Kapitel 2

Y - ein Prototyp

2.1 Konzept, Werte, Ziele der Software

2.1.1 Zielpersonen

Was ich persönlich in der Menge der mir bekannten Software vermisse, ist im vorigen Kapitel hoffentlich deutlich genug zum Ausdruck gekommen. Dabei will ich niemanden missionieren, im Gegenteil entspricht es meiner Ambition nach Divergenz, dass Komponisten ihren eigenen Weg gehen, technisch wie musikalisch. Wem meine theoretische oder Softwarearbeit dabei als Inspiration dient, der ist herzlich willkommen, aber ich möchte niemandem meine Wertvorstellungen aufdrängen. Auf der anderen Seite möchte ich bei Y keine Kompromisse eingehen, die durch Feature-Wünsche von anderen Leuten entstehen.

> Im Prinzip verselbständigt sich so ein Projekt mit der Größe des Kundenkreises, da zu über 90% nur noch die (teilweise genialen) Vorschläge unserer Kunden umgesetzt werden müssen, wenn sich ein Programm erst einmal etabliert hat.[9]

...sagte ein langjähriger Entwickler der Software-Firma Steinberg über ein Programm, das mittlerweile ausgestorben ist.

2.1.2 Musikalische Ziele

An der Konzentration auf Computertonbandmusik möchte ich auch bei der praktischen Umsetzung festhalten. Dass natürlich immer ein gewisser Spielraum zur Dehnung des Anwendungsfeldes besteht, ist auch klar. Die offene Struktur der Software ermöglicht über flexible Schnittstellen wie OSC auch die Anbindung an Multimedia-Software wie Processing oder Jitter/GEM.

Unter anderem plane ich eine Studie über lineare Beschleunigungen und die De-Instrumentation eines Orchesterstücks auf adaptive Filtervorgänge. Bei Projekte werden nicht in der angestrebten Qualität realisierbar sein ohne entscheidende Durchbrüche bei der Arbeit mit Metapartitur und der Rückführung von manuellen Änderungen an der Partitur in Syntheseprozesse.

Außerdem habe ich vor, einen Satie-Zyklus für CD zu produzieren, mit Mezzosopran, seltenen Instrumenten und Elektronik. Dabei möchte ich ein System zur Variantenbildung einsetzen, das einen engen und sehr ergonomischen Austausch zwischen Klangmontage, Klanganalyse und Klangsynthese erfordert. Auch dafür werde ich diese neue Software brauchen.

Eine Musikalische Anwendung, die bewusst ausgeklammert wird, ist das Sequencing auf der Basis von MIDI-Noten und Mehrzweck-Synthesizern. Dabei handelt es sich um ein teuflisches Verfahren, das mit sehr schnellen Klangergebnissen und einfacher Datenreduktion lockt. Gerade bei einem an Ergonomie und freier Exploration orientierten Arbeitsfluss ist jedoch die die Gefahr der Entstehung von Keyboardmusik zu groß.

2.1.3 Angestrebte Arbeitsweise

Die Möglichkeiten zu divergentem, experimentellem, explorativem Arbeiten sollen im Vordergrund stehen. Das Bilden von Varianten, Auswählen, Verwerfen, Trennen und Verbinden von Klang- und Partiturfragmenten soll durch gute Infrastruktur schnell und ergonomisch ablaufen können. Das Ziel ist ein kreativer Fluss mit sehr kurzen Interaktionszyklen im Übergang zwischen Komposition und Improvisation. Zeitkritische Momente von Inspiration und Motivation sollen effizient ausgeschöpft werden und nicht in redundanter Klickerei versacken.

Grundsätzlich halte ich konvergentes Arbeiten für ebenso wichtig, und die entwickelte Software ermöglicht natürlich auch eine stark vorausplanende, problemorientierte Vorgehensweise. Allerdings stellt konvergente Produktion nicht die gleichen Ansprüche an den Zeitpunkt, an dem sie statt findet. Darüber hinaus sind konvergent erreichbare Ziele bereits zufriedenstellend durch andere Software abgedeckt.

Audioseitig könnte man das Ziel von Y als Erweiterung von traditionellem Tonschnitt betrachten, wobei im Idealfall die Bandmaschine durch eine flexible Samplingtechnik und das Mischpult durch einen modularen Synthesizer ersetzt wird.

Einige Konzepte aus der Studiotradition sollen dabei übernommen werden: Die kontextbezogene Tonaufnahme (Overdubbing) direkt in der Partitur halte ich für eine wichtige Funktion. Dazu lege ich Wert auf ein ergonomisches Schnitt verfahren und das sogenannte „Reamping“[1].

Wichtig ist mir auch kontextsensitive, rückgekoppelte Partitursynthese.

[1]Darunter versteht man ein Verfahren, bei dem ein aufgenommenes Audiosignal zu einem Verstärker plus Lautsprecher geleitet wird, um dann per Mikrofon wieder aufgenommen zu werden. Der Zweck ist eine Klanggestaltung, die die spezifischen Eigenschaften von Verstärker(n), Lautsprecher(n) und Mikrofon(en) sowie der akustischen Umgebung nutzt. Leider bietet keiner der mir bekannten Audiosequenzer eine ergonomische Unterstützung dieses Verfahrens.

2.1.4 Ansprüche an die Software

- Grafischer wie schriftlicher Zugriff auf Partiturdaten
- Bearbeitung von Klängen im Kontext der Partitur
- Sinnvolle Gruppierung von zusammenhängenden Daten
- Alternative zum Spuren-Konzept im Sequenzer
- Schnittstelle zum Betriebssystem und anderen Programmen
- Flexible Signalverarbeitung

2.2 Technische Realisierung

2.2.1 Allgemeines zum Prototyp

Bjarne Stroustrup schreibt in seinem Standardwerk „Die C++-Programmiersprache“:

> Zu Beginn eines ambitionierten Entwicklungsprojekts ist die beste Möglichkeit zur Strukturierung des Systems noch nicht bekannt. Oft weiß man noch nicht einmal präzise, was das System leisten soll, da Details erst im Verlauf der Bemühungen, das System zu entwickeln, zu testen und zu benutzen, klar werden. [...] Die populärste Form des Experiments scheint die Entwicklung eines Prototyps zu sein. Dabei handelt es sich um eine verkleinerte Version des Systems oder eines Teil davon. [...] Das Ziel besteht darin, so schnell wie möglich eine laufende Version zu bekommen, um eine Untersuchung von Design- und Implementierungsalternativen vornehmen zu können. [6, S. 762]

Der Prototyp stellt nicht die endgültige Software dar, sondern dient als Modell, um heraus zu finden, ob und wie bestimmte geplante Konzepte funktionieren oder nicht. „Fertig“ ist der Prototyp dann, wenn durch ihn ausreichend Erkenntnis gewonnen ist, um die eigentliche Software von vorne, sozusagen in Reinschrift, zu implementieren. Dieser Neubeginn baut möglicherweise auf anderer Software auf, mit verstärktem Schwerpunkt auf Performance, Stabilität, Veröffentlichbarkeit (u.a. Lizenzen), und so weiter. Der Quellcode des Prototyps wird dabei nicht weiter verwendet. Daher macht es Sinn, im Prototyp nur diejenigen Elemente auszuformulieren, die für das Skelett unbedingt notwendig sind. Optimiert, geputzt, erweitert, abgesichert und verschönert wird in einer anderen Entwicklungsphase, zu der ich vorhabe, mich nach dieser Arbeit mit einem erfahrenen Informatiker zusammen zu tun. Vor allem GUI-Programmierung ist ein Fachgebiet, das in

seiner vollen Komplexität meine technischen Fähigkeiten und Ambitionen übersteigt. Auch wenn mir die Umsetzung der Grundfunktionalität im Rahmen des Entwurfs nach meinen Vorstellungen gelungen ist, sind bei der Arbeit doch Ideen für die Endversion entstanden, die nicht ohne Expertenwissen realisiert werden können.

Der Arbeitstitel „Y“ entstammt dem Bedürfnis nach einem Namen, der schnell zu tippen ist. Er wird „Ü“ ausgesprochen.

2.2.2 Offenes System

Der Großteil verbreiteter Software wird heute in Form von einzelnen, geschlossenen Anwenderprogrammen mit einer einzigen, meist grafischen Benutzerschnittstelle angeboten. Besonders bei kommerziellen Produkten ist das die Regel, denn die Geschlossenheit einer Applikation begünstigt entscheidende Qualitätsmerkmale, darunter einfache Installation und Programmstart, einsteigerfreundliche Bedienung, kurze Datenwege und Stabilität.

Stattdessen überwogen für die Realisierung eines Prototypen von Y die Kriterien

- Kurze Entwicklungszeit
- Universelle Skriptsprache (Objektorientierung, Unix-Einbindung)
- Gleichberechtigung von Partitursynthese und -Montage

Es bot sich daher an, das Projekt in drei Aufgabenbereiche einzuteilen - grafische Benutzerschnittstelle, Signalverarbeitungs-Einheit und Skript-Schnittstelle - und jeden dieser Bereiche mittels bestehender Spezialsoftware zu implementieren.

2.2.3 Grafische Benutzerschnittstelle

Die grafische Benutzerschnittstelle wurde realisiert mittels “Processing“. Processing ist eine auf Java basierende Entwicklungsumgebung für Multimediaprojekte. Die Entscheidung für Processing hat mehrere Gründe:

- Processing-Framework und Library

  Aufgrund des unkonventionellen Bedienansatzes geht die grafische Schnittstelle über Standard-Knöpfe, -menüs und -textfelder weit hinaus, deswegen mussten spezielle GUI-Elemente entwickelt werden, wobei die Klassen und und Primitiven von Processing sehr hilfreich waren, nicht zuletzt wegen der Grafikbeschleunigung per OpenGL. Zudem erfordert der doppelte Datenzugriff sowohl per GUI als auch per Skript ein erhöhtes Maß an kinetischer Kontrolle, wofür die üblichen GUI-Libraries meines Wissens im allgemeinen nicht ausgelegt sind. Wie für viele andere Sprachen, existiert auch für Processing eine mächtige und schnelle XML-Library, die zum Speichern und Laden des Projekts zum Einsatz kommt, und eine OSC-Library zur Kommunikation mit anderen Teilen der Software siehe unten.

- Java

  Processing basiert auf der Programmiersprache Java, welche für die Aufgabe einen akzeptablen Kompromiss darstellt. Geschwindigkeit ist für die grafische Darstellung vieler Elemente ein Faktor. Die weitgehende Objektorientierung, Nebenläufigkeit und umfangreiche Klassenbibliothek bilden eine ausreichende Grundlage für das Programmdesign, welches sich am Architekturmuster “Document-View“ orientiert Eine Abwandlung des populären “Model-View-Controller“-Musters, bei dem die View-Klasse auch Steuerungsfunktionen übernimmt. Durch die strenge Typisierung, umfangreiche Syntax- und Semantikprüfung zur Übersetzungszeit wird die Fehleranfälligkeit verringert und damit der Entwicklungsprozess maßgeblich beschleunigt.

- Überschaubarer Arbeitsaufwand

  Nach einjähriger Erfahrung mit interaktiver Grafikprogrammierung in Processing konnte ich im Voraus relativ zuverlässig abschätzen, dass das Vorhaben im begrenzten zeitlichen Rahmen der Diplomarbeit umsetzbar sein würde. Mit anderen GUI-Systemen hatte ich bisher wenig Erfahrung und wollte daher kein Risiko eingehen.

2.2.4 Signalverarbeitung

Für Klangerzeuger, Effekte, Analysen, das Routing und die Ein- und Ausgabe von Audio und MIDI kommt ausschließlich SuperCollider zum Einsatz. Für die grundlegenden Aufgaben (Abspielen von Samples, Rauschen, Impulse, Grundwellenformen, Verstärker, Filter, Kompressor, Hall, Denoiser, etc.)[2] steht eine Datenbank von "Synths" zur Verfügung. Weil alle Synths in SuperCollider programmiert sind, können ohne viel Aufwand neue Synths dazu programmiert werden. Weil hierbei beliebiger SuperCollider-Code erlaubt ist, können eigene Synths außer Klang natürlich auch Dateizugriff, MIDI-Steuerung, etc. als Seiteneffekte beinhalten. Mittels OSC und Unix-Einbindung kann ein Synth auch andere Synthesesoftware wie Max/PD oder CSound ansteuern. Zur Weiterverarbeitung innerhalb von Y sind allerdings nur Audiosignale zulässig. Dabei habe ich mich bewusst gegen Mehrkanaligkeit entschieden, zumindest im Rahmen des Prototyps, um hemmende Konditionalitäten zu vermeiden. Das Signal-Routing ist auch in Mono kompliziert genug.

[2] nicht alle im Prototyp implementiert

2.2.5 Scripting

Wenn zwei Schnittstellen grafisch und schriftlich um ein und denselben Datensatz (die Partitur) konkurrieren, bieten sich allgemein drei Möglichkeiten an, wo die Daten gespeichert werden: bei der grafischen Schnittstelle, bei der Skript-Schnittstelle, oder außerhalb der beiden Schnittstellen, z.B. in einer Datei.

Aus Performancegründen, da die GUI in sehr kurzen Zeitabständen aktualisiert werden und daher schnell auf die Daten zugreifen können muss, blieb keine andere Wahl, als die Daten in der Java-Applikation abzulegen. Daher stellte sich für das Scripting die Frage, welcher Mechanismus sich zum Datenaustausch mit der laufenden Java-Applikation am besten eignet. Es gibt bereits eine Reihe von Softwarelösungen (“BeanShell“, “Groovy“, “Rhino“, “JRuby“, “Jython“ uvm.) die die Sprachen Python, Ruby, JavaScript, etc. für diesen Zweck anbieten. Daneben auch eine Lösung für SuperCollider in Form der Software “SwingOSC“ von Hanns Holger Rutz. Da sich auch die Programmiersprache von SuperCollider (“sclang“) hervorragend als Skriptsprache eignet, wurden hier zwei Fliegen mit einer Klappe geschlagen. Mittels SwingOSC wird nun die GUI-Applikation, die alle wichtigen Daten enthält, von sclang angesteuert. Damit wird sowohl die Vermittlung zwischen Partitur und Klangsynthese den Signalverarbeitungs-Einheit als auch von benutzergeschriebenen Skripten per SwingOSC angesteuert.

In der Praxis sieht das so aus: Zuerst startet man SuperCollider und initialisiert mit dem Befehl „Y.init“ die Klasse, die alle weiteren Vorbereitungen trifft (Server starten, OSC-Responder einreichten, etc.). Danach startet man das Processing-Applet, das sich automatisch bei SuperCollider anmeldet. Daraufhin teilt SuperCollider dem Applet die Einträge der Synth-Datenbank mit. Wird im Applet eine Session geladen, meldet das Applet automatisch das Session-Objekt bei SuperCollider an, und die Software ist einsatzbereit.

- Man kann das Applet neu starten, während SuperCollider weiter läuft.
- Man kann SuperCollider neu starten, während das Applet weiter läuft.

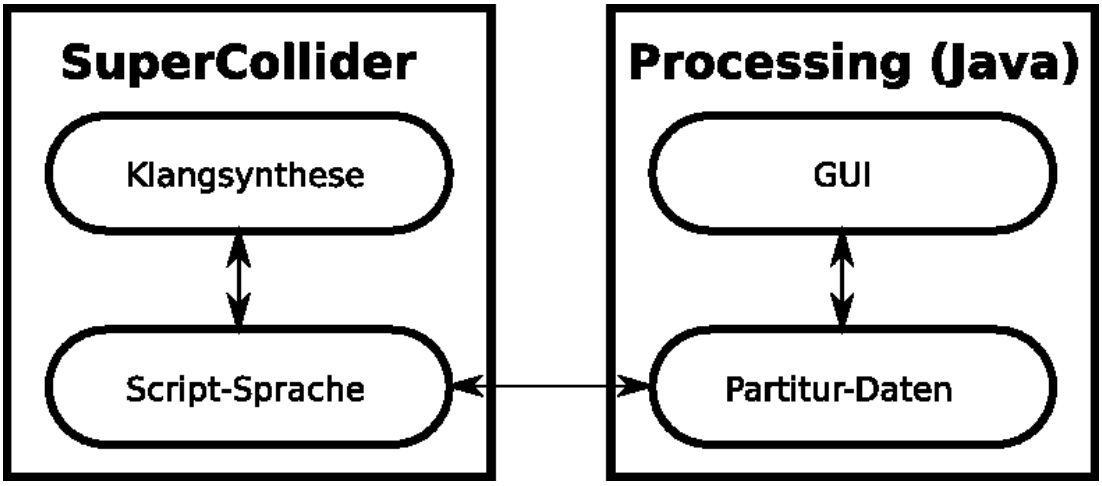

Abbildung 2.1: Kommunikation zwischen SuperCollider und Java

2.2.6 Datenspeicherung

Alle Daten einer Session werden in einer XML-Datei gespeichert. XML ist ein strukturiertes Textformat, man kann es mit jedem Texteditor öffnen, es ist sehr übersichtlich zu lesen und auch bequem in eigenen Programmen zu verarbeiten. Die Daten werden dort so gespeichert, wie sie auch in der GUI und damit im Scripting-Interface repräsentiert sind, nämlich als Baum.

Es gibt automatische Speicherung nach jedem Arbeitsschritt, zudem Undo und Redo, also die Gefahr von Datenverlust ist gering. Jeder Speichervorgang legt eine neue Datei im Backup-Verzeichnis an, so dass jederzeit auf ältere Versionen zurück gegriffen werden kann. Undo und Redo sind auch über Programmstarts hinweg verfügbar.

Voreinstellungen (Preferences), die das Programm betreffen, sind in der Software hart kodiert.

2.2.7 Performance

Die Performance des Prototyps ist überraschend gut. Spitzen in der CPU-Auslastung, die durch die Garbage-Collection von Java entstehen, fallen geringer aus als erwartet, es sind trotz der aufwendigen GUI gute Latenzzeiten möglich (2.9ms auf meinem System). Bei mehr als 1000 Partiturobjekten geht allerdings die Leistung in die Knie. Für den Prototyp ist das ein akzeptables Maß, schließlich wurde sowohl bei der Signalverarbeitung als auch bei der grafischen Anzeige auf prozessorfreundliche Sparmaßnahmen verzichtet, zugunsten einer kürzeren Entwicklungszeit.

2.3 Funktionsbeschreibung

Abbildung 2.2: Eine einfache Session in Y

2.3.1 Regionen und Breakpoints

Y übernimmt das Prinzip der für Sequenzer üblichen Zeitleiste, die Positionsmarke und die Anordnung der Partiturobjekte in der Zeit von links nach rechts. Partiturobjekte teilen sich bei Y in zwei Kategorien: Regionen und Breakpoints.

Breakpoints haben nur einen Zeitpunkt und einen skalaren Wert. Sie dienen zur Beschreibung von Hüllkurven, daneben finden sie aber auch Verwendung als zeitliche Markierungen, etwa zur Definition von Form- und Rhythmusrastern.

Regionen ähneln ihren gleichnamigen Pendants in Pro-Tools, Nuendo, Ardour, etc.. In der GUI erscheinen sie als längliche Rechtecke, die man mit der Maus hin- und herschieben kann. Sie haben einen Startzeitpunkt, eine Dauer, einen Namen, eine ganzzahlige Kennung (ID) und gegebenenfalls weitere Inhalte. Was sie von Regionen in Standard-Sequenzern unterscheidet, ist ihr Inhalt und ihre strukturbildende Funktion.

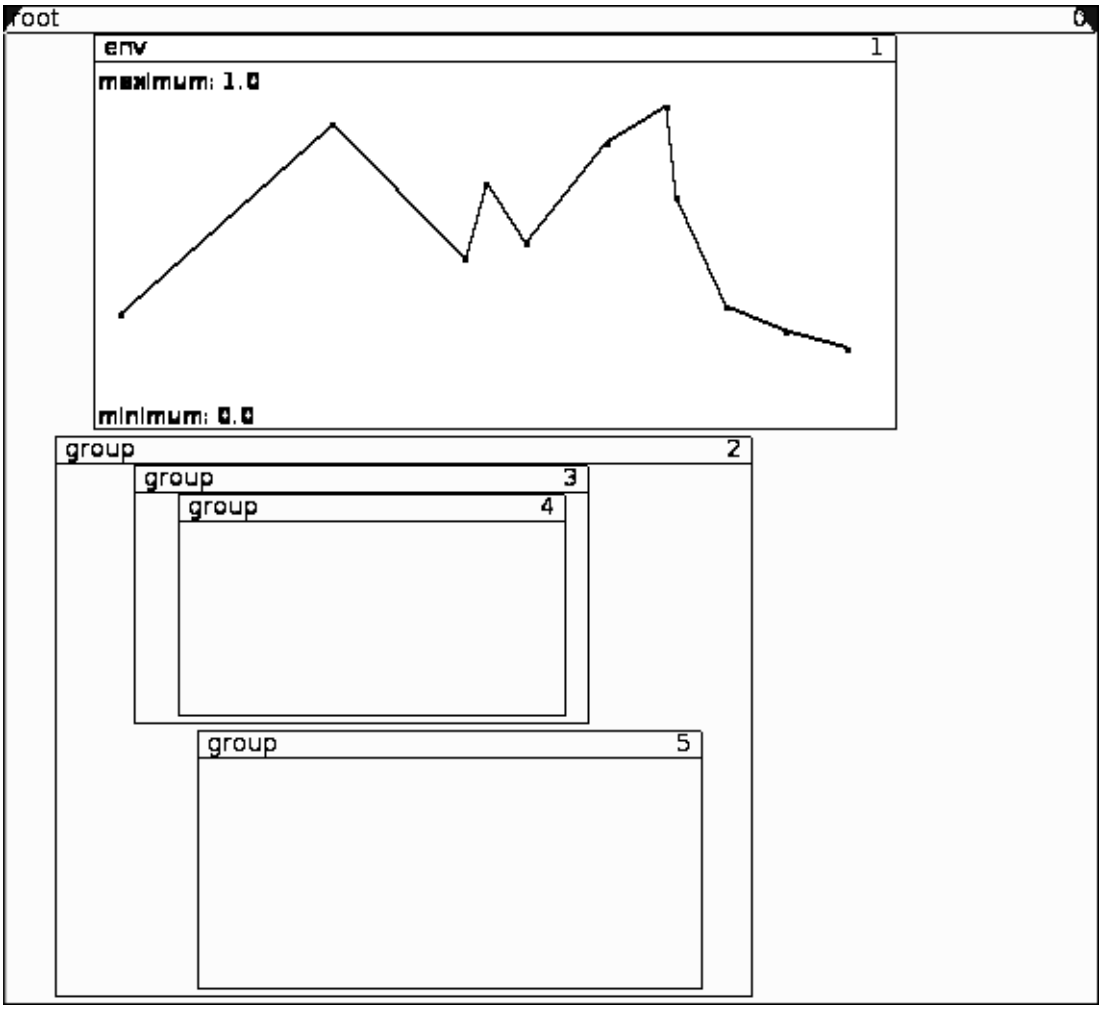

Abbildung 2.3: Verschachtelte GroupRegions und eine EnvRegion.

Die vier Typen von Regionen

- GroupRegion: Sie enthält nichts außer ggf. anderen Regionen
- SynthRegion: enthält einen Synth und ggf. andere Regionen
- OPRegion: enthält einen OP und ggf. andere Regionen
- EnvRegion: enthält Breakpoints

Regionen können ihrerseits wieder Regionen („Kinder“) enthalten[3]. Das ermöglicht eine baumförmige Verschachtelung der Partiturobjekte. Diese hierarchische Gruppierung dient der Abstraktion, vergleichbar mit einem Verzeichnisbaum oder mit Subpatches bei Max/PD. Vor allem wenn algorithmisch gearbeitet wird, können Anzahl, Dichte und Polyphonie schnell ein Maß erreichen, in dem das bloße Nebeneinander auf der Ebene keine übersichtliche und handhabbare Organisationsform mehr darstellt. Stattdessen bilden alle RegionenMitKindern zusammen eine dynamische Hierarchie. Auf höchster Ebene liegt eine einzige Region namens „Root“ oder Wurzel, die als einzige kein Parent[4] besitzt. Alle anderen Regionen sind entweder deren Kinder, Enkel, Urenkel, etc., bis in beliebige Generationstiefe. RegionenMitKindern können frei innerhalb der Hierarchie auf- und abwärts bewegt werden. Auch die Wurzel kann durch Veränderung der Baumstruktur ihres Amtes enthoben werden, so dass eine andere RegionMitKindern ihren Platz einnimmt. Insofern herrscht völlige Gleichberechtigung unter den RegionenMitKindern: Jede von ihnen kann einen beliebigen Platz in der Hierarchie einnehmen. Man kann es wirklich gut mit einem Verzeichnisbaum vergleichen.

[3]EnvRegionen bilden eine Ausnahme. Sie können keine Kinder haben, denn das würde grafisch mit der Darstellung der Breakpoints konkurrieren. Im folgenden wird die Bezeichnung „RegionMitKindern“ verwendet für eine Region, die Kinder haben *kann*, wohlgemerkt unabhängig davon, ob sie tatsächlich welche *hat*.

[4]Der deutsche Begriff „Elternteil“ wird vermieden, weil er meines Erachtens suggeriert, dass es *zwei* davon geben müsste. Für die Baumstruktur ganz wesentlich ist aber die Tatsache, dass es genau *ein* Elternteil gibt, mit Ausnahme wie gesagt der Wurzel, die keines hat.

Wenn RegionenMitKindern Verzeichnisse wären, dann wären EnvRegionen wohl Dateien. Sie können nur als Blätter im Baum auftreten. In ihnen liegen maximal noch Breakpoints, und danach ist Schluss mit Verschachtelung.

Mit den Einzelheiten über Synth- und OP-Regionen befassen sich die folgenden Abschnitte. Nur so viel sei vorweg genommen: Wie vielleicht schon erahnt, können Synth-Regionen Audiosignale erzeugen und verarbeiten. Group- und OP-Regionen können Audiosignale nur bündeln und weiterleiten. Die Hauptaufgabe von OP-Regionen besteht in der nondestruktiven Veränderung von Partiturdaten.

2.3.2 Elemente der grafischen Benutzerschnittstelle

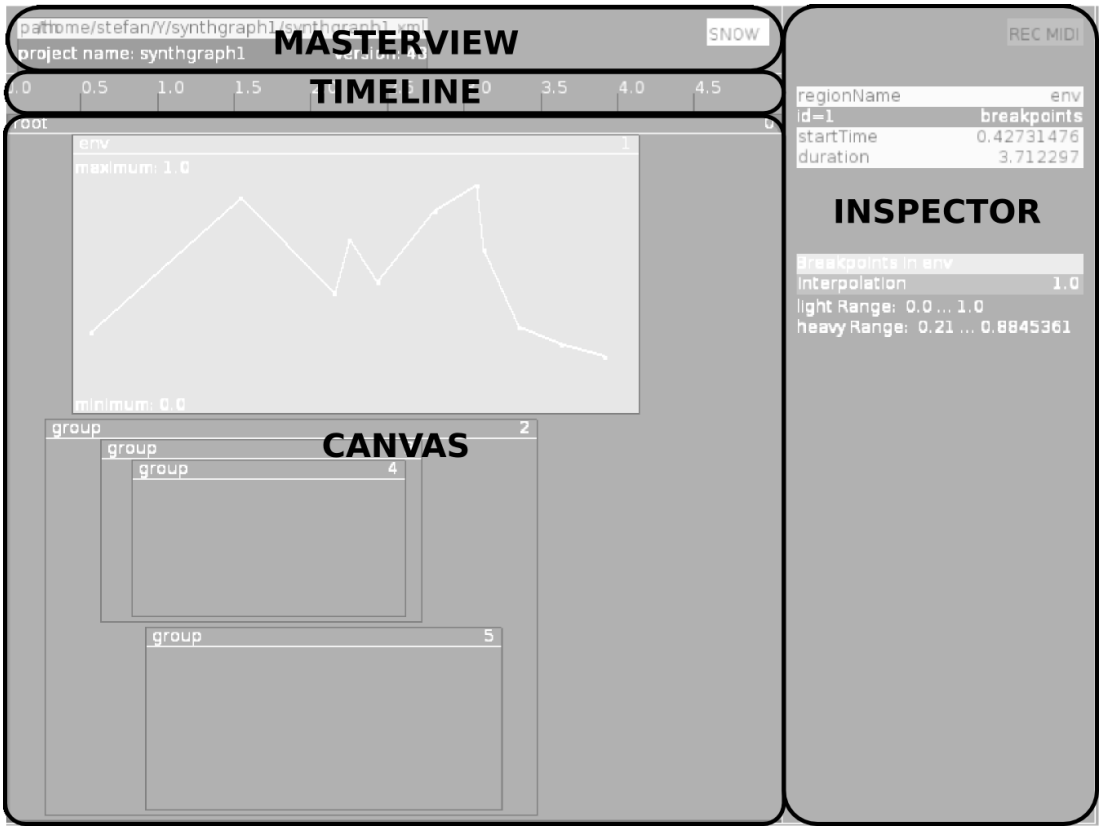

Abbildung 2.4: Aufteilung des GUI-Fensters

MasterView

Im linken oberen Teil des Fensters befindet sich der MasterView. Hier wird Funktionalität untergebracht, die unabhängig davon ist, ob gerade eine Session geladen ist oder nicht. Sinnvollerweise befindet sich hier die Möglichkeit, eine Session per Pfadangabe zu laden, und der „Snow“-Knopf zur Entspannung.

Timeline

Die rote Zeitleiste markiert die globale Zeit in Sekunden. Auf sie kann wie üblich mit der Maus geklickt werden, um die Positionsmarke zu setzen. Durch Ziehen mit der Maus wird eine „Range“ definiert.

Canvas

Als Canvas (Leinwand) wird der große rechteckige Bereich bezeichnet, auf dem die „RegionViews“ liegen, welche die tatsächlichen Regionen für die GUI repräsentieren[5]. RegionViews stellen außer der visuellen Darstellung auch Methoden zum verschieben, verlängern, etc. bereit. Sie bestehen aus einer Kopfzeile, die Name, ID und Fade-In/Out-Kurven der Region anzeigt, und aus einem darunter liegenden Rumpf, wo Kinder bzw. Breakpoints visualisiert werden.

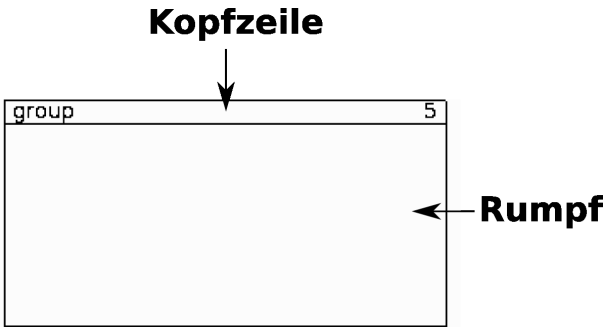

Abbildung 2.5: Ein RegionView mit Name und ID

[5]Im Gegensatz dazu wird beim Scripting direkt auf die Regionen zugegriffen.

Abbildung 2.6: In der Kopfzeile werden Fades als Kurven angezeigt

Außerdem bietet die Kopfzeile Angriffsfläche für Mausoperationen wie Ziehen, Verlängern, usw.. Mit Rechtsklick öffnet sich ein Drop-Menü mit möglichen Befehlen, die die Region betreffen. (siehe Tabelle im Anhang)

Der Canvas bietet die übliche Funktionalität zum horizontalen *scrollen* und *zoomen*. Darüber hinaus kann man einen angewählten RegionView auf die ganze Höhe und Breite des Canvas vergrößern. Man taucht sozusagen in die Äste und Zweige des Baumes und kann so in beliebigen Tiefen präzise sehen und arbeiten.

Inspector

Der Inspector im rechten Fensterbereich zeigt alle wissenswerten Daten über die jeweils im Canvas hervorgehobene Region an. Für jede Region sind Name, Startzeit, Dauer variabel. Die ID einer Region kann nicht verändert werden, sie wird automatisch vom System zugewiesen.

Ist die inspizierte Region eine SynthRegion, wird hier im Inspector ihr Synth mit all seinen Parametern angezeigt[6]. Dementsprechend erscheint bei einer OPRegion ihr „OP“ samt Parameter. Bei einer EnvRegion gibt es einen variablen Parameter für die Interpolation und eine Anzeige über Minimal- und Maximalwerte.

Zusätzlich bietet der Inspector Fade-Daten und AudioSends zur Bearbeitung an.

[6]Eine frühere Version des Prototyps bot die Sonderfunktion, dass die Synths einzeln verschachtelter SynthRegionen im Inspector automatisch als „Kanalzug“ untereinander dargestellt werden. Diese GUI-Abkürzung schien mir ursprünglich wichtig für die Ergonomie, hat sich aber im praktischen Umgang mit Y als ziemlich irrelevant herausgestellt. Eine GUI ohne Ausnahmen ist mehr wert als eine Ausnahme, die nicht unbedingt nötig ist.

Ganz oben rechts befindet sich der Record-Knopf. Er ist rot, wenn die Region Audio führt, und gelb, wenn es eine EnvRegion ist. Dementsprechend aktiviert er jeweils die Aufnahme von Audiosignalen bzw. MIDI-Controllern.

2.3.3 Klangsynthese

SynthRegionen

Jede SynthRegion besitzt einen Synth. Es wird kein struktureller Unterschied zwischen Tonerzeugern und Effekten gemacht, da auch Tonerzeuger Eingangsparameter in Form von Audiosignalen haben können, vergleichbar mit Inlets bei Max/PD. Welche Parameter ein Synth hat, und welche Datentypen diese Parameter als Eingang haben, ist in der Synth-Datenbank festgelegt. Implementiert sind die folgenden Parameter-Typen:

- String (Zeichenkette)
- Float (reelle Zahl)
- Audio (Ausgangssignal einer anderen Region)

String-Parameter sind unter anderem als Pfadangabe wichtig, wenn ein Synth z.B. eine Datei von Festplatte abspielt. Ein Float-Parameter ist ein konstanter Zahlenwert. Audio-Parameter können ebenfalls konstante Werte annehmen, oder eben auch Audiosignale. Dazu gibt man ins Parameterfeld statt einer Zahl eine Referenz auf eine andere Region ein, z.B. „r53“, damit wird der Ausgang der Region mit der ID 53 in diesen Audio-Parameter geroutet. Auf diese Weise kann man Synths in Reihe schalten und Insert-Effekte realisieren.

Busse und Sends

Zusätzlich dazu haben alle Regionen außer EnvRegionen[7] einen „Bus“, auf den andere Regionen per „Send“ ihr Ausgangssignal schicken können. So können mehrere Signale auf einen Bus gemischt werden, ähnlich wie am Mischpult. Auf diese Weise können auch Send-Effekte und überhaupt alle Mischpult-typischen Signalwege realisiert werden. Einfaches Beispiel, Hall: mehrere Regionen senden auf eine Region namens „schöner Hall“. Diese ist eine SynthRegion mit einem Synth, der das Signal vom Mixbus der Region verhallt und das Resultat an den Ausgang der Region legt. Von da aus kann es weiter gesendet werden, z.B. an Root. Root gibt ihren Mixbus auf den Hardwareausgang.

Spezial-IDs für Parent und Root

Man erreicht die jeweilige Root-Region per Send entweder über ihre ID oder über die Spezial-ID „-2“. Daneben gibt es die Spezial-ID „-1“, was den jeweiligen Parent einer Region bezeichnet. Letzteres ist die Voreinstellung beim Erzeugen neuer Synth-, Group- oder OPRegionen. Dadurch entsteht eine lockere Korrelation zwischen Baumstruktur und Signalfluss, die aber nicht verbindlich ist. Die Idee ist, dass Signale grundsätzlich vom Blatt Richtung Wurzel fließen, wenn man es nicht anders festlegt. Wenn man also 200 Grains als Kinder der Region „Haufen“ hat, spielen die Grains zuerst in ihre Parent-Region „Haufen“. Von dort aus kann man sie gesammelt bearbeiten, in dem man der Region „Haufen“ ein neues Parent namens „Schaufel“ hinzufügt, welches dann z.B. eine SynthRegion mit Filter oder eine OPRegion mit Zeitquantisierung sein kann.

[7] Das ist wieder eine andere Ausnahme, diesmal die Signalverarbeitung betreffend. Es hat nichts damit zu tun, dass EnvRegionen keine Kinder haben können.

EnvRegionen

EnvRegionen werden wieder einmal speziell behandelt. Sie haben keinen Mixbus, man kann nicht auf sie routen. Stattdessen ergibt sich das Ausgangssignal aus der Interpolation zwischen den Breakpoints. Verschiedene Interpolationskurven können stufenlos über einen numerischen Parameter eingestellt werden: treppenförmig, linear oder parabolisch.

Sie sind als Signalquelle für Audio-Parameter gedacht und bieten so ein hochauflösendes Mittel zur dynamischen Steuerung von Signalprozessoren. Sie ersetzen die im Sequenzer übliche Automation und MIDI-Controller.

Eine EnvRegion hat keine Audio-Sends, das heißt, man kann keine von EnvRegionen erzeugten Signale auf die Mixbusse anderer Regionen senden[8].

Audiodateien

Y kann große Samples direkt von der Festplatte „streamen“, wie es bei Audiosequenzern üblich ist. Das wird bei Bedarf über einen Synth erledigt, der als Parameter Dateipfad und Startposition bekommt.

Interessanter für die Klangsynthese ist jedoch das Abspielen aus dem Arbeitsspeicher, weil damit eine beliebige Modulation des Abspielzeigers möglich ist: Transposition, Waveshaping, Scratching, etc.. Das passiert über Buffer in SuperCollider. Jede Session hat, genau wie bei Tracker-Programmen, eine globale Tabelle mit Audiosamples[9]. Bei jedem Öffnen der Session werden die Samples geladen, falls sie nicht bereits im Speicher liegen. Synths können diese Samples sehr flexibel abspielen.

[8] Damit soll die Möglichkeit eingedämmt werden, dass (ultra)tieffrequente Steuersignale versehentlich in einen Teil des Signalwegs gelangen, in dem sie nichts verloren haben: nämlich den für hörbare Frequenzen. Und hörbare Frequenzen per Breakpoint-Hüllkurve zu erzeugen, dürfte ein wirklich seltener Spezialfall sein. Wer das trotzdem einmal brauchen sollte - vielleicht für Impulse - kann sich einfach einen Synth definieren, der das Signal von einem Audio-Parameter-Eingang auf den Ausgang legt.

[9] Im Gegensatz zum Tracker werden die Sampledaten nicht in der Session-Datei mit gespeichert. Die XML-Datei von Y enthält nur Referenzen auf die Dateien (Pfadangaben)

Wellenformdarstellung

Der Nachteil beim flexiblen Abspielen von Samples ist, dass die vom Audiosequenzer gewohnte Wellenformdarstellung oder das Spektrogramm nicht so einfach möglich ist. Wenn der dynamische Verlauf der Abspielposition von einem Audiosignal gesteuert wird, welches wiederum aus einem Synthesizer kommt, usw., ist auch der Verlauf der resultierenden Wellenform nicht vorhersehbar. Erst beim tatsächlichen Abspielen kann man den Verlauf des Signals messen und darstellen. Wie das in Y geht, ist weiter unten im Abschnitt „Klanganalyse“ beschrieben.

2.3.4 Klangeingabe

Mit Record und Play wird wie im Sequenzer eine Audioaufnahme synchron zur Partitur gestartet, und mit Stop beendet. Daraufhin liegt ein neuer Buffer mit dem aufgenommenen Material im Arbeitsspeicher, und als Kind der im Canvas markierten Region liegt eine neue SynthRegion, die auf das Abspielen dieses Buffers eingestellt ist.

Im Prototyp kann man nur *eine* Spur aufnehmen, vom ersten Audioeingang. Das ist auf jeden Fall übersichtlich. Nichts ist ärgerlicher, als wenn man den Take seines Lebens eingespielt, auch die Spur scharf geschaltet aber den falschen Eingang gewählt hat. Das passiert in Audiosequenzern gerne ab und zu, weil diese Einstellungen in der GUI oft weit auseinander liegen und nicht immer gleichzeitig sichtbar sind. Zum Beispiel befindet sich das Eingangsrouting im Mixerfenster, das gerade versteckt ist. Je öfter die Signalquelle gewechselt wird, desto größer die Gefahr. Viel sinnvoller fände ich es, diese Funktion direkt am Record-Knopf anzubringen, oder gleich mehrere Record-Knöpfe nebeneinander zu haben, für jeden benutzten Eingang einen. Und direkt an jedem Knopf eine kleine Pegel-Anzeige, so dass man vor jeder Aufnahme sicher gehen kann, dass dort auch etwas rein kommt, wo man scharf schaltet. Die „richtige“ Version von Y wird dafür eine angemessene Lösung bieten. Prinzipiell ist Mehrkanalaufnahme überhaupt kein Problem, und da das Aufnehmen in den Arbeitsspeicher kaum Performance kostet, wäre es auch denkbar, grundsätzlich alle Kanäle aufzunehmen, und erst nach der Aufnahme die eindeutig nicht benutzten automatisch zu entfernen. Das würde zumindest die Gefahr von Kanalverwechslungen ausräumen.

Zum Laden und Verwalten von Samples in Buffern hat der Prototyp keinen speziellen GUI-Mechanismus eingebaut. Es geht aber sehr einfach in der XML-Datei oder per Scripting.

2.3.5 Klanganalyse

Über das Canvas-Kontextmenü (Rechtsklick auf RegionView) und das Scripting-Interface stehen verschiedene Analysemethoden zur Auswahl: Pegelmessung, Tonhöhenerkennung, Finden von Anschlagsphasen, Messungen über spektrale Eigenschaften wie Höhenlastigkeit, etc..

Dazu wird das Ausgangssignal der Region temporär in einen privaten Buffer in SuperCollider oder in eine Audiodatei exportiert. Von dort aus wird die Analyse durchgeführt und anschließend die Messergebnisse als EnvRegion(en) in die Partitur eingefügt. Da die EnvRegionen als Kinder der analysierten Region und synchron zu dieser eingefügt werden, stehen die Analysedaten direkt im Kontext mit dem Audiomaterial und sind strukturell an dieses gebunden. Zum Beispiel werden sie als reguläre Kinder beim Verschieben oder Entfernen der Region normalerweise mitgenommen. Sie können aber auch in der Baumhierarchie nach oben bewegt somit unabhängig von der ursprünglichen Region behandelt werden. Auf jeden Fall können die Analysedaten sofort zur weiteren Synthese oder Montage in der Partitur verwendet werden.

2.3.6 Klangmontage

Realtime vs. Non-Realtime

Über die eingebauten oder benutzerdefinierte Synths lassen sich beliebige Echtzeit-Klangbearbeitungen realisieren. Wie beim Audiosequenzer mit Plugins lassen sich Effekte zu Ketten verbinden. Es sind aber auch kompliziertere Verschaltungen möglich, wie man es von Max/PD gewohnt ist. „Echtzeit“ heißt, das Material wird direkt beim Abspielen und synchron dazu bearbeitet, was unter anderem den Vorteil bringt, dass man in sehr kurzen Zyklen hören und bearbeiten kann.

Ein Nachteil ist jedoch, dass Synths zwar die Vergangenheit kennen[10] aber nicht in die Zukunft blicken können. Um ein Signal beispielsweise zu beschleunigen, muss man es aus dem Echtzeitkontext heraus heben. Mit Buffern ist das in Y möglich. Dazu wird eine Region in eine Datei ausgespielt, die dann sofort als Buffer geladen und als SynthRegion synchron zum Original eingefügt wird.[11]

Schnitt

Ob man „Schnitt“ zur Klangmontage oder zur Partiturmontage zählt, hängt für meine Begriffe davon ab, ob die Ergebnisse des Schnitts in der Partitur festgeschrieben sind oder nicht. Schneidet man etwa im Audiosequenzer das mittlere Drittel aus einem Sample heraus, bedeutet das eine Änderung der Partiturparameter. Die Positionen des Schnitts sind in der Partitur sichtbar und variabel. Tut man das gleiche im Sample-Editor, ist der Vorgang zwar reversibel (Undo), aber nicht variabel. Der Schnitt ist in der Partitur nicht sichtbar, sondern man hat etwas am Klang montiert.

Mischung

Mischung fällt dagegen für meine Begriffe klar in den Bereich der Klangmontage, als Teilmenge von Echtzeitbearbeitung. Die dazu nötigen Synths (Verstärker, Equalizer, Hall, ...) sind in der Implementierung trivial, und die erforderliche Infrastruktur (Sends, Busse) ist ebenfalls vorhanden.

[10] mit Hilfe von Delays, also temporären Speichern

[11] Bouncen inklusive Reimportieren ist in Y genau ein einziger Mausklick. Mit dieser ergonomischen Brücke wird bezweckt, dass die traditionell als distinkt wahrgenommenen Konzepte von *flüchtigem* und *konserviertem* Signal (Echtzeit und Nicht-Echtzeit) einander angenähert werden. Genau so schnell kann eine Region in einem externen Sampleeditor zur Bearbeitung geöffnet werden.

2.3.7 Partitursynthese

Klassische Partitursynthese heißt: Imperativ Regionen und Breakpoints erstellen. Das geht in Y, während die GUI läuft. Die bestehenden Regionen werden über ihre ID angesprochen und um programmierte Kinder oder Parents erweitert. Diese sind sofort automatisch in der GUI sichtbar, und man kann sie direkt manuell weiter bearbeiten oder mit *Undo* wieder entfernen, falls man sich verprogrammiert hat.

Das Scripting-Interface greift dabei direkt auf die Java-Klasse „Session“ zu, in der die Partitur repräsentiert ist. Über diese Klasse hat es Zugang zu den Klassen der Regionen und Breakpoints. Somit stehen alle Java-Funktionen, die die GUI benutzt, auch für das Scripting zur Verfügung. Hier in die technischen Details zu gehen, würde zu weit führen, statt dessen wollen wir es bei zwei einfachen Code-Beispielen belassen.

```
Y.getRegion(104).newChildSynth("‘Krach"’, 10, 5, "‘WhiteNoise"’)
```

Die obige Zeile fügt der Region mit der ID 104 eine new SynthRegion als Kind hinzu. Dieses trägt den Namen „Krach“, beginnt bei Sekunde 10, dauert 5 Sekunden und erzeugt weißes Rauschen mit dem Synth „WhiteNoise“.

```
a = Y.getRegion(104);
b = a.newChildEnv("‘Treppe"’, 0);
10.do({arg i; b.newBreakpoint(i*2, i/10)});
```

Diese Zeile erzeugt eine neue EnvRegion namens „Treppe“ als Kind von Region 104 mit gleichem Startpunkt und Dauer und ohne Interpolation (0). Dort hinein kommen 10 Breakpoints im Abstand von 2 Sekunden und mit Werten von 0 bis 0.9.

2.3.8 Partitureingabe

Die meisten Werte sind in der GUI als Zahlen dargestellt und können dort auch direkt als solche eingegeben werden.

Breakpoints kann man manuell eingeben oder in Echtzeit per MIDI-Controller aufnehmen. Ihre Werte liegen immer zwischen 0 und 1. Um den Wertebereich zu skalieren, verwendet man OPRegions (siehe Abschnitt über Partiturmontage). Breakpoints werden in der GUI per Maus oder als Zahlen in der XML-Datei eingegeben.

2.3.9 Partituranalyse

Das Scripting-Interface hat vollen Zugriff auf alle Partiturdaten. Also können diese zum Erkenntnisgewinn, zum Export oder als Eingabe für Partitursynthese (Metapartitur) ausgelesen werden.

Eine Verwendung sind z.B. Time-Tracks. Das sind EnvRegions, deren Breakpoints Zeitpunkte markieren. Das können formale Abschnitte in der Partitur sein, oder auch metrische Wertigkeiten. Bei normalen Sequenzern wird ersteres bei Bedarf mit Marken gelöst[12] und für letzteres dient die Zeitstrukturierung in Takten[13]. So kann man beliebige Zeitnetze oder Rhythmen definieren, die dann in Partitursynthese oder Nicht-Echtzeit-Klangbearbeitung berücksichtigt werden. Prinzipiell auch in Klangmontage, das möchte ich später gerne einbauen. Regionen und Breakpoints könnten beim manuellen Verschieben auf diese Zeitraster eingerastet werden, ein weit verbreitetes und nützliches Feature.

[12] die aber keinen weiteren Einfluss auf die Partitur haben
[13] die sehr wohl Einfluss auf die Partitur haben

2.3.10 Partiturmontage

„Schnitt“ wird zurecht oft mit „schneiden“ assoziiert. Ausgerechnet das kann Y nicht[14]. Auch nicht kleben. Fade-In und Fade-Out schon.

Was allerdings recht flott implementiert ist, ist das Verlängern und Verkürzen von Regionen, sowie das Erstellen, Verschieben und Entfernen von Regionen und Breakpoints in der GUI. Dazu kommt das Konzept der Op-Regionen. Sie beeinflussen die Werte ihrer Kinder[15] innerhalb ihrer zeitlichen Ausdehnung nach festgelegten Parametern. Ein sehr mächtiges Werkzeug zur Partiturmontage.

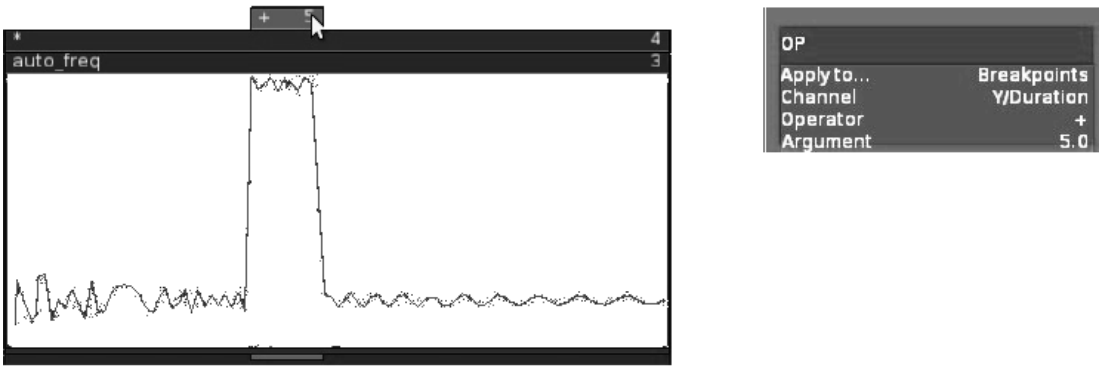

Abbildung 2.7: OPRegionen wirken innerhalb ihrer Zeitgrenzen auf Kinder

Die Abbildung zeigt, wie Breakpoints von einer darüber liegenden OPRegion per Addition verschoben werden. Derzeit gibt es die Operatoren plus, mal, hoch, maximum, minimum, log, abs und quantize. Sie können speziell auf Regionen oder Breakpoints oder beide angewendet werden. Außerdem betreffen sie entweder die Zeit oder die Dauer (bei Regionen) oder den Wert (bei Breakpoints).

[14]Während das bei Env-, Group- und OPRegions sehr einfach umzusetzen wäre, ist das erwartete Verhalten bei SynthRegionen abhängig von der Funktion des jeweiligen Synths. Hier möchte ich keine voreiligen Entscheidungen treffen und lieber die Entwicklung zukünftiger Synths abwarten. Bis jetzt habe ich die Funktion auch noch gar nicht vermisst, was mich selber überrascht.

[15]genauer gesagt *Nachfahren*, OPRegionen wirken auf alle inneren Generationen

2.4 Zur Ästhetik der GUI

2.4.1 Raumaufteilung

Die meisten Editoren, die ich kenne, haben den Inspector links vom Canvas. Rechts ist mir lieber, das schließt das Bild in Leserichtung mit etwas Sinnvollem ab. Sonst rollt der Cursor so sehr in Richtung „nichts“. Im Prototyp von Y bewegt sich der Cursor zur Kontrolleinheit hin, anstatt von ihr weg.

Zeitleiste und Masterview hatte ich zuerst auch unten, aus reiner Protesthaltung, aber oben hat sich als handlicher erwiesen.

2.4.2 Farbgebung

Farben sind wichtig für die Orientierung im Canvas. die Root-Region ist Blau, die zweite Generation rot, usw., die Farbtöne wiederholen sich erst nach 10 Ebenen. So weiß man immer, welche Regionen im Baum auf einer Höhe liegen, was nicht ganz unpraktisch ist[16]. Die Helligkeit einer Region zeigt an, ob diese angewählt ist oder verschoben wird. In letzterem Fall wird durch Helligkeit markiert, ob man gerade die ganze Region samt Nachkommen verschiebt oder nur Start- oder Endzeitpunkt. Auch im Inspector sind zusammengehörige Elemente farblich gruppiert, dieses allerdings ziemlich wahllos und ohne bedeutendes Konzept, was ich als angenehm empfinde.

Insgesamt ist die GUI deutlich bunter als andere. Die großen einfarbigen Flächen haben natürlich atmosphärische Wirkung. Auch wer immun gegen C64-Nostalgie ist, spürt beim Benutzen deutlich, dass er/sie sich nicht in einem Tabellenkalkulationsprogramm befindet.

[16] In späteren Versionen findet sich eine Lösung, die auch für Menschen mit beeinträchtigtem Farbsehvermögen geeignet ist.

2.4.3 Formen und Texturen

Das Rechteck dominiert als häufigste geometrische Struktur. Die nächste grafische Benutzerschnittstelle werde ich im Hundertwasser-Stil programmieren, aber für Y sind Rechtecke OK. Hauptsache keine „schicken“ Holz- oder Chrom-Texturen mit Lichtreflexen mehr. Von allen ästhetischen Determinationen ist mir das die unsympathischste, und es tut wirklich gut, so etwas nicht dauernd sehen zu müssen. Auch keine imitierten Potentiometer oder Lämpchen.

2.4.4 Animationen

Jeder blinkende Cursor ist ein Metronom. Jedes Metronom stört grundsätzlich[17]. Ich habe bewusst darauf geachtet, dass die GUI keine Zeitkomponente beinhaltet. Auch eine einfache Blende hat schon eine gewisse Härte oder Weichheit, die nicht mit jeder ästhetischen Grundhaltung harmoniert oder auf produktive Weise disharmoniert. Nur in einem Punkt war die Versuchung stärker: Das Dropmenü wirft einen Schatten, der etwa eine Sekunde braucht, um einzublenden. Dieser Anblick löst bei mir jedes Mal 0.041 Nanogramm Glückshormone aus, weshalb es medizinisch unverantwortlich wäre, ihn zu entfernen.

2.4.5 Schnee

Der Schnee-Knopf erzeugt Schnee.

[17] Selbst wenn es zufällig im richtigen Tempo sein sollte, ist es mit größter Wahrscheinlichkeit phasenverschoben.

2.5 Projektverlauf und Geschichte

Y hat beileibe nicht so begonnen, wie es heute aussieht - und wird wahrscheinlich in einigen Monaten wieder ganz anders aussehen.

2.5.1 Frühere Ansätze

Erste Denkanstöße gaben mir die quälend unergonomischen Import-/Export-Vorgänge bei der experimentellen Arbeit mit Logic und CSound. Recherchen in Richtung Nuendo haben die Lage auch nicht verbessert, also habe ich Logic verkauft und mich Ardour zugewandt. Wenn mir ein Sequenzer schon auf die Nerven geht, soll er das wenigstens kostenlos tun. Zu der Zeit habe ich viel mit Guile (Scheme) programmiert, und so verquirlte sich der Gedanke an den offenen Quellcode von Ardour mit dem an die einfache Einbindung von Guile in C-Programme. Also war mein erster praktischer Ansatz, einen Guile-Interpreter in Ardour zu integrieren. Die Grundidee war, Import und Export von Partitur-Fragmenten zu skripten, für schnelle Klangbearbeitungen mit CSound. Bevor dieser erste Ansatz richtig Gelegenheit zum Scheitern hatte, kamen sich schon der zweite und dritte Ansatz gegenseitig dazwischen: Der zweite war eine eigene, minimale Skriptsprache, der dritte ein OSC-Interface für Ardour. Beide verunglückten an der relativ konsequenten Programmierung von Ardour, welche, wie sich nach wochenlanger Verzweiflungstätigkeit herausstellte, kurz gesagt wirklich nicht dafür ausgelegt ist, dass irgendwelche halbstarken Interfaces der GUI ihre Alleinherrschaft streitig machen. Gleichzeitig mit dieser Einsicht kamen immer mehr Bedenken an dem Sequenzer-Konzept an sich. Noch nicht so sehr das mit den Spuren, vor allem fehlten mir die Abstraktionen und Datentypen für den zunehmenden Wunsch nach Analysedaten und Metapartitur in der Partitur. Dann kam die Erwägung, einen eigenen Editor oder zumindest einen grafischen Viewer zu schreiben, der parallel zu Ardour läuft und all das darstellt, was Ardour nicht darstellen kann.

Ardour wäre dabei zunehmend zum LADSPA-Mischpult degradiert worden. Zum Glück kam es nicht so weit, denn der nächste Entwurf war bereits die Idee mit den verschachtelten Regionen. Ab dem Moment war klar, dass dafür auch eine spezielle Synthese-Engine nötig war, und die Optionen grenzten sich auf die hier beschriebene Lösung ein.

2.5.2 Ausblicke

In den nächsten Monaten wird die Arbeit mit dem Prototyp hoffentlich weitere Stärken und Schwächen aufzeigen, die die technische und konzeptuelle Weiterentwicklung betreffen. Einige Aspekte habe ich bereits im Auge. Unter anderem:

Datenstrukturen

Das Eingrenzen der Partiturobjekte in Regionen und Breakpoints fiel sehr schwer, da diese Entscheidungen das ganze Konzept der Software und somit indirekt kompositorische Denkmuster determinieren. Besonders der Umstand, dass Breakpoints nur *einen* Wert über der Zeit haben. Stets lockte die Versuchung, diesen Datentyp um beliebige Dimensionen zu erweitern. Viele Anwendungen rufen geradezu danach, darunter Analyseverfahren wie LPC, Tonhöhenerkennung, Filterbankanalyse, die zu bestimmten Zeitpunkten einen zusammenhängenden Satz von Zahlen ausgeben. Auch MIDI-Noten ließen sich vernünftig in Breakpoints abbilden, wenn diese mehr als nur einen Wert zu einem Zeitpunkt speichern könnten. Für tonale Musik mit Partitursynthese wäre es hilfreich, im Sinne einer Metapartitur Melodik, Harmonik und Rhythmik ausdrücken zu können, auch hier wären komplexere Datenstrukturen von Nöten. Nun ist es so, dass jeder Datensatz im Computer erst durch seine Interpretation Sinn erhält. Wenn Daten ausgetauscht werden, muss deren Interpretation standardisiert sein. MIDI-Noten sind erst dadurch transportabel, dass sie von den immer gleich funktionierenden

Synthesizern abgespielt werden. Bei Y besteht die Konvention darin, dass Breakpoints zu Audiosignalen gewandelt werden, welche ihrerseits das Standardformat darstellen, das zwischen Synths und Regionen ausgetauscht wird. Für andere Datenstrukturen[18] gibt es keine Konvention, also müssen diese Daten bei Bedarf fest an interpretierende Mechanismen gebunden werden. Möchte man etwa einen speziellen Datentyp „Streichernote“ einsetzen, mit den Parametern Tonhöhe, Dauer, Bogenposition und Lautstärkenverlauf, definiert man einen Synth, der genau diese Parameter erhält und entsprechend verarbeitet. Jede *Instanz* dieses Datentyps, also jede Streichernote, ist dann eine SynthRegion mit dem entsprechenden Synth und den Parametern. Die Lautstärkehüllkurve realisiert man über eine EnvRegion als Kind der SynthRegion, damit sind die Daten sinnvoll aneinander gebunden. Auch Akkordsymbole oder sonstige Textdaten kann man als String-Parameter eines speziell definierten Synths[19] angeben. Partituranalyse-Skripts können diese Angaben dann weiter verarbeiten. Eine Grenze wird erreicht bei Datenstrukturen, deren Parameter nicht in Anzahl und Typen fest definiert sind, oder bei nicht-primitiven Parametern. Im Extremfall möchte man vielleicht beliebigen SuperCollider-Code direkt in die Partitur schreiben, dafür gibt es momentan noch keinen ausgedachten Mechanismus.

GUI-Ergonomie

Das Manipulieren von Regionen in der GUI ist rudimentär implementiert. Man kann Regionen samt Kindern mit der Maus verschieben, oder Start- und Endzeitpunkte unabhängig von den Kindern per Maus oder Texteingabe verändern. Möglicherweise wächst der Wunsch, detailliertere Zusammenhänge ausdrücken zu können. Wie zum Beispiel: „Das Ende dieser Region soll immer mit Ende ihres Parents synchron sein, ihre Länge soll gleich bleiben.“ Oder „Die zeitliche Ausdehnung dieser Region passt sich automatisch der der Kinder an.“ Oder: „Die

[18] z.B. Akkorde, FFT-Frames, Taktmetren, Phoneme ...
[19] dessen Ausgangssignal ja Null sein kann

Geschwister knüpfen ohne Überlappung zeitlich aneinander an (oder mit einer Überlappung von ...).“ Wenn sich herausstellt, dass so etwas häufig gebraucht wird, kämen verschiedene Ansätze für die GUI in Frage, oder auch skriptbasierte Lösungen im Bereich Partituranalyse/-synthese.

Es hat sich für mich bis jetzt als überraschend ergonomisch herausgestellt, ohne Dateidialogfenster zu arbeiten. Beim Aufnehmen oder Exportieren aus Y werden automatisch nummerierte Dateien mit Präfix und Versionsnummer der Session im Projektverzeichnis erstellt, z.B. „exp000012_815.wav“. Auf diese Weise ist sichergestellt, dass keine Namen doppelt auftreten, keine Dateien überschrieben oder an falsche Stellen gespeichert werden. Darüber hinaus ist die neueste Datei immer an hervorgehobener Stelle in der Desktop-Umgebung sichtbar[20], und ältere Dateien können über die nachgestellte Session-Nummer auf ihre Entstehung zurückgeführt werden. Regionen können durch Exportieren mit einem Mausklick zwischengesichert werden, ohne die geringste Ablenkung durch Dialogfenster. Das Importieren von Audiodateien aus der Desktop-Umgebung[21] in die XML-Datei geht per *Copy und Paste* so schnell, dass mir kaum eine elegantere Lösung einfällt.

Allenfalls Texteingabefelder in der GUI von Y sollten auf jeden Fall um *Copy und Paste*-Fähigkeit erweitert werden, um bequem einzelne Werte zwischen Texteditor und GUI austauschen zu können.

[20] nach Namen oder Datum sortierte Listendarstellung im Verzeichnisfenster vorausgesetzt.
[21] zumindest bei „GNOME“ und „gedit“ unter Linux