Beschreibung der funktionellen Aspekte der Applikation „GeoViewer“

1. Allgemeines

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 die Visualisierung „dynamisiert“ werden.

Zur Interpretation dieser Beweisschemata und Behandlung der algebraischen Fragen wird eine Implementierungdes 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 Gleichungstyp) 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.

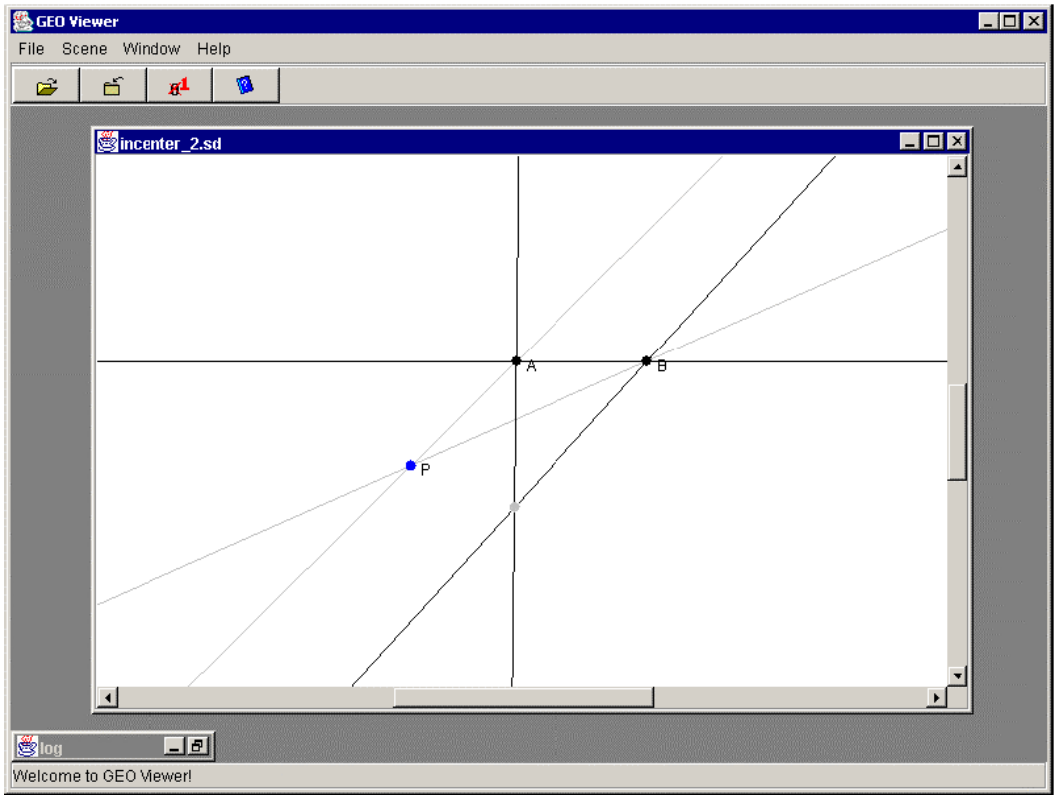
2. Produktübersicht

Die Applikation wird durch den Kommandozeilen-Aufruf

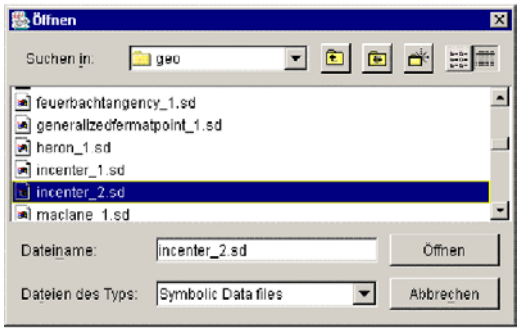
java geometry.GeoViewer

im Wurzelverzeichnis der Pakethierarchie gestartet. Während der *Initialisierung* wird die Verbindung zu Maple hergestellt und im Erfolgsfall das Hauptfenster aufgebaut.

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 geöffnet werden können sowie ein spezielles Fenster für Fehlermeldungen zu finden ist.

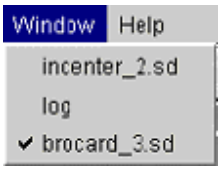


Das Menu hat die Einträge File, Scene, Window, Help. Mit File  wird das *Laden eines GEO-Records* gestartet. Dies geschieht durch einen Dateidialog, in welchem man entweder in einer Verzeichnisstruktur per Mausklick oder in einem Textfeld per Tastatur das zu ladende Geoprojekt, eine .sd Datei auswählt.



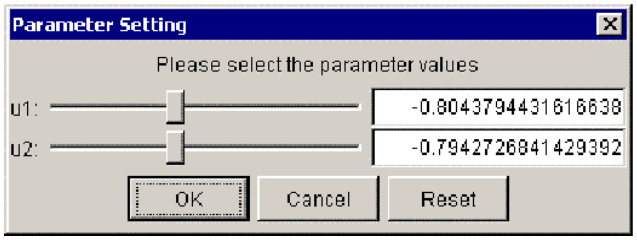
Über Scene 

gelangt man in einen Dialog zur Änderung der Parameter. Im Window Menupunkt kann man zwischen allen geöffneten Zeichenflächen und dem Fehlerprotokoll hin und her springen, es kann immer nur eins der Fenster aktualisiert und in den Vordergrund geholt werden.



Über Help gelangt man zu einer Infobox.

Der Toolbalken hat die Funktionen Zeichenfläche öffnen  , Zeichenfläche schließen  , Parameter verändern  und die Hilfefunktion  .
Über Zeichenfläche öffnen gelangt man in den gleichen Dialog zum Öffnen eines Geoprojekts wie über den Menueintrag File.
Mit Zeichenfläche schließen wird die aktuelle Zeichenfläche geschlossen, dies kann auch optional über den  Button des Zeichenfensters selber geschehen. Mit Parameter ändern wird ein Dialog zur Veränderung aller freien Parameter des aktuellen Zeichenfensters geöffnet.



Dieser enthält für jeden Parameter einen Schieberegler und ein Feld mit dem Wert als Gleitkommazahl. Optional kann der Wert mit beiden geändert werden. Die Veränderung der Parameter kann nun entweder angenommen „OK“, nicht beachtet „Cancel“ oder zurückgesetzt „Reset“ werden. Geschlossen wird der Dialog über dessen  Button.

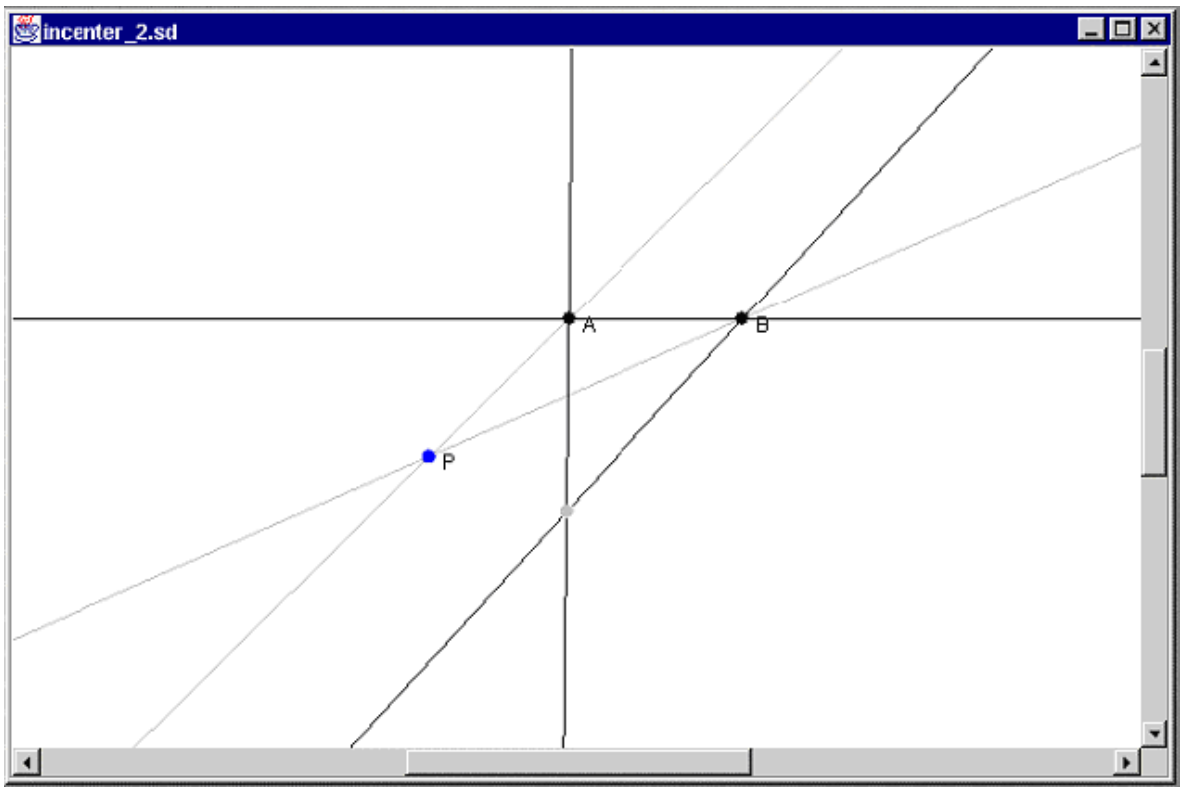
Die Hilfefunktion öffnet wie der Menueintrag Help eine Infobox.

Die beiden Einträge Zeichenfläche schließen und Parameter ändern sind nur dann aktiv wenn auch wirklich eine Zeichenfläche existiert, dies äußert sich dadurch das die beiden Schaltflächen bei Inaktivität grau unterlegt sind .

Im Statusbalken wird die eben ausgeführte Aktion für eine gewisse zeit angezeigt danach erscheint wieder ein vorgegebener Text.

Die Desktopfläche enthält beim Öffnen nur ein Fehlerprotokoll 

, die Zeichenflächen werden später dazu geladen. Das Fehlerprotokoll zeigt alle Error und Ausnahmewarnungen in einem Textfeld an. In der Zeichenfläche wird das Geoprojekt gezeichnet,



sein Name erscheint im oberen Balken des Fensters 

, weiterhin verfügt die Zeichenfläche über ein horizontalen und einen vertikalen Scrollbalken. Es kann zwischen allen Zeichenflächen und dem Fehlerprotokoll per Mausklick oder optional dem Menupunkt Window hin und her geschaltet werden. Alle Zeichenflächen lassen sich minimieren maximieren oder schließen 
. Letzteres gilt auch für die Applikation selber.

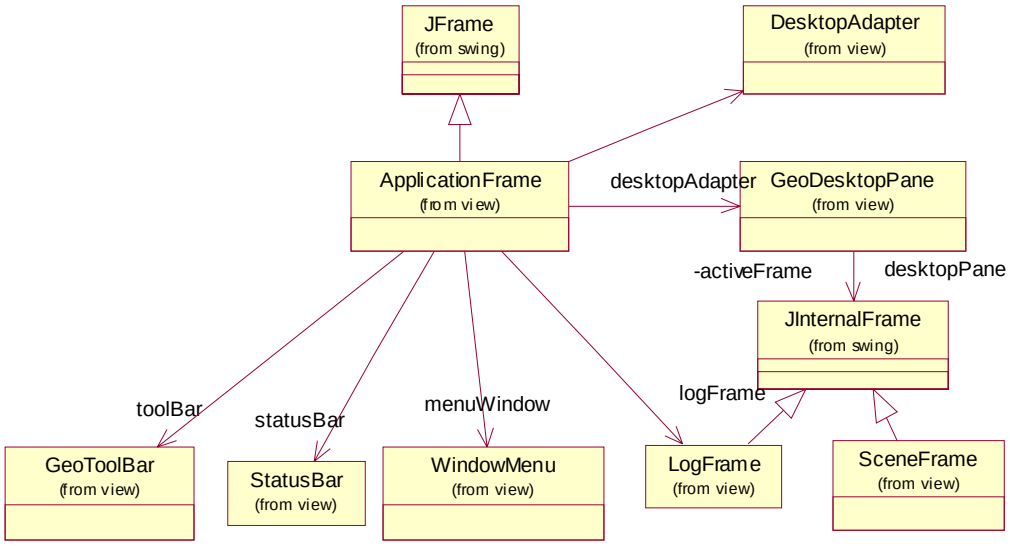
3. Grundsätzliche Design-Entscheidung

View

Der View des Programms GeoViewer setzt sich zusammen aus einem Hauptfenster dem ApplicationFrame, abgeleitet von JFrame, einem Dialog zum Öffnen einer Datei (geoFileChooser abgeleitet vom Typ JFileChooser) und einem Dialog zur Änderung von unabhängigen Parametern eines Geoprojekts.

ApplicationFrame

Das ApplicationFrame besteht aus Menü-, Status- und ToolBars, sowie einem Desktop dem GeoDesktopPane. Weiterhin sind mit ihm alle EventHandler (DesktopAdapter, OpenAction, usw.) verbunden.



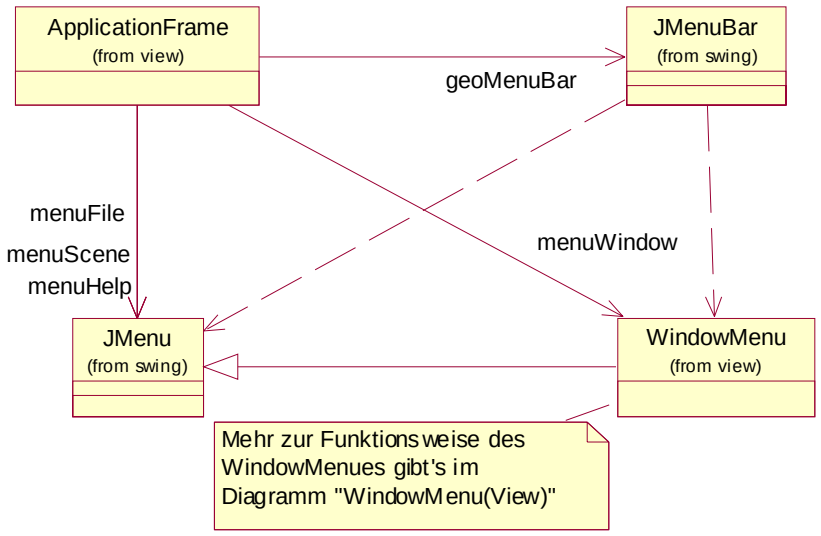
MenuBar

Die MenuBar (im Hauptfenster ganz oben) ist eine Instanz von, der von javax.swing gelieferten Klasse, JMenuBar. Sie enthält die Einträge menuFile, menuScene und menuHelp vom Typ JMenu. Die Inhalte und Funktion dieser Einträge werden über AbstractAction-Objekte definiert. Des weiteren enthält die MenuBar noch den Eintrag menuWindow vom Typ WindowMenu.

WindowMenu

WindowMenu ist eine Unterklasse von JMenu und erlaubt dem Nutzer ein aktives Fenster auf dem Desktop auszuwählen. Bei Initialisierung wird eine Referenz auf das Desktop-Objekt (GeoDesktopPane) übergeben. Diese gibt Auskunft über eventuell offene Fenster. Im Menü selber werden die Frames mit Hilfe von FrameButtons dargestellt. Dabei wird jedem FrameButton (abgeleitet von JCheckBoxMenuItem, implementiert ActionListener, PropertyChangeListener) ein JInternalFrame-Objekt übergeben, um seine Beschriftung bestimmen zu können. Die Auflistung aller Frames wird in einer HashTabelle (java.util.HashMap) gespeichert. Dort enthält jeder Listeneintrag ein JFrameObjekt, vom Typ JInternalFrame, und den zugehörigen FrameButton im WindowMenu. Über verschieden Methoden des WindowMenu können Button hinzugefügt, gewechselt, gelöscht sowie aktiviert und deaktiviert werden. Um zu gewährleisten das immer nur ein Fenster aktiv ist

werden alle Buttons in einem java.swing.ButtonGroup-Objekt verwaltet. Dies sorgt dafür das immer nur ein Button markiert ist. Um auch den Zustand ‚kein Button markiert' zu ermöglichen wird standardmäßig ein DummyButton in der Gruppe angemeldet, der keinen Eintrag hat und deshalb unsichtbar ist. Um die Aktualität des WindowMenu zu gewährleisten muss es über eventuelle Events des Desktop unterrichtet werden. Dies erledigt die innere Klasse DesktopAdapter (implementiert DesktopListener) der Klasse ApplikationFrame, welche weiterhin dafür sorgt, dass die FrameButtons gegebenenfalls gelöscht, beziehungsweise hinzugefügt oder gewechselt werden.

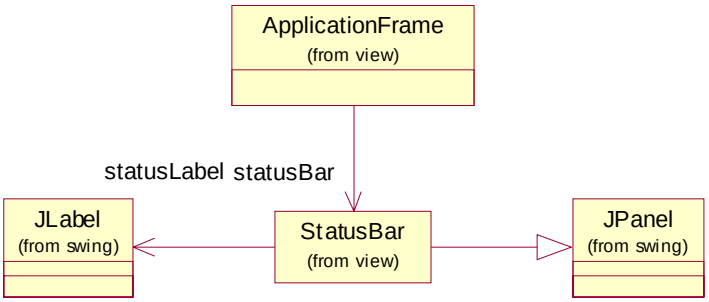


ToolBar

Die Toolbar (unter MenuBar) ist eine Symbolleiste vom Typ GeoToolBar (abgeleitet von JToolBar) und enthält mehrere Buttons. Diese werden per AbstractAction-Objekte hinzugefügt. Siehe Event-Managment. Die Veränderung zur Oberklasse, der JToolBar, besteht darin das die beiden Toolbar Einträge close-Action und parameter-Action nur dann aktiv sind, wenn auch wirklich ein Fenster (sceneFrame) geöffnet bzw. ausgewählt ist.

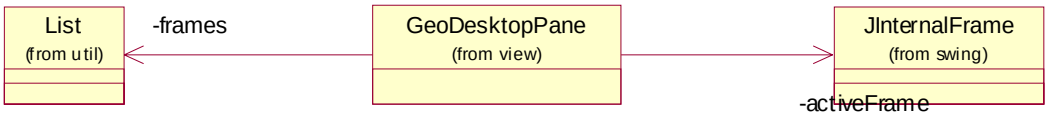
StatusBar

Die StatusBar (geometry.view.StatusBar) besteht aus einem JPanel welches mit einem JLabel versehen ist, das sich ganz links in der StatusBar (West, BorderLayout) befindet. Jeder Statusbartext muss „manuell" an- und abgeschaltet werden. Während des Ausführens einer Aktion wird für ein bestimmten Zeitraum ein erklärender Text angezeigt. Ansonsten wird ein selbstdefinierter Standarttext eingeblendet.



GeoDesktopPane

Die Klasse GeoDesktopPane ist abgeleitet von JDesktopPane und repräsentiert die Arbeitsfläche des Geoviewer. Über sie wird das Multi-Documents-Interface realisiert. Das Management der Child-Window (JInternalFrame) übernimmt die Klasse JDesktopPane. Das GeoDesktopPane dient also zur Verwaltung der Multi-Document-Umgebung. Alle angezeigten Fenster werden als Frames vom Typ JInternalFrame bei ihm registriert. Des weiteren verfügt GeoDesktopPane über eine Anzahl von fire[...]-Methoden die Nachrichten an die registrierten Listener interessierter Objekte schicken, falls Änderungen der Fensterzahl stattgefunden haben. Sie informieren sowohl über Zustand der Fenster, als auch welches Fenster aktiv ist. In dem Desktop sind wiederum ein LogFrame und optional mehrere SceneFrames enthalten. Beide sind von der Klasse JInternalFrame abgeleitet.

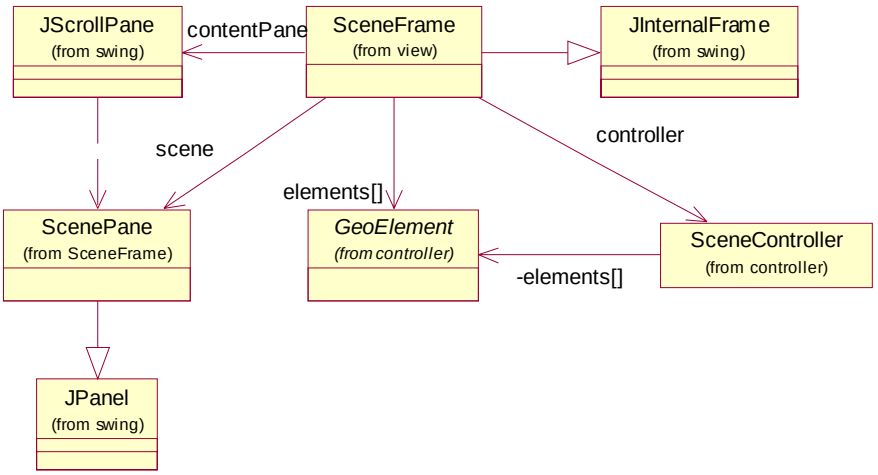


LogFrame

Das LogFrame, das alle anfallenden Fehler (Errors, Exceptions) in ein Textfeld, der textArea vom Typ JTextArea zeichnet. Die Überwachung der Umgebung wird durch einen Deamon-Thread realisiert. Die Klasse LogDeamon ist eine innere Klasse von LogFrame und abgeleitet von Thread. Der Thread wird nur in vordefinierten Zeitabständen aufgerufen. Maßgebend ist dafür die Konstante SLEEPTIME=100. Damit der LogDeamon automatisch beim Anzeigen des LogFrame startet, wird die Methode start() überschrieben und der Deamonprozess gestartet. Zum Scrollen im LogFrame wir eine ScrollBar zur Verfügung gestellt, die scrollPane vom Typ JScrollPane.

SceneFrame

Das SceneFrame stellt das Fenster, in das die Geoprojekte (.sd-Files) gezeichnet werden. Die SceneFrame besitzt eine Liste aller geometrischen Elemente (GeoElements) und besteht aus einem ScenePane (JPanel) und einem JScrollPane, ersteres zeichnet alle GeoElements des Sceneframes, in dem es die paintComponent() überschreibt, letzteres zeigt den sichtbaren Teil der ScenePane an, es stellt Scrollbalken zur Veränderung der Sicht zur Verfügung. Die ScenePane macht sich die Prioritäten der einzelnen geometrischen Objekte zu nutze um eine Abfolge für das Zeichnen zu erstellen. Die contentPane wird sozusagen von ihr überschrieben, da die ScenePane einen weit höheren Umfang (2000,2000) hat als die contentPane, wird sie in das JScrollPane übergeben, die nun immer nur einen Ausschnitt, welcher mit den Scrollbars verändert werden kann, ausgibt.
Wird ein SceneFrame erstellt so wird auch ein dazugehöriger SceneFrameController erstellt und mit ihm verbunden, wird das SceneFrame gelöscht, existiert auch der Controller nicht mehr. Für das SceneFrame hat der Controller nur die Bedeutung, dass er die GeoElemente aus den GeoProjekten liefert.

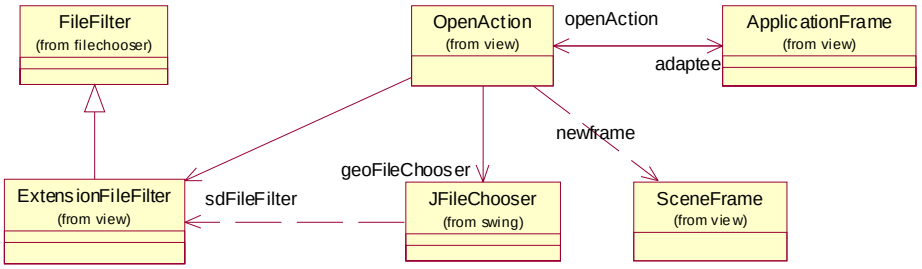


AbstractAction

Wie schon zu Beginn erwähnt, werden die Einträge des Menu und der ToolBar über AbstractAction-Objekte definiert, sogenannten Button-Events. Die Event-Handler sind spezialisierte Versionen der AbstractAction-Klasse, die registrierten Objekte werden per Konstruktor übergegen. Für Tooltips und Ähnliches wird der Hash von AbstractAction genutzt: Das Interface Action enthält einige statische Klassenvariablen, die vordefinierte Werte für die Hash-Tabelle bereithalten. Unter diesen Werten lassen sich dann Dinge wie Tooltips und Icons speichern. Jeder Menueintrag kann vollständig über ein AbstractAction-Objekt charakterisiert werden(Icon und Beschreibung).

Exit-, Open-, Close-, About-, ParameterAction

Ein Button-Event ist die ExitAction. Sie wird vom ApplikationFrame registriert und schließt dieses.
Die OpenAction ist ebenfalls ein Button-Event und wird vom Hauptfenster registriert um den File-Open-Prozess zu initiieren. Sie besitzt ein Icon das beim Anklicken ein Dialog zum Datei-Öffnen, den GeoFileChooser vom Typ JFileChooser öffnet. Es wird ein benutzerdefinierter Dateifilter (ExtensionFileFilter, *.sd) hinzugefügt. OpenAction erhält von GeoFileChooser ein File-Objekt und gibt dieses an ein neu erstelltes SceneFrame weiter, das seine Daten aus der ausgewählten Datei empfängt. Das neue Frame wird zum DesktopPane hinzugefügt und mit maximaler Größe angezeigt.

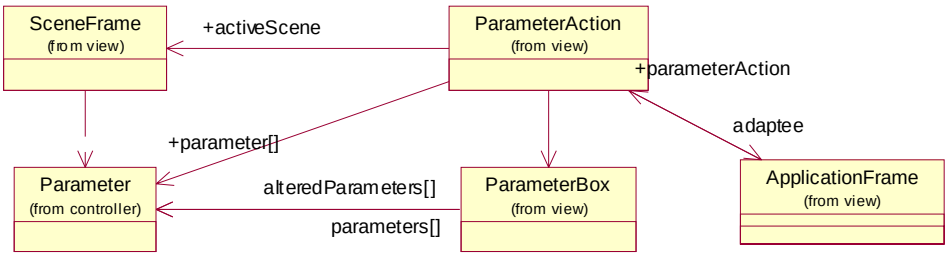


Das Button-Event CloseAction besitzt ebenfalls ein Icon. Wird diese Icon angeklickt holt sich CloseAction von GeoDesktopPane das aktive Fenster und führt dessen dispose()-methode aus. Klickt man auf das Icon von aboutAction in der Menübar, wird die Klasse AboutBox (abgeleitet von JDialog, implementiert ActionListener) aufgerufen und ein About-Dialog angezeigt. Dieser wird zentriert dargestellt.

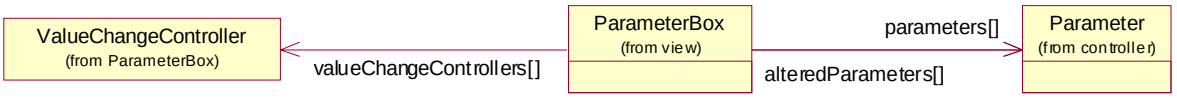
Das Button-Event ParameterAction öffnet ein Dialogfenster, die Parameterbox (wird später noch näher erklärt), mit Schiebereglern um die freien Parameter des aktuellen Fensters zu ändern. Vom Desktop wird dabei das aktuelle SceneFrame geliefert: activeScene vom Typ SceneFrame. Dieses übergibt über seinen Controller (controller(SceneController) eine Liste an freien Parametern: parameter(Parameter[]). Nun wird das Dialogfenster mit SceneFrame und Parameterliste initialisiert. Der Dialog gibt ein Liste mit den geänderten Parametern an die activeScene zurück die per update()-Methode aktualisiert werden. (Hier gibt es im Programm anscheinend einen Fehler: es werden immer die gelieferten Parameter ausgewertet, auch wenn sie per Reset-Taste zurückgestellt und der Dialog mit Cancel abgebrochen wurde. Abhilfe könnte hier ein Vergleich der Parameterobjekte schaffen.) Während des Updates wird eine Nachricht auf der StatusBar dargestellt und dann das Fenster neu gezeichnet.

ParameterBox

Wie schon angedeutet, stellt die Parameterbox ein Dialogfenster mit Auflistung aller vorhandenen unabhängigen Parameter dar.



Die Klasse Parameterbox ist abgeleitet von JDialog und implementiert ActionListener. Die Parameter werden per Constructor übergeben und in einem Array von Parametern gespeichert. Für jeden Parameter wird im Dialogfenster in einer Zeile der Bezeichner, ein Slider und ein Textfeld mit dem aktuellen Wert bereitgestellt. Die geänderten Werte werden in einem zweiten Array gleichen Typs abgespeichert um Änderungen per Button ‚Reset' rückgängig machen zu können. Jeden Parameter wird ein ValueChangeController zugeordnet welche in einem Array von ValueChangeController[] gespeichert werden. Die Klasse ValueChangeController implementiert ActionListener, FocusListener und ChangeListener und verwaltet das Zusammenspiel zwischen dem Textfeld (field(JTextField)) und der Sliderbar (slider(JSlider)) und liest die Beschriftung des Labels (label(JLabel)) aus den übergebenen Parameterobjekten ab. Wenn Slider verschoben wird (stateChanged-Event(ChangeListener)) muss Textfeld aktualisiert werden. Andersherum muss die Sliderposition angepasst werden, wenn der Inhalt des Textfeldes verändert wird. Das Textfeld verliert dann den Focus und ein actionEvent tritt auf, der die Sliderposition aktualisiert. Wenn Reset gedrückt wurde, werden die alten Parameterwerte wieder übergeben und das Textfeld, sowie der Slider aktualisiert.

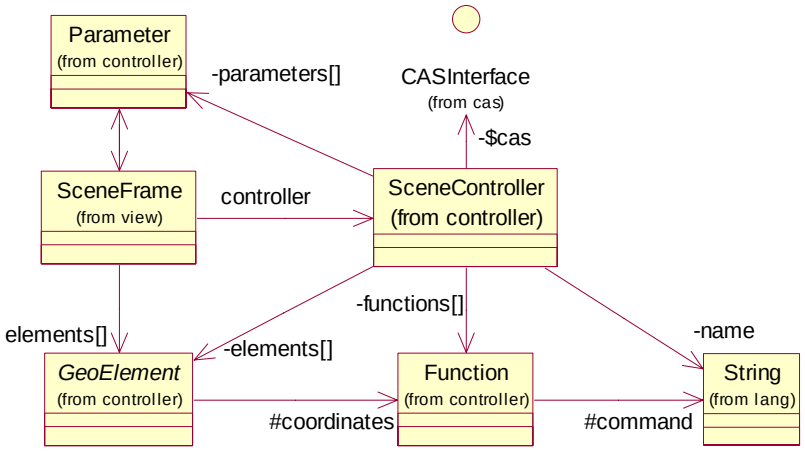


Controller

In den Bereich der Controller können die Klasse geometry.control.SceneController, das Interface geometry.cas.CASInterface und seine implementieren Klassen geometry.cas.MupadInterface und geometry.cas.MapleInterface eingeordnet werden, wobei natürlich für unsere Betrachtungen nur das MapleInterface von Bedeutung ist. Weiterhin können noch den Klassen des JLex-Packages und des java_cup-Packages Controllereigenschaften zugeschrieben werden, da sie in ihrer Funktion als Parser für die Interpretation des GEO-Syntax in den .sd-Files zuständig sind und damit dafür sorgen, dass die Daten von den persistenten Speicherungen an die verarbeitenden Komponenten weitergereicht werden.
Beginnen wir mit der Klasse SceneController, da diese eine zentrale Position in der Kommunikation zwischen Zeichenfenster (SceneFrame) und den Daten, gespeichert in den .sd-Files, spielt.

SceneController

Das SceneController Objekt sammelt und verwaltet Informationen über die aktuelle Scene. Dabei wird eine Referenz eines SceneControllers von jedem SceneFrame-Objekt gehalten. D.h. jedem SceneFrame-Objekt wird genau ein SceneController zugeordnet. Die Grundlage für den Informationsgehalt des SC bildet eine gewählte .sd-Datei. Um die benötigten Informationen aus der .sd-Datei zu erhalten, wird die statische Methode SceneController.load() verwendet. Sie nutzt den externen Parser um die Datei zu untersuchen und liefert ein fertig initialisiertes SceneController-Objekt.



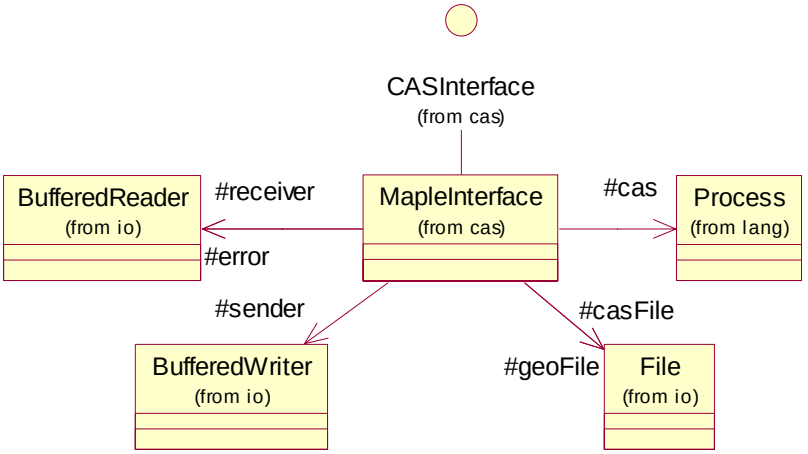
Der SC enthält eine Referenz auf die Schnittstelle zur entsprechenden CAS. In diesem Fall wird das Interface durch eine Schnittstelle zu Maple implementiert (MapleInterface). Über diese Schnittstelle werden zu berechnende Ausdrücke an Maple gesendet und die Resultate wieder empfangen und abgespeichert. Weiterhin speichert der SC eine Liste von Parametern(geometry.controller.Parameter[ ]), ein Liste von Funktionen (geometry.controller.Function[ ]) sowie eine Liste von geometrischen Elementen (geometry.controller.GeoElement[ ]). Die Liste der Parameter enthält alle freien Parameter, die zu Beginn mit Zufallswerten initialisiert und später über die Dialogbox geometry.view.ParameterBox eingestellt werden können. Die Initialisierung durch Zufallswerte wird in SC durch die Methode SceneController.initializeParameters() ausgeführt. Die Liste der Funktionen wird an das CAS übermittelt und legt den Grundrahmen für die Berechnungen der Scene. Die Liste der geometrischen Elemente wird vom SceneFrame übernommen und mit Hilfe von Zeichenroutinen (abgestimmt auf die Unterklassen von GeoElement) der Klasse ElementGraphics auf dem Bildschirm ausgegeben. Jegliche Änderungen der Parameter, ausgelöst durch z.B.: ParameterBox, wird von SC behandelt und es erfolgt eine Neuberechnung der abhängigen Funktionen und Elemente.

CASInterface

Das CASInterface ist ein abstraktes Interface für alle Schnittstellen zu CAS. An das CASInterface werden Ausdrücke übergeben, die von ihm berechnet werden sollen. Dabei wird ein externes CAS verwendet, an welches die Ausdrücke weitergeleitet werden und das die entsprechenden Resultate zurückgibt. Das CASInterface bietet Methodenrümpfe für grundlegende Operationen mit CAS an. Verbindungsoperationen: open() und close() und Operationen auf Ausdrücke: evaluate(), delete(), transmit(). Die Fehlerbhandlung erfolgt über IOExceptions und spezielle geometry.controller.CASExceptions. Eine besondere Implementation des CASInterface wäre die Klasse MapleInterface.

MapleInterface

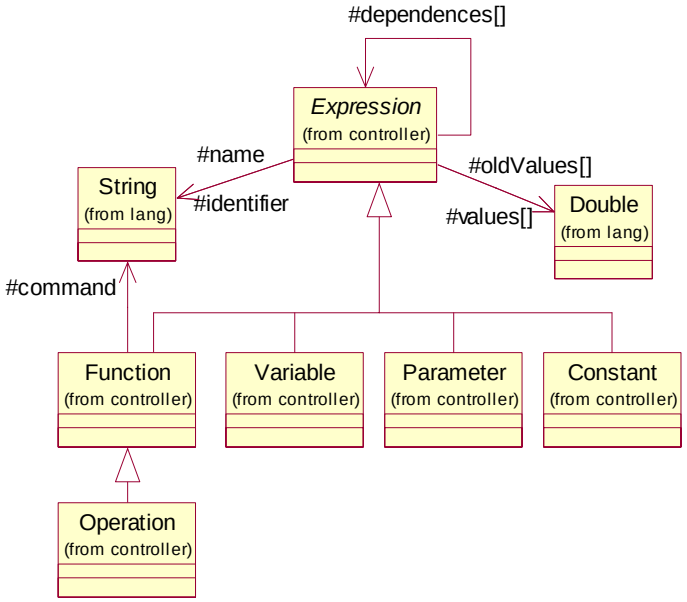
Das MapleInterface ist eine spezielle Implementation des CASInterface zugeschnitten auf die Kommunikation mit der CAS Maple ab Version 5.



Das MapleInterface erweitert das allgemeine CASInterface um einige wichtige Funktionen. Bei dem Verbindungsaufbau zu Maple wird die, in GeoViewer.rc definierte, Maple-Kommandozeilen-Datei in einem extra Prozess gestartet (Runtime.getRuntime().exec()). Dabei wird überprüft ob die Angaben über die ausführbare Maple-Datei und die Datei geoprover.mpl korrekt sind. Ist das der Fall wird zu dem Standarteingabekanal stdin des MapleProzess eine Verbindung per BufferedReader und zu den beiden Standartausgabekanälen stdout und stderr eine BufferdWriter-Verbindung aufgebaut. Diese Kanäle werden zur aktiven Übermittlung der Informationen und zum Abfangen kritischer Fehlermeldungen benötigt.
Zum Schluß wird die Initialisierungssequenz in Form eines String an Maple gesendet. Bei der Übermittlung von Formeln, Aufrufen und Werten wird darauf geachtet, dass alles in einem String an Maple versendet wird, der entsprechend des Maplesyntax formatiert wurde. Z.B.: werden Pfadangaben mit Hilfe von String-Tokenizern an die Maple-Konventionen angepasst. Der Maple-Output wird zusätzlich nach Fehlermeldungen geparst. Das MapleInterface implementiert Methoden zum Senden von Ausdrücken, zum Berechnen von ausdrücken und zum Entfernen von Ausdrücken aus der Berechnungsumgebung von Maple. Von Maple gelieferte Ergebnismengen können als Listen in Form einer einzelnen Zeichenkette oder als Arrays der Klasse Double interpretiert werden. Beim Schliessen der aktiven Maple-Verbindung werden zwei Wege eingeschlagen. Der sogenannte „soft way“ besteht daraus an Maple die Sequenz ‚quit’ zu schicken und 5sec zu warten, ob die Maple-Verbindung erfolgreich geschlossen wurde. Falls dies nicht der Falle ist, wird auf dem „hard way“ die Methode Prozess.destroy() verwendet. Damit wird garantiert, dass die Verbindung zur CAS auf jeden Fall abgebaut wird.

Model

Die Model-Ebene umfasst die Menge der .sd-Files, als persistente Speicherung der Geo-Algorithmen, die CAS inklusive der GeoProver-Bibliothek, zur Bereitstellung wichtiger Algorithmen zum Lösen algebraischer Probleme sowie die Implementationen geometrischer Elemente (geometry.controller.GeoElement) und Ausdrücke (geometry.controller.Expression). Auf die Auswertung von Geo-Syntax und die Erläuterung der Funktionsweise der CAS wird im Folgenden nicht weiter eingegangen, da diese Konzepte in der Projektspezifikation nicht berücksichtigt werden. Viel relevanter ist die Strukturierung der verschiedenen Ausdrücke.



Die Klasse Expression dient als abstrakte Oberklasse für alle verwendeten Ausdrücke. Von ihr sind die Unterklassen Funtcion, Variable, Parameter und Constant abgeleitet. Eine besondere Spezialisierung der Klasse Function ist die Klasse Operation.

Expression

Ein, durch Expression abstrahierter, Ausdruck besteht aus einer ID(String) zur Identifikation innerhalb der CAS, einem optionalen Namen(String), einem Feld von Werten(Double[ ]) und einer Liste(Expression[ ]) von weiteren Ausdrücken von denen der aktuelle Ausdruck abhängig ist. Jeder Ausdruck speichert seine aktuellen Werte sowie seine Werte vor der letzten Änderung. Ausdrücke werden in verschiedene Prioritätsklassen unterteilt, welche durch Klassenvariablen vom Typ int repräsentiert werden: SUPORTING, MAIN, CONCLUSION, MOVEABLE. Dabei wird ein Ausdruck mit dem Wert SUPORTING initialisiert. Mit der abstrakten Methode getInitializer() wird ein String zurückgegeben, der von Maple ausgewertet werden kann. Wie diese Zeichenkette genau aussieht hängt von der jeweiligen Unterklasse ab.

Function

Die Unterklasse geometry.controller.Function modelliert GEO-Funktionen und algebraische Basisfunktionen. Der Funktionskopf wird in einer separaten Zeichenkette gespeichert(command: String). Eine Funktion wird mit einer CAS-ID, einem Funktionskopf und einer Liste anhängiger Ausrücke() initialisiert. Die Liste der Ausdrücke repräsentiert die Menge der Argumente und damit die Stelligkeit der Funktion. Die Methode getInitializer() liefert eine Zeichenkette der Form: ‚command(argument1, argument2, …)

Operation

Die Klasse Operation spezialisiert die Menge der Funktionen auf unäre und binäre Operationen. Sie erweitert die Klasse Function um die Forderung nur ein oder zwei Funktionsargumente zu akzeptieren. Der Funktionskopf repräsentiert in diesem Fall das Operationszeichen.

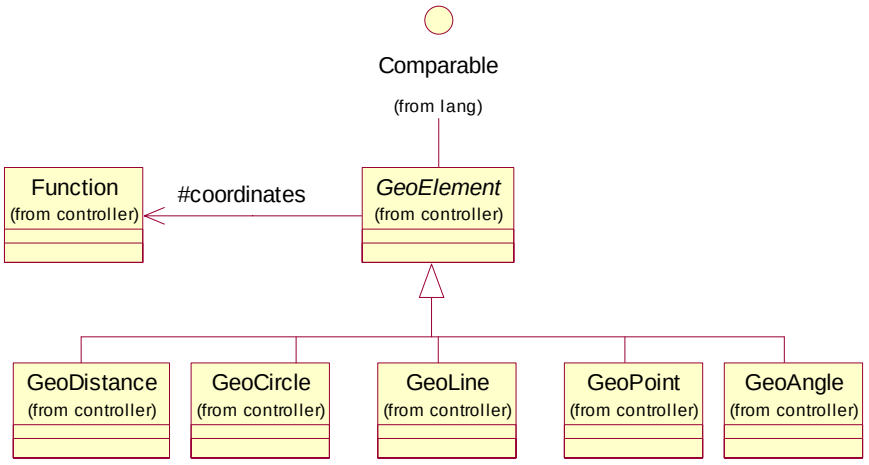
Parameter

Ein Parameter repräsentiert einen freien Parameter des Gleichungssystems, der später über die Parameterbox frei gewählt werden kann. Er erweitert die Klasse Expression durch die Tatsache das sein Name mit dem Namen des Parameter im .sd-File initialisiert wird.

Constant

Die Unterklasse Constant repräsentiert konstante Ausdrücke. Der Wert der Konstanten wird als String an den Constructor übergeben. Dieser Wert wird als Expression-Id sowie als value verwendet (Konvertierung in einen double-Wert). Die Werteliste ist in diesem Fall eine Liste mit nur einem Element.

Neben der Strukturierung von Funktionen, spielt auch die Modellierung von geometrischen Elementen eine wichtige Rolle. Als Ausgangspunkt steht hier die abstrakte Oberklasse GeoElement, von der die einzelnen Elemente GeoDistance, GeoCircle, GeoLine, GeoPoint und GeoAngle abgeleitet werden.



GeoElement

Ein GeoElement wird mit seinen Koordinaten initialisiert, welche in Form eines Function-Objekts übergeben werden. Die Koordinatenwerte eines GE können über ein Array der Wrapperklasse Double über die Methode getValues() ermittelt werden. Jedes GE wird einer Prioritätsklasse zugeordnet, die die Reihenfolge in der die Elemente gezeichnet werden, festlegt. GE implementiert das Interface Compareable und überschreibt die Methode compareTo, sodass es möglich ist zwei Elemente entsprechend ihrer Prioritätsklassen miteinander zu vergleichen. Da die abgeleiteten Unterklassen sich nur in ihrer Prioritätsklasse voneinander unterscheiden, wird im Folgenden nur diese aufgeführt:

*GeoCircle (*Prioritätsklasse=2), *GeoLine(*Prioritätsklasse=3), G*eoAngle(*Prioritätsklasse=4), *GeoDistance(*Prioritätsklasse=5), *GeoPoint(*Prioritätsklasse=6)

Auch hier gilt: Je höher die Pri.-klasse je höher der Rang beim Neuzeichnen.

Neben ihrer Funktion bei der Kapselung der Pri.-klasse, dienen die Unterklassen auch als Identifikator zur Auswahl der passenden draw()-Methode in der Klasse geometry.view.ElementGraphics.

4. Paket- und Klassenstruktur

Paketstruktur

geometry
Das Paket geometry besteht aus den 4 Paketen cas, controller, parser, view.

cas
Das Paket cas enthält alle Klassen bzw. Interfaces die zur Verbindung zwischen Programm (GeoViewer) und ComputerAlgebraSystem (CAS) notwendig sind, dazu gehören das CAS- und MapleInterface aber auch das MupadInterface und 2 Exceptionsklassen. Das Paket gehört logisch zum Controller.

controller
Das Paket controller enthält zum einen die Klasse SceneController, welche noch logisch zum Punkt Controller zu zählen ist, aber auch die Klassen die zum Model gehören. Dazu zählen Expression und alle davon abgeleiteten Klasse (z.B. Function, Parameter) und GeoElement und dessen Unterklassen (z.B. GeoPoint, GeoLine).

parser
Ds Paket parser enthält alle Klassen zum Übersetzen zwischen Programm- und CAS- Syntax. Dazu werden Parser (GeoParser), Scanner (GeoScanner) und Übersetzer (Translator) benötigt. Weiterhin gibt es eine Klasse zur Identifizierung der Symbole (GeoSymbols) und eine Exceptionklasse.

view
Das Paket view enthält alle im Unterpunkt View der Grundsätzlichen Designentscheidung besprochenen Klassen. Darunter z.B. das ApplicationFrame, die ParameterBox und GeoDesktopPane.

Klassenstruktur

Die Klassenstruktur kann der Datei Reversengineer.mdl entnommen werden.