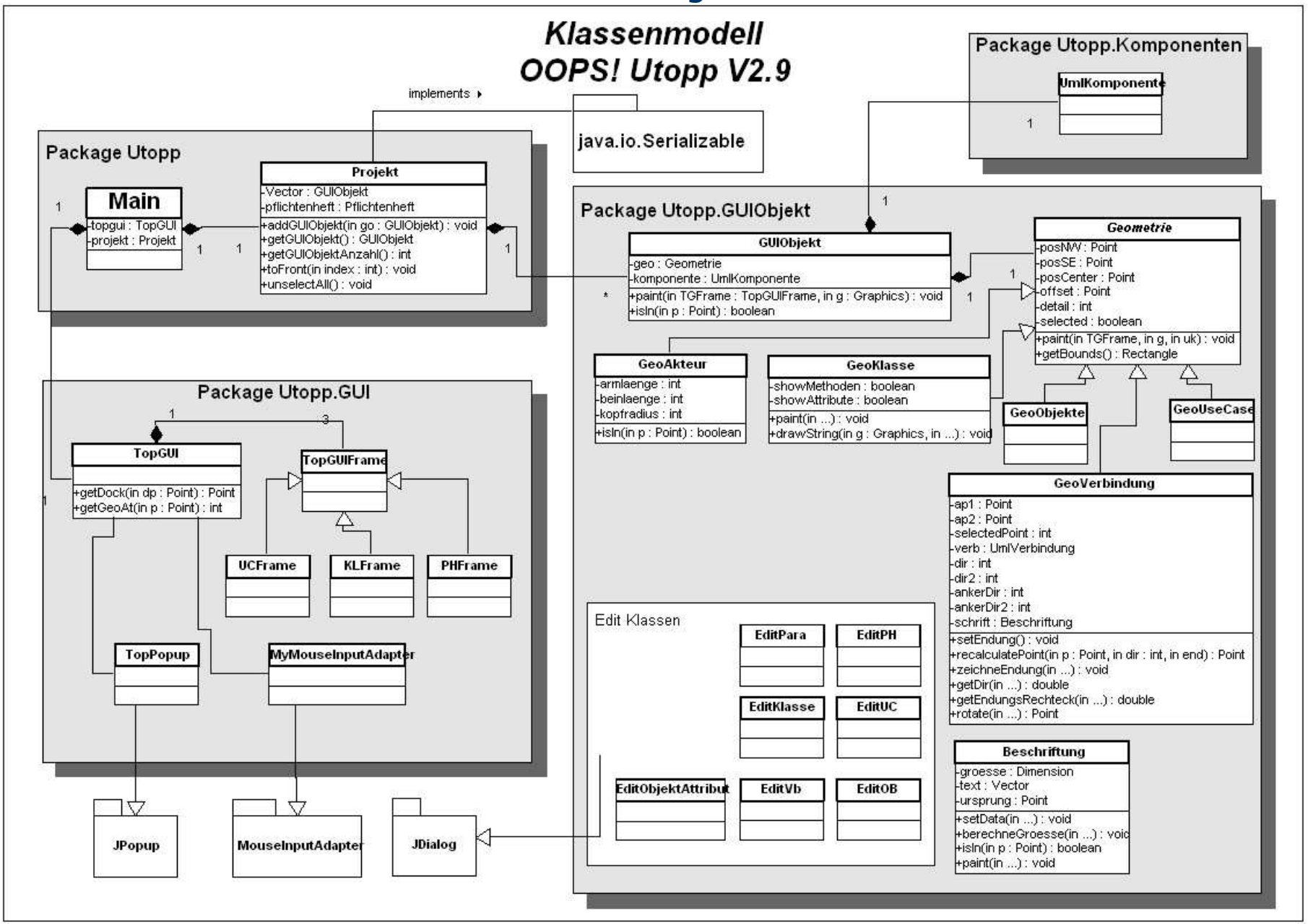
Ablauf:

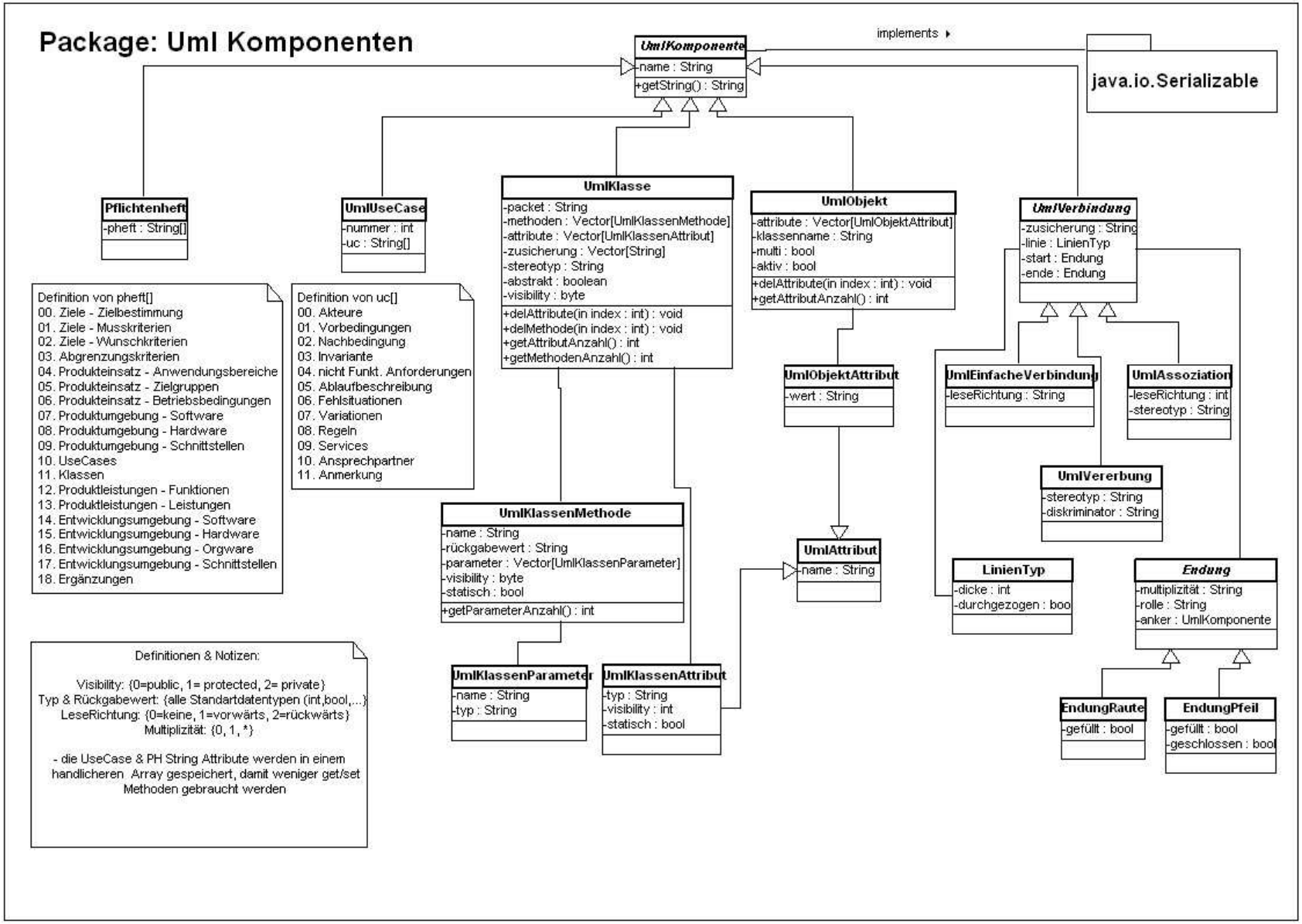
- Benutzer wählt im Menü des Hauptfensters eine Ansicht aus
- Zur Auswahl stehen: Pflichtenheft, Use-Case-Diagramm und Klassendiagramm
- Die momentan dargestellte Diagramm-Ansicht wird durch die ausgewählte ersetzt.

Nachbedingung:

- Die gewählte Diagrammansicht wird angezeigt

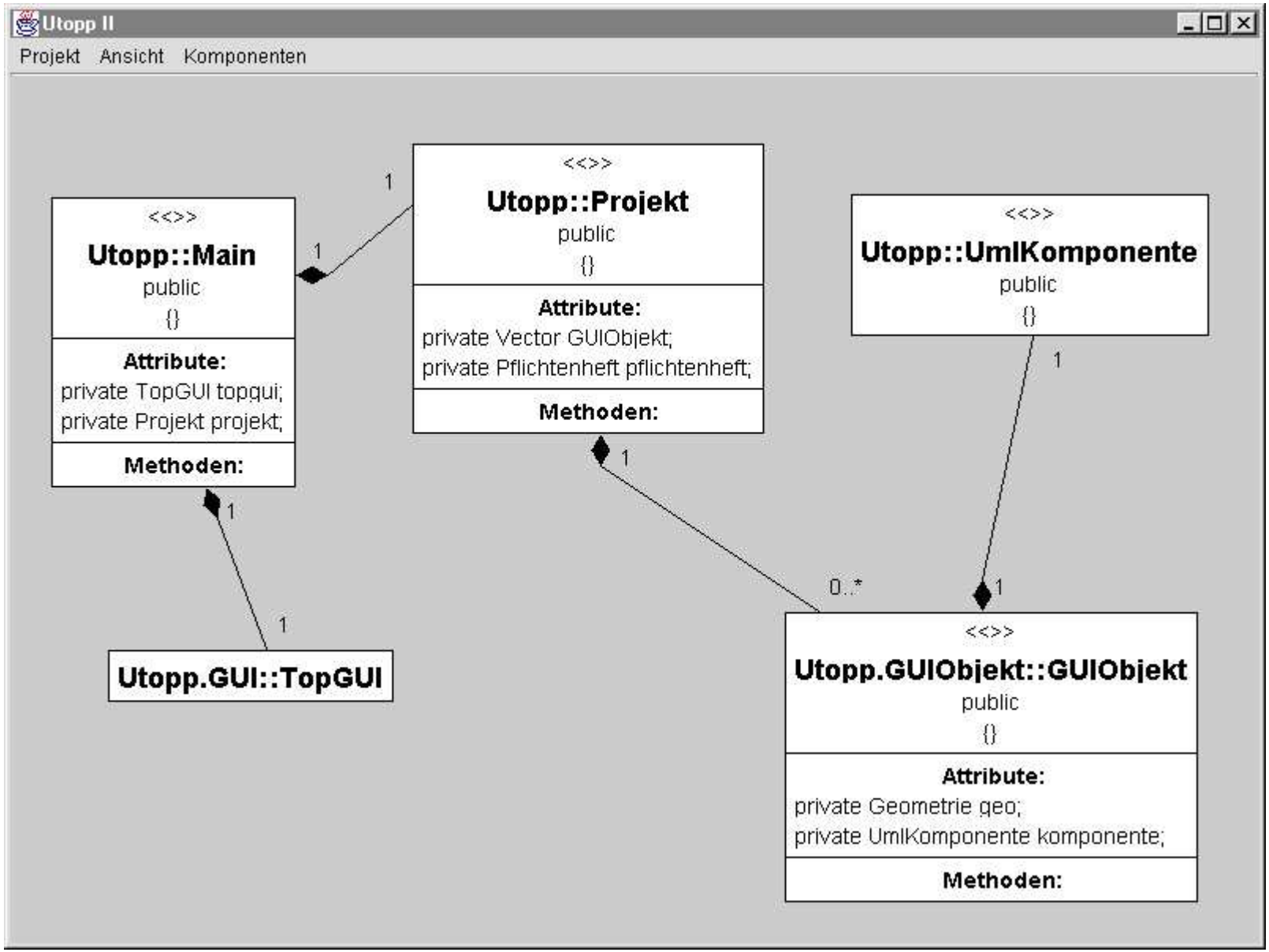
5.Klassendiagramme:



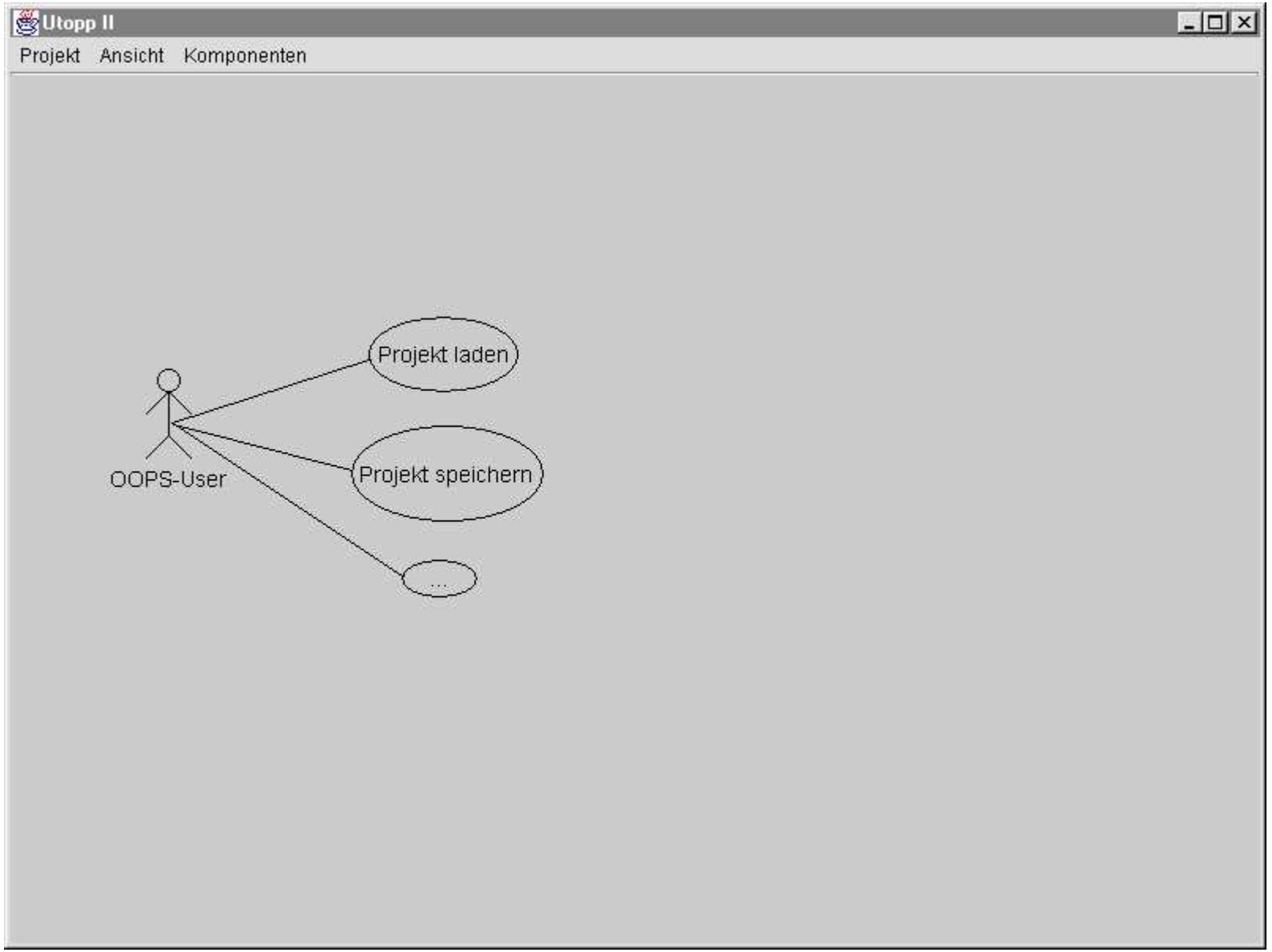

Package: Uml Komponenten
implements
UmlKomponente
-name : String
+getString() : String
java.io.Serializable
Pflichtenheft
-pheft : String[]
UmlUseCase
-nummer : int
-uc : String[]
UmlKlasse
-packet : String
-methoden : Vector[UmlKlassenMethode]
-attribute : Vector[UmlKlassenAttribut]
-zusicherung : Vector[String]
-stereotyp : String
-abstrakt : boolean
-visibility : byte
+delAttribute(in index : int) : void
+delMethode(in index : int) : void
+getAttributAnzahl() : int
+getMethodenAnzahl() : int
UmlObjekt
-attribute : Vector[UmlObjektAttribut]
-klassenname : String
-multi : bool
-aktiv : bool
+delAttribute(in index : int) : void
+getAttributAnzahl() : int
UmlVerbindung
-zusicherung : String
-linie : LinienTyp
-start : Endung
-ende : Endung
Definition von pheft[]
00. Ziele - Zielbestimmung
01. Ziele - Musskriterien
02. Ziele - Wunschkriterien
03. Abgrenzungskriterien
04. Produkteinsatz - Anwendungsbereiche
05. Produkteinsatz - Zielgruppen
06. Produkteinsatz - Betriebsbedingungen
07. Produktumgebung - Software
08. Produktumgebung - Hardware
09. Produktumgebung - Schnittstellen
10. UseCases
11. Klassen
12. Produktleistungen - Funktionen
13. Produktleistungen - Leistungen
14. Entwicklungsumgebung - Software
15. Entwicklungsumgebung - Hardware
16. Entwicklungsumgebung - Orgware
17. Entwicklungsumgebung - Schnittstellen
18. Ergänzungen
Definition von uc[]
00. Akteure
01. Vorbedingungen
02. Nachbedingung
03. Invariante
04. nicht Funkt. Anforderungen
05. Ablaufbeschreibung
06. Fehlsituationen
07. Variationen
08. Regeln
09. Services
10. Ansprechpartner
11. Anmerkung
UmlObjektAttribut
-wert : String
UmlEinfacheVerbindung
-leseRichtung : String
UmlAssoziation
-leseRichtung : int
-stereotyp : String
UmlVererbung
-stereotyp : String
-diskriminator : String
UmlKlassenMethode
-name : String
-rückgabewert : String
-parameter : Vector[UmlKlassenParameter]
-visibility : byte
-statisch : bool
+getParameterAnzahl() : int
UmlAttribut
-name : String
LinienTyp
-dicke : int
-durchgezogen : boo
Endung
-multiplizität : String
-rolle : String
-anker : UmlKomponente
UmlKlassenParameter
-name : String
-typ : String
UmlKlassenAttribut
-typ : String
-visibility : int
-statisch : bool
EndungRaute
-gefüllt : bool
EndungPfeil
-gefüllt : bool
-geschlossen : bool
Definitionen & Notizen:
Visibility: {0=public, 1= protected, 2= private}
Typ & Rückgabewert: {alle Standartdatentypen (int,bool,...}
LeseRichtung: {0=keine, 1=vorwärts, 2=rückwärts}
Multiplizität: {0, 1, *}
- die UseCase & PH String Attribute werden in einem handlicheren Array gespeichert, damit weniger get/set Methoden gebraucht werden

6 Screenshots:

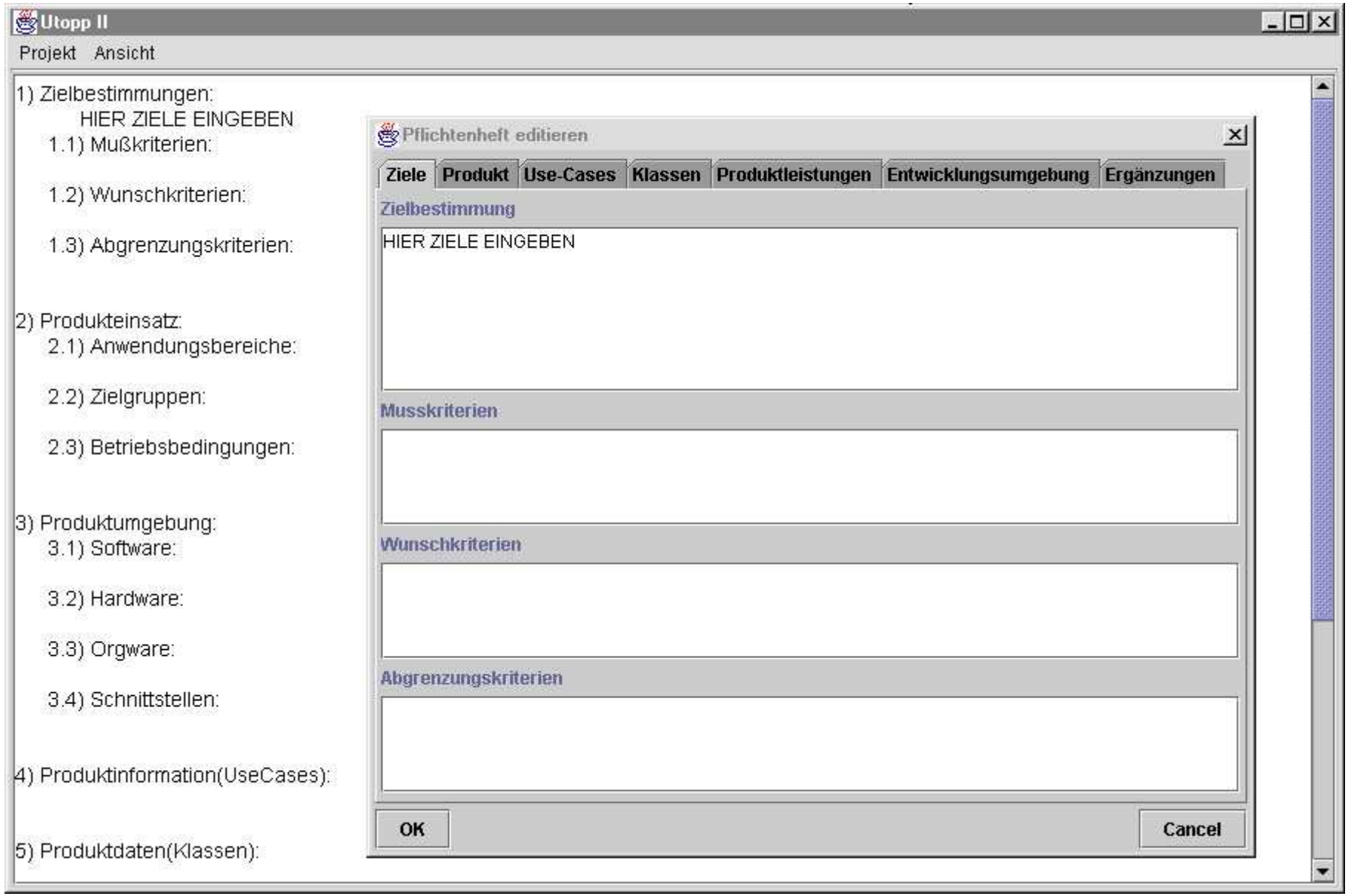
Klassenansicht



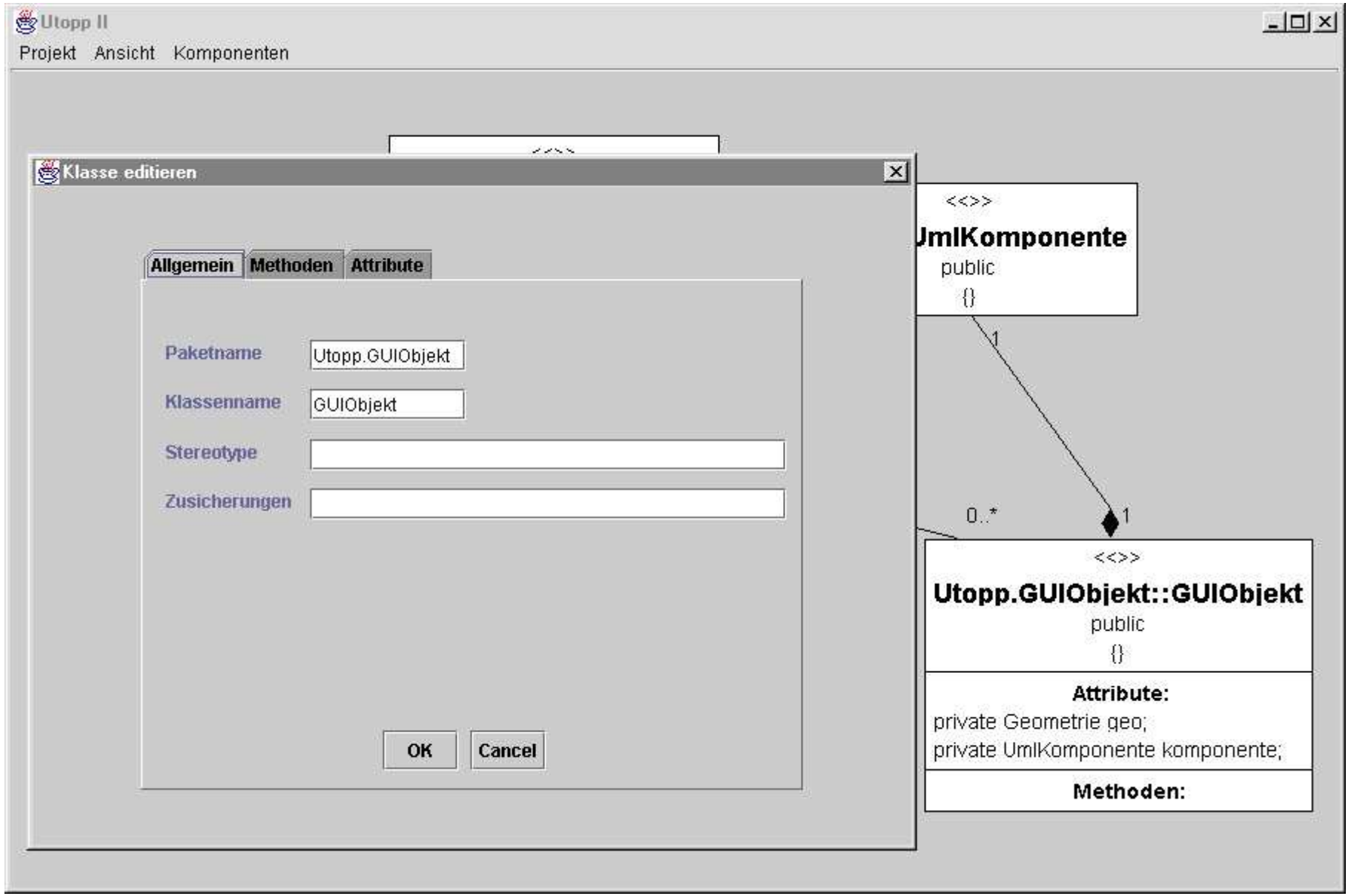
UseCase Ansicht



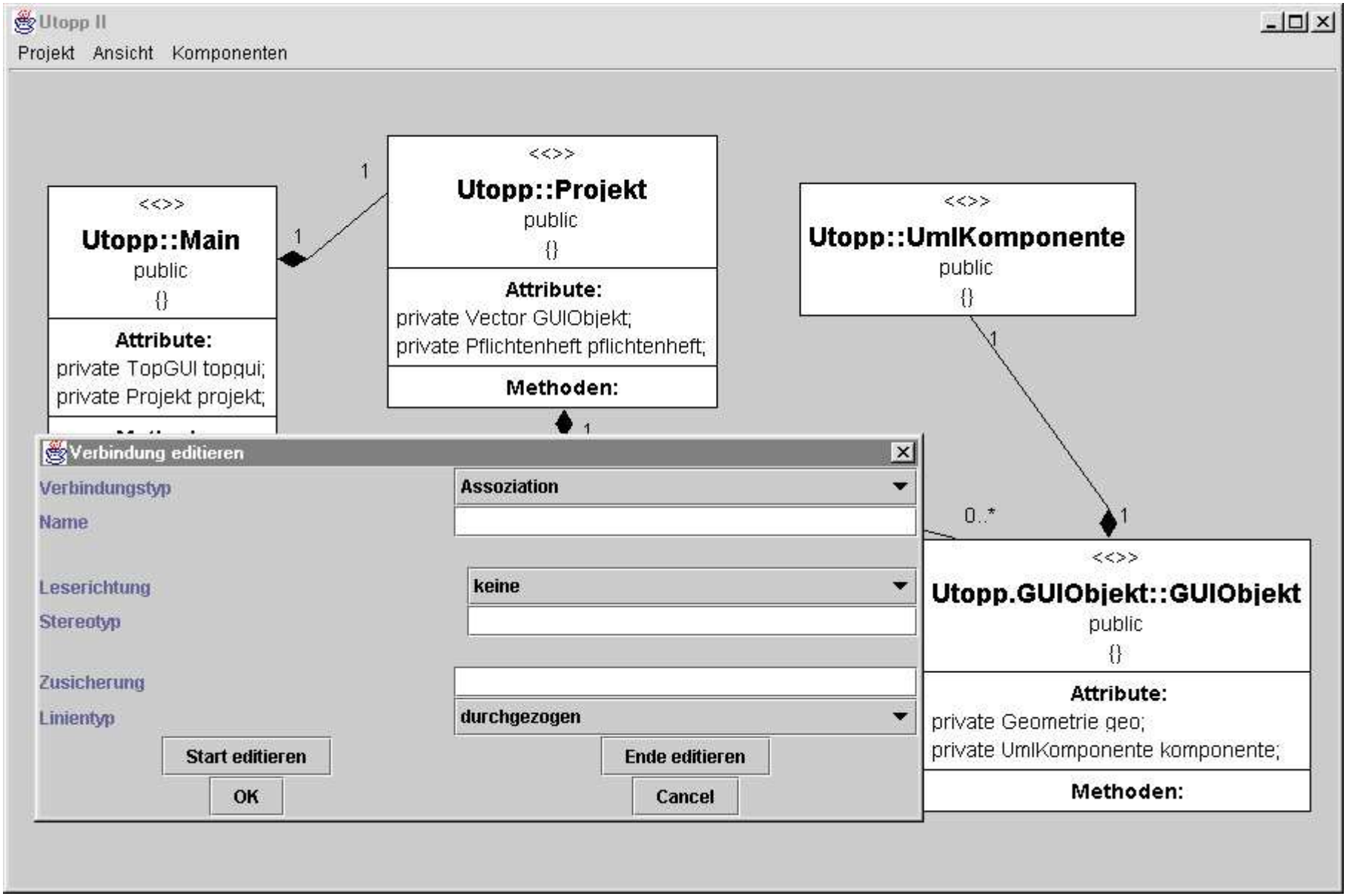
Pflichtenheft Ansicht



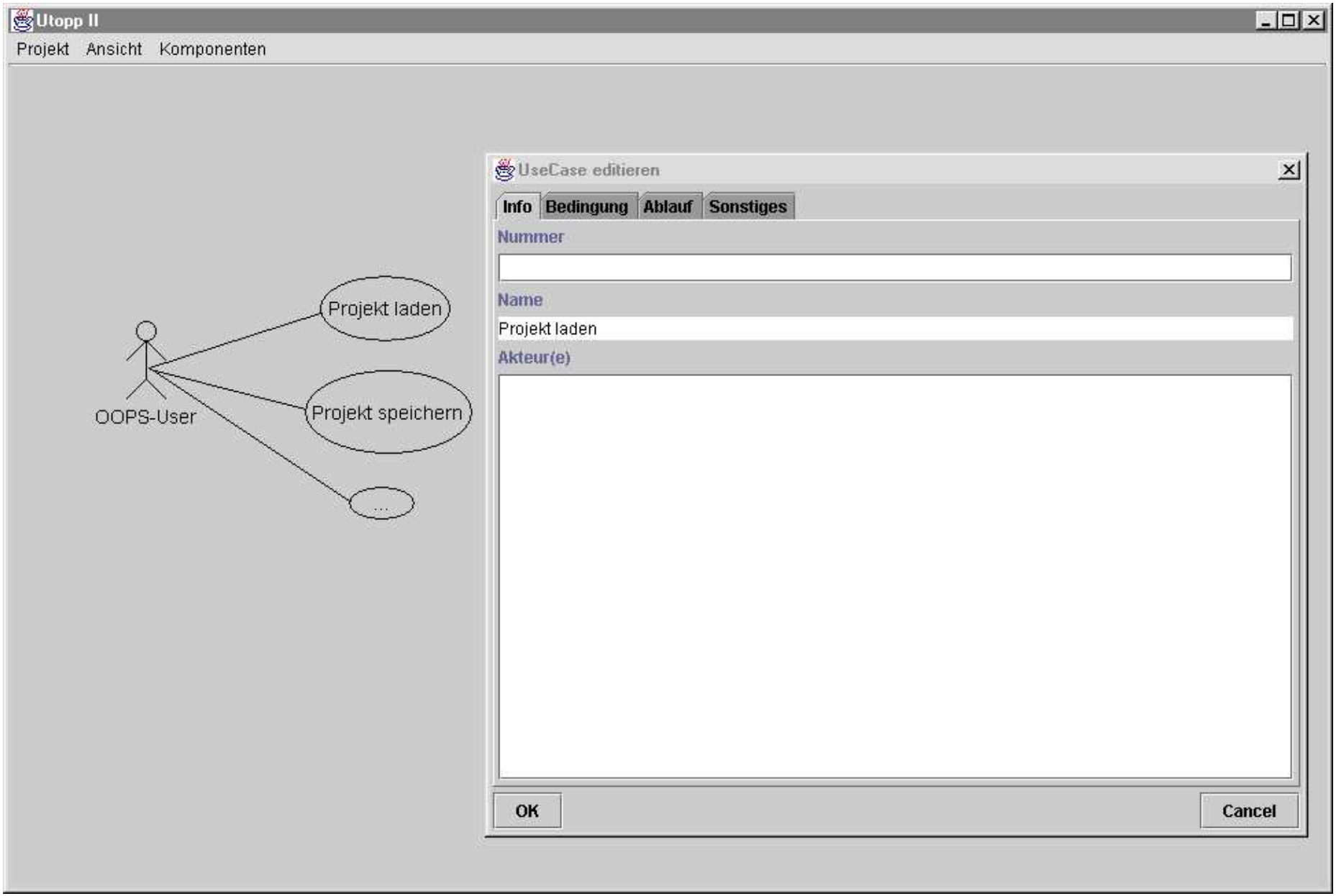
Klassen Editieren



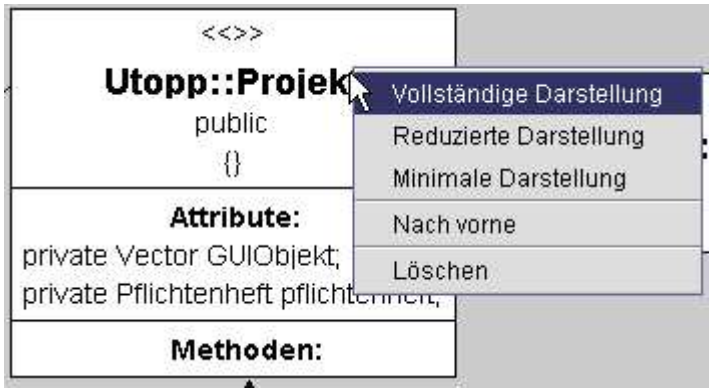
Verbindung Editieren



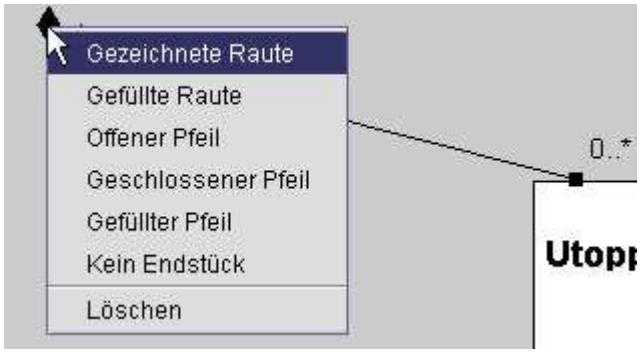
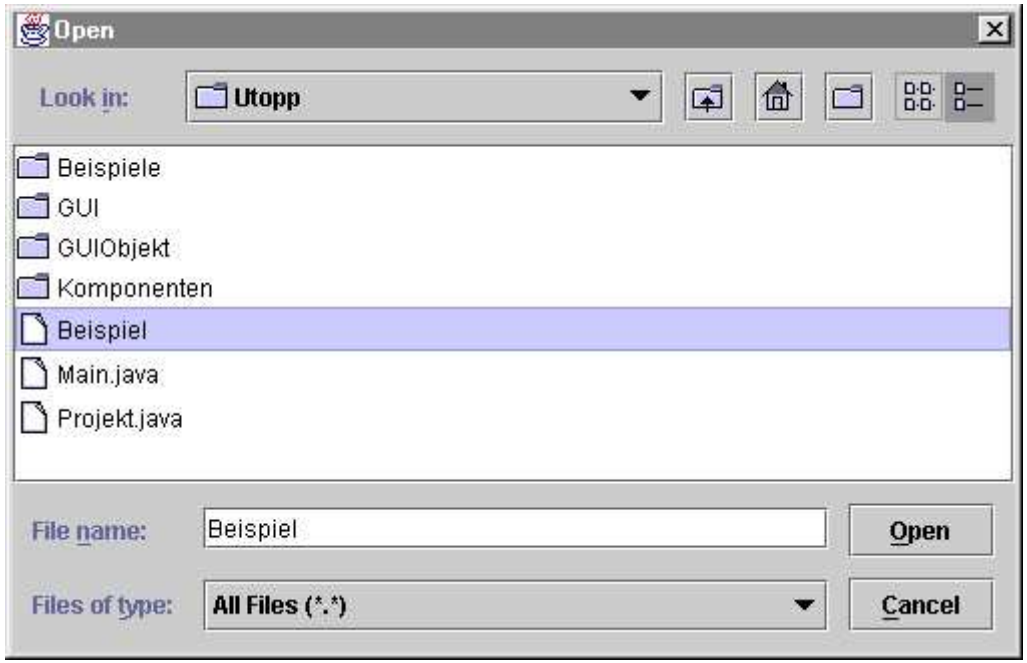
UseCase Editieren



Klassen Popupmenü



Verbindung Popupmenü



Speichern / Laden Dialog