UOC
www.uoc.edu

#Artnodes

July 2005 ISSN: 1695-5951
www.uoc.edu/artnodes

Article

Code as performative speech act

http://www.uoc.edu/artnodes/eng/art/arns0505.pdf

Inke Arns

Artnodes
July 2005 ISSN: 1695-5951

#Artnodes

UOC
www.uoc.edu

Article

Code as performative speech act

http://www.uoc.edu/artnodes/eng/art/arns0505.pdf

Inke Arns

Abstract

Software art involves an artistic activity which, in the medium —or rather, the material— of software, allows for critical reflection on software (and its cultural impact). It thus highlights the aesthetic and political subtexts of seemingly neutral technical commands. In this article Inke Arns argues that, in the context of software art, a far more interesting notion than the "generative" nature of code is its "performativity". This notion —borrowed from speech act theory— not only involves the ability to generate in a technical context, but also encompasses the implications and repercussions of code in terms of aesthetics, politics and society. This article proposes the notion of the performativity of code as one of the reasons for contemporary artists' growing interest in using software as an artistic material.

Keywords

Software art, generative art, code, performativity

Introduction

Let's start at the very *beginning* (I won't spare you this classic). Before discussing my theses, I would like to give you three historical examples of the performativity of speech or texts in general: one biblical, one mythological and one example from Russian Futurism, ie, from the early 20th century.

According to the Bible, God created the world through speech. God spoke and the world came into being. According to *Sefer Yesira* (the 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 letters so as to form reality (David R. Blumenthal,[1] The Creator and the Computer article, late 1970s).

A similar, but indeed far more interesting, story for our context is that of the Golem. In 1580 in Prague, as the legend goes, 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 come alive simply by the Rabbi engraving the word "Emet", or truth, אמת 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 generally, the creative power) from the word "Emet", the word now reads "met", מת, which stands for "dead man", "dead" or "death". Thus, *one small 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 the first computer main-

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. Wohlgelernter (New York: Columbia University Press, 1980) 114-29. http://www.emory.edu/UDR/BLUMENTHAL/CreatorandComputer.html.

frame was presented in Israel in 1965, the Kabbalah specialist Gershom Scholem named it "Golem"[2] (see the name on the chip).

Another example of the performativity of speech is the Russian Futurist poet Velimir Khlebnikov.[3] It would definitely be rewarding to research this poet's experimental work with language more closely, as he could be considered a kind of precursor to software and code art (Olga Goriunova has already pointed to this fact in March this year). Khlebnikov'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 (such as Dadaist sound poetry, *zaumnyj jazyk* from the Russian Futurists, *immaginazione senza fili / parole in libertà* from the Italian Futurists, and the Surrealists' *écriture automatique*). Khlebnikov's "star language" is special because it combines extremely archaic and utopian elements. For example, for Khlebnikov, the first letters of words (and numbers) embody ontological relations 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 throughout time.

These scarce episodes in a yet to be written history of the performative are entertaining and useful to form the basis of my argument, as I wish 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 entirely 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 *the performativity of code* as one of the reasons for contemporary artists' interest in using software as an artistic material. The performativity of code in this case refers to its ability to act and perform as in speech act theory.

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 mathematical or pragmatic instructions. By following these pre-defined rules or instructions, once set in motion these "self-organising" processes run 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, and 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 architecture)[7] 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 in 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, involves an artistic activity which, in the medium —or rather, the material— of software,

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 by M. Idel, *Golem* (New York: SUNY Press, 1990). In 1981 Stanislav Lem (*1921) described in *Also sprach GOLEM* the lectures of the computer GOLEM 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*, Fritz Mierau (ed.), p. 177-210; Roman Jakobson [1921]: "Novejsaja russkaja poezija / Neueste russische Dichtung", in: W.-D. Stempel (ed.), *Texte der russischen Formalisten* vol. II, Munich 1972, 18-135; Jurij Tynjanov, "Velimir Chlebnikov" (1928), in: *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*, Milan 2003, p. 4, http://www.philipgalanter.com/pages/acad/media/ga2003%20proceedings%20paper.pdf. The mailing list eu-gene 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.

allows for critical reflection on 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 highlights the aesthetic and political subtexts of seemingly neutral technical commands. By doing so, software art can take place on different levels: it can be located at the level of the source code, at the level of abstract algorithms or at 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 executables), to experimental web browsers (eg, I/O/D's *WebStalker*, 1997), and fully-executable programs (eg, 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.

My first example is the *insert_coin*[10] project by Dragan Espenschied and Alvar Freude. As part of their diploma work in 2000/2001, which they carried out under the motto "two people control 250 people", the two media art students secretly installed a web proxy server at the Merz Academy in Stuttgart, Germany, which, via a Perl script, manipulated all the web traffic of students and professors on 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 URLs entered to other addresses, modified HTML code, transformed the content of the latest news on news websites via a simple search-and-replace function (eg, by replacing the names of politicians) as well as the content of private e-mails that were read through web interfaces such as Hotmail or Yahoo!. During four weeks this project manipulated web access for the entire Academy —and 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 affected took the time to make the minor adjustment in order to regain access to unfiltered information. Several months after the end of the experiment web access from most of the Academy's computers was still being filtered.

Image 1. *Insert_coin* (2000) by Dragan Espenschied and Alvar Freude

My second example, *walser.php* (2002) by textz.com/Project Gnutenberg (ie, Sebastian Lütgert), has been called "political" or "literary"[12] software. It could be called anticopyright-activist software, 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-print 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 unfavourable circumstances, called back by the publisher (nice try: calling back a digital document). Rather, *walser.php* (or rather *walser.pl*) by textz.com 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 code.[13] 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 Licence. However, it may be only *executed* with the written permission of the Suhrkamp publishing house.[14]

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.online-demonstration.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.
13. http://textz.com/trash/walser.pl.txt.
14. *Cf.* textz.com: "Suhrkamp calls back walser.pdf, textz.com releases walser.php", http://textz.com/trash/readme.txt.

```
<?

/*
    walser.php v1.01 (includes makewalser.php)
    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 suhrkamp verlag, frankfurt am main, germany.
    source code: http://textz.com/trash/walser.php.txt
    release notes: http://textz.com/trash/readme.txt
    bug reports: http://heise.de/newsticker/foren/go.shtml?list=1&forum_id=30648
    no copyright 2002 textz.com
*/

set_time_limit(0);

$z .= "64000a20212728292c2d2e2f303132333435363738393a3b3f414243444546474849 4a4b4c4d4e4f";
$z .= "505152535455565758595a6162636465666768696a6b6c6d6e6f707172737475767778797a849293";
$z .= "9697a9b4c4d6dcdfe0e1e4e7e8e9edf1f3f4f6fbfc0c022635474a40460a373c48504351103b5147";
$z .= "154a4f554d5c1b3c5e5662585b566466244d6564596 72a466e707264314877697066 6d6c394544 45";
$z .= "483d8e4069867a857f76838749757f8d887e85506c9383918f8b9b999c5a8b985d808f989e626e6d";
$z .= "6e0d021a40413b08303e3d4350420f554f5344484c46525b4d57201c555b615155646161585a685c";
$z .= "295d5b6e2d61637131856466767 86b7b827e7872423e72748342a2797a7a848b84827d83818b4f7b";
$z .= "8f9396958 58c99589b98a194915e9294a261b695994346453 3c4b453f0a3e504e404610385650 47";
$z .= "4a5a5452195e584f1d3953615e64575b59632c28596e5d632d62676d7a666e716977376 16d727670";
$z .= "443d5f74797f436d797e82487c7e8d4c7982908a8594538785978c5891975b949e94939d94969ba1";
$z .= "9943031f434743080c3d4f4d3f450f2b4f55514a56464c504d1f1b3e5558605e565a5e6025645a5c";
$z .= "6a2a5b6960627064305d67756a666e796d773f3c7b757c744184757b867e7c8b84827d83814e6b84";
$z .= "8e868a908d8c9b958f5a8e909f5e8a94a29d939a994403453945484 63c4e5444414f52104f454755";
$z .= "155a545b4d5b1b47515f65545e5567615b255b635d646e6d6b6b677263696775355e7f7a7c6e7770";
$z .= "3d837381718374787d897b0b4e4a90808e9387848c87aa8f988e8d9b59988e909e5ea494a293a498";
$z .= "9d453747054c3b493c3e48130c363f535a25122d565a49522c19405d555b625454225b6125425b69";
$z .= "655a68742c382f3b313e3341354437473987 3b454d3e484f414b51444e534751554a54574c594e6b";
$z .= "a479536d73705c587278755c7f787d837b62838886231d201f290730272d240b330e391035122c14";
```

Image 2. *Walser.php* (2002) by textz.com/Project Gnutenberg

Image 3. *Sealed Computers* (since 1992) by Amurizio Bolognini

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 remain unconnected to any kind of output device (eg, 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 and Freude's experiment in filtering and censorship of internet content points to the relatively unlimited potential for control 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 produces a dystopian scenario in the form of manipulated software, textz.com with its *walser.php* project develops genuinely utopian "countermeasures 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 predominating in the field of design. Philip Galanter for example, whom I quoted above, 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 Politecnico di Milano University and organiser of the *Generative Art*[16] conferences, defines "generative art" as a procedural 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, the *Codemuse* website also 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 from 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 serve to avoid intentionality and produce an unexpected, arbitrary and inexhaustible diversity of form. The *n_Gen Design Machine* by Move Design, submitted to the *Read_Me* Festival 2003 in Helsinki, or 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]

15. Cramer, *ibid.*, p. 77.
16. *Cf.* http://www.generativeart.com/.
17. Soddu, C.: "Generative Art and Architecture", Abstract, undated, http://www.nyu.edu/studio/generative.html.
18. http://www.codemusenet/html_files/GenerativeArt.html [Dec. 23, 2003].
19. Cornelia Sollfrank, *net.art generator* (1999), http://soundwarez.org/generator/.
20. *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.

insert_coin and *walser.php* go beyond such definitions of "generative art" or "design" insofar 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 to question software and code as culture, and to question culture as implemented in software. To do so, they develop "experimental software" (in *insert_coin* a proxy server and in *walser.php* a Perl script), which not only generates arbitrary surfaces but which critically investigates the technological, cultural or social impact of software. Furthermore, the writing of "experimental software" is closely connected to *artistic subjectivity*, as can be seen in the use of different private languages, and less so with proving evidence of machinic creativity (whatever that 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] as the emphatic definition by Florian Cramer and Ulrike Gabriel states, 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 which you can see in Table 1.

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 soft-

Table 1.

Generative art	*Software art*
Focus on the surface ("phenotext") created by a generative process ("black box problem")	Focus on the generative process (set in motion by a "genotext") which may generate surfaces or other results
Software as a pragmatic/neutral tool serving to create a certain result; the tool itself is not being questioned	Software as culture which is being questioned; interest in aesthetic and political subtexts; software can be "experimental" and "non-pragmatic"
Software as a pragmatic-generative tool	Software or code as a work of its own (possibly experimental)
Efficient code ("beautiful algorithms"*)	Code as excess, code as extravagance, not necessarily efficient
Employment of generative processes in order to negate intentionality	"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, software art is highly concerned with artistic subjectivity and its reflection and extension into generative systems."** (Cramer/Gabriel)
Fascination of the generative	Interest in the "performativity" of code

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

** Florian Cramer / Ulrike Gabriel, quoted after Andreas Broeckmann, "On Software as Art", in: *Sarai Reader 2003: Shaping Technologies*, New Delhi 2003, pp. 215-218, here: p. 216

21. 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.
22. 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 Goruinova and Alexei Shulgin (Moscow 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 websites. 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 Gershuny and Ira Schneider (http://www.radicalsoftware.org/ [Dec. 31, 2003]).
23. 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].

ware"[25] or "unpragmatic" and "irrational"[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' artwork, but rather as a generative material consisting of machinic and social processes. Software art can be the result of an autonomous and 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 the 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: the 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]

Performativity of the Code v Fascination with the Generative

The current interest in software, according to my hypothesis, is not only attributable to a fascination with the generative aspect of software, ie, 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 the *performativity of code* I mean its ability to act and perform in terms of speech act theory.

I am thinking here of a series of lectures by John Langshaw Austin at 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 speech act theory meets code's assumed performativity: "[when] a word not only means something, but performatively generates exactly that which it names".[31]

Austin identifies three distinct linguistic acts in all speech acts: the locutionary,[32] the illocutionary[33] and the perlocutionary[34] 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

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': "if 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 italics].
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 such as: "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.

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.

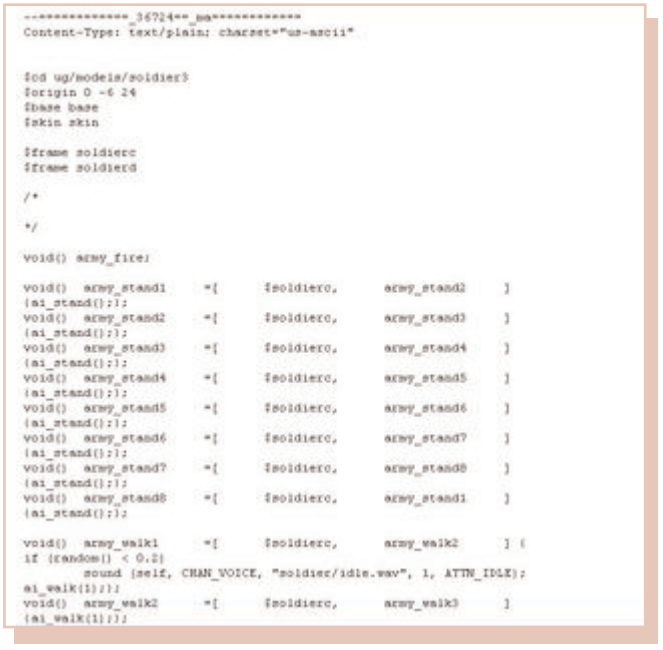
According to Austin, speech acts can also be acts, without necessarily having to be effective (that is, without having to be 'successful'). If these acts are unsuccessful, they represent failed performative utterances. Thus, speech acts are not always *effective* acts. "A successful performative utterance [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, nonetheless, non-executable code also has a purpose.

Accordingly, when I speak of the performativity of code, I mean that this performativity is not to be understood as a purely technical performativity, ie, it does not only happen in the context of a closed technical system, but affects the realm of the aesthetic, political and 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 even "kills", a process.[40] This "coded performativity"[41] has immediate and 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 immobilises* its users. Code thus becomes Law, or, as Lawrence Lessig put it in 1999, "Code [already] is Law".[42] This is the reason why software art is more interested in the "performance" than in the "competence" (terms coined by Noam Chomsky), more interested in the *parole* than the *langue*[43] (the 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 *insert_coin* and *walser.php* projects, 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 the *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 the surfaces or graphical user interfaces usually created by this code. Works by Jodi, Netochka Nezvanova, aka antiorp, and mez[46] thus highlight the existence of a hidden, "invisible shadow world of process",[47] as Graham Harwood has called it. Technically speaking, these "Codeworks" are located at the opposite end 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 the supposed executabil-

36. The work on speech acts was extended by John Searle in his *Speech Acts* (1969) book. He attempted to classify speech acts in terms of 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" on 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.

ity, thus functionality, and non-executability, ie, 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, the non-specialist reader remains unaware of whether in another location on the computer this text could in fact be compiled, and thus turned into machine-readable algorithms, and, ultimately, executed.



```
Content-Type: text/plain; charset="us-ascii"

$cd ug/models/soldier3
$origin 0 -6 24
$base base
$skin skin

$frame soldierc
$frame soldierd

/*

*/

void() army_fire;

void()  army_stand1     ={      $soldierc,      army_stand2     }
(ai_stand());
void()  army_stand2     ={      $soldierc,      army_stand3     }
(ai_stand());
void()  army_stand3     ={      $soldierc,      army_stand4     }
(ai_stand());
void()  army_stand4     ={      $soldierc,      army_stand5     }
(ai_stand());
void()  army_stand5     ={      $soldierc,      army_stand6     }
(ai_stand());
void()  army_stand6     ={      $soldierc,      army_stand7     }
(ai_stand());
void()  army_stand7     ={      $soldierc,      army_stand8     }
(ai_stand());
void()  army_stand8     ={      $soldierc,      army_stand1     }
(ai_stand());

void()  army_walk1      ={      $soldierc,      army_walk2      } {
if (random() < 0.2)
        sound (self, CHAN_VOICE, "soldier/idle.wav", 1, ATTN_IDLE);
ai_walk(1);};
void()  army_walk2      ={      $soldierc,      army_walk3      }
(ai_walk(1));
```

Image 4. *Walkmonster_start* (2001) by Jodi

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 also 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 readers themselves. (Parallels can be seen in J.R. Searle's 1969 further development of *Speech Acts*).

In contrast to generative art, software art directs our attention to the fact that our (media) environment is increasingly reliant upon 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 oscillation between simplicity and totality of execution, the "Codeworks" and, generally speaking, software art as a whole point to the potentially totalitarian dimension of algorithmic program code, the "invisible shadow world of process".

Recommended citation:

ARNS, Inke (2005). "Code as performative speech act". *Artnodes,* issue 4 [article online].
DOI: http://dx.doi.org/10.7238/a.v0i4.727

48. Cox, G. / McLean, A. / Ward, A.: "The Aesthetics of Generative Code" (2000), http://generative.net/papers/aesthetics/.

Inke Arns
Artistic director of Hartware MedienKunstVerein
inke.arns@snafu.de

Inke Arns (PhD) is an independent curator and author focussing on media art, net cultures and Eastern Europe. 2005 saw her made artistic director of Hartware MedienKunstVerein in Dortmund, Germany (www.hmkv.de).
After spending four years in Paris (1982-86), she studied Eastern European cultural studies, Slavistics, Political Science and Art History at the Free University Berlin and the University of Amsterdam (Erasmus scholarship 1992); 1996 M.A. thesis *Neue Slowenische Kunst* (NSK) *–analysis of their artistic strategies in the context of Yugoslavia in the 1980s* (published 2002). 1998-2000 PhD grant from the Berlin Senate (NaFöG). 2000-2001 lecturer at the Institute of Slavistics at the Humboldt University, Berlin, Germany. 2002-2004 guest-lecturer at the Hochschule für Grafik und Buchkunst (HGB), Leipzig.
In 2004 she completed her PhD degree at the Institute of Slavistics at the Humboldt University, Berlin, Germany. Her dissertation, entitled Objects in the Mirror may be Closer Than They Appear: The Avant-garde in the Rear View Mirror, researches a paradigmatic shift in the way artists reflect the historical avant-garde and the notion of utopia in visual and media art projects of the 1980s and 1990s in the former Yugoslavia and Russia.