Kapitel 1

Motivation

1.1 Grundlagen von Mensch-Computer-Interaktion

Alan Kay hat Recht mit seiner These, dass die 1960 von J.C.R. Licklider[3] beschriebene „Symbiose“ von Mensch und Computer bis heute nicht Realität geworden ist[7]. Auch von einer partnerschaftlichen Kooperation zu sprechen, wäre noch weit übertrieben. Es ist kein faires Verhältnis, wenn einer von beiden sich ständig bedienen lässt. Den allergrößten Teil seiner Lebensspanne verbringt ein durchschnittlicher Personalcomputer nicht mit dem Lösen von Problemen, sondern mit dem Warten auf Dateneingabe vom Menschen, welcher seinerseits übermäßig viel Zeit damit verbringt, seine Probleme in einer für den Rechner verständlichen Form zu artikulieren.

> Wir Endbenutzer werden quälend langsam in Programmen und Maschinen heimisch, die ihrerseits kein Wort verstehen, nur Kommandozeilen, Mausklicks und seit neuestem Befehle per Mikrofon. [...] Wir „kommunizieren“ gar nicht mit Programmen, sondern fügen uns in ihre Dummheit.[4, S. 88]

So gesehen ist der Mensch, obwohl er über weit größere Intelligenz verfügt, in puncto Zusammenarbeit mit Denkmaschinen im Allgemeinen der Dumme, da er sich von den Interaktionsbarrieren in seinem Arbeitsfluss ausbremsen lässt.

Wenn man also übereinkommt, dass heutzutage Menschen und Computer eben nicht in der Lage sind, miteinander zu einem nahtlos funktionierenden, produktiven Hybridorganismus zu verschmelzen, wie lässt sich das Arbeitsverhältnis realistischerweise beschreiben? Sind Computer reine Werkzeuge? Dienen sie als Erweiterungen des menschlichen Gehirns, so wie Hammer oder Teleskope den Arm oder das Auge erweitern? Liefern sie Ideen? Welche Rolle nimmt der Benutzer als Bestandteil eines interaktiven Systems ein? Wer ist zuständig für welche Aufgaben?

Licklider beschreibt in seiner Utopie die Aufgaben des Menschen innerhalb eines gut funktionierenden Mensch-Computer-Systems folgendermaßen:

> Men will set the goals and supply the motivations [...] They will formulate hypotheses. They will ask questions. They will think of mechanisms, procedurues, and models. [...] They will make approximate and fallible, but leading, contributions, and they will define criteria and serve as evaluators, judging the contributions of the equipment and guiding the general line of thought. [...] Men will fill in the gaps, either in the problem solution or in the computer program, when the computer has no mode or routine that is applicable in a particular circumstance.[3]

Über Computer schreibt er:

> The information-processing equipment [...] will convert hypotheses into testable models and then test the models against data (which the human operator may designate roughly and identify as relevant when

> the computer presents them for his approval). The equipment will answer questions. [...] In general, it will carry out the routinizable, clerical operations that fill the intervals between decisions.[3]

Dabei sieht er für unsere Zeit eine Computerwelt voraus, die stark von der Orientierung an künstlicher Intelligenz geprägt ist. Der Computer, der sich am Verstehen, Erklären, Planen, Vereinfachen, Bewerten und Modellieren beteiligt, ist nach wie vor Zukunftsmusik, weil sich der Softwaremarkt in den letzten Jahrzehnten anders entwickelt hat. Als letzten revolutionären Durchbruch auf dem Gebiet der Verständigung zwischen Mensch und Computer kann man die Erfindung der grafischen Benutzerschnittstelle (GUI) um 1980 bei Xerox PARC betrachten. Seitdem wurden Details verfeinert, aber im Wesentlichen bieten uns Programme heute entweder textbasierte oder mauszeigerbasierte Schnittstellen an. Diese beiden Konzepte reichten offenbar aus, um den allergrößten Teil der von Nutzern dringend gewünschten Anwendungen abzudecken. Alan Kay spricht von „Current Goals“[7], also kurzfristigen Zielen, die von berufstätigen Erwachsenen hauptsächlich verfolgt werden und zur Implementierung von Spezialsoftware führen. Diese erfüllt zwar ihren unmittelbaren Zweck, z.B. Bilddaten von einem Format in ein anderes zu konvertieren, der Lerneffekt für die Weiterentwicklung der Mensch-Computer-Interaktion ist allerdings gleich Null. Im Gegensatz dazu wäre zur Umsetzung von Lickliders Zielen eine Orientierung an langfristigen Verbesserungen auf Kommunikationsebene nötig, die in der Industrie zu wenig wertgeschätzt wird.

Lickliders Prophezeiung ist nicht die einzige, die ihren Zeitplan verfehlt hat. Stanley Kubricks Film „2001: A Space Odyssey“ sollte aus Protest umbenannt werden, denn 2007 ist von einem Computer, der sich mit Menschen auf Englisch unterhält, noch lange keine Spur. Zumindest die Fähigkeit, finstere Pläne zu schmieden und seine Benutzer mit Lügen und Spionage willentlich ins Verderben zu stürzen, wird kaum jemand an seinem Heimrechner vermissen. Wahrscheinlich will man

in den meisten Fällen überhaupt nicht, dass der Rechenknecht sich eine eigene Meinung zu den Daten bildet, die er verarbeiten soll. Dass er aus den gegebenen Informationen Rückschlüsse auf die Ziele des Menschen zieht, sie bewertet und mit seinen persönlichen Vorstellungen in Bezug setzt, kann je nach dem, wie der Computer charakterlich gestrickt ist, eine sehr unerwünschte Eigenschaft sein.

Im Allgemeinen wird man zumindest übereinstimmen können, dass, egal wie tiefgreifend die Mensch-Computer-Beziehung auf intellektueller Ebene auch immer sein mag, der Mensch bei Entscheidungen, die ihn betreffen, gern das letzte Wort hat. Konstruktive, intelligente Vorschläge vom Computer in passenden Momenten[1] sind gern willkommen, aber was letztendlich in die Tat umgesetzt wird, will der Benutzer selbst bestimmen können. Sonst wird er unglücklich. Und zwar unmittelbar - und wohlgemerkt unabhängig davon, welche Entscheidung nun tatsächlich die richtige gewesen wäre. Das kann ich behaupten aufgrund der Beobachtung, dass Leute sich auch dann über unerwartete Systemabstürze ärgern, wenn sie dadurch lediglich beim Zeitverschwenden unterbrochen werden.

Noch einmal Licklider:

> Men will set the goals and supply the motivations.[3]

[1] Die animierte Büroklammer bei Microsoft Word ist ein anschauliches Gegenbeispiel

1.2 Software und Kreativität

User schätzen es nicht, wenn sie von Soft- oder Hardware ganz offensichtlich am Erreichen ihrer unmittelbar fokussierten Ziele gehindert werden. Aber wie verhält es sich mit anderen Zielen, die nicht aktuell im Bewusstsein präsent sind? Kann Software die Ziele im Voraus oder während der Arbeit unterschwellig beeinflussen? Wo liegen die Toleranzgrenzen in Fällen, in denen Manipulation bewusst wird? Wie wird in der Praxis mit diesen Grenzen umgegangen?

Bevor diese sehr allgemeinen Fragestellungen auf den konkreten Fall Computermusik angewandt werden, möchte ich den kreativen Umgang mit Software vom Rest der Mensch-Computer-Interaktion abgrenzen. Die nachfolgend beschriebenen Prinzipien sind auf verschiedene mediale Bereiche übertragbar, Audio- ebenso wie Bild- und Videobearbeitung und andere. Die Einteilung nach Guilford[2] in konvergentes und divergentes Denken bei der Produktion von Ergebnissen soll hierbei als Maßstab dienen.

1.2.1 Konvergente versus divergente Produktion

Konvergente Produktion

> ist die Entwicklung logischer Schlussfolgerungen aus gegebenen Informationen. Die Betonung liegt, im Gegensatz zur Divergenten Produktion, auf dem Erreichen der einzigen bzw. besten Lösung. Die gegebene Information determiniert das Ergebnis. Ein Beispiel sind Rechenaufgaben: 3+5= ?[12]

Divergente Produktion

> beschreibt die Entwicklung logischer Alternativen aus gegebenen Informationen, wobei die Betonung, im Gegensatz zur Konvergenten Produktion, auf der Verschiedenheit, der Menge und der Bedeutung der Ergebnisse aus der gleichen Quelle liegt. Es gibt also mehr als genau eine richtige Lösung.[12]

Komposition ist ein komplexer Vorgang, der meiner Auffassung nach grundsätzlich beide Denkweisen zu verschiedenen Phasen und in verschiedenen Gewichtungen erfordert. An welcher Stelle dabei die Software zum Einsatz kommt, ist ein ganz entscheidender Faktor.

Unabhängig von ihrem Design kann die meiste Software nämlich sowohl konvergent als auch divergent eingesetzt werden. Betrachten wir zwei Extrembeispiele. Wenn ich mir in Fall A vornehme, eine Bilddatei auf 800x600 Pixel zu skalieren, sie daraufhin im Bildbearbeitungsprogramm GIMP öffne, die Skalierungsoperation durchführe, speichere und beende, dann ist das ein rein konvergenter Prozess. Ich kann aber auch in Fall B die gleiche Datei im gleichen Programm öffnen, und völlig ziellos drauf los pinseln und verformen, was ein divergenter Prozess ist. Im ersteren Fall ist die Wahl der Software egal, so lange sie die entsprechende Funktionalität bietet. Exakt das gleiche Ergebnis hätte ich ohne GUI mit dem Konsolenprogramm ImageMagick erreichen können, vielleicht ein paar Sekunden schneller oder langsamer, aber das Resultat wäre messbar identisch. In Fall B macht es einen enormen Unterschied, ob ich ein GUI- oder ein textbasiertes Programm zur Hand nehme. Und darüber hinaus, sofern ich mich für die grafische Variante entscheide, spielt es eine Rolle, ob es GIMP oder Photoshop oder Corel Draw ist. Nicht nur stehen unterschiedliche Möglichkeiten zur Manipulation zur Verfügung, sie stehen auch an unterschiedlichen Stellen und sind je nach Softwaredesign unterschiedlich leicht oder schwer zu erreichen. Im Extremfall, wenn

die Arbeit völlig ohne inhaltliche oder ästhetische Zielsetzungen beginnt, entstehen diese Ziele entweder irgendwo auf dem Weg, durch wiederholte Zyklen von Bearbeitung, Wahrnehmung und Bewertung, oder die resultierenden Inhalte samt ihren ästhetischen Eigenschaften werden von den technischen Gegebenheiten geleitet. Halten wir die These fest, um sie später zu präzisieren:

Software beeinflusst *divergente* Produktionsvorgänge.

In der Praxis sind die Grenzen oft fließend, zudem muss man einerseits differenzieren zwischen inhaltlichen Zielen, ästhetischen oder technischen[2] Zielen, andererseits zwischen formalisierten und unformalisierten Zielen. Letzteres betrifft vor allem die Ästhetik, da hier viele Komponenten mitspielen, die schwer genug mit Worten und praktisch unmöglich in Zahlen auszudrücken sind. Daher sind sie nicht immer über rein konvergente Strategien zu erreichen. Trotzdem kann der Komponist eine klare Vorstellung haben von dem, was er will. Diese Vorstellung hilft ihm bei der Auswahl divergent erzeugter Lösungsansätze.

1.2.2 Planung und Realisation

Wenn ich ein Stück auf dem Papier vollständig durchplane, beispielsweise eine mathematische Komposition mit Sinustönen, und nachdem alle Parameter festgelegt sind, setze ich mich an den Computer und realisiere das Stück exakt nach (meiner eigenen) Vorgabe mit irgendeiner geeigneten Programmiersprache, dann ist der Herstellungsprozess klar in eine divergente, kreative Phase am Papier und eine konvergente Softwarephase geteilt. Wieder besteht hier der technische Teil aus reiner Problemlösung, nicht kreativer als Datenbankprogrammierung oder das Entfernen von Knacksern aus alten Aufnahmen, und ebenso wenig anfällig gegen subtile Einflüsse durch Softwaredesign. Auch wenn diese zweite Phase technisch möglicherweise hochkomplex ist: Aus künstlerischer Sicht stellt sie einen trivia-

[2] handelt es sich z.B. um einen Softwaretest oder ein Tutorial

len Handgriff dar, bei dem Fehler zusammen mit Entscheidungen ausgeschlossen sind.

Falls allerdings während der Realisation irgendwelche zu setzenden Parameter auftauchen, die ich bei der Planung versehentlich nicht berücksichtigt habe, fängt die eindeutige Trennung zwischen Planungsphase und Realisationsphase bereits an zu bröckeln. Dann befinde ich mich nämlich schon in einem durch Software beeinflussten Kontext und soll von dort aus eine künstlerische Entscheidung treffen. Möglicherweise ist diese Entscheidung leicht zu fällen, und der zwischenzeitliche Sprung zurück in die Planungsphase ist technisch gesehen nur ein kleiner Hopser. Mit etwas Pech handelt es sich aber auch um einen Grenzfall, wo inhaltliche oder ästhetische Ziele mit technischen konkurrieren. Bin ich bereit, eine kleinere ästhetische Einbuße in Kauf zu nehmen, wenn mir das mehrere Stunden Programmierarbeit spart? Oder gehe ich zurück und muss mit der Realisation komplett von vorne anfangen? Gerade weil diese Frage mir ausgerechnet mitten in der Programmierarbeit begegnet, wo mein Kopf voller Software ist und ich auf technische Erwägungen fixiert bin - möglicherweise auch nicht mehr ganz frisch nach langer, ermüdender Problemlöserei - könnte die Versuchung schon groß sein, dem schnelleren Vorankommen den Vorzug zu geben, wenn der Unterschied an die Qualität des Endprodukts marginal ist oder im Bereich der „Geschmacksfrage“ liegt.

Nicht immer ist es *möglich*, Entscheidung im Voraus planen. Auch wenn über die Inhalte von Anfang an Klarheit besteht: Je komplexer das Projekt ist, desto häufiger treten ästhetische Zusammenhänge erst beim Hören deutlich zutage, und wichtige technische Fragen tauchen erst beim Machen auf.

Wirklich *nötig* wird Planung oft aus rein logistischen Erwägungen, wo Zeit, Geld und Personal eine Rolle spielen. Wird das Stück aber von der Idee bis zum Tonträger im Alleingang produziert, mit eigenen Mitteln und in einem lockeren Zeitrahmen, sind die logistischen Grenzen in der Regel gering. Der typische Elektroaku-

stische Komponist kann oft so frei zwischen Planung und Realisation hin- und her springen, dass eine Trennung der Begriffe kaum mehr Sinn macht. Sie verschmelzen zu einem interaktiven, kreativen Prozess. Teile einer Komposition werden einzeln realisiert, gehört, bewertet, variiert, wieder gehört, usw.. Aus einzelnen Ergebnissen entstehen neue Motivationen und Ideen, die in die Komposition einfließen. Strategien und letztendlich Ziele werden dynamisch entwickelt auf Basis des bisher Erarbeiteten[3]. In Verbindung mit einer unüberschaubar großen Methodenvielfalt kann dies einen Grad an Gestaltungsfreiheit zur Folge haben, der manchen Komponisten glatt den Boden unter den Füßen raubt.

1.2.3 Paradoxon Freiheit

Zu viel Auswahl erzeugt Paralyse und Unzufriedenheit.[5]

> Electroacoustic composers often regard externally imposed limitations as welcomed challenges around which to design their compositional strategies.[1]

Diese Feststellung ist nicht ganz neu und wurde schon an vielen anderen Stellen beobachtet. Gerade deshalb halte ich sie in diesem Zusammenhang für untersuchenswert: Wenn es intern an Strategien fehlt, helfen externe Limitationen?

Externe Limitationen können sein: Hardware, Software, Zeit, Geld, Personal, Räumlichkeiten... Interne Limitationen können sein: Vorstellungskraft, handwerkliche Fähigkeiten, körperliche Verfassung, mentale Verfassung, Motivation, Ziele, Strategien...

Dass es tatsächlich an Limitationen mangelt, wage ich zu bezweifeln. Knappheit gibt es immer und überall im Überfluss. Eher mangelt es an Aufmerksamkeit

[3] ...und die ganze Zeit über rattert die Software leise im Hintergrund und jubelt uns heimlich ihre strukturelle Denke unter.

den bereits bestehenden Limitationen gegenüber, und an einer entsprechenden Bewertung. Wenn ich mir zum Beispiel Klarheit darüber verschaffe, dass mir zur Fertigstellung von 60 Studioalben bis Weihnachten nur noch rund elfeinhalb Monate bleiben, dann kann das durchaus einen Zeitplan darstellen, der klare Strategien erzwingt, nach denen sich meine Ziele, Motivation, Verfassung, Fähigkeiten und Vorstellungskraft zu richten haben, so wahr Gedeih und Verderb davon abhängen. Sofort verlagert sich der scheinbar grenzen- und regellose Raum in ein Problemfeld, in dem es Lösungen zu finden gilt. Ist der Spielraum so eng, dass nur noch eine einzige akzeptable Lösung in Aussicht steht, kann bzw. muss diese konvergent erarbeitet werden. Wenn auf diesem Weg ein Arbeitstempo und eine Richtung initiiert werden, reicht das häufig schon als Orientierung für spätere Schritte.

Eine Gefahr bei übermäßiger Bewertung bestehender, externer Limitationen kann sein, dass man, anstatt sie effizient zur Beschleunigung zu nutzen, anfängt, gegen sie anzukämpfen. Im Einzelfall muss man abwägen. Manche externen Grenzen sind extrem hartnäckig und kosten viel Zeit und Kraft beim Dehnen, die dann bei der eigentlichen, kreativen Tätigkeit fehlen. Barry Eaglestone[4] hat in einigen Experimenten versucht, Elektroakustische Komponisten bei der Arbeit zu beobachten und nimmt eine lockere Einteilung in drei Gruppen vor:[1]

- Komponisten, die Software-Werkzeuge in enger Verbindung mit Musik entwickeln und sich dabei leicht an technische Gegebenheiten anpassen
- Komponisten, die mehr auf Musik konzentriert sind, sich aber trotzdem Mühe geben, dafür technisch hochentwickelte Software zu schreiben, teils in der Annahme, dass ein hochentwickelter Kompositionsprozess hochentwickelte Musik hervorbringt
- Komponisten, die so sehr auf Musik konzentriert sind, dass sie keine Minute

[4]Ralf Nuhn, Barry Eaglestone, Nigel Ford, Adrian Moore, Guy Brown, siehe Literaturverweis

> an Softwareentwicklung verschwenden wollen und lieber aus der unorthodoxen Gebrauchsweise bestehender Werkzeuge originelle Resultate gewinnen

Weiter bemerkt Eaglestone, dass häufiges Wechseln zwischen verschiedenen Softwareprodukten (z.B. Sequenzer und Klangsynthese-Software) eher förderlich für kreatives Verhalten ist. Es koste zwar Zeit, aber verhindere auch das lähmende „Festbeissen“ an einem konkreten Detail oder einem konkreten Denkmuster:

> Diversion catalyses the expansion of ideas and possibilities.[1]

Der Trick basiert auf der Annahme, dass verschiedene Softwareprogramme verschiedene Denkmuster implizieren. Löst man sich von einem konkreten Programm und hält Ausschau nach anderen Programmen, fasst man damit gleichzeitig alternative Denkmuster und damit Lösungswege ins Auge. Man erzwingt also Divergenz, wo Konvergenz zu versagen droht. Vergleicht man diesen Trick mit dem vorigen, wo es um externe Limitationen ging, kann man schlussfolgern:

- *Mehr Software* hilft gegen *Konvergenz*blockaden
- *Weniger Software* hilft gegen *Divergenz*blockaden

Ferner kritisiert Eaglestone die vom Hören ablenkende Wirkung grafischer Darstellungen am Computer. Dagegen erwähnt er die Eignung von Papier und Stift, um auf intuitive, ergonomische Weise Zusammenhänge zu markieren, zu gruppieren, zu betonen, zu kommentieren, usw..

All diese obigen Beobachtungen sind am Beispiel Computermusik festgemacht, gelten aber wie gesagt genauso für andere mediale Bereiche.

1.3 Spezialfall Computer-Tonbandmusik

Computermusik ist Musik, die mit Hilfe von Computern gemacht wird. Tonbandmusik ist Musik, die nicht von Menschen aufgeführt wird, sondern das Endprodukt wird auf einem Tonträger oder einer Tondatei gespeichert. Die Schnittmenge aus beiden sei die Musik, um die es hier vorrangig geht.

Obwohl diese Musik von einem Drang zur kompositorischen Grenzüberschreitung und unorthodoxen Herangehensweisen geprägt ist, möchte ich versuchen, die vielen möglichen Einsatzgebiete für den Computer in eine überschaubare Anzahl zu gliedern.

1.3.1 Klang versus Partitur

In den meisten Fällen kann man zwischen Partitur- und Klangdaten unterscheiden. Klangdaten, das sind die große Datenmengen, die über eine triviale Vorschrift (z.B. PCM) direkt hörbare Signale repräsentieren. Dagegen sind Partiturdaten Eingangswerte für Syntheseprogramme, welche anhand dieser Werte Klangdaten erzeugen. Allein die Annahme, dass Partiturdaten auch dynamische Werte beinhalten können, zum Beispiel die Beschreibung einer Lautstärkehüllkurve, weicht dieses Konzept bereits wieder auf. Denn dynamische Werte können sich ja durchaus auch im hörbaren Bereich ändern und damit an sich wieder unter die Definition von Klangsignalen fallen, obwohl ihre *Verwendung* klar auf Partiturebene liegt. Aber wie gesagt, eindeutige Grenzen wird man in diesem Bereich nicht definieren können. Meine Einteilung orientiert sich eher an der Funktion der Partitur als etwas halbwegs überschaubares, das man als Mensch direkt schreiben und lesen kann bzw. könnte, und das zum Hörbarmachen eines interpretierenden Klangerzeugers bedarf. Den meisten verbreiteten Konzepten, die uns bei Musiksoftware begegnen, wird diese Einteilung weitgehend zweifelsfrei gerecht.

1.3.2 Eingabe, Analyse, Synthese, Montage

Was man nun mit diesen beiden Sorten von Daten machen kann, möchte ich wiederum kategorisieren in folgende vier Bereiche: Eingabe, Analyse, Synthese, Montage. Mit der Ausgabe von Klang- und Partiturdaten beschäftige ich mich hier nicht weiter, da sie sich in der Regel aus dem jeweils besprochenen Kontext ergibt, bzw. einen trivialen Vorgang darstellt.

Jeweils auf Klang und Partitur angewendet, ergeben sich acht zusammengesetzte Begriffe:

Tabelle 1.1: Typische Klang- und Partitur-Operationen

	Klang	Partitur
Eingabe	**Klangeingabe**	**Partitureingabe**
Analyse	**Klanganalyse**	**Partituranalyse**
Synthese	**Klangsynthese**	**Partitursynthese**
Montage	**Klangmontage**	**Partiturmontage**

Tabelle 1.2: Typische Hilfsmittel für Klang- und Partitur-Operationen

Operation	Hilfsmittel
Klangeingabe	Audio-Sequenzer, Sample-Editor
Klanganalyse	Analyse-Software, Sample-Editor, Synthese-Software
Klangsynthese	Synthese-Software, Audio/MIDI-Sequenzer, Sample-Editor
Klangmontage	Sample-Editor
Partitureingabe	Audio/MIDI-Sequenzer, Score-Datei
Partituranalyse	?
Partitursynthese	Programmiersprache
Partiturmontage	Audio/MIDI-Sequenzer

Der Inhalt dieser Tabelle, die den genannten acht Operationen gängige Hilfsmittel gegenüber zu stellen versucht, kommt ein wenig krumm daher. Wenn man der von mir gewählten Nomenklatur eine gewisse Sinnhaftigkeit unterstellt, kommt als Sündenbock nur noch eine ungleich gewichtete Repräsentation der Aufgabenbereiche durch heute bestehende Hilfsmittel in Frage.

Klangeingabe ist noch sehr einleuchtend: Klang gelangt in eine Software entweder als Datei oder durch den Analog/Digital-Wandler. Die Klanganalyse scheint bereits durch mehrere Arten von Software repräsentiert zu sein: Reine Analyse-Software, die hauptsächlich zur Anzeige von Daten gedacht ist, daneben Editoren, die zur manuellen Bearbeitung dienen, und solche Programme, die vorwiegend als Synthese-Werkzeuge wahrgenommen werden, daneben aber auch Analysefunktionen bieten. Dagegen ist die Verwendung von Analyse-Software zur Klangsynthese weniger gängig, weil der größte Teil der Verfahren bereits durch Synthese-Software abgedeckt ist. Im Allgemeinen kann man sagen, dass die meisten Computermusikprogramme mehr Synthese- als Analysefunktionalität bieten, was auch nicht verwundert, schließlich ist das erzielte Endprodukt Klang, da bestehen höhere Ansprüche an dessen Erzeugung. Klangmontage im Sample-Editor ist klar. Partitureingabe wird schon spannender. Man möchte erwarten, dass hier mehr verschiedene Konzepte angeboten werden, aber in der Regel konzentrieren sich die Möglichkeiten auf die Eingabe per Maus oder MIDI im Sequenzer und das Schreiben in speziellen Textformaten. Oder übersehe ich hier etwas wichtiges? Ein weiterer Begriff, den die Kategorisierung hervorbringt, ist computergestützte „Partituranalyse“. Mir ist keine Software bekannt, die sich auf das Analysieren von Computermusik-Partituren spezialisiert. Wozu sollte ein Komponist auch seine eigene Partitur analysieren wollen? Und dann auch noch computergestützt? Das klingt auf den ersten Blick nach viel überflüssiger Arbeit, wäre in der Tat in vielen Fällen redundant und bedarf zweifellos einiger Klärung. Ich komme später darauf zurück. Jedenfalls bitte ich jetzt schon zu bemerken, dass auf Partiturebene ein noch stärkeres Gefälle zwischen Analyse und Synthese besteht als bei Klang. Und darüber hinaus, dass im Partiturbereich Analyse (was immer das sein mag), Synthese und Montage praktisch nie im selben Programm stattfinden. Dagegen gibt es auf Klangebene immerhin den Sample-Editor, der als Grenzgänger notfalls alle vier Arten von Klang-Operationen abdecken kann.

1.3.3 Universelle vs spezielle Software

Eaglestone rät ja vom Traum der „eierlegenden Wollmilchsau“ tendenziell ab. Trotzdem entwickeln sich viele Programme über die Jahre zu solchen, oder zeigen zumindest deutliche Tendenzen zur Alleskönnertum.

Dabei ist das Argument von Eaglestone nicht von der Hand zu weisen: Jede auch noch so klug entworfene Software funktioniert auf *eine* bestimmte Weise, und bereits dieser Umstand macht es nachteilhaft, sich über zu lange Zeit ununterbrochen in ihr aufzuhalten.

Eaglestone spricht aber auch einen ökonomischen Nachteil an, nämlich den Zeitverlust beim Wechseln. Je nach dem, wie häufig gewechselt wird, kann der Import und Export zwischen den Programmen sogar mehr Zeit in Anspruch nehmen als die eigentliche kreative Tätigkeit. Als einfaches Beispiel sei der Export eines Audiofragments aus Logic genannt, welches in CSound bearbeitet[5] und zurück importiert werden soll. Es beginnt in Logic:

Start- und Endzeit markieren - Spur oder Region *Solo* schalten - Export-Menü anwählen - Dateinamen angeben - Format einstellen - mit OK bestätigen - warten bis Export abgeschlossen ist - *Solo* wieder raus - Spur oder Region stumm schalten - Konsole öffnen - Score-Datei bearbeiten - Audiodateipfad lokalisieren und einfügen - Dauer einfügen - weitere Parameter einfügen - CSound-Kommandozeile eintippen mit Pfad von Ausgabedatei, Orchestra-Datei und Score-Datei - warten bis Berechnung abgeschlossen ist - wieder Logic anwählen - neue Audiospur erstellen und anwählen - Importdialog öffnen - Datei lokalisieren - mit OK bestätigen - Region mit der Maus an die richtige Stelle ziehen - abspielen.

23 Arbeitsschritte, großzügig gerechnet, und mögliche Fehler nicht mitgezählt. In der Praxis gehört es selbstverständlich dazu, dass man das *Solo*-Schalten vergisst

[5] Eine bereits funktionierende CSound-„Orchestra-Datei“ vorausgesetzt

oder aus Versehen aus Logic in 24 Bit oder aus CSound in 32 Bit exportiert und daraufhin einige Schritte wiederholen muss. Aber wahrscheinlich bleibt man unter 30 Arbeitsschritten und unter 3 Minuten. Gemessen an 3 Arbeitsschritten in 5 Sekunden, die es braucht, um statt CSound ein Plugin innerhalb von Logic zu benutzen, zeigt das schon einen beachtlichen Unterschied in Sachen Workflow[6]. Es ist wirklich eine Kosten-Nutzen-Rechnung:

- Brauche ich an dieser Stelle wirklich die Spezialsoftware?
- Kann mein Allround-Werkzeug nicht auch so etwas ähnliches?
- Kann der Kollege[7] schon wieder eine Kaffeepause verkraften?
- Habe ich die Nerven für die Extra-Klickerei?
- Verträgt die Arbeitsatmosphäre das, oder vergeht uns allen die Lust?
- Verträgt der Zeitplan das?
- Was mache ich, wenn das Klangresultat nicht auf Anhieb begeistert?

Die utopische Annahme, dass das Klangresultat auf den ersten Versuch begeistert, ist zugegeben in den obigen Fall mit eingerechnet. Im realistischeren Fall müssen natürlich nicht alle 23 Schritte wiederholt werden (das nochmalige Exportieren fällt weg), sofern mit dem gleichen Fragment weitergearbeitet wird. Trotzdem bleiben in dem Fall noch ungefähr 10 für jede Berechnung mit neuen Parametern, was ein exploratives Arbeiten kaum möglich macht, zumindest nicht effizient. Könnte man mit CSound direkt nach Logic hineinspielen, ließen sich die Arbeitsschritte auf akzeptable 3 oder 4 herunter schrauben, und die Welt sähe ganz anders aus. Aber meines Wissens geht das nicht. Also sucht man im Eifer des Gefechts lieber nach einer schneller greifbaren (Not-)Lösung innerhalb des

[6] Von *Flow* im Sinne von Mihaly Csikszentmihalyi ganz zu schweigen.
[7] Mitkomponist, Musiker, Choreograph, Regisseur...

Sequenzers, was natürlich wieder auf die uralten Standard-Effekte hinausläuft - mit Chorus, Flanger, Delay, Distortion, Pitchshifter und noch ein paar anderen, tendenziell langweiligen Filtertypen.

Logistische Barrieren zwischen Programmen haben also weit mehr als nur zeitliche Konsequenzen. Und dabei stammt das gewählte Beispiel wohlgemerkt aus der Klangmontage, was der Audiosequenzer ja noch relativ gut kann. Aber eben nur *relativ* gut. Hätte er überhaupt keine eingebauten Effekte, wäre die Entscheidung eine ganz andere. Der umständliche Import-/Exportvorgang würde als notwendiges Übel hingenommen werden, insgesamt würde man wahrscheinlich weniger Bearbeitungen am Klang vornehmen und seine *Kompositionsstrategien um die deutlich wahrgenommenen externen Limitationen herumbauen.* Und trotzdem erlitte man die softwarestrukturelle Vereinsamung nach Eaglestone.

Mein Vorschlag zur Weltverbesserung liegt also in der Verbesserung der Infrastruktur beim Datenaustausch zwischen möglichst verschiedenen Programmen. Audiodateien reichen als Währung nicht aus, „Jack“ ist ein Schritt nach vorne, OSC auch, aber mehr dazu später.

1.3.4 Wo der Computer versagt

Die acht Operationen Klangeingabe, Partitureingabe, Klanganalyse, Partituranalyse, Klangsynthese, Partitursynthese, Klangmontage und Partiturmontage könnte man als Handlungsfelder oder Wege der Datenmanipulation bezeichnen. Diese Einteilung ist technisch gedacht und orientiert sich an Informationen, die dem Computer mitteilbar, also sinnvoll in Zahlen artikulierbar sind.

Natürlich besteht Musik aus mehr als nur Zahlen, und es stellt sich die Frage, über welche Wege diese nicht-numerischen Komponenten vom Hirn auf den Tonträger gelangen.

> I find it so amazing when people tell me that electronic music has not got soul and they blame the computers: „There is no soul here.“ It's like you couldn't blame the computer: if there's no soul in the music it's because nobody put it there.[8]

Dieser viel zitierte Satz, den Björk einmal in einem Fernsehinterview gesagt hat, profitiert in unserem Zusammenhang von einer Verallgemeinerung:

If there's no ..... in the music it's because nobody put it there.

Eine Liste von Begriffen zu finden, die hier gut hinein passen, sei dem Leser selbst überlassen. Jedenfalls hoffe ich damit folgendes verdeutlicht zu haben: Sofern uns lediglich die obigen „8 Operationen“ zur Verfügung stehen, muss jedes erzielte, wahrnehmbare, musikalische Merkmal über einen dieser acht Wege aufs Tonband gelangen. Merkmale, die nicht von der trivial-arithmetischen Arbeitsweise des Computers erfasst und verarbeitet werden können, müssen geschickt hinter dessen Rücken vorbei geschleust werden. Klangeingabe macht die Sache einfach: Man nimmt ein Stück Audio, das das gewünschte Merkmal trägt, und fügt es unbearbeitet oder nur leicht angepasst in die Komposition ein. Mit Synthese wird es schon schwieriger. Sofern die erzielten Merkmale im wissenschaftlich erschlossenen Bereich der Akustik liegen, wie etwa Raumbewegung, Reflexion, klangliche Eigenschaften bestimmter physikalischer Materialien usw., kann man sie normalerweise durch physikalische Modelle nachbilden, die im Computer nicht viel mehr sind als einfache Filtergleichungen. Viele dieser Merkmale sind durch bestehende Synthesizer repräsentiert, und der Komponist muss nicht einmal die physikalischen Gesetze im Detail verstehen, um trotzdem gezielt die erwünschten

Merkmale anzusteuern. Im erschlossenen Bereich der Psychoakustik werden vorgefertigte Lösungen seltener, aber auch hier kann der ambitionierte Komponist mit Mühe und Geschick ein erzieltes Ergebnis mittels eigener Syntheseprogramme zielgerichtet erarbeiten.

An der Häufung des Begriffs *Ziel* erahnt der Leser vielleicht, worauf ich hinaus will: Sobald man sich im akustischen und psychoakustischen Wilden Westen bewegt, kommt eine konvergente Produktionsmethode nicht mehr in Frage. Hier liegen die größten Chancen in der kreativen Variantenbildung.

1.4 Verbreitete Konzepte in der Musiksoftware

Dieses Kapitel stellt keinerlei Anspruch auf Vollständigkeit. Ein Versuch, jedes Programm im audionahen Bereich, das jemals geschrieben wurde, zu untersuchen und aufzulisten, würde bereits beim Ausfindigmachen scheitern. Also wird nicht alles dran kommen. Nicht nur einzelne Programme fallen komplett aus der Beobachtung heraus, auch ganze Kategorien, zum Beispiel werde ich nicht über Sampleeditoren sprechen.

Stattdessen geht es in diesem Kapitel um Produktionskonzepte, die mir aus eigener, intensiver Auseinandersetzung in der Vergangenheit hinreichend vertraut sind. Von diesen möchte ich einige Aspekte beleuchten, die mir im Rahmen meiner Recherche besonders aufgefallen sind. Meistens, weil sie technische Limitationen implizieren, die meines Erachtens die Tendenz haben, unterschwellig in die kreative Arbeitsweise einzufließen.

1.4.1 Audio/MIDI-Sequenzer

Unter dieser Bezeichnung fassen wir zusammen:

- Apple Logic
- Ardour
- Digidesign ProTools
- MusE
- MOTU Digital Performer und AudioDesk
- Propellerhead Reason
- Rosegarden
- Steinberg Cubase
- Steinberg Nuendo
- und viele andere

Eine früher gängige Einteilung in Audio-Sequenzer und MIDI-Sequenzer macht heute weniger Sinn, da die Programme zunehmend beides anbieten. Welche Programme ihren Anfang als virtuelle Bandmaschine-Mischpult-Effektrack-Simulation hatten, und welche als reine MIDI-Sequenzer begonnen haben, ist bisweilen gar nicht mehr klar erkennbar. In beiden Fällen kann man jedoch sagen, dass die Architektur der Software an Verfahren orientiert ist, die in Musikstudios eine lange Tradition haben. Tonaufnahme, Schnitt, Sequencing[8], Synchronisation, Mischung und Mastering wurden nach und nach aus ergonomischen und ökonomischen Gründen von der Hardware-Welt in die Software-Welt verlagert.

[8]Fernsteuerung vorwiegend von elektronischen Tasten- und Schlaginstrumenten

Typisch für diese Softwarekategorie sind:

- Es gibt eine globale Zeitleiste, auf der Audio- und MIDI-Regionen angeordnet werden. Die Auflösung ist sehr hoch, Darstellung und Rasterung erfolgen wahlweise in Sekunden, Timecode, Metrum, etc.
- Gleichzeitige Regionen werden auf verschiedene Spuren verteilt.
- Es gibt Audiospuren und MIDI-Spuren.
- Jede Audiospur ist an einen Mischpultkanal gekoppelt.
- Jede MIDI-Spur ist an einen Synthesizer gekoppelt, der seinerseits in einen Mischpultkanal mündet.
- Jeder Mischpultkanal bietet Insert-Effekte als Plugins.
- Mischpultkanäle können auf Busse oder Summenkanäle geroutet werden, nicht jedoch auf andere Mischpultkanäle.
- Audiodateien können entweder direkt in der Timeline oder im Sampler verwendet werden. Beides hat Vor- und Nachteile.
- MIDI wird verwendet für Notenereignisse und skalare Steuerdaten (Controller, Mischpult-/Effektautomation).
- eingebaute Hilfsmittel wie Sample-Editor, Spektralanalyse, Videosynchronisation, einfacher Notensatz

Audio-Regionen Eine Audio-Region in Ardour besitzt außer Einsatzzeitpunkt und Dauer nicht viel mehr als eine Referenz auf eine bestimmte Audiodatei und ein Offset, das angibt, ab welcher Startposition das Sample ausgelesen werden soll. Abspielgeschwindigkeit und -richtung sind nicht variabel, was sich historisch

damit begründen lässt, dass Spuren auf einem Mehrspurband ja auch nicht verschieden schnell laufen können. Eine aktueller, technischer Grund hängt mit den langsamen Zugriffszeiten von Festplatten zusammen. Diese erlauben es nicht, beliebig schnell in großen Dateien hin- und her zu fahren oder zu springen. Darum liest man Dateien per „Streaming“ aus, also in kurzen, gleichmäßigen Häppchen. Der Vorteil dieser Lösung, in der Audiodateien nicht komplett ins RAM geladen werden müssen, besteht im verzögerungsfreien Umgang mit beliebig großen Datenmengen, was für professionelle Schnittprogramme unabdingbar ist. Außerdem würde eine variable Abspielgeschwindigkeit das Verhalten bei Schnitten verkomplizieren.

MIDI-Regionen Daneben ist die Verwendung von MIDI-Regionen üblich, in denen im Wesentlichen MIDI-Messages (Noten- und Steuerdaten) enthalten sind. Diese Daten werden an einen externen oder nativen Klangerzeuger geschickt, der sie interpretiert und in Klang umwandelt. Das entscheidende Wort hierbei ist interpretiert. Man darf nicht vergessen, dass der MIDI-Standard eine umfangreiche Menge von Konventionen definiert, von temperierten Stimmung bis zur festen Zuweisung von Controllernummern auf genormte Parameter[9]. Der Beginn einer Note wird außer dem Zeitpunkt noch durch zwei andere Werte beschrieben: „Tastennummer“ und „Anschlagsstärke“. Mit Ausnahme einiger weniger (Tasten)instrumente zielen diese beiden Parameter zwar völlig an der Realität vorbei, aber trotzdem gilt die Norm, und jeder gewöhnliche MIDI-Synthesizer weiß diese Angaben in Klangparameter umzusetzen. Üblicherweise werden daraus bei gestimmten Instrumenten eine statische Tonhöhe und maximale Lautstärke abgeleitet, bzw. bei Drumkits und dergleichen Samplenummer und Lautstärke. Ein Vorteil ist, dass eingegebene MIDI-Regionen in den meisten Fällen auch auf anderen Synthesizern abgespielt werden können, und es kommen die gleichen Ton-

[9] 7=Gesamtlautstärke, 10=Panorama, etc.

höhen heraus. Als allgemeiner Datentyp für beliebige Klangsynthesen kann das natürlich nicht durchgehen. Das ist buchstäblich so, als hätte man bei CSound nur zwei freie P-Felder (p4 und p5) mit ganzzahligen Werten von 0 bis 127 zur Verfügung.

Aufgrund ihrer Tradition sind diese Programme bestens auf die Bedürfnisse von Standard-Musikproduktionen spezialisiert, daher verwenden die meisten kommerziellen Studios eines der oben genannten Programme als zentrale Steuereinheit. Logic, Cubase & Co. werden nicht so sehr als Spezialwerkzeuge betrachtet, viel mehr präsentieren sie sich als Allzweck-Studioumgebung zur Realisierung ganzer Musikproduktionen von der Aufnahme bis zum Mastering. Tatsächlich ist jeder kompositorische Eingriff, der sich entweder als Bandschnittvorgang, Notenbearbeitung oder Mischung deuten lässt, innerhalb der Software mit Leichtigkeit zu realisieren. Ebenso jede Klangsynthese, die durch vorhandene Synthesizer-Plugins repräsentiert ist oder durch Reihen- und Parallelschaltung von vorhandenen Effekt-Plugins erreicht werden kann.

Andere mögliche Eingriffe

- werden von der GUI nicht ermöglicht oder ‘verschwiegen’
- erfordern häufig ein Verlassen des Programms, verbunden mit unergonomischen Export- und Import-Maßnahmen
- erscheinen angesichts des allgemeinen Bedienkomforts noch unbequemer als sie es objektiv sind.

Klassische Audio/MIDI-Sequenzer Seite begegnen dem Problem von Masse, Dichte und Polyphonie prinzipbedingt mit einer starren Hierarchie. Besonders im Umgang mit Polyphonie trennen sich die Welten in Audio- und MIDI-Sequencing.

Polyphonie bei Audio

- Es gibt eine feste Anzahl von Audiospuren.
- Jeder Audiospur ist ein Mischpultkanal fest zugeordnet.
- Audiospuren spielen nie mehr als eine Audioregion gleichzeitig ab[10].

Um beispielsweise ein normales Klavier-Multisample ohne Zuhilfenahme eines MIDI-Samplers zu benutzen, bräuchte man daher bis zu 88 Spuren und ebenso viele Mischpultkanäle. Die Session würde sehr schnell unübersichtlich werden.

Polyphonie bei MIDI

- Eine MIDI-Region enthält MIDI-Noten in beliebig hoher Polyphonie.
- Die Noten müssen von einem Synthesizer/Sampler interpretiert werden.
- Heraus kommen Noten nicht einzeln, sondern zu 1 Audiospur gemischt.

Auf diese Weise kommt ein Klavier-Multisample mit nur einer Audiospur aus.

Der Nachteil bei MIDI-Polyphonie ist, dass die gespielten Noten nicht mehr einzeln auf Audioebene bearbeitet werden können, was beim Klaviersample im Normalfall nicht weiter stört. Kritischer wird es, um ein populäres Beispiel zu nennen, bei Schlagzeug-Multisamples. Dabei kann es durchaus wünschenswert sein, z.B. einzelne Beckenschläge mit dem Equalizer zu bearbeiten, was in dem Fall an einem Strukturproblem scheitert. Daher verzichten viele Produzenten bei Schlagzeugsamples auf die Vorzüge von MIDI und platzieren grundsätzlich die Audiosamples der einzelnen Schläge als einzelne Regionen direkt in Audiospuren. Damit geht neben MIDI-Bearbeitungsfunktionen ein wichtiges Mittel zur Abstraktion verloren, nämlich die Gruppierung in Regionen. MIDI-Noten sind automatisch immer

[10] abgesehen von Crossfades, da sind es kurzzeitig zwei

in MIDI-Regionen gruppiert, während Audioregionen grundsätzlich auf globaler Ebene liegen. Mag es dabei für den HipHop-Produzenten noch hinnehmbar sein, 200 HiHat-Schnipsel hintereinander auf einer Spur liegen zu haben, wird es spätestens für den Computermusiker ärgerlich, wenn es sich um 200 Grains pro Sekunde handelt, die sich auch noch bis zu fünfzigfach überlappen. Ein und das selbe Masse- und Polyphonieproblem kann sich je nach Musikrichtung in ganz unterschiedlichen Dimensionen ausprägen.

Abstraktion Daher braucht eine Software, die allgemein für Tonbandmusik taugen will, ausdrucksstarke Mechanismen zur Abstraktion: Einzelheiten müssen sinnvoll zu größeren Einheiten gebündelt werden können.

In der rein instrumentalen Musikproduktion ergeben sich die Abstraktionsmuster in der Regel automatisch aus den Gegebenheiten bei der Tonaufnahme: jedes Instrument eine Spur, jeder "Take" pro Spur eine neue Region. Da stellt sich kaum die Frage nach sinnvollen Alternativen. In Anbetracht der langen Tradition von Musikproduktionen, die so funktionieren, mag es nicht weiter überraschen, wenn diese bewährten Organisationsmuster weitgehend unreflektiert auch in instrumental geprägte Musikproduktionen mit elektronischer Klangerzeugung einfließen. So lange die Musik in einer greifbaren Zahl von Stimmen gedacht ist, lässt sich diese Einteilung in aller Regel so reibungslos auf das vom Audio/MIDI-Sequenzer angebotene Abstraktionssystem übertragen, dass es gar nicht als solches auffällt.

Flexible Abstraktionsmittel werden zunehmend wichtiger

- mit steigender Anzahl der Regionen insgesamt
- mit steigender Polyphonie
- je weiter die Musik von instrumentalen Standardmustern abweicht

Es gibt auch in verbreiteten Audio-/MIDI-Sequenzern Bestrebungen, auf die bestehenden Abstraktionsmittel weitere aufzusetzen. Ein mir bekanntes Beispiel sind Folder in Logic, mit denen selten gebrauchte Spuren zur visuellen Entlastung zusammen gefasst und versteckt werden können. Trotzdem löst das nicht das grundlegende Strukturproblem. In dem oben genannten Beispiel mit den 200 Grains kommt als Lösung weder Audio-Sequencing in Frage, da eine Grainwolke 50 Spuren am Mixer belegen würde, noch MIDI-Sequencing, weil dafür die Notenparameter fehlen. Um ein Grain abzuspielen, braucht es mindestens die Angaben: 1. Audiodatei als Buffer-Index oder Dateipfad, 2. Startposition. Allein schon diese beiden wären in den für „Tastennummer“ und „Anschlagsstärke“ reservierten 8 Bit nur mit viel Phantasie und geringen Ansprüchen an Genauigkeit und Bufferanzahl unterzubringen, spätestens wenn weitere Parameter wie Lautstärke, Trasposition, Hüllkurve, etc dazu kommen sollen, ist Feierabend.

Automatisieren von Handgriffen Generell sind Sequenzer für manuelle Bearbeitung ausgelegt, wie ja auch ihre Hardware-Vorfahren mit der Hand bedient und bearbeitet werden. Mit dem Umzug auf Software haben sich die Handgriffe auf Mausklicks reduziert, und viele von ihnen, besonders in den Bereichen Schneiden, Kopieren, Kleben brauchen nur noch einen Bruchteil der Zeit im Vergleich zu früher. Andere fallen komplett weg, z.B. Bänder wechseln, spulen, Mischpult zurücksetzen, usw.. mit der Konsequenz, dass die gleiche Arbeit in viel kürzerer Zeit bzw. in der gleichen Zeit viel mehr Bearbeitungsschritte erledigt werden können. Letzteres gibt dem kreativen Studiobenutzer Freiraum für Experimente. Trotzdem gibt es ein Limit bei der Anzahl von Mausoperationen, die ein geübter ProTools-Operator in der Sekunde schafft. Zum Glück sind die gängigsten Mainstream-Anwendungen in professioneller Software als automatische Prozeduren repräsentiert, z.B. Entfernen aller stumm geschalteten Regionen, automatisches Schneiden einer Audiodatei nach Lautstärkepegel, Exportieren aller Audio-

spuren als einzelne Dateien. Nuendo (und wahrscheinlich auch andere) erlaubt das Verketten mancher GUI-Operationen zu sogenannten Macros. Damit lassen sich effektiv einige Mausklicks einsparen, in dem Anweisungen wie „Tu damit dies, danach das, anschließend jenes“ gebündelt werden können. Die Funktionalität ist jedoch stark eingeschränkt, da sprachliche Ausdrucksmittel wie Funktionsparameter, Rückgabewerte, Kontrollstrukturen, etc. fehlen. An Partitursynthese oder intelligentes Verhalten mittels Macros ist folglich nicht zu denken. Schon in dem einfachen Fall „schneide Region A an jeder stimmlosen Stelle“ geht meines Wissens auch in der teuersten Digital Audio Workstation die Handarbeit los. Je weiter die Anwendung vom Mainstream entfernt ist - und die Rede ist hier von Computermusik - desto eher stößt der Benutzer an die Grenzen des Implementierten. Zum Zeitpunkt des Schreibens ist mir kein Audiosequenzer bekannt, der neben einer ausgereiften Funktionalität mit GUI auch eine ausgereifte, interaktive Script-Schnittstelle ähnlich wie GIMP bietet[11]. Also was kann man tun, wenn etwas automatisch ablaufen soll? Im Zweifelsfall wird man wohl die problematische Stelle in einer sprachbasierten Umgebung wie CSound realisieren müssen. Falls die problematische Stelle sich schlecht isolieren lässt, weil sie von ihrem Kontext abhängig ist, muss dann auch der ganze Kontext mit in die andere Umgebung.

Import / Export Eine Kompromisslösung kann sich über die vom Sequenzer unterstützten Dateiformate ergeben. Logic- oder Nuendo-Sessions sind Binärdateien, die man nicht so ohne weiteres lesen kann[12]. Das MIDI-File-Format bietet sich leider nur für Daten an, die innerhalb des Sequenzers den MIDI-Weg gehen, was Computermusikern oft wenig nützt. Die Software *Mix* von Øyvind Hammer, ein einfacher Audiosequenzer mit rudimentären Mischfunktionen, nutzt zum Laden und Speichern Textdateien mit einer sehr einfachen Syntax. Diese kann man per Programmiersprache erstellen (siehe weiter unten: Partitursynthese) und auch

[11] Fundmeldungen und sachdienliche Hinweise bitte an *kreitmayer@folkwang-hochschule.de*
[12] Teilweise ist das von den Herstellern sogar beabsichtigt.

mit geringem Aufwand einlesen. Ähnlich verhält es sich mit Ardour, das seine Daten in standardisiertem XML aufbewahrt. Leider ist die Syntax[13] nicht dokumentiert und erschließt sich nicht vollständig aus dem Studium bestehender Session-Exemplare. Auch der offene Quellcode von Ardour lässt keine zuverlässige Analyse zu, die garantiert, dass selbstgenerierte XML-Sessions in dieser oder der nächsten Ardour-Version funktionieren werden. Aber im Prinzip ist das möglich. Mit Geduld und Glück kann man theoretisch auch komplizierte Ardour-Sessions skriptweise bauen. Auch von Nuendo ist mir eine gewisse XML-Unterstützung bekannt, zum Beispiel das Ex- und Importieren einzelner oder mehrerer Spuren per XML. Dazwischen kann man theoretisch einiges manipulieren, so lange man nicht versehentlich gegen die Syntax- oder Semantikregeln verstößt. Aber es ist wie gesagt ein riskanter Weg, die Programme sehen das nicht vor und sind auch nicht für besonders hilfreiche, detaillierte Fehlermeldungen im Fall von korrupten Dateien bekannt.

1.4.2 Melodyne

Eine Software, die sich gezielt gegen das strukturelle Audio/MIDI-Problem im Sequenzer wendet, ist Melodyne. Es vereint Konzepte eines mehrspurigen Editors mit dem Paradigma, dass Audiomaterial in seinem zeitlichen Verlauf, so wie in Tonhöhe (falls vorhanden) und spektralen Eigenschaften frei gestaltet werden kann.

Aufgenommene oder geladene Audiodaten werden analysiert mit hochentwickelten Verfahren zur Erkennung von Klangereignissen (Noten). Die Software weiß sozusagen, im Rahmen der modernsten technischen Möglichkeiten, über den Inhalt des Materials Bescheid - eine Eigenschaft, die ich generell für wertvoll halte. Denn auf diese Weise können darauf folgende Montage- und (Re-)Synthese-Schritte sen-

[13] Gemeint ist nicht die XML-Syntax, sondern die Reihenfolge, in der Ardour seien Daten erwartet, und wie diese Daten strukturiert sein müssen

sibel und intelligent mit dem Material umgehen. Bei Melodyne bedeutet das eine ungleich flexiblere Gestaltung und bessere Klangqualität als es mit „blinden“ Verfahren möglich ist. Aber auch andere Sequenzer würden in meinen Augen stark profitieren von der Fähigkeit, Audiodaten in Zusammenhang mit Analysedaten zu behandeln.

Bemerkenswert ist weiterhin die wahlweise Verwendung als eigenständige, mehrspurige Schnittsoftware, sowie die Einbindung als Plugin in eine Audio-Sequenzer-Spur.

1.4.3 Tracker

In der deutschsprachigen Wikipedia findet man eine Liste mit zur Zeit 346 Programmtiteln, die zu dieser Kategorie gerechnet werden [10], über 70 allein für den C64. Die englischsprachige Abteilung zählt immerhin noch 266 Stück [11], über die Dunkelziffer mag man spekulieren. Jedenfalls läge es mir fern, diese Liste im Detail zu verifizieren. Mir persönlich bekannte Beispiele beinhalten: Ultimate Soundtracker, Fast Tracker, Protracker, OctaMED, Technosound Turbo II, Data Becker MusicMaker, Oktalyzer, PlayerPRO.

Weit verbreitet sind Tracker auf älteren Plattformen und solchen mit bedeutender Spieletradition. Tatsächlich stellten Tracker aufgrund ihres Ressourcen sparenden Konzepts lange Zeit die einzige Möglichkeit dar, der begrenzten Hardware klanglich vielfältige Musik abzugewinnen. Noch heute wird das von Trackern generierte MOD-Datenformat im Bereich Computerspiele eingesetzt, wo Speicher und Rechenleistung (besonders bei Internet- und Handyspielen) eng begrenzt sind.

Trotz unzähliger Varianten und Grenzgänger lässt sich doch ein allgemeiner Stereotyp von Tracker beschreiben:

- Typischerweise besteht ein Tracker aus der Kombination von Sequenzer und Sampler.
- Optisch auffällig ist meist die Leserichtung von oben nach unten. Eine Liste von Zeitpunkten wird durchlaufen, zu denen Noten ausgelöst und Steuerbefehle (Tempowechsel, Glissando, etc) gesetzt werden können.
- Die Zeit ist dabei grob aufgelöst, wie im übrigen alle Parameter.
- Als einziges Mittel zur Abstraktion stehen „Patterns“ zur Verfügung, mit denen Partiturabschnitte wiederholt werden können.
- Die Samples liegen grundsätzlich im RAM und können transponiert werden.
- Transposition und Effekte werden pro Note festgelegt, nicht pro Spur.
- Weil Spuren keinem Routing unterliegen, spielt es klanglich keine Rolle, auf welcher (freien) Spur ein Sample liegt[14].
- Gehaltene Noten können auch nach ihrem Einsatz noch gezielt in ihren Parametern verändert werden.
- Arpeggios sind eine Spuren sparende Methode, Polyphonie zu simulieren.
- Die Notwendigkeit, Speicherplatz zu sparen, hat den Gebrauch von Sample-Loops eingebürgert.
- Ein integrierter Sample-Editor gehört in der Regel dazu.

[14] Ausnahme: Bei älteren Amiga-Modellen liegen ungeradzahlige Kanäle am linken, geradzahlige am rechten Audioausgang. Bestimmt gibt es noch mehr Ausnahmen.

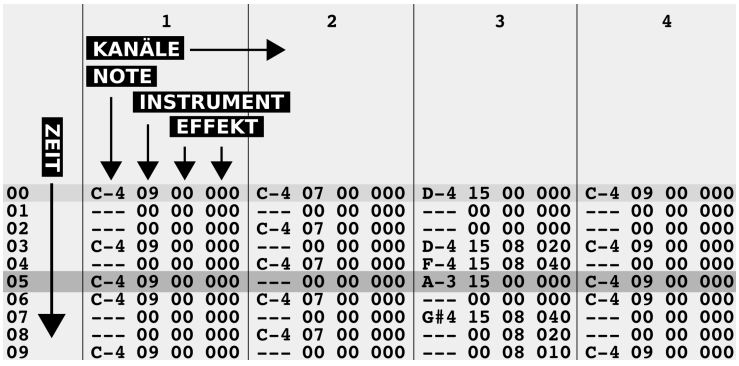

Abbildung 1.1: Typische Tracker-Struktur.

Natürlich gelten diese Eigenschaften nur als grobe Typisierung, und niemand hat den Begriff *Tracker* wirklich patentiert. Z.B.„Soundmonitor“ von Chris Hülsbeck, der oft als „der erste Tracker“ bezeichnet wird, konnte überhaupt keine Samples abspielen, sondern nutzte in spezieller Weise die drei Oszillatoren des C64.

Besonders älterer MOD-Musik hört man ihre zugrundeliegende Software deutlich an, und der allergrößte Anteil der heutigen Stücke versucht auch gar nicht, anders zu klingen als nach Vierspur-Spielesoundtrack. Tatsächlich machen die enge Kopplung des Sequenzers an den Sampler und die fehlenden Abstraktionsmittel das Tracker-Prinzip meines Erachtens für kompliziertere Arrangements relativ ungeeignet. Die übliche Darstellung der Spuren als Textspalten hat ihre Grenzen bei der Übersichtlichkeit. 20 Noten gleichzeitig bedeutet: 20 Spuren gleichzeitig, was in aller Regel die Bildschirmbreite überschreitet. In der Pop-Produktion verbreitete MIDI-Anwendungen (Klavier oder Streicher einspielen, Controllerbewegungen aufzeichnen und bearbeiten) scheitern schnell an der eingeschränkten Polyphonie und der groben Zeitauflösung. Die allgemeine Beschränkung der Sample-Datenmenge durch den Hauptspeicher und sicher noch einige andere Gründe haben dazu geführt, dass die Familie der Tracker es nie wirklich in die Tonstudios geschafft hat - mit Ausnahme derer, die Musik für Computerspiele und die Demoszene[15] produzieren.

[15]Sportlich orientierte Gattung von Multimediaprogrammierung mit dem Ziel, möglichst spektakuläre Echtzeit-Animationen mit Musik auf möglichst alten Rechnern laufen zu lassen

Im Bezug auf das vorhin erwähnte Grain-Beispiel könnte man sagen, dass das Tracker-Konzept in diesem Fall die Nachteile von Audio- und MIDI-Sequencing vereint: Jede Polyphonie kostet Spuren, *und* es fehlen die nötigen Startparameter, um ein Grain abzufeuern. Was ich jedoch noch einmal positiv betonen möchte, ist die grundsätzliche Möglichkeit, einzelne Noten nach dem Start noch zu modulieren. Ich kann ein Klaviersample auf E abspielen und halten, eine Viertel später ändere ich die Tonhöhe per Spurbefehl mit Glissando auf D, danach lasse ich die selbe Note in 32stel zwischen drei anderen Tönen arpeggieren. Logic kann das nicht, weder per Audio noch mit MIDI[16]. In CSound geht es mit Umwegen, richtig gut funktionieren diese Dinge mit SuperCollider, wo Noten über eine eindeutige Kennung angesprochen werden können.

Beim Tracker wird das Privileg der dynamischen Note jedenfalls teuer erkauft, dadurch dass jede Note wie gesagt eine Spur belegt. Wenn man sich nun experimenthalber vorstellt, ein polyphones MIDI-Sampler-Plugin in Logic würde das gleiche versuchen - und statt einer einzigen Audiospur viele Audiospuren ausgeben, eine pro Note - dann könnte man die einzelnen Stimmen immerhin mit Audioplugins bearbeiten. Zum Beispiel der im Abschnitt über Audio-/MIDI-Sequenzer erwähnte einzelne Beckenschlag wäre mit Equalizer gestaltbar. Um das Gedankenexperiment aber zu Ende zu spielen: Die Tonhöhe, also die Abspielrate des Samples, ist ein Syntheseparameter und gehört zum Sampler. Die erzeugten Audiospuren müssten also auch noch Automationsdaten zurück an ihr Plugin senden können, um die Abspielrate dynamisch zu variieren.

[16]Pitchbend und andere Controller gelten kanalweise, nicht auf einzelne Noten bezogen.

1.4.4 CSound / Music-N

Von allen direkten Nachfahren der „Music-N“-Familie ist CSound heute der mit Abstand populärste. Verwandte Programme sind unter anderem CMix, RTcmix, CMusic und SAOL, mit denen ich aber keine praktische Erfahrung habe. Daher werde ich mich im Folgenden auf CSound beschränken.

- Es existiert eine große Auswahl an eingebauten Algorithmen für Klangsynthese und Klanganalyse.
- Die Partitur besteht im Wesentlichen aus einer Liste von (Klang-)Ereignissen und liegt schriftlich in Form einer Textdatei vor.
- Diese Datei kann sowohl vom Menschen als auch vom Computer bequem geschrieben, bearbeitet und gelesen werden.
- Über eine einfache Syntax werden Synthese- und Analysebausteine zu Synthesizern zusammen gesetzt.
- Audiosamples können von Festplatte oder aus dem Speicher gelesen werden.
- Die Synthese kann sowohl interaktiv in Echtzeit erfolgen, als auch „offline“.
- Der Polyphonie sind keine strukturellen Grenzen gesetzt.
- CSound kann als Shell-Kommando manuell aufgerufen werden, oder auch von externen Programmen, etwa zur Stapelverarbeitung von Analyse- / Syntheseprozessen.
- Außer der üblichen Audiosumme können auch andere Signale oder Steuerwerte zur Weiterverarbeitung als Audio- oder Textdatei ausgegeben werden.
- Nach Beendigung eines Offline-Prozesses terminiert das Programm.

Über die technischen Stärken und Schwächen von CSound ist bereits viel diskutiert worden, auch das hätte ein Schwerpunkt dieser Arbeit werden können. Kurz gesagt, es gibt wohl kaum eine Anwendung, zu der man CSound nicht auch in gewissen Grenzen teilweise missbrauchen könnte, das Erzeugen von Partituren eingeschlossen. Deshalb halte ich mich hier mit Aussagen über die realen Grenzen der Software zurück und beschränke mich auf die vorsichtige These, dass der Einsatz von CSound als Synthese- und Analysehilfsmittel und die Auslagerung von so viel Intelligenz wie möglich in externe Prozesse (ausgereifte Programmiersprachen) sich bisher oft als erfolgreiche Strategie erwiesen hat.

1.4.5 Partitursynthese

Unter Partitursynthese verstehe ich das algorithmische Erzeugen von Daten, die anschließend von einem separaten Syntheseprozess gelesen und verarbeitet werden. Dabei könnte die Synthese auch instrumentaler Natur sein, aber ich möchte mich hier auf Tonbandmusik beschränken. CSound-Score-Dateien mittels einer universellen Programmiersprache wie Scheme, C, etc. zu programmieren, wäre ein typisches Beispiel. Daneben gibt es auch spezielle Partitursynthese-Software wie Common Music, Projekt 1/2, PPP, RTC-lib und viele andere, die jeweils bestimmte Strukturmuster betonen, hier aber nicht weiter vertieft werden, da es um Partitursynthese im Allgemeinen geht.

- Aus einem kleinen Parametersatz werden große Datenmengen erzeugt.
- Es können Strukturen generiert werden, die man aus Zeitgründen manuell nicht zu errechnen bereit wäre.
- In Verbindung mit Zufallsalgorithmen oder manueller Variation der Eingabewerte wird experimentelles Arbeiten effizient.

- Es hat Vorteile, wenn das Partiturformat „human-readable“ ist, wie im Fall von CSound. Der Komponist kann das Ergebnis seiner Programmierarbeit auf Fehler prüfen, bevor es zur Synthesephase gelangt.

Leider ist Partitursynthese häufig als Einbahnstraße ausgelegt: Wenn die Score-Daten einmal erzeugt sind, fehlen oft die Mittel zur Korrektur (Analyse, Montage). Kleinere Korrekturen sind manchmal, im Fall von Text-Scores, noch manuell mit einem Texteditor möglich. Seltener können Korrekturen an der Partitur algorithmisch oder grafisch vorgenommen werden. Dazu müsste man die Partitur zumindest parsen und ggf. grafische Hilfsmittel entwickeln.

Ein weiteres Kriterium ist das kontextfreie Zielformat Audiodatei. Man programmiert eine Score, diese wird weitergeleitet an eine Klangsynthese, und heraus kommt eine Audiodatei, die liegt in einem Verzeichnis irgendwo im Filesystem. Wenn man das mehrmals macht, hat man hinterher einen Ordner voller Klänge, die man dann sehr bequem im Audio-Sequenzer hintereinander und übereinander montieren kann. Eine völlig legitime und weit verbreitete Arbeitsweise, die meines Erachtens nicht unwesentlich von der Logistik der Software beeinflusst ist. Die Tatsache, dass eine Audiodatei keinerlei Information darüber enthält, wann und wo und wie sie abgespielt werden soll, und Programme wie CSound nun mal Audiodateien erzeugen, legt nahe, dass die Prozesskette Partitursynthese -> Klangsynthese weitgehend ohne konkrete Hintergedanken zum späteren Kontext des entstehenden Klangresultats betrieben wird. Beweisen kann ich das leider nicht, es wäre jedoch interessant, festzustellen, wie sich die Arbeitsweise mancher Komponisten verändern würde, könnten sie ihre Partiturfragmente direkt in den Audio-Sequenzer hinein synthetisieren. Um wieder auf das Grain-Beispiel zurück zu kommen:

- Fall A: Das könnte zum Beispiel so aussehen, dass Scheme eine Score-Datei erzeugt mit 200 Grains in einer Sekunde, CSound macht daraus eine Audi-

odatei. Dann sagt Scheme dem Audio-Sequenzer „Erstelle eine neue Spur und setze diese Datei an die Stelle 14 Minuten 20 Sekunden“. Letzterer Arbeitsschritt, das Importieren, wäre damit auch automatisiert, was im *Einzelfall* keinen wesentlichen ergonomischen Vorteil bringt.

- Fall B: Anders sieht es aus, wenn Scheme CSound veranlasst, jeden Grain einzeln zu berechnen, und dann dem Audio-Sequenzer sagt, wie diese 200 Dateien im Arrangement auf 50 Spuren in der Zeit zu platzieren sind. „Ja, wer will denn so viele einzelne Grains in seinem Arrangement sehen?“ hört man es aus dem finsteren Walde ächzen. Darauf könnte man antworten, dass Masse / Dichte ja nicht unbedingt statisch sein muss. Was ist, wenn es am Anfang nur wenige, längere Samples sind, die dann allmählich immer mehr werden und sich zu einer Vielzahl von kurzen Grains verdichten? Dann will man unter Umständen die ersten fünf oder zehn fein säuberlich als einzelne Regionen mit der Pinzette bearbeiten - mit Equalizer, Panorama und Effektautomation - während einem mit zunehmender Dichte und Kürze die späteren Einzelsamples kontinuierlich egaler werden und man lieber alle 50 Schichten mit einer großen Schaufel traktieren möchte. Dass mir bei Sequenzern die Abstraktionsmechanismen fehlen, hatte ich ja schon erwähnt. Fall B wäre wohl noch in Ardour realisierbar, es würde nur sehr hässlich aussehen, und man wäre bestrebt, das Projekt schnell zu Ende zu bringen, um nicht dauernd diese vielen hässlichen Spuren sehen zu müssen.

- Fall C: Oder man *betrügt* und realisiert „die Wenigen“ und „die Vielen“ auf unterschiedliche Weisen. Sofern sich irgendwo die Grenze ziehen lässt, und nicht andere Gründe ein Kontinuum erzwingen, ist es natürlich kein Problem, die ersten zehn Samples als Einzeldateien und den Rest zusammen als eine Datei zu erzeugen. Was aber, wenn es gerade doch ein kontinuierlicher Verlauf sein muss?

Das ist natürlich ein sehr spezielles Beispiel, aber es macht deutlich, dass irgendwo eine Grenze gezogen werden muss, wo Synthese aufhört und Montage anfängt, denn man will natürlich das beste aus beiden Welten. Diese Entscheidung sollte man sich als Komponist nicht von der Software abnehmen lassen, sofern man Wert darauf legt, im Netzwerk des Kompositionsprozesses eine leitende Position einzunehmen.

Metapartitur Jeder Synthesevorgang basiert auf bestimmten Eingangsparametern. Die Gesamtheit der Eingabeparameter für einen Partitursynthesevorgang möchte ich als Metapartitur bezeichnen. Vielleicht könnte man sagen, dass sich die Metapartitur zur Partitur verhält wie das Leadsheet zum ausformulierten Arrangement. Sie definiert bestimmte Regeln, an die die Partitur sich zu halten hat. Gemäß einem musikalischen Syntheseparadigma *„Wir machen aus einfachen Daten komplizierte Daten“* ist die Metapartitur ein sehr überschaubarer Datensatz. Meistens so überschaubar, dass sich seine Synthese per Meta-Meta-Partitur in der Regel schon nicht mehr lohnt, also werden diese Daten häufig von Hand eingegeben. Zum Beispiel als Kommandozeilenparameter, in einem speziell dafür eingerichteten Textformat oder hart kodiert in einem Partitursyntheseprogramm, dessen Quellcode dauernd verändert wird.

Noch stärker als die Partitur neigen diese Daten meiner Beobachtung[17] nach zur Flüchtigkeit. Je nach Arbeitsweise geht die Verbindung unterwegs verloren, welche Partitur- oder Audioresultate von welchen Ur-Daten abgeleitet sind. Oft wären zur Dokumentation logistische Maßnahmen nötig, für die man bei der Arbeit „gerade keinen Nerv“ hat, die den kreativen Fluss stören oder die Gefahr bergen, vom eigentlichen Ziel des Schaffens abzulenken. Oder wertvolle Ergebnisse entstehen unerwartet schon beim Debuggen oder beim lockeren Herumprobieren, bevor man sich Gedanken über eine ergonomische Buchführung gemacht hat.

[17] bei meiner eigenen Tätigkeit und auch bei anderen Komponisten

Selten werden die Metapartitur zusammen mit der Partitur dokumentiert (Beispiel: die Superformel im Licht-Zyklus von Karlheinz Stockhausen). Ob das Sinn macht oder nicht, hängt von der Kausalität der Synthesealgorithmen ab. Spielen Zufallsparameter eine überwiegende Rolle, kann die Metapartitur mitunter auch ein ziemlich wertloser Datenhaufen sein. Ein Spezialfall ergibt sich, wenn Metapartitur und Partitur zwar in einem deterministischen Verhältnis zueinander stehen, die Algorithmen zur Übertragung aber nicht durchschaubar sind. Das wäre zum Beispiel der Fall, wenn in der Metapartitur der Anfangswert für einen Pseudozufallsgenerator (Random Seed) festgelegt ist. Dann wird trotz zufälliger Zusammenhänge aus einer bestimmten Metapartitur immer die gleiche Partitur errechnet, und der Vorgang der Partitursynthese ist rekonstruierbar.

Besteht die Metapartitur nur aus wenigen, streng deterministischen Eingabewerten für ein festgelegtes Programm, kann man diese Werte relativ bequem im Dateinamen des jeweiligen (Audio-)Resultats unterbringen und hat somit einen Anhaltspunkt, falls später ein Ergebnis rekonstruiert oder eine Variation gebildet werden soll. Darüber hinaus verfügen die gängigsten Audioformate über einen Kommentar-Chunk, in dem man theoretisch auch Entstehungsdaten speichern könnte. Das ist aber keine sichere Methode, da dieser Chunk beim Import und Export durch andere Programme leicht verloren geht oder überschrieben wird. Eine einfache Möglichkeit, die sich vor allem bei größeren Datensätzen empfiehlt, ist die Anlage eines Logfiles, anhand dessen Metapartituren und Ausgabedateien einander zugeordnet werden können.

Fazit Sofern Partitursynthese auf vom Komponisten selbst geschriebener Software beruht, liegt es in dessen Verantwortung, ob er erstere als gerichteten oder rückgekoppelten Prozess versteht. Ob so eine Rückkopplung Sinn macht oder nicht, muss im Einzelfall entschieden werden, genau wie die Frage, wie viel Information man bereit ist, beim Verbacken von Partiturdaten zu Klangdaten zu

verlieren. Fakt ist, dass man den Rückweg nicht geschenkt bekommt, so wahr Computerprogramme nicht rückwärts laufen. Synthese macht in der Computermusik normalerweise aus wenigen Daten viele Daten - trotzdem verliert man dabei Information.

1.4.6 SuperCollider

- SuperCollider besteht aus einer modernen Programmiersprache (sclang) und einem Server für die Klangsynthese (scsynth).
- Beide sind interaktiv programmierbar und können voneinander unabhängig betrieben werden.
- Sprache und Server kommunizieren miteinander über OSC-Nachrichten.
- Im Gegensatz zur Partitursynthese mit CSound ist kein menschenlesbares Textformat von Haus aus zwischen Struktur- und Klangsynthese geschaltet, was die Versuchung weckt, ohne eine solche Zwischenschicht zu arbeiten. Wird allerdings der Aufwand betrieben, eine solche Schicht extra dazu zu implementieren, besteht der Vorteil der frei definierbaren Syntax.
- Auch hier ist Echtzeit- und Offline-Betrieb möglich, auch beides gemischt.
- Syntheseprozesse sind direkt in die Programmierumgebung eingebunden. Zum Beispiel kann eine Audiodatei geladen, analysiert, und danach mit den Analysedaten weiter bearbeitet werden. Mit CSound würde das mehrere Programmaufrufe und das Parsen von Analysedaten erfordern.
- Jedes einzelne „Klangobjekt“ ist während seiner gesamten Dauer modulierbar.
- Es können intelligente Programme mit Unix-Integration und grafischer Benutzerschnittstelle erstellt werden.

1.4.7 Visuelle Audioprogrammiersprachen

- Cycling74 Max/MSP
- Pure Data (PD)
- Native Instruments Reaktor
- Plogue Bidule
- IRCAM jMax
- und andere...

Wenn gelegentlich die Formulierung Max/PD auftaucht, dann ist das als Oberbegriff für diese Kategorie von Programmen gemeint[18]. Max/MSP wird auch kurz als Max bezeichnet.

- grafische Programmier- und Benutzerschnittstelle
- Darstellung eines Patches als Flussdiagramm mit „Kisten“ und „Kabeln“
- Kapselung von Programmeinheiten zu Subpatches
- eigene Objekte als „Externals“ in C, Java, Python, Scheme, etc. programmierbar
- große Nutzergemeinde, viele freie Patches und Externals im Internet
- wie bei SuperCollider kein „Hausformat“ für Partituren
- starke Gewichtung auf Echtzeitanwendung

[18]Im Kern konzentriert sich die Betrachtung auf Max/MSP und Pure Data, da ich mit den anderen Vertreter dieser Gattung nur oberflächlich vertraut bin. In wie weit die besprochenen Punkte auch für Reaktor, und andere gelten, muss sich der Leser bei Interesse in Eigeninitiative erkundigen.

- Es können intelligente Programme mit Unix-Integration (bei PD) und grafischer Benutzerschnittstelle erstellt werden.

Vor allem Max und Reaktor sind bei Musikern mit Interesse an neuartigen Klängen und Strukturen sehr beliebt, weil sie auch ohne Programmierkenntnisse Zugriff auf eine große Menge innovativer Musikalgorithmen erlauben. Neben dem Erstellen eigener Patches ist auch die musikalische Auswerwertung fremder Patches gängig. Diese Programme laden durchaus dazu ein, sie einfach einzuschalten und Musik zu machen.

Visuelles Programmieren Das Erstellen und Lesen einfacher Programme, die bereits Klang verarbeiten, ist in der Regel sehr intuitiv. Das macht es vor allem Einsteigern leicht. *Einfach* heißt in dem Fall nicht unbedingt *wenige Objekte*, sondern eher *visuell klar erkennbare Zusammenhänge*. Selbst wenn 100 Objekte in einer Linie hintereinander geschaltet sind, ist es noch ein einfaches Programm, dessen Struktur vom Auge innerhalb von Sekundenbruchteilen verstanden wird. Nennen wir es Programm A. Programm B dagegen besteht aus 5 Objekten und 9 Kabeln, und man kann minutenlang rätseln, was es eigentlich tut.

Fehlerbehandlung Programm A ist ganz einfach zu debuggen. Wenn am Ende (Objekt 100) nicht das heraus kommt, was heraus kommen soll, dann prüft man einfach in der Mitte (Objekt 50). Wenn das Signal bis dahin in Ordnung ist, prüft man ein Stück weiter hinten (Objekt 75), und spätestens nach sieben Versuchen hat man den Fehler eingekreist.

Für Programm B gibt es kaum Strategien. Unter Umständen ist der Ablauf des Programms gar nicht eindeutig anhand des Graphen ersichtlich[19] oder nur schwer. In dem Fall mangelt es an verlässlichen Strategien.

[19]Das kann bei PD passieren, wo sich die Reihenfolge beim Abarbeiten im Zweifelsfall aus der Reihenfolge beim Verkabeln ergibt.

Wie bei jeder Art von Programmierung empfehlen sich die üblichen Methoden zur Fehlervermeidung: eine disziplinierte, vorausschauende Herangehensweise, Bemühung um klar lesbare Strukturierung des Programms, Vermeidung von Redundanz und Mehrdeutigkeiten, aussagekräftige Namensgebung und Kommentare, sinnvolle Kapselung, visuelle Hygiene usw.. Es ist auch möglich, Teile des Programms als „Externals“ in einer Textsprache zu implementieren, wo Programmabläufe durchschaubarer zu formulieren sind. Trotz allem herrscht, denke ich, weitgehend Konsens darüber, dass mit der Komplexität von Max/PD-Patches die Gefahr schwer zu behebender Fehler dramatisch steigt. Daher findet man in diesem Bereich vor allem einfach gestrickte Programme, in denen Max und PD ihre Stärken optimal ausspielen können.

Polyphonie Polyphonie ist einer der großen Schwachpunkte, sowohl bei Max als auch PD. Grundsätzlich existiert jedes Objekt, das man auf der Leinwand platziert, genau einmal. Das macht Sinn, denn so kann man es direkt eindeutig ansprechen. Wenn man zwei von der gleichen Sorte haben möchte, setzt man einfach ein zweites daneben. Ab wie vielen Objekten diese Praxis als „schlechter Stil“ zu gelten hat - manche sagen ab zwei, andere sehen da überhaupt kein Problem - soll hier nicht diskutiert werden, es hängt sicher viel vom Einzelfall ab. Abgesehen von der Einbindung polyphoniefähiger Externals[20] gibt es Mechanismen, gekapselte Max/PD-Subpatches mehrfach zu instantiieren. Die Referenzierung und Steuerung dieser Instanzen von außen kann sich aber durchaus umständlich gestalten, so dass man Polyphonie in Max-Patches auffallend seltener findet als beispielsweise in CSound-Scores.

[20]Max unterstützt z.B. VST-Instrumente