

Code as performative speech act

Inke Arns (Berlin)

Lecture held at the Read_Me conference, Aarhus/DK, August 23, 2004

I dedicate this text to Gereon Schmitz († Sept 2004).

Introduction

Software art describes an artistic activity which in the medium – or rather: the material – of software allows for a critical reflection of software (and its cultural impact). It thus makes visible the aesthetic and political subtexts of seemingly neutral technical commands.

In this lecture I will argue that in the context of software art a far more interesting notion than the »generative« is that of the »performativity« of code. This notion – borrowed from the context of speech act theory – does not only comprise the ability to generate in a technical context, but also encompasses the implications and repercussions of code in the realm of the aesthetic, the political and the social. My lecture proposes the notion of performativity of the code as one of the reasons for contemporary artists' growing interest in using software as an artistic material.

¹ In the beginning God created the heavens and the earth. ² The earth was without form and void, and darkness was upon the face of the deep; and the Spirit of God was moving over the face of the waters. ³ And God said, "Let there be light"; and there was light. (Genesis 1)¶

But let's start at the very beginning (I won't spare you this classic). Before discussing my theses, I'd like to give you three rather historical examples for the performativity of speech or of texts in general: one biblical, one mythological and one example from Russian Futurism - i.e. from the beginning of the 20th century.

According to the Bible, God created the world through speech. God spoke and the world came into being. According to *Sefer Yesira* (Book of Creation), God's "speech" was not talking in the sense of someone speaking, but rather a

manipulation of the letters of the Hebrew alphabet. These letters, the Book of Creation teaches, are not merely linguistic symbols. They are real, they are made of a special spiritual substance and, hence, could be formed, weighed, shaped, etc. by God. Creation, then, was the process of shaping the letters so as to form reality (see on this David R. Blumenthal¹ in his article “Creator and the Computer”, first published in the late 1970s).

| | | |
|-------------------------|------------|-----------------------|
| Emet (truth) | אמת | = 400 + 40 + 1 |
| Met (dead/death) | מת | = 400 + 40 |

A similar, but indeed far more interesting story for our context, is that of the Golem. In 1580, as the legend goes, in Prague the famous Rabbi Loew created a Golem, an artificial human being made from clay. This artificial mute man was created in order to protect the Jewish people in times of persecution. What is interesting in the context of a “pre-history” of the performative is the fact that this Golem would become alive simply by the Rabbi engraving the word “Emet”, or truth, on the Golem’s forehead. But as soon as the first letter of the word “Emet” is erased, the Golem immediately collapses. By deleting the first letter Aleph (which represents God, or more general, the creative power) from the word “Emet”, the word now reads “met”, which stands for “dead man”, “dead” or “death”. Thus, *one tiny letter* - aleph - marks the border between life and death, creation and destruction of the clay man - reminding one of the need to stick strictly to orthographic rules when programming: here, the difference between a comma and a semicolon could be fatal. Interestingly enough, when in 1965 the first computer mainframe was presented in Israel, the Kabbalah specialist Gershon Scholem named it “Golem”.² (see the name on the chip)



Another example for the performativity of speech is the Russian Futurist poet Velimir Chlebnikov.³ It would definitely be rewarding to research this poet's experimental work with language more closely, as he possibly could be considered a kind of precursor of software and code art (Olga Goriunova has pointed to this fact already in March this year). Chlebnikov's "star language" (*zvezdnyj jazyk*), a kind of universal language which he developed between 1915 and 1922 (the year of his death), and which also included an "alphabet of numbers", is set in the context of early 20th century avant-garde experimentation with language, but also very clearly distances itself from all these practices (like Dadaist sound poetry, *zaumnyj jazyk* by the Russian Futurists, *immaginazione senza fili / parole in libertà* by the Italian Futurists, and the Surrealists' *écriture automatique*). Chlebnikov's "star language" is special because it combines extremely archaic and utopian elements. For example, for Chlebnikov in the first letters of words (and of numbers) ontological relations are embodied linking words to history, time and the cosmos. The "star language" is thus not created at random, but presents (through the letters' ontological relations) a complex system functioning globally and through time.

These very few episodes in a yet to be written history of the performative may be entertaining -- they certainly are useful to me to prepare my argument. I would like to stress the importance of the notion of the performative - in contrast to the notion of the generative, or generative art which has become fashionable over the past few years. Very often, generative art is used as a synonym for software art -- which, to my understanding, is not really correct. To shed some light on this *connection between generative art and software art* is thus one aim of this presentation. The other is to propose the notion of *performativity of the code* as one of the reasons for contemporary artists' interest in using software as an artistic material. Performativity of the code in this case refers to its ability to act and perform in the sense of speech act theory.

I. Generative art ≠ software art

According to Philip Galanter (2003), generative art refers to »any art practice where the artist uses a system, such as a set of natural language rules, a computer program, a machine, or other procedural invention, which is set into motion with some degree of autonomy contributing to or resulting in a completed work of art.«⁴

Generative art thus describes processes defined by rules, processes which are automated to a certain degree by the use of a machine or computer, or by using mathematic or pragmatic instructions. By following these pre-defined rules or instructions, once set in motion these »self-organizing« processes are running independently from their authors or artist-programmers. Depending on the technical context in which the process is unfolding, the result is »unpredictable« and thus less the result of individual agency or authorship, than much more the result of the respective production conditions, or, if you wish, the result of the technical ecology of a certain system.⁵ Galanter's definition of generative art is, like definitions by other authors, an »inclusive«, all-embracing and comprehensive definition, leading Galanter to the conclusion that, surprisingly or not, »generative art is as old as art itself«⁶. But let's return to the essential feature: The most prominent element in all these attempts to define generative art – in electronic music and algorithmic composition, computer graphics and animation, the Demo scene and VJ culture and industrial design and architecture⁷ – is the employment of generative processes for the *negation of intentionality*. Most generative art is interested in generative processes (and in software or code) only insofar, as they generate »unpredictable« events. In this sense – and in this context – software and code are understood as pragmatic tools which remain unquestioned themselves. Exactly because of this – because of generative art's focus on the *negation of intentionality* and the fact that its main interest does not lie in the questioning of the tools employed – the notion of generative art cannot be used as a synonym for software art.

Software art, on the contrary, describes an artistic activity which in the medium – or rather: the material – of software allows for a critical reflection of software (and its cultural impact). Software art does not regard software merely as a pragmatic, invisible tool generating certain visible results or surfaces, but on the contrary focuses on the program code itself – even if this code is not explicitly being laid open or put in the foreground. According to Florian Cramer, software art makes visible the aesthetic and political subtexts of seemingly neutral technical commands. Doing so, software art can happen on different levels: it can be located on the level of the source code, on the level of abstract algorithms or on the level of the result generated by a certain program code. Thus it comes as no surprise that there is a broad spectrum of software artworks ranging from so-called »Codeworks« consisting predominantly of ASCII-Code (not being *executables*), to experimental web browsers

(e.g. I/O/D's *WebStalker*, 1997), and fully-executable programmes (e.g. Antoine Schmitt's *Vexation 1*⁸, 2000, or Adrian Ward's *Auto-Illustrator*⁹, 2000). In generative art, software is only one material amongst others. Software art, on the other hand, *can* contain elements of generative art but does not necessarily have to be generative in a strict technical sense (see the »Codeworks«). Software art and generative art can therefore not be used synonymously. Rather, these two notions function in *different registers*, as I hope to show in the following examples.



insert_coin

My first example is the project *insert_coin*¹⁰ by Dragan Espenschied and Alvar Freude. In the framework of their diploma work which they realised under the motto »two people control 250 people« in 2000/2001, the two media art students secretly installed a Web proxy server at the Merz Academy in Stuttgart, Germany, which via a perl script manipulated the entire Web traffic of students and professors in the

Academy's computer network. According to Espenschied and Freude, the aim of this project was to critically assess the »competence and the critical faculty of the users concerning the everyday medium Internet«¹¹. The manipulated proxy server redirected the entered URLs onto other addresses, modified HTML code, transformed the content of the latest news on news websites via a simple search-and-replace function (e.g. by replacing the name of politicians) as well as the content of private e-mails that were read through Web interfaces like Hotmail or Yahoo!. During four weeks this project was manipulating the Web access of the entire Academy – and it remained unnoticed. When Espenschied and Freude announced the project publicly, almost nobody was interested. They even published a simple-to-follow instruction manual which would enable everybody to independently switch-off the filter that was manipulating the Web content. But only a minority of those concerned took the time to make the minor adjustment in order to regain access to unfiltered information. Still several months after the end of the experiment the Web access from most of the Academy's computers was filtered.

```

# /usr/bin/perl

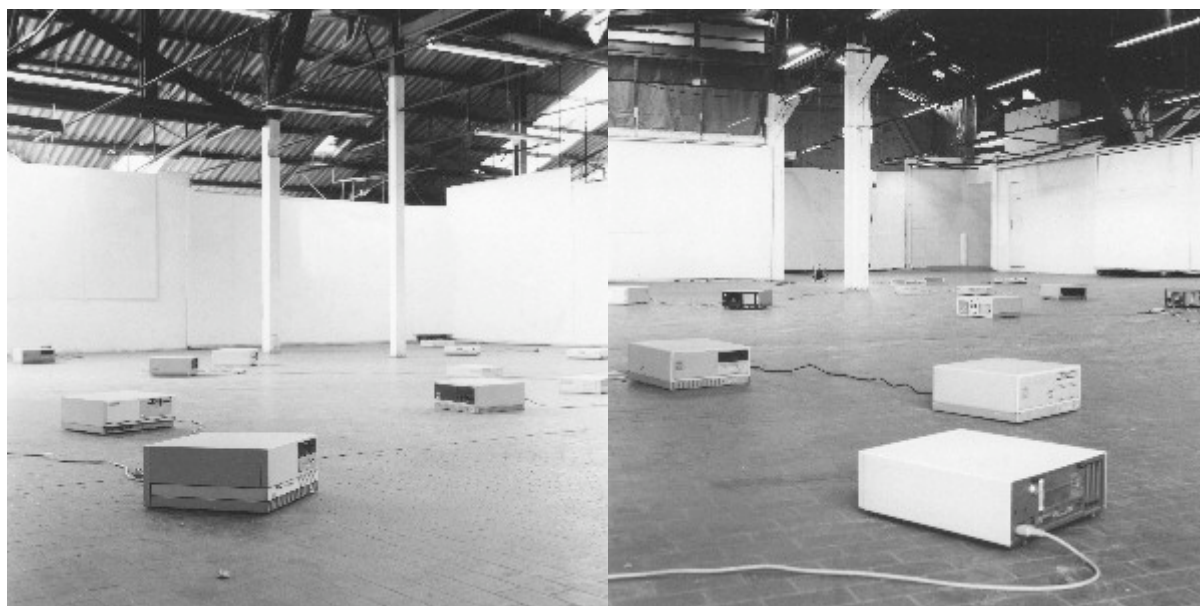
# walsner.pl v0.41 (still without makewalsner.pl)
#
# this script generates the plain ascii version
# of "tod eines Kritikers" by martin walser.
# it can be redistributed and/or modified under
# the terms of the gnu general public license
# as published by the free software foundation,
# but may not be run without written permission
# by subkamp verlag, frankfurt am main, germany.
#
# source code: http://tests.com/trash/walsner.pl.txt
# release notes: http://tests.com/trash/readme.txt
# bug reports: http://waise.de/newsticker/foceen/go.html?list=1&forum_id=10648
# no copyright 2002 tests.com
#
# perl port no copyright 2002 luther blissett

[2] .* "6400a1021272829c2d2e2f303132333435363738393a3b3c3d3e3f404142434445464748494a4b4c4d4e4f";
[2] .* "505152535455565758595a5b5c5d5e5f606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f";
[2] .* "8997a9b4c4d5d6d7d8d9dele4e7e8e9edf1f2f3f4f5f6f7f8f9f0022635474a40460a373c48504351103b5147";
[2] .* "154a4f55465e1b3c5e5642585b564662446656459672a466e7072643148776970666d6c39454445";
[2] .* "483d8e4c6967a057f7683874975783087e85504c938391828b9b999c5a8b985a000f989e424e4d";
[2] .* "6e03d21a0413b08030e304350420f554f5344484e46525b4d57201c555b413155446161585a885c";
[2] .* "29545b4e2d6163713185646878786b7b827e7872423e71748342a2797a7a840864827883318b4f7b";
[2] .* "82939695858c9958989a194915e9294a263b8959943464533c4b453f0a3e504e40461030545047";
[2] .* "4a5a452195e584f1d3953615e64575b59432c28594e5d632d62476d7a666e71697737616d727870";
[2] .* "443d5274797e436d797e82487c7e6d4c7982908a8594538785978c5891975b949e94939d98989a1";
[2] .* "9943031f434743080c3d4f4d3f450f2b4f5514a56464c504d1f1b3e5558605e565a5e6025645a5c";
[2] .* "6a2a2b6960627064305d67756a666e796d77323c7b757c764184757b8e7e7c8b84827d83814e8b84";
[2] .* "8e85a9068c9c9050f1a8e909f15e8a94a29d93a99440345394548463c4e5444414f521042454755";
[2] .* "155a42b4d3e1b47511265545e5567615b255b635d646e6d6b6b67726369677355e727a7c6e7770";
[2] .* "3d83738178174787d897b8b4e4a90800e9387848c87aa82988e8d9c5998e909e5ea494a193a488";
[2] .* "9d453747054c3b490c3e48130c363f535a25122d565e49522c19405d555b625454225b6125425b69";
[2] .* "655a69742c382f3b31e33413544374739873b454d3e4842414b51444e534751554a54574c594e4b";
[2] .* "a479536d73705e587278755c7f787d837b42838886231d201f29073027d240b330e395035122c14";
[2] .* "2163f184219431c471e44203b224a24432651285129432c552e583059325134483661386239593c";
[2] .* "603e5a403b426444334671487149574b4855427b4b797b7d7c746f78837d773b748e5f9c919f63e8";
[2] .* "a312034038470e084f3a462c56520f524349544849474b551924653145a575158225c58686f6a28";
[2] .* "6e5e805c6a695f0474328956c716b3d197736e74813e7482887385887a8a4849682a04d868187";

```

walsner.php

My second example, *walser.php* (2002) by textz.com/Project Gutenberg (i.e. Sebastian Lütgert), has been called »political« or »literary«¹² software. We might call it an anticopyright-activist software which has been written in response to one of the biggest literary scandals in Germany after the Second World War. The file name *walser.php* is not only an ironic allusion to the file »walser.pdf«, a digital pre-printed version of Martin Walser's controversial novel which was distributed by the Suhrkamp publishing house as an e-mail attachment – and later on, due to the unfavorable circumstances, called back by the publisher (nice try: calling back a digital document). Rather, *walser.php* (or rather *walser.pl*) by textz.com was/is a Perl script which via an appropriate Perl interpreter can generate a human-readable ASCII text version of Walser's novel *Death of a Critic* from the 10.000 lines of source code¹³. While the source code written in Perl contains the novel itself in an »invisible«, machine-readable form and thus can be distributed and modified as free software under the GNU General Public License, it may be *executed* only with the written permission of the Suhrkamp publishing house.¹⁴



Sealed Computers

My third example is Maurizio Bolognini's installation *Sealed Computers* (since 1992). It consists in total of over 200 computers producing and exchanging a continuous flow of random images. These computers are sealed and left to work indefinitely, and also remain unconnected to any kind of output device (like, e.g., a monitor). As there is no visible result of what is generated and exchanged by these

machines, we are left with a somewhat uncomfortable feeling of an invisible, self-contained performative which cannot be controlled.

While Espenschied & Freude's experiment on filtering and censorship of Internet content points to the relatively unlimited potential of control that is contained in software, *walser.php* offers a practical solution for dealing with the commercial restrictions which threaten the freedom of information on the Internet in the form of Digital Rights Management Systems (DRM). While *insert_coin* temporarily realises a dystopian scenario in form of manipulated software, textz.com with its *walser.php* project develops genuinely utopian »counter measures in the form of software«¹⁵. Bolognini's *Sealed Computers*, on the other hand, embody the invisible performativity of code as a mute and autistic entity or process.

All of these projects are *generative* in the best sense of the word. However, neither *insert_coin* nor *walser.php* comply with the definitions of »generative art« currently found predominantly in the area of design. Philip Galanter for example, whom I quoted in the beginning, defines generative art as a process *contributing to or resulting in a completed work of art*. Similarly, Celestino Soddu, director of the Generative Design Lab at the Polytechnical University of Milano and organiser of the *Generative Art*¹⁶ conferences, defines »generative art« as a processual tool enabling the artist or designer to »synthesize [...] an ever changing and unpredictable series of events, pictures, industrial objects, architectures, musical works, environments, and communications«. ¹⁷ And finally, also the *Codemuse* web site defines generative art as a process with parameters that the artist should experiment with »until the final results are aesthetically pleasing and/or in some way surprising«¹⁸.

What becomes apparent in these quotations is the fact that generative art is interested predominantly in the *results* created by generative processes. Software in this context is seen and employed as a pragmatic-generative tool or device for the creation of certain results – without being questioned itself. The generative processes brought about by software here primarily do serve to avoid intentionality and to produce an unexpected, arbitrary and inexhaustible diversity of forms. The *n_Gen Design Machine* by Move Design, submitted to the *Read_Me* Festival 2003 in Helsinki, as well as Cornelia Sollfranks *net.art Generator*¹⁹ (1999) which at the push of a button generates net art, should be seen as ironic commentaries on »generative design« (mis-)understood in such a way.²⁰

insert_coin and *walser.php* go beyond such definitions of »generative art« or »design« in so far as these projects are interested far more in the *coded processes* generating certain results or surfaces. This interest in the coded processes, or, to be more precise, in the significance and implications of software and coded structures, sharply distinguishes them not only from generative art but also from many interactive installations of the 1990s which displayed their disinterest in software by hiding the program code in *black boxes*. Instead, projects like *insert_coin* and *walser.php* aim at questioning software and code as culture – and at questioning culture as implemented in software. For this, they develop »experimental software« (in *insert_coin* a proxy server and in *walser.php* a perl script), which does not only generate arbitrary surfaces but which critically investigates the technological, cultural or social impact of software. What's more, the writing of »experimental software« is very well concerned with *artistic subjectivity*, as can be seen in the usage of different private languages, and less with proving evidence of a machinic creativity (whatever this may be): »Code can be diaries, poetic, obscure, ironic or disruptive, defunct or impossible, it can simulate and disguise, it has rhetoric and style, it can be an attitude«²¹, thus the emphatic definition by Florian Cramer and Ulrike Gabriel, both members of the *transmediale* software art jury in 2001.

I have tried to set up a somewhat polemical comparison between generative art and software art you can see here:

| Generative art ¹ | Software art |
|---|--|
| Focuses on the <i>surface</i> (« <i>phenotext</i> ») created by a generative process (" <i>black box</i> problem") | Focuses on the generative <i>process</i> (set in motion by a « <i>genotext</i> »), which may also generate surfaces or other results |
| Software considered a pragmatic/neutral <i>tool</i> with which to create a certain (artistic) result. The tool itself is not called into question. | Software considered <i>culture</i> that is questioned; interest in its aesthetic and political subtexts; software can be «experimental» and «non-pragmatic» |
| Software considered a <i>pragmatic-generative tool</i> | Software considered a <i>self-contained (experimental) work</i> |
| <i>Efficient code</i> _The «art» of programming is often described as being «elegant» or «aesthetic» because it develops particularly small, memory-efficient, pragmatic solutions to problems («beautiful algorithms»).^2 | <i>Code as excess, code as extravagance</i> (potentially) _Useless, wasteful code, running wild, deliberately «anti-elegant» and «experimental» code _ Software art can also contain non-wasteful, curt code |
| Uses generative processes to <i>negate intentionality</i> | »Software artists [...] seem to conceive of generative systems not as negation of intentionality, but as balancing of randomness and control. [...] Far from being simply art for machines, <i>software art is highly concerned with artistic subjectivity</i> and its reflection and extension into generative systems.« ³ |
| Fascination with the <i>generative</i> | Focus on the <i>performativity</i> of the code |

Comparison of 'generative art' with 'software art'

The term "software art" was first defined in 2001 by the Berlin media art festival *transmediale*²² and introduced as one of the festival's competition categories.²³ Software art, referred to by other authors as «experimental»²⁴ and «speculative software»²⁵ as well as «non-pragmatic» and «non-rational»²⁶ software, comprises projects that use program code as their main artistic material or that deal with the cultural understanding of software, according to the definition developed by the *transmediale* jury. Here, program code is not considered a pragmatic-functional tool that serves the 'real' art work, but rather as a generative material consisting of machinic and social processes. Software art can be the result of an autonomous and

¹ The characteristics of 'generative art' and 'generative design' listed here are in part also valid for computer art of the 1960s and many interactive installations of the 1990s.

² See Donald Knuth, *The Art Of Computer Programming: Vol. 1, Fundamental Algorithms*, Reading, Mass. 1997.

³ Florian Cramer / Ulrike Gabriel, citing Andreas Broeckmann, "On Software as Art," in: *Sarai Reader 2003: Shaping Technologies*, New Delhi 2003, pp. 215–218, here: p. 216 [my emphasis].

formal creative practice, but can also refer critically to existing software and the technological, cultural, or social significance of software.²⁷

Interestingly, the difference between software art and generative design is reminiscent of the difference between software art that was developed in the late 1990s and the early computer art of the 1960s. The difference can be described as follows: Works from the field of software art »are not art created *using a computer*, but art that *takes place in the computer*; it is not software programmed by artists in order to create *autonomous works of art*, but *software that is itself a work of art*. With these programs, it is *not the result* that is important, but the *process* triggered in the computer (and on the computer monitor) by the program code.«²⁸ Computer art of the 1960s is close to concept art in its privileging of the concept as opposed to its realisation. However, it does not follow this idea through to its logical conclusion: its work, executed on plotters and dot matrix printers, has an emphasis on the final product and not the program or process that created the work.²⁹

II. Performativity of the Code vs. Fascination of the Generative

The current interest in software, according to my hypothesis, is not only attributable to a fascination with the generative aspect of software, that is, to its ability to (pro)create and generate in a purely technical sense. Of interest to the authors of these projects is something that I would call the *performativity* of code. By *performativity of the code* I mean its ability to act and perform in terms of speech act theory.

I am thinking here of a series of lectures held by John Langshaw Austin at the Harvard University in 1955. In these lectures, entitled *How to Do Things With Words*, Austin outlined the groundbreaking theory that language does not only have a descriptive, referential or constative function, but also possesses a performative dimension.³⁰ According to Austin, linguistic utterances by no means only serve the purpose of describing a situation or stating a fact, but are used to commit acts. Austin's speech act theory regards speech essentially as action and sees it as being effective not on the merit of its results, but in and of itself. This is precisely where the speech act theory meets the code's assumed performativity: »[When] a word not only means something, but performatively generates exactly that what it names.«³¹

Austin identifies three distinct linguistic acts in all speech acts: the locutionary³², the illocutionary³³, and the perlocutionary³⁴ act. He defines the *locutionary act* as the

propositional content, which can be true or false. This act is not of further interest to us in this context. *Illocutionary acts* are acts that are performed by the words spoken. These 'performatives' (that create or do what they describe) are defined as acts in which a person who says something also does something (for example, a judge's verdict: »I sentence you« is not a declaration of intent, but an action.) The message and execution come together: Here, simply »uttering [the message] is committing an act.«³⁵ This draws attention to the importance of the *context* of a performative utterance. Illocutionary, or performative utterances have certain consequences and can either succeed or fail, depending on whether certain extra-linguistic conventions are fulfilled or not.³⁶ *Perlocutionary acts*, on the other hand, are utterances that trigger a chain of effects. The speech itself and the consequences of that speech do not occur at the same time. As Judith Butler notes, the »consequences are not the same as the speech act, but rather the result or the 'aftermath' of the utterance.«³⁷ Butler summarises the difference in a succinct formula: »While illocutionary acts take place with the help of linguistic conventions, perlocutionary acts are performed with the help of consequences. This distinction thus implies that illocutionary speech acts produce effects without delay, so that 'saying' becomes the same as 'doing' and that both take place simultaneously.«³⁸ Insofar as 'saying' and 'doing' coincide, program codes can be called illocutionary speech acts.

According to Austin, speech acts can also be acts, without necessarily having to be effective (that is, without having to be 'successful'). If these acts are unsuccessful, they represent failed performative utterances. Thus, speech acts are not always *effective* acts. »A successful performative utterance [however] is defined in that the act is not only committed,« writes Butler, »but rather that it also triggers a certain chain of effects.«³⁹ Program codes, viewed very pragmatically, are only useful as successful performative utterances; if they do not cause any effect (regardless of whether the effect is desired or not), or they are not executable, they are plain and simply redundant. In the context of functional pragmatic software, only executable code makes sense. In software art, on the contrary, also non-executable code has a purpose.

Accordingly, if I speak of the performativity of code, I claim that this performativity is not to be understood as a purely technical performativity, i.e. it does not only happen in the context of a closed technical system, but affects the realm of the aesthetical, the political and the social. Program code is characterised by the fact that

here »saying« coincides with »doing«. Code as an effective speech act is not a description or a representation of something, but, on the contrary, it directly affects, and literally sets in motion – or it even »kills« a process.⁴⁰ This »coded performativity«⁴¹ has immediate, also political consequences on the actual and virtual spaces (amongst others, the Internet), in which we are increasingly moving and living: it means, ultimately, that this coded performativity *mobilises or immobilizes* its users. Code thus becomes Law, or, as Lawrence Lessig has put it in 1999, »Code [already] is Law«⁴². This is the reason why software art is rather more interested in the »performance« than in the »competence« (terms coined by Noam Chomsky), rather more interested in the *parole* than the *langue*⁴³ (famous opposition coined by Ferdinand de Saussure). In our context, *performance* and *parole* refer to the respective actualisations and concrete realisations and consequences a certain program code has on, let's say, social systems, and not only what it does or generates in the context of abstract-technical systems. In the projects *insert_coin* and *walser.php* the generative is deeply political – specifically because changing existing texts covertly (in the case of *insert_coin*) and extracting copyrighted text from a Perl script (in the case of *walser.php*) is interesting - *not* in the context of *technical* systems, *but* rather in the context of *social and political systems* that are becoming increasingly dependent on these technical structures.

Certainly one of the »most radical understanding[s] of computer code as artistic material«⁴⁴ can be found in the so-called »Codeworks«⁴⁵ and the artistic use they make of program code. »Codeworks« almost exclusively consist of texts which are sent to mailing lists like *Nettime* or *7-11* in the form of simple e-mails. »Codeworks« make use of formal ASCII instruction code and its aesthetic – without relying on surfaces and graphical user interfaces usually created by this code. Works by Jodi, Netochka Nezvanova aka antiorp and mez⁴⁶ thus recall the existence of a hidden, »invisible shadow world of process«⁴⁷, as Graham Harwood has called it. Technically speaking, these »Codeworks« are located on the opposite side of an imaginary spectrum of generativity. However, the status of these languages or these language-like bits and pieces remains ambivalent: In the perception of the recipient they oscillate between supposed executability, thus functionality, and non-executability – i.e. dysfunctionality – of the code; in short: between significant information and meaningless noise. This phenomenon can be seen very clearly in Jodi's *walkmonster_start ()* e-mail which was sent to the *Nettime* mailing list on October 22,

2001. While the text contained in this e-mail resembles executable program code, for the non-specialist reader it remains completely open whether in another location in the computer this text could in fact be compiled, and thus be turned into machine-readable algorithms, and thus, ultimately, be executable.

What plays a major role here rather than the actual technical execution is the understanding of the fact that the code fragments used in the Codeworks can *potentially* be executed and thus become performative. However, in »The Aesthetics of Generative Code« Geoff Cox, Alex McLean and Adrian Ward claim that »the aesthetic value of code lies in its execution, not simply in its written form«. ⁴⁸ While I can agree with this assertion for projects like *insert_coin*, *walser.php* and the *Sealed Computers* – because their critical (and perhaps even poetic) momentum lies exactly in their technical execution – this definition would have to be extended regarding the structure of the »Codeworks«. The aesthetic and poetic value of these »Codeworks« indeed is constituted not only by their textual form, but by the fact and the knowledge that they might *potentially* be executable. I would like to broaden the notion of the generative in the sense that code is not only executable in technical environments, but can become extremely productive as »imaginary software« in the reader him- or herself. (one can find a parallel in J.R. Searle's 1969 further development of *Speech Acts ...*)

In contrast to generative art, software art directs our attention on the fact that our (media) environment is increasingly relying on programmed structures. In doing so, the »Codeworks« use the »poor« medium of text which at the same time appears to be performative, or executable in the context of the command line. By using precisely this ambivalence or this oscillation between simplicity and totality of execution, the codeworks, and, more generally speaking, software art as a whole, point to the potentially totalitarian dimension of the algorithmic program code, the »invisible shadow world of process«.

- ¹ This article first appeared in David R. Blumenthal: *Understanding Jewish Mysticism* (New York: Ktav Publishing: 1978) 22-29. It was subsequently published as an article in *History, Religion, and Spiritual Democracy: Essays in Honor of Joseph L. Blau*, ed. M. Wohlgeleuter (New York: Columbia University Press, 1980) 114-29. <http://www.emory.edu/UDR/BLUMENTHAL/CreatorandComputer.html>
- ² Cf. on this Gershom Scholem, "The Idea of the Golem," *On the Kabbalah and its Symbolism* (New York: Schocken, 1965) 158-204 with special attention to 165-73, 184-95 as well as the full-fledged study of M. Idel, *Golem* (New York: SUNY Press, 1990). In 1981 Stanislaw Lem (*1921) described in *Also sprach GÖLEM* the lectures of the computer GÖLEM XIV. Golem XIV belongs to the machine intelligentsia of the 21st century superior to the human race who in his lectures deals with the position of the human race in the cosmos.
- ³ Cf. Roman Jakobson: *Über die neueste russische Poesie* (1919), in: *Die Erweckung des Wortes. Essays der russischen Formalen Schule*, hrsg. von Fritz Mierau, pp. 177-210; Roman Jakobson [1921]: „Novejsaja russkaja poezija / Neueste russische Dichtung“, in: W.-D. Stempel (Hrsg.), *Texte der russischen Formalisten* Bd. II, München 1972, 18-135; Jurij Tynjanov, „Velimir Chlebnikov“ (1928), in: ders., *Die literarischen Kunstmittel und die Evolution in der Literatur*, Frankfurt 1967, p. 69.
- ⁴ Galanter, P.: »What is Generative Art? Complexity Theory as a Context for Art Theory«, *Generative Art Proceedings*, Milano 2003, p.4, <http://www.philipgalanter.com/pages/acad/media/ga2003%20proceedings%20paper.pdf>. The mailing list eugene is devoted to the discussion of generative art, see <http://www.generative.net/>.
- ⁵ See also Cox, G.: *anti-thesis: the dialectics of generative art (as praxis)*, MPhil/PhD Transfer Report 2002, <http://www.anti-thesis.net/>. A similar definition can be found in Adrian Ward: »Generative art is a term given to work which stems from concentrating on the processes involved in producing an artwork, usually (although not strictly) automated by the use of a machine or computer, or by using mathematic or pragmatic instructions to define the rules by which such artworks are executed.« (http://www.generative.net)
- ⁶ Galanter, *ibid.*, p. 1
- ⁷ Galanter, *ibid.*, p.2, calls these four areas the four »main clusters« of generative art.
- ⁸ <http://www.gratin.org/as/>
- ⁹ <http://www.signwave.co.uk>
- ¹⁰ http://www.odem.org/insert_coin/
- ¹¹ Cf. Espenschied/Freude's text for the *Internationaler Medienkunstpreis 2001*, http://www.onlinedemonstration.org/insert_coin/imkp2001.html
- ¹² Cramer, F.: »walser.php« In: Goriunova, O. / Shulgin, A. (eds.): *Read_Me 2.3. Reader*. Helsinki: Nifca, 2003, pp.76–78, here: p.76 <http://textz.com/trash/walser.pl.txt>
- ¹³ <http://textz.com/> »Suhrkamp calls back walser.pdf, textz.com releases walser.php«, <http://textz.com/trash/readme.txt>
- ¹⁴ Cramer, *ibid.*, p. 77
- ¹⁵ Cf. <http://www.generativeart.com/>
- ¹⁶ Soddu, C.: »Generative Art and Architecture«, Abstract, without date, <http://www.nyu.edu/studio/generative.html>
- ¹⁷ http://www.codemusenet/html_files/GenerativeArt.html [Dec. 23, 2003].
- ¹⁸ Cornelia Sollfrank, *net.art generator* (1999), <http://soundwarez.org/generator/>
- ¹⁹ Cf. Goriunova, O. / Shulgin, A.: »n_Gen Design Machine« In: Goriunova, O. / Shulgin, A. (eds.): *Read_Me 2.3. Reader*. Helsinki: Nifca, 2003, pp.66–67, here: p.66.
- ²⁰ Cramer, F. / Gabriel, U.: »Software Art« In: Broeckmann, A. / Jaschko, S. (eds.): *DIY Media – Art and Digital Media: Software - Participation - Distribution. Transmediale.01*. Berlin, 2001, pp.29–33, here p.33
- ²¹ Other notable events: "Kontrollfelder" (Dortmund 2001, curated by Andreas Broeckmann and Matthias Weiß, [<http://www.hartware-projekte.de/programm/inhalt/kontroll.htm>] [Jan. 2, 2004]); the "Read_Me" Festival, conceived by Olga Goriunova and Alexei Shulgin (Moskau 2002, Helsinki 2003, [http://www.m-cult.org/read_me/] [Dec. 31, 2003]) and the exhibitions "Generator" (GB 2002, curated by Geoff Cox, [<http://www.generative.net/generator.html>] [Dec. 31, 2003]), "CODEDOC"(New York, Sept. 2002, curated by Christiane Paul, [<http://artport.whitney.org/commissions/codedoc/>] [Dec. 31, 2003]), "I love you - computer_viren_hacker_kultur" (Frankfurt/Main, Jan. 31-Feb. 5, 2003, [http://www.digitalcraft.org/index.php?artikel_id=269] [Jan. 2, 2004]) and the software art repository "Runme", launched in January 2003 ([<http://runme.org>] [Dec. 31, 2003]). Further examples of software art can be found on these Web sites. The most historically significant year in terms of software art is 1970, during which three software art-related events took place: Jack Burnham's exhibition "Software – Information Technology: Its New Meaning for Art", which took place at the Jewish Museum; the exhibition curated by Kynaston McShine at MoMA in New York, entitled "Information"; and the foundation of the magazine "Radical Software" by Beryl Korot, Phyllis Gerhuny, and Ira Schneider ([<http://www.radicalsoftware.org/>] [Dec. 31, 2003]).
- ²² For an early, programmatic concept paper on software programming and art, see Geoff Cox / Alex McLean / Adrian Ward, "The Aesthetics of Generative Code" (2000), [<http://generative.net/papers/aesthetics/>] [Dec. 18, 2003]. An attempt to formally define and research the archaeological history of software art using literary and artistic examples can be found in Florian Cramer, "Concepts. Notations. Software. Art", Mar. 23, 2002, [http://userpage.fu-berlin.de/~cantsin/homepage/writings/software_art/concept_notations/concepts_notations_software_art.html] [Nov. 19, 2003].

- ²⁴ Tilman Baumgärtel, "Experimentelle Software. Zu einigen neueren Computerprogrammen von Künstlern", in: *Telepolis*, Oct. 28, 2001, [<http://www.heise.de/tp/deutsch/inhalt/sa/9908/1.html>] [Dec. 31, 2003].
- ²⁵ Matthew Fuller, for example, distinguishes between 'critical', 'social', and 'speculative software'. See Matthew Fuller, "Behind the Blip: Software as Culture," in: *Nettime*, Jan. 7, 2002, [<http://amsterdam.nettime.org/Lists-Archives/nettime-l-0201/msg00025.html>] [Dec. 19, 2003].
- ²⁶ Olga Goriunova and Alexei Shulgin define 'artistic software' as 'unpragmatic' and 'irrational': "[I]f conventional programs are instruments serving purely pragmatic purposes, the result of the work of artistic programs often finds itself outside of the pragmatic and the rational." (Olga Goriunova / Alexei Shulgin, »Artistic Software for Dummies and, by the way, Thoughts About the New World Order," in: *Nettime*, May 26, 2002, [<http://amsterdam.nettime.org/Lists-Archives/nettime-l-0205/msg00169.html>] [Nov. 19, 2003]).
- ²⁷ See [http://www.transmediale.de/04/pdf/tm04clubtm04_formular_ausschreibung.pdf] [Nov. 17, 2003]. See also the panel discussion from transmediale.03 (Künstlerhaus Bethanien, Feb. 4, 2003), [<http://www.softwareart.net/>] [Nov. 19, 2003], and Olga Goriunova / Alexei Shulgin (Hg.), *Read_Me 2.3 Reader – about software art*, Helsinki 2003, [http://www.m-cult.org/read_me] [Nov. 17, 2003].
- ²⁸ Baumgärtel, *ibid.* [my accentuation]
- ²⁹ Typical in this context are the artworks by the so-called »Algorists«, who were co-founded by Roman Verostko. Cf. Verostko, R.: »Epigenetic Painting: Software As Genotype, A New Dimension of Art « (1988), <http://www.verostko.com/epigenet.html>; Verostko, R.: »Epigenetic Art Revisited: Software as Genotype«, in: Schöpf, C. / Stocker, G. (eds.): *Ars Electronica 2003: Code - The Language of Our Time*. Ostfildern: Cantz, 2003, pp.156–167. Here one finds formulations like: »The essential character of each finished work is derived from the ›form-generating-procedure‹ or ›algorithm‹ acting as genotype. For this reason one could say that the finished work is an epiphany, or manifestation, of its generator, the *code*. For me each work celebrates its code [...].«
- ³⁰ For a discussion of Austin's elementary distinction between performative and constative utterances cf. Kent Bach: Performatives, in: *Routledge Encyclopedia of Philosophy*, <http://online.sfsu.edu/~kbach/perform.html>; Richard van Oort: Performative-Constative Revisited: The Genetics of Austin's Theory of Speech Acts, in: *Anthropoetics II*, no. 2 (January 1997), <http://www.humnet.ucla.edu/humnet/anthropoetics/Ap0202/Vano.htm>:
- ³¹ Judith Butler, *Hass spricht. Zur Politik des Performativen*, Berlin 1998, p. 67.
- ³² Locutionary: The speech act as meaningful utterance.
- ³³ Illocutionary: A meaningful utterance with a certain conventional – performative – force.
- ³⁴ Perlocutionary: A meaningful utterance with a certain conventional force non-conventionally bringing about a certain effect.
- ³⁵ *Ibid.*, p. 67.
- ³⁶ The work on speech acts was extended by John Searle in his *Speech Acts* (1969) book. He attempted to classify speech acts into five classes: representatives (informing), directives (request), commissives (promise), expressives (thanking), and declarations (declare marriage).
- ³⁷ Butler., p. 31.
- ³⁸ *Ibid.*, p. 31
- ³⁹ *Ibid.*, p. 31.
- ⁴⁰ Cf. Arns, I.: »Texte, die (sich) bewegen: zur Performativität von Programmiercodes in Netzkunst und Software Art« In: Arns, I. / Goller, M. / Strätling, S. / Witte, G. (eds.): *Kinetographien*. Bielefeld: Aisthesis, 2004 [forthcoming]
- ⁴¹ Grether, R.: »The Performing Arts in a New Era«, *Rohrpost*, July 26, 2001, <http://coredump.buug.de/pipermail/rohrpost/2001-July/000353.html>
- ⁴² Lessig, L.: *Code and other Laws of Cyberspace*. New York: Basic Books, 1999
- ⁴³ The distinction between competence and performance is credited to Noam Chomsky's generative transformation grammar (see Chomsky, N.: *Aspects of the Theory of Syntax*, Cambridge, MA., 1965); the distinction between *langue* and *parole* is attributed to Ferdinand de Saussure (see de Saussure, F.: *Cours de linguistique générale*, Paris 1967 [1916]).
- ⁴⁴ Cramer, F.: »Exe.cut[up]able statements: Das Drängen des Codes an die Nutzeroberflächen«, in: Schöpf, C. / Stocker, G. (eds.): *Ars Electronica 2003: Code - The Language of Our Time*. Ostfildern: Cantz, 2003
- ⁴⁵ Cf. on this Sondheim, A.: »Codework« *American Book Review*, Vol. 22, Issue 6 (September/October 2001), <http://www.litline.org/ABR/PDF/Volume22/sondheim.pdf>
- ⁴⁶ Cf. for more examples Florian Cramers »<nettime> unstable digest« auf <http://www.nettime.org/archives.php>
- ⁴⁷ Harwood, G.: »Speculative Software« In: Broeckmann A. / Jaschko, S. (eds.): *DIY Media - Art and Digital Media: Software - Participation - Distribution. Transmediale.01*. Berlin, 2001, pp.47–49, here p.47
- ⁴⁸ Cox, G. / McLean, A. / Ward, A.: »The Aesthetics of Generative Code« (2000), <http://generative.net/papers/aesthetics/>