Zur Performativität und Medialität von Code

Reflexionen zu Code aus der künstlerischen Perspektive

shusha Niederberger, 8. Juli 2013

Der Text basiert auf einem Vortrag, gehalten am Workshop „out of control – der performative Spielraum in Kunst und Kybernetik“ 20.4.2013, Corner College, Zürich

Inspiriert wurde ich zu diesem Vortrag von Erfahrungen und Fragestellungen aus meiner eigenen künstlerischer Arbeit und einem Text von Sibylle Krämer. Sie schlägt in ihrem Text „Sprache – Stimme – Schrift: Sieben Gedanken über Performativität als Medialität“ vor, Performativität als Medialität zu denken (Krämer: 2002). Ihr Begriff von Medialität entspricht dabei im Allgemeinen dem Performativitätsbegriff der Kunst- und Kulturwissenschaften, wie er v.a. von den Theaterwissenschaften (v.a. Fischer-Lichte: 2004) formuliert wurde. Im Gegensatz dazu entwickelt ihn Krämer in ihrem Text aus den Sprachwissenschaften. Diesen Umstand finde ich sehr interessant, weil die Sprachwissenschaften mit ihren Vorstellungen von Sprache, Kommunikation und Information massgeblich an der Entstehung der Kybernetik beteiligt waren (Wiener: 1989/[1950]). Und die Kybernetik ihrerseits hat beginnend mit der Erfindung von FORTRAN 1954 eine eigene Art von Sprachlichkeit hervorgebracht: die Programmiersprachen.
Diese formalen Sprachen werden im Allgemeinen innerhalb ihrer Disziplin rezipiert und weiterentwickelt, ihre Performativität scheint recht einfach technischer Natur zu sein und sich auf das Erfüllen ihrer technischer Zwecke zu beschränken. Warum ist dann aber die Rede von „Sprachen“? Warum gibt es so viele unterschiedliche davon, mit ihrer jeweiligen treuen Anhängerschaft? Es scheint also doch etwas komplexer zu sein als aus der instrumentellen Perspektive sichtbar ist.
Ich möchte in diesem Vortrag einen Wechsel in die künstlerische Perspektive vorschlagen. Aus dieser heraus frage ich, wie die Medialität von Programmiersprachen beschrieben werden kann. Ich werde dafür zwei künstlerische Arbeiten vorstellen, eine von mir und eine von einem isländisch/amerikanischen Künstler. Anhand dieser Arbeiten und Erfahrungen aus meiner künstlerischen Arbeit möchte ich versuchen, einige Aussagen zur Medialität von Programmiersprachen zu formulieren.

Ich habe in meiner künstlerischen Arbeit wiederholt mit Programmiersprachen als sprachliche Objekte gearbeitet. Dabei muss ich gleich zu Beginn eine wichtige Differenzierung machen. Meine künstlerische Arbeit unterscheidet sich grundlegend von der generative Perspektive, d.h. der rein instrumentelle Einsatz von Programmierungen. Meine Arbeitsweise steht im

Gegensatz dazu der Software Art nahe und muss im Kontext der Konzeptkunst gesehen werden.[1]
Was habe ich also aus meiner künstlerischen Arbeit über Programmiersprachen erfahren? Als erstes wäre der erstaunlich grosse Widerstand des Materials zu nennen, auf den ich gestossen bin. Diese Widerständigkeit möchte ich hier als einen Anteil an der Medialität von Programmiersprachen beschreiben.

Meine Arbeit **„der Raum der abwesenden Dinge“** von 2011 ist eine ortsspezifische Installation. Ich möchte die konzeptionelle Anlage zum besseren Verständnis kurz erläutern: Das Zimmer befindet sich in einem erst kürzlich renovierten historischen Gebäude, das seine Geschichtlichkeit in allen Räumen inszeniert – ausser in diesem einen. Hier ist alles neu, aber behauptete eine Historität, die sich bei genauerer Betrachtung als synthetisch herausstellt. Auf Nachfrage habe ich erfahren, dass in diesem Raum die Küche untergebracht war.

Bild: Einladungskarte zur Ausstellung

Die absolute Abwesenheit dieser Küche in der inszenierten Authenzität des Raumes ist frappant. Ich habe diese anwesende Abwesenheit in eine Beschreibung umgesetzt, und dafür eine künstliche Sprache verwendet (Java/Processing). Der Code ermöglicht es mir, von an- und abwesenden Dingen gleichzeitig zu sprechen, also einen medialen Raum zu eröffenen. Ausserdem konnte ich dabei die sprachlichen Möglichkeiten von Code in einer konkreten räumlichen Situation ausprobieren. Die Anlage dieser Arbeit ist also von zwei Interessen getragen: einerseits aus einem Interesse für die spezifische Situation vor Ort, andererseits einer Fragestellung an das Medium der Programmiersprachen. Und weil Programmiersprachen nur in ihrem technologischen Zusammenhang als solche existieren, war

[1] zur Abgrenzung von Software Art von Generative Art siehe Arns: 2004

es mir sehr wichtig, dass der Code funktionsfähig ist (er compiliert ohne Fehler, tut aber nichts). Es ist kein symbolischer Code, damit hätte ich meine Fragestellung verlassen, sondern eher ein absurder Code.

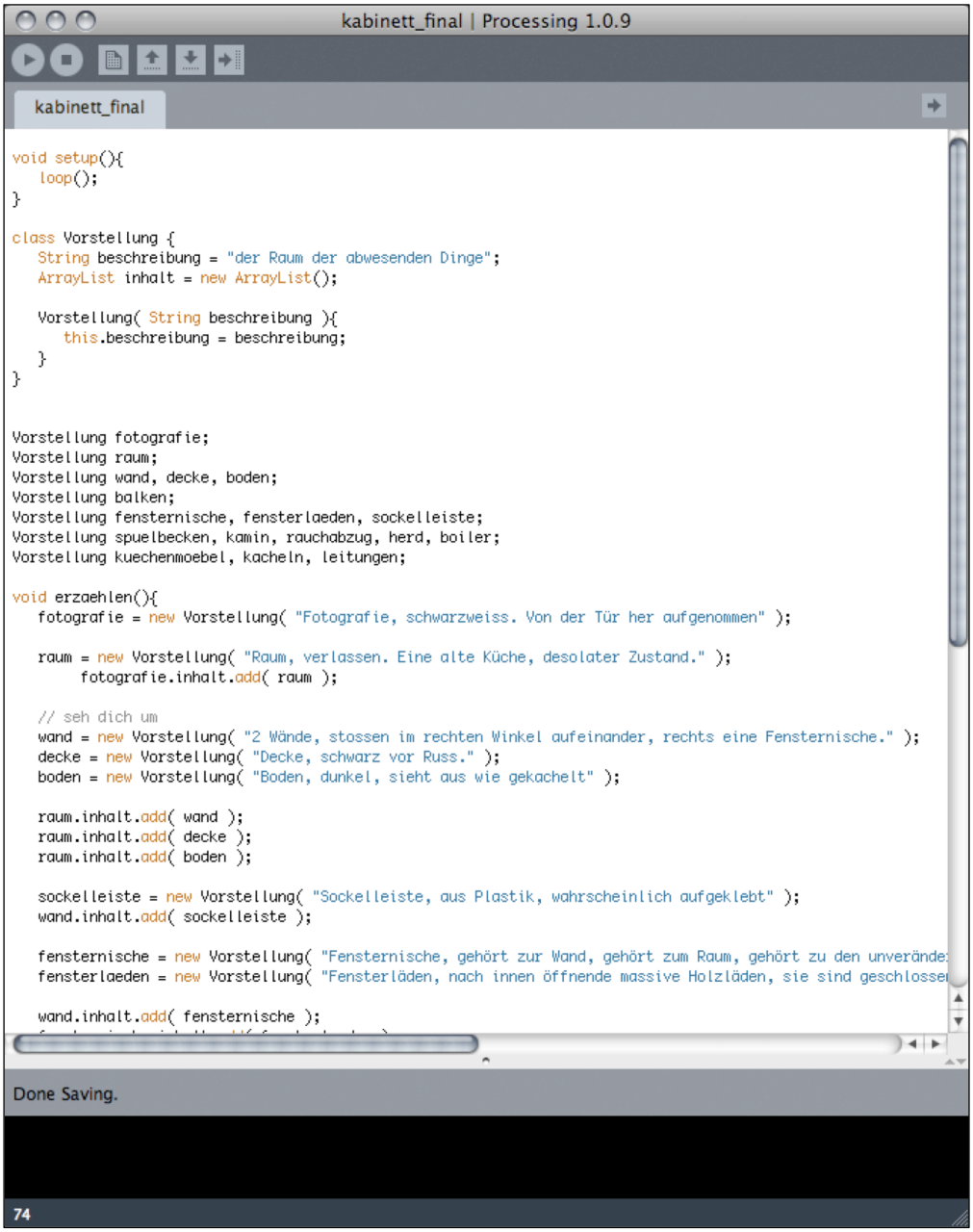


Bild: Screenshot aus der Entwicklungsumgebung von Processing auf den Anfang des Codes.

Die verwaltende Sprache

Bei der Arbeit habe ich festgestellt, dass man mit Programmiersprachen erstaunlich wenig sagen kann. Man kann eigentlich fast gar nichts sagen. Das einzige, was sprachlich funktioniert, sind das *Festlegen und Verarbeiten von Verhältnissen*.
Das hat einerseits damit zu tun, dass Java (die Programmierspache, in der der Code geschrieben ist) zu den sogenannten „objektorientierten Programmiersprachen“ gehört. Das

objektorientierte Programmieren gehört zum aktuellen Paradigma des „structured programming. Aus der historischen Perspektive ist diese Konzeption von Programmierung der aktuelle Stand einer langen Entwicklung von der phyischen Maschine und den manuellen Tätigkeiten an ihr zu einer linguistischen Praxis. Diese Entwicklung kann als einen kontinuierlichen, wiederholten Prozess der Abstraktionen beschrieben werden. Ich möchte hier gerne etwas darauf eingehen.

Einer der ersten Computer war der ENIAC (Electronic Numerical Integrator And Computer), der 1946 an der Universität Pennsylvania der Öffentlichkeit vorgestellt wurde und bis 1955 betrieben wurde. Das Programmieren des ENIACS war eine manuelle Aufgabe: die Verbindungen zwischen den Elementen wurden mit einem Kabel von Hand gesteckt (wie bei den Telefonanlagen) und die gewünsche Operation (Addition, Subtraktion, Multiplikation oder Division) per Drehregler eingestellt. Als „Programmer“ wurden die Menschen (meist Frauen) bezeichnet, welche diese manuelle Arbeit verrichteten. Sie standen ganz am Ende (und hierarchisch zuunterst) einer Arbeitsteilung, welche die zu berechnenden Probleme in drei Schritte aufteilten: der „Analyst“ entwickelte die Logik eines Programmes, welche anschliessend vom „Coder“ in mathematische Schritte übersetzt wurde, diese wurden dann von den „Programmer“ in die physische Logik der Maschine übertragen.

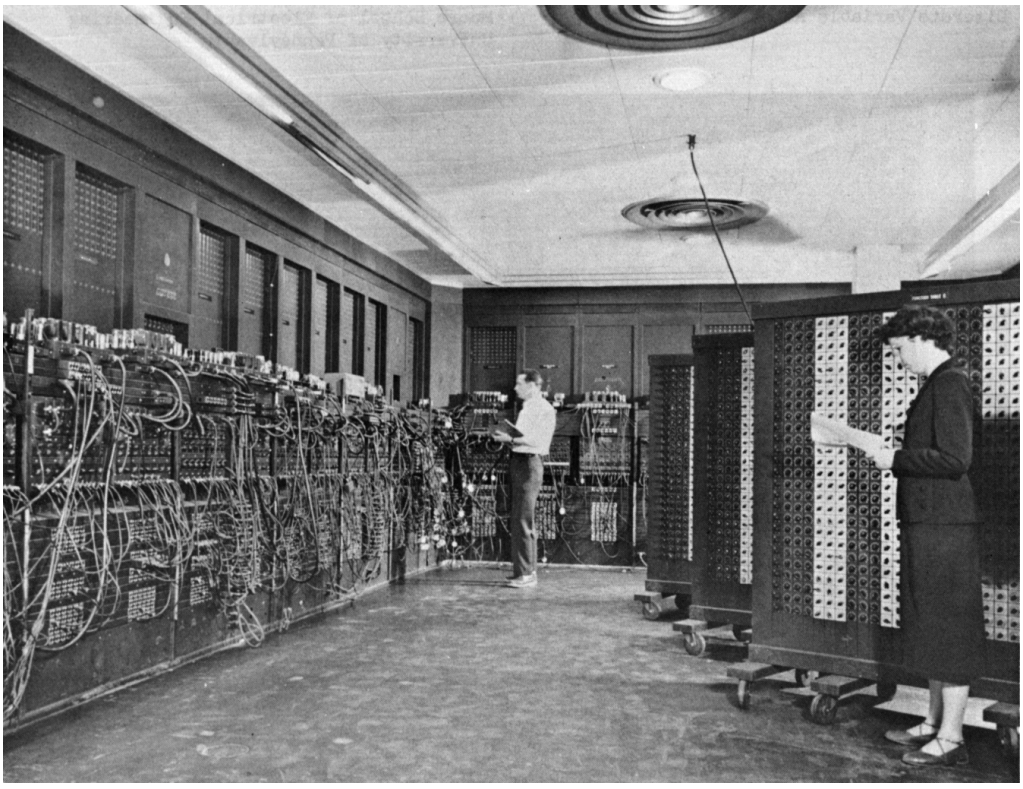


Bild: Glen Beck (background) and Betty Snyder (foreground) program the ENIAC, Ballistic Research Lab, Aberdeen, 1947-1955. Quelle: wikipedia

Mit dem Aufkommen der ersten grösseren Speichermodule wurde es möglich, häufig verwendete Programmschritte zu speichern und auf Abruf wiederzugeben bzw. auszuführen. Diese ersten Programme hiessen „Pseudo Codes“, ihre Erstellung wurde „automatic

Programming“ genannt. Das Programmieren direkt an der Maschine wurde in Abgrenzung dazu als „programming proper“ bezeichnet. Auf dieser Stufe ist der Abstand zur Maschine in den Begriffen noch benannt. In einem nächsten Abstraktionsschritt wird die Maschine aber ganz verlassen, und mit dem „structured programming“ die Steuerung der Maschine ganz auf sprachlicher Ebene gefasst. Die Programme heissen jetzt „Source Codes“, ein Begriff, welcher die Maschine begrifflich unterschlägt und die linguistische Praxis als Quelle jeder Tätigkeit der Maschine einsetzt.

	Logik	Mathematik	Maschine
ENIAC	Analyst	Coder	Programmer
Automatic programming	Analyst	Programmer	Pseudo Code
Structured Programming	Programmer	Source Code	

Dieser zunehmende Abstand zur Maschine wird von Hartmut Winkler als „Abspaltung von der Praxis bezeichnet“ (Winkler: 1998), und Wolfgang Hagen weist darauf hin, dass Computer als Medien erst mit diesem Übergang des Zugangs zur Logik in die Sprache möglich wurde (Hagen: 2006). Dieser Abstraktionsprozess von Tätigkeiten an Maschinen hin zu rein sprachlich-logischer Repräsentation äussert sich auf sprachlicher Ebene in einer „Verwaltungs“-tätigkeit. Josef Weizenbaum sagte:

> „Programmieren ist eher eine Erfindung von Bürokratie als eine Erfindung von Maschine“ (zitiert nach Chun: 2011)

Und man möchte anfügen: eher auch als eine Erfindung von Sprache. Der Begriff „Programmiersprache“ ist irreführend: es handelt sich nicht um Sprachen, auch wenn sie als Set von Ausdrücken und Regeln daherkommen.

Programmiersprachen sind also das Ergebnis eines langen Abstraktions- und Formalisierungsprozesses. Hartmut Winkler nennt diesen Prozess ein „Umschlagen von Ereignis in Struktur“. Und er führt weiter aus, dass diese Schemenbildung sich „auf ihrem eigenen Terrain inszeniert.“ (Winkler: 1998). Damit meint er, dass die Programmiersprache die Abstraktions- und Automatisierungsprozesse, aus denen sie hervorgegangen sind, in ihrer eigenen sprachlichen Anlage abbilden. Programmieren ermöglicht es, Strukturen möglichst knapp zu formulieren: die Ausdehnung einer Schleife in der Ausführung bildet sich in ihrer Notation eben gerade nicht ab. Und in dieser Knappheit der Formulierung zeigt sich die Beschränkung des sprachlichen Ausdrucks, welche der technischen Leistungsfähigkeit (Eindeutigkeit und Geschwindigkeit) geschuldet ist.
Was Winkler „Inszenierung auf dem eigenen Terrain“ nennt, kommt dem nahe, was in den Kulturwissenschaften mit Performativität gemeint ist. Auch Programmiercodes sprechen von

mehr, als die technische Funktionalität verlangt. Aber um dieses Mehr zu vernehmen, muss man die Codes sehen und lesen können – und dies ist meistens nur einem kleinen Kreis von Fachleuten möglich, und nur zu einem gewissen Punkt im Herstellungsprozess.

Damit bin ich beim zweiten Punkt, den ich unter der Medialität von Programmiersprachen vorschlagen möchte:

Die Flüchtigkeit von Code

Code ist immer auf seine Ausführung hin geschrieben[2]. Denn Code ist nicht dasselbe wie das Programm – dies ist eine wichtige und oft vernachlässigte Unterscheidung. Ein Code wird mittels einer Übersetzungsleistung in ein Programm überführt. Das Programm selbst ist die reine technische Performativität. Programme bestehen aus Maschinencode, der für Menschen im allgemeinen nicht lesbar ist[3]. Dieser Maschinencode bezieht sich im Gegensatz zum Source-Code spezifisch auf eine bestimmte Art von Maschine bzw. auf den Prozessor, auf dem das Programm laufen soll. Die Übersetzung in ein Programm geschieht entweder beim Compilieren oder bei Scriptsprachen in Echtzeit durch einen Interpretor. Bei der Übersetzung gehen alle sprachlichen Eigenschaften dies Codes verloren, die Logik wird wieder mit der Maschine enggeführt und verliert im Gegenzug die Lesbarkeit, Weiterverwendbarkeit und Flexibilität des Codes. Wenn wir uns erinnern, durch welche Schritte der Abstraktion die sprachliche Gestalt von Code gebildet wurde, wird die Dynamik des Übersetzungsprozesses sichtbar: in ihm implodiert der in den beschriebenen Abstraktionsschritten über Jahrzehnte aufgebaute und in Sprache abgelegte Abstand zur Maschine in kürzester Zeit. Über diesen Schritt zurück von der Abstraktion zum Ereignis, diese Implosion des Abstands, diese „Wiederkehr des Fleisches“ (nochmal Winkler) wissen wir sehr wenig, was angesichts der Verbreitung der programmierten Medien doch etwas erstaunlich ist.

Ich habe bisher darüber gesprochen, die Medialität von Programmiersprachen und der in ihnen notierten Codes als Ergebnis des Abstraktions- und Formalisierungsprozess ihrer historischen Entwicklung zu sehen. Als zweiten Punkt habe ich die Flüchtigkeit von Codes besprochen, die immer drauf und dran sind, etwas anderes zu werden: ein Programm, das den Bereich des Sprachlichen schon wieder verlassen hat.

Die künstlerische Arbeit mit Programmiersprachen und Codes als sprachliche Objekte bedeutet also das Arbeiten in einem Vakuum: zwischen Abstraktion und Sprache sowie zwischen Notation und Ausführung. Das Herauslösen der Codes aus der Ausrichtung auf eine technische Ausführung als Programm kann auch als eine Übersetzung des

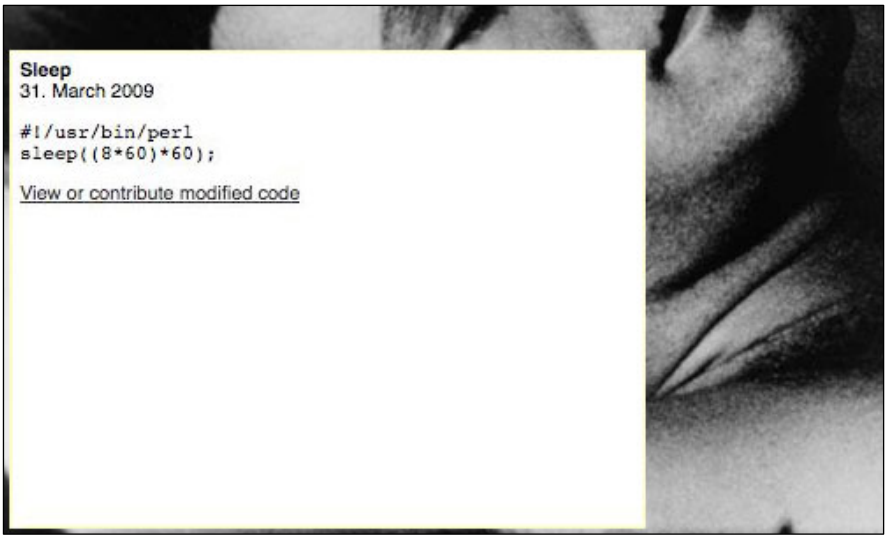
---

[2] das schliesst die Kommentare mit ein, welche als Strukturierungs- und Memorierungshilfen in den Code hineingeschrieben werden. In Kommentaren bildet sich eine Haltung zu Arbeit und Produktion ab, die ebenfalls auf ein Produkt zielen.

[3] auch von den meisten Programmierern nicht, obwohl man natürlich auch Maschinencode lesen und schreiben lernen kann.

Medialisierungsschrittes der Compilierung/Interpretation in einen Rezeptionszusammenhang in der Kunst gedacht werden.

Zum Abschluss möchte ich noch eine Arbeit von Pall Thayer zeigen. Es ist ein Beispiel aus der Reihe Microcodes: **Sleep, 2009**. Die Microcodes wurden von Thayer über seine Webseite und über Mailinglisten distribuiert, sind gegenwärtig leider aus mir nicht bekannten Gründen offline.



Quelle: Interview mit Pall Thayer auf CONT3XT.NET
http://cont3xt.net/blog/?p=1743 (last access: 10.6.2013)
Der Hintergrund zum Code wurde mutmasslich von der Redaktion eingefügt,
da alle weiteren mir bekannten Microcodes reine Textdokumente sind.

Betrachten wir den Microcode. Als erstes fällt auf, dass der Microcode aus verschiedenen typografisch gekennzeichneten Textsorten besteht: zuoberst stehen ein Titel und eine Datierung, die dem Genre der Werkbeschriftung nahe stehen. Dann folgt der eigentliche technische Code, er ist in einer monospace-Schrift gesetzt, wie es in Programmierumgebungen der Übersichtlichkeit halber üblich ist. Der Code beginnt mit einer Adressierungszeile, die mit „#“ anfängt, hier wird der Pfad zum Interpretor angegeben, also der Instanz, welche für die Übersetzung in Maschinencode und ein lauffähiges Programm zuständig ist. Mit der Nennung des Interpretors wird einerseits der Abstand zur Maschine benannt, andererseits wird auch die verwendete Programmiersprache ersichtlich: /perl. Perl ist wie Java eine objektorientierte Programmierspache, sie ist für ihre Nähe zu natürlichen Sprachen bekannt.
Die zweite Code-Zeile ist der eigentliche Code. Er ist sogar für Nicht-Programmierer lesbar: sleep ist die Anweisung zu schlafen (zur Aussetzung der Ausführung allfälliger weiterer Anweisungen), und zwar 8 mal 60 Minuten mal 60 Sekunden, also 8 Stunden lang.
Am Ende des Microcodes ist ein Link aufgeführt, der zur Webseite führt. Der Link fordert dazu auf, selber Microcodes zu schreiben und zur Sammlung beizutragen. Es ist ein Verweis auf die Kultur der OpenSource- Bewegung und deren Kultur des sharings von Codes, der

gemeinsamen Arbeit am Code, also der Vorstellung von Code als kulturellem Gut, welches allen gehört.

Der Code selber ist funktionsfähig, jeder kann mit einem Perl-Interpretor (der für alle gängigen Betriebssysteme gratis erhältlich ist) den Code ausführen und das Kunstwerk in seiner technischen Performativität beobachten. Damit eröffnet Payer eine zweite Rezeptionsform, die er wie im Interview erwähnt, auch explizit mitdenkt. Der Code ist also sowohl für die sprachliche als auch die ausführende Ebene konzipiert. Diese doppelseitige Anlage entspricht auch der inhaltlichen Ebene. Schlafen ist eine Tätigkeit, die wir zwar vollziehen, aber dabei nicht beobachten können. Wir können nur den Schlaf der anderen beobachten. Und erst in dieser Beobachtung entfaltet sich die Zeitlichkeit des Schlafs, wird das Schlafen als Handlung sichtbar. Die Arbeit ist eine Referenz auf das Sleep-Video von Andy Warhol (1963), in dem er seinen Freund beim Schlafen filmt. Das Video dauert 8h (die durchschnittliche Zeit, welche ein Mensch täglich schläft). Das Video von Warhol und der ausgeführte Microcode erlauben es dem Betrachter, eine Perspektive einzunehmen, welche Schlafen als Tätigkeit sichtbar macht. Denn auch das Programm ist nicht einfach inaktiv, sondern es arbeitet weiter im Hintergrund und zählt die Tausendstelsekunden, um zur richtigen Zeit wieder aufzuwachen.

Die künstlerische Methode des Herauslösens von Codes aus der Ausrichtung auf eine ausschliesslich technologische Ausführung ermöglicht, dass der in den sprachlichen Codes abgelegte Bedeutungsüberschuss (welcher Programmiersprachen überhaupt zu sprachlichen Objekten macht: ihre Metaphern und ihr Assoziationsspielraum) frei wird, um Verbindungen mit aussersprachlichen und aussertechnologischen Referenzen einzugehen. Diese Verbindungen können entfaltet und gestaltet werden.
Der Code spricht von mehr als nur von Funktionalität.

Meine Frage wäre abschliessend, inwiefern die Medialität (in Umkehrung zu Krämer: als Performativität) von Code nicht auch in ungebrochenen technologischen Settings wirksam ist, sich hinter der oberflächlichen Funktionalität noch andere Strömungen und mikropolitische Wirksamkeiten verbergen.

Vielen Dank.

Referenzen:

- Arns Inke: *„Code as performative speech act“*, in: Artnodes - e-journal on art, science and technology, 2005/7
- Chun Wendy Hui Kyong: *„Programmed Visions: Software and Memory“*, MIT Press, 2011
- Fischer-Lichte Erika: *„Ästhetik des Peformativen“*, Suhrkamp 2004
- Hagen Wolfgang: The Style of Sources: Remarks on the Theory and History of Programming Languages, in: Chun Wendy Hui-Kyong / Keenan Thomas: *„New media, old media: a history and theory reader“*, Routledge 2006
- Krämer Sibylle: *„Sprache – Stimme – Schrift: Sieben Gedanken über Performativität als Medialität“*; in: Uwe Wirth: Performanz, Suhrkamp 2002
- Pflüger Jörg: „Writing, Building, Growing: Leitvorstellungen der Programmiergeschichte“, in: Hans Dieter Hellige: *„Geschichten der Informatik: Visionen, Paradigmen, Leitmotive“*, Springer 2004
- Wiener Norbert: *„The Human Use of Human Beings; Cybernetics and Society“*, Free Association Books, 1989 (1950)
- Winkler Hartmut: *„Über Rekursion. Eine Überlegung zu Programmierbarkeit, Wiederholung, Verdichtung und Schema.“* Vortrag an der Ars Electronica Linz 1998, online: http://homepages.uni-paderborn.de/winkler/rekursio.html (last access: 10.6.2013)

Erwähnte Kunstwerke:

- Niederberger shusha: *„der Raum der Abwesenden Dinge“*, 2011, online: http://www.shusha.ch/?page_id=827 (last access: 10.6.2013)
- Pall Thayer: *„Microcodes: Sleep, 2009*“, gegenwärtig offline (http://pallit.lhi.is/microcodes/)
  Interview mit Pall Thayer auf CONT3XT.NET, Januar 2011, online: http://cont3xt.net/blog/?p=1743 (last access: 10.6.2013)