DESIGN AND IMPLEMENTATION OF A FINITE STATE MACHINE QUEUING TOOL FOR EPICS

J.A. Perlas, D. Beltrán, J. Rosich
Synchrotron Light Laboratory, LLS-IFAE
Edifici Cn, Campus UAB, E-08193 Bellaterra, Barcelona (Spain)

Abstract

In the process of making a detailed design for a synchrotron accelerator in Barcelona, we have designed and implemented the control system for a magnetic measurements system [1], conceived as an evaluation for the accelerator control system which is planned to use the EPICS toolkit [2].

In this article, we introduce a new sequencing tool (called *Finite State machine Queuing Tool* or *FSQT*) and we discuss its design and its advantages using a new implementation of our test bench applications. *FSQT* has born from the limitations of the equivalent EPICS tool (the *Sequencer*).

1 WHY A NEW SEQUENCING TOOL

Although we have found the Sequencer [3] to be a very useful tool for small applications, the necessity of developing a new design arises mainly from its limitations. These are listed below:

- It is not possible to determine the new state *dynamically* (during the execution of the current transition).
- A new state is required for the introduction of a timeout timer.
- Accesses can be made via the Channel Access (CA) facility in either synchronous or asynchronous mode, but this can only be decided at compilation time.
- The "put" to a Process Variable (PV) in asynchronous mode does not provide any function that indicates the completion of execution.
- There exists an intermediate "engine" (i.e. the Sequencer) between the Operating System (OS) and the code. This complicates the debugging phase.
- The C-language code can be "escaped" in an application but this makes it much less readable. In addition, there are strong limitations in the inclusion of subroutines and system calls. Furthermore, the syntax used for CA accesses inside subroutines is different to that used outside them, making very difficult the readability and reusability of the code.
- It is not portable.

After analysis of all these issues and considering the desirable features that a minimally powerful sequencing tool has to provide, we arrived at two options: *a)* upgrade the Sequencer, or; *b)* create a new tool. Taking into account that there are intrinsic limitations in the current sequencing tool, which are not possible to overcome (mainly because of the decision to provide its own language/syntax) our decision has been to create a new tool, which we call *FSQT*.

2 THE *FSQT* DESIGN CONCEPTS

2.1 General Background

The main idea in the new design is to provide a centralized *manager* that *listens* to all the stimuli sources (internal and external) of a real-time program and acts accordingly and chronologically upon them.

Very often in a program it is necessary to wait for *events* that are generated from several sources. When the arrival of one such event takes place, it is desirable to take action to perform a specific task. To standardize the process of generation, waiting and management of events, the *FSQT* tool has a centralized (and prioritized) FIFO queue where the different generated events in the system are received, waiting for later treatment. In this way, to each of the stimuli of our system it will be assigned a different "facility" that will allow to univocally identify the stimulus that generated the event. These facilities will be inserted (with priority) by the "stimuli generators" in an unique queue, where they will remain stored until their extraction and subsequent execution of their associated actions. In this way, all the stimuli are processed in a chronological way in the order in which they were generated avoiding, in addition, any loss.

For an application of medium or high complexity, it is desirable to implement it using the Finite State Machine FSM approach [4]. In this case these stimuli can invoke transitions between the previously designed states of our State Diagram.

2.2 FSQT design requirements

In the *FSQT* conception process, besides solving the Sequencer problems, we imposed the following requirements:

- To use a standard programming language like C and thus avoid the creation of a special syntax.
- The tool has to be well layered, containing independent support libraries for the different functionality that it has to provide. An application

program can use all or part of these support libraries.
- Portability across different platforms.
- The package has to include a graphical tool to edit the State Diagram and generate the source code.
- To support nested states (according to Booch [5]).
- Reusability for other control systems different to EPICS.

3 *FSQT* IMPLEMENTATION

3.1 Tool components

Following the above philosophy and requirements, the *FSQT* tool has been implemented as three completely independent C-libraries. They have been ported to several OS and their APIs are very well defined. These libraries are:

1. Queue action library (*fsqt_qa*). This centrally manages the different events generated in the system and allows the execution of a routine that is associated to every possible event.
2. FSM library (*fsqt_fsm*), which supports the hierarchical implementation of State Diagram based programs.
3. Channel Access library (*fsqt_ca*). This is a simple interface with the EPICS CA Client layer used to establish communication with the Process Variables, both synchronously and asynchronously, including transaction-completion functions.

Fig. 1 shows the *FSQT* software layer design and how it is integrated inside the EPICS architecture.

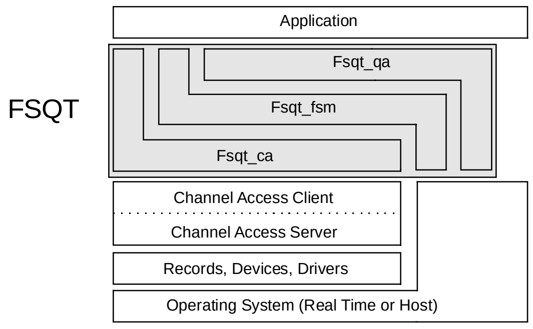

Figure 1. *FSQT* software layers.

One of the implementation choices for these libraries has been the use of the Posix 1003.1b standard wherever possible to ensure that the code is portable to any other RTOS. Where this has not been possible, a platform-dependent library has been used. Nevertheless, our intention is to eventually use the EPICS OSI Layer [6] that is currently being developed.

All these library functions include a VxWorks-like error code system. These error codes are associated to a character string that explains the cause of failure.

The *FSQT* libraries also contain initialization and cleanup functions that, due to their autonomous and modular design, must be called from the application. It is important to note that in spite that a full-featured *FSQT* application should use the three libraries, those are provided as “building blocks”, in the sense that any application may use only one or two of them, e.g. a non-real-time application in a Unix host machine, for a non-EPICS control system could use only *fsqt_fsm* to do the sequencing of user input data in an accelerator operational procedure.

This set of libraries is complemented with a graphical tool (called *GFSQT*), implemented in Tcl/Tk that allows, in a comfortable and simple way, the graphical design of Statecharts, and the generation of the associated C code. An example is shown in Fig. 2.

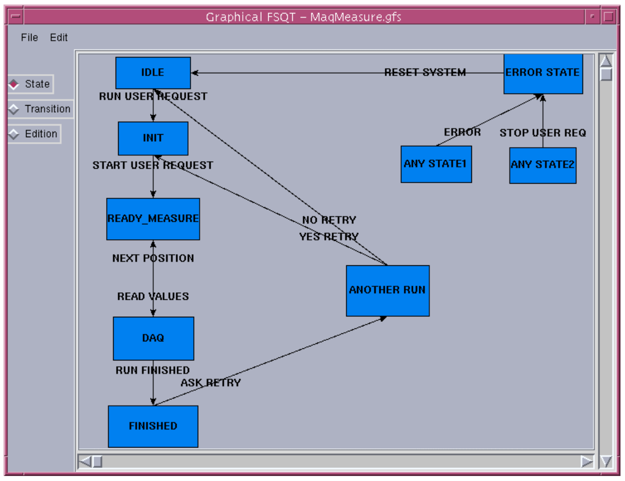

Figure 2. Graphical *FSQT* example.

3.2 CA performance

One of the important aspects for any re-design is to check that the performance of the new software (having more functionality) has not been degraded.

The performance in the CA access times to the PVs in the current *fsqt_ca* version has been evaluated and compared with the Sequencer. Using the hardware timers in the CPU board, we have measured the mean times for the remote access (gets) of software PVs, for both the *fsqt_ca* functions and the Sequencer equivalents, making a comparison. The results are shown in Table 1.

Table 1. Comparison of average remote (get) times (*msec/PV*) between *FSQT* and Sequencer.

	Synchronous	**Asynchronous**
FSQT	4.58	2.77
Sequencer	4.76	2.71

This test was carried out in a 68040 CPU board accessing to 100 PVs residing in a remote database. We can clearly see that the *fsqt_ca* library doesn't introduce a significant overhead, obtaining similar access times to those using the Sequencer.

4 ADVANTAGES OF THE *FSQT* DESIGN

FSQT has solved most (and potentially all) the problems of the Sequencer and it further extends its functionality. This is summarized in the following enhancements list:

Major additions:

- Permits *hierarchical* state diagram designs.
- *Portability* to be executed in any (supported) host computer.
- Includes a *graphical* FSM editor with skeleton-code generation.
- Optionally permits several state machines share a single task.
- It is much *easier to debug* applications (no intermediate "engine" present, and no multi-tasking used).
- Provides full C-language support, including ISRs.

Minor additions:

- Allows *pre* and *post*-actions to be executed.
- Supports Channel Access *put* with *callback*.
- Allows an action not to reset timers.

5 CASE STUDY: THE MAGNETIC MEASUREMENTS SYSTEM

Our "old" Sequencer code for the magnetic measurements system has been ported to *FSQT*. A partial State Diagram of the new code can be seen in Fig. 3.

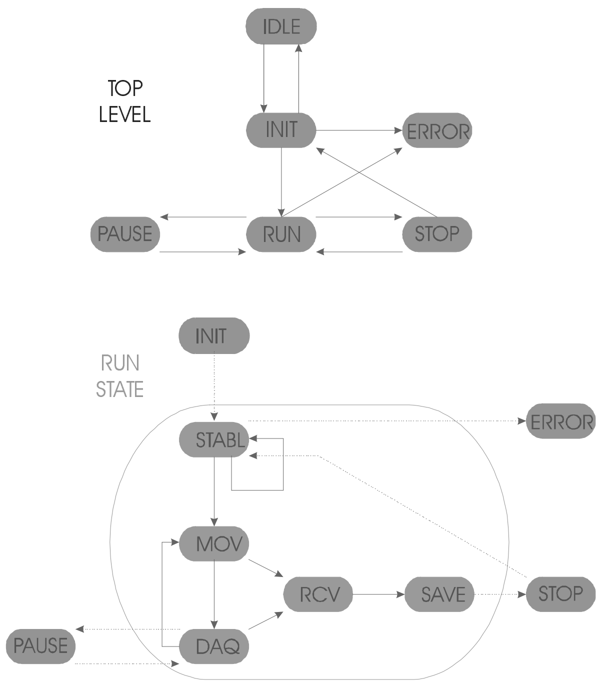

Figure 3. Partial view of the application FSM.

This experience has been very useful to demonstrate the big differences over the old version. These are:

1. Programming simplicity (through hierarchies and the use of the graphical editor).
2. Faster reaction time to user intervention.
3. No loss of events.
4. Same CA performance.
5. Some applications now run in the Unix host.
6. Code more readable and maintainable.
7. Better code quality through the use of sub-routines (avoiding duplication, resulting in 25 % less code).
8. Allow full use of OS resources.

6 CONCLUSIONS

We have performed a software analysis and produced a new design and implementation for a new sequencing tool (*FSQT*). This overcomes the problems of the current EPICS Sequencer and adds much more flexibility and power to future sequencing applications, both for EPICS and non-EPICS.

Amongst the most important features, we emphasize that it supports hierarchical state machine designs, it is portable and runs in the host computer, it includes a graphical FSM editor with C-code generation, it is much easier to debug applications, and finally, since no proprietary syntax is used it has the support of the C-language and any other third-party C libraries.

The CA accesses performance of *FSQT* is as good as that of the Sequencer and it has proved very satisfactory for controlling our magnetic measurements system, after porting the software that was using the EPICS Sequencer to this new approach. Big improvements have been attained in programming simplicity, faster reaction time to user intervention and no loss of events. Furthermore, using the graphical tool and the portability to the Unix host, applications for prototyping and testing have been developed in record time producing better code quality.

The testing of the *FSQT* tool is practically completed and we intend it to be the next generation of the EPICS sequencing tool. Although the library support is well developed and tested, the *GFSQT* application is still under development and will be essential for the proper support of hierarchical statechart designs.

REFERENCES

[1] D. Beltrán, J.A. Perlas et al., "The LLS magnet test facility...", EPAC'98, Stockholm, June 1998.
[2] L.R. Dalesio et al., "EPICS: past, present and future", NIM A, **352** (1994) 1-5.
[3] A. Kozubal, W. Lupton, Sequencer v1.9.2 (1996).
[4] D. Harel, "Statecharts: a Visual Formalism for Complex Systems", Sci. Comp. Prog. 8 (1987).
[5] G. Booch, "Object-Oriented Analysis and Design with Applications", Ed. Addison-Wesley, 1994.
[6] M. Kraimer, Operating System Independent Layer for EPICS, http://epics.aps.anl.gov/asd/controls/epics.