
Model Based
RDIR
PDIS
RDIR
PTOC
TCTR
PSCH
MMXU
PTUF
MMXU
PSCH
TCTR
PDIS
TVTR
TVTR
PDIF
PTUF
PTOC
PTIC
TVTR
PDIR
PSCH
MMXU
PTUF
TVTR

by Iony Patriota de Siqueira, Chesf, Brazil

Testing and Diagnosis of IEC 61850 Systems

This article describes the method's use for fault diagnosing through a proof-of-concept prototype developed in a research project.

The adoption of IEC 61850 standard on Substation Automation Systems (SAS) requires new ways of testing and diagnosis. Testing is required to ascertain their correct behavior, while diagnosis is required to find and correct any faulted function detected in a test. The object oriented aspects of IEC 61850 suggests a model based equivalent concept for testing and diagnosis. This issue has been partially addressed by Cigré Working Group B5.32 in proposing a structured object oriented methodology for Functional Testing of IEC 61850 Based Systems.

This article reviews this method and describes its use for fault diagnosing through a proof-of-concept prototype developed in a research project. In this tool, fault diagnosing in Substation Automation Systems (SAS) is done using a model-based approach whereby an object oriented model of the system is provided that can be simulated while faults are identified and pinpointed. Testing is also performed in an objected oriented way, conducted by following a test script according to the scheme proposed by Cigre WG B5.32. In addition, fault diagnosis may be performed by correlating the test results with the model architecture.

This article reports the results of a Brazilian research project jointly sponsored by CHESF (Companhia Hidro Elétrica do São Francisco) and ANEEL (Agência Nacional de Energia Elétrica), and conducted by UFCG (Universidade Federal da Paraiba), to develop a proof-of-concept software tool that enables automation engineers to build, run and debug functional tests for IEC 61850-based systems. The testing is based on the specification produced by WG B5.32. An extension of this work is being developed to allow diagnosis, using the same tool.

Automation Model

A simple example will illustrate the test method proposed by Cigre, taken from the work done by WG B5.32. Only a brief sketch of a test scenario will be given due to space restrictions. Please refer to the full WG B5.32 technical brochure

by Iony Patriota de Siqueira, CHESF, Brazil

for details. The approach is based on UML, text and XML formats, used to specify Functional Use Cases and other UML artifacts. Consider the SAS for a simple one-line diagram of a transformer bay in a substation shown in Figure 1, taken from WG B5.32 Technical Brochure, with the corresponding logical nodes defined by IEC 61850. This figure uses a UML communication diagram to specify the message types exchanged by the logical nodes of the SAS.

This type of diagram can be part of a functional specification of an SAS system, not covered by current version of IEC 61850, that includes a *Functional Implementation Conformance Statements* (FICS), in a format proposed by Cigre WG B5.32 shown in Table 1.

In addition, other UML diagrams may be used for functional specification. For example, a UML sequence diagram is shown in Figure 2 that may be part of the FICS document. In this Figure the numbers shown are PICOM types (12 = Operated, 22 = Trip, etc.). The left-hand side of the figure also shows the performance requirements as UML time delay restrictions.

Further details about the SAS could be included in a UML deployment diagram, showing the physical distribution of logical nodes in servers, their labels, network addresses, etc., as shown in Figure 3.

Test Model

Cigre WG B5.32 suggested a test architecture consisting of several test components, following the UML Test Profile of OMG (Object Managing Group), as depicted on Figure 4. In this architecture, the test objects are instantiated and connected to the SAS objects as defined in Figure 2. Figure 5 shows the objects instantiated from the test classes necessary to exercise this example SAS. The figure also shows their logical connection to the SAS Logical Nodes (LN).

In this model each breaker is modeled by instances of DigitalOutput and DigitalInput classes, to simulate their command and response messages, while each current transformer is modeled by an instance of CurrentOutput

table 1 Functional implementation conformance statement

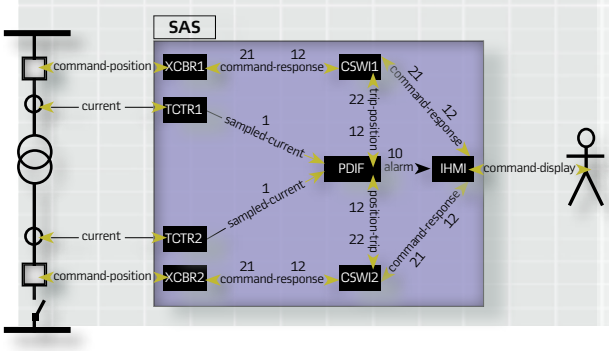
Functional Implementation Conformance Statement	
Code	
Name	
Description	
Customer	
Substation	
SCL File	
Primary User (Actor)	
Secondary User (Actor)	
Stakeholder & Interest	
Function Description	
Trigger	
Components or Logical Nodes	
Process Equipment	
Performance	
Preconditions	
Post conditions on Success	
Post conditions on Failure	
Use Case Description	
Basic Course Description	
Alternative Course Description	
Exception Course Description	
Extensions	

table 2 Functional test case template

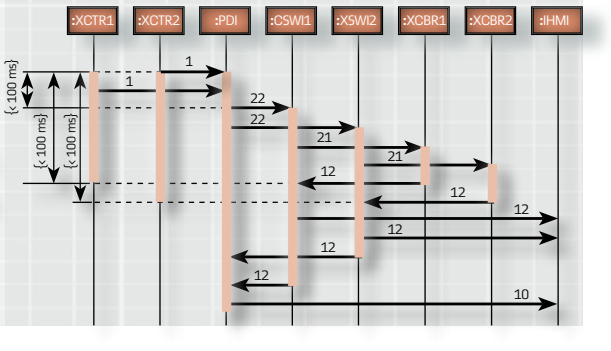
Functional Test Case	
Code	
Name	
Description	
Use Case Description	
Customer	
Substation	
SCL File	
FSR File	
FICS File	
Test Description	
Test Connection	
Test Setup	
Test Start	
Test Stop	
Test Disconnection	
Test Verdict	

Xxxx PRP and HSR are new and offer additional redundancy levels. This is required especially in Processes Bus applications with sampled values but can work at any level. In a process bus with sampled values, seamless or zero-recovery-time-redundancy is a prerequisite.

1 Example - substation layout diagram



2 Functional specification by UML sequence diagram



class, simulating their sampled currents. A network simulator (or analyzer) is instantiated and assigned to monitor the messages related to logical node PDIF, to measure its response time. Messages sent and/or received by the operator are modeled by an Operator object. This setup can be described more fully as a functional test case, described as a UML Use Case artifact. This is shown in Table 2 using a format proposed by Cigre WG B5.32.

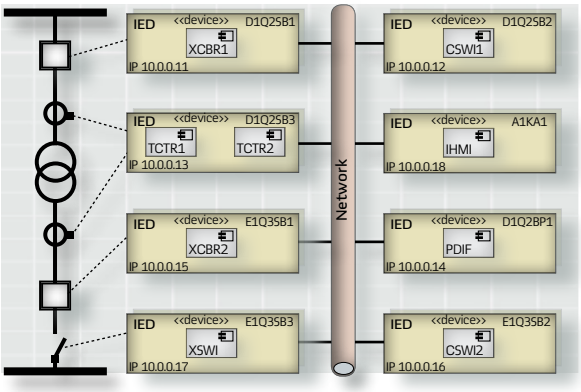
Test scripts can specify signals to be injected in the system as well as expected signals. Each command in this script is a method call supported by the instantiated class. The last commands (verdicts) evaluate the results of the test case. Test cases can also be specified in XML. The reader is referred to the Cigre WG B5.32 report for the full description of this test and the associated XML schema.

Functional Test Requirements

To automate the execution of test scripts, a tool was specified following requirements developed by CHESF and implemented by UFCG. The tool will allow automation and protection engineers to develop and debug SAS designs, and check their correctness through test scripts. The requirements set by CHESF specify that initially the system will be used to build and debug tests in a simulated SAS environment only. In a subsequent phase, the system may be used during actual operation on a real SAS, by injecting actual messages at appropriate SAS access points, recording appropriate messages and evaluating the performance and functionality through test scripts. It should be possible to execute test scripts and report on the test verdicts, with full support for all B5.32 test objects (like VoltageOutput, CurrentOutput, DigitalInput, DigitalOuptut, NetworkSimulator, Operator, TestTimer, TestScheduler, TestArbiter).

According to CHESF requirements, the SAS should be represented by a model and should be simulated by the tool. The LNs most commonly used in SAS design should be supported. The design should be component-oriented to allow third parties to develop new LNs and plug them

3 UML deployment diagram



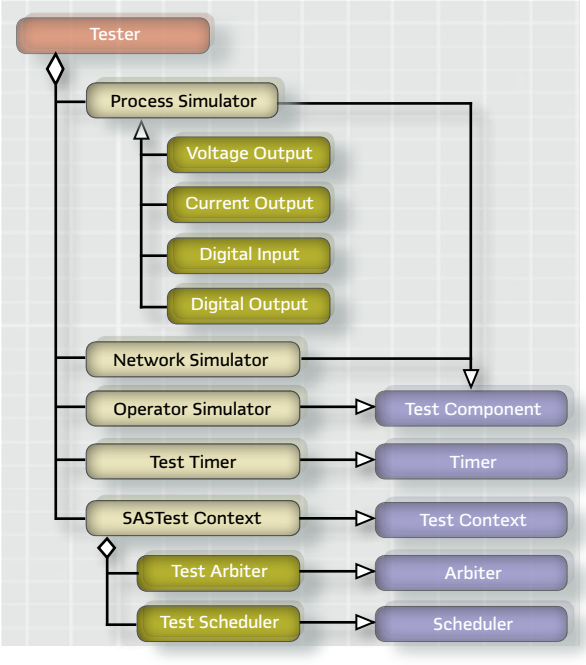
This research provides an example of a proof-of-concept implementation of a tool to help automation and protection engineers design and test SASs.

to the system. The system should also provide test script management (script creation, visualization, editing, addition, removal, etc.) and should be able to read in SAS models provided in IEC 61850 Substation Configuration Language (SCL).

The tool should provide for visualization and editing of test scripts in a command language and also in an XML format. Automatic conversion between these formats should be provided. The tool should perform syntax guidance and checking during test script editing according to the XML schema developed by Cigre WG B5.32. The test execution environment should provide execution commands (Run all, Run selected, Pause, Stop, etc.), Debugging mode (Run debug, breakpoints, single step, variable watch, etc.), and Simulated time speed control to accelerate or decelerate the simulation as compared to real time.

For diagnosing, the execution environment should provide mechanisms for the insertion of faults in any place on the SAS model. The tool should provide fault

4 Test profile as a UML class diagram



Iony Patriota de Siqueira was born in São José do Egito, Brazil. He has an MsC degree in Operations Research, an MBA in Information Systems, and a finishing PhD level in Electrical Engineering. He is a member of Cigré and IEEE, former convenor of Cigré WG B5.32 on Functional Testing of IEC 61850 Based Systems, Secretary of Cigré Study Committee B5, and convenor of Brazilian Technical Committee of IEC TC 57. Currently his is vice-director of Brazilian Maintenance Association and Manager of Protection and Automation at Chesf, with interest in protection, automation, reliability, maintenance and system performance.

by Iony Patriota de Siqueira, CHESF, Brazil

diagnosis functionality through an automatic fault diagnosis algorithm, thus allowing the source of faults to be pinpointed, down to the level of Logical Nodes.

Functional Test Tool

As tool called Smash (Smart SAS Test and Fault Diagnosis) is being developed to satisfy the requirements outlined above. This section describes its architecture, interfaces and current development status, according to Figure 6. The SAS is modeled by LNs which are components that simulate the behavior of functions such as differential protection (PDIF), circuit breakers (XCBR), etc., as per IEC 61850 standard. Since the architecture is componentized (the LNs are components that obey a standard discovery interface), new LNs can be added by third parties to the tool. The LNs are "active" classes, meaning that they run in a separate thread. This enables time delays to be introduced in their behaviors. All Publish-Subscribe communication between LNs and other test components is controlled by a common software bus. This allows the simulated environment to include network delays in the simulation. The main data structures are the LNs themselves as well as the Configuration component containing an in-memory version of the SCL file and a Script component containing an in-memory version of the script being executed. The TestScheduler, as described in the UML Test Profile, is the main simulator that interprets and executes script commands. Simulated time control is provided by the Time Control component. This is where speed control is implemented. All components requiring time service must interface with this component.

User Interface

The Smash User Interface follows the traditional layout of software integrated development environments. Figure 6 shows an outline of the Smash User Interface.

At the top left of Figure 7, is a menu that provides test script management, execution control, etc. Below the menu is a tool bar for test script execution and debugging, further detailed in Figure 8. Observe that a debugging mode is available to single-step execution, set breakpoints, examine the value of variables, etc.

Below the tool bar (Figure 8) is an area that shows the available tests. A test can be chosen and run from this list. Figure 9 shows the result of the test verdicts after a test run, showing that Verdict 6 did not pass for the Transformer Differential Protection SAS.

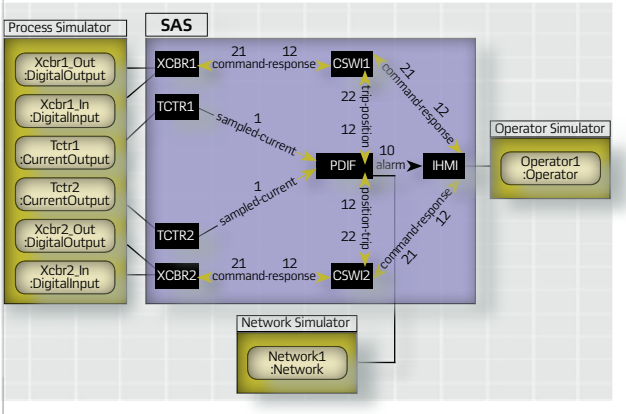
On the bottom left of the test environment there is an area that more fully describes the functional test case and provides access to the "Functional Implementation Conformance Statement" (FICS) and "Functional Specification Requirement" (FSR) files and version information, as suggested by Cigre WG B5.32. On the center panel, the test script is exhibited, either in a scripting language or as XML text. Debugging allows one to set breakpoints in the test script on this panel. Figure 9 shows the script with two breakpoints with execution stopped at the second one. The bottom panel can be used to show error list found on the test script, or to watch any variable defined on the model, as shown on Figure 10.

Fault Simulation

Fault simulation is a requirement set by CHESF to verify the fault coverage and completeness of a test plan. As suggested by Cigre WG B5.32, FMEA and HAZOP are two methods standardized by IEC that can be used to analyze possible failures of a system, and to avail the fault coverage of any proposed test plan. Software components like logical nodes may present many failure modes, such as:

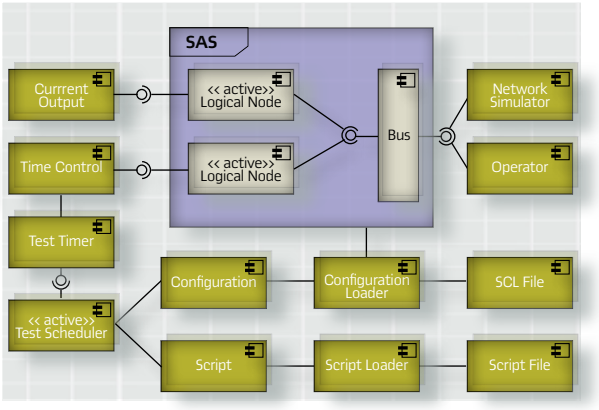
- Wrong parameters
- Wrong code or software bugs
- Wrong or absence of input/output signal/messages
- Wrong timing (delay) for input/processing signals/ messages.

5 Test setup as a UML communication diagram

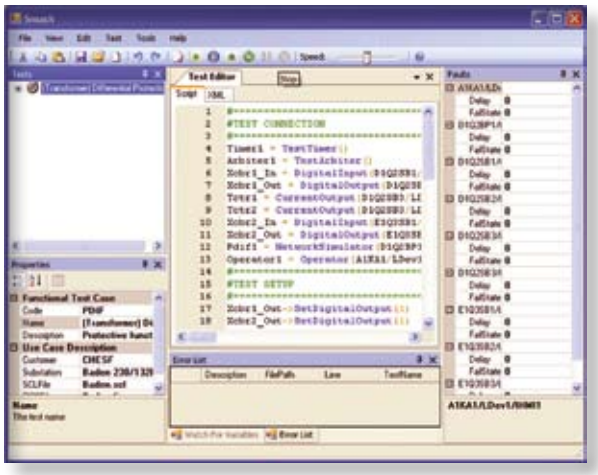


6 Smash Architecture

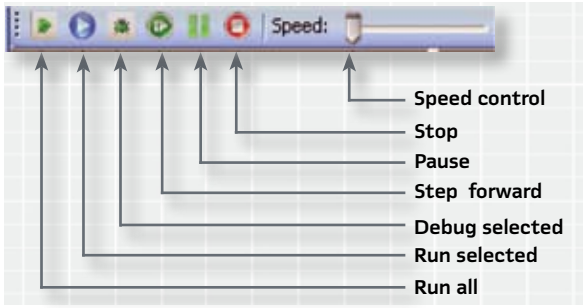
Smash (Smart SAS Test and Fault Diagnosis)



7 Smash Main Screen



8 Test Script Execution Control Tool Bar



> Full tests with real users and systems will validate the design of the tool.

For testing purposes, logical nodes are treated as black boxes, so that their failure modes are detected by loss or degradation of an expected external behavior.

Table 3 shows a simplified FMEA table proposed by Cigre WG B5.32 that can be used to relate possible failure modes (FM) of a system to functional failures (FF) they impact, and to the test cases (T) capable of detecting each failure mode, as shown on the lower part of the table.

To avail the test coverage and the corresponding diagnosing of each test plan, the right panel of the Smash test environment can be used to inject faults on the model and simulate the verdicts generated by the test script. For each set of faults injected, the test verdicts can be used to avail the coverage of the test set, and also to suggest the fault location in the SAS.

Conclusion: This research provides an example of a proof-of-concept implementation of a tool to help automation and protection engineers design and test SASs. Preliminary results show that simple SASs can be modeled, simulated, exposed to failures and automatically tested and diagnosed using an implementation of the Cigre WG B5.32 specification. A GOOSE Viewer and a GOOSE Sniffer are two recent additions to the tool, allowing it to inspect real messages exchanged between the model and the tested SAS. Full tests with real users and systems will validate the design. Further developments are underway to increase the library of models to cover all IEC 61850 logical nodes. ■

The author would like to thank all members of Cigre WG B5.32, and the Smart Diagnostics team at UFCG for the fruitful discussions and implementation effort, and CHESF and ANEEL for the financial support to this research.

9 Depicting test script verdicts

10 Variables' watch panel

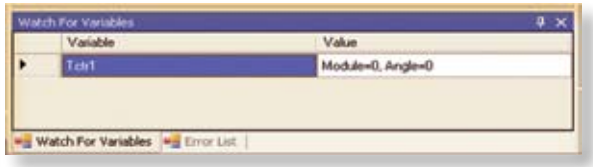


table 3
Failure mode & effects analysis

FMEA		Failure Mode				
		FM1	FM2	FM3	...	FMn
FUNCTIONAL FAILURE	FF1		X		...	
	FF2	X		X	...	X
	...	...	...	...	...	...
	FFn		X	X	...	X
COVERAGE		X	X	X	...	X
TEST CASE	T1	X			...	X
	...			X	...	X
	Tn		X		...	X