Projektgruppe geo08

Designbescheibung GeoViewer

Aufgabe 3 SWT Praktikum

Inhaltverzeichnis

1. Allgemeines[1]

1.1. Kurzcharakterisierung

GeoViewer ist eine menügesteuerte graphische Java-Applikation, mit der sich geometrische Konfigurationen visualisieren lassen, die in Form von GEO-Records vorliegen. Die algebraischen Rechnungen zur Bestimmung der Koordinaten der geometrischen Objekte erfolgt mit dem GeoProver-Paket und dem Computeralgebrasystem Maple

1.2. Systemvoraussetzungen

GeoViewer ist als eigenständige Applikation konzipiert, die über eine grafische Nutzerschnittstelle mit gängigen Fenster-Techniken bedient wird. GeoViewer benötigt als Voraussetzungen zum Starten die Java JRE 1.4., Maple (ab Version 5), das GeoProver-Paket für Maple sowie GEO-Records zur Eingabe. Die genaue Lage der einzelnen Ressourcen kann in einer Datei `GeoViewer.rc` spezifiziert werden.

1.3. Beschreibung der Produktumgebung

GEO-Records enthalten Beschreibungen geometrischer Beweisschemata, die in einem speziellen XML-artigen Format abgelegt sind, das auf der GeoCode-Spezifikation aufsetzt. Beides ist näher in der Dokumentation des SymbolicData-Projekts (http://www.symbolicdata.org) beschrieben. Ein GEO-Record enthält neben der Beschreibung der geometrischen Konfiguration eine Liste von unabhängigen Parametern, die für die konkrete Visualisierung mit geeigneten Zahlenwerten zu belegen sind. Die Parameterwerte haben Einfluss auf die Lage einzelner geometrischer Objekte (freier Punkte und Gleiter) und damit auch auf die Lage abgeleiteter Objekte. Verschiedene Parameterwerte ergeben also verschiedene Bilder derselben geometrischen Konfiguration. In diesem Sinne kann dieVisualisierung „dynamisiert“ werden. Zur Interpretation dieser Beweisschemata und Behandlung der algebraischen Fragen wird eine Implementierung des GeoCode-Standards im GeoProver-Paket für das Computeralgebrasystem Maple
herangezogen.

1.4. Abgrenzung

Ein GEO-Record kann auch abhängige Parameter enthalten, deren Werte sich als Lösungen eines Gleichungssystems ergeben, dessen Koeffizienten mit den unabhängigen Parametern variieren. Da die Behandlung von (nichtlinearen) Gleichungssystemen mit variierenden reellen Koeffizienten auch für Maple schwierig ist, werden derartige GEO-Records (vom Gle ichungstyp) nicht visualisiert. GEORecords ohne abhängige Parameter bezeichnet man als Records vom konstruktiven Typ. Eine „Dynamisierung“ der Visualisierungen durch direkte Mausmanipulation ist nicht implementiert, da hierfür weitergehendes Event-Handling erforderlich ist. Parameterwerte einer Visualisierung können nur über ein Dialogfeld geändert werden.

2. Produktübersicht

Die Applikation wird durch den Kommandozeilen-Aufruf

```
java geometry.GeoViewer
```

[1] Abschnitte 1 und 2 sind ausgenommen von Abb. 2.1 von Prof. Gräbe übernommen

im Wurzelverzeichnis der Pakethierarchie gestartet. Während der *Initialisierung* wird die Verbindung zu Maple hergestellt und im Erfolgsfall das Hauptfenster aufgebaut. Dies wird durch das folgende Sequenzdiagramm deutlich gemacht:

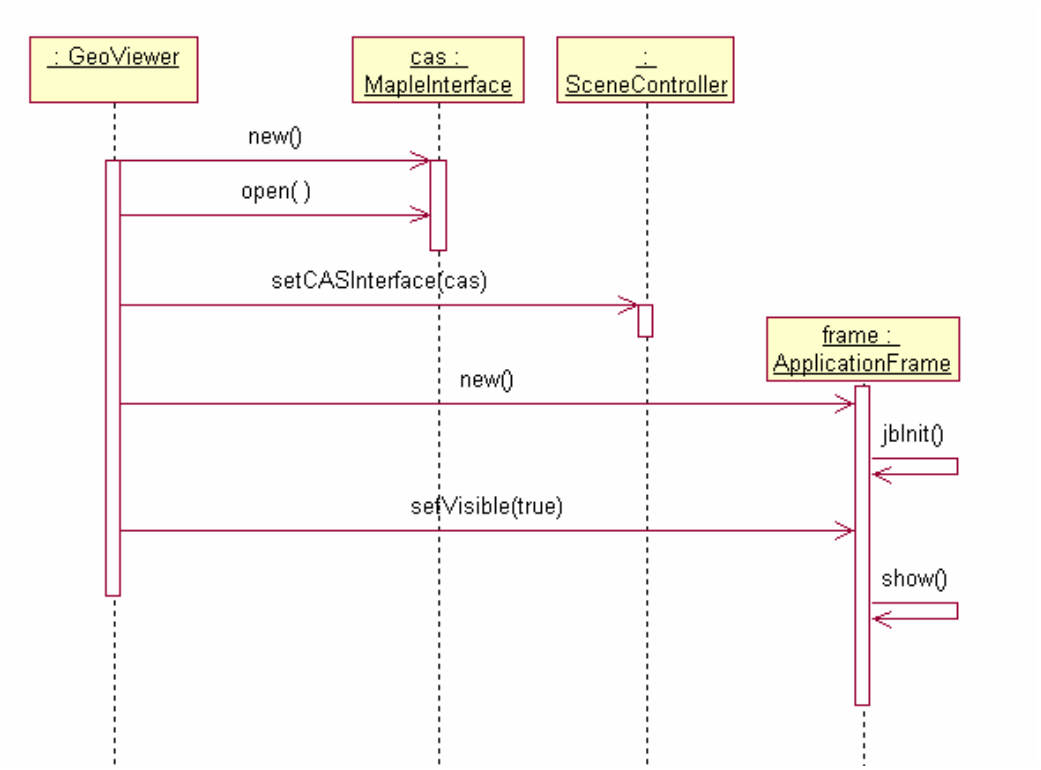
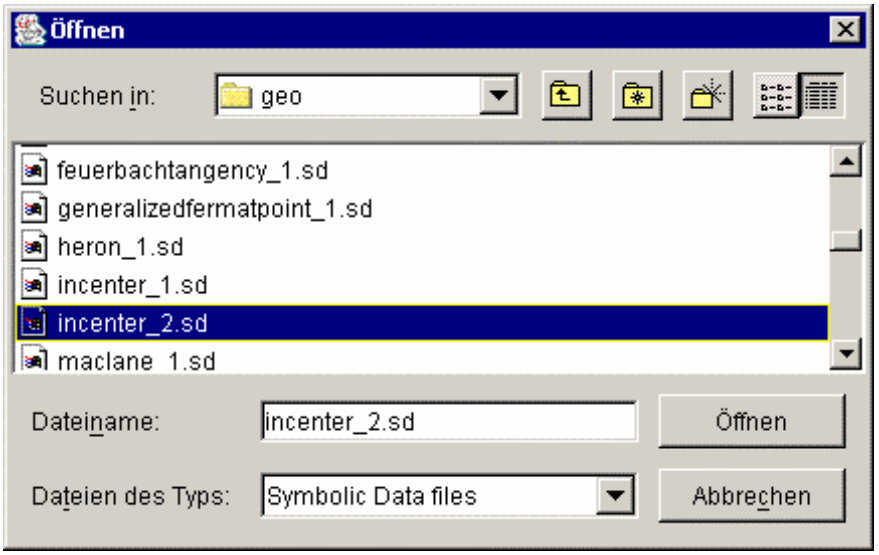
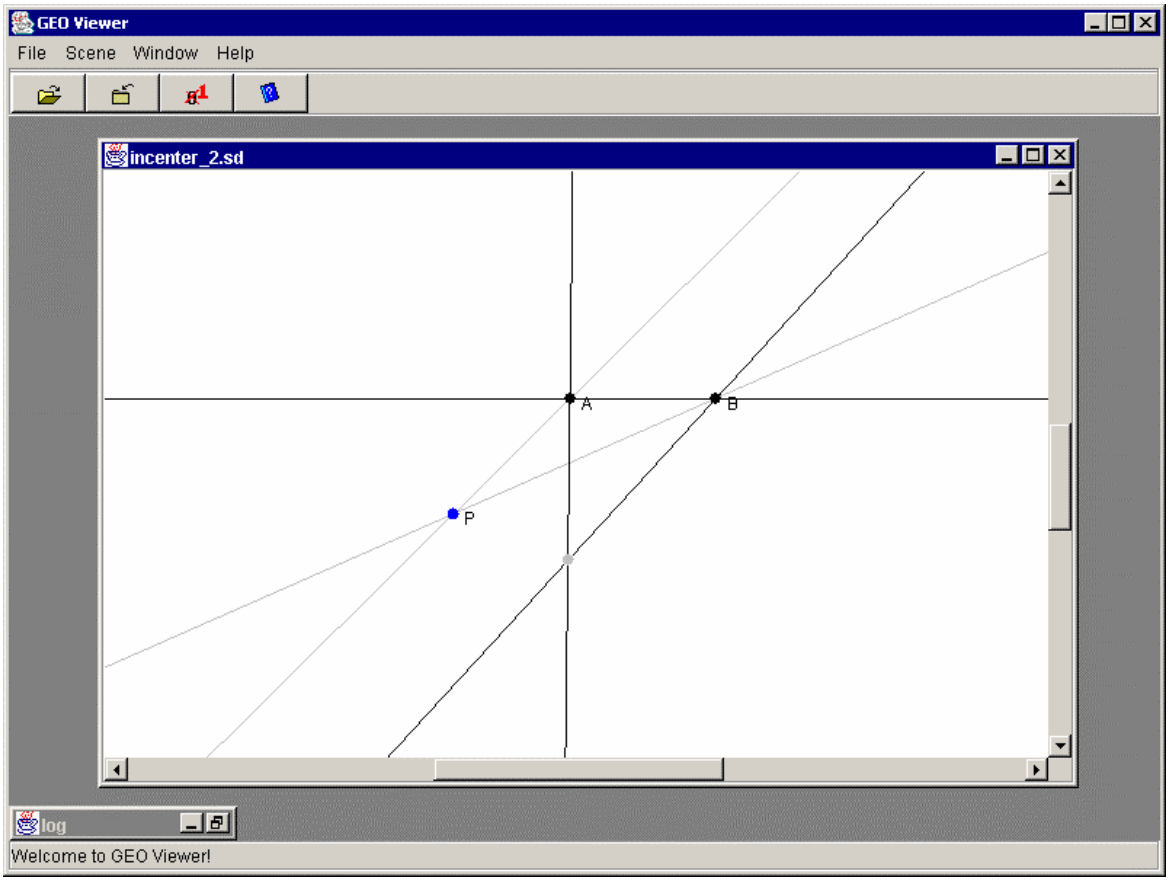


Abb 2.1 Objekt Nachrichten beim Starten der Applikation

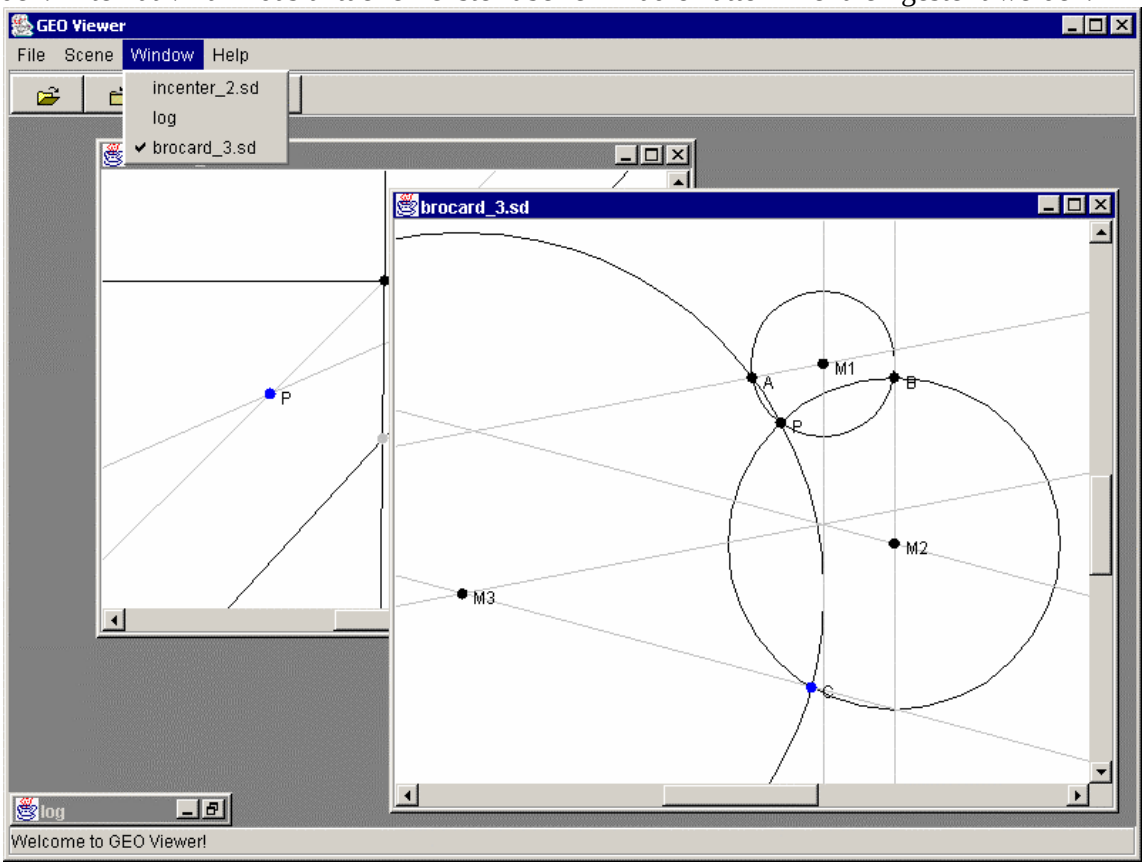
Das *Hauptfenster* enthält am oberen Rand einen Menü- und Toolbalken, am unteren Rand einen Statusbalken und in der Mitte eine Desktop-Fläche, in der im weiteren Verlauf die verschiedenen Zeichenflächen sowie ein spezielles Fenster für Fehlermeldungen geöffnet werden können (siehe Abb 3.2). Über den Menüeintrag *File* oder das linke Icon wird das *Laden eines GEO-Records* gestartet.



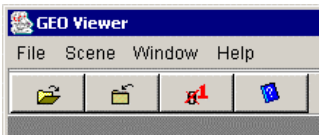
Handelt es sich um ein konstruktives Schema, so wird die zugehörige Visualisierung generiert und in einer neuen Zeichenfläche im Desktopbereich angezeigt und aktiviert. Die Parameterwerte werden dabei zunächst mit Zufallszahlen initialisiert.



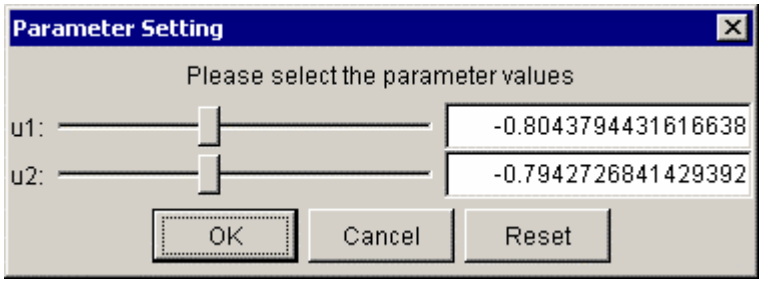
Zwischen verschiedenen Fenstern der Desktopfläche kann mit der Maus hin- und hergeschaltet werden. Alternativ kann das aktuelle Fenster über ein RadioButton-Menü eingestellt werden.



Wird als Fenster eine Zeichenfläche ausgewählt (aktive Zeichenfläche), so werden für diese die Menüpunkte und Icons „Parameterwerte ändern“ und „Zeichenfläche schließen“ aktiviert.



Die Parameterwerte können für die aktive Zeichenfläche über einen Menüeintrag geändert werden, der ein Dialogfenster mit Schiebereglern öffnet. Danach werden die Koordinaten der einzelnen Objekte von Maple neu berechnet und die Visualisierung in der aktiven Zeichenfläche neu dargestellt.



Über einen weiteren Menüpunkt, Icon oder einen speziellen Knopf am Fensterrand wird die aktive Zeichenfläche geschlossen. Die Applikation wird über den Exit-Eintrag im File -Menü beendet.

3 Grundsätzliche Designentscheidung

3.1 Überblick

Da es sich bei der Applikation um eine Geometriesoftware handelt, wurde darauf Wert gelegt eine angemessene Bildschirmpräsentation zu realisieren, die eine übersichtliche und kompakte Oberfläche zur Betrachtung von einzelnen Szenen[2] unterstützt. Dies wurde mittels eines Multi Dokument Interface implementiert das multiple Frames zur Darstellung verschiedener geometrischer Sachverhalte bereitstellt. Diese bieten dem Benutzer keine Mausinteraktion. Parameter der Szenen können lediglich statisch über einen separaten Dialog geändert werden.

Zur Darstellung der Geometrien musste aber zunächst das Einlesen (Parsen) von GEO-Records und der Interpretation der daraus resultierenden geometrischen Beweisschemata mittels eines CAS[3] realisiert werden. Dies stellt auch die interne Kernfunktionalität dar.

[2] Eine Szene wird durch ihre geometrischen Objekte definiert und logisch in einer Datei abgespeichert.
[3] Computer Algebra System

Es wurde versucht die logische Abtrennung von Kernfunktionalität und Bildschirmpräsentation mittels der Packetstruktur umzusetzen. Dabei spielen die Packete **`geometry/view`** und **`geometry/contoller`** eine zentrale Rolle. Richtet man sich nach dem MVC-Konzept ist dies allerdings nicht gut gelungen. Im nächsten Abschnitt soll dies näher erläutert werden.

3.2 MVC

MVC setzt sich aus den drei Klassen Model, View und Controller zusammen. Das Model-Objekt stellt die Kernfunktionalität und das Anwendungsobjekt dar, das View-Objekt seine Bildschirmrepräsentation, und das Controller-Objekt bestimmt die Möglichkeiten, mit denen die Benutzungsschnittstelle auf Benutzereingaben reagieren kann. Die klare Schichtentrennung die daraus resultiert wurde nur ungenügend in der Applikation umgesetzt. Aufgrund der Namensgebung bei den Packeten kann man allerdings davon ausgehen, dass dies zumindest versucht wurde.

Auffällig ist, dass die Sichten auf die Anwendungsdaten (View) und die Benutzerschnittstelle (Controller) in ein Packet **`geometry/view`** gepackt wurden. Dieses Konzept kommt dem UI-Delegation[4] Paradigma gleich. Allerdings schliesst es auch aus, dass weitere Packete unter der Bezeichnung controller existieren. Das Packet **`geometry/controller`** stellt somit eine Verletzung der UI-Delegation dar und zudem befinden sich in ihm Klassen der Kernfunktionalität wie SceneController[5] und die geometrischen Klassen, also dem Modell.

Weitere Klassen der Kernfunktionalität, d.h. des Parsers und der CAS Schnittstelle befinden sich wiederum alleine ausserhalb dieser Strukturen (**`geometry/parse`** und **`geometry/cas`**). Es existiert kein Packet model.

Veranschaulicht wird dieser Sachverhalt in Abbildung 4.1. Im Folgenden soll trotz Mangel an logischer Packetunterteilung auf die Konzepte von Model, View und Control eingegangen werden.

3.2.1 Model

Zum Modell der Applikation und somit zur Kernfunktionalität zählen das Parsen, Berechnen und Speichern der geometrischen Sachverhalte. Das Parsen wird im Packet **`geometry/parse`** und mit Hilfe des **`java_cup`** Packetes realisiert. Zum Berechnen der geometrischen Beweisschemata dient das Maple Interface das im Packet **`geometry/cas`** implementiert wurde. Die interne Definition der Geometrien in Klassen und die Speicherung in einem Java Container ist in **`geometry/controller`** und in **`geometry/view`** realisiert. Im Folgenden wird dies im Detail aufgeführt.

3.2.1.1 Parsen

[4] Darstellung und Benutzerschnittstelle werden in einem Packet vereinigt
[5] siehe Punkt 3.1.1.3 SceneController würde einer Zuordnung zu einem Model Packet eher entsprechen

Da die geometrischen Sachverhalte in Form von Szenen als GEO-Records textuell in einer Datei definiert sind, wird nach öffnen solch einer Datei (mit Endung sd) zunächst eine Klassenmethode **`SceneController.load()`** gestartet, die den Parser GeoParser initiiert, der wiederum zusammen mit verschiedenen ‚Helfer Klassen' arbeitet. Dazu gehören die Klassen **`GeoSymbol`**, **`GeoScan`**, **`Translator`** und zuletzt **`GeoScan`**, das das Interface **`java_cup.runtime.Scanner`** implementiert.

Beim Parse-Vorgang wird sichergestellt, dass Ausdrücke in Form von korrekten geometrischen Beweisschemata erzeugt werden, die von Klassen die auf das CAS Interface erweitern weiterverwendet werden können (siehe nächster Punkt).

3.2.1.2 Berechnen

Zum implementieren einer Schnittstelle zu einem Algebra System steht uns das Interface **`geometry.cas.CASInterface`** zur Verfügung. Dieses definiert Prototyp-Methoden mit deren Hilfe eine einheitliche Schnittstelle zum Verbinden zu einem Algebra System dargeboten wird. Hier eine Auswahl der Klassenmethoden:

`CASInterface.open()`
Öffnet und initialisiert die Verbindung zum CAS. Es werden gebufferte Streams(Input, Output) zur Datenübertragung verwendet.

`CASInterface.close()`
Schliesst die Verbindung.

`CASInterface.evalutate(Expression expr)`
Wertet einen geometrischen Ausdruck **`expr`** aus.

`CASInterface.transmit(Expression expression)`
Übermittelt den aktuellen Wert des Ausdruckes **`expression`** zum CAS

Will man also in Zukunft eine Schnittstelle zu einem CAS einführen muss man u.a. diese Methoden implementieren. Zukünftige Versionen sollten die Möglichkeit anbieten verschiedene Schnittstellen über einen Dialog auszuwählen und zu konfigurieren.

Der Applikation stellt z.Z. Schnittstellen zu Maple (**`geometry.cas.MapleInterface`**) und Mupad (**`geometry.cas.MupadInterface`**) zur Verfügung. Wobei allerdings das letztere nicht Funktionsfähig ist – die Applikation benutzt Maple.

Die Klasse **`MapleInterface`** implementiert also **`CASInterface`** und entwirft die Methoden um die eingelesenen und formatierten Geo-Records die in Form von Ausdrücken angenommen werden an die Maple Applikation zu übergeben. Zunächst ist es aber erforderlich die Verbindung aufzubauen (*‚das Interface initialisieren'*). Steht die Verbindung können die geometrischen Kommandos und Parameter ausgewertet werden und dem GeoViewer zur Darstellung zur Verfügung gestellt werden. Dies geschieht in Form eines Java-Containers der Objekte vom Typ GeoElemts enthält, die über draw() Funktionen verfügen und das direkte Zeichnen ermöglichen.

3.2.1.3 Speichern

Alle geometrischen Objekte einer Scene werden in einem Container innerhalb eines **`geometry.view.SceneFrame`** Objekts abgespeichert. Jedes Objekt besitzt jeweils eine Referenz auf eine Instanz von Typ SceneController. Ein Objekt vom Typ **`geometry.view.ScenePane`**, das in SceneFrame instanziiert wird, ist zum Zeichnen jedes einzelnen Elements aus diesem Container verantwortlich. Beziehen tut **`SceneFrame`** die Elemente mittels **`SceneController.getElements()`** das dem Konstruktur der Klasse **`SceneFrame`** mitgegeben wird. In der folgenden Abbildung sieht man eine Darstellung dieser Sachverhalte.

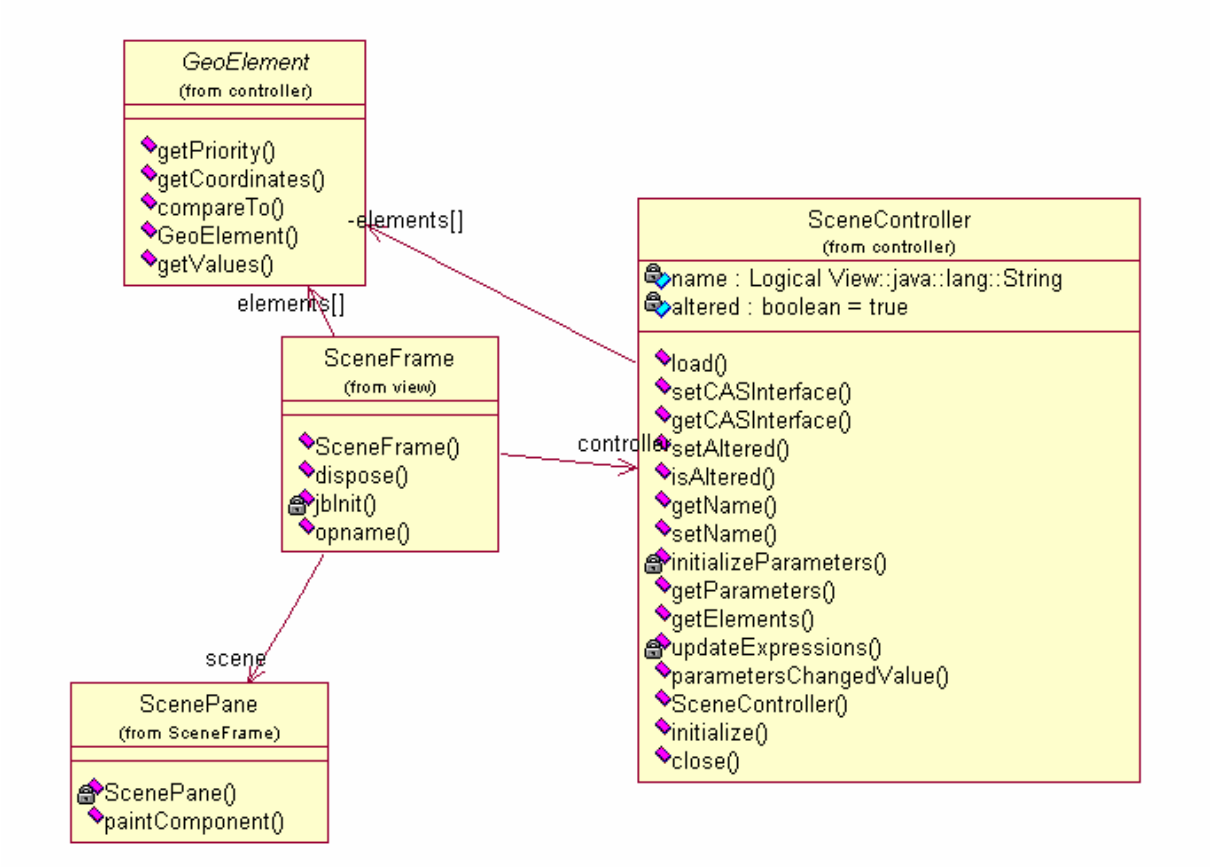


Abb. 3.1 Beziehung von View und Modell

Wie man deutlich erkennt bietet die Klasse SceneController die eigentliche Kernfunktionalität des Anwendungsmodels. Es initialisiert nicht nur die CAS Schnittstelle sondern lädt auch die Geo-Record Dateien, startet den Parser und stellt die geometrischen Elemente den zeichnenden Klassen zur Verfügung. In einer zukunftigen Version würde man diese Klasse in ein Packet ‚model' überführen und als Basisklasse für die weiteren Modell-Komponenten kennzeichnen. Dies wird durch die Abbildung 4.2 verdeutlicht.

Die verschiedenen geometrischen Klassen, die die eigentlichen Komponenten einer Szene darstellen, leiten sich alle ausnahmslos von der Superklasse GeoElements ab.

3.2.2 View

Die Applikation soll es ermöglichen mehrere Szenen auf dem Bildschirm darzustellen. Eine Szene wird durch eine Datei repräsentiert. Wird die Szene geöffnet, so wird innerhalb des Hauptfensters auf dem desktopPane vom Typ GeoDesktopPane ein neuer Frame hinzugefügt der einen ScenePane enthält und verantwortlich für die Darstellung aller Elemente ist.
Dies wird in der folgenden Abbildung veranschaulicht.

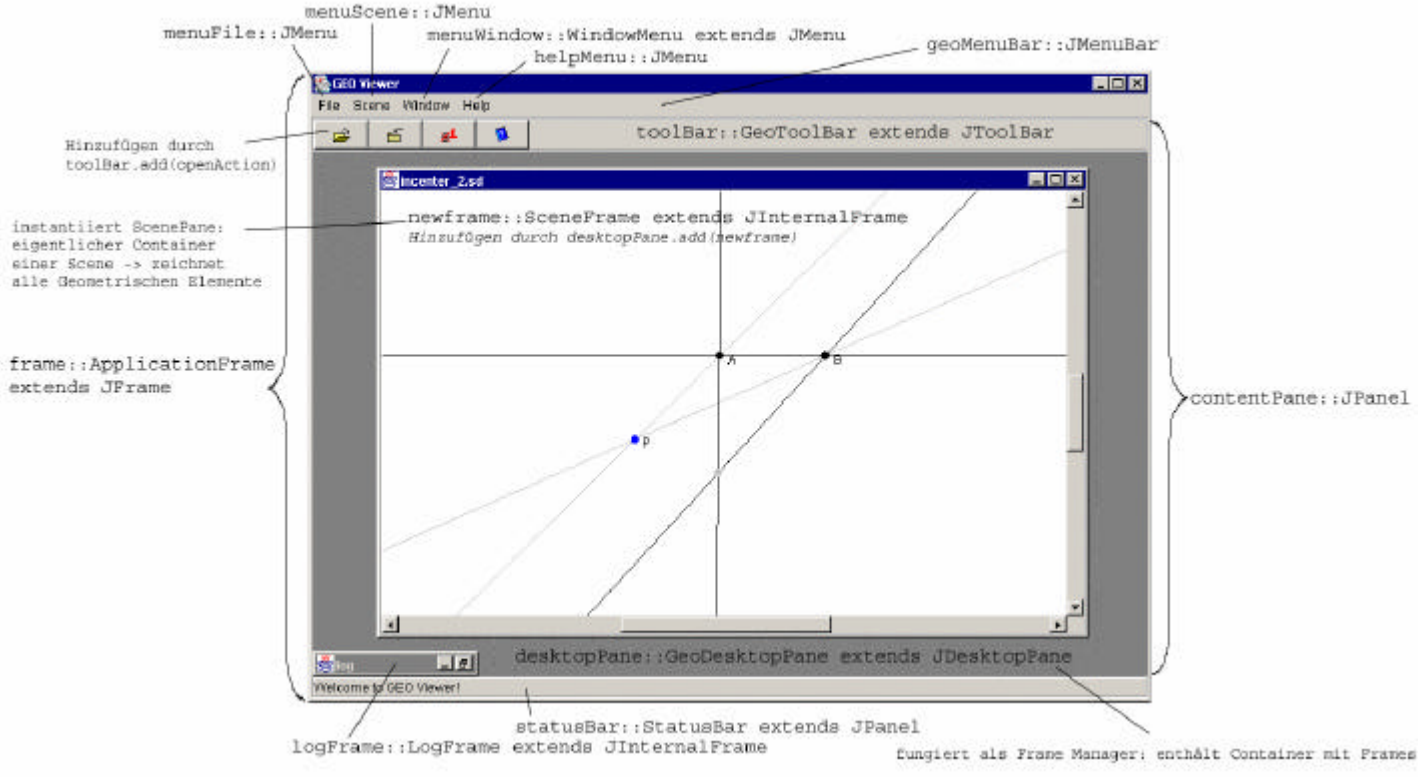


Abb 3.2 Die GUI und ihre Objekte

Die Klassenstruktur und die Assoziationen kann man sich auch nochmal in Abbildung 4.4 näher verdeutlichen.

3.2.3 Control

Die Benutzungschnittstelle reagiert auf Benutzereingaben wenn ein Fenster in seiner Form und Position geändert wird und falls eines der Aktionen aus Abschnitt 3.3 ausgeführt wird. Dazu zählen das Drücken eines der 4 Buttons oder das Klicken im Hauptmenu.

3.3 Eventhandling

Das Eventhandling für den GeoViewer besteht aus 2 Modulen. Zum einen wurde in der Klasse ApplicationFrame die Methode `processWindowEvent(WindowEvent e)` überschrieben um `WINDOW_CLOSING` Events abzufangen und schliesslich die Methode dispose() auf das frame::ApplicationFrame Objekt aufzurufen. Dieses gibt die zugeordneten Windows Resourcen frei und entfernt das Fenster aus der Owner-Child-Registrierung.

Zum anderen werden Button Klicks und Menuauswahl Klicks abgefangen. Dies wurde mittels Klassen implementiert, die sich von der Klasse `AbstractAction` aus `java.awt.event.*` ableiten. Diese befinden sich mit in der Datei ApplicationFrame.java. Folgende Klassen wurden implementiert: `ExitAction, OpenAction, CloseAction, ParameterAction` und `AboutAction.` Dabei wurden der Konstruktor und die Methode `actionPerformed (ActionEvent e),` welche die Abhandlung zur jeweiligen Action gewährleistet, realisiert.

Das Hauptfenster, also das Objekt **`frame::ApplicationFrame`** initialisiert Instanzen zu jeweils einer dieser Klassen und registriert diese an den entsprechenden Elementen (Buttons und Menuitem) der Benutzerschnittstelle. Wird eines von diesen durch Mausklick aktiviert so löst dies die Ausführung der **`actionPerformed()`** Funktion des jeweiligen Objektes aus, dass zu dem betreffendem Bildschirmelement assoziert ist. Die folgende Abbildung veranschaulicht die Abhängigkeiten.

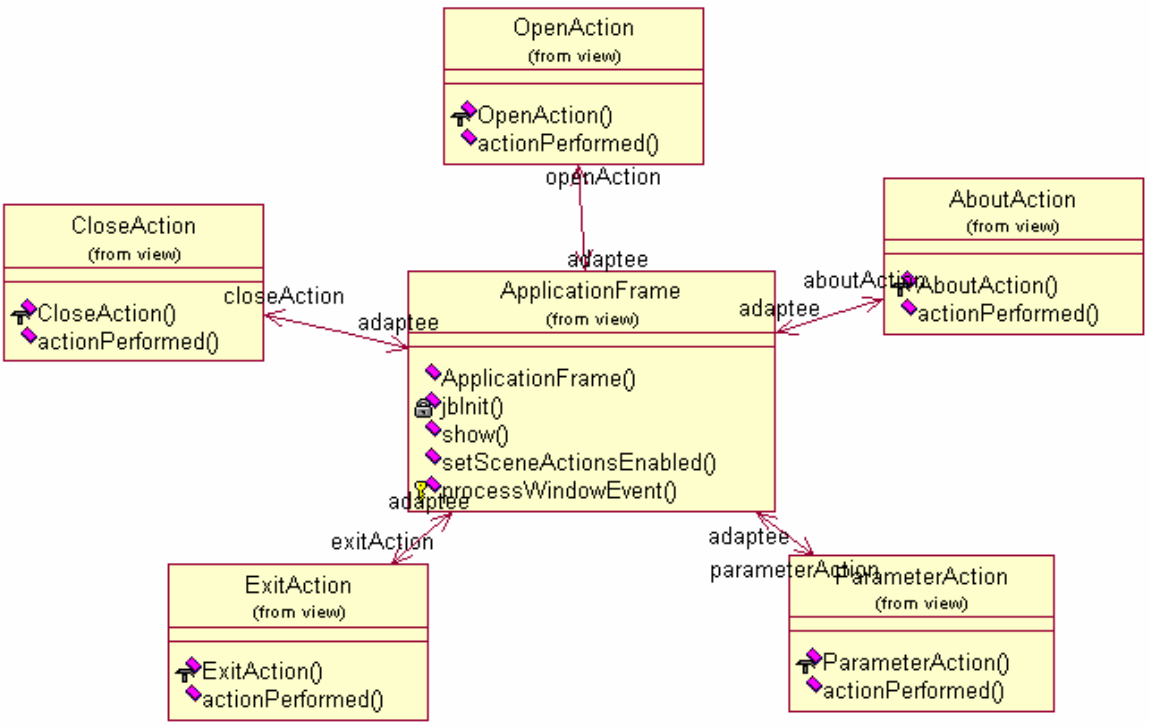


Abb. 3.1 Klassenbeziehung des Hauptfenster zu seinen Aktionen

Folgende Aktionen werden durch die jeweiligen actionPerformed() Methoden initiiert:

`OpenAction.actionPerformed()`

Ein Dialog zum Öffnen von Dateien mit Endung *sd* wird aufgerufen. Ein **`newFrame`** Objekt wird erzeugt (siehe Abb 3.2) dessen Konstruktor ein SceneController Objekt entgegen nimmt. Als Übergabeparameter an den Konstruktor wird die Klassenmethode **`SceneController.load()`** geladen in der das Parsen und die Kommunikation zu Maple stattfindet.

`CloseAction.actionPerformed()`

Schliesst den aktiven Frame. Hierzu wird **`die dispose()`** Funktion aufgerufen.

`ParameterAction.actionPerformed()`

Zeigt einen Dialog an in dem Parameter der aktuellen Szene geändert werden können. Diese werden über den SceneController des aktiven SceneFrames (**`activeFrame::SceneFrame`**) an das CAS übermittelt.

`AboutAction.actionPerformed()`

Zeigt einen Dialog mit einer kleinen Information über das Programm und den Urheber an.

`ExitAction.actionPerformed()`

Beim Schliessen des Hauptfensters werden mit Hilfe der **`dispose()`** Methode (**`ApplicationFrame.dispose()`**) die zugeordneten Windows Resourcen frei gemacht und das Fenster entfernt.

3.4 Aussicht

Bei Weiterbenutzung des Codes wäre es von Vorteil die Packetstruktur noch einmal hinsichtlich eines klaren MVC-Konzepts zu überarbeiten. Es wäre weiterhin empfehlenswert das Programm mit einem Editor Fenster auszustatten um das Ändern/Anpassen von sd-Dateien zu vereinfachen. Eine reichhaltige Hilfe, die die Geo-Records erklärt wäre auch empfehlenswert. Die Auswahl und Konfiguration von CAS Schnittstellen sollte Dialogbezogen realisiert werden.

Nichtsdestotrotz spielen statische Geometrieanwendungen wie der GeoViewer heute eine weniger bedeutende Rolle auf dem Markt für Geometrie-Lernsoftware. Es können zwar komplizierte geometrische Sachverhalte selbst mittels Texteditor erstellt und dargestellt werden aber die Benutzerfreundlichkeit und der Zugang zu den Inhalten ist für Laien schwer fassbar.

Plant man die Erstellung eines dynamischen geometrischen Systems so könnte man allerdings von der CAS Schnittstelle gebrauch machen. Nicht als zentrales Anwendungsobjekt sondern als Erweiterung für Maple und Geo-Records Spezialisten.

4 Packet- und Klassenstruktur

4.1 Packetstruktur

Die folgende Abbildung der Packetstruktur zeigt das Vorhandensein von **`controller`** und **`view`** Packet aber das Fehlen eines **`models`**. Wie schon erwähnt wurden View und Controller Komponenten im das Packet **`view`** zusammengefasst (UI-Delegation). Beim Betrachten des unten aufgeführten Packetdiagrammes fällt allerdings noch die starke Verkettung von **`cas`**, **`parser`** und **`controller`** Packeten auf. Dies hängt damit zusammen, dass wie schon erwähnt alle Klassen im Packet **`controller`** eigentlich genau wie **`cas`** und **`parse`** Packet der Kategorie Model, also der Kernfunktionalität, zugeordnet werden sollten.

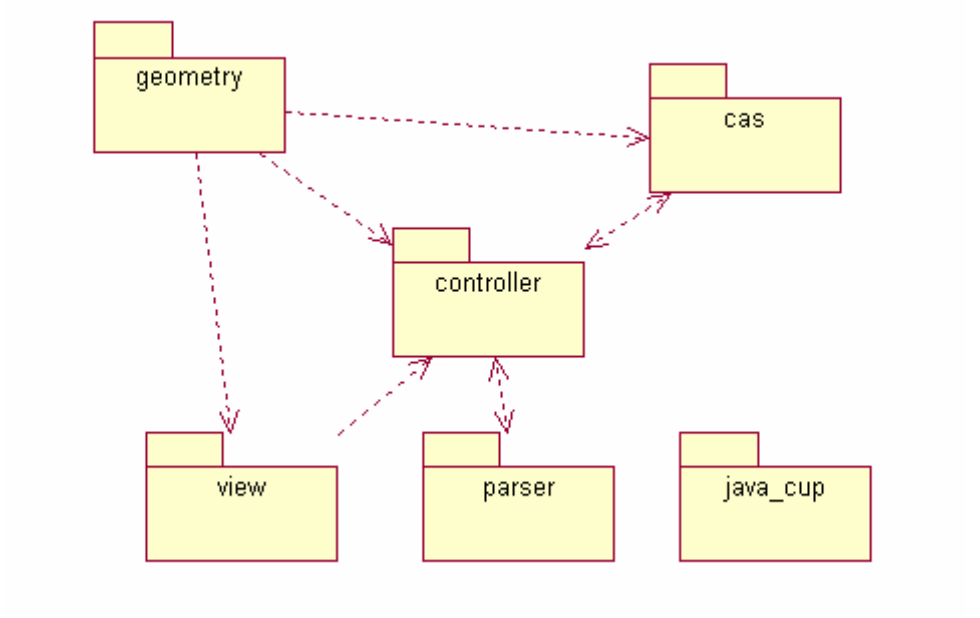


Abb 4.1Übersicht der Packete

Die nächste Abbildung soll zeigen wie eine mögliche Packetstruktur in zukünftigen Versionen unter Einberufung des UI-Delegations Konzept aussehen könnte. Die Klassen in **`model`** verwalten dabei die Szenen, das Parsen und das CAS Interface.

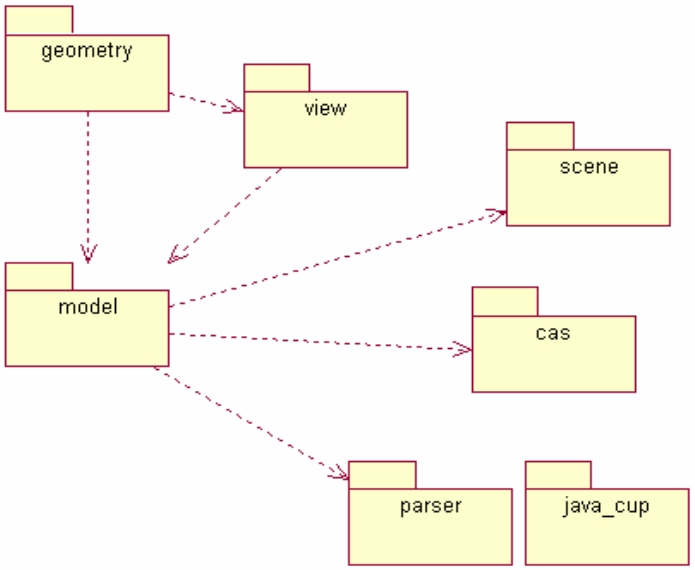


Abb 4.2 Packetstruktur Aussicht (UI-Delegation)

4.2 Klassenstruktur

Es ist nicht von Vorteil hier die gesamte Klassenstruktur darzustellen. Neben denen im Dokument schon erwähnten soll dieser Abschnitt nur die Interessantesten zeigen auf die im Text mindestens einmal Bezug genommen wurde.

Das folgende Diagramm skiziert die Spezialisierung der **geometrischen Klassen** die eine Szene charakterisieren. GeoElement bietet elementare Funktionalität und eine gemeinsame Schnittstelle aller geometrischen Instanzen die davon ableiten. Jedes SceneController Objekt benutzt dies um einen Container mit GeoElements zu halten und den grafischen Komponenten zur Verfügung zu stellen.

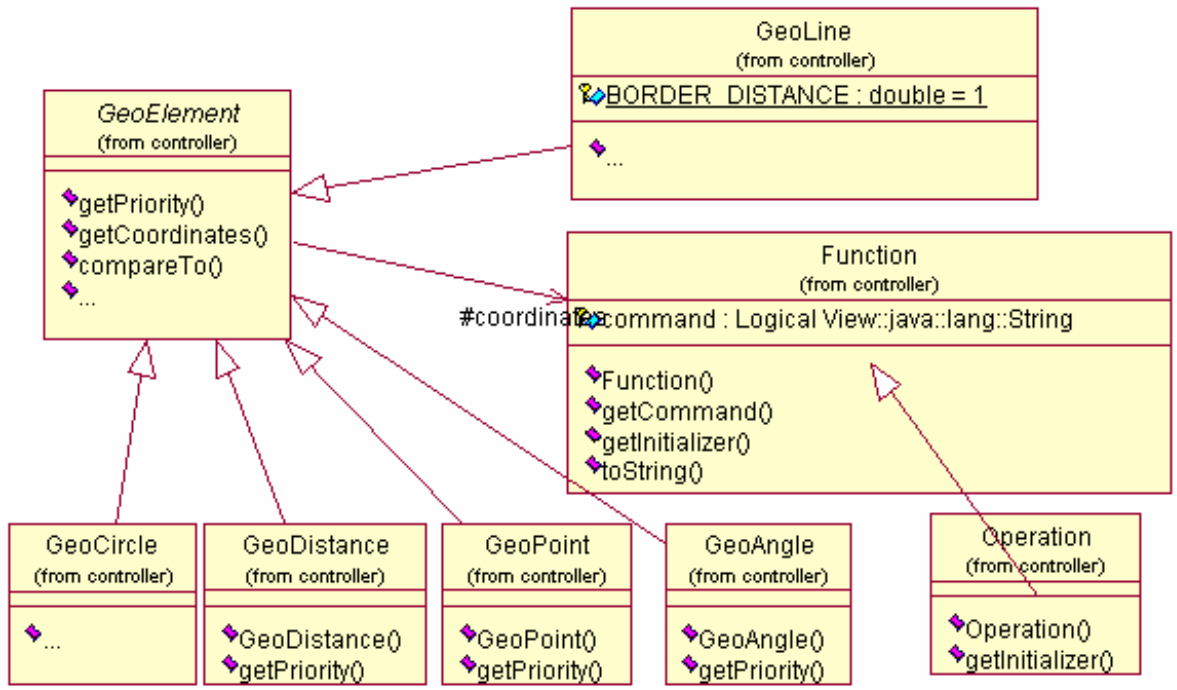


Abb. 4.3 Klassendiagramm der geometrischen Klassen

Die nächste und letzte Abbildung zeigt das Klassenaufbaudiagramm der GUI wie sie unter Abschnitt **3.2.2 View** schon Schematisch mit ihren Objekten in der Oberlfäche angezeigt wurde. Die folgende *Abb. 4.4* verdeutlicht auch die Assoziationen untereinander.