USING WINDOWS MESSAGING TO CONTROL AUTOMATIC TEST EQUIPMENT

James R. Evans
Software Engineering Division (TISAD)
Ogden Air Logistics Center
Hill Air Force Base, Utah 84056
Phone 801-777-4527
jevans2@sisna.com or
evansj@software.hill.af.mil

James C. Lisonbee and Lloyd G. Allred
Software Engineering Division (TISAD)
Ogden Air Logistics Center
Hill Air Force Base, Utah 84056

Abstract - Traditionally, Test Program Sets (TPSs), written in ATLAS for Automatic Test Equipment (ATE), were developed for use with sequential, single tasking systems. For the F-16 Analog Test Station Sustainment (FATSS) project, TPSs were implemented in a separate thread, enabling the operator to activate controls on the Test Executive while the TPS thread executes. When the operator activates a control on the Test Executive screen, a window message is sent to the TPS thread, causing it to be suspended, single stepped, or aborted. Without multi-thread support, the operator would be forced to wait until a task completes before making changes. The operator would also be forced to reboot the computer or kill the test session in order to stop a test from executing. Under runaway burn conditions, or with extremely long delays, waiting for a task to complete becomes unacceptable. Legacy system flow controls such as the Auxiliary Keypad on the Honeywell 2600 used hardware interrupts to control the flow of the TPS. These controls are now implemented in software on the Test Executive screen. Using this method, the operator will be able to cancel the program execution by pushing the Station Clear button, or to change flow control values such as Repeat Test, or Hold on NoGo buttons, while the translated ATLAS executes in a separate thread. Rehosting these TPSs to a windows environment also enables us to attach new control devices to the system such as the mouse and voice control.

INTRODUCTION

Under the National Instruments Test Executive Version 2.0, when a segment of code is called, the user interface loses all control of the flow of the software until the end of the code segment is reached. This was designed for executing short segments of code, probably communicating directly with a piece of test equipment. The user cannot alter or stop the flow of the code, and has no indication of the progress of the code until the test is finished. This situation is unacceptable when there are long delays and multiple tests built into the code being executed. Some of the delays built into the legacy ATLAS code are as long as 3 minutes, waiting for equipment to warm up. With one of our primary goals in mind of having little or no impact on the TPS coding, the best approach to solving this problem was to have the translated ATLAS code execute in a separate thread from the Test Executive. This approach allows the user to interact with the Test Executive screen even while the code executes in the separate thread.

THE USE OF WINDOWS MESSAGING

Under the system developed for the FATSS test station, the Test Executive makes a call into a TPS_Main function, passing it the line number of the ATLAS code to start executing. The TPS_Main function launches a TPS thread, which makes calls into the translated ATLAS code. When the TPS thread is launched, it sends a message to the Test Executive, registering its own handle with the Test Executive. This gives the Test Executive the information needed to suspend or terminate the TPS thread at any time, even during execution of a call to an instrument driver. As the translated ATLAS executes, it sends messages back to the Test Exec querying for the current status of all screen controls. If the operator has changed one of the controls, such as turning on the Single Step or Hold On NoGo switches, the ATLAS can respond and start behaving in that manner. The TPS_Main function executes a loop waiting for the TPS_Thread to finish. When the TPS_Main function is complete, control is then returned to the Test Executive which updates the status of the test on the screen and allows the user to call the next Entry Point.
In the legacy ATLAS code, there are calls to hardware peripherals such as Clear Screen, and Print, which send commands directly to hardware on the Honeywell 2600. Most of these commands will now need to be directed to the new Test Executive. During the process of translation from ATLAS to C, all of these hardware IO functions are replaced by calls to a group of IO functions, which send messages to the Test Executive. As the code executes in the TPS thread, when one of these commands is reached, a windows message is sent back to the Test Executive which handle communications with the screen and printer. When input is required from the user, a message is sent to the Test Executive, which handles all user input.

On the Honeywell 2600, the Time Delay Generator was used to implement a "Drop Dead Timer" to guard against having a test lock up while taking a measurement with a piece of equipment such as the counter or the DMM. For the new FATSS test station, this has been implemented by setting up a windows timer in the Test Executive. If the ATLAS code needs to set up a Drop Dead Timer, it sends a message back to the Test Executive informing it to start a timer and sending it the time to go off in seconds. If the Drop Dead Timer goes off, the Test Executive kills the ATLAS thread, and restarts it at the previously designated line number. If the ATLAS code completes the task successfully and needs to turn off the Drop Dead Timer, it sends another message to the Test Executive, which in turn kills the timer before it has a chance to fire.

The Honeywell 2600 has an Auxiliary Keypad, which has four test mode buttons and several other controls such as Station Clear and Begin Test and Advance which uses hardware interrupts to control the flow of the ATLAS code. Adapting this for use with the new FATSS station would be an expensive and time-consuming effort. These controls will now be implemented in software as buttons and switches on the Test Executive screen on the new FATSS test station. Through the use of windows messaging and threads, we will be able to allow the user to control the flow and direction of the ATLAS code as it executes without development of new hardware to replace or interface with the previous Auxiliary Keypad. The placement of these software controls on the Test Executive screen can be seen in (Figure 1).

Another use of windows messaging which will be implemented on the new FATSS station will be the use of Voice Control to control the flow of the TPS and to issue voice responses from the computer to the operator. This will be achieved through the use of the Voice Control systems tray application developed by O0-ALC/TISAD and Microsoft's Text To Speech and Voice Command Recognition engines. This will assist the operator in controlling the flow of the TPS when he needs to operate the station hands free, such as when he is making adjustments or taking manual measurements.

A diagram of the general communications scheme between the Test Executive, the TPS Main, the TPS Thread, and the Legacy ATLAS code and the Hardware Drivers is shown in (Figure 2). When the Operator selects Begin Test, the Test Executive calls TPS_Main function passing it the line number of the ATLAS Entry Point code to execute. The TPS_Main function then launches the TPS_Thread. This thread is a loop which executes until the Entry Point code is finished. The TPS_Thread looks at the line number to be executed and calls the correct ATLAS entry point function which has been identified by the ATLAS-to-C translator developed by O0-ALC/TISAD. As the Entry Point executes, Status Messages are sent to the Test Executive by using the API SendMessage() and PostMessage() functions. Status of screen controls is also requested by these same messages. When the TPS_Thread was created, the handle to the thread was handed back to the Test Executive which uses this handle to pause or kill the TPS_Thread if necessary. This is an especially important capability in an ATE environment.

The Entry Point code will make calls to Hardware Drivers, which talk directly to the instruments in the test environment.
station. Some of these instruments in the station, such as the DMM, Counter, and Waveform Analyzer have displays on the Test Executive screen. When one of these instrument drivers needs to update the display on the screen, it sends a Windows Message to the Test Executive, notifying it to update the display. When the Entry Point is complete, the thread is finished and calls the API function ExitThreadO. All global variables used in the Entry Point code are retained, since they are in the scope of the TPS_Main function.

For the Waveform Analyzer and DMM, large sets of data need to be transferred between the hardware driver and the Test Executive. The transfer of this amount of data from a buffer in the instrument driver to a buffer in the Test Executive could be relatively slow, especially while the TPS is executing. In order to speed up this transfer, a static shared memory area has been established between the instrument hardware driver and the Test Executive. As the driver collects data and needs to update data on the Test Executive screen, it fills the shared memory area and sends a windows message to the Test Executive informing it to update the screen with the data in the shared memory area. This is much faster than transferring the data from buffer to buffer.

CONCLUSION

Through the use of Windows Messaging, threads, and shared memory areas, the FATSS test station will be able to allow the user to control the flow of legacy ATLAS TPS code as it executes. The use of Windows Messaging also lowers the development cost of the FATSS station by alleviating the need to support the legacy hardware such as the Auxiliary Keypad. It also has allowed the use of new controls such as the mouse and Voice Control to be used for executing and controlling the flow of ATLAS code.

Figure 2. Communications Scheme