Studienarbeit
zur
Multiresolution Analysis

von
Julia Schneider
aus
Wuppertal

vorgelegt am
Lehrstuhl für Praktische Informatik IV
Prof. Dr. Effelsberg
Fakultät für Mathematik und Informatik
Universität Mannheim

Dezember 2001
Betreuerin: Dipl.-Math. oec. Claudia Schremmer

Inhaltsverzeichnis

1. Vorwort

Während der Multimedia Vorlesung im Wintersemster 2000/2001, gehalten von Professor Effelsberg, wurde ich zum ersten Mal auf das VIROR-Projekt aufmerksam. Via Internet wurde die Vorlesung in einer Live-Konferenz-Schaltung in Ton und Bild nach Freiburg und Karlsruhe übertragen. Sowohl die Vorlesungsfolien als auch der dazu gehaltene Vorlesungsbeitrag von Professor Effelsberg konnte im Internet heruntergeladen werden. Außerdem wurden dort interaktive Lernmodule in Form von Applets bereitgestellt. Der Student hatte also verschiedenste Möglichkeiten, den in der Vorlesung gehörten Stoff aufzunehmen und zu verarbeiten. Besonders begeisterten mich dabei die Potentiale, die die Nutzung dieser neuen Lernumgebung bereithält.

Neben meinem allgemeinen Interesse am VIROR-Projekt, gefielen mir die Vorlesungsinhalte zum größten Teil gut. Mein allgemeines technisches Interesse wurde zum ersten Mal auf die Grundlagen multimedialer Anwendungen gerichtet. Deshalb wählte ich das Thema Wavelets für meine Studienarbeit aus. Dabei war das Ziel zum einen das theoretische Konstrukt zu erfassen, verarbeiten und darzustellen. Zum anderen habe ich mir aber auch immer wieder überlegt, dass eine schematische, einfache Darstellung den didaktischen Implikationen des Teleteachings am ehesten gerecht wird. Als Erweiterung und Gegenpol zum klassischen Lernen in Frontaluntericht oder in der Gruppe vor Ort, wirft das Teleteaching völlig neue lerntechnische Fragestellungen auf. Als Wirtschaftsinformatikerin konnte ich natürlich diesen Fragestellungen nicht nachgehen. Daher habe ich zumindest eine schematische, aggregierte und somit übersichtliche Darstellung, die dem Lernenden hilfreich sein kann, anvisiert.

Mannheim, den 03.12.2001 Julia Schneider

2. Einleitung

Mit dem VIROR-Projekt (**VIR**tuelle Hochschule **O**ber**R**hein) haben sich vier oberrheinische Universitäten zum Ziel gesetzt, in verschiedenen Fächern ein multimediales und netzgestütztes Lehrprogramm aufzubauen. Angeboten werden Lernmodule (CBTs/WBTs), die zur selbstgesteuerten Nutzung entwickelt wurden. Außerdem bilden Teleteaching-Veranstaltungen eine zentrale Komponente der Konzeption. Sie werden über das Internet übertragen. Die Aufzeichnungen sind online abrufbar oder werden über CD-ROM vertrieben. So soll es dem Studierenden ermöglicht werden, unabhängig von Zeit und Raum zu lernen.

In diesem Kontext ist auch meine Entwicklung des Multiresolution-Analysis-Applets zu sehen. Es stellt ein interaktives Lehrmodul zum Thema Bildkompression der Vorlesung „Multimediatechnik“ dar und wird in die Modulsammlung der „Virtuellen Hochschule Oberrhein“ übernommen. Die Multiresolution Analysis konzentriert sich auf die Betrachtung eines Signals in immer gröberen, aufeinander aufbauenden Auflösungen.

Diese Studienarbeit besteht aus zwei thematisch voneinander getrennten Teilbereichen. Im ersten Teil steht die Illustration von Wavelets als Funktionen im Mittelpunkt. Die Wirkung des Dilatationsparameters *a* (Stauchung/Streckung) und des Translationsparameters *b* (Verschiebung) auf das Mutterwavelet wird dabei mit Hilfe von Schiebereglern dargestellt. Der zweite Teil der Studienarbeit zeigt die praktische Anwendung der Signaltransformation. Hier wird der Algorithmus für die Analyse eines digitalen Signals mittels Wavelet-Filtern durch eine Animation visualisiert. Low- und High-Pass-Filter einer Wavelet-Filterbank werden dabei über das Signal geschoben und an jeder Position mit diesem gefaltet. Falten bedeutet, dass die übereinanderliegenden Werte multipliziert werden und danach alles aufaddiert wird. Auf die gleiche Weise wird sukzessive mit den Ausgangssignalen verfahren. Der programmatische Teil dieser Studienarbeit wurde mit dem Ziel entwickelt, diese komplexen Sachverhalte einfach und übersichtlich darzustellen.

2.1 Aufgabenstellung

Die Studienarbeit zur Visualisierung der Multiskalenanalyse unterliegt folgender Aufgabenstellung.

Inhaltlich werden entwickelt:

1. Darstellung (d.h. Plot) einiger Wavelets, die in geschlossener Form vorliegen (z.B. Haar, Mexican Hat, Morlet, etc)

2. Verschiebung, Stauchung und Streckung des gewählten Mutterwavelets Ψ durch Parameter a und b:

$$\Psi_{a,b}(t) := \frac{1}{\sqrt{a}} \Psi \left(\frac{t-b}{a} \right)$$

3. Analyse eines eindimensionalen Signals mit der gewählten Waveletschar, entweder

   (a) als geschlossene Form ODER
   (b) als Filter.

Die äußeren Vorgaben sind:

- Java 1.3 mit Swing GUI,
- Englische Benutzerführung,
- „intuitive“ Oberfläche,
- ausführliche help Seiten,
- abschließende schriftliche Ausarbeitung von ca. 30 Seiten Umfang.

2.2 Entwicklungsumgebung

2.2.1 Software

- Übersetzung des Quelltextes durch Java™ 2 SDK (Software Development Kit) version 1.3.1 der Firma Sun
- Entwicklungumgebung: *KAWA 3.51a* der Firma Tek-Tools
- Erstellung der UML-Diagramme durch *Rational Rose 2000 Enterprise Edition* der Firma Rational
- Microsoft Word 2000

2.2.2 Hardware

- Betriebssystem: Microsoft Windows 95 und Microsoft Windows 98
- Prozessor: Intel Pentium mit 233MHz und Intel PentiumII mit 333MHz
- Arbeitsspeicher: 16 MB-RAM und 64 MB-RAM

2.3 Systemvoraussetzungen

2.3.1 Software

Da die Anwendung eine Swing-GUI besitzt und bei der Entwicklung auch andere Klassen des Pakets javax.swing benutzt wurden, ist das Java 1.2 Plugin unbedingte Vorraussetzung, um das Applet in einem Browser anzeigen zu können. Es kann kostenlos im Internet auf der Sun-Webseite http://java.sun.com/products/plugin/ heruntergeladen werden. Es ist jedoch zu empfehlen, sich direkt die neuste Version des Java plugins herunterzuladen.

Desweiteren benötigt man einen Internet-Browser, der fähig ist Java-Applets anzuzeigen. Wenn das Applet nicht lokal sondern über das Internet geladen wird, ist natürlich eine Internetverbindung unabdingbar.

Die Anwendung wurde zwar unter dem Betriebssystem Windows 95/98 entwickelt, da Java aber eine plattformunabhängige Programmiersprache ist, dürften auch auf anderen Betriebssystemen keine Probleme auftauchen. Darüberhinaus wurden die meisten betriebssystemabhängigen AWT-Oberflächenkomponenten durch die ab Java Version 1.2 gängigen Swing-Komponenten ersetzt. Dadurch sollte das Applet auf jeder Plattform dieselbe Erscheinungsform besitzen.

2.3.2 Hardware

Bei langsameren Prozessoren kann es passieren, dass die Animation nicht mehr flüssig dargestellt werden kann und Stockungen auftreten.

2.3.3 Orgware

Applets, die in einen Java-fähigen Browser geladen werden, ist es nicht erlaubt, Dateien auf dem lokalen Rechner zu lesen. Das Multiresolution-Analysis-Applet kann deswegen nicht korrekt gestartet werden, wenn die Anwendung lokal gespeichert ist. Die meisten Browser erlauben dem Applet jedoch Zugang zu Dateien die auf der Control Acces List des JDK

stehen. Durch Aufnahme des Dateinamens „Image.jpg“ in diese Liste, können evtl. diesbezüglich auftretende Probleme behoben werden. Das funktioniert folgendermaßen:

Zuerst öffnet man die *~/.hotjava/properties* Datei in einem Texteditor. Dann fügt man die Zeile

acl.read=/Pfad zum Ort der Anwendung/Image.jpg

dieser Datei hinzu. Nun speichert man die Änderung und schließt die Datei wieder.

3. Theoretischer Teil

3.1 Die Fourier Transformation

Eine wichtige wissenschaftliche Errungenschaft erlangte der Mathematiker Joseph Fourier (1768-1830) mit der nach ihm benannten Fourier-Reihe. Kernausage dieses Theorems ist, dass jedes periodische Signal (Funktion) als eine Summe von verschiedenen Sinus- und Cosinus-Schwingungen dargestellt werden kann. Dabei multipliziert man Sinus- und Cosinus-Kurven mit passenden Koeffizienten und verändert auf diese Weise deren Amplituden. Dann verschiebt man diese Funktionen so, dass sie sich zu der gewollten Funktion aufaddieren lassen. Die beiden Funktionen, also die Usprungsfunktion und die aus Sinus- und Cosinus-Schwingungen zusammengesetzte (transformierte) Funktion, verkörpern nun dasselbe Objekt.

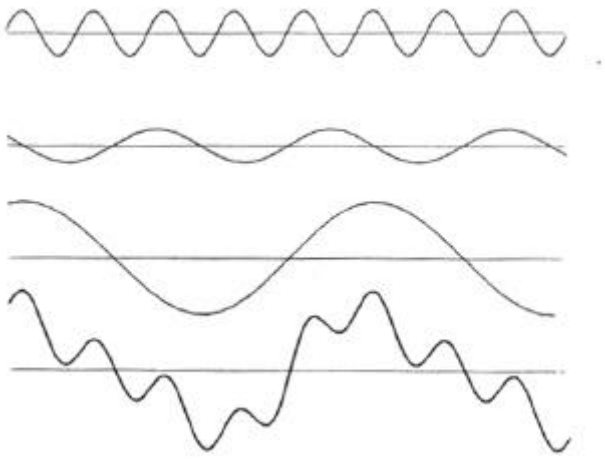

Abb. 1: Die letzte Funktion ist aus den drei darüberliegenden Funktionen zusammengesetzt.

Bei dieser Transformation gehen also keine Informationen verloren. Die beiden Gleichungen unterscheiden sich nur in der Art ihrer Basen. Die Ursprungsfunktion ist im Ortsraum dargestellt, d.h. ihr Verlauf ändert sich abhängig von der Zeit. Die transformierte Funktion hingegen lässt sich eindeutig durch die Koeffizienten der einzelnen Schwingungen darstellen. Deshalb spricht man auch von dem Frequenzraum. Aussagen über die Zeit und den Ort lassen sich in dieser Darstellung nicht mehr machen.

3.2 Schwächen der Fourier-Transformation

Durch die Fourier-Transformation konnten plötzlich numerische Gleichungen für Probleme gefunden werden, die zuvor unlösbar schienen, wie z.B. die Betrachtung der Wärmedurchdringung in einem Metallstab. Bei dieser Problemstellung untersucht man, wie sich die Temperatur in Abhängigkeit von der Entfernung (entlang des Metallstabs) und der Zeit in einem abkühlenden Metallstab verhält. Lange Zeit war es den Wissenschaftlern nicht gelungen, für dieses Problem eine Gleichung zu finden, die sowohl den Ort- als auch den Zeitaspekt berücksichtigte. Durch die Eigenschaft der Fouriertransformation, vom Ortsraum in den Frequenzraum zu wechseln, war es gelungen, dieses Problem als eine Funktion in Abhängigkeit von der Frequenz auszudrücken.

Obwohl diese Eigenschaft für viele Problemstellungen Lösungen bereithält, stellt sie für andere eher einen behindernden Faktor dar. Ein Fourier-transformiertes Signal kann nämlich keinerlei Auskunft über die Zeit geben. Man kann exakte Aussagen über den Anteil einer bestimmten Frequenz am Signal treffen, kann aber nicht sagen, wann diese Frequenz gesendet wurde. Zum Beispiel kann ein Fourier-transformiertes Musikstück genau angeben, welche Noten (=Freuqenzen) in ihm enthalten sind. Wann diese gespielt werden, bleibt jedoch unklar.

Eine weitere daraus resultierende Schwäche der Fourier-Transformation besteht darin, dass lokale Charakteristika globale Auswirkungen auf das gesamte transformierte Signal haben. Denn jede kleinste Unregelmäßigkeit bedeutet eine Überlagerung von vielen verschiedenen Frequenzen. Wenn also bei der Kodierung eines einstündigen Signals in den letzten fünf Minuten ein Fehler auftritt, erhält man eine Frequenzverteilung, die dem Originalsignal in keinster Hinsicht mehr ähnelt.

Diesen Verlust des Zeitaspekts versuchte Gabor mit der Gefensterten Fouriertransformation auszugleichen. Dabei teilt man das gesamte Signal in gleichgroße Zeitfenster ein. So erhält man eine gewisse Anzahl von Signalstücken, die nun einzeln der Fourier-Transformation unterzogen werden. Auf diese Weise lassen sich zumindest über gewisse Zeiträume Aussagen treffen. Problematisch ist bei dieser Methode jedoch die Wahl der Fenstergröße. Je kleiner man das Zeitfenster wählt, desto besser können plötzliche Veränderungen im Signalverlauf bei der Transformation aufgefangen werden.

Kleine Fenster aber haben zum Nachteil, dass große Schwingungen (also tiefe Frequenzen) nicht in sie hineinpassen. Sie werden somit in der Fourier-Darstellung nicht repräsentiert. Große Fenster hingegen verlieren den gewünschten Aspekt der Zeitlokalität.

3.3 Wavelets und ihre Darstellung

Ein Wavelet ist eine Funktion Ψ, welche die Zulässigkeitsbedingung

$$0 < C_\Psi := 2p \int_R \frac{\left|\hat{\Psi}(w)\right|^2}{|w|} dw < \infty$$

erfüllt. Daraus folgt:

$$0 = \hat{\Psi}(0) = \int \Psi(x) e^{-2pi0x} dx = \int \Psi(x) dx$$

Ein Wavelet ist also eine Funktion, die ein Integral von Null besitzt. Das Wavelet selbst ist nur auf einem begrenzten Intervall ungleich Null. Die eine Hälfte davon ist positiv, die andere negativ. Skaliert (Stauchung oder Streckung) man das Wavelet durch Multiplikation mit einer Konstanten, verändern sich die positiven und negativen Bereiche gleichermaßen und das Integral bleibt weiterhin Null.

Die von einem Mutterwavelet Ψ abgeleitete Waveletfamilie $\Psi_{a,b}(t)$ kann folgendermaßen beschrieben werden:

$$\Psi_{a,b}(t) := \frac{1}{\sqrt{a}} \Psi\left(\frac{t-b}{a}\right) \qquad \text{mit a>0 und a,b} \in \Re$$

Der Dilatationsparameter a staucht und streckt also das Wavelet. Der Translationsparameter b verschiebt es.

Der erste Teil des Multiresolution-Analysis Applets beschäftigt sich mit der grafischen Darstellung eines Wavelet. Als Beispiele stehen das Haar-, das Mexican-Hat- und das Morlet-Wavelet zur Verfügung. Anhand von Schiebereglern können die Parameter a und b verändert und das enstandene Baby-Wavelet betrachtet werden.

Bei der Implementierung wurden folgende Wavelet-Funktionen benutzt:

Haar Wavelet:

$$\psi(t) = \begin{cases} 1 & : \ 0 \le t < \frac{1}{2}, \\ -1 & : \ \frac{1}{2} \le t \le 1, \\ 0 & : \ \text{else.} \end{cases}$$

Mexican Hat Wavelet: $\Psi(t) = (1 - t^2)e^{-t^2/2}$

Morlet Wavelet: $\Psi(t) = \frac{1}{\sqrt{2p}} e^{-t^2/2} \cos(2p\nu_0 t)$ mit $\nu_0 = 0.8$

3.4 Die Wavelet Transformation

Die Wavelelet Transformation verfolgt denselben Ansatz wie die Fourier Transformation. Die Transformation besteht auch hier im Wechsel der Basis. Man überführt ein Ausgangssignal in eine andere Darstellungsform, die aus der Summe eines Mutter-Wavelets in verschiedenen Ausprägungen besteht. Diese verschiedenen Baby-Wavelets unterscheiden sich in Skalierung und Position. Ein gestauchtes Wavelet entspricht einer höheren Frequenz, ein gestrecktes einer tieferen Frequenz. Die Position ergibt sich durch das Verschieben des Wavelets über das Signal, bis es eine Stelle gefunden hat, an der es sich „wiederfindet“. Damit ist die Darstellung eines Wavelet-transformierten Signals nicht nur von der Frequenz, sondern auch von der Zeit abhängig!

Ein weiterer Unterschied zur Fourier Transformation besteht in der *Multiresolution Analysis*: Ein Signal wird sukkzesive in immer gröberen Auflösungen betrachtet. Gestauchte Wavelets werden benutzt, um kurze Komponenten mit hohen Frequenzen zu betrachten; Gestreckte für lange Komponenten mit tiefen Frequenzen. Die Größe des Betrachtungsfensters ist also nicht fest vorgegeben. Eine grobe Auflösung wird benutzt, um einen Gesamtüberblick über das Signal zu bekommen. Eine feinere Auflösung entdeckt die kleinen Details des Signals.

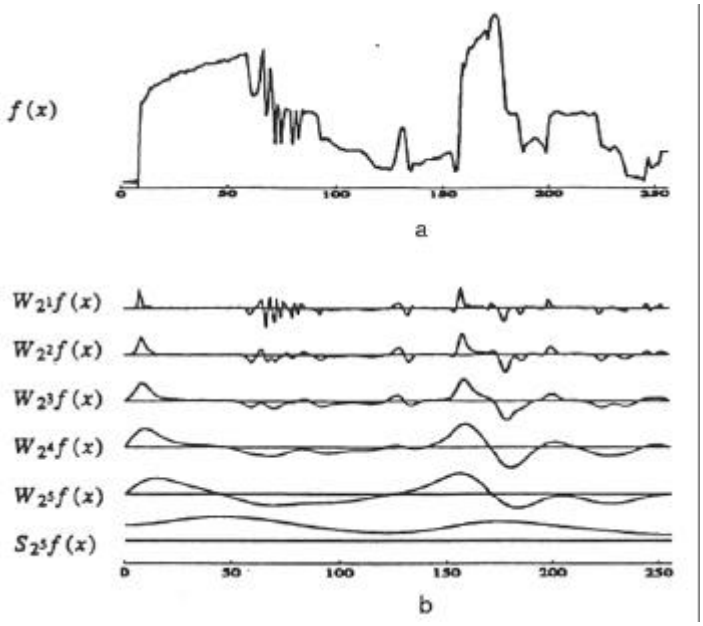

Abb. 2: Darstellung eines Signals in fünf verschiedenen Auflösungen.

Durch die besonderen Eigenschaften von Wavelets, ist die Wavelet Transformation besonders dann im Vorteil gegenüber der Fourier Transformation, wenn Signale analysiert werden sollen, die eine hohe Lokalität aufweisen. Denn ein Signal kann so durch sehr viel weniger Wavelet-Koeffizientent dargestellt werden, als es selbst an Werten besitzt.

3.5 Die Wavelet Transformation mit Filtern

Im Jahre 1986 machte Stéphane Mallat die Entdeckung, dass die Multiresolution Analysis von Wavelets nicht nur auf dem Forschungsgebieten der Mathematik beheimatet ist, sondern dass sie schon längst -unter anderem Namen- in anderen Disziplinen wie z.B. der Bildverarbeitung der Elektroingenieure Anwendung fand. Er erklärte in einer wissensschaftlichen Veröffentlichung, dass die Betrachtung eines Signals in verschiedenen Auflösungen durch Wavelets equivalent zu der sukzessiven Anwendung von Filtern auf ein Signal ist. Wenn man nämlich ein Signal mit langgestreckten Wavelets analysiert, filtert man die hohen Frequenzen heraus. Führt man die Analyse hingegen mit stark gestauchten Wavelets durch, filtert man tiefe Frequenzen heraus und übrig bleiben die hohen Frequenzen.

Auf Grundlage dieser Entdeckung entwickelten Mallat und Yves Meyer die schnelle Wavelet Transformation (*Fast Wavelet Transformation, FWT*). Um diese Transformation durchzuführen, benötigt man die Skalierungs-Funktion. Durch sie wird es ermöglicht, ein Abbild des zu analysierenden Signals in halber Auflösung zu bekommen. Bei der Transformation geht man nun folgendermaßen vor:

Zuerst wird ein Low-Pass Filter, der mit der Skalierungsfunktion verbunden ist, über das digitalisierte Signal geschoben. Dabei verändert sich die Position des Filters jeweils um den Faktor zwei. An jeder Position wird dieser mit den darunterliegenden Signalwerten gefaltet, d.h. multipliziert und dann aufaddiert. Die mathematische Formel für die Faltung ist

$$(a*b)_k = \sum_{j=0}^{k} a_j b_{k-j} \quad \text{mit } a_j, b_j > 0 \forall j$$

wobei a die Werte des Filters, b die Werte des Signals und k die Anzahl der Signalwerte darstellen. Der Low-Pass-Filter hat die Eigenschaft, hohe Frequenzen aus dem Signal herauszufiltern und nur die tiefen Frequenzen „passieren“ zu lassen. So erhält man also ein geglättetes Signal, das dem Ausgangssignal in einer gröberen Auflösung entspricht und nur halb so viele Werte wie dieses besitzt.

Nun wird auf dieselbe Weise ein High-Pass Filter über das Signal gelegt. Dieser Filter entspricht stark gestauchten Wavelets. Deswegen lässt er nur die hohen Frequenzen „passieren“ und filtert die tiefen aus. Deshalb besteht das Resultat dieser Filterung auch in den Detailwerten. Um das Ursprungssignal zu rekonstruieren, müssten diese Werte dem geglätteten Signal hinzugefügt werden.

Als nächstes wiederholt man das gesamte Verfahren auf dem geglätteten Signal. Dieses wird also wieder in zwei Teile geteilt, in den Low-Pass- und in den High-Pass-Teil. So fährt man rekursiv fort. Die Detailwerte jeder Transformationsstufe werden festgehalten.

Der große Vorteil dieser Methode ist, dass man während der Transformation keine Informationen verliert. Um das Ursprungssignal wieder herzustellen, führt man spezielle synthese-spezifische Low- und Higpass Filter über das geglättete Signal und über die Detailwerte der Transformation. Diesen Vorgang nennt man Synthese.

Bei der FWT (Fast Wavelet Transformation) erhält man nach jeder Transformationsstufe exakt genau so viele Koeffiezienten, wie das Ausgangssignal Werte besitzt. Einen Kompressionseffekt erzielt man, wenn man gewisse Detailwerte vernachlässigt. Das hat zur Folge, dass sich das Ursprungssignal nicht mehr vollständig rekonstruieren lässt. Die Rekonstruktion entspricht dann nur noch einer geglätteten Version dieses Signals.

Werden bei der Transformation Filter benutzt, die mehr als zwei Werte besitzen, treten Randwertprobleme auf. Dieses Problem kann auf verschiedene Weise gelöst werden. Eine mögliche Lösung betsteht in der periodischen Fortsetzung des Signals. Weitere Möglichkeiten sind das Spiegeln, das Fortsetzen oder das Auffüllen der Randwerte.

Der zweite Teil des Multiresolution-Applets versucht, diese Wavelet Transformation mit Filtern durch eine Animation zu visualieren. Dabei hat der Benutzer die Möglichkeit, zwischen der Transformation mit einer Haar-Filterbank und der mit einer Daubechies2-Filterbank zu wählen. Bei letzterer wurde das Randwertproblem durch die periodische Fortsetzung des Signals gelöst. Außerdem kann sowohl die Analyse als auch die Synthese eines digitalen Signals dargestellt werden.

4. Klassenstruktur

Java ist eine objektorientierte Programmiersprache. Deshalb besteht eine Java-Anwendung nicht aus einer einzigen compilierten Quellcode-Datei, sondern aus einer Vielzahl von Klassen und Interfaces, die in einer bestimmten Beziehung zueinander stehen und Nachrichten über Methoden austauschen können. Mögliche Beziehungen sind z.B. Generalisierung-Spezialisierungs-Beziehungen, Assoziationen und Aggregationen.

Die rein textuelle Dokumentation solch komplexer Klassenstrukturen ist für außenstehende Betrachter nur schwer verständlich und die Kommunikation daher stark eingeschränkt. Die Unified Modelling Language (UML) ist eine ausdruckstarke Modellierungssprache, die verschiedene Diagrammarten zur Visualisierung von objektorientierten Problemen und Problemlösungen anbietet. Als Ergänzung zur traditionellen Dokumentation ist sie ein mächtiges Werkzeug, welches das Verständnis von komplexen Zusammenhängen erleichtert (ein Bild sagt mehr als tausend Worte). Die in den nächsten Kapiteln folgende Beschreibung der Klassenstruktur ist deshalb um UML-Klassendiagramme erweitert.

Um die Übersichtlichkeit dieser Diagramme auch im Din-A4-Format zu garantieren, sind einige Klassen zu Packages zusammengefasst, die der eigentlichen Klassenstruktur nicht vollständig entsprechen. Im einzelnen wird darauf in den betroffenen Kapiteln hingewiesen.

Außerdem ist zu beachten, dass die getXY() und setXY() Methoden in UML-Diagrammen i.a. nicht angeführt werden. Da sie außer dem Setzen und dem Wiedergeben eines Attributwertes keine weiteren Funktionalitäten aufweisen, wird es nicht als sinnvoll erachtet, sie im einzelnen anzuführen. Man geht schlicht davon aus, dass sie existieren. Im weiteren habe ich deshalb sowohl in den Diagrammen als auch in den Beschreibungen der einzelnen Klassen darauf verzichtet, diese Methoden zu erwähnen.

4.1 Das Modellmanagement

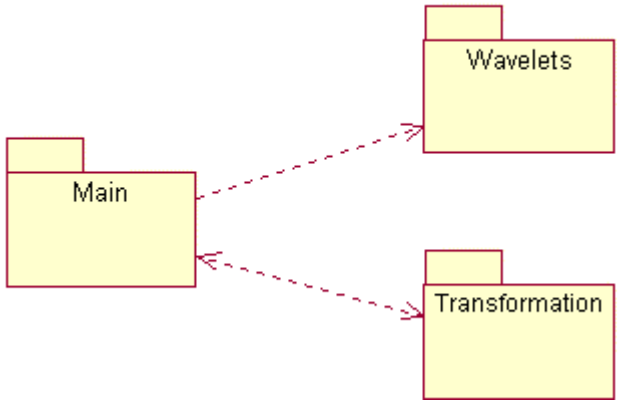

Abb. 3: Top-Level Ansicht

Wie aus der Einleitung (siehe Kapitel 2) hervorgeht, besteht diese Studienarbeit aus zwei thematisch voneinander getrennten Aufgabenstellungen. Die eine hat die Darstellung einiger Wavelets, die in geschlossener Form vorliegen, und deren Manipulation durch die Parameter a und b zum Inhalt. Die andere beschäftigt sich mit der Analyse eines eindimensionalen Signals mit Filtern.

Das Package *Wavelets* enthält alle unmittelbar mit der ersten Aufgabenstellung verbundenen Klassen. Das Package *Transformation* die Klassen der zweiten Aufgabenstellung. *Main* umfasst jene Klassen, die sich weder zum Package *Wavelets* noch zum Package *Transformation* eindeutig zuordnen lassen.

Main gehört zu den Packages, die nur der Übersichtlichkeit halber hinzugefügt worden sind. Auf realer Klassenebene existiert es nicht. Die in ihm enthaltenen Klassen sind hierarchisch auf derselben Ebene wie die beiden Packages *Wavelets* und *Transformation* anzusiedeln. Wie im nächsten Kapitel zu sehen ist, würde die Aufnahme jener Klassen in diesem Diagramm den Rahmen sprengen und in keinster Weise zu einem besseren Verständnis beitragen.

4.2 Das Package Main

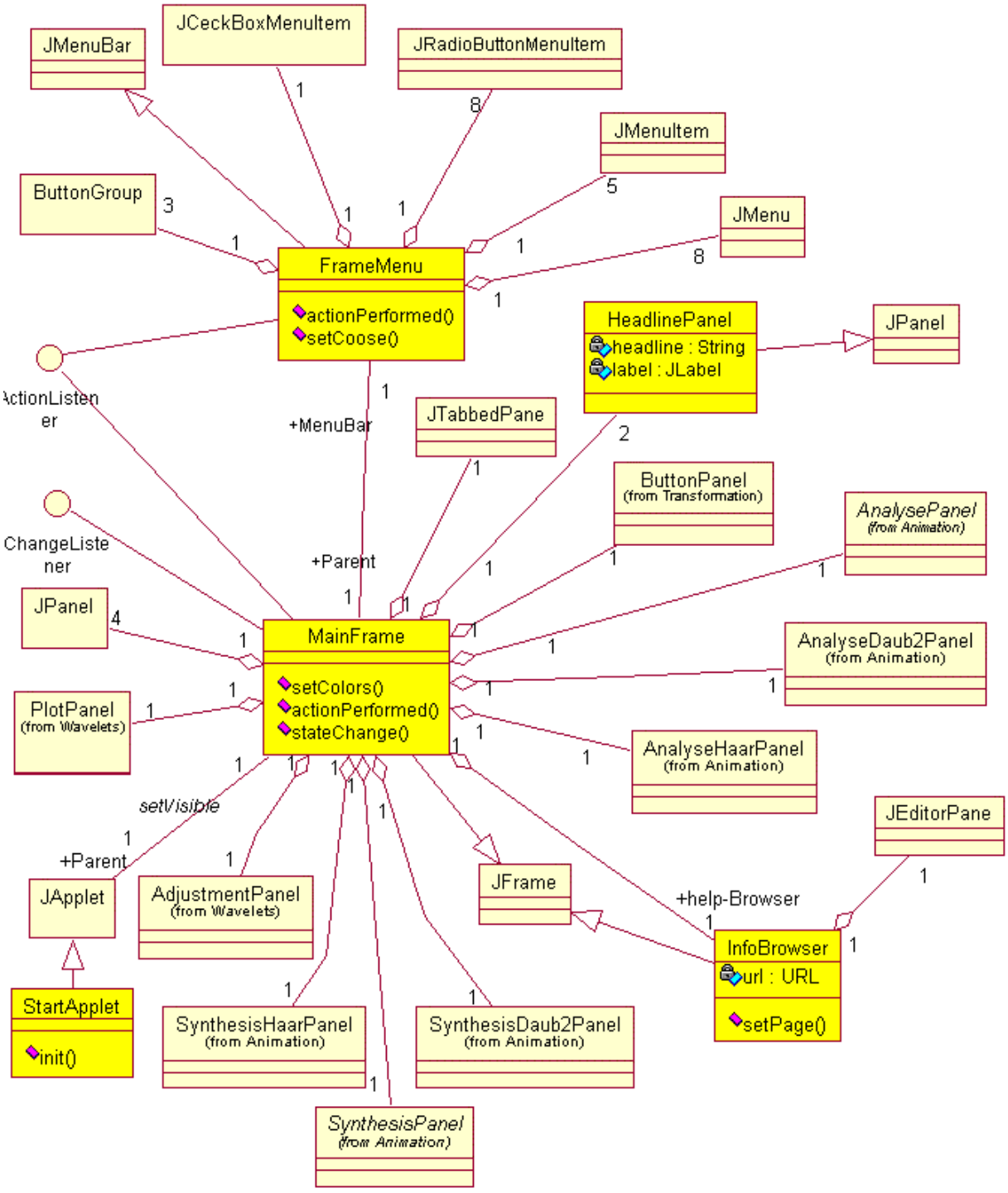

Abb. 4: Die Klassenstruktur des Package Main.

Den Ausgangspunkt, von dem die gesamte Anwendung gestartet wird, bildet das *StartApplet* , eine Subklasse des *javax.swing.JApplets*. Seine einzige Funktionalität besteht in einem

javax.swing.JButton, der, wenn er gedrückt wird, eine Instanz der Klasse *MainFrame* erzeugt und diese anschließend sichtbar macht.

Der *InfoBrowser* -eine Subklasse des *javax.swing.JFrame*- stellt einen kleinen Browser mit minimaler Funktionalität dar. Mit ihm können nur bestimmte HTML-Seiten angezeigt werden. Die Navigation muss durch die HTML-Seiten gewährleistet sein. Die Seiten werden auf einem *javax.swing.JEditorPane* dargestellt. Über die Methode *void setPage(URL path, String page)* wird die Seite *page* unter dem Pfad *path* im InfoBrowser angezeigt.

Das *FrameMenu* stellt die zu *MainFrame* gehörige Menueleiste dar. Sie ist von der Klasse *javax.swing.JMenubar* abgeleitet. Inhalt des *FrameMenus* sind Objekte der Klassen *JMenu, JMenuItem, JRadioButton* und *JCheckBoxMenuItem* (alle aus *javax.swing*). Dem Benutzer werden einige Aktionen zur Verfügung gestellt. So können verschiedene Einstellungen wie z.B. die Animationsgeschwindigkeit oder der Farbwechsel für rot/grün-blinde Benutzer vorgenommen werden. Außerdem eröffnet dieses Menue eine weitere Möglichkeit, eine der beiden Karten des *javax.swing.JTabbedPane* des *MainFrames* zu selektieren. Auch die help-Seiten der Anwendung werden über dieses Menue mittels eines *InfoBrowsers* aufgerufen. Die wichtigste Funktionalität besteht aber in der Auswahl eines der verschiedenen Wavelets in der Waveletansicht bzw. einer Filterbank in der Transformationsansicht.
Das *FrameMenu* implementiert zwar den *javax.swing.ActionListener*, die meisten *javax.swing.event.ActionEvents* werden jedoch an den Parentframe *MainFrame* weitergeleitet.

Das *HeadlinePanel* ist von *javax.swing.JPanel* abgeleitet und dient allein der Darstellung einer Überschrift –dem Attribut *headline*. Es wird in der Waveletansicht dazu benutzt, dem Benutzer zu verdeutlichen, welches Wavelet gerade dargestellt wird, bzw. welche Filterbank in der Transformationsansicht benutzt wird.

Die Klasse *MainFrame*, abgeleitet von *javax.swing.JFrame*, stellt schließlich das Herzstück der Anwendung dar. Zur Gestaltung des Layouts wird der Container *javax.swing.JTabbedPane* benutzt, der seine Komponenten grafisch suggestiv als Karteiregister aufbaut. In der Waveletansicht wird ein *AdjustmentPanel* und ein *PlotPanel* des *Wavelets*-Packages gezeigt (siehe Kapitel 4.3).
In der Transformationsansicht wird ein *ButtonPanel* und je nach gewählter Transformationsart ein *AnalysePanel* oder ein *SynthesisPanel* des *Transformation*-Packages

(siehe Kapitel 4.4) gezeigt. Ob das *AnalysePanel* auf das *AnalyseHaarPanel* oder auf das *AnalyseDaub2Panel* bzw. das *SynthesisPanel* auf das *SynthesisHaarPanel* oder auf das *SynthesisDaub2Panel* desgleichen Packages verweist, hängt von dem im FrameMenu gewählten Wavelet bzw. der dort gewählten Filterbank ab.

Von MainFrame werden die beiden Listener *java.awt.event.ActionListener* und *javax.swing.event.ChangeListener* implementiert. Von ihnen werden auftretende *java.awt.event.ActionEvents* bzw. *javax.swing.event.ChangeEvents* verarbeitet.

4.3 Das Package Wavelets

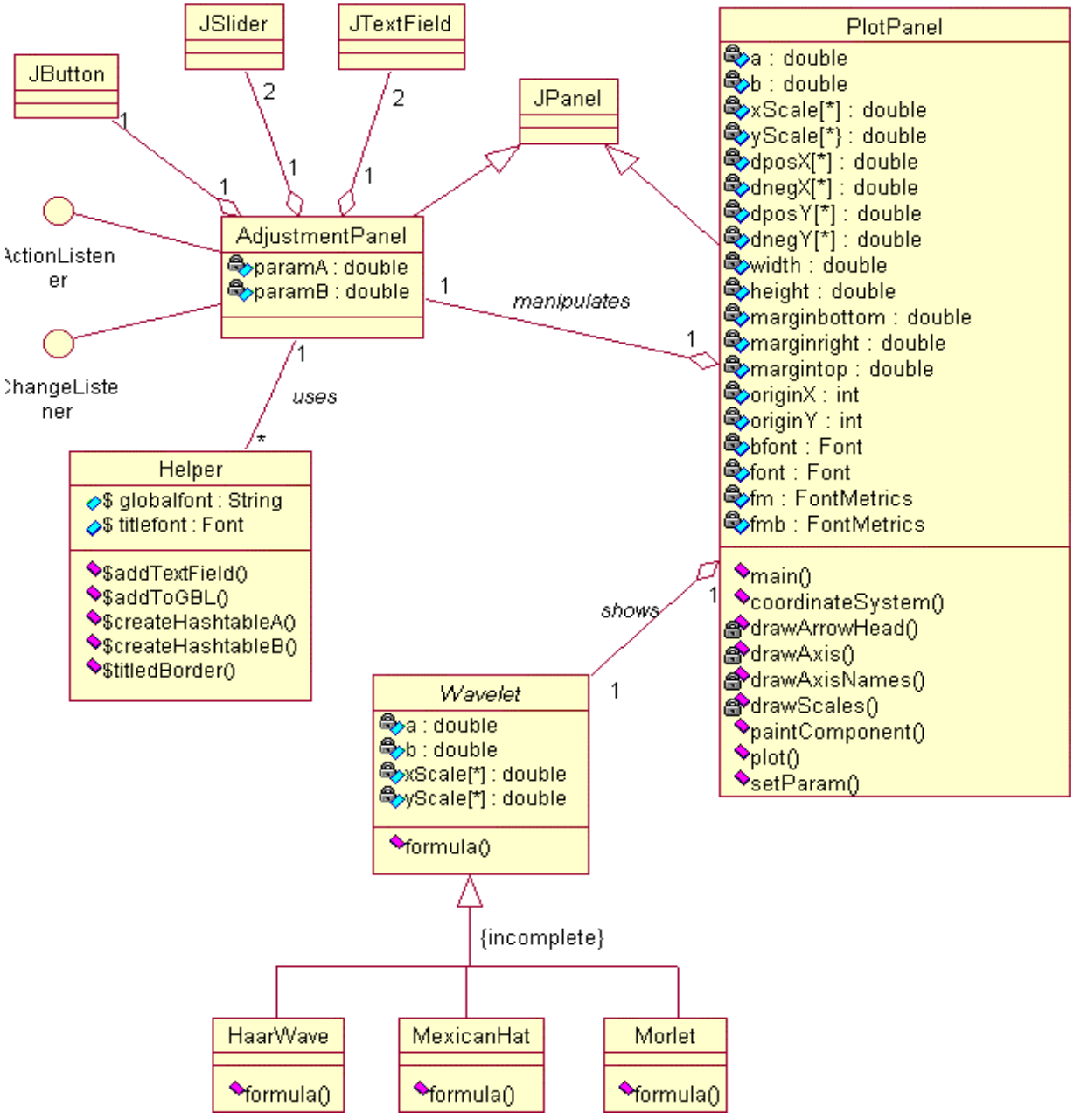

Abb. 5: Die Klassenstruktur des Package Wavelets.

Das Package *Wavelets* umfasst alle Klassen, die für die Darstellung eines Wavelets und für dessen Verschiebung, Stauchung und Streckung notwendig sind.

Alle für die Darstellung eines Wavelets wichtigen Daten sind in der abstrakten Klasse *Wavelet* enthalten: der Dilatationsparameter *a*, der Translationsparameter *b* und jeweils ein Array für die Werte der X-Achse (*xScale*) und für die der Y-Achse (*yScale*). Die Methode *double formula(double t)* berechnet wie die Waveletfunktion $\Psi_{a,b}(t)$ den Funktionswert des Wavelets an der Stelle t. Diese Methode ist in der Klasse *Wavelet* abstrakt, d.h. sie ist nicht implementiert. Alle abgeleiteten Klassen wie *HaarWave* (Haar Wavelet), *MexicanHat* (Mexican Hat Wavelet) und *Morlet* (Morlet Wavelet) überschreiben diese Methode gemäß ihrer Waveletfunktion. Sollen noch weitere Wavelets hinzugefügt werden, so müssen sie von der Klasse *Wavelet* abgeleitet sein, die Methode *double formula(double t)* überschreiben und den beiden Achsen (*xScale und yScale*) des Graphen die gewünschten Werte zuweisen.

Die grafische Darstellung des jeweils aktuellen Wavelets in einem Koordinatensystem erfolgt durch die Klasse *PlotPanel*, die von der Klasse *javax.swing.JPanel* abgeleitet ist. Jedes *PlotPanel* besitzt genau ein *Wavelet*, das es darstellen soll. Auch in ihm existieren die beiden Parameter *a* und *b*, die der aktuellen Ausprägungen der gleichnamigen Paramter des darzustellenden *Wavelet*s entsprechen. Die arrays *xScale, yScale, dposX, dnegX, dposY* und *dnegY* halten auf verschiedene Weise die benötigten Achsenwerte. Die übrigen Attribute werden für das Layout sowie für die Berechnung der richtigen Position des Achsenkreuzes verwendet. Bei Aufruf der Methode *paintComponent(Graphics g)* wird ein beschriftetes Koordinatensystem sowie das darzustellende *Wavelet* gezeichnet. Die Achsen sind nach den Achsenwerten des *Wavelet*s skaliert. Jedesmal wenn einer der beiden Parameter durch die Methode *void setParameter(double a, double b)* geändert wird, zeichnet sich das Panel durch den Aufruf der Methode *repaint()* neu.

Für die Manipulation -also die Stauchung, Streckung und Verschiebung- des Wavelets durch den Benutzer ist das *AdjustmentPanel* zuständig, das auch von der Klasse *javax.swing.JPanel* abgeleitet ist. Jedes *AdjustmentPanel* besitzt genau ein *PlotPanel*, das es über die Parameter *paramA* (Dilatationsparameter) und *paramB* (Translationsparameter) manipulieren kann. Die Werte dieser Parameter können auf verschiedene Arten verändert werden: Über zwei *javax.swing.JSlider* und zwei *javax.swingJTextField*s können verschiedene Parameterwerte gewählt werden. Der Reset-*javax.swing.JButton* setzt die Parameter wieder auf ihre

Ursprungswerte zurück. Das *AdjustmentPanel* implementiert die beiden Interfaces *java.awt.event.ActionListener* und *javax.swing.event.ChangeListener*. Die Methoden *public void actionPerformed(ActionEvent e)* und *public void stateChanged(ChangeEvent e)* setzen nach der Benutzereingabe die gewählten Parameterwerte des darzustellenden *PlotPanels.*.

Bei der Gestaltung seines Layouts bedient sich das *AdjustmentPanel* der Methoden der *Helper*-Klasse. Diese Klasse besteht ausschließlich aus static Attributen und Methoden, die evtl. noch für andere Anwendungen nutzvoll sein können.

4.4 Das Package Transformation

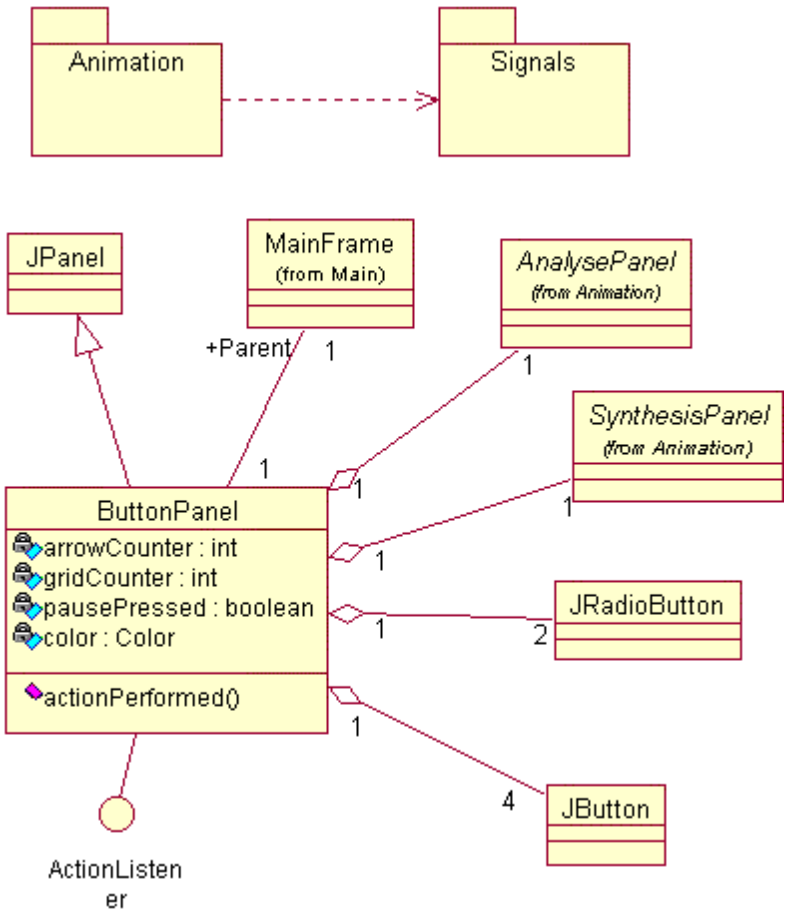

Abb. 6: Die Klassenstruktur des Package Transformation.

In *Transformation* existieren die zwei Unterpakete *Signals* und *Animation*. Während sich das Paket *Signals* (Kapitel 4.4.1) auf die Darstellung von Signalen und Filtern konzentriert, beinhaltet das Paket *Animation* (Kapitel 4.4.2) all jene Klassen, die direkt mit der Animation der Multiskalenanalyse verbunden sind.

Beide Pakete sind wie das Paket *Main* auf realer Klassenebene nicht existent. Sie dienen allein der Übersichtlichkeit dieser Studienarbeit. Ihre Klassen sind auf der selben Ebene wie das *ButtonPanel* anzuordnen.

Die Klasse *ButtonPanel* –Subklasse von *javax.swing.JPanel*- ermöglicht einem externen Benutzer, auf die Animation Einfluß zu nehmen. Die zwei *javax.swing.JRadioButtons* entscheiden darüber, welches *TransformationPanel* –*AnalysePanel* oder *SynthesisPanel*- gerade aktiv ist. Das *ButtonPanel* besitzt außerdem vier *javax.swing.JButtons*. Mit dem Startbutton wird je nach gewähltem *TransformationPanel* entweder die Animation der Analyse oder die der Synthese gestartet. Diese kann mit dem Pausebutton angehalten werden. Der Resetbutton setzt alle Anfangswerte neu und stellt den Anfangsbildschirm der Analyse bzw. der Synthese wieder her. Schließlich erhält der Benutzer bei angehaltener Animation durch den Detailbutton Detailinformationen über die Filter der aktuellen Transformation. *ButtonPanel* implementiert den *java.awt.event.ActionListener*, behandelt aber nur die auftretenden *java.awt.event.ActionEvents* der *JButtons* selbst. Die Behandlung der *ActionEvents* der *JRadioButtons* übernimmt der Vaterframe *MainFrame* aus dem Package *Main*.

4.4.1 Signals

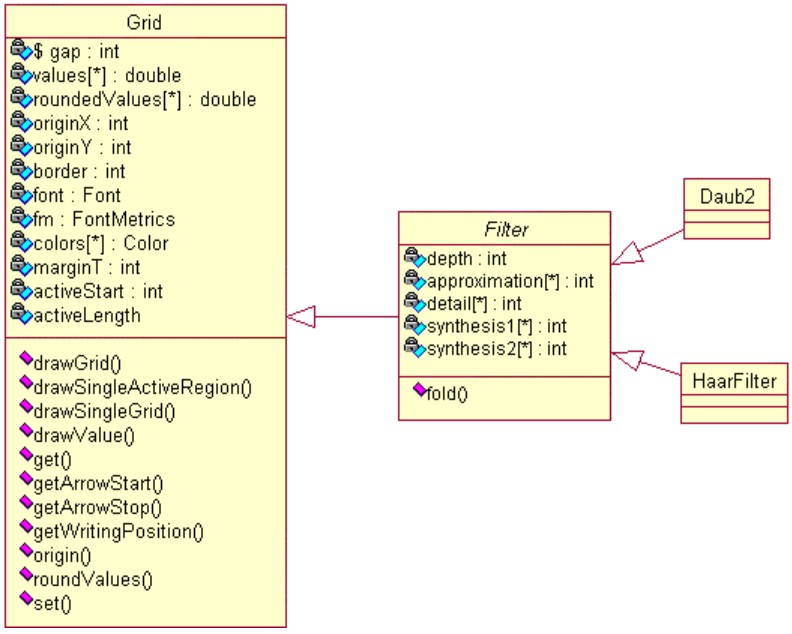

Abb. 7: Die Klassenstruktur des Package Signals.

Um ein eindimensionales, digitales Signal in einem Gitter darzustellen, benutzt man eine Instanz der Klasse *Grid*. Im array *values* werden die einzelnen Werte des Signals gehalten, im array *roundedValues* das auf zwei Nachkommastellen gerundete Pendant. Das static Attribut *gap* gibt den Abstand zwischen zwei Kästchenlinien an, die Attribute *originX* und *originY* die Koordinaten der oberen linken Ecke des Grids. Mit Hilfe der Methode *double get(int x)* bzw. *void set(int x)* erhält bzw. setzt man den Wert an der Stelle x. Die Andockkoordinaten für hereinkommende oder herausgehende Pfeile erhält man durch die Methoden *Point getArrowStart(int x)* und *Point getArrowStop(int x)*. Die zahlreichen *draw....(....)* Methoden stellen Möglichkeiten dar, ein Gitter sowie die dazugehörigen Werte des Signals zu zeichnen.

Die abstrakte Klasse *Filter* ist von der Klasse *Grid* abgeleitet. Sie erbt deshalb all ihreEigenschaften. Das Besondere an Filtern ist, dass sie die Arrays *approximation, detail, synthesis1* und *synthesis2* für die Speicherung der verschiedenen Low- und High-Pass-Filter einer Filterbank zur Verfügung stellt. Außerdem kann diese Klasse durch die Methode *void fold(Grid x, Grid[] y)* eine Multiskalenanalyse der Tiefe *y.length* des Signals *x* durchführen. Dabei wird das in der Analysetiefe *depth* erhaltene transformierte Signal in das Grid *y[depth]* übertragen.

Die konkreten Klassen *Daub2* und *HaarFilter* sind Subklassen von *Filter*. Der einzige Unterschied besteht darin, dass diese Klassen in ihrem Konstruktor konkrete Werte für die jeden einzelnen Filter einsetzen.

4.4.2 Animation

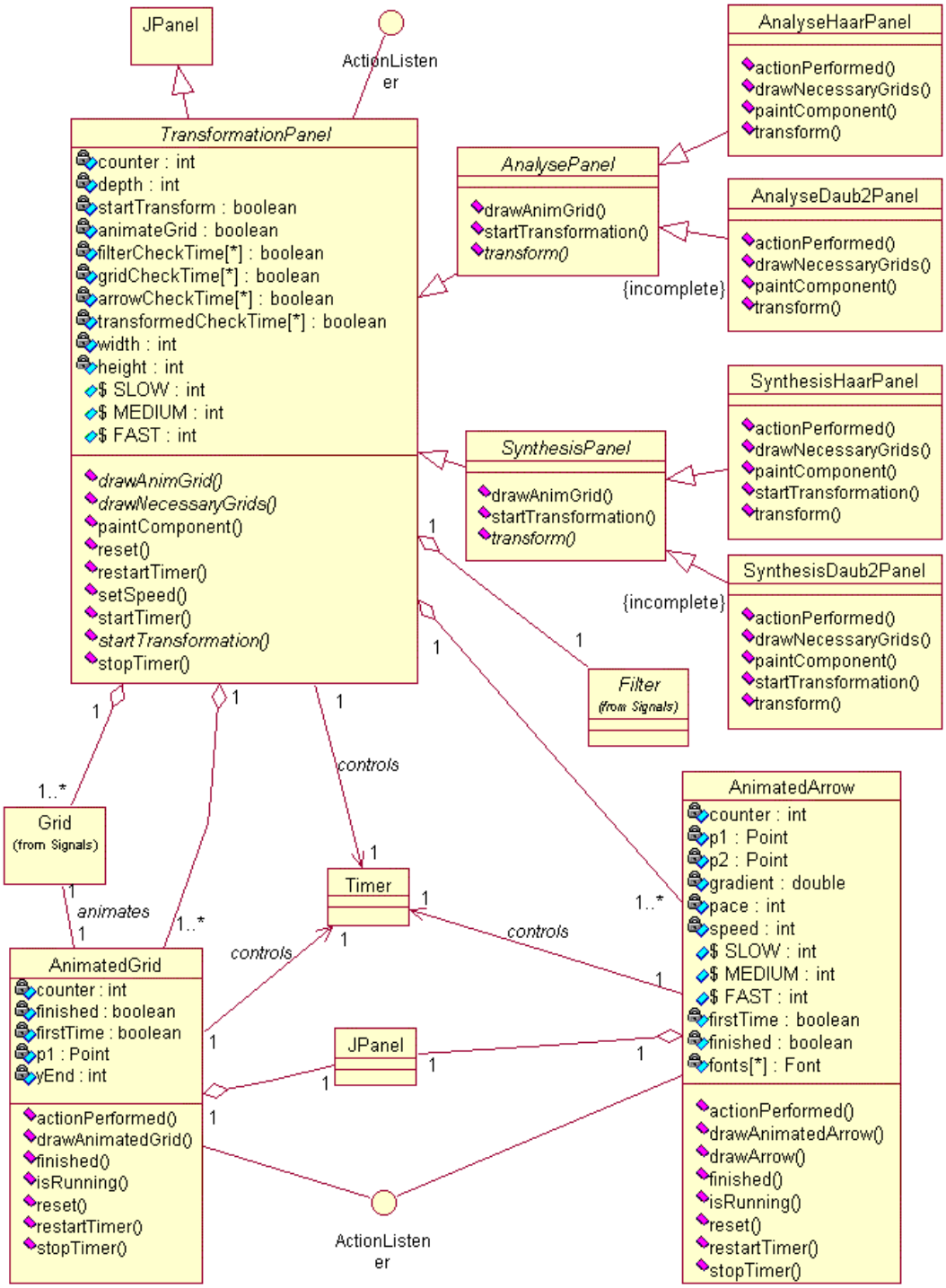

Abb. 8: Die Klassenstruktur des Package Animation.

Das generelle Problem bei der Erstellung von Animationen bzw. von animierten Objekten unter Java ist, dass es solche nicht schon in vorgefertigten Klassenbibliotheken gibt. Man muss also wie beim Daumenkino „per Hand“ einzelne Bilder zeichnen, die man entsprechend

schnell hintereinander ablaufen lässt, so dass sie dem Betrachter wie ein animiertes Bild erscheinen. Da eine Klasse an sich ein statisches Gebilde ist, tritt ein weiteres Problem auf. Wie kann eine Klasse in Abhängigkeit von der Zeit verschiedene Bilder zeichnen, ohne dass jedes Bild einzeln abgespeichert werden muss? Eine mögliche Lösung ist der Einsatz von booleschen Variablen, die je nach Zeitpunkt ein- oder ausgeschaltet werden. Aber wie bestimmt man diese Zeitpunkte und wie trennt man die Schaltung der booleschen Variablen von der grafischen Darstellung? Die Implementation der folgenden Klassen stellt einen Lösungsversuch für diese Probleme dar.

Um eine animierte Multiskalenanalyse grafisch darzustellen benutzt man eine Subklasse der abstrakten Klasse *TransformationPanel*, die von *javax.swing.JPanel* abgeleitet ist. Elementares Element dieser Klasse ist ein *javax.swing.Timer*-Objekt. In gewissen Zeitabständen feuert dieses Objekt *java.awt.event.ActionEvent*s ab. Diese werden implizit von der Methode *public void actionPerformed(ActionEvent e)* abgefangen. Bei jedem Aufruf dieser Methode wird das Attribut *counter* inkrementiert. Wenn es einen bestimmten Wert erreicht hat (somit ist eine Transformation des Signals abgeschlossen), wird das Attribut *depth* inkrementiert und *counter* wieder auf null gesetzt. In Abhängigkeit der Ausprägungen dieser beiden Variabeln werden die boolean Attribute des *TransformationPanel*s auf „true“ oder auf „false“ gesetzt. Zum Schluss wird dann die Methode *repaint()* aufgerufen, die wiederum für das erneute Zeichnen des Panels verantwortlich ist.
Gezeichnet wird in der Methode *void paintComponent(Graphics g)*. Die Ausprägungen der boolean-Attribute entscheideen darüber, welche Komponenten -*Grids, Filter, AnimatedGrids* und *AnimatedArrows*- an welcher Stelle des Panels gezeichnet werden sollen. Zuerst wird die Methode void *drawNecessaryGrid(Graphics g)* aufgerufen, die in Abhängigkeit der Transformationstiefe (*depth*) die schon transformierten Signale in der oberen Hälfte des Panels untereinander aufreiht. Diese Methode ist in *TransformationPanel* noch abstrakt und muss deshalb von Subklassen überschrieben werden. Wenn die Animation bereits gestartet wurde, wird nun die Methode *void startTransformation(Graphics g)* aufgerufen, in der bei abgeleiteten Klassen die eigentliche Transformation implementiert werden soll. Nach Abschluss einer Transformation wird jeweils einmal die Methode *void drawAnimGrid(Graphics g)* aufgerufen, die das gerade transformierte Signal vom unteren Bildrand zu den oben aufgereihten, bereits transformierten, Signalen bewegt. Auch diese Methode ist abstrakt und muss noch von Subklassen überschrieben werden.

Die Methode *void startTimer()* aktiviert den *Timer* das erste Mal, so dass er von nun an *ActionEvents* abfeuert. Gestopt wird das *Timer*-Objekt durch Aufruf der Methode *void stopTimer().* Alle an der Transformation beteiligten, animierten Objekte (*AnimatedGrid*- oder *AnimatedArrow*-Objekte), die zu diesem Zeitpunkt aktiv sind, werden ebenfalls angehalten. Die Animation wird somit vollständig unterbrochen. Sie wird durch die Methode *void restartTimer()* erneut gestartet.
Durch Aufruf der Methode *void setSpeed(int speed)* mit den static Argumenten *SLOW, MEDIUM* und *FAST* als Argument kann die Geschwindigkeit der gesamten Animation eingestellt werden.

Die abstrakten Klassen *AnalysePanel* und *SynthesisPanel* sind direkte Subklassen des *TransformationPanels*. Während das *AnalysePanel* als Superklasse für solche Klassen gilt, die die Analyse eines Signals darstellen, gilt das *SynthesisPanel* als Superklasse für solche Klassen, die die Synthese eines bereits analysierten Signals darstellen. Beide Panels überschreiben die Methoden *void drawAnimGrid(Graphics g)* und *void startTransform(Graphics g)* der Superklasse. Hinzu kommt die abstrakte Methode *void transform(Graphics g, int i, int pos, Color color)*. Während *startTransform(Graphics g)* zur analyse- bzw. synthesespezifischen Vorbereitung der Animation dient, wird in *transform(...)* die eigentliche Transformation wie bereits oben besprochen implementiert.

Die Klassen *AnalyseHaarPanel* und *AnalyseDaub2Panel* sind schließlich von *AnalysePanel* abgeleitet. Die eine implementiert die Analyse mit einer Haar-Filterbank, die andere die Analyse mit einer Daubechies2-Filterbank. Beide überschreiben in der oben genannten Art die noch übriggebliebenen abstrakten Methoden *drawNecessaryGrids(Graphics g)* und *transform(...)* der Superklasse sowie die Methode public void *actionPerformed(ActionEvent e)* des Interfaces *java.awt.event.ActionListener*.

Analog sind die Klassen *SynthesisHaarPanel* und *SynthesisDaub2Panel* als Subklasse des *SynthesisPanels* implementiert.

Weitere Animationsobjekte dieses Packets sind Instanzen der Klassen *AnimatedGrid* und *AnimatedArrow*. Sie funktionieren nach dem gleichen Prinzip wie die *TransformationPanels*. Ein Timer-Objekt feuert in regelmäßigen Abständen *java.awt.event.ActionEvents* ab, die von

der Methode *public void actionPerformed(ActionEvent e)* abgefangen werden. So steuert das jeweilige *Timer*-Objekt die gesamte Animation dieser Klassen.

Ein *AnimatedGrid* ist eine Klasse, die genau ein *Grid* besitzt, welches auf einem Panel von dem Startpunkt *p1* zu dem Endpunkt (p1.x, *yEnd*) bewegt wird. Es wechselt also während der Animation nicht seine X-Koordinate, sondern bewegt sich senkrecht nach oben oder nach unten. Realisiert wird diese Animation in der Methode *void drawAnimatedGrid(Graphics g, Point p1, int y).* Beim ersten Aufruf dieser Methode wird das *Timer*-Objekt aktiviert und das *Grid* 0.75 Pixel oberhalb (bzw. unterhalb) seiner Ausgangsposition gezeichnet. Bei jeder Ausführung der Methode *actionPerformed(ActionEvent e)* wird erneut die Methode *drawAnimatedGrid(...)* aufgerufen, die das *Grid* um weitere 0.75 Pixel nach oben (bzw. nach unten) verschiebt. Wenn das *Grid* seinen Endpunkt erreicht hat, wird das *Timer*-Objekt deaktiviert.

Die Klasse *AnimatedArrow* zeichnet einen animierten Pfeil vom Startpunkt *p1* zum Endpunkt *p2*. Beim Aufruf der Methode *void drawAnimatedArrow(Graphics x, Point p1, point p2, String label)* wird beim ersten Mal das *Timer*-Objekt gestartet und die Steigung (*gradient*) zwischen den beiden Punkten *p1* und *p2* berechnet. Bei diesem und bei allen übrigen Malen wird dann ein Pfeil gezeichnet, der die Steigung von *gradient* besitzt und um 1 Pixel länger ist als der vorherige Pfeil. Die Methode drawAnimatedArrow(...) wird bei jeder Ausführung der Methode *actionPerformed(ActionEvent e)* erneut aufgerufen. Wenn der *AnimatedArrow* seinen Endpunkt erreicht, wird das *Timer*-Objekt deaktiviert.

5. Benutzeroberfläche

Nachdem der Button „Start Multiresolution Analysis“ des, in der HTML-Seite eingebundenen, Applets gedrückt wurde, öffnet sich das Hauptfenster außerhalb des Browsers. In ihm sind alle wichtigen Funktionalitäten der Anwendung enthalten.

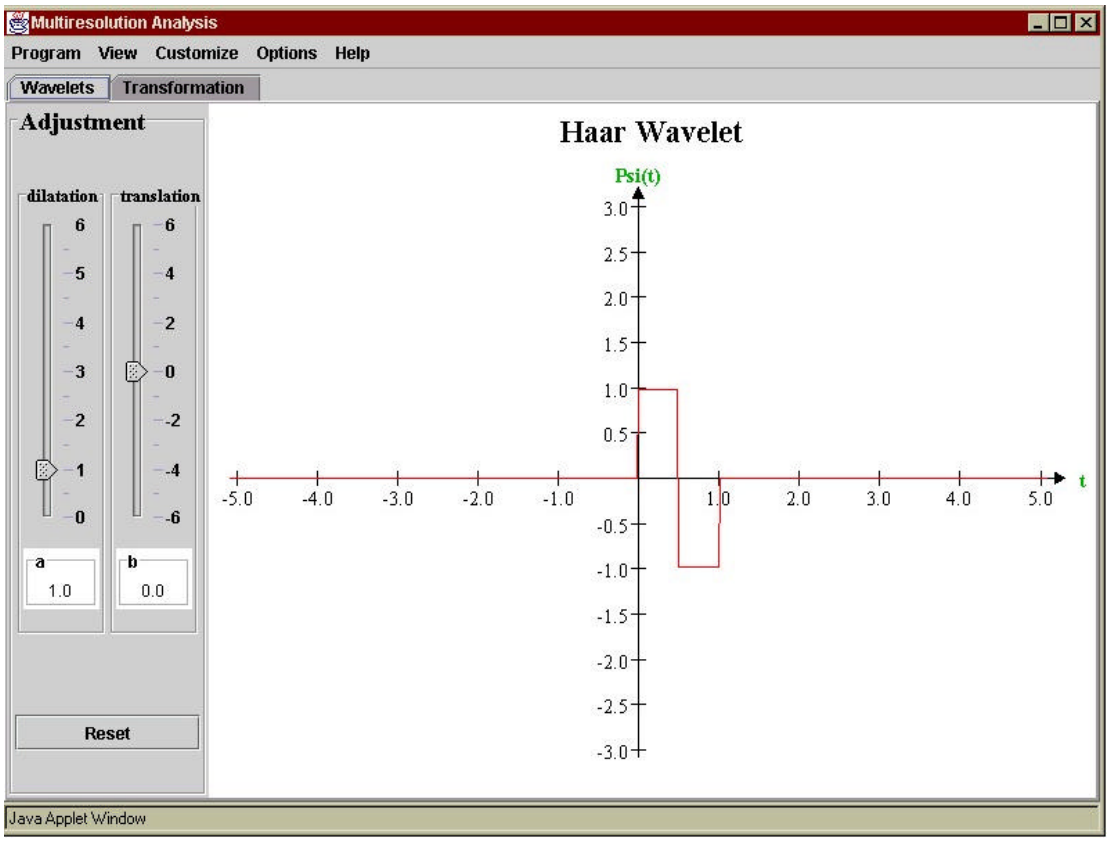

Abb. 9: Das Hauptfenster nach dem ersten Ladevorgang.

Das Layout des Hauptfensters wird durch ein JTabbedPane bestimmt. Dieser Container ordnet seine Komponenten suggestiv als Karteiregister auf. Es kann jeweils immer nur eine Karteikarte aktiv selektiert sein. Das Blättern durch die einzelnen Karteikarten erfolgt einfach durch Mausklick auf die Registerlaschen. Dieses Layout verdeutlicht besonders die Aufteilung der Studienarbeit in zwei unterschiedliche Teilbereiche. Die erste Karteikarte mit der Registerlasche „Wavelets“ beschäftigt sich mit der Darstellung von Wavelets als Funktionen und entspricht der Waveletansicht (siehe Kapitel 5.1). Die Zweite mit der Registerlasche „Transformation“ enthält die Animation der Multiskalenanalyse mit Wavelet-Filterbanken und entspricht der Transformationsansicht (siehe Kapitel 5.2).

Die Menueleiste des Hauptfensters ermöglicht dem Benutzer verschiedene Einstellungen vorzunehmen bzw. bestimmte Aktionen durchzuführen. So besteht -neben dem Mausklick auf die Registerlaschen- eine weitere Möglichkeit zwischen den beiden Ansichten zu wechseln:

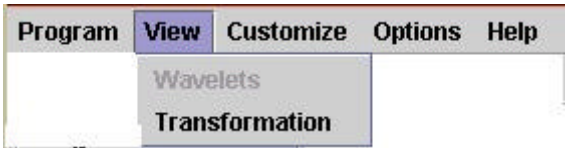

Abb. 10: Das Menue View.

Im Menue *View* der Menueleiste wählt man zwischen *Wavelets u*nd *Transformation* aus und gelangt so zu der gleichnamigen Ansicht.

Die beiden Menues *Customize* und *Options* bieten beide Aktionen zur Auswahl an, die an die jeweils selektierte Karteikarte gebunden sind. Sie werden daher ausführlich in den jeweiligen Kapiteln besprochen.

Über das Menue *Help-Index* werden die Help-Seiten des Applets geladen. Um schließlich die Anwendung zu beenden, selektiert man im Menue *Program* das Untermenue *Quit*. Eine weiter Möglichkeit besteht darin, den Close-Button der Kopfzeile zu drücken. Mit den übrigen dort angesiedelten Buttons lässt sich das Hauptfenster minimieren, maximieren oder ikonifizieren.

5.1 Die Waveletansicht

Die Wavelets-Karteikarte dient der Darstellung jeweils eines Wavelets, das durch Benutzereingaben gestaucht, gestreckt und verschoben werden kann.

Das Plotpanel auf der rechten Seite der Karteikarte zeigt die jeweils aktuelle Waveletfunktion $\Psi_{a,b}(t)$ in einem Koordinatenkreuz. Das darüber angesiedelte Headlinepanel gibt an, welches Wavelet gerade selektiert ist. Ein anderes Wavelet kann man unter dem Menuepunkt Options-ChooseWavlet der Menueleiste des Hauptfensters wählen.Zur Auswahl stehen das Haar-, das Mexican-Hat und das Morlet-Wavelet.

Abb. 11: Menue Options-Choose Wavelet.

Auf der linken Seite der Karteikarte befindet sich das Adjustmentpanel. Dieses ermöglicht dem Benutzer die Manipulation am Wavelet durch die Parameter a und b.

Abb. 12: Das Adjustment-panel.

Bewegt man den linken Slider des Adjustmentpanels, verändert man den Dilatationsparameter a. Durch ihn kann der Benutzer das Wavelet stauchen oder strecken. Diesen Parameter kann man aber auch durch die Eingabe einer Fliesskommazahl in das linke Textfeld a erreichen. Bewegt man hingegen den rechten Slider des Adjustmentpanels, verändert man den Translationparamter b. Erverschiebt das Wavelet nach an der X-Achse des Koordinatenkreuzes. Denselben Effekt erzielt man durch Eingabe einer Fließkommazahl in das rechte Textfeld b. Darüberhinaus bietet das Adjustment Panel die Möglichkeit alle vorgenommenen Änderungen am Wavelet durch Drücken des Reset-Buttons wieder zurückzunehmen.

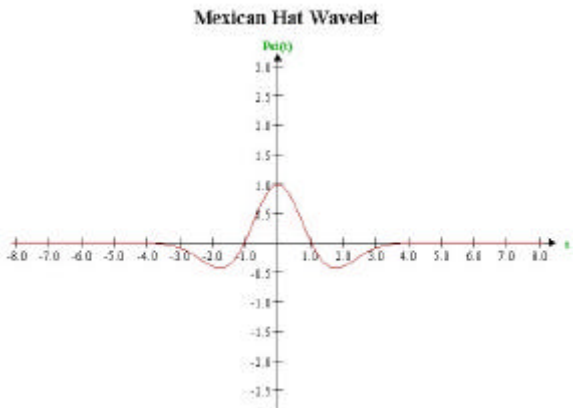

Abb. 13: Das Mutter-Mexican-Hat-Wavelet mit a=1 und b=0 .

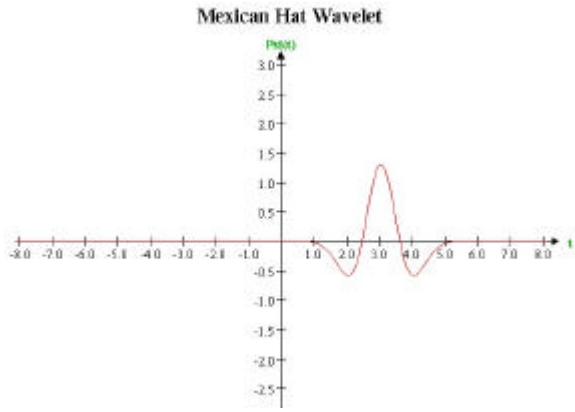

Abb. 14: Das Mexican-Hat-Wavelet mit a=0.5 und b=3.0 .

5.2 Die Transformationsansicht

Auf der Karteikarte Transformation findet die animierte Multiskalen-Analyse mittels Wavelet- Filtern statt. Es kann sowohl die mehrfache Analyse des Signals, als auch die mehrfache Synthese durchgeführt werden.

Das Transformationpanel zeigt die jeweilige Animation. Dabei durchläuft in der Analyse-Animation ein digitales Ausgangssignal (*Signal*) mehrere Analyse-Transformationen. In jedem Durchlauf werden die Low-Pass- und High-Pass-Filter sukzessiv über das Signal gelegt. Die so berechneten, transformierten Signalwerte werden in ein neues transformiertes Signal (*x.Transformed*) übertragen. Nach Beendigung jedes Durchlaufs wird das vorher

transformierte Signal zum Inputsignal der nächsten Analyse. So wird fortgefahren, bis keine weiteren Durchläufe mehr möglich sind. Für die Synthese nimmt man das letzte transformierte Signal als erstes Input-Signal. Dieses unterläuft nun der Faltung mit den Synthese-Low- und –High-Pass-Filtern. Wie bei der Analyse wird so lange fortgefahren, bis das Ausgangssignal wieder vollständig rekonstruiert ist.

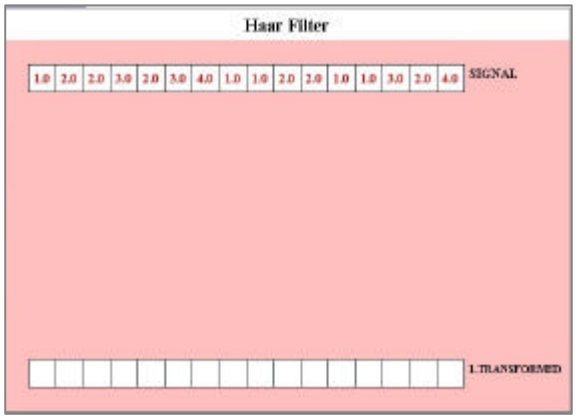

Abb. 15: Das Transformationpanel in seinem Analyse-Ausgangszustand.

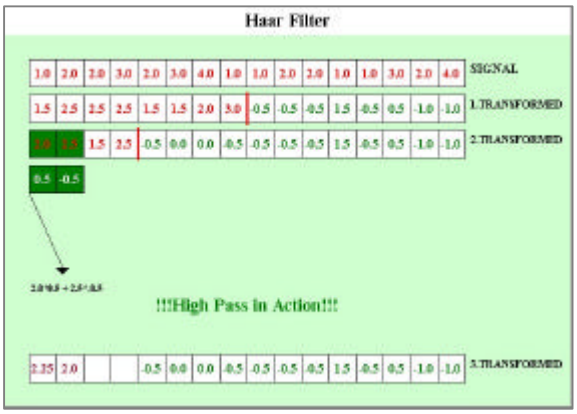

Abb. 16: Das Transformationpanel bei der Analyse mit dem Haar-High-Pass –Filter.

Das Headlinepanel über dem Transformationpanel gibt an, welche Filterbank gerade in der aktuellen Animation verwendet wird.

Abb. 17: Das Buttonpanel.

Mit Hilfe des Buttonpanels auf der linken Seite der Karteikarte kann der Benutzer nun diese Animationen steuern. Benutzt er den Startbutton, wird die Animation gestartet. Durch den Pausebutton kann er die Animation anhalten. Der Resetbutton schließlich setzt die Animation in ihre Ausgangssituation zurück. Die beiden Radiobuttons Analysis und Synthesis ermöglichen dem Benutzer zwischen eine der beiden Transformationsarten zu wählen. Wechselt er die Transformationsart, wird der Zustand der letzten Animation gemerkt. Über den Detailbutton kann sich der Benutzer Detailinformationen über die aktuelle Filterbank einholen. Ein Informationsdialog zeigt bei Betätigung des Knopfes alle Filter der Filterbank an.

Um nun aber die Filterbank zu wechseln, muss man wieder auf die Menueleiste des Hauptfensters zurückgreifen. Unter dem Menue *Options-Choose Filter* kann zwischen der Haar-Filterbank (*Haar Filter*) und der Daubechies2-Filterbank (*Daubechies2 Filter*) gewählte werden. Weitere mögliche Einstellungen, die über die Menueleiste vorgenommen werden können, sind die Einstellung der Animationsgeschwindigkeit über das Menue *Customize-Speed* und die Änderung der Animationsfarben für rot-grün-blinde Menschen über die Checkbox *Red-green-Blinds* im gleichen Menue.

Abb.18: Das Menue Options-Choose Filter.

6. Literaturverzeichnis

[1] Martin Schader / Lars Schmidt-Thieme. *Java – Eine Einführung, 3.Auflage*, Springer-Verlag 2000.

[2] Barbara Burke Hubbard. *The World according to Wavelets, Secon Edition*, A K Peters Natick-Verlag 1998.

[3] Wolfgang Effelsberg. *Vorlesung Multimediatechnik WS00/01*, Universität Mannheim 2000. Siehe auch:
http://www-mm.informatik.uni-mannheim.de/veranstaltungen/ws20002001/multimedia/

[4] Christoph Esser. *Wavelet Transformation von Standbildern*, Universität Mannheim 2001. Siehe auch:
http://www-mm.informatik.uni-mannheim.de/veranstaltungen/animation/multimedia/wavelet/WaveletDemo.html