VLB ARRAY MEMO No. 398

October 11, 1984

From:
Joseph J. (Jay) Johovich
NRAO-VLBA Control and Monitor

To:
VLBA Memo Series

This paper was originally presented to the "Green Bank Computer Conference" in April, 1984.

Crisis in Radio Astronomy

There is a crisis in radio astronomy, and it is part of a problem that is pandemic in science. It concerns the tools we choose to use for computation; specifically the languages, operating systems, and hardware.

In regard to languages, I would like to quote Dr. Kenneth G. Wilson from a paper that appeared in the January, 1984 issue of the "Proceedings of the IEEE, vol. 72, no. 1". The paper's title is "Science, Industry, and the New Japanese Challenge".

"There is a very serious barrier that is today blocking the whole process of computerization of science and subsequent exploitation of this computerization in industry. This barrier occurs in the way that the scientific computer programs are written. When scientists explain their work to each other, they do this using a mixture of mathematics and human language (the standard language for scientists the world over is currently English). The most effective scientific expositions are carefully broken down into chapters of a textbook or sections of a scientific article which fellow scientists can master chapter by chapter or section by section. Unfortunately, the current language predominantly used by scientific programmers to address computers, namely Fortran, is neither a human language nor a mathematical language. The worst aspect of Fortran is that the ideas underlying a Fortran program get all jumbled up in the Fortran description. Many different ideas usually are needed to build major scientific programs, which the scientist would normally explain in separate chapters. However, each and every line of a Fortran program typically draws on many of these ideas at once, making documentation or reading or modification of a Fortran program an endlessly difficult, time-consuming, and frustrating task. To make matters worse, there is a very great pressure to optimize these programs to minimize their running costs; this optimization is usually done relative to a specific computing system, including its precise arrangements for data storage and graphics display. Both the writing of Fortan and its optimization are highly error-prone processes; full confidence in these programs can rest heavily on twenty years of usage of them, combined with hundreds of man-years spent improving and optimizing these programs and then discovering and removing errors.

"There are two consequences of the 'Fortran barrier'. The first is that programs presently running on a specific mainframe often cannot be moved to a more powerful supercomputer even when growing usage of the program has made the mainframe inadequate, even when a major product line is at stake. The effort and the delays involved in moving the program and then re-establishing confidence in it are too overwhelming to contemplate. The second consequence is that there is now a very major reluctance to build new industrial applications programs, just because of the enormity of such tasks. This is especially true in cases where it is not certain that present computers are powerful enough to handle the application, once the program is established..."

Dr. Wilson goes on to say: "... Modern computer science has started to develop a startling array of ideas for easing the difficulties of interacting with a computer. These ideas include many different language frameworks; they also include ideas from the so-called artificial intelligence community...

"Unfortunately, recent developments in computer science followed a long period of gestation during which modern computer science became very isolated from the many worlds of specific computer applications and from the computer manufacturers. This isolation seems to exist whether the computer scientists live in universities, industrial laboratories, or whatever. Computer scientists speak a language which is unintelligible to the average scientific programmer or computer designer. Because of this isolation, the strange languages and other products of computer science do not fully meet the needs of specific application areas and are largely ignored by the real scientific world..."

Dr. Wilson concludes his comments on the "Fortran barrier" with this recommendation:

"Another very critical need is for universities to experiment with new ways to train students to write software. The parallel architectures to come will seriously strain the current Fortran-based frameworks for building scientific software. It is especially important that universities that combine top-quality science and computer science departments encourage interdisciplinary projects combining both computer scientists and scientists to attack the Fortran barrier."

It is unfortunate that Dr. Wilson frames his remarks in the context of the "Japanese Threat". But if some healthy competition can induce change in the scientific community, I am all for it.

The problem extends beyond the Fortran barrier. The whole scientific community is being sucked in by sexy software, software permanently locked into the "old technology":

* UNIX

  (Pipelines are good, but not good enough! The UNIX system does not work well with "Super-computer" architectures, or in true multiprocessing environments because the basic structures of UNIX were not designed for them)

* VAX

  (What's so hot about the VAX? SOFTWARE! Hardware systems abound that can outrun the VAX at fractions of the cost)

* FORTRAN

  (Computational systems an order of a magnitude larger than those in use today will not be programmed in FORTRAN! Some will be programmed in languages like Ada)

Fortunately, the "new technology" is rapidly becoming available. Driving that "new technology" is a philosophical framework that has contributed much to our understanding of computational systems. By far the most important contribution is the concept of the "object".

OBJECTS

The ability to create in a scientific environment is directly related to the tools the scientist uses.

Mathematical notation is a prime example. Before the standardization of formula representation, mathematicians labored to explain their ideas. The notation not only helped to communicate those ideas, it provided a framework for manipulating mathematical concepts with just a pencil and a piece of paper. The notation itself helps us to create new mathematical concepts, and even new notations...

The job of computer programming became much easier when the ability to program a computer in mathematical notation first appeared; For-Tran, for "formula translator". Fortran gave the computer programmer the power of mathematical notation. And today, we even have programs that manipulate the symbols of mathematical notation for us (SMP).

Every science has its own notation. That is because each branch of science views the world with a different conceptual framework. Although we all, at times, use mathematics, our larger ideas are framed using patterns that can only be described using non-mathematical notations. These notations help us to manipulate these larger ideas with dexterity.

The computer scientist knows this. It has been the job of the computer scientist to transform these notational conventions to machine level actions. Fortran was the first step (I'll agree, it was the most important step, due to the ubiquitousness of mathematical notation).

Most programmers were willing to continue this transformation process, but, in the 1960's some computer scientists decided to look at the way we program computers. The end result is a notational system for expressing the conversion of notational systems to computer models. Although the system of computer notation is still very much in flux, one essential element has come forth: the object.

The concept of the object is a fundamental structure that allows ALL concepts to be represented in a cohesive, manageable manner. It is atomic, it is modular, it forms the basis for the task, the sub-program, the procedure, types, elements, bytes and bits! And, of course, it can be applied in the other direction as well; to higher level structures that we have yet to imagine.

The concept of the object is the most powerful tool that computer science has had to offer us. It is a tool that will revolutionize science in every field, by allowing us to emerge from the morass of IO computing. It is the tool that will allow us to "manipulate concepts like we now manipulate bits".

The object is expressed in programming languages using the following concepts:

1. Modularization
2. Task-task communication
3. Strong typing (creation of new types)
4. Separate compilation of modules
5. Information hiding (implementation hiding)
6. Scope rules

Presently there is only one language that supports all of the above features and can still boast that it is a "real", potentially useful, language; and that language is Ada.

Hardware designs that support objects are now becoming available, and will, soon, be quite common. But, because of the simplicity of our systems, we do not yet need extensive object support in hardware. That support is now provided by operating system utilities and compiler software. Of course, if the language does not support objects, true object programming becomes very expensive. In terms of programmer support and complexity, forcing unqualified languages into an "object" mold is an exasperating task. I think that the SAIL experience fits that description. But, that early failure was because SAIL was an immature technology, and experimental at that; not because the concepts that drove you to use SAIL were incorrect. We have the chance to try again, this time with a technology that is mature, standard, and supported. And it will only get better!