Artif Life Robotics (2003) 7:124–131
DOI 10.1007/s10015-003-0246-8

ORIGINAL ARTICLE

Anthony J. Hirst · Jeffrey Johnson · Marian Petre
Blaine A. Price · Mike Richards

What is the best programming environment/language for teaching robotics using Lego Mindstorms?

Received and accepted: May 16, 2003

Abstract We are in the process of producing a range of educational materials to teach robotics to a variety of audiences using the LEGO Mindstorms Robotics Invention System. We briefly review the programming environments currently available, and consider how appropriate they are for each of our candidate audiences. There is the usual trade-off between ease of use and power. It is suggested that no single programming environment is suitable for all audiences. Instead, a *progression* of environments from microworlds, through graphical programming environments, to textual languages seems to provide the best way to develop our teaching.

Key words Programming education · Robotics education · Progressive disclosure · Microworlds

Introduction

Robotics has been shown by a number of researchers to be motivating and beneficial in teaching science and technology.[1] We believe that robots are a powerful way to motivate learning. The construction and programming of robots uses a wide range of scientific and engineering principles, which are key skills in the modern technological economy.[2] This range of skills necessitates teamwork, planning, and record keeping.

We have taught subjects related to robotics for many years, and are beginning to formulate a new robotics curriculum. In collaboration with the international RoboFesta[3] and RoboCup[4] movements, we are engaged in a program to teach robotics in schools as well as in this university. Previous experience with LEGO-based teaching materials has made us well disposed toward the LEGO Mindstorms Robotics Invention System as a possible hardware platform for robotics, engineering, and computing courses in school- and undergraduate-level teaching. An inevitable question is: *What is the best programming environment and language for teaching robotics using Lego Mindstorms?*

Given the depth and breadth of things that we intend to teach using Mindstorms, from simple programming to engineering principles and simulation, and given the range of audiences we intend to serve, from young children to mature university students, the language issue is both complex and crucial. Because the large-scale production of good-quality teaching materials is expensive, the issue has economic as well as pedagogic ramifications.

To a certain extent, the choice of environment is further complicated by the functionality required of the RCX brick itself. Different PC-based programming environments expose the actual RCX brick hardware to varying degrees of complexity. For a general course on programming, this is not usually an issue. However, if the aim is to teach low-level control, or interfacing techniques, then this factor must be considered.

The way in which the user will be required to interact with the robot is also an important consideration. For example, it is possible to download a program to a robot so that it runs autonomously. However, it is also possible to control the robot via an infrared tether, in which case a control program may be run on a host PC, issuing commands and receiving sensor data via the IR link. For science-based activities, where the robot is used as a data-logging platform, it may be desirable either for the robot to collect a batch of data before a host PC polls it and uploads

A.J. Hirst (✉)
Department of Telematics, The Open University, Walton Hall, Milton Keynes MK7 6AA, UK
Tel. +44-1908-652789; Fax +44-1908-653658
e-mail: a.j.hirst@open.ac.uk

J. Johnson
Department of Design and Innovation, The Open University, Milton Keynes, UK

M. Petre · B.A. Price · M. Richards
Department of Computing, The Open University, Milton Keynes, UK

This work was presented in part at the 7th International Symposium on Artificial Life and Robotics, Oita, Japan, January 16–18, 2002

the data, or alternatively the data logger may return data to the host PC continuously.

In this paper, we are not concerned with the division between environment and language, and we give both the terms language and environment a wide interpretation. For example, we treat a drop-and-drag environment for creating code as a "language" in the same way as a conventional textual language within an editing environment.

This paper is a synthesis of our research and analysis to date. We do not attempt to give a definitive answer to the question at this stage, and we invite readers to contribute to the discourse.

What are we teaching, to whom, and why?

There is currently a widespread appeal of robotics to adults and children of both sexes. This is evident in the success of television programmes featuring robots, and the growing number of robot competitions. We have broad educational aspirations, and would like to harness the interest and enthusiasm of all groups in this audience for wider educational purposes. The programming environment–language choice must accommodate those we are teaching, what we are trying to teach them, and our deeper educational aims.

To whom are we trying to teach?

- Young children, less than 10 years old.
- School children, 10–18 years old.
- University students, 18+ years old.
- Adults in life-long learning.
- Teachers who are learning to support students.

The breadth of this list complicates the choice of environment and language. Although we assume that some students will commence our courses as novices to robotics, the assumptions we can make about existing skills, speed of learning, and appropriate conceptual level will differ among groups. The needs of newly literate children are different from those of highly literate university students, which are different again from the needs of mature students returning to education. This suggests that there is no one perfect programming environment. Our goal must be pragmatic: to serve as many students as possible while making the best use of our resources.

What are we trying to teach and why?

Our plan is twofold:

- to teach robotics (and in particular, programming as it relates to robot control) per se;
- to use robotics as a springboard to further motivate learning in a variety of subject areas.

Robotics itself is multidisciplinary, encompassing subjects such as mechanical engineering, electronics, control, communication, vision, real-time parallel computing, and systems design. All these are relevant in our teaching.

Robotics is also a vehicle for developing key skills (e.g., teamwork, critical thinking, planning, scientific observation, and record keeping), for reinforcing skills in elementary physics, mathematics, and numeracy, and for introducing advanced concepts in simulation, artificial intelligence (AI), and cognition. There is an increasing emphasis on using robotics to support science activities, for example by providing programmable (and often mobile) data-logging platforms.

Furthermore, robots raise profound questions about our relationship with advanced technologies and their potential that allow us to address ethical and social issues surrounding technology use.

Using robots to bridge the gulf between concept and practice

Traditional methods of teaching computing tend to be abstract, and students often have difficulties in reasoning about program behavior and recognizing the relevance of their activities. The trouble is that general-purpose languages are complex, in order to afford the necessary richness to the programmer. Unfortunately for the novice, this often means *you need to know a lot to do a little.*

Many languages require the user to type in a large amount of syntactically obscure programming code to produce relatively trivial results. Either students have to learn the syntax before they can write any programs (which is frustrating), or they have to enter code that is effectively meaningless to them. An alternative approach is to use a graphical programming environment.

There are several advantages to teaching programming (within the context of robot control) using a tailored environment that provides strong visual cues and supports syntactic correctness. For example, this approach:

- is concrete, since students program things they can handle, to behave in ways they can observe in the physical world;
- is incremental;
- is creative;
- admits of many solutions;
- allows manipulation within a constrained context;
- provides immediate feedback;
- has behavior (and thus encourages anthropomorphization);
- uses a variety of skills;
- allows complete novices to create interesting outcomes (e.g., "go collect a tennis ball" rather than "print 'Hello, world'").

Our experience so far is that programming with robots helps learners to bridge the gap between concept and practice, and to derive principles for themselves from their own experience.

Robots are appealing

The appeal of robots is evident in the success of television programmes featuring robots, such as *RobotWars* and *TechnoGames* in the UK, that attract large audiences across a wide range of ages. For over 75 years, robots have been a staple of popular culture. Recent films such Steven Spielberg's *A.I.* have stimulated popular debate about the potential of robotics, and the debut of the Sony AIBO has attracted substantial media attention. Competitions involving robots are popular with participants and audiences alike. Robots are attractive to adults and children of both sexes.

How will students study what we teach?

- Supported distance learning.
- Classroom lesson.
- Extracurricula school clubs.
- Family learning/self-help groups.
- Independent exploration.

Multiple disciplines, multiple audiences, multiple learning modes; all of these mean that our choice of programming environment is sufficiently complex that there is unlikely to be a single solution. Instead, we might ask: what is the best *progression* of environments and languages for teaching robotics using Lego Mindstorms?

The system context

The RCX brick

Programming the Mindstorms processor *brick* (the RCX brick) as supplied requires a standalone computer where code is composed, edited, and compiled. The compiled code is downloaded to the brick, where it executes using a small operating system implemented as the brick's *firmware*.

Early releases of Mindstorms for the consumer market (such as the Robotics Invention System (RIS) version 1.5) were shipped with three integrated software components.

- *Firmware* that can be downloaded to the microcontroller at the heart of the brick. This firmware implements a virtual machine that will run bytecode downloaded from a host machine (a disassembly of the original firmware is also available[5]).
- *An ActiveX control* (the Spirit OCX) that can be used as component-ware on an external host machine to write programs that can be downloaded to run on the brick, as well as sending direct commands to the brick running the Lego firmware. Technical documentation released by Lego as SDK1 describes the functions provided by the Spirit OCX. Since this component is no longer available, an open-source replacement has recently appeared.[6]
- *A graphical programming language-environment* (RCX code) that uses a Lego block metaphor to construct programs out of small functional units.

The most recent release (RIS 2.0) provides a richer graphical programming environment that uses a subset of a textual programming language, the Lego PBrick script code, as well as an improved version of the firmware. Technical documentation for the Lego script code is available as part of SDK2. Programs are generated and saved from the graphical environment as script code programs, which means that as users increase in programming confidence, they can move seamlessly from the graphical environment to a text-based one. The script code is then compiled to a bespoke byte code assembly language, the Lego assembler language (LASM), which runs directly on the virtual machine implemented by the Lego firmware. It is possible to author programs directly in LASM if required. The ocx component has also been replaced by the VPBrick COM server, which can be reused within custom-built programming environments.

For the educational market, Lego produce RoboLab, which is implemented using LabView and uses the LabView programming metaphor, specifically wiring functional blocks together. The idea of progression is supported within the RoboLab environment. A Pilot phase, for beginners, uses a highly constrained environment in which users can create linear programs from a small number of functional blocks (for example, turn motor on/off, wait for a particular sensor report). An Inventor phase offers a more flexible graphical environment which supports a far richer medium, including various forms of control flow, multitasking, and variable manipulation. Different levels within the Inventor area offer the same programming metaphor, but control the amount of functionality available to the programmer. Although a textual representation of the program can be viewed, it is not possible for the student to author this code directly.

The programming environments run almost exclusively on the user's computer, although there are exceptions. For example, the leJOS On-Board Programmer[7] uses the RoboLab Pilot metaphor to allow users to program the brick directly from the buttons on the brick itself. Typically, the PC-based programming environments allow users to compose, edit, and compile code, which they then download to the brick to run on the firmware. The firmware shipped with the brick imposes limitations on the types of command that may be executed and on the number of variables available, for example, as well as tying the user into a proprietary product. However, replacement firmware can be downloaded to provide different functionality. Indeed, there is a very active community of open-source developers producing a wide variety of packages for use with the RCX brick. As a result, choosing a particular programming environment may require downloading new firmware.

Hardware and operating system choice

The Open University (OU), which specializes in distance education on a global basis, specifies the so-called Wintel machine for its students. For better or worse, this policy is based primarily and pragmatically on the fact that some

90% of our students have this hardware–software platform, and it is easier to support a single platform from a generic help-desk servicing hundreds of thousands of students world-wide.

Given this hardware default, the operating system is virtually a fait accompli. The obvious contenders are variants of Windows and Unix (Linux or Macintosh System X). The OU's commitment to being as *open* and *inclusive* as possible contradicts a one-platform approach. Therefore, a language solution that is platform or OS "agnostic," such as Java, would receive special consideration.

Using traditional programming languages

If the aim is to teach a traditional programming language using a familiar environment but in the context of robotics, this can be achieved in a variety of ways. For example, a program may be written in a particular language and then compiled into a raw form that can be run on the brick's microcontroller directly. An example of this approach is given by brickOS (formerly known as legOS), which provides a set of C libraries that implement core functionality (motor and sensor drivers, for example). These libraries can be included in a vanilla C program, which can then be cross-compiled down to the RCX brick.

Alternatively, some replacement firmware can be downloaded to the brick, and then a language can be compiled to run on this virtual machine. leJOS provides a compact implementation of a Java virtual machine (JVM), and a set of useful robotics-related Java classes. These classes can be included in a native Java program, which can compiled to run on the JVM implemented on the RCX.

A third approach makes use of the functionality provided originally by the Spirit.ocx component, and more recently by the VPBrick API. In this case, the component provides functionality that allows the user to communicate with the brick, and write and download programs to it. For example, this extra functionality can be straightforwardly embedded within a Visual Basic or Visual C program. Many of the earlier programming environments adopted this approach as a way of creating the environments themselves.

Choosing a programming environment

Our experience in teaching computing,[8,9] and the current trends in software engineering and AI, give us some general guidance in terms of desirable characteristics for programming environments/languages.

An object-based approach would support and integrate with our existing curriculum, and is now considered the basis of sound software engineering. Object-oriented programming also makes it easy to represent and present complex behaviors to novices.[8] We emphasize the importance of providing software which is suitable for novices. Any programming environment for novices must be robust: it should behave reliably and consistently, and it must not crash. Errors if they appear at all, must be meaningful.

The human–computer interaction, end-user programming, and visual programming literatures give us some guidance about relevant concepts in language selection, as described below.

Separation of domain manipulation from programming per se

Microworlds are an educational tool, originally developed by the MIT Logo Group, that allow students to explore and manipulate a domain in a controlled way.[10] The user can manipulate data or phenomena in the microworld through GUI devices such as push buttons and fill-out boxes, and see the subsequent changes reflected on the screen.

In effect, users are "programming" the microworld (albeit only to the extent of combining operations and manipulating program parameters), but the syntax and structure of the language are hidden under the interface. Hence, the implementation is hidden, and users can concentrate on the domain concepts, independently of the implementation language. Moreover, users can learn fundamental programming concepts that generalize across languages without having to learn language syntax (cf. Soloway's[11] environment, which is designed to allow high-school students to program by combining conceptual units or "plans" rather than in a programming language).

The types of concept that can be learned from such an environment include:

- that algorithms can be used to solve problems;
- that solutions can be decomposed into relatively small components;
- that most tasks can be accomplished by using sequence, iteration, choice;
- object concepts.

Microworlds have been used in the entry-level Open University course *Computing: An Object-Oriented Approach* to teach the concepts behind object-oriented (OO) technology. In an early example, the students are able to send messages to an on-screen frog, telling it to hop left, right, up, or down, setting its colour, and so on. In later lessons they create subclasses of frogs with some inherited properties and some novel properties particular to that subclass.

Simulation: separation of control logic from physical control

Simulation is a method which is commonplace in the field of autonomous mobile robots for working out and testing control strategies in isolation from the physical system. Ideally, the same program drives both the simulator and the robots. Although simulations are often different from real systems, simulators allow ideas to be tested, and they are good for detecting bugs when the vagaries of real machines in real environments are not available. This is pertinent to Mindstorms, where the performance of individual sensors and motors may vary. The effects of physical variations can

be addressed when the logic of the program and its implementation are correct. Although various RCX simulators are available, we do not feel that they are stable enough for student use at the present time.

Direct manipulation

An important characteristic of the microworlds approach is the direct manipulation of screen objects, without the imposition of linguistic devices or explicit syntax. Hutchins et al.[12] believe that with direct manipulation, novices can learn basic functionality quickly, experts can work extremely rapidly to achieve complex ends, and users can see immediately if their actions are furthering goals. Hence, direct manipulation is seen as highly desirable, characterized by the provision of rapid, incremental, reversible operations whose impact on the object of interest is immediately evident.[13]

Layering: progressive disclosure

A generalization of the microworlds approach is the "direct manipulation programming environments" (e.g., the Alternative Reality Kit[14,15]), which provide both a domain-level representation (e.g., a microworld or a control surface) and an underlying code representation. A key advantage of layering is that it is possible for the user to build their conceptual model through interaction with the microworld (i.e., in a controlled environment), and hence not get near the underlying syntax until they have a well-established model of the domain.

This sort of "layered" approach, which provides a gradual revelation of functionality, so that the user can have the simplest environment that meets their immediate needs, but expose more functionality as needed, has long been espoused.[16,17] It has been incorporated into some of the most effective programming environments for novices and young users, such as Repenning's AgentSheets,[18] a system which also allows users to move from a simple, accessible graphical environment to a textual environment when more sophistication and precision is required. AgentSheets has been used to create a rule-based programming environment – LEGOsheets – as a forerunner to the Lego RCX brick, MIT's Programmable brick. To our knowledge, no version of LEGOsheets has been produced for the RCX brick.[19]

Layering is also supported to a limited extent by the RCX SDK2, which introduced the Mindscript language. One apparent intention behind this language was to allow users to see a script language version of the programme produced using the graphical RCX language.

The BricxCC control centre (formerly RcxCC), an editor originally developed to support nqc programming, uses layering to reveal the compiled LASM byte code generated from a particular nqc program. The BricxCC also offers support for a range of programming environments (lejos, brickOS, nqc, lsc), and as it matures it looks like a strong candidate for a layered, textual programming IDE. Students using our "frogWorld" are only introduced to the implementation language (in this case Smalltalk) after fully exploring the microworld. By then, they should have a firm grounding in the concepts and be able to see how they are applied in a more conventional programming interface.[8]

Readership

Graphical environments are seen as accessible and fun, and direct manipulation potentially reduces the need for text generation, which may be problematic for newly literate children. Yet graphical environments have associated issues of readership,[20] such as:

- significant limits on the number of elements that fit on a screen;
- discriminability of graphical elements;
- the need to develop effective reading or inspection strategies;
- the difficulty of indexing into the code, of searching for and identifying desired graphical entities;
- scalability;
- the importance of an effective graphical editor.

Criteria for choice

We derived a list of criteria for language selection. Our primary concern has been an entry-level university course. However, we also wish to reuse materials for use in schools, and to support students in competitions such as RoboFesta and RoboCup. Hence, the detailed decisions refer to university level, but the higher-level decisions (e.g., OO, layering, multimode environments) are meant to generalize across our diverse audience.

Relevant criteria for selecting a language include:

- ease of understanding and use (and suitability for novices);
- rapid development;
- scalability (from simple programs to complex systems);
- general-purpose programming;
- convenient control of physical devices;
- robustness;
- support for maintenance;
- cost;
- compatibility with existing course and curriculum decisions;
- ease and cost of updating;
- longevity.

Comparison of RCX programming environments

From its first release, Lego Mindstorms proved very popular with the technically sophisticated hobbyist community. Faced with the limited power of the standard RCX programming environment described above, several people

Table 1. A comparison of first generation Mindstorms programming environments

			Summary					
Package	Language type	Requires	LEGO fw	Novice	Low cost	CS	Power (~)Y	Development environment (partially) applicable
RCX language	Custom graphical (Lego)	VPB	Y	Y	(Y)			Drag-and-drop, plug together program blocks
Robolab	Custom graphical (Labview)		Y	Y		?	Y	Drag-and-drop, wire together program blocks, supports communication between bricks
MindScript	Script language	VPB	Y		Y		Y	Text editor
LASM	Byte-code	VPB	Y		Y		Y	Text editor
Brick command	Spirit commands	OCX	Y		Y			Syntax checking text editor
Gordon's brick programmer	Spirit commands	OCX	Y		Y			Drag-and-drop editor
BotCode	Resembles Spirit commands	OCX	Y		Y			Syntax checking text editor
Pro-Bot	Resembles Spirit commands	OCX	Y		Y	~Y		Text editor
Finite-state machine	Resembles Spirit commands	OCX	Y		Y	Y		Dialogue
Visual Basic	Visual Basic (using VPB API)	VPB	Y	~Y			Y	Microsoft Visual Studio
Visual C	Visual C (using VPB API)	VPB	Y			~Y	Y	Microsoft Visual Studio
JavaScript	(using embedded ActiveX control)	OCX	Y		Y			Preferred editor
Bot-Kit	Dolphin Smalltalk	OCX	Y	~Y	Y	Y	~Y	Language-sensitive text editor
nqc	C-like		Y		Y		Y	Language-sensitive, visual editor available (Bricxcc)
Ada	Ada	nqc	Y		Y	Y	Y	Language-sensitive editor
legOS/brickOS	C	gcc			Y		Y	Preferred editor
librcx	C	gcc			Y			Preferred editor
leJOS	Java	JDK			Y	Y	Y	Preferred editor (visual interface available)
pbForth	Forth			Y	Y			Console
MIT YBL	Logo			Y	Y			Console
TinySoar	Soar	brickOS			Y			Preferred editor
legolog	Prolog	prolog, nqc			Y	Y	Y	Preferred editor

Legend	
Requires	OCX = Spirit OCX; VPB = VPBrick component; JDK = Java Developer's kit; gcc = cross-compiler
Lego fw	Does the programming language use the Lego firmware?
Novice	Is the language suitable for novice users, incorporating direct-manipulation, layered functionality, multi-mode environment (graphical and textual), robustness?
Low cost	Is the programming language cheap to buy?
CS	Is the language suitable for teaching principles of computer science?
Power	Is the language powerful enough for advanced students to create complex systems?

created their own. Many made use of the ActiveX component and the Lego firmware provided, but some approaches led to the creation of new firmware in the form of software libraries that could be linked into "traditional" programming languages. Table 1 gives a summary of the most popular community-sourced programming environments, and Table 2 gives their availability.

Conclusions

We believe that robotics is a suitable vehicle for teaching a wide range of students, no matter what their age or background. The Lego Mindstorms kit is an appropriate low-cost solution. Even though our work comparing pro-

Table 2. Sources of Mindstorms programming environments

Package	URL
RCX language/ Mindscript/LASM	http://mindstorms.lego.com/eng/community/resources/default.asp
Robolab	http://www.ceeo.tufts.edu/Robolab/
Brick command	http://www.geocities.com/Area51/Nebula/8488/lego.html
Gordon's brick programmer	http://www.umbra.demon.co.uk/gbp.html
BotCode	[http://www.workshop3d.com/rcx/botcode.htm]
Pro-Bot	http://mapageweb.umontreal.ca/cousined/lego/4-RCX/PRO-BOT/
Finite-state machine	http://www.idi.ntnu.no/~petrovic/fsm
Bot-Kit	http://www.object-arts.com/Bower/Bot-Kit/Bot-Kit.htm
nqc	http://www.baumfamily.org/nqc/
Ada	http://www.usafa.af.mil/dfcs/adamindstorms.htm
legOS/brickOS	http://brickos.sourceforge.net/
librcx	http://graphics.stanford.edu/~kekoa/rcx/tools.html#Librcx
leJOS	http://lejos.sourceforge.net
pbForth	http://www.hempeldesigngroup.com/lego/pbFORTH/index.html
MIT YBL	http://el.www.media.mit.edu/projects/ybl
TinySoar	http://tinysoar.sourceforge.net/rcx.html
legolog	http://www.cs.toronto.edu/cogrobo/Legolog/
BrainStorm (Logo)	Archived at http://robofesta.open.ac.uk/RCXprog

gramming environments/languages for Mindstorms is incomplete, the investigations to date allow us to draw provisional conclusions.

First, Mindstorms robotics provides an opportunity to offer a microworld that bridges the gap between computing abstractions and real-world activity. Well-designed microworlds and simulations are useful teaching methods, providing a low-risk, controlled environment in which to learn and develop a firm footing for further learning. Using such systems fosters confidence in using skills as well as teaching those skills.

More advanced microworlds, in which the user can see genuine program code being constructed and executed, are excellent primers to advanced computer programming with integrated development environments.

Second, although a wide range of programming environments has been created for the Mindstorms brick, *none fully meets our requirements for an introductory course*. With the exception of RoboLab, none of the graphical environments is powerful enough for students to continue to advanced work. The minimalist textual environments (text editors and command-line compilers) are not robust or supportive enough for a novice – especially a young novice – to use.

Finally, we conclude that we need to take a progressive approach, starting with a custom-built, graphical, microworld-based system, and later moving to a more sophisticated programming environment.

The microworld-based system would introduce concepts and simple programming in a language-independent, object-based methodology, would use progressive disclosure (e.g., a pseudocode view linked to the microworld view) to help students map between domain actions and code, and would serve as a bridge to a more traditional programming environment such as one of those reviewed.

References

1. Beer RD, Chiel HJ, Drushel RF (1999) Using autonomous robotics to teach science and engineering. Commun ACM 42(6):85–99
2. Wasserman E (2002) Why industry giants are playing with Lego. Fortune 144(10):101–106
3. RoboFesta and RoboFesta-UK. http://www.robofesta. net, http://www.robofesta-uk.org
4. Robo Cup. http://www.robocup.org
5. RCX hurdware internals and lirmware disassenbly. http://graphics.stanford.edu/~kekoa/rcx/
6. Opensource replacement for Spirit Active X Component. See the phantom.dll at http://21405.gel.ulaval.ca/fichiers/
7. RCX brich on-board programming environment. http://robofesta.open.ac.uk/OBP
8. Griffiths R, Holland S, Woodman M, et al. (1999) Separable UI architectures in teaching object technology. Proceedings of the 30th International Conference on Technology of Object-Oriented Languages and Systems, Tools USA '99, Santa Barbara, IEEE Computer Society, Washington DC, pp 290–299
9. Woodman M, Griffiths R, Robinson H, et al. (1998) An object-oriented approach to computing. Proceedings of the ACM Conference on Object-Oriented Programming, Systems and Languages, OOPSLA '98, Vancouver, ACM Press, New York
10. Papert S (1980) Mindstorms, children, computers and powerful ideas. Basic Books, New York
11. Soloway E (1986) Learning to program = learning to construct mechanisms and explanations. Commun ACM 29:850–858
12. Hutchins E, Hollan J, Norman D (1986) Direct manipulation interfaces. In: Norman D, Draper S (eds) User-centred system design. Lawrence Erlbaum, London, pp 87–119
13. Schneiderman B (1982) Designing the user interface, 2nd edn. Addison Wesley, Reading
14. Smith R (1987) Experiences with the Alternate Reality Kit: an example of the tension between literalism and magic. Proceedings CHI + GI 87, ACMPress, New York, pp 61–67
15. National Instruments Labview solfware. http://www.natinst.com/labview
16. Carroll JM, Carrithers C (1984) Training wheels in a user interface. Commun ACM 27(8), 800–806
17. Carroll JM (1987) Minimalist design for active users (enhancing system usability). Readings in HCI: a multi-disciplinary approach. Morgan-Kaufmann, Los Altos, pp 621–626
18. Repenning A (2000) AgentSheets: an interactive simulation environment with end-user programmable agents. Interaction 2000,

Tokyo. Available from http://www.cs.colorado.edu/~ralex/papers/PDF/Interaction2000.pdf
19. Gindling J, Ioannidou A, Loh J, et al. (1995) LEGOsheets: a rule-based programming, simulation and manipulation environment for the LEGO programmable brick. Proceedings of the Visual Languages Conference (Darmstadt). IEEE Computer Society Press, Silver Spring, pp 172–179
20. Petre M (1995) Why looking isn't always seeing: readership skills and graphical programming. Commun ACM 36(6):33–44