TOWARDS A WESTERN FIFTH-GENERATION COMPUTER SYSTEM PROJECT

by

Maarten van Emden

Logic Programming Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Research Report CS-84-14
June 1984

To appear in the Proceedings of the ACM Annual Conference
San Francisco, October 1984

TOWARDS A WESTERN FIFTH-GENERATION COMPUTER SYSTEM PROJECT

Maarten van Emden

Logic Programming Group
Department of Computer Science
University of Waterloo

ABSTRACT

A Fifth-Generation Computer System (FGCS), as usually understood, requires breakthroughs in artificial intelligence and parallel processing. We identify "near-term" FGCS's relying only on existing developments in hardware, user interfaces, and software techniques integrating databases and a wide range of programming paradigms to achieve much of the social utility expected of FGCS's of the long-term variety. We identify these developments and argue that logic programming is too good to be left to the Japanese: that it is an economical basis for a Western near-term FGCS project.

1. Introduction and summary.

In 1981 the Japanese government inaugurated [15] a project to develop a "Fifth-Generation Computer System" (FGCS). Such a system features, among other functions,

a) voice input using natural language

b) learning, associating, and inferring.
Of course, rudimentary forms of these functions are feasible at the present time. To give an idea of the scope invisaged for inclusion in an FGCS we quote Moto-Oka [15]: "With such abilities, computers would be able to clarify even vague requests given by man."

c) natural language translation.

The FGCS project is centered at ICOT (for "Institute for New Generation Computing Technology"), a newly formed organization whose researchers are mainly on loan from established industrial and governmental laboratories.

From soon after its announcement the Japanese project has caused a considerable furore all over the world which has only recently begun to show signs of subsiding. The ambitious level of its goals have been the target of much comment. In addition, government and industry in the UK, the US, and the European Communities have responded by initiating special funding programmes for research and development in computing technology.

This paper serves to guide those concerned with these developments. Brief summaries of the successive sections of the paper follow below.

What is new?
It is not clear to everyone what is special about the situation we find ourselves in: haven't we had a computer revolution ever since ENIAC?

Near-term FGCS's are easier than many believe.
The ambiguity of the notion of FGCS gives rise to a great deal of misunderstanding. There are in fact two different interpretations: a near-term version, taking up most of the effort currently expended by ICOT, and a long-term version getting most of the attention. It is the long-term version that requires several breakthroughs in parallel processing and artificial intelligence. The near-term version requires only development, no breakthroughs. Yet the power of near-term FGCS's is sufficient to carry the computer-based revolution that many are anticipating. Another topic about which confusion exists is the role played by Artificial Intelligence research in realizing FGCS's. We argue that no AI research is needed for near-term FGCS's.

The technical basis of the Japanese FGCS project is important.
The Japanese FGCS project has received plenty of attention, but its technical basis (logic programming and Prolog) has been largely ignored. We argue that this choice of technical approach is not a whim, but has important advantages for a near-term FGCS project.

The West should have an FGCS project.
When considering the question whether the West should have FGCS projects, the distinction between near-term and long-term FGCS's is crucial. The currently prevailing funding structure in the US (NSF, ARPA) is suitable for research leading to long-term FGCS's. It is here that AI plays an important role; probably not more, nor less important than hardware.

For near-term FGCS's, however, the existing funding structure is entirely inappropriate. This is the area most important to the medium term economic prospects of the West. To develop a near-term FGCS, a mission oriented, closely managed project is essential.

2. What is new?

There are several developments in computing technology that have proceeded up till now more or less independently. They have, however, the potential of converging and interacting in a powerful way.

- VLSI design is rapidly becoming sufficiently automated to make custom hardware more widely available. Thus it will cease to be necessary to squeeze all of computation into the straightjacket of convential architecture.
- Database technology: the relational data model has permitted the application of first-order predicate logic as a conceptual tool, allowing at last a unification of database modelling with knowledge representation in artificial intelligence [2]. Another important development is the emergence of "application generators": basically database management systems with some computation and report generator facilities added (MAPPER, NOMAD; see [14]). This very limited unification of database with computation already has great practical impact [14].
- Advances in relational programming (Prolog) and functional programming (Lisp machines).
- Expert systems research has contributed the "Conceptual Interface", which makes it possible to communicate with software using the common sense concepts of facts, questions, answers, rules, and explanations [8]. Although it has only been applied in expert systems so far, there is great potential in allowing nonprogrammers to communicate effortlessly with all software without requiring further advances in natural language processing.

3. Near-term FGCS's are easier than many believe.

The Japanese FGCS project has two components: a near-term one culminating in PSI, the Personal Sequential Inference machine, and a long-term one aiming at an FGCS with all the features usually attributed to FGCS's. It is the near-term project that currently gets most of the resources, while the long-term aims get most of the attention. The Japanese authors [15] of papers on the aims of the FGCS project seem to suggest, and Western observers seem to agree, that the revolutionary impact of FGCS's in society will only happen through long-term FGCS's.

This is an important point, and we disagree on it: near-term FGCS's will give society all the revolution it will be able to stand for some time. The Conceptual Interface, mentioned above, can give a large segment of the nonprogramming population access to computers, even when using a conventional terminal as I/O device. It is true that serious difficulties exist in knowledge representation and knowledge acquisition, but these difficulties only stand in the way of large scale proliferation of expert systems with hard expertise.

But no such difficulties stand in the way of what we call "mundane expert systems", where the expertise is not difficult to obtain, but nonetheless such that its computerization has profound economic effects [8]. This kind of expertise exists on a large scale in administrative law, regulations of all kinds existing in institutions, companies, government agencies, and so on; in short, the kind of thing that keeps a large part of white collar workers busy. Mundane expert systems will be able have an enormous impact in this part of the economy, which has so far resisted attempts at automation.

A third reason for believing that near-term FGCS's will have a very strong effect, is that mundane expert systems with the Conceptual Interface make only moderate demands on computing power. All that is needed is the LISP machine analog for Prolog. And this is just the role played by PSI, ICOT's personal sequential inference machine. In December 1983 ICOT has taken delivery of the first PSI prototype from Mitsubishi. By the end of 1984 ICOT expects to have the system software completed.

4. The Japanese project should be taken seriously.

Feigenbaum and McCorduck [7] take an alarmist point of view: they argue that, unless massive efforts are undertaken in the US, the Japanese will soon dominate information technology, the commanding heights of the economy of the future. It would seem to behoove an adherent of this opinion to pay some attention to the technical basis of the Japanese FGCS project. But that is not done; whenever there is mention of these topics, they are dismissed with a few disparaging remarks. One cannot have it both ways: if indeed the Japanese made a mistake in basing their project on logic programming and Prolog, then there is no reason to consider their project an economic threat.

We believe that the Japanese were right in their choice of technical approach. This does not imply that they will be successful. The circumstances of the FGCS project are completely different from those of the successes they have achieved in the past.

What is so great about logic programming? It offers important advantages for the nonprogramming user as well as for the systems developer. For the user it is important that the Conceptual Interface is inherent in logic programming; it is not something that has to be simulated at another level.

The advantages for the systems developer are that the major paradigms of computer use are all aspects of the same kernel concept of Horn clause logic programming. This kernel is conceptually simple; implementation studies have demonstrated feasibility, with modest resources.

The paradigms are:

- Relational databases.

  Roughly speaking, the relational database model is a subset of logic programs. See [10,12] for a detailed account of the relationship.

- Recursive processing of lists and trees.
  This was the first application of logic programming and it is still the best developed part of Prolog implementations. See [3,6] for this aspect of logic programming.
- Programming with loops and flowcharts.
  This is a very poorly developed aspect of logic programming. It is so for a good reason: conventional languages take care of this. But it is important to realize that this paradigm is also covered by logic programming [5] so that it is not necessary to switch to a different system should this paradigm be needed.
- Concurrent programming.
  Certain versions of Prolog [4,17] have been designed giving a formalism for concurrent programming similar to, but more powerful than, Hoare's CSP.
- Object-oriented programming.
  Shapiro's version of concurrent Prolog [17] has been shown to be an attractive formalism for object-oriented programming [18].
- Metalevel programming.
  Part of the charm of LISP is in the recursive processing of lists. Another important part is its facility for metalevel programming: the ease with which functions produce functions, etc, to any desired number of levels. In implementation this aspect of logic programming is not yet well developed. Conceptually, however, the proposal of Bowen and Kowalski [1,12] for amalgamating object and metalevel in logic programs gives an extremely powerful way of providing metalevel programming and several other facilities.
- Functional programming.
  Functional programming in its wider sense emphasizes metalevel programming and recursive processing of list and trees (see above). In its narrower sense it means the use functional notation. This facility can be provided in Prolog by a simple front end, as shown in [3].

For the systems developer the availability of all these paradigms is important. How can all these be made available?

We believe that logic programming is the *only* way. Currently a well-equipped computer centre will be able to run one or more systems implementing each paradigm. But no programmer uses them all: he is deterred by the mass of documentation and the exasperating way in which the common features are handled differently. The Ada experience has shown how hard it is to design and implement a much narrower spectrum of paradigms in a single language when that language is not based on a powerful unifying principle. Logic programming is such a principle: all the above paradigms are minor variations on a single compact kernel. It is the only way.

5. The West should have an FGCS project.

When considering the question whether the West should have an FGCS project, the distinction between near-term and long-term FGCS's is crucial. For long-term FGCS's breakthroughs are needed, but not for near-term FGCS's. Therefore, though it may be possible to hurry up the long-term FGCS's to some extent, it cannot be predicted when these will be achieved. The appropriate mode of government support in an area where breakthroughs are needed is the present mechanism for funding scientific research, whether this is in computing, or in chemistry, or whatever.

Near-term FGCS's need no breakthroughs; only development is needed. And the economic prize is great. Therefore there is a strong case for an extraordinary initiative to fund a special project to develop such a system. It is essential that such a project be mission-oriented and tightly focussed (like the Japanese project). None of the Western responses so far (ESPRIT, the UK result (does it have a name?) of the Alvey committee report, and MCC in the US) are in this category.

A Western near-term FGCS project should be different from its Japanese counterpart in several interesting ways: it should

have a more specific goal
: The Japanese project is only specific in its short-term goal, the Personal Sequential Inference machine. Its long-term goal is necessarily vague. The Western project proposed here should aim for a specific demonstration system. See next section for a suggestion.

be mainly university-based
: Unlike Japan, the West has in many of its universities a strong capability for software development. Our experience suggests that a significant part of the manpower can be supplied by beginning graduate students.

be distributed rather than centralized
: Experience suggests that a suitable computer mail facility supplemented by a face-to-face meeting a few times a year is almost as good as working on the same campus.

6. A suggestion for near-term FGCS project

We consider successively the following questions about the project: "What will it do?", "How will it be done?", and "How much will it cost?". Although the answers below are neither detailed nor definitive, they should provide a useful starting point for further discussion.

6.1. What should it do?

A near-term FGCS should both make currently available software more accessible as well as open up new areas of application. To explain what we mean by the problem of accessibility of currently available software, let us consider the computer aids in principle available to, for example, an engineer.

In his daily activities he makes calculations, searches tables, standards, textbooks, drafts reports, receives and sends mail (and other documents), retrieves and studies drawings and textual library material, and so on. In all these application areas separately, computer programs exist to provide powerful aids. The rapidly falling hardware cost makes these programs potentially accessible to every engineer. But this is happening only slowly and in a piecemeal fashion: if an engineer is to utilize the full potential of all available computer aids, he would need half a dozen software experts available at a few seconds notice.

Even then, he will not be able to use the computer as a truly congenial tool: that is only possible when no intermediary is needed. Currently he needs an intermediary for most applications because of the complex and idiosyncratic interface provided by most software. Or he becomes a computer "hacker" himself and stops being productive in his own area. The situation sketched here holds not only for engineers, but also for professionals in government, business administration, and scientific research.

Part of the proposed project is to use the expert systems user interface to make this software available to the nonprogramming user as a congenial tool without the need for a human intermediary. Such use of expert systems is suggested by the SACON expert system.

Expert systems are also the key in opening up new application areas. An expert system like MYCIN has achieved two valuable goals: it has developed the Conceptual Interface (a new interface for user as well as for the system developer) and it has automated a nontrivial expertise. It is often overlooked that the application of the same software methods to less challenging areas of expertise is extremely valuable. An example is the development by Hammond and Pickup of an expert system for supplementary benefits of the British Social Security system. This was completed in five days (civil servants' working days). Indeed quite untypical. The reason was that the rules were not hard to elicit from the experts: they knew them already and could even find them in the text of the relevant government regulations.

But, although this application is not impressive from the point of view of the researcher in expert systems, it is a great improvement compared to software development by conventional methods. There are many applications in the same category: hard to do conventionally, but trivial as expert system.

6.2. How will it be done?

It will be done by logic programming, because it has the advantage that the expert systems interface is natural to it and because it is the way to make all paradigms of computer use available to the systems developer (see above).

6.3. How much will it cost?

Let us assume that machinery in the form of suitable Prolog machines is supplied. If not, then the project will have as additional part one that develops such a machine. We will not try to say anything about the cost of this part. The develpment of LISP machines should provide relevant experience.

What remains is the software development component. The work of the Logic Programming Group at Imperial College has suggested many ideas on which this proposal is based. Not only the group's results, but also its operation as a group can give useful information: the group itself can be regarded as the seed of a near-term FGCS project. Without such a model it is hard to find a starting point for discussing a suitable size of project.

Not counting the part of the group engaged in the Alice functional programming machine, there were in 1982-1983 three workers with University appointments who were therefore working only part time on logic programming. In addition there were about eight full-time grant-supported workers. This is the starting point. The first step is to imagine that the Imperial College group were all full-time and that their work were more focussed (namely on a near-term FGCS) rather than the way it actually was. What would be the resulting percentage of the effort required for development of a near-term FGCS in a suitable time period? Let us say it would be between 30 and 50%.

This group suggests impressive leverage for a near-term FGCS project: a great effect at a modest cost.

7. References

[1] K.A. Bowen and R.A. Kowalski: Amalgamating language and metalanguage in logic programming. pp 153-172 in "Logic Programming", K.L. Clark and S.A. Tarnlund (eds.). Academic Press, 1982.

[2] M.L. Brodie and S.N. Zilles (eds.): Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling, SIGPLAN Notices Volume 16, Number 1 (Jan 1981).

[3] K.L. Clark and F.G. McCabe: micro-Prolog: Programming in Logic. Prentice-Hall, 1984.

[4] K.L. Clark and S. Gregory: PARLOG: a parallel logic programming language. Research Report DOC 83/5, Dept. of Computing, Imperial College, 1983.

[5] K.L. Clark and M.H. van Emden: Consequence verification of flowcharts. IEEE Transactions of Software Engineering, vol. 7 (1981), pp 52-59.

[6] H. Coelho, J.C. Cotta, and L.M. Pereira (eds.): How to solve it with Prolog, 2nd edition. LNEC, Lisbon, 1980.

[7] E.A. Feigenbaum and P. McCorduck: The Fifth Generation. Addison Wesley, 1983.

[8] P. Hammond: micro-Prolog for expert systems. pp 294-319 in [3].

[9] R.A. Kowalski: Predicate logic as a programming language. Proc. IFIP 74, North Holland, 1974, pp 556-574.

[10] R.A. Kowalski: Logic for data description. In "Logic and Data Bases", H. Gallaire and J. Minker (eds.), Plenum Press, 1978.

[11] R.A. Kowalski: Logic for Problem-Solving. North Holland, 1979.

[12] R.A. Kowalski: Logic as a database language. Department of Computing, Imperial College, 1981.

[13] R.A. Kowalski: Logic programming. Proc. IFIP 83.

[14] J. Martin: Application Development without Programmers. Prentice Hall, 1982.

[15] T. Moto-Oka (ed.): Fifth-Generation Computer Systems. North Holland, 1982.

[16] M. Sergot: A Query-the-User facility for logic programming. Proc. European Conf. on Integrated Computing Systems. P. Degano and E. Sandewall (eds.), North Holland, 1983.

[17] E. Shapiro: A subset of concurrent Prolog and its interpreter. ICOT Technical Report TR-003.

[18] E. Shapiro and A. Takeuchi: Object-oriented programming in concurrent Prolog. New Generation Computing, vol. 1 (1983).