A progress report on NEBULA

By A. R. Rousell

It is now close on two years since Ferranti came to the conclusion that implementation of NEBULA should proceed, and that COBOL should be ignored, at least for the time being. The NEBULA compiler for Orion is nearing completion and good progress has been made with NEBULA for Atlas. It therefore seems an opportune moment to go back and examine the reasons for reaching this decision, and to see how far the situation has changed in this period.

The first report of the CODASYL Committee was made available in mid-1960. At about this time it was necessary to decide first whether Orion business users should be provided with an autocode, secondly to what extent an autocode should remove the need for using conventional machine orders, and lastly the extent to which object-program efficiency was important.

In view of the magnitude of the programming effort which the early Orion customers were expected to expend on the very large applications planned for their machines, it was considered essential that a good comprehensive autocode should be provided. It was further considered that the aim should be to eliminate completely the need for machine orders, or at least to reduce their use to an absolute minimum.

COBOL 60 was examined with this object in view, but it was found to lack a number of facilities required by Orion users. It could, nevertheless, have been implemented together with additional facilities defined by the manufacturer, and this is in fact what has happened in some cases. Ferranti's view, however, was that the object of a common language was to some extent defeated by such an approach, and we preferred to implement a language, which, although similar in many respects to COBOL, was in no way bound by decisions taken in the light of business requirements in the U.S.A., requirements which do not in many cases reflect the state of data processing in this country. It was in our view essential that, if customers were to have any confidence in the ability of the language to meet their requirements, the specification should be as near as possible to completion before programming started. There appeared to be little or no chance of achieving this object owing to the delay which inevitably must arise when decisions depend on a committee sitting many thousands of miles away. It seemed far more useful to get on with the task of specifying the language and writing the compiler. To return to COBOL as specified at that time, it was fairly obvious after only a cursory study that it was orientated towards a character rather than a binary machine. It is of course perfectly feasible to write a COBOL compiler for a binary machine, but it is extremely difficult to make the compiler really efficient. Since the Ferranti machines for which a business language is required are binary machines, it seemed unreasonable to penalize users for the sake of conformity with a language over which we have little influence.

The chief difficulties for the user arise when the physical appearance of data on external media such as cards or paper tape is considered. COBOL starts at the point where data is read from magnetic tape and ends with the writing of an updated magnetic-tape file and results file. This is a perfectly feasible proposition for those installations which handle input and output on off-line equipment, or use small "pup" computers for these operations, but it is hardly satisfactory when the complete application is carried out on one large machine. Manufacturers' additions to COBOL have overcome this difficulty in some cases, but a complete plain-language program is still only possible if rather rigid rules are adhered to. NEBULA attempts to give not only a complete freedom of choice of data media, but also freedom of format on the chosen media.

In NEBULA the data description is divided into two parts—the Logical and Physical Descriptions. The Logical Description is similar and equivalent to the Data Description in COBOL, that is it deals with the appearance and characteristics of data within the computer store. The physical description, however, includes sufficient facilities to enable data held according to a variety of external conventions to be read and converted automatically. Thus the standard input and output verbs READ and WRITE are completely machine and data-medium independent. To take punched cards as an example, it is possible to read and convert cards punched to any of the four different Hollerith conventions, two Powers (ICT) conventions, and the Bull and IBM conventions.

This spotlights one of the major difficulties in providing comprehensive facilities at this particular point in time. Many users are at the point where they wish to supplement existing punched-card equipment with a computer. They naturally wish to avoid the cost and upheaval involved in any major repunching of existing card files, or the alteration of practices with which the organization is thoroughly familiar. Many of these practices originated many years ago and might be considered non-standard nowadays. Old Powers cards in particular have given rise to many tricky problems. The block overpunching technique, that is the use of overpunch positions to provide additional columns rather than to extend the value of a column, is used quite often. Similarly, one finds many examples where a column has been used to provide 12 Yes/No answers rather than to hold a quantity of any sort. In market-survey work, practically the

entire card is sometimes punched in this way. Cards like this can be read and processed by a NEBULA-compiled program.

This is not to say that we have satisfied all our customers' demands. There are at present ten Orion commercial customers who are attempting to write all their programs in NEBULA. It requires an act of faith to order a machine from the drawing board. It requires a similar act of faith to write programs in a language for which no compiler exists. We are delighted that so many of our customers have chosen to indulge in both. As program writing has progressed, situations have inevitably arisen which could not be handled easily within the framework of NEBULA as it was originally specified. The feedback of difficulties of this sort has fortunately been prompt, and it has therefore been possible to provide further facilities to cope with these situations. This is something which is only possible where close liaison exists between the customer and the compiler team. However, it would be foolish to pretend that we have been completely successful in this respect.

We have sometimes found it possible to provide a facility to meet one particular customer's problem but have refrained from doing so because it is impossible to make the facility general enough to avoid other customers misusing it. There is one case where it has been completely impossible to meet a customer's requirements due to the time-scale involved. This particular case involved the reading of cards containing fields used in a variety of ways. Some followed more or less conventional patterns, but others were punched in a sort of binary-coded decimal, that is, a three-digit number was contained in one column of a card. It has been necessary to revert completely to machine orders to read and convert these cards.

There are other cases where users have chosen to take advantage of the facility to include machine orders in otherwise plain-language programs. For example, there is often a requirement to represent a number of Yes/No answers by individual bits within the store. As I have mentioned, the reading of this type of data from cards can be handled quite reasonably by NEBULA, but the Procedure Description which handles reference to the individual bits tends to be rather clumsy and somewhat inefficient. It is, however, true to say that users are adopting this course only on very rare occasions. Well over 95% of all programs are being written completely in autocode. Although the facility to use machine orders has been included, we feel that it is not a solution which should be encouraged. Mercury autocode has demonstrated this point fairly conclusively. Many Mercury autocode users have at some stage used machine orders. They are running into difficulties now that their machines are being replaced or supplemented by more modern machines.

Another unforeseen difficulty has arisen over the size of object programs. The advent of the daily updating run, in Insurance work for example, has meant that facilities which might have been spread over several programs, run on different days, have now been crammed into one large program. The provisions for packing data into variable-length records have also encouraged users to hold data on files which might otherwise have been discarded as being uneconomic. In one case a program length of between 50,000 and 75,000 instructions seemed likely to result. A program of this size is well outside the scope of the NEBULA compiler, and for the time being some changes in systems have been necessary. However, this restriction, and others affecting the size and efficiency of the object program, will be removed by a new version of the compiler which is currently being planned.

Our experience in advising customers who are using NEBULA leads us to believe that we have been working along the right lines. One result of attempting to meet all customers' requirements has been that the language has become more complicated than we at first hoped. We would certainly no longer claim that it would be learnt overnight. It has nevertheless been possible to avoid the division of labour between systems and programming teams which has been a feature of earlier installations.

In conclusion I would say that our experience in advising customers who are using NEBULA has brought to light many problems, but these have only served to confirm that we appear to be working on the right lines.

Reference

BRAUNHOLTZ, T. G. H., FRASER, A. G., and HUNT, P. M. (1961). "NEBULA: A Programming Language for Data Processing," *The Computer Journal*, Vol. 4, p. 197.