The future of specialized languages

by F. B. THOMPSON and B. H. DOSTERT

California Institute of Technology
Pasadena, California

INTRODUCTION

The prediction of the future, like the review of the past, has as its sole usefulness, the organization and purposeful direction of the present. Thus, in situations where there is rapid change, as there certainly is in the field of computing, prediction—though more difficult—is all the more necessary. Suppose that you, as a computer specialist, are considering whether to design and implement a language for your own special application area or to consolidate your programming within the framework of a single, general language. You will certainly want your decision to be responsive to, though perhaps not dictated by, your perception of the directions computers and computing will be taking.

Our own perception concerning programming language development rests on two considerations. The first consideration is to understand the nature of the pressures that will bring about change. New languages for specialized application areas don’t just happen; they arise because of a felt need, a dissatisfaction with trying to write programs for a specialized domain using programming languages not specifically tailored to that domain. Our first task, then, is to understand the essential nature of this need.

The second consideration on which our perception of future developments rests is to assess the technological and economic restraints and resources for change. Big computers, batch processing, and further professionalization of programmers will stimulate certain trends in programming language design; mini-computers, interactive processing, and widespread knowledge of programming will stimulate different trends. Thus, our second task will be an assessment of the major developments in hardware and software, and of the economic environment within which the pressures for change will be resolved.

These considerations should then put us into a position where we can bring into some focus what to expect in language development for specialized application areas. The past and current situation has been reviewed by Jean Sammet in *An Overview of Programming Languages for Specialized Application Areas*. Our final task, therefore, is to identify the major trends these developments are likely to take in the future.

SOURCES OF PRESSURE FOR LANGUAGE DEVELOPMENT

In talking about languages for computing, we should first like to consider certain aspects of the nature of language. To do this it will be useful to develop an analogy between a language, on the one hand, and the design engineer’s laboratory, on the other. Imagine yourself to be an electrical engineer designing a new computer central processing unit. As your design progresses, you and your team actually construct a model or “breadboard” out of flip-flops, amplifiers, “and” gates and other components. You wire these together often using clips rather than solder, because as you try out your design, you may wish to modify it as your breadboard model reveals unexpected relationships and possibilities for improvement. This model has a second important function; it expresses in a most articulate way the sum total of ideas contributed by the various members of the design team and is thus a vehicle for communication between them.

The design engineer’s laboratory provides a great variety of basic components and the means of hooking them together into meaningful arrays. As the design proceeds, selection is made of certain components and they are built into combinations that express the functions that the designer wishes to execute. The same is true of a language. A language is based upon a great variety of basic concepts and its syntax is the means of hooking them together into meaningful arrays. As a programmer proceeds, he selects certain components, building them into combinations that express the functions that he wishes to execute. This then is the analogy...
between design laboratory and programming language we now wish to exploit.

Suppose you were called upon, as a computer specialist, to develop the configuration of a new computer for your installation. You would have to choose the main CPU, the size of high speed memory, what peripherals you would want, etc. Suppose you were told that the components you had at your disposal were diodes, resistors, capacitors and the like and that you had to build your computer installation from this point up. You would surely go about your job far differently than if you had been provided as building blocks the major pieces of computer equipment available from computer manufacturers. Consider these two situations carefully. If you had diodes and resistors you would have to be a different kind of engineer than if you had disk drives and printers to work with; your design would take considerably longer, and your expectation, design considerations and final output would be distinctly different. Our activities, the direction of our thinking, our efficiency, and our results are all greatly influenced by the basic components with which we must work and the means at our disposal for building them into more meaningful structures.

These same observations apply with equal force to the languages we use in dealing with our various problem areas. Language is the principal tool of our conceptual processes. And different styles of thought reflect and are reflected in the languages we use. Notice the correlation between artificial intelligence and LISP; engineering mathematics and FORTRAN or PL/1; operating systems programming and assembly language. This notion that language conditions the modes of thought is not new. Benjamin Whorf, the MIT chemist-turned-linguist, expresses the matter in the following words:

“We dissect nature along lines laid down by our native languages. The categories and types that we isolate from the world of phenomena we do not find there because they stare every observer in the face; on the contrary, the world is presented in a kaleidoscopic flux of impressions which has to be organized by our minds—and this means largely by the linguistic systems in our minds.”

Whorf was referring, of course, to natural languages. But surely there is every reason to suspect that the programming languages we use will also have an effect on the kinds of problems and the character of the programs to which we apply the computer.

To illustrate this interdependence, consider the relation between languages and the kinds of management information systems that are currently in vogue. The original development of concepts for dealing with business data came at a time when punch cards and magnetic tape were the only large size repository for machine readable data. Thus the whole nomenclature of sequential files, fields, records, file update, report generation, etc., became an integral part of the language of data processing and the conceptual environment for the development of COBOL. Now we find that the inertia resulting from these common language concepts inhibits the acceptance of new systems based upon random access to data and time shared environments. A whole generation of programmers, information systems designers, and middle management people have come to think of systems for formatted file management and report generation as coextensive with management information systems, even though such systems are a travesty on what management wants and needs.

If Whorf’s hypothesis is true, it can be turned around and applied in reverse. We, ourselves, build the artificial languages for programming computers. As language builders, we will have significant effects on those who use them. Thus, when we apply this Whorfian notion to artificial languages we ourselves can build, a new perspective opens up. The computer is a uniquely powerful instrument for conceptual design and structuring. The “special application” languages we provide are the major interface between the “designer” and this powerful design laboratory. The conceptual elements and syntactic tools made available to him cannot but influence the effectiveness, even the direction, his work will take.

Who is this “designer” in the case of the computer? He is the industrial manager who is seeking to direct the intricate interactions of product lines, production capabilities, markets and finance. He is the anthropologist searching for those causal laws of human behavior that lurk within his voluminous data on family relationships, vocational histories and patterns of ownership. He is the test engineer whose task is to chart the strengths and weaknesses of an aircraft wing from the thousands of data points of a destructive test procedure. He is the government agency head whose response to public need is dependent on his sensitivity to changing conditions as reflected in census statistics and spot surveys. Can we indeed say that all of these are well served by FORTRAN or PL/1? Is it not obvious that the basic conceptual elements of each of these diverse areas of computer application are themselves equally diverse, each with their own logical interrelationships and implicit assumptions? This logic and these assumptions can be built once and for all into the software support of a special application language. They can exist at that tacit level which is natural for the given application area and free the creative mind of the manager, researcher or engineer to build his pro-
grams on a conceptual platform appropriate and natural for his concerns.

We would like to illustrate this point by a reference to experience in the area of theorem proving on the computer. The resolution strategies of Robinson and their refinement by a number of researchers has progressed to the point where interesting applications can be made to highly restricted areas such as the control of robots that move blocks. However, application of theorem proving to problems of everyday business appears to be a long way away. One of the reasons for this is the necessity, if theorem proving is to give realistic results, to describe in specific axioms the very large number of tacit assumptions that are implicitly understood in these areas. Using explicit statements of these assumptions turns out to be very much more expensive in processing time than building them implicitly into heuristics and procedures. Think of the task of stating all of the tacit assumptions that underly just the personnel files of a modern business. But these same assumptions, known implicitly to the applications programmer, are built into the procedures for processing that data. To be sure, these procedures are specific to the application area. Indeed it is just such idiosyncratic procedures that become embodied as the interpretive routines of a language which is “natural” to such a specialized area. An example is the inclusion of efficient differential equation solving algorithms in applied mathematics systems such as NAPSS algorithms that are automatically invoked in response to natural syntax for the expression of such an equation. From there on, they are no longer conscious considerations in further program writing, or processed internally as axioms. They are now implicit, pressed down below the level of the explicit considerations of the researcher or manager who is using his own natural language. There are two advantages here. One is the advantage that accrues to the user in the efficiency of his language in dealing with matters that are directly of his concern. The second is the advantage in the underlying processing algorithms that make use of the underlying implicit assumptions of the domain. These enormous economic advantages that the computer can thereby put at the fingertips of the specialist in his area are the true source of the pressure for special application languages.

THE PRESENT RESOURCE ENVIRONMENT

When the computer was the scarce resource and programming was the domain of a small community of professionals—and indeed these conditions still residually apply—the powerful multipurpose language was the natural tool for man/computer communication. But this situation has been changing. We will examine four significant developments in this regard.

Training of computer scientists

In the first place, our universities are turning out a swelling stream of graduates in each of the various professions who, over and above their professional specialty, are also knowledgeable in computing. Our major schools, and even many smaller institutions, include as an integral part of their professional curriculum creditable courses in programming and computer systems. So far, most of these are limited to a few widely used algebraic languages such as FORTRAN. However, more and more, specialized languages are being taught: COBOL in business schools, LISP in psychology, SIMSCRIPT in industrial management, various applied mathematics languages, such as NAPSS, in areas of engineering, data analysis languages, such as SPSS, in the social sciences.

More important than the large number of students taking these computing courses is that much smaller minority in the various disciplines who take the more advanced courses in the computer science curriculum. They go into their professional areas fully competent to bring to their disciplines the full power of the computer, and they are highly motivated to do just that. It is these cross disciplinary people who will spawn the new languages for their specialized application areas. They will know the conceptual elements and implicit logic essential to their substantive discipline and they will also know how to embody these within efficient programs and with linguistic sophistication reflecting their training in computer science.

Let’s review a case history in this regard. Dr. Harry Markowitz received his doctor’s degree in economics, concentrating in areas which required a high degree of knowledge of mathematics and computing. At the RAND Corporation his work involved the development of some economic models and their implementation on the computer. He then became one of the leading scientists in the RAND logistics project where he was instrumental in the development and application of simulation techniques. Subsequently Markowitz worked within the executive office of the General Electric Company, applying digital simulation to manufacturing and corporate problems. As he developed his simulation programs, his style and program organization became clearer and more modularized. Also, the scope of application of his work grew. Upon returning to the RAND Corporation, it was natural for him to distill from all of the various simulation programs he had written and supervised, a general technique that was...
widely applicable. The result was one of the first and still one of the best discrete simulation languages, SIMSCRIPT. The number of such able, interdisciplinary people need not be large to make a significant impact, and both the number and quality are growing.

Advances in hardware

The second development we would like to cite is in the hardware area. The price of computers and peripheral equipment continues to come down. However, as computer costs drop, not only is the market widened, but the experienced user can afford to shift more of his task onto the computer and thus demand that the computer do more for him. One form of this extra service is to move the man/machine interface closer to the man. One of the principal ways this can be accomplished is through the development of languages that are more natural for the various application areas.

However, more significant than the general downward trend of cost per cycle time is the advent of the mini-computer. Many project teams in industry, in the university and other research institutions, in all walks of life are now finding they can afford their own computer. Up to now the main market for mini-computers has been in areas of process control or in connection with special input devices and sensing equipment. Those applications of computers which are not tied directly to special sensors or control devices but which require complex application programs have tended to gravitate to central computing centers with their large programming staffs and facilities for batch processing the many debugging runs that characterize application software development. But the cost advantage of one's own mini-computer which avoids the growing overhead costs of the big computer centers will tend to reverse this. A preliminary look indicates that very high level conversational languages can be implemented as dedicated mini-computer systems.

The development of effective programming languages for specialized application areas can be instrumental in the opening up of sizable markets for the mini-computers. As these versatile devices are brought to bear on the sophisticated problems of competent professionals, with their small group of captive programmers, we can again and again expect to see the evolution from special application programs, to module libraries, to monitors, to special application languages. Once such languages are developed, they stimulate similar installations elsewhere. The heavy attendance at specialized area sessions at computer conferences attest to the interest in learning about just such developments. Because of the low cost and considerable capability of the mini-computer, one can expect a general shift toward single user installations with a corresponding increase in independence and innovation. The result cannot help but have a considerable effect on the development of languages for specialized application areas.

Developments in systems programming

The third general area of development is that of systems programming of which extensible programming language research is a part. Here the rapid growth of interest and importance of extensible programming languages is a key development that will have a profound effect on the proliferation of specialized application languages. To see this, one needs to be aware of the nature of the problems that those working in the area of extensible languages are attacking. The deep problems of this domain have to do with the building of optimizing compilers. It is no great trick for the experienced systems programmer to build a programming language that provides for complex structures declarations, at least when he can implement these without consideration of run time efficiency or storage management. The real problems are how to achieve optimization of both storage and computing efficiency. Important inroads into this difficult and central area are being made. We cite, for example, the work of Cheatham and his group at Harvard.

As work progresses on extensible programming languages, one of its primary applications will surely be to the definition of higher and higher level languages, languages that include more and more implicit structural logic, indeed languages that fit closely the specialized needs of significant application areas. The ability to produce such languages efficiently and at the same time to retain reasonable levels of optimization in the actual encoding provides a powerful tool for specialized language building. These specialized application languages need not be limited to domains outside computer science. The language Planner, developed by the Artificial Intelligence Group at MIT, is an excellent example of such a language to be used by computer scientists themselves. In this case it is built on LISP which is surely an example of an extensible language.

Science of linguistics

The fourth area of development is linguistics itself. We are indeed learning more and more about the struc-
tures of languages and the underlying reasons why language is such a powerful tool for conceptual structuring and communication. In particular, we are rapidly gaining sufficient knowledge of the mechanisms of natural language so that useful segments of natural language can be understood and responded to by the computer. Our own work on the REL system is a good example of where this is being successfully accomplished. In our system, we combine results from recent work in theoretical linguistics, namely modern deep case grammar as developed by Fillmore, with an efficient application of the notions of syntax-directed compiling. We believe we are only a very few years away from an English driven system for conversational data analysis with levels of efficiency that will markedly improve upon present formal language batch systems. Another system, that of Terry Winograd at MIT has demonstrated the ability to control intelligently the complex behavior of a robot in response to directions given in a comprehensive subset of natural English.

The importance of these types of systems does not lie in some magical quality of "English." There is a rather general understanding by computational linguists working on actual systems that "natural" languages are full of idiosyncratic notions and expressions that derive from the particular domain of discourse for which they are adapted. What is important about this natural language research is not the vocabulary, which is surely not a universal of the native fluent speaker. Rather the important insights from this research concern syntactic mechanisms and their interaction with semantics. We have referred to the need in a language not only for the right conceptual elements but for a sufficiently powerful syntax that will allow the efficient expression of interrelationships, as indeed is found in natural language.

Natural language has a variety of powerful mechanisms for combining conceptual elements. Ambiguity is an excellent example. Usually, when ambiguity in a language is mentioned, one thinks of ambiguous sentences. However, consider how ambiguity is involved in phrases, i.e., segments of sentences. In the phrase:

"the computer just moved into the laboratory"

the words "computer," "moved," and "laboratory" are essentially all ambiguous when standing alone in that they do not designate a unique piece of equipment, action or location. The phrase "moved into the laboratory" also could be used in many contexts where it would have quite different meanings. However, the above phrase, when taken as a whole and in context is not ambiguous at all. This ability to use general nouns and verbs in such a way as to balance off their ambiguous referents to achieve in a parsimonious way a totally unambiguous description is a powerful and ubiquitous tool of natural language, and one that is not difficult at all to include in computer languages where the context is delimited to a specialized application area. Thus, work on natural language will indeed go far in providing both understanding and specific techniques for building specialized application languages with sufficient expressiveness to truly serve as effective design laboratories for conceptual structuring.

Our growing knowledge of computational linguistics goes beyond knowledge of natural language to all areas of language communication. In particular in the domain of graphic languages and man/machine communications using graphic consoles, specialized man/machine communication languages have been and surely will increasingly be developed.

The rapid growth in linguistic knowledge will stimulate advances beyond the immediate effect of more effective computer languages and language systems. When a domain of human knowledge expands as linguistic knowledge is expanding today, it has far reaching consequences. The simulation of more effective systems built upon these new linguistic insights, insights that span from English syntax to psycholinguistics and the mechanisms of language acquisition, cannot help but be great. And greater understanding has always led to higher degrees of specialization, specialization that spells efficiency.

TRENDS IN PROGRAMMING LANGUAGES

The pressures and developments discussed in the previous two sections give evidence that programming languages for specialized application areas will continue to proliferate. Moreover, they imply several more specific trends. We shall identify and examine several of these.

As knowledge of linguistics grows and the application of computers to specialized areas continues, we realize that our old algebraic languages—FORTRAN, ALGOL, PL/I—are really quite special purpose after all. Certainly there is a need for good algebraic languages in physical science and engineering. However, other language developments, for example LISP, have already demonstrated the need for programming languages with quite different characteristics.

The attempt to include in PL/I a wide range of data structure declaration capabilities has led to an interesting and perhaps significant development. PL/I has not replaced FORTRAN for writing special application programs. It does appear that it is being used by some systems programmers and by programmers who
are writing languages for special application areas. The problem with using PL/I for these systems programming purposes is that the resulting code is not sufficiently optimized. But it is exactly in this area of optimizing compilers that progress of the greatest importance is being sought in extensible language research. The confluence of these developments will lead to a movement toward powerful system programming languages of the extensible type. They will be based upon our increasing understanding of the nature and importance of data structure.

The advent of these systems programming languages will mean that we will have greatly increased ability to create languages that are carefully tailored to the conceptual and structural needs of specialized areas. The mini-computer, the computer utility and the general lowering of computing costs are creating a ready market for such language developments.

Thus we foresee that systems programmers will be turning away from their traditional preoccupation with the architecture of, and compilers for, classical algebraic programming languages. System programming will turn more of its attention to the efficient implementation of a much wider class of language mechanisms and the building of these mechanisms into a far more diverse family of application languages. These developments will obviously be greatly influenced by the spreading acceptance of conversational, interactive computing.

A second major trend will be toward natural language systems. We emphasize that by natural language we do not mean the general English of some mythical native fluent speaker. By natural we mean computer languages which reflect the conceptual elements and the syntax of the application areas in which they are employed. These languages will be specialized languages, idiosyncratic, reflecting the physical as well as the conceptual environment in which they are to be employed. For example, they will be influenced by the kinds of terminals or displays to be used, whether they refer to very large files of data, whether the application area naturally employs advanced mathematics, the interactive reaction times required, etc.

There are two technical characteristics, however, which many of these languages for specialized application areas will share. First they will tend more and more in the years ahead to be problem definition languages rather than procedural languages. The distinctions between problem definition languages and procedural languages is extensively discussed in the computer science literature without definitive conclusions. Thus it will do no harm if we add to that discussion a distinction that we feel to be an important one. A procedural language talks essentially about the computer; statements in such a language are instructions to the computer about what it is to do. A problem definition language talks essentially about the domain of application; statements in such a language either describe, ask for descriptions or direct the computer concerning the subject matter of the application. It appears to us that higher level languages and languages for more highly specialized languages tend to be closer to problem definition than to procedural languages. We feel that there will be an increasing trend toward problem definition languages.

The second technical characteristic that we foresee is a trend toward natural language syntax. English as a programming language has been discussed for a good number of years and often fervently wished for but thought of as something for the distant future. Through these same years solid progress has been made—in theoretical understanding of linguistic structure, in computational linguistic practice, and toward operational English language systems. Pragmatic work on machine translation has been showing practical results, contrary to some expressed opinions. As a result, the next few years will see operational systems for specialized areas where the structure of the language is closely related to our native English. Once this capability has been conclusively demonstrated, prejudice against it will be forgotten and we will make use of the powerful syntactic mechanisms of natural language in many application areas.

SUMMARY

Return for a final look at the Whorfian hypothesis, that language shapes the thought and culture of those who use it. As we develop powerful systems programming languages for application language development, as we incorporate the power and expressibility of natural language syntax into our application languages, as the economic costs of hardware and language software come down, and particularly as our rapidly expanding knowledge of linguistics continues to grow, a new dimension in artificial language development will come into being. We will recognize that language design is a powerful means of direction and control. The tasks of professionals can and will be directed by the languages they must use in communicating with their computers.

REFERENCES

1 B L WHORF

Language, thought and reality

MIT Press 1964 p 213
2 J S ROBINSON
The present state of mechanical theorem proving
Proc of Fourth Systems Symposium 1968

3 C GREEN B RAPHAEL
The use of theorem proving techniques in question answering systems
Proc 23rd Nat Conf of ACM 1968

4 L SYMES R ROMAN
NAPSS: Syntactic and semantic description for the numerical analysis programming language
Comp Sci Dept Purdue Univ 1969

5 H MARKOWITZ B HAUSNER H KARR
SIMSCRIPT: A simulation programming language
The RAND Corp Santa Monica 1963

6 N NIE D BENT C H HULL
SPSS: Statistical package for the social sciences
McGraw-Hill 1970

7 B WEGBREIT
The ECL programming system
Div of Eng and App Phy Harvard 1971

8 C HEWITT
PLANNER: A language for proving theorems in robots
Proc of Internat Joint Conf on Arti Intell 1969

9 F THOMPSON P LOCKEMAN B DOSTERT R DEVERILL
REL: A rapidly extensible language system
Proc 24th Nat Conf of ACM 1969

10 B H DOSTERT F B THOMPSON
The syntax of REL English
REL Project Calif Inst of Technology 1971

11 C J FILLMORE
The case for case
Universals in Linguistic Theory ed E Bach and R Harms
Holt Rinehart and Winston 1968

12 J E SAMMET
Programming languages: History and fundamentals
Prentice-Hall 1969 p 20-22 p 726

13 J E SAMMET
The use of English as a programming language
Comm of ACM 9 pp 228-230 1966

14 F B THOMPSON
English for the Computer
Proc FJCC pp 349-356 1966

15 T WINOGRAD
Procedures as a representation for data in a computer program for understanding natural language
Project MAC MIT 1971

16 C KELLOGG J BURGER T DILLER D FOGT
The converse natural language data management system
Proc of Sym on Info Stor and Retrieval Univ of Maryland 1971

17 W A WOODS
Transition network grammars for natural language analysis
Comm of ACM 13 pp 591-606 1970

18 S PERSCHKE
Machine translation, the second phase of development
Endeavour 27 pp 97-100 1968

19 J SAMMET
An overview of programming languages for specialized application areas
To be published in Proc SJCC 1971

20 SIMSCRIPT II.5 handbook
California Analysis Centers Santa Monica 1971

21 Proc of ACM Sigplan Conference on Extensible Languages Sigplan Notices 6 #12 December 1971

22 M D HALPERN
Foundations of the case for natural-language programming
Proc FJCC 29 1966