SeTT: Testing-tool for Measurement System DEWESoft

TOMAŽ KOS[1], TOMAŽ KOSAR[2], MARJAN MERNIK[2], JURE KNEZ[1]
[1]DEWESoft d.o.o.
Gabersko 11a, 1420 Trbovlje
SLOVENIA
{tomaz.kos, jure.knez}@dewesoft.si http://www.dewesoft.si
[2]Faculty of Electrical Engineering and Computer Science
Smetanova ulica 17, 2000 Maribor
SLOVENIA
{tomaz.kosar, marjan.mernik}@uni-mb.si

Abstract: - Measurement procedures are becoming more and more complex within the DEWESoft measurement system. Consequently, the manual-testing of acquired data is becoming more and more demanding. Therefore, it was decided to build a tool for the automated-testing of measurement system. The key idea is that each piece of hardware and software needs a test to ensure proper functioning. This paper introduces an SeTT integrated within a domain-specific modelling tool Sequencer. In this paper the testing-tool architecture and implementation details are revealed. Moreover, some advantages and disadvantages are provided and the experience of using this testing tool within a real-case scenario.

Key-Words: - domain-specific modelling language, testing domain-specific modelling language, unit-testing, automated testing, measurement system

1 Introduction

Construction of the measurement procedure is not the last phase in the development life-cycle of a measurement system. After the measurement procedure (also called "sequence") has been designed, end-users also need to check for potential errors or defects, and in the case of their presence, locate and improve them. The testing of each system module consists of comparing the acquired data with the referenced results. The testing of the system can be manual or automated. Manual-testing is a laborious and time-consuming process, which requires concentration and precision at work. After sloppy work, errors can easily appear and they can be critical for the proper functioning of the whole measurement system. Testing should be repeated every time a minimal modification is made within the system (i.e., electronics, firmware, software) and it takes up additional engineering time. Measurement systems, hardware, sequences, etc. are also becoming more and more complex. Consequently, manual-testing acquired data has become demanding. Therefore, an SeTT (Sequencer Testing Tool) has been built for automated testing of different components of measurement system. The key idea is that each piece of hardware and software needs a test to ensure proper functioning.

The development of test-automation software from scratch is demanding. This is especially true in the case of testing tool developed over dedicated software. Such software is called domain-specific language (DSL) [1] or domain-specific modelling language (DSML) [2, 3]. End-users, who have no formal computer knowledge, but have a skill regarding the problem domain, can write their own domain-specific programs/models to solve the specific needs within their domain [4]. DSMLs, which use visual notation instead of textual ones as DSLs, remain even more expressive at a higher abstraction level than DSLs.

SeTT is integrated within an existing DSML Sequencer [5] which is included within the DEWESoft measurement software [6]. The Sequencer enables domain experts to model their own measurement procedures. The traditional unit-testing concepts [7] have been adopted to the modelling environment, such as: test assertions, definitions of expected values, and test-reports. Using this testing-tool, domain-experts can create test cases for each part of the measurement system, in order to check and evaluate its behaviour. Preliminary results show that automated testing in a Sequencer reduces the cost of failures and increases the quality of the measurement system.

This paper is divided as follows. Section 2 introduces the principle of testing within the measurement system. Section 3 provides details about testing-tool implementation. Use of the testing-tool is presented in Section 4. Finally, Section 5 summarises the main features of the testing-tool for the measurement systems, and provides the concluding remarks.

2 Measurement system-testing

Measurement system-testing in a Sequencer is black-box testing [8]. The starting-point represents a configuration of the complete system within a controlled-environment. The purpose of testing is to validate system's accuracy, performance, and completeness of designed functions. Moreover, system-testing simulates scenarios that occur within a real system environment, and tests all the required functions of the system. In general, a testing measurement system is completed when the actual results and expected results are either equal or the differences are explainable and acceptable.

The diagram in Fig. 1 shows the testing process for a measurement system. Testing begins when a domain expert designs test-cases (1). This phase consists of the selection/definition of the measurement procedure undergoing a test, the selection of testing-objects (i.e., signals), as well as definitions of the expected values. The test-cases can be constructed with DSML language Sequencer in a visual manner. The end-user can easily model an arbitrary test case, even though they do not have any knowledge about programming or experiences with unit-testing in general-purpose languages. In the next step the test-cases are executed (2) one after another. This step can take a long time, since it depends on the type of test. When all the tests have been completed, the testing-tool generates a test-report (3). If one of the test-cases is faulty (4), the engineers need to locate it (5) and fix the errors (6), then repeat the execution of the tests.

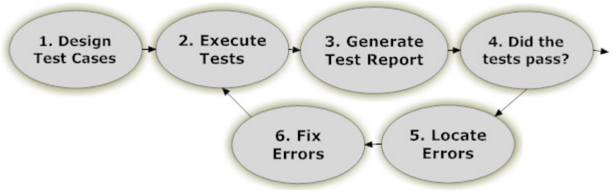


Fig. 1: Process diagram

3 DSML Testing Tool

This section introduces the SeTT, its architecture, construction, and the environment engine that is the main part of the testing tool.

3.1 Testing-environment and Architecture

As previously mentioned, the SeTT is included within the product DEWESoft. The whole product is implemented in the Delphi programming language and operates under MS Windows platform.

In general, the SeTT's environment is divided on three levels (Fig. 2). The user level, intended for end-users to design test-cases and observe the testing results, is on the first level. Test-cases can be programmed or modeled in three different notations (visual, textual, or XML). Every language is translated into a single-execution model which is a unique internal-representation of the model.

The core of the SeTT is on the second level. Here, the test-cases are delivered to the domain-test engine, which constructs the execution model. The execution model is then further served to the domain framework, which is responsible for executing the model. The domain framework is in charge of the entire process during the test case and also controls and supervises the hardware.

The hardware, the most important part of the measurement system, is on the third level. At this level, hardware components are tested, such as AD/DA cards, CAN, GPS receivers, telemetry boards, and others involved in the measurement system. Smaller defects on the circuit or problems with firmware may lead to a total measurement error. In order to control and verify the results obtained from the hardware components, there are reference devices (calibrations, function generator, etc.) to help simulate and verify the accuracies of the obtained results.

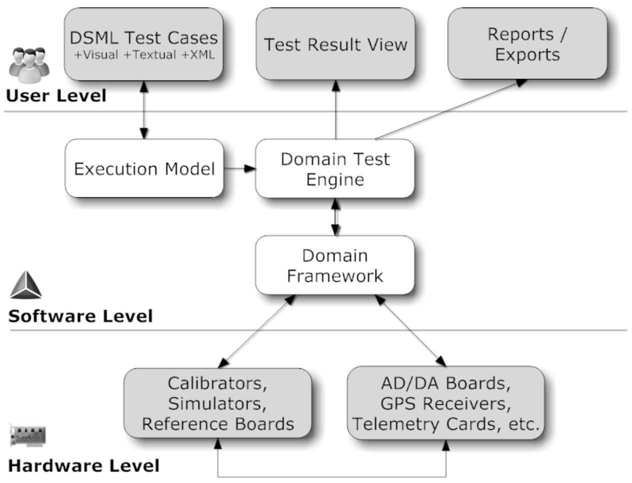


Fig. 2: DSML Testing-environment

3.2 Extension of DSML

Unit-testing was taken as a reference and the DSML Sequencer was expanded with new functionalities. The language Sequencer was extended with a new concept "Assert". The purpose of this construct is to validate specific system outcomes. Fig. 3 presents a hierarchical decomposition of "Assert" concept, in FDL notation [9]. The root-concept `AssertStatement` contains text information (`text_inf`), output voice (`output_voice`), type of assertion (`AssertType`), etc.

Generally, DSML languages with visual notation are developed using metamodels that define the abstract syntax of a DSML. A metamodel is a model of models and includes domain concepts that are used to model the concrete problem. Also, Sequencer is constructed using a metamodel. In the definition of the metamodel, a set of classes and base classes are defined, and a set of attributes for each class, as well as the relationships between classes and their limits. For the purpose of the testing-tool the metamodel was extended with a new class "Assert", associated attributes, the relationship, and constraints between classes.

```
AssertStatement: all(text_inf, output_voice,
      AssertType, ExpectedResult, error_message)
AssertType: one-of(AssertUser, AssertValue,
      assert_acq_running)
AssertUser: all(yes_button, no_button)
AssertValue: more-of(Expression,
      opt(LogicalOperator))
Expression: all(Value1, Value2,
      ComparisonOperator)
Value: one_of(constant, global_variable,
      local_variable)
ComparisonOperator: one-of(equal, not_equal,
      greater, less, greater_equal, less_equal)
LogicalOperator: one-of(and, or)
ExpectedResults: one-of(true, false)
```

Fig. 3: FDL specification of Assert concept

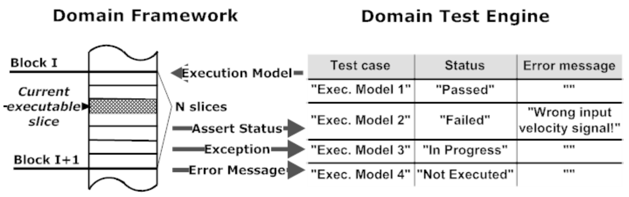
In addition to visual notation, textual notation has also been extended for the testing tool. The textual version of Sequencer was developed using compiler/interpreter implementation. For the purpose of testing tool, the lexical, syntax, and semantic definitions were extended to support the "Assert" statement. In the lexical analysis, the state-machine was upgraded with new keywords, whilst syntactical analyzer was extended with a new production (Fig. 4).

```
NT_START ::= "Begin" NT_LINE "End"
NT_LINE ::= "Action" NT_ACTION
         | "LoadSetup" NT_LOADSETUP
         | "If" NT_IF
         | "Repetition" NT_LOOP
         | "Wait" NT_WAITFOR
         | "Delay" NT_DELAY
         | "AvdioVideo" NT_AVDIOVIDEO
         | "Calculation" NT_FORMULA
         | "CustomBlock" NT_CUSTOM_BLOCK
         | "FileManager" NT_FILEMANAGER
         | "Macro" NT_MACRO
         | "Export" NT_EXPORT
         | "RemoteControl" NT_REMOTECONTROL
         | "Assert" NT_ASSERT
         | "Form" NT_FORM
         | epsilon
...
```

Fig. 4: Partial Sequencer's grammar

3.2 Domain-test Engine and Framework

The Domain-test Engine is the main component of the testing tool. It is responsible for processing and handling test cases. Each test case is transformed into a single-execution model, which is then executed within the domain framework. Execution models are executed in the same order as test cases. The test, in turn, is sent to the domain framework for execution. Furthermore, the domain-test engine maintains a status for each test case separately and it can be "Pass", "Failed", "Error", "Not Executed" and "In Progress". The test case starts with the "In Progress" state. The test remains in this state until assertion produces a result (pass, fail, or error). If a test case does not have an assert statement, the test always produces a "Pass" result. Since the execution model and, thus, the assert statement are executed within the domain framework, the test engine receives information about the produced status of the domain framework. In addition to status and execution, the Domain-test Engine receives an error-message (Fig. 5).



Test case	Status	Error message
"Exec. Model 1"	"Passed"	""
"Exec. Model 2"	"Failed"	"Wrong input velocity signal!"
"Exec. Model 3"	"In Progress"	""
"Exec. Model 4"	"Not Executed"	""

Fig. 5: Program execution

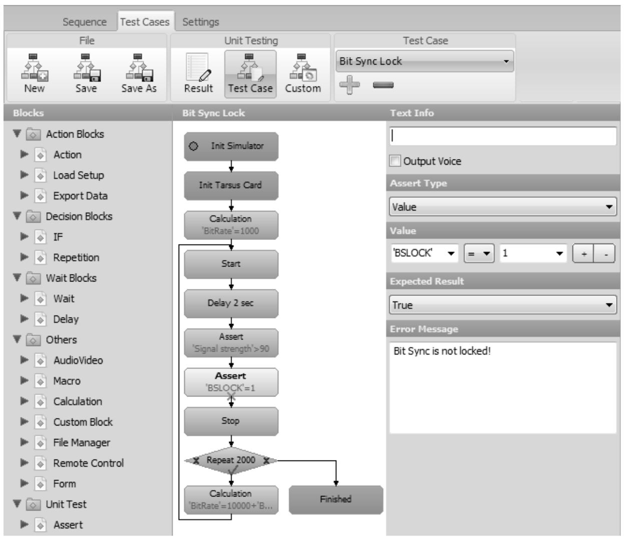
To ensure a good response and synchronization between the different parts of the measurement system (test cases, DEWESoft, hardware, etc.), the individual actions of the execution model are processed in time slices, as shown in Fig. 5.

4 An Example

Nowadays, it is hard to imagine large and complex systems, such as satellites, rockets and spacecraft, without the use of PCM telemetry [10]. Such a technique allows for the automated monitoring, saving, and alerting of the measured data necessary for a safe and efficient operation. Space agencies such as NASA, ESA, and other agencies, use telemetry to acquire data from spacecraft and satellites. Thousands of telemetry data are transmitted via the PCM signal from a spacecraft to the earth-system, where the data are captured, decoded, displayed and subsequently analyzed. Such systems must be of high quality, and therefore need to be tested to perfection. In order to demonstrate the idea of testing with DSML, the testing-tool was used on PCM telemetry. A telemetry measurement unit (together with this software) as produced by DEWESoft, is already in use by the aerospace industry (e.g. NASA, Raytheon, Honeywell and Lockheed Martin).

Automated testing with SeTT is presented on a telemetry acquisition station. Since the measuring system consists of many components (AD card, IRIG Timing card, GPS receiver, etc.), the focus is only on the TarsusHS-PCI-01 Processor Board [11]. In the following example we present a test case which is used for validating the status of a lock-signal for the bit-synchronizer of different bit-rates.

In order to verify the correct behaviour of the PCM bit-synchronizer, a simulator (PCM transmitter) is needed. The simulator sends a PCM signal, onto which our tested system is able to lock. During the testing the transmitter and receiver bit rate (from 1 kbps to 20 Mbps) are changed and verify the status. On this basis, the decision is made as to whether this test case has succeeded or not.



Assert Type: Value

Value: 'BSLOCK' = 1

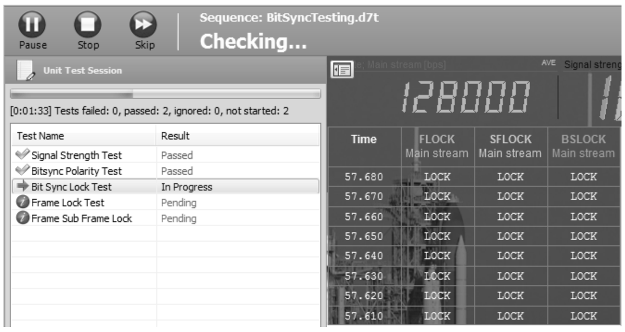
Expected Result: True

Error Message: Bit Sync is not locked!

Fig. 6: Creating a test case with a Modelling Tool

Fig. 6 shows the modelling tool with the "Bit-sync Locking" test case. On the left-side of the SeTT testing tool, there are domain-specific modelling constructs. The working panel is in the middle, where the end-user can model a test case. The right-side of the user interface contains properties that can be configured for the selected block, from a model. Since the "Assert" block was selected in this test case, its properties can be observed. In the "Bit-sync Locking" test case, the procedure begins by calling two functions (blocks Init Simulator and Init Tarsus Card), which initialize the simulator and testing board where some settings are defined such as PCM code type, signal polarity, and others (the starting block is marked with a circle). The measurement procedure continues with the settings of a stream bit rate. Then it starts with data acquisition (block Start). The purpose of the Delay block in the sequence is to stabilize the signal. After that, the next "action" block validates the strength of a signal (signal-strength must be higher than 90 percent, otherwise the test case fails) and the bit-sync status is checked. The second assertion determines whether the bit-sync status is locked (e.g. "BSLOCK" = 1) or not. Also, the expected result and error-message are defined. The latter is reported beside a failure in the Test Result View (Fig. 7). At the end of the test case, the data acquisition is stopped. The data acquisition and testing is repeated 2000 times (block Repeat 2000 in Fig. 6). For each repetition, the bit rate is increased by 10 kbps (block Calculation) to obtain 20 Mbps (10 kbps * 2000 = 20 Mbps). The test case finishes after 2000 repetitions.

During the execution of the test, the domain expert can observe the results and acquired values. Fig. 7 shows the execution of the test case, as defined in Fig. 6. On the left-side of the screen, the domain-expert can observe the elapsed time of the test-execution. This session-header view also indicates the total number of failures (in this bit-sync locking test case, there were zero failures), number of tests passed (two passes), number of ignored tests (zero ignored tests), and the number of tests as yet tested. Note, that more test-cases can be activated within one execution. The list of test-cases underneath the header-view indicates all the names of the test-cases and their status. During the test-execution, the acquired data can be observed on the right-side of Fig. 7. For this test case, input signal can be monitored together with the bit rate, and bit-status, which are used for validation.



[0:01:33] Tests failed: 0, passed: 2, ignored: 0, not started: 2

Test Name	Result
Signal Strength Test	Passed
Bitsync Polarity Test	Passed
Bit Sync Lock Test	In Progress
Frame Lock Test	Pending
Frame Sub Frame Lock	Pending

Time	FLOCK Main stream	SFLOCK Main stream	BSLOCK Main stream
57.680	LOCK	LOCK	LOCK
57.670	LOCK	LOCK	LOCK
57.660	LOCK	LOCK	LOCK
57.650	LOCK	LOCK	LOCK
57.640	LOCK	LOCK	LOCK
57.630	LOCK	LOCK	LOCK
57.620	LOCK	LOCK	LOCK
57.610	LOCK	LOCK	LOCK

Fig. 7: SeTT session on PCM telemetry system

5 Advantages and Disadvantages

Since the year 2000, more than one thousand measurement procedures have been developed using the DEWESoft. To increase the software quality, reduce development time and consequently a faster time to market, development teams need to have a flexible, easy to use, and effective testing-environment. Over these years, the advantages of having SeTT, have already been revealed.

Most measurement procedures are tested manually, which is a time-consuming. Nevertheless, the objective of system-testing is to isolate a part of the system and validate its accuracy without human influence. Test-automation can achieve this and enable other benefits, such as the elimination of human errors, test repetition, and reusability. In addition to these benefits, probably the most important advantage of all is that the automated tests are significantly faster than the manual tests performed by humans. Practice has shown that automated tests can be 5 times quicker than manual tests. In Table 1, four different measurement procedures are tested using the SeTT and compared with manual tests. In Table 1 it can be observed that the manual testing of Test 1 takes approximately 8 hours and 39 minutes, whilst the automated testing takes only one hour and 43 minutes.

Table 1: Comparison of Automated / Manual Testing

System	Manual testing	Automated testing
Test 1	08:39	01:43
Test 2	01:28	00:40
Test 3	04:45	02:47
Test 4	00:29	00:17

The Sequencer is focused on a specific domain. The level of abstraction has been raised, and thus increases productivity in the development of applications for measurement systems. Practice has shown that end-users can develop applications faster, whilst the program length is reduced [12]. It was discovered that this also applies for DSML testing. In Table 2, four different tests using SeTT are compared with a test-application written in DUnit [13]. The size of the program is measured using our textual language, because the number of lines of code (LOC) is difficult to compare with visual models. Note that mapping between our own textual and visual concrete syntaxes is almost one-to-one. In Table 2 it can be observed that Test 1 using the DUnit tool has 111 LOC, whilst the DSML Test Unit has only 11 LOC.

Table 2: Comparison of the tests in LOC

System	DSML testing	DUnit testing	Ratio
Test 1	11	111	10.1
Test 2	29	232	8.0
Test 3	5	71	14.2
Test 4	14	136	9.7

The development of DSMLs and supporting tools (i.e. testing tool) brings both advantages and disadvantages. At the beginning, it is necessary to invest in the development of a tool. For instance, the implementation of SeTT took two engineer months (which does not include time for maintenance and extensions).

6 Conclusion

This paper presented a testing-tool integrated within DEWESoft measurement systems. This testing-tool offers testing hardware (different analog boards, different PCM boards, etc) as shown in “Bit-sync Locking” test case. With SeTT, it is also possible to test those models written with the Sequencer. Now the domain-experts can create and trace measurement procedures faster, and more efficiently.

Acknowledgements

This work was partly sponsored by the European Social Foundation.

References:

[1] Mernik M., Heering J., Sloane A.: When and how to develop domain-specific languages. *ACM Computing Surveys,* Vol. 37, No. 4, 2005, pp. 316-344.

[2] Kelly S., Tolvanen J.-P.: *Domain-Specific Modeling Enabling Full Code Generation*, John Wiley & Sons, Inc., 2008.

[3] Sprinkle J., Mernik M., Tolvanen J.-P., Spinellis D.: Guest Editors' Introduction: What Kinds of Nails Need a Domain-Specific Hammer?, *IEEE Software*, Vol. 26, No. 4, 2009, pp. 15-18.

[4] Kosar T., Oliveira N., Mernik M., Varanda Pereira M. J., Črepinšek M., da Cruz D., Henriques P. R.: Comparing General-Purpose and Domain-Specific Languages: An Empirical Study. *Computer Science and Information Systems*, Vol. 7, No. 2, 2010, pp. 247-264.

[5] Kos T., Kosar T., Mernik M.: Development of Data Acquisition Systems by Using a Domain-Specific Modeling Language, Accepted for publication in *Computers in Industry*, 2011.

[6] Data acquisition software DEWESoft 7: http://www.dewesoft.com/.

[7] Unit testing: http://en.wikipedia.org/wiki/Unit_testing/.

[8] Beizer B., *Black-box testing: Techniques for functional testing of software and systems*, Wiley, Inc., 1995.

[9] van Deursen A., Klint P.: Domain-specific language design requires feature descriptions, *Journal of Computing and Information Technology*, Vol. 10, No. 1, 2002, pp. 1-17.

[10] Telemetry Tutorial, L-3 Communications Telemetry West, 2002.

[11] Ulyssix Technologies, Inc.: http://www.ulyssix.com/

[12] Kos T., Kosar T., Knez J., Mernik M.: From DCOM interfaces to domain-specific modeling language: A case study on the Sequencer, *Computer Science and Information Systems*, Vol. 8, No. 2, 2011, pp. 361-378.

[13] DUnit tool: http://dunit.sourceforge.net/.