USING PROCESS-ORIENTED PROGRAMMING IN LABVIEW

Vladimir E. Zyubin
Institute of Automation and Electrometry, Russian Academy of Science, Siberian Branch
1, Prosp. Acad. Koptyuga, Novosibirsk, 630090
Russia
e-mail: zyubin@iae.nsk.su

ABSTRACT
The paper presents an implementation of the process-oriented style of programming by means of the core LabVIEW facilities. The differences of the approach comparing to the known event-driven approaches for complex control application programming are discussed.

KEY WORDS
Process-oriented programming, complex control algorithms, event-driven structure, synchronism, logic parallelism, Reflex language, LabVIEW

1. Introduction
Software plays an ever-increasing role in industrial automation. With this, the associated software cost increases, even to the point that it becomes the highest part of the total expenses. This circumstance encourages activity in the field of formal languages and techniques for implementation of complex control algorithms. In the last ten years, some of the most powerful approaches have become so popular that they change the status quo in industrial automation and noticeably press the IEC 1131-3 languages – the most prevalent formal languages for control algorithms at the moment. Ousting of the IEC 61131-3 languages is to be observed in the two directions:

- by programming techniques conceptually kindred to finite state machine (e.g. switch-technology, hierarchical state machine, process-oriented programming, quantum programming)
- by programming tools that provide not only control algorithms description but human-machine interface as well. LabVIEW is the most popular tool of the class.

Although the native language of LabVIEW is the G dataflow language, LabVIEW provides tools to integrate third-party add-ons. So, the idea to enhance LabVIEW by means that are oriented on complex control algorithms is very attractive.

Because control applications have a different ratio between the control and interface parts, it seems there is no universal approach for all possible cases. There are a number of solutions the programmer can choose from to provide a reasonable balance between effectiveness of the enhancement and difficulty of switching to the new programming pattern.

In section 2 we overview the most popular programming models to create control algorithms, the main implementations of the concepts. In section 3 we focus on the process-oriented programming style. Section 4 is dedicated to using of the LabVIEW environment in control applications and comparing of the known mechanisms to enhance LabVIEW by event-driven behaviour. In section 5 we discuss the way to implement process-oriented style of programming by the core LabVIEW facilities. Finally, in section 6 we sum up the results of the previous sections.

2. Overview
Present-day digital control system is the only way for industrial automation. The heart of such a system is programmable logic controller (PLC). PLC contains a microcomputer based control device, usually with multiple inputs and outputs, and software, which implements specific functions such as I/O control, logic, timing, counting, PID control, communication and arithmetic.

To develop a PLC program we need a formal language and design pattern based on fundamental formalisms such as a control algorithm model, conceptual means and terminology that reflect main aspects of control algorithms. In the last thirty years, a variety of different formalisms for specifying control and event-driven systems were introduced, like Hoare’s model of Communicating Sequential Processes [1], Harel’s Statecharts [2], Input / Output Automata [3], Esterel [4], Calculus of Communicating Systems [5], and their timed extensions [6, 7], to name a few. In general, each of them was introduced targeting to a specific area of investigation: reactive systems, embedded programming, logic control, parallelism, homeostatic systems, so called real-time systems, robot-technique, etc. So, for the engineer it is not always obvious which model to select for a specific application domain. As a result the industry has chosen very disputable and heterogeneous facilities of the IEC 61131-3 standard [8]: relay logic language, functional block diagram, and sequential function chart. Experts criticize the standard for the low expressiveness of the languages, their
serious limitations and yet mention the standard as an example of a misuse of standardization process [9].

The common dissatisfaction and believes that we can reduce expenses of control algorithms programming and overcome the limitation of the standard encourage many new initiatives currently being developed. The main directions of the activities are:

- styles of programming in 3rd generation languages (mostly in C or C++) [10, 11];
- 4th generation languages for PLC [12, 13, 14];
- event-driven extensions for workbenches primary targeted to using in the adjoining fields [15].

Noteworthy is the fact that finite state machines (FSM) and FSM-like models to describe algorithm are easily distinguished in all the cases. In first two cases the aspect is directly declared. In the last case National Instruments provides templates for FSMs with LabVIEW.

3. Process-oriented programming

Ease of handling is at the bottom of the FSM using popularity. FSM has a number of attractive characteristics to describe control algorithms, which includes presence of external environment, event-driven behavior, and cyclic functioning, although they are not expressed in the model explicitly [16].

Other necessary features of the control algorithms cannot be expressed by means of FSM, namely: operations with time intervals, mass logical parallelism, hierarchical structure, divergence and convergence are not supported within the model [17-19].

The cause of the mentioned problem is in the historical circumstances of the model creation [20]. The FSM model was designed for a hardware implementation and negative effects became apparent even on conceptual level, e.g. the terms “input alphabet” and “output alphabet” are obsolete and unused in IT practice for forty and more years. The FSM model is extremely hard for studying, especially under the nowadays condition of totally computerized living environment and education. This leads to common misunderstanding and discourages using of this potentially very powerful concept [21] and forces researchers make attempts to get a revision of FSM and a linguistic implementation of the model that meets all the requirements.

The hyper-process (HP) modeling framework [19] is a mathematical framework that supports the description and analysis of control systems. The fundamental object in the HP framework is a process, which is a radically revised type of state machine. The process is a polymorphic function, but in contrast to the OOP-like polymorphisms, polymorphism of process is event-driven one. So, it has several predefined functions and at any moment the process is represented by its current function. The current function is one of process’s functions. The current function can be switched by events. There are several types of events that can lead to the current function switching: timeouts, changes of variables, and switching of the current function of a process. In addition to user-defined functions, any process has two passive functions: STOP and ERROR functions that do nothing but the indication of the process execution state for outside processes.

The passive functions mechanism provides a powerful means for concurrent and coherent execution of processes, which can be organized in a hierarchical and flexible structure – hyper-process. The timeouts allow generate time-dependent events and provide synchronism of the hyper-process.

The Reflex language is a C-like programming language that provides notations for describing hyper-process as a set of processes precisely.

Practical use of the Reflex language shows:

- natural and deep decomposition of control algorithms into concurrent processes (in real projects, the number is up to a thousand processes),
- possibility to create a hierarchy corresponding to the technological description of controlled object by the structure and terminology,
- wide set of particular strategies and methods that are not present at the other branches of programming.

These notable results make us speak of process-oriented programming (POP).

POP is based on the process concept proposed in [19] that allows create a complex hierarchical structure, which is composed of logically parallel and intercommunicated processes. The process has three key features:

1) functional polymorphism, which has been called event-driven polymorphism.

2) embedded timer that is suitable for develop time-dependent behaviour in concurrent systems.

3) native facilities to monitor and change the current function of other processes those concurrent execution is organized in a round robin fashion.

As a result, the programming structure created by means of POP:

1) reflects the main features of control algorithm, namely:

- presence of an external environment to communicate with,
- cyclic and event-driven functioning,
- synchronism,
- mass logical parallelism,
- hierarchical structure.

2) has terminology that is adequate to the modern tendencies in education and IT.

3) provides easy software implementation and gives conceptual framework for reasoning about control algorithm in terms of the target technology.

Application area of POP has been researched in a number of projects.

1. The Reflex language (also known as “C with processes”) [14] has been designed for control algorithms specification. The Reflex language is an example of linguistic implementation of POP semantics in C-like syntax.
2. Applicability of POP for complex control systems has been investigated [22]. Distributed control system for a furnace used to grow silicon single-crystals with a diameter of up to 250 mm has been developed with the Reflex language. Growth of silicon single-crystals by pulling from the melt is a typical nanotechnology process, which is extremely complex for integrated automation because of multi-parametric control, multi-criteria requirements, and lack of the scale invariance. The control system must regulate several parameters. The final product must comply with a set of strong criteria. The control laws are changed during the process. In spite of the fact that this type of processes has wide spread occurrence in chemistry, metallurgy, nano-materials production, and biosynthesis, control problem solutions for these processes are very interesting from both a theoretical and practical point of view. The developed event-driven algorithm consists of about 900 weakly connected processes. Maximal latency of the system on external event can be decreased down to 10 ms (the further decrease is limited by the I/O updating procedures).

3. Applicability of POP for mass service systems has been investigated [23]. Common Lisp Process System (CLIPS) was introduced as a POP extension of the LISP language targeted on mass service systems. The features of POP on CLIPS are:
   - possibility to transfer parameters at the start of process;
   - floating period of the hyper-process activation and full utilization of the computing system resources.

4. Process-oriented enhancement of the ST language from the IEC 61131-3 set [24] has been introduced and investigated. The ST language has been extended by the "process" and "function-state" program components, and by operations with time intervals. As a result, possibility to describe parallelism, event-driven polymorphism and synchronism has appeared in the extended ST language. The introduced features allow programming of a wide class of control algorithms in linguistically homogenous environment, without using of the other IEC 61131-3 languages (namely, SFC, IL, LD, and FBD).

5. A set of virtual educational boards for practical training on applied programming of control algorithms has been created. The Reflex language is used as the formal notation for control algorithms. Controlled objects and their animation are simulated by means of the native LabVIEW environment. Control algorithms written in the Reflex language are translated for encapsulation in the LabVIEW environment by means of either the formula node mechanism (C-like notation) [25] or the Python interpreter [26]. The approach drastically improves the way students learn. Rather than focusing on sometimes-tedious work with physical model of a controlled object, educators and students can focus on results and concepts of control algorithms programming. The source code in the Reflex language is translated in the Python language, and then is executed by the interpreter.

The LabVIEW environment is attractive because of the native WYSIWYG-means allowing create end-user interface and advanced means allowing integrate third-party equipment. In many cases these possibilities compensate imperfection of the G language based on the dataflow concept.

The idea to extend the base LabVIEW toolkit for control algorithms and event-driven cases is not new. A large amount of related work has been published over the years. The attempts were inspired by the fact that complex control algorithms cannot be easily implemented with neither the G language nor a LabVIEW add-on FSM module.

4. LabVIEW: event-driven extensions for complex control systems

To solve the problem, how to enhance LabVIEW with event-driven structures, several approaches are being used in the practice:
   - the LabHSM toolkit – seamless extension integrated in LabVIEW with undocumented features [17];
   - manually created code integrated by means of the Formula Node Express VI [27];
   - automatically created code integrated by means of the Formula Node Express VI [28, 25];
   - integration by means of the dll-mechanism;
   - seamless integration by means of a third-party interpreter [26].

The summary characteristics of these approaches are shown in Table 1.

As one can see in the table, in the case of serious necessity for process-oriented behavior we can choose an appropriate extension of the native LabVIEW toolkit to better meet the specificity of the project to be solved. While the complexity of the event-driven part of a developed control algorithm is notable but not high, we can reasonably try to reject third-party add-ons at all, especially in the case we ought to intensively use the LabVIEW library functions. Moreover if the last circumstance appears in a project we cannot use the Formula Node Express VI as well. And there is no programming pattern for that case (see Table 1).

5. Process-oriented programming by the native LabVIEW toolkit

For the case when project both demands frequently using of the LabVIEW library functions (VIs) and does not allow third-party add-ons we introduce process-oriented style of programming by means of the G language, i.e. a way to construct the event-driven part of program as a set of intercommunicating processes.

As described in [29], recommended FSM implementation formulated in the state terms is the following. Each state in a state machine does something unique and calls other states. State communication depends on some condition or sequence. To translate the state diagram into a LabVIEW diagram, the programmer needs the following infrastructure:
   - While loop – continually executes the various states,
Case structure – each case contains code to be executed for each state,
Shift register – contains state transition information,
Transition code – determines the next state in the sequence.

The flow of the state diagram (Fig. 1) is implemented by the loop, while the individual states are replaced by cases in the case structure. A shift register on the while loop keeps track of the current state, which is fed into the case structure input.

There are four common methods to determine which state to transition to next. All of them are based on the using of an enum constant as it is shown on the figure.

Unfortunately, the implementation leads to the enum constant modification problem. Enums are used as case selectors in state machines. So, if the programmer attempts to add or delete a state from this enum, the remaining connected wires to the copies of this enum will break. This is one of the most common obstacles when implementing state machines with enums that makes event-driven programming very

<table>
<thead>
<tr>
<th>Name</th>
<th>Examples</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
</table>
| Third-party add-on modules for LabVIEW | LabHSM toolkit | • unified scope and seamless using of variables for interface and event-driven parts of the program
• error detection
• low processing costs | • using of undocumented features in the toolkit and possible incompatibility in future
• the necessity to use the LabVIEW IDE to modify control algorithm |
| Manually created code integrated by means of Formula Node Express VI | Manual implementation of the switch technology or process-oriented programming in C | • absence of necessity to use third-party software
• low processing costs | • the necessity to use the LabVIEW IDE to modify control algorithm
• Modbus-like (provided by data layer) data exchange between control and user interface parts of algorithm
• lack of event-driven error detection and code reliability problems |
| Automatically created code integrated by means of Formula Node Express VI | Visio2switch toolkit, Reflex2CFN translator | • error detection
• low processing costs | • the necessity to use the LabVIEW IDE to modify control algorithm
• Modbus-like (provided by data layer) data exchange between control and user interface parts of algorithm
• the necessity to use third-party commercial software (in the case of Visio2switch toolkit) |
| Integration by means of a dll | Reflex2C translator or manual implementation one of the event-driven approach | • possibility of creation free of charge applications (independency from the LabVIEW IDE)
• event-driven error detection (in the case of automatic implementation)
• low processing costs | • the necessity to use a third-party toolkit to modify the control part of algorithm (e.g. Visual C)
• Modbus-like (provided by data layer) data exchange between control and user interface parts of algorithm
• for manual implementation: lack of event-driven error detection and code reliability problems |
| Integration by means of a third-party interpreter | Virtual laboratory board toolkit (based on the Reflex2Py translator and the Python interpreter) | • possibility of creation free of charge applications (independency from the LabVIEW IDE)
• on-line seamless modification of code
• error detection | • time-consuming interpretation
• Modbus-like (provided by data layer) data exchange between control and user interface parts of algorithm |
uncomfortable (besides of the discussed problem of algorithm presentation with single FSM).

One of the known solutions is use of a string control to select the cases. The select case structure does not need to be synchronized with the select variable. This method does require a 'Default' case. It always contains a dialog box that generates a trap message to indicate the problem.

The proposed solution of the POP implementation by the native LabVIEW toolkit is shown on Fig. 2. As well as in the FSM implementation, every process is implemented with the case structure and the string variable of process function (SVPF), whose value consists of a current function name of the process (the “PROC P1” … “PROC P6” variables on Fig. 2). The passive functions of a process are implemented with the “STOP” and “ERROR” marked states (PROC P5 and PROC P6). Default value of SVPFs is “STOP”. To launch a process we must transform the SVPF value (or its local copy) to the “Start” word, which is reserved for the marked first function of process (PROC P1).

Execution control of a process is implemented by monitoring of the SVPS value for the “STOP” or “ERROR” values (PROC P2). Cyclic functioning and the activation period of processes are provided by the while loop VI and the ‘Wait Until Next ms Multiple’ function. Figure 2 (PROC P3) demonstrates the "Timeout?" function using the standard implementation of synchronism for the hyperprocess base model. Using of the millisecond timer VI is acceptable as well.

There are the following problems the manual implementation has:

- impossibility of error detection for closed-loop functions and inaccessible functions,
- a lot of implementation specific operations that hinders the programmer in focusing on results and concepts of control algorithms programming,
- relatively low readability of processes whose field of view is limited to the one function only.

To trap a use of a not-specified function name, the case structure must contain a 'Default' case that generates an error message to indicate the problem (see PROC P3).

The possibility to use arbitrary language for function names sufficiently simplifies the analysis and modification of algorithms for non-English-speaking programmers.

Among nuances of the implementation, we have discovered the difficulty to construct hierarchies with disturbed tree-like structure (e.g. in the case there is a necessity to change the current function of the parent process). Seemingly, the problem is caused by internal details of local variables implementation in LabVIEW. The problem can be solved only partly, by using of logical variable for state selector of a case structure that
The proposed implementation of the process-oriented style of programming by means of the G language was successfully used in devices for automated testing of electric circuits by the adaptor method.

6. Conclusions

Complex control algorithms can be implemented by using various techniques in LabVIEW. The introduced method makes it possible to create and maintain complex control algorithms in LabVIEW in the process-oriented style of programming as a collection of weakly-connected processes.

Comparing to the main programming patterns applied for design of complex control software in LabVIEW we empathize the following features of the method:

- flexible hierarchical structure of algorithm in the process-oriented style,
- unified scope and seamless using of variables for the interface and event-driven parts of program,
- independence from third-party toolkits and add-ons,
- low processing costs,
- possibility to use non-English identifiers.

Practical use proves that the control systems designed with the pattern are reliable, maintainable and, at the same time, inexpensive one.

References


[18] M. Samek, Déjà vu, C/C++ Users journal (June), 2003


[24] V.E. Zyubin, The ways to expand the ST language from IEC 61131-3 for industrial automation tasks, Devices and Systems (2) 2009, 29-33


[27] K.V. Vavilov, A.A. Shalyto. LabVIEW and SWITCH-technology // Industrial Control Systems and Controllers (6), 2006, 43–45


[29] Application Design Patterns: State Machines, NI Developer Zone (Oct 23), 2006. [URL: http://zone.ni.com/devzone/cda/tut/p/id/3024]