# USE OF OBJECT ORIENTED INTERPRETIVE LANGUAGES IN AN ACCELERATOR CONTROL SYSTEM

N. Yamamoto, A. Akiyama, S. Araki, T. Katoh, T. Kawamoto,
I. Komada, K. Kudo, T. Naito, T.T. Nakamura,
J-I. Odagiri, Accelerator Lab., KEK, JAPAN
M. Kaji, Mitsubishi Electric Co. Ltd., JAPAN
N. Koizumi, Mitsubishi Space Software Co. Ltd., JAPAN
M. Takagi, S. Yoshida, Kanto Information Service, JAPAN
T. Kitabayashi, K. Yoshii, Mitsubishi Electric System & Service Engineering Co. Ltd., JAPAN

*Abstract*

In a control system for a high energy accelerator, like KEKB, quick application development/modification is required. This short turn-around time is especially important during a commissioning of the accelerator. In KEKB control system, we have achieved this goal by introducing interpretive programming languages, Python and SAD, in the control system. SAD is the language originally developed at KEK for accelerator lattice design. The other, Python, is the language system distributed as a public domain software.

These languages are used not only for prototyping application but also for developing application software used in daily operation. These languages are easier to learn and safer to use compared to compiled languages such as C or C++. Interface to the appropriate widget library from these interpretive languages, such as Tk/gtk+ widget greatly reduces effort needed for developing graphical user interfaces.

Modular and object oriented features in the languages allow incremental development of application software. This method benefits reliability and maintainability of the software.

Authors will discuss productivity, performance and maintainability of the system based on experiences in the KEKB control system.

## 1 KEKB CONTROL SYSTEM

KEKB[1] is an asymmetric electron-positron collider for B-meson physics built in KEK, Japan. It started its operation on December 1st, 1998 after 5 years of construction. The control system for KEKB accelerators[2] were built using EPICS Toolkit[7] as a framework of the control system.

The core of EPICS toolkit is a runtime database running on VME single-board computers and a network protocol, called channel access, to exchange data between computers. On the top of channel access, EPICS provides various tools to construct a control system. The tools include DM/EDD (Display Manager/Display Editor), MEDM(Motif based Display Manager /Editor), ALH(Alarm Handler), AR(Data ARchiver). These tools can cover 95% of needs in the most control systems. To fulfill the remaining 5% of needs, you may take one of three ways:

1. case 1: Just ignore the remaining 5%.

2. case 2: Extend functionality of existing tools by yourself (and bring it back to the collaboration).

3. case 3: Create new tools or applications for your own needs

Of course, we cannot take the first way. Usually, the remaining 5% is an essential part of the control system.

If we take the second way, there may be several disadvantages.

1. *Synchronization*: Modification of the tool can be made at several places, merging process can be additional load. Use of CVS can reduce the load, however, the integrated software should still be tested.

2. *Fat software*: If you merge every features developed in the collaboration, which may be too specific to the particular project, the tool may grow too large. When one functionality is found useful for general use, this should be integrated with the tool, of course.

3. *Longer development time*: All EPICS applications are written in C/C++. Only expert of these languages and X library can develop these applications. A need of the communication between users(an operator or an accelerator physicist) and a programmer( C/C++ expert) adds another overhead .

So we actually chose the third way. As discussed above, it is not feasible to develop new application from scratch using compiled languages like C/C++. Experience in the TRISTAN[3] control system suggested us to use ”two languages” development system.

## 2 TWO LANGUAGES SYSTEM

### *2.1 Nodal and PCL in the TRISTAN control system*

The TRISTAN control system uses NODAL[4] and PCL(Process Control Language)[5] for software development. NODAL is an interpreted language with Basic-like syntax and features specific to distributed control system. PCL is a compiled language with Fortran-like syntax and extensions for real-time system. PCL is used mainly to develop data modules in Nodal to access hardware. Most of user interfaces and applications in TRISTAN control system were written in NODAL mainly by accelerator physicists.

This kind of ”two languages” system is widely seen in the successful computer system[6]. Unix, for example, has shell scripting languages and C. Elisp and C in the EMACS is another good example.

This observation suggests that successful user extendable system has (at least) two programming languages, an interpreted language and a compiled language.

### *2.2 modularity*

In two languages system, a compiled language is used to extend capability of interpreted language. This extension is modular so that user can add or delete this extension at anytime. The interpreted languages are used as glue to combine these modules. This approach is also useful to avoid a fat application which includes everything(Fat software). Modular design of software also makes development/test/maintenance easier.

### *2.3 User participation*

Another advantage of ”two language” system is participation of users. An interpreted language usually has simpler syntax and is easy to learn and use for the people including *NON*-professional programmers. In KEKB, application programs which needs knowledge of accelerator physics are written by accelerator physicists. In this case, there is large overlap in users of an application and its developer. It reduces overhead of communication between the user and the developer. They use mostly SAD[8] language because they use it for their research anyway.

Python is used to develop an application where an accelerator model is not required. Hardware engineers and accelerator operators can develop there own application to simplify their own daily tasks. Clean and simple syntax of python makes learning it easier. Although there are several textbooks on python in English, only one textbook[9] on Python is available in Japanese. This text book and a python tutorial translated into Japanese, which is available on WWW, were used in KEKB. Self-training with these text is sufficient to start using Python. User can also find good examples of python code in its standard library modules.

| SAD | Python | medm | java |
|---|---|---|---|
| 119 | 24 | 12 | 1 |

Table 1: Number of applications registered in the KEKB control program launcher.

## 3 USE OF EXISTING RESOURCES

It is also important to use existing resources as much as possible to reduce cost of application development. SAD, an accelerator modeling code with Mathematica like script language, is a valuable resource we have. We can also find variety of *OPEN SOURCE* software and free software on the internet. We have developed extensions to access these resources from SAD and Python when these APIs do not exist. APIs to CA library and Tk widgets are examples of these extensions. [ Note that a library to access Tk widget is a standard part of Python. Python includes interfaces to other standard libraries as well.]

## 4 PERFORMANCE

Execution time of program written in an interpreted language is generally slow compared to one in the compiled language. In the KEKB control system, interpreted languages are used to develop graphical user interfaces. Most CPU hungry part are written in C or Fortran as a module. Overhead due to interpreted languages is not significant in most applications. We will show an example of performance improvement with this approach in section 6.

## 5 PRODUCTIVITY AND MAINTAINABILITY

User participation into the development of the system increases productivity in the application development for the control system. Table 1 shows numbers of programs registered into one of the main task launchers in KEKB control system[ Figure 1]. Most of applications are developed since Dec. 1st, 1998. These applications are used in the commissioning and improved by the users. The application may grow with understanding of the accelerators.

Only disadvantage of this approach is lack of proper documentation on these applications. An application can be changed so quickly and users( = authors) spend their time for the commissioning rather than writing documentations of the software.

Use of CVS for version control of the software is encouraged, to avoid accidental corruption of the software. Frequent change of software may introduce a new bug in the software. CVS repository can be used to track when this bug was introduced. And it is possible to go back to the older version when it is necessary.

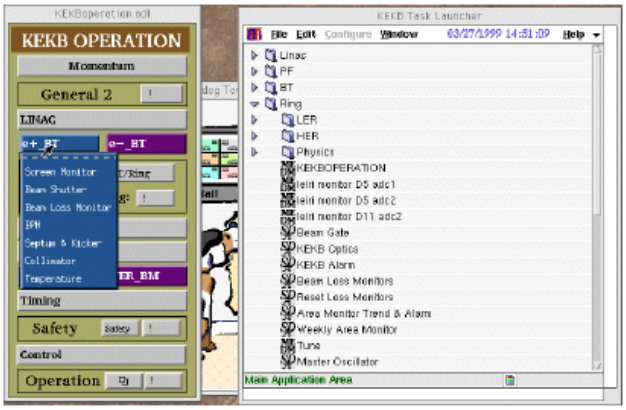


Figure 1: KEKB control program launchers. The left one is MEDM application and the right one is a SAD application.

| ncalls | tottime | cumtime | function |
|---|---|---|---|
| 1 | 0.000 | 0.000 | __init__ |
| 120566 | 26.770 | 39.930 | addData |
| 141534 | 79.550 | 119.480 | do_line |
| 1 | 29.050 | 148.530 | readFile |
| 23 | 0.000 | 0.000 | __init__ |
| 1 | 0.240 | 148.770 | arr |
| 120566 | 13.160 | 13.160 | addData |
| 1 | 0.020 | 150.870 | arr.arr('sample.data') |

| ncalls | tottime | cumtime | function |
|---|---|---|---|
| 1 | 16.780 | 16.780 | readFile |
| 1 | 0.000 | 16.780 | arr |
| 23 | 0.000 | 0.000 | __init__ |
| 1 | 0.000 | 0.000 | __init__ |
| 1 | 0.040 | 17.280 | arr.arr('sample.data') |

Table 2: Profile of arr.py with (lower) and without(upper) C-extension.

## 6 EXAMPLE OF SOFTWARE IMPROVEMENT IN THE TWO LANGUAGE SYSTEM

In this section, we describe how we could improve performance of our application.

A Python module, arr.py, is a program to read archived data from EPICS archive program. It reads a data file and stores data into Python object. Using Python library, user can manipulate archived data and create outputs, including graphs. We also created an application, arr_plot.py, as a generic archived data retriever with graphical user interface, on top of arr.py.

The first version of arr.py was developed as a pure Python program. Although it was useful for small data files, it was very slow. Main reason of this slowness was found to be a number of function calls in Python. The arr.py reads a file line by line and calls a function for each line. Execution profile[ Table 2 upper] shows that a call of a single function, readFile(), in arr.py dominates the total execution time.

We have converted Python function readFile() and functions used in this function into C-extension. Execution profile of arr.py with the C-extension is shown in the lower part of Table 2. An archived datafile of 6MB was used to measure this profile. Result shows arr.py with C-extension is almost ten times faster than the arr.py of Python only version[Upper table].

## 7 CONCLUSION

Using the lesson we learned from the TRISTAN control system, we adopted "two languages system" in the KEKB control system. It uses both compiled and interpreted languages. This approach allows non-professional programmers to develop applications used in the operation of KEKB accelerators and increases number of application developers and number of developed applications. On the other hand, it causes lack of proper documentation of each application. Python provides a way to embed a documentation of each unit( module, class, function) in itself. Users can access these documentation texts using an attribute, __doc__, at run-time. Encouraging use of this mechanism may improve the situation. It is also shown that a modular program ming style allows efficient improvement of the system with the minimal effort.

## 8 REFERENCES

[1] "KEKB B-Factory Design Report", KEK Report 95-7, August 1995

[2] T. Katoh et al., "Present Status of the KEKB Control System", ICALEPCS '97, Beijing, China, November 3-7, 1997

[3] S-I Kurokawa, et al., "The TRISTAN Control System", Nucl. Instr. and Meth., A247, (1986) pp. 29-36. ; T. Mimashi et al., "The rejuvenation status of TRISTAN accelerator control system", Nucl. Instr. and Meth., A352 (1994) 128-130.

[4] M.C. Crowley-Milling and G.C. Shering, "The NODAL System at the SPS", CERN 78-08

[5] "PCL Manual", Hitachi Limited.

[6] John K. Ousterhout, "Scripting: Higher Level Programming for the 21st Century", http://www.scriptics.com/people/john.ousterhout/scripting.html

[7] L. Dalesio et al. , "The Experimental Physics and Industrial Control System Architecture: Past, Present, and Future", Proc. ICALEPCS, Berlin, Germany, 1993, pp 179-184. W. McDowell et al.:"EPICS Home Page", "http://epics.aps.anl.gov/asd/controls/epics/EpicsDocumentation/EpicsGeneral/"

[8] "SAD home page" at: "http://www-acc-theory.kek.jp/SAD/sad.html"

[9] M. Lutz, "Introduction to Python",O'Reilly Japan Inc, Japan 1998, translated from "Programming Python", O'Reilly & Associates, Inc. USA, 1996

[10] 'Python Home Page', "http://www.python.org/"