Automation of industrial serial processes based on finite state machines

A. Jamhour\textsuperscript{a}, C. Garcia\textsuperscript{b} a*

\textsuperscript{a}PETROBRAS, Excellence Center of Technology Application in Industrial Automation, São Paulo, Brazil
\textsuperscript{b}USP – University of São Paulo, Polytechnic School, São Paulo, Brazil

Abstract

An automation solution based on finite state machine (FSM) for a flow measurement testing unit is proposed. This FSM was able to automate not only the execution of one test but the whole sequence of tests. Commercial software and hardware tools were employed. As a result an unattended, continuous, high productivity and low cost operation was achieved. This solution can be applied to automate other serial and batch processes.

© 2012 Published by Elsevier Ltd. Selection under responsibility of the Congress Scientific Committee (Petr Kluson) Open access under CC BY-NC-ND license.

Keywords: Finite state machine (FSM); process automation; batch automation; serial automation.

1. Introduction

Bench units for studies and development of equipment and processes demand that a great number of tests are conducted even for the simplest cases. These tests would generally involve several repetitions. The testing unit cited in this paper is not a bench for the calibration of flow meters. It is a unit to develop inferences for flow measurement. There are many sensors that collect signals from different components of the unit. These sensors, the valves and the fittings are replaced or repositioned between the sequences of tests. It is very difficult to return the unit to the exact conditions in which it was before a test was run and changes in its configuration were performed. For example, to ensure that the sensors are positioned in the same place. For these reasons and to suppress the need of assisted operation it was necessary to

* Corresponding author.

E-mail address: jamhour@petrobras.com.br
develop a system capable to identify and repeat tests that did not meet the minimum established parameters, minimizing the rejection of tests during the analysis stage.

So it was developed a system, controlled by a finite state machine. The achieved automation allows that, after planning the experiment and insertion in a database of the desired conditions for each test, the unit can execute all these planned tests in an optimally, unattended and uninterrupted sequence.

2. A state machine

A finite state machine (FSM) also called state machine or finite state automaton is a particular type of Petri nets, a mathematical model, used to model a large number of problems. It is an abstract machine that can be in one of a finite number of states.

State Transition. The machine changes to the next state when a triggering event occurs. This is called a transition. There are transition functions that map one state to the next state.

Current state. The machine is in only one state at a time. Current state is the state the machine is at any given time.

More definitions can be found in [1] and [2], and introductory texts in [3], [4] and [5].
A finite-state machine can model engineering and biological applications, among which are:

- communication protocol design,
- electronics design,
- automation,
- to describe grammars of natural languages,
- to describe neurological systems and many others.

In the present case a state machine is used not only to model the execution of a test, but to model and automate the whole testing sequence.

3. Description of the bench unit

A simplified diagram of the unit is shown in figure 1.

The bench unit consists of a fluid input selector, primary pressure controller, reference flow meter, control valves, sensor devices and output alignments.

Currently the unit only works with air or water and it is being used to develop inferences for flow measurement based on cyclic pressure oscillations caused by the valves and by the pipe fittings that are installed.

When air is aligned, the primary pressure controller is a pressure reducing valve (PRV). When the unit is operating with water, the pressure control is made indirectly by the speed variation of the water pump.

The solenoid valves shown in the figure 1 allow to select the control valves to be used in the test. These valves have different sizes and different internals and are changed to extend the ranges in study.

A reference flow meter (type Coriolis) and the valve that is aligned for the test are used to set the flow rate. The signals are collected by many sensors installed in the unit.

Test procedure. The test procedure includes the following steps:
1. selection and installation of the hardware (valves, fittings, instruments and sensors),
2. adjustment of the unit until a steady flow be established,
3. storing of the readings of the sensors, of the reference meter and of the other process parameters to be further analyzed.
A phase. The mesh of tests to be run is generated based on the parameters that define the operational window to be studied. These parameters with the specific conditions of each test are called phases. The phases are stored in a database.

A sequence of tests. The phases for the same hardware configuration of the unit are grouped in the database with the same code. A sequence of tests is the execution of the tests with the same code.

According to the conditions of the unit, a new phase is selected, the conditions are automatically adjusted, and once achieved, the test start to run.

During the test all manipulated variables are maintained as stable as possible and the signals are captured by the sensors and recorded at 50 kHz acquisition rates for post processing. Also some averages and standard deviations are calculated and compared to limits previously established by the user. Once the levels of acceptance are reached, the test is completed and a new phase is selected. If the levels are not met, a new test with equal conditions will be repeated until a maximum number of times, after which the phase is canceled and another automatically selected.

The set-points and the conditions where the tests were performed, as well as a continuous record of the conditions of the unit are stored in a database. The signals detected at high frequency are stored in binary TDMS files [6] for further processing.

4. System overview

The main components of the system and some of the functions they perform are shown in figure 2. The state machine runs on the CPU in a parallel process to the main program and it is responsible for controlling the execution of the sequences and for starting other processes.

5. System hardware and software

5.1. Hardware

The data acquisition cards used were National Instruments NI92XX series. These cards are all installed in chassis and connected by USB ports to a CPU. The minimum required components for implementing the system are the field devices, the data acquisition cards and a computer running the software.

Fig. 1. Simplified diagram of the bench unit
In addition to the traditional analog communication, other protocols are also possible such as OPC or Ethernet.

5.2. Software

5.2.1. Database.

In this architecture a database needs to be integrated into the system as it responds to various queries made by the main program such as the selection of the next test to be run. The information below is stored in the database:
- cause/effect diagram that interlocks the unit and assists in the safety of persons and equipment
- set-points of the variables to scan the operational window in study
- parameters for tests pre-acceptance;
- results of the tests collected at low frequencies
- operational conditions of the unit
- event log.

5.2.2. Planning of experiments

Any method or software for the planning of experiments can be used. It is only necessary to store all the phases, the set-points of each test to be performed, in the database (see phases definition).

5.2.3. Development of the system.

For the development of a system based on a state machine free and commercial software like [7], [8] or a FSM generator, like [9], [10] and [11], that generates codes for common programming languages can be used. The direct specification for some object-oriented languages [12] can also be used.

The developed system integrates and controls all phases of testing. It is a program that automatically switches between the different phases, including:
- reading of the conditions of the unit
- selection of the test to be performed
- adjustment of the unit
- execution of the tests
- preliminary analysis of the test in real time
- storage of low frequency data in the database
- storage of High-frequency data in binary files.
5.2.4. Visualization of data and data processing

The data acquired at high frequencies were stored in binary files. For viewing and processing of these files it was used a proprietary software [13], also able to export these binary files for analysis to other platforms.

6. The goal of the system

6.1. From planning to the results of the tests

In a briefly description, a test is a set of measurements that are recorded when the unit is operating in steady state under predefined conditions. The complete sequence from planning to execution of a series of tests can be summarized as:

6.1.1. To set the test conditions:
- fluid selection
- hardware selection: alignments, valves, sensors (type, number and position of sensors)

6.1.2. To adapt the unit:
- to install sensors and other instruments and equipment
- to perform loop-tests

6.1.3. To establish the safety and operational limit conditions of the unit:
- to set levels of pressure, temperature, flow, speed and others to be observed, based on data from equipment, instruments and other operational limits of the unit
- to run preliminary exploratory tests to avoid searching areas where conditions are unfeasible
- to adjust the operating limits of the unit and the cause-effect diagram, feeding these data in the corresponding tables in the database.

6.1.4. Planning experiments:
- to set within the limits identified in the previous step, the operating conditions of the tests to be performed
- to detail the experiments using techniques and software that are known and dominated by the user in order to scan the operating window under study
- to insert in the database the conditions under which the tests shall be performed.

6.1.5. Running the tests:
- to align the fluid and start the unit
- for each of the phases set on the step (6.1.4) :
  - to adjust the operating conditions
  - to store the tests data.

In addition to the physical changes to the unit, there are many other conditions that can be adjusted on a test, such as the inlet pressure, the flowrate, the valve opening, the outlet pressure, the fluid to be used and others, enabling the implementation of many tests.
6.1.6. Collecting test data:
- to collect data stored in the binary files and in the database and export them to the formats necessary to be analyzed

6.1.7. Analyzing data and issuing the corresponding reports:
- It is often necessary to repeat some tests or to refine the mesh for confirmation of results, i.e., to return to step (6.1.4).
  The purpose of this system is to automate the step (6.1.5).

6.2. User actions after the implementation of the system

Figure 3 summarizes the operation of the main program, with emphasis on the tasks performed by the user. Note that the interventions by the user listed in 2.3.2 and 2.4 are for spurious changes in controllers, for graphically displaying of the process variables, to prior break the cycle of testing or to handle error or interlock situations of the unit. To run a whole sequence of tests, if no abnormalities occur, nothing more needs to be done by the user beyond inserting the sequence code and starting the test cycle.

7. System details

The system was developed in LabVIEW. As described in section V, many other programming languages could be used. In this section some parts of the program and some relevant aspects of this system will be cited.

7.1. The states of the FSM

The core system which controls transitions between the states is the state machine. Figure 4 shows the states created for automating the execution of the sequence of tests for the unit shown in this case and the various transitions between them.
Following the transitions it is possible to understand how the tests are selected, implemented and pre-analyzed. When a test is validated the phase is completed, otherwise, the test is automatically reprogrammed to be repeated. The above cycle is repeated until all the provided phases have been completed. Using the same principle one can develop state machines that achieve equivalent levels of automation in other processes, even for more complex units. The success of a similar project will depend on a conditional analysis that prevents the system to be trapped in "dead-locks" and that properly treats, if not all, most of the situations that could occur. In [14] and [15] there are explanations of state minimization techniques that can assist in developing more complex state machines.

7.2. Internal data flow

7.2.1. Local and global variables

Data used only in the execution of the routine are stored in local variables. When an information needs to be exchanged between the various processes that run in parallel or subsequently be employed, it is stored in a specific repository, the repository of global variables. Figure 5 shows a small part of the globals repository. A repository serves as a memory pointer and can even be displayed during program execution, while the variables are dynamically updated.
Global variables and data stored in the database are the only mechanisms for exchanging data between processes, which greatly simplifies the development and debugging of the program. Data read from OPC servers also communicate through global variables.

7.2.2. Data from the database

Many values, such as the parameters that define the operating conditions and those for the acceptance of the tests are read directly from the database. When the same value needs to be used in different parts of the program it is read and assigned to a global variable. When its use is punctual, the access is made internally by the routine that needs it in a similar way as a local variable.

8. System integration with database

Event records are just log files, however when an event occurs, a dedicated subroutine receives the event code and, respecting the cause and effect diagram (that is stored in a database table), the pre-defined actions are automatically performed, such as stopping an equipment or opening a valve and even the full stopping and depressurization of the unit. This error handling in a targeted manner greatly facilitates the maintenance of the program and updates in the interlocks, mainly due to the modifications and revamps of the unit. As an example of the integration with the database, figure 6 shows the subroutine that checks if the boundaries of the unit, that do not require manual reset, have been violated. It is compact and does not demand updates, even when new events are created. The limits are read from the database only once at the beginning of the routine that keeps monitoring the unit in loops until it is turned off. This and other features were based on commercial distributed control systems (DCSs). These implementations were possible due to the integration of the system with the database. This figure also exemplifies the modular programming: the routine identifies when a threshold is violated and when it returns to normal; the treatment of the event is done by another routine (i.e., in another module) that is called internally by this one. Figure 7 shows a simplified version of the steps 2 and 12 of the FSM. In these parts of the code it is possible to identify that local and global variables and that queries to the database are used in the internal decisions taken. These decisions generate the transitions among the states that emulate some of the actions that would be taken by an operator. In the serial processes these actions are predictable, which allows a high degree of automation, as in this presented case.
9. Advantages and disadvantages of this method

9.1. Advantages

- Low cost of hardware and of software development.
- Security requirements of the unit can be easily incorporated.
- It is not necessary to use hardware from a single brand or series. The requirement is that they have drivers for communication with the system.
- By including a database and a FSM in the control structure an almost complete automation of a bench or of a small unit can be achieved. The FSM program structure is modular and can be easily adapted when changes in hardware occur or for use in other units.
- It was possible to incorporate into the system some of the distributed control systems (DCSs) facilities, such as event handling based on cause-effect diagram, log of alarms and of process data historian.
- If a CPU is not sufficient to control the entire process, multiple CPUs can be interconnected and the processes can run in parallel. Physical and logical trigger events can be used to synchronize these processes.
- The system capacity can be greatly expanded by the choosing of hardware and software able to exchange data via OPC, Ethernet or other digital protocols.
- Performs analysis and stores data in real time.
- Optimization tasks can be implemented in the FSM to increase the overall performance, like the pre-acceptance and automatic repetition of tests cited in this case.
- Executable programs can be compiled. Running them in the field CPUs implies that there is no risk of alteration or unauthorized access to the code.
- The main parameters, such as the operational limits of the unit, are stored in the database and not in the structure of the program. This minimizes the need to alter the code and to recompile the executable.
9.2. Disadvantages

- The first implementation is difficult and may not be suitable for situations where a few tests will be conducted since it demands a great effort to be developed.
- Demand a little knowledge of software (of SQL and of the programming language chosen to build the FSM).

10. What software should be employed?

Due to the large number of databases and FSM generators one may ask: which set of software may be used to develop an equivalent system? And the expected answer should be: the programming languages that you know well. This applies to the database, but not necessarily for the FSM development language. The reason is that some data acquisition cards do not have drivers for communication with other languages. Then the development of the system needs to begin with the learning of the language. Fortunately the most of these FSM oriented programming languages are easy to learn and to debug, very productive and most importantly, are adequate to control and automate even the most complex processes.

11. Conclusions

In this paper were presented some aspects of an adequate structure for the automation of sequence in serial processes, like batch units or units for testing. Depending on the process knowledge, available resources and on the creativity and skill of the programmer, a high degree of automation can be obtained employing commercial data acquisition cards. There are several possible applications in a same industry or laboratory, and once implemented in one unit, the effort to adapt the system to other units is greatly reduced because the FSM architecture is modular and a large part of the code can be fully reused.

In this real case, to the unit for which the system was developed, it became possible to run several tests with a high approval in a period much lower than expected. The inclusion of monitoring of the safety and of the process conditions increased productivity and reduced operating costs by allowing unattended, continuous and secure operation. Gains were also noticed due to the implementation of an internal mechanisms for pre-acceptance of the tests that reduced the need for repetitions after further analysis.
References


