FAR: An End-User WYSIWYG Programming Language for E-speak: Interim Report

TR 00-60-10, October 2, 2000

Margaret M. Burnett and Sudheer Kumar Chekka
Oregon State University
{burnett, chekka}@cs.orst.edu

Abstract

For the past few months, we have been working to design and prototype a WYSIWYG programming tool allowing end users to offer and deliver services through e-speak. This document begins with a brief summary of the design and progress on the prototype to date. We then use an evaluation device called representation design benchmarks *to evaluate FAR's design against some issues derived from cognitive research into programming. The purpose of the evaluation is to find flaws at this early stage that predict problems with usability. As a result of the evaluation, we summarize design changes planned and required.*

1. Introduction

FAR ("Formulas and Rules") is a WYSIWYG programming tool allowing end users to offer and deliver services through e-speak without these users having knowledge of e-speak. It is a one-year exploratory project, and as such is very focused on just one aspect of end-user e-speak: the ability for an end user to offer their own customized services. If the project is continued beyond the one-year exploratory period, the ability to programmatically consume services will eventually also be investigated.

The design philosophy underlying the approach is to provide programming devices aimed directly at supporting the "what" perspective of the user's goals (such as "I want to deliver a custom web page about a flower"), not the "how" perspective of the underlying e-speak componentry required to fulfill the goals. The e-speak basic services supported by FAR allow an end user to publish a service contract (interface) using an existing or a new vocabulary and to provide the service. For example, an owner of a commercial garden shop with a database of information about certain types of flowers, stored on her own PC and maintained using some PC software such as Access, could offer “flower encyclopedia” services as a cottage contribution to a larger digital library.

2. Using FAR to Create E-Services

The way the garden shop owner would use FAR to set up the flower encyclopedia services would be by laying out a sample web page via direct manipulation and specifying rules and spreadsheet-like formulas for dynamically filling in parts of the page based on the incoming query. The sample web page defines a template of how the garden shop owner wants to use the contents from her database to create a custom web page responding to inquiries about flower care and gardening that would come into her computer. Behind the scenes, the ability to receive such queries and to send out a customized web page would occur through the use of e-speak (JESI calls and XML documents), but the garden shop owner does not know this. Rather, her task is simply to lay out the web page and to provide the formulas and/or rules that explain how she wants the fields to be filled in.

These rules and formulas either would use the terminology of an existing e-speak vocabulary or would be a vehicle for establishing a new vocabulary. The user would specify as part of these rules the way to retrieve the necessary information about the flower in order to deliver the requested service, such as by accessing an Access database to look up information on her PC. When the garden shop owner has completed the creation of the sample web page with its rules and formulas, her specifications are saved (in XML format). Pushing a button then invokes the run-time system, which makes the services available until the user stops making them available. In e-speak terms, this means the system knows how to connect and disconnect to e-speak, define and publish a new service contract, find vocabularies, publish new vocabularies, and deploy and deliver services that are dynamically-created web pages (also in XML format).

FAR combines three features to support this kind of programming: support for WYSIWYG web-page layout, ability to incorporate logic by defining spreadsheet-like formulas for objects the user has placed on this web page, and ability to alternatively view and edit these formulas as rules whenever desired.

The combination of these three things is a unique feature of this work. The motivation for the use of formulas to incorporate logic is that millions of end users have demonstrated that they are able to both lay out web pages and use formulas; hence both are proven to be usable by this audience. The integration of rules into this paradigm is due to several reasons.

One reason why rules seemed like a good addition to this paradigm is that there is empirical evidence that end users choose to think in terms of rules when programming some kinds of situations [Pane et al. 2000]. Further, several end-user programming languages have had success with this paradigm [Ambler et al. 1997; Rader et al. 1997; Rader et al. 1998]; examples include KidSim/Cocoa/Stagecast (a series of names for an evolving programming tool for children) [Cypher and Smith 1995; Heger et al. 1998], Altaira [Pfeiffer 1998], and Visual AgentSheets [Repenning and Ambach 1996]. Another reason for the integration of rules comes from our own experience as programmers in our previous spreadsheet-based language, Forms/3 [Burnett and Gottfried 1998]. When writing Forms/3 applications, we found that, whereas in some applications, we wanted to express logic in "pull" terms (referring to cells of interest to Cell A in the usual spreadsheet way), in other applications we wanted to express logic in "push" terms (enumerating how Cell A should affect other cells). The former is easily expressed by formulas and the latter is easily expressed by rules; we wanted to provide users with the flexibility of working in both these alternative expressions of logic.

Thus, in FAR, rules are the mirror-image of the formulas: they express the formulas in "push" terms. The user, not the language implementer, decides which formulas to view as rules. That is, the user can select any cell and drag it to the Rules view, which collects all cells impacted by the same condition into a single rule. She can edit either the rule view or the formula view at any time for any cell. Thus, the user has complete flexibility in switching between the formula-based paradigm and the rule-based paradigm, or working in both simultaneously as desired. See Figure 1 for a snapshot of our current implementation of FAR (in progress).

3. FAR's Representation Design Benchmark Scores

It is possible to bring research into cognitive issues of programming to bear upon visual programming language (VPL) design decisions by considering Green et al.'s *cognitive dimensions*, a distillation of psychology of programming knowledge into a form usable by non-psychologists [Green and Petre 1996].

As a concrete application of the cognitive dimensions, *representation design benchmarks* [Yang et al. 1997] were designed in an earlier collaboration between Oregon State University and Hewlett-Packard. The benchmarks are a flexible set of measurement procedures for VPL designers to use when designing new static representations for their languages. They focus on the static representation part of a VPL, and provide a designer with a yardstick for measuring how well a particular design fulfills design goals related to the static representation's usefulness to programmers. These benchmarks measure the VPL's navigable representation (S, NI) where S is the VPL's static representation and NI is the VPL's navigational instrumentation. *Navigational instrumentation* is the set of devices that take a static representation as input and map it to a subset of that static representation as output.

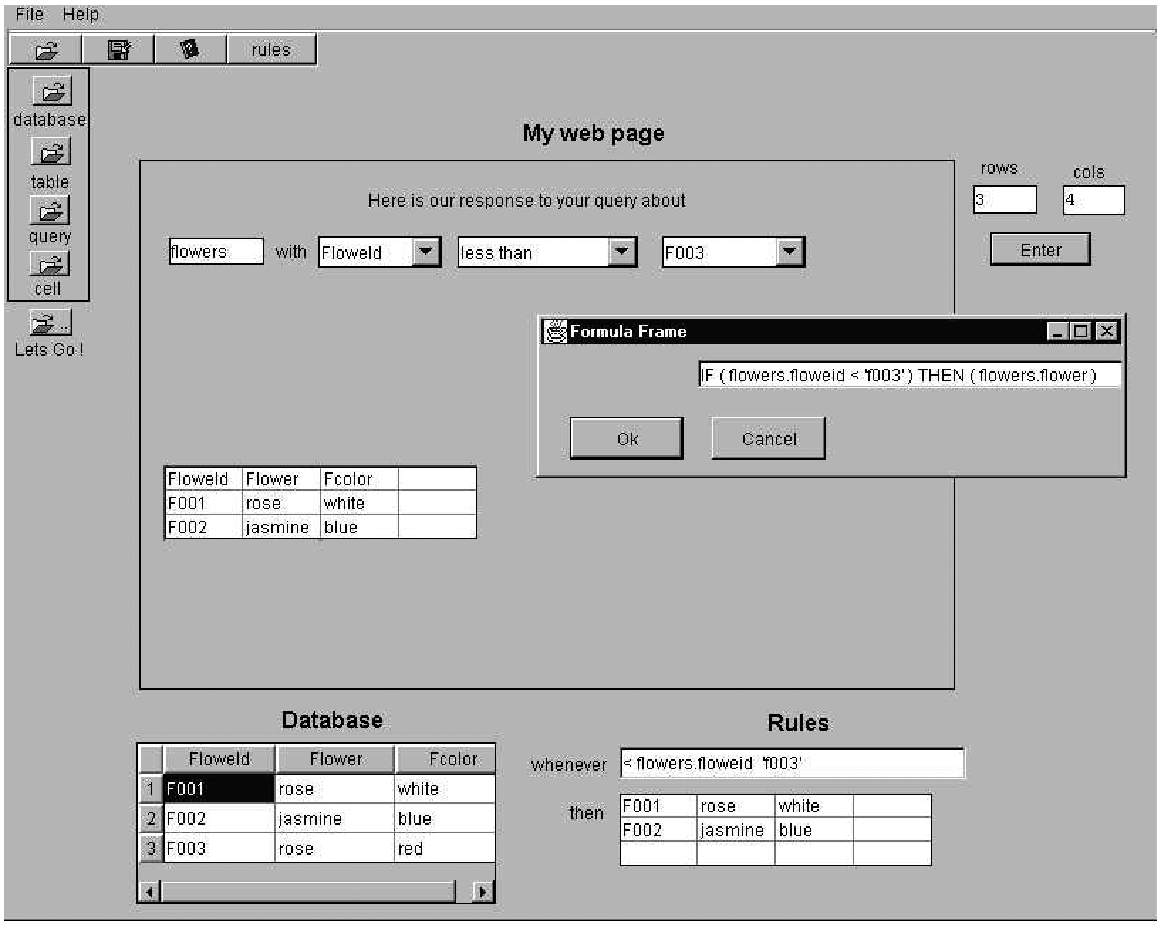
The purpose of representation design benchmarks is to provide a set of early (design-time) measures that VPL designers can use to measure and improve their design ideas. Any design problems found relating to eventual human usage at this early stage are considerably less expensive to both find and fix than would be the case if design problems were not found until a prototype complete enough for usability testing were ready.

The evaluation of FAR using these benchmarks follows.

3.1 Understandability Benchmarks

3.1.1 Visibility of Dependencies (D1, D2)

There is a *dependency* between P1 and P2 if changing some portion P1 of a program changes the values stored in or output reported by some other portion P2. Dependencies are the essence of common programming/maintenance questions such as “What will be affected if P1 is changed?” and “What changes will affect P2?” Green and Petre noted hidden dependencies as a severe source of difficulty in understanding programs [Green and Petre 1996]. In FAR, program



Here is our response to your query about

flowers with FloweId less than F003

rows: 3 cols: 4

IF (flowers.floweid < 'f003') THEN (flowers.flower)

FloweId	Flower	Fcolor	
F001	rose	white	
F002	jasmine	blue	

	FloweId	Flower	Fcolor
1	F001	rose	white
2	F002	jasmine	blue
3	F003	rose	red

whenever: < flowers.floweid 'f003'

then:

F001	rose	white	
F002	jasmine	blue	

Figure 1: Snapshot of FAR prototype.

portions are databases, tables, queries, and cells (examples of each are in Figure 1).

D1 measures whether the dependencies are explicitly depicted in the static representation of a VPL. D1 = (Sources of dependencies explicitly depicted) / (Sources of dependencies in system). In FAR, the sources of dependencies are direct (2), transitive (2) and query (2) dependencies. In the figure, a formula is shown in the small frame for the selected cell (here the "rose" cell is selected by a single click on it). One direct dependency is shown: the cell references that are used directly in the formula. The other direction, which cells the "rose" cell affects, can be seen in the rules view. Query dependencies (the dependency of the results in the table on the query) are not explicitly shown at all. Also transitive dependencies (that the results in the table depend on the query which in turn depends on the database) are not shown. Since only two of these possibilities explicitly are shown, D1 = 2/6.

D2 is the worst-case number of steps required to navigate to the display of dependency information. It takes *n* steps to display all formulas (one at a time), where *n* is the number of cells, so D2 = *n*.

Design implications: D1's low score shows that the formula representation scheme needs improvement. We could reduce the D2 score by including a "Show All" button to display all the formulas corresponding to the cells. This would reduce D2 to *n*/2. We have not yet decided whether *n* is large enough to warrant this extra button.

3.1.2 Visibility of Program Structure (PS1, PS2)

Program structure is the relationships among all the modules of a program. From the programmer's standpoint, a depiction of program structure answers questions such as "What modules are there in this program?" and "How do these modules logically fit together?" Example depictions of program structure in other languages include call graphs, inheritance trees and diagrams showing the flow of data among program modules. In FAR, the main modules are the database, the query, and the sample web page.

PS1 measures the presence or absence of program structure in the static representation and takes Yes/No. In FAR, there is no explicit representation of how the above-mentioned modules of the program logically fit together, so PS1 = "No." However, since there are only three modules, we do not plan to change this.

PS2 measures the worst-case number of steps required for a programmer to navigate to the display of the program structure. Since PS1 is "No," PS2 is not applicable.

Design implications: None.

3.1.3 Visibility of Program Logic (L1, L2, L3)

If the fine-grained logic of a program is included in a static representation, we will say the program logic is *visible*. We will say the visibility of the program logic is *complete* if the representation includes a precise description of every computation in the program. Textual languages traditionally provide complete visibility of fine-grained program logic in the (static) source code listing, but some VPLs have no static view of this information. Without such a view, a programmer's efforts to obtain this information through dynamic means can add considerably to the amount of work required to program in the language. For example, one study of spreadsheet users found that experienced users spent 42% of their time moving the cursor around, most of which was to inspect cell formulas [Brown and Gould 1987].

L1 measures whether the static representation provides visibility of the fine-grained program logic and takes Yes/No. The formulas of the cells in the corresponding formula frames do show how an element is computed, so L1 = "Yes." Alternatively, logic can be seen in the rule view, but the cost is the same.

L2 measures the worst-case number of steps required to display the code. As pointed out for benchmark D2 above, if the only way to show the formulas is one formula at a time, it will take *n* steps to display all the formulas (L2=*n*). If we add the "Show All" button, the average number of steps is *n*/2 (L2 = *n*/2).

L3 is the number of sources of misrepresentations of generality. There is a source of misrepresentation in the use of concrete values in the displayed formulas, such as "f003" in the formula in Figure 1. The comparison as shown is not really the formula being used by the system; behind the scenes, the formula actually refers to abstractly to the current query's value. Thus L3 = 1, which indicates there is a problem.

<u>*Design implications:*</u> The misrepresentation problem revealed by L3 will need to be solved to avoid misleading the user as to exactly what computation is being done here.

3.1.4 Display of Results with Program Logic (R1, R2)

This group of benchmarks measures whether it is possible and feasible to see a program's partial results displayed with the program source code fragment that produces each partial result. The ability to display fine-grained results (values of each cell, etc.) at frequent intervals allows fine-grained testing while the program is being developed, which has been shown to be important in debugging [Green and Petre 1996].

R1 measures whether or not it is possible to see the partial results displayed statically with the program source code and takes Yes/No. It is possible to do this in FAR. In both formulas and Rules views, results of the query and all formulas are displayed immediately with the code, so R1 = "Yes."

R2 is the worst-case number of steps required of the programmer to display results with the source code. All final and partial results are automatically and immediately displayed and updated, so R2 = 0.

<u>*Design implications:*</u> None.

3.1.5 Secondary Notation: Non-Semantic Devices (SN1, SN2)

Secondary notation is the collection of optional non-semantic devices that a programmer can include in a program to clarify its meaning, such as comments in traditional languages.

Changing an instance of secondary notation, such as a textual comment, does not change a program's behavior. Petre argues that secondary notation is crucial to the comprehensibility of graphical notations [Petre 1995]. The use of secondary notations allows clarifications and emphases of important information such as structure and relationships. Four such devices are measured by SN1 and SN2: optional naming, layout devices with no semantic impact, textual annotations and comments, and static graphical annotations.

SN1 measures the presence of these notational devices. SN1 = SNdevices/4. In FAR, naming is optional and the tables and cells in the Rules view can be grouped. It is possible to add comments and static graphic notations anywhere simply by adding cells that have the attribute "hidden", and whose formulas produce strings or graphics.

Layout devices have semantic impact in the web page view, because they affect the position of elements on the final output. However, each unique if-condition defines a rule antecedent (labeled "whenever" in the figure's Rules view), and the union of all the corresponding then-conditions comprises the rule's "then." When the rules are displayed in the Rules view, the user can rearrange the elements within a "then" without semantic impact, so in this view, layout devices are not semantic. Even so, the Rules view is transient, so layout devices used are not permanent parts of the program; thus we consider them about "half" present in SN1. Thus, SN1 = 3.5/4.

SN2 is the worst-case number of steps required to navigate to the secondary notations. Most of the above-mentioned devices are always visible, except for the hidden cells, and a "Show Hidden" button is planned to make those visible in 1 button click. Thus SN2 = O(1) in the worst case for most of the non-semantic devices. However, the Rules view requires one or more cells to be selected first, and the worst-case cost to get all cells in that view is O(r), where r is the number of unique rule conditions ($\leq n$, the number of cells).

Design implications: Both SN1 and SN2 can be improved if the Rules view layout is made a permanent part of the saved program.

3.2 Scalability Benchmarks

3.2.1 Abstraction Gradient (AG1, AG2)

The term *abstraction gradient* refers to the extent to which a VPL supports abstraction. Abstraction is a well-known device for scalability in programming languages, because it usually reduces the number of logical details a programmer must understand in order to understand a particular aspect of a program. It also allows a larger fraction of a program to fit on the physical screen, since replacing a collection of details by an abstract depiction almost always saves space.

AG1 measures the sources of details that can be abstracted away from a representation. In a VPL, 4 such sources of details that can be abstracted away are data details, operation details, other fine-grained portions of the program, and details of navigational instrumentation devices. AG1=AGsources / 4.

In FAR, the data details (table, etc.) can be abstracted away only to some extent by adjusting the table's size, which brings up scrollbars (score for this: 1). The operational details in formulas are abstracted by the "Hide All / Show All" button or by collapsing the formulas one at a time (1). The other portion of the program, the database, can be shrunk to the same extent as the tables (1). The navigational instrumentation devices like control panel that includes tool palette cannot be abstracted away (0). Summing up these partial scores gives AG1 = 3 / 4.

AG2 measures the worst-case number of steps required to abstract the above details away. In FAR, as mentioned above, data details of a table can be abstracted away by grabbing a table and shrinking it (O(*n*) where *n* is the number of tables) and operation details in *n1* (or *n1*/2 steps using a "Hide All / Show All" feature as mentioned above) where *n1* is the number of cells. We can combine these factors by taking a worst-case view that the number of tables is the same as the number of cells, giving AG2 = O(*n*).

<u>*Design implications:*</u> The lack of abstraction ability for the navigational instrumentation devices does not seem like a serious problem, since there aren't many such devices so far. However, the navigational cost of O(*n*) to abstract away the other kinds of details could become excessive if the user includes a large number of tables on the web page. Thus, this is an aspect that may need further attention in the future.

3.2.2 Accessibility of related information (RI1, RI2)

This measures a programmer's ability to display desired items side by side. Green and Petre argued that viewing related information side by side is essential, because the absence of side-by-side viewing amounts to a psychological claim that every problem is solved independently of all other problems.

RI1 measures whether it is possible to include all related information, side by side, in a VPL's static representation. The formulas view in FAR is not a strong Yes, because rearranging cells and tables on the sample web page also rearranges the final output. However, the Rules view is a Yes, since any set of cells and tables can be arranged side by side in the Rules view.

RI2 measures the number of steps by the user to accomplish this. In the Rules view, all related tables could be displayed and rearranged as desired, so in this sense, side-by-side placement of related information is possible. Still, the window's size is limited, and scrollbars eventually replace physical space if too many related tables are present, so RI2 in the worst case = O(*n*), where *n* is the number of objects the user must scroll past to view the desired part of the Rules view. Further, this navigation must be done each time the program is re-loaded, since the Rules view is transient.

<u>*Design implications:*</u> See SN1 and SN2.

3.2.3 Use of Screen real estate (SRE1, SRE2)

Screen real estate denotes the size of a physical display screen, and connotes the fact that screen space is a limited and valuable resource. This group of benchmarks provides measures of how much information a representation's design can present on a physical screen without obscuring the logic of the program.

SRE1 is the maximum number of program elements that can be laid out on the common screen size expected for the particular VPL. SRE2 measures the number of intersections and overlaps that occur while measuring SRE1. In the case of FAR, expected screen size is an ordinary PC-sized screen. Making enough space for the file menu, tool palette, database and Rules view as shown in the snapshot, but expanding the window's width to fill a PC-sized screen, the total number of cells that can be placed—without formulas—in the formulas view is 228 without obscuring each other or overlapping the web page section border (allowing SRE2 to be 0). If formulas are shown below each cell (average length: about 4 value-sized columns), half the number of rows and one fourth the number of columns are possible. Thus, with formulas showing, 228/8, or approximately 28 cells, can be viewed at once if SRE2 is held to 0.

Design implications: The ability to view 28 formulas at once (including the accompanying values) seems adequate for defining a web page template, so no changes are planned.

3.3 Audience-Specific Benchmarks (AS1, AS2, AS3)

These benchmarks compare the representation elements with the prerequisite background expected of the VPL's particular audience. This considers the question of whether programming in a given language is similar to the way its audience might solve similar problems without using FAR. ASn = ASyes's/ASquestions—where ASyes's = the number of "yes" answers, and ASquestions = the number of itemized questions—given questions of the general form: "Does the static representation element look like the *object /operation / spatial composition mechanism* in the intended audience's prerequisite background?"

Note that this set of benchmarks does not predict whether the task is easy or hard for the user. Rather, it compares what the language designers are prepared to require as the prerequisite background to program in the language with the way programs in that language appear on the screen. For FAR, the programmers are end-users. Our assumption of their background is that a FAR user has worked with some spreadsheet product such as MS Excel, has worked with some database product such as MS Access, and also has basic familiarity with the web.

AS1 compares all the *objects* in the representation with the corresponding elements in the audience's experience and background. In FAR the objects are free-standing cells, queries, tables, and databases. Except the freestanding cells, the others are in accordance with the audience background; that is, we assume from their prerequisites that they have seen queries, tables, and databases that look similar to the ones in FAR. Hence AS1 = 3/4.

AS2 considers *operations*. In FAR, these are the formulas and rules. Although spreadsheet formulas are generally within their background, many of these formulas use the *if* operator, which is not commonly known by most spreadsheet users. Further, many of them do not have prior experience using rules. Thus, we do not consider FAR operations to be in accordance with the audience background, and AS2 = 0.

AS3 considers *spatial composition mechanisms* for the objects and operations. The web page will show the results of a query in the traditional database style of rows and columns. Further, the web page layout depicts the intended layout of a web page. Thus, these spatial composition mechanisms are the same as those in the prerequisite background, and AS3 = 1.

Design implications: AS2's low score shows that neither the formulas nor the rules are within the audience's prior experience. Of course, the score does not actually prove that they will not find it easy, and in fact some rule-based formats have been empirically shown to be successful for end-user programmers. Still, this could be critical to the users' understanding of FAR. We are reworking the syntax of conditional formulas, but leaving the rules unchanged at this point. If pilot studies later show that the rules are difficult for FAR users, a more graphical rule syntax, such as those used by Stagecast or Visual AgentSheets, could be considered.

4. Conclusion

As a result of applying the representation design benchmarks to FAR, several design issues were uncovered. Some may or may not become problems, depending on the sizes of programs users eventually write. These include:

- Possibly adding a "Show All Formulas" button (from benchmark D2).
- Make the transitive dependencies explicitly visible (from D1).
- Lower the cost of abstracting away details (AG2).
- Abstracting away the Navigational Instrumentation (NI) devices like the control panel.
- Consider alternative, more graphical, syntax for rules (AS2).

Three issues arose that clearly must be addressed, and we are already planning design changes to do so. These issues are:

- Making the query dependency explicitly visible (from benchmarks D1 and L3).
- Make the Rules view layout permanent (from SN2, RI2).
- Change the formula syntax to something closer to the users' background (from AS2).

References

[Ambler et al. 1997] Allen Ambler, Thomas Green, Takayuki Dan Kimura, Alexander Repenning, and Trevor Smedley, "1997 Visual Programming Challenge Summary," *1997 IEEE Symposium on Visual Languages*, Capri, Italy, 11-18, Sept. 23-26, 1997.

[Brown and Gould 1987] P. Brown and J. Gould, "Experimental Study of People Creating Spreadsheets," *ACM Transactions on Office Information Systems* 5, 258-272, 1987.

[Burnett and Gottfried 1998] Margaret Burnett and Herkimer Gottfried, “Graphical Definitions: Expanding Spreadsheet Languages through Direct Manipulation and Gestures,” *ACM Transactions on Computer-Human Interaction* 5(1), 1-33, March 1998.

[Cypher and Smith 1995] A. Cypher and D. Smith, “KidSim: End User Programming of Simulations,” *Proc. CHI’95: Human Factors in Computing Systems*, Denver, CO, 27-34 May 7-11, 1995.

[Green and Petre 1996] T. R. G. Green and M. Petre, “Usability Analysis of Visual Programming Environments: A ‘Cognitive Dimensions’ Framework,” *Journal of Visual Languages and Computing* 7(2), 131-174, June 1996.

[Heger et al. 1998] N. Heger, A. Cypher, and D. Smith, “Cocoa at the Visual Programming Challenge 1997,” *Journal of Visual Languages and Computing* 9(2), 151-169, April 1998.

[Pane et al. 2000] J. F. Pane, C.A. Ratanamahatana, and B.A. Myers, “Studying the Language and Structure in Non-Programmers’ Solutions to Programming Problems,” *International Journal of Human-Computer Studies*, 2000 (to appear).

[Petre 1995] Marian Petre, “Why Looking Isn’t Always Seeing: Readership Skills and Graphical Programming,” *Communications of the ACM* 38(6), 33-44, June 1995.

[Pfeiffer 1998] Joseph J. Pfeiffer Jr., “Altaira: A Rule-based Visual Language for Small Mobile Robots,” *Journal of Visual Languages and Computing* 9(2), 127-150, April 1998.

[Rader et al. 1997] Cyndi Rader, Cathy Brand, and Clayton Lewis, “Degrees of Comprehension: Children’s Understanding of a Visual Programming Environment,” *ACM Conference on Human Factors in Computing Systems (CHI’97),* 351-358, March 22-27, 1997.

[Rader et al. 1998] Cyndi Rader, Gina Cherry, Cathy Brand, Alexander Repenning, and Clayton Lewis, “Designing Mixed Textual and Iconic Programming Languages for Novice Users,” *1998 IEEE Symposium on Visual Languages*, Halifax, Nova Scotia, Canada, Sept. 1-4, 1998

[Repenning and Ambach 1996] A. Repenning and J. Ambach, “Tactile Programming: A Unified Manipulation Paradigm Supporting Program Comprehension, Composition and Sharing,” *1996 IEEE Symposium on Visual Languages*, Boulder, Colorado, Sept. 3-6, 1996, 102-109.

[Yang et al. 1997] S. Yang, M. Burnett, E. DeKoven, and M. Zloof, “Representation Design Benchmarks: A Design-Time Aid for VPL Navigable Static Representations,” *Journal of Visual Languages and Computing*, October/December 1997.