CHAPTER 13

Machines and the Web

In communicating between people using the Web, computers and networks have as their job to enable the information space, and otherwise get out of the way. But doesn't it make sense to also bring computers more into the action, to put their analytical power to work making sense of the vast content and human discourse on the Web? In part two of the dream, that is just what they do.

The first step is putting data on the Web in a form that machines can naturally understand, or converting it to that form. This creates what I call a *Semantic Web*—a web of data that can be processed directly or indirectly by machines.

Consider the limited amount of help we have received so far on the Web from machines. Search engines have proven remarkably useful in combing large indexes very rapidly and finding obscure documents. But they have proven remarkably useless, too, in that they have no way to evaluate document quality. They

return a lot of junk. The problem is that search engines generally just look at occurrences of words in documents—something that is a hint at but tells very little about what the document really is or says.

A bit more sophisticated are automated brokerage services, which began to emerge in 1998. These are Web sites that try to match buyers and sellers. From the buyer's perspective, such a service can look like a metashop—a store of stores. One metashop to emerge is webmarket.com: Give it a book title, and it will search all the online bookstores it knows, check the prices, and present a competitive list. To actually search the bookstores' catalogues, it has to pretend to be a browsing buyer, run their search engines, then extract the resulting data about product, price, and delivery. It can then prepare a table comparing each deal.

The trick of getting a computer to extract information from an online catalogue is just that: a trick. It is known as *screen scraping*—trying to salvage something usable from information that is now in a form suitable only for humans. It is tenuous because the catalogue could change format overnight—for example, putting the ISBN number where the price used to be—and the automatic broker would be confused.

As people learn to use the Web, they analyze it in many ways. Ego surfing—looking for occurrences of one's own name—is a simple example. It may seem narcissistic, but it is a reasonable quest, because we have a certain responsibility to figure out where we fit into the world. Online research is a more serious example: One tries to find not only the answer to a question, but also what structures might be out there in the information.

Take a writer who wants to influence decision makers in Pakistan and India who are toying with the possible use of nuclear weapons. He wants to give them a deep awareness of the horrible aftermath of the atomic bombing of Nagasaki. He needs to know the forums in which these people operate, what they read. He needs sources of information on nuclear weapons. He needs the

current connections between the people, forums, and information sources. The structures and interrelations are important.

The same sort of Web analysis could uncover new markets. It could help a project team leader evaluate the workings of her team by mapping all the dependencies and relationships among people, meeting minutes, research, and other materials involving the group, which together define how the project is going. A CEO would like to be able to analyze his company's entire operation. Imagine receiving a report along the lines of: "The company looks fine, except for a couple of things. You've got a parts division in Omaha that has exactly the same structure and business patterns as a company in Detroit that just folded: You might want to look at that. There's a product you make that is completely documented but completely unused. And there seem to be a few employees who are doing nothing that contributes to the company at all."

None of this analysis can be automated today, partly because the form of intelligence that can draw such conclusions is difficult enough to find in people, yet alone in a computer program. But a simpler reason is that very little of the information on the Web is in a form that a machine can understand. The Semantic Web tackles this simpler problem—perhaps in the end as a foundation for tackling the greater problem.

Today, when one person posts a notice on a Web site to sell, say, a yellow car, it is almost impossible for another person to find it. Searching for a "yellow car for sale in Massachusetts" results in a useless huge list of pages that happen to contain those words, when in fact the page I would want may be about a "Honda, good runner, any good offer" with a Boston phone number. The search engine doesn't understand the page, because it is written for a human reader with a knowledge of English and a lot of common sense.

This changes when the seller uses a program (or Web site) that allows him to fill out a form about an object for sale. This

could result in a Web page, in a machine-readable format, that maintains the significance of the document and its various parts. If all notifications of cars for sale were posted using the same form, then it would be easy for search engines to find, exclusively, yellow cars in Massachusetts. This is the simplest first step toward machine-understandable data.

The next step is a search engine that can apply logic to deduce whether each of the many responses it gets to an initial search is useful. This would allow us to ask general questions of our computerized agents, such as "Did any baseball teams play yesterday in a place where the temperature was 22°C?" A program—call it a logic engine—would apply mathematical reasoning to each item found. The search engine might find six thousand facts involving baseball teams, and two million data items about temperatures and cities. The logic engine would analyze which bits of data refer to where a baseball team is, ascertain what the temperature was in certain towns, filter both sets of data, strip out all the junk, and respond: "The Red Sox played in Boston yesterday and the temperature was 22°C. Also, the Sharks played in Tokyo, where it was 22°C." A simple search would have returned an endless list of possible answers that the human would have to wade through. By adding logic, we get back a correct answer.

While Web pages are not generally written for machines, there is a vast amount of data in them, such as stock quotes and many parts of online catalogues, with well-defined semantics. I take as evidence of the desperate need for the Semantic Web the many recent screen-scraping products, such as those used by the brokers, to retrieve the normal Web pages and extract the original data. What a waste: Clearly there is a need to be able to go publish and read data directly.

Most databases in daily use are *relational databases*—databases with columns of information that relate to each other, such as the temperature, barometric pressure, and location entries in a

weather database. The relationships between the columns are the *semantics*—the meaning—of the data. These data are ripe for publication as a *semantic* Web page. For this to happen, we need a common language that allows computers to represent and share data, just as HTML allows computers to represent and share hypertext. The consortium is developing such a language, the Resource Description Framework (RDF), which, not surprisingly, is based on XML. In fact it is just XML with some tips about which bits are data and how to find the meaning of the data. RDF can be used in files on and off the Web. It can also be embedded in regular HTML Web pages. The RDF specification is relatively basic, and is already a W3C Recommendation. What we need now is a practical plan for deploying it.

The first form of semantic data on the Web was metadata: information about information. (There happens to be a company called Metadata, but I use the term here as a generic noun, as it has been used for many years.) Metadata consist of a set of properties of a document. By definition, metadata are data, as well as data about data. They describe catalogue information about who wrote Web pages and what they are about; information about how Web pages fit together and relate to each other as versions, translations, and reformattings; and social information such as distribution rights and privacy codes.

Most Web pages themselves carry a few bits of metadata. HTML pages have a hidden space in the document where certain items can be encoded, such as the page's title, its author, what software was used to create it, when it was created, and when it was last modified. Often this is also put in human-oriented form, in plain English, at the bottom of a Web page in small type. Legal information, such as the copyright owner and privacy practice of the publisher, might be there, too. Metadata already out there also include catalogue information, such as keywords and classification numbers, and all the things libraries tend to put on library cards. There is endorsement information, such as PICS labels.

And there is structural information about which Web pages on a site act as cover page, table of contents, and index. There is no end to metadata, and a common RDF language for metadata should make a consistent world out of it.

RDF's introduction has not been straightforward—and there has been a lot of discussion about how and even whether it should be introduced. This is because, like many new languages, it confronts a basic dilemma inherent in the design of any language. HTML is a limiting language: You can use it only to express hypertext documents. Java, by contrast, isn't: You can write a bit of Java to do almost anything. Limiting languages are useful because you can, for example, analyze an HTML page element by element, convert it into other formats, index it, and whatever. It is clear what every bit is for. People do all kinds of things with HTML pages that the pages were never originally intended for. A Java applet is different. Because Java is a *complete* programming language, you can use it to do anything, including creating a penguin that does somersaults. However, because Java is so powerful, the only way to figure out what a Java applet will do is to run it and watch. When I designed HTML for the Web, I chose to avoid giving it more power than it absolutely needed—a "principle of least power," which I have stuck to ever since. I could have used a language like Donald Knuth's "T_eX," which though it looks like a markup language is in fact a programming language. It would have allowed very fancy typography and all kinds of gimmicks, but there would have been little chance of turning Web pages into anything else. It would allow you to express absolutely anything on the page, but would also have allowed Web pages that could crash, or loop forever. This is the tension.

There is a fear that one day the big brother of RDF will become a programming language, and library cards will start composing music, and checks will be made payable to a person whose name can be calculated only by using two hundred years of computer time. Looking at my plans for the Semantic Web,

computer scientists at MIT and consortium members alike have been known to raise their eyebrows and suggest that we should keep the strength of the total language down. Should we, then, prevent the presence of powerfully descriptive languages on the Web?

The answer is that within many applications on the Web we should, but that in the Web as a whole we should not. Why? Because when you look at the complexity of the world that the Semantic Web must be able to describe, you realize that it must be possible to use any amount of power as needed. A reason for the success of the Web is that hypertext is so flexible a medium that the Web does not constrain the knowledge it tries to represent. The same must be true for the web of meaning. In fact, the web of everything we know and use from day to day is complex: We need the power of a strong language to represent it.

The trick here, though, is to make sure that each limited mechanical part of the Web, each application, is within itself composed of simple parts that will never get too powerful. In many places we need the transparent simplicity of HTML—so each application, like an ATM machine, will work in a well-defined way. The mechanisms for metadata, privacy, payment, and so on will all work in a well-defined way. The art of designing applications in the future will be to fit them into the new Web in all its complexity, yet make them individually simple enough to work reliably every time. However, the total Web of all the data from each of the applications of RDF will make a very complex world, in which it will be possible to ask unanswerable questions. That is how the world is. The existence of such questions will not stop the world from turning, or cause weird things to happen to traffic lights. But it will open the door to some very interesting new applications that do roam over the whole intractable, incalculable Web and, while not promising anything, deliver a lot.

. . .

To keep a given application simple, RDF documents can be limited so that they take on only certain forms. Every RDF document comes with a pointer at the top to its RDF *schema*—a master list of the data terms used in the document. Anyone can create a new schema document. Two related schema languages are in the works, one for XML and one for RDF. Between them, they will tell any person or program about the elements of a Web page they describe—for example, that a person's name is a string of characters but their age is a number. This provides everything needed to define how databases are represented, and to start making all the existing data available. They also provide the tools for keeping the expressive power of an RDF document limited and its behavior predictable. It allows us to unleash, bit by bit, the monster of an expressive language as we need it.

As the power is unleashed, computers on the Semantic Web achieve at first the ability to describe, then to infer, and then to reason. The schema is a huge step, and one that will enable a vast amount of interoperability and extra functionality. However, it still only categorizes data. It says nothing about meaning or understanding.

People "come to a common understanding" by achieving a sufficiently similar set of consistent associations between words. This enables people to work together. Some understandings that we regard as absolute truths, like the mathematical truth that a straight line is defined by two different points, are simple patterns. Other understandings, such as my understanding of someone's anger at an injustice, are based on complex patterns of associations whose complete anatomy we are not fully aware of.

When people "understand" something new, it means they can relate it to other things they already understand well enough. Two people from different planets can settle the difference between red and blue by each making a prism, passing light through it, and seeing which color bends farther. But the difference between love and respect will be hashed out only in interminable discussions. Like words in the dictionary, everything—until we tie things down to the physical world—is defined in terms of other things.

This is also the basis of how computers can "understand" something. We learn very simple things—such as to associate the word *hot* with a burning feeling—by early "programming" of our brains. Similarly, we can program a computer to do simple things, like make a bank payment, and then we loosely say it "understands" an electronic check. Alternatively, a computer could complete the process by following links on the Semantic Web that tell it how to convert each term in a document it doesn't understand into a term it does understand. I use the word *semantic* for this sort of machine-processible relative form of "meaning." The Semantic Web is the web of connections between different forms of data that allow a machine to do something it wasn't able to do directly.

This may sound boring until it is scaled up to the entirety of the Web. Imagine what computers can understand when there is a vast tangle of interconnected terms and data that can automatically be followed. The power we will have at our fingertips will be awesome. Computers will "understand" in the sense that they will have achieved a dramatic increase in function by linking very many meanings.

To build understanding, we need to be able to link terms. This will be made possible by *inference languages*, which work one level above the schema languages. Inference languages allow computers to explain to each other that two terms that may seem different are in some way the same—a little like an English-French dictionary. Inference languages will allow computers to convert data from one format to another.

Databases are continually produced by different groups and companies, without knowledge of each other. Rarely does anyone stop the process to try to define globally consistent terms for each of the columns in the database tables. When we can link terms,

even many years later, a computer will be able to understand that what one company calls "mean-diurnal-temperature" is the same as what another company calls "daily-average-temp." If HTML and the Web made all the online documents look like one huge book, RDF, schema, and inference languages will make all the data in the world look like one huge database.

When we have the inference layer, finding the yellow car for sale becomes possible even if I ask for a yellow automobile. When trying to fill in a tax form, my RDF-aware computer can follow links out to the government's schema for it, find pointers to the rules, and fill in all those lines for me by inference from other data it already knows.

As with the current Web, decentralization is the underlying design principle that will give the Semantic Web its ability to become more than the sum of its parts.

There have been many projects to store interlinked meanings on a computer. The field has been called *knowledge representation.* These efforts typically use simple logical definitions such as the following: A vehicle is a thing, a car is a vehicle, a wheel is thing, a car has four wheels—and so on. If enough definitions are entered, a program could answer questions by following the links of the database and, in a mechanical way, pretend to think. The problem is that these systems are designed around a central database, which has room for only one conceptual definition of "car." They are not designed to link to other databases.

The Web, in contrast, does not try to define a whole system, just one Web page at any one time. Every page can link to every other. In like fashion, the Semantic Web will allow different sites to have their own definition of "car." It can do this because the inference layer will allow machines to link definitions. This allows us to drop the requirement that two people have the same rigid idea of what something "is." In this way, the European Commission can draw up what it thinks of as a tax form. The U.S.

government can draw up its own tax form. As long as the information is in machine-understandable form, a Semantic Web program can follow semantic links to deduce that line 2 on the European form is like line 3A on the U.S. form, which is like line 1 on the New York State tax form.

Suppose I ask my computer to give me a business card for Piedro from Quadradynamics, but it doesn't have one. It can scan an invoice for his company name, address, and phone number, and take his e-mail address from a message, and present all the information needed for a business card. I might be the first to establish that mapping between fields, but now anyone who learns of those links can derive a business card from an e-mailed invoice. If I publish the relationships, the links between fields, as a bit of RDF, then the Semantic Web as a whole knows the equivalence.

Forgive the simplified examples, but I hope the point is clear: Concepts become linked together. When, eventually, thousands of forms are linked together through the field for "family name" or "last name" or "surname," then anyone analyzing the Web would realize that there is an important common concept here. The neat thing is that no one has to do that analysis. The concept of "family name" simply begins to emerge as an important property of a person. Like a child learning an idea from frequent encounters, the Semantic Web "learns" a concept from frequent contributions from different independent sources. A compelling note is that the Semantic Web does this without relying on English or any natural language for understanding. It won't translate poetry, but it will translate invoices, catalogues, and the stuff of commerce, bureaucracy, travel, taxes, and so much more.

The reasoning behind this approach, then, is that there is no central repository of information, and no one authority on anything. By linking things together we can go a very long way toward creating common understanding. The Semantic Web will work when terms are generally agreed upon, when they are not,

and most often in the real-life fractal mess of terms that have various degrees of acceptance, whether in obscure fields or global cultures.

Making global standards is hard. The larger the number of people who are involved, the worse it is. In actuality, people can work together with only a few global understandings, and many local and regional ones. As with international and federal laws, and the Web, the minimalist design principle applies: Try to constrain as little as possible to meet the general goal. International commerce works using global concepts of trading and debt, but it does not require everyone to use the same currency, or to have the same penalties for theft, and so on.

Plenty of groups apart from W3C have found out how hard it is to get global agreement under pressure of local variations. Libraries use a system called a MARC record, which is a way of transmitting the contents of a library catalogue card. Electronic Data Interchange (EDI) was created a decade ago for conducting commerce electronically, with standard electronic equivalents of things like order forms and invoices. In both cases, there was never complete agreement about all the fields. Some standards were defined, but there were in practice regional or company-wide variations. Normal standards processes leave us with the impossible dilemma of whether we should have just one-to-one agreements, so that a Boeing invoice and an Airbus invoice are well defined but quite different, or whether we should postpone trying to do any electronic commerce until we define what an invoice is globally.

The plan for the Semantic Web is to be able to move smoothly from one situation to another, and to work together with a mixture. XML namespaces will allow documents to work in a mixture of globally standard terms and locally agreed-upon terms. The inference languages will allow computers to translate perhaps not all of a document, but enough of it to be able to act

on it. Operating on such "partial understanding" is fundamental, and we do it all the time in the nonelectronic world. When someone in Uruguay sends an American an invoice, the receiver can't read most of it because it's in Spanish, but he can figure out that it is an invoice because it has references to a purchase-order number, parts numbers, the amount that has to be paid, and whom to pay. That's enough to decide that this is something he should pay, and to enable him to pay it. The two entities are operating with overlapping vocabularies. The invoice is consistent with those drafted in Uruguay, and U.S. invoices are consistent as well, and there is enough commonality between them to allow the transaction to be conducted. This happens with no central authority that mandates how an invoice must be formulated.

As long as documents are created within the same logical framework, such as RDF, partial understanding will be possible. This is how computers will work across boundaries, without people having to meet to agree on every specific term globally.

There will still be an incentive for standards to evolve, although they will be able to evolve steadily rather than by a series of battles. Once an industry association, say, sets a standard for metadata for invoices, business cards, purchase orders, shipping labels, and a handful of other e-commerce forms, then suddenly millions of people and companies with all sorts of computers, software, and networks could conduct business electronically. Who will decide what the standard fields for an invoice should be? Not the Web Consortium. They might arise in different ways, through ad hoc groups or individual companies or people. All the Web Consortium needs to do is set up the basic protocols that allow the inference rules to be defined, and each specialized slice of life will establish the common agreements needed to make it work for them.

Perhaps the most important contribution of the Semantic Web will be in providing a basis for the general Web's future evolution.

The consortium's two original goals were to help the Web maintain interoperability and to help it maintain "evolvability." We knew what we needed for interoperability. *Evolvability* was just a buzzword. But if the consortium can now create an environment in which standardization processes become a property of how the Web and society work together, then we will have created something that not only is magic, but is capable of becoming ever more magical.

The Web has to be able to change slowly, one step at a time, without being stopped and redesigned from the ground up. This is true not only for the Web, but for Web applications—the concepts, machines, and social systems that are built on top of it. For, even as the Web may change, the appliances using it will change much more. Applications on the Web aren't suddenly created. They evolve from the smallest idea and grow stronger or more complex.

To make this buzzword concrete, just take that all too frequent frustration that arises when a version-4 word processor comes across a version-5 document and can't read it. The program typically throws up its hands in horror at such an encounter with the future. It stops, because it figures (quite reasonably) that it cannot possibly understand a version-5 language, which had not been invented when the program was written. However, with the inference languages, a version-5 document will be "self-describing." It will be provide a URI for the version-5 schema. The version-4 program can find the schema and, linked to it, rules for converting a version-5 document back into a version-4 document where possible. The only requirement is that the version-4 software needs to have been written so that it can understand the language in which the rules are written. That RDF inference language, then, has to be a standard.

When we unleash the power of RDF so that it allows us to express inference rules, we can still constrain it so that it is not such an expressive language that it will frighten people. The

inference rules won't have to be a full programming language. They will be analyzable and separable, and should not present a threat. However, for automating some real-life tasks, the language will have to become more powerful.

Taking the tax form again, imagine that the instructions for filling out your tax return are written in a computer language. The instructions are full of *if*s and *but*s. They include arithmetic, and alternatives. A machine, to be able follow these instructions, will need a fairly general ability to reason. It will have to figure out what to put on each line by following links to find relationships between data such as electronic bank statements, pay slips, and expense receipts.

What is the advantage of this approach over, say, a tax-preparation program, or just giving in and writing a Java program to do it? The advantage of putting the rules in RDF is that in doing so, all the reasoning is exposed, whereas a program is a black box: You don't see what happens inside it. When I used a tax program to figure out my 1997 taxes, it got the outcome wrong. I think it got confused between estimated tax paid *in* 1997 and that paid *for* 1997, but I'll never know for sure. It read all my information and filled in the form incorrectly. I overrode the result, but I couldn't fix the program because I couldn't see any of its workings. The only way I could have checked the program would have been to do the job completely myself by hand. If a reasoning engine had pulled in all the data and figured the taxes, I could have asked it why it did what it did, and corrected the source of the problem.

Being able to ask "Why?" is important. It allows the user to trace back to the assumptions that were made, and the rules and data used. Reasoning engines will allow us to manipulate, figure, find, and prove logical and numeric things over a wide-open field of applications. They will allow us to handle data that do not fall into clean categories such as "financial," "travel planning," and "calendar." And they are essential to our trusting online results,

because they will give us the power to know how the results were derived.

The disadvantage of using reasoning engines is that, because they can combine data from all over the Web in their search for an answer, it can be too easy to ask an open question that will result in an endless quest. Even though we have well-defined rules as to who can access the consortium's members-only Web site, one can't just walk up to it and ask for admittance. This would ask the Web server to start an open-ended search for some good reason. We can't allow our Web server to waste time doing that; a user has to come equipped with some proof. Currently, users are asked under what rule or through which member they have right of access. A human being checks the logic. We'd like to do it automatically. In these cases we need a special form of RDF in which the explanation can be conveyed—if you like, a statement with all the whys answered. While finding good argument for why someone should have access may involve large searches, or inside knowledge, or complex reasoning, once that argument has been found, checking it is a mechanical matter we could leave to a simple tool. Hence the need for a language for carrying a proof across the Internet. A proof is just a list of sources for information, with pointers to the inference rules that were used to get from one step to the next.

In the complexity of the real world, life can proceed even when questions exist that reasoning engines can't answer. We just don't make essential parts of our daily business depend on answering them. We can support collaboration with a technical infrastructure that can respect society's needs in all their complexity.

Of course, our belief in each document will be based in the future on public key cryptography digital signatures. A "trust engine" will be a reasoning engine with a bolted-on signature checker giving it an inherent ability to validate a signature. The trust engine is the most powerful sort of agent on the Semantic

Web. There have been projects in which a trust engine used a less powerful language, but I honestly think that, looking at the reality of life, we will need a very expressive language to express real trust, and trust engines capable of understanding such a language. The trick that will make the system work in practice will be to send explanations around in most cases, instead of expecting the receiver to figure out why it should believe something.

Creating the actual digital signature on a document is the simpler part of the trust technology. It can be done regardless of the language used to create the document. It gives the ability to sign a document, or part of a document, with a key, and to verify that a document has been signed with a key. The plan is for a standard way to sign any XML document. The consortium in 1999 initiated this activity, combining earlier experience signing PICS labels with new ideas from the banking industry.

The other part of trust, which actually weaves the Web of Trust, is the mesh of statements about who will trust statements of what form when they are signed with what keys. This is where the meat is, the real mirroring of society in technology. Getting this right will enable everything from collaborating couples to commerce between corporations, and allow us actually to trust machines to work on our behalf. As the Web is used to represent more and more of what goes on in life, establishing trust gets more complicated. Right now, the real-life situation is too complicated for our online tools.

In most of our daily lives, then, even in a complex world, each step should be straightforward. We won't have to unleash the full power of RDF to get our job done. There is no need to fear that using RDF will involve computers in guesswork.

However, now that we are considering the most complex of cases, we must not ignore those in which computers try to give reasonably good answers to open questions. The techniques they use are *heuristics*—ways of making decisions when all the alternatives can't be explored. When a person uses a search engine, and

casts her eye over the first page of returns for a promising lead, she is using a heuristic. Maybe she looks at the titles, or the first few lines quoted, or the URIs themselves; in any case, using heuristics is an acquired art. Heuristic programs at a bank are the ones that sound a warning when a person's credit-card spending pattern seems to differ from the usual.

The interplay between heuristic and strictly logical systems will be interesting. Heuristics will make guesses, and logic will check them. Robots will scan the Web and build indexes of certain forms of data, and those indexes will become not definitive, but so good that they can be used as definitive for many purposes. Heuristics may become so good that they seem perfect. The Semantic Web is being carefully designed so that it does not have to answer open questions. That is why it will work and grow. But in the end it will also provide a foundation for those programs that can use heuristics to tackle the previously untacklable.

From here on it gets difficult to predict what will happen on the Semantic Web. Because we will be able to define trust boundaries, we will be inclined, within those boundaries, to give tools more power. Techniques like viruses and chain letters, which we now think of as destructive, will become ways of getting a job done. We will use heuristics and ask open questions only when we have made a solid foundation of predictable ways of answering straightforward questions. We will be sorcerers in our new world when we have learned to control our creations.

Even if the blueprint of technologies to achieve the new Web is not crystal clear, the macroscopic view I've presented should at least convey that a lot of work has to be done. Some of it is far along. Some of it is still a gleam in the eye.

As work progresses, we will see more precisely how the pieces fit together. Right now the final architecture is hypothetical; I'm saying it *could* fit together, it *should* fit together. When I try to explain the architecture now, I get the same distant look in

people's eyes as I did in 1989, when I tried to explain how global hypertext would work. But I've found a few individuals who share the vision; I can see it from the way they gesticulate and talk rapidly. In these rare cases I also have that same gut feeling as I did a decade ago: They'll work for whomever they have to work for, do whatever it takes, to help make the dream come true. Once again, it's going to be a grassroots thing.

The blueprint for the new Web is also much like my 1989 proposal for the original Web. It has a social base, a technological plan, and some basic philosophy. A few people get it; most don't. In the very beginning I wrote the World Wide Web code, then went out into the world to promote the vision, made the technology freely available so people could start working on their little piece of it, and encouraged them.

Today the consortium might write some of the code, or at least coordinate the writing of the code. Perhaps the computer community will share the vision and complete the pieces according to a business model that spans a number of years. Or perhaps someone watching from the sidelines will suddenly realize: "I know how I can do this. I don't know how to figure out a business model for it, but I think I can write the code in two weeks."

Work on the first Web by people in various places progressed in a fairly coordinated way because I had written the early code, which gave other people something to write to. Now we have two tools we didn't have then. One is the consortium—a place where people can come together as well as a source of advanced software platforms like Jigsaw and Apache that people can use to try out their new ideas. The second tool is the Web itself. Spreading the word will be so much easier. I can publish this plan to the world even when it's only half finished. The normal academic way Robert Cailliau and I could spread the original proposal was to get it into the hypertext conference proceedings—and it was rejected. This blueprint is not conference ready either, and I'm not inclined to make it so. We'll just get the information out there

so people can point to it and discuss it. Once a seed is sown it will contain pointers back to where it came from, so ideas will spread much more rapidly.

Cynics have already said to me, "You really think this time around people are going to pick up on the architecture, and spend hours and hours on it as Pei Wei and all the others did?" Yes. Because that's just what the cynics said in 1989. They said, "Oh, well, this is just too much to take on." But remember, it takes only a half dozen good people in the right places. Back then, finding that half dozen took a long time. Today, the world can come to the consortium, plug in their ideas, and have them disseminated.

Indeed, the danger this time is that we get six hundred people creating reasoning engines in their garages across the land. But if they try to patent what they're doing, each one of them thinking they've found the grand solution first, or if they build palisades of proprietary formats and use peculiar, undocumented ways of doing things, they will just get in the way. If, through the consortium, they come openly to the table for discussion, this could all work out remarkably soon.

I mention patents in passing, but in fact they are a great stumbling block for Web development. Developers are stalling their efforts in a given direction when they hear rumors that some company may have a patent that may involve the technology. Currently, in the United States (unlike in many countries), it is possible to patent part of the way a program does something. This is a little like patenting a business procedure: It is difficult to define when something really is "novel." Certainly among some patents I have looked at I have found it difficult to find anything that gives me that "ah-ha" feeling of a new idea. Some just take a well-known process (like interlibrary loan or betting on a race) and do it in software. Others combine well-known techniques in apparently arbitrary ways to no added effect—like patenting

going shopping in a striped automobile on a Thursday. They pass the test of apparent novelty because there is no existing document describing exactly such a process. In 1980, a device for delivering a book electronically, or a device for online gambling, might have seemed novel, but now these things are just obvious Web versions of well-known things. The U.S. Patent and Trademark Office, ill-equipped to search for "prior art" (earlier occurrence of the same idea) in this new field, seems to have allowed through patents by default.

It is often difficult to know what a patent is about at all because it is written obscurely using language quite different from that which a normal programmer would use. There is a reason for this: The weapon is fear of a patent suit, rather than the patent itself. Companies cross-license patents to each other without ever settling in court what those patents actually mean. Fear is increased by uncertainty and doubt, and so there is an incentive for obscurity. Only the courts can determine what a patent means, and the legal effort and time involved dwarfs the engineering effort.

This atmosphere is new. Software patents are new. The Internet ethos in the seventies and eighties was one of sharing for the common good, and it would have been unthinkable for a player to ask fees just for implementing a standard protocol such as HTTP. Now things are changing. Large companies stockpile patents as a threat of retaliation against suits from their peers. Small companies may be terrified to enter the business.

The lure of getting a cut of some fundamental part of the new infrastructure is strong. Some companies (or even individuals) make a living only by making up patents and suing larger companies, making themselves immune to retaliation by not actually making or selling any products at all. The original aim of patents—to promote the publication and deployment of ideas and to protect the incentive for research—is noble, but abuse is now a very serious problem.