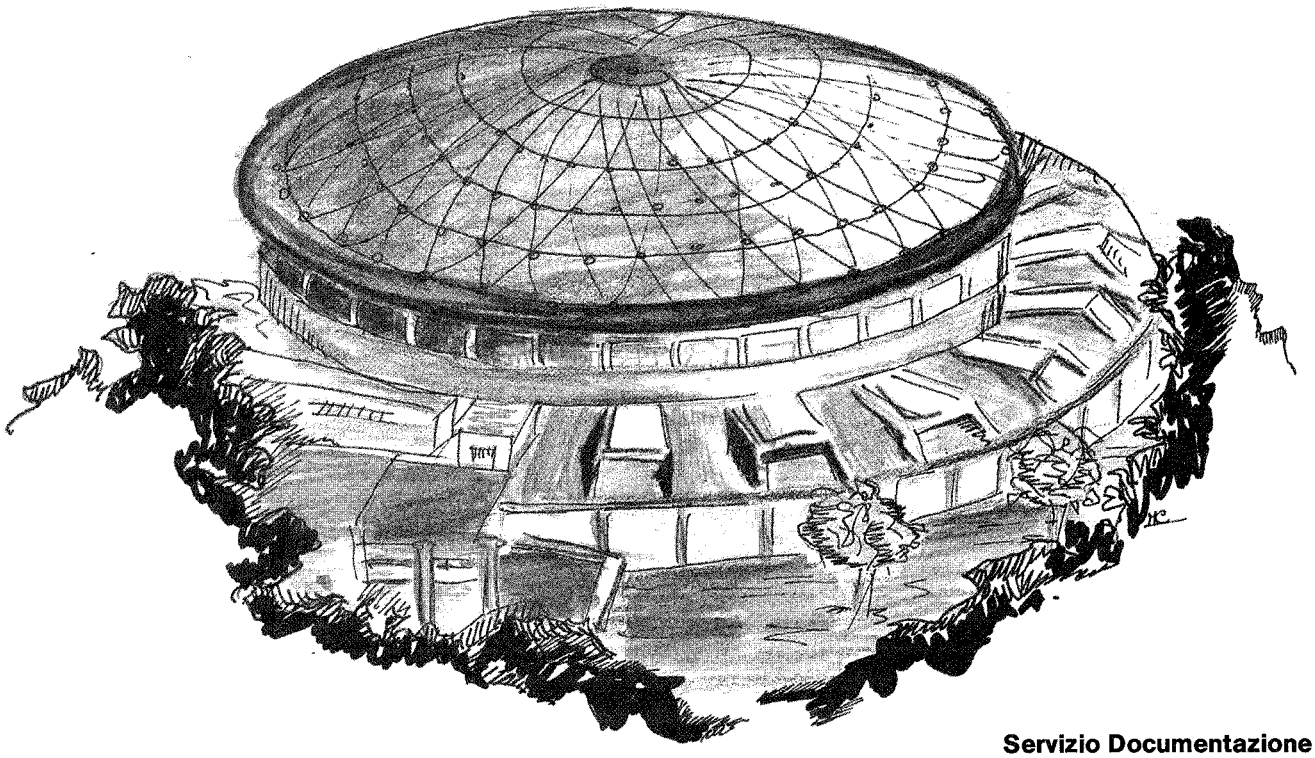
ISTITUTO NAZIONALE DI FISICA NUCLEARE - ISTITUTO NAZIONALE DI FISICA NUCLEARE - ISTITUTO NAZIONALE DI FISICA NUCLEARE - ISTITUTO NAZIONALE DI FISICA NUCLEARE - ISTITUTO NAZIONALE DI FISICA NUCLEARE - ISTITUTO NAZIONALE DI FISICA NUCLEARE - ISTITUTO NAZIONALE DI

# Laboratori Nazionali di Frascati

LNF90/021(R)
2 Aprile 1990

L. Trasatti:

## NEW LANGUAGES, HYPERCARD AND ADONE



**Servizio Documentazione**
**dei Laboratori Nazionali di Frascati**
**P.O. Box, 13 - 00044 Frascati (Italy)**

INFN - Laboratori Nazionali di Frascati
Servizio Documentazione

LNF90/021(R)
2 Aprile 1990

# NEW LANGUAGES, HYPERCARD AND ADONE

L. Trasatti
INFN - Laboratori Nazionali di Frascati, I-00044 Frascati

## ABSTRACT

"Old" and "New" languages are compared. Experience with Hypercard on a Macintosh IIx to generate a human interface program is reported. A set of Hypercard XCMD's has been developed to interface a Macintosh to the VME data acquisition standard

## 1. - "OLD" LANGUAGES

Computer languages, as we usually define them, have been with us for a long time: FORTRAN, PASCAL, C, BASIC and so on have been present in our discussions for at least 20 years.

While CPU computing power has increased by several orders of magnitude and memories have gone from kilobytes to megabytes, we are still using the same tools to manage the completely new monsters that are sitting on our desks: there must be something wrong.

All of these languages have been written for machines which sported 16 or 32 Kbytes of memory! And they show it.

There is a fundamental difference between a 100 line program and a 100,000 line program: the probability to make a mistake is one thousand times higher.

There is such a thing as an error prone language. Think of a couple of simple examples: in BASIC, it is perfectly legal to have a scalar variable with the same name as an array: in other words, A and A(i) are two completely separate entities. How about a simple way to generate very involved errors?

In FORTRAN, column 6 on our screen is still reserved for continuation characters: remember when columns 73-80 were used to number cards, in case you let them fall on the floor?

In C, if you write a number as 10, it is what it looks like. But if you happen to write 010 (zero - one - zero), it is Octal! It means 8! Again, why all of these dirty tricks?

That's what it is all about: tricks. And a very large part of our computer "experts" are extremely proud of them. Unfortunately, the intricacies of a language are not only fatal to the newcomer, but generate an unavoidable amount of time waste and of simple, honest mistakes from the experts.

The Macintosh revolution has been a clear demonstration of the whole process. The "experts" have strongly reacted for a long time to the idea of wasting much of the newly available processing power to ease the process of handling the human interface. Think of the difference between a command line like:

COPY A:\subdirectory:myprog.eas B:\newdir:oldname.asy

and a dragging operation, where you click on a file icon and drag it to another window on the same screen.

I am not considering the difficulty of the process, but only the probability of making a mistake.

This new trend has been slowly and quietly infiltrating the programming language scene for a long time, although not many have recognized it until recently. The first examples have been in the database-spreadsheet environment. How long has DB2 (and later on DB3 and DB4) used a simplified language of its own? Of course you could still write your own database from scratch using the official languages, but it certainly was much faster not to have to worry about dimensioning arrays, reserving space for variables and creating your own fields. After all, it was always the same old and obvious operations, and if somebody did it for you, so much the better. All this reflects a growing worry by the industry, which is much more time conscious than the research environment, in the fact that software costs have now soared much higher than hardware costs. It doesn't pay to write the same software twice.

Following this need, a large variety of languages has infiltrated the programming environment, one for each successful program, until every application has its own. The consequent proliferation of manuals is better suited to a library than to a software house: if you try to use one of the last and most famous spreadsheets, Microsoft Excel, and try to use the Help facility to find out how to get the sum of two fields, you end up with the statement "Study the manual from page 256 to page 257", which is worse than the old FORTRAN manual, if you plan to use more than one or two programs in the next five years.

It is clear that the need for languages working at a higher level than the classic ones is with us. On the other hand, it is not at all clear what "higher level" means.

New capabilities generate new problems: people have been complaining for years about the high level of obscurity of the famous Inside Macintosh manual, but I have heard somebody with experience in DecWindows, which is the equivalent for VaxStations, complaining about exactly the same problems: obscure manuals, no examples, involved subroutine nests, and so on.

What I think this means is that it is very difficult to write low level software for new facilities. Data fields have been with us for a long time, but graphic windows and menus are relatively new. It takes a while to get used to new concepts and to make it easier to handle them.

## 2. - "NEW" LANGUAGES

A new generation of programming languages is finally appearing. The new trend is not anymore to write an ever more complicated way to handle the same things, but to give new tools to handle as simply as possible the new capabilities, without having to repeat a hundred times the same hundred lines of code to open one hundred very similar windows.

Hypercard is only one of these new tools: and it is full of limitations, unusual shortcuts, strange ideas. But it allows you to create a button or a field in a matter of seconds, instead of hours. It reduces the lines of code by a few orders of magnitude, with a proportional decrease in the probability of errors. It also introduces new ideas, more in touch with reality than the usual way of handling things (remember, usual means something created for a 16 Kbyte machine with a few hundred Kbytes mass storage and a computing power of a few tenths of a MIP).

It is an interpreted language: which means it is slow to execute and fast to develop. On the other hand, you can write your own external commands or functions (XCMD and XFCN) in whatever language you choose, from FORTRAN to C to Assembler, and recall them very simply. Remember the old 95/5 rule? 95% of the running time of a program is spent inside 5% of the code: so you only have to worry about the efficiency of 5% of your code.

All variables are stored as ASCII. It takes a bit more memory, but is it a big concern compared with the possibility of ALWAYS looking at a file (stack) and seeing with your own eyes what it contains?

Everything is saved as you work, without explicitly having to enforce it with a command. How many times did you forget to save before recompiling or just before a crash?

There are a lot of other similar, revolutionary ideas. Not all of them are viable, and a lot more progress will be needed. But the direction is right.

I have tried to use Hypercard in data acquisition and process control in an upgrade of the ADONE control system.

A set of XCMDs and XFCNs has been developed using C and Assembler to take care of the actual interface to the VME hardware through the MICRON interface. All of the human interface is handled by Hypercard itself.

The results will be published in a separate report. The overall impression is that such a complicated program took a very short time to write and an even shorter time to debug, thanks to the immediate response of a simple and interpreted language. Corrections and upgrades in the field were very easy to implement.

The only real limitation, speed, is after all not so important when you have to deal with the human hand and the human eye. Moreover, faster and faster machines keep appearing on the market. If only we could have color.....