INFORMATION REFINING

With the proliferation of e-mail, electronic publishing, bulletin boards, on-line documentation and the like, the volume of text distributed and stored electronically is overwhelming. We have some tools to retrieve individual pieces of text if we know what we're looking for, but we need better ways to categorize texts and even summarize their meaning. We have covered many of them over the past few years; here's a more conceptual overview of the issues and some additional, more recent examples. (See also Release 1.0, 87-9, 87-11, 88-1, 89-3, 89-7, 89-12, 90-2.)

Information is intangible and reusable; its value lies in what you make of it. The value added by refining information can be far greater than that of the raw information itself. Consider: A name and address is a useful nugget of data. That same name and address mentioned casually in a letter is information that needs refining. Finally, that name and address in context -- for instance, in a letter indicating that the individual may be in the market for a two-story brownstone -- is potentially the most valuable of all. (There's a fourth possibility: The letter is to a friend and should not be refined: Each nuance has great meaning; he's telling her he's about to move out. But we're concerned mostly with commercial information here.)

Our topic, "information refining," is a term coined by John Clippinger of Starr King Communications (now moving to Coopers & Lybrand to work on this area). More or less, it is the process of converting information ore into something useful -- the "refining" of textual information into more structured information that can be represented and manipulated by computers. (Analysis of data in databases is another matter, and analysis of sound and images is still mostly beyond us in a commercially realistic sense.)

Information can be refined by people -- editors, for example -- but a number of techniques ranging from string search to so-called natural-language understanding can automate or at least assist the process. The more regular the initial format and content, the more efficient the tools can be; the more irregular, the more value can be added by refinement. Information can be classified, organized, filtered/selected and combed.
for relevant facts. Once refined, the output could be a database of financial transactions or earnings reports or a list of job candidates or a selection of articles for a human to read or a set of grammatical errors for correction. It could trigger (and provide input for) the execution of some action, such as entering an appointment in a schedule and sending a letter of confirmation, selecting a news item from a stream of uninteresting news, sending a letter of apology to an angry customer or transferring $2.9 million to an overseas account. Or selected pieces of it could be assembled into a personalized newsletter or marketing piece (see Release 1.0, 90-2). Items concerning the same subject can be linked to each other; cross-references can be represented electronically.

We're dealing here with practical problems; see the box across for a discussion of "meaning." The goal of classification or understanding of texts for our purposes is simply to represent a text as useful, structured information so that it can be manipulated electronically. In short, the system doesn't need to understand the language; it just needs a way to represent what is relevant about its topic or its meaning. It strips the text to find concepts it recognizes and the relationships among them.

Topic and summary: Cast and plot

The tools divide into two basic but overlapping areas -- ones that deal with words and concepts, for text classification and filtering, and ones that try to derive summaries of the content (or "meaning"), generally by parsing sentences and recognizing standard "scripts" or story lines. In short, there are ones that determine the text concerns a boy and a girl, and ones that can figure out the story line -- whether the boy won or lost the girl. Both can be useful; the parsers tend to work in far more limited domains, with a small range of possible story lines -- for example, mergers and acquisitions, earnings reports, disaster reports for an insurance company, purchase orders, sports scores.

Underlying each representation is a model. The model may be as simple as a database or a list of index terms by which text chunks are classified, or as complex as a set of rules, a semi-hierarchical tree, a semantic net, an object-oriented database or a set of situations and analogies. The model for filtering and classification systems is generally some set of relationships among lexical items; the model for the parsers is the story lines (or simply correct sentences, in the case of grammar/style checkers, which check to see if the text follows rules but generally make no attempt to determine content). You could say that classifiers attempt to fit each item as a concept into a model, whereas the summarizers try to create a model of interacting concepts for each chunk. Of course, meaning being what it is, both the means and the results may involve a combination of these things (a mixture of concepts, in fact).

The four sections following explore four approaches, three for text classification and the final one for summarization:

I - simple queries, no model;
II - simple queries, complex model of relationships among concepts;
III - complex queries (using rules and parsers), any kind of model; and
IV - complex model for representation of individual text items, not just the relations among items.
I -- Tools for determining topics (simple queries, no model)

The basic tool is plain old string search for text retrieval: you index text -- words, phrases, etc. -- and then you search by those strings. (The indexing has no conceptual role; it's simply a list of all the possible words/phrases in the text base, with pointers to where they can be found, and is a convenience for finding and counting words.) In a simple Boolean system, a query is just a list of words whose presence indicates the concept. Most systems allow you to edit and store queries and give them names, which is a convenience for the user. You can also build more complex Boolean queries, specifying combinations or the presence of one word within a given distance from another. (See Release 1.0, 88-1.)

Similarity ranking

Next, you can build more complex queries, which may weight words according to their relative frequency in the overall textbase and the target chunk of text, co-occurrences of words, and other parameters. "Similarity" ranking figuratively builds an N-dimensional space, in which each chunk of text is a vector, with its dimensions determined by the number of occurrences of each word. (Each word represents one dimension; most text chunks have 0 length in most dimensions. Alternatively, you could create a one-dimensional vector for each text chunk; each position in the string is a word, and a 1 in any position indicates that the text chunk contains the word referred to by that position in the string. This system doesn't weight words at all, but can also be useful. See Release 1.0, 88-1.)

A variety of formulas are used to determine the "similarity" of any text chunk to any other; figuratively, how close are the vectors to each other? Do they point in the same direction? And so on. Each kind of formula will produce slightly different results, but in the end similar articles are likely to be so identified. The vectors can represent either the entire database, an individual text chunk, or the concept/query itself. This allows for a simple form of so-called natural-language query, where the system regards all the neutral words as noise, and pays attention to those that discriminate one topic from another (in italics in these examples):

Will OS/2 ever reach its potential and beat out UNIX?

What's the name of the river in Budapest?

When did Juan meet Alice?

You won't get a direct answer, but you can probably find the answer in the responses. The use of "potential" in the first example might pull in some stories about electric circuits, but those can be discarded. The better systems allow for refinement of queries by selecting or rejecting particular texts the system finds and running a query again -- in essence, the user is saying "weight the words in these texts more" or "weight them negatively."

Most of the industrial-strength systems now use some variant of these techniques, including Individual Inc., Dow Jones' DowQuest (which runs on the Connection Machine), and other learn-as-you-go systems which refine queries by adjusting the weightings of words. They are all more or less based on the work of Gerard Salton at Cornell University.

Release 1.0 16 March 1990
Language and meaning

Yes, I'd love to have lunch. He left the firm for personal reasons.
Your talk was memorable! I'll call you. We'll be back on track next
quarter. Of course I remember you! Fine, thank you. I love you. I
sent it yesterday. Send him a letter thanking him, blah, blah, blah.
I'll take care of it. Mom, I have a tummyache. Well, that's Fred!
Such a pretty color! We have a little weather up ahead. I don't re-
call... Our family is very close. He's terrific; you'll love him!
Oh, no, really, I didn't want it anyway. I am a Berliner! Yes.

First of all, it's not language we're dealing with; it's meaning.
But what is "meaning"? Can anyone really understand natural lan-
guage, as illustrated in the samples above? The only being that can
totally understand everything you say is an exact clone, with the
same experiences as you up to this very moment. And even then, are
you hiding your true feelings from yourself?

Computers can transform information from one representation to anoth-
er for easier understanding by humans. But the "meaning" itself,
some would say, exists only in the minds of speaker and listener (or
writer and reader). The meaning is the purpose, in the sense of what
the speaker wants the listener to do, even if it is only to believe
something to be true.... What about meaning as a description of the
world? That is only data, not meaning. An inert piece of text (or a
movie) can have meaning only as interpreted by someone. (But if in-
terpreted by a computer program? Perhaps it has no realized or prac-
tical meaning until its results reach a human somewhere.

Communication occurs when there is some consonance between the mean-
ings of the speaker and the listener. Frequently, the speaker may
mean more than the listener hears, or they may work from different
assumptions. It can be easier to talk to strangers or foreigners
than to friends, because with them you're aware of the limits to un-
derstanding that you tend to forget among friends. What script do
you bring to the phrase "I love you," or to the family dinner table?
Does your family consider reading in company rude? Our family thinks
it shows a cozy togetherness which allows each person to do his own
thing. Consider the old saw about England and America, two countries
divided by a common language.

Things we don't talk about: The power of explicitness

What are the implications of euphemism and explicitness? There are
lots of things you know but don't talk about, for a variety of rea-
sons: not to hurt someone's feelings, not to make too vivid unpleas-
ant (death, illness) or powerful things (sex, authority). Resistance
to groupware is partly founded on the same issue. It reveals who
works and who shirks, who reports to whom, where people rank. And
quantifying and objectifying removes the sacredness of, say, a love
relationship. But then why is it okay to say "the best mother a guy could have" (Jimmy Stewart), but not "the best sex I've ever had" (Marla Maples, reportedly)? Because you can have several lovers but only one mother? The difference between sex and love plays a part, but how?

On the other hand, explicitness or naming can also give you the distance -- the perspective -- to handle a confusing onslaught of emotions and perceptions. "I feel this way because I'm a teenager." "Boys will be boys." "Oh, he's just angry...."

Language helps us to understand -- or perhaps it merely oversimplifies -- complexity. Imagine looking at the world without perspective (as people with certain neurological disorders do). You would see what a computer sees -- lots of stuff, varying lights all over the place, but no sharp edges. Now put in visual pattern-recognition, the skill that took you several months to learn as an infant. Aha! This is a discrete object. But what is it? Oh, it's a cup. There's something inside it, must be a liquid... Things come into focus when you recognize them; seeing gives you the model, but the model helps you see.

Language is an additional level -- one more way of defining things so you can manipulate them in your mind without having to hold all the details. Vision lets you see things one at a time; language lets you see them as examples of metaobjects: This is a cup, different from all other cups yet like them in important ways. This is Alice, the same person she was yesterday, yet in a different mood, in different clothes. You can think about "Alice" so you don't need to think about freckles, the set of a jaw, the curl of a lip, the notes passed in meetings, the conversations. "Alice" is a representation of all you know about her and all you have forgotten because it didn't fit your model -- first impressions filtered through subsequent events. It's a way to organize your experience and your interactions with Alice, who may not have meant to say what you understood.

The model thing

The first time we ever encountered the object/model dichotomy was when we were fourteen. We opened a bathroom cabinet in search of toothpaste. Inside there was a tin of dental floss from Newbery's, a drugstore in Maine, but there was no toothpaste. Thinking idly about the store, we closed the door, and found we couldn't remember the store's name...

*****

What is the meaning of this little essay? From the writer's perspective, it may be musings of a sleepless night, notes to a friend whose baby is starting to talk, a love letter in disguise. For you the reader, its assessment of our (in)ability to communicate may be profoundly sad. Or it may explain why you can't talk to your spouse and cheer you up. Or maybe it's just words on a page, and you pass on.

Release 1.0

16 March 1990
Missing model

However, the complexity and power of the query mechanism has little to do with the model of the concept space in which a particular text chunk may be placed. The query mechanism has to do with matching the text to a concept; the second has to do with how the concepts themselves are organized. All these systems basically offer one-to-one measures of closeness between the concept and the target items.

There's no model beyond the user's own -- an alphabetical listing, say, and whatever meaning he assigns to the names he gives to the concepts. You're just measuring the match or the closeness of the queries to each concept. In this context, a "concept" is the input to which target chunks of text are matched: It could be "'Juan' or 'Alice' and 'budget',' or "Please list stories on French wines," or simply a single word in a book index. Thus a book index is a list of the concepts covered in the book (and is very different from an electronic index which lists all the words in the book).

II -- Structured semantic space (simple queries, complex model of concepts)

Now we come to the notion of organizing the concepts into a model. You may decide you want more than just answers to queries or rankings of similarity. You want a model of the domain you're studying, and a way of placing the text chunks you examine within it. In essence, you've got a model of the world, and queries (simple or complex) that identify text chunks concerning the various concepts in that world. Then you can use the query technologies described above to link target items to concepts in a model of the domain.

A thesaurus indicates what words are linked to each other; a semantic net with numerical values or types on the links between concepts, as in the work of Vladimir Pokhilko (see "Software as medium," Release 1.0, 89-11), holds even more value. Other possible structured models that might be relevant, if not universal, are a chronology or a table of contents or a course syllabus. A writer wanting to categorize source material according to a book outline would find a book outline useful; a student studying for an exam or teacher preparing a course could match texts to the syllabus.

Note: Hypertext is what you get when you link things. When you link things in some automated way, as described above, you usually get something more structured and predictable -- such as a hierarchy. Human-generated hypertext may be a brilliant piece of work, or it could be a lot of unconnected rubbish. If you allow a lot of processes, or a lot of editors or readers you may get a self-organizing system, a text base with some valuable structure that couldn't be predicted (cf. hypertext publishing, Release 1.0, 89-7, or a self-structuring support system such as Answer's Apriori, Release 1.0, 89-7, or "agenda-setting," Release 1.0, 90-2).

Verity: Model matches queries

An interesting tool in this context is Verity's Topic, a premier example of a system that helps users to build hierarchical taxonomies ("topics") and the related queries. It builds and displays relationships between topics and sub- and super-topics. In Topic, the queries are hierarchically related just as the concepts (or nodes) are: If the concept of "the Soviet Union" includes "Sverdlovsk," "Kiev" and "Moscow," then the query for "Moscow" "in-
cludes" the words in the queries for "Sverdlovsk," "Kiev" and "Moscow," with weightings assigned by the builder-user. You could of course request articles about the Soviet Union but not about Kiev.

In fact, Topic is not a true hierarchy but (potentially) a more complex directed acyclic graph; a node can have multiple parents. For example, the query and the concept for "Moscow" could be a node both under "cities" and under "Soviet Union." On the text display, you see only three levels at a time, but if you walk down the Topic tree on a high-resolution Sun display, you may end up seeing duplicate nodes displayed in several locations. Thus it has implicit rules and weights in the way it combines nodes, but it does not easily support the kind of rules with variables ("acquired" followed by a company name) described below.

III -- Complex queries (with rules) for classifying texts by concept

We originally figured that sheer word statistics will always suffice to determine concepts: You can find a story without the precise words you might use if you simply look for co-occurrences and similarity rankings. For example, not every story about France will have the word France in it, but there will be similarities of people, places, named in it to other stories about France. So if you have a good enough query of weighted words, you can determine any topic. With a Connection Machine, for example, you can simply look for articles similar to a target. The lack or presence of any single word isn't deciding; it's a fuzzy sort of match. It just seems very costly in terms of resources to employ rules and parsers.

Two problems: One, the Connection Machine is still fairly expensive, even though (or because?) the software itself is simple. Two: Ultimately, statistical methods are less precise and less extensible. Concepts are explicit, and can be quite complex. From a business perspective, it's ultimately more comforting to define a topic explicitly than to point to an article and say, "something like this." Moreover, because the rules are explicit and compact, they are easier to interpret, modify or extend, and to interpret. It allows you to define concepts by rules (what a computer does and is) rather than by lists ("computers" is IBM, Apple, Compaq, Tandem, Cray).

Finally, rules are more reusable, since you can change the variables in a rule and reuse it, whereas in a word-based approach, you have to create a whole new set of words for each new concept. For example, compare "Juan, Alice, Fred, Sasha, Brian, Ann, Bill, Bill, Gwen, Arkady, Julia..." with this rule: "Any person whose employer is Fancy Functions." The rule can easily be edited to apply to workers at another company simply by changing the company name rather than by building a new list; it also doesn't require updating every time the boss gets angry and fires someone, or whenever the human resources director puts a new relative on the payroll. Rules give you the extensibility that differentiates intelligence from string-recognition, however many strings you may recognize. In the end, rules are generally easier to validate, debug and extend than a statistical approach, although lists may be easier to generate and work quite well in simple cases.

1 Yes, concepts are explicit, but "moods" are not. For some purposes, such as assessing opinions or morale, evaluating moods can be valuable, but that's beside the point here.
A case study: How Reuters classifies stories

One example of powerful categorization is Topic Identification System (TIS), a classifier built jointly in London by Reuters and Carnegie Group around Carnegie's Text Categorization Shell. The first implementation, working since September and live this month, classifies news articles for Country Reports, an archival database of articles indexed by 196 countries and into economics, politics, sports and general, for a total of 200 possible discrete classifications.

A more complex service will start live operation this summer indexing Reuters' Textline service into several hundred topics such as asset transfers, privatizations, labor and currencies, plus company codes. Textline handles business news from more than 1000 publications.

Development manager Steven Weinstein at Reuters estimates that the two systems will save more than $1 million a year on indexing, a singularly boring job, by allowing editors to move to more interesting work. In addition, accuracy, consistency and speed should be improved. The Country Reports version took about six months to develop and has accuracy (both recall and precision) above 90 percent. The Textline classifier took a little more than another person-year to develop.

This effort is an excellent example of the issues and approaches involved (and will form the basis of presentations at two conferences this May; see calendar). Rather than just vague categories, the company wanted specific classifications: For example, a story about the yen that mentions its value against the dollar should be classified under currency and Japan, but not under US. And an economic-outlook story that mentions either currency briefly should not be classified as a currency story at all.

Because the world is hybrid, so is TIS. It begins with concepts, based on weighted words and phrases, including constructions such as "'acquired' followed by a company name." There can be a wide variety of synonyms and related phrases for any concept. Once there is some categorization, discriminating rules can be applied. For example, it resolves the potential confusion among a variety of "dollars" with this rule (paraphrased):

Assign the concept "Australian dollar" if a dollar concept not specific to any country matches, and the Australia concept matches, and the US dollar and the Singapore dollar concept do not match.

Although the entire project took three years and eight person-years, Carnegie's Phil Hayes estimates, much of that was devoted to development of the basic engine, later abstracted out as the Text Categorization Shell. Using TCS and benefiting from their experience, Carnegie or Reuters would probably now need only one person-year of effort writing rules and lexicons to build an equivalent system now -- similar depth and accuracy for a different set of concepts. TCS has a runtime system, and also includes a workbench of tools and debuggers for analyzing word and phrase counts and weights, editing concept definitions and categorization rules, and testing the system for precision and recall against classifications by humans.
Measures of accuracy

Accuracy is a function of recall -- what percentage of the total matches did you find? -- and precision -- how many of the ones you found were in fact matches? What constitutes 100 percent, of course, may be arguable, just as doctors don't always agree on diagnoses, and tax experts and IRS personnel don't always agree on tax items. With traditional word-list systems the sum of precision and recall generally totals about 100, whereas with Reuters' rule-based Text Identification System each alone approaches 100 (at least in the example cited above, for a given domain and after considerable work).

Proper names provide a good illustration of the issue. String-search systems are very good at finding them -- with 100 percent precision (as long as you include synonyms like Big Blue or Ma Bell). However, the goal is to find stories where they are the focus, rather than a passing mention. In tests at Reuters, for example, TIS got high recall (98 percent) for categories linked to specific names but lower precision (84 percent), because text items were flagged whether or not the name was relevant. "'You're cuter than Liz Taylor,' he murmured," may not imply a story about the film industry. The 135 economic categories, by contrast, were tougher to recall (89 percent) than for precision because of the breadth of the concepts involved and the large variety of ways to express them. Could you flag "Rhymes with Rich" as a story about the hotel business? That was Newsweek's title for its cover on Leona Helmsley.

The Reuters system described across provides such fine discrimination, assigning stories to a variety of carefully constructed classifications. By contrast, Desktop Data is using simpler string search (using customer-defined strings and information-vendor story classifications as available) on a pc to select news articles received in real-time over an FM band. It has signed up 15 customers since its launch last month, at $15,000 or so a year, but that number is growing quickly, with Wall Street the quickest to sign up for now. Desktop's NewsEDGE works in real-time more as an alerting mechanism, and is designed to be easily modifiable; the user can simply enter a new keyword. But it suffers from traditional trade-offs, typically with high recall and low precision -- or a lot of extraneous stories that the user can simply ignore.

IV -- Determining story lines: Complex models of individual text items

All the systems above classify text items, so that you can get appropriate pieces of text about any concept defined in your concept model. But more interesting is the notion of summarization, so that you can get precisely the information you need, rather than articles about it.

2 In a complex assignment of intellectual property, Reuters owns the original rule base it got from Carnegie as well as the specific implementation (TIS), an extended rule base and communications interfaces, for modification and reuse within Reuters. But Carnegie Group owns and licenses the underlying categorization and pattern-matching engine, Text Categorization Shell. If Reuters were to remarket TIS, it would owe royalties for TCS to Carnegie.

Release 1.0 16 March 1990
Determining a model of the content, as opposed to a model of the concepts in which an item can be located, is a tough but worthwhile task. If you want to know when Federal Express zapped ZapMail, for example, you don’t want a bunch of articles about ZapMail; you want a single date — the date you can’t specify in a string-search because that’s the information you lack. Or if you want to know what restaurant chains Feed’n’Frenzy acquired, you’d rather get a list than a pile of articles that include "Feed’n’Frenzy" and the words "acquired," "bought" and "merged," including one about the purchase of a new French-fry facility. In other words, you want your information not just filtered but also digested (or refined), with the contents of these various redundant or irrelevant articles reduced to a model that you can query for precisely what you need.

The basic tools for finding meaning are a variety of pattern-recognizing parsers. They know grammar rules -- the stuff of traditional natural language/AI -- and they also have models of the domain: scripts, relationships among concepts, standard actions and transactions, etc.

One part of the system recognizes linguistic constructs -- verbs and nouns, phrases and antecedents, synonyms and complex clauses. Grammar rules deal with parts of speech that the system recognizes through lists and other cues. They also have a customized lexicon of domain-specific synonyms for various calculations, field names, and application commands; e.g., salary = pay x hours worked, emp.no = employee number, etc.³

Another part of the system looks for domain-specific scripts, or generic "meanings," such as acquisitions, transactions, or whatever the content may be. For example: "Country Cousins was acquired by MetroToys for $49 million." "Please wire $200,000 to Mr. X in Nicaragua." "Please book a room at the Berkeley in London." "Earnings fell 10 percent." (But we’re sure they’ll make it up next quarter!)

Some commercial products

For example, Natural Language Incorporated’s Connector is a tool for connecting such scripts (as represented by nouns and verbs) with a database that holds the model -- i.e. the relationships implied by the scripts. I.e., "X pays Y" implies a transaction, with an amount, a date, and an object (an action or a thing) that was paid for. The Connector helps a user/builder to define terms (pay, salary, particular products that the people in this domain might be buying and selling, invoice, bill, due, etc.) and build the model they pertain to. Each transaction is one line in the database, but the system can answer related queries by joining various records. For example, "Does anyone who works for a company in Chapter XI owe us money?" This system might combine internal company records with a database populated by external financial news, set up to collect information about companies whose employees we do business with (among other things).

³ Without the domain-specific content such parsers are useful for grammar-checking, but you can’t have "meaning" without a model of a specific domain -- and some intention. Theoretically you could avoid being domain-specific by having a model of the whole world and everyone’s intentions, but let’s get real!

Release 1.0

16 March 1990
The results of such analysis can fill a database or knowledge base, generate a financial transaction, or trigger a letter of response. The database can also be used to answer queries or as a source of data for traditional applications, as in SCISOR, below. At this point most such systems are in early stages or are considered a source of proprietary advantage by their owners and not discussed; one exception is several installations of Cognitive Systems' ATRANS tool, first installed in July 1986 at Citibank. (Cognitive Systems was founded in 1981 by Roger Schank, a former Yale professor who propounded the notion of conceptual dependencies, including scripts, or standard models of behavior: Juan gives an object to Alice; Alice pays money and takes it; ownership is transferred. Or, Michael is hungry; Michael eats food; Michael is no longer hungry.)

ATRANS monitors incoming telexes for money-transfer instructions, and is in use at Citibank, Irving Trust, and Generale Bank in Brussels, among other places. It has an extremely limited domain -- telexes for money-transfers -- but it is able to save banks considerable time and data-entry. And for banks more than most of us, time is money. Paul Callens, assistant head of banking operations for Generale Bank, for example, says that the system identifies and correctly handles 80 to 85 percent of telexed payment orders, which number 120 to 150 a day. The system automatically generates formatted S.W.I.F.T. messages (Society for Worldwide Interbank Financial Telecommunications, a banking standard). About 15 percent are kicked back and once in a while a mistake occurs in the generated messages, so it's necessary for a human to compare the original telexes to the ATRANS output. Still, use of ATRANS allows the bank to accept properly formatted telexes as late as 10 am, whereas other messages must come in by 8.30 to be handled the same day.4

GE's SCISOR: A database of financial news

SCISOR stands for System for Conceptual Information Summarization, organization and Retrieval, a project at General Electric's R&D Center in Schenectady, NY. Developed by Lisa Rau and Paul Jacobs, SCISOR builds a database about mergers and acquisitions, using stories from financial wires. Stories are first classified as about, not about or maybe about a takeover, in a four-stage process that starts with headline analysis. For example, "Dow Jones Stock Averages" and other recurring headlines are automatically discarded. Then the system uses a keyword filter, looking for words such as "buy," "takeover" and "acquisition" that indicate a story about a merger or acquisition. Next, it applies pattern matches, looking for both positive patterns such as "acquired [COMPANY] for $[AMOUNT]," or negative matches, such as "sale of [COMPANY] unit," indicating a divestiture rather than a merger. Finally, if the system is still unsure it does linguistic and conceptual analysis, knowing that the object of a takeover is a company.

4 On the other hand, messages in the S.W.I.F.T. syntax itself can be received until 11 am. You'd think that this is such a limited domain with such well-defined transactions (or scripts) that most bankers would simply adopt S.W.I.F.T. syntax to save themselves trouble -- and kill the market for ATRANS. In fact, S.W.I.F.T. has been around since the Seventies but it is still used by only half the world's banks, albeit for 75 about percent of transactions. Think of the implications for Electronic Data Interchange. Seems like natural language will be with us for a long time!
At this stage, SCISOR also starts to define the information that will be used to fill the database. Take the complex sentence illustrated above: "Revere said it had received an offer from an investment group to be acquired for $16 a share, or about $127 million." By determining that the investment group is making the offer, SCISOR decides that it is the acquirer, and that Revere must be the company being acquired. It can also distinguish the per-share price from the total value of the acquisition.

SCISOR does this through the cooperation of two modules, TRUMP (for TRansportable Understanding Mechanism Package) and TRUMPET (for TRUMP Expectation Tool). TRUMP is a general-purpose parser and semantic interpreter; it finds the filter words, and knows how to interpret them. It knows (for example)
if an offer is from an object, the object is making the offer. TRUMPET, on
the other hand, contains the domain-specific model: It knows that a unit
making an offer is the acquirer, and matches words to concepts. This is
generally sufficient to generate a full-fledged database entry, as shown in
the form at the bottom of the illustration across. And the system can also
handle questions in much the same way, parsing the questions and deriving
the answers from the database.

Last year, Rau and Jacobs and colleague George Krupka took the basic tech-
nology of SCISOR to build a second system for classifying Navy intelligence
messages as part of a demonstration project for a DARPA conference, and were
able to construct a high-accuracy database generator in one person-month.
(How accurate is, so to speak, classified.) The team has also started work-
ing with other GE units on possible commercial applications of SCISOR, but
that is preliminary.

SCISOR runs on Sun SPARCstations and takes about 10 seconds per story, com-
pared to TIS's 5 seconds on MicroVAX servers. ATRANS takes a second or less
to parse brief messages on a mainframe, about 10 seconds on a LISP machines
and 15 to 30 seconds on a VAX. Of course, these tasks are completely dif-
ferent and performance is hardware-dependent, but these times indicate that
the technology makes sense for commercial use. All the systems are written
in LISP.

Other components of meaning

Other components of meaning are even harder to determine, and are generally
supplied by a human. They include:

talking about spreadsheets. Start by defining words such as here and now.

Intent: The goals of speaker and listener. "The cat is in the kitchen." Why is he
telling you this? Should the cat be brought into the living room? Is it waiting
for a walk? Is it eating the turkey? Or is that where it belongs? "You look
great today!" Why did he say this? Not a problem to address (electronically, commercially) at the moment.

Speech transactions: Acts, not just meanings. "I'll get it to you tomor-
row." "We'll buy them!" "Please file this report." Action Technologies'
The Coordinator asks users to identify their speech transactions more or
less explicitly, and uses those tags to generate a database that monitors
the status of interpersonal obligations: Who owes an answer or a work pro-
duct to whom? Who has requested something but not received an answer?

Overall, we're still at an early stage in dealing with natural text, al-
though we can already go far with structured text that looks natural. "We
still don't realize how complex it is," says Clippinger. "We're fish; we're
swimming in it!"

The moral is: When you use natural language, you need to understand what
you're talking about, with a model, a (n implicit) context/domain. The nat-
ural language part is easy!
MIX YOUR METAPHORS: TRADE-OFFS IN VISUAL PROGRAMMING

We have spent a long time looking for the ideal visual programming language: It could fully represent any program; it would be intelligible to humans and easy for them to use. Recently we've seen a number of attempts ranging from database tools that string icons together words would do fine, to large-scale systems that try to represent everything as geometric constructions. But like other powerful tools, visual programming involves certain trade-offs; it's not uniquely suited to every aspect of every problem.

The ideal approach lies in a combination of visual and symbolic techniques. It's not just a question of whether the user is right- or left-brained; it's a question of what each software component does. Just as it is impossible to comprehend any reasonably complex program in a single glance, so is it impossible to represent (not implement) it fully in a single language. Any single kind of representation may be correct, but incomplete.

The false lure of consistency

There are many different facets of a program, and each can best be represented in different ways, some visual, and some not. Take the notion of higher and lower levels of language -- which refers to closeness to the machine (files, registers, subroutine calls) vs. closeness to human abstractions (employees, tasks, account numbers, letters). While one can be transformed into the other, you lose some control in return for the abstractions.

But there are vertical as well as horizontal divisions. While higher- and lower-level languages are different representations of the same program, there are also different aspects -- including specifically the sequence, the algorithms, the data relationships and the user interface of a program. (Each of these may incorporate the others.)

The market's fascination with graphical user interfaces, direct manipulation, data visualization and other visual approaches has led to some misguided hopes for the existence of a grand, unified visual programming language. This essay is an attempt to illuminate the trade-offs in the use of visual programming -- in terms of the preceding article, to develop a model of this buzz-concept so the words make sense. Let's start with the parts:

VISUAL...

The advantage that visual display gives you over any other medium is high resolution in two dimensions in addition to time. Text is fundamentally linear, and sound, while you can get stereo, is low-resolution, if only because we aren't trained to hear like bats. A visual display can change over time, but it can sit still so you can look at it and define or examine relationships among different components; you can map things. (See how hard it is to discuss things without resorting to visual metaphors.) In visual displays you can transform time into space and represent things occurring simultaneously in different locations so they can be easily distinguished.

Visualization is especially appropriate for displaying or defining those aspects of a program which exist in virtual dimensions: inclusion or exclusion, entity relationships, the sequence of a program (the flow of control or of data), a menu hierarchy, and other aspects which are fundamentally re-
15

relationships. And of course it's ideal for building and displaying a visual interface itself: It's generally easier to design an interface by drawing it than by listing coordinates and colors.

Perhaps the first and most powerful use of visualization was the formatting of code that came with structured programming -- a perfect combination of visual display of structure and textual display of symbols.

While visual displays are ideal for expressing or manipulating the relationships among concepts, symbols are generally better to represent the concepts themselves. An icon is simply a visual symbol instead of a textual one. It may have more associations than a word, and it may be more easily visible from afar. It may be a handy way to portray a member of a class of things, with the name of the individual instance attached. But an icon is not inherently visual; it does not occupy "space" or dimensions in the same way as the visual display of relationships among the concepts represented by icons.

...PROGRAMMING...

"Look Ma! I'm programming!" Take the old story about how everyone in the US would become a telephone operator by the middle of the century if phone usage continued to rise... Of course, it did and they did; every time you dial (011 7095) 921-0902 you are in effect acting as a telephone operator. The analogy to programming has been made more than once. But explore a little deeper: You can call just about anyone, but can you set up a conference call? Leave a voice message that will keep ringing periodically until someone answers? A broad definition of programming focuses on the end -- building an application; a stricter one on the means -- building a sequence of computer instructions. But both imply creation rather than just interaction -- building a program that lasts after the programming is over.

Programming in the strict sense sits at one end of a spectrum, where the programmer has a multitude of choices of processes (not data) and can construct as he pleases any arrangement (loops, branches, sequences) of events, logic, data. At the other, the user has no choices; he is presented with an application that asks him for data. In the middle, he may have an "application" such as a word-processor or a spreadsheet which is part application, part tool, possibly part language. His choices may generate another application -- a spreadsheet model, say, or a database application.

The breadth of choices available orients a system on the spectrum from programming to non-programming. Whether the user types in words or selects them off a list (and whether they're represented as words or icons) is beside the point, although it has a great impact on ease of use and the strain on a user's memory. (The difference between declarative programming, where you specify what you want, and procedural programming, where you describe how to achieve it, is a difference among programming techniques, rather than between them and something else. Declarative constructions can be much easier to formulate -- for the right problems.)

There are many useful ways to generate applications that do not really qualify as programming, ranging from customizing an application by answering questions to selecting options from a menu. Some extremely useful visual tools replace programming by allowing people to reuse objects. You can put visual objects inside other objects, move a document icon to a printer icon,
link buttons to scripts and assemble buttons on cards or screens to create applications (or interfaces to applications). That's a powerful level of abstraction that lets an end-user manipulate things, after someone has written the tough code. (Interleaf (page 20) and HyperCard are examples of this; both also offer textual programming languages so that users can create and modify the objects represented by the icons.) Finally, programming by example is a way of "programming" by direct manipulation that allows the computer to derive the instructions -- and typically lacks the flexibility of true programming.

Return on investment

All other things being equal, which they never are, the practical impact of a system's position along the programming spectrum is the level of flexibility given to the programmer -- or call it the confusion of choices inflicted on the user.

As it happens, the capabilities available to users through graphical environments, object-oriented applications, and links to existing applications will contribute greatly to productivity in the years ahead. Most users don't need the flexibility of strict-sense programming; they need accessible tools and applications that they can configure to their own needs. Many of the tasks they want to do are generic; all that is unique is each user's particular combination of tasks. So although these assembled-component systems may represent only a small amount of "programming" effort, they may end up performing a large part of the actual work that users do. By any measure, that's a superior return on investment.

...LANGUAGE

A language is a representation system; it consists of symbols, not things themselves. (Geometry and mathematics are counter-examples, but we'll let that slide.) The tokens of a language represent concepts, classes of things of which there are multiple instances, procedures which occur many times, etc. The ways the tokens can be combined are a language's grammar. Together, they give you the power to express and manipulate whole classes of things at a time, as represented by the symbols.

Symbols vs. direct manipulation

As we frequently discover when writing this newsletter, there are strange correspondences between seemingly disparate topics. For example, both visual representations and string search allow you to deal with individual objects in an easy-to-understand, direct way. What you see, or what you ask for, is precisely what you get. But just as ultimately it's easier to define a concept than to provide an exhaustive list of examples, so at some point is it easier to express a program with symbols than to represent it all visually, however that can be done.

Direct manipulation allows you to manipulate lists of things, or icons representing classes. You can directly manipulate, for example, graphical objects on a screen when you're designing an interface or a page. You can also directly manipulate icons as symbols, representing classes of objects, and it becomes a matter of taste and screen real-estate how to represent those symbols -- with words in boxes or with icons.
Symbols give you the extended manipulative power that programming is all about. By contrast, dealing directly with representations of individual objects gives you specificity and a sense of tangibility, but limited control.

**VISUAL PROGRAMMING AND...**

...and icons: If it's graphical it's got to be good!

So while visual representations are ideal for laying out sequences and relationships, symbols are better for representing the concepts -- variables, values, algorithms, mathematical transformations, comparisons, rules, whatever. Die-hard visualists want to represent these with icons too. On some occasions that's entirely appropriate, but there are two fallacies: that merely representing a symbol as an icon rather than as a word is "visual programming" in any meaningful way, and that icons are easier to understand than words.

Why do we need symbols? Because there are a variety of concepts, processes and variables, to manipulate. In fact, when you get beyond 20 or so symbols, it becomes pretty hard to remember which is which, and words help. Did the pen represent the word-processing program or the letter templates or that hot new layout program? Is it the blue box or the yellow box that means ...? Does that little box with the B in it represent the buyer record or the booking form? Better you should represent the sequence of events in a visual arrangement of icons, but the "content" of events -- what data fields will be displayed, for example, or what data the user is asked for -- with textual details. A text item in a box may be a perfect mixture of symbolic and visual cues. Icons can be a handy way of giving "grabbability" to symbols identified by text labels.

So use icons where they're convenient and intelligible, but don't give up on text for fear of mixing metaphors. The trick is to use the right metaphors for the right programming constructs. Visual display is great for showing how the employee table is linked to the department table -- or how sales employees are a subclass of employees, but when you want to represent running payroll it's easier to use a word for the action and a word for the employees rather than a tiny dollar bill and a smiling face.

None of this suggests that anyone -- end-user or programmer -- should ever spend unnecessary time typing. Most of the information you need to use is already in the computer -- field names, commands, variables, and so forth. A correctly designed system should have enough hierarchical menus, dialogue boxes, maps, browsers and other display tools that anyone should generally be able to pick something rather than type it in. Which they prefer to do is a matter of convenience. And as most users' fingers know, selecting a sequence of items off a standard sequence of menus is just typing anyway. It's not the typing that's the problem; it's knowing what to type.

...and user-interface design tools: Representing vs. programming

User interfaces are easy to lay out and design directly, through a sort of visual declarative programming, by manipulating graphical objects on a screen. They are correspondingly hard to represent textually. But representing their symbolic behavior is best done through text programs.
Even for visual output, whether you’re a programmer building an application or an end-user building a long document, you need symbols to get the manipulative power that programming is all about. For example, rather than lay out each page of an Interleaf document or each menu of a multi-level program, you want to express layout rules that will apply correctly in a variety of circumstances. You want the title centered on the page whether it’s two lines deep or six; you want each chapter to begin on odd pages; you want the diagram reference to change if you insert an extra diagram. And perhaps you want to associate a routine with a certain table so that it will automatically fetch the latest data from a certain spreadsheet.

Likewise, it’s all very well to draw a single interface for an application, but what you actually want is an interface template with rules that will display a variety of information in a similar way, just as in the many pages generated by a page-layout program. And beyond the page, you want the appropriate behavior assigned to each menu object without having to physically attach it to a button; you should be able to specify that all the items in a list should be linked, and so forth. Or the simple name of a menu item should be sufficient to link it to the appropriate piece of code.

Certainly you want to be able to reuse these capabilities by selecting them from a menu, but there’s no particular need for them to be icons. Their names may have a particular appearance on the screen, such as boldface italics, but they are symbols of the procedures they implement.

...and groupware: Process and content

To take another angle on it, consider any program as groupware. There’s the content part -- calculations and transformations, finding items and doing things to them -- and the coordination part -- first do this, then do that, keep doing this until that happens, do this or do that, go back to this sequence, and so forth. It so happens that visual representation is perfect for the second part, and terrible for the first. It’s easy to represent the relationships spatially, and tough to represent the content that way.

This has interesting implications for groupware: The coordination/workflow part of it may be an ideal target for visual programming languages, although content will still be the province of more traditional approaches such as Lotus Notes. Visual programming can show the flow of work from one person to another; the branches, where it goes to Juan if it’s over $100 and to Alice if it’s under; a hierarchy of delegation from Juan to Alice to Gwen to Arkady. And of course it can also be used for setting up the "database" part: the org charts, the assignments of projects, maps to show the relations between departments, the connections between people and tasks and resources, the schedules. What are GANTT and PERT charts but visualizations of some complex relationships?

...and object-oriented programming: Objects =/= icons

Much current comment confuses object-oriented programming with visual programming, because object-oriented approaches are so handy for manipulating graphics. HyperCard, for one, has a programming language, HyperTalk, that is far from visual, plus the ability to arrange and link buttons and other icons on cards. Call it "reusable code objects identified by buttons and icons." Likewise, NewWave is not a visual programming language; it’s an object-oriented programming/execution environment with graphical objects.
There's a spectrum of object-oriented systems:

- object-oriented applications, which allow users to interact with objects, which in practice are mostly represented visually;

- extensible object-oriented applications, which allow users to interact with (visual) objects and even modify them to some extent, and to create new instances of the objects; and

- object-oriented programming, which requires a programmer to create or reuse and modify robust, reusable objects, usually in some cryptic language such as C, C++ or even LISP. In a very low-level sense any language allows you to reuse code, but not in the modular way object-oriented programming does.

...and artificial reality

All the arguments about visual programming languages also apply to artificial reality; call it a 3D visual programming environment. Consider the two roles of software: agent and medium. Artificial reality is a great medium. You can manipulate things directly; you can imagine you're there; etc. But the power of software really comes from its power to do work for you, when you're not even there. For that you need an agent, one to which you can give directions, rules, cases, examples, etc. With artificial reality, you're pretty much limited to examples. Symbols allow you to manipulate things at a distance, overcoming time and complexity with remote control.

Thus artificial reality is a wonderful idea, but it is incremental, not the final goal of all software development. It will make some things easier -- dealing with individual physical objects, representing individual physical objects exactly. But it will not work so well when you want to describe things in general: This piece should always be twice as high as that piece (vs. this piece should be 3 inches high and that should be 6 inches); how can you represent the relationship? Put two of A inside B? What about A should be 1.46 times the size of B? And should it grow proportionally in width as well, or only in height?

Like multimedia, artificial reality is more a medium of representation than an environment for building agents that can follow your instructions. It doesn't do things; it represents them. And of course, once you've built an agent in some other environment or with explicit tools inside artificial reality, that agent can operate within artificial reality.
APPLICATIONS AS ENVIRONMENTS: NEW WINE IN OLD BOTTLES

This new-product assessment could be about object-oriented programming, cooperative processing, client/server architecture, interprocess communication, groupware, or application integration -- all items any self-respecting new product claims to incorporate -- or even "active documents," the buzzword Interleaf is seeking to apply to the technology in its forthcoming release of the Interleaf Technical Publishing System (TPS), 35,000-plus copies already in use. But given that we have to pick one angle, we think it's about the technical and marketing implications of turning an established application into a powerful, full-fledged environment... for groupware and application integration, using object-oriented programming and cooperative processing with external applications as the technical approach.

What's the toughest thing about new software? Training users, of course. So what's the most valuable thing any software vendor could have? An installed base. That's why more and more vendors are trying to extend their applications rather than create entirely new products. It's not that they can't conceive of something insanely great like, say, NextStep (or ON Technology's ultimate product). It's that they want to be able to sell it. This is the motivation behind Lotus LEAF (Lotus Enhanced Application Facility), Agility's and Beyond's tools for mail-enabled applications (which will mostly enable existing applications; see Release 1.0, 89-10), Wolfram Research's repositioning of Mathematica into a general-purpose programming/control environment (with MathLink), Windows 3.0, and most recently Interleaf's Active Documents.

All the comforts of home

The pitch is the same in each case: "Stay in the cozy application(s) you know and love, but get the power to manipulate other applications." Without even leaving Interleaf, or 1-2-3, Excel or whatever, get all the power of SQL Server, Dow Jones News Retrieval, Mathematica, or your favorite graphing package. With a computer, it's possible to do anything; with multi-tasking and cooperative processing and the kind of support Interleaf and others will provide, it's may even be possible to do this relatively easily.

Basically, you use your application/environment as an environment from which to program and execute other applications. The native environment acts as a client to other applications acting as servers. It's not magic; you don't get to avoid the programming, but it's easier for a user-builder to do the programming and make the capabilities he has programmed flexible and accessible to the end-users who know only the "native" application. Thus you can have a little menu or perhaps a button attached to some part of your native program that will launch a variety of "foreign" application routines to run a 1-2-3 macro, execute a remote database query or transaction, etc.

Call these little subprograms "agents," if you want. They can be triggered by some action you take, by some separate event, or you can launch one directly through your application interface. Because they are triggered from your familiar application (or when you're not even there), they look like extensions of that application. Basically, they let you get the power of other applications through the application you're most familiar with.

All it requires is that the other applications have an API (application program interface) so that the builder-user can write programs to run them, and

Release 1.0 16 March 1990
that they don't get in each other's way, which is a result of multi-tasking.
(For the user to start programming them himself, we're going to need some-
thing closer to a standard macro language, an SML for commands instead of
SQL for data. Candidates include Extended BASIC from Microsoft, LEAF from
Lotus, Personal REXX from IBM, and HyperTalk and supersets from Aldus, Spin-
naker and others. Such a standard would also let a user feel at home with a
variety of applications.) And there's probably a need for communications in
there somewhere too, since it's unlikely all the necessary applications and
data will be available on a single machine. At the least, there will need
 to be some message-passing between the applications; at the most, commands
and data may travel across phone lines to remote sites.

Interleaf: Not just another pretty document

Let's take Interleaf's new version of Technical Publishing System/Active
Documents as an example (available later this year). It can do almost any-
thing, the promotional material says, although it neglects to mention that
it must be programmed to do so. Be that as it may, the results are delight-
ful. Within Interleaf itself there is already a broad range of capabilities
heretofore available only to its programmers; now they are open to anyone
who cares to use them through Interleaf's own visual API. That is, there
are lots of objects with parameters; the user simply finds the appropriate
object and gives the appropriate parameters to the instances.

The document and its components are all objects with their own intelligence
(behavior). Because they contain information about themselves and the
people and applications that contributed the data they contain, they're able
to do surprisingly "intelligent" things. For example, a document could
automatically send itself to people it's addressed to, present different
subsets of its information in different formats to different people. In ad-
tion to traditional live links -- get me the data from this named range in
that spreadsheet and graph them -- it could size the different (named) parts
of a bicycle in a drawing so as to reflect their manufacturing costs as
listed in the spreadsheet.

Better yet, there are "creative" links (nice name) to run other applications
and do things with the results. For example, a table in a manager's report
could periodically query 12 salespeople for their latest results, use 1-2-3
to consolidate them and incorporate the results into a pie chart (or bar
graph if any result is negative). Or it could send a message to each sales-
person, wait for an answer from each, and then print out an appropriate
response (a congratulatory letter if sales are up, a polite one if they're
flat, and a warning if they are down), along with a summary of all sales-
person's progress to the sales manager.

As for groupware, the active document can route itself through an approval
process, assemble itself based on data from a variety of databases and other
sources and reconfigure itself according to realtime criteria (someone's
answers to some questions in a dialogue box) or which person is viewing it.
An active document properly programmed can create personalized versions of
almost anything (cf. our discussion of personalized newsletters in Release
1.0, 89-7 and 90-2), including not just newsletters but catalogues, training
manuals, schoolbooks (as in a recent project at McGraw-Hill), documentation
customized for the particular configuration of a system shipped, etc.

Release 1.0 16 March 1990
As the PR material points out, most documents except for novels are assembled from information already available somewhere else (although it may have to be scanned in, in extremis). Thus a document is really a collection of data; ideally, it's a collection of up-to-date data that can be maintained automatically through links to those sources. In addition, the behavior of components of a document may be determined either by their contents or by external attributes. For example, if a certain field on an expense report contains a number greater than $100, it should be sent to Alice; otherwise, to Juan. External attributes might be who looked at something, a value in a database, the system date, or the results of a query performed each Tuesday.

Call it a document; it's really a representation of several programs, a report generator and a document-processor rolled into one. To the users (including all the people parts of it are automatically sent to) it looks like a document; to the other applications it looks like a control program.

How to sell it

Cleverly, Interleaf is not trying to position itself as an object-oriented development environment competing with, say, NextStep -- not that this looks like a position to emulate anyway. Instead, it's a document processor that has a broader vision of documents: A document is anything that presents data; all the power behind it is applications. The document is the interface to all the other data and applications.

With a product such as Windows 3.0, Microsoft has the marketplace clout and its own applications to bring tangible value to the environment without making it an extension of an existing application. By contrast, NextStep and more especially NewWave provide the framework without enough content.

Interleaf may be a development environment, but it already includes a substantial amount of code for dealing with such basics as presentation of textual, tabular, graphical and image data; manipulation of text (formatting, intelligent linking as in outlines, tables of contents, indexing, citations, footnotes and hypertext); interface objects such as dialogue boxes, scroll bars, menus and windows; and so forth. (Mathematica has the same sort of application richness in the scientific/technical arena.) So even if you don't need or even understand all the capabilities at first, you can find TPS a useful product.

From C to LISP

Interestingly, while the original Interleaf was object-oriented, its objects were passive in the application itself which was written in C for performance. That is, you could manipulate the system objects (paragraphs, graphic items, etc.), and create new instances of them, but you couldn't create new object classes, such as a new kind of document with a link to another application. Interleaf's current incarnation, which allows you to define document components and add attributes that control whether or not something is printed or displayed, were a new kind of object added to the earlier release, but they were hand-coded in painstakingly by experienced programmers. The trick will be to make it easy to create new objects.
RESOURCES & PHONE NUMBERS

Chuck Digate, Beyond, Inc., (617) 621-7123 (new number)
Phil Hayes, Carnegie Group, (412) 642-6900
Steve Mott, Cognitive Systems, (203) 773-0726
Don McLagan, Desktop Data, (617) 890-0042
Lisa Rau, Paul Jacobs, General Electric R&D, (518) 387-5000
Paul Callens, Generale Bank, (011 322) 516-2024
Dave Weinberger, Steve Pelletier, Interleaf, (617) 577-9800
John Manferdelli, Jerry Ginsparg, Natural Language Inc., (415) 841-3500
Steven Weinstein, Reuters, (011 441) 324-7585
John Clippinger, Starr King Communications, (617) 876-4533
Danny Hillis, Dave Waltz, Thinking Machines, (617) 876-1111
Cliff Reid, Verity Corporation, (415) 960-7600
Steve Wolfram, Wolfram Research (Mathematica), (217) 398-0700

SUGGESTED READING

This is a list of books we find of interest currently, rather than an exhaustive bibliography on language understanding:


"Grammatical Man," by Jeremy Campbell. Simon & Schuster, 1982. Grammar is not just what you learn in English class; it's the intersection of choices and constraints. A lucid exposition of information theory and other such topics. (This is the book that finally made us understand music, which has a grammar too.)

"The Intentional Stance," by Daniel Dennett. MIT Press, 1987. We intend to read this book on our next long trip. It has more to do with meaning than with natural language, and with intelligence than with AI. Our comments on pages 4 to 5 and 13 would no doubt have been richer had we already read this book.


COMING SOON

- Groupware from a credible vendor.
- Software hygiene.
- On-line services.
- Distribution in Eastern Europe.
- Network navigation.
- The Douglas brothers -- Hofstadter and Lenat.
- And much more...

Release 1.0 16 March 1990
## RELEASE 1.0 CALENDAR

<table>
<thead>
<tr>
<th>Date Range</th>
<th>Event Description</th>
<th>Contact Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 27-29</td>
<td>1990 AAAI spring symposium series - Stanford. Case-based reasoning, text-based intelligent systems, AI &amp; molecular biology, other neat topics, with all the right people. Sponsored by the American Association for Artificial Intelligence. Call Carol McKenna Hamilton, (415) 328-3123.</td>
<td></td>
</tr>
<tr>
<td>March 27-29</td>
<td>DB/Expo '90 - San Francisco. &quot;Databases, Tools &amp; Connectivity.&quot; With Chris Date, Richard Finkelstein, Vaughan Merlyn, Michael Stonebraker. Contact: Dana NeNardi at (800) 2DBEXPO or (415) 941-8440.</td>
<td></td>
</tr>
<tr>
<td>March 31-April 1</td>
<td>Conference on participatory design - Seattle, WA. Come participate in a conference devoted to figuring out how to bring customer self-service to design. With Lucy Suchman, Xerox PARC. Sponsored by Computer Professionals for Social Responsibility. Call Paul Cyzewski, (415) 967-7079, or Jeff Johnson, (415) 857-7661. Cleverly scheduled to precede...</td>
<td></td>
</tr>
<tr>
<td>April 1-4</td>
<td>CHI'90 - Seattle. &quot;Empowering people&quot; with better, more intelligent interfaces to functional systems. Sponsored by ACM SIGCHI. Call Toni MacHaffie, (503) 591-1981.</td>
<td></td>
</tr>
<tr>
<td>April 1-2</td>
<td>1990 Computer Game Developers' Conference - San Jose, CA. Call Dave Menconi at (408) 942-0387.</td>
<td></td>
</tr>
<tr>
<td>April 2-4</td>
<td>Patricia Seybold's Technology Forum - Cambridge, MA. Distributed network computing and object-oriented environments.</td>
<td></td>
</tr>
</tbody>
</table>

Release 1.0  
16 March 1990
Hear about them from pioneer Doug Englebart; Adele Goldberg, ParcPlace; Bob Frankenberg, HP; Chris Stone, Object management Group; Dan Cheifetz, Odesta; Orion developer Won Kim, MCC; Mark Hatch, Netwise; Robert Sellinger, AT&T; Larry Loucks, IBM; Bertrand "Eiffel" Meyer; Tim Andrews, Ontologic. Call Deb Hay, (617) 742-5200 or (800) 826-2424.

April 2-4


April 2-4


April 5-7

MicroVision Summit 1990 - Newport Beach, CA. For people with something to sell, to meet people looking for something to resell. Group and private meetings with US and overseas resellers such as ComputerLand (Ed Anderson), Inacomp (Rick Inatome), MicroAmerica (Peter Brumme & Jack Littman-Quinn), MicroAge (Jeff McKeever), Ingram Micro D (David Dukes), Metrologie (Alain Schwartzmann). Call Micky Dude, (603) 888-5626.

April 6-7

Advanced workshop in computer access and use for persons with disabilities - Toronto, Ontario. Sponsored by Trace R&D Center. Contact: Gregg Vanderheiden, (608) 262-6966.

April 7-8


April 9-12

*AIIM - Chicago. The annual conference of the Association for Information and Image Management. Sessions on new topics such as image compression, digital paper and transaction processing as well as vertical markets. Call James Breuer at (301) 587-8202.

April 9

*Software Law '90 - San Francisco. Planned to lead right into MacWorld: Patents, copyrights, viruses & liability therefore. Sponsored by Elias & Goodman and Wes Thomas PR. With Adam Osborne; Esther Dyson; Sue Morgan, SoftView; Bob Kohn, Borland; Jeff Cherniss, Advanced Software (DocuComp and the "redlining patent"). Call Karen Thomas at (516) 266-1652 or Paul Goodman at (212) 421-6000.

April 10-13

*Macworld - San Francisco. The usual festival, later this year. Call Peggy Kilburn, (617) 326-9955.

April 11-12


Release 1.0

16 March 1990

April 18-20  Sun Expo Europe '90 - Netherlands Congress Center/The Hague. Sponsored by The Sun Observer Europe. Call Brona Stockton, (512) 331-7761.


April 23-26  First international conference on systems integration - Morristown, NJ. Sponsored by ACM and IEEE groups. Keynotes: Arno A. Penzias, AT&T Bell Laboratories, Robert Berland, IBM. Call Peter Ng, (201) 596-3387.


April 25  Massachusetts Computer Software Council's spring membership meeting - Boston. Call Joyce Plotkin at (617) 437-0600.

April 25-27  Conference on office automation systems - Cambridge, MA. Sponsored by ACM and IEEE groups. Call Joan Staunton, (212) 869-7440, or Robert Allen, (201) 829-4315.


April 30-May 3  Structured Development Forum XI - San Diego. Sponsored by SDF. Call Judith Hays, (503) 745-5692.

May 1-3  Amy Wohl's office systems and networks dialogue - Cambridge, MA. "The office is re-inventing itself." With Bill Campbell (Claris), Bill Crow (Hewlett-Packard), Tony Mondello (IBM), John Scull (MacroMind), Alan Hald (MicroAge). Call Karen Krebs-Wellerstein or Florence Wohl at (215) 667-4842.

May 1-3  Second annual conference on innovative applications of artificial intelligence - Washington, DC. Sponsored by the American Association for Artificial Intelligence; chaired by

Release 1.0

16 March 1990
Alain Rappaport of Neuron Data. With presentations on TIS by Phil Hayes and Steven Weinstein, page 8, and on Prism, a case-based telex classifier from Cognitive Systems, and others. Contact: Julia Munde, (415) 328-3123.

**May 4-5**

Advanced workshop in computer access and use for persons with disabilities - Madison, WI. Sponsored by Trace R&D Center. Contact: Gregg Vanderhelden, (608) 262-6966.

**May 6-9**


**May 7-9**


**May 7-9**

Fortunes outside the 1000 - Chicago. Sponsored by Computer Reseller News and Salomon Bros. With Bobby Orbach, 47th Street Computer; Randy Fields, Mrs. Fields'; Avery More, Compucom; Rick Inatome, Inacomp; Jeff McKeever, MicroAge. Call John Russell, (516) 562-5717.

**May 9-11**


**May 9-11**


**May 10-12**

LANDEX '90 Midwest - St. Charles, IL. The most focused industry event for LAN resellers, distributors and manufacturers. Call Cindy Froelich, (708) 279-2255.

**May 14-17**

*Expert Communication '90 - Austin, TX. Sponsored by Graphic Communications Association and Davis Review. This is where to come to hear about the ideas explored in this issue. With John Clippinger, Paul Doeble, Mills Davis, Steven Weinstein, Esther Dyson, others. Call Mills Davis, (202) 667-6400, or Patty Hill, (703) 841-8160.

**May 16**


**May 21-22**

LAP & PALMTOP '90 - New York City. Call Peter O'Connor at (212) 682-7968.

**May 22-24**

*Second annual executive UniForum symposium - Santa Barbara, CA. Sponsored by Patricia Seybold's Office Computing Group with UniForum and X/Open. "The applications development environment of the 1990s: Can Unix set the innovation agenda? Now that the UNIX market is emerging, it needs its own conference without a trade show; this is it. Call Judith Hurwitz, (617) 742-5200 or (800) 826-2424.

May 22-25  Visualization in biomedical computing - Atlanta. Sponsored by Georgia Institute of Technology and Emory University. Co-chair: Ed Catmull of Pixar. With topics such as "Trends in molecular modeling," "Chaos and fractals in electroencephalography," and "Visualization and man-machine interaction in clinical monitoring tasks," and "Linking a relational database of biological features to computer-aided reconstruction of tissue." Contact: Mary Simmons, (404) 894-3964.

May 28-June 1  Avignon '90 - Avignon, France. Tenth international workshop on expert systems and applications. Sponsored by ARC, ECCAI and JSAI. The major European expert system event. Call Jean-Claude Rault, (331) 47 80 70 00, or fax, 47 80 66 29.

June 3-6  Spring Comdex - (back in) Atlanta. Sponsored by the Interface Group. Call Elizabeth Moody at (617) 449-6600.

June 3-6  *ADAPSO annual spring conference - Washington, DC. With an international flavor this year. Call Ellen Kokolakis at (703) 522-5055.


June 6-12  Computex '90 - Taipei, Taiwan. Sponsored by Taipei Computer Association and The China External Trade Development Council. Call (886 2) 725-111 or fax (886 2) 725-1314.


June 14-17  *International Computer Club inaugural conference - Moscow. Scheduled to lure people east from the SPA event in Cannes. For information, call Esther Dyson at (212) 758-3434 or Levon Amdilyan in Moscow at 921-09-02.


June 19-21  PC Expo - New York City. Targeted at multi-unit buyers of pc products. Sponsored by H.A. Bruno. Contact: Steve Feher, (201) 569-8542 or (800) 444-EXPO.

Release 1.0  16 March 1990
June 19-21  

June 24-28  
Design Automation Conference - Orlando, FL. Sponsored by IEEE and ACM groups, for vendors and users of design tools. Call P.O. Pistilli, (303) 530-4333.

July 2-6  
ACM symposium on parallel algorithms and architectures - Crete, Greece. Contact: Tom Leighton, (617) 253-3662.

July 10-16  
*PC World Forum - Moscow. Sponsored by IDG. An exposition, with a software development conference. Contact: Frank Cutitta, (508) 879-0700, or Karin Griffhorn in West Germany at (49) 893 60860.

July 16-18  

July 27-29  

July 29-August 3  
*AAA-90 - Boston, in the heart of AI-land East. Sponsored by the American Association for Artificial Intelligence. Contact: Claudia Mazzetti, (415) 328-3123.

August 6-10  

August 13-17  
International parallel processing conference - St. Charles, IL (25 miles from O'Hare). Sponsored by Pennsylvania State University. Contact: David Padua, (217) 33-4223 or Benjamin Wah, (217) 244-7175, or Roger Anderson, (415) 422-8572.

September 5-7  
*Breakaway 90 - New Orleans. Sponsored by ABCD. With a panel featuring Mike Shabazian, Mike Pickett, Mike Swavely, moderated by Esther Dyson. Contact Jeff Rosenberg, Computer Emporium, (914) 565-6262.

September 9-12  
18th mini/microcomputer industry conference - Boston, MA. Sponsored by Gowen & Co. Contact: Amy Burns, (617) 523-3221.

September 10-13  
NetWorld '90 - Dallas. Sponsored by H.A. Bruno. Call Annie Scully or Mark Haviland, (201) 569-8542 or (800) 444-EXPO.

September 23-25  
*Agenda 91 - Laguna Niguel, CA. ...enhanced by a great new place. Sponsored by P.C. Letter/PCW Communications; staged by Stewart Alsop. Call Tracy Beiers, (415) 592-8880.

September 25-27  
International expert systems conference and exposition - London. Focusing on mainstream concerns: applications, integration, reliability and quality. Sponsored by Learned In-
formation. Contact: Jean Mulligan, 011 (44 867) 730275 or fax (865) 736354.

September 25-27  PC Expo - Chicago. Sponsored by H.A. Bruno. Contact: Steve Feher, (201) 569-8542 or (800) 444-EXPO.


October 3-5  Seybold Conference - San Jose. Electronic publishing in all its guises. Call Kevin Howard, (213) 457-5850.


October 7-10  *CSCW '90 - Los Angeles. Computer-supported cooperative work, with a slight (but lessening) academic flavor. Sponsored by ACM. Call Frank Halasz (back at PARC after a tour at MCC) at (415) 494-4750, or Tora Bikson, (213) 393-0411.


October 18  Massachusetts Computer Software Council’s fall membership meeting - Boston. Call Joyce Plotkin at (617) 437-0600.

October 21-24  **EDventure East-West High-Tech Forum - Budapest, Hungary. By popular demand. Explore the problems and opportunities of high-tech business in Eastern Europe and meet your peers in a limited-attendance conference focused on contacts, not speeches. Sponsored by EDventure Holdings, with a roster of speakers and attendees from both sides. Call Daphne Kis, (212) 758-3434. By invitation only.


October 31-November 2  UNIX Expo - New York City. Sponsored by National Expositions. Contact: Don Berey, (212) 391-9111.

November 4-7  *ADAPSO management conference - Phoenix. Contact: Ellen Kokolakis at (703) 522-5055.

Release 1.0 16 March 1990


November 7-9  Software Development Fall '90 - Boston. Sponsored by Miller Freeman. Call Lynne Mariani or Angela Hoyte, (415) 995-2471.


November 14-16  Pacific Rim international conference on artificial intelligence '90 - Nagoya, Japan. Sponsored by the Japanese Society for Artificial Intelligence. Special sessions for AI in engineering and AI and large-scale information. Call Shigero Sato at (813) 479-5535 or fax (813) 479-7433.

December 5-8  *CASE '90 - Irvine, CA. The fourth international workshop on computer-aided software engineering. Sponsored by Index Technology, IEEE and several academic institutions. Call Ron Norman, (619) 594-3734.

1991


March 3-7  *Seybold Seminars '91 - Boston. Call Kevin Howard, (213) 457-5850.

March 10-13  **EDventure Holdings PC (Platforms for Computing) Forum - Tucson, AZ (again). Sponsored by us! Contact: Daphne Kis, (212) 758-3434.


Please let us know about any other events we should include.
-- Denise DuBois

*The asterisks indicate events we plan to attend. Lack of an asterisk is no indication of lack of merit.

Release 1.0  16 March 1990
SUBSCRIPTION FORM

Please enter my subscription to **Release 1.0** at the rate of $495 per year in the U.S. and Canada. Overseas subscriptions are $575, airmail postage included. Payment must be enclosed. Multiple-copy rates on request. Satisfaction guaranteed or your money back.

Name_____________________________________________________

Title ______________________________________________________

Company ___________________________________________________

Address_____________________________________________________

City_________________________ State_________ Zip ____________

Telephone _________________________________________________

How did you hear about Release 1.0? _____________________________

____________________________________________________________

Please fill in the information above and send with your check payable to: EDventure Holdings Inc.

375 Park Avenue, Suite 2503

New York, NY 10152

If you have any questions, please call us at (212) 758-3434.

Daphne Kis
Associate Publisher

Release 1.0

16 March 1990