netzspannung.org
Wissensraum für digitale Kunst und Kultur

DE CAMPO, Alberto
ROHRHUBER, Julian

else if – Live Coding, Strategien später Entscheidung

Publiziert auf netzspannung.org:
http://netzspannung.org/positions/digital-transformations
02. Dezember 2004

Erstveröffentlichung: FLEISCHMANN, Monika; REINHARD, Ulrike (Hrsg.): Digitale Transformationen. Medienkunst als Schnittstelle von Kunst, Wissenschaft, Wirtschaft und Gesellschaft. Heidelberg: whois verlags- und vertriebsgesellschaft, 2004.

IMK
Fraunhofer Institut Medienkommunikation

The Exploratory Media Lab
MARS Media Arts & Research Studies

whois

ALBERTO DE CAMPO, JULIAN ROHRHUBER

ELSE IF – LIVE CODING, STRATEGIEN SPÄTER ENTSCHEIDUNG.

Abstract

Wird danach gefragt, wie ein Kunstwerk entstanden ist, so fällt der Blick früher oder später auf die lange Kette von getroffenen Entscheidungen und deren Motivationen. Die Frage, wann sie getroffen werden mussten (oder müssen) wird dabei meist übergangen. Gerade in der Musik spielt dieser Zeitpunkt insofern eine Rolle, als er Musik als eher »improvisiert« oder »komponiert« charakterisiert. In welcher Weise und bis wann es offen bleibt, welche Wege ein Musikstück geht, gehört daher auch zu den Grund legenden Überlegungen und Entscheidungen, die einer Computermusik-Komposition vorausgehen. Obwohl bekanntlich die Mikrostruktur eines jeden Computer-Programms aus unzähligen Entscheidungen besteht, sind diese Wege nicht zu jedem Zeitpunkt zugänglich – Programme sind in der Regel fixiert, bis auf wenige »Hebel«, die den Zugang zu einigen wenigen Entscheidungen für später offen halten.
Im Gegensatz dazu sind einige Programmiersprachen explizit darauf ausgelegt, Entscheidungen möglichst spät zu treffen. Die Techniken des live coding, der Programmierung von bereits aktivem Code, führen dazu, dass dieser offene Zugang die Entscheidungsstrukturen zum Teil künstlerischer Arbeit werden lässt.

Programme und Sprachen

Die Mikrostruktur eines Computerprogramms bildet einen feinen Teppich aus unzähligen Entscheidungen – abhängig von den Spannungswerten bestimmter Register werden weitere Register gesetzt; das Ergebnis, das sich in Folge dieser Wege bemerkbar macht, ist die Konsequenz aus diesem Labyrinth von Verzweigungen und Zwischenstufen.
Aus Effizienzgründen werden einmal errechnete Zwischenergebnisse für den späteren Gebrauch meist gespeichert; man kann sie aber auch noch einmal nachrechnen, wenn das Ergebnis erneut gebraucht wird. Durch die Zugänglichkeit von immer schnelleren Rechnern außerhalb hoch spezialisierter Zentren bietet es sich immer häufiger an, den zweiten Fall dem ersten vorzuziehen, und sich nicht auf gespeicherte vergangene Entscheidungsketten zu verlassen. Diese Alternativen sind nicht nur im Programm verwirklicht, sie finden sich auch in der Computersprache, die zur Beschreibung der Prozesse verwendet wird.
Unter anderem weichen dadurch Sprachen wie Smalltalk, Self oder Forth den Gang des Programms quasi auf, so dass dieses selbst während der Laufzeit noch umgeschrieben werden kann; und dadurch, dass das System sehr viel über sich selbst weiß, können Entscheidungen abhängig davon getroffen werden. Dieses Konzept ist in vielen Situationen zu riskant – offensichtlich können Fehler in einem solchen System auch erst dann auftauchen, wenn es benutzt wird. Je offener die Struktur, je später die Entscheidungen, desto mehr solcher Laufzeitfehler sind möglich, die nicht schon beim Kompilieren gefunden und entfernt werden können.
Für die künstlerische Arbeit ist es dennoch interessant: Ähnlich wie bei einer Improvisation oder einem Fluxuswerk muss die Bereitschaft zunehmen, sich unvorhersehbaren Konsequenzen zu stellen. Was das Computerprogramm als Werkzeug verfestigt, nämlich die Investition an Zeit, die später die Verfertigung von geschlossenen »Werken« effizienter und sicherer machen soll, entfällt und macht den Blick frei auf eine Ebene, in der die Sprache selbst Teil des künstlerischen Prozesses wird.
Unter diesen Bedingungen wird die Programmiersprache zur Umwelt, in der die Entscheidungen, welche Veränderungen welche Auswirkungen haben, immer neu getroffen werden können. Das macht den traditionellen Zugang zur Software-Entwicklung weniger interessant als die Technik der späten Entscheidung.

Musik

Musik ist in diesem Zusammenhang besonders aufschlussreich, weil sie idealtypisch Zeit gebunden ist; hier gewonnene Erkenntnisse scheinen auf andere algorithmisch formulierbare Kunstformen durchaus übertragbar. Das Ideal des Komponisten aus der Zeit der seriellen Musik ist, möglichst alles vorwegzunehmen, am besten aus einem konzeptionellen Bauplan abzuleiten; etwa die Gestaltung sämtlicher Arbeiten Stockhausens aus der Entfaltung seiner 1977 gefundenen »Superformel« demonstriert dieses Ideal prototypisch. Die klassische elektronische Studiokomposition und ihre übliche Aufführungsform folgen diesem Modell auch weitgehend: Das erklingende Material ist vollständig durchkomponiert, der zeitliche Ablauf fixiert, lediglich die räumliche Gestaltung und – meist in geringerem Maß – die Dynamik wird für die Aufführung live interpretiert.
Demgegenüber steht die Tradition der Improvisation, die das Üben als lebenslange Vorbereitung auf die ideale Flexibilität im Moment des Spielens versteht; inwieweit diese Flexibilität eingesetzt wird, um codifizierte traditionelle Regelsysteme mit Leben zu erfüllen – wie etwa in der indischen Musik – um alle Rückbezüge auf existierende Musik zu vermeiden, oder, um mit möglichen neuen Querverbindungen zu spielen, ist dabei sekundär; das Interessante für das Publikum ist das unmittelbare Miterleben des kreativen Prozesses. Dabei ist Virtuosität lediglich ein Faktor von vielen; schon die Kommunikation selbst ist interessant zu verfolgen und die diskursive Entwicklung von Ideen kann ganz analog zu einem interessanten Gespräch verlaufen.

Aktuelle Systeme

SuperCollider3[1] ist eine open source Programmierumgebung, deren Interpreter-Sprache Elemente funktionaler und Objekt orientierter Sprachen wie Smalltalk, Lisp, Scheme oder auch HMSL implementiert. Sie verbindet eine große Bibliothek von Unit Generators mit Strukturen zur algorithmischen Komposition zu einem erweiterbaren System, das zur Laufzeit verändert werden kann. Sie kann bereits auf eine Entwicklungszeit von 10 Jahren zurück blicken.

Just In Time Library, JITLib,[2] ist eine für SuperCollider3 verfasste Bibliothek, die die Konzepte dieser Programmiersprache für live coding weiterentwickelt und in Rechner-Netzwerken zugänglich macht.

BreakBeatCut Library[3] ist speziell für live coding im break beats-Stil gedacht und wird u.a. von Nick Collins und Fredrik Olofsson[4] eingesetzt.

ChucK [5] ist eine von Grund auf neu geschriebene Sprache, die dieselben Ziele wie SuperCollider verfolgt; sie ist jedoch noch nicht so umfangreich und ausgereift.

Zwei Arbeitsweisen
JITLib für Film

Die in der JITLib entwickelte Technik zur Programmierung laufender Programme wurde unter anderem im Filmprojekt von Volko Kamensky »Alles was wir haben« erarbeitet und erprobt. Der Film untersucht die Konstruktion von Heimatverbundenheit, indem er scheinbar beliebige Bilder einer norddeutschen Kleinstadt mit scheinbar dokumentarischen Klängen versieht, die jedoch rein synthetischen, algorithmischen Ursprungs sind.
Die Vorgehensweise, sich in gemeinsamer Arbeit durch Konversation und Programmierung schrittweise an Portraits imaginierter akustischer Phänomene anzunähern, führt zu einem interessanten Kreislauf der Beeinflussung zwischen der schriftlichen Repräsentation des Programmtextes in seiner potentiellen Veränderlichkeit, dem sich parallel verändernden Klangbild und der gleichzeitigen Konversation darüber. Zum Beispiel die Vorstellung, wie ein Rasenmäher oder eine Windböe klingen sollen, überkreuzen sich in diesem Verfahren der real sound synthesis mit den

Wegen, die die Entwicklung eines Programms geht – mit mal äußerst unerwarteten, mal äußerst vorhersehbaren Resultaten.
Die Möglichkeit, das Programm während der Laufzeit umzuschreiben, erlaubte uns die Wahrnehmung von Differenzen, die für diese Arbeit essentiell wurde und die zu Klängen führte, die weniger Ergebnis eines physikalischen Modells als das eines perzeptuellen und konversationellen Prozesses sind. Auch wenn dem Betrachter des Films diese Vorgehensweise nicht mehr präsent ist, meinen wir, dass sie sich im Charakter der Tonspur niederschlägt und rekonstruiert.

Warteraum/PowerBooks_UnPlugged

Aus dem »Warteraum1«-Seminar[6] ist das Ensemble »PowerBooks_UnPlugged« hervorgegangen. Seine Mitglieder sind: Alberto de Campo, Echo Ho, Hannes Hölzl und Jankees van Kampen. Bei »PowerBooks_UnPlugged« wird die Verwendung des Laptops als vollständiges Instrument konsequent weiter geführt, u.a. durch bewussten (ausschließlichen) Einsatz der internen Lautsprecher der Mobilcomputer, Sprachsynthese aus dem Betriebssystem, und nicht zuletzt durch live coding.
Die Praxis wird durch JITLib ermöglicht: Es gibt einen gemeinsamen Pool von Dateien mit Programmtext für die Performances. Jede Datei besteht aus einem oder mehreren kleinen Tasks, die Klang-Texturen erzeugen; das können Streams aus einzelnen (noten-artigen oder granularen) Ereignissen, komplexe kontinuierliche Syntheseprozesse oder Mischformen sein. Manche dieser Texturen haben fixierte Anfangs- und Endzeiten, so dass die Musik aus Überlagerungen dieser Texturen entsteht, und alte Texturen von selbst enden; andere werden ausgetauscht oder modifiziert, während sie laufen. Neu entdeckte interessante Modifikationen können beim Spielen über chat-fenster als Code ausgetauscht werden, genauso wie andere Mitteilungen wie zum Beispiel Absprachen, wie weiter vorgegangen werden soll.
Diese Variationen und andere während des Spielens formulierte neue Ideen können auf Wunsch in neuen Dateien gesammelt und Teil des code pools werden. Das implizite Arbeitsmodell ist ebenso demokratisch und symmetrisch wie auch die räumliche Disposition der Musik: Jede/r kann alle generierten Klänge auf dem eigenen, auf einem fremden, oder (sequentiell oder quasi-simultan) auf allen Rechnern klingen lassen. Die dabei entstehenden Unsicherheiten gehören zu den (für uns) reizvollsten Folgen von live coding.[7]

Organisation

Im Laufe des Symposions »changing grammars«, Hamburg 2004, hat sich TOPLAP formiert, die »Temporary Organisation for the Proliferation for Live Algorithm Programming« – so eine der möglichen Interpretationen des Akronyms[8]. TOPLAP diskutiert live coding in einer mailing list und auf einem Wiki; eine gute Übersicht über die Standpunkte der teilnehmenden KünstlerInnen.[9]

Literatur

Collins, Nick/Alex McLean/Julian Rohrhuber/Adrian Ward: Live Coding Techniques for Laptop Performance, in: Organised Sound 8/3 (2004), S. 321-330.

1 James McCartney et al; http://supercollider.sourceforge.net

2 Julian Rohrhuber http://supercollider.sourceforge.net

3 Nick Collins http://www.sicklincoln.org

4 http://www.klippav.org

5 Ge Wang, Perry Cook http://chuck.cs.princeton.edu

6 Das Seminar wurde von Julian Rohrhuber und Alberto de Campo, an der Kunsthochschule für Medien, Köln im Mai 2003 gehalten.

7 Rohrhuber, Julian/de Campo, Alberto: Uncertainty and Waiting in Networked Sound Synthesis, in: Proceedings of the 2004 International Computer Music Conference, San Francisco [erscheint September 2004].

8 http://www.toplap.org

9 http://www.toplap.org/wiki?Read_Me_Paper