Chinese Whispers.
Semiotically Mediating Between Idea and Program

Matthias Krauss, Frieder Nake, Susanne Grabowski
Informatik Universität Bremen
P.O. Box 330440
D-28334 Bremen, Germany
{krauss, nake, susi}@informatik.uni-bremen.de

Abstract

The relation of aesthetics and computation has intrigued researchers, artists, and philosophers throughout the history of Western mind. In a long-range project, compArt*, we take computer art as a case to explore and create sites for learning, art, and programming. We set up situations for students to gain insight into the algorithmic fabric of certain classes of graphic art. An indicator of the students‘ comprehension is the reformulation of hidden algorithms. An easy exercise is to use tools of a GUI to generate replicas. A greater challenge is a precise symbolic description of classes of graphics. We use the children's play of Chinese whispers as an interface metaphor for the description of data flow. The user is not required to know anything about data flow when he sets up a communication structure between agents and thus creates a total algorithmic behavior from local semantics. We bridge the gap between vague idea and precise formalism by semiotic embedding.*

1. Introduction

Programming may be considered the activity of transforming a vague idea into the precise description of an algorithmic process. The question, "How to make our ideas clear", allows for a simple answer: write a program! This answers Peirce‘s question [16] because the computer now serves as an impartial interpreter.

If we accept this proposition, the question is one of algorithmic communication or of communicative algorithmics. The situation between a user and a software artifact appears as if it was a communicative one, yet it is nothing but algorithmics. The difference is: communication usually allows for free interpretation while algorithmics yields exactly determined responses.

The linguistic means for programming must therefore be capable of double-faced expression. Such means should permit the expression of vague ideas and the precision needed to control a computer, simultaneously. A user who is an expert in her subject domain, but no expert in programming, should still find a way to tell a computer what it is supposed to do. Descriptive means must allow for an intuitively correct interpretation by the user. To the computer, however, they must determine exactly one operation without any interpretative freedom.

"Do what I mean, not what I say", is a nice way of saying the same. But this slogan is misleading. It suggests that the machine could be capable of finding out about secret desires and transforming them into formal operations. We do not subscribe to such a view of the communicative situation between computer and user. Our approach to end user programming takes a different starting point: We view the situation semiotically, as a sign process. We don‘t take the approach of tough education nor that of the intelligent machine.

We start by asserting that human user and computer are considered two autonomous systems. Either system is characterized by capabilities in which it cannot be outperformed. Humans are superior to computers in all respects that rely on the bodily experience of being-in-the-world [7]. Nevertheless, humans have developed computers to incredible algorithmic efficiency. This efficiency is, however, restricted to the precisely defined field of computability.

As a consequence of the autonomous-systems assumption, each system belongs to the environment of the other one. They are losely coupled in pursuit of their respective processes. The user‘s interest is of an instrumental character: he wants the computer to carry out particular tasks. To achieve this, he must control the interface which appears as a soft layer to protect the computer against users' errors or misinterpretations. They are steps in the usual state of affairs and appear as erroneous only from a more general perspective, e.g. from an efficiency consideration.

A human-*centric* perspective views everything from the human‘s position. It is slightly different from a human-*centred* perspective. Centredness looks at the human from the outside and observes her in pursuing the task. A certain amount of detachment remains. Centricness, on the other hand, views the task from the user‘s perspective and subsumes everything to that.

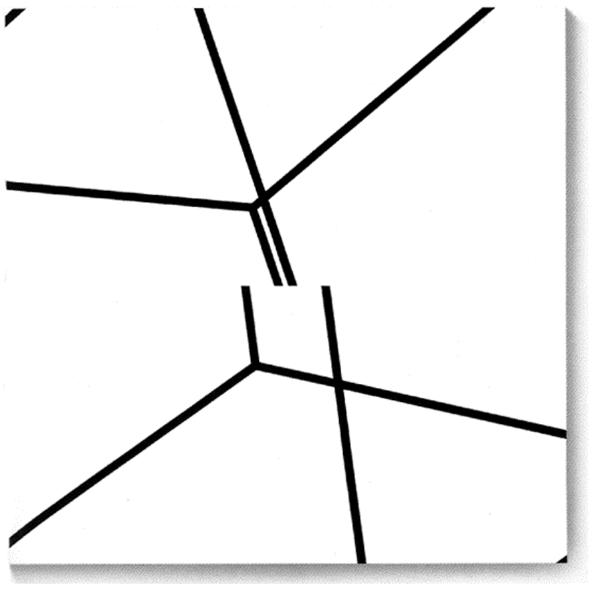

Fig. 1. Example of algorithmic art used to probe students' comprehension of algorithmic contents (Manfred Mohr: From series P-200-K, 1977-79. Permission by the artist)

The work task is socially determined (not technically) even though the machine is made to carry out important steps. This generates a tendency of considering the task as the activity of the computer. End-user programming appears as an attempt to revert this false tendency.

Since end-users are distinguished from systems programmers by lack of technical expertise, they have no way but to metaphorically approach the computer.

Over the past 20 years, *metaphor* has become an important concept in software[1]. We will take up the metaphor of Chinese whispers to talk about a programming situation in a playful way. We look at the data flow of a program as a Petri net without using the terminology of Petri nets. The metaphor should make that possible. Our current application is algorithmic art, particularly that of Manfred Mohr. Students are supposed to identify algorithms behind computed works of art.

We first give some background to the application domain. In section 3, we give a direct manipulation interface to the algorithm. Section 4 explains the Chinese whispers metaphor. We discuss educational issues connected to the approach and generalize to programming issues in section 6. In section 7, we briefly look at related work, before we draw a conclusion.

[1] A general remark on metaphor may be in place. For detail we refer the reader to [11] and, much briefer, to [14]. Coyne, in the subtitle of his remarkable book [3], announces a paradigmatic shift of the focus of systems design. Before these, Canfield Smith had introduced powerful metaphors that have ever since governed graphic interfaces [19]. As we use it here, a metaphor is a sign, or system of signs, to denotate an object (or system of objects) in such a way that two interpretants are invoked. The two interpretants belong to distinctly different cultural domains. The *similarity* of the interpretants is entirely subjective. We hope for some kind of *transfer* from the source domain of the first interpretant to the goal domain of the second. The source domain should be familiar to users. Metaphor is supposed to facilitate learning by analogy.

The work described here is part of a project aiming at a hypermedium as a space for computer art. We are interested in issues of learning about aesthetics and algorithms. The contribution of this paper is not to go beyond the state of visual data flow languages. Others have more to say about that, and the language of our project is nothing remarkable. We try to take human-centricness seriously. This means to use metaphor in a wrap-in manner and view programming as a semiotic activity. Our contribution is to identify the interaction between human and computer as a primarily semiotic exchange hiding instrumental aspects of programming. For this aim we base language on metaphor, not the other way.

2. A case from algorithmic art

Have a look at Figure 1. What could it possibly show? You observe straight black lines in a seemingly random arrangement. You also detect vertices with three emerging edges; some lines look parallel to others.

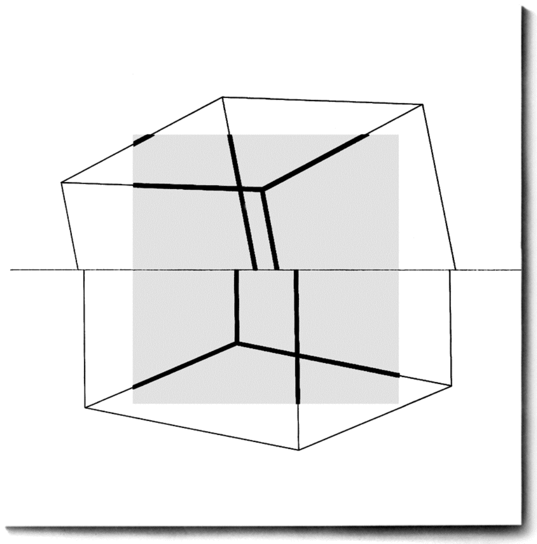

Fig. 2. The divided cube of Fig. 1, with different rotation and not clipped (Manfred Mohr: P-200-K, 1977-79. Permission by the artist)

Humans are good in interpreting pictures with a figurative content. We usually try to detect in a picture features resembling objects of our daily experience. Take, as an example, a painting like that by Leonardo da Vinci known by the name "Mona Lisa". If a typical adult in the West is confronted with this picture, he would most likely answer "Some old-fashioned lady", or even "The Mona Lisa", when asked the question "What do you see?" Could the same become true in our present case? Couldn't there be some equally obvious relation of the paint on the flat surface (form) to some object depicted by the paint (content)? In fact, there is.

We provide a bit more context. Figure 2 is a second view of the object under consideration. The two figures are similar in the central area of thick black lines. In Figure 2, the lines extend to build the parts of cubes in projection. This indicates how we should interpret Figure 1. Think of two cubes and rotate them independently. Project them onto the plane of the front face in original position, and draw just those sections of edges that fall inside that square face (grey in Fig.2). Do this in such a way that the first cube‘s projection is blanked out in its lower part whereas the second cube‘s projection is blanked in the upper half. Obviously these instructions may be turned into a precise algorithm. The content of Fig. 1 is detected by embedding the lines into context.

We present pictures like Figure 1 to students and ask them: describe an algorithm that generates such pictures. We hope to motivate students to take a thorough look at the work of art. By asking them to determine an algorithm they are bound to make explicit their understanding.

Figures 1 and 2 are representations of paintings by Manfred Mohr, a New York based German artist who has been using computers for more than thirty years [10]. Many of his pieces relate to algorithms in subtle ways. It is important to compare series of Mohr‘s paintings or prints. Only the serial principle is capable of presenting the unity in variety that Mohr is interested in. The algorithm behind the scene stands for the generative principle and the content shared by all pictures of a series. It keeps them together and may be viewed as the secret to be detected in those works of art.

To be sure, the preference given here to this artist‘s works is due to their clear algorithmic nature. Algorithms have been in use to produce works of art since the mid 1960s (one of the authors has contributed to the first such experiments). There is a wealth of international literature on the subject, e.g. [12], [21]. The journal, Leonardo, publishes regularly on procedural art. Within the broader context of the compArt project we pursue other aspects of algorithms and beauty.

We now turn away from interpreting Mohr‘s paintings by only looking at them and thinking about them. We arrange a situation of direct manipulation that should support understanding in a learning-by-doing manner [6]. The doing here is semiotically determined. It is not about oil on canvas.

3. A direct manipulation interface

Figure 3 shows a screenshot from an implementation that we prepared for the experiment. After starting the program, the student is confronted with nothing but the four edges of a square. Nothing hints at the fact that the square is the parallel projection of a cube orthogonal to its face. To the right of the display we observe a column of graphic signs and numerical values. From experience with GUIs we probably assume they stand for interactive devices. Most likely, we do not immediately understand what the icons mean.

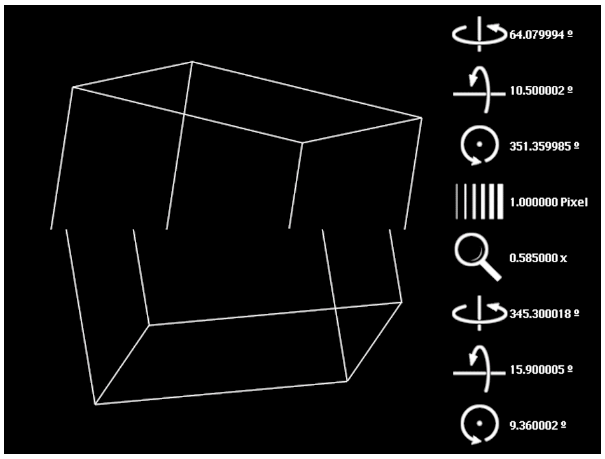

Fig. 3. A view of user interface of “Mohr‘s cubical signs”

We bet that most people, even if only little familiar with the use of computers, will grab the mouse and move it around. When they do so they implicitly ask “What can I do here?” Nothing happens if they move the cursor over the square. But the interaction devices react visibly when touched by the cursor. In all likelihood, the user will click on one of the devices thereby declaring it to become the active tool. Further movement of the cursor will produce an appropriate effect on the only object present, i.e. on the square.

When a first operation is applied to it, the square reveals an important fact. It seems to consist of two halves, which can be manipulated independently. In fact, the first three icons stand for rotations about the three axes of the upper cube, whereas the same three icons to the bottom are to manipulate the lower cube. When we rotate a cube, its projection is no longer perpendicular to the face. Some of the cube‘s hidden edges show up, and when we rotate about different axes, we obtain an image of all the edges. We discover that the projections are cut.

The two icons in the center of the panel allow to set a different line width, and a magnifying factor. All devices show their current numerical value.

Our observations of students show that, after a first period of getting used, users discover what they can do here. The exercise indicates how easy it has become to program an interactive device for structures of Mohr style. However, the user is reduced to be a *user*. He can try out parameter settings rapidly and interactively. But this will not be enough in a more challenging learning situation. We therefore take a second step to get closer to opportunities for the evolution of mental models.

4. A visual programming interface

We have developed an interface and language that we hope to be useful for learning. We describe the interface itself, and give an example of its use. The interface is based on a metaphor well-known to pre-schoolers.

4.1 Programming metaphor

Children in various countries are fond of a birthday party game called “Telephone”, but also “Chinese whispers”, in English. It is *Stille Post* in German, *Le téléphone Arabe* in French. A crowd of children line up one behind the other. The first child thinks up some funny phrase which is to become the message passed down the line. She whispers it into her successor‘s ear. He continues doing the same by whispering the phrase he has heard—whatever it was. The last child in the line utters aloud what she has perceived.

It is fun to compare the first and the last phrase. Usually they are far apart from each other. Although the rule of the game says that every child should faithfully pass the message on to the next relay station, the fun is in pretending to obey the rule but just slightly twisting the message. The children display some predetermined behavior yet they are free to do whatever comes to their mind. We have a structure of autonomous agents generating some total behavior. This makes it attractive for transfer to our programming situation.

We use the metaphor for programming by organizing a data flow. We arrange a structure of specialized agents. An agent is capable of performing a particular task and has access to several phones. It is connected to other agents via phone lines. Phones are distinguished as input and output. On an input phone, the agent is expecting data. On an output phone it is delivering data as a result of its processing of inputs.

This is where the metaphor of Chinese whispers carries through: a message is passed down the line but each time it reaches a connecting station it may be changed. An external observer cannot tell directly. She would need to draw conclusions from observations.

To give an example, one agent is an *adder.* It is capable of adding two numbers (or other items) and deliver the result. It becomes active to do its job whenever there are numbers waiting to be processed on each of the two input phones. After having produced the sum, the agent delivers it on the output phone.

4.2 Language Structure

Programs in Chinese Whispers are constructed by combining the basic elements: agents, telephones, cables, alarm clocks, triggers, and books. Telephones, triggers, alarm clocks and books can be given to an agent as its assets. Cables are used to establish unidirectional connections between telephones, or from a trigger to an alarm clock. Triggers are activated by an agent in order to tell another agent that it should do something. When an agent’s alarm clock is triggered, it rings to awake the agent from passive state. Phone connections are for data flow, trigger connections for control flow.

Agents work on packets of data: geometric objects, axes, and numbers. Geometric objects are wire frame polyhedra. Axes are the three principal directions of Cartesian space. Numbers come as reals or integers.

Books contain data as constants. When a book is given to an agent as part of its repository, it contains some constant that the agent may look up.

An agent is highly specialized but sensitive to the kind of input it receives. An input line may offer data of types that the agent cannot handle. If the data is known to the agent, it adjusts. E.g., the adder agent can add two numbers, or superimpose two pictures. This way, agents represent generic data-driven operations.

There are agents for adding, multiplying, rotating, cutting, randomizing, comparing, and projecting. Agents change state as they do their jobs. Once activated, an agent looks for data on its input phones. If not enough data is available, it calls on someone else and asks for the data. When enough data has been collected, the agent performs its action and tells successors of the result. That may be some sort of data or an event, used to trigger someone else’s alarm clock.

We develop Chinese Whispers bottom-up. We choose an example and try to describe its solution with the currently implemented set of agents, assets, and packets. If the language means are not suited well, we improve and elaborate their definitions. At present, the language does not reach far. We neither strive for a minimalistic syntax vs. maximum generative power, nor do we currently allow for generally available programming structures (there are no loop structures, recursion, or functional abstraction).

Our focus is not language but usage. We are, of course, aware of the intricate relation between the two. But we follow the hypothesis that language will evolve from usage if the general framework is suitable. This is a consequence from the premise to adopt a human-centric position. Language should therefore follow from what it is used for, not the other way around. The childish metaphor is not meant as a trick to instruct users. It rather helps us communicate about an evolving artifact. It is our way of talking about the local semantics of primitive operations and about structure.

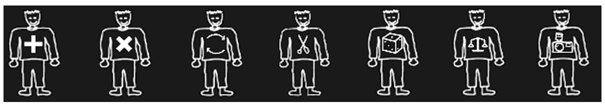
Fig. 4. Collection of agents of “Chinese Whispers”: add, multiply, rotate, cut, randomize, compare, project.

4.3 Visual elements of the interface

Figure 4 shows seven agents as currently implemented. We have chosen an anthropomorphic visualization. Human figures display their special capability on the work shirts. We will redesign the figures such that their appeal will be less sturdy.

The state of an agent is reflected in its visual appearance (waving a hand, lying in bed). We borrow from work by Beaudouin-Lafon [1] for the general appeal of the interaction. There are two mice controlling a red and a green cursor pointing left and right upwards. Each mouse can adopt the other's function. We also borrow the tool glass as a convenient device to locally arrange and change visual settings (Figure 5).

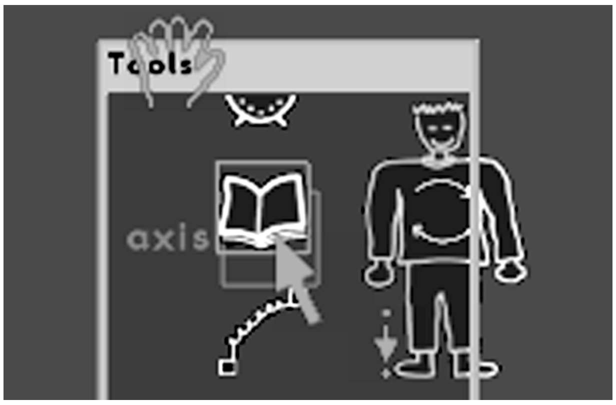

Fig. 5. The tool glass put into position to move an agent (the transparent overlay is colored on the screen)

The visual interface is equipped with three vertical bars (or palettes), one speed indicator, and the Chinese Whispers ground (Figure 6). The latter is the work area where the user places agents, assets, and phone lines and thus creates a structure for whispering.

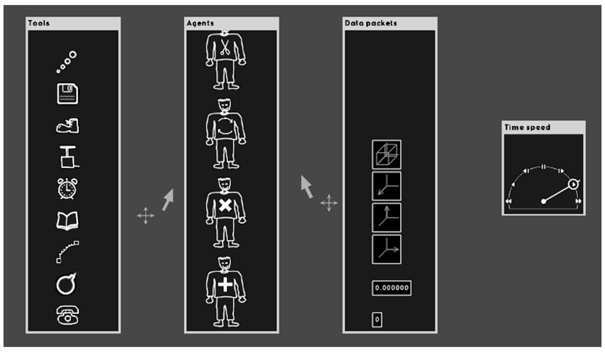

Fig. 6. Interface of "Chinese Whispers": tools, agents, packets, time speed control, and two cursors

The three bars collect tools, agents, and packets resp. It may be necessary to scroll elements within a bar. The speed indicator resembles the dial of a potentiometer.

The speed indicator is a neat device to control, and thereby test-run, the crowd of agents. In central position (hand pointing straight up), all activity is stopped. Moving the hand to the right activates all agents. The distance from neutral position determines the speed of the agents' actions. Placing the hand to the left from neutral position causes time to run backwards.

4.4 An example: Mohr's cubical signs

We take up the example from section 2. We want to arrange a stuctured crowd of agents such that the class of computer art pieces in the style of Mohr's cubes gets generated. For introductory purposes we reduce complexity. Figure 7 displays a structure of five agents that do the simplified job.

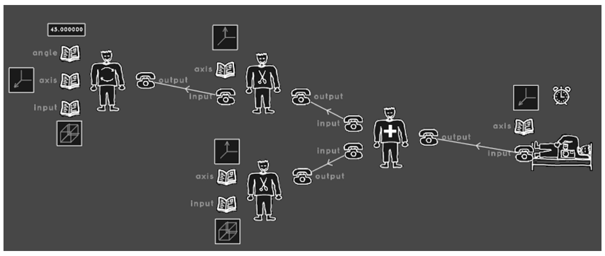

Fig. 7. Connected assembly of agents for a simplified version of Mohr's cubical signs (a data flow diagram)

To the right, a projector agent is in sleeping state. If its alarm goes off, it wakes up and orthogonally projects the object delivered to it onto the picture plane.

The next agent to the left is capable of superimposing two pictures. It calls upon two scissoring agents to provide the pictures. The lower one of them takes a cube and cuts off half of it. The upper one does the same but to an object that has to be delivered by a rotating agent. The rotator takes a cube and rotates it about the x-axis by some angle.

The working principle of the metaphor should become clear from this short exposition. Rotations about the other axes would have to be compounded for "real" Mohr cubical signs.

4.5 Interface behavior

As indicated before, we see an important task for the programming environment to tighten the link between program structure and its semantics. In semiotic terms, programs are complex signs. As such they relate a *representamen* to an *object* and generate an *interpretant* [17]. The representamen is the syntactic structure, the object is what the program execution produces, and the interpretant is what sense the user makes out of this (or puts into). The mental gap in programming is the gap between formal structure (expressed in the representamen) and vague idea (interpretant). The link between the two runs through the object. Usually what we learn in programming is to get our ideas closer to the objects that are designated by the syntactic structure. In end user programming we should be able to narrow the gap on a different level.

We have implemented mechanisms and interactive behaviors into this language and system to assist the process of describing a whispering situation. The most important aspect is ignoring the distinction between program design time and program execution time. Similar to Pygmalion [19], agents can be active without the existence of a complete program. The program graph can be modified while the program is running. The state of the program elements can be inspected at any time. The speed indicator is useful for various tests. If an exceptional situation is reached, flow is stopped automatically and feedback is given. By setting a negative speed, the data and control flow can be traced back to the cause of the unintended behavior.

5. Educational issues

We prepare an application of the system of the following kind. A group of paintings by Manfred Mohr is presented to an interested student, preferably, but not necessarily, in print. He or she is told that there exists an algorithm that generated these pictures. The job is to formulate an algorithm in the visual language of the agents. This job should be done interactively in the interface medium.

The learning situation is characterized in the following way. The student is supposed to analyze a small set of structurally similar paintings. This analysis is, by and large, a test of the student‘s ability to detect an abstract, formal principle hidden in a sample set of pictures. As a result of this phase, the student may talk about what he or she believes is the case. He (or she) expresses in plain language whatever comes to mind as a vague idea about the algorithmic background.

The second step consists of carefully formulating the algorithm in Chinese Whispers. Since the system can be changed while running, any change is immediately reflected in the behavior of some agent, and thus of the system. We may logically distinguish a third phase of learning, the phase of testing the algorithm.

To summarize the learning situation, there are elements of sensual observation, mental reflection, explicit formulation, automatic testing, correcting a formal system, and comparing constructed results with givens. These steps contain experimental as well as reflective, iconic as well as symbolic, concrete as well as abstract components. Learning succeeds in so far as the student has a chance to approach the subject matter from different perspectives. Experiencing differences and drawing conclusions from them leads to changes of ideas.

Not much actual evaluation has taken place. We have experimented with the system on an ad hoc basis. Results are encouraging but not conclusive (section 8).

6. Programming without a programming language

The student‘s activity culminates in a successful set-up of a crowd of whispering agents. This set-up is the result of an experimental, visual, interactive, and, we hope, joyful activity. Granted, it looks a bit childish. We will sooner or later know just which degree of childishness is tolerable to students, and where we should resort to more serious, i.e. more abstract, kinds of visual elements.

In any case, the crowd of agents constitutes a whispering society. Its members work cooperatively towards some common goal. A structure emerges, not necessarily the most brilliant one. But the student is implicitly asked to arrange whispering agents in such a way that their combined whispers amount to a great song, a song of successful achievement.

The student experiences the success of her efforts immediately without another person intervening. She compares the crowd's products with Mohr‘s art. The student decides when the job is done.

This state of affairs is characterized by the existence of an algorithm hidden behind the backs of the agents. We have the ingredients of any algorithmic situation: primitives that cannot be changed and do something basic, the agents; ways for these primitives to communicate what they produce, the phones; means to define structure, the phone lines; and means to express variety: the states of agents and packets of data.

These constructive elements are capable of expressing linear sequence. There is one agent doing some basic comparison (the scales). Given sequence and conditional, iteration can be expressed, if back phone lines are allowed. There is some degree of parallelism whenever agents work independently on their data. No provision is made to deal with structured data.

Students are not requested to learn an abstract syntax in order to formulate algorithmic processes. All they need is an understanding of the Chinese whispers metaphor and its use in the present context. A brief introduction should suffice for that. The metaphor and the visual direct manipulation environment offer feedback. What students do is some sort of programming but possibly without knowledge of a programming language syntax.

This is well known to the visual programming community. Our contribution is in the approach and the clientele we hope to reach: students of fine art. We stress the combination of media: reproductions of actual works should be available for immediate comparison.

The balance to strike is that of complexity versus simplicity. Our starting situation contains some works of recognized art (no toy examples). Now students are requested to produce an algorithm that demonstrably creates such pieces. Even if these miss decisive features of the given works, they should clearly belong to the same class. If this happens, the student is in a position to develop a feeling for the relation between beauty and computability. The computable formulation arrives after the fact of the work of art, but it existed before, when the artist did his programming. We expect insight into complexity to emerge from the approach.

Our premise is that one important kind of thorough understanding can be demonstrated by algorithmic expression. Algorithmic expression necessitates some formal language. Programming languages are such means. Although unavoidable in the end, we try to stay away from them for this type of learning out of the following reasons.

Learning happens as a sequence of ever more plausible hypotheses is created (the path of abduction [9]). Such hypotheses are rarely ever true of false. They are something inbetween. We *believe* them and use them to *argue* with. Students are looking for adequate descriptions, not for logically true ones. Learning is more akin to expressive means (signs) than to truth values (propositions). Semiotics is closer to this than to logic.

Since we take algorithms as a final proving ground, we are tight to logic, and precise formal syntax cannot be avoided. We acknowledge this fact but we try to hide it and wrap it into the metaphor. We base the language on metaphor and let the metaphor unfold its creative potential instead of attaching a metaphor on top of a language. The latter is what gets usually done on a merely didactic level.

We are aware of the problematic value of metaphor [2]. Not in all situations, and not for all stages, are metaphors a good advice. They remain useful for qualitative purposes when vague ideas are in the first stages of gradually becoming clear. We do not expect their effects to be quantitatively measurable. As is the case with sign processes in general, the dynamics of the situation is what matters most.

7. Related work

At times, a simple diagram highlights a problem more convincingly than a long treatise. For the relation of the user of a technical system to that technical system itself, Don Norman has presented such a diagram [15]. Between the mental “representation” of a problem and its explicit algorithmic representation, there is a gulf that cannot be bridged but has to be mastered. Going from human mental goals (vague ideas) to operations of a technical system (program) is to cross the gulf of execution. Taking the results of the program and interpreting them in the context of human ideas is to cross the gulf of evaluation.

As Norman suggests, there are two ways of bridging the gulf: move the system closer to the user, or move the user closer to the system. The usual programming course takes the second way (tough education). Many programming-by-example approaches take the first. We take a slightly different look. We doubt that there *are* representations in the brain—at least we do not know of them other than as a hypothesis. But users are confronted with all sorts of external representations. Some are offered to them by technical systems. Others are generated by himself.

The use of a software system amounts to a play with representations. This is a typical semiotic situation. Since any use of an artifact is a dynamic process, everything changes and nothing remains fixed on the user’s side. Use enforces learning. It is therefore misleading to assume that the user *has* a representation. He continually *forms* one, and, in doing so, follows an interest. Even the goal of an activity changes as the activity progresses. Therefore we abandon any static assumption in the context of usage and prefer a media perspective instead. (Problems of representation have puzzled much of AI research, viz. [8]).

Explicit representation and implicit interpretation are the topic of semiotics. A sign appears as representation of some other entity that is absent. We take an interest in representations only when we interpret. Vague interpretation and formal representation are united in Peirce's concept of the relational sign [17]. We are convinced that a treasure is to be discovered for the theory of programming in Peircean semiotics. The recent interest by a number of authors in *contexts* of software usage and design is an indication of this (e.g. [5, 13]).

On a graphic display, the user encounters visual signals. They are produced algorithmically and, at first, have no meaning. Meaning is a human product. Embedding signals into contexts generates meaning. This is what the user does in an act of permanent interpretation. Semiotics is the art and science of interpretation [14]. Fine art is on subjectively interpreting given representations. Programming is on generating precise representations of sloppy concepts. This way, programming and fine art are related inversely, and semiotics builds a bridge between them.

Within the narrower field of programming, we owe much to David Canfield Smith's pioneering work on Pygmalion ([19], see also his contribution to [4]). His invention of the icon has become a paradigmatic ingredient of all graphic interfaces. It is a genuinely semiotic concept and of great power for user immediacy.

Smith observed that “editing an artifact rather than typing statements” was easy for people. He also viewed his editing system, Pygmalion, as a medium. In both respects, he defines our starting point.

Where Smith et al. [20] and Repenning & Perrone [18] in their successful and path-breaking work on end-user programming focus on rules, we take a, perhaps, more conventional approach. Ours is to invite the user to describe, in a metaphorical way, a total algorithmic structure. It connects locally meaningful agents, and thus combines global *and* local aspects. The other approach is to concentrate on local semantics only, and leave the rest to a powerful general rule interpreter.

8. Conclusion

We have presented the metaphor of Chinese whispers as a framework for a programming exercise. Programming was hidden in so far as students had to detect an algorithm from samples. The Chinese Whispers system helps students (of fine art, or in school) to arrive at some explicitly formulated algorithmic understanding in so far as the direct manipulation interface is close to everyday habits: two hands may be used, and manipulations are directed towards objects.

We have not yet extensively tested the system. But we have collected preliminary observations in playful settings. During a seminar at the Juske Akademi for Kunst in Aarhus, Denmark, in the fall of 2000, we split participants into three groups. Each group had to solve the same problem but was given different means. The problem was, as always: find an algorithm, and describe it as rigorously as possible, that generates the class of pictures some given samples belong to.

The three groups of students differed in so far as two of them were working on the computer, whereas the third one had only paper and pencil at their disposal. The computer groups were given the first direct manipulation program (no agents yet) in different versions (with and without explanations). A major observation was that students without computer developed a more abstract understanding. Students working with the computer developed a better feeling for spatial effects of the situation. This may be attributed to the immediate appeal of rotating an object in 3D space even if this is viewed on a flat display screen. We are currently preparing for tests that will take up more complex structures.

We hide the programming task behind interactive playful operations. We borrow test cases from fine art. Taken together, we have a typical media situation. We expect to be able to show that the media perspective (as opposed to a tool perspective) is *the* end unser perspective. Since media are complex signs, semiotics becomes important in developing frameworks for programming without explicit attention to programming language. Much of this has long been around with the visual languages community. Our additional expectation is that understanding visual expressiveness can gain from fine arts and semiotics, and from their combination.

Acknowledgement. We are indebted to Manfred Mohr for granting permission to reproduce his work. During a stay in 1998 at the University of Colorado at Boulder, two of us have learned a lot from discussions with Alexander Repenning. Important hints from anonymous reviewers helped us improve the paper.

References

[1] M. Beaudouin-Lafon: Instrumental interaction: an interaction model for designing post-WIMP user interfaces. *Proc. CHI 2000*, April 2000, pp. 446-453

[2] A. F. Blackwell, T. R. G. Green: Does Metaphor Increase Visual Language Usability? *Proc. 1999 IEEE Symp. Visual Lang.*, 246-253

[3] R. Coyne: *Designing Information Technology in the Postmodern Age: From Method to Metaphor*. Cambridge: MIT Press 1995

[4] A. Cypher (ed.): *Whatch What I Do. Programming by Denmonstration.* Cambridge, MA: MIT Press 1993

[5] B. Dahlbom, L. Mathiassen: *Computers in Context*. Cambridge, MA: NCC Blackwell 1993

[6] J. Dewey: *Democracy and Education*. New York: The Free Press 1997

[7] H. L. Dreyfus, St. E. Dreyfus: *Mind over Machine*. New York: The Free Press 1986

[8] J. A. Fodor: *Representations. Philosophical Essays on the Foundations of Cognitive Science.* Cambridge, MA: MIT Press 1981

[9] J. R. Josephson, S. G. Josephson (eds.): *Abductive Inference. Computation, Philosophy, Technology*. Cambridge University Press 1994

[10] M. Keiner, Th. Kurtz, M. Nadin (eds.): *Manfred Mohr.* Zürich: Waser-Verlag 1994

[11] G. Lakoff, M. Johnson: *Metaphors We Live By*. Chicago: University of Chicago Press 1980

[12] P. McCorduck: *Aaron‘s Code. Meta-Art, Artificial Intelligence, and the Work of Harold Cohen*. New York: W.H. Freeman 1991

[13] B. A. Nardi (ed.): *Context and Consciousness. Activity Theory and Human-Computer Interaction.* Cambridge, MA: MIT Press 1996

[14] W. Nöth: *Handbook of Semiotics*. Bloomington: Indiana University Press 1993

[15] D. A. Norman: Cognitive Engineering. In: D. A. Norman, S. Draper (eds.): *User Centered System Design*. Hillsdale, N.J.: Erlbaum 1986

[16] Ch. S. Peirce: How to make our ideas clear. In: J. Buchler (ed.): *Philosophical Writings of Peirce*. New York: Dover Publ. 1955

[17] Ch. S. Peirce: The Theory of Signs. In: J. Buchler (ed.): *Philosophical Writings of Peirce*. New York: Dover Publ. 1955

[18] A. Repenning, C. Perrone: Programming by analogous examples. In: H. Lieberman (ed.): *Your Wish is my Command. Programming by Example.* San Francisco, CA: Morgan Kaufmann 2001

[19] D. Canfield Smith: *Pygmalion: A Computer Program to Model and Stimulate Creative Thought*. Basel: Birkhäuser 1977

[20] D. Canfield Smith, A. Cypher, L. Tesler: Novice Programming comes of Age. In: H. Lieberman (ed.): *Your Wish is my Command. Programming by Example.* San Francisco, CA: Morgan Kaufmann 2001

[21] A. Spalter: *Computers in the Visual Arts*. Reading, MA: Addison Wesley 1999