CHAPTER

Six

End Users and GIS: A Demonstration Is Worth a Thousand Words

CAROL TRAYNOR
Saint Anselm College

MARIAN G. WILLIAMS
University of Massachusetts, Lowell

S
R
L

Abstract

This chapter presents a programming-by-demonstration (PBD) approach to Geographical Information Systems (GIS). The aim of our approach is to enable non-specialist users to avail themselves of the software without having to resort to the help of expert users. We begin with a story of one group of non-specialist users who encountered difficulty with GIS software. Next we summarize findings of a study of why GIS software is hard for non-specialist users to use. Then we describe in detail the PBD approach for GIS and explain how this component may be integrated into a GIS.

6.1 Introduction

Geographic information systems (GISs) are in wide use by city planners, landscape architects, natural resource managers, and other specialists who have the expertise—or the trained staff—to use them. Many nonspecialists (e.g., community activists lobbying for change in their inner-city neighborhoods) would like to be able to use GIS. However, GIS software is not accessible to them, because, in its current incarnation, it requires a knowledge of geography, cartography, and database systems. Despite an enormous pool of potential nonspecialist users, GIS is not at this time a mainstream, mass-marketed application.

In this chapter, we present a programming-by-demonstration (PBD) approach to GIS that offers promise for enabling nonspecialists to avail themselves of the power of GIS. We begin with a story of a group of nonspecialists who encountered major difficulties when they tried to use a GIS. We summarize the findings of a study about why such users find GIS so very difficult to use. We then describe a PBD approach to GIS that shows promise for making GIS accessible to nonspecialist users, and we explain how this PBD component may be integrated into a GIS.

6.2 A Story of End Users and GIS

Several years ago, we had the opportunity to observe a group of faculty from a local university as they embarked on a project to make a GIS available to residents of a nearby inner-city neighborhood. The goal of the project was

to equip residents of the economically depressed neighborhood, many of whom were recent immigrants, so that they could effectively influence governmental decisions impacting their quality of life. With a GIS, the residents should be able to study and present maps showing the distribution of crime across the city, thereby bolstering their arguments for the placement of a new police precinct within the neighborhood. With a GIS, they should be able to arm themselves with maps showing the proximity of toxic waste sites to their children's schools and lobby effectively to have the sites cleaned up.

The faculty members came from diverse disciplines, including community psychology and public health nursing. They were smart, capable people with advanced degrees and good problem-solving skills. They were regular computer users, proficient with applications for word-processing their academic papers, making slides for presentations, and exchanging email. They approached the adoption of GIS with careful planning. To help them select the right GIS software and get them started using it in the neighborhood project, we engaged a GIS expert for them to consult (Kuhn, Richardson, and Williams 1994). After several long sessions with the consultant, they purchased one of the best-known mass-marketed GIS products.

Despite the careful approach and the consultations with the expert, they found it dauntingly difficult to use the GIS. They were, of course, perfectly capable of going out and taking training courses and understanding the material in such courses. But their time, attention, and brainpower needed to be directed at the content of the project, not at how to use a complicated software tool. Needless to say, it was obvious from the start that if the faculty members could not use the GIS, then it would be even further beyond the abilities of non-computer-using neighborhood residents.

What did the faculty members do? They hired computer science graduate students to use the GIS for them. We call these grad students their *surrogate users.* In styling them thus, we mean no disrespect for the faculty members, who retained responsibility for the intellectual content of the project and who specified the nature of the information displays to be created with the GIS. The surrogate users provided a human interface between the faculty members and the software interface. It was the graduate students who produced the displays that the faculty members and neighborhood residents needed.

Engaging surrogate users has been observed in a variety of other circumstances, when technology has been (intentionally or unintentionally) unavailable for direct use by nonspecialist end users. For example, directory assistance operators have long served as surrogate users of technology to search for telephone listings, though some of that technology has now been

made available to end users on the Web. Like us, Garson and Biggs (1992) and Egenhofer (1995) have observed the need for surrogate users for GIS software.

What exactly did the grad students do for the faculty members? The first task, after spending a nontrivial amount of time learning how to use the software, was to locate, reformat, and import data files. Once the data sets were installed, they used the GIS to create information displays on maps. They also attempted, unsuccessfully, to use the GIS's built-in scripting language to create an application that the social scientists could use. However, the possible customizations were too minimal to help solve the problems. In simple terms, the grad students were needed for (1) getting data in and (2) getting information out. We focus here on the problem of getting the information out of a GIS and on the characteristics of the current crop of GIS software that make it so very difficult for nonspecialists to use.

6.3 Why Is GIS Software So Hard To Use?

What is it about GIS software that makes it so hard to use, so hard to get the information out?

To answer this question, we looked closely at seven commonly used GIS software packages, including the one the faculty members were using. We chose some simple tasks that GIS users are apt to perform repeatedly—for example, opening a map—and analyzed the knowledge and steps necessary for performing them (Traynor and Williams 1995). This task analysis showed that, in general, the GISs had three serious obstacles to use by nonspecialists:

1. They used *technical terminology* and *technical concepts* from cartography, geography, and database systems (e.g., the user needs to know what an overlay is in the cartographic sense and what a query is in the database sense).
2. They required the user to have a mental model of the *software architecture* to perform a task, since the sequence of steps was based on the way data were stored and represented (e.g., the user needs to know the structure of the database).

In other words, each of the GISs required users to translate a task from their own language into its language, as well as to understand its software

architecture. Little wonder, then, that the grad students hired to work on the neighborhood project became immersed in the terminology of GIS and in a way of thinking about their work that the software imposed on them. Even when coached, they had trouble communicating with the users they served, because of the burden of translating between GIS-speak and the real-world terminology of the users.

Another issue common to all of the GISs we examined was the following:

3. They provided *no record of how a display of information on a map was created,* other than, perhaps, a representation in a database query language.

The GIS users we observed created information displays by trial and error, with the result that once they arrived at a useful display, they had no record of how they had achieved it. Also, they had no way to create a similar, but different, display, other than more trial and error. Without a record of how an information display was created (except, perhaps, in a database query language such as SQL that nonspecialists cannot read), a GIS user has no way to *program* the GIS. He or she tends to start over from scratch all the time.

Heuristics for designing this next-generation GIS software follow naturally from the problems we identified with the current generation:

1. Present the human-computer interaction in terms of the user's task.
2. Protect the user from needing technical expertise from cartography, geography, or database systems.
3. Protect the user from having to know about the software architecture.
4. Provide a program representation of the steps for creating an information display.

Some of these heuristics are widely accepted principles of software human factors and familiar rules of thumb for designers of interactive systems. However, they have special applicability to software, such as GIS, that has arisen out of a particular technical community. Interaction that fits the task model and the expertise of a specialist may be incomprehensible to a nonspecialist. Similarly, a program representation, if present, may be represented in a language familiar to the specialist but alien to the nonspecialist.

We observe a pattern to the way technical software gets mainstreamed and becomes successful in the mass market. First, its applicability beyond

the original technical domain becomes widely recognized. There follows a phase in which new users become specialists, or engage the help of surrogate users. Then attempts are made to modify the software to make it usable by nonspecialists. Finally, the software is reconceptualized for the nonspecialist end users. Only then does it succeed in the mass market. Spreadsheets, CAD, and Web browsers are examples of software that has traveled this route.

GIS originated in the geographers' community. Its enormous applicability beyond that community is clear, though at the moment, nonspecialists need surrogate users to exercise the software for them. Now the question is whether we are still tweaking the software in the hopes that small changes will make a difference to nonspecialist users, or whether designers are finally starting to think about GIS from the nonspecialist's point of view.

6.4 Are Things Improving for GIS Users?

We recently revisited the GIS packages that we looked at a few years ago and can report that things have improved for GIS specialist users. The software packages now reside on mainstream platforms, such as Windows. The vendors have started to take usability issues seriously. And more research has been done on how people use spatial data. But what about nonspecialist users?

One of the major vendors of GIS software now has a product aimed at first-time GIS users and has published a book to accompany it. The company's Web site suggests that "everyone" can now use GIS: "Within minutes, you'll be able to create attractive and accurate digital maps that you can print, embed in documents, e-mail, or publish on the World Wide Web." Unfortunately, interacting with the software requires learning many of the same technical concepts and terms (e.g., the technical meanings of "theme" and "coverage") that full-fledged GIS software requires. Queries are just thinly disguised SQL.

In research labs, new ways of querying a GIS are being explored. For example, Standing and Roy (1997) have proposed a visual functional query language for creating macros. Aufaure-Portier (1995) has devised a visual gesture language for describing spatial relationships, such as adjacency and inclusiveness. Richards and Egenhofer (1995) have designed a drag-and-drop interface that lets the user stack up icons, as if they were blocks; the resulting stack shows how information should be used on a map. Ahlberg and Shneiderman (1994) have shown the usefulness of letting users create dynamic database queries by manipulating alphasliders to select

values for the various parameters. Some of these approaches require technical knowledge (e.g., Standing's query language uses icons representing functions such as Union and Buffer). None of them provides a savable, retrievable, editable program representation written in a language that nonspecialists can read.

6.5 How Can Programming by Demonstration Help?

In the PBD paradigm, the software "watches" while the user "demonstrates" how to do a task. The software makes inferences about what the user is doing and creates a program that will do the same thing. Depending on the sophistication of the software, the resulting program may be hard-wired for the specific task the user demonstrated (in which case it functions like a macro), or it may be generalizable to similar tasks. (See Cypher 1993 for extensive discussion and case studies of PBD.)

The PBD software creates a representation of the program it has inferred. Sometimes, the representation is internal and cannot be viewed by the user. Often, it is accessible to the user but written in a language best understood by a trained programmer (e.g., Lisp). However, to be usable by nonspecialists, the program representation needs to be written in a language that is accessible to nonprogrammers. The program representation language needs to be

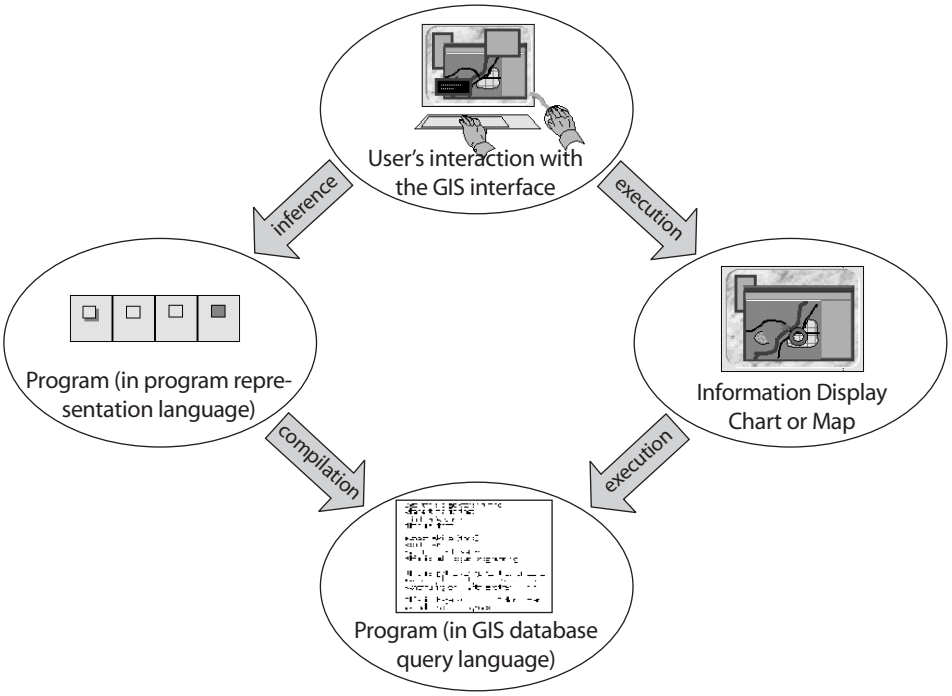
- free of technical terminology,
- independent of the software architecture, and
- expressed in terms of the user's task.

For the program representation to be useful, the user must be able to do more than just see it and read it. The user must also be able to use it in the following ways:

- *view it while it is being constructed,* in order to check whether the software is making correct inferences;
- save and retrieve it for later use;
- *execute it* on demand; and
- *edit it* for performing similar tasks.

S
R
L

FIGURE 6.1



How a user's interactions with the GIS interface are processed.

For a GIS, having a program representation that can be understood by non-specialist users means that a user's interactions with the interface of the GIS are processed in two different ways. First, the instructions are executed and the results shown in an information display, either in a chart or on a map. Second, a program is inferred from the user's actions and encoded in the program representation language. If the user wishes to run the program later on, it needs to be compiled into the GIS's database query language for execution. Figure 6.1 shows the two different ways that the users' interactions with the interface are processed.

If the program has been inferred from the user's actions and the program is run later, the resulting information display should be identical to the one that the user constructed by hand. Of course, if the user has edited the program, then the information display would differ accordingly.

S
R
L

6.6 A Programming-by-Demonstration Approach for GIS: C-SPRL

We are using a PBD approach to making GIS accessible to the nonspecialist users. The specific user population we are designing for can be characterized as people who

- wish to focus on their own domain tasks;
- use a computer as a vehicle to accomplish those tasks;
- have little or no programming knowledge; and
- have no expertise in geography, cartography, database systems, or GIS.

Examples of this kind of user include the concerned mother who wants to ask, "Show me the bars and liquor stores within a mile of my child's school" or the community activist who wants to say, "Show me data on traffic problems in all of the city's neighborhoods"—and who should not need to understand cartography and database systems to formulate their questions.

Figure 6.2 shows the PBD GIS user interface. There are two main parts to the interface: the GIS component (upper part) and the PBD component (lower part).

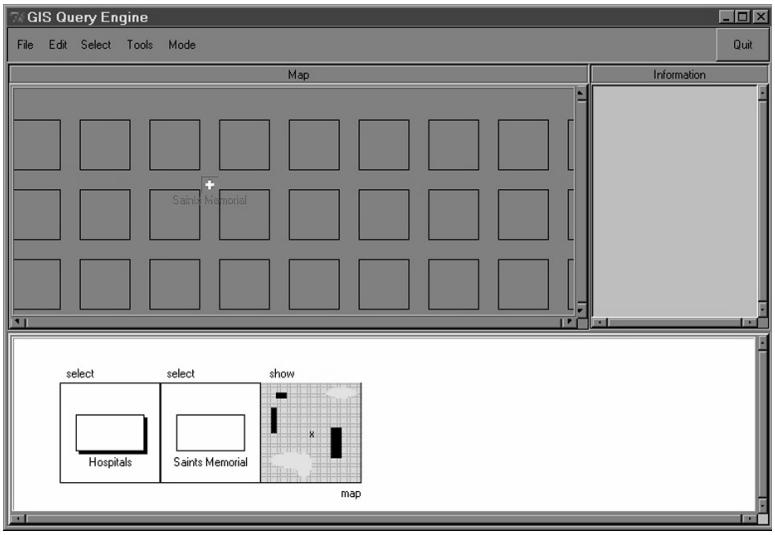
The architecture of the GIS for nonspecialist users corresponds to Figure 6.1. The user interacts with a graphical user interface (GUI) designed to support the way the nonspecialist user thinks about his or her tasks (e.g., *map* is used in the common, everyday sense, not the cartographic sense). His or her instructions are carried out, and an information display is created on a chart or a map or both. Simultaneously, a program representation is created and displayed.

The PBD component for GIS allows the user to interact in two modes: record and edit. In record mode, the user interacts with the GIS GUI, and the system infers the program representation based on the user's actions. Record mode is the default mode. Edit mode allows the user to edit the program representation directly so that he or she can correct or modify a query or create a new query. The user can switch to edit mode by choosing Edit from the Mode menu from the GIS GUI. In Edit mode a menubar is displayed in the PBD window. (See Figure 6.8 later for the options available in Edit mode.)

The program representation language uses a comic-strip metaphor. A program consists of a series of panels, just the way a comic strip does. A

S
R
L

FIGURE 6.2

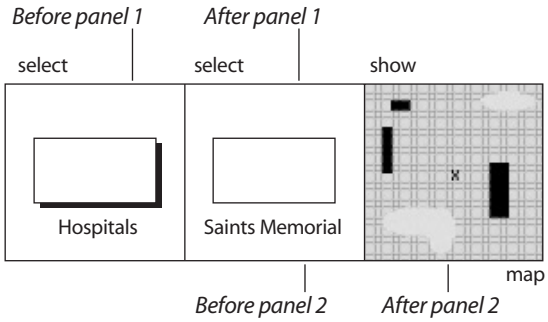


The PBD GIS interface in Record mode. The query depicted is "Show me the location of Saints Memorial Hospital."

panel to the left of another is a *before panel;* one to the right is an *after panel.* An operation on the contents of the before panel results in the contents of the after panel. Thus, a series of panels represents a sequence of transformations, in the temporal order in which they occur. We call the language the Comic-Strip Program Representation Language (C-SPRL, pronounced, "c-spiral"). C-SPRL is an extension of the Pursuit program representation language devised by Modugno (1995). C-SPRL uses the notion of categories of data, in which *category* is used in the normal everyday sense of the word. A category may be composed of one or more subcategories and/or of a number of individual category members. For example, the school system in many U.S. school districts consists of three categories of schools—namely, elementary schools (kindergarten to fifth grade), middle schools (sixth to eighth grade), and high schools (ninth to twelfth grade). In this case the overall category would be "Schools." The category Schools would contain three subcategories: High Schools, Middle Schools, and Elementary Schools. Each of these three subcategories would contain a list of the individual schools in that subcategory. Categories form the basis for queries. Categories, subcategories, and individual members of categories

S
R
L

FIGURE **6.3**



Programs consist of before and after panels. This program depicts the query "Show me the location of Saints Memorial hospital."

are represented by rectangles. A shadow under a rectangle indicates a category or subcategory. A rectangle without a shadow represents an individual member of a category. Data displays are represented by chart panels (called "info") and by map panels (called "map"). (Figure 6.7 later shows the symbols for both the chart and map panels.) The data display panels are abstractions, not realistic representations of actual screen objects as in, for example, Mondrian (Lieberman 1993).

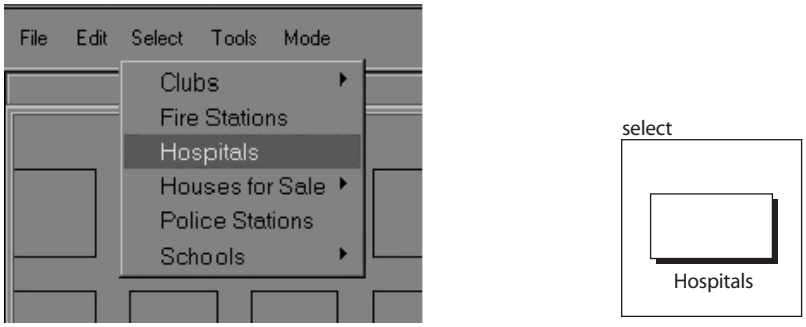
Figure 6.3 shows a very simple C-SPRL program representation with three panels. The query represented is "Show me the location of Saints Memorial hospital." When a user interacts with the GUI of the GIS, the software infers a program and builds a representation of it, panel by panel, in a program display window. To construct the simple example of Figure 6.3, the user interacts with the GUI and begins by selecting the category Hospitals from the Select menu on the menubar (Figure 6.4 [a]).

Once this action has been completed, the first panel appears in the PBD program area (Figure 4[b]). Notice that the category Hospitals is represented by a shadowed rectangle.

A pop-up window with a list of choices appears in the GIS GUI (Figure 6.5). Since there are no subcategories in this category, a list of individual hospitals and a list of the actions or information that can be requested about the available hospitals are displayed.

The user clicks on Saints Memorial. At this point the user can also indicate that he or she would like to see the location displayed on a map by clicking on the map button on the pop-up window. When the user has fi

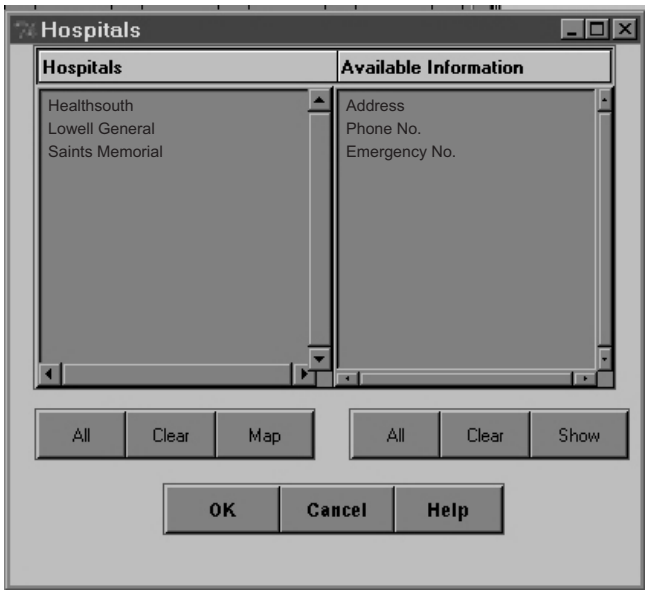
FIGURE 6.4



(a) (b)

(a) The categories from the select menu on the GIS GUI; (b) the panel created in the PBD area as a result of choosing Hospitals (a shadowed rectangle) from the select menu.

FIGURE 6.5



When the user selects Hospitals from the Select menu, a pop-up window with the list of hospitals and actions and/or information that can be requested about the available hospitals appears on the screen.

S
R
L

nished choosing items from the Hospitals popup window, he or she clicks on the OK button. The second and third panel of the program representation now appear on the PBD screen. The location of Saints Memorial Hospital also appears on the map in the GIS GUI. Figure 6.2 shows both the GIS and PBD GUI for the results of this query. Notice also in the program representation that Saints Memorial hospital is represented by a rectangle without a shadow. The third panel is an example of a map panel.

More complex user actions would result in multiple panels being added to the program representation. In building a query, the user uses a top-down approach. At each step in the query-building process, the user refines and narrows the search until reaching the level of granularity that he or she is looking for.

The program representation does a variety of things for the user. First, it provides a visualization of the software's inferences about the user's actions, so the user can correct the software's understanding, if necessary. Each panel depicts a user action, so the user has a step-by-step account of how the query was created. This is also helpful in identifying user errors. Second, the gradual construction of the program representation helps the user to learn C-SPRL by watching what happens as he or she interacts with the GIS's GUI. The representation provides the user with continuous feedback, so if the program representation depicts an "incorrect" action, the user can immediately backtrack and rectify the situation. Third, the program representation is editable. The user can edit the program to modify it or to create a similar, but different, program. The user saves time by being able to run a previously constructed query without having to repeat all the steps. The user can even create an entirely new program from scratch. Fourth, the user can save and retrieve programs for later use. Once again this saves the user time by not having to repeat work/steps already done. Users can easily share results of queries with others. Finally, a program can be executed on demand. Figure 6.6 summarizes the various uses that a user can make of a program representation.

The set of symbols used in C-SPRL program representation is not large in number (Figure 6.7). As a result, users do not have to spend much time familiarizing themselves with the terminology. All program symbols have self-identifying labels that aid user comprehension. C-SPRL also has a set of commands: select, show, and draw. The s*elect* command indicates the selection of a category, subcategory, category member(s), or one or more items from the list of available information about a category or category member. The show command indicates the display of information on a map or in text format. The draw command indicates that a user has narrowed the search to the area highlighted on the map. These commands appear above the panels in the program representation.

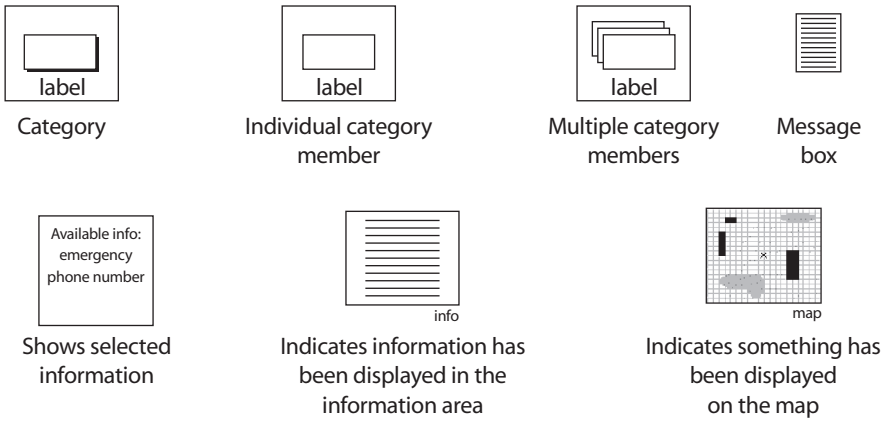
S
R
L

FIGURE 6.6

- Check whether the software is making correct inferences
- Learn C-SPRL by watching the correspondence between GUI interactions and program constructs
- Save and retrieve a program
- Edit a program to modify it or to create a similar program
- Write a program from scratch

Summary of uses of a C-SPRL program representation.

FIGURE 6.7



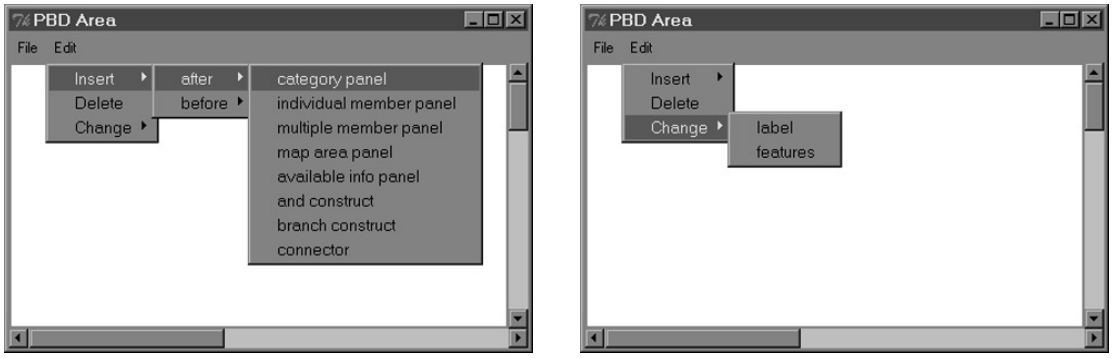
C-SPRL panel icons.

C-SPRL program representations can be edited directly with a special-purpose editor. Figure 6.8 shows the menu options available in the PBD program editor. Panels and program constructs may be added, edited, and deleted. If a before panel is edited or deleted, then all of its after panels are deleted, when their preconditions are no longer met.

C-SPRL includes a branching construct, called *do-one,* and an and-ing construct called *do-all.* Figure 6.9 shows a program that uses the *do-all.* The

S
R
L

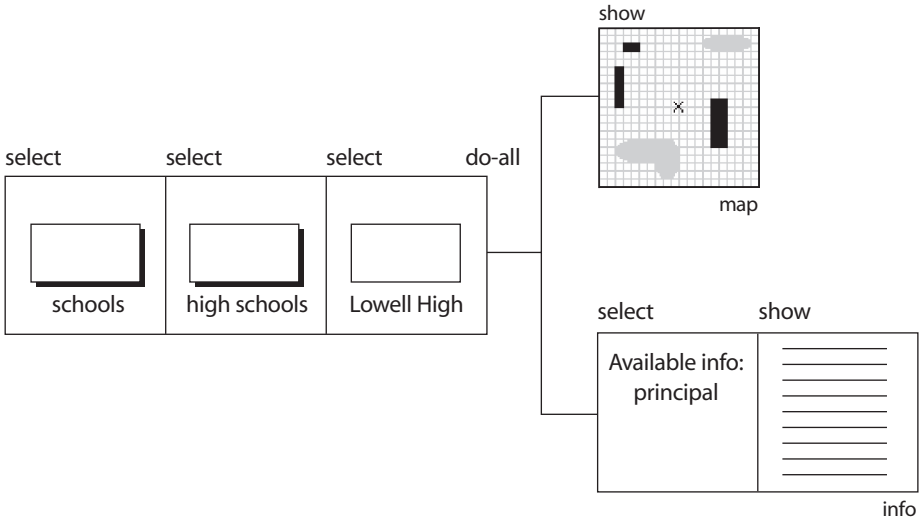
FIGURE 6.8



(a) (b)

Menu options from the C-SPRL editor.

FIGURE 6.9



A C-SPRL program with a do-all construct.

S
R
L

schools category is selected, the selection is refined to include just the high schools, and then it is further refined to an individual school, Lowell High School. The panel for Lowell High is the before panel for two after panels, because two simultaneous operations are performed with the Lowell High data: (1) the school's location is shown on a map, and (2) information about the school's principal is displayed in a chart.

User tests of the C-SPRL program representation language indicate that non–GIS specialists are able to read C-SPRL programs (program interpretation), edit programs to create new programs (program modification), and write programs from scratch (program creation). We also gave C-SPRL programs to study participants and asked them to use the GUI to perform the steps that would cause the software to infer the programs (program navigation). The fact that they were able to navigate the GUI successfully indicates that the participants understood the encoding of actions into the C-SPRL representation.

Figure 6.10 gives samples of the different types of queries used in the studies. Two different studies were performed. As a result of the first study, some of the C-SPRL panel icons were modified; hence some of the panel icon symbols in the program navigation may differ slightly from those in Figure 6.7.

The tasks chosen for the user studies were based on a taxonomy of the types of queries that our end users are likely to perform on a GIS. The taxonomy was derived from a requirements definition for a GIS intended for use by one of our target user populations (Traynor 1998). The taxonomy contains six classes of queries:

1. Show me one or more **objects.**
2. Show me one or more **objects** with/without one or more **features.**
3. Show me one or more **objects** with/without one or more **attributes.**
4. Show me one or more **objects** with/without one or more **features** and with/without one or more **attributes.**
5. Show me one or more **objects** within "this" **distance** of one or more **objects.**
6. Show me one or more **attributes** for one or more **objects.**

More details of these studies can be found in Traynor and Williams (1995, 1997).

S
R
L

FIGURE 6.10

Program Interpretation

Show me the location of all houses for sale with a fireplace.
Show me the practice hours for Pawtucket soccer club.
Show me all the liquor stores within a mile of Lowell High and 1.5 miles of my house.

Program Modification

Original Task

Show me all swimming pools.
Show me all parks within a half-mile of Pawtucket Soccer Club and the number of teams in the club.

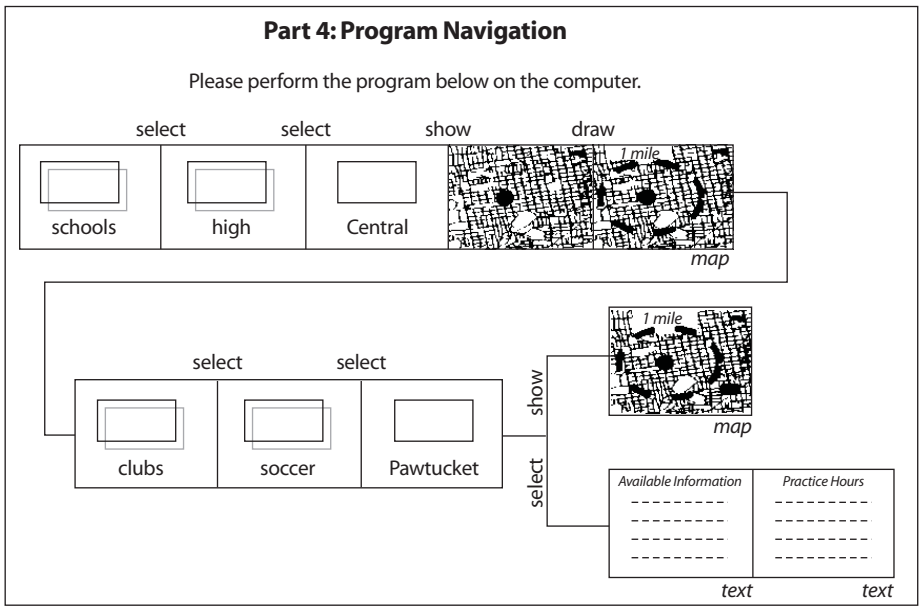
Modified Task

Show me all parks.
Show me all the swimming pools within a mile of the Wang Middle School and the number of teachers in the school.

Program Creation

Show me the Acre Police Station and the number of policemen assigned to it.
Show me all the fire stations within a mile of Murphy's restaurant and the number of fire engines.

Program Navigation



Sample of queries used in the user studies.

S
R
L

6.7 Conclusion

C-SPRL is one possible representation language for a GIS for nonspecialist users. Many others are possible. For example, the block-stacking metaphor of Richards and Egenhofer's (1995) GIS interface would lend itself to a program representation that contains images of stacked blocks. Whatever the representation, it appears possible to create new ways of interacting with GIS software—ways that are not steeped in the traditions of geographers and cartographers. The experts already have GIS software that meets their needs. Attempting to tailor their highly technical software for nonspecialists, either by modifying the interface slightly or by providing a subset of the functionality, has not worked. What is needed is a fresh start, with nonspecialist users viewed as a first-class population of GIS users, rather than an add-on population. Only then will GIS come into its own as a "killer application."

Acknowledgments

We would like to thank the following people for their insightful comments and help in preparing this chapter: the HCI Research Group at the University of Massachusetts Lowell, in particular Dr. J. Nicholas Buehler and Guillermo Zeballos; and Gordon Paynter of the University of Waikato, Hamilton, New Zealand.

References

Ahlberg, C., and Ben Shneiderman. 1994. "Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays." In *Proceedings of Human Factors in Computing Systems Conference (CHI '94)*. New York: ACM Press.

Aufaure-Portier, M.-A. 1995. "Definition of a Visual Language for GIS." In *Cognitive Aspects of Human-Computer Interaction for Geographic Information Systems,* ed. T. Nyerges, D. Mark, R. Laurini, and M. J. Egenhofer. Dordrecht, The Netherlands: Kluwer Academic.

Cypher, Allen. 1993. *Watch What I Do: Programming by Demonstration.* Cambridge, Mass.: MIT Press.

Egenhofer, M. J. 1995. "User Interfaces." In *Cognitive Aspects of Human-Computer Interaction for Geographic Information Systems,* ed. T. Nyerges, D. Mark, R. Laurini, and M. J. Egenhofer. Dordrecht, The Netherlands: Kluwer Academic.

Garson, G. David, and Robert S. Biggs. 1992. *Analytic Mapping and Geographic Databases.* Sage University Paper series on Quantitative Applications in the Social Sciences, series no. 07–087. Newbury Park, Calif.: Sage.

Kuhn, Sarah, Charles Richardson, and Marian G. Williams. 1994. "Meeting of the Minds: The Challenges of Interdisciplinary and Inter-occupational Communication." In *Proceedings of the Participatory Design Conference (PDC '94),* ed. R. Trigg, S. I. Anderson, and E. A. Dykstra-Erickson. Palo Alto, CA: CPSR.

Lieberman, Henry. 1993. "A Teachable Graphical Editor." In *Watch What I Do: Programming by Demonstration.* Cambridge, Mass.: MIT Press.

Modugno, F. 1995. "Extending End-User Programming in a Visual Shell with Programming by Demonstration and Graphical Languages Techniques." Ph.D. diss., Carnegie Mellon University, Pittsburgh, Pa.

Richards J. R., and M. J. Egenhofer. 1995. "A Comparison of Two Direct-Manipulation GIS User Interfaces for Map Overlay." In *Geographical Systems,* 2, no. 4: 267–290.

Standing, Craig, and Geoffrey G. Roy. 1997. "Developing Macro Queries in Geographical Information Systems." In *Proceedings of the Fourth Conference of the International Society for Decision Support Systems (ISDSS '97).*

Traynor, Carol. 1998. "Programming by Demonstration for Geographic Information Systems." Ph.D. diss., University of Massachusetts, Lowell.

Traynor, Carol, and Marian G. Williams. 1995. "Why Are Geographic Information Systems Hard to Use?" In *Conference Companion of Human Factors in Computing Systems Conference (CHI '95).* Boston: Addison-Wesley.

———. 1997. "A Study of End-User Programming for Geographic Information Systems." In *Proceedings of Empirical Studies of Programmers: Seventh Workshop.* New York: ACM Press.

S
R
L

S
R
L