On Signifying the Complexity of Inter-Agent Relations in AgentSheets Games and Simulations

Marcelle P. Mota, Ingrid T. Monteiro, Juliana J. Ferreira,
Cleyton Slaviero and Clarisse S. de Souza
Semiotic Engineering Research Group
Departamento de Informática, PUC-Rio
Rua Marquês de São Vicente 225
22451-900 Rio de Janeiro, RJ - Brazil
{mmota, imonteiro, jferreira, cslaviero, clarisse}@inf.puc-rio.br

ABSTRACT
This paper reports the results of an empirical study about the semiotic engineering of signs of complexity for live documentation of games and simulations built with a visual programming learning environment. The study highlights the essence of the semiotic engineering process and shows how its outcome has been received by a group of users who can speak for a large portion of the live documentation system's user population. It also shows how the communication of complexity is, in and of itself, a major design challenge, especially when mastering complexity is one of the prime purposes of the documented object. Because the study was carried out in the context of a live documentation system, its conclusions can also illustrate how to conduct semiotically-inspired interaction design.

Categories and Subject Descriptors
H.5.2 [**Information Interfaces and Presentation**]: User Interfaces - Theory and methods; Training, help, and documentation. K.3.2 [**Computers and Education**]: Computers and Information Science Education.

General Terms
Documentation, Design, Human Factors.

Keywords
Meaning of program representations, Semiotic engineering in practice, Live documentation, Computational thinking acquisition, AgentSheets.

1. INTRODUCTION
The importance of developing children's computational thinking (CT) skills at school has been repeatedly emphasized over the last few years [12][13]. The aim of an increasing volume of research and technology has been to facilitate the teaching and learning of basic computer programming, an inherently complex cognitive activity even if achieved with the aid of 'fun tools' like toys and robots, games and animations. The challenge for researchers and educators is not so much one of *simplifying complexity*, but rather one of providing the appropriate means to *deal with complexity* and *gain mastery* of computers.

This paper reports on empirical research with program representations in AgentSheets, which was carried out to inform the design of extensions to its accompanying live documentation system, PoliFacets. AgentSheets is a visual agent-oriented programming environment with which learners can build games and simulations [17]. PoliFacets supports the exploration of various aspects (facets) of AgentSheets programs, like the depiction and behavior of agents, the structure of game space, the experience of game play and the program code [11]. The interest of this research for design and communication studies is that we follow directives proposed by Semiotic Engineering, an HCI theory that views human-computer interaction as a specific kind of computer-mediated human *communication*. With it systems producers *tell* systems users their design vision as well as how, when, where, why and what for they can use the system [3].

The purpose of the proposed extensions to PoliFacets is to help CT learners and teachers detect and understand the sources of program complexity. One of the main sources of complexity in games and simulations built with AgentSheets is to define and control how agents affect one another when the program is executed. Together, AgentSheets and PoliFacets provide various kinds of representations for understanding *local* and *global* relations, the former being expressed within an agent's behavior rules, and the latter throughout the entire game program. The leap from one to the other, however, is cognitively very challenging for learners. We thus set out to create an intermediate level of representation to deal with a limited scope of inter-agent relations, larger than *local*, yet smaller than *global*. Representations of *regional* inter-agent relations are thus meant to support explorations of bounded chains of influence that one agent has upon the behavior of other agents. By using them, learners (and teachers) should have more options to divide and conquer program complexity while trying to understand and explain the logic of games and simulations.

SIGDOC 2013, September 30 - October 01 2013, Greenville, NC, USA

http://dx.doi.org/10.1145/2507065.2507070

In the following sections we briefly describe AgentSheets and PoliFacets (section 2), then report how we used a qualitative methodology to carry out a formative evaluation study with users and finally highlight how its procedures and results elicit and express the quality of communication we have achieved (section 3). In the last section we discuss our contribution in view of relevant related work dealing with the signification and communication of complexity in computer-supported program comprehension tasks.

2. AGENTSHEETS AND POLIFACETS

AgentSheets is a visual programming tool specifically designed to promote CT acquisition through the development of games and simulations. Programming is mainly done through drawing and direct manipulation of interface elements and the ultimate targeted users are school children [18].

Although program representations in AgentSheets are very rich (which is appropriate for learning environments), previous studies about this system's interface [5][6] have shown that the overall *communication* of the role and meaning of such representations, as well as that of relations between them, could be improved. The technical outcome of such studies was a live documentation system called PoliFacets [11], with which teachers and learners can explore how the meanings expressed during game play or simulation execution have been *encoded* in program structures by their creators. PoliFacets thus supports *reflection* upon AgentSheets programs, a critically important element in the overall CT learning process. It has been implemented as a Web extension to AgentSheets, following previous successful experiences with the Scalable Game Design Arcade, a Web-based cyber learning infrastructure where learners can assess their progress based on automatically extracted CT patterns that they have used in their games and simulations [2].

To illustrate how AgentSheets and PoliFacets work and relate to each other, we will use a simulation of how industrial pollution affects the environment. An important aspect of this simulation is that it is only *an expression of its creator's understanding* of environmental damage caused by atmospheric pollution and by no means a *computational model* of the actual chemical processes in place. Likewise, the names of program elements have been arbitrarily chosen by the programmer; they aren't necessarily natural language words denoting what such elements *mean* to the programmer or the game players. This characteristic is very important as will be seen in subsequent sections of the paper.

2.1 Creating Simulations with AgentSheets

AgentSheets games and simulations consist of two fundamental components. The first is a set of one or more *agents*, which have a visual depiction (possibly many) and whose behavior is defined by if-then production rules. The other is a set of one or more *worksheets* (or game spaces), where agents are deployed at programming time and where they perform at run time. In Figure 1 we show the deployed worksheet of the environmental pollution simulation we have used in our study. In it there are *agents* like trees and clouds, for instance, placed on top of a background image with a green field, blue sky, wild flowers, and so on.

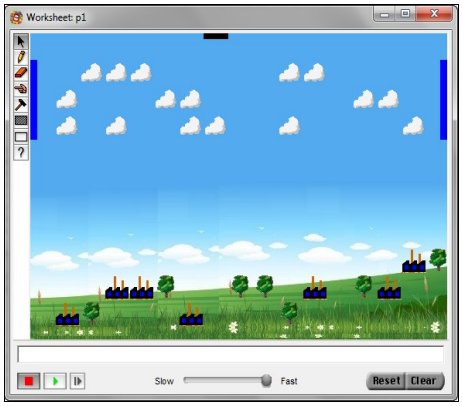

Figure 1. The worksheet

Although, as mentioned, all agents behave in accordance with if-then rules established by the programmer, some may have *void* behavioral rule. This is the case of agents whose sole purpose is to compose the structure of the game space or constrain the behavior of other agents. As an illustration, Figure 2 shows the agents gallery and part of the behavior of agent 'A'. Agents can have multiple depictions, as is the case with agent 'A'. It has two depictions, which are changed as the simulation executes. To specify the agent's behavior a user can build rules by dragging and dropping conditions, actions and even other rules into the appropriate slots. Rules are formed by multiple rows with conditions on the left (If) and actions on the right (Then). When all conditions on the left side are satisfied, actions on the right side of the same row are executed in sequence.

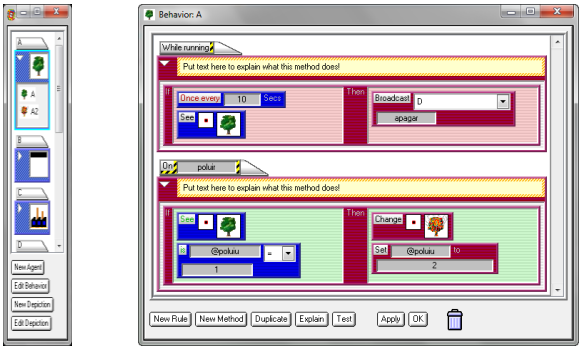

Figure 2. Gallery and part of the behavior of agent 'A'

Figure 2 (when in colors) also shows the effects of AgentSheets' conversational programming style. While a game or simulation is running, the user can see if rule conditions are true or false for a specific agent at particular moment in time. Sequentially tested rules are shown in green (bottom of Figure 2) if they are true and red if they are false (top of Figure 2). Color and animation help users understand why agents do (or don't do) specific actions at run time.

Furthermore, AgentSheets allows users to generate a report with a list of all agents' depictions and behavior rules (see a snapshot in Figure 3). The report is displayed as a Web page, with hyperlinks for quick access to related parts of the report. The language used in the report is exactly the same visual language as used in AgentSheets' programming interface.

Figure 3. A snapshot of AgentSheets' program report

In spite of its interactive attractiveness, CT learning processes with AgentSheets might benefit from extended representations [6]. This is mainly because while AgentSheets makes *action* easy to take, it doesn't support *reflection on action* [20] to the same extent. This finding has seeded the development of PoliFacets [11].

2.2 Exploring Program Facets with PoliFacets

PoliFacets is a Web-based active documentation system for AgentSheets programs. Once games are uploaded, a range of *facets* are automatically generated. The system allows users to explore such facets by following structured *conversational threads* about facets and significant relations between them. For example, they can ask questions like 'How many agents are there in this game? And what do they do?' or 'Are there stacked agents on the worksheet? Where?'. Content provided in conversations can be automatically generated by the system (based on the parsing and analysis of uploaded games and simulations) or be included by students and instructors, *a posteriori*, by means of structured annotations to program facets.

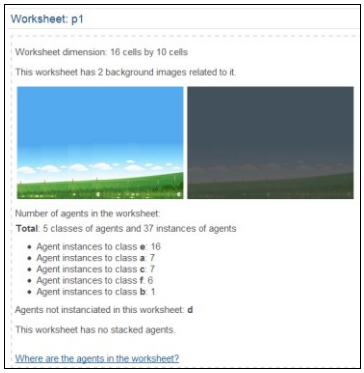

Figure 4. Worksheet details

Insights and understanding promoted by PoliFacets are not easily obtained with interactions with AgentSheets. For example, the worksheet, a critically important component in a game or simulation, may have alternate background images that are switched under certain conditions to produce great visual effects. However, in order to *see* if and how background images are used while analyzing the program in AgentSheets a learner should not only be able to separate (or visually *parse*) the background image from forefront agents in Figure 1, for example, but also spot the specific rule in one of the various agents' behavior where there is a command to switch background image. In PoliFacets this is made immediately clear when the user explores the worksheet facet (Figure 4). Note that other relevant details of the game space, which may actually go unnoticed in the AgentSheets' programming interface, are explicitly represented and communicated (*e. g.* how many cells there are in the worksheet, how many instances of different classes of agents are deployed, etc.).

In Figure 5 we see a representation of the worksheet in *grid* style, showing the exact positioning of all selected agent intances (compare this structural view with the game space rendition in Figure 1, against the background image that is not seen in Figure 5). In PoliFacet's grids, the user can explore different renditions of the worksheet (viewing the number of existing agent instances in each class, where the agents are located, enabling and disabling agents, viewing and hiding agent stacks, etc.). All such explorations contribute to grasping how the programmer has represented and structured the message that he wants this simulation to communicate.

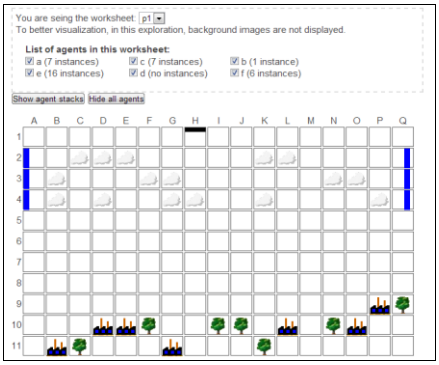

Figure 5. The grid

Another facet called *rules* presents a natural language textual rendition of AgentSheets' visual rules. The rule content is the same, but the variation in form (from visual to textual) supports a more fluid articulation and communication of the rule's *logic*. Figure 6 shows the English translation of the rule shown in the insert of Figure 3. This feature supports different teaching strategies. For example, teachers can begin with the visual programming and then resort to textual rule descriptions when students are *creating* games, but do it the other way around when students are *analyzing* (their own or someone else's) games.

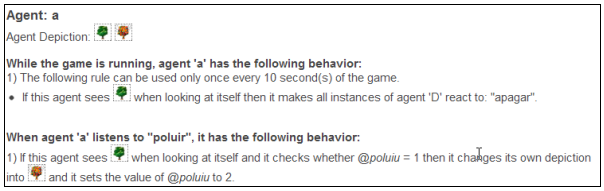

Figure 6. Part of the rules of agent A

One of the main features of PoliFacets [11] is the emphasis on *communicating the designers' intent and message* (namely to stimulate and facilitate reflection upon meanings of/in game and simulation programs). This is achieved by carefully following a Semiotic Engineering perspective [3] on interaction design. Part of this perspective involves having a keen eye for how AgentSheets *signs* are communicated and received by users, especially in view of the overall complexity of programming. Besides various aspects already mentioned and illustrated in paragraphs above, the *signification* (choice of representation) and *communication* (expression of meaning and achievement intent) of relations between agents in the program structure is essential for mastering the cognitive complexity of programming tasks. This is even more important if we consider that visual renditions of agents' behavior during the game play can suggest logical relations that are actually

not encoded *as such* in the underlying program [5]. In other words, there may be a number of different program structures that yield identical visual effects when the program is executed. Calling the learner's attention to this is of paramount importance in CT acquisition if we want them to appreciate (and eventually master) the power of computing as a new means of expression and action is society.

Table 1: Signs of inter-agent relations in AgentSheets and PoliFacets

	Signs of LOCAL Inter-Agent Relations	Signs of REGIONAL Inter-Agent Relations	Signs of GLOBAL Inter-Agent Relations
AgentSheets	**Static:** Agent's behavior rules **Dynamic:** Conversational Programming Visualizations	**Static:** None **Dynamic:** None	**Static:** Game program report and location in Worksheet **Dynamic:** Game play (*applet*)
PoliFacets	**Static:** Textual (automatic and manually annotated) descriptions of the agent's behavior rules **Dynamic:** None	**Static:** **PROPOSED** **Dynamic:** None	**Static:** Textual (automatic and manually annotated) descriptions of the entire game and agents' locations in *grid* **Dynamic:** Game play (*applet*)

An examination of both AgentSheets and PoliFacets in their current state of development showed us that when trying to trace relations among agents – be it because some agent is misbehaving during execution (which calls for debugging) or because he wants to see what will happen if an agent's behavior is changed (which calls for programming experimentation) – a programmer has two choices. He can either take a *local* perspective and inspect agents' behavior one at a time, or take a *global* perspective and analyze the whole program structure and execution (see Table 1). Stepping from one perspective directly into the other is difficult, which motivated us to create a new sign of complexity for PoliFacets. In keeping with PoliFacets' Semiotic Engineering rationale, our proposal is meant to communicate (and support subsequent exploratory communications about) an agent's *regional* scope of influence upon other agents and to bridge an important gap in both systems. The next section reports what we have found in the ongoing course of Semiotic Engineering research to achieve this end.

3. EMPIRICAL STUDY

The semiotic engineering of a system's interface must always address two aspects of communication: the emission of the designer-to-user intended communication and its reception [4]. Therefore, in view of strong evidence that users need, want or may benefit from some particular piece of communication, the semiotic engineering process starts with the elaboration of *signs* for the emission of the designers' message. In our case this involved the elaboration of signs to communicate a bounded area of influence for all agents in a game or simulation. So, based on previous research studies and on a careful analysis of AgentSheets commands and rule structures, we built a diagrammatic representation of bounded inter-agent relations (see Figure 8 and additional details in sub-section 3.1). Next, we tested the reception of our message with a qualitative exploratory study, carried out initially with six participants.

In order to create a realistic and semiotically adequate situation for participants to engage in productive communication with PoliFacets' and AgentSheets' representations, we created a specific simulation where we intentionally introduced various kinds of inter-agent relations using moderately complex program structures. That is, we deliberately programmed mutual agent behavior with command structures that *could not* be completely figured out by looking exclusively at individual agents' behavior rules.

The six participants in the first phase of the study (P1-P6) were chosen among teachers and teacher-support team members of the *Scalable Game Design Brasil* project (SGD-Br) [19]. The recruiting criterion was that they had previous experience teaching AgentSheets to beginners or that they qualified to teach introductory lessons about AgentSheets programming. Three of the recruited participants were basic-level game and simulation programmers (P1, P2 and P3), whereas the others had more advanced knowledge (P4, P5 and P6). P4 was the most experienced participant, having taught basic and advanced classes in one of SGD-Br partner schools. P5 and P6 had relatively less experience with AgentSheets, but they had additional programming abilities. P5 had taught programming classes using Scratch [10] whereas P6 not only taught IT classes for middle and high school children regularly, but he also had an active interest in and practice with non-professional programming.

3.1 Procedure and Materials

Our study included two iterations of the following steps: (i) elaboration of the sign; (ii) sign reception test with participants; and (iii) analysis of results. The first iteration (or first phase of the study) showed us how our new message was received and what elements in it were missed or misunderstood. The second iteration (or second phase), after semiotic engineering improvements were made in our message, showed us more clearly if and how the expression of *regional* inter-agent relations can help game and simulation creators gain new perspectives on the complexity of AgentSheets' programs. In the second phase of the study we collected data from five participants, four of which also participated in the first phase.

The materials used in the study were a combination of existing AgentSheets and PoliFacets representations for a specifically designed simulation, along with a manually produced Web page with our proposed diagrammatic representations of regional inter-agent relations (with hyperlinks and tooltips for certain elements of the diagram). A picture of the simulation worksheet has already been presented in Figure 1 (sub-section 2.1) and is shown in more details in Figure 7. Examples of diagrams appearing on the manually composed Web page in the first phase are shown in Figure 8, in this sub-section, and in Figure 9, in sub-section 3.3.

The simulation represents factories that pollute the environment, calling the palyer's attention to the consequences of such pollution. The names of the agents are single letters: A, B, C, D, E and F. We avoided using meaningful names in order to stimulate participants to develop their own interpretation of what the agents are and do. As part of intentional program complexity, we used agent and game attributes or properties set to local and global program variables, as well as methods (to encapsulate a set of

rules). Variable and method names, however, were meaningful words, linked to their purpose or content. We estimated that such meaningful names would help participants interpret an agent's behavior more easily and thus be able to assign meaning to them (and possibly name them).

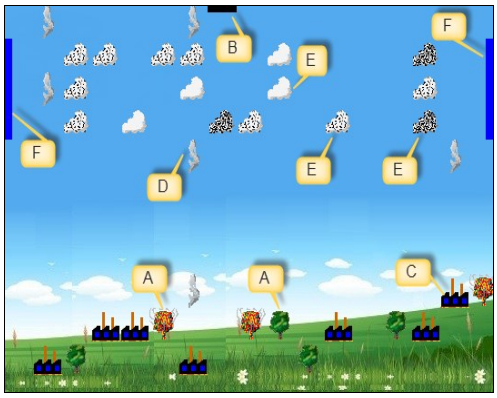

Figure 7. Agents during the simulation

In Figure 7 we show an annotated image of various agents while the simulation is running. The use of letters instead of meaningful agents' names gives the reader (as it also gave the participants) a sense of the complexity of the task, which requires that we constantly combine names and depictions in sense making. This sign-association task lies at the heart of a substantial part of professional programming activities, which justifies the case. The top-level logic of the game is as follows. If agent A is depicted as a healthy tree (), it asks agent D to execute the method "erase" (which may or may not make D erase itself, depending on certain contextual conditions). If agent D () touches agent E, E's depiction changes to the next polluted stage (to or to). Agent B () increments a variable used by C () to control the release of D () in the environment. Agent F (,) generates instances of agent E () on the left side of the worksheet and erases them on the right side. Agent E () moves from left to right under certain contextual conditions. If its depiction is that of a heavily polluted cloud (), it asks agent A to run method "pollute", which turns its depiction into that of a "dead" tree (). When all instances of A are depicted as dead trees, instances of D are no longer erased. The simulation stops when there is a column of D agents ( ) straight from some C () into the sky above.

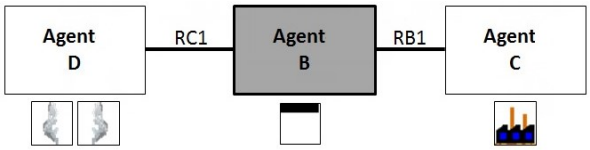

Figure 8. Connections from the agent B

Although the reader is by no means expected to tease out inter-agent relations from the preceding paragraph, the fact that there *are* significant and complex relations among agents should, however, have clearly come across. In Figure 8 we show the proposed diagram for the bounded scope of influence of agent B. Lines indicate that RB1 (rule number 1 of agent B) and RC1 (rule number 1 of agent C) establish the relations between B and C and B and D, respectively. The diagram provides navigation links to rules RB1 and RC1 in the program report (an excerpt is shown in Figure 3). RB1 connects B and C because the two agents share a property, that is, they manipulate the same global variable. RC1 connects B and D because it uses a property shared with agent B in combination with a depiction of D. Just by looking at the simulation it is not possible to identify the relations among B and C or B and D. Moreover, an examination of the entire program code (a *global* perspective) or of the involved agents' behavior rules one set at a time (a *local* perspective) makes it extremely difficult not to get lost in the logic of mutual relations *unless* we use some tracking notation like a diagram, for example.

Our intended message to the users of the proposed diagrams (as illustrated in Figure 8) can be expressed using de Souza's *metacommunication template* [3]. What we meant to say with the diagram and manipulations afforded by the user interface was:

"*We [the design team] understand that you [the user] are a CT teacher. We've learned that you need and want to have a deeper understanding of AgentSheets programs in order to decide what strategies you will use in class. We have therefore built a diagram to help you understand relations between agents, realize which ones are important and see how they influence the behavior of agents during game play. As you gain deeper understanding of this facet you will also gain improved skills to express your ideas through games and simulations. Notice how agents are related within certain boundaries – not through the entire game. You can click on hyperlinks to see the complete behavior rules that establish the relations you see. Links lead you to different facets of game representation*."

In the first iteration of the procedures, we showed them a run of the simulation and then asked them to give us a verbal description of what they saw. Next, we asked them to tell what were the relations between agents B and C, first, and A and C, second, using only the material they had at hand (the running simulation, the open simulation program in AgentSheets, the program report generated by AgentSheets and the textual rendition of the rules generated by PoliFacets). When they finished this, we showed them the inter-agent relations diagram with representations of bounded scope relations for each one of the agents. We asked them to comment on the diagrams and then tell us how agent D was related to agent F. In order to find the right answer, they should *preferably* use and manipulate the diagram, but they could also use the material provided for previous steps.

3.2 First Iteration: Analysis and Results

The qualitative analysis of the data revealed some recurring categories of meanings from the participants' discourse, which was supported by the observation of their interaction with the provided materials. Categories have been named as follows: affect; iconicity; dominant language function throughout the entire discourse; and transition across referential and metalinguistic language functions.

3.2.1 Affect

Most participants *liked* the diagram from start. A positive reaction to the diagram is important because it increases the chance that users will be initially willing to engage in this sort of communication.

P1 at the first sight of the diagram said: "I already like it! I love icons, graphs and (...) there are figures. I already like it!" P3 had the same kind of reaction: "It's very nice, very clear (...) it will [call the attention to] the need of understanding the code, (...) establishing a relation [between agents]." This participant even anticipated how the proposed diagrams might change the user's

experience with PoliFacets: "[The diagram] changed the whole [navigation in] PoliFacets (...) I mean, we begin with an image, in this case with a diagram or a graphic image, and then choose to see the code or [the rules]."

As a counterpoint, P6 reacted badly when he first saw the diagram. He said: "You take something simple and make it so complicated!" However, after he carefully analyzed diagram representations to answer test questions, P6 had a considerably different opinion: "This is good; people [can] understand what is going on (…), it is very useful (…), it is a map (…)."

One of the six participants didn't like of the diagrammatic representation at all. Even after using the diagram to answer our questions, and getting to learn more about what it meant, P2 said: "I got lost with this diagram (...) I think [understanding inter-agent relations] is an incremental success or failure [process], you see? I mean, I don't know why I would see the relations better here than in the [rules], honestly." He summarized his affective reaction to the diagram by saying: "It scares me more than supports me."

Affect, however, did no more than pre-dispose the participants positively (in most cases) or negatively (in a single case) towards trying to understand and use the diagram. The other categories brought up stronger insights on the result of our semiotic engineering effort.

3.2.2 Iconicity

Iconicity is the name we chose to express the fact that participants showed a tendency to interpret relations in a more concrete way (*i. e.* keeping similarities with physical reality) than the style of abstract symbolic signs used to name agents or to compose the diagram suggested. For example, most of participants talked about "toxic rain", "acid rain" or "cataclysm", although none of these expressions had been used in the program. We interpreted this as evidence of how strongly the participants' attention was focused on signs and agents shown during the simulation's execution, which is a natural choice for explanations centered on the *contextual* message of the simulation.

As an example, when asked to explain the simulation, P2 said: "This tiny polluting smoke [here], when it comes near the cloud, it makes the cloud [turn] darker; it pollutes the cloud. Next, you have toxic rain." We must remember, however, that the questions being asked from participants referred to how *program components* (and not domain elements) related to each other. If we look at evidence provided by P2, we see that although he was able to *guess* the relation between clouds and smoke (because the program happened to encode visual effects in a similar way as the phenomenon is articulated by common sense observation and reasoning), he failed elsewhere. For instance, he did not realize the relations between clouds and trees, nor the one between trees and the effects of calls to the *pollute* method.

Likewise, P5 (who is a Science teacher) tried to analyze the simulation scientifically. He, more than any other participant, needed to see acid rain destroy the trees. His explanation of the simulation was: "This is a simulation of factories throwing smoke [in the air]. Because I am a teacher, I know that there is CO_2, which is generating acid rain and the trees are [like] burned out and everything will be burned out." He even complained: "You showed no rain, there is supposed to be rain [falling] from this cloud." This participant's attachment to domain-centered (iconic) signs was so strong that he simply refused to look at the program and explain what the relation between selected agents in it was.

Iconicity also misled participants when trying to tell the relation between agents D ( ) and F (,). The agents themselves are, semiotically speaking, signs of different types (and intentionally so). While D ( ) is acceptably an iconic representation of *smoke*, F (,) might be the iconic representation of whatever holds a similarity with a vertical bar. This was in itself a disorienting fact for an iconic interpretation of *all* signs in the simulation, given that there aren't vertical bars hanging in the skies. As a reminder, agent F (,), as explained in the previous section, generates instances of agent E ( ) on the left side of the worksheet and erases them on the right side. This, however, can be more clearly seen in an examination of the program *code* than in an execution of the simulation or an inspection of agents' depictions.

When giving the answer for the relation between agents D and F, P5 looked at the simulation being executed – not at the encoded program. His explanation for the relation shows how completely misled he was by choosing to privilege icons over symbols: "There is a relation only when this tiny smoke is here, in this area comprised between the two [instances of agent F]. What we see is that the clouds are all within this demarcated area (…). In fact, what you wanted to do was to show the relation between the cloud and the smoke, which is communicated by these little bars here."

P6 gave us very powerful evidence of how iconicity and abstract computing representations and processes generated conflicting interpretations in his mind. At some point he said "Some drop should be falling on the tree (...). Did something happen that I didn't notice? (…) I just saw the tree turning [orange] (...) I do not know if [drops] fell so fast that I didn't see them] Let me play the simulation over again." Later, when trying to tell the relation between agents B and C, P6 went back to the absence of iconic representations of what he thought was happening: "Visually, nothing happens, so let me see. (…) But he [agent B] is a counter, you know, this is what it says here [in the agent's behavior rules]. He's counting something there." So, P6 knew B was a counter, and yet he could not tell what was being counted.

An interesting hint at the meaning of iconicity was given by P4, who is a Math teacher and also the most experienced in programming with AgentSheets among all participants. He was the only one who did not invoke contextual signs (like *rain* and the like) to explain the behavior of agent A. With a remarkably abstract perspective centered on the assumed logic of the program, he explained the relation between B and C like this: "The relation between B and C is that B counts. It looks like C is checking the count; when B counts, [C] checks if B is at 2; then [C] does something; and when the counter reaches 3, [C] will do something else." P4 simply did not care about what the simulation might *mean* and *communicate*. He gave a correct and straightforward explanation based solely on how the rules were programmed. We took this piece of evidence very carefully, however, because although P4 had much less difficulty in dealing with program structures, the ultimate intent of our semiotic engineering of inter-agent relation diagrams is to help teachers and learners realize how abstract (and considerably different) programming alternatives *signify* meaningful things to program creators and program users. Therefore the strength of *iconicity* in trying to make sense of inter-agent relations was not a negative result in terms of previous and ongoing efforts made by the developers of PoliFacets. Participants were, in general, under the influence of the meanings expressed by the simulation. What most of them failed (or had a

lot of difficulty) to realize is how those meanings resulted from inter-agent relations in the program.

3.2.3 Dominant Language Function

If we think about the overall goal of PoliFacets – to support teaching and learning of CT and computer programming with an emphasis on meanings – it is critical that PoliFacets be able to communicate how *linguistic* constructs *effect* computations. In other words, how (visual) programming language commands *cause* the kind of agent behavior seen in games and simulations. From a communicative perspective, the relation between language and computation can be framed using a well-known set of *language functions* proposed by Roman Jakobson [9]. According to this author, in communication there are six objects of study: the sender; the receiver; the communication channel; the communication code; the message; and the context. Human language can be used to direct the participants' attention to any one of these objects. So, for example, when communication is full of first-person pronouns ("I", "my", "me", "mine"), the *function* of language is *expressive*. Likewise, when communication provides numerous explanations about the terms being used in a message (*e. g.* "CT stands for *Computational Thinking*" or "*Semiotics* is the study of signs"), language is being used to effect a *metalinguistic* function. The other four functions that language can effect are *phatic* (directing attention to the channel of communication), *connative* (directing attention to the receiver), *poetic* (directing attention to the message itself) and finally *referential* (directing attention to the context of communication, including the physical and social setting where it takes place, the purpose and effect of communication, etc.).

The program *code* is critically important for any piece of software, given that it causes all observable (and non-observable) computations. Therefore, one of the categories that emerged from the evidence collected in our study was the *dominant language function*. It follows from the purpose and the design of our study that we would like participants to use the *metalinguistic* function very abundantly. That is, when asked to explain relations between agents, we wanted them to direct their attention to the code of the simulation, rather than to the message (the simulation itself) or the broader context of communication (the fact that factories can pollute the air and then cause the death of trees, for example).

Three of our participants used predominantly referential function (P1, P5 and P6), with occasional comments on how the message was expressed (poetic function). The other three (P2, P3 and P4) focused on the simulation code, using the metalinguistic function of language more productively.

P1's use of the *poetic* function (focusing on the message), for example, can show the effects of failing to look at the program code. His attempt to explain the relation between B and C goes like this: "Agent C is generating smoke and B… when he sees the cloud, it stops to generate [smoke], right?." Notice how all signs in this piece of discourse (except for agents' names) are borrowed from the *message*, rather than the program code.

A similar situation came about with P5. He was so focused on the context and meaning of the simulation that he expressed the relations between B and C like this: "They are setting a limit to [others], right? It is a limit for [the sky] not to get full of that little smoke." Even after looking at the encoded rules, his (equivocal) conclusion was: "Sure, the role of B is to limit the emission of C." This confusion between what the program is *doing* (the underlying computation) and what it is *causing* (the observable effects in the simulation) can be expected to be a major barrier for program comprehension and other highly frequent programming tasks such as debugging and program modification.

The discourse of participants could, however, be more closely focused on a *programming* perspective. P2, for example, explained the relation between many agents in the whole simulation by saying: "It gets darker in two stages. In the first one it's somewhat dark, and then it gets [darker] (...). So there should be a counter to say 'when I see [this agent] next [to me], I get somewhat dark first (...) Then when I, for the i^{th} time, get myself in this condition, I get darker' (...)." Interestingly, this *metalinguistic* awareness came about by looking at the simulation, mainly – not the code.

A noteworthy *negative* effect of predominant metalinguistic function was observed in P4's discourse. This participant was so focused on the code that he barely bothered to look one more time at the simulation in order to tell the relations between agents. He looked only at the agents' behavior rules. As a consequence, his initial explanation about the simulation (before he answered specific questions about inter-agent relations) was correct and precise: "After (…) [all] trees 'catch fire' […] smoke keeps building up (...) because it only begins to accumulate after all trees have burned." However, he affirmed quite positively that "agents A and C have no relation with each other." That is, because there aren't any rules for agent A in which agent C is affected (acted upon) and vice-versa, P4 said that the two agents were not related. He missed an *indirect* relation through agent D that was captured in his spontaneous explanation of the simulation (that factories, instances of agent C, generate smoke, instances of agent D, which accumulates after all trees, instances if agent A, have burned). The *negative* effect in this case was not to be guided by the code in trying to answer the question – which was exactly what should be done – but the fact that the interpretation of a *relation* seemed to be strictly *local* (within the scope of an agent's set of rules) and structural, rather than broader and more expressive (taking a *regional* or *global* perspective).

The dominant language function in participants' discourse showed us two things. First, it indicated each participant's positioning when trying to explain inter-agent relations. Explanations where referential or even poetic functions were dominant suggested that their creators were *farther* from the level of abstraction at which *we* – as designers trying to communicate a particular facet of meanings – signified our message and on which we were trying to get users to focus. Second, and possibly more importantly, it showed that in this group there *was not* a systematic correspondence between the dominant function and the level of programming skills. Two skilled participants chose to explain relations choosing mainly the referential function, whereas two less skilled participants chose the metalinguistic one.

3.2.4 Transitions across Language Functions

Although the dominant language function in participants' discourse did not show a direct correspondence with their programming skills, this does not mean that those who favored non-metalinguistic functions failed to recognize that AgentSheets program constructs determined what was seen in the simulation. Throughout the entire discourse produced by participants during the study, we were able to detect transitions across language functions. These transitions were substantially influenced by the

study procedures themselves, considering that we asked participants to begin by explaining what they saw in the simulation prior to viewing the underlying program. Only then did we gradually delve into more abstract meanings, by asking them about inter-agent relations as signified in the simulation playback and other program representations. For lack of space (because evidence emerges from contrasting long pieces of discourse collected throughout the entire study), in this sub-section we will only comment on our observations, some of which are supported by quoted passages above.

P5 and P6, for example, had no difficulty in realizing the connection between program code and simulation. They could also express themselves using the metalinguistic function of language in discourse. However, they *preferred* to express themselves using the referential function. They could transition from one to the other effectively. Likewise, P4, who favored the use of metalinguistic expression, had no difficulty in interpreting the simulation with reference to the domain context. Again the transition between functions did not represent a problem.

However, although P2 and P3, for instance, recognized that program structures determine the behavior of agents during simulation (which represents the essence of the metalinguistic function of any language in relation to another), we have no evidence of a complete piece of their discourse with a coherent explanation expressed *only* in metalinguistic terms. The transition between language functions was hard for them; they got confused when trying to establish a connection between program structures (metalanguage) and simulation (object language). Something similar happened to P1, whose explanatory discourse during the study was predominantly expressed with reference to the domain of the simulation. The transition between relevant language functions was not observed in his case.

The importance of looking at how participants transitioned from one language function to the other becomes clearer as we go deeper in the analysis of this category of findings. All participants were teachers or instructors. A good teacher's discourse in class is typically full of language function transitions, which are skillfully used to give the students multiple perspectives on the topic being taught. Consequently, this finding – like the others – called our attention to something we had not thought of in our original design.

After we analyzed findings from the first iteration with a group of PoliFacets users, we improved the semiotic engineering of the metacommunication message conveyed through the prototyped interface and proceeded with the next iteration.

3.3 Second Iteration: Analysis and Results

In her pioneering book about the meaning of computer programming for various groups of people (including children), Turkle [23] distinguishes between hard and soft masters. Hard masters have a plan in mind and work rationally to implement it in the computer. Soft masters interact with the computer and eventually compose a program with meaningful patterns that emerge from interaction. In both cases there is a lot of complexity to be dealt with, although the form it takes and the way it evolves can be considerably different for hard and soft masters.

The first round of evaluation of the semiotic engineering of metacommunication to support the detection and understanding of program complexity in AgentSheets simulations taught us important lessons. We learned: that a visual representation of complex inter-agent relations was well received (*affect*); that to support correct interpretations of inter-agent relations our extensions to PoliFacets should signify more explicitly elements of both the simulation domain and the program code (*iconicity*, *dominant language function*, *transitions across language function*); and finally that a critical feature in our piece of metacommunication, if we want to help PoliFacets users to deal with program complexity, is to provide abundant support for them to navigate smoothly between simulation representations and program structure representations (*transitions across language functions*). The latter, in particular, means that the navigation itself is a semiotically engineered *sign* of how to relate agents to one another. Figure 9 and Figure 10 show the contrast between the first and second (revised) version of the inter-agent relations diagram, respectively.

Looking at how agent C – the center of the diagram – relates to other agents in the simulation, we can see that whereas the old (first) sign had more agents (rectangles) and less connections (lines), the new (revised) sign has more connections and fewer agents. This is because the study showed that representing indirect connections between agent C and agents A and E, in a single diagram, was more confusing than helpful. The new sign thus favors a richer (more scaffolded) representation of how agent C connects only with agents B and D. The annotations to the diagram nodes and edges have also changed considerably. The revised version attempts to evoke program elements that *cause* the relation between agents, in an attempt to support the transition between domain-centered interpretations and explanations of inter-agent relations to program-centered ones.

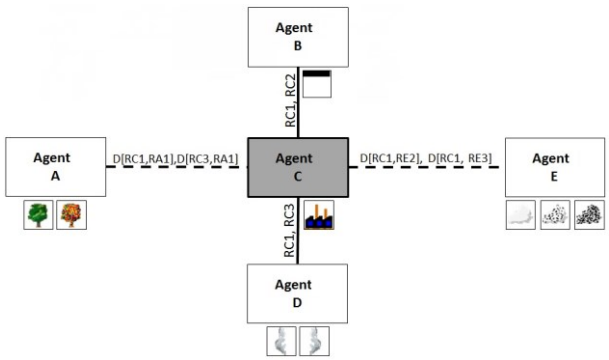

Figure 9. Old diagram of agent C

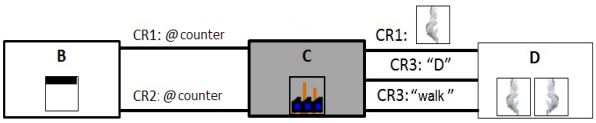

Figure 10. New diagram of agent C

The trade-off in our second round of semiotic engineering has been to shrink the scope of the *region* of influence (agents A and E are no longer represented in Figure 10) and to amplify the communication of the *meaning* of "influence". For example, annotations on edges of the revised diagram explicitly include not only the rule that *causes* the influence (*e. g.* CR1), but also the program element that originates it (@counter). As a result, the mutual influence between agents C and A can no longer be seen in the *region* of influence of agent C. It will be up to PoliFacets users to find out that C and A are related by exploring agent A's

diagram while navigating through the various facets of program meanings in this live documentation system for AgentSheets.

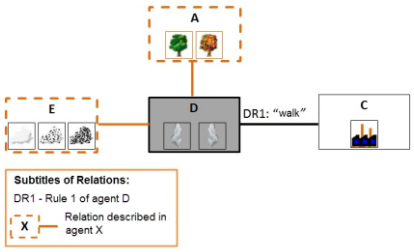

Figure 11. New diagram of agent D

Another element of reengineered signification is the expression of relations defined in the rules of a related agent. In Figure 11, dashed orange squares around A and E mean that their relations with D can be checked only by looking at A's and E's own diagrams. Compared with Figure 9 and Figure 10, this shows that the relation between C and A, or C and E, shown in Figure 9, is still unaccounted for by this strategy. This is because the origin of such relations was not to be found in their own rules (but in agent C's rules, as shown in edge annotations depicted in Figure 9).

We presented the new version of our diagrams to all the participants of our study and to volunteers who had helped us in preliminary pilot tests with the old diagram. We collected the impressions from five of them (P1, P2, P3, P4 and one of the volunteers with advanced knowledge of AgentSheets programming) by means of a questionnaire with open-ended questions. We asked them about: how they compared both versions of the diagram; if/how they would use the new diagram to explain to a learner the effects of a specifically proposed program modification; if/how they would use the diagram to teach AgentSheets programming online or offline; suggestions for improving the diagram; and additional comments.

The result of this second iteration showed that considerable improvements were made in our semiotic engineering process. For example, P2 – the only participant who commented that he did not like to reason based on visual representations – made the following comment: "This new version is more provocative to learning; it teases your chain of thought more strongly by directly expressing relations with attribute names and properties, for example. (...) [It's] more intriguing, I think." Moreover, participants that would fall in Turkle's *hard masters* category (P4 and P7) productively answered questions involving learning-teaching situations. They proposed making sensible use of the new diagram and explored good teaching strategies. However, *soft masters* who participated in this second phase (P1, P2 and P3), although they liked the diagram, showed that they are not yet prepared to use it effectively in dealing with complex situations. We thus conclude that *soft masters* need us to support them further with appropriate explanatory discourse about program complexity and how inter-agent relations play a role in it. In other words, we *can* use the proposed diagrams in metacommunication, but they need to be part of a much richer discourse, with various other signs to convey the essential message.

4. CONCLUSION

This paper presents and discusses the semiotic engineering of representations to signify and communicate complexity in computer-generated live documentation discourse about AgentSheets games and simulations. The emphasis of our research is not on the proposed representations themselves but on the rationale, the criteria and the process we have used to produce them, alternating perspectives between a *sender's* (what *we* mean to say) and a *receiver's* perspective (what *others* take us to mean and would like to mean themselves). This is our contribution to the design of communication.

Our research goals and findings are related to previously published work in different fields of study. To begin with, our choice of diagrammatic representations was motivated by research showing that, unlike sentential representations, diagrammatic ones can communicate more effectively the information about problem components, their roles and relations [7]. Moreover, our attention to the need of *regional* complexity representations was inspired by the work of Stenning and Oberlander [22]. These authors propose that there are three classes of abstract representation systems: those supporting only minimal one-to-one abstractions; those supporting limited one-to-many abstractions; and those supporting unlimited key-switched many-to-many abstractions. Their study on the cognitive roles played by the three systems in learning processes concludes that the expression and manipulation of limited abstractions is critically important for supporting a learner's reasoning.

Limiting the scope of required program abstraction is also supported by previous work in program visualization. Kapec [16], for instance, whose work discusses software visualizations using hypergraphs, concludes that even in small software projects large hypergraphs can be required to express all underlying relations. These are not at all easy to comprehend. By taking one agent at a time as the focus of communication and showing its immediate inter-agent relations, we leave it up to subsequent interaction (like navigation across various facets of meanings in PoliFacets or sequential exploration of agents' area of influence, one after the other) to communicate elements with which users can gradually construe the complexity of the program. The overall complexity is redundantly expressed by mutually-supporting signs like the simulation playback (in visual language) and the program report (in textual language), for example.

Previous work comparing three CT learning environments [24] (Scratch [21], Alice [1] and Greenfoot [8]) concludes that they try to simplify the job of building computer programs in different ways. Alice and Scratch mainly eliminate difficulties with the syntax of program encoding, whereas Greenfoot restricts programming to a tiny set of Java resources. Additionally, all three systems value simplicity and aim at removing or hiding accidental conceptual complexity. This allows users to work only with fundamental programming constructs. The same can be said about AgentSheets [17], which has been the reason for developing PoliFacets [11]. The separation between the *programming* and the *reflection on programming* spaces is how participants of the *Scalable Game Design Brasil* project resolved the tension between AgentSheets learners' difficulty in dealing with program complexity and their need to master it in order to build programs that they really wish to build. So, we keep with the findings from previous work about how the programming environment should be, but elaborate on the semiotic richness that can be explored in live documentation that accompanies such environments. This decision is in line with the opinions expressed by one of the most influential thinkers behind user-centered design, Don Norman [15]. In his 2011 book dedicated to discussing how we live with

complexity [14], Norman underlines that: "Modern technology can be complex, but complexity by itself is neither good nor bad: it is confusion that is bad. Forget the complaints against complexity; instead, complain about confusion."

Our study shows that *soft masters*, as Turkle refers to programmers who build software by experimenting with program pieces and looking at interesting emerging effects [23], are not yet helped by the communication we propose to include in PoliFacets. We seem to be speaking only to *hard masters*, whose approach is to work from general goals and principles down into the actual encoding of ideas in the form of program constructs. However, the positive results concerning the *affect* of proposed signs suggests that we can – and must – attempt to build new scaffolds and more elaborate interactive discourse about the diagrams we have proposed. Possibly, this communicative strategy will support *soft masters* more effectively and empower them to deal with the meanings and expressive opportunities lying beneath program complexity.

The next steps in our research is to implement what we have proposed. With a working prototype, we will be able to make empirical observations of how *hard masters* receive our communication in actual teaching-learning situations using AgentSheets and PoliFacets. We must also conduct additional rounds of semiotic engineering studies like the one presented in this paper, elaborating on signs of complexity so that *soft masters* can benefit from them. In one case or another, the longer-term goal of our research is to use Semiotic Engineering principles and methods to design efficient and effective communication about inter-agent relations in AgentSheets games and simulations. This communication should be primarily used in live documentation about AgentSheets, but we believe that it should also provide valuable information for improving AgentSheets' user interface itself.

5. ACKNOWLEDGMENTS

Authors would like to thank *CNPq*, *CAPES* and *FAPERJ*, the Brazilian agencies that support their research in different ways. They also thank *The AMD Foundation* for sponsoring the Scalable Game Design Brasil project, as well as all the participants who volunteered to help us in this research.

6. REFERENCES

[1] Alice - http://www.alice.org/

[2] Bennett, V., Koh, K. H. and Repenning, A. 2011. Computing learning acquisition? Visual Languages and Human-Centric Computing.

[3] de Souza, C. S. 2005. The semiotic engineering of human-computer interaction. Cambridge: Mass. The MIT Press.

[4] de Souza, C. S. 2013. Semiotics and Human-Computer Interaction. In: Soegaard, Mads and Dam, Rikke Friis (eds.) The Encyclopedia of Human-Computer Interaction, 2nd Ed. Aarhus, Denmark: The Interaction Design Foundation. Available online at http://www.interaction-design.org/encyclopedia/semiotics_and_human-computer_interaction.html.

[5] de Souza, C. S., Garcia, A. C. B., Slaviero, C., Pinto, H., and Repenning, A. 2011. Semiotic traces of computational thinking acquisition. EUD'11, Berlin, 155-170.

[6] Ferreira, J. J., de Souza, C. S., Salgado, L. C. C., Slaviero, C., Leitão, C. F. and Moreira, F. 2012. Combining cognitive, semiotic and discourse analysis to explore the power of notations in visual programming. VL/HCC'12.

[7] Glasgow, J., N. H. N. and B. Chandrasekaran. 1995. Diagrammatic Reasoning: Cognitive and Computational Perspectives. MIT Press, Cambridge, MA, USA.

[8] Greenfoot - http://www.greenfoot.org/

[9] Jakobson, R. 1060. Closing statements: Linguistics and Poetics, Style in language. T.A. Sebeok, New-York.

[10] Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. 2010. The Scratch Programming Language and Environment. Trans. Comput. Educ. 10, 4, Article 16.

[11] Mota, M .P, Faria, L.S. and de Souza, C.S. 2012. Documentation Comes to Life in Computational Thinking Acquisition with AgentSheets. In Proceedings of the 11th Brazilian Symposium on Human Factors in Computing Systems (IHC '12). Brazilian Computer Society, Porto Alegre, Brazil, 151-160.

[12] National Research Council Committe for the Workshops on Computational Thinking. 2010. Report of a Workshop on The Scope and Nature of Computational Thinking. Washington, D.C.: The National Academies Press.

[13] National Research Council Committee on Information Technology Literacy. 1999. Being Fluent with Information Technology. Washington, D.C.: National Academy Press.

[14] Norman, D. A. 2011. Living with Complexity. Cambridge, Mass.: The MIT Press.

[15] Norman, D. A. and Draper, S. W. 1986. User Centered System Design. Hillsdale, N.J. Lawrence Erlbaum.

[16] Peter Kapec. 2010. Visualizing software artifacts using hypergraphs. In Proceedings of the 26th Spring Conference on Computer Graphics (SCCG '10). ACM, New York, NY.

[17] Repenning, A. and Ioannidou, A. 2004. Agent-based end-user development, Communications of the ACM, v.47 n.9.

[18] Repenning, A., Webb, D., and Ioannidou, A. 2010. Scalable game design and the development of a checklist for getting computational thinking into public schools. In Proceedings of the 41st ACM technical symposium on Computer science education (SIGCSE '10). ACM, New York, 265-269.

[19] Scalable Game Design Brasil - http://www.sgd-br.inf.puc-rio.br

[20] Schön, D. 1983. *The reflective practitioner*. New York, NY. Basic Books.

[21] Scratch - http://scratch.mit.edu/

[22] Stenning, K. and Oberlander, I. 1995. A Cognitive Theory of Graphical and Linguistic Reasoning: Logic and Implementation. *Cognitive Science*, vol. 19 (1).

[23] Turkle, S. 2005. The Second Self. Computers and the Human Spirit. Twentieth Anniversary Edition. Cambridge, Mass.: The MIT Press.

[24] Utting, I., Cooper, S., Kölling, M., Maloney, J., and Resnick, M. 2010. Alice, Greenfoot, and Scratch - A Discussion. Trans. Comput. Educ. 10, 4, Article 17.