OPAL, an Open Physics Analysis Language.

A.G.Shamov,

Budker Institute of Nuclear Physics
630090 Novosibirsk, Russian Federation

OPAL is an open language which includes most important FORTRAN-77 operators and can be easily expanded with arbitrary user-defined statements. The subset of OPAL called XBOOK provides a convenient interface to HBOOK/HPLOT packages of the CERN program library. The OPAL toolkit is rather compact and can be built in any application. Currently OPAL is available under VAX/VMS only.

Key words: Analysis; programming languages; software tools.

1 Introduction

The last years of HEP show that the PAW program [1] is an efficient tool of data analysis. The drawback of it is the necessity to use very different languages: that of KUIP [2] and FORTRAN in the COMIS frame [3].

The paper present the *open* language which includes simple and convenient commands for histogram manipulating together with the most important FORTRAN-77 operators and can be expanded with arbitrary user-defined statements.

The language was called OPAL in honor of PAW and was employed for data analysis and presentation in some experiments [4], in the test beam and prototype experiments software [5], as an input language of the simulation tool [6] and as a general command language of the XonLINE data acquisition system [7].

2 OPAL kit

The OPAL kit is the object (or shareable) library which can be built in any application program. The library includes the initialization procedure, the command line interpreter (CLI), the set of procedures implementing the standard OPAL commands and the interfacing routines allowing an application to establish its specific environment.

An OPAL application can be considered as consisting of a static part designed at coding time and a dynamic one created during the interactive session with OPAL interpreter.

Using the OPAL interfacing routines any static objects (variables, arrays, common blocks, functions and subroutines) can be declared for interactive use at the OPAL level. Dynamically created objects can be accessed from the static parts of an application by their names.

This allows to develop an efficient and flexible tools for off-line and on-line data analysis and simulation.

3 OPAL compiler/interpreter

The OPAL syntax is similar to that of Digital Command Language DCL but allows multilevel command parameters:

$$
\begin{aligned}
&command &&:= [label]\ \ [c\_keyword]\,[qualifier...]\,[parameter\ \ ...]\\
&qualifier &&:= \backslash q\_keyword\,[=\ parameter]\\
&parameter &&:= parameter\,[qualifier]\mid parameter_1\\
&parameter_1 &&:= parameter_1\,d_1\,parameter_2\mid parameter_2\\
&parameter_2 &&:= parameter_2\,d_2\,parameter_3\mid parameter_3\\
&parameter_N &&:= [p\_keyword\ \ p\_k\_d]\ \ expression\mid source\_string
\end{aligned}
$$

where *label* - statement label starting with a digit, *keyword* - user specified keyword starting with a letter, d_1, d_2, ... - delimiter set specific for given command, *p_k_d* - parameter keyword delimiter, *expression* - FORTRAN-77 expression (integer, real etc.), *source_string* - character string which is not to be interpreted by the CLI.

Such syntax, with the additional rule that a delimiting symbol enclosed in brackets is not considered as a delimiter, allows to describe all FORTRAN-77 operators with the reasonable limitations and gives a simple way to implement application-specific commands.

To allow more or less free usage of blanks for source line formatting, the following rule is applied:

delimiter := [*blank*] *non_blank_delimiter* [*blank*]

Processing a source line, the CLI extracts the command label, if specified, and the command keyword, looks up the command table and parses command qualifiers and parameters using the corresponding delimiter set. The command keyword can be omitted for assignments and procedure calls (LET and CALL commands).

To define an additional command, an user must call the library routine to specify its parameter structure and other attributes and supply the procedure implementing the command.

The CLI generates an executable machine code for each command processed. Parsing the command line, it generates a code to obtain a value of each parameter specified. The command-dependent part of code generation can be accomplished in the two different ways.

If the command is declared as a 'system mode' command, the code must be generated by the command implementing procedure which is called once after parameter parsing.

If the command is declared as a 'user mode' command, the CLI generates the code to call a command implementing procedure which must analyze parameters specified and perform actions required.

System mode commands are generally more efficient than user mode ones. To generate an executable code, a command implementing procedure can use the special library calls or recursively call the CLI. The user mode command procedures are similar to those of KUIP.

Due to machine code generation, OPAL is approximately 10 time more efficient than COMIS.

4 Example of a command implementation

Let us consider how to implement in the OPAL frame the *select* command of the Kinematical Analysis Language (KAL) developed for the data analysis of the ARGUS experiment [8]:

select particle_type[, ...]

This command starts all loops required to select all combinations of particles of specified types in an event record. To declare the commands one call of the OPAL library is required:

```
call xdcom( SELECT,0,'XP1','SELECT','R9TSD2',CC)
```

This call specifies the external command implementing procedure (SELECT), the command mode and number of parameter description strings ('XP1'), the command keyword ('SELECT') and the parameter description string 'R9TSD2' which defines the parameter repetition count 'R9', the parameter type 'TS' (source string) and the delimiter set 'D2' (commas).

Using the OPAL library, the command implementing procedure must retrieve the particle types and generate the executable code required. For two particles it might look like:

```
call x_cli('DO LV1=1,PDIRL(NPC1)')
call x_cli('IP1=PDIR(LV1,NPC1)')
if(npc1.eq.ncp2) call x_cli('DO LV2=LV1+1,PDIRL(NPC1)')
if(npc1.ne.ncp2) call x_cli('DO LV2=1,PDIRL(NPC2)')
```

call x_cli('IP2=PDIR(LV2,NCP2)')
where NPC1,NCP2 are numerical particle codes, PDIR, PDIRL are functions for accessing particle directory of an event and IP1,IP2 are the particle pointers which must be used by other KAL operators.

5 OPAL interface to HBOOK/HPLOT

OPAL includes the set of commands and the interfacing procedures for the histogram manipulating called XBOOK ('X' can be considered as the cyrillic letter matching the latin 'H').

In the XBOOK frame histograms are identified by names and can be arranged in multidimensional arrays. It provides most part of the histogram service available in PAW and, additionally, allows to control interactively histogram filling process both in the static and dynamic parts of an application.

The last feature is very important for an on-line data analysis. Histogram filling with XBOOK is faster than with normal HBOOK calls.

6 Conclusion

A few years of exploitation of OPAL in BINP in various applications have shown its efficiency and convenience. VAX/VMS version of OPAL is available for HEP community. Porting it to other platforms requires an external support.

References

[1] Brun R. et al., CERN program library Q121.

[2] Brun R., Zanarini P.. CERN program library I202.

[3] Berezhnoi V. et al., CERN program library L210.

[4] S.E.Baru et al., Phys. Rep. 267 (1996) 73-160.

[5] V.M. Aulchenko et al., NIM A379 (1997) 475; ibid p.360; ibid p. 491.

[6] A.G. Shamov, A.R. Buzykaev. LCE, Light Collection Efficiency simulation tool. This proceeding.

[7] I.B.Logashenko, A.G.Shamov. Int. Conf. on Computing in High Energy Physics'95, Rio de Janeiro, Brasil, 1995.

[8] H. Albrecht et al., http://www.Physik.Uni-Dortmund.DE/ARGUS/KAL