Cleogo: Collaborative and Multi-Metaphor Programming for Kids

Andy Cockburn*
Department of Computer Science
University of Canterbury
Christchurch, New Zealand
andy@cosc.canterbury.ac.nz
+64 3 364 2987/2569 (voice/fax)

Andrew Bryant
Department of Psychological Medicine
Christchurch School of Medicine
Christchurch, New Zealand
abryant@chmeds.ac.nz
+64 3 364-0640/372-0407 (voice/fax)

Abstract

Cleogo is a novel groupware environment that allows several users to simultaneously develop programs through any mixture of three alternative programming metaphors: a direct manipulation language for programming by demonstration; an iconic language; and a standard text-based language. Cleogo is motivated by the pedagogical values of peer-learning and of collaborative problem solving, and by our desire to investigate flexible and appropriate user-interfaces for programming, particularly for youthful users. Through its real-time groupware facilities Cleogo provides a shared conversational artifact around which students can talk, gesture and work on programming tasks. Its three concurrently active programming metaphors allow students to choose a method of program expression that best suits their task and skill level. Critical issues in the design and motivation of Cleogo are described.

Keywords: Collaborative programming, multi-paradigm programming, real-time groupware, user interface design, educational environments.

1 Introduction

There is little doubt that computer literacy will be an essential skill in the coming millennium. Computers are embedded into many household appliances such as video recorders and cookers, and the advent of "ubiquitous computing" technologies (Weiser 1993) indicates that computing devices will continue to pervade our lives at home and at work. People lacking the ability to communicate with, or program, these devices are likely to be severely disadvantaged.

We are investigating novel interfaces that help people, particularly children, to learn how to program computers. Our work on Cleogo, a groupware multiple-paradigm programming environment, is motivated by two mutually reinforcing philosophies. First, from a pedagogical perspective, we firmly believe in the value of peer-learning and collaborative problem solving. Collaborative problem solving prompts learners to "express beliefs in ways that serve to organize what they know and to identify gaps in their understanding" (Edelson, Pea & Gomez 1996). Cleogo's real-time groupware facilities in which each student has their own screen, keyboard and mouse allow it to serve as a "conversational prop" (Hill, Brinck, Patterson, Rohall & Wilner 1993) around which users can talk, gesture and problem solve.

Second, we are interested in providing "appropriate" interfaces for end-user programming. We do not believe that any single programming paradigm can suit all tasks and all users. Instead, we provide three equivalent and mutually consistent programming paradigms—programming actions expressed as input in any one of the paradigms cause corresponding output expressions in the other two paradigms. The three programming paradigms are a standard text-based language, an iconic language and a direct manipulation environment for programming by demonstration. Users are free to select

*Author for correspondence.

whichever paradigm best suits their task or their skill level. Cleogo's groupware facilities ensure that one user's programming actions are immediately communicated to all other users.

The structure of the paper is as follows. Section 2 reviews related work on groupware educational systems and on novel programming environments for children. Cleogo and its design rationale are described in section 3. Issues of evaluation and further work are discussed in section 4, and section 5 summarises the paper.

2 Background

Cleogo's motivation and design draws on three main areas of related work.

Learner centred design (Soloway & Pryor 1996) emphasises the role of technology in enabling new styles of learning, particularly group learning, that are tailored to the needs, skills and interests of the learners.

Novel programming paradigms investigate new ways of issuing sets of instructions to computers, such as programming by demonstration and visual means of program specification. Several systems focus on providing programming environments that are tailored to the needs of children.

CSCW and groupware investigate the subtleties of making computer systems usable by groups of people.

This section reviews related work on learner centred design and on novel programming environments. Related work on groupware design is incorporated into Cleogo's design rationale which is described in section 3.

2.1 Learner Centred Design

Papert's visionary work with Logo (Papert 1980) stimulated substantial interest in the role of computers in education. Based on a Piagetian pedagogical philosophy (Gruber & Voneche 1987), Papert argued that Logo could provide a foundation, or "seed", for a constructivist approach to learning in which children test personally derived hypotheses using simulations that they create within Logo's "microworlds". Educationalists' opinions on the pedagogical value of Logo differ enormously, from fervent followers to skeptics (Maddux 1985). It is clear, however, that Logo's original interface (command line and primitive graphics) was severely constrained by the lack of processing power.

In the 1990s the availability of abundant processing power effectively removes these constraints. As a result, there has been a resurgence of interest in computer support for education that is encapsulated by the banner term "learner centred design" (Soloway & Pryor 1996). Furthermore, with many politicians advocating Internet connected computers in every classroom (for instance, US President Bill Clinton's 1996 State of the Union address), there is substantial interest in the new styles of *collaborative* educational software that are enabled by the Internet.

Norman & Spohrer (1996) argue that educational systems need to satisfy three fundamental requirements: engagement, effectiveness and viability. Engagement at the interface critically effects student motivation levels. "Effectiveness" concerns the educational content of the environment. "Viability" concerns the practicality of the system: whether it is affordable, extensible, and so on. Norman and Spohrer note that although computers in the classroom are certainly engaging, their effectiveness and viability are much harder to assess.

Most of the collaborative educational systems developed to date are shared hypertextual databases, such as CSILE (Scardamalia & Bereiter 1996) and the Collaboratory Notebook (Edelson, O'Neill, Gomez & D'Amico 1995). Our interests with Cleogo's groupware properties concern real-time collaboration in which all users are simultaneously aware of each other's actions and can freely share all of the system's controls. The Color Matcher (Bricker, Tanimoto, Rothenberg, Hutama & Wong 1995) was developed to examine collaboration styles during simultaneous activity. It allows three students to collaboratively merge Red, Green and Blue colour values to match a 'target' colour, but it is highly constrained in that each user can only manipulate a single pre-assigned colour control. The domain extent of the Color Matcher is also severely constrained. TurboTurtle (Cockburn & Greenberg 1997) provides a fully collaboration-aware real-time Newtonian microworld in which all students share control of the physical properties of a "turtle", such as its velocity, mass, size and the force of a rocket on its back. Although certainly engaging, its pedagogical effectiveness has not been evaluated, and its viability is constrained by the in-built Newtonian domain.

To our knowledge, Cleogo is the only real-time collaborative programming environment for children.

2.2 Novel Programming Paradigms

Logo's turtle graphics output was intended to motivate children and to provide a concrete metaphor for the results of program execution. Logo's input language, however, was a standard text-based command language and although meaningful mnemonics replaced abstract command codes (such as "FIRST" instead of Lisp's "CAR"), typing and memorising command languages can be burdensome for youthful users. Many systems have investigated interface improvements for children's programming environments. Some of these systems are reviewed below.

AlgoArena (Kato & Ide 1995) aims to increase the motivation of student programmers by enhancing the graphical output of programs. Based on a sumo-wrestling simulation, students program the properties of sumo-wrestlers, and animated graphical output shows the execution of the specified behaviour. AlgoArena's input language is a text-based dialect of Logo.

AlgoBlock (Suzuki & Kato 1995) is another Logo-based programming environment which can be used by very young users. AlgoBlock's input language is made tangible and concrete by using electronic building blocks which are connected to form a series of instructions. The execution of the program is displayed on a computer monitor. The expression syntax in the language is very limited, but AlgoBlock's large building blocks allows several students to work together around a table during the construction of programs.

KidSim[1] (Smith, Cypher & Spohrer 1994) is an exciting and radical departure from Logo-based educational programming environments. Derived from work on visual programming and programming-by-demonstration (Cypher 1993), KidSim programmers use "graphical rewrite rules" to specify *before-and-after* conditions that change the state of the visual display. While KidSim's clock runs, any time the *before* image is generated in a portion of the display, it is replaced with the *after* image, and the new state of the display may trigger subsequent *before* conditions. Abstraction and parameterised values are supported in the environment and "property tests" can be used to trigger conditional branches. Now a commercial product from Apple, KidSim is likely to become many people's first programming environment.

ToonTalk (Kahn 1996) is a further extension of programming-by-demonstration in which the programmer issues instructions to a robot which operates within a simulation of a city. Videogame technology is used throughout the simulation to maximise engagement at the interface.

In almost all cases, systems that provide a novel method of program expression support only a *single* syntax for articulating the program.

3 Cleogo

Cleogo is substantially different from any of the systems reviewed in the previous section. First, it is fully group-aware, allowing distributed users to simultaneously develop programs and observe their execution. Second, because Cleogo supports three alternative paradigms for program expression, its users are free to choose whichever paradigm best suits their current task and their skill-levels. Users can fluidly intermix any of the three paradigms at any time.

The following two sections describe first, Cleogo's three programming environments, and second, its support for collaboration.

3.1 Multiple Programming Paradigms

Cleogo is a collaborative version of the single-user system Leogo. Leogo's design and evaluation is described in Cockburn & Bryant (1996) and Cockburn & Bryant (1998). Cleogo's multiple programming paradigms are identical to those of Leogo, but they are described here for completeness.

Cleogo's programming domain is similar to that of Logo (section 2.1. One user's view of Cleogo is shown in figure 1. The three programming environments are, from left to right, the iconic programming environment, the direct-manipulation environment for programming by demonstration and the text-based environment. Each environment is briefly described below.

Text-based programming. The text-based environment (right hand side of figure 1) supports a standard dialect of Logo, without list processing commands. Program lines are typed into the text-entry widget at the bottom of the screen and are executed when the user clicks the "Do It" button or when they press the return key. The history of previously executed commands, which may have been expressed in any of the paradigms, is shown in a scrollable list-box. Users can re-execute lines or sequences of

[1] Renamed Cocoa.

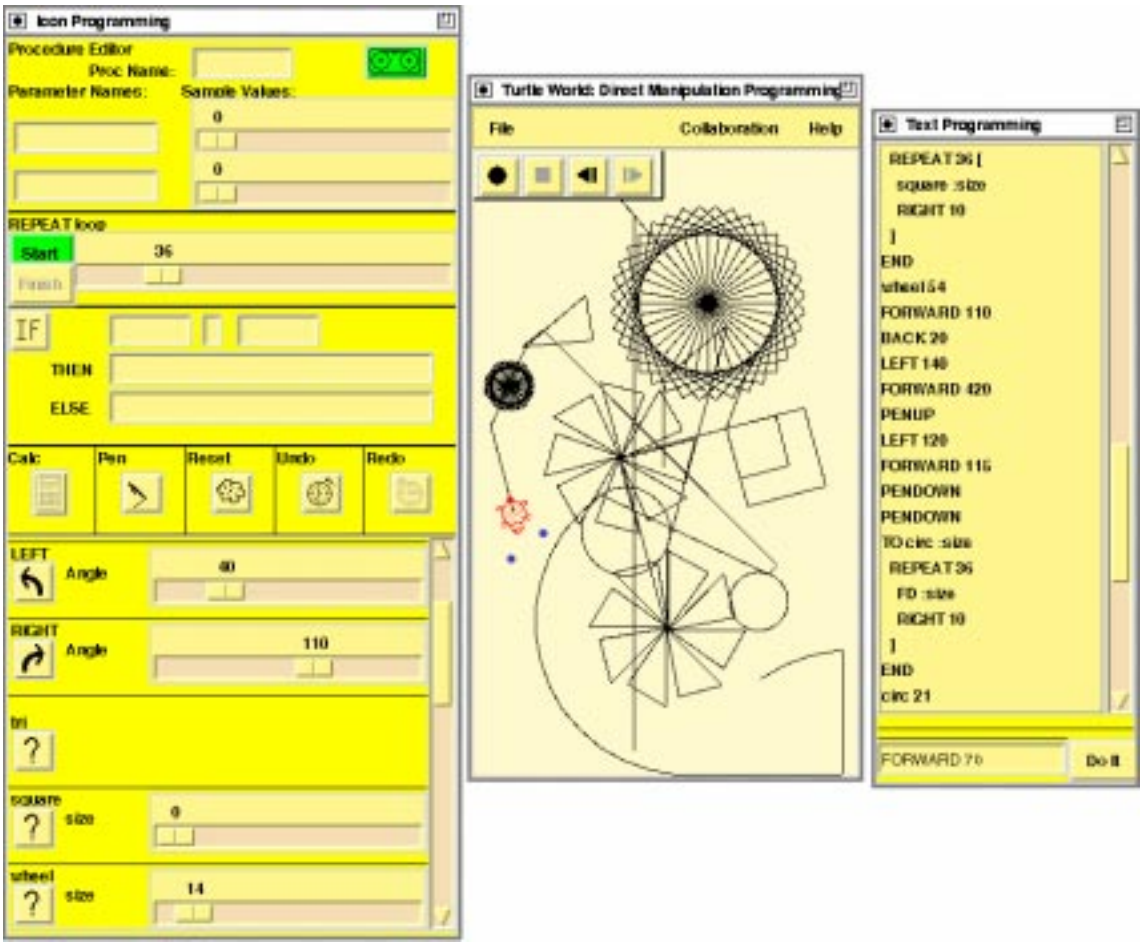

Figure 1: Cleogo's three programming paradigms: iconic, direct manipulation and textual.

lines in the history list through a point and click interface.

Iconic programming. The left-hand window of Figure 1 provides iconic representations of all of Cleogo's language elements. Iconic mechanisms for Cleogo constructs include mechanisms for defining new procedures with parameters and sample parameter values, constructs for creating repeat loops, constructs for generating conditional statements, a "Calculator" for generating expressions, mechanisms for calling turtle-motion procedures, and icons allowing the user to call user-defined procedures with parameter values. All iconic programming actions cause corresponding actions to be displayed in the text-based programming window and in the direct manipulation programming window.

Parameter values for any procedure call are set through the slider widgets alongside the procedure icons. User-defined procedures can display an arbitrary number of parameter sliders, but to save screen real-estate procedures defined within the iconic programming environment are limited to two parameters.

Cleogo's calculator (figure 2) provides an iconic-programming mechanism for entering expressions such as $: size < \ : height * 5$. Expressions are used to determine conditional statements and to give the bounds of `REPEAT` loops. The calculator allows local parameters to be included in expressions.

To define a new procedure in the iconic programming environment the user types the name of the procedure and its parameters (at the top of the window). Sample values for each of the parameters can be set, allowing the user to see the effect of the procedure while it is defined: in standard Logo, the effect of a procedure can only be viewed after it has been defined and called, making procedure declaration a highly abstract process. To start recording the procedure, the user clicks the tape-reel icon (top right hand corner), which becomes animated, and a continuous whirring sound notifies the user that

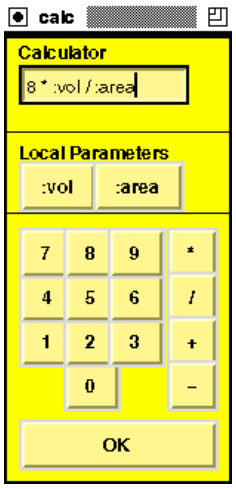

Figure 2: Cleogo's calculator for entering expressions.

all actions (which may be expressed in any of the paradigms) are being encapsulated in the procedure. To finish recording the user either clicks the tape icon (in the iconic environment), types END (in the text environment), or clicks the tape-stop button in the direct-manipulation environment. A new button is created at the bottom of the window to allow the procedure to be called, and an associated slider is created for each of the procedure's parameters.

Direct manipulation programming. In standard Logo, the middle window of figure 1 would be the output region displaying the turtle-motion described by a text-based program. In Cleogo, however, this region supports limited facilities for direct manipulation programming by demonstration. Users generate Logo commands by dragging different segments of the turtle with the mouse. Corresponding programming actions that produce identical turtle motion are simultaneously displayed in the iconic and text programming windows. Dragging the turtle's head causes it to rotate on the spot. Dragging its body causes straight line motion forwards or backwards, and clicking the turtle's tail toggles between Pen-up (no trail on motion) and Pen-down (leaving a trail).

The tape recording icons at the top of the window allow the user to record procedures and to undo and redo previous actions. Undo and redo are particularly valuable in educational environments as they encourage students to explore without concern for the consequences of erroneous actions. To start recording a set of actions the user clicks the tape-record icon, and all actions prior to stopping the tape (in any of the environments) are encapsulated into a procedure that is named through a pop-up dialogue box.

The program expression facilities of the direct-manipulation environment are weaker than the other two environments. Parameters, conditions, expressions and loops are not supported, and procedures cannot be called directly from the direct-manipulation environment. For some time we considered a variety of metaphors to increase the programming functionality of the direct-manipulation environment, but rather than risk weak metaphors and interface kludges to partially solve these limitations, we decided to avoid the issue with the philosophy that certain programming tasks are not suited to direct-manipulation. Children using the system appear to have no problem with the lack of complete equivalence between the environments (Cockburn & Bryant 1996).

3.2 Groupware Programming

Our goal in adding groupware facilities to our earlier system Leogo is to encourage children to collaboratively solve problems. We do not intend the collaborative facilities to make programming more efficient or effective. Rather, we want to provide a learning environment which supports peer learning through the articulation of problem-solving strategies while the children manipulate the shared computer-supported environment. Suzuki & Kato (1995) also investigate children's programming tools which engender collaborative problem solving, but

their system, AlgoBlock, enables collaboration through sharing of physical objects on a work-surface (section 2). Although physical blocks have some significant advantages over computer-supported input mechanisms, there are also substantial disadvantages, including the separation of paradigms for input (physical blocks) and output (computer screen), the inability to save and re-use programs, and the requirement that all collaborators be co-located.

Each Cleogo user has their own screen, keyboard and mouse, and they share simultaneous control of all of the interface mechanisms. There are no built-in limits on the number of simultaneous users, but four is a realistic maximum before degradation of system response begins to affect collaboration. Users may be co-located (in the same room), or physically distributed across the Internet, in which case a separate audio channel (speaker-phone) is required. Most of Cleogo's groupware design decisions are governed by the fundamental requirement that it serve as a conversational prop around which the children talk, gesture and work. For this reason, Cleogo is fully WYSIWIS (what you see is what I see) (Stefik, Bobrow, Foster, Lanning & Tatar 1987), ensuring that all Cleogo users see an identical system state. The only relaxation to the WYSIWIS principle is in the location of each of the programming windows: users are free to manage their own screen real-estate without affecting their colleagues' window positions.

There are no restrictions on the actions that each user can make at any time, and there are no access controls to determine which parts of the interface each user can manipulate. If two users simultaneously try to set different values for a parameter, or if one user tries to make the turtle walk forward while another makes it walk backwards, there are no software policies to resolve the conflict. Although this 'free-for-all' policy may seem to invite conflict, it is a direct metaphor for real-world activity in which there are no constraints on simultaneous manipulation of the same artifact (such as a mutually desired toy). Resolution of conflict is left to the users through social protocols. The key issue for the software is to ensure that all users are aware of each others' actions. In Cleogo, this activity awareness is provided by 'telepointers' (Tang 1991) which continually show the position of each user's cursor across all of the users' screens (two telepointers can be seen close to the turtle in figure 1). Telepointers also carry out a critical role in supporting the users' deictic references in which verbal utterances such as "this", "that" and "put it there" require a gestural activity to clarify the statement's context (Tatar, Foster & Bobrow 1991).

4 Discussion

A textual description of Cleogo's interface, such as that above, does not provide an accurate depiction of the users' sense of engagement that is engendered by its dynamic, noisy and colourful interface. For instance, consider a group of users recording a procedure to draw a square. Once the tape-icon is clicked, all users hear the whirring sound of the tape reels. One user may click the 'REPEAT' button in the iconic programming environment, and another may then set the value 4 for the loop bound. Another user may then drag the turtle forward in the direct manipulation environment, and yet another may turn it 90 degrees by typing the command into the text-based environment. Finally, one of the users will end the repeat loop, and another will click the tape icon to finish the procedure. These actions could have been expressed in any of the paradigms, and as soon as the actions are taken in one paradigm their equivalent expression is shown in the other two paradigms, both to the local user and to all other users. It is our intention that all of these actions will be embedded within extensive discussion about what to do next, and who should make each of the actions at the interface.

We have not yet begun evaluating Cleogo, but the evaluation of it's fore-runner, Leogo, was extremely encouraging (Cockburn & Bryant 1996, Cockburn & Bryant 1998), and indicated that youthful users readily adapt to the availability of multiple platforms for program expression. The primary school in which it was tested is enthusiastic to continue evaluating the system, and Cleogo will be evaluated during the 1998 academic year.

One major area that we hope to make observations on during the evaluations is the effectiveness of collaboration through the multiple programming paradigms. Given that the simultaneous users of Cleogo may prefer different paradigms for program expression, we are interested to see how well the equal opportunity interface supports the users' comprehension of programming tasks in one paradigm while the task is being expressed through another paradigm.

5 Summary

The Internet is entering the classroom, and networked computers enable many new styles of educational collaboration. The extent and nature of these new styles of collaboration are yet to be determined.

In this paper we described the motivation and design considerations that governed the development of Cleogo, an innovative system that aims to ease, motivate and assist the development of programming skills. Its groupware facilities allow learners to collaboratively solve programming problems: fostering team-work skills, and promoting peer-learning. Cleogo's multi-paradigm equal-opportunity interface for expressing programs allows collaborating users to select whichever programming method best suits their skills and tasks.

Availability

Cleogo is written in Tcl/Tk (Ousterhout 1993) and GroupKit (Roseman & Greenberg 1996). Cleogo has been tested on Sun Sparc stations and on PCs running the Linux operating system. It is available on request from the first author.

References

Bricker, L., Tanimoto, S., Rothenberg, A., Hutama, D. & Wong, T. (1995), Multiplayer activities that develop mathematical coordination, *in* 'ACM Conference on Computer Supported Cooperative Learning (CSCL '95). Bloomington, Indiana. October 17–20, 1995', Lawrence Erlbaum Associates, Inc, pp. 32–39.

Cockburn, A. & Bryant, A. (1996), Do it this way: Equal opportunity programming for kids, *in* 'OzCHI'96: The Sixth Australian Conference on Computer-Human Interaction. Hamilton, New Zealand. November 24–27.', IEEE Press, pp. 246–251.

Cockburn, A. & Bryant, A. (1998), 'Leogo: An equal opportunity user interface for programming', *Journal of Visual Languages and Computing* . In Press.

Cockburn, A. & Greenberg, S. (1997), 'The design and evolution of TurboTurtle, a collaborative microworld for exploring newtonian physics', *Interactive Learning Environments* . In Press.

Cypher, A., ed. (1993), *Watch what I do: programming by demonstration*, MIT Press.

Edelson, D., O'Neill, K., Gomez, L. & D'Amico, L. (1995), A design for effective support of inquiry and collaboration, *in* 'ACM Conference on Computer Supported Cooperative Learning (CSCL '95). Bloomington, Indiana. October 17–20', Lawrence Erlbaum Associates, Inc, pp. 107–111.

Edelson, D., Pea, R. & Gomez, L. (1996), 'The collaboratory notebook', *Communications of the ACM* **39**(4), 33–34.

Gruber, H. & Voneche, J., eds (1987), *The Essential Piaget: An Interpretive Reference and Guide*, New York: Basic Books.

Hill, R., Brinck, T., Patterson, J., Rohall, S. & Wilner, W. (1993), 'The Rendezvous language and architecture', *Communications of the ACM* **36**(1), 62–67.

Kahn, K. (1996), 'ToonTalk—an animated programming environment for children', *Journal of Visual Languages and Computing* **7**(2), 197–217.

Kato, H. & Ide, A. (1995), Using a game for social setting in a learning environment: *AlgoArena* — a tool for learning software design, *in* 'ACM Conference on Computer Supported Cooperative Learning (CSCL '95). Bloomington, Indiana. October 17–20', Lawrence Erlbaum Associates, Inc, pp. 195–199.

Maddux, C. (1985), 'The need for science versus passion in educational computing', *Computers in Schools* **2**(2/3), 9–10.

Norman, D. & Spohrer, J. (1996), 'Learner-centered education', *Communications of the ACM* **39**(4), 24–27.

Ousterhout, J. (1993), *An Introduction to Tcl and Tk*, Addison-Wesley.

Papert, S. (1980), *Mindstorms — Children, Computers, and Powerful Ideas*, Harvester Press, Brighton.

Roseman, M. & Greenberg, S. (1996), 'Building real time groupware with GroupKit, a groupware toolkit', *ACM Transactions on Computer-Human Interaction* **3**(1), 66–106.

Scardamalia, M. & Bereiter, C. (1996), 'Student communities for the advancement of knowledge', *Communications of the ACM* **39**(4), 36–37.

Smith, D., Cypher, A. & Spohrer, J. (1994), 'Kidsim: Programming agents without a programming language', *Communications of the ACM* **37**(7), 55–67.

Soloway, E. & Pryor, A. (1996), 'The next generation in human-computer interaction', *Communications of the ACM* **39**(4), 16–18.

Stefik, M., Bobrow, D., Foster, G., Lanning, S. & Tatar, D. (1987), 'Wysiwis revised: Early experiences with multiuser interfaces', *ACM Transactions on Office Information Systems* **5**(2), 147–167.

Suzuki, H. & Kato, H. (1995), Interaction-level support for collaborative learning: *AlgoBlock* — an open programming language, *in* 'ACM Conference on Computer Supported Cooperative Learning (CSCL '95). Bloomington, Indiana. October 17–20', Lawrence Erlbaum Associates, Inc, pp. 349–355.

Tang, J. (1991), 'Findings from observational studies of collaborative work', *International Journal of Man-Machine Studies* **34**, 143–160.

Tatar, D., Foster, G. & Bobrow, D. (1991), 'Design for conversation: Lessons from cognoter', *International Journal of Man-Machine Studies* **34**(2), 185–209.

Weiser, M. (1993), 'Some computer science issues in ubiquitous computing', *Communications of the ACM* **36**(7), 75–85.