in Proceeding of Visual Languages 1996, Boulder, Colorado: IEEE Computer Society.

Tactile Programming: A Unified Manipulation Paradigm Supporting Program Comprehension, Composition and Sharing

Alexander Repenning and James Ambach

Department of Computer Science
Center for LifeLong Learning and Design
University of Colorado, Boulder CO 80309-0430
(303) 492-1349, ralex@cs.colorado.edu
(303) 492-1503, ambach@cs.colorado.edu
Fax: (303) 492-2844
http://www.cs.colorado.edu/~ralex/
http://www.cs.colorado.edu/~ambach/

Abstract

Although visual programming techniques have been used to lower the threshold of programming for end users, they are not sufficient for creating end user programming environments that are both easy to use and powerful. To achieve this, an environment must support the definition of programs that are not just static representations of behavior, but are instead dynamic collections of program objects which can be applied in a number of contexts rather than just a program editor. We describe an approach to end user programming called *tactile programming* which extends visual techniques with a unified program manipulation paradigm that makes programs easy to comprehend, compose and, most importantly, share over the World Wide Web. Tactile programming's inherent ability to support the social context in which programming takes place along with its ability to ease program comprehension and composition is what differentiates this approach from others. In the context of the Agentsheets programming substrate, we have created an instance of a tactile programming environment called Visual AgenTalk which is used to create interactive simulations.

Keywords

graphical rewrite rules, end user programming languages, scripting, direct manipulation, agents, object-oriented programming, visual programming, visual object-oriented programming languages

VL Topics

HCI issues for VLs, Visual Programming languages for programming on the Internet

Tactile Programming: A Unified Manipulation Paradigm Supporting Program Comprehension, Composition and Sharing

Alexander Repenning and James Ambach

Department of Computer Science
Center for LifeLong Learning and Design
University of Colorado, Boulder CO 80309-0430
(303) 492-1349, ralex@cs.colorado.edu
(303) 492-1503, ambach@cs.colorado.edu
Fax: (303) 492-2844
http://www.cs.colorado.edu/~ralex/
http://www.cs.colorado.edu/~ambach/

Abstract

Although visual programming techniques have been used to lower the threshold of programming for end users, they are not sufficient for creating end user programming environments that are both easy to use and powerful. To achieve this, an environment must support the definition of programs that are not just static representations of behavior, but are instead dynamic collections of program objects which can be applied in a number of contexts rather than just a program editor. We describe an approach to end user programming called *tactile programming* which extends visual techniques with a unified program manipulation paradigm that makes programs easy to comprehend, compose and, most importantly, share over the World Wide Web. Tactile programming's inherent ability to support the social context in which programming takes place along with its ability to ease program comprehension and composition is what differentiates this approach from others. In the context of the Agentsheets programming substrate, we have created an instance of a tactile programming environment called Visual AgenTalk which is used to create interactive simulations.

1. Introduction

The goal of this work is to enable a wide range of end users, ranging from children to professionals, to create their own SimCity®-like interactive simulations. Central to this goal is the problem of creating a programming environment that strikes a careful balance between ease-of-use and expressiveness [1]. General purpose programming languages such as C and Pascal, on the one hand, are highly expressive; they enable professional programmers to create complex computational artifacts, but they are not accessible to the majority of end users. Visual and end user programming approaches, on the other hand, are often meant for the casual programmer with little or no programming background. Especially in the domain of interactive simulations, visual programming environments based on graphical rewrite rules, such as Agentsheets [2] BitPict [3], ChemTrains [4], KidSim [5] and Vampire [6] are effective end user programming approaches allowing users to define behavior by editing before and after pictures. These rule-based, visual programming environments provide ease-of-use but are limited in their expressiveness. The question raised is whether it is possible to create programming environments which are both easy-to-use and expressive.

To address this tradeoff, we believe that it is not enough to focus solely on mechanisms that attempt to make programming more intuitive. Although this will help, achieving a qualitative increase in expressiveness, while not sacrificing ease-of-use, requires support for the social context in which programming can take place. Currently, most end user programming environments assume a solitary programming process in which one person programs one computer. Nardi [7] and MacLean et al. [8] indicate that there are many advantages to supporting the development of programming communities. MacLean specifically points out that a programming community encourages its members to help each other by sharing insights and expertise. While the existence of this social dimension is recognized, it is not often supported in current end user programming environments.

Programming environments that are easy to use and expressive must contain mechanisms that enhance the user's ability to *comprehend* programs and program fragments, to *compose* complex programs from simpler primitives, and most importantly, to easily *share* programs and program fragments within a community of users. Although other programming environments have addressed these concerns individually, we believe that it is essential to find a *unified program manipulation paradigm* that supports the following issues simultaneously:

- ***Comprehension***: An end user programming environment should include mechanisms that increase program comprehension. Users should be able to easily determine the effects of programs and program components within their applications. This becomes increasingly important in environments that support collaboration and the sharing of programs. When programs are shared between users it is essential that these programs are not only executable but can also be easily understood by other users who may need to modify them.
- ***Composition***: It should be easy to construct complex behaviors by composing them from simpler ones. Programming environments should not only allow this kind of composition but should actively support the composition of meaningful programs by helping users to avoid the composition of incompatible components.
- ***Sharing***: Environments should allow end users to share, with very little effort, programs and program fragments. The programming environment must enable users to recognize sharable program fragment boundaries to ensure that the shared components will work in other users' applications.

This paper develops the notion of *tactile programming* as a unified program manipulation paradigm that supports these issues from a user interface perspective. We introduce an instance of a tactile programming environment called Visual AgenTalk that was designed based on our experience with graphical rewrite rules and rule-based visual programming languages [9].

2. Visual AgenTalk: A Tactile Programming Environment

Visual programs use non-textual visualization to represent programs and programming constructs. These representations, unless they provide some kind of feedback during program execution, are static representations of the program. Viewing these programs in a dynamic medium, like a computer, does little to make them more comprehensible than viewing the same program in a static medium like paper. Tactile programming extends the framework of visual programming by adding *perception by manipulation.*

In tactile programming, visual program representations are enhanced with a sense of tactility by elevating program fragments to the status of first class interface objects which can be dragged and dropped [10] into different contexts besides just program editors. Applying program objects by dropping them into different contexts triggers dynamic audio-visual feedback that creates perception by manipulation. The ability to apply program objects in different contexts using the same interface manipulation paradigm is the primary concern of tactile programming and provides support for program comprehension, composition and sharing.

Visual AgenTalk is a tactile programming environment that has been added to the Agentsheets system [11] in order to enable end users to create complex, interactive, SimCity-like simulations. Tactile programming in Visual AgenTalk is centered around the notion of objects that can move fluently between three different worlds: the *application world*, the *programming world* and the *collaboration world*.

2.1. The Application World

The application world (Figure 4) contains the objects of direct interest to the end user. In Agentsheets, application worlds consist of communicating *agents* organized in a grid. In Figure 1 agents represent real-world objects such as cars, , roads, , and traffic signals, . In comparison, application world objects in HyperCard® include things like buttons, Time, and fields, It is now 2:42 PM. Application world objects are manipulated or viewed by the end user of the application.

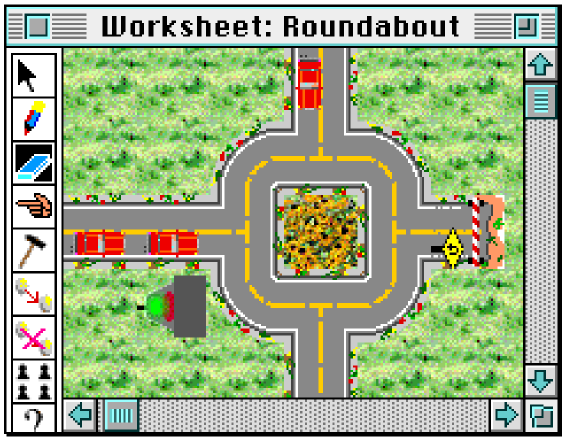


Figure 1: An Agentsheets worksheet allowing users to simulate interactions between cars and traffic lights.

2.2. The Programming World.

The programming world (Figure 4) contains objects that describe the behavior of application world objects. In HyperCard, programming world objects consist of HyperTalk® functions and handlers describing the behavior of application world objects such as buttons. In comparison, Visual AgenTalk features three different types of programming world objects: *commands*, *rules* and *triggers*.

- ***Commands*** are small interactive forms representing programming primitives that can be manipulated by end users. Commands have interfaces that consist of familiar direct manipulation widgets. A command (Figure 2) consists of a name and an arbitrary number of parameters. Commands have a visual syntax allowing the combination of textual and pictorial components. End users set the values of the typed parameters via *type interactors* such as number fields, text fields, check boxes, and textual or iconic pop up menus. Type interactors can limit user input to valid choices. For instance, a sound type interactor is a pop up menu offering only the names of sounds that are available in the system.

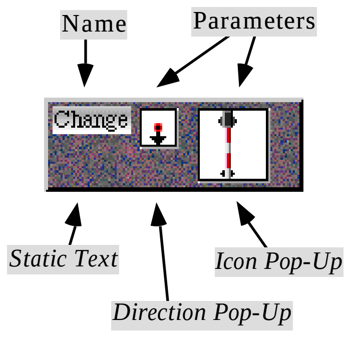


Figure 2: A Visual AgenTalk command to change the look of an agent.

Parameters, such as the crossing gate in Figure 2, can be references to application world objects. The ability to have application world objects appear in the programming world as they do in the application world significantly helps end users to map between the two worlds [12].

- ***Rules*** contain condition and action commands. For instance, the rule below (Figure 3) is used to program a car. The IF part, on the left, and THEN part, on the right, of each rule are represented by flexibly sized containers into which commands can be dragged and dropped. The IF part is an implicit conjunction of all the conditions and the THEN part is an implicit action sequence.

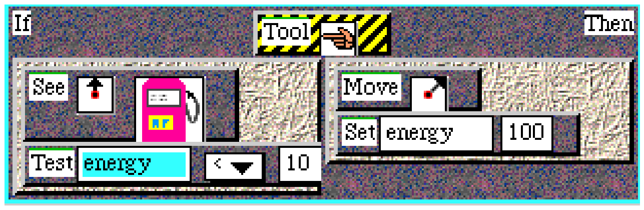


Figure 3: "Check for a Gas Station" Rule. If there is a gas station above the car, and if the energy level of the car is less than 10, then the car will pull up next to the gas station and set it's energy level to 100.

- ***Triggers***, such as the Tool trigger, , used in the rule above (Figure 3), allow the end user to define the type of event that will cause the rule to be tested. In this case the "check for a gas station" rule will be tested if a user applies the hand tool to a car agent. Other trigger types include timers, mouse clicks, keyboard events and new messages defined by end users.

2.3. The Collaboration World

The collaboration world (Figure 4) contains application world objects and programming world objects that are shared within the programming community. In Visual AgenTalk, shared items can include simulations, agents, rules, commands and triggers .

The ability to easily share program fragments is something that end user programming environments tend to ignore. Instead, the focus is on lowering an individual's threshold to programming by providing more intuitive interfaces. However, there is a significant amount of evidence that shows that individual learning is enhanced when the individual is a part of a larger community that can share each other's work [13-15]. For programming environments, the Internet seems a likely medium to allow sharing amongst a distributed community, but current mechanisms that allow this exchange (FTP and email) are difficult to use, and are not directly integrated into the programming environment. To perceive the benefits of being in a community of practice, the mechanisms that allow sharing must be easy-to-use, and tightly integrated with the rest of the end user programming environment.

2.4. Moving between Worlds

Tactile programming relies on the ability of end users to easily cross the boundaries between the application world, the programming world and the collaboration world. The philosophy behind tactile programming is not to think of the integration of the collaboration world as an afterthought, but to create a programming environment that takes all three worlds into account and provides a *unified* approach for accessing them.

3. Comprehension

Tactile programming enhances a user's ability to comprehend programs by augmenting visual perception, used in visual programming, with perception by manipulation. In tactile programming, objects in the programming world are elevated to the status of concrete, tangible objects that can be manipulated by the end user. In Visual AgenTalk the manipulation of commands via drag and drop does not only allow end users to *move* commands and rules between worlds but also to comprehend (Figure 4 1) their functionality by *applying* them to agents. Along the lines of dynamic programming languages such as Lisp and Smalltalk, any command or rule can be executed at any point in time resulting in immediate feedback.

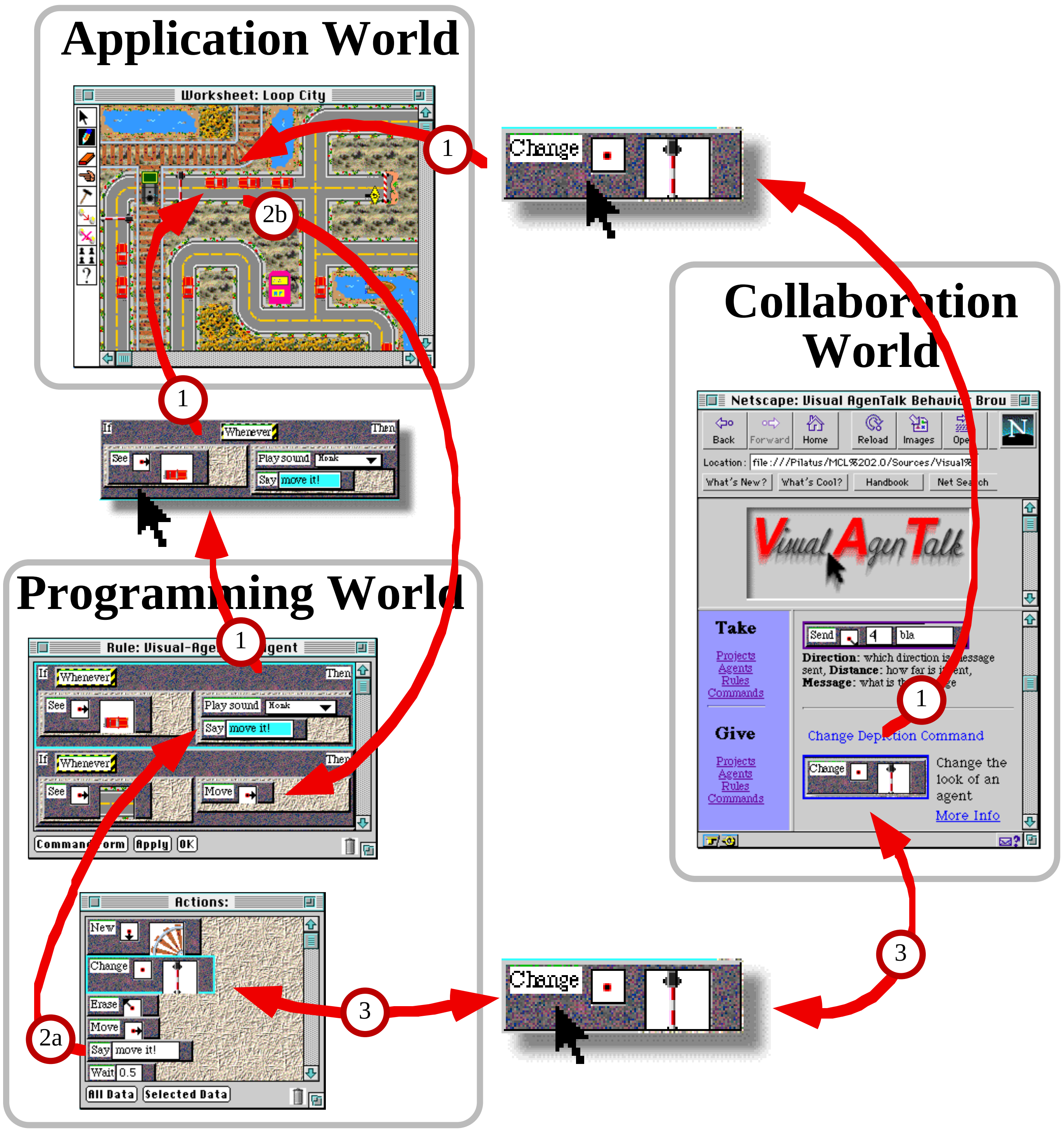


Figure 4: Moving fluently between worlds: ① *Comprehension*: test the functionality of commands and rules by moving them from the programming world or collaboration world into the application world. ②a *Direct Composition*: select commands and compose them into rules. ②b *Composition by Example*: compose rules by manipulating the application world. ③ *Sharing*: share with a community of users entire simulations, agents, rules and commands through the World Wide Web.

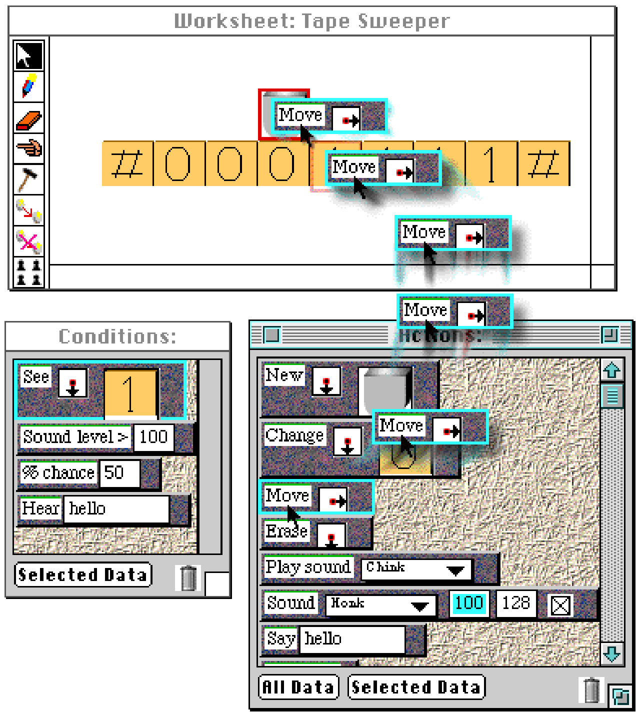


Figure 5: Commands can be dragged and dropped onto agents to explore their functionality and to modify agents

A command is applied to an agent and executed by selecting the command from a palette, specifying the command's parameters, and dragging and dropping the command onto the agent in the worksheet. Figure 5 shows a simple Agentsheets Turing machine. The top window is the worksheet representing the application world. It contains a number of agents that act as the tape pieces and the Turing machine head. Below the worksheet are two command palettes representing part of the programming world. The condition command palette, on the left, holds commands that query the state of the application world. The action command palette, on the right, holds commands that can effect the application world. In the figure, the "move" action command was selected, its direction parameter was set to ➡ (right) via a graphical pop up menu, and the command was then dragged on top of the Turing machine head. Dropping the command on the Turing head results in the command's execution. The Turing head will move according to the direction indicated in the command, in this case one position to the right. This powerful[1] application and execution paradigm allows the end user to explore and test the repertoire of commands and parameters in the context of any agent, simplifying the task of program comprehension.

Even more significant for comprehension than exploring the repertoire of commands is the ability to test entire rules. Problems identified regarding rule comprehension [16] and debugging [17-19] stand in contrast to claims of modularity in rule-based approaches [20]. While the modularity of rule-based approaches can indeed simplify the process of writing rules it can increase the complexity of debugging them. We believe that this problem is at least partly due to the lack of the ability to test individual rules. Even the ability to trace rule execution, found in many rule-based systems, is limited as it will only provide insights about rules that are actually executed. In Visual AgenTalk each part of the rule, the IF part, the THEN part, individual condition or action commands, and even entire rules can be dragged and dropped onto any agent at any point in time to test them. Applying a rule to an agent via drag and drop will execute the rule with audiovisual feedback. For instance, applying the car rule in Figure 3 to an agent will sequentially highlight the condition commands (testing for the presence of a gas station, then testing the energy level.) If all conditions succeed then the actions will be executed sequentially (the agent moves next to the gas station and sets its energy to 100). Should any condition fail, the execution feedback plays a sound while providing visual feedback indicating which condition failed.

Although the tactile approach to testing is helpful in assisting a single user to comprehend his or her own programs, it becomes crucial to support the comprehension of programs created by others. The tactile nature of programming world objects, enabled by the drag and drop interaction mechanism extends the notion of object-orientation. Tactility encourages a more exploratory style of programming in which end users perceive the functionality of programming world objects by "touching" them in a direct manipulation sense.

4 Composition

The process of tactile programming includes the creation of programs by composing them from programming world objects. Similar to Boxer [21], and LiveWorld [22], Visual AgenTalk employs a spatial metaphor of nested containers to represent programs. Composition consist of putting programming world objects into containers. Programming objects can be containers themselves. Visual AgenTalk supports two different approaches for putting objects into containers: Direct composition and composition by example.

[1]It is difficult to convey the power of tactility in a static medium such as paper.

4.1 Direct Composition

End users *directly compose* programs by selecting commands from the condition or action palettes, and by dragging them into rules contained in a Visual AgenTalk rule editor (Figure 4 2a). For instance the Turing machine (Figure 5) could be programmed with rules in order to behave according to this table:

If	Then
there is a "0" below	move right
there is a "1" below	change it to "0" and move right

The resulting Visual AgenTalk program consists of two rules:

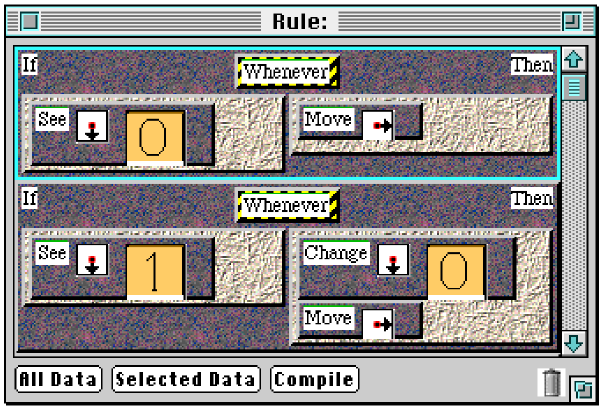


Figure 6: Turing machine rules.

Rule sets are interpreted from top to bottom. If the rule interpreter finds a rule for which all the conditions are true then it will execute all of its actions, from top to bottom, and return from the matching cycle.

4.2 Composition by Example

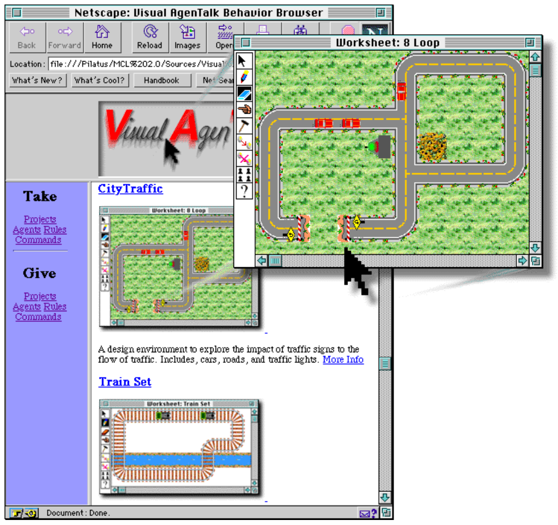
Since tactile programming reduces the barriers between the application word and the programming word, it is also possible to use programming by example techniques [23] to allow the composition of programming word objects through the manipulation of objects in the application word (Figure 4 2b). For instance, the first Turing machine rule in Figure 6 can be composed in Visual AgenTalk by holding down a special "composition by example" key, and by manipulating the objects that are to be programmed. By holding the key down, and clicking on the "0" tape segment below the machine head in Figure 5, Visual AgenTalk infers that an important part of the rule's condition is that a "0" is located below the tape head. Then, the user simply drags the head to the right, and Visual AgenTalk adds the Move command to the rule action. Both the condition and action parts of the rule can be composed of an arbitrary number of commands that can be specified by manipulating application world objects.

5. Sharing

In a tactile programming environment, the same manipulation paradigm used to minimize the barriers between the application world and the programming world can also be used to access the collaboration world. The World Wide Web along with its increasing acceptance is turning into a promising medium enabling collaboration. However, the current Web programming environments (Java®, Java Script®) provide little support to end users. While Java may serve an essential role for networking, similar to the role that Postscript® plays in printing, it will be necessary to add higher level program manipulation paradigms such as tactile programming to make Web programming accessible to end users. The ideal environment would take advantage of tactility and allow the easy sharing of programming world and application world objects.

The Visual AgenTalk Behavior Exchange extends our initial system, called the Remote Explorium [24], by adding different levels of sharing granularity. Using the same tactile manipulation paradigm end users can share entire Agentsheets simulations, individual agents, rules and commands (Figure 4 3).

- ***Sharing Simulations:*** Simulations are the coarsest objects users can share. Once end users find simulations of interest on the Visual AgenTalk Behavior Exchange Web page they can, in the spirit of tactile programming, simply drag the simulation out of the page into Agentsheets. The simulation includes all the objects necessary to use it: including the agents with their rules, commands and resources such as sounds, icons and documentation.

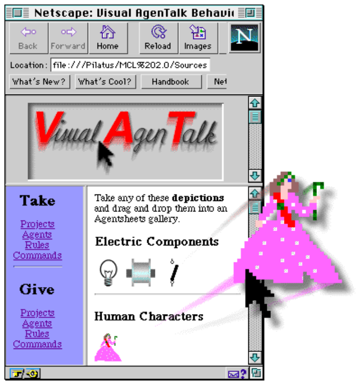


A design environment to explore the impact of traffic signs to the flow of traffic. Includes, cars, roads, and traffic lights. More Info

Figure 7: Dragging A Simulation from the Web Page

Figure 7 shows the City Traffic simulation being dragged out of the Web page. In doing so, all of the necessary programs and resources are automatically downloaded to the user's computer.

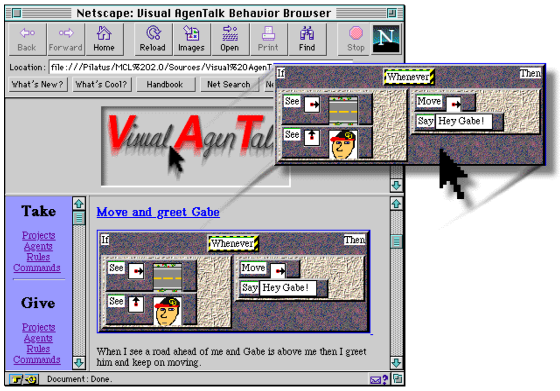
- ***Sharing Agents***: Dragging agents out of a Visual AgenTalk Behavior Exchange page will copy its look and behavior into Agentsheets. That is, the agent comes with all of its rules and the commands contained in the rules.



Take any of these **depictions** and drag and drop them into an Agentsheets gallery.

Figure 8: Dragging an Agent from the Web Page

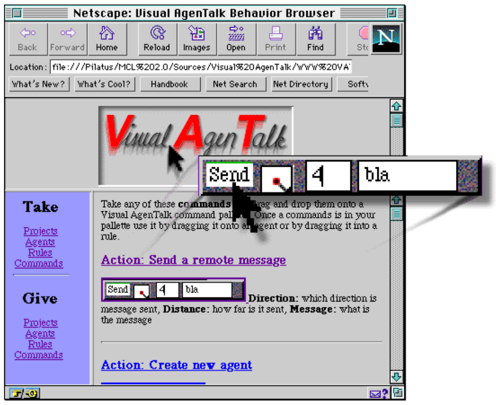
- ***Sharing Rules***: Rules can be dragged either into a Visual AgenTalk rule editor (Figure 4 3) to get a copy of them, or directly onto application world objects in order to comprehend what they do (Figure 4 1).



When I see a road ahead of me and Gabe is above me then I greet him and keep on moving.

Figure 9: Dragging a Rule from the Web Page

- ***Sharing Commands***: Like rules, commands can be dragged into the programming world to get a copy (Figure 4 3) or they can be dragged into the application world to comprehend them (Figure 4 1). Visual AgenTalk is an open programming language allowing language designers to create their own specialized commands and share them with the programming community. In the figure below, a Send command used to send messages to distant agents is dragged off the Web page. Dragging a new command into the programming world will compile the new command's function, create a user interface for the command and install it into the command palette.



Take any of these **commands** ... drag and drop them onto a Visual AgenTalk command pal... Once a commands is in your pallette use it by dragging it onto a... agent or by dragging it into a rule.

Action: Send a remote message

Send 4 bla **Direction**: which direction is message sent, **Distance**: how far is it sent, **Message**: what is the message

Action: Create new agent

Figure 10: Dragging a Command from the Web Page

Visual AgenTalk gains power from the collaboration world by allowing a community of users to create a growing collection of easily sharable components. Depending on the experience of the users, they can get involved to different degrees within the community. They may only wish to access simulations without doing any programming themselves, or they can extend existing simulations by programming in Visual AgenTalk. Users can create a new simulation from scratch and share it with others or they can even create their own programming language featuring specialized commands and share the commands with other users.

6. Conclusion

In order to create end user programming environments that allow program comprehension, composition and sharing to occur easily, it is not enough to develop better visualizations of languages. Instead, we argue that such an environment needs to employ the tactile programming techniques described in this paper to provide a unified program manipulation paradigm that allows users to easily cross the boundaries between the application world, the programming world and the collaboration world. Allowing this to happen not only lowers the threshold to programming, but also provides a flexible framework that allows the definition of highly expressive languages and programs. Visual AgenTalk exists as a working prototype, and we are currently exploring the creation of tactile programming environments in other languages including Java.

Acknowledgments

We wish to thank Gerhard Fischer and the Center for LifeLong Learning and Design for all of the thoughtful discussions. We also want to thank Corrina Perrone for her excellent work, and Kurt Schneider, Kumiyo Nakakoji, Tamara Sumner and Anders Morch for their feedback. This work has been supported by the Advanced Research Projects Agency under Cooperative Agreement Number CDA-940860, and the National Science Foundation under grant number RED925-3425.

References

1. Bell, B., J. Rieman and C. Lewis, "Usability Testing of a Graphical Programming System: Things We Missed in a Programming Walkthrough," *Proceedings of CHI'91*, New Orleans, LA, 1991, pp. 7-12.

2. Repenning, A., "Bending the Rules: Steps toward Semantically enriched Graphical Rewrite Rules," *Proceeding of Visual Languages*, Darmstadt, Germany, 1995, pp. 226-233.

3. Furnas, G. W., "New Graphical Reasoning Models for Understanding Graphical Interfaces," *Proceedings of CHI'91*, New Orleans, LA, 1991, pp. 71-78.

4. Bell, B. and C. Lewis, "ChemTrains: A Language for Creating Behaving Pictures," *1993 IEEE Workshop on Visual Languages*, Bergen, Norway, 1993, pp. 188-195.

5. Smith, D. C., A. Cypher and J. Spohrer, "KidSim: Programming Agents Without a Programming Language," *Communications of the ACM*, Vol. 37, pp. 54-68, 1994.

6. McIntyre, D. W. and E. P. Glinert, "Visual Tools for Generating Iconic Programming Environments," *Proceedings of the 1992 IEEE Workshop on Visual Languages*, Seattle, 1992, pp. 162-168.

7. Nardi, B., *A Small Matter of Programming,* MIT Press, Cambridge, MA, 1993.

8. MacLean, A., K. Carter, L. Lövstrand and T. Moran, "User-Tailorable Systems: Pressing the Issues with Buttons," *Proceedings of CHI'90*, Seattle, WA., 1990, pp. 175-182.

9. Gindling, J., A. Ioannidou, J. Loh, O. Lokkebo and A. Repenning, "LEGOsheets: A Rule-Based Programming, Simulation and Manipulation Environment for the LEGO Programmable Brick," *Proceeding of Visual Languages*, Darmstadt, Germany, 1995, pp. .

10. Wagner, A., P. Curran and R. OBrien, "Drag Me, Drop Me, Treat Me Like an Object," *Proceedings of CHI '95*, Denver, CO, 1995, pp. 525-530.

11. Repenning, A. and T. Sumner, "Agentsheets: A Medium for Creating Domain-Oriented Visual Languages," *IEEE Computer,* Vol. 28, pp. 17-25, 1995.

12. Halbert, D. C., "SmallStar: Programming by Demonstration in the Desktop Metaphor," in *Watch What I Do: Programming by Demonstration*, A. Cypher, Ed., The MIT Press, Cambridge, MA, 1993, pp. 103-124.

13. Roschelle, J., "Learning by Collaboration: Convergent Conceptual Change," *The Journal of the Learning Sciences*, Vol. 2, pp. 235-276, 1992.

14. Brown, J. S., A. Collins and P. Duguid, "Situated Cognition and the Culture of Learning," *Educational Researcher,* Vol. January-February, pp. 32-42, 1989.

15. Pea, R. D. and L. M. Gomez, "Distributed Multimedia Learning Environments: Why and How?," *Interactive Learning Environments,* Vol. 2, pp. 73-109, 1992.

16. Gilmore, D., K. Pheasey, J. Underwood and G. Underwood, "Learning graphical programming: An evaluation of KidSim," *Proceedings of the Fifth IFIP Conference on Human-Computer Interaction*, London, 1995, pp. .

17. Carver, S. M. and S. C. Risinger, "Improving Children's Debugging Skills," in *Empirical Studies of Programmers: Second Workshop*, G. M. Olson, S. Sheppard and E. Soloway, Ed., Ablex Publishing Corporation, Norwood, New Jersey, 1987, pp. 147-171.

18. Nanja, M. and C. R. Cook, "An Analysis of the On-Line Debugging Process," in *Empirical Studies of Programmers: Second Workshop*, G. M. Olson, S. Sheppard and E. Soloway, Ed., Ablex Publishing Corporation, Norwood, New Jersey, 1987, pp. 172-184.

19. Papert, S., *Mindstorms: Children, Computers and Powerful Ideas,* Basic Books, New York, 1980.

20. Cooper, T. and N. Wogrion, *Rule-based Programming with OPS5,* Morgan Kaufman Publischers, Inc., San Mateo, CA, 1988.

21. diSessa, A. A., "An Overview of Boxer," *Journal of Mathematical Behavior,* pp. 3-15, 1991.

22. Travers, M., "LiveWorld: A Construction Kit for Animate Systems," *Proceedings ofCHI '94*, Boston, MA, 1994, pp. 37-38.

23. Cypher, A., Watch What I Do: Programming by Demonstration, MIT Press, Cambridge, MA, 1993.

24. Ambach, J., C. Perrone and A. Repenning, "Remote Exploratoriums: Combining Networking and Design Environments," *Computers and Education,* Vol. 24, pp. 163-176, 1995.