CrawLogo: empowering end-users to program the Web

Kevin McGee and Johan Nilsson

Department of Computer and Information Science,
Linköping University
581 83 Linköping, Sweden

kevmc@ida.liu.se
x03johni@ida.liu.se

Abstract

In order to create Web-enabled applications that programmatically use the Web as an expressive medium, the current choice is largely between conventional programming languages that are difficult to learn and use – and less expressive alternatives. In order to address this issue, we have been developing CrawLogo, a Logo-inspired programming environment in which Web-elements are programmable, body-syntonic "Crawltures" that exist within a 5-dimensional Crawlture Geometry. In this paper we briefly summarize related work, describe the CrawLogo environment, some sample applications, and the initial response of end-user programmers who have successfully used it to build Web-enabled applications. We conclude with a discussion of some insights into the larger question of empowering end-user programming of the Web, the development of a Crawlture Geometry, and future research challenges.

1 Introduction

The ability for non-programmers to build innovative Web-enabled applications is still quite limited.

There are, of course, many end-user systems that help people program Web sites, Web-enabled games, and other applications that use the Web as an infrastructure for communication and coordination. However, for the most part, when people speak about "Web-based applications", they usually means such things as those that primarily use the Web as an extended database to be searched and catalogued, as a front-end display medium, or as a delivery-mechanism for applications, updates, and the like.

In this paper the term *Web-enabled application* refers to applications that not only have a Web interface – they also use aspects of the Web as programmable data-types to retrieve, manipulate, and transmit Web-content. The aim here is not to make it easier for end-users to build typical Web-sites, networked games, or search engines (although such goals are certainly worthy). Rather, it is to enable end-users to use the content, mechanisms, protocols, and very connectivity of the Web as an expressive, *programmable* medium – much as a painter uses oils, brushes, and canvas as such a medium. Said another way, the interest is to empower end-users to program and invent Web-enabled applications that they *could* imagine and realize if the threshold of programming expertise was not set so high.

Interesting examples of such Web-enabled applications include "collaborative browsing" (*Let's Browse* [14]), collaborative collage-making (*CollageMachine* [12]; group-editing and publishing (Web logs ("blogs"), Wikis); file-sharing (*Napster*, *Friendster*); chat, IM, and various kinds of coordination software; Web-enabled games (MUDs, MMORPGs, interactive story-telling), art, and entertainment. These are often the kinds of Web-enabled applications that spark the imaginations of users – and also suggest interesting variations that they would like to create.

2 Survey

In general, few existing end-user tools for the creation of Web-enabled applications provide an expressive programming language in combination with a meaningful metaphor and ease-of-use. Available tools for programming Web-enabled applications are either too difficult for end-users or are simply not expressive enough to support the development of applications of much complexity or innovation.

Since the requirements of end-user programming environments differ widely depending on the users and their purposes, there are a number of approaches to end-user programming that are being explored, including: making the programming language more like natural spoken or written languages [15]; the creation of "scripting" or special-purpose languages (for arguments and discussion, see [21, 8]); the development of various interface metaphors [4, 19, 24]; reconceiving the notion of a "programming language" in various ways (visual programming [10], programming by physical manipulation [17] and hybrid approaches [6]); and "adding intelligence" to the programming tools ("programming by demonstration" [7, 13], agents [24], and the like). For a more extensive survey of languages for non-expert programmers, see [11].

The most common examples of Web-oriented end-user programming systems involve simplifying the development of different kinds of Web sites, from "standard" ones to Wikis and blogs. Of the systems that specifically support end-user programming of Web-enable applications, there are a few for customizing Web crawlers or newsreaders (*FeedDemon*); some for authoring and publishing to the Web (digital libraries [23]); tools to support network-based tasks such as messaging and online collaboration (ToonTalk [10], NetLogo [25]); and game construction kits [3,5].

3 Research Problem

The success of many different end-user programming systems to date is encouraging – and the dearth of such systems for creating Web-enabled applications is both an opportunity and a challenge. The research problem, then, is to develop a system that empowers end-users to build interesting Web-enabled applications similar to those they already see – such as collaborative collage-makers – or even to spontaneously invent their own.

4 Method

The approach taken to address this problem involved the development of CrawLogo, a Logo-inspired programming environment in which Web-elements are "*Crawltures*" – and in which the body-syntonic metaphor of Turtle Geometry is extended to a (Web) Crawler Geometry. In the sections below, the CrawLogo environment and the initial version of Crawlture Geometry is described – as are some sample applications and the initial response of end-user programmers who successfully used CrawLogo to build Web-enabled applications. (For a more extensive treatment, see [18].)

In order to better contextualize the contribution of CrawLogo, a brief summary of Logo and Turtle Geometry is now provided.

4.1 CrawLogo: Turtle Ancestry

Two of the major end-user programming innovations introduced in Logo [19] included the design of a special-purpose language with intuitive syntax and primitives (a variation on Lisp) – and the introduction of the Turtle as a programming interface-metaphor (or "object to think with"). Both of these strategies have become standard in systems intended to empower end-user programming. However, the notion of an "object to think with" has not been as widely elaborated upon. To be sure, there are many examples of end-user systems where agents are used as an interface metaphor. But the innovation of the Logo Turtle was more profound: it was part of a reconceptualization of geometry to make it more *syntonic*, or resonant, with the epistemology and interests of children. That is, "Turtle Geometry" was created as a bridge between the interests of children (drawing various kinds of images), the computer as a formal medium of expression, and formal geometry (as an "adult" representation that is particularly empowering for the creation and manipulation of graphics).

One of the consequences of Logo's design was that young children were able to engage in programming – indeed, it was clear that the change in representation (the "domain redesign", in the words of Papert) empowered young children to think, design, and construct in ways that were previously thought only possible by older children (the creation of systems based on formal rules and operations, etc.). Another consequence is that children often spontaneously

"know what to do" when they are introduced to the Logo Turtle; this "knowing what to do" means more than "understanding how to control the Turtle" – it also involves what Eleanor Duckworth calls "the having of wonderful ideas." That is, the domain redesign empowers children to conceive of – and execute – exciting projects that formerly might not have been meaningful or possible.

A key design insight from the work on Turtle Geometry was the importance of redesigning a domain to *leverage existing knowledge*. The Turtle is particularly syntonic or convivial in the sense that most people have fairly well-developed "body knowledge" that can be immediately leveraged in the new context of "controlling a Turtle to make pictures." These insights have been explored in the development of Logo-influenced systems for music [2], modeling decentralized systems [20], and game design [3], among many, many others; and, of course, Logo and Turtle Geometry have had an impact on many other aspects of end-user programming in general.

To conclude this review, in Turtle Geometry, the programmer controls a Turtle via a programming language. The Turtle commands (*back*, *forward*, *turn-left*, *turn-right*) correspond to the programmer's body-knowledge about movement – and this syntonic language enables a programmer to quickly create complex geometric designs, drawings, or animations. From the perspective of geometry, then, the Turtle embodies a mathematical *point* – as well as additional characteristics, such as *heading*, *speed*, *age*, and the like.

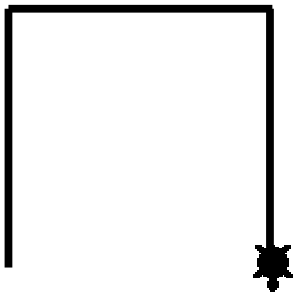

Figure 1: A "Turtle trip"

As an example, consider the image of a "Turtle trip." This illustrates the result of starting with a heading of *0* (north) – and then executing the following commands: *FORWARD 100*, *TURNRIGHT 90*, *FORWARD 100*, *TURNRIGHT 90* and finally *FORWARD 100*.

4.2 CrawLogo: The Implementation

In the section below, aspects of the CrawLogo implementation, language, and objects are described.

In general, applications are developed in the CrawLogo environment by creating and managing procedures and various objects such as Web browsers and filters, using Logo-like syntax and semantics. These applications can be local to one computer – or distributed across two or more.

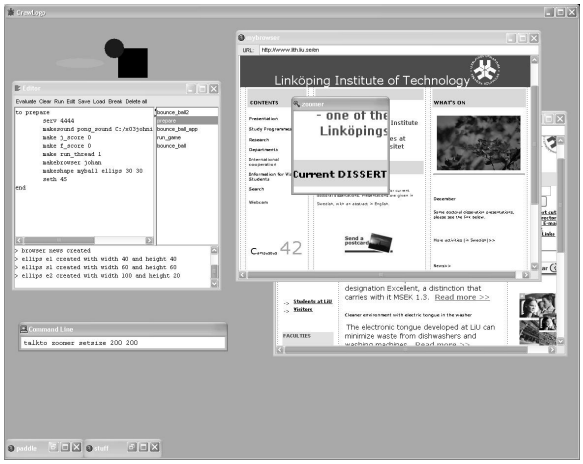

Figure 2: CrawLogo workspace

The core parts of the environment are the user interface, the CrawLogo interpreter and a number of programmable objects (see **Figure 2**). Users interact with the tool via the command line or the editor – or by direct manipulation of objects placed in the workspace. The GUI in CrawLogo consists of a workspace where the user can create and control various CrawLogo objects. A programmer has access to all the commands and primitives of CrawLogo from the command line or the editor. As in other programming environments, the command line is primarily intended for instantly invoking procedures and shorter commands or series of commands; this is useful for controlling and manipulating CrawLogo objects "on the fly", e.g. during the runtime of an application (such as a game). The editor is used to create and manage longer procedures and consists of four parts – a procedure editing frame, a list of existing procedures, a message frame (where messages from other users and system messages are displayed) and a panel with action buttons for evaluating, executing, editing, saving and loading procedures.

CrawLogo Language. Similar to other versions of Logo, CrawLogo contains a fairly standard set of primitives (mathematics, symbols, control, graphics, I/O, and the like). There are also custom primitives that include support for threading, networking, and creating and manipulating CrawLogo objects. The new CrawLogo primitives were designed to extend the CrawLogo metaphor to correspond to some of the task-specific operations that programmers might want to have their CrawLogo applications perform. For example, starting a server that can be accessed by other users does not require the programmer to know or use details about socket handling and sending packages; it is possible within the CrawLogo metaphor to quickly get a server up and running and start interacting with other users.

CrawLogo Objects. As traditional Logo has Turtle objects, CrawLogo has *Crawltures* – objects that, in addition to screen positions, headings, and the like, also have a URI ("uniform resource locator") location in Web space. Crawltures can also potentially have different sensors and effectors, allowing them to respond to – and act upon – other Crawltures and I/O data (display, audio, and the like). Thus, for example, a Crawlture can be programmed to "click on" certain kinds of hyperlinks – and "have them spawn as new Crawltures." There are a number of different categories of Crawltures, including Web browsers (which can retrieve and display Web pages); shapes (most similar to the traditional Logo Turtle, these are simple graphical objects – quadrangles and ellipses of different sizes and colors – that can be instructed to move on the screen and interact with other objects); and filters (which can graphically modify other Crawlture*s* that they come across – blurring or embossing them).

Crawlture Geometry. The "geometry" of Web space presents a number of representational challenges for CrawLogo. In particular, it is not clear to what extent it is possible to have a fixed, absolute geometric reference framework for a "space" made up of "coordinates" that update dynamically relative to each other. This is a longer-term research problem, and there are a number of research projects to visualize and represent the geometry of Web space (see [9] for a survey of this work).

This initial version of Crawlture Geometry involves a 5-dimensional, physical space. A Crawlture exists in a position comprised of the X- and Y-coordinates of the screen – as well as "something like" a three-dimensional URI-space. The URI space is three-dimensional in the following sense: the URI links explicitly referenced on the page of the Crawlture's current URI location are mapped as a plane (X- and Y-coordinates in URI-space) of discrete, Crawlture-relative nodes – and the "absolute" directory/file-structure space of the current URI location is represented as a Z-axis in URI-space (with nodes potentially "above" and "below" the Crawlture). One way to visualize this is to think of the computer screen as "looking down" on the "top surface" of a 3-dimensional URI-space.

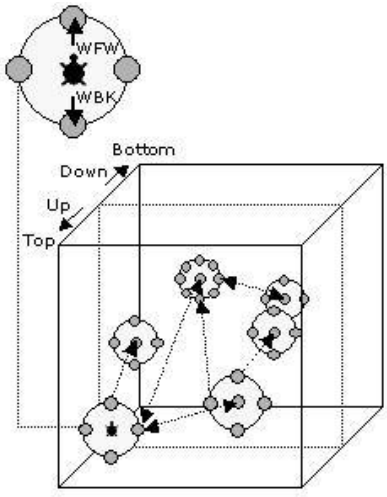

Figure 3: Conceptual model of a URI-space

Thus, body-relative Crawlture movement consists of the traditional *forward* and *back* on the screen-surface – as well as *web-forward* and *web-back*, which move the Crawlture along the Web XY-plane. Additionally, the Crawlture can move *up* and *down*, where *up* is "towards" the programmer (from behind the screen) along the Z-axis, and *down* is "away from" the programmer (in the direction of the screen) along the Z-axis.

The Web-coordinates are only "absolute" to the extent that such coordinates actually exist in the structure of Web URIs at the moment of code-execution – and CrawLogo makes no assumptions about the existence or the endurance of such coordinates. Crawlture movement along any one of the five Crawlture Geometry dimensions is *disjoint* from movement in any other dimension.

When a Crawlture moves to a new location, it automatically extracts all hyperlinks contained in the URI page and maps them around itself as "exits" to

other locations (see Figure 3). The "width" of each exit is calculated by 360/*n*, where *n* is the number of hyperlinks of the current URI. When there is at least one exit, a Crawlture will always be facing a specific one, and all exits have the same width. In some sense, a URI coordinate can be thought of as a house, with numerous doors (hyperlinks) to rooms (other URIs) on different floors (directories and subdirectories). Telling the Crawlture to move *web-forward* will take it to the URI it is currently facing, and telling it to turn *web-right* or *web-left* will direct it towards the URI position it has generated. Note that the number of degrees of a URI node is determined by the number of links the URI has; arguments to *web-right* and *web-left* are evaluated relative to the number of degrees in the node.

For example, in Turtle Geometry, the calculation for a Turtle that has a heading of 0 and turns right 90 is: (/ (modulo 90 360) 360) = 1/4 turn. In Crawlture Geometry, the calculation for a Crawlture that has a heading of 0 and turns right 90 (on a node of 20 degrees) is: (/ (modulo 90 20) 20) = 1/2 turn.

Moving *web-forward* (or *web-back*) takes a Crawlture to a new node. In the case where the Crawlture is on a node with no connecting nodes, the only way for it to move is via *web-up* or *web-down*.

4.3 CrawLogo: Programming

Programming in CrawLogo is very similar to programming in other Logo environments. The actual syntax is similar and there are commands for making and controlling Crawltures; there is support for creating, applying, and saving complex procedures and sub-procedures; and there are mechanisms for managing different aspects of URI-space, network activity, and connectivity. Additionally, there is an initial implementation of a "recording" feature that allows programmers to do things such as have a Crawlture "run all night, looking for interesting things" – and then "play back" some portion of its activity-history.

4.4 CrawLogo: Sample Applications

A number of demonstration CrawLogo applications have been developed; four are briefly described here.

CrawLogo Pong. This is a version of the classic Atari game in which players compete across the network, and in which Crawltures are the "ball" and "paddles" – and in which different state-conditions for both the balls and paddles have unexpected consequences for the players (such as changing the speed, size, or direction of the ball – or modifying the player's ability to control the paddles).

Collaborative Browsing. This is an application in which users can browse the Web together, show each other interesting Web pages and chat about what they see.

Image Slideshow. This is an application in which someone can programmatically specify a Web-generated slide-show.

"Guess Who?" This a Web-enabled multiplayer guessing game in which Crawltures are programmed to find other Crawltures with, say, pictures of rock-stars. The pictures are then blurred (or otherwise disguised) and players send either guesses or Crawltures to "de-blur" the image a bit.

4.5 CrawLogo: Actual Use

To date, most of the CrawLogo research effort has gone into the initial design and development of the environment and the Crawlture Geometry. However, there have been some informal meetings with end-users who have tried the system. Participants were told about the Logo Turtle, the idea behind CrawLogo, and some examples of applications one could make with it. They were then given access to CrawLogo and a short manual of CrawLogo commands. The principle developer was also present as they used the system, answering any questions and providing "system feedback" in cases where it wasn't available. For the most part, the programming experience of the participants was "a single introductory course a few years ago"; none had ever developed a Web-enabled (or networked based) application.

In the discussions, participants were generally enthusiastic about the idea of being able to quickly create their own applications that might include the ability to communicate and collaborate with others over the network – as well as create applications that made use of the Web itself as an expressive medium. In the case of one pair of participants, after about five minutes they "made contact" with each other over the network and began spontaneously sending each other Crawltures (Web-browsers, geometric shapes, and actual procedures). They also quickly discovered that they themselves continued to have control over Crawltures they sent to other participants, and began to make them do "interesting things" on each others' screens. This quickly evolved into a collaborative

browsing and playing situation. A couple of the participants also spontaneously created their own versions of some of the applications described above.

Participants were also presented with a specific challenge: create a Web-crawler that would generate an interesting slide-show by finding a URI that matched some criteria, displaying it, waiting a certain amount of time, and then finding another URI (based on programmer-specified criteria). They all succeeded quite easily in creating such an application.

The immediate impression of participants was that not only did the CrawLogo environment empower them to make interesting applications, but the actual *process* of making the applications was fun.

One difficulty that became apparent was the lack of a "visible heading" for certain kinds of Crawltures. A WebCrawler browser window, for example, can have a heading just like a traditional Turtle – and although there is some indication of heading if the image is upside-down, more subtle variations are not currently represented visibly. This is something currently being developed, but the absence made it difficult for participants to create some of the effects they wanted. Not surprisingly, the participants also found it difficult to conceptualize certain aspects of Crawlture Geometry. We discuss these issues in more detail below.

5 Discussion

Empowering users to create and control Crawltures that move along complex and unpredictable paths calls for a meaningful and intuitive representation of such movement – and the geometry of the space within which such movement takes place.

This work is still in its early stages and there are already a number of obvious research challenges related to the CrawLogo language, the design of Crawlture Geometry, and the visualizations of different phenomena.

Syntonicity and Design of Primitives. Much of the learnability of Logo's Turtle Geometry lies in its syntonicity – the possibility for a user to identify with the Turtle and mentally (or physically) "play Turtle." Currently, the CrawLogo primitives fall into three broad categories, depending on the degree to which they can be said to be consistent with the Crawlture metaphor: consistent with "playing Crawlture" (commands such as *forward*, *up*, and the like); consistent with "talking to the Crawlture" (commands such as *setcolor*, *setpower* and *setURL*); and those that are "outside the metaphor" (*startserver* and the like). This raises an ongoing design issue about whether (and how) to try and "force" certain programming activities into a consistent metaphor.

In order to reduce complexity for the programmer, the current implementation of CrawLogo does not support a screen Z-dimension – nor various controls for orientation ("pitch", "yaw", and the like). In their efforts to create versions of Logo that support 3-dimensional movement and graphics, others [1] have noted some of the difficulties. The very dynamics and structure of "Web space" raises additional issues for an appropriate geometry; for example, there is no guarantee that executing the same series of CrawLogo movement commands from the same starting point at different times will result in either the same *path* or the same *terminal-URI*.

Turtle Goes Crawling. Certain aspects of representing the Web geometrically raise problems that are not present in traditional 2-dimensional Turtle Geometry. To name only a few examples, consider that in Turtle Geometry, headings of 0 and 360 are equivalent; it is not clear to what extent it is meaningful to think of Web space as being "closed" in the same sense. Even more problematic, it is not clear where "the screen" is located along the Web z-axis: is it the "origin" – and if so, is such an origin best conceived in terms of a polar coordinate system? Similarly, the notion of "reversible operations" within Crawlture Geometry is not straight-forward: whereas moving *up* is unambiguous (e.g. moving up from any particular URI will always lead to a parent URI) moving *down* from the same parent URI can potentially lead to a different URI (not to mention the inherent ambiguity of *down* in the context of multiple choices).

Crawlture Visualization. One issue that is clearly problematic is how to visualize certain aspects of a Crawlture's state; for example, *heading* and *position* in CrawLogo space. Some issues, such as the visual representation of a rectangular browser's orientation on the screen, will not be difficult to solve. Others, such as providing orientation or movement cues along other dimensions, will be more challenging. In particular, the current implementation does not try to visualize a unified, 5-dimensioanl space; much of the spatial movement and orientation of Crawltures is left to the imagination of the programmer. As a related visualization problem, it is not obvious what the equivalent of "pen down" should be for CrawLogo. Programmers in traditional Logo benefit from seeing

the history of the pictures they try to make; it is not clear how to provide similar, concrete feedback about Crawlture histories. (It may be worth exploring solutions similar to "salient still" visualizations [22]).

In addition to the control and visualization of certain standard state-information, a future goal is to increase the expressive potential in other ways. For example, in traditional versions of Logo the dimensionality of the Turtle is zero. In CrawLogo, the programmer should be able to programmatically control dimensionality – in all the dimensions of Crawlture Geometry. As an example of application, consider the potential to create Crawltures that can be more like agents in different ways: they could expand/shrink along different dimensions ("inhaling" and "exhaling"), "feed on" other Crawltures, and "hide" in different dimensions. In the current version of CrawLogo it is possible for Crawltures to respond to each other physically (Pong balls bouncing off of paddles) to a limited extent. There are many interesting possible extensions – but they clearly raise many of the challenges involved with designing an appropriately syntonic geometry.

It is currently possible to create a Crawlture that moves from URI to URI and displays the page contents. However, the current possibilities for programmers to specify this movement is still quite limited. Similarly, the current implementation provides only a limited ability to specify data-types and properties of Web content. It would be useful if programmer- and user-interaction could be more intelligent – Crawltures could have various ways of representing meta-data, histories, and the like.

The major drawback of the current implementation has to do with the 5-dimensional geometry – and certain implementation choices based upon it. In a way, the single largest future research problem is how to develop a Crawlture Geometry of *six* dimensions – one that is syntonic and visually empowering. One possible solution is to represent the two, 3-dimensional "halves" of Crawlture Geometry as separate screens: with visual feedback from "screen-space" displayed in one, and visual feedback from "URI-space" displayed in the other. It would be interesting to see whether programmers were able to "create a mental synthesis" of the entire 6-dimensional space. An alternative is to explore more esoteric techniques for visualizing spaces with large numbers of dimensions.

Another major research topic involves the further refinement of how to compute the number of degrees in a URI node – and how to best correlate such a solution with the programmer's intuitions and standard assumptions. It is troubling to have the classic Turtle "right 90" mean different things at different times and places in URI-space; it is not clear that there is an easy alternative.

6 Conclusion

A long-term goal of this research is to empower computational creativity of different kinds; in particular the ability of end-users to program the Web as an expressive medium. In addition to such success criteria as ease-of-use, the evaluation metrics include such things as the pleasure with which people use the tools, the pride they take in their creations, and the degree to which they are empowered in "the having (and realizing) of their own wonderful ideas." The current version of CrawLogo still requires much work to fulfill this goal, but the results to date are encouraging.

7 References

1. Abelsson, H., diSessa, A. (1980). Turtle Geometry: the computer as a medium for exploring Mathematics. Cambridge: MIT Press.
2. Bamberger, J. (1979). Logo Music Project: Experiments in musical perception and design. A.I. Memo 523, M.I.T. Artificial Intelligence Lab., Cambridge, Mass.
3. Begel, A.B. (1997). Bongo: A Kids' Programming Environment for Creating Video Games on the Web. MS Thesis. MIT.
4. Blackwell, A.F. (2002). What is programming? In *Proceedings of PPIG 2002*, pp. 204-218.
5. Bruckman, A. (1997). MOOSE Crossing: Construction, Community, and Learning in a Networked Virtual World for Kids. PhD Thesis, MIT.
6. Cockburn A. and Bryant A. (1997). Leogo: An equal opportunity user interface for programming. *Journal of Visual Languages and Computing*, 8: 601–619. Academic Press, New York, NY.
7. Cypher, A. Ed. (1993). Watch What I Do – Programming by Demonstration. The MIT Press, Cambridge, MA 02142, 1993.
8. Eisenberg, M. (1995). Programmable Applications: Interpreter meets Interface. In SIGCHI Bulletin, 27(2), pp.
9. Geisler, G. (1998). Making Information More Accessible: A Survey of Information Visualization

Applications and Techniques. URL: http://www.ils.unc.edu/~geisg/info/infovis/paper.html (2003-11-21).
10. Kahn, K. (1996). ToonTalk: An Animated Programming Environment for Children. Journal of Visual Languages and Computing, 7(2):197.
11. Kelleher, C. and Pausch, R. (2003). Lowering the Barriers to Programming: a survey of programming environments and languages for novice programmers. Computer Science Department, Carnegie Mellon University. Pittsburgh, PA: CMU-CS-03-137.
12. Kerne, A. (2001). Collage Machine: Interest-Driven Browsing through Streaming Collage. In Proceedings of Cast01: Living in Mixed Realities. (Bonn Germany, 2001), pp. 241-244.
13. Lau, T. and Weld, D.S. 1999. Programming by Demonstration: An Inductive Learning Formulation. *In Proceedings of the International Conference on Intelligent User Interfaces*, 145–152.
14. Lieberman, H., van Dyke, N. and Vivacqua, A. (1999). Let's Browse: A Collaborative Web Browsing Agent. In Proc. Intl. Conf. on Intelligent User Interfaces, January 1999.
15. Miller, L. A. (1981). Natural Language Programming: Styles, Strategies, and Contrasts. IBM Systems Journal, 20(2), 184-215.
16. Miller, R. C. (2003). End-user Programming for Web Users. End User Development Workshop, Conference on Human Factors in Computer Systems (CHI), April, 2003.
17. Montemayor, J., Druin, A., Farber, A., Simms, S., Churaman, W., and D'Armour, A. (2001). Physical Programming: Designing Tools for Children to Create Physical Interactive Environments. CHI 2002, ACM Conference on Human Factors in Computing Systems, CHI Letters, 4(1), 299-306.
18. Nilsson, J. (2004). CrawLogo: An Experiment in End-User Programming for Web-Enabled Applications. Unpublished MSc Thesis. Linkoping University, Linkoping, Sweden.
19. Papert, S. (1980). Mindstorms: Children, Computers, and Powerful Ideas. (Second Edition, 1993). New York: Basic Books.
20. Resnick, M. 1996. StarLogo: An Environment for Decentralized Modeling and Decentralized Thinking. In Proceedings of the 1996 Conference Companion on Human Factors in Computing Systems.
21. Smith, D.C., Cypher, A. Schmucker, K. (1996). Making Programming Easier for Children. Interactions, v.3 n.5, p.58-67, Sept/Oct 1996.
22. Teodosio, L, Bender, W. (1993). Salient video stills: content and context preserved, Proceedings of the first ACM international conference on Multimedia, p.39-46, August 02-06, Anaheim, California, United States.
23. Theng, Y. L., Mohd-Nasir, N., Buchanan, G., Fields, B., Thimbleby, H. & Cassidy, N. (2001). Dynamic Digital Libraries for Children. *Joint Conference on Digital Libraries* pp 406–415. ACM Press.
24. Travers, M. (1996). Programming with Agents: New Metaphors for Thinking about Computation. PhD Thesis, Massachusetts Institute of Technology.
25. Wilensky, U. & Stroup, W. (2000). Networked Gridlock: Students Enacting Complex Dynamic Phenomena with the HubNet Architecture. The Fourth Annual International Conference of the Learning Sciences. Ann Arbor, MI. June 14 - 17, 2000.