THE SYSTEM SPORA

S. Lavrov
Institute of Theoretical Astronomy
Leningrad - U.R.S.S.

"SPORA" is an abbreviation for "Software Package Oriented to Research in Astronomy" (in Russian - "Specializirowannoie Programmnoie Obespechniie Rabot po Astronomii").

The system is designed for the wide range for users who are not inclined to call themselves computer programmers. We assume that most of them only wish to state their problems as simply as possible - just to say which information they have and what they want to get. On the other hand some of these users have rich programming experience and they are fond of using sophisticated languages, like PL/1 or ALGOL 68, and writing complicated programs.

The system SPORA must respond to interests of all this range of users.

We have to deal with some difficulties arising in our area (as well as in other theoretical and applied sciences) :

- a variety of physical representations (measurement methods) of most quantities,

- a variety of programming implementations of a given physical representation in common programming languages,

- very different forms of existing information sets and program modules (lack of standards),

- low actual level of existing universal (so called "high level") programming languages and usually lower level of input languages of data base control systems, operating systems and other programming tools.

We do not yet consider some very important features, such a distributed data bases, telecommunication and a variety of computer types.

One of the starting points in developing the system SPORA was that

C. Jaschek and W. Heintz (eds.), Automated Data Retrieval in Astronomy, 135–143.

common programming languages (CPL), such as FORTRAN and ALGOL 60 or even PL/I, ALGOL 68 and PASCAL, have had their golden age about ten years ago. During the last decade programming practice has brought in use new programming tools - namely data bases (DB) and application program packages (APP) - both with appropriate control systems and input languages. These languages are often not compatible with CPL which are still more appropriate for writing new algorithms. On the other hand DB and APP are more suitable for storing and using previously accumulated information and knowledge.

In the system SPORA all these means - DB, APP and CPL - are tied together and form a (hopefully) balanced integrity.

One more concept of the 70-ies is that of abstract data types (ADT). In contrast with DB and APP this concept has come not from practice, but rather from the intention of language designers to raise the level of programming languages, to make these languages closer to natural languages and simpler for use and understanding. Therefore this concept was accepted and widely used in the system SPORA and in its input language "Descartes".

There are two kinds of abstract objects in the system : an abstract type and an abstract map (or mapping). An abstract type may be either a primary one or a relation.

A primary type is introduced by a declaration, an example of which is given on fig. 1. The declaration contains the name of the type, the names and types of operations that can be performed on values of the type, and the axiomes expressing the properties of these operations. In the example identifiers calend and julian are names of so called representations. Each representation is bound with a method of physical measurment of a value of the type. Each physical representation may in its turn allow different implementations in various programming languages. The properties of a representation can be described, e.g., as in fig. 2.

We cannot go into details of the programming implementation description which tell the system how to denote a value of the type, to duplicate it and to convert it from one physical representation into another. This part of the type declaration is denoted by "..." in the figure.

An example of a map declaration is shown on fig. 3. Again the implementation details of this declaration are omitted.

We distinguish between relations of a data base, which are called tables, and relations of an application program package, which may have more complex (hierarchical) structure. Let us consider closely only the first kind of relations.

The typical example of a table declaration is given in fig. 4. It contains a table name, names and types of columns, a unique key of the table, and possibly some indications of more efficient ways of access

to the table.

The next concept is that of a data base schema (fig. 5). A data base schema describes tables that are stored permanently in this data base. For each specific program it is possible or even necessary to describe a subschema of the schema (fig. 6). The names of tables and the names and types of their columns must be the same as in the data base schema. The number and order of the columns may however be different.

One possibly feels that the language of such declarations is too complicated, especially in the part dealing with representations. It may be, but these declarations have to be written only once for a relatively long period. This has to be done by an advanced user. We call him (or her) a designer in contrast with the broader range of users. Their task is to describe a specific problem by means of one or more Descartes statements.

The main part of a statement is a request, which consists of a projection, a line description and a condition. Only the line description is obligatory. In a request, presented on fig. 7, we ask for an intermediate table, containing observations of Mercury at third contacts made during an eclipse in 1973 having the maximum value of O-C.

Each statement is to be placed in a program, written in some host (base) programming language. In the case of the examples given in fig. 8 and 9 this language is ALGOL 60.

The part of a program presented in fig. 8, calculates the average value of O-C for all observations contained in the personal archive of a user. The fragment, exemplified on fig. 9, recalculates those values of O-C, which do not lie within interval $[C1, D1]$. The construction r.oc -> cvt julian (A) bounds for each line r of the table personal-archive the value of the attribute oc in the representation julian with the host language variable A. During the execution of the statement B : = B + A (fig. 8) this variable assumes that value.

A user must have only a basic knowledge of host programming language - how to declare variables and to describe the simplest calculation.

We have not mentioned here another part of the language "Descartes" that deals with application program packages. Using this part a problem can be stated even more simply and on a more abstract level.

The general structure of the system is shown on fig. 10. The model of an application area is created by designer. The user specifies his problem.

A problem is always stated in one specific area model. There may be many models in the system and many problems per model. A model defines terms used in its area and their connections and properties. All the

https://doi.org/10.1017/S025292110008283X Published online by Cambridge University Press

```
type  time  : calend, julian
      op same year ≤ : (time, time) ---> (boolean) ;
      +  : (time, time) ---> (time) ;
      abs : (time) ---> (time) ;
      axiom  same year (t, t) ;
         (same year (t, t1) impl same year (t1, t)
         (( same year (t, t1) and  same year (t1, t2))
            impl  same year (t, t2))
         order (≤ ) ;
      ...
endtype
```

Fig. 1

```
calend : representation algol  / fortran
         declaration  day : "REAL" / REAL
                    month : "STRING" / CHARACTER (5)
                     year : "INTEGER" / INTEGER

      ...
      endrepr
```

Fig. 2

```
map ≤ : (time, time) ---> (boolean) infix
      ....
      endmap
```

Fig. 3

https://doi.org/10.1017/S025292110008283X Published online by Cambridge University Press

```
Table  mercury transit
     (nobs : observation number ; nc : contact number ;
     lambda, phi : angle ; h : distance ;
     dc : contact description ; cond : observation condition ;
     o minus c : time ; reduced time : time) ;
     key (nobs) : index  (reduced time, nc),...
endtab
```

Fig. 4

```
schema  observations ;
primary
     type observation number, contact description, ...
          ...., time as julian
     map within interval
table mercury transit .... endtab ;
table transit definition
     (nobs : observation number ; obs   observatory ; name : name ;
      telescope : instrument ; meth : method ;
      method description, source, notes : text) ;
     key (nobs)
endtab ;
table personal archive
     (nobs : observation number ; dc : contact description ; oc : time)
     key (nobs) ; order (nobs ::⩽ )
endtab
endschema
```

Fig. 5

https://doi.org/10.1017/S025292110008283X Published online by Cambridge University Press

```
db observations ;
     dbtab mercury transit
       (nobs : observation number ; nc : contact number ;
        dc : contact description ; o minus c, reduced time : time) ;
           personel archive
       (nobs : observation number ; oc : time ;
        dc : contact description) ;
     worktab  wt
       (nobs : observation number ;
        dc : contact description ; oc : time)
enddb
```

Fig. 6

```
r in mercury transit ::
     ((same year (r : reduced time, dnt calend (1973))
         and  (r.nc. = dnt contact number (3)))
     and (all t in mercury transit
           ((same year (t. reduced time, dnt calend (1973))
                 and (t.nc. = dnt contact number (3)))
           impl  (abs (r.o. minus c) >> abs (t.o. minus c)))))
```

Fig. 7

https://doi.org/10.1017/S025292110008283X Published online by Cambridge University Press

```
....
"INTEGER" N ; "REAL" A, B ; ...
B : = 0 ; N : = 0 ;
foreach r in personal archive
     with r. oc. ---> cvt  julian (A) do
          B : = B + A ; N : = N + 1
     enddo ;
"IF" N  ≠ 0 "THEN" B : = B/N ;
...
```

Fig. 8

```
...
"REAL" A, C1, D1 ; "INTEGER" K ;
"REAL" "PROCEDURE" F (X, I) ; ...
update r in personal archive ::
     not  ((cvt julian (C1) ≤ r. oc) and  (r.oc ≤  cvt julian (D1)))
     with  r. oc. <---> cvt  julian (A),
           r. nobs ---> cvt   observation number (K) do
        ... ; A : = F (A, K) ; ...
       enddo ;
...
```

Fig. 9

https://doi.org/10.1017/S025292110008283X Published online by Cambridge University Press

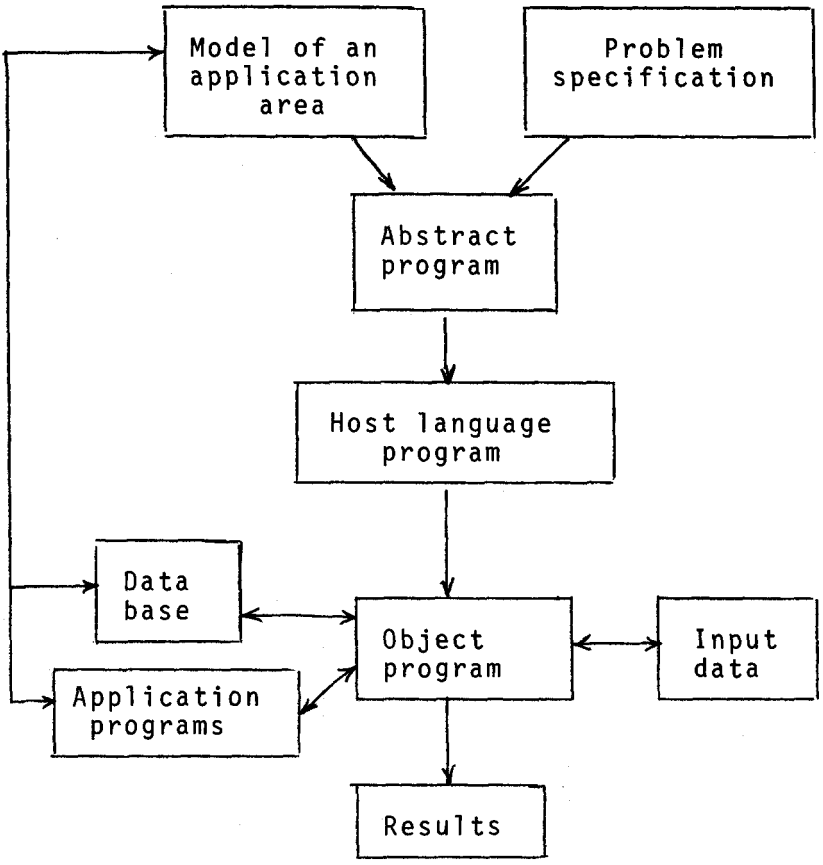

Fig. 10

https://doi.org/10.1017/S025292110008283X Published online by Cambridge University Press

primitive types and maps used in a model must be declared beforehand.

From a model and a problem specification the system synthetizes an abstract program, expressed in terms of abstract types and maps. The system uses here properties of abstract objects, contained in their declarations.

Then the abstract program is translated into a host language. This is performed using properties of physical representations and their programming implementations.

At the next stage the host language program is compiled into an object code. The compilation is done by a serial host language compiler.

The object program accepts input data prepared by the user and returns to him the results of its execution. It may also get information from the data base related to the supporting area model and put some part of its results there. It may also use application programs (modules) from the package and deliver new ones to it.

The sources of the underlying ideas used in SPORA are :

- relational model of data bases (E. F. Codd)
- the language UTOPIST (E. H. Tyugu)
- abstract data types (B. Liskov, E. Moss)
- abstraction levels (E. W. Dijkstra)

and others.

REFERENCE

I.O. Babaiev, F. A. Novikov, T. I. Petrushina. Iazyk Dekart - vhodnoj iazyk sistemy SPORA : - In "Prikladnaia informatika", vyp. 1, 1981, "Finansy i statistika", p.p. 35-73