Testing with StateWORKS

Verification and testing

Any program must be checked against the requirements. There are several stages of testing while developing software: testing while programming, verification, testing of ready programs, customer use.

Coding is an error prone task. Therefore a program has to be tested continuously in its development phase. Testing is an integral part of coding. The development tools prove the correctness of syntax (compiler) and the completeness of required software modules (linker). Before the program is completely ready a programmer begins to run the already created code trying to catch logical errors and the deviations from the requirements. In some cases it is the way to complete the specification with missing details. In extreme cases the working program shows the programmer what he really wanted to program.

If the software is designed before coding, using some formal methods, there is a chance to verify the model which lies behind the software. Normally there is no model of the entire software but only some parts can be verified. As a rule the verifying process is applied to the control paths of the software which can be modelled for instance using finite state machines. The verification process is able to detect some logical failures in the design as for instance: deadlocks, infinite loops, not reachable states, improper terminations, etc. The verification process is done already in the design phase. Of course, coding may corrupt the model; hence, the process should be applied again to the completed program if possible. Verification of software models belongs to a class of rather overvalued activities but it is popular because it is feasible. Unfortunately, software verification is not able to catch logical errors in software (finite state machine) specifications which lead to failure in conformance to requirements.

The answer to the last problem can be achieved by testing of a running program. Testing means here a repetition of stimuli which are expected while the software will be in use and a comparison of responses with the expected values. The obvious scenarios are for instance the UML Use Cases which represent the “sunny days scenarios” (when everything works as expected). A true testing process has to simulate also the unexpected situations which may occur infrequently but could have disastrous effects if software fails. We are not able to cover all imaginable situations but we have to invent as many testing scenarios as possible. The testing scenarios play an extremely important role in software changes. A changed program is normally tested against the expected effects. Unfortunately, any changes to a program can have side effects which are unpredictable. Therefore after all changes we should repeat most if not all testing scenarios which may detect those side effects.

The last software testers are customers. While using the software they create situations which are never expected by professional testers and programmers. Customer feedback is valuable for software developers allowing them to improve the software quality and tune it to the real challenge. An automated request to send a message to Microsoft while the Windows operating program breaks down or encounters a problem is a well known example of this kind of feedback.

Testing with StateWORKS Monitors

StateWORKS specification tools and RTDB based run-time systems offer effective means for testing of programs being developed as well as running applications. As the StateWORKS concept is based on finite state machines (VFSM) testing concentrates on running scenarios which check the

correctness of state sequences. In addition, the specific of RTDB based applications allows easy access to all objects (variables, data) which can be checked against expected values. Monitors are used for testing. There are 3 monitors with different characteristics.

The SWMon is the basic monitor used for manual testing. It displays all objects properties in several forms (state machine, I/O-unit or user defined). Due to its presentation variety and simultaneous display of selected objects SWMon is especially useful in the development phase for solving difficult state machine specification problems. Its very strong point is the ease of using a service (force) mode and the logical debugging using step mode while executing finite state machines. These debugging features are described in [1] and [2].

The SWTerm was the first monitor to exploit command files for automatic testing. The command language used is described in [3].

The SWQuickPro is an expansion of its predecessor SWQuick. It is the most powerful tool considering automatic testing. It uses an extended version of the command language introduced for SWTerm. The extensions are pseudo commands which are used to control the progress of the test sequences.

Testing of an RTDB based application is understood as sending a stimulus to an object (for instance a command to a finite state machine), wait a while, read a value from the RTDB and compare it with the expected value. If the comparison delivers a false result an error is written into a log file. The sequence is repeated many times, whereas the delays between commands can be defined in the command file. Optionally, a wait for an event can be used as a break in the command sequence instead of a fixed time delay.

SWQuickPro

The SWQuickPro monitor is a Window dialog application; its graphical user interface (GUI) is shown in Figure 1. The GUI contains 3 zones: connecting, monitoring and testing.

Connecting to RTDB

The upper part is the connecting part. The buttons **Connect** and **Disconnect** are used to connect / disconnect the SWQuickPro to / from an RTDB based application whose IP address is defined in edit windows **Host** and **Port**. If required the edit window **Password** is used for entering the password. The state of the connection, the application configuration file and some basic information about the application (number of objects and number of state machines) is displayed in rows below the buttons.

Monitoring functions

The middle part is the Monitoring part. If connected, SWQuickPro displays a list of RTDB objects in the left pane. On selecting an object in the list all attributes of that object are displayed in the middle pane. A click on the button **Get** displays the object attribute value for selected attribute or all attribute values if the pseudo attribute *.all* is marked. A click on the button **Set** writes the value (entered into the edit box to the right) into the selected object attribute (the pseudo attribute *.all* does not make sense in that case). Only those buttons are enabled which can be used for a selected attribute. The button **Set** is of course disabled for the pseudo attribute *.all*.

Logging

The bottom part is the Testing part. It contains several buttons as well as edit and text windows for handling tests. All communication with the RTDB (Get and Set operation done in the

monitoring part) are written automatically into a log file *xxx.log* where *xxx* is the configuration file name (file *.swd). The default directory of the log file equals the application configuration file directory and cannot be changed.

Creating a test file

The log file can be copied into the command file using the buttons **Save to Test File**. The default name of the command file *xxx.swc* can be redefined in the provided edit window.

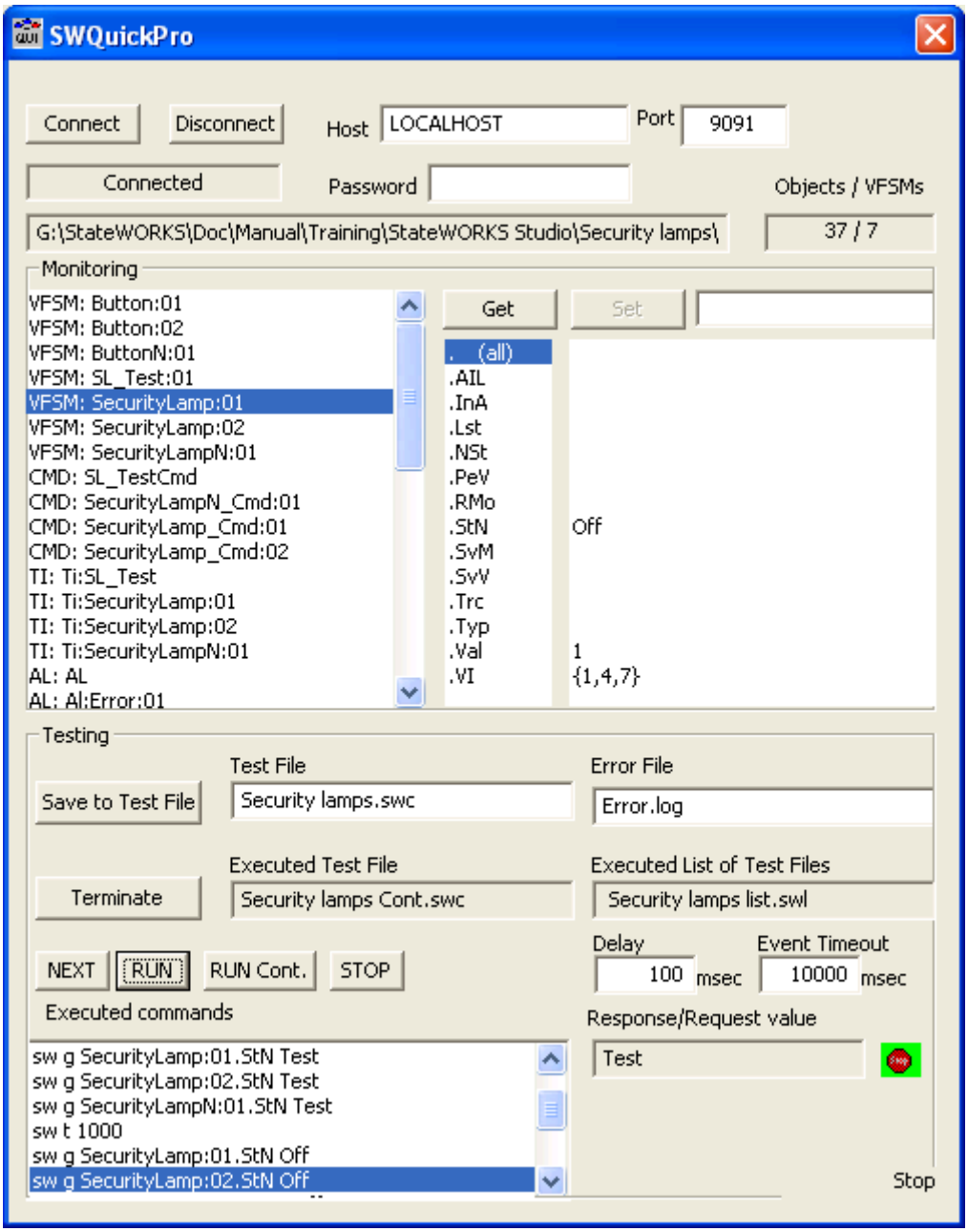

Figure 1: SWQuickPro monitor

By saving the log file to the test file we get a choice of overwriting of the existing command file or appending the log file content to the test file (see Figure 2). If the test file does not exist it will be created.

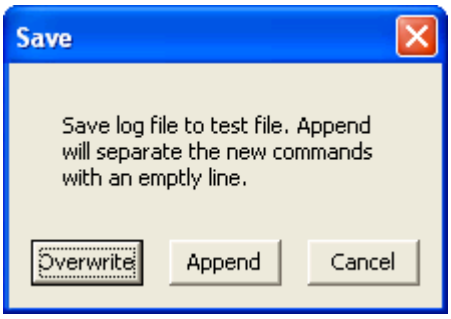

Figure 2: The Save dialog window

Command files

The command files created from log files can be used for simple testing or exercises but a true, demanding, tailored to our needs command file has to be written by hand. The log file may be of course the initial file where we sort out the redundant commands and add pseudo commands to achieve proper timing. While creating a command file for automatic testing we use the commands defined in [3] for communication with the RTDB, especially the Set and Get commands:

sw s objectname[attributename] Value

sw g objectname[attributename] ExpectedValue

These commands have been completed by two pseudo commands **sw t** and **sw e** which are used for defining waiting time between commands sent to the RTDB.

Command Delay

sw t DelayValue

sw t ObjectName.AttributeName

The **sw t** (delay) command has a parameter which can be:

- DelayValue that is an integer defining a delay in msec.
- ObjectName.AttributeName that is a name of an RTDB object which defines a delay. Normally this object will be a timeout of a certain timer.

For instance:

```
sw t 5000
```

defines a 5000 msec pause before the next command will be performed

```
sw t Ti:SecurityLamp:02.CnC
```

defines a pause which equals the timeout value of the timer `Ti:SecurityLamp:02`. In that case the timer clock base will be considered to define the proper value in msec.

Command Wait for Event

sw e ObjectName.AttributeName Value [DelayValue]

The **sw e** (wait for event) command may have up to three parameters:

- ObjectName.AttributeName is a name of an RTDB object that defines an event to be waited for. Typically, this object is a state machine (VFSM) and its attribute is the state name (*.StN*).

- Value is the event to be waited for.

- DelayValue is an optional parameter (integer) that defines the timeout (in msec) for the event.

For instance:

```
sw e SecurityLamp:01.StN Off 2000
```

reads: wait for the state `Off` of the state machine `SecurityLamp:01` or continue after `2000` msec.

If the parameter DelayValue is missing an EventTimeout value (default = 10000 msec) is taken.

Using command files for automatic testing

A button **Terminate** in Figure 1 has a double function. In the initial state it is named **Load test file** and is used to begin the test by loading the test file. In the initial state a click on this button opens a file open dialog window. Either a single command file (extension *.swc) or a file containing a list of command files (extension *.swl) can be selected. While running a test this button is named **Terminate** and is used to terminate the test and return to the initial state accompanied by cleaning all relevant displays.

The errors occurring during test is written into the error file with the default name *Error.log*. The name of the error file can be changed in the text window **Error File**. The default directory of the error file equals the the application configuration file directory and cannot be changed. An example of an error file is shown in the Appendix C.

A click on the button **NEXT** performs one command from the selected command file. The just executed command is displayed as selected in the list window **Executed commands.** The executed command parameter of the *Set* operation is shown in the text windows **Response/Request value**. The same window displays the response for the *Get* command. If the response differs from the expected response an error icon is displayed to the right to the window R**esponse/Request value**.

While executing a single test file the commands as shown in the **Executed commands** window are executed sequentially. Reaching the end of the list the execution begins again from the first command in the list. While performing a continuous test (buttons **RUN** or **Run Cont.)** the execution returns to the beginning of the list but stops at the first command and has to be restarted.

A click on the button **RUN** starts a continuous execution of a command file. The delays between two consecutive commands is determined by a default value defined in the edit window **Delay**. This value can be redefined at any time. The delay between two consecutive commands is replaced by the command **sw t** in the command file. If an error occurs the testing is stopped. Testing can be resumed by pushing a button: **NEXT**, **RUN** or **RUN Cont**. Similarly, the default value of the event timeout is replaced by a parameter of the command **sw e**. The default value of the event timeout is displayed and can be redefined at any time in the edit window **Event Timeout**.

Similarly, a click on the button **RUN Cont.** starts a continuous execution of a command file. In that case a test error does not stop testing.

A click on the button **STOP** stops continuous testing which can be resumed by pushing a button: **NEXT**, **RUN** or **RUN Cont**.

A click on the button **Terminate** terminates testing.

If running a list of command file (*.swl*) all command files in the list are processed sequentially.

The Appendices contains examples of a command file and a list of command files.

Conclusions

The testing facilities for application built around the RTDB stem from the principle of using ready run-time system. The standard access to all relevant data concentrated in the RTDB plus the control model based on finite state machines allows definition of a standard test procedure and corresponding tools like SWQuickPro which have universal character. They can be used in any RTDB based application specified with StateWORKS Studio.

References

[1] Wagner, F. et al., *Modeling Software with Finite State Machines – A Practical Approach*, Auerbach Publications, New York, 2006.

[2] Technical note: Debugging State Machines. Nov. 2004.

[3] Manual: StateWORKS Terminal client.

Appendices

A. Example of a command file

```
// Test of Security lamps

// Start test
sw s SL_TestCmd.PeV 1

// Read states of all state machines
sw g SecurityLamp:01.StN Test
sw g SecurityLamp:02.StN Test
sw g SecurityLampN:01.StN Test

// Break for 3 sec
sw t 3000

// Wait for SecurityLamp:01.StN being in the state Off
// otherwise timeout after 1 sec
// Check whether the state machine is in the state Off
sw e SecurityLamp:01.StN Off 1000
sw g SecurityLamp:01.StN Off

// Wait for SecurityLamp:02.StN being in the state Off
// Check whether the state machine is in the state Off
sw e SecurityLamp:02.StN Off
sw g SecurityLamp:02.StN Off

// Check whether the state machine is in the state Off
```

```
sw g SecurityLampN:01.StN Off
```

B. Example of a list of command file

```
// List of of Security lamps tests
Security lamps.swc
Security lamps Cont.swc
Security lampsE.swc
```

C. Example of an error file

```
G:\StateWORKS\Doc\Manual\Training\StateWORKS Studio\Security lamps\Conf\Security
lamps list.swl
started at 30-May-07 11:30:00


G:\StateWORKS\Doc\Manual\Training\StateWORKS Studio\Security lamps\Conf\Security
lamps1.swc
started at 30-May-07 11:30:00
End of test




G:\StateWORKS\Doc\Manual\Training\StateWORKS Studio\Security lamps\Conf\Security
lamps Cont.swc
started at 30-May-07 11:30:35
          Line 6: g SecurityLamp:01.StN Off : Test <> Off (expected)
          Line 7: g SecurityLamp:02.StN Off : Test <> Off (expected)
          Line 8: g SecurityLampN:01.StN Off : Test <> Off (expected)
End of test




G:\StateWORKS\Doc\Manual\Training\StateWORKS Studio\Security lamps\Conf\Security
lampsE.swc
started at 30-May-07 11:30:37
          Line 8: Timeout while waiting for event: SecurityLamp:02.StN Off
End of test



End of test
```