# Designbeschreibung

GeoFuchs 1.0 – Stand 7.7.03

## *Inhaltsverzeichnis*

# 1 Allgemeines

## 1.1 Einleitung

Dieses Dokument begleitet Version 1.0 von unserem Programm **geofuxx**[1]. Es wurde wert darauf gelegt alle zur Zeit zur Verfügung stehenden Funktionen lückenlos zu beschreiben. Die dynamischen und statischen Aspekte des Modells wurden überarbeitet. Im Team wurde darüber diskutiert, inwiefern die Vorschläge, die in der letzten Inspektion gebracht wurden, umsetzbar sind. Näheres dazu im Text. Die Applikation kann als Applet unter folgender Adresse abgerufen werden.

http://pcai003.informatik.uni-leipzig.de/~geo08

## 1.2 Kurzcharakterisierung

GeoFuxx ist eine menügesteuerte graphische Java-Applikation, die es dem Benutzer ermöglicht, planare, geometrische Objekte wie Punkte, Geraden und Polygone zu zeichnen, zu bewegen und zu löschen. Es handelt sich hierbei hauptsächlich um eine „Dynamisierung“ der Visualisierung durch direkte Mausmanipulation. Weiterhin werden Funktionen wie das Bestimmen von Senkrechten, Parallelen, Mittelpunkten und Schnittpunkten zur Verfügung gestellt auf die über Buttons oder das Hauptmenu zugegriffen werden kann.

## 1.3 Systemvoraussetzungen

GeoFuxx ist als eigenständige Applikation konzipiert, die über eine grafische Nutzerschnittstelle mit gängigen Fenster-Techniken bedient wird. GeoFuxx benötigt als Voraussetzung zum Starten die Java JRE 1.4. Eine Mindestanforderung an die Performance des Systems ist nicht bekannt.

## 1.4 Entwicklungsumgebung

Das Klassendesign wurde in Rational Rose ausgearbeitet. Die Implementierung wurde mittels eines Texteditors durchgeführt (UltraEdit). Das Rational Rose Projekt dient als Basis für die Ausarbeitung des statisches Klassenmodells mit dessen Hilfe in Zukunft der Code von neu hinzugefügte Klassen generiert werden soll.

[1] Auch GeoFuxx (trademark)

# *2 Produktübersichten*

## *2.1 Modulbeschreibung*

Die Applikation setzt sich aus einer graphischen Komponente, dem GUI, und der Programmlogik, die die Darstellung von geometrischen Konfigurationen ermöglicht und Events behandelt, zusammen.
Das Programm wird durch den Kommandozeilen-Aufruf `java geofuchs.GeoFuchs` im Wurzelverzeichnis der Pakethierarchie (Verzeichnis geo08) gestartet. Nach erfolgreicher Initialisierung wird das Hauptfenster (*GMainFrame*) aufgebaut. Dies ist das visuelle Hauptmodul, das Applikation und Applet implementiert.

Das Hauptfenster ermöglich dem Benutzer Fenster zu erstellen die geometrische Konfigurationen in sich darstellen. Ein Menupunkt im Hauptmenu steuert dies. Das Modul zum Laden und Speichern von Konfigurationen wurde in dieser Version noch nicht berücksichtigt. Dafür ist es möglich zu einer schon bestehenden Konfiguration weitere Fenster zu öffnen. Alle Änderung an den geometrischen Sachverhalten (keine View Optionen) werden unverzögert in den anderen mit dargestellt. Verschiedene Konfigurationen können natürlich auch erstellt werden.

Das Modul, das die Kernfunktionalität realisiert macht es z.Z. möglich geometrische Objekte wie Punkte, Linien, Dreiecke, Kreise und Vierecke zu zeichnen. Einzelne Punkte können verschoben werden wobei evtl. damit verbundene Linien in die Bewegung miteinbezogen werden. Dazu mehr unter Punkt **3.1.2.**.

Des Weiteren wird die Berechnung von Mittelpunkten und Schnittpunkten unterstützt. Es können auch Senkrechten und Parallelen zu Linien generiert werden..

Das Skalieren und Verschieben des Ausschnittes des Koordinatensystems ist auch möglich. Zur Übersichtlichkeit wurde die Funktion Koordinatenachse ein/ausblenden bzw. Gitter ein/ausblenden realisiert sowie Punkte und Koordinateninformationen der geometrischen Objekte an/ausschalten mit eingebracht.

## 2.2 Abgrenzung

Es werden nur zweidimensionale Objekte und Koordinatensysteme benutzt. Außerdem ist es nicht möglich Funktionsgraphen oder komplexe mathematische Sachverhalte zu visualisieren.

# 3 Grundsätzliche Designentscheidung

3.1 MVC

Da es sich bei der Applikation um eine Geometriesoftware handelt, wurde darauf Wert gelegt eine angemessene Bildschirmpräsentation umzusetzen, die eine übersichtliche und kompakte Oberfläche zur Betrachtung von einzelnen geometrischen Konfigurationen unterstützt. Dies wurde mittels eines Multi Dokument Interface implementiert das mehrere Frames zur Darstellung verschiedener geometrischer Sachverhalte bereitstellt. Diese können vom Benutzer mittels der Maus manipuliert werden.

Zur Konstruktion des User-Interface wird das Model-View-Controller-Paradigma angewandt, das aus den drei Klassen Model, View und Controller besteht. 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. Das MVC-Konzept erlaubt somit eine klare Schichtentrennung zwischen den Anwendungsdaten, den Sichten auf die Anwendungsdaten und der Benutzungsschnittstelle.

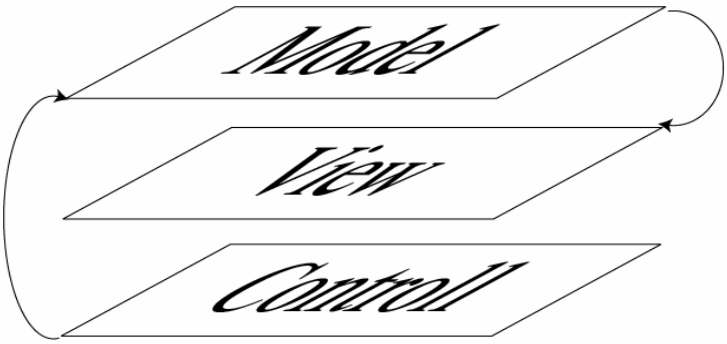



*Abb3.1 MVC Architektur mit Verknüpfungen*

Einen Überblick über die Realisierung dieses Paradigmas in unserem Projekt wird aus den Darstellungen der Packet- und Klassenstruktur unter Punkt 4 deutlich. In den nächsten Unterpunkten soll im Detail darauf eingegangen werden.

### 3.1.1 Model

Zum Modell der Applikation, also zur Kernfunktionalität, zählt die Bereitstellung von verschiedenen Geometrie Klassen und der geometrischen Konfiguration die in der vorliegenden Version Komponenten umfasst.

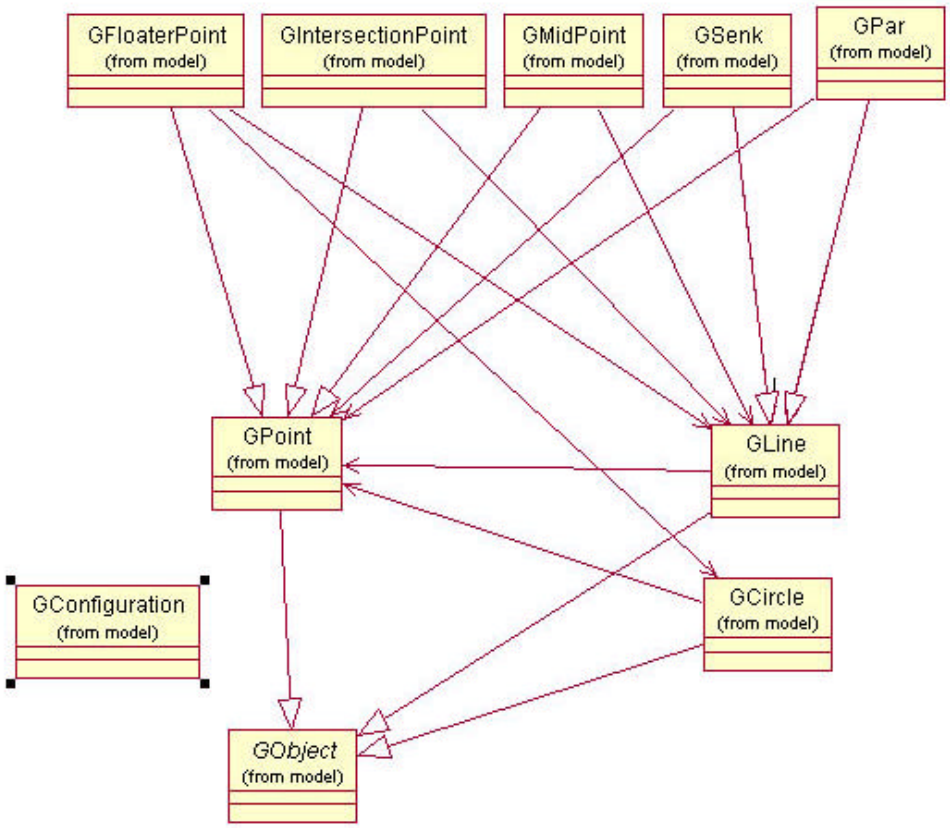



*Abb.3.2 Allgemeiner Überblick über das Modell*

Alle Klassen (ausgenommen GConfiguration) werden Programmintern als geometrische Klassen betrachtet und haben *GObject* als Basisklasse. Jedes Objekt hat eine eindeutige Identifikationsnummer (id). *GObject* stellt eine einheitliche Schnittstelle für die geometrischen Objekte dar. Außerdem definiert sie die abstrakte Klasse draw(), die von den jeweiligen Kinder-Klassen implementiert werden muss. In späteren Versionen könnte man abstrakte Funktionen zu *GObject* hinzufügen die zum Beispiel für Speichern/Laden oder Kopieren/Einfügen verantwortlich sind.

### Mehr Über Punkte und Linien:

- Jede Linie (GLine) ist genau mit zwei Punkte assoziieret
- Jeder Gleiter (GFloaterPoint) ist genau mit einer Linie assoziiert
- Jeder Schnittpunkt (GIntersectionPoint) ist genau mit zwei sich schneidende Geraden assoziiert
- Alle Punkte ausser GIntersectionPoint können mit der Maus bewegt werden

Die geometrischen Objekte[2] einer Konfiguration werden in *GConfiguration* in einer Liste gespeichert. Darüber hinaus stellt sie Attribute und Operationen zur Verwaltung der Objekte und deren Darstellung in einem Zeichenfenster bereit.

Eine typische Member Methode der Klasse GConfiguration ist z.B. addObject(GObject), das die Funktionalität eines der oben erwähnten geometrischen Objekte zu der Liste von GObjects hinzuzufügen, bereitstellt.

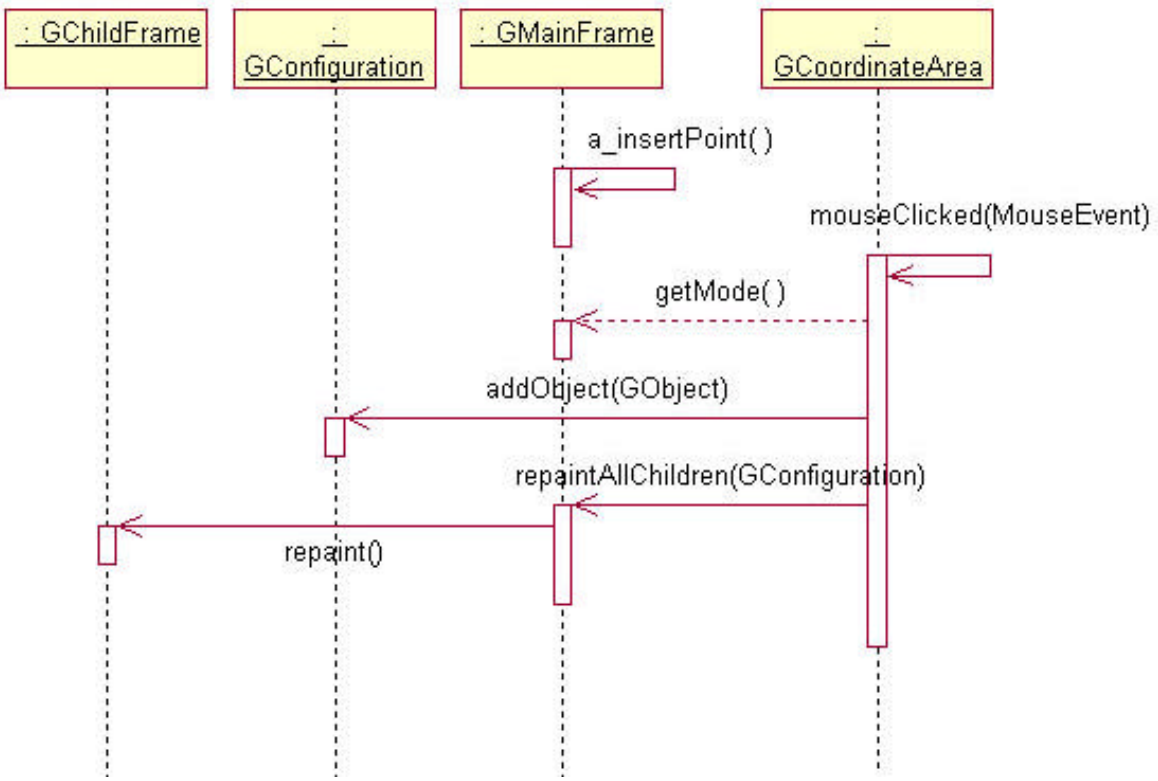



*Abb 3.3 Dynamisches Modell - Punkt hinzufügen*

[2] die geometrischen Objekte umfassen z.Z. Objekte vom Typ: Linie (GLine), Kreis (GCircle), Schnittpunkt (GIntersectionPoint), Gleiter (GFloaterPoint), Freier Punkt (GPoint), Parallele (GPar), Senkrechte (GSenk) und Mittelpunkt (GMidPoint)

Das Bewegen von Punkten wird durch Klicken und Ziehen der Maus realisiert. Wenn die Maus über einem Punkt ist, und die linke Maustaste gedrückt wird, so wird der Punkt zur Bewegung vorbereitet. Wenn die Maus bewegt wird und dann losgelassen, dann wird der Punkt an die Stelle bewegt, wo die Maustaste losgelassen wurde.

**In der aktuellen Version wurde die von uns zuvor zu Berechnungszwecken eingesetzte Klasse GCalculator entfernt und deren Funktionalität in die entsprechenden davon Nutzen tragenden geometrischen Klassen implementiert (GIntersectionPoint, GFloaterPoint).**

### 3.1.2 View

Initialisiert wird das graphische User Interface über die Startklasse *GeoFuxxs* (im Packet *geofuchs)*, die ein Objekt frame des Typs *GMainFrame* definiert. Dieser wiederum initialisiert über seinen Konstruktur alle graphischen Komponenten der Applikation und richtet die Liste der geometrischen Konfigurationen ein. Es kann mehrere Konfigurationen und Instanzen vom Typ *GChildFrame* haben, wobei jedes *GChildFrame* genau einer Konfiguration zugeordnet ist. Die folgende Abbildung soll eine Übersicht über das statische Modell der View geben.

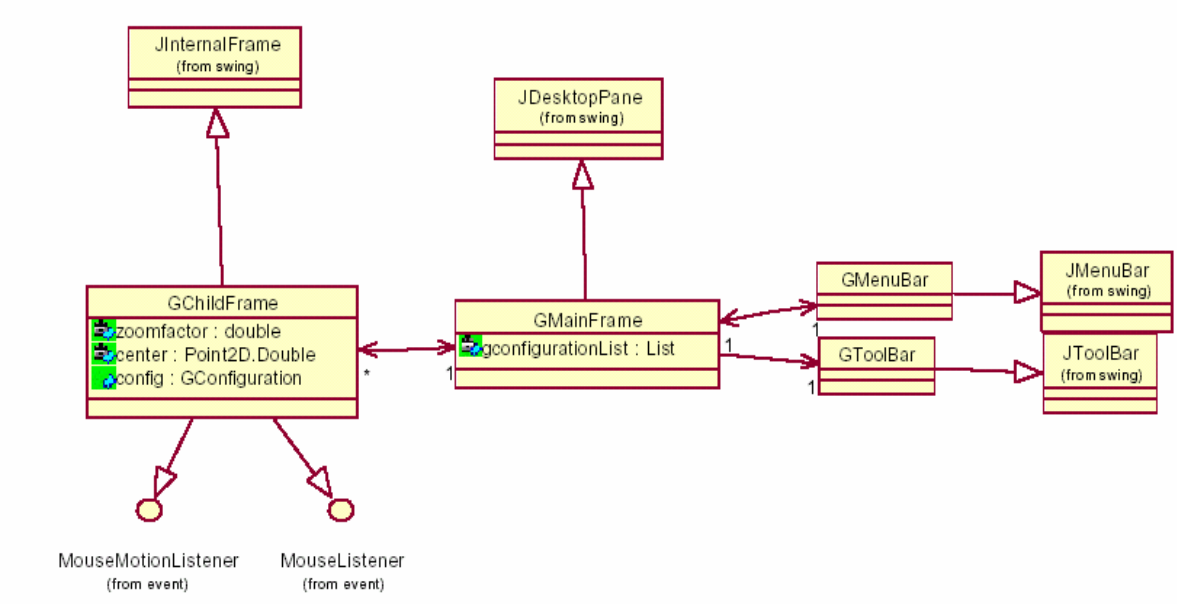



*Abb 3.4 Statisches Modell – View Komponenten*

Beim wiederholten Zeichnen (repaint()) auf eine Instanz vom Typ *GChildFrame* wird auf die Konfiguration zurückgegriffen, so dass alle Elemente im sichtbaren Bereich der aktuellen Ansicht gezeichnet werden. Dabei werden Veränderungen, die die Ansicht auf die Zeichenebene betreffen wie z.B. Zoomfaktor und Zentrum der Ansicht sowie Grid oder Koordinatenattribute im GChildFrame Objekt abgespeichert. Alle Änderungen an den geometrischen Sachverhalten werden in dem entsprechenden Konfigurationsobjekt festgehalten.

Der tatsächliche Zeichenmodus wurde nach langer Diskussion als global festgelegt. Orientiert haben wir uns dabei an weit verbreiteter graphischer Anwendungssoftware.

Wenn die Konfiguration geändert wird (siehe **Control**), müssen auch alle 'child frames', die mit der aktuellen Konfiguration assoziiert sind, neu gezeichnet werden. Das bedeutet, dass *GMainFrame* eine Liste aller Instanzen von *GChildFrame* und *GConfiguration* haben muss, um diese Funktionen realisieren zu können. Beim Skalieren wird lediglich das aktuelle *GChildFrame* aktualisiert. Panning (Ansicht bewegen/Zentrum der Ansicht bewegen) wird durch Scrollbars realisiert.

### *Wichtige Funktionen auf einem Blick*

**Punkt/Punkt verschieben**: Punkt zeichnen und Punkt verschieben (sobald Maus über einen Punkt wird in den Verschiebemodus geschaltet)

**Linie zeichnen**: Linie spannt sich zwischen 2 schon existierenden Punkten auf

**Kreis zeichnen:** Spannt sich zwischen Mittelpunkt und einem Kreispunkt auf

**Dreieck zeichnen**: Objekt bestehend aus 3 Punkten und 3 Linien wird bei Mausklick generiert.

**Viereck zeichnen**: Objekt bestehend aus 4 Punkten und 4 Linien wird bei Mausklick generiert.

**Gleiter:** Erstelle Gleiter auf Linie. Diese Funktion wird noch bearbeitet.

**Mittelpunkt**: Errechne und zeichne Mittelpunkt einer Linie durch Mausklick auf diese

**Schnittpunkt bestimmen**: Bestimme Schnittpunkt von 2 sich schneidenden Linien.

**Senkrechte zeichnen**: Erstellt eine Senkrechte zu einer Linie durch einen ausgewählten Punkt.

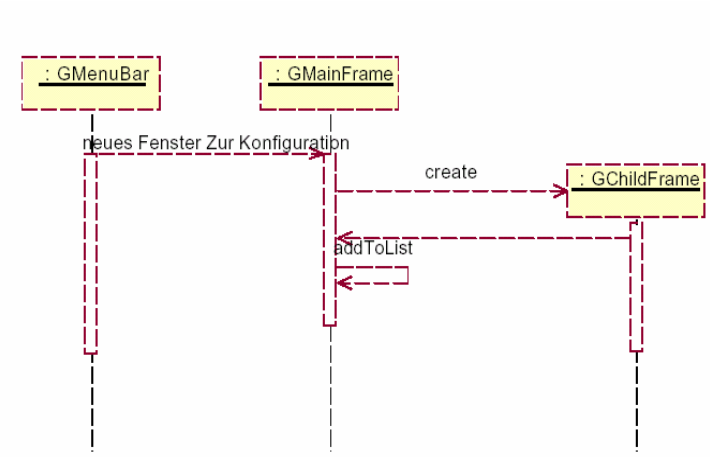
**Parallele zeichnen**: Erstellt eine Parallele zu einer Linie durch einen ausgewählten Punkt.

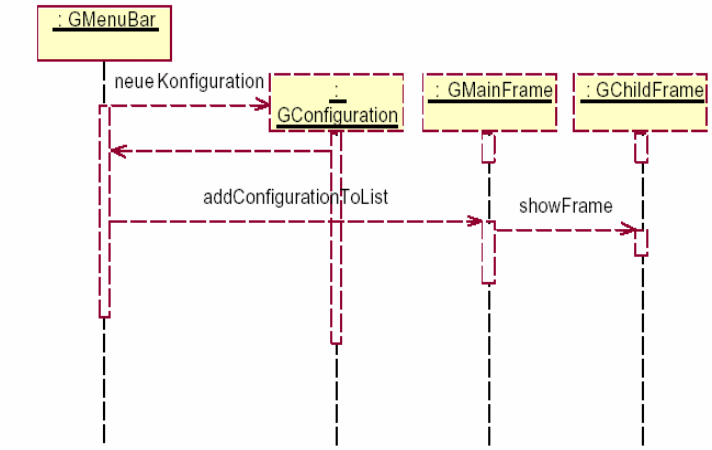
Zu jedem der oben genannten Punkte existiert eine von AbstractAction abgeleitete Klasse, die die Nachricht bei einem Buttonklick via der actionPerformed() Methode des Buttons entsprechend weiterverarbeitet.

## GChildFrame

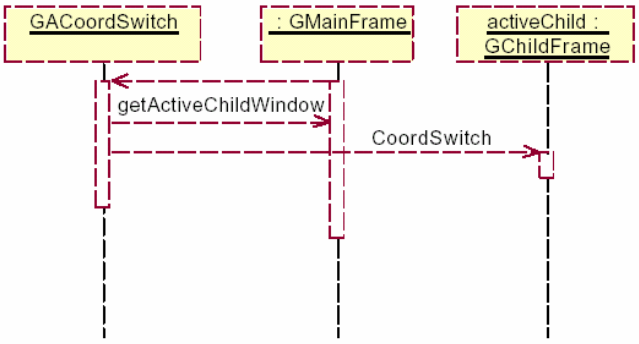
Besteht aus einer Zeichenfläche, Scrollbars und einem StatusBar. Die Aktuelle KonfigurationsID (z.B. Konfiguration1) wird der Text des Fensters sein. Der Skalierungsfaktor, Zentrum des Koordinatensystems und der aktuelle Status der Mausposition (Maus über Objekt) sind im StatusBar zu sehen. Darüber hinaus werden auch verschiedene Buttons zur Kontrolle der Sicht auf die Konfiguration bereitgestellt.

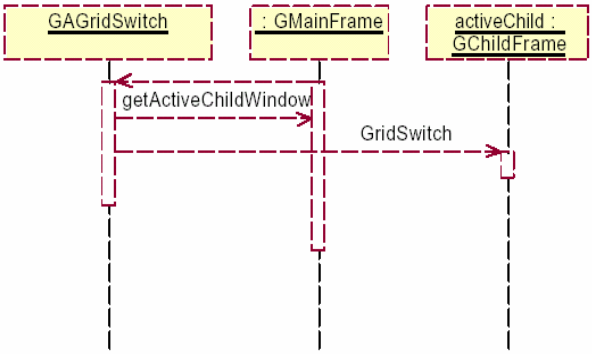
Einige elementare Operationen sollen in folgenden dynamischen Modellen gezeigt werden. Im folgendem wird eine neue Konfiguration erstellt (Rechts) und ein Fenster zu einer schon bestehenden Konfiguration hinzugefügt (Links).




Die 2 folgenden Diagramme zeigen auf welche Weise das Gitter in einer Ansicht An und Aus geschaltet wird bzw. das Koordinatensystem.



#### 3.1.2.1 Snapping

Snapping wird gebraucht um festzustellen über welchem Objekt der Konfiguration die Maus sich z.Z. befindet. Es erlaubt auch den Cursor der Maus durch Bewegen und Annähern an ein benachbartes Objekt in der Zeichenebene oder am Grid auszurichten. Unter benachbarte Objekte versteht man: Den Schnittpunkt zweier Objekte, Der Endpunkt einer Gerade, einen Punkt oder ganz einfach einen beliebigen Punkt auf einem Objekt.

Die Klasse *GCoordinateArea* implementiert zu diesem Zweck die Funktion getSnapObjekt(). Diese macht wiederum von der Funktion GObject.isSnapQualified() gebrauch, die als ‚abstract' in GObject definiert ist und von jeder geometrischen Klasse die davon ableitet überschrieben wird. Die Funktion gibt das zu untersuchende Objekt zurück falls der Abstand vom Cursor zu dem Objekt im Toleranzbereich des Snap-Radius liegt. Anderfalls ist die Rückgabe null und es findet kein ‚snapping' statt.

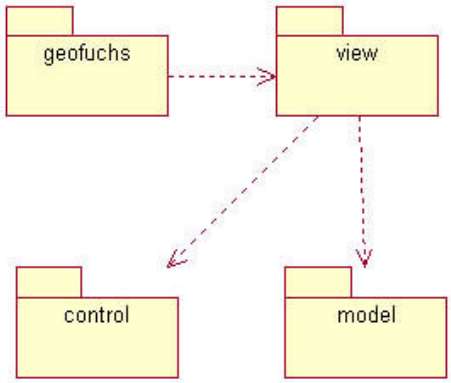
### 3.1.4 Control (Event handling)

In der Control-Schicht werden zwei Arten von Nachrichten abgefangen: Maus (Click, Drag, Release, Buttonpress) und Keyboard Events. Die erste Art von Nachricht ist für Einfügen und Auswählen von Elementen und Aktivierung von Menüpunkten und Toolbar-Buttons zuständig. Die zweite Art ist verantwortlich für die Skalierung des aktuellen Zeichenfensters und der Veränderung dessen Mittelpunktes.

Jede Aktion die durch einen Klick auf einen Punkt im Menu oder auf einen Button auf dem ToolBar ausgelöst wird wurde mit Hilfe eines ActionListener Objekts implementiert, das die actionPerformed() Funktion jeweils überschreibt. Die gesamte Abhandlung dieser Routinen findet in der Klasse GMainFrame statt. Dabei wird einfach zu jeder zu behandelnden Komponente die addActionListener() Funktion aufgerufen und ein entsprechendes Objekt ActionListener übergeben. Der Inhalt der actionPerformed() Funktion weist jeweils auf Funktionen die mit a_*() gekennzeichnet sind und in *GMainFrame* implementiert wurden. Es wurde auf die Benutzung von separaten Klassen die AbstractAction implementieren verzichtet um die Quantität der Java-Klassen relativ bezogen auf den Umfang des Projektes abzustimmen.

# 4 Paket und Klassenstruktur

Die Packetstruktur beruht auf dem MVC Konzept. Um die logische Abtrennung der 3 Schichten zu gewährleisten wurde Wert darauf gelegt die zu Model, View und Control gehörenden Klassen in gleichnamige Packete zu packen. Das folgende Diagramm soll dies veranschaulichen.




Die Klassenstruktur geht aus den schon gezeigten statischen Diagrammen aus Punkt 3 **grundsätzliche Designentscheidungen** hervor.