Shrager, J (2010) From Wizards to Trading Zones: Computers in Scientific Collaboration. In M. Gorman (Ed.) Trading Zones and Interactional Expertise. MIT Press.

6 From Wizards to Trading Zones: Crossing the Chasm of Computers in Scientific Collaboration

Jeff Shrager

The Chasm

Scientists are the prophets of the modern age. Whereas in the past prophets represented God to the masses, today scientists represent reality. As consumers, we are all quite used to computers as our interface to reality—your phone conversations are transmitted by computers, your bank account is virtual money, and nearly everything that you see or hear in the media is refracted through computers. The same is true, and even more so, in science: scientists don't fly around in space sketching pictures of the earth's weather, nor do they peer into atoms with microscopes. These are the jobs of instruments whose outputs are nearly always computer-mediated. But whereas consumers don't worry much about being held at arm's length from reality by computers (perhaps we should worry more), scientists worry about it quite a lot. Although we have nominated scientists as our interpreters of reality, it's actually not scientists but engineers, and more and more often software engineers, who have direct access to reality. Software engineers stand between scientists and the instruments that *really* read the photons. Thus scientists are left to interpret the "massaged reality" offered up by software engineers. But software engineers are essentially carpenters with no special training in science, and so the relationship is problematic.[1] No prophet would last long if, rather than claiming a direct line to God, he was just repeating what some carpenter told him!

I've spent nearly three decades working to bridge this chasm between scientists and reality, most recently between molecular biologists and computer scientists—to arrange it so that molecular biologists don't need to have reality mediated by software engineers, or at least so that they have a deep enough understanding of computing that they don't worry about this arrangement. Some of my early attempts were, in retrospect, misguided. However, I believe that they are worth describing for two reasons. First, there are still many similar projects being pursued today; perhaps the

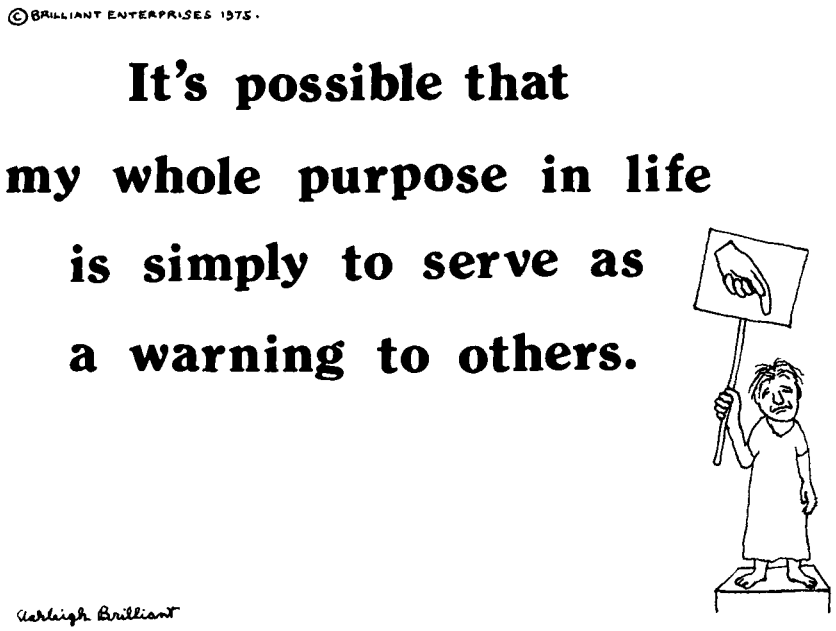

Figure 6.1

present analysis can help those researchers avoid plummeting into the chasm. More directly, the history of my (mostly misguided) ideas eventually did lead me to what I think is actually a very good solution to the problem of software engineers mediating science. Of course, I could be wrong again; this new idea could be yet another misguided one. Time will tell.

Before getting into my (mostly failed) attempts at bridge building, let's examine a few examples of the breadth and depth of the chasm that we need to cross.

The history of statistics in biology is quite interesting. Biology provided many fundamental examples in the development of modern statistics. But with the advent of molecular biology, biologists' ability to access the discrete components of life (DNA and such) has led them to feel that they could use simple tools like microscopes, gels, and imaging to access the machinery of life, making statistics irrelevant. As a result, molecular biologists over the past three decades were almost never trained in serious statistics. Unfortunately, and inevitably, as the complexity of our understanding of life mounts, sensitive instrumentation is taking over the field; one can no longer make much use of the view through the microscope lens to understand what's going on in a cell; instead one needs laser spectrometers, microarrays, quantitative PCR, pathway analysis, multiple testing corrections, dynamical systems models, Bayes nets, and other

such machinery that require quite subtle computation and statistics. Although biologists are becoming increasingly aware that they need to undertake statistical analyses, they have little idea how to do them, and so they rely upon statisticians (or, more often, students who could only play statisticians on TV), taking these folks' word for the results. This gives these (sometimes pretend) statisticians enormous say and sway over what gets done in a lab, because statistics demands a certain amount of data to get significant results. Unfortunately, the lab side of biology requires enormous time and labor, and getting even small amounts of data can cost a great deal. More than once, biologists have come to me with some data that they spent (not unusually) a year and (not unusually) hundreds of thousands of dollars obtaining, only to find out, after I do just a few minutes of math, that that they would need many times that amount of data to get any useful results, because of variability in the data. The upshot of this sort of interaction is almost never pretty; the biologists often end up putting the paper in journals with less and less prestige until they manage to get it through peer reviewers who also don't understand statistics. This problem is slowly being dealt with through new kinds of Bayesian statistical tools that incorporate biologists' subjective priors, and biologists are (re)learning statistics. The most important positive trend, which foreshadows where I'm going in this chapter, is that biologists are starting to design experiments in close collaboration with statisticians and computer scientists. Put another way, biologists are becoming willing to include statisticians and computer scientists as first-class members of their research collaborations, as opposed to treating them like hired help and expecting them, as an afterthought, to "find the results" in the data.

Another disconnect between biologists and computer scientists, implicit in the above example, has to do with the speed with which results can be obtained. I described the way that biologists will regularly spend years and hundreds of thousands of dollars producing data which the statisticians proceed to demolish in minutes (or at most, a few days) sitting at a computer, essentially for free. The opposite is also true, and in some ways more problematic: computer scientists can spend the same few days at their computer (essentially for free) and obtain *new* results by reanalyzing the biologists' own data, after (or sometimes even before) it is published! Imagine that you are a biologist who has sweated blood, perhaps spending your whole dissertation on producing a small pile of data, which your lab has poured perhaps a million dollars into. You publish your interpretation of these data, and (as is required) make the data public.[2] A week later you find that some computer scientist has downloaded your data, and has not only demonstrated that your interpretation was *wrong*, but has produced a new, *more correct* conclusion based upon computational analyses of *your own* data.

If the computer scientist is feeling particularly magnanimous, he'll ask you to coauthor a paper, but there's no ethical requirement for him to do that; your data are public—all he ethically need do is to cite you. This may feel to you like intellectual theft, but it's no more intellectual theft than it is to use $E = mc^2$ in a calculation, referencing Einstein (which no one does, but it's obvious where that came from). I have, more than once, found myself on either the short or the long end of this stick, or worse, straddling both sides trying to keep scientists from having one another hauled up in front of ethics boards for just this sort of thing. Not only does it happen all the time (not so often involving ethics boards, thank goodness), but it will be happening more often as more and more data become available and as computers become more and more powerful, able to mine larger and larger quantities of data. Unless biologists get to the point where they can mine their own and others' data in this way, or else form early and strong collaborations with statisticians and computer scientists, the biologists are the ones who will end up as "mere" worker bees, gathering data, while the computer scientists will be the producers of theoretically important results.

Into the Chasm

So this is the chasm that we want to find our way across: biologists need to be able to deeply understand computation and, to a large extent, to do it themselves (or so it seemed to me). Before diving into a survey of my failed attempts to cross this chasm, I'd like to clarify that I will mostly discuss molecular biology here because that's the field through which I have come to this problem. The problem is not the same for all fields. Many scientists learn to program, and do so all the time. For example, the whole concept of computer simulation was developed in large part by and for physicists, and many physicists are highly competent programmers. Unfortunately, biologists don't have such a tradition, and, as we shall see, computation hit biology hard at the wrong time, and has nearly overwhelmed the field. Although biocomputing is ahead of other fields in knowledge representation (what is sometimes called "semantics" or "ontology"), biological knowledge bases are rather haphazard affairs, connected to one another in an ad hoc manner. In order to do computing in this space, one needs to have a strong stomach for intense kludgery.[3]

But the same biologists who kludge things together in their labs all the time don't seem to have the stomach for it in computers. Why do physicists? Perhaps it's because mathematics—the knowledge representation of physics—isn't as bad a kludge as is most biological knowledge representation? Maybe it's that computers, not being alive, simply don't interest biologists? Maybe it's because biologists have traditionally been

able to get along without them? Another reason comes from an interesting historical accident: biologists began finding it impossible to live without computation around the early 1990s. This was coincidentally about the same time that computers had begun to *appear* simple, by virtue of the Web's emphasis on simple pushbutton functionality. As a result of this accident of historical timing, biologists came to see software in the model of a vending machine: you put in a quarter (some data), press a button, and out pop results. I've often encountered the attitude among biologists that "There must be a Web page somewhere that does this. … I'll just surf around until I find it, and then put my data in and press the 'compute' button, and results will pop out." This is generally true for consumer needs—if you want to buy a book or schedule a flight—but not so much if you're doing science. And even if you do get lucky and find that someone has thought about the same problem before, and has left you a way to do it, it's usually nontrivial to get it working, and in any case you'd better understand what it's doing or you're likely to misinterpret the results. These are the capabilities that are expressly missing in biology, and these are what lead to the chasm that I'm trying to cross.

Wizard: Computers as Collaborators

The buttress of my attempts to build bridges is my master's degree project, advised by Tim Finin at the University of Pennsylvania. As an undergraduate computer scientist, I developed help systems for the Penn engineering community's large time-sharing systems. In my master's work, Tim Finin and I went a step farther, developing an AI-based assistant for computer users, called Wizard (Shrager and Finin 1982). Although I doubt that I realized it at the time, our idea had significant subtlety, and it directly underlies the rest of my work. Rather than just being another help system, Wizard fundamentally changed the role that help systems played with respect to their users: instead of being mere tools, the help system could become a collaborator. What is the difference between a tool and a collaborator? Tools don't understand the goals of the project and don't take initiative to reach those goals. Collaborators, on the other hand, do understand and share the goals of the project, and by that virtue can take initiative to reach them. Humans wield tools, but do not wield their collaborators! Instead, we work with them; the project becomes a shared one, with shared goals and negotiation about who's going to do what. You don't negotiate with your tools. With this in mind, Tim and I focused on goals, and specifically on *goal recognition*—that is, giving the computer enough of an understanding of the user's goals that it could recognize them from the user's actions, and take initiative to help. Out of this arose Wizard, which

was probably the world's first computer program that was able to take the initiative in offering assistance to a user (Shrager and Finin 1982).[4]

Now, as a young computer scientist working in AI, I believed that psychologists knew something that I didn't—specifically, that they understood things like goals and how goal recognition could work. I had taken some psychology courses as an undergraduate and decided, somewhat insanely, that I didn't just want to study psychology, I wanted to actually *be* a psychologist. (Watch as this irrational need to go native reappears as a theme in my life!) Not wanting to leave computing too far behind, I undertook doctoral work at Carnegie Mellon University (CMU), where there is a strong computational cognitive psychology program.

At CMU I pursued several apparently unrelated topics: computational modeling of cognitive development (with David Klahr and Bob Siegler), computational modeling of adult cognition (with John Anderson), and computational modeling of scientific discovery (with Pat Langley and Herb Simon). I shall leave for another time the interesting analysis of how computer wizards, cognitive development, and scientific discovery relate. Suffice it to say that goal recognition is an observational discovery task (Cohen, Perrault, and Allen 1982), analogous to scientific discovery, and that one important way to obtain an understanding of how goals work in a given culture is to have grown up (developed) in that culture (Collins, Clark, and Shrager 2008).

My first real foray in bridging the chasm between biologists and computation lay at the confluence of the concepts of "computers as collaborators" and "computers as scientists." Whereas it seemed to me impossible at that time (and still does today) to build an artificial scientist, I thought that perhaps we could build computational tools that could participate with scientists as true collaborators, at least at the level of wizards like Wizard. Such tools could have a sense of the goals and, even more importantly, could provide explanations. Explanation was a concept that we had only a glimmer of in Wizard, but it is obviously an important part of science (e.g., Hempel 1965). In the current context of collaboration: you don't explain to your power tools what you're doing with them, and you don't expect explanations from them when they make mistakes, whereas you do explain things to, and expect explanations from, your collaborators.

I thought that by endowing the program with an ability to understand scientific goals (even in a simple sense), and to understand and provide scientific explanations (even in a simple sense), scientists would be that much closer to not having to depend upon software engineers (even in a simple sense), because they would be able to collaborate with the program (even in a simple sense). To be somewhat less fantastical,

one can think of explanations on the input side as a sort of programming language, and on the output side as a sort of debugging log. These would be rendered into "plain" language—or, in this case, *scientific* language, that is language that scientists can understand without having to be software engineers. If the scientists could interact with the program's logs in the scientists' own terms, and perhaps even write programs in the same terms, there would (I believed) be less need for scientists to engage software engineers to speak in computational tongues.

Afferent, a Computational Scientific Collaborator: A First Run at the Chasm

At this time—around 1996—another AI engineer, David Chapman, asked me to help him productize a robotic drug-discovery tool, called Afferent, that he had prototyped while working at a biotech. Afferent and its relationship to drug discovery is described elsewhere (Shrager 2001). Here I highlight the ways in which Afferent could be thought of as a collaborator, in that it could be said to understand the medicinal chemists' goals and to offer explanations.

One of the most important approaches to discovering new drug leads is called "high throughput combinatorial chemistry" (combi-chem). Combi-chem works by running numerous chemical reactions, thereby creating numerous molecules that could become drugs. Now, chemists realize that running numerous reactions is a robotic problem (at least for large values of "numerous"), but chemists are too busy being chemists to learn to be robot programmers as well. Chapman was a robot programmer, and I worked in AI, human-machine interaction, and computational models of science. We, along with several other engineers, produced Afferent, a program that would allow medicinal chemists to conveniently get a robot to do combi-chem.

Afferent approached this task through a combination of subtle technologies woven together through a user interface that walked a difficult line between power tool and collaborator. Afferent's interface enabled the chemists to "talk chemistry" to the robot. The chemists entered chemical reactions and protocols into Afferent in more or less the same format that the chemists were used to—the sort of thing that you'd see in a chemistry textbook (figure 6.2).

Given reactions and execution protocols for the chemistry, Afferent would operate the robot to run the reactions. At the same time, Afferent would *simulate* the chemistry. There are two reasons for bothering with this very complex simulation step. First, the chemists want to be able to tell what products to expect from the reactions. Moreover, chemistry is finicky; many (sometimes most) of the desired reactions won't work as predicted, and you want to be able to tell which ones failed. This is done by various

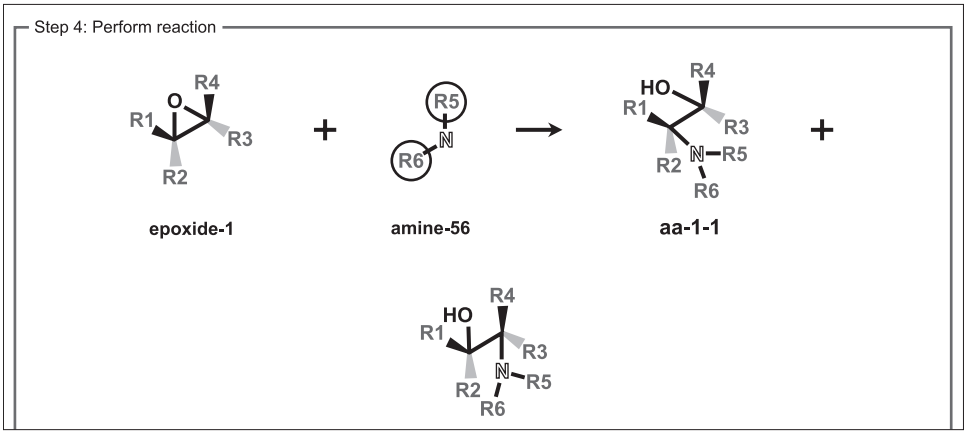

Figure 6.2
A chemical reaction as entered into Afferent. Note its similarity to the notation used by chemists, such as that found in a textbook.

analytical measures that compute the amount of product in each resulting reaction vessel. Failed reactions can be traced through the reaction simulations to locate offending reaction pathways. This sort of analysis and the corresponding traces constitute one important type of explanation in chemistry. (See Shrager 2001 for more details.)

Whereas Afferent could be said to "read" chemistry, "think" (simulate) chemistry, and "speak" chemistry, it wasn't anything like a contributory expert in chemistry. It contributed nothing in terms of guiding heuristic search through reaction space.[5] Regardless, Afferent went far beyond a simple combi-chem tool, and could fairly be said to be, in some sense, a collaborator in the drug discovery process.

Successful as Afferent was in combi-chem, the goal of a true computational scientific collaborator remains elusive. No computer program can (yet) rise to the level of subtlety required of a true collaborator. Collins and Kusch (1998) argue on theoretical grounds that computers will never be able to rise to this level (but see Collins, Clark, and Shrager 2008). There is also an issue of "explanatory trust" that cannot be bridged through "mere" AI: although human software engineers might misunderstand specs, make mistakes, and miss milestones, at least you can have a conversation with them. They can explain what they are doing or why they are not succeeding, and one can bring managerial or social pressure to bear on them. It's impossible to bring social pressure on a computer program. Nonetheless, Afferent was perfectly good as a power tool, and sold well. In 2000 the company was sold to MDL, Inc., and I made enough money to fund my next attempt at crossing the chasm.

Becoming a Molecular Biologist: Trying to Leap the Chasm

Software engineering and psychology stand apart from most other technical fields in that they are, in an interesting way, "domain independent." When you write a program, the program has to be computing *about* something else. It is similar in psychology: when you study someone's thinking, the person you are studying has to be thinking *about* something else. This makes computation and psychology quite flexible, in that, as a practitioner of these fields, one can work on whatever in the world one likes. Suppose, for example, I wanted to live on top of a mountain and work in astronomy; all I'd have to do, as a software engineer, is hire on to write programs for astronomers. Similarly, as a psychologist of science, I could decide that how astronomers think is somehow particularly interesting, and that I want to study them. Many other fields that are together called the "social sciences" share this property, but not so most technical fields, which are usually about just what they are about, not what some other field is about. So in being a computational psychologist of science, I'm able to study whatever I like in the scientific world. And because of the critical role that software engineers play in science, I could potentially make significant (if silent) contributions to it.

Now, there is a strong tendency for practitioners of metafields to get pulled into the target field—to begin seeing the world as the community under analysis sees it. After all, that community must be pretty interesting to you; otherwise, why would you be studying it? This tendency has long been recognized in anthropology, where they call it "going native." To "go native" as a scientist means to begin finding the domain of the science more interesting than the metascience, and to adopt the scientists' explanations for their own actions. In the limit, going native in science might mean, in Collins and Evans's (2007) terms, becoming a "contributory expert"—becoming a *real* astronomer, rather than merely working as a programmer for astronomers, or as a psychologist studying astronomers. I felt this tendency quite strongly with biology, and for a time even went native myself.

After Afferent was sold, I made up my mind to leave computers behind and do "real" science. To this end, I stopped programming altogether and set out to become a "real" molecular biologist. The fact that computers had become essential at that time, yet also essentially mysterious to biologists, offered me a way into the lab of my choice. I made a deal with the biologists: you teach me to be a molecular biologist, and I'll do your computing for you. Who could refuse?

In addition to satisfying my romantic need to go native in biology, becoming a molecular biologist myself was, in my fantastical imagination, another route for

scientists to understand computers. In this fantasy, no one would begin in biology per se; instead, one would always undergo a dual training: first you'd become a computer scientist, and only then would you study biology. Under this plan you become a contributory expert in biology, but with a strong background in computing. These disciplines could be undertaken nearly simultaneously; and indeed they now are—for example, in specialized biocomputing degree programs. This is not a brand-new idea: Seymour Papert (1993) had the idea that everyone should learn to program in elementary school, and he invented a simplified programming language, called Logo, that supposedly made programming easy and motivating for kids. But Papert theorized that programming somehow prepared one better for general thinking. My idea was much more specific and practical: all scientists needed to be programmers in order to cross the chasm between themselves and reality—ground now held fast by software engineers. In my model everyone is first a software engineer, only later applying computers in biology. You end up as a contributory expert in biology, but you arrive with a strong background in computing.

Of course, there's nothing particularly new about the idea of learning the important background (computing or math or whatever) and then taking on a specific domain. It was just apparent to the biologists, and to me, that biology was becoming a much more computational and mathematical field that current biologists were unprepared for—thus the chasm. Because working biologists have no time to retrain in computing, and because computation has the flexibility of needing a domain, computer scientists could be turned into biologists, thus bridging the chasm.

The fastest way to perform this experiment was, of course, to inject the drug myself (figure 6.3). This was, for me, in large part, a successful experiment: I coauthored numerous almost purely biological papers (e.g., Labiosa et al. 2006), and was co-principal investigator of several almost purely biological grants.

Unfortunately, once in the lab, no matter how hard I ran from computers, I was unable to hide from them, and programming came to occupy more and more of my time. Not only did I have my own biological computing to do, but as none of the other "real" biologists knew much about computers, I was servicing their ever-increasing computing needs as well. (Of course, this was the explicit deal I had made with them in exchange for their teaching me molecular biology.) I was also, as it turned out, pretty poor as a bench biologist. (I can't cook very well either and don't enjoy it, but unfortunately, cooking is closest to the operational skills utilized in a lab.) As a result, I was much more valuable to the lab as a computer scientist than as a molecular biologist. So, after a couple of years of actual pipetting, as shown in figure 6.3, I found myself working almost all the time at a computer rather than at a bench.

Figure 6.3
The author engaged in some mysterious bench molecular biology. Note the tools of the trade: multiple timers (hanging), pipettes, various bottles, safety glasses, notebooks. (See Shrager 2004 for more detail.)

The inverse was true of the other biologists in the lab; they were about as good at computing as I was at pipetting, so I ended up with a long line of biologists at my back in need of computational help. My remaining strong romantic desire to be a real molecular biologist, rather than a software engineer working *for* biologists, led directly to the my next approach to the chasm's edge.

BioBike: You Can Drag Biologists to Computers, but You Can't Make Them Hack

There was so much computing to do, even in my small lab of about fifteen people, that it was beyond my abilities to keep up with. And I still harbored the romantic fantasy of doing biology, not programming for biologists. In fact it was this tension—my trying to pull toward being a molecular biologist, and the molecular biologists trying to pull me into doing their computing for them—that led me to realize that the chasm existed in the first place, and also to the next idea of a way across it. If

computers can't themselves be scientific collaborators, and programmers can't effectively become biologists, maybe it would work the other way around: enable biologists to do their own computing! Again, there is precedent for this in physics and other fields. The language of physics is math, and programming math is what programming was for in the first place; but what is the language of biology?

As I mentioned above, biological representation and computing were (and remain) quite a mess. You could put a full-time software engineer in every lab to help biologists apply computers to their specific work, and still not be able to satisfy this ever-increasing need. If biologists were to be enabled to do their own computing, one would need to reduce many time costs: the cost of gathering the relevant data and knowledge and wiring it all together, the cost both of learning to program and of learning the numerous representational and programming languages needed to get even simple things done in biocomputing, among other costs.

Onerous as these obstacles may seem, my colleagues and I elegantly addressed all these needs, to some extent, through a novel Web-based technology called a KnowOS—a Knowledge Operating System (Travers, Massar, and Shrager 2005), on which we developed a specialized Web-based biological programming environment called BioBike—a sort of Logo for biologists (figures 6.4 and 6.5; Massar et al. 2005).

BioBike offered unprecedented integration capabilities and made biocomputing as easy as we were able to imagine. I taught a course at Stanford in using BioBike, and my colleague Jeff Elhai, a biologist at Virginia Commonwealth University, taught it there. Elhai continues to work to make BioBike even more biologist-friendly (figure 6.5). He even sent a postdoctoral biologist, Arnaud Taton, on a world tour to train biologists to use BioBike *in situ,* helping them conduct their own research in their own labs. Still, to date, significant numbers of biologists have not adopted BioBike or any other programming paradigm.

Let me give you just one example of how difficult this chasm is to cross, and to understand. Once BioBike was up and running, I began to introduce it to the long line of biologists wanting me to do computing for them. I started by giving lectures in order to acquaint them with the general concept of biocomputing, and specifically how to use BioBike to do real work. These were well attended by enthusiastic biologists. When I worked on a particular project for a biologist, I used BioBike to build a small quick-and-dirty application for them; all that they usually had to do was to make small modifications to the code to get their work done. Figure 6.4 depicts such a program, which does molecular functional phylogeny. This was as close to the scientific vending machine as one could imagine; it was even better, in fact, because the functionalities were tuned to the particular task at hand. I even

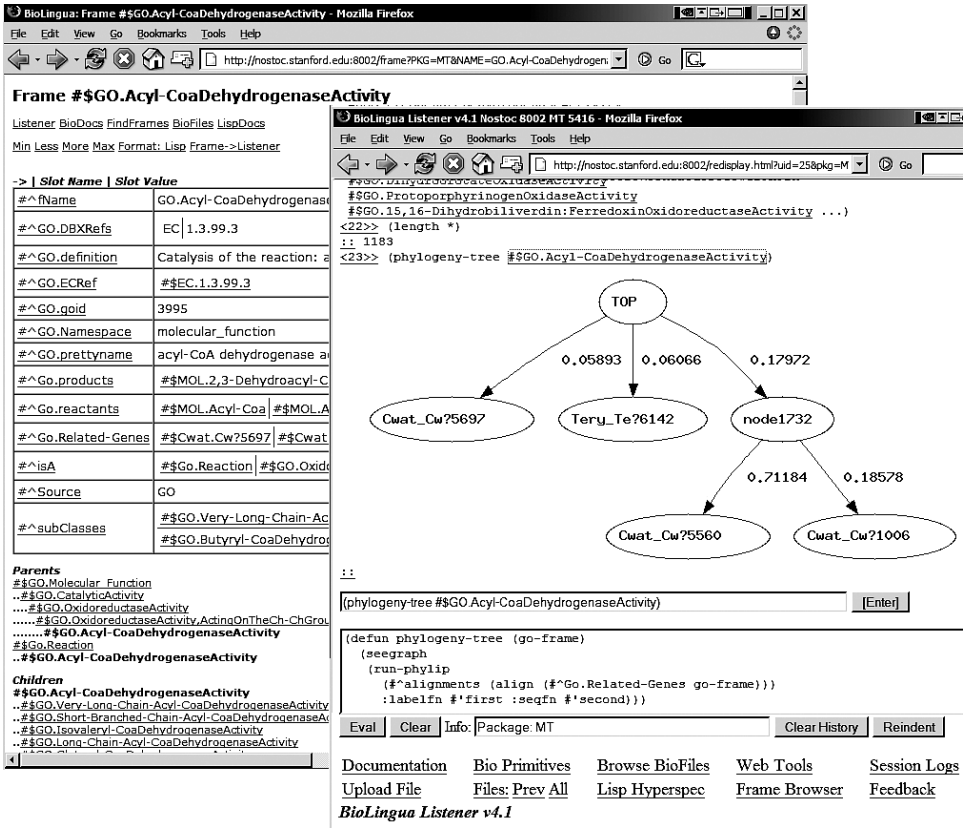

Figure 6.4

BioBike screens depicting (left, partly occluded) the built-in knowledge framework and (right, overlapping) the command interface. The program depicted (right bottom, beginning "defun ...") is all that is needed to compute molecular phylogeny among organisms based upon functional orthology. (See Massar et al. 2005 and Travers, Massar, and Shrager 2005 for more detail.)

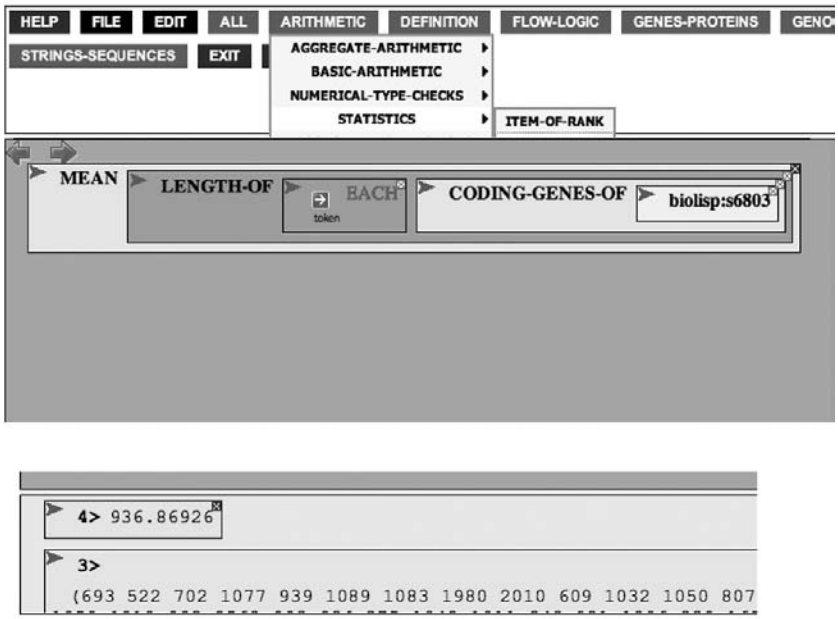

Figure 6.5
The BioBike/BBL visual programming interface. The computation depicted here (green frame) was created interactively through the Web. It computes the mean length of the coding genes of the cyanobacterium *Synechocystis* sp. PCC 6803. (Results are in the blue frame.) Contrast this with the messy complexity of figure 6.4.

wrote detailed instructions, in lablike protocol format, explaining how to use these programs.

Nevertheless, I discovered you can drag biologists to computers, but you can't make them program! I once created a three-line BioBike program for a biologist in our lab. If he had used my code, his computation would have taken about ten minutes. Several weeks later he came back to me reporting that he'd finally found the time to accomplish the computation. *The time?* Yes, he'd spent all weekend on it! It turns out that he had used BioBike only to download the data into a spreadsheet. (He had actually given me the data in a spreadsheet in the first place, and I had simply uploaded it into BioBike, so why he had to re-download it into a spreadsheet is entirely beyond me.) Next he had arduously applied spreadsheet functions (sorting, deleting, and simple math) to get the desired result. My program (which I'd given him detailed instructions on how to use) would have done all of that work for him automatically, taking ten minutes to get the same result that he had probably spent twenty hours on.

This was not an isolated case; this sort of thing happened to me over and over again. A biologist would come to me to ask for help; I would try to explain BioBike and create a very simple example program that she or he could modify and use almost immediately. It would have taken perhaps an hour of preliminary attention to learn how to use BioBike, and then perhaps another hour (and these are high estimates!) to get the work done. Not once that I can recall in my five-plus years of trying to get this to work did a biologist ever take the short time it would have taken to save himself or herself hours, or days. They would either walk away saying that they would do it, and then wouldn't; or would walk away confused; or would just walk away.[6] Most commonly they would simply keep annoying me about it until I did the work myself. In this last case BioBike saved *me* enormous time; it was, and remains, an excellent biocomputing platform—saving biocomputing engineers from the most painful and repetitive parts of their work—and in this sense it was (and continues to be) successful. But as far as biologists doing their own computing, so far BioBike has, in my experience, failed to live up to its promise.

Although I have given up on that goal, Jeff Elhai and his coworkers have persisted, taking BioBike even farther toward the Logo vision of biocomputing by biologists. They have created a simplified push-button, domain-specific Web tool for genomic computing, including a special-purpose, simplified programming language, called BBL (Massar et al. 2005), and they have developed an impressive visual programming interface for BBL, depicted in figure 6.5.

BioBike as Trading Zone: Collaborating *without* Crossing the Chasm

I failed in my effort to create a computational collaborator. I failed in my effort to exemplify the transition between computer scientist and biologist. I failed in my effort to train biologists to program, even after providing them with a platform that made programming simple. But from all these failures has arisen a potentially important, interesting, and entirely unexpected idea.

Although biologists did not themselves end up programming in BioBike, a community arose which does, in the end, accomplish the desired goals. Importantly, this community does *not* consist of biologists who are autonomously programming their own solutions. Rather, because BioBike is a Web-based platform, the biologists are able to interact in a very tight collaboration with programmers *directly through* BioBike, even if they are all remote from one another. In some cases this collaboration has been effectively elbow to elbow: biologists and programmers sitting in different locations—often thousands of miles apart—working on the same problem through shared BioBike screens. In this BioBike "trading zone" (borrowing a term first coined in a

scientific context by Peter Galison [1997]), real problems are really getting solved, *not* by AI-based computational collaborators, *not* by the programmers becoming biologists, and *not* by the biologists becoming programmers, but by the enabling artifact of BioBike, a biocomputing-specific, knowledge-rich, Web-based computational trading zone (Shrager 2007).

The specific role played by BioBike in this trading zone is complex and interesting. On the one hand, it can be thought of as a sort of platform for the development of what Star and Griesemer (1989) have called "boundary objects," forming a shared focus for the communities of practice that meet in the trading zone. But a program is more than an object; it carries in and of itself a semantics, and so BioBike is more like a dynamic book—a dictionary, perhaps, of the creole that forms in the trading zone—than like a static object. Discourse in the trading zone runs through BioBike rather than, or in addition to, running around (about) it. There are many features that enable BioBike to serve this role as locus of a scientific trading zone, and I have discussed these in some detail elsewhere (Shrager 2007). The key point is that I neither foresaw this result nor made an explicit effort to reach it. Instead, the BioBike trading zone emerged from the activities of the community of BioBike users and developers.

So I am left, now, with this last way across the chasm between biologists and computing: scientists and engineers interacting as peers through a shared science-specific collaborative computing platform. With twenty-twenty hindsight, it is clear to me that this was the correct solution all along; the biologists needed to think of the computer scientists as collaborators, not as power tools (or the operators of power tools). In a strong sense, BioBike is not what is enabling this, so much as the historical fact of the computer scientists' not being under the thumb of the biologists. As long as I was in the lab, working *for* the biologists, I was never going to be more than a carpenter to them. But by enabling scientists—whether biological scientists or computational scientists—who are at a similar level of contributory expertise in their own fields to work *with* one another, not *for* one another, to cooperate—*no, to collaborate!*—the needs of shared goals, explanation, and getting things done are all met.

Or perhaps not. Time will tell.

Appendix: Popular Press regarding Shrager and Finin 1982

Each of the following was the very first sentence of popular news reports that arose from Shrager and Finin 1982. The entire articles can be read at http://nostoc.stanford.edu/jeff/personal/vita/pubs/1982WizardPress.pdf:

H. W. Pierce, "Computer Advises without Being Asked," *Pittsburgh Post-Gazette*, August 16, 1982, 6: "Most people don't like back-seat drivers. But what if the back-seat driver were a computer?"

T. Hritz, "Busybody Computer," *Pittsburgh Post-Gazette*, August 19, 1982, 4: "When I read earlier this week about the National Conference on Artificial Intelligence which is meeting here, and about the computer out at Carnegie Mellon University that gives advice without being asked for it, I found it hard to believe. I always thought that you couldn't get anything out of a computer unless you put that thing into it."

T. Henkel, "Digital Big Brother: Sassy Program Keeps Programmers in Line," *Computerworld*, September 6, 1982, 16: "Thanks to the marvels of technology, computers can now be taught to torment their users."

Notes

Mike Gorman edited in detail several drafts of this chapter, helping to hone the concepts and language, but those are the least of his contributions! This paper arose from a talk presented at the first symposium on Trading Zones and Interactional Expertise, hosted by Arizona State University. I came to that meeting with a completely different—and much more shallow and boring—talk in hand. After sitting through the first day of talks, especially those of Mike, Peter Galison, and Harry Collins, I completely rewrote my talk, calling it (originally) "My History of Dumb Ideas." After hearing the talk, Mike hounded me to put it on paper for the present volume, and helped me to fine-tune my thinking and to connect the concepts better to the themes of the symposium. As a result, Mike has fundamentally helped me to understand my own intellectual history.

1. This issue lies at the heart of Galison's 1997 *Image and Logic*: "To get at the shifting meaning of 'the experiment and the experimenter,' we will need to empathize with the anxiety that physicists have often felt at their loss of control over the rapidly expanding laboratory, at the complex confrontation of physicist with engineer" (p. 5).

2. Although it must be made public (especially when the project is publicly funded), data is often not actually made easily available. In the best case, it is just downloadable, but I have many times found that I have to ask multiple times for the data, and then get it in dribs and drabs, or in completely uninterruptible formats.

3. A "kludge" (the *u* is pronounced as *oo* in "boot") is a term of art in computer science, referring to a usually messy, ad hoc, and poorly constructed solution to a complex problem. A kludge gets the job done, but because it is not a good solution, it is usually hard to understand, modify, and support.

4. The reader will be most familiar with the wizard concept in the guise of the much-maligned Microsoft (MS) Paperclip, which appeared about a decade after Finin and I published our work. Although the MS Paperclip did essentially the same job as Wizard, what people didn't like about

the Paperclip seems to have been its arrogance and annoying animatronics. The Paperclip was retired around the turn of the century. It was pretty clear, even to nontechnical observers, that the concept of a computer that could take its own initiative was going to be problematic. In the appendix, I reproduce some responses to our work that appeared in the popular press at the time of its publication.

5. We did have in mind trying to do this, but never got around to it. And, in retrospect, we probably wouldn't have got very far.

6. Jack Carroll and Mary Beth Rosson refer to this as the "production paradox" in their eloquent and classic paper "The Paradox of the Active User" (Carroll and Rosson 1987).

References

Carroll, J. M., and M. B. Rosson. 1987. Paradox of the Active User. In *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, ed. J. M. Carroll. Cambridge, MA: MIT Press.

Cohen, P. R., C. R. Perrault, and J. F. Allen. 1982. Beyond Question Answering. In *Strategies for Natural Language Processing*, ed. W. Lehnert and M. Ringle, 245–274. Hillsdale, NJ: Lawrence Erlbaum Associates.

Collins, H. M., A. Clark, and J. Shrager. 2008. Keeping the Collectivity in Mind? *Phenomenology and the Cognitive Sciences* 7:353–374.

Collins, H. M., and R. Evans. 2007. *Rethinking Expertise*. Chicago: University of Chicago Press.

Collins, H. M., and M. Kusch. 1998. *The Shape of Actions: What Humans and Machines Can Do*. Cambridge, MA: MIT Press.

Galison, P. 1997. *Image and Logic*. Chicago: University of Chicago Press.

Hempel, C. G. 1965. *Aspects of Scientific Explanation and Other Essays in the Philosophy of Science*. New York: Free Press.

Labiosa, R., K. Arrigo, A. Grossman, T. E. Reddy, and J. Shrager. 2006. Examination of Diel Changes in Global Transcript Accumulation in Synechocystis. *Journal of Phycology* 42 (3):622–636.

Massar, J. P., M. Travers, J. Elhai, and J. Shrager. 2005. BioLingua: A Programmable Knowledge Environment for Biologists. *Bioinformatics* (Oxford) 21 (2):199–207.

Papert, S. 1993. *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books.

Shrager, J. 2001. High Throughput Discovery: Search and Interpretation on the Path to New Drugs. In *Design for Science*, ed. K. Crowley et al., 325–348. Hillsdale, NJ: Lawrence Erlbaum.

Shrager, J. 2004. On Being and Becoming a Molecular Biologist: Notes from the Diary of an Insane Cell Mechanic. In *New Directions in the Study of Science and Technology*, ed. M. Gorman et al. Mahwah, NJ: Lawrence Erlbaum.

Shrager, J. 2007. The Evolution of BioBike: Community Adaptation of a Biocomputing Platform. *Studies in History and Philosophy of Science* 38:642–656.

Shrager, J., and T. Finin. 1982. An Expert System that Volunteers Advice. In *Proceedings of the Annual Conference of the American Association for Artificial Intelligence*, 339–340. Menlo Park, CA: AAAI Press.

Star, S. L., and J. R. Griesemer. 1989. Institutional Ecology, Translations and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907–39. *Social Studies of Science* 19:387–420.

Travers, M., J. P. Massar, and J. Shrager. 2005. The (Re)Birth of the Knowledge Operating System. International Lisp Conference, Stanford University, June.