IEC 61850, Applications and Benefits, Testing of Devices, Distributed Functions and Systems

Design and Automatic Testing of IEC 61850 Substation Automation Systems

Ubiratan Carmo[1], Jacques Sauvé[2], Wagner Porto[2] , Iony Patriota[1], Tadeu Pereira[1],

[1]Companhia Hidro-Elétrica do São Francisco, [2]Federal University of Campina Grande

Brazil

uacarmo@chesf.gov.br

1 - INTRODUCTION

This paper describes a proof-of-concept software tool that enables automation engineers to build, run and debug functional tests for IEC 61850-based systems in a simulated environment using the test philosophy proposed by Cigre WG B5.32 . The paper is structured as follows. Section 1.1 described the work of Cigré WG B5.32 on Functional testing and section 1.2 describes the problems that this paper discusses. Section 2 describes the architecture and prototype of the tool. Section 3 describes a smash specification. Section 4 an example of a functional test and smash application and finally Section 5 give the conclusion and provides ideas for future directions.

1.1 Context: Testing IEC 61850 systems

The introduction of the IEC 61850 has resulted high added value in the implementation of Substation Automation Systems (SAS). However, although conformance and interoperability tests are subject to standardized approaches, functional and performance testing are not yet subject to standards. Cigré Workgroup B5.32, entitled *Functional Testing of IEC 61850-Based Systems* was formed in 2006 to propose a solution for such testing activities.

The approach taken by WG B5.32 revolves around black-box testing which is a quality assurance process that verifies that an application's functionality works accurately, reliably, predictably and securely [1]. Functional testing consists of a series of tests that emulate the interaction between IEC 61850 intelligent electronic devices (IEDs) and the application in order to verify whether or not the application does what it was designed to do. The proposed solution allows the construction of "test scripts" that can verify functional behavior and performance characteristics.

The solution proposed by WG B5.32 is being submitted to IEC for standardization in the near future.

1.2 The problems

Two problems have been identified and are the reason for implementation this tool:

1. The work done by WG B5.32 needs a proof-of-concept implementation to test its viability in practice.
2. A very useful extension to the testing process can be performed by analyzing the test results and producing a diagnosis of where faults may reside in an SAS.

The paper describes the architecture and preliminary results of a software solution to these problems called Smash.

2 SMASH (SAS TEST AND FAULT DIAGNOSIS) TOOL

The SAS Test and Fault Diagnose Tool named (SMASH), establishing a simulation environment of a SAS where the IEDs use IEC 61850 to perform the communication between them. The smash have the capacity to create a SAS environment simulation that is formed by LNs and substation information read of a SCD file . The LNs are components that simulate the behavior of functions such as differential protection (PDIF), circuit breakers (XCBR), etc..., and realize the communication between them using the 61850 standard specification. The new LNs can be added by third parties to the smash tool.

The smash simulation environment is formed with functional components named SAS , Process Simulator (current Output), Time Control, Test Schedule, Network Simulator, Configuration Loader and Script Load. The Smash architecture is shown in Figure 5.. The definition of Smash component is conform the specification WG B5.38 brochure. And can be resumed as following:

- The SAS component are formed by LN and communication bus. The LNs are “active” classes, meaning that they run in a separate thread. This enables time delays to be introduced in their behaviors. 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

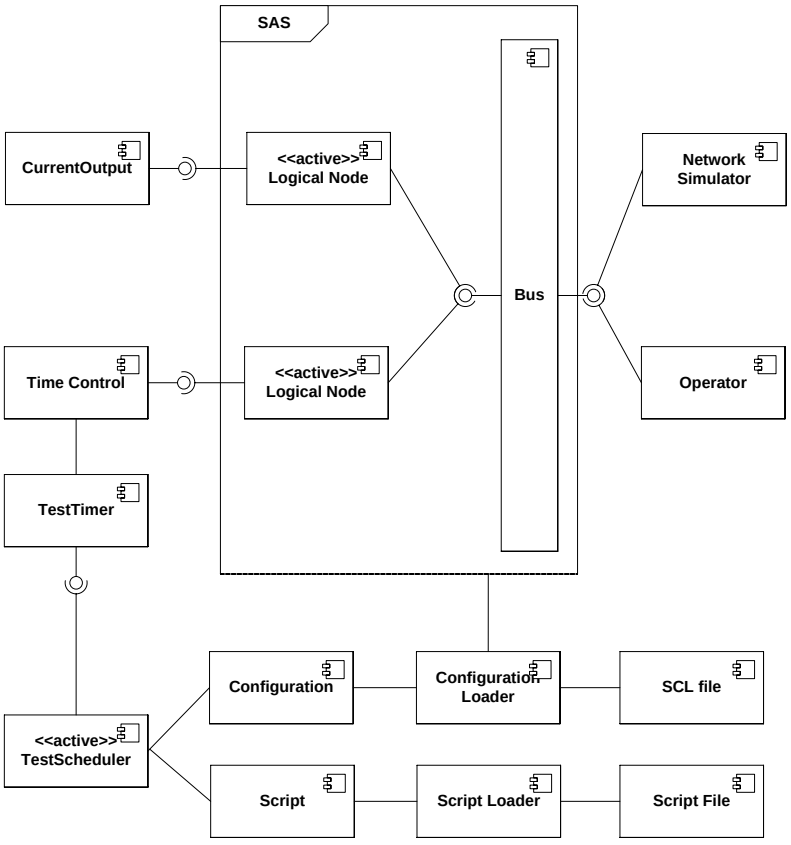

Figure 5: Smash Architecture

All Publish-Subscribe communication between LNs and other test components is controlled by a common bus. This allows the simulated environment to include network delays in the simulation.

- The Process Simulator is a package of classes to emulate the signals that are received and sent from and to the process. These include classes to monitor and send analog, digital and sampled values and messages [1].
- The Network Simulator is a package of classes to supervise and generate network messages related to any logical node. These include methods used to monitor and send network messages containing sampled and digital values.
- The Test Timer is a package of classes to support time related operations, such as real time clocks, timer start and stop, event timing and tagging. It is formed mainly by a class TestTimer derived from a Timer interface, as defined on UML Test Profile.
- The Test Scheduler is a package of classes to start, stop and sequence the steps of a functional testing. It is formed mainly by a class TestScheduler derived from a Scheduler interface, as defined on UML Test Profile.
- Finally the Test Arbiter is a package of classes to avail the results of any test sequence. It is formed mainly by a class TestArbiter derived from an Arbiter interface, as defined on UML Test Profile. For more detail about this class you can consult the [1].

  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. The TestScheduler is the main simulator that interprets and executes script commands.

3 SMASH SPECIFICATION.

The smash functionality of the smash are illustrated as the following specification:

- In a first phase, the system will be used to build and debug tests in a simulated SAS environment only. In a second 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.
- The system execute test scripts and report on the test verdicts. There is full support for all B5.32 test objects (VoltageOutput, CurrentOutput, DigitalInput, DigitalOuptut, NetworkSimulator, Operator, TestTimer, TestScheduler, TestArbiter).
- The SAS is represented by a model and simulated during execution.
- The LNs most commonly used in SAS design is supported.
- The design is component-oriented to allow third parties to develop new LNs and plug them into the system.
- The system provide test script management (script creation, visualization, editing, removal)
- The tool read in SAS models provided in IEC 61850 Substation Configuration Language (SCL)
- The tool provide for visualization and editing of test scripts in a script language and also in XML.
- The tool provide automatic conversion between the script and XML versions of a test.
- The tool perform syntax checking during script editing.
- The test execution environment provide
    - Execution command: Run all, Run selected, Pause, Stop.

- Debugging mode (Run debug, breakpoints, single step, variable watch)
- Simulated time speed control to accelerate or decelerate the simulation as compared to real time.

- The execution environments provide mechanisms for the insertion of faults
- The tool must fault diagnosis functionality through an automatic fault diagnosis algorithm, thus allowing the source of faults to be pinpointed, down to the level of Logical Node.

4 FUNCTIONAL TESTING EXAMPLE

We choice the same SAS differential example used by WG B5.32 for a brief sketch of an this test scenario, can be given here. Please refer to the full WG B5.32 technical brochure for details [1]. The motivation for providing this example is to give the reader unfamiliar with WG B5.32's proposal an outline of the approach so that the rest of the paper may be better understood. The approach is object-oriented and UML, text and XML formats to specify the applications through Functional Use Cases and other Functional Specification documents. We do not show all these documents here due to lack of space. Consider the substation layout diagram shown in Figure 1.

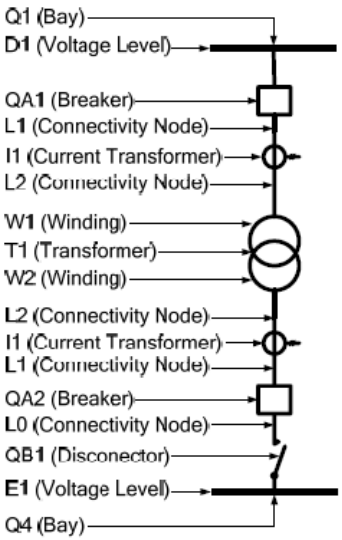

Figure 1: Example substation layout diagram

The functional specification of the system may include Functional Implementation Conformance Statements (not shown), specifying, for example, that XCBR1 and XCBR2 must trip in less than 100 ms upon inception of an internal short circuit in the transformer. In addition, other UML diagrams may be used in the functional specification. For example, a UML communication diagram is shown in Figure 2 and a UML sequence diagram is shown in Figure 3. In Figure 3, the numbers shown are PICOM messages types (12=Operated, 22 = Trip, etc.). The left-hand side of the figure also shows the performance requirements as time delay restrictions.

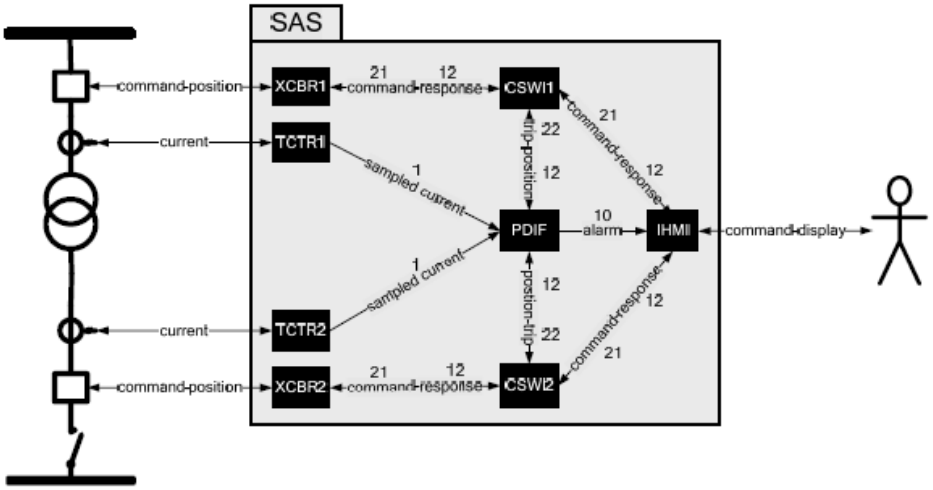

Figure 2: Functional specification by UML communication diagram

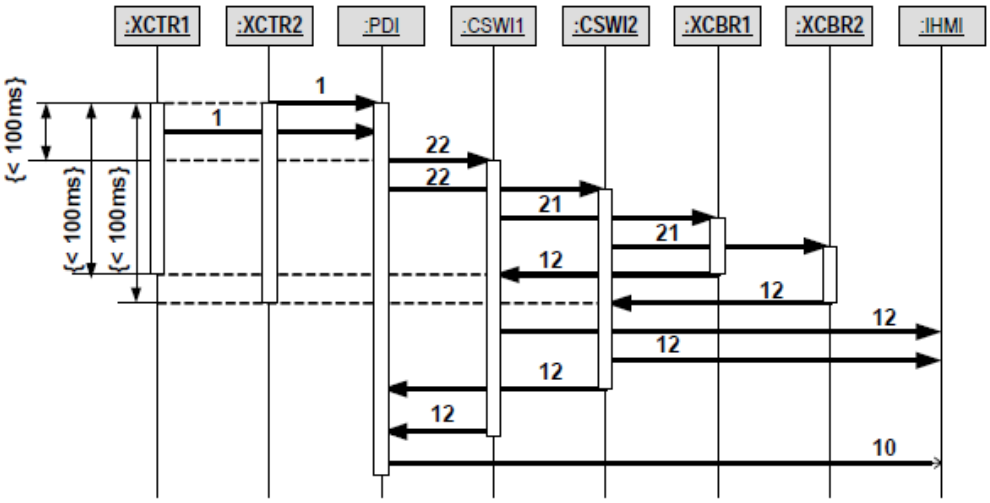

Figure 3: Functional specification by UML sequence diagram

We now move on to the specification of functional tests. WG B5.32 recommends that Failure Modes and Effects Analysis (FMEA) and Hazard and Operability Analysis (HAZOP) be used tools to drive the design of tests and also to investigate the fault coverage attained by test plans. WG B5.32 has suggested a test architecture consisting of several test components used in automatic testing activities. Figure 4 shows the testing objects instantiated from the test device classes necessary to test this example SAS. The figure also shows their connection to the SAS Logical Nodes (LN).

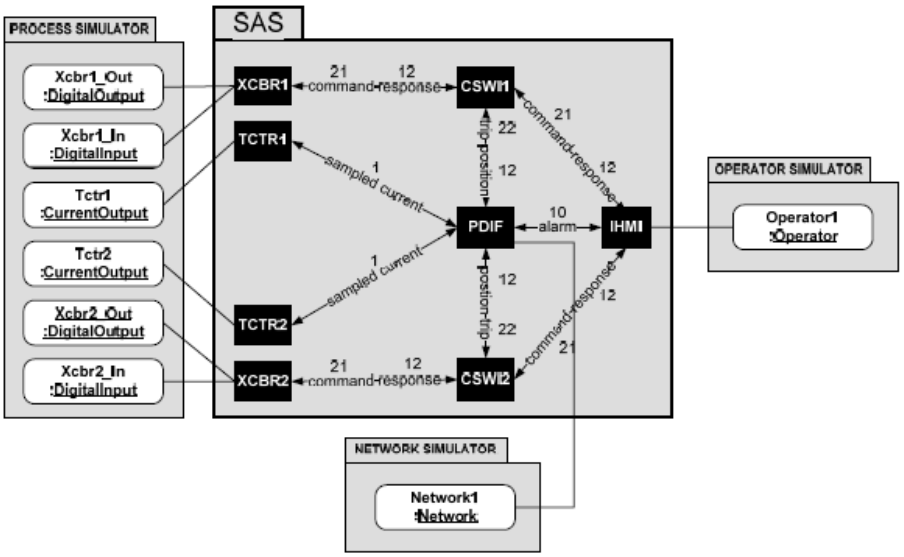

Figure 4: Test setup as a UML communication diagram

Reference [1] describes this setup as follows: “Note that each breaker is modeled by a DigitalOutput and a DigitalInput object, to simulate their command and response messages, while each current transformer is modeled by a CurrentOutput object, to simulate 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 [1]. This is shown below] for the three functions specified in this example SAS.” As one can see, test scripts can specify signals to be injected in the system as well as signals to be expected. The functional test specification is divided into the following parts: test connection, test setup, test start, test stop, test disconnection and test verdict. The table 1 illustrate the test scrip for the example subject this paper.

Each command in this script is a method call supported by the instantiated class. The last 7 commands (verdicts) evaluate the results of the test case. These commands check the time performance of the SAS against the specification (<100ms), as well as operator notification of breaker trippings and operation of the differential protection. Test cases can also be specified in XML.

Table 1: Example test script of SAS with differential relay

Test Connection		
1.1	Timer1 = TestTimer()	Create a timer to measure events
1.2	Arbiter1 = TestArbiter ()	Create a test arbiter to emit verdicts
1.3	Xcbr1_In = DigitalInput (XCBR1)	Create a digital input connected to XCBR1
1.4	Xcbr1_Out = DigitalOutput (XCBR1)	Create a digital output connected to XCBR1

1.5	Tctr1 = CurrentOutput (TCTR1)	Create an analog output connected to TCTR1
1.6	Tctr2 = CurrentOutput (TCTR2)	Create an analog output connected to TCTR2
1.7	Xcbr2_In = DigitalInput (XCBR2)	Create a digital input connected to XCBR2
1.8	Xcbr2_Out = DigitalOutput (XCBR2)	Create a digital output connected to XCBR2
1.9	Pdif = NetworkSimulator (PDIF)	Create a network simulator linked to PDIF
1.10	Operator1 = Operator (IHMI)	Create an operator connected to IHMI
Test Setup		
2.1	Xcbr1_Out->SetDigitalOutput (1)	Prepare to close breaker XCBR1
2.2	Xcbr2_Out->SetDigitalOutput (1)	Prepare to close breaker XCBR2
2.3	Xswi_Out->SetDigitalOutput (1)	Prepare to close switch XSWI
2.4	Tctr1->SetACCurrentOutput (0,0)	Prepare to zero current on node TCTR1
2.5	Tctr2->SetACCurrentOutput (0,0)	Prepare to zero current on node TCTR2
2.6	Xcbr1_Out->StartDigitalOutput ()	Close breaker XCBR1
2.7	Xcbr2_Out->StartDigitalOutput ()	Close breaker XCBR2
2.8	Tctr1->StartCurrentOutput ()	Zero current on transformer TCTR1
2.9	Tctr2->StartCurrentOutput ()	Zero current on transformer TCTR2
2.10	Pdif->GetMessageSequence (1min)	Record messages for 1min to and from PDIF
2.11	Xcbr1_In->GetDigitalIinputSequence (1min)	Record input sequence for 1min from XCBR1
2.12	Xcbr2_In->GetDigitalIinputSequence (1min)	Record input sequence for 1min from XCBR2
Test Start		
3.1	Tctr1->SetACCurrentOutput (5,0)	Prepare 5A on current on transformer TCTR1
3.2	Timer1->Start ()	Start time to measure function delays
3.3	Pdiff->StartNetworkSimulator()	Start recording messages to/from PDIFF
3.4	Time1=Tctr1->StartCurrentOutput ()	Apply 5A to node TCTR1 and record time
Test Stop		
4.1	Wait (2min)	Wait for 2min without processing the script
4.2	Tctr1->SetACCurrentOutput (0)	Prepare to zero current on node TCTR1
4.3	Tctr1->StartCurrentOutput ()	Zero current on transformer TCTR1
4.4	Pdiff->StopNetworkSimulator()	Stop recording messages to/from PDIFF

Test Disconnection		
5.1	Time2 = Pdif->FirstPICOMTo (CSWI1,22)	Get time of first trip from PDIF to CSWI1
5.2	Time3 = Pdif->FirstPICOMTo (CSWI2,22)	Get time of first trip from PDIF to CSWI1
5.3	Time4 = Xcbr1_In->FirstDownInputTransition ()	Get time of opening of breaker XCBR1
5.4	Time5 = Xcbr2_In->FirstDownInputTransition ()	Get time of opening of breaker XCBR2
Test Verdict		
6.1	Verdict1 = Arbiter1->TestArbiterConfirm (Time2-Time1 <100)	Trip of PDIF to CSWI<100ms
6.2	Verdict2 = Arbiter->TestArbiterConfirm (Time3-Time1<100)	Trip of PDIF to CSW2<100ms
6.3	Verdict3 = Arbiter->TestArbiterConfirm (Time4-Time1<100)	Trip of breaker XCBR1<100ms
6.4	Verdict4 = Arbiter->TestArbiterConfirm (Time5-Time1<100)	Trip of breaker XCBR2<100ms
6.5	Verdict5 = Operator1->OperatorConfirm (“PDIF Trip”)	Confirm PDIF trip indication
6.6	Verdict6 = Operator1->OperatorConfirm (“XCBR1 Trip”)	Confirm XCBR1 trip indication
6.7	Verdict7 = Operator1->OperatorConfirm (“XCBR2 Trip”)	Confirm XCBR2 trip indication

4.1 Testing IEC 61850 SAS transform bay

The first step to use the smash is and open an existing project or create a new project. Note to the creation of a new project you must have in hand scl files, test scripts and test requirements. The figure 04 illustrate a smash use interface after open or create new project.

Note that after you upload or creating the project the smash interface (test windows, properties windows, edit windows, fault windows and) show the various elements of the project. In the script windows we can see and edit the scripts test.

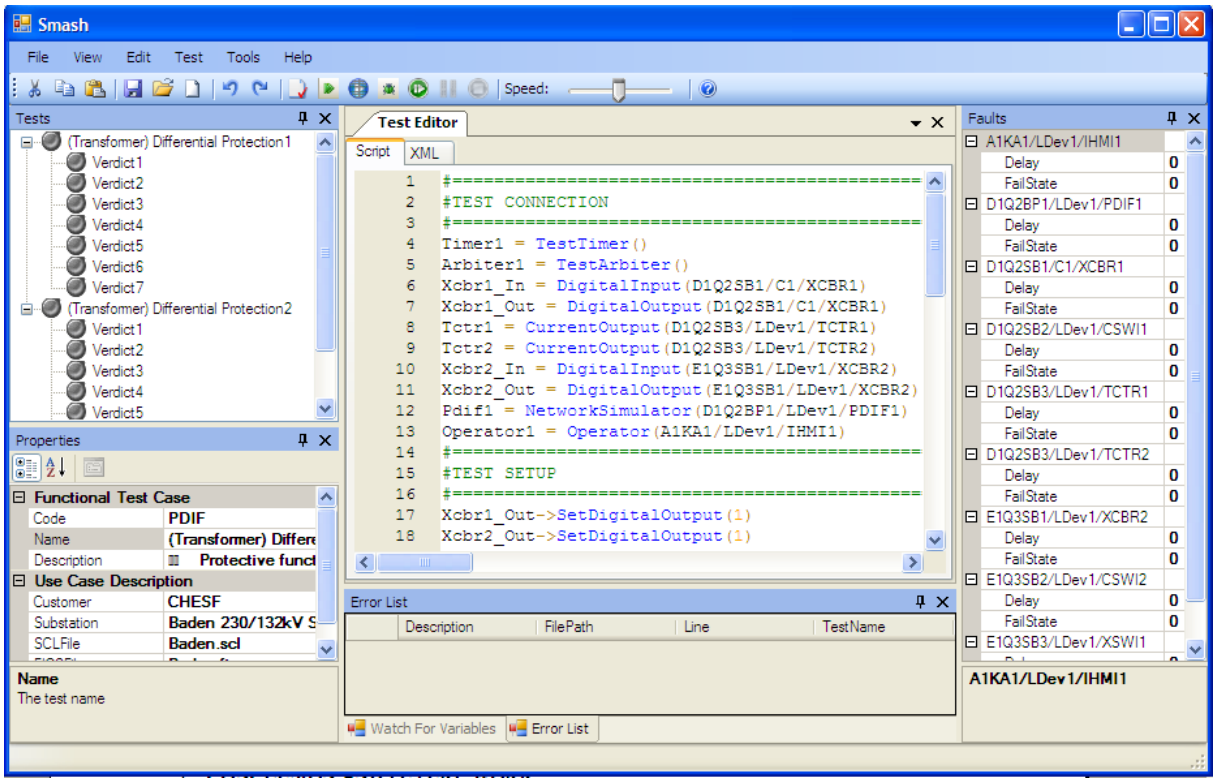

Figure 4: Smash Main Screen

Other interesting smash's functionality is that the user can force error in the IED communication through the fault windows. The spot mark with color red identify the script verdict that is in failure [2]. The figure 05 illustrate a script that had a real verdict failure or a verdict failure caused by the user.

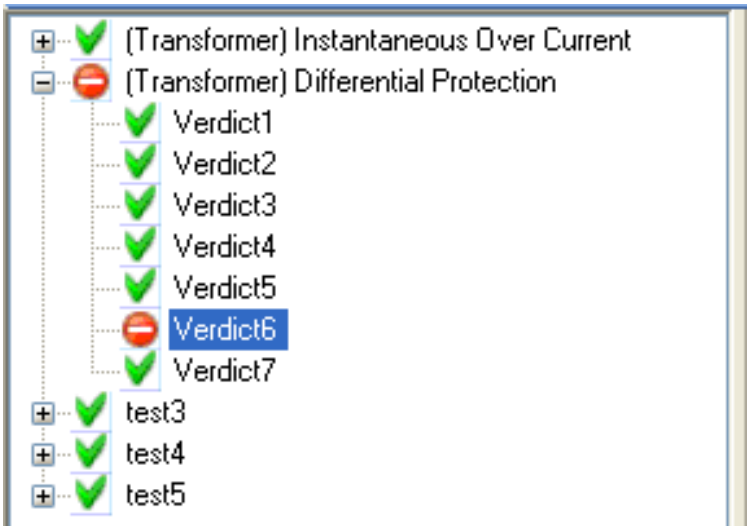

Figure 8: Depicting Test Script Verdicts

The smash's user has the ability to introduce break point when the smash tool is in debug operation mode. The figure 06 illustrate smash script using a break point (color yellow and brow) to asset the functionality failure in the smash's debug mode.

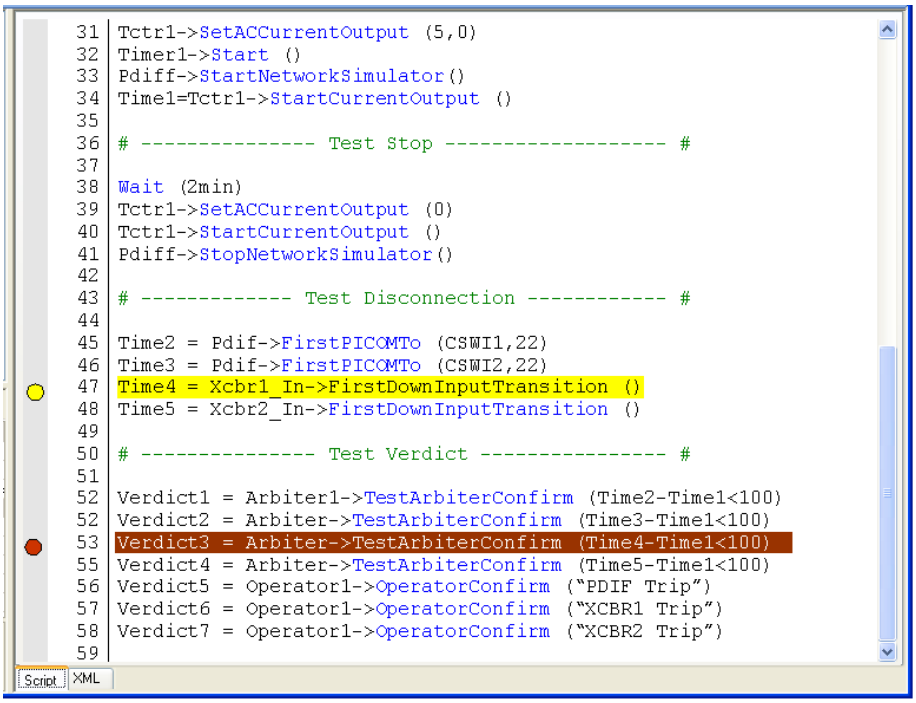

Figure 9: Smash breakpoints

The smash tool also have the capability to show to the user a GOOSE map where the user can see the GOOSE IN and GOOSE out by IED. The GOOSE map is a kind of wire "From/To" table used by the commissioning team for long time. The figure 10 illustrate the GOOSE map created by the smash tool.

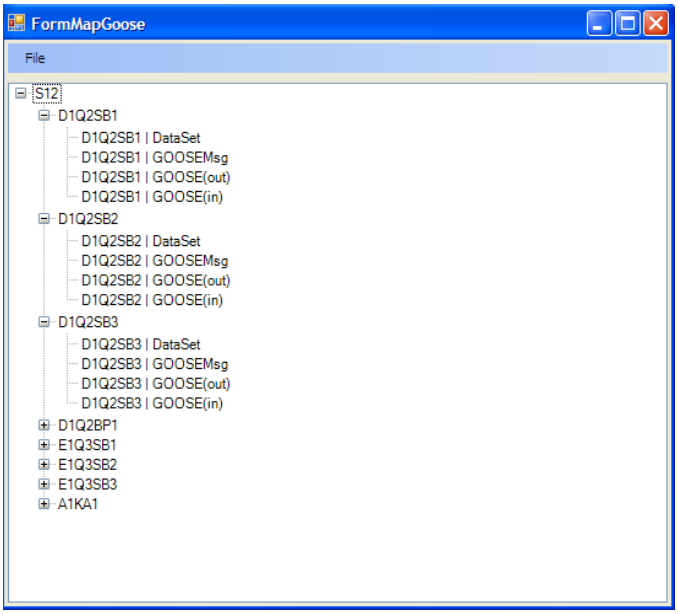

Figure 9: Smash GOOSE map

5 CONCLUSION AND FUTURE WORKS

As illustrated in this paper the tool smash made SAS simulation of the example of the transformer bay studied by the WG B.32 Cigré attesting to full compliance as specified in the brochure produced by this working group.

News testing shall be performed by Hidroelectric Company of San Francisco – CHESF and Campina Grande Federal University - UFCG using real SAS scenarios and the test scripts adhering to these scenario in order to test the performance and scalability of the smash tool.

The problem we set out to solve is to provide a proof-of-concept implementation of a tool to help automation and protection engineers design and test SASs.

6 ABSTRACT

This paper discusses the design and effectiveness of a tool meant to ease the design and test of IEC 61850 systems. The tool is based on the work of CIGRÉ group B5.32. The B5.32 script language is fully supported by Smash, thus allowing the substation automation engineer to test the SAS design in simulated mode. Smash offers a syntax-driven editor to build test scripts.

Also the Smash offers several views of the SCL files to ease understanding. For example, a table offers a view of GOOSE messages exchanged by logical nodes, logical device datasets, etc.

Smash can be connected to the substation network and interfaces itself to test boxes. There are two ways to perform tests on a real SAS. First, Smash can convert the B5.32 test scripts to the test box's script language and let the test box run tests; second, Smash can remain in full control of testing, letting the test box merely generate the required signals to the SAS.

The importance of the above features is that an SAS can now be designed and tested before any implementation and commissioning is performed. Additional testing is then done during commissioning. Furthermore, we expect that, if the B5.32 work is successfully accepted by the community, the specification of an SAS will be possible by providing SCL and tests scripts which will serve not only as a design specification but also as a set of acceptance tests run during commissioning.

7 REFERENCES

[1] *Functional Testing of IEC 61850-Based Systems Technical Brochure*, Cigré Workgroup B5.32, December 2008.

[2] *Smash User Handbook, Version 4.0,CHESF – UFCG, November 2009.*

8 BIOGRAPHY

Ubiratan Alves do Carmo, born Brazil, on April 03, 1955. He graduated in Electrical Engineering from Universidade Federal de Pernambuco - UFPE in 1979 and won the title of Master in Computer Science from UFPE in 2003. She currently works in the Hydroelectric Company of San Francisco - CHESF since the year 1980 and holds the post of manager of the Measurement and Process Control Division - DOMC. Member CIGRÉ and IEEE since 2003. Actually is member of B5 JWG D2-B5-30 - Communications for HV Substation Protection & Wide Area Protection Applications.