NEW PROGRAMMING TECHNOLOGIES FOR BEAM DIAGNOSTICS

S. A. Kryukov, Institute for Nuclear Research, Moscow, Russia

Abstract

The MOON Lab, new software for solving various problems of beam diagnostics has been developed at the Moscow Meson Factory Linac. ("MOON" stands for "Multitasking Object-Oriented Network") This program tool takes few advantages when compared with the LabView®, its closest prototype. Several novelties of the programming technology are introduced: multitasking sharing of the apparatus via the driver-reside semaphores, asynchronous process control via the user interface, network-transparent control and more. All that extremely simplifies developing and debugging applications for beam diagnostics, data processing and presentation through a network. The new program technology has been tested on PC applications operating with beam loss monitors, neutron counters and beam current transformers.

1 INTRODUCTION

The control system of the Moscow Meson Factory Linac is very old and it needs in essential upgrade. A problem to be solved first is a problem of beam diagnostics software. The Linac is of significant length, so another problem to be solved is accessing and viewing the diagnostic data everywhere along the Linac.

MOON Lab, projected to solve many of the problems, is an environment for running measurement and monitoring applications on a local area network or a single computer. These MOON Lab-specific applications can be developed with a easy-to-use interface library supplied.

This software is presently available for Microsoft® Windows™ 3.11 and all compatible versions for IBM PCs. It can be easily ported to 32-bit Windows 95 and Windows NT systems.

Strong Object-Oriented Programming (OOP) approach is used for the MOON Lab design. The interface library can be linked in using many programming languages without OOP capabilities. It is presently supplied as source code written in C, C++ and Pascal; a FORTRAN interface is presently under development. For the interface usable with non-OOP languages every possible effort has been made to follow the OOP style itself.

The system does not provide real-time possibilities. The real-time properties of the software are important for data acquisition in nuclear physics but are not necessary for solving most problems for diagnostics. The real-time problems must be solved separately on a lower level of the control system.

2 WHY MOON LAB?

We have decided to develop a new laboratory software system for the following reasons:

- Special program tools like LabView are too expensive for our purposes; with MOON Lab we use general-use compilers only; one license per developer is required.
- MOON Lab applications can be written in virtually any procedural programming language.
- Any type of measurement and data acquisition hardware can be used, including non-standard ones and those unfamiliar to brand-name software.
- All the code of the MOON Lab application can be run in connection with the MOON Lab system under a stepwise debugger. The CPU-level code of, for example, low-level device driver can also be debugged as a part of an entire application.

3 PROGRAM ARCHITECTURE

To run a MOON Lab application it is necessary to load it via a special program, referred to below as the ***MOON Lab Shell*** Application to be run under the Shell is referred to as the ***User Process***. The User Process is written by user with use of the interface library, referred to as the ***MOON Lab API*** (Application Program Interface).

In present version of the software the executable file of the User Process is made as a Windows **DLL** (Dynamic Link Library) running as part of the Shell task. Multiple User Processes can be run concurrently on the same computer, each one as a part of a separate ***instance*** of the same MOON Lab Shell.

The MOON Lab Shell works as a GUI (Graphical User Interface) server for the User Process. It provides a flexible program interface to advanced event-oriented 2-dimensional graphics. 3-dimensional graphics is reserved by the MOON Lab API, but not implemented (see below).

The MOON Lab Shell provides two threads of execution: one for user interface and another one for sequential code of process itself. Special attention has been paid for implementation of the multi-threading semantics for the non-preemptive Windows 3.11.

MOON Lab introduces a few novelties of programming technology. Some of them are mentioned below.

3.1 Asynchronous process control

Traditionally, parameter input is a difficult problem of practical programming, especially in the field of computer-guided experiments. In research experiments it is difficult to decide which data should be changeable during run-time. The event-oriented approach does not solve this problem completely, because many experiments have a complex but strict sequential scenario and must be treated as "statement-driven" or "code-driven".

MOON Lab introduces a special technique of ***registering*** of the process parameters. Parameters to be modified during run-time are registered by the User Process and stored by the Shell in a special user-accessible list. Each parameter corresponds to a variable of the programming language used. *Integer, floating-point* and *enumerated* data types are supported.

For every parameter registered its name and comment are specified. For each numeric parameter a range of valid values is also specified.

The fact of registering does not affect the User Process behavior until the user tries to modify a parameter. After he clicks an item on the list of parameters, the proper parameter editor (depending of the parameter type) is used to modify the parameter selected. After successful modification (in particular, if the value entered is within the parameter range) the User Process continues execution using new value of the parameter.

We tested this technique for both beam diagnostic tasks and numerical simulation problems and proved its advantages in comparison with the traditional approach.

For special purposes the User Process can synchronize its execution with the input of parameters using a call to the MOON Lab API to pause execution and possibly to send a text message to user. In this case the user can continue execution of the User Process via a menu or a toolbar of the Shell.

The user's decision to start, pause, continue or close a process is a special aspect of the asynchronous process control. The user can do this at any time via the Shell. For the proper "post-mortal" action after the asynchronous closure of a process some sort of "destructor" is present in the code of the User Process.

3.2 Event-oriented features of process

From the above discussion one can understand that the idea of the User Processes is to couple a "code-driven" scenario of an experiment with the event-driven structure of the MOON Lab Shell.

Some event-oriented features of the User Processes are also available and very useful. To use these features the programmer of a process has to supply ***callback*** procedures to be called asynchronously relative to the User Process.

There are few points where the callback can be used.

1) A callback procedure can be specified for any registered parameter (see above). It is called each time parameter is modified, if its new value differs from previous. For example, if graphical data is updated in an operational loop from time to time, it also might be useful to update it immediately after modification of some parameters.

2) During run-time a set of push-buttons can be created. A user-defined callback procedure is called immediately after the user presses or clicks a push-button. Its behavior should depend on a push-button identifier.

3) A special case of the callback is the process destructor performing some "post-mortal" actions (see above). The destructor must always be present in the code of the User Process.

3.3 Exception handling

MOON Lab enables the User Process to use structured exception handling, similar to that of CLU [1], Ada® [2] and Ada-95 [3] programming languages. The C++ try-throw-catch feature, introduced by the draft ANSI C++ standard, gives a more flexible structure for exceptions [4].

One problem of exceptions for MOON Lab is that many languages do not have this feature. Another problem is that exception handling is not compatible at the level of executable files. That means that if the User Process raises an exception and does not handle it, it cannot be handled properly within the MOON Lab Shell. This is because the User Process is a separate executable (possibly written in another language than the Shell).

This problem has been solved by implementing the fully functional exception mechanism with the use of language-independent low-level programming.

The author's contribution is a clear syntax put forward for the User Process code of an exception handler. It allows the implementation of the non-procedural exception semantics via appropriate calls to the MOON Lab API, accessible with most procedural languages.

3.4 Driver-reside semaphore

We use the MOON Lab system with CAMAC hardware, but present discussion is not specific to CAMAC and is applicable to any other interface to physical measurement apparatus.

Some of the tasks for beam diagnostics are very simple and occupy a small part of a standard CAMAC crate. So we use a single crate for several independent tasks for beam monitoring. For example, we need to visualize a set of beam loss monitors, neutron counters

---

® Ada is a registered trademark of the U. S. Government, Ada Joint Program Office

and current transformers. Several ADCs or counters occupy some of crate space left by another independent installation, interfacing via the same computer.

Each multitasking technology is faced with the problem of ***shared resources***. The typical one is a crate, shared by few tasks. The problem is that the internal state of the crate and its controller, modified by one task, should be taken into account by each other. From the other hand, a consistent multitasking technology must isolate independent tasks as possible.

A key to solution of the problem is based on Dijkstra semaphores [5]. The semaphore takes no task-specific information, but merely sets up a protected fragment of a code by opening (P) and closing (V). The protected fragment sandwiched between P and V is sure to be run by only one task at a time. Task switching is possible within the fragment, but other tasks are delayed at the call to P until first one calls V. The protected fragment is used to access shared resources.

Since the crate controller under consideration is a shared resource, the MOON Lab handles it via the special interface driver, designed to resolve concurrent calls via the single driver-reside semaphore. It turns out that it is not necessary to track out of the state of the crate or its controller, if every task follows proper semaphore usage.

3.5 Networking

In the present version all the network capabilities have to be linked in the User Process code. MOON Lab only supports a definite style of the inter-task communications via the network. This style is supported by an additional network library, a special command line parameter of the MOON Lab Shell and a respective interface parameter of the MOON Lab API.

According to this style, each User Process performing measurements can be accompanied by its counterpart anywhere on the network. It is referred to as the ***remote process***. All the interactions between the user and the User Process via the correspondent instance of the MOON Lab Shell are doubled with the remote process.

4 FURTHER DEVELOPMENT

4.1 New Windows versions

New MOON Lab versions for Windows and Windows NT are presently under development. Most important improvements are the following:

- Components for 3-dimensional graphics have been developed. These will be used for complete implementation of the MOON Lab API.
- A new model of interaction between the User Process and the MOON Lab Shell has been developed. The User Process is to be developed as a separate task. This significantly improves system performance.
- A universal network-transparent message dispatcher is being presently developed.

4.2 UNIX edition

A new project has been started. The goal is to make a MOON Lab edition for the UNIX X Window system. The User Process code should be made portable at the level of source texts.

4.3 Data file formats

A universal data format has been projected. It will support a structure of multi-dimensional arrays defined on a space of argument arrays, and a structure of dependencies. Some other data is also supported, such as version signature, denotations, comments, date, time, duration of an experiment and so forth.

Two forms of the format are projected: binary and plain ASCII. The second one, being much less compact, is human-reading and suitable for exporting to other systems.

The MOON Lab data format will cover most needs of experimental physics. It will be recognized using the MOON Lab data viewers and it will be made possible to pass it with virtually no additional explanations.

Similar data formats have been implemented for a former prototype of the MOON Lab for DOS system.

5 CONCLUSION

The MOON Lab facilitates programming of the measurement applications for beam diagnostics and gives wide capabilities and flexibility. It is a good choice for solving of several relatively small independent problems and linking them together.

The novelties of the programming technology introduced can be of interest for many fields of experimental science.

REFERENCES

[1] Barbara H. Liskov, Alan Snyder, ‘Exception handling in CLU’, *IEEE Transactions on Software Engineering* SE-5(6), pp. 546-558 (1979)

[2] Narain Gehani, ‘Ada: an advanced introduction including Reference manual for the Ada programming language’, London: Prentice Hall, 1984

[3] Annotated Ada Reference Manual. Language and Standard Libraries. International Standard ISO/IEC 8652: 1995 (E)

[4] Bjarne Stroustrup, ‘The C++ Programming Language’, 2nd Edition, Reading, Mass.: Addison-Wesley, 1991 (repr. 1995)

[5] E. W. Dijkstra, ‘Cooperating Sequential Processes’. In *Programming Languages* edited by F. Genuys, London: Academic Press, 1968