Dokumentation zu

UTOPP advanced

der Gruppe

CvK Mber(t)

Version vom 04. 02. 2001

Inhaltsverzeichnis

1 Zielbestimmungen

1.1 Musskriterien

Bei »Utopp advanced« handelt es sich um ein Programm zur Modellierung von Programmen mit der Sprache UML. Utopp stellt hierzu die Komponenten Klasse, Objekt, Usecase, Verbindung und Pflichtenheft zur Verfügung. Diese werden alle unter einer Einheit – dem »Projekt« – miteinander verbunden, das Projekt faßt also mehrere Instanzen der einzelnen Komponenten unter einem Rahmen zusammen. Dies betrifft sowohl die Speicherung als auch die Sichtbarkeit. Es ist stets nur ein Projekt sichtbar.Für alle Komponenten wird Werkzeug zu Erstellung und Editierung in Form von Eingabemasken zur Verfügung gestellt, wobei sich ein Pflichtenheft nicht mehrfach erstellen läßt. Desweiteren lassen sich Klassendiagramme (die auch Objekte beinhalten können) und UseCaseDiagramme in einem gesonderten Rahmen erstellen. In diesem Rahmen kann man aus einer Liste der im Projekt existierenden Komponenten wählen und diese hinzufügen. Ihre Plazierung im Diagramm erfolgt durch Steuerung mit der Maus. Verbindung zwischen Klassen/Objekten oder UseCases können ebenfalls aus einer Liste ausgewählt werden, sie können in den Komponenten der graphischen Darstellung »verankert« werden, d.h. sie bewegen sich bei einer Änderung des Diagramms mit. In der graphischen Darstellung ist durch Doppelklick auf die Komponenten eine Änderung des Detailierungsgrades möglich. Die Klassen- und UseCaseDiagramme sind druckbar

1.2 Wunschkriterien

Utopp advanced bietet Möglichkeiten zur Erstellung von Sequenzdiagrammen. Die Klassen- und UseCaseDiagramme sind druckbar.

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.

2 Produkteinsatz

2.1 Anwendungsbereiche

2.2 Zielgruppen

Das Programm U-Topp advanced 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.

2.3 Betriebsbedinungen

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

as Produkt benötigt eine virtuelle Java-Maschine, z.B. das Java-Runtime-Environment von Sun, das auch bei der Entwicklung verwendet wird. Diese sollte mindestens in der Version 1.2 vorliegen.

3.2 Hardware

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

i. Genügend ausgestatteter PC

ii. Sun® -Workstations im mathematischen Institut

Darüber hinaus wird keine besondere Hardware benötigt.

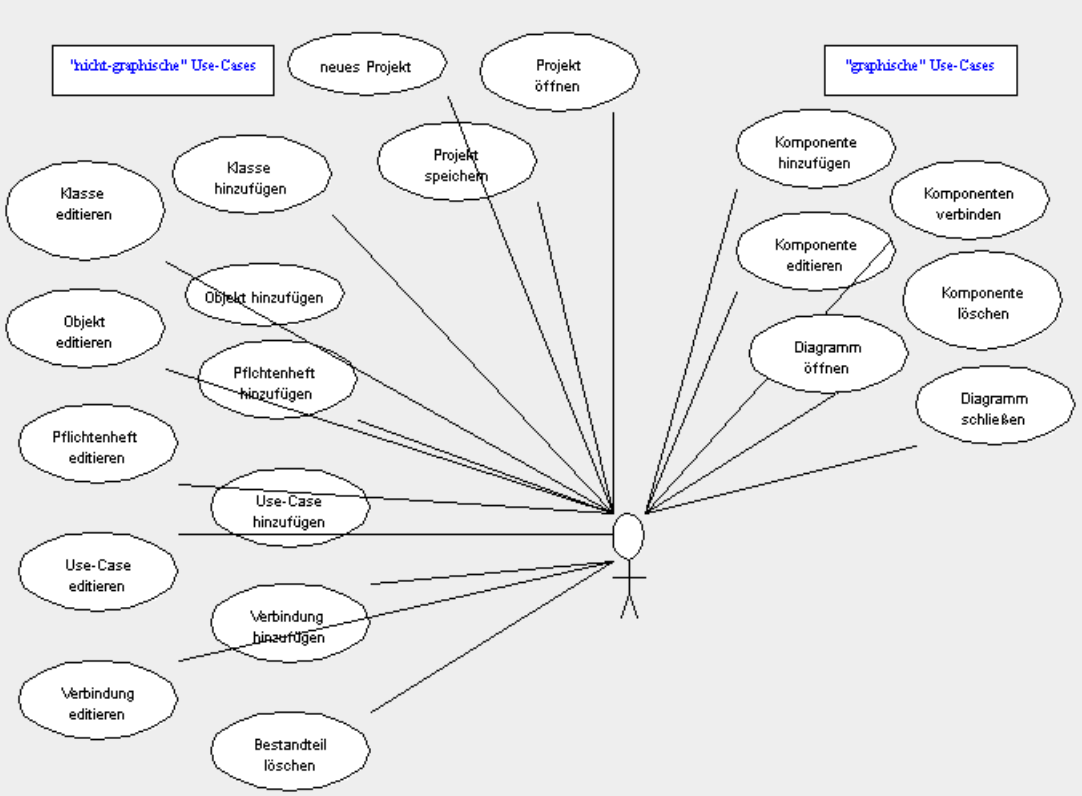

Abbildung 1: Use-Case-Diagramm

3.3 Orgware

Es wird keine besondere Orgware benötigt.

3.4 Schnittstellen

Es werden keine besonderen Schnittstellen benötigt.

4 Produktinformation

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 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.

4.3.7 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.

4.3.8 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

4.3.9 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

4.3.10 Use-Case zum Diagramm hinzufügen

- **Akteure:** Benutzer
- **Vorbedingungen:** Der Use-Case muss erstellt und zum Projekt hinzugefügt sein. Im weiteren sollte der Use-Case in die Auswahlliste des Use-Case Diagrammfensters geladen sein.
- **Nachbedingungen:** Der Use-Case ist in das Use-Case Diagrammfeld eingefügt.
- **Ablaufbeschreibung:**

    1. Benutzer wählt in der Menüleiste, nachdem er ein Projekt angelegt hat, unter Use-Case Diagramm den Punkt Use-Case Diagramm hinzufügen
    2. Dann wird in der Use-Case Auswahlliste ein Use-Case ausgewählt
    3. Durch die Bestätigung wird der Use-Case in das Nebenstehende Feld eingefügt. Dabei ist grafisch der Use-Case-Name im ovalen Rahmen zu sehen.

4.3.11 Akteur zum Diagramm hinzufügen

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein Projekt sollte angelegt und geöffnet sein. Das Use-Case Diagrammfenster sollte geöffnet sein.
- **Nachbedingungen:** Der Akteur ist in das Use-Case Diagrammfeld eingefügt.
- **Ablaufbeschreibung:**

    1. Benutzer wählt in der Use-Case Diagramm Oberfläche den Punkt Akteur hinzufügen, und benennt diesen nach dem Akteur den es symbolisch darstellt.
    2. Es erscheint ein Strichmännchen mit seinem Namen im Use-Case Diagrammfeld und steht zur weiteren Bearbeitung zur Verfügung.

4.3.12 Verbindung zum Use-Case Diagramm hinzufügen

- **Akteure:** Benutzer
- **Vorbedingungen:** Es sollte bereits ein Akteur und mindestens ein Use-Case zum Diagrammfeld eingefügt sein.
- **Nachbedingungen:** Es ist eine Verbindung zwischen Use-Case und Akteur zum Diagrammfeld eingefügt, und nach seiner Verbindungseigenschaft benannt.
- **Ablaufbeschreibung:**
  1. Benutzer wählt in der Use-Case Diagramm Oberfläche aus der Verbindungsliste die Verbindung mit jeweiliger Eigenschaft die er bestimmt.
  2. Der Verbindungspfeil der zunächst unabhängig im Diagrammfeld erscheint, kann durch Angabe von Anfangs- und Endpunkt positioniert werden.

4.3.13 Neuen Use-Case erstellen

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein Projekt muss geöffnet sein.
- **Nachbedingungen:** Eine neue Instanz des Typs Use-Case ist erstellt worden.
- **Ablaufbeschreibung:**
  1. Der Benutzer ruft in der Use-Case Diagramm Oberfläche den Punkt neuen Use-Case erstellen auf.
  2. Es erscheint die Use-Case Eingabemaske. Nun wird wie in 4.3.10 fortgefahren

4.3.14 Verbindung an einer Klasse verankern

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein Projekt ist geöffnet und sowohl eine >Verbindung als auch mindestens eine Klasse wurden dem Diagramm bereits hinzugefügt
- **Nachbedingungen:** Ein Objekt wurde zum Diagramm hinzugefügt.
- **Ablaufbeschreibung:**

1. Durch Anklicken zieht der Benutzer eine Verbindungsspitze auf das Feld einer Klasse
2. Doppelklick auf die Spitze verknüpft die Verbindung dauerhaft mit der Klasse

4.3.15 Klassendiagramm schliessen

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein Klassendiagramm ist geöffnet
- **Nachbedingungen:** Das Klassendiagramm wurde geschlossen
- **Ablaufbeschreibung:**
  1. Der Benutzer klickt das Kreuz am rechten oberen Rand an
  2. Der Benutzer bekommt die Gelegenheit, Änderungen abzuspeichern bzw abzubrechen

4.3.16 Verbindung zum Klassendiagramm hinzufügen

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein Projekt ist geöffnet und in der Auswahlliste sind Verbindungen vorhanden sowie eine ausgewählt
- **Nachbedingungen:** Eine Verbindung wurde zum Diagramm hinzugefügt
- **Ablaufbeschreibung:**
  1. Öffnen eines Projektes
  2. Durch die Menüleiste lässt der Benutzer den Rahmen für ein >Klassendiagramm erscheinen
  3. Doppelklick auf den Verbindung–Hinzufügen Button
  4. Die Verbindung wird im Rahmen angezeigt

4.3.17 Objekt zum Klassendiagramm hinzufügen

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein Projekt ist geöffnet und in der Auswahlliste >sind Objekte vorhanden sowie eines ausgewählt

- **Nachbedingungen:** Ein Objekt wurde zum Diagramm hinzugefügt
- **Ablaufbeschreibung:**
    1. Öffnen eines Projektes
    2. Durch die Menüleiste lässt der Benutzer den Rahmen für ein >Klassendiagramm erscheinen
    3. Doppelklick auf den Objekt–Hinzufügen Button
    4. Das Objekt wird im Rahmen angezeigt

4.3.18 Klasse zum Klassendiagramm hinzufügen

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein Projekt ist geöffnet und in der Auswahlliste >sind Klassen vorhanden sowie eine ausgewählt
- **Nachbedingungen:** Eine Klasse wurde zum Diagramm hinzugefügt
- **Ablaufbeschreibung:**
    1. Öffnen eines Projektes
    2. Durch die Menüleiste lässt der Benutzer den Rahmen für ein >Klassendiagramm erscheinen
    3. Doppelklick auf den Klasse–Hinzufügen Button
    4. Die Klasse wird im Rahmen angezeigt

4.3.19 Projekt anlegen

- **Akteure:** Benutzer
- **Vorbedingungen:** das Programm ist gestartet
- **Nachbedingungen:** Ein neues Projekt ist angelegt. Das vorher geöffnete ist eventuell gespeichert
- **Ablaufbeschreibung:**
    1. Menüpunkt Projekt → neues Projekt
    2. War vorher ein anderes Projekt geöffnet, wird der Benutzer gefragt, ob er abspeichern möchte.
    3. die Anzeigeoerfläche wird geleert

4.3.20 Projekt speichern

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein Projekt ist geöffnet.
- **Nachbedingungen:** Das Projekt ist gespeichert.
- **Ablaufbeschreibung:**
  1. Menüpunkt Projekt → Projekt speichern
  2. Es erscheint ein FileSaveDialog
  3. Der Benutzer wählt den gewünschten Pfad und Dateinamen aus.
  4. Das Projekt wird unter dem gewählten Namen gespeichert.

4.3.21 Projekt öffnen

- **Akteure:** Benutzer
- **Vorbedingungen:** Das Programm läuft.
- **Nachbedingungen:** Das gewünschte Projekt ist geöffnet. Das vorher geöffnete ist eventuell gespeichert
- **Ablaufbeschreibung:**
  1. Menüpunkt Projekt → Projekt öffnen
  2. War vorher ein anderes Projekt geöffnet, wird der Benutzer gefragt, ob er abspeichern möchte.
  3. Es erscheint ein FileOpenDialog
  4. Der Benutzer wählt die gewünschte Datei.
  5. Das Projekt wird geladen und es stehen die entsprechenden Informationen zur Verfügung.

4.3.22 Komponente eines Projekts löschen

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein Projekt ist geöffnet.
- **Nachbedingungen:** Der gewünschte Bestandteil ist aus dem Projekt gelöscht.

- **Ablaufbeschreibung:**
    1. Menüpunkt <Typ der Komponente> → löschen wird aufgerufen
    2. Es erscheint eine Auswahlliste mit den zum Projekt gehörigen Komponenten des entsprechenden Typs
    3. Der Benutzer wählt die gewünschte Komponente.
    4. Die Komponente wird aus dem Projekt und allen Diagrammen (zusammen mit den Verbindungen) gelöscht.

4.3.23 Komponente aus einem Diagramm löschen

- **Akteure:** Benutzer
- **Vorbedingungen:** Das betreffende Diagramm ist geöffnet.
- **Nachbedingungen:** Der gewünschte Bestandteil ist aus dem Diagramm gelöscht.
- **Ablaufbeschreibung:**
    1. Die Komponente wird durch rechten Mausklick ausgewählt.
    2. Die Entfernen-Taste wird gedrückt
    3. Die Komponente wird aus dem Diagramm gelöscht. Alle zugehörigen Verbindungen werden ebenfalls entfernt.

4.3.24 Komponente aus einem Diagramm heraus editieren

- **Akteure:** Benutzer
- **Vorbedingungen:** Das betreffende Diagramm ist geöffnet.
- **Nachbedingungen:** Der gewünschte Bestandteil ist nach den Wünschen des Benutzers verändert.
- **Ablaufbeschreibung:**
    1. Die Komponente wird durch doppelten Mausklick ausgewählt.
    2. Die zugehörige Editier-Maske erscheint
    3. Der Benutzer nimmt die gewünschten Änderungen vor
    4. Nach Bestätigung der Änderungen erscheint wiederum das Diagramm, in welchem die geänderten Eigenschaften sichtbar sind.

4.3.25 Komponente eines Projekts löschen

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein Projekt ist geöffnet.
- **Nachbedingungen:** Der gewünschte Bestandteil ist aus dem Projekt gelöscht.
- **Ablaufbeschreibung:**
    1. Menüpunkt <Typ der Komponente> → löschen wird aufgerufen
    2. Es erscheint eine Auswahlliste mit den zum Projekt gehörigen Komponenten des entsprechenden Typs
    3. Der Benutzer wählt die gewünschte Komponente.
    4. Die Komponente wird aus dem Projekt und allen Diagrammen (zusammen mit den Verbindungen) gelöscht.

4.3.26 Neues Use-CaseDiagramm erstellen

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein Projekt muss geöffnet sein.
- **Nachbedingungen:** Eine neue Instanz des Typs Use-CaseDiagramm ist erstellt worden.
- **Ablaufbeschreibung:**
    1. Der Benutzer ruft in der Menüleiste den Punkt Use-CaeDiagramm –>Use-CaseDiagramm erstellen auf
    2. Es erscheint die Use-CaseDiagrammoberfläche

4.3.27 Neues Klassendiagramm erstellen

- **Akteure:** Benutzer
- **Vorbedingungen:** Ein Projekt muss geöffnet sein.
- **Nachbedingungen:** Eine neue Instanz des Typs Klassendiagramm ist erstellt worden.
- **Ablaufbeschreibung:**

1. Der Benutzer ruft in der Menüleiste den Punkt Klassendiagramm– >Klassendiagramm erstellen auf
2. Es erscheint die Klassendiagrammoberfläche

5 Fachkonzeptklassen und Produktdaten

5.1 Klassendiagramm

*S*iehe Abbildungen auf den folgenden Seiten

5.2 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 ???

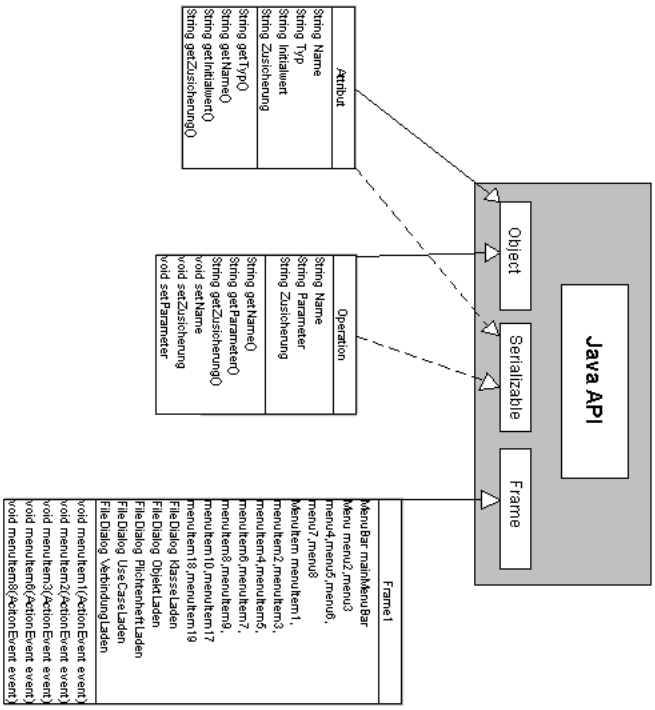

Abbildung 2: Das Klassendiagramm für das Hauptmenü

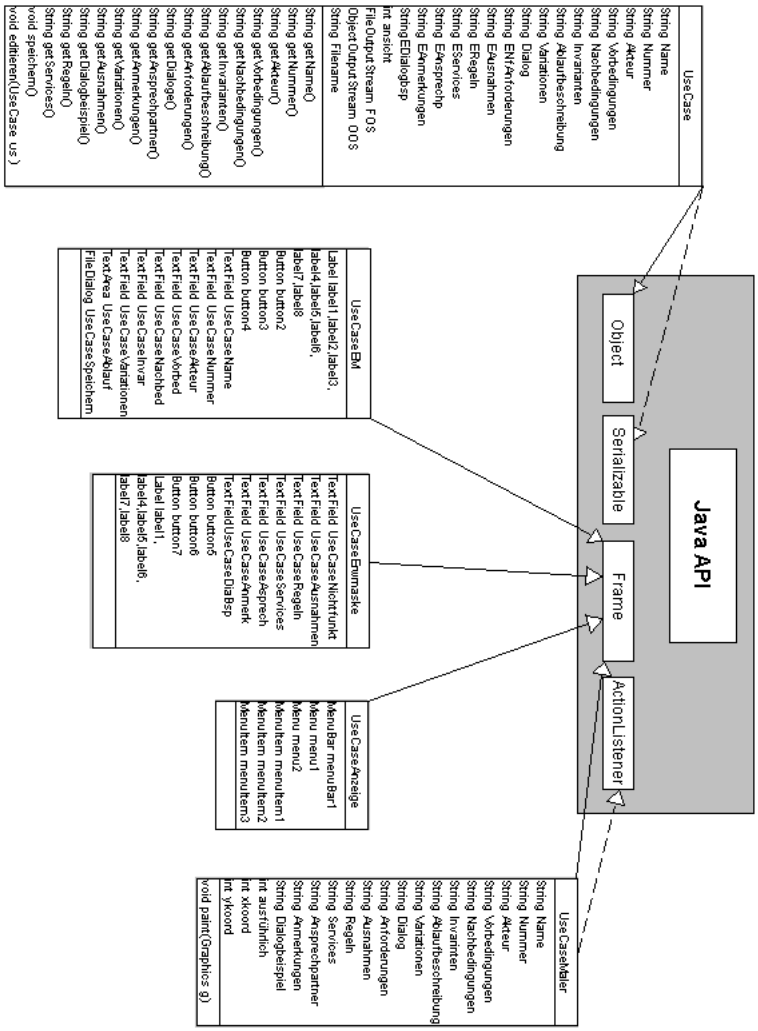

Abbildung 3: Das Klassendiagramm für den Bereich Use-Cases

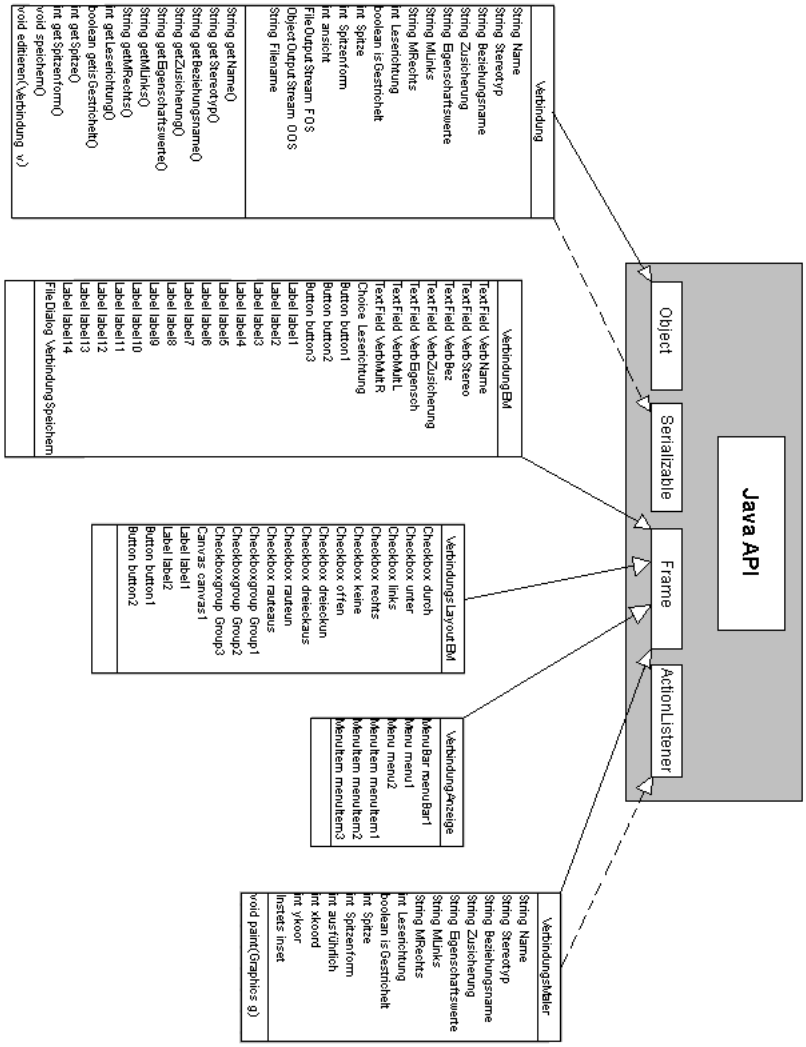

Abbildung 4: Das Klassendiagramm für den Bereich Verbindungen

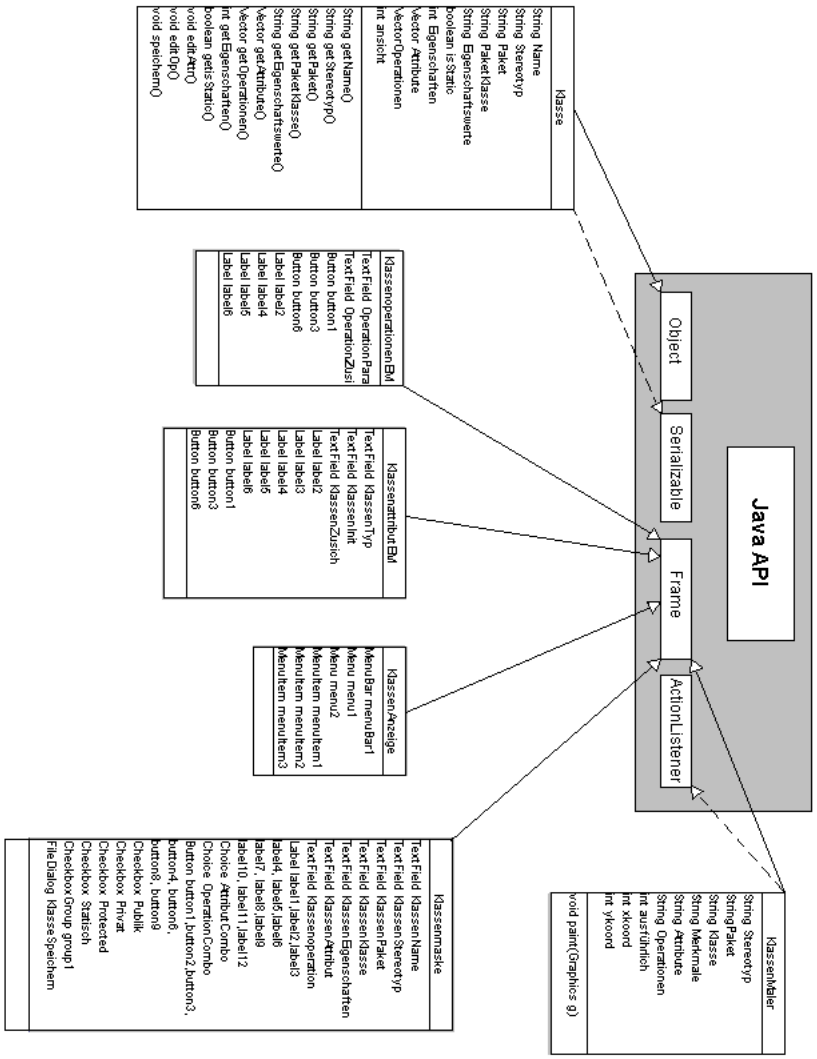

Abbildung 5: Das Klassendiagramm für den Bereich Klassen

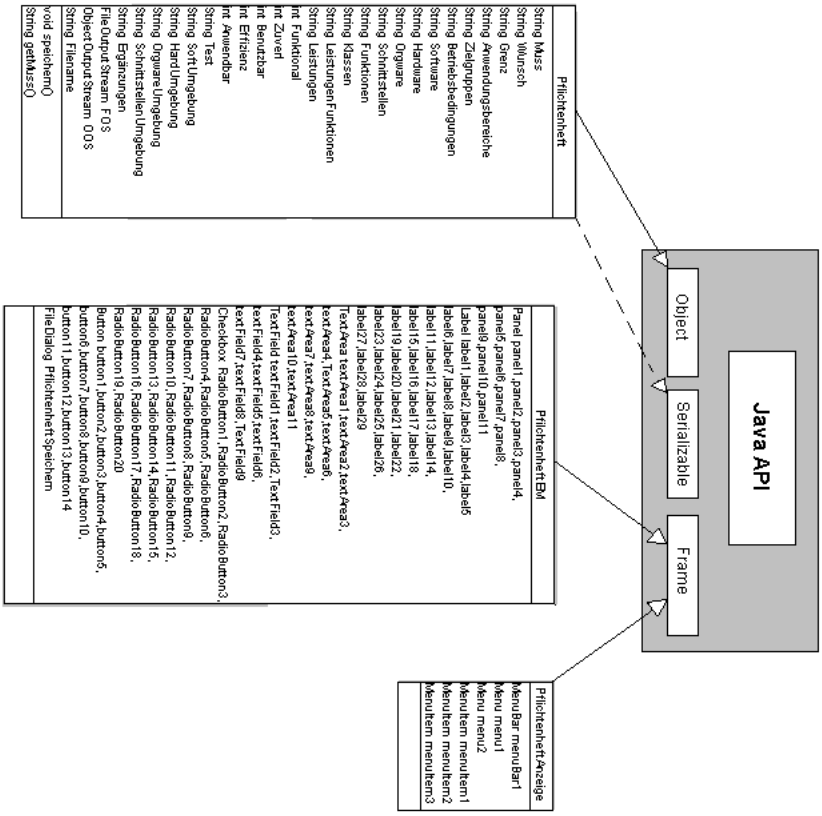

Abbildung 6: Das Klassendiagramm für den Bereich Pflichtenheft

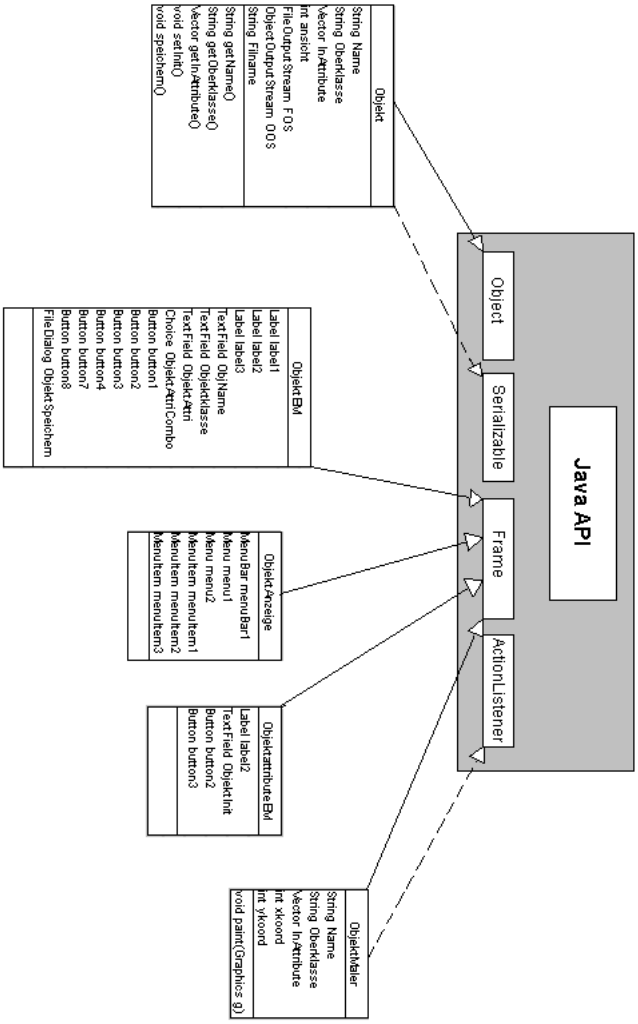

Abbildung 7: Das Klassendiagramm für den Bereich Objekte

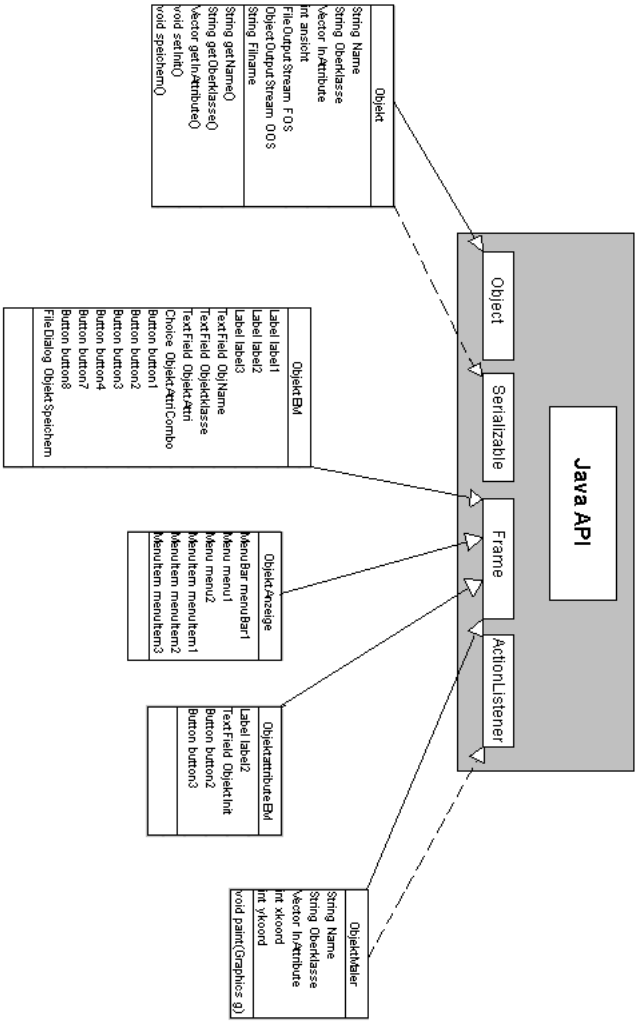

Abbildung 8: Das Klassendiagramm für den Bereich Objekte

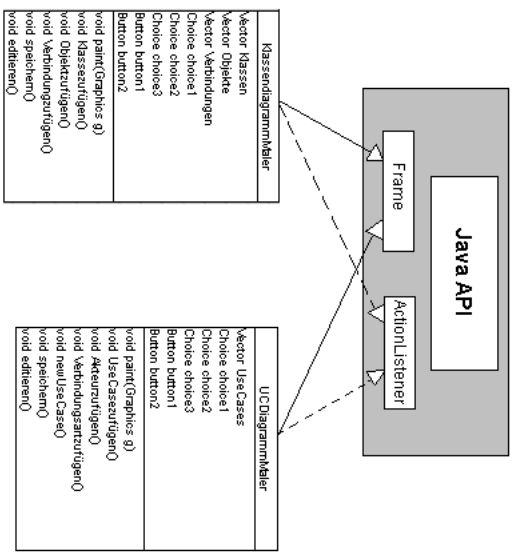

Abbildung 9: Das Klassendiagramm für die Diagrammmaler

6 Produktleistungen: Funktionen

7 Benutzeroberfläche

7.1 Screenshots

Abbildung 10: Menü »Objekt«

Abbildung 11: Menü »UseCaseDiagramm«

7.2 Textuelle Beschreibung der wichtigsten Klassen

7.3 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.

Abbildung 12: Menü »KlassenDiagramm«

Abbildung 13: Menü »Verbindung«

Abbildung 14: Menü »Projekt«

Abbildung 15: Modell der Diagrammanzeige

7.4 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.

7.4.1 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.

7.4.2 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.

7.5 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.

7.6 Klasse BildVonUns

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

7.7 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).

7.8 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.

7.9 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.

7.10 Verbindung

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

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).

7.11 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.

7.12 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.

7.13 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.

7.14 QuitDialog

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

7.15 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.

8 Qualitätsziele

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

9 Testszenarien/Testfälle

9.1 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.

9.2 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.

9.3 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.

9.4 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.

9.5 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.

9.6 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.

9.7 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.

9.8 Einträge speichern

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

9.9 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.

9.10 Programm beenden

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

9.11 Projekt öffnen

Das ausgewählte Projekt wird angezeigt, dh die in ihm enthaltenen Klassen, UseCases usw werden im Hauptframe angezeigt.

9.12 Projekt speichern als

Speichern mit gültiger Namensangabe: die Projektdaten werden auf dem aktuellen Stand abgespeichert. Speichern unter ungültigem, dh schon verwendetem Namen: eine Fehlermeldung erscheint, der User wird aufgefordert, einen anderen Namen einzugeben. Der User bricht die Speicherung ab: das Projekt wird auf dem alten Stand der Speicherung belassen.

9.13 Projekt speichern

Die Projektdaten werden auf den aktuellen Stand des Projekts abgespeichert.

9.14 Drucken

Gültige Auswahl getroffen: dh »UseCase-Diagramm, Klassendiagramm oder Gesamtanzeige drucken« wurde ganz ausgewählt: Die entsprechende Projektkomponente wird gedruckt. Ungültige Auswahl: die einzelnen Diagramme lassen sich nicht nur teilweise ausdrucken.

9.15 Projekt schliessen

Nach einer bestätigten Speicherabfrage wird das Projekt aktuell gespeichert und geschlossen, dh allein die HauptMenüleiste bleibt auf dem Bildschirm. Wird die Speicherung abgelehnt, wird das Projekt auf dem alten Speicherungsstand belassen und geschlossen, dh allein die HauptMenüleiste bleibt zu sehen.

9.16 Beenden

Das laufende Projekt wird wie oben geschlossen und das Programm beendet, indem auch die Menüleiste verschindet.

9.17 UseCase/Objekt/Klasse/Verbindung zum Diagramm hinzufügen

Aus einer Auswahlliste wird ein UseCase/Objekt usw ausgewählt und im Diagramm im dort gewählten Modus angezeigt.

10 Entwicklungsumgebung

Software

1. **Symantec® Visual Café** wurde zur Erstellung eines Teils der Benutzerdialoge verwendet
2. **ArgoUML** zum zeichnen der Diagramme
3. **Kawa®** und **JPad®** wurden zum editieren »per Hand« verwendet

Hardware
übliche **PCs, Sun®** Workstations

Orgware
Es wurde keine besondere Orgware benötigt

Schnittstellen
Es wurden keine besonderen Schnittstellen benötigt

11 Ergänzungen