401

Functional Testing of IEC 61850 Based Systems

Working Group
B5.32

December 2009

Functional Testing of IEC 61850 Based Systems

Working Group B5.32

December 2009

Members

Iony Patriota (BR) – (Convener), Aitzol García (ES), Pascal Postec (FR),
Marcelo Paulino (BR) , Alex Apostolov (US) , Tetsuj Maeda (CH), Dennis Holstein (US), Fred Steinhauser (AT), Stephen Thompson (GB), Marco C. Janssen (CH),
Hyuk Soo Jang (KR), Jian-cheng Tan (CA)

Corresponding Members

Damien Tholomier (FR), Marcus Steel (AU), Gareth Baber (GB),
Benemar Alencar (BR), Ubiratan Carmo (BR), Benton Vandiver (US) ,
Ricardo Cartaxo (PO), Allan Cascaes (BR), Ren Yanming (CN),
Sun Bo (CN), Byung-Tae, Jang (KO)

ISBN: 978-2-85873-088-9

Table of Contents

List of Tables

List de Pictures

Abstract

Introduction of IEC 61850 standard on Substation Automation Systems requires new ways to test their performance and functionality. This brochure describes a structured method to specify functional tests on systems based on this standard. An object oriented approach is proposed, using UML, text and XML formats. Conformance and interoperability tests are not treated, being already standardized.

Key words
IEC 61850, Functional Test, Performance Test, Substation Automation Systems.

Executive Summary

Today's IEC 61850 solution providers and product vendors face a triple challenge. They must optimize the quality of increasingly complex protection and automation systems – and to do so more quickly and cost-effectively than ever before – in order to deliver reliable systems that yield a high return-on-investment and provide a competitive advantage.

Time and time again, IEC 61850 solution providers and product vendors have successfully demonstrated the power of automated functional testing to thoroughly test, rapidly develop and reduce the cost of delivering high-quality/reliable protection and automation applications. Yet, 85% of those vendors that attempted comprehensive functional testing fail. This high failure rate is symptomatic of a mindset that views comprehensive functional testing as a quick, turnkey solution.

There is no question that functional test automation can allow organizations to produce higher-quality products while leveraging existing resources. However successful organizations realize that a comprehensive functional testing solution is only one part of the answer. They must make an investment in planning a quality-driving development process, training quality-related personnel and developing the right test framework.

This technical brochure provides practical insight into the lessons learned by those that have successfully followed the guidelines described. It answers questions such as:

- What is the strategic role of functional testing of IEC 61850 systems?
- What are the value-added benefits associated with implementing a comprehensive functional test program?
- What is the best approach to ensure the success of your functional testing efforts?
- What should you look for in functional testing of IEC 61850 systems?

To be more specific, this brochure builds on the concept of black-box testing which is a quality assurance process used to verify that an application's functionality works accurately, reliably, predictably and securely. Functional testing can involve either manual or automated methods. Either way, it entails a series of tests that emulate the interaction between IEC 61850 intelligent electronic devices and the application in order to verify whether or not the application does what it was designed to do.

As described in this brochure, the comprehensive approach uses the Unified Modeling Language and Use Cases to describe the applications from a user point of view. Failure Mode and Effects Analysis augmented with Hazard and Operability Analysis as defined by IEC are recommended as suitable tools to list the possible faults on components, logical nodes and functions to ensure complete test coverage. Such an approach delivers the long term advantages of reliability, predictabili-

ty and consistency, and productivity that shorten test cycles and increase product quality.

Collectively, these advantages enable quality assurance to accurately assess quality levels, make sound decisions regarding release readiness and minimize deployment risk.

Résumé

Aujourd'hui, les fournisseurs de solutions CEI 61850 et les vendeurs de produits associés sont confrontés à un triple défi. Ils doivent optimiser la qualité de plus en plus complexe des systèmes de protections et d'automatismes - et le faire plus rapidement et efficacement que jamais – dans le but de fournir des systèmes fiables pour produire un fort retour sur investissement et obtenir un avantage compétitif.

À maintes reprises, les fournisseurs de solutions CEI 61850 et les vendeurs de produits ont démontré avec succès la capacité des tests fonctionnels automatisée à tester complètement, à développer rapidement et réduire les coûts de fourniture de solutions de protections et d'automatismes de haute qualité et de haute fiabilité. Pourtant, 85% des vendeurs qui ont tenté ces tests fonctionnels complets ont échoué. Ce fort taux d'échec est symptomatique d'un état d'esprit qui perçoit ces tests fonctionnels complets comme une solution clé en main rapide.

Il n'est pas question de remettre en cause le fait que l'automatisation des tests fonctionnels puisse permettre de produire des systèmes de qualité supérieure tout en exploitant les ressources existantes. Mais les organisations qui l'ont réalisé avec succès constatent qu'une solution de tests fonctionnels exhaustifs n'est qu'une part de la réponse. Ils doivent faire un investissement dans une planification de qualité du processus de développement, la qualité de la formation du personnel et la mise en oeuvre d'un bon environnement d'essais.

Cette brochure technique fournie une application pratique issues des leçons retenues par ceux qui ont suivi avec succès les directives proposées. Il répond à des questions telles que:

- Quel est le rôle stratégique des tests fonctionnels des systèmes basés sur la norme CEI 61850?
- Quels sont les bénéfices associés à la mise en oeuvre d'un programme exhaustif de tests fonctionnels?
- Quelle est la meilleure approche pour garantir la réussite de vos efforts en matière de tests fonctionnels ?
- Que devez-vous rechercher dans la réalisation de tests fonctionnels sur les systèmes CEI 61850 ?

Pour être plus précis, cette brochure s'appuie sur le concept de tests type « boîte noire » qui est un processus d'assurance qualité utilisé pour vérifier que la fonctionnalité d'une application répond correctement, de manière fiable, prévisible et sûre. Les essais fonctionnels peuvent être réalisés par des méthodes manuelles ou automatisées. Quoi qu'il en soit, il comporte une série de tests qui émulent l'interaction entre équipements CEI 61850 et l'environnement afin de vérifier si l'application peut réaliser ce pour quoi elle a été conçue.

Comme décrit dans cette brochure, l'approche globale utilise l'« Unified Modeling Language » (UML) et des cas d'utilisation pour décrire les applications d'un point

de vue utilisateur. L’analyse des modes de défaillance et des conséquences complétés par l'analyse de risque et d'exploitabilité tel que défini par la CEI, sont les outils pratiques recommandés pour lister les éventuels défauts sur les composants, les nœuds logiques et les fonctions afin d’assurer une couverture complète des essais. Cette approche offre les avantages à long terme de fiabilité, de prévisibilité et de cohérence, ainsi que de productivité qui raccourci les cycles d’essais et augmente la qualité du produit.

Ensemble, ces avantages permettent d'obtenir les garanties nécessaires à l’obtention, avec précision, des niveaux de qualité requis, de prendre des décisions judicieuses concernant la libération du produit et de minimiser les risques liés au déploiement.

1. Introduction

Over the last few years considerable work has been carried out in the development of a new standard IEC 61850, “Communication Networks and Systems in Substations” which is set to make wide ranging changes to the way substations and power systems are designed, built, commissioned, operated, maintained and extended. This standard is unique in technology respects in that whilst it is implemented in individual devices, it is directed at system wide benefits.

The current state of the art in substation design is based on a master-slave relationship between Intelligent Electronic Devices (IEDs). As such, communication between networked IEDs is very deterministic – the master requests an action and the slave executes the response in a prescribed manner. IEC 61850 introduces mechanisms to operate networked IEDs in a peer-to-peer relationship. For example, IEC 61850 introduces the use of a state change message that notifies all listening IEDs that one or more of the IED’s functions has changed state. It is the responsibility of the receiving IED to respond in an appropriate manner. This capability introduces functional testing challenges, which is the subject of this report.

The purpose of this report is therefore to provide utility management and their technical support staff an overview of the functional testing of IEC 61850 based systems. IED vendors, tool suppliers, and system integrators should use this technical brochure to ensure that their products and services provide the capabilities needed to execute the functional test plans and procedures tailored for each utility’s 61850 based system configuration.

1.1 Limitation on the scope of functional testing

Part 10 of IEC 61850 (61850-10) addresses conformance (type and interoperability) testing and as such this subject is not addressed in this report; nor are the IED physical characteristics addressed. Furthermore, the members of the CIGRE Task Force decided that this brochure should not attempt to comprehensively address all possible testing scenarios. Specifically it should not address testing of non-functional requirements such as maintainability, reliability, availability, usability, security, etc. They decided to use a few Use Case and other UML examples to describe the functional testing methodology, and leave it to future working groups to address the scenarios in their area of interest.

1.2 A unified approach for functional testing

Any approach to functional testing of IEC 61850 must consider the requirements for real-time performance, failure modes and effects, and tools needed to efficiently conduct the testing. Clearly a coherent and unified approach for functional testing is needed, which led the task force members to select the Unified Modeling Language (UML) to describe the IEC 61850 functions and test environment which can also be used for describing the architectural arrangement of the system-under-test (SUT), test drivers and measurement units. The relationships among SUT components and the test environment components are defined as objects in

the UML vernacular. The UML diagrams proposed are more fully described in Section 2.4. Failure Modes and Effects Analysis (FMEA) and Hazard and Operability Analysis (HAZOP) are suggested as tools to investigate the fault coverage attained by test plans. FMEA and HAZOP methods are more fully described in Section 4.5.

1.3 An IEC 61850 functional test framework

We conclude the introduction by reiterating the objective of this technical brochure. The number of protection and automation applications based on IEC 61850 and the functional test scenarios is only limited to the imagination of the substation protection and automation engineer design team. In effect, there is no limit to the number of applications and test cases that can be envisioned. For this reason, this technical brochure defines a framework or methodology for functional testing of IEC 61850 based systems and evaluation of its fault or test coverage, which validates this approach using a complete example. We leave it to others to use this framework to explore the functional testing requirements for applications and detailed scenarios in their area of interest. Their feedback and recommendations to improve this framework is expected and should result in a future update of this technical brochure.

1.4 A guide to reading this report

This technical brochure is not a tutorial describing the technical details of IEC 61850 or the underlying technologies of functional testing. We assume the reader has this knowledge base and will use the cited references in this report as a supplement. We have attempted to write each chapter using the language of protection and automation engineers, and we have provided intuitively clear visuals (diagrams, pictures) to ease the pain of translating these recommendations into concrete test plans and procedures.

The second chapter, Functional Requirements, describes the methods for specifying functional and performance requirements for protection and automation systems that are useful for designing functional test cases. UML is described as the recommended specification tool, supplemented by design documents using IEC 61850-6's Substation Configuration Language (SCL). SCL is only related to the UML specification during test execution, after the system has been integrated and both specifications include the necessary level of detail. The basic idea is for the test tool, to get access points for signal injection and monitoring by importing the UML specification as a script and importing the SCL package to gain access to all network points.

The third chapter, Test Requirements, is a review of the different types of functional test requirements common to protection and automation systems, and their attributes that distinguish them from conformance and interoperability tests as defined in IEC 61850-10.

The fourth chapter, Test Coverage, the concept of test coverage is introduced as a tool to define the required number of test cases. Test coverage is the method used

to ensure that all fault and function test objectives are addressed. Chapter 4 also describes the use of FMEA and HAZOP tools to list the possible faults of a component, logical nodes and functions of IEC 61850 based systems. These tools are proposed to assess the fault coverage of any proposed test plan.

The fifth chapter, Functional Test Tools, describes the architecture of the proposed functional testing system. UML Test Profile, as defined by the Object Management Group (OMG), is used as a conceptual framework to define the test components, simulators, and their behavior as needed for testing protection and automation systems.

The sixth chapter, Functional Test Specification, shows how to design a functional test case based on the proposed architecture. Test cases are structured in phases, which are described in an object-oriented way complemented with a user-friendly table, and optionally in XML format. XML details are presented in an appendix of this report.

The seventh chapter, Test Case Example, illustrates the proposed method by specifying the functions of an example SAS system, designing the test cases, and evaluating their fault coverage. The example will strictly obey the approach defined in previous chapters.

The eighth chapter, Conclusions, summarizes the testing approach taken in the brochure, and explores future developments in this field.

Appendix A, Functional Specification Template, is a proposed model for specifying the functions of an SAS system, according to the approach taken in this brochure.

Appendix B, Functional Test Specification Template, is a proposed model for specifying functional testing of an SAS systems, based also on the approach taken in this brochure.

Appendix C, XML Schema for Functional Test, contains the listing of an XML Schema for specifying test cases, according to the test architecture proposed in this brochure.

Appendix D, UML Overview, introduces the basics of UML and describes how different UML diagrams can be used to provide a complete picture of the system being described.

2. Functional Requirements

2.1 Introduction

Functional requirements express a project specification from an external or user point of view. For SAS testing purpose, functional requirements should be identified and documented as the standard against which functional test results should be compared for approval.

This chapter covers the different ways SAS functions can be specified, and the changes in functional specifications brought by the introduction of IEC 61850.

2.1.1 Methods to define a function

Nowadays, automation and protection functions are defined by utilities in a textual form, possibly together with logical equations, tables and state diagrams. The actual method of specification varies from utility to utility, depending on the existing philosophy. For instance, the interlocking function of a high voltage apparatus can be specified by means of an equation, a ladder diagram or even a table listing the conditions that permit (or inhibit) its operation.

This approach, whilst being the historical way of doing it, has the disadvantage of not leading to a standardized way of specifying functions. To overcome this limitation, new specification methods have to be adopted and this is the aim of the chapter.

As previously discussed in Section 1, it has been decided to use the UML 2.0 (Unified Modelling Language), as the basis for that purpose, and SCL (Substation Configuration Language), the language defined by the IEC 61850 to achieve interoperability among configuration and test tools, thus automating the testing procedure.

It must be stressed, however, that the IEC 61850 does not define functions, only their interface to the communication network, so the SCL schema presented in this standard does not cover function definition. Consequently it will have to be extended to cover the internal logic of the IEDs.

This chapter defines formal ways to specify functions and their performance requirements. Some example cases will be used throughout the chapter, illustrating how the different elements of the functional and performance specification (i.e. text, SCL, UML) compare and interact.

2.1.2 Introduction to the next sections

In the second section the chapter functions will be grouped into classes and, for each one, the applicable ways of specification are indicated. Reference is made to the SAS Functional Specification Template, in the appendix.

The third section is devoted to the functional specification using the SCL. This language can describe the allocation of user functions, the data structure of the devices and the communication between Logical Nodes, via the communication network.

The fourth section will use UML to specify SAS functional requirements. A subset of diagrams from UML will be used in this brochure. These are

- Use Case Diagrams;
- Communication Diagrams;
- Sequence Diagrams;
- Activity Diagrams;
- State Diagrams; and
- Deployment Diagrams.

System performance requirements will be covered in the fifth section of the chapter. They will be split into 'functional performance requirements' (for which classes of performance are defined) and 'logical node performance requirements' (defined by means of the PICOM type and classes defined in IEC 61850-5).

Finally, in the sixth section, UML will be used to specify the performance requirements, primarily using Sequence Diagrams of each function.

2.2 SAS Functional Specification

2.2.1 Definition of a function

Mathematically, a function is a transformation that, given the values of n inputs, produces the values of m outputs. In a SAS, the situation is quite similar, that is, to define a function there must be a way to express the outputs in terms of the inputs. Typical inputs and outputs for SAS functions are:

Inputs

- Digital values from the process (e.g. status of the circuit breaker, tap position);
- Analogue values from the process (e.g. currents, voltages);
- Commands from local and remote operators (e.g. open isolating switch, raise voltage);
- Internal variables.

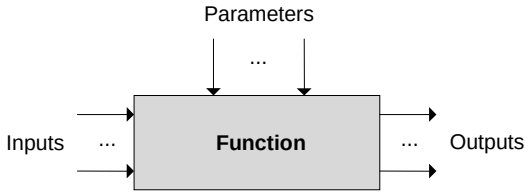
Outputs

- Commands to the process (e.g. open breaker);
- Events and alarms to the operator (e.g. protection pick-up);
- Events and alarms to the data-logging;
- Internal variables (e.g. authorization to operate a circuit breaker).

The function's behavior is also determined by its parameters, which are defined in IEC 61850-4 as 'variables that define the behavior of the functions of the SAS and the IEDs within a given range of values'. Examples are listed below:

- Maximum voltage difference in synchronism-check function;
- Reclose time in Autoreclose function;
- Start value in Overcurrent protection function.

Picture 1 depicts the concept of a function. It must be stressed that the outputs are not necessarily activated simultaneously with the inputs, since functions may incorporate time delays.



Picture 1 – Representation of a Function

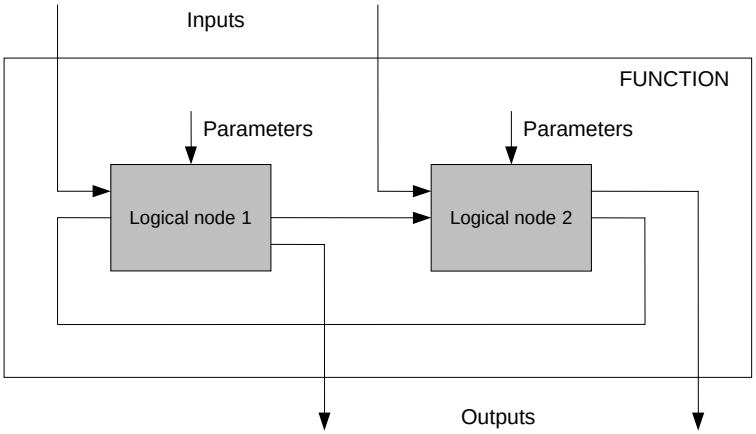
2.2.2 Specification in terms of IEC 61850

The IEC 61850 standard defines entities called Logical Nodes (LN), which are 'the smallest part of a function that exchanges data'. A complete function is performed by means of the interaction of one or more LNs, thus some functions may be completely realized inside a single LN and others may need several acting together. In addition, the deployment of logical nodes across IEDs is also a factor in the way that a specific function is realized. A function may be completely encapsulated inside a single IED (itself containing several LNs), or it may be performed by 2 or more IEDs using IEC61850 communication between them.

So, the specification of a function requires the definition of the following:

- Listing of the IEDs involved;
- Listing of the LNs involved;
- Inputs/outputs of the LNs;
- Parameters of the LNs;
- Behavior of the LNs;
- Exchanged messages between LNs;
- Performance requirements.

Picture 2 shows an example, in a generic way, of a function executed by two LNs. This illustrates their interaction as well as the communication with the exterior of the function.



Picture 2 – Decomposition of a Function

2.3 SCL Functional Specification

As defined on IEC 61850, SAS functions can be specified directly using SCL, the Substation Configuration Language. As its name implies, this XML standard is ideal to define the design functional specification, which may differ from a user functional specification. Although based on pure text, XML and SCL need specialized skills, patience and training to be read, being ideal for computer interpretation. Users normally are not expected to be trained to specify SAS functionality in SCL as this would be produced by appropriate tools.

Being a design specification, SCL contains details needed to allocate logical nodes to functions, not necessary from a user functional specification, but required for functional testing. This aspect is important as it allows the test engineer to get network access to all logical nodes and messages, as points for signal injection and monitoring during testing.

In summary, a SCL functional specification is necessary to design functional tests, together with a user functional specification. The former provides access to network points and logical nodes, while the later supplies desired functional behavior from a SAS. The user functional specification can possibly be expressed in text of using UML.

2.4 UML Functional Specification

2.4.1 UML Overview

The Unified Modeling Language (UML 2.0) is a standard means by which complex conceptual structure and functionality can be represented and communicated using easy to grasp diagrams and tables. UML is applicable to a wide variety of applications from business processes to software engineering, although it is perhaps in the latter application area that it has its greatest use.

This section describes a subset of UML diagrams that are recommended for use in describing the functions within the substation under test. It is assumed that the

reader is familiar with UML diagramming concepts. However, Appendix D gives a short introduction to UML diagrams for those who aren't.

2.4.2 Applicable UML Subset

The full UML 2.0 specification describes a large number of different diagram forms and structures that are applicable to a multitude of procedures and situations. Many of these diagrams have limited or insignificant use in the description of SAS functionality. To avoid unnecessary complexity in diagram usage, and to impose conformity on the use of UML in the modeling of SAS functionality, it is recommended that a primary subset of diagrams is used to describe SAS functionality, with a secondary subset of UML diagrams being employed only to provide additional explanation or clarification of SAS functionality where necessary.

The primary subset of UML diagrams consists of the following:

- **Use Case Diagrams** – to indicate expected behavior from a user or client perspective of the function being described
- **Communications Diagrams** – to describe the message flows expected in the relevant function
- **Sequence Diagrams** – to describe the sequence of events and messages within the relevant function
- **Deployment Diagrams** – To describe the physical allocation of logical nodes, IEDs and network apparatus.

The secondary subset of UML diagrams can optionally be used to document additional information which may be necessary and appropriate to the functional description. Typically these will be

- **Activity Diagrams** – To elaborate further on expected behavior of the function being described, especially in terms of procedural flow.
- **State Diagrams** – To indicate that the function being described has several states, possibly each one of which exhibits differing behavior.

These diagrams are recommended to expand on functionality which is complex or has several possible modes/states of operation. They are not required to be provided for all functional descriptions where those functions can be adequately described by the primary subset.

Note that examples of the use of the various diagram types appear in this section in order to demonstrate how they can be applied. These examples use everyday functions and requirements to illustrate their use. Examples of their use in a SAS are given in chapter 7.

2.4.3 Use Case tables and Diagrams

2.4.3.1 Use Case Overview

A Use Case presents a view of the behavior of a function or a system from the perspective of a client of that function or system. It is a top-level view which doesn't provide details of how functions are performed, instead it focuses on expected outcomes (Goals) from user/client provided input and data. Users/clients in a use case are known as Actors, who have interaction with a system/function in order to achieve the goal. Actors provide input (requests, data) into the system/function to initiate or continue a function. The following types of actors are common to SAS:

- **Operators** – station and remote system operators that use and request functions from the SAS;
- **Process** – station equipment connected to the SAS;
- **Engineering** – maintenance and other engineering people that design, test, configure or maintain the SAS.

A Use Case consists of a number of Scenarios which capture different possible outcomes that can occur in the system or function. The normally expected outcome, i.e. the use case showing successful achievement of the goal along the normally expected route, is known as the Main Success Scenario (MSS). Scenarios resulting in the same goal achieved by a different route are possible and are known as 'Alternatives'. Other scenarios in which the outcome is different from the MSS, i.e. failure conditions where the main goal is not achieved, are known as 'Extensions'. A combination of an MSS, with optional Alternatives and Extensions is generally sufficient to describe even very complex requirements.

2.4.3.2 Use Case Presentation

A Use Case is most usefully presented using text which can describe the MSS and the Extensions. In the earlier example use case of a driver starting a car the MSS would describe the case of the car starting correctly first time. An alternative is defined where the car is started by a different method, for example a push start or a starting handle. Extensions describe the scenarios where the car fails to start for various reasons (i.e. the goal is not achieved).

Now let's take an SAS related example of an operator commanding the system to manually close the circuit breaker, either locally or remotely. The MSS could be as follows:

MSS (Local Operation)

1. Local workstation enters command to close the circuit breaker
2. System checks if local commands are enabled
3. System checks if closure of CB is permitted
4. System starts manual close timer and waits for expiry
5. System performs interlocking check to validate that CB can be closed
6. System initiates CB closure and waits for closure to complete.
7. System checks that CB has closed successfully
8. System informs local workstation of successful closure of CB

Extensions

2a. Local commands are not enabled;
.1 System advises local workstation of command failure
Exit

3a. closure of CB is blocked;
.1 System advises local workstation of command failure
Exit

5a. Interlock checks prevent closure of CB;
.1 System advises local workstation of command failure
Exit

7a. CB fails to close;
.1 System advises local workstation of CB closure failure
Exit

MSS (Remote Operation)

1. Remote Workstation enters command to close the circuit breaker
2. System checks if remote commands are enabled
3. System checks if closure of CB is permitted
4. System performs interlocking check to validate that CB can be closed
5. System initiates CB closure and waits for closure to complete
6. System checks that CB has closed successfully
7. System informs remote workstation of successful closure of CB

Extensions

2a. Remote commands are not enabled;
.1 System advises remote workstation of command failure
Exit

3a. closure of CB is blocked;
.1 System advises remote workstation of command failure

Note that in the remote operation use case the manual close timer is not required.

In an IEC61850 based system the actions provided by ‘System’ will be performed by logical nodes. The logical nodes involved in this sort of operation would typically be

- IHMI – Human Interface (local);
- ITCI – Telecontrol interface (remote);
- CSWI – Switch Controller;

- XCBR – Circuit Breaker Controller;
- CILO – Interlocking function.

Furthermore some operations in both use cases are common and could be moved into a secondary use case that could then be referenced from the main use case. Thus, moving common use case operations into a secondary use case, and substituting the actors for logical nodes, the resulting use cases could be as follows –

MSS (Local Operation)

1. Operator enters command at local workstation to close the circuit breaker
2. IHMI checks if local commands are enabled
3. <<Check and Close CB (local)>> (Reference to a secondary use case)
4. IHMI informs workstation of state of CB

Extensions

2a. Local commands are not enabled;

.1 IHMI advises workstation that local commands are not enabled

Exit

MSS (Remote Operation)

1. Operator enters command at remote workstation to close the circuit breaker
2. ITCI checks if remote commands are enabled
3. <<Check and Close CB (remote)>> (Reference to a secondary use case)
4. ITCI informs workstation of state of CB

Extensions

2a. Local commands are not enabled:

Check and Close CB

1. CSWI checks if closure of CB is permitted
2. CSWI starts manual close timer and waits for expiry
3. CILO performs interlocking check to validate that CB can be closed
4. XCBR initiates CB closure.
5. XCBR checks that CB has closed successfully
6. CSWI updates state of CB to ‘Closed’

Extensions

1a. closure of CB is blocked;

.1 CSWI update state of CB to ‘blocked’

Exit

2a. Operation is remote, no timer required

.1 Continue from 3

3a. Interlock checks prevent closure of CB;

.1 CSWI update state of CB to ‘interlock blocked’

Exit

5a. CB fails to close;

2.4.3.3 Use Case Template

A use case template is proposed which is used to formalize, provide consistency and aid completeness, in the definition of use cases. The overall template is known as the Functional Implementation Conformance Statement (FICS) and contains three sections.

a) **Functional Overview**

This section is the main introductory section and describes the general details of the function that the use case is going to describe.

Functional Implementation Conformance Statement	
Code	Short distinctive name of the function from the SAS
Name	Phrase that declares the main objective of the function
Description	Longer description or summary of the function objective
Customer	Identification of owner, contractor or integrator of substation
Substation	Identification of substation
SCL File	Substation configuration language file and version, if available
Primary User (Actor)	Role name or description of the primary functional actor/user of the use case among people, system, etc. (ex. Operation, Engineering, Process, Dispatch)
Secondary User (Actor)	Role name or description of the secondary functional actor/user of the use case among people, system, etc. (ex. Operation, Engineering, Process, Dispatch)
Stakeholder & Interest	Name of the stakeholder and interest of the stakeholder in the use case

b) **Function Description**

This section describes the related aspects, such as triggers and post conditions, of the function that the use case is going to describe.

Function Description	
Trigger	Which action(s)/event(s) of the primary/secondary users initiate the Use Case
Components or Logical Nodes	(Codes of) Component(s) architecture or Logical Node(s) that implement or realize the function or use case, taken from the SCL file, if available
Process Equipments	(Codes of) Associated substation process equipments, affected by the function
Performance	Goal or quantification of function objective (Ex. execution time, records accuracy, etc.)
Preconditions	Expected state of the automation system, substation or its environment before the use case may be applied (Ex. Closed breakers, switches, etc.)
Post conditions on Success	Expected state of the automation system, substation or its environment after successful completion of the use case (Ex. Tripped breakers, alarms, recordings, etc.)
Post conditions on Failure	Expected state of the automation system, substation or its environment after unsuccessful completion of the use case (Ex. Breaker failure, etc.)

c) **Use Case Description**

This section describes the scenarios of the use case. One instance of this section will be used to define the MSS, and other instances of the section describe Alternative Scenarios (i.e. successful achievement of goals by a different path) and Sub-Scenarios that are <<used>> by the MSS and the alternatives (and by the other sub-scenarios in this set).

Use Case Description				
Use Case Name	See below			
Basic Course Description	Flow of events performed during normal state			
	1	Actor	Event, step or condition of successful execution	
	…	….	….	
	N	System	Event, step or condition of successful execution	
Extensions	Flow of events performed during abnormal states			
	1	Failure & Failure Mode	Actor	Execution step
			….	….
			System	Execution step
	…	…	…	…
	N	Failure & Failure Mode	Actor	Execution step
			….	….
			System	Execution step

In the FICS it is proposed that a convention for Use Case names is defined.

The top level Use Case in the FICS should be called **MSS**. This establishes the main success scenario of the use case.

Alternative success scenarios are named **ASS (- description -)**, for example **ASS (Remote Request).**

Sub-scenarios which describe functionality common to several other scenarios, and which are therefore <<used>> from those scenarios, are named **USS (- description -)**, for example **USS (Check Interlock).**

These are illustrated in the example FICS below.

d) **FICS Example**

Applying these templates to the example of the closure of the circuit breaker yields the following:

Functional Implementation Conformance Statement	
Code	CB Manual Closure
Name	Operator Closure of Circuit Breaker
Description	Operator requests the closure of the circuit breaker which is performed by the SAS after various checks have been performed. The closure can be requested locally or remotely
Customer	Anytown Power Inc.
Substation	Number 1

SCL File	n/a		
Primary Actor	Operator		
Secondary Actor	System		
Stakeholder & Interest			
Function Description			
Trigger	Operator enters local command		
Components or Logical Nodes	IHMI, ITCI, CSWI, CILO, XCBR		
Process Equipments	Circuit Breaker		
Performance	No intentional delay is expected except due to normal component processing		
Preconditions	Circuit Breaker is open		
Post conditions on Success	Circuit breaker is closed		
Post conditions on Failure	Circuit breaker is open		
Use Case Description			
Use Case Name	**MSS**		
	1	Operator	Enters command to manually close CB
	2	IHMI	checks if local commands are enabled
	3	<< USS(Check and Close CB) >>	
	4	IHMI	informs operator of successful closure of CB
Extensions	**2a.** Local commands are not enabled		
	1	IHMI	advises operator that local commands were not enabled
	-	-	exit
Use Case Description			
Use Case Name	**ASS (Remote Operation)**		
	1	Remote Operator	Enters command to manually close CB
	2	ITCI	checks if remote commands are enabled
	3	<< USS(Check and Close CB) >>	
	4	ITCI	informs remote operator of successful closure of CB
Extensions	**2a.** Remote commands are not enabled		
	1	IHMI	advises operator that local commands were not enabled
	-	-	exit
Use Case Description			
Use Case Name	**USS(Check and Close CB)**		
	1	CSWI	checks if closure of CB is permitted
	2	CSWI	starts manual close timer and waits for expiry
	3	CILO	performs interlocking check to validate that CB can be closed
	4	CSWI	initiates CB closure
	5	XCBR	Closes CB and checks that CB has closed successfully
	6	CSWI	Updates state of CB to ‘Closed’
Extensions	**1a. Closure of CB not permitted**		
	.1	CSWI	Updates CB state to ‘CB Closure Not Permitted’
	-	-	Exit
	2a. Request is from ITCI		
	.1	CSWI	Manual closure timer does not need to be run

	-	-	Continue from 3
	3a. Interlock checks prevent closure of CB		
	.1	CSWI	Updates CB state to ‘CB Interlocked’
	-	-	exit
	5a. CB fails to close		
	.1	CSWI	Updates CB state to ‘CB Closure Failed’
	-	-	exit

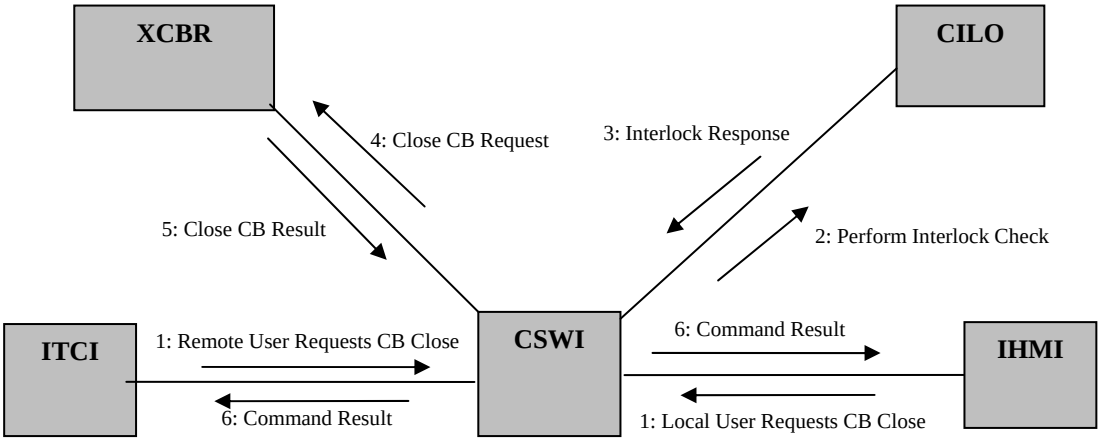
2.4.4 Communications Diagrams

Communication diagrams show how different objects in a system communicate and collaborate in order to achieve the functional goal. Communication diagrams consist of

- Objects (Boxes) representing blocks of functionality within the system;
- Lines between the objects showing pathways of communication;
- Message names that depict the messages flowing along the lines;
- Arrows that show the direction of message flow.

Optionally a communication diagram can also have sequence numbers that show the sequence in which messages are sent. The style of numbering used is open to debate as UML2.0 proposes a nested numbering scheme that shows levels of messages as well as sequence (e.g. 5.1.2.5.1), but this can get very cumbersome if there are many levels and many messages. It is therefore recommended in this brochure that a ‘flat’ numbering scheme is used which is just a single number showing sequence of messaging without indicating levels. Sequence diagrams – described in section 2.4.5 – provide the means for describing sequencing of messages in more detail.

A communication diagram describing the objects and message flows for the CB closure example defined in section 2.4.3.1 is shown below. It can be seen that logical nodes lend themselves to being represented as objects in the communications diagram.



Picture 3 – Example Communication Diagram

In a typical SAS Communication Diagram the objects will be Logical Nodes and the messages between them will be commands, responses, alarms, values and other signals that are defined in the IEC61850 standard.

2.4.5 Sequence Diagrams

Sequence diagrams show an alternative view of collaboration between various objects in the system. A sequence diagram has a fixed format which shows

- sequences of messages;
- elapsed time;
- periods of control within objects.

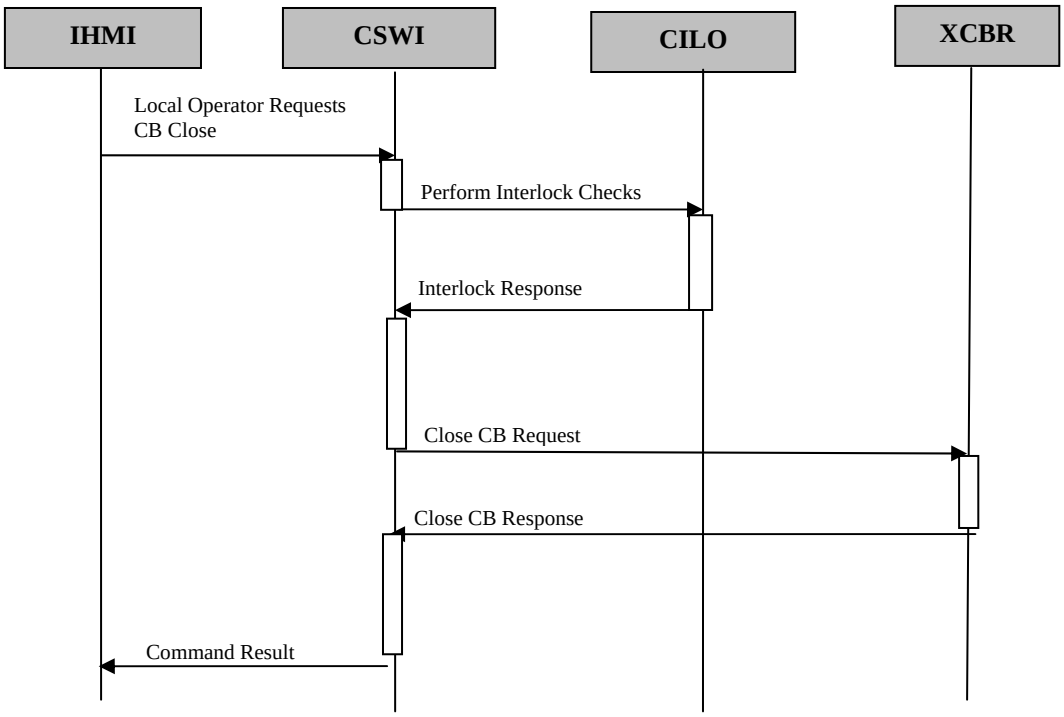
The basic element of a sequence diagram is an object (or Participant), which is represented as a box at the top of the diagram, with a vertical line (Lifeline) descending from the box. The lifeline represents time. Activation bars on the lifeline show periods when that participant is active. A diagram consists of two or more participants with messages shown as horizontal lines between the lifelines. The horizontal message lines have arrows showing the direction of message flow.

The following diagram shows the Circuit Breaker closure example described in section 2.4.3.1 presented as a sequence diagram.

Note that a message flow from one lifeline to another does not necessarily mean that control also moves to the target participant. Participants may send out several messages to other participants before finally relinquishing control to another.

Also note that the example diagram shows the use case of a successful achievement of the goal as described by the MSS in section 2.4.3.1. This is a good relationship between a sequence diagram and a use case where the diagram directly relates to a single use case scenario. Alternatives and Extensions may be shown on the same sequence diagram (using Interaction Frames) but this is not a strong feature of sequence diagrams, and if the alternatives and extensions are complex or multifunctional then the diagrams become overly complicated. Instead it is recommended that alternatives and extensions are each shown on separate sequence diagrams. Thus the use cases described in section 2.4.3.1 will result in several sequence diagrams.

The diagram below illustrates a single sequence diagram for the case where the local operator successfully closes the CB.



Picture 4 – Example Sequence Diagram

2.4.6 Deployment Diagram

Deployment diagrams show how the various components identified in previous diagrams, specifically the logical nodes, are configured on the hardware components within the system. Deployment of logical nodes is outside the scope of IEC61850 but knowledge of it is paramount to successful testing of the system.

A deployment diagram consists of

- **Device Nodes** – identifying the hardware or software devices in the system on which software components are deployed. These are drawn as three-dimensional boxes. Outer level nodes represent the IEDs themselves. Inner level nodes represent logical devices inside the IED.
- **Deployed Artifacts** – identifying the software components (artifacts) present in each device node. These are 2-dimenional boxes containing the name of the artifact. For an IEC61850 based SAS the artifacts will be Logical Nodes.
- **Communication Path** – identifying the means of communication between the device nodes. (Note that in the case of SAS communications for IEC61850 this is assumed to be Ethernet.)
- **Tagged Values** – supporting information in each device node or deployed artifact, such as address or device name, to further specify the deployment environment.

Tagged Values in the IED device nodes should specify the following items:

- Device Node IP Address

- Device Node IED name

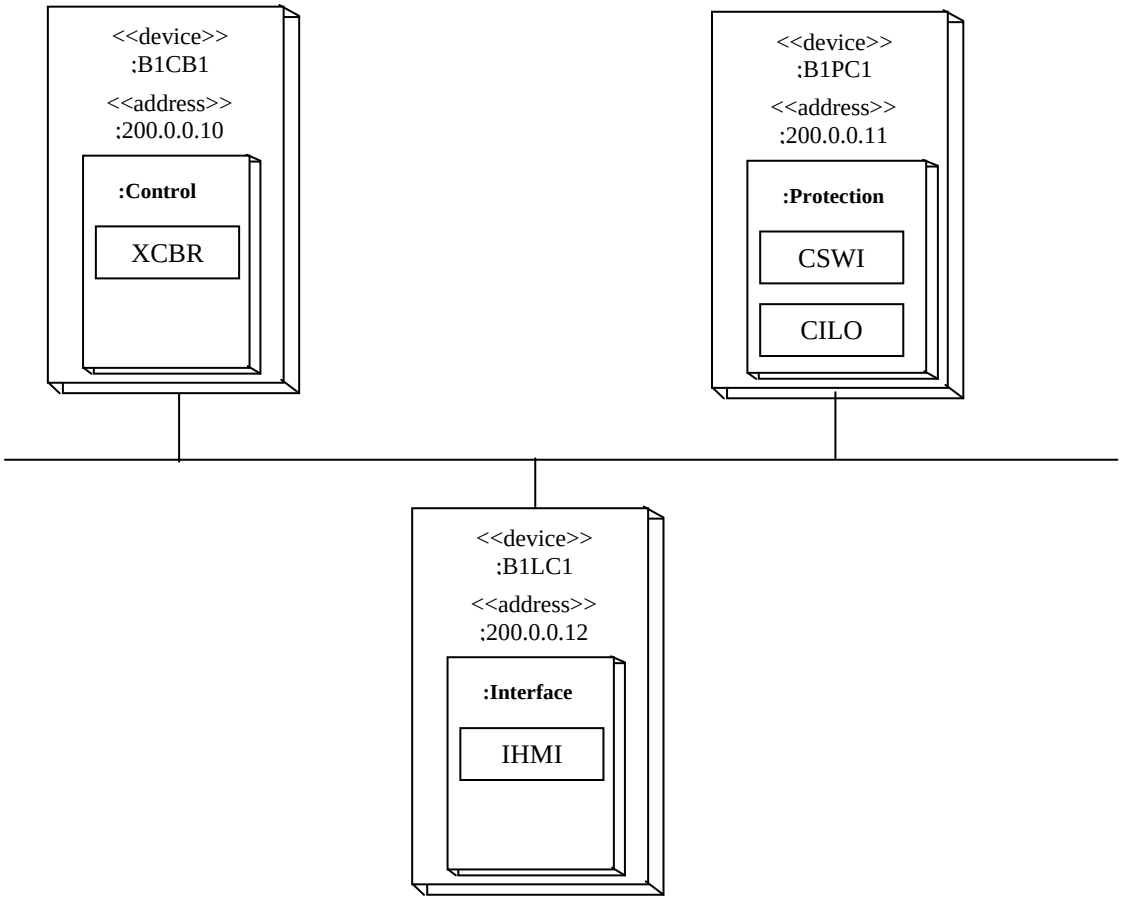
Tagged Values in the LD device nodes should specify the following items:

- Name of the Logical Device

Tagged Values in the Deployed Artifacts should specify the following items:

- Name of the Logical Node

The deployment diagram below illustrates the deployment of logical nodes involved in the local control of the circuit breaker.



Picture 5 – Example Deployment Diagram

2.4.7 Activity Diagrams

Activity diagrams show procedural flow in a system where Actions are performed in response to decisions, as part of loops or sequentially. Activity diagrams can also be used to show parallel flows of activity. An Activity diagrams consist of:

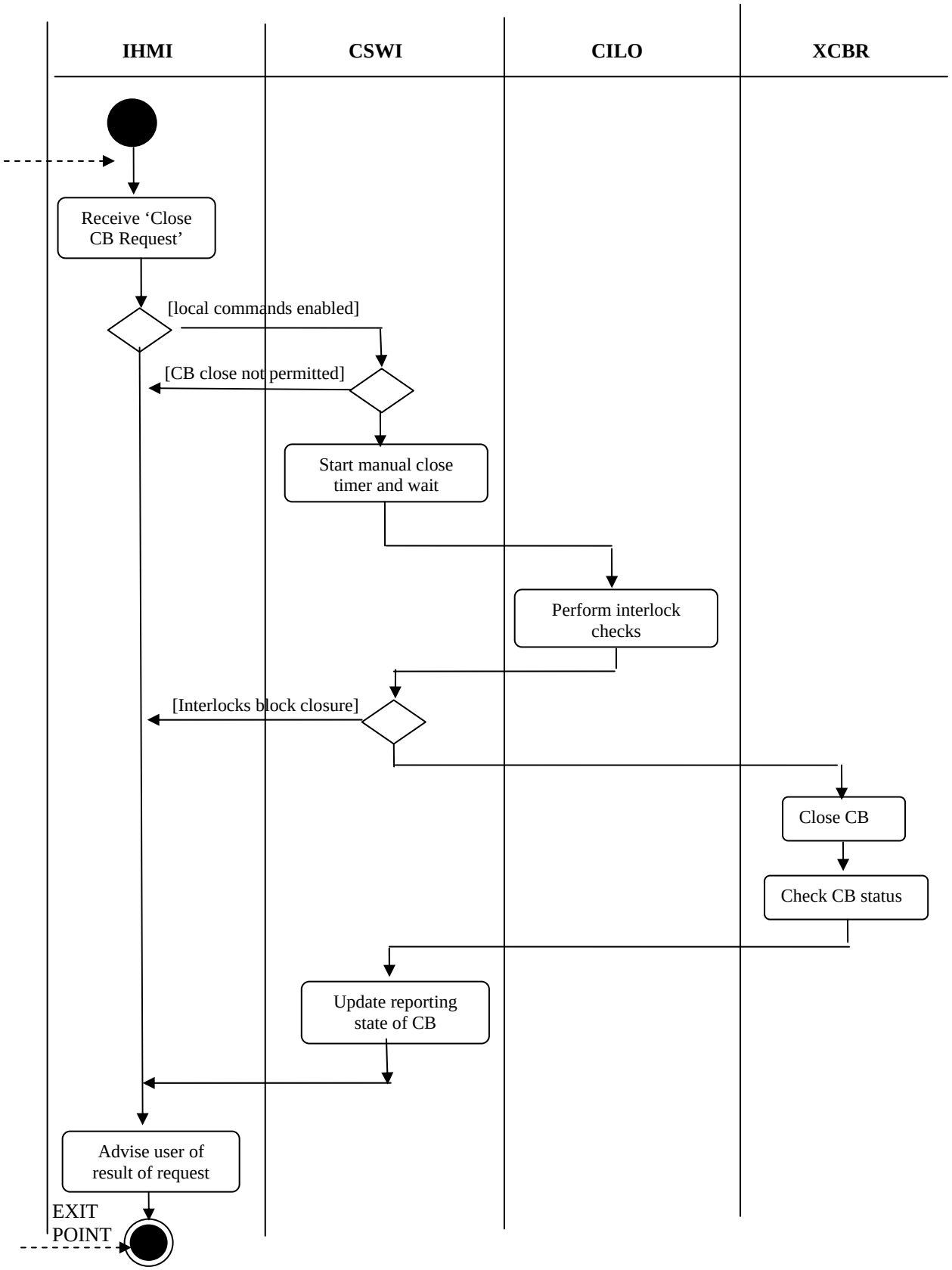
- **Actions** – tasks performed by the function;
- **Flows** - lines linking actions and other features;

- **Decisions** – selecting from alternative flows;
- **Forks** – initiation of 2 or more parallel flows;
- **Joins** – synchronization point for 2 or more parallel flows.

Activity diagrams may show actions that are performed by different parts of a system or different modules of a program, and to aid in the visualization of this, an Activity diagram allows Partitions to be shown where the different actions (and other features) may be located in the different partitions on the diagram to show where they are performed. These partitions, sometimes called Swim Lanes, appear on the Activity diagram as vertical (or horizontal) lines dividing the diagram into Activity Partitions.

In an SAS activity diagram it may be useful to show the partitions (swim lanes) representing the different logical nodes that perform the function being described.

The activity diagram below shows the local CB Closure example. A similar diagram will exist for remote closure.



Picture 6 – Example Activity Diagram

2.4.8 State Diagrams

State diagrams show the different states that a system may occupy while performing the functions being described. They are useful to show behavior of a system in terms of events and resulting actions, especially if the states and actions are modeled in several use cases. A State diagram consists of:

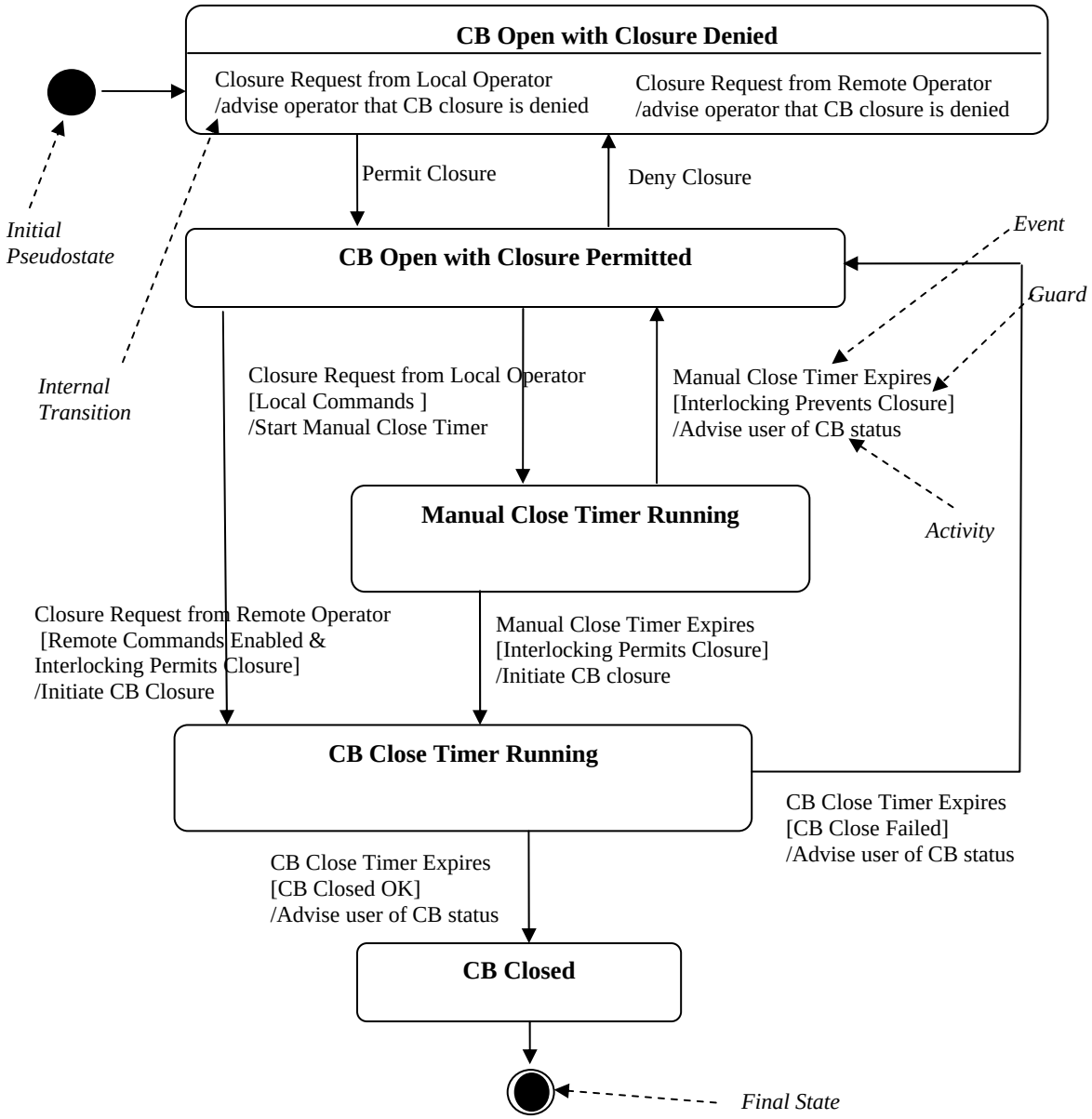
- **States**; these may be
  - Passive - waiting for 1 or more events which must occur before the state can be exited
  - Active - performing some work after which there is an automatic transition to another state (although 1 or more events can also occur to cause the state to exit)

- **Transitions**; each of which consist of
  - Event (or Trigger) – condition that initiates the transition
  - Guard – condition that determines if the transition is acted on
  - Activity – some behavior that is executed as part of the transition

A state is represented diagrammatically as a box and a transition as a line between two boxes (states) supported by a text label which defines the Event, Guard and Activity. An Event is a spontaneous condition, such as external signals, timers expiring and user input, which are processed by the functions being modeled. A Guard is a condition that allows, or denies, the acceptance of the event. For example, an overcurrent event may be ignored if overcurrent protection is disabled. An Activity allows for some actions to be initiated as part of the transition. Typical actions include starting timers, output of data (to a user) or generation of a log entry.

Event, Guard and Activity are all optional, so that a transition may occur without a guard or without an activity, or without both. A transition without an event is possible and is usually used to show exit from an activity state once the activity has completed. Transitions usually cause a change in state but Internal Transitions can occur where an event, protected by a guard, causes an activity without a change in state being made. These are shown on a state diagram by the text describing the Event, Guard and Activity of the transition appearing inside the box representing the state.

The diagram below shows a sample state diagram for the local/remote request of the CB closure.



Picture 7 – Example State Diagram

Strictly speaking a Guard condition on a transition is really stating that there are two states to consider, one where the transition is accepted and one where it isn't. However, state diagrams can become over complicated if too many states are shown, so Guard conditions on transitions allow diagrams to be simplified. But complex Guard conditions can make the diagram too complicated as well. A good state diagram is one which achieves a balance between states and guards. In the above example it would have been perfectly legal to remove the 'CB Open with Closure Denied' state and put additional guard conditions on the transitions from the remaining 'CB Open' state, but this adds complexity to the diagram which could result in poorly understood requirements for testing.

2.5 SAS Performance Requirements

2.5.1 General

In addition to the functional requirements of the intended substation automation system, it is also necessary to specify its performance requirements. The flexibility afforded to the IEC 61850 system designer by the free allocation of functions to devices (both logical and physical) and the use of mainstream communication technologies such as Ethernet is limited by the performance requirements, which cover issues such as response times, availability and reliability, and time synchronization. Together with other constraints (which may include aspects such as environmental conditions, substation topology, redundancy of main protections and so on), the performance requirements impact on the allocation of functions and logical nodes to IEDs and also influence the topology of the communication system and choice of its components. The performance requirements are clearly an input when determining the test specification for an IEC 61850 system. Performance requirements may be classified as functional and non-functional but, as discussed in part 1.1, this document is concerned only with functional performance requirements.

The performance requirements of an IEC 61850 SAS can be specified in a hierarchical structure, with system-level requirements broken down into functional and logical node requirements. In [5] it was observed that different approaches to the specification process could be envisaged depending on a particular customer's level of involvement in specifying the IEC 61850 details. The level at which performance requirements are specified by the customer will equally be dependent on this level of involvement. A customer who requests an IEC 61850 compliant system but has no involvement in the IEC 61850 details will issue performance requirements at the system level only, leaving functional and logical node requirements to be established by the system designer. On the other hand, a customer who desires greater involvement in the IEC 61850 details may well issue a specification which includes performance requirements at the level of functions or logical nodes.

2.5.2 System Performance Requirements

System-level performance requirements are likely to be specified for IEC 61850 substation automation systems in much the same manner as for conventional systems. The requirements tend to be expressed informally in words, and may be quantitative or qualitative or both. The content varies from customer to customer but typically performance parameters such as those listed below are specified under the most onerous system loading conditions:

- Time from operation of a protection element to opening of circuit breaker.
- Time from initiation of a command at the HMI to execution at the process level.
- Time from change of state at the process level to indication at the HMI.

- Time from generation of an alarm or event at the process level to logging at the station level.
- Resolution for time-tagging of alarms and events.
- Time to change between HMI displays.
- Availability.

Some customers may specify performance parameters at different system loading levels, classified for example as Normal, High and Peak loading. The system-level performance requirements are used to determine requirements at the functional and logical node levels.

2.5.3 Functional Performance Requirements

A more formalized approach to specifying performance requirements at the functional level can be realized by the creation of classes of time requirements. This is similar to the concept of message performance classes described in sub-clause 13.6 of IEC 61850-5 which are applied at the PICOM level. Here we wish to identify functional performance time requirements rather than message performance and so it is proposed that functional performance classes are designated using the letter 'F'. The time requirements of all functions can then be identified according to these performance classes, for example:

Table 1 – Classification of Functional Performance

Class	Time	Example of Function
F1	100 ms	Tripping of CB by distance protection operation
F2	500 ms	Closing of CB by synchronized switching
F3	1 s	Closing of CB by operator control
F4	5 s	Disturbance record retrieval
...	...	...

The performance class identified for each function can be entered in the Functional Implementation Conformance Statement (FICS) created for the function.

Note that Annex D of IEC 61850-5 defines the total response time of a function as the sum of the starting time, the internal processing time, the overall transfer time of a PICOM, and the delay time in the related process interface. Annex G gives guidance on typical performance requirements for each function type.

2.5.4 Logical Node Performance Requirements

Functions are decomposed into logical nodes (LNs), and the LNs making up a particular function are identified in the FICS for that function. For example, LNs associated with the distance protection tripping operation in Table 1 would be TCTR, TVTR, PDIS, CSWI, XCBR and IHMI. Performance at this level is dependent both on the execution time of individual LNs or combinations of LNs and on the transmission delay for messages passed between LNs. The user can specify the required time performance for a group of LNs, including their

interconnecting PICOMs, by showing a time constraint in a UML sequence diagram. This method will be described in section 2.6.

The performance attributes of individual PICOMs are described in clause 13 of IEC 61850-5. The performance requirements can be formalized by presenting the PICOM message types and performance classes in tabular form, as shown for example in Table 2. PICOM messages flow from logical source LNs to logical sink LNs. They are classified by message type (types 1 to 7) according to the type of function they fulfill, and by performance class (P1, P2 and P3) depending on the criticality of the application, with P3 being the most onerous class, as described in sub-clause 13.6 of IEC 61850-5. A full list of available PICOM types, including type ID numbers, is given in annex B of IEC 61850-5.

Table 2 could be extended as required to cover other aspects of PICOM performance other than transmission time, such as data integrity and cause of transmission.

Note that time synchronization requirements for the system must also be specified, and these have their own performance classes described in sub-clause 13.7.6 of IEC 61850-5.

Table 2 – Formalized Specification of LN/PICOM Performance

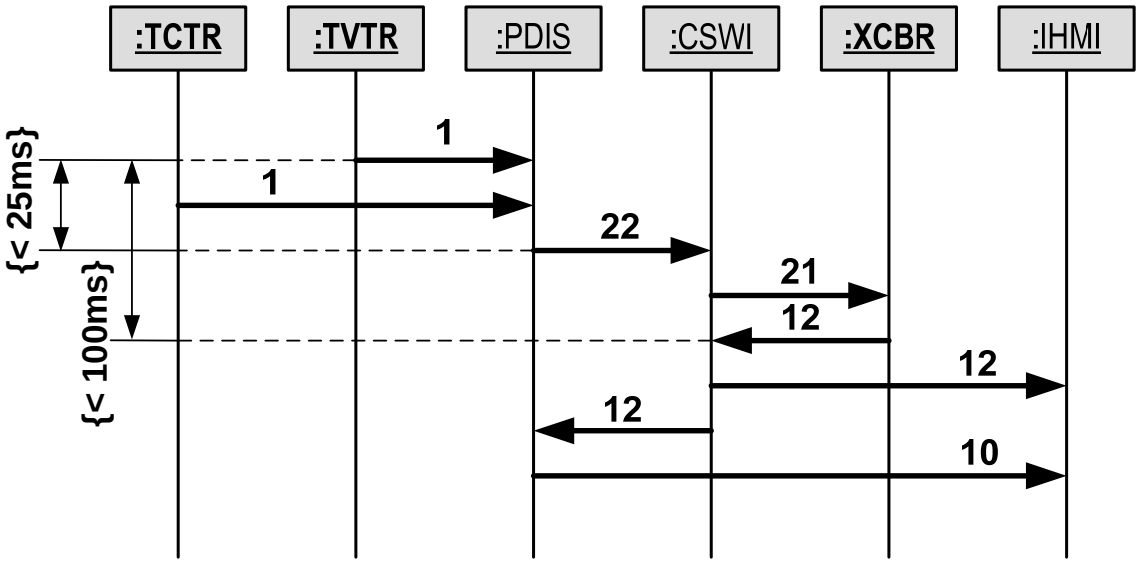
Source LN	Sink LN	PICOM Content	PICOM Type ID	Message Type (Class)
TCTR	PDIS	Process Value (sample)	1	4 (P3)
TVTR	PDIS	Process Value (sample)	1	4 (P3)
PDIS	CSWI	Trip	22	1A (P3)
	IHMI	Event / Alarm	10	3
CSWI	XCBR	Trip	1	1A (P3)
	PDIF	Response	12	1B (P3)
	IHMI	Event / Alarm	10	3
XCBR	CSWI	Response	12	1B (P3)

2.6 UML Performance Requirements

The performance of a function in an IEC 61850 SAS is dependent upon the execution time of the individual LNs and combinations of LNs that make up the function and on the transmission delay for their interconnecting PICOMs. The UML sequence diagram, introduced in 2.4.5, can be extended to include time constraints in order to specify the performance requirements of groups of LNs and their interconnections within a function.

Considering again the example used in 2.5.4 of the tripping of a circuit breaker by distance protection, we can define time constraints as shown in Picture 8. The passage of time flows from the top to the bottom on a UML sequence diagram, and time constraints define time critical portions within the lifetime of the function. Here, separate constraints are shown for two such time critical processes. Firstly, the required operating time of the distance protection is defined, including generation of current and voltage data by the instrument transformer LNs, TCTR

and TVTR, and the transmission of the trip command from the distance protection PDIS to the switch checkcontroller CSWI. Secondly, the total time from initiation of the function to opening of the circuit breaker is specified, illustrating that time constraints may have to take into account the operation of devices external to the SAS, such as the mechanical opening time of the circuit breaker as reported by XCBR, in this example. Indication at the HMI is not considered time critical in this example and so no time constraint is specified for this part of the function.



Picture 8 – Functional Performance Specification by UML Sequence Diagram

Note that timing constraints could be added to other types of UML diagrams, notably communication diagrams, and it might be beneficial to do so for certain types of function. However, it is in the sequence diagrams that timing constraints are largely to be defined.

3. Test Requirements

3.1 Introduction

For the purpose of this report, a System under Test (SUT) is an assembly of IEDs which individually have been conformed to the IEC 61850 standard. That means that each IED has been certified by an independent test authority or supplier. A certified IED is supposed to be able to operate with all other certified IEDs, for the functions which have been tested.

This chapter presents a brief review of the different test requirements which shall be performed over a SUT based on IEC 61850, including:

- Conformance Tests
- Factory Acceptance Tests (FAT)
- Interoperability Tests
- Site Acceptance Tests (SAT)
- Functional Tests
- Performance Tests.

3.2 Conformance tests

One of the first requirements for the functional tests of an integrated SAS is the conformance test of every IED or Device Under Test (DUT) which belongs to the SAS. The conformance tests are performed to prove the adherence of a specific IED to the IEC 61850 standard requirements. A conformance test is the type test for the communication system of the incorporated IEDs. The IEC 61850 standard is focused on the interoperability, using data, function and device models, including all services available at the application level. Consequently the conformance tests shall demonstrate the capability of the DUT to operate with other IEDs from several makers in a specified way, and in accordance with the requirements of the IEC 61850 standard.

Among the goals of conformance tests the following are applicable:

- to reduce the risks of non interoperability to an acceptable level;
- to provide the client with the maximum guarantee that the IED under test will interoperate with other certified devices;
- to provide a type test of the communication interface of the device.

3.3 Factory Acceptance Tests (FAT)

Even for a conventional system, the factory acceptance testing (FAT) is normally a complete test of an integrated system at the vendor facility, with application functions, data base, HMI, displays and logs performed on a specific utility system. Before performing these tests, all the wiring interconnecting the different panels and devices must be completed. During these tests, all specified functions

included in the client specification shall be tested. Any malfunction detected must be repaired.

FAT tests are performed in a per unit basis and are quite similar to the factory acceptance tests performed on conventional digital systems. The main difference is the communication module of the IED under test which must consider the IEC 61850 standard, as well as the communication network. The FAT must include the verification of the vertical messages, from the IEDs to the substation level equipment, as well as the horizontal messages, from one IED to the others. The performance of the specified functions must also be verified during the FAT.

One alternative for the FAT is to perform the system tests at a third party site, known as laboratory acceptance tests (LAT), which may be closer to the client's office or even at the client's offices. This alternative has the advantage of allowing a closer participation of the utility technical staff, leading to a faster process of engineering actualization with the new technology and less problems during the site acceptance tests.

3.4 Interoperability Tests

Differently from the conformance tests, interoperability tests are applied to an assembly of IEDs of the same or different manufacturers. They shall demonstrate that these IEDs, when interconnected by a proper communication system, may operate together, sharing data and other information and performing their functions in a secure way and with acceptable performance.

One of the basic requirements for interoperability tests is that all the involved IEDs have passed the conformance tests, especially with respect to the distributed functions. Distributed functions are those functions which have logical nodes in different physical devices, and exchange horizontal messages using a publisher-subscriber communication mode. Interoperability tests shall also include the verification of vertical messages, using client-server communication mode. These are the messages exchanged between the IEDs at the bay level and the substation level equipment.

In order to perform the interoperability tests it is necessary to have the required means to simulate the communication network that will interconnect the IEDs under test and to simulate the operation of those IEDs which are not available. The test system shall be able to simulate the messages which are exchanged by the IEDs assembly during the operation of the different functions. The worst case scenario can also be considered when performing the tests.

The interoperability tests shall also include the verification of the vertical messages, using a client-server communication mode. These are the messages exchanged between the IEDs at the bay level and the substation level equipment. IED servers must manage one context per client, such as the list of reports, the list of events, the switchgear equipment statuses and commands, the values of the analogic measures like currents, voltages and temperatures, the transfer of files

etc. Therefore, the performance of the system under test will also depend on the number of clients to be considered. The data mapping shall be verified as well, in order to assure that the interpretation of data names is consistent between different suppliers.

3.5 Site Acceptance Tests (SAT)

Site Acceptance Tests (SAT) are those tests performed on the deployment site to confirm the correct functioning of each IED after mounting. The following are requirements to initiate the SAT:

- All primary equipment has been installed, connected and tested;
- The control building and rooms are prepared to receive the integrated SAS;
- The communication LAN equipment are installed, connected and tested;
- All the IEDs and other equipment which are part of the integrated SAS are installed and connected;
- The substation level equipment (HMI, gateways, servers, GPS equipment etc) with the respective data base and operation screens are installed or configured, connected and preliminarily tested;
- All documentation listed above is available at the site;
- Appropriate test equipment, as well as test computer with the required engineering tools are available at the site;

Considering that system tests have been successfully performed during the FAT or LAT period, the SAT will be very much simpler. It shall be done in stages. The utility may need to disable some control or protection features to preclude any undue operation.

3.6 Functional Tests

System functional tests supplement conformance and interoperability tests to guarantee that all functional requirements are met. Because IEC 61850 is a communication standard, it does not standardize the functions of IEDs. Therefore, the customer has to perform functional tests himself or delegate the responsibility to a third party.

While conformance and interoperability tests are applied to a specific IED, system functional tests are applied to an IED assembly, including the communication network, and are intended to verify their joint correct operation. For functional testing of IEC 61850 based systems, it is necessary to describe which types of functions they must perform, derived from the user specification.

Functional tests shall comply, as close as possible, with some characteristics. They have to be:

- Simple – adequate to user knowledge;
- Repeatable – same output for same input sequence;

- Documented – structured and formalized;
- Automatable – computer executable;
- Human readable – textual or graphical form;
- Customer oriented – focus on user requirements;
- Tool independent – common to all testers;
- Supplier independent – common to all IEDs;
- Extensible – easily edited;
- Systematic – structured methods;
- Standardized – based on accepted standards.

In addition to these requirements, the system under test (SUT) must comply with the customer specification requirements which may include a mixing of devices of different suppliers. Since there are virtually unlimited combinations of possible configurations, it is unlikely that all test results will be provided. Therefore, an adequate level of test coverage must be established in order to define the number of test cases required. Detailed information on test coverage methods will be given in the next chapter.

Functional tests must consider SAS with substation bus only, and systems with substation bus and process bus. Depending on the complexity of the SAS, the tests may be performed step-by-step, as follows:

- Functional testing of the individual components of the SUT;
- Functional testing of bay level distributed applications;
- Functional testing of substation level distributed application;
- Functional testing of process bus functionalities.

Before functional tests are initiated, the following documentation must be available:

- System specification, including the single line diagram;
- List of functions and functional behavior;
- Performance requirements of each function;
- Specific and catalog information of all IEDs and other devices to be tested;
- Data model in accordance with the IEC 61850 or signal list;
- Communication system architecture and components;
- Requirements for migration scenarios;
- Definition of responsibility in multi-vendors systems;
- SCD file of the SAS system;
- CID file of each IED which makes up the SUT;
- PICS, MICS and PIXIT files of each IED which makes up the SUT;
- Version control documentation as per IEC 61850 standard;
- Set of engineering tools required;
- Detailed test plan for the FAT and SAT tests.
- FICS – Function Implementation Conformance Statement;

The FICS format, described on Chapter 2, is proposed to standardize the functional specification of Substation Automation Systems.

3.7 Performance Tests

Conformance tests do not verify the correct functioning of an IED assembly, including the communication system. Also, they do not guarantee that the performance will be adequate in time response and accuracy of the IEDs assembly, especially when considering distributed functions. So, conformance tests must be supplemented by functional and performances testing.

The performance of a distributed function is based on GOOSE messages and on the publisher-subscriber communication mode. Therefore, it will depend on the number of GOOSE messages in transit on the network and on their frequency of occurrancy. The tests shall not only check their interoperability and time performance, but their endurance in nominal and degraded situations.

Desired performance of a system may be expressed and measured by its time response and measurement accuracy. These should be specified and tested at different operational conditions, like voltage collapse, communication failures and network loading. Normal and worst case conditions should be verified against the desired behavior.

Performance tests aim to verify that a designed system can attain the specified requisites related to time and accuracy. Like functional tests, they aim at the assembled system, being done usually at the user installations by systems integrators. When possible, they could be done at the factory or Lab, since if the performance of the SUT is not adequate, it may be necessary to make deep changes in the project. Their requirements and characteristics are very similar to performance tests, being usually done at the same time.

As the number of possible functional and performance tests of a system is unlimited, acceptable test coverage criteria must be defined, as proposed in the next chapter.

4. Test Coverage

4.1 Introduction

In any test plan, it is important to show the level of test coverage of the plan. Test coverage is a measure of completeness of a test plan, expressed by the set of possible failures checked by a given test set. FMEA (Failure Mode and Effects Analysis) [22] and HAZOP (Hazard and Operability Analysis) [21] are two methods standardized by IEC to analyze the possible failures that a system may present. In this chapter, these two methods are suggested as possible ways to asses the test coverage of a SAS.

FMEA is a structured method to correlate all possible failure modes of a system to their functional failures and effects. It can be used as a design tool, to decide on project options, or as a maintenance tool, to help plan maintenance policies. In this chapter, FMEA will be used as a tool to evaluate the failure modes and functional failures detected by a test case or test plan. The aim is to decide on how much testing is necessary to prove the effectiveness of the plan, measured by the set of failure modes it can detect.

HAZOP is also a structured method to identify all possible failure modes of a given system or installation. It will be used with FMEA to evaluate the fault coverage of the test cases and plans of SAS.

In the remainder of this chapter, a sequential method to determine test coverage is proposed. Starting from the functions as described on Chapter 2, all functional failures of a given SAS will be listed. Then all physical or logical components that take part on each function will be identified, as well as their failure modes. FMEA tables and HAZOP matrices will correlate them, in the test design stage, to evaluate their test coverage.

4.2 SAS Functional Failures

Functional failures represent abnormal states or results from system functions. They can vary from a complete loss of the function, to a partial degradation of its expected performance level. Usually, functional failures may be associated with a function, without correlation to the component or failure mode that causes it. This distinction is important, for the correct application of FMEA, as the same function can be the result of many alternate failure modes of several components.

In an SAS, typical functional failures can be associated to the following conditions:

- Absence of output signals/messages
- Wrong output signals/messages
- Wrong destination of output signals/messages
- Wrong timing (delay or pre-emption) of output signals/messages

- Wrong sequence of output signals/messages.

Usually, the same function can fail in several ways; each way is considered a different kind of functional failure. In a protection function, for instance, absence of tripping during a fault, or the trip of the wrong breaker, or even an excessive delay in tripping the right breaker should be considered distinct functional failures of the same protection function. Similar failures can be listed for command and control functions, supervision, recording, teleprotection, etc.

The correct identification of all possible functional failures is important for the design of test cases and their test coverage. A complete functional coverage means the test is able to discover the presence of all possible failures of each function. When possible, they should be identified independently from the associated components. The result will be a list of all possible failures of each SAS function. That is, there are no failures dissociated from a system function.

A standardized format should be followed for the description of each failure. Usually, a code and a short title are sufficient to completely define a functional failure. A noun, derived from the main verb that describes the function, supplemented by any verbs and adverbs are adequate to qualify the fault, such as unattended performance limits. The following examples show possible ways to describe failures, with the corresponding codes:

- *FNT1: Absence of trip for an internal transformer fault;*
- *FNT2: Unnecessary trip of a remote breaker during a bus fault.*

An adequate coding scheme should be used for failures, so that each failure can be uniquely identified and distinguished from others. The same coding should be followed consistently for the whole SAS. Preferably the coding should associate the failure with the function, such as FNT1.FLR2, to denote failure FLR2 of function FNT1, and so on.

4.3 SAS Components

For FMEA and HAZOP analysis, components are defined as any part of a system where failure modes can originate. Physically, SAS systems are composed mainly by IED (Intelligent Electronic Devices) and network components. IEDs based on IEC 61850 are modeled as sets of Logical Nodes, while network components are typically structured as switches, hubs, etc.

Physical components like IEDs and switches are typically programmed electronic devices (PED), based on digital technologies. Abstract components like logical nodes are usually software modules or blocks of code that externally act as a component. Conceptually they can be encapsulated as a single component or firmware, distributed on several servers or grouped in a unique server. Servers are also modeled as logical nodes by IEC 61850.

4.4 Physical and Logical Node Failure Modes

These failure modes represent abnormal events that may occur on components. Like functional failures, they can vary from a complete stop of component behavior, to a partial degradation of its expected performance. Usually, failure modes should be related to specific components, without correlation to system functions or functional failures. This distinction is important for the correct application of FMEA, as the same failure mode can affect several functions, causing many concurrent functional failures.

Failure modes of physical components are usually related to environmental or physical causes, like short-circuits, broken or worn-out pieces, etc. Software components and logical nodes may present novel failure modes, caused by:

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

For testing purposes, software modules like logical nodes are treated as black boxes, so that their failure modes are limited to loss or degradation of an expected external behavior of the component.

4.5 Hazard and Operability Studies

To identify all possible failure modes of a given component, it is proposed to use HAZOP as defined by IEC 61882 Hazard and Operability Studies (HAZOP Studies) - Application Guide. According to this standard, failure of any component can be associated to a Guide Word, following the meaning listed on Table 3. The table also shows examples of possible logical node (LN) failure modes related to each guide word.

Table 3 – IEC 61882 HAZOP Guide Words

Guide Word	Interpretation
No	Complete negation of the design intention.
More	Quantitative increase of the design intention.
Less	Quantitative decrease of the design intention.
As well as	All design intention is achieved with additions.
Part of	Only some of the design intention is achieved.
Reverse	The logical opposite of the design intention is achieved.
Other than	Complete substitution of the original design intention.
Early	Something happens earlier than expected relative to clock time.
Late	Something happens later than expected relative to clock time.

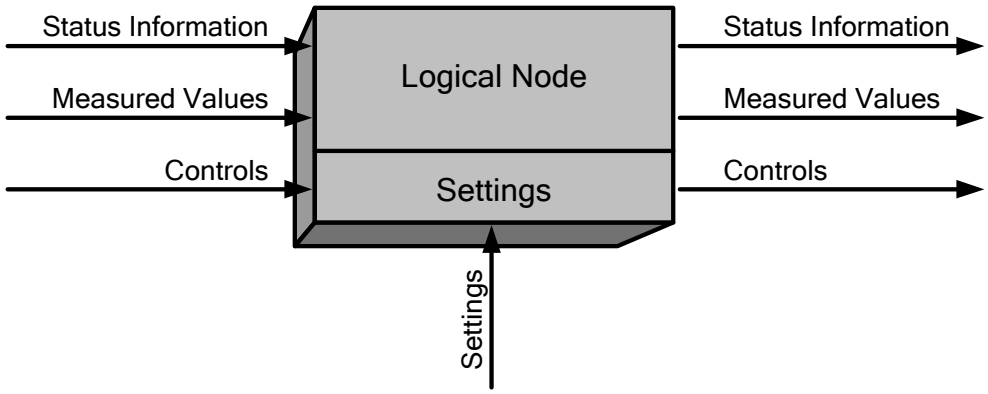
Before	Something happens before it is expected.
After	Something happens after it is expected.

For Programmable Electronic Systems (PES), the HAZOP standard suggests specific meanings to each Guide Word, according to Table 4.

Table 4 – IEC 61882 HAZOP Guide Words for PES

Guide Word	Interpretation for Programmable Electronic System (*)
No	No data or control signal passed
More	Data is passed at a higher rate than intended
Less	Data is passed at a lower rate than intended
As well as	Some additional or spurious signal is present
Part of	The data or control signals are incomplete
Reverse	Normally not relevant
Other than	The data or control signals are incorrect
Early	The signals arrive too early with reference to clock time
Late	The signals arrive too late with reference to clock time
Before	The signals arrive earlier than intended within a sequence
After	The signals arrive later than intended within a sequence

Logical nodes, as defined by IEC 61850, have all the properties of PES, so it is suggested that these guide words can be applied to them as well. To explore the way logical nodes can fail, consider the general input output model shown on Picture 9, taken from IEC 61850 standard.



Picture 9 – Simplified Model of Logical Node

According to this model, failure modes of a logical node will manifest as loss of their ability to respond correctly to status, measured, control or setting signals. These possibilities can be seen on the following Table 5.

Table 5 – IEC 61882 HAZOP Guide Words for Logical Nodes

Guide Word	Status	Measures	Controls	Settings
No	No status	No measurement	No control	No setting
More		Measure > expected		Setting > expected
Less		Measure < expected		Setting < expected
As well as			Wrong control	
Part of	Not all status	Not all measures	Not all controls	Not all settings
Reverse	Inverted status	Inverted measure	Inverted control	Inverted setting
Other than			Unknown control	Unknown setting
Early				Too few timing setting
Late	Status delay	Measuring delay	Control delay	Excess timing setting
Before		Sample out of order		
After		Sample out of order		

All these methods can be used in a Failure Mode and Effects Analysis, to determine the functions affected by any failure mode.

4.6 Failure Mode and Effects Analysis

FMEA is a formal method to analyze the effects of all component failure modes on the functions of a system. FMECA (Failure Mode, Effects and Criticality Analysis) is a variant of FMEA where a criticality level is assigned to each failure mode, according to its consequent impact on system functions. Traditionally, FMEA is documented in a table format, like Table 6.

Table 6 – Failure Mode and Effects Analysis Table Format

Function	Functional Failure	Failure Mode	Effect
FNT1			
…	**…**	**…**	**…**
FNTn			

When the effects are known or similar, like not performing a given function or not passing a given test, a somewhat simplified table can be used, like the model of Table 7, where the failure modes (FM1, …, FMn) are related to the functional failures (FLR1, …, FLRn) they impact, on a given system. All failure modes of this table can be listed using the HAZOP or other method.

Table 7 – Simplified FMEA Format

FMEA		FAILURE MODE								
		FM1	**FM2**	**FM3**	**FM4**	**FM5**	**…**	**…**	**…**	**FMn**
FAILURE	**FLR1**		**X**			**X**	**…**	**…**	**…**	
	FLR2	**X**		**X**	**X**		**…**	**…**	**…**	**X**

	…						…	…	…	
	FLRn		X	X		X	…	…	…	X

4.7 Test Coverage

This FMEA format is easily extended to evaluate failure coverage of a given test plan, by adding the functional test cases affected by each failure mode. Table 8 shows the test cases planned to a given system, added as new lines on the FMEA table. A line is also added to indicate test coverage.

Table 8 – FMEA Test Coverage

FMEA		FAILURE MODE								
		FM1	FM2	FM3	FM4	FM5	…	…	…	FMn
FUNCIONAL FAILURE	FLR1		X			X	…	…	…	
	FLR2	X		X	X		…	…	…	X
	…	…	…	…	…	…	…	…	…	…
	FLRn		X	X		X	…	…	…	X
COVERAGE		X	X	X	X	X	…	…	…	X
TEST CASE	T1	X					…	…	…	X
	…			X		X	…	…	…	X
	Tn		X				…	…	…	X

For each test case line, an X is put on the column corresponding to every failure mode tested. An X is also put on the crossing of this column with the coverage line. The number of Xs in the coverage line is an evaluation of the number of failure modes covered by the test plan. In this way, the test designer can identify which failure mode is tested, and mainly, which ones are not tested.

Test coverage can also be shown on a table relating the tested components (C1, …, Cn) to the HAZOP guide words that describes their failure modes, as shown on Table 9.

Table 9 – HAZOP Test Coverage

	Component								
Guide Word	C1	C2	C3	…	…	…	…	…	Cn
No	OK	X	OK	…	…	…	…	…	OK
More				…	…	…	…	…	
Less			X	…	…	…	…	…	
As well as	X			…	…	…	…	…	
Part of		X		…	…	…	…	…	X

Reverse				…	…	…	…	…	
Other than	**OK**	**X**	**OK**	…	…	…	…	…	**OK**
Early				…	…	…	…	…	
Late	**OK**	**OK**	**X**	…	…	…	…	…	
Before				…	…	…	…	…	**X**
After			**OK**	…	…	…	…	…	

In this table, the word “OK” in a cell means that the failure mode of the component column related to the guide word row is tested by at least one test case. An “X” in a cell means that the failure mode exists but is not tested by any test case. Otherwise, a blank cell means there is no component failure mode related to that guide word.

5. Functional Test Tools

5.1 Introduction

The development and application of complex IEC 61850 based substation or power plant protection and automation systems requires the development of tools for their testing that will ensure the correct operation of protection, control, monitoring, recording and metering functions under normal and abnormal system conditions.

This chapter discusses the requirements for the tools needed for testing of complex substation automation systems. The testing tools need to correspond to the functional hierarchy of the substation automation system. The tools should support three levels of testing:

- Functional element testing
- Integration testing
- System testing

The tools will also have some differences depending of the level of implementation of the standard. Two typical types of IEC 61850 based substation automation systems are considered:

- A Hybrid system - with Substation Bus (IEC 61850-8-1) only
- A Complete system - with Process Bus (IEC 61850-9-2) and Substation Bus (IEC 61850-8-1)

Tools for testing of both types of systems are proposed based on the following system components tests:

- Testing of IEC 61850 protocol compliance of the individual components of the system
- Testing of Merging Units
- Testing of IEC 61850 compliant IEDs
- Testing of bay level distributed applications
- Testing of substation level distributed applications

Requirements for tools that support the configuration, monitoring and reporting of the testing of complex systems are discussed. Solutions for the testing of the individual components of the IEC 61850 based system, as well as for the end-to-end testing of distributed applications are also described.

5.2 System Testing Tools Requirements

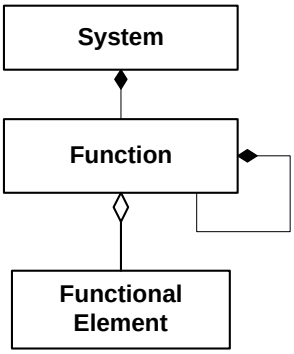
IEC 61850 defines a system as “The logical system is a union of all communicating application-functions performing some overall task like

“management of a substation”, via logical nodes. The physical system is composed of all devices hosting these functions and the interconnecting physical communication network. The boundary of a system is given by its logical or physical interfaces. Within the scope of the IEC 61850 series, ‘system’ always refers to the Substation Automation System (SAS), unless otherwise noted” [1].

The above definition is in the core of the testing tools requirement definitions. The definition of system boundary and the distinction between logical and physical interfaces play an important role in the development of the functional testing tools. Each component of the IEC 61850 system interacts or is related to at least one other element. Any object which has no relationship with any other element of the system is obviously not a component of that system.

Depending on the complexity of the system, its components can be simple functional elements, subsystems or combination of the two. A subsystem is then defined as a set of elements, which is a system itself, and also a part of the whole system. Each of these has to be defined in a way that meets the requirement for testability. This is a characteristic which allows the status (operable, inoperable, or degraded) of a system or any of its sub-system or elements to be confidently determined in a timely fashion. Testability attempts to qualify those attributes of the system which facilitate detection and isolation of faults that affect system performance.

In the substation protection and automation domain we can consider different functions performed by the system as subsystems. The hierarchy of a complex system is shown in Picture 10 as a UML diagram.



Picture 10 – System Hierarchy UML Class Diagram

From Picture 10 it can be seen that the system can contain 1 to many functions that can have several layers of 1 to many sub functions and at the bottom – a sub function can contain 1 to many functional elements. The functional elements correspond to the IEC 61850 logical nodes.

Functional system testing tools are required for testing conducted on a complete, integrated substation automation system, subsystem or distributed function. The goal of the test system tools is to help evaluate the IEC 61850 system's compliance with its specified requirements.

System testing tools should support the principles of Black Box Testing. This means that the test system does not have to have any knowledge of the internal logic and the behavior of the different subsystems or functional elements included in it.

The system testing tools should allow tests to be performed in a top-down or bottom-up approach. This is to a great extent dependent on the purpose of the test. If the test is a factory acceptance test it might be a good idea to use the bottom-up approach. In this case the testing starts first with the individual parts of the system – the functional elements. They are then grouped together to form sub functions or functions, which are in turn linked into more complex functions until the complete system is tested.

When we do commissioning or maintenance testing we assume that the individual functional elements are operating properly, especially if there are no alarms in any of the IEDs that are included in the system test. In this case a top-down approach is suitable, since we are interested in the overall performance of the tested system function and not in the behavior of the components of the system. This fits the Black Box approach, which means that we take an external perspective of the test object to derive the test cases and analyze the results.

Tools for performing functional testing of any function or sub function requires the ability from the test designer to select a set of valid or invalid inputs and determine the correct expected output for each test condition defined in the test plan. This will serve to define the evaluation criteria to determine if the test result is PASS or FAIL.

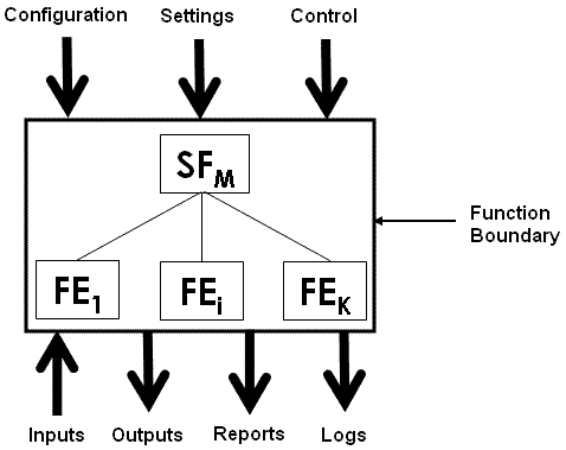
Since the purpose of functional element testing is to determine if the tested element has the expected behavior under different realistic test conditions, the testing tools should allow the simulation of such conditions and at the same monitor the behavior of the tested element.

One of the key reasons for integration testing is to detect any potential interoperability problems between the functional elements and/or sub functions that are integrated together in a function or a system. So the functional testing tools should not only test the performance of the system, but also observes the exchanges between the different components of a distributed function being integrated into a system.

The system testing tools should look at the overall performance of the system from an external observer point of view. It should support the top-down testing model in which the system is defined as a whole with its boundaries and behavior, without considering the details for any part of it. Each sub-part of the system then can be tested using the same approach until we get to the bottom of the functional hierarchy where we perform the functional elements testing.

At the same time it should support the bottom-up testing - starting with the functional elements testing and then going up the functional hierarchy by testing sub-functions until we finish with the overall system testing.

In all cases it is important for the testing tool to be able to clearly identify the system or function boundary that will define the requirements for simulation by the test system and monitoring the behavior of the tested function or component.



Picture 11 – Function Boundary Definition

In Picture 11 above SF indicates a sub function that contains K functional elements. The functional elements are the smallest component in the system that can be defined with a function boundary, interface and behavior, i.e. that can be tested.

The testing tools should also include components that will allow the testing of the different types of systems as described below.

5.3 IEC 61850 System Types

The different requirements for functional testing of the two typical types of IEC 61850 based substation automation systems can be defined based on the interface with the primary substation equipment: It has to be well understood, due to the fact that it determines functional boundaries of many different types.

5.3.1 Systems with Partial Implementation of IEC 61850

In systems with partial implementation of IEC 61850 the interface with the process is identical to the conventional substations, i.e. hardwired connections between:

- secondary side of current and voltage instrument transformers and the analog inputs of the IEDs
- auxiliary contacts of the breakers and the IED optical inputs
- IED binary outputs and the process control (for example breaker trip coils or transformer tap changers)

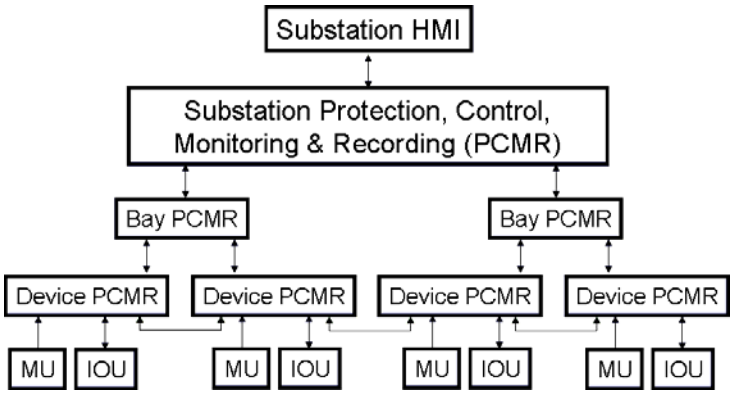
The interface between the devices in the substation is based on communications messages exchange over the substation local area network.

5.3.2 Systems with Full Implementation of IEC 61850

A full implementation of IEC 61850 in a substation protection and automation system indicates the use of both Substation Bus (IEC 61850-8-1) and Process Bus (IEC 61850-9-2).

The interface between all devices in the system in this case is based on communications, with the use of copper cables being limited to:

- DC or AC power
- secondary of the instrument transformers and the merging units
- breaker auxiliary contacts and trip coils and the secondary devices in the substation.

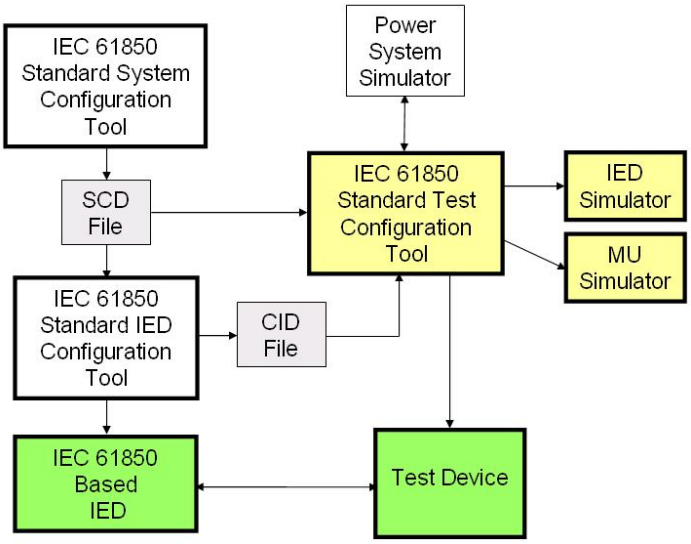


Picture 12 – Full Implementation Architecture

When we analyze the system in Picture 12, it becomes clear that the requirements for testing tools will change significantly depending on where we draw the system boundary. It will also be affected by the use of conventional or non-conventional sensors.

5.4 IEC 61850 Test System Components

A test system designed for IDs (Intelligent Devices), distributed applications or systems based on IEC 61850 have multiple components that are needed for the testing of the individual functions, as well as a complete application. A simplified block diagram of such a system is shown in Picture 13.



Picture 13 – System/Configuration Tool, Simplified Block Diagram

The first component of the test system is the test Configuration Tool. It takes advantage of one of the key components of the IEC 61850 standard – the Substation Configuration Language. The Configuration Tool is used to create the files required for configuration of different components of the test system. It imports or exports different configuration files defined by Part 6 of IEC 61850.

The test system Configuration Tool reads the information regarding all IEDs, communication configuration and substation description sections. This information is in a file with .SCD extension (for Substation Configuration Description) and is used to configure the set of tests to be performed.

The overall functionality of any IEC 61850 compliant device is available in a file that describes its capabilities. This file has an extension .ICD for IED Capability Description.

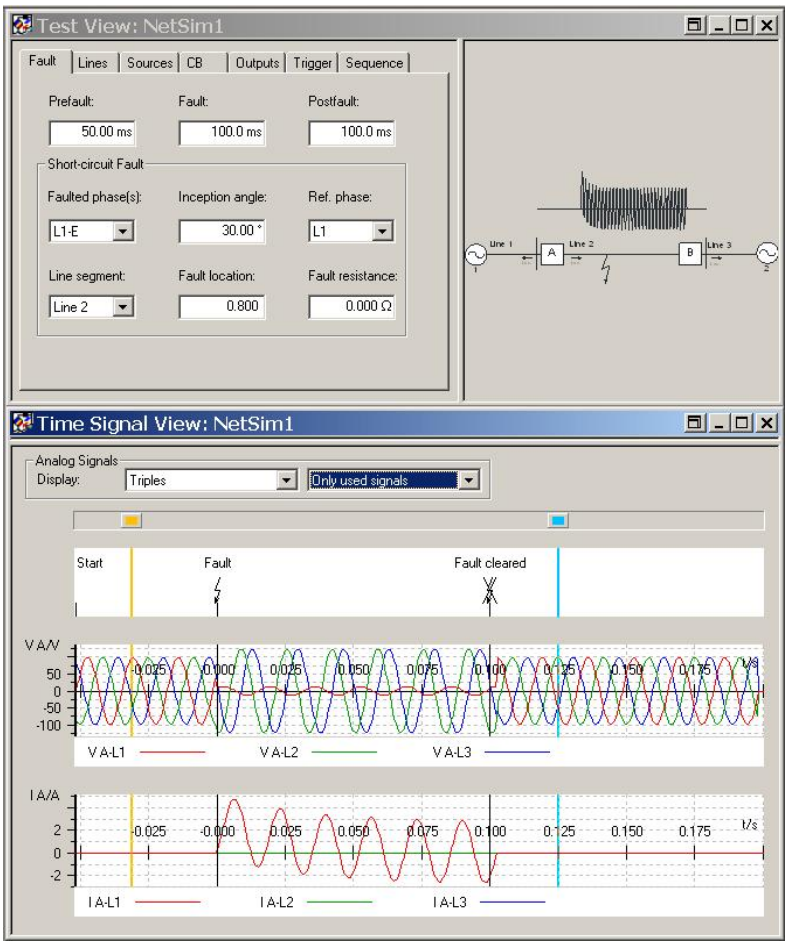
The IED configuration tool sends to the IED information on its instantiation within a substation automation system (SAS) project. The communication section of the file contains the current address of the IED. The substation section related to this IED may be present and then shall have name values assigned according to the project specific names. This file has an extension .CID (for Configured IED Description).

The second component of such a system is a Simulation Tool that generates the current and voltage waveforms. The specifics of each simulated test condition are determined by the complete, as well as the configured functionality of the tested device or application.

The simulation tool requirements will also be different depending on the type of function being tested. For example, if the tested function is based on RMS values or phasor measurements, the simulation tool may include a sequence of steps with the analog values in each of the steps defined as phasors with their

magnitude and phase angle. Based on these configuration parameters the simulation tool will generate the sine waveforms to be applied as analog signals or in a digital format to the tested components or systems.

If the tested functions are designed to detect transient conditions or operate based on sub-cycle set of samples from the waveform, an electromagnetic transient simulation will be more appropriate.



Picture 14 – Network Simulator Interface

Picture 14 shows the simulation tool interface that allows the user to configure the specifics of the network model, type of fault, fault location, etc. that are then used to calculate the waveforms to be applied to a device or system under test.

The third part of the test system is the Virtual Merging Unit simulator. While under conventional testing the waveforms generated by the simulation tool will be applied to the tested device as current and voltage analog signals, a Virtual Merging Unit will send sampled measured values as defined in IEC 61850 over the Ethernet network used for the testing.

The Virtual Merging Unit simulator should support multiple sampling rates and allow the user to select a protection, power quality, or recording mode. As agreed

in IEC 61850 9-2 LE, in the first case the simulator should send 80 samples/cycle in 80 messages/cycle. Each message contains one sample of the three phase currents and voltages (WYE class). In the second mode, 256 samples/cycle are being sent in groups of 8 samples in a single message, thus requiring 32 messages/cycle.

The fourth component of the test system is the Virtual IED simulator that is used to represent components of the system that are not available at the time of testing, for example during factory acceptance testing. During the testing this module send GOOSE messages that the function or Sub function under test uses as inputs that determine its behavior under the test conditions applied.

The fifth component of the test system is the Test Evaluation Tool that includes the monitoring functions used to evaluate the performance of the tested elements within a distributed sampled analog value based system. Such evaluation tool requires multiple evaluation sub-modules that are targeted towards the specifics of the function being tested. They might be based on monitoring the sampled measured values from a tested merging unit, GOOSE messages from a tested IED, as well as reports or waveform records from the tested device.

The sixth component of the test system is the Reporting Tool that will generate the test reports based on a user defined format and the outputs from the simulation and evaluation tools.

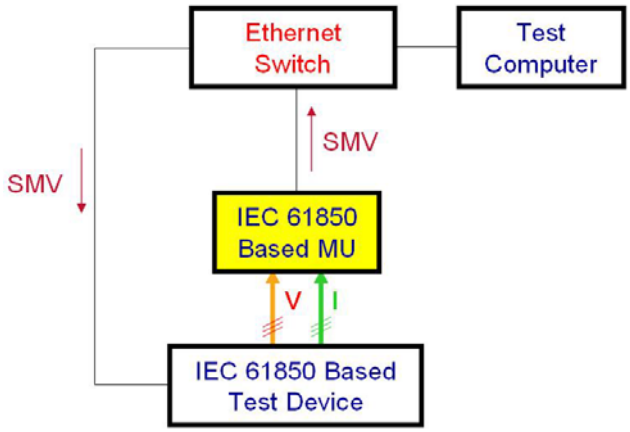
5.5 Tools for Functional Testing of IEC 61850-9-2 Based Merging Units

Since Merging Units are an essential component of any IEC 61850 process bus based application, they have to be tested to ensure that they provide the required sampled measured values.

The currents and voltages applied to the Merging Unit will be based on current and voltage waveforms produced from the network simulator in order to simulate different system conditions, such as high current faults or low current minimum load conditions.

At the same time the Test Evaluation tool will need to receive the sampled analog values from the tested merging unit and compare the individual sampled values from the Merging Unit with the samples coming from the network simulator. The testing of Merging Units will require first of all a very accurate time synchronization of both the test device and the tested MU.

It is necessary to analyze the phase (time) and magnitude differences of the individual samples and compare these to the calibration specifications of the MU. Proper documentation and reporting is required in the same manner as meter testing is performed today.



Picture 15 – Testing of Merging Units

Keeping in mind that the standard allows different sampling rates, as step one the Merging Unit test module shall support the sampling rates defined in IEC 61850 9 – 2 LE. This means that depending on what is the mode of the Merging Unit being tested, the evaluation tool will receive different messages:

- For protection applications – 80 samples/cycle in 80 messages/cycle
- For power quality and recording applications – 256 samples/cycle in 32 messages/cycle (8 samples per message)

Things are more complicated with a generic implementation of the IEC 61850 9-2 process bus, when the sampling rate can have any value. This will require appropriate configuration tools and support by the Merging Units simulator.

5.6 Tools for Functional Testing of IEC 61850-9-2 Based IEDs

The testing of different functions in IEDs that are based on sampled measured values can be achieved in a couple of different ways depending on the requirements of the specific test. One approach is acceptable when testing the IED only, while another can be used if the testing includes the complete MU/IED system. The difference is that in the first case there is no hard wiring between the test device and the tested IED – i.e. the test system can be communications based only.

The key component of this module is the Merging Unit simulator described earlier in the paper. It will have to take the waveforms generated from the Network Simulator and then format them in the required 80 samples/cycle and multicast the individual sampled values to the LAN 80 times per cycle (e.g. 80 messages/cycle).

For power quality and recording applications the Merging Unit simulator will have to take the waveforms generated from the Network Simulator and then format them in the required 256 samples/cycle and multicast the individual sampled values to the LAN 32 times per cycle (8 samples per message).

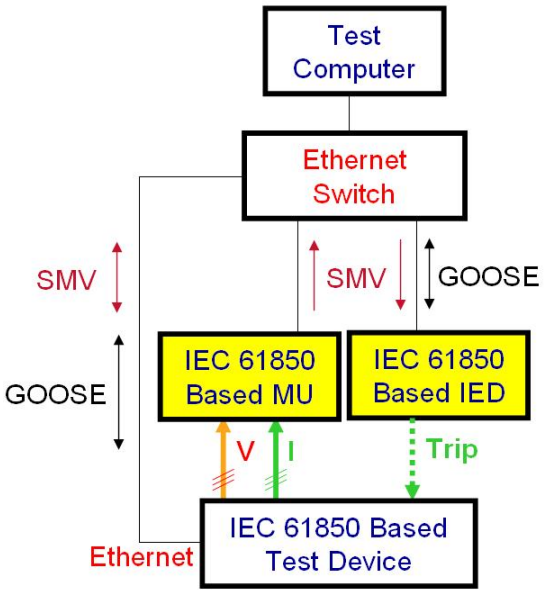
The functions that can be tested in an IEC 61850 9-2 based IED are:

- Protection
- Measurements
- Recording
- Fault location

The testing of these different types of functions available in the IED will be similar to what was described earlier for the hybrid device. This applies to both the configuration and analysis modules of the test system.

The test system needs to subscribe to and monitor the GOOSE messages received from the tested IED that represent the operation of the tested functional elements in order to determine if the devices operated as required. If the tested device has relay outputs as well, they will have to be wired into the test device and their operation (time tag) will be compared with the received GOOSE messages to determine if the performance of communications based solutions is analogous to the hard-wired case.

The test system may also retrieve the waveform records from the tested device and again compare them with the original waveforms from the simulation tool.



Picture 16 – Testing of IED With Process Bus and Hard Wired Interface

Picture 16 shows the system configuration for hybrid testing of IEDs that have relay outputs and at the same time support GOOSE messages.

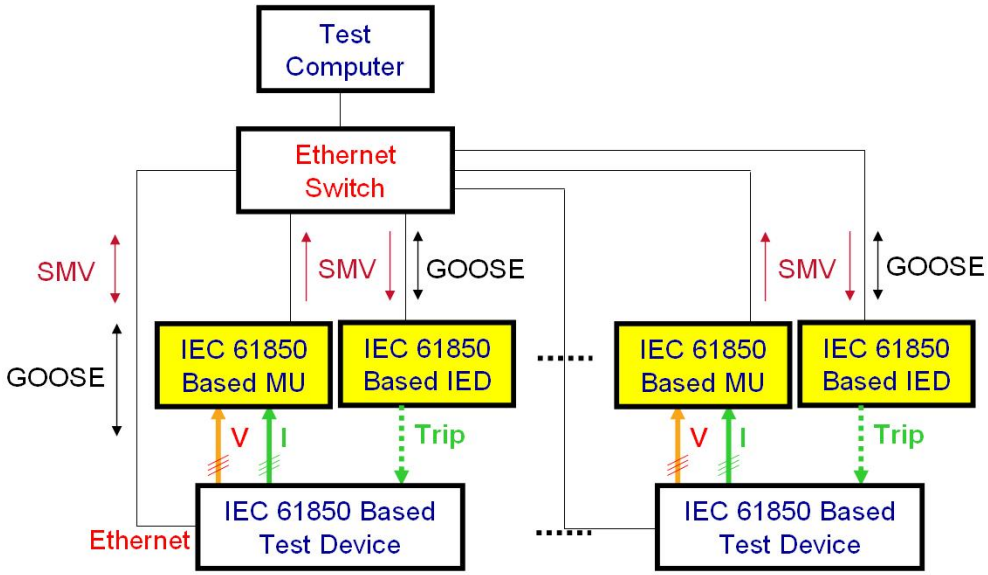
5.7 Tools for Functional Testing of IEC 61850-8-1 and IEC 61850-9-2 Based Bay and Substation Level Distributed Applications

The testing of distributed bay and substation level functions that are based on communications only – IEC 61850 8 – 1 or 9 – 2 – will be similar functionally to the testing an individual IEDs. The main difference is that in this case there will be multiple test devices with virtual simulators or analog outputs. The simulation of the substation and system environment required for the functional testing of bay and system level functions will require the simulation of multiple merging units (IEC 61850 9-2 interface) and other IEDs (IEC 61850 8-1 interface).

Considering the fact that 100 MB/s is the common Ethernet today, the number of Merging Unit simulators may require multiple computers to simulate all the required sampled analog values and GOOSE messages.

The simulation tool will also be different, because first of all it will require a multi-node system simulator. Once the results from the simulation are available, it requires the development of methods to split the results from the Network System Simulator and distribute them between the individual physical devices that perform the simulations, as well as to make them available as sampled measured values from the virtual merging units that participate in the test.

The evaluation of the performance of the distributed functions in this case will be based on the subscription of the test system components to the GOOSE messages from the different IEDs participating in the tested distributed applications. If these devices also have relay outputs hardwired to the test devices, their operation will have to be monitored as well in order to evaluate the performance of the tested system and if necessary compare the communications based to hardwired solutions. [2]. A simplified block diagram of this test system is shown in the Picture 17.



Picture 17 – Testing Bay or System Level Distributed Applications Testing

5.8 Functional Test System Architecture

From a user perspective, it is important that a test system provide means to specify and realize functional tests that are as independent of SAS and tools suppliers as possible. In this respect, the following requisites are judged necessary for functional testing of Substation Automation Systems:

- No specific vendor and manufacturer tool or technology;
- Only generic and normalized requisites;
- Specified by a tool independent language.

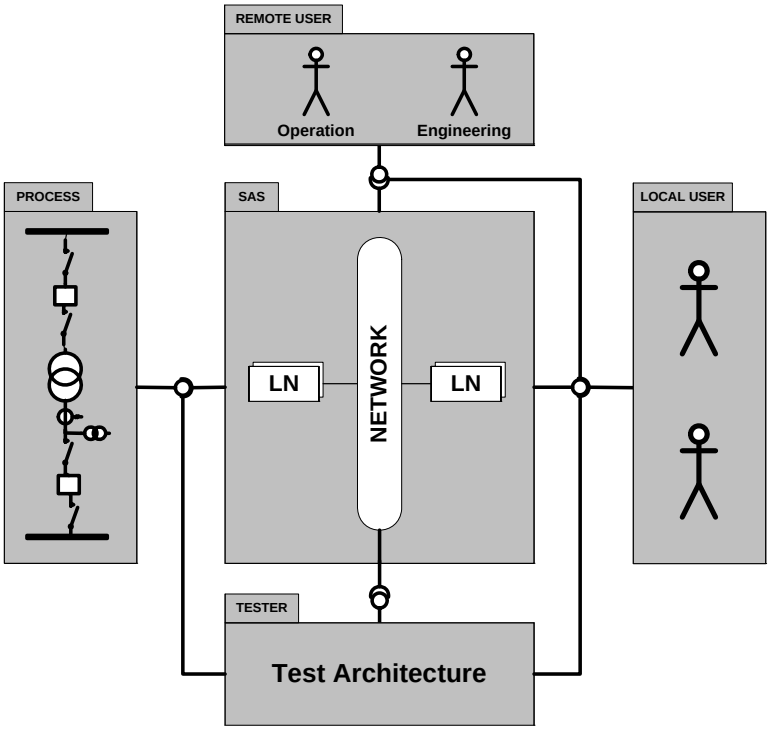
To attain these requisites, some suggestions are pertinent to the test architecture:

- Physical devices should be used as proxies/gateways to direct access to the physical and logical devices in the substation;
- Testing devices should be used with the ability to address, send and receive messages to/from each selected node;
- Object oriented (UML) classes should be defined to be used in the gateways/proxies to specify their behavior.

In this respect, the following concepts are borrowed from the UML Test Profile from the OMG (Object Management Group) to specify the capabilities functional tests tools:

- **Test Architecture** - concepts related to test structure and test configuration (containing the relationship of elements involved in a test project);
- **Test Behavior** - concepts related to the dynamic aspects of test procedures;
- **Test Data** - structures and meaning of values to be processed in a test;
- **Test Time** - concepts or a time quantified definition of test procedures.

To introduce these concepts, the following picture shows a UML package diagram of a general architecture for a test system, connected to an SAS. The SAS is seen as a network of logical nodes, linked to a substation process, and accessed by local and remote operators. The package representing the Test Architecture should be able to intercept the interfaces among users and process equipment to the SAS, and to have access to specific logical nodes through the station and process networks. These accesses should be bidirectional, that is, the test system should be capable to monitor or simulate any kind of messages and signals in these interfaces. These include analog and digital signals to and from the station equipments, network GOOSE and SV messages among logical nodes, and operator actions and messages sent and received by human-machine interfaces.

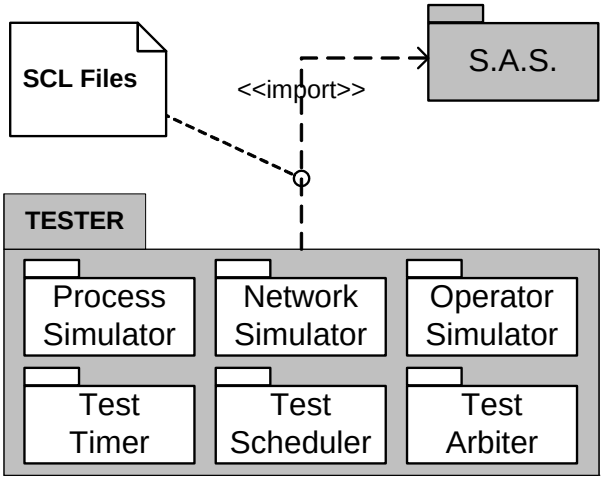


Picture 18 – Functional Test Setup

In order to generate and monitor all these signals and messages, it is recommended, following the suggestion of the UML Test Profile form OMG, that the following components be part of the Test architecture:

- Process Simulator;
- Network Simulator;
- Operator Simulator;
- Test Timer;
- Test Scheduler;
- Test Arbiter.

These components are shown as internal packages of the Test Architecture, in the following picture.



Picture 19 – Test Package

Note that the Tester package imports the SCL files from the SAS package, to get access to all logical nodes and network addresses that form the automation system under test. The following is a description of each package forming the Tester.

5.8.1 Process Simulator

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. The following classes are suggested as a minimum to form a Process Simulator in the Test Architecture:

- **VoltageOutput** – class to generate voltage waveforms;
- **CurrentOutput** – class to generate current waveforms;
- **DigitalInput** – class to generate digital input waveforms;
- **DigitalOutput** – class to read digital input waveforms;

Table 10 to Table 13 list suggested attributes and methods of these classes to support the simulation of a substation process in a test architecture. These classes and functions will be used to illustrate the test specification in the remaining chapters.

Table 10 – Class VoltageOutput Specification

VoltageOutput : TestComponent	
Attribute	Explanation
+ Node	Reference to a Logical Node
- File	Reference to a Comtrade File
Method	Explanation
+ VoltageOutput (Node) : VoltageOutputName	Constructor of a voltage output object connected to Node
+ SetVoltageOutputSequence (File)	Read a Comtrade File with voltage output for Node

+ GetVoltageOutputSequence (Duration)	Record voltage output sequence for Duration from Node
+ SetACVoltageOutput (Module, Angle)	Set an AC Module & Angle voltage output to Node
+ GetACVoltageOutput () : (Module, Angle)	Read current AC Module & Angle voltage output to Node
+ SetDCVoltageOutput (Module)	Set a DC Module voltage output to Node
+ GetDCVoltageOutput (Module)	Get current DC Module voltage output to Node
+ StartVoltageOutput () : Time	Start voltage output (set before) to Node and return Time
+ StopVoltageOutput () : Time	Stop voltage output to Node and return Time

Table 11 – Class CurrentOutput Specification

CurrentOutput : TestComponent	
Attribute	Explanation
+ Node	Reference to a Logical Node
- File : Private	Reference to a Comtrade File
Method	Explanation
+ CurrentOutput (Node) : CurrentOutputName	Constructor of a current output object connected to Node
+ SetCurrentOutputSequence (File)	Read a Comtrade File with current output for Node
+ GetCurrentOutputSequence (Duration)	Record current output sequence for Duration from Node
+ SetACCurrentOutput (Module, Angle)	Set an AC Module & Angle current output to Node
+ GetACCurrentOutput () : (Module, Angle)	Read current AC Module & Angle current output to Node
+ SetDCCurrentOutput (Module)	Set a DC Module current output to Node
+ GetDCCurrentOutput (Module)	Set current DC Module current output to Node
+ StartCurrentOutput () : Time	Start current output (set before) to Node and return Time
+ StopCurrentOutput () : Time	Stop current output to Node and return Time

Table 12 – Class DigitalInput Specification

DigitalInput : TestComponent	
Attribute	Explanation
+ Node	Reference to a Logical Node
- File	Reference to a Comtrade File
Method	Explanation
+ DigitalInput (Node) : DigitalInputName	Constructor of a digital input object connected to Node
+ GetDigitalInputSequence (Duration)	Record input sequence in File for Duration from Node
+ GetDigitalInput () : Boolean	Read current Boolean input from Node
+ FirstUpInputTransition () :Time	Get Time of first positive transition from Node in File
+ FirstDownInputTransition () :Time	Get Time of first negative transition from Node in File
+ LastUpInputTransition () :Time	Get Time of last positive transition from Node in File
+ LastDownInputTransition () :Time	Get Time of last negative transition from Node in File

Table 13 – Class DigitalOuptut Specification

DigitalOutput : TestComponent	
Attribute	Explanation
+Node	Reference to a Logical Node
-File	Reference to a Comtrade File
Method	Explanation
+DigitalOutput (Node) : DigitalOutputName	Constructor of a digital output object connected to Node
+ SetDigitalOutputSequence (File)	Read a Comtrade File with digital output for Node
+ GetDigitalOutputSequence (Duration)	Record digital output sequence for Duration from Node
+ SetDigitalOutput (Boolean)	Set a Boolean output to Node
+ GetDigitalOutput () : Boolean	Read current Boolean output to Node
+ StartDigitalOutput () : Time	Start digital output (set before) to Node and return Time
+ StopDigitalOutput () : Time	Stop digital output to Node and return Time
+ FirstUpOutputTransition () :Time	Get Time of first positive transition from Node in File
+ FirstDownOutputTransition () :Time	Get Time of first negative transition from Node in File
+ LastUpOutputTransition () :Time	Get Time of last positive transition from Node in File
+ LastDownOutputTransition () :Time	Get Time of last negative transition from Node in File

5.8.2 Network Simulator

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 following table is suggested as a minimum specification for the attributes and methods of a class to model a Network Simulator in the Test Architecture. It will be used to illustrate the test specification in the remaining chapters.

Table 14 – Class NetworkSimulator Specification

NetworkSimulator : TestComponent	
Attribute	Explanation
+ Node	Reference to a Logical Node
- File	Reference to a message File
Method	Explanation
+ NetworkSimulator (Node):NetworkSimulatorName	Construct a network simulator object connected to Node
+ SetMessageSequence (File)	Read a File with network messages to and from Node
+ GetMessageSequence (Duration)	Record in File messages for Duration to and from Node
+ RepeatMessageSequence (File,Interval)	Read a File at Interval with messages to and from Node
+ StartNetworkSimulator () : Time	Start network messages to and from Node & return Time
+ StopNetworkSimulator () : Time	Stop network messages to and from Node & return Time
+ FirstPICOMFrom (FromNode,Picom) :Time	Get Time of first Picom from FromNode to Node in File
+ LastPICOMFrom (FromNode,Picom) :Time	Get Time of last Picom from FromNode to Node in File

+ FirstPICOMTo (ToNode,Picom) :Time	Get Time of first Picom from Node to ToNode in File
+ LastPICOMTo (ToNode,Picom) :Time	Get Time of last Picom from Node to ToNode in File

5.8.3 Operator Simulator

The **Operator Simulator** is a package of classes to emulate and monitor the messages received and sent from and to an operator. These should cover text and messages exchanged by local and remote IHM. The following table is suggested as a minimum specification for the attributes and methods of a class to model an Operator Simulator in the Test Architecture. It will be used to illustrate the test specification in the remaining chapters.

Table 15 – Class Operator Specification

Operator : TestComponent	
Attribute	Explanation
+ Node	Reference to a Logical Node
Method	Explanation
+ Operator (Node) : OperatorName	Constructor of an operator object at a station in Node
+ OperatorAct (Action) : Verdict	Perform and confirm manual Action at staton in Node
+ OperatorConfirm (Action) : Verdict	Confirm automatic Action by SAS at station in Node

5.8.4 Test Timer

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. Table 16 and Table 17 define a minimum specification of these classes, to be used to illustrate the test specification in the remaining chapters.

Table 16 – Class TestTimer Specification

TestTimer : Timer	
Attribute	Explanation
- Time	Internal time count
Method	Explanation
+ TestTimer () : TestTimerName	Constructor of a timer object named TestTimerName
+ Start ()	Start timer counting
+ Stop ()	Stop timer counting
+ GetTime () : Time	Get current time count

Table 17 – Class Timer Specification

<<interface>> Timer

Attribute	Explanation
{readOnly} isRunning : Boolean	True when the timer is running
Method	**Explanation**
start(expire : Time)	Start timer counting
stop()	Stop timer counting
read() : Time	Get current time count

5.8.5 Test Scheduler

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. Table 18 and Table 19 define a minimum specification of these classes, to be used to illustrate the test specification in the remaining chapters.

Table 18 – Class TestScheduler Specification

TestScheduler : Scheduler	
Attribute	**Explanation**
-File	Internal SCL File
-Script	Internal reference to a test Script
Method	**Explanation**
+ ReadSCL (File)	Imports SCL file for Substation Automation System
+ ReadScript (Script)	Imports XML Script file for testing
+ StartTestCase (Name)	Start running test case named Name
+ Wait (Duration)	Stop further script processing for Duration ms

Table 19 – Class Scheduler Specification

<<interface>> Scheduler	
Attribute	**Explanation**
Method	**Explanation**
startTestCase()	
finishTestCase(t : TestComponent)	
createTestComponent(t : TestComponent)	

5.8.6 Test Arbiter

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. Table 20 and Table 21 define a minimum

specification of these classes, to be used to illustrate the test specification in the remaining chapters.

Table 20 – Class TestArbiter Specification

TestArbiter : Arbiter	
Attribute	Explanation
Method	Explanation
+ TestArbiter () : TestArbiterName	Constructor of an arbiter object named TestArbiterName
+ TestArbiterConfirm (Expression) : Verdict	Evaluate and return value of Expression as a Verdict

Table 21 – Class Arbiter Specification

<<interface>> Arbiter	
Attribute	Explanation
Method	Explanation
getVerdict() : Verdict	Evaluate a verdict
setVerdict(v : Verdict)	Define a verdict

5.9 Conclusions

IEC 61850 peer-to-peer communications based systems require a different approach and set of tools for proper testing of the individual components of the systems, as well as the evaluation of the performance of the distributed functions.

This chapter presents the concept of distributed functions based on sampled analog values and GOOSE messages and describes the components of the system. The chapter describes the approach to system testing, as well as the different components of a test system designed to enable the functional testing of IEC 61850 based functions, including:

- Configuration tool based on the Substation Configuration Language defined in Part 6 of IEC 61850.
- Simulation tool that generates the current and voltage waveforms
- Virtual Merging Units and IED simulators
- Test Evaluation tool
- Reporting tool

If the tested device has relay outputs as well, their operation will be compared with the received GOOSE messages to determine if the performance of communications based solutions is analogous to the hard-wired case.

The test system may also retrieve the waveform records from the tested device and again compare them with the original waveforms from the simulation tool.

The chapter describes the test system architecture for the testing of individual devices using sampled analog values, as well as protection or recording schemes that involve multiple devices.

The following chapter will illustrate how to use this architecture to design functional test cases.

6. Functional Test Specification

6.1 Introduction

Once the SAS has been specified, all the requirements defined in the Functional Requirements Specification (FRS) have been defined, and the capabilities of the system in the Function Implementation Conformance Statement (FICS) have been produced, functional tests can be designed. Using the tools defined in the previous chapter, testing will aim to check and validate the functional specification described in these documents. Test specification will mainly be directed to simulate the analogue inputs and digital information sent to the SAS, and to verify the proper operation of all functions involved in the SAS.

In general terms a test sequence to verify SAS functional requirements is built as follows:

- Injection of (GOOSE/SMV/MMS/GSSE /Analog/Discrete) messages at appropriate access points of station, bay or process network, simulating the start events of the function, as defined by IEC 61850-5 module [5].
- Recording of (GOOSE/SMV/MMS/GSSE/ Analog/Discrete) messages at appropriate access points of station, bay or process networks which attest the correct execution of the function.
- Evaluation of any functional/performance criteria based on these recordings, and comparing with functional requirements

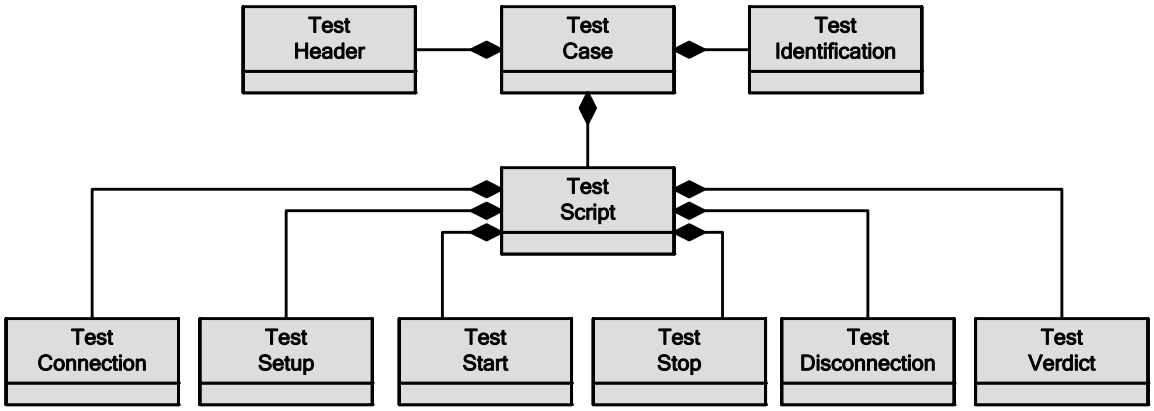
This chapter describes how to construct a functional test specification, using the test architecture and tools proposed in the previous chapter. The proposed test architecture allows the specification of tests to be designed and read by a user and executed by computers. That is, any user test specification that obeys these rules can be processed by a test tool that follows the suggested architecture. Use of Extensible Markup Language (XML) as the test script language [25], associated to a user oriented test template can attain both ends.

6.2 SAS Test Specification

A functional test specification can be divided in seven parts that must be designed in sequence to verify a functional specification. These sections correspond to the following sections in a test script:

- Test Specification;
- Test Connections;
- Test Setup;
- Test Start;
- Test Stop;
- Test Disconnection; and
- Test Verdict.

These sections can be organized in a class diagram shown on Picture 20. These classes serve as a base to organize the XML schema of Appendix C.



Picture 20 – Diagram of the Test Case Components

According to this picture, each Test Case is composed of a Test Identification that nominates the test case, a Test Header that gives information about the Test Case, and a Test Script that contains the test steps.

The Test Identification and Test Header can be specified in a table format, or as comments in a Test Script. The following table shows an example of a possible format, where each test case is identified by a Code, a Name, a Description, the Customer, Substation, and the corresponding SCL and FRS files, according to the template described in Appendix B.

Table 22 – SAS Test Specification

Functional Test Case	
Code	PDIFF
Name	(Transformer) Differential Protection
Description	Protective function that operates on a percentage or phase angle or other quantitative difference of two currents
Use Case Description	
Customer	CHESF
Substation	Baden 230/132kV
SCL File	XMLExmpleSCL.scl
FSR File	FSR file
FICS File	FICS file

The same information could be inserted in the header of an XML script file, following the schema of Appendix C:

```
<?xml version="1.0" encoding="utf-8"?>
<TestCase xmlns="http://tempuri.org/XMLSchema.xsd">
  <Code>PDIF</Code>
  <Name>(Transformer) Differential Protection</Name>
  <Description>
    Protective function that operates on a percentage or phase angle
    or other quantitative difference of two currents
  </Description>
  <Customer>CHESF</Customer>
```

```
<Substation>Baden 230/132kV Substation</Substation>
<SCLFile>Baden.scl, version 2.0</SCLFile>
<FICSFile>Baden.fics</FICSFile>
```

In this script, the SCL file should be specified by a universal locator like a place in network folder or an internet site, in order to be automatically located by the test computer.

6.3 SAS Functional Test Connections

Before starting the tests it is necessary to identify the function to be tested and the relation with the other functions of the SAS. The functional specification and the test requirements described in chapters 2 and 3, provide this information.

Circuit breakers, switches, voltage and current transformers, blocking and trip signals, commands, oscilographic registers, events, etc can be classified as Process, network, or operator classes. In addition, there are other modules as timers, the sequencer, and the arbiter to be taken into account when a function is to be tested.

The identification of all components involved in the test is followed by the definition of all connections between these components and the SUT. Such connections have to be declared at the beginning of the test script so that the Test Tool can select the modules and connections to perform the test. The specification is built by calling the constructor of every object needed in the test, which can be formatted as shown in Table 23.

Table 23 – SAS Test Connection

Test Connection		
Step	**Command**	**Meaning**
1	Call test component constructor 1	Create a test component and connect it to SUT
2	Call test component constructor 2	Create a test component and connect it to SUT
N	…	…

The test components and their connections can be declared as:

- **Instantiation of process simulator classes** – for each source or destination of signals exchanged by the SAS with the process an object should be created, derived from the appropriate class in the process simulator. These classes model the behavior of primary process equipments like breakers, potential transformers, current transformers, power transformers, etc. A current transformer TCTR1 could be modeled, for example, by the constructor of the class CurrentOutput, in a test script by the command

  ```
  Tctr1 = CurrentOutput(TCTR1)
  ```

  or using an XML script line like

```
<CurrentOutput CurrentOutputName="Tctr1" Node="TCTR1"/>.
```

- **Instantiation of network simulator classes** - for each logical node selected for message injection or monitoring during the test, an object should be created, derived from the appropriate class in the network simulator. A differential protection logical node PDIF could be modeled, for example, by the constructor of the class NetworkSimulator, in a test script by the command

  ```
  Pdif = NetworkSimulator(PDIF)
  ```

  or using an XML script line like

  ```
  <NetworkSimulator NetworkSimulatorName="Pdif" Node="PDIF"/>.
  ```

- **Instantiation of operator simulator classes** - for each operator node selected for message injection or monitoring during the test, an object should be created, derived from the appropriate class in the operator simulator. An operator connected to human machine interface node IHMI could be modeled, for example, by the constructor of the class Operator, in a test script by the command

  ```
  Operator1 = Operator(IHMI)
  ```

  or using an XML script line like

  ```
  <Operator OperatorName="Operator1" Node="IHMI"/>.
  ```

- **Instantiation of test sequencer class** – this class does not need an explicit instantiation from the user. It is automatically created by the test instrumentation as a script sequencer.

- **Instantiation of test timer class** – timers are used to measure or delay the occurrence of events in a test. A timer named Timer1 can be created by calling the constructor of class TestTimer, by the command

  ```
  Timer1 = TestTimer()
  ```

  or using an XML script line like

  ```
  <TestTimer TestTimerName="Timer1"/>.
  ```

- **Instantiation of test arbiter class** - for each condition verified during the test, a message should be directed to the appropriate object created in a previous step, and derived from the appropriate TestArbiter class. The constructor for this object can be, for example

  ```
  Arbiter1 = TestArbiter()
  ```

or using an XML script line like

```
<TestArbiter TestArbiterName="Arbiter1"/>.
```

Using this object, a difference in time between two events Time2 and Time1 could be checked by the command

```
Verdict1 = Arbiter1->TestArbiterConfirm (Time2-Time1<100)
```

or using an XML script line like

```
<TestArbiterConfirm TestArbiterName="Arbiter1" Expression="100 > Time2-Time1" Verdict="Verdict1"/>.
```

6.4 SAS Functional Test Setup

Test requirements define particular parameters and set points on test components. Before starting the test some initial conditions have to be defined. Initial position of circuit breakers and switches, initial values of current and voltage, initial status of blocking and trip signals, etc. Most of the times SUT has to be set with particular parameters before performing the test.

For each object needing initialization during the test, a message should be directed to the appropriate object created in the previous step, and derived from the appropriate class in the process simulator. These messages can be formatted as shown in Table 24.

Table 24 – SAS Test Setup

Test Setup		
Step	**Command**	**Meaning**
1	Set parameters to test component 1	Prepare test component to apply signal to SUT
2	Set parameters to test component 2	Prepare test component to apply signal to SUT
N	…	…

To initialize to zero the output current from current transformer Tctr1, for example, a call to its object function SetCurrentOutput could be included in a test script by the command

```
Tctr1->SetACCurrentOutput(0,0)
```

or using an XML script line like

```
<SetACCurrentOutput CurrentOutputName="Tctr1" Module="0" Angle="0"/>.
```

6.5 SAS Functional Test Start

Once all objects are initialized, the setup specification is ready with the connections and initial conditions of all components specified. Next part of the script is directed to apply the analogue and digital signals to the SUT taking into

account the requirements needed by the test. These can be formatted as shown in Table 25.

Table 25 – SAS Test Start

Test Start		
Step	**Command**	**Meaning**
1	Start test component 1	Apply test component signals to SUT
2	Start test component 2	Apply test component signals to SUT
N	…	…

All defined objects that need to be started are commanded in this part of the script:

- **Start of process simulator objects –** Using the process objects created and initialized previously, the occurrence of events like injection of current by Tctr1 could be recorded in variable Time1 by the command

  ```
  Time1=Tctr1->StartCurrentOutput ()
  ```

  or using an XML script line like

  ```
  <StartCurrentOutput CurrentOutputName="Tctr1" Time="Time1"/>.
  ```

- **Start of network simulator objects -** Using the network objects created and initialized previously, the occurrence of events like sending of messages by Pdiff could be started by the command

  ```
  Pdiff->StartNetworkSimulator()
  ```

  or using an XML script line like

  ```
  <StartNetworkSimulator NetworkSimulatorName="Pdif"/>.
  ```

- **Start of operator simulator objects -** Using the operator objects created and initialized previously, the occurrence of events like sending of messages by Operator1 could be started by the command

  ```
  Operator1->StartOperatorSimulator()
  ```

  or using an XML script line like

  ```
  <StartOperatorSimulator OperatorSimulatorName="Operador1"/>.
  ```

- **Start of test timer object** - Using the timer objects created and initialized previously, the counting of time intervals and events by Timer1 could be started by the command

  ```
  Timer1->Start ()
  ```
  or using an XML script line like

  ```
  <Start TestTimerName="Timer1"/>.
  ```

6.6 SAS Functional Test Stop

This section of the specification is mostly related to the previous one. In this part the test specification defines the conditions to end the test and stop applying analogue and digital signals to the SUT. These conditions can be specified as shown in Table 26.

Table 26 – SAS Test Stop

Test Stop		
Step	**Command**	**Meaning**
1	Stop test component 1	Suspend test component signals to SUT
2	Stop test component 2	Suspend test component signals to SUT
N	…	…

The following actions can be necessary to stop a test script:

- **Stop of process simulator objects** – for each process simulator object that is generating signals, an appropriate message should be sent to stop the signal generation. To stop the current signal from Tctr1, the following commands could be inserted in the script:

  ```
  Tctr1->SetACCurrentOutput(0)
  Tctr1->StartCurrentOutput()
  ```

  or using an XML script line like

  ```
  <SetACCurrentOutput CurrentOutputName="Tctr1" Module="0"
  Angle="0"/>
  <StartCurrentOutput CurrentOutputName="Tctr1" Time=""/>
  ```

- **Stop of network simulator objects** – for each object representing a logical node for which the test script is sending or recording messages, an appropriate message should be sent to end it. To stop the recording of messages coming from the logical node Pdiff, the following command could be inserted in a script

  ```
  Pdiff->StopNetworkSimulator()
  ```

  or using an XML script line like

  ```
  <StopNetworkSimulator NetworkSimulatorName="Pdif" Time=""/>
  ```

It is necessary to remark that SAS Functional Test Start and SAS Functional Test Stop sections can be included in the specification several times depending on the type of the test. For example, these sections will be included only once when the test is performed on an overcurrent unit, while these sections will be included several times when the recloser is tested.

6.7 SAS Functional Test Disconnections

Before closing and disconnecting all test components from SUT the specification must collect all necessary events, reports, etc. from the components and SUT. It is assumed that all objects are closed and their resources liberated by the test sequencer class, as soon as the test script is ended. These actions can be collected as shown in Table 27.

Table 27 – SAS Test Disconnection

Test Disconnection		
Step	**Command**	**Meaning**
1	Log test component 1 results	Register results of test component from SUT
2	Log test component 2 results	Register results of test component from SUT
N	…	…

To collect data from each object created, like the instant that the first PICOM type 22 is sent from logical node Pdiff to CSWI1, the following command, for example, can be inserted in the test script

```
Time2 = Pdif->FirstPICOMTo(CSWI1,22)
```

or using an XML script line like

```
<FirstPICOMTo NetworkSimulatorName="Pdif" ToNode="CSWI1" Picom="22" Time="Time2"/>.
```

6.8 SAS Functional Test Verdict

The last section of the script specifies the conditions that should be checked before the emission of a verdict. A verdict is a pass or fail conclusion about some results collected from the test. All verdicts can be collected and inserted in a Test Verdict specification as shown in Table 28.

Table 28 – SAS Test Verdict

Test Verdict		
Step	**Command**	**Meaning**
1	Check log results from component 1	Publish test component verdict
2	Check log results from component 2	Publish test component verdict
N	…	…

As an example, to verify that the first trip signal (PICOM type 22) sent from the logical node Pdiff after the inception of a fault occur in less than 100ms, the following command could be inserted in the test script:

```
Verdict1 = Arbiter1->TestArbiterConfirm(Time2-Time1<100)
```

or using an XML script line like

```
<TestArbiterConfirm TestArbiterName="Arbiter1" Expression="100 > Time2-Time1" Verdict="Verdict1"/>.
```

In these commands, Time2 and Time1 are instants of time collected by the test script in previous steps.

The verdict section will have as many statements as needed to provide the test tool with the necessary information to determine the correct functioning of the SUT. In case of failure, this information will be useful for the human tester to identify the problem in detail.

All these commands will be illustrated in a complete test case example, in the next chapter.

7. Test Case Example

7.1 Introduction

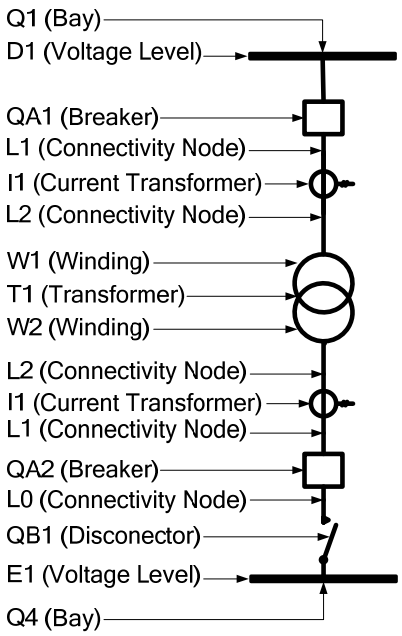
This chapter shows the design of functional and performance tests for a simple substation automation system, based on the methodology described on previous chapters. Starting from a transformer bay station layout, a functional specification is built for an automation system, according to the format proposed on Chapter 2. An SCL design specification is assumed, as a model of the SAS system for the station. Performance requirements are also specified for the automation system.

Failure mode and effects analysis (FMEA) and Hazard and Operability Analysis (HAZOP) are conducted on the example system, following the format of Chapter 4, to identify failure modes, and to establish test coverage criteria. Finally, both functional and performance tests are derived, based on these specifications, formatted in UML diagrams, tables and XML format, according to the test architecture of Chapter 5, the specification formats of Chapter 6 and the XML Schema proposed on Appendix C.

It is expected that this example demonstrates the joint application of all proposed techniques of previous chapters, and suggests similar applications to other SAS systems.

7.2 Substation Specification

Picture 21 shows a single line diagram of a simple transformer bay in a station. It is a simplified and modified version of Baden 220/132kV substation example listed on IEC 61850-6 part. This system will be used in the remaining of this chapter to illustrate the testing methodology.



Picture 21 – Substation Layout Diagram

In this station, transformer T1 has two windings, W1 and W2. W1 is connected to a 220 kV voltage level D1 at bay Q1, through connectivity nodes L1 and L2. Winding W2 is connected to bay Q2 at 110 kV voltage level E1. There is a current transformer at each of the transformer windings.

The 132kV bay E1Q2 contains a circuit breaker QA2 and a bus bar disconnector QB1, both electrically connected at connectivity node L0, as well as current transformer I1 between connectivity nodes L1 and L2.

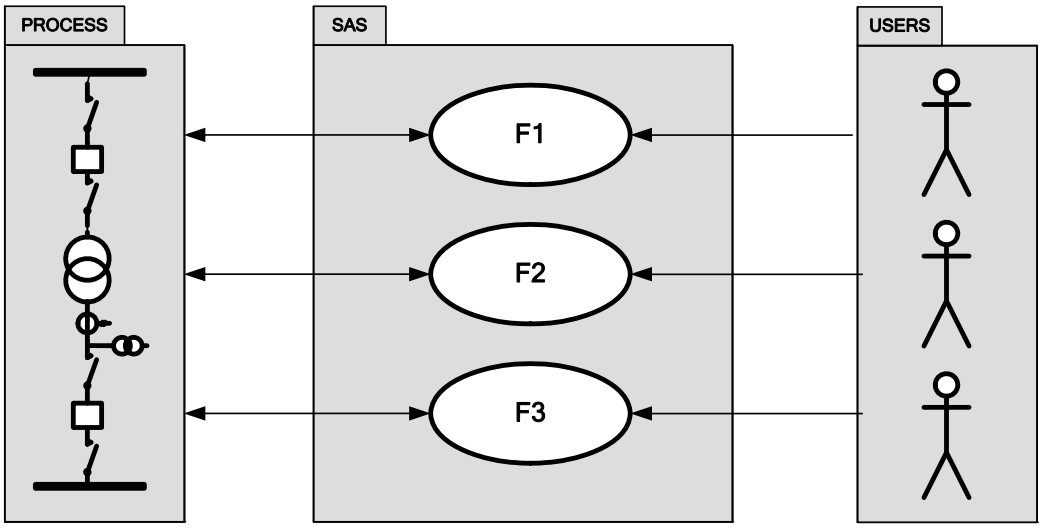
This diagram or some other graphical representation can be recovered, with adequate software tools, from the substation section of the SCL file included in Appendice E. The file is a syntactically correct, but not fully completed SCL file for the example used in this chapter, including the communication section.

7.3 SAS Functional Specification

Suppose that the following functions and performance requirements were defined by the user, for this bay:

F1 - Trip breaker QA1 in less than 100ms when there is a fault on transformer T1, and report on operator console;
F2 - Trip breaker QA2 in less than 100ms when there is a fault on transformer T1, and report on operator console;
F3 - Operate a differential protection in less than 100ms when there is a fault on transformer T1, and report on operator console.

A UML Use Case diagram can be constructed, as part of the above functional specification, as shown on the following diagram.



Picture 22 – Functional Use Cases

The following table shows the Functional Specification of a proposed SAS, using a differential protection, formatted as a FICS (Functional Implementation Conformance Statement) file, as defined on Chapter 2, for the station transformer bay described on the previous topic, covering functions F1, F2 and F3 above.

Table 29 – Functional Implementation Conformance Statement

Functional Implementation Conformance Statement	
Code	PDIFF
Name	(Transformer) Differential Protection
Description	Protective function that operates on a percentage or phase angle or other quantitative difference of two currents
Customer	CHESF
Substation	Baden 220/132kV Substation
SCL File	Baden.scl, version 2.0
Primary User (Actor)	T1 transformer
Secondary User (Actor)	Station operator
Stakeholder & Interest	Operation of this function prevents major damage on transformer
Extensions	None
Function Description	
Trigger	An internal short circuit on the transformer
Components or Logical Nodes	XCBR1, XCBR2, PDIF, TCTR1, TCTR2, CSWI1, CSWI2

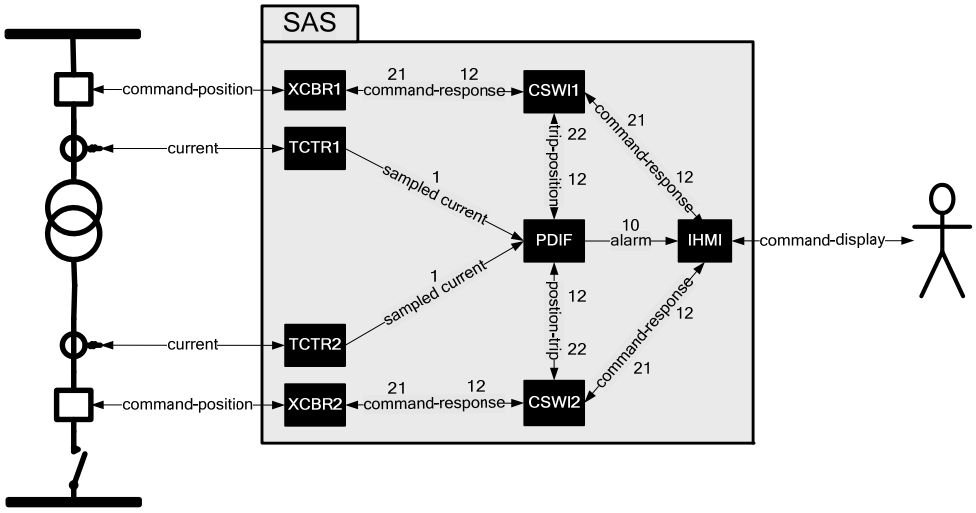
Process Equipments	Transformer T1, Breakers BR1 and BR2		
Performance	Tripping time of XCBR1 & XCBR2 < 100ms upon inception of short circuit		
Preconditions	Switch SSWI closed, Breakers XCBR1 & XCBR2 closed		
Post conditions on Success	Breakers XCBR1 & XCBR2 open, alarm on operator console IHMI		
Post conditions on Failure	Breakers XCBR1 or XCBR2 still closed, possible damage of transformer, opening of remote breakers		
Use Case Description			
Use Case name	MSS		
Basic Course Description	1	Transformer	Internal short circuit on Transformer
	2	XCTR1 or XCTR2	Mismatch between currents
	3	PDIF	Detect, trip CSWI1 & CSWI2, alarm IHMI
	4	CSWI1 & CSWI2	Send trip to XCBR1 & XCBR2
	5	XCBR1 & XCBR2	Trip and respond to CSWI1 & CSWI2
	6	CSWI1 & CSWI2	Respond to PDIF and report to IHMI
Extensions	5a Breaker 1 or breaker 2 failure		
	1	XCBR1 or XCBR2	Activate breaker failure protection

The following table shows the functions related to each logical node, according to this specification. This table will also be used in the test coverage analysis, to demonstrate the test effectiveness.

Table 30 – Multiple Uses of Logical Nodes

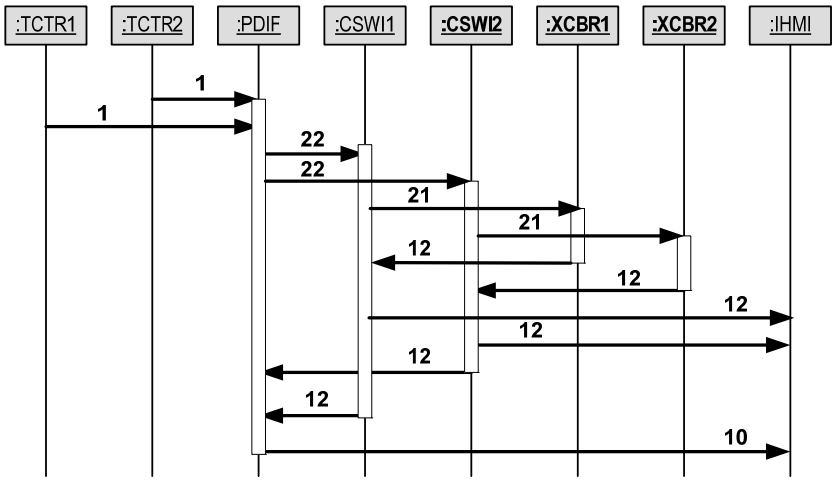
S.A.S.		LOGICAL NODE							
		XCBR1	XCBR2	CSWI1	CSWI2	TCTR1	TCTR2	PDIF	IHMI
FUNCITON	**F1**	**X**		**X**		**X**	**X**	**X**	**X**
	F2		**X**		**X**	**X**	**X**	**X**	**X**
	F3					**X**	**X**	**X**	**X**

To complement the specification, a UML communication diagram can be supplied, as shown on Picture 23, showing the type of messages exchanged between the logical nodes, according to the Use Case scenario of an internal short circuit on the transformer. Note that the exact sequence of messages is not shown as these are asynchronous events.



Picture 23 – Functional Specification by UML Communication Diagram

A UML sequence diagram can also be used to complement this information, as shown on Picture 24 for the same scenario. The messages are identified by the same numbers as shown on Picture 23. Again, the event messages are asynchronous.



Picture 24 – Functional Specification by UML Sequence Diagram

The numbers that identify the messages in these diagrams correspond to PICOM (Piece of Information Communication) types, according to Table 31, extracted from IEC 61850-5 – Annex B, for the logical nodes used on this automation system. Other message identification methods could be used, as long as they are unique for each logical node. Just a subset of these messages is used in this example.

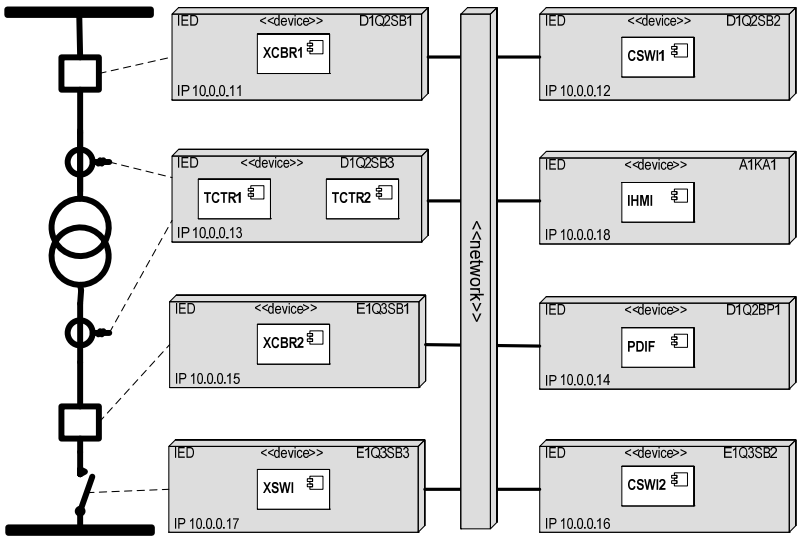
Table 31 – PICOM Type and Description

Type	Description	Range (ms)	Type
XCBR	(Circuit Breaker)		
12	Operated	10 to 1000	Event Spontaneous

CSWI	(Switch Controller)		
21	Command to Switchgear	1 to 1000	Command Spontaneous
12	Operated	10 to 1000	Event Spontaneous
TCTR	(Current Transformer)		
1	Current/Voltage	10	Value Cyclic
PDIF	(Differential Protection)		
10	Trip Indication	100 to 1000	Event Spontaneous
22	Trip Command	1	Command Spontaneous

7.4 SAS Design Specification

The design specification of this example contains all project details like node allocation among IEDs, network address (IP) of each server. The following picture shows a UML deployment diagram for the SAS of this example, with the allocation of logical nodes in eight IEDs. Each IED is identified by a label, with corresponding network address (IP) shown inside each device block. A single network connects all IEDs.

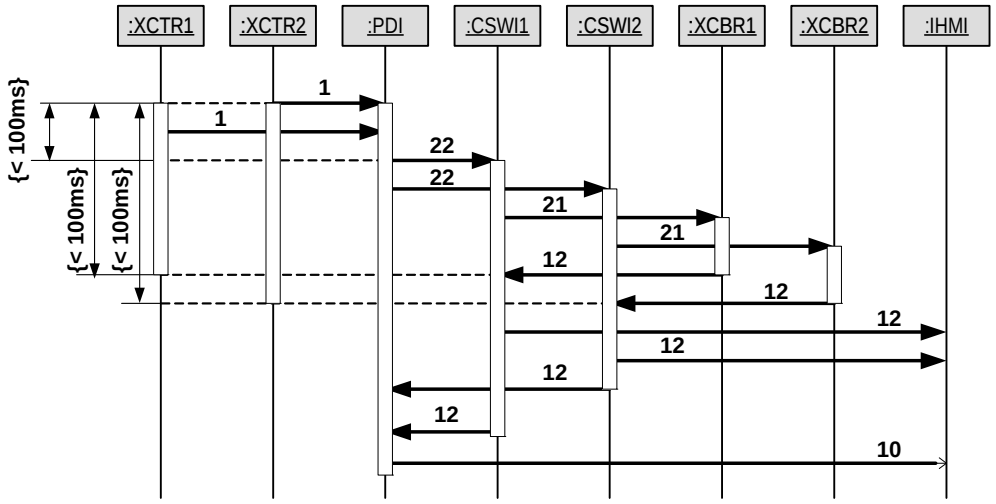


Picture 25 – Physical Design by UML Deployment Diagram

This SAS system is described in the SCL file in Appendice E. It has been checked against the IEC 61850 XML schema for SCL, and completely defines the addresses of each logical node.

7.5 SAS Performance Requirements

A UML sequence diagram can also be used to show the time performance requirement of each function, as shown on Picture 26. The messages are also identified by their PICOM numbers.



Picture 26 – Performance Specification as UML Sequence Diagram

Note that timing performance is shown as constraints for external input output signals, as well as for specific node messages of the SAS. Observe also that the timing constraint includes the opening of the circuit breaker, a time delay that is dependent on the operation of devices external to the SAS, measured by the response time of logical nodes XCBR1 and XCBR2.

7.6 Failure Mode and Effects Analysis

Before designing the tests for this system, it is necessary to perform an FMEA analysis to identify its possible faults. HAZOP will be used to identify all failure modes of each node. Network devices can also be analyzed using the same method, but in this example their defects will be assumed to be reflected on a logical node failure. Table 32 shows the relationship of HAZOP guide words with all logical nodes used in this system. Each “X” on this table represents a possible failure mode of the corresponding logical node.

Table 32 – HAZOP Analysis of Logical Nodes

Guide Word	XCBR1	XCBR2	CSWI1	CSWI2	TCTR1	TCTR2	PDIF	IHMI
No	X	X	X	X	X	X	X	X
More					X	X	X	
Less					X	X	X	
As well as							X	
Part of								X
Reverse					X	X	X	
Other than	X	X	X	X	X	X	X	X
Early					X	X		
Late	X	X	X	X	X	X	X	

Before					**X**	**X**		
After					**X**	**X**		

To this example, only the failure modes associated to the HAZOP guide word “No” will be shown on the FMEA matrix (Table 33), as they seem to be sufficient to test the specified functions. Similar tables can be constructed for the other guide words, as necessary.

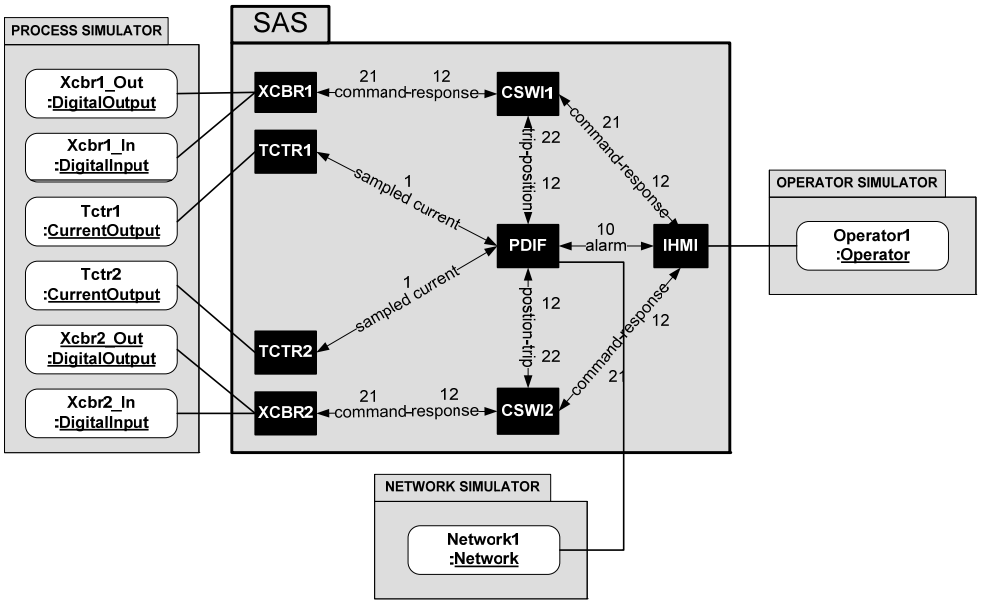
Table 33 – Failure Mode and Effects Analysis

FMEA		**LOGICAL NODE**							
		XCBR1	XCBR2	CSWI1	CSWI2	TCTR1	TCTR2	PDIF	IHMI
FUNCITON	**F1**	**X**		**X**		**X**	**X**	**X**	**X**
	F2		**X**		**X**	**X**	**X**	**X**	**X**
	F3					**X**	**X**	**X**	**X**

7.7 SAS Functional Test Specification

Having identified the potential failures of the SAS, it is possible now to design the tests to detect them. The test specification will describe the setup of the test components, instantiated from the classes of devices and functions available in the test architecture, and the test script to be followed, described as method calls or messages derived from the instantiated classes.

The following Picture 27 shows the testing objects instantiated from the test device classes, as described on Chapter 5, necessary to test this example SAS. The picture shows also their connection to the SAS logical nodes, in a UML communication diagram.



Picture 27 –Test Setup as a UML Communication Diagram

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, in a table format as described on Chapter 6. Table 34 shows the test case for the three functions specified in this example SAS.

Table 34 – Functional Test Case

Functional Test Case		
Code	PDIFF	
Name	(Transformer) Differential Protection	
Description	Protective function that operates on a percentage or phase angle or other quantitative difference of two currents	
Use Case Description		
Customer	CHESF	
Substation	Baden 230/132kV	
SCL File	XMLExmpleSCL.scl	
FSR File	FSR file	
FICS File	FICS file	
Test Description		
Step	**Command**	**Meaning**
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->GetDigitallinputSequence (1min)	Record input sequence for 1min from XCBR1
2.12	Xcbr2_In->GetDigitallinputSequence (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

Observe that each command in this script is a message or function call to a method supported by the instantiated class. It is dependent on the object oriented architecture of the test devices. The last commands evaluate the results of the test case. The seven verdicts (Verdict1 to Verdict7) check the time performance of the SAS against the specification (<100ms), as well as the operator notification of each breaker tripping and differential protection operation.

This same test case can also be specified in XML, using the XML Schema defined on Appendice C. The schema depends on the same object oriented test architecture. The following listing details this specification, after being checked against the schema.

```
<?xml version="1.0" encoding="utf-8"?>
<TestCase xmlns="http://tempuri.org/XMLSchema.xsd">
  <Code>PDIF</Code>
  <Name>(Transformer) Differential Protection</Name>
```

```
  <Description>
    Protective function that operates on a percentage or phase angle
    or other quantitative difference of two currents
  </Description>
  <Customer>CHESF</Customer>
  <Substation>Baden 230/132kV Substation</Substation>
  <SCLFile>Baden.scl, version 2.0</SCLFile>
  <FICSFile>Baden.fics</FICSFile>
  <Script>
    <!--=================================================================-->
    <!--TEST CONNECTION-->
    <!--=================================================================-->
    <TestTimer TestTimerName="Timer1"/>
    <TestArbiter TestArbiterName="Arbiter1"/>
    <DigitalInput DigitalInputName="Xcbr1_In" Node="XCBR1"/>
    <DigitalOutput DigitalOutputName="Xcbr1_Out" Node="XCBR1"/>
    <CurrentOutput CurrentOutputName="Tctr1" Node="TCTR1"/>
    <CurrentOutput CurrentOutputName="Tctr2" Node="TCTR2"/>
    <DigitalInput DigitalInputName="Xcbr2_In" Node="XCBR2"/>
    <DigitalOutput DigitalOutputName="Xcbr2_Out" Node="XCBR2"/>
    <NetworkSimulator NetworkSimulatorName="Pdif" Node="PDIF"/>
    <Operator OperatorName="Operator1" Node="IHMI"/>
    <!--=================================================================-->
    <!--TEST SETUP-->
    <!--=================================================================-->
    <SetDigitalOutput DigitalOutputName="Xcbr1_Out" Boolean="true"/>
    <SetDigitalOutput DigitalOutputName="Xcbr2_Out" Boolean="true"/>
    <SetACCurrentOutput CurrentOutputName="Tctr1" Module="0" Angle="0"/>
    <SetACCurrentOutput CurrentOutputName="Tctr2" Module="0" Angle="0"/>
    <StartDigitalOutput DigitalOutputName="Xcbr1_Out" Time=""/>
    <StartDigitalOutput DigitalOutputName="Xcbr2_Out" Time=""/>
    <StartCurrentOutput CurrentOutputName="Tctr1" Time=""/>
    <StartCurrentOutput CurrentOutputName="Tctr2" Time=""/>
    <GetMessageSequence NetworkSimulatorName="Pdif" Duration="P1M"/>
    <GetDigitalIinputSequence DigitalInputName="Xcbr1_In" Dura-
tion="P1M"/>
    <GetDigitalIinputSequence DigitalInputName="Xcbr2_In" Dura-
tion="P1M"/>
    <!--=================================================================-->
    <!--TEST START-->
    <!--=================================================================-->
    <SetACCurrentOutput CurrentOutputName="Tctr1" Module="5" Angle="0"/>
    <Start TestTimerName="Timer1"/>
    <StartNetworkSimulator NetworkSimulatorName="Pdif"/>
    <StartCurrentOutput CurrentOutputName="Tctr1" Time="Time1"/>
    <!--=================================================================-->
    <!--TEST STOP-->
    <!--=================================================================-->
    <Wait Duration="P2M"/>
    <SetACCurrentOutput CurrentOutputName="Tctr1" Module="0" Angle="0"/>
    <StartCurrentOutput CurrentOutputName="Tctr1" Time=""/>
    <StopNetworkSimulator NetworkSimulatorName="Pdif" Time=""/>
    <!--=================================================================-->
    <!--TEST DISCONNECTION-->
    <!--=================================================================-->
    <FirstPICOMTo NetworkSimulatorName="Pdif" ToNode="CSWI1" Picom="22"
Time="Time2"/>
    <FirstPICOMTo NetworkSimulatorName="Pdif" ToNode="CSWI2" Picom="22"
Time="Time3"/>
```

```
    <FirstDownInputTransition DigitalInputName="Xcbr1_In" Time="Time4"/>
    <FirstDownInputTransition DigitalInputName="Xcbr2_In" Time="Time5"/>
    <!--============================================================-->
    <!--TEST VERDICT-->
    <!--============================================================-->
    <TestArbiterConfirm TestArbiterName="Arbiter1" Expression="100 >
Time2-Time1" Verdict="Verdict1"/>
    <TestArbiterConfirm TestArbiterName="Arbiter1" Expression="100 >
Time3-Time1" Verdict="Verdict2"/>
    <TestArbiterConfirm TestArbiterName="Arbiter1" Expression="100 >
Time4-Time1" Verdict="Verdict3"/>
    <TestArbiterConfirm TestArbiterName="Arbiter1" Expression="100 >
Time5-Time1" Verdict="Verdict4"/>
    <OperatorConfirm OperatorName="Operator1" Action="PDIF Trip" Ver-
dict="Verdict5"/>
    <OperatorConfirm OperatorName="Operator1" Action="XCBR1 Trip" Ver-
dict="Verdict6"/>
    <OperatorConfirm OperatorName="Operator1" Action="XCBR2 Trip" Ver-
dict="Verdict7"/>
  </Script>
</TestCase>
```

7.8 Test Coverage

To verify the test coverage of this test case, Table 35 expands the FMEA analysis of Table 33 to include the failure nodes of every logical node covered by this test case, related to HAZOP guide word "*No*". Note that, with exception of node XSWI, not related to functions F1, F2 and F3, all failure modes of the other logical nodes are tested by this test case.

Table 35 – Test Coverage Analysis for HAZOP Guide Word No

FMEA		LOGICAL NODE							
		XCBR1	XCBR2	CSWI1	CSWI2	TCTR1	TCTR2	PDIF	IHMI
FUNCITON	**F1**	**X**		**X**		**X**	**X**	**X**	**X**
	F2		**X**		**X**	**X**	**X**	**X**	**X**
	F3					**X**	**X**	**X**	**X**
TEST COVERAGE		**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
TEST CASE	**T1**							**X**	
	T2							**X**	
	T3	**X**		**X**		**X**	**X**	**X**	
	T4		**X**		**X**	**X**	**X**	**X**	
	T5					**X**	**X**	**X**	**X**
	T6	**X**		**X**		**X**	**X**	**X**	**X**
	T7		**X**		**X**	**X**	**X**	**X**	**X**

More detailed test coverage could be achieved using all the failures generated by all HAZOP guide words, as seen on Table 36. It can be seen that all failure modes

identified by letter “X” are possible failures not covered by this test case. Failures identified by the word “OK” are covered by the test case. This should help the test designer to decide on the need of further testing of this system.

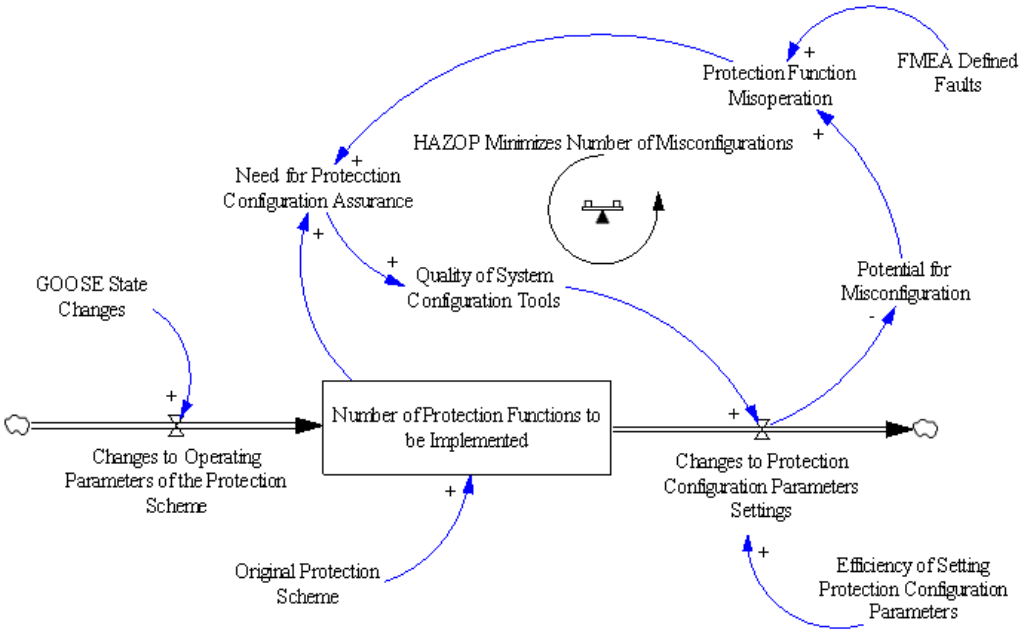
Table 36 – Test Coverage Analysis for all HAZOP Guide Words

Guide Word	XCBR1	XCBR2	CSWI1	CSWI2	TCTR1	TCTR2	PDIF	IHMI
No	**OK**	**OK**	**OK**	**OK**	**OK**	**OK**	**OK**	**OK**
More					**X**	**X**	**X**	
Less					**X**	**X**	**X**	
As well as							**X**	
Part of								**OK**
Reverse					**X**	**X**	**X**	
Other than	**OK**	**OK**	**OK**	**OK**	**OK**	**OK**	**OK**	**OK**
Early					**X**	**X**		
Late	**OK**	**OK**	**X**	**X**	**X**	**X**	**OK**	
Before					**X**	**X**		
After					**X**	**X**		

8. Conclusions

8.1 FMEA and HAZOP ensures adequate test coverage

To highlight the conclusions of this work a Systems Dynamics Model is used to visualize the interaction of contributions to system test specification. As illustrated below, a simple example is used to show how the procedure recommended in this brochure offers the possibility to minimize the number of misconfigurations that degrade protection function operation.



Picture 28 – Interactions of Contributions to System Test Specification

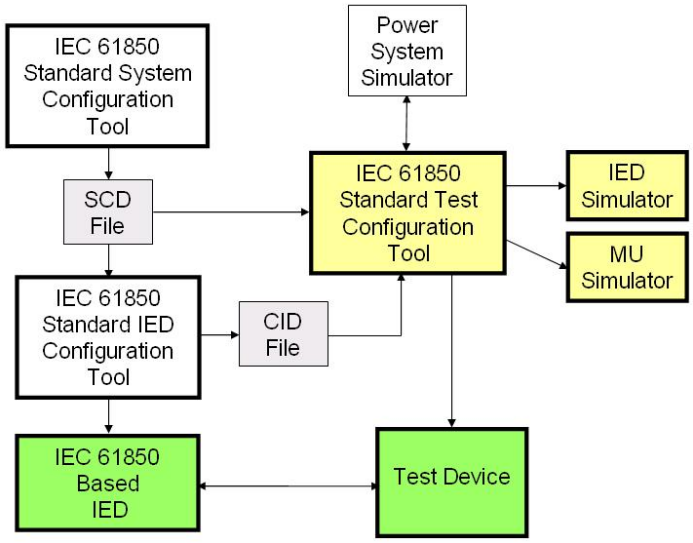
A well defined protection scheme must be described by the utility protection engineer. Such a scheme defines the number of protection functions to be implemented. Each protection function requires proper setting of the protection configuration parameters which is strongly influenced by the efficiency of setting these parameters using quality system configuration tools. Because of the need for protection configuration assurance, IEC 61850 defines the requirements for a system configuration language to be used by these tools. These contributions are shown in the lower causal loop shown in the figure above.

The counterclockwise upper causal loop integrates the recommendations of this brochure to minimize the number of misconfigurations. Based on FMEA defined faults, the potential for misconfiguration is translated into protection function misoperation. Using the HAZOP tables described in Chapter 7, test coverage should be improved. In turn, the need for protection configuration assurance is enhanced and the upper loop is closed by applying the FMEA and HAZOP analysis.

When system tests are run, GOOSE state change will result in changes to the operating parameters of the protection system. The results of these tests provide the means to measure the sensitivity of protection system operating reliability to the settings selected for the protection functions.

8.2 System configuration tools are the key to success

System configuration tools are the key to successful system test planning, execution and evaluation. Chapter 5 provides several examples of how these tools are used to. Automation of the test processes are needed to minimize labor intensive tasks and perform the tests in a timely manner. This is best illustrated in the figure below.



Picture 29 – System Configuration Tools

IEC 61850 system configuration tools and IED configuration tools are used to generate the Substation Configuration Description (SCD) file and Configured IED Description (CID) file respectively. Both files are needed by the standard test configuration tool proposed by WG B5.32 in this brochure.

8.3 UML provides the ability to manage system level complexity

To manage system level complexity, the functional testing should be viewed as “black-box” testing. That is the System Under Test (SUT) is characterized as inputs, outputs and the behavior of the transfer function of the SUT. The inputs, including the environment, are the test stimulus and the outputs are used to measure the correctness of the transfer function. WG B5.32 found that the best approach to understand the complexity of 61850 system functionality is use the Unified Modeling Language (UML). UML class diagrams and transaction sequences should be derived from the functional decomposition of the protection and automation functions under consideration.

WG B5.32 recommends using the eXtensible Markup Language (XML) to automate test specification checklist and the system functional test setup. Several examples are included to show how this approach is applied. Table forms of FMEA and HAZOP are incorporated in these examples.

8.4 Test specification template is the formal checklist

Although many test specification templates are available, a tailored template is needed to provide automated (machine readable) integration of IEC 61850 configuration tool outputs. For this reason, WG B5.32 designed a test specification template to be used as a formal checklist to address all features of the system test planning, execution and evaluation. Particular attention was given to tailoring the following fields.

- Each test case contains a unique identifier to track all inputs to the test specification.
- Definitions, acronyms and abbreviations are matched to IEC 61850 nomenclature.
- Test items are matched to the tools described in Chapter 5.
- Features to be tested and pass/fail criteria are matched to the output of FMEA and HAZOP describing what is tested and what is not tested.

8.5 Next steps for future CIGRE SC B5 research

Because SC B5 wanted first to address the basic technical approach to system level functional testing, the scope of WG B5.32 work was intentionally narrow. Technical issues and other system level functional test methods which exceeded the scope of this work provide the recommendations for future CIGRE SC B5 research. Listed below is a summary of the recommended research initiatives.

- Develop functional specifications for typical functions based on the templates provided in this brochure.
    - Develop example implementations in SCL for each case.
    - Develop functional test script in XML for each case.
- Propose to IEC TC57 WG10 to standardize the templates and XML for system level testing.

Appendice A. Functional Specification Template

<The following template is provided for use with IEC 61850 based functional specification, suggested to be included in procurement or test specification for Substation Automation Systems. Text enclosed in square brackets is included to provide guidance to the functional specification author and should be substituted by normal text before publishing the document>

A.1 Revision History

<The following table should register the main revision of this document >

Table 37 – Functional Use Case Revision History

Date	Version	Description	Author
<dd/mmm/yy>	<x.x>	<details>	<name>

A.2 Introduction

[The introduction of the Functional Requirements Specification (FRS) provides an overview of the entire FRS. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the FRS.]

[Note: The FRS document captures the complete functional requirements for a Substation Automation System, or a portion of it. Following is a typical FRS outline for a project using UML Use Case modeling.

Purpose

[Specify the purpose of this FRS. The FRS fully describes the external behavior of the application or subsystem identified. It also may describe nonfunctional requirements, design constraints, and other factors necessary to provide a complete and comprehensive description of the functional requirements for the Substation Automation System.]

Scope

[A brief description of the SAS application that the FRS applies to, the feature or other subsystem grouping, what Use-Case model(s) it is associated with, and anything else that is affected or influenced by this document.]

Definitions, Acronyms, and Abbreviations

[This subsection provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the FRS. This information may be provided by reference to a project's Glossary.]

References

[This subsection provides a complete list of all documents referenced elsewhere in the FRS. Identify each document by title, report number if applicable, date, and publishing organization. Specify the sources from which the references can be obtained. This information may be provided by reference to an appendix or to another document.]

Overview

[This subsection describes what the rest of the FRS contains and explains how the document is organized.]

A.3 Overall Description

[This section of the FRS describes the general factors that affect the SAS and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in detail in Section 3, and makes them easier to understand. Include such items as:

- project perspective
- project functions
- user characteristics
- constraints
- assumptions and dependencies
- requirements subsets].

A.4 Functional Requirements

[This section of the FRS contains all functional requirements to a level of detail sufficient to enable (designers to design a SAS to satisfy those requirements, and) testers to test that the system satisfies those requirements.]

[For many applications, this may constitute the bulk of the FRS package and thought should be given to the organization of this section. This section is typically organized by function types like protection, supervision, interlocking, etc.

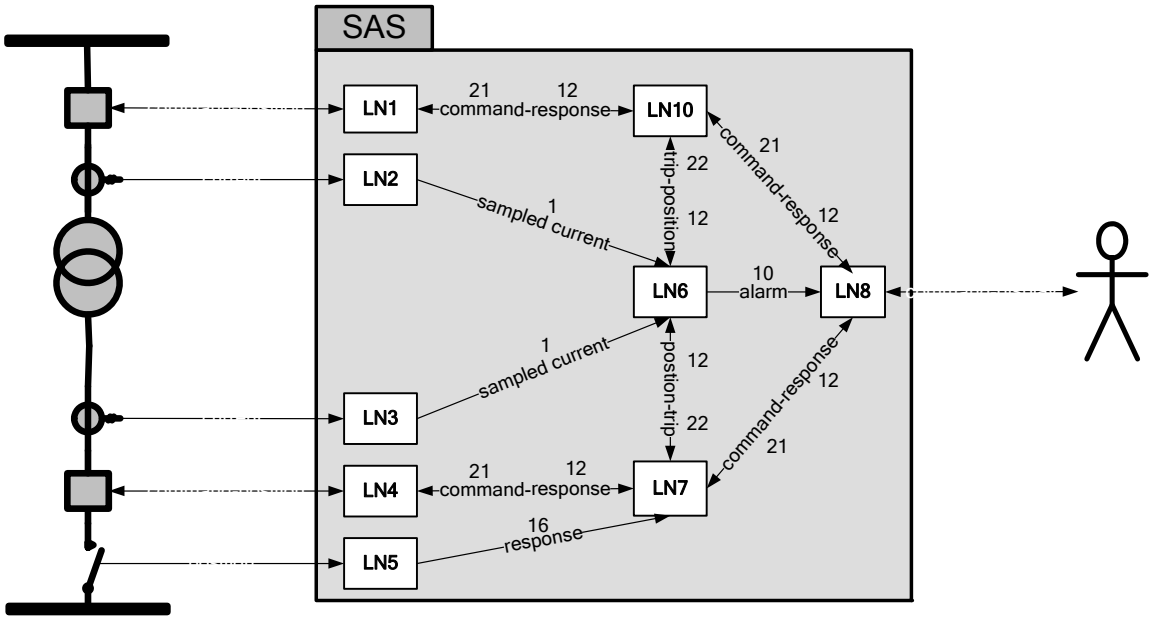
Where application development tools, such as requirements tools, modeling tools, and the like, are employed to capture the functionality, this section of the document would refer to the availability of that data, indicating the location and name of the tool used to capture the data.

Typically, the requirement would contain the following items, for each function:

- <Inputs of SAS functions – where signals come from>

- <Outputs of SAS functions – where signals go to>
- <Exceptions of SAS functions – what if it happens>
- <Processing of SAS functions – which nodes transform signals>
- <UML Object Diagram of Logical Nodes – may be provided in the FICS>
- <UML Messages (PICOM) among Objects – annotated on the diagram>

[The following picture shows a UML object communication diagram showing the logical nodes, PICOM types and messages exchanged in a typical SAS function]



Picture 30 – UML Communication Diagram

[The system's performance characteristics are also outlined in this section, associated to each function, like:

- Response time for a transaction (trip, alarm, etc.) (average, maximum)
- Throughput, for example, transactions or messages per second
- Capacity, for example, the number of transactions (alarms, messages, etc) the system can accommodate
- Degradation modes (what is the acceptable mode of operation when the system has been degraded in some manner (like loss of a switch, etc)
- Resource utilization, such as memory, disk, communications, and so forth, as applicable.

Vendors and system integrators are expected to supply a **FICS – Function Implementation Conformance Statement** – that summarizes the functional capabilities of the system or device to be tested, and a detailed object model supported by the product;

[FICS and functional requirements can be expressed in a typical UML Use Case, according to the following model, for each specified function]

Table 38 – Functional Use Case Template

Functional Implementation Conformance Statement			
Code	Short distinctive name of the function from the SAS		
Name	Phrase that declares the main objective of the function		
Description	Longer description or summary of the function objective		
Customer	Identification of owner or integrator of substation		
Substation	Identification of substation		
SCL File	Substation configuration language file and version, if available		
Primary User (Actor)	Role name or description of the primary functional actor/user of the use case among people, system, etc. (ex. Operation, Engineering, Process, Dispatch)		
Secondary User (Actor)	Role name or description of the secondary functional actor/user of the use case among people, system, etc. (ex. Operation, Engineering, Process, Dispatch)		
Stakeholder & Interest	Name of the stakeholder and interest of the stakeholder in the use case		
Function Description			
Trigger	Which action(s)/event(s) of the primary/secondary users initiate the use case		
Components or Logical Nodes	(Codes of) Component(s) architecture or Logical Node(s) that implement or realize the function or use case, taken from the SCL file, if available		
Process Equipments	(Codes of) Associated substation process equipments, affected by the function		
Performance	Goal or quantification of function objective (Ex. execution time, records accuracy, etc.)		
Preconditions	Expected state of the automation system, substation or its environment before the use case may be applied (Ex. Closed breakers, switches, etc.)		
Post conditions on Success	Expected state of the automation system, substation or its environment after successful completion the use case (Ex. Tripped breakers, alarms, recordings, etc.)		
Post conditions on Failure	Expected state of the automation system, substation or its environment after unsuccessful completion the use case (Ex. Breaker failure, etc.)		
Use Case Description			
Basic Course Description	Flow of events performed during normal state		
	1	Actor	Event, step or condition of successful execution
	2	System	Event, step or condition of successful execution
	...	**...**	**...**
	N	System	Event, step or condition of successful execution
Extensions	Alternative flow of events performed during normal state		
	1	Actor	Event, step or condition of successful execution
	2	System	Event, step or condition of successful execution
	...	**...**	**...**
	N	System	Event, step or condition of successful execution

Appendice B. Functional Test Specification Template

<The following template is provided for use with IEC 61850 based functional test specification. Text enclosed in square brackets is included to provide guidance to the functional test specification author and should be substituted by normal text before publishing the document>

B.1 Revision History

<The following table should register the main revision of this document >

Table 39 – Functional Test Specification Revision History

Date	Version	Description	Author
<dd/mmm/yy>	<x.x>	<details>	<name>

B.2 Introduction

[The introduction of the Functional Test Specification (FTS) provides an overview of the entire FTS. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the FTS.]

[Note: The FTS document captures the complete functional test specification for a Substation Automation System, or a portion of it, using UML Testing Profile and object oriented testing techniques.

Purpose

[Specify the purpose of this FTS. The FTS fully describes the external testing of the application or subsystem identified by a corresponding FRS (Functional Requirement Specification) or FICS (Functional Implementation Conformance Statement). It also may describe nonfunctional test requirements, design constraints, and other factors necessary to provide a complete and comprehensive testing of the functions of the Substation Automation System.]

Scope

[A brief description of the SAS application and its FRS/FICS, that the FTS applies to, the feature or other subsystem grouping, what Use-Case model(s) it is associated with, and anything else that is affected or influenced by this document. Explicit mention should be made of the exclusion of conformance and interoperability tests, and other environmental tests, as applicable]

Definitions, Acronyms, and Abbreviations

[This subsection provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the FTS. This information may be provided by reference to a project's Glossary, and to this Cigré document.]

References

[This subsection provides a complete list of all documents referenced elsewhere in the FTS, specially the FRS that it applies to. Identify each document by title, report number if applicable, date, and publishing organization. Specify the sources from which the references can be obtained. This information may be provided by reference to an appendix or to another document.]

Overview

[This subsection describes what the rest of the FTS contains and explains how the document is organized.]

B.3 Overall Description

[This section of the FTS describes the general factors that affect the SAS testing and its requirements. This section does not state specific test requirements. Instead, it provides a background for those requirements, which are defined in detail in Section 3, and makes them easier to understand. Include such items as:

- project perspective
- project functions
- user characteristics
- constraints
- assumptions and dependencies
- requirements subsets]

B.4 Functional Test Specifications

[This section of the FTS contains all functional test specifications to a level of detail sufficient to enable testers to test a SAS to satisfy a specific FRS/FICS.]

[For many applications, this may constitute the bulk of the FTS package and thought should be given to the organization of this section. This section is typically organized following the FRS/FICS, by function types like protection, supervision, interlocking, etc.

Where testing development tools, such as simulator tools, modeling tools, and the like, are employed to capture test functionality, this section of the document would refer to the availability of that data, indicating the location and name of the tool used to capture the data.

<Typically, the test specification would contain the following items, for each tested function:>

- <Processing of SAS functions – which nodes transform signals>
- <UML Object Diagram of Logical Nodes – may be provided in the FICS>
- <UML Messages (PICOM) among Objects – annotated on the diagram>
- <**Test script** containing a:
  - **Test Connection** – Create all test components and connect to SUT
  - **Test Setup** – Define all parameters and set points of test components
  - **Test Start** – Apply signals to SUT
  - **Test Stop** – Suspend signals to SUT
  - **Test Disconnection** – Close test components and evaluate statistics
  - **Test Verdict** – Emit result of test case

[Vendors and system integrators are expected to supply a **FICS – Function Implementation Conformance Statement** – that summarizes the functional capabilities of the system or device to be tested, and a detailed object model supported by the product;]

[Functional test specifications can be expressed in a table, according to the following model, for each tested function]

Table 40 – Functional Test Case Template

Functional Test Case		
Code	Short distinctive name of the function from the SAS	
Name	Phrase that declares the main objective of the function	
Description	Longer description or summary of the function objective and its test	
Use Case Description		
Customer	Identification of owner or integrator of substation	
Substation	Identification of substation	
SCL File	Substation configuration language file and version, if available	
FSR File	Functional Specification Requirement file and version	
FICS File	Function Implementation Conformance Statement by SAS vendor or supplier	
Test Description		
Step	**Command**	**Meaning**
Test Connection		
1	Call test component constructor 1	Create a test component and connect it to SUT
2	Call test component constructor 2	Create a test component and connect it to SUT
N	…	…
Test Setup		
1	Set parameters to test component 1	Prepare test component to apply signal to SUT
2	Set parameters to test component 2	Prepare test component to apply signal to SUT
N	…	…
Test Start		
1	Start test component 1	Apply test component signals to SUT
2	Start test component 2	Apply test component signals to SUT
N	…	…
Test Stop		
1	Stop test component 1	Suspend test component signals to SUT
2	Stop test component 2	Suspend test component signals to SUT
N	…	…
Test Disconnection		
1	Log test component 1 results	Register results of test component from SUT
2	Log test component 2 results	Register results of test component from SUT

N	…	…
Test Verdict		
1	Check log results from component 1	Publish test component verdict
2	Check log results from component 2	Publish test component verdict
N	…	…

B.5 Test Coverage

[This section of the FTS contains a demonstration of test coverage of failure modes and functional failures.]

[The section is typically organized following the FRS, by function types like protection, supervision, interlocking, etc., but can also condense the coverage of a whole test suite with several test cases.]

[Functional test coverage can be expressed in a series of tables, according to the following models, for each tested function.

[HAZOP (Hazard and Operability Analysis), following IEC 61882 standard, is suggested as a systematic mean of description of failure modes. A table for one or several logical nodes can resume the meaning of each HAZOP guide word as applied to their status, measuring, control and setting functions.]

Table 41 – HAZOP Guide Word Meaning for Logical Nodes

Guide Word	Status	Measures	Controls	Settings
No				
More				
Less				
As well as				
Part of				
Reverse				
Other than				
Early				
Late				
Before				
After				

[Or a summary table to condense the failure modes of all Logical Nodes involved in one or several functions to be tested, if the meaning of each failure mode is evident from the guide word].

Table 42 – Logical Nodes Failure Modes

Guide Word	LN1	LN2	…	LNn
No			…	
More	X		…	X
Less			…	X
As well as		X	…	

Part of			…	
Reverse	X	X	…	
Other than			…	X
Early	X	X	…	
Late	X		…	
Before		X	…	X
After	X		…	X

[Finally, a coverage table detailing the failure modes tested by one or several test cases, signaled on the previous table by the label OK, can be included to demonstrate the test coverage.]

Table 43 – Functional Test Coverage

Guide Word	LN1	LN2	…	LNn
No			…	
More	OK		…	X
Less			…	X
As well as		X	…	
Part of			…	
Reverse	X	OK	…	
Other than			…	OK
Early	X	X	…	
Late	OK		…	
Before		X	…	X
After	X		…	X

[Test coverage can also be shown for a whole test suite, for all functional failure and failure modes tested in an FMEA summary table.]

Table 44 – FMEA Test Coverage

FMEA		FAILURE MODE				
		M1	M2	M3	...	Mn
FUNCTIONAL FAILURE	F1			X	...	
	F2		X		...	X
	F3			X	...	
	...	X		X	...	X
	Fm				...	
COVERAGE		X	X	X	...	...
TEST CASE	T1				...	
	T2		X	X	...	X
	T3				...	
	...			X	...	
	Tm		X		...	X

Appendice C. XML Schema for Functional Test

C.1 Introduction

The following XML Schema is provided for use with IEC 61850 based functional test design, in procurement or test specification processes.

C.2 Revision History

Table 45 – Functional XMLTest Revision History

Date	Version	Description	Author
02/16/2007	1.0	First Meeting of Cigré Task Force B5.92	Cigré TF B5.92

C.3 Purpose

This XML Schema defines the syntax rules for designing functional tests of IEC 61850 based systems, according to the test architecture described in this brochure. It does not describe nonfunctional test requirements, conformance and interoperability tests, design constraints, and other factors necessary to provide a complete and comprehensive testing of the functions of a Substation Automation System.

C.4 Schema Organization

The main classes of the test architecture and their methods are defined, and identified by comments on the XML Schema. Each possible method is also defined as an XML element, with attributes according to the parameters defined for the corresponding method of the class of the test architecture. A complete example of the use of this schema is included on Chapter 7.

C.5 XML Schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="http://tempuri.org/XMLSchema.xsd"
           elementFormDefault="qualified"
           xmlns="http://tempuri.org/XMLSchema.xsd"
           xmlns:mstns="http://tempuri.org/XMLSchema.xsd"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!--=========================================================-->
  <!--Possible Test Case results, from UML Test Profile-->
  <!--=========================================================-->
  <xs:simpleType name="verdict">
    <xs:restriction base="xs:NCName">
      <xs:enumeration value="pass"/> <!--The system under test adheres to
the expectations-->
```

```
      <xs:enumeration value="fail"/> <!--The system under test differs from
the expectation-->
      <xs:enumeration value="inconclusive"/> <!--The evaluation cannot be
evaluated to be pass or fail-->
      <xs:enumeration value="error"/> <!--An error has occurred within the
testing environment-->
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="TestCase">
    <xs:complexType>
      <xs:sequence>
        <!--=========================================================-->
        <!--Test Case Identificacion-->
        <!--=========================================================-->
        <xs:element name="Code" type="xs:string" />
        <xs:element name="Name" type="xs:string" />
        <xs:element name="Description" type="xs:string" />
        <xs:element name="Customer" type="xs:string" />
        <xs:element name="Substation" type="xs:string" />
        <xs:element name="SCLFile" type="xs:string" />
        <xs:element name="FICSFile" type="xs:string" />
        <xs:element name="Script">
          <!--=========================================================-->
          <!--Test Case Script-->
          <!--=========================================================-->
          <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <!--=========================================================-->
              <!--Digital Output Simulator-->
              <!--=========================================================-->
              <!--Constructor of a digital output DigitalOutputName
connected to Node -->
              <xs:element name="DigitalOutput">
                <xs:complexType>
                  <xs:attribute name="DigitalOutputName" type="xs:ID" />
                  <xs:attribute name="Node" type="xs:string" />
                </xs:complexType>
              </xs:element>
              <!--Read a Comtrade File with digital output for Node -->
              <xs:element name="SetDigitalOutputSequence">
                <xs:complexType>
                  <xs:attribute name="DigitalOutputName" type="xs:IDREF" />
                  <xs:attribute name="File" type="xs:string" />
                </xs:complexType>
              </xs:element>
              <!--Record digital output sequence for Duration from Node -->
              <xs:element name="GetDigitalOutputSequence">
                <xs:complexType>
                  <xs:attribute name="DigitalOutputName" type="xs:IDREF" />
                  <xs:attribute name="Duration" type="xs:duration" />
                </xs:complexType>
              </xs:element>
              <!--Set a Boolean output to Node -->
              <xs:element name="SetDigitalOutput">
                <xs:complexType>
                  <xs:attribute name="DigitalOutputName" type="xs:IDREF" />
                  <xs:attribute name="Boolean" type="xs:boolean" />
                </xs:complexType>
              </xs:element>
```

```
            <!--Read current Boolean output to Node -->
            <xs:element name="GetDigitalOutput">
              <xs:complexType>
                <xs:attribute name="DigitalOutputName" type="xs:IDREF" />
                <xs:attribute name="Boolean" type="xs:boolean" />
              </xs:complexType>
            </xs:element>
            <!--Start digital output (set before) to Node and return Time
-->
            <xs:element name="StartDigitalOutput">
              <xs:complexType>
                <xs:attribute name="DigitalOutputName" type="xs:IDREF" />
                <xs:attribute name="Time" type="xs:time" />
              </xs:complexType>
            </xs:element>
            <!--Stop digital output to Node and return Time -->
            <xs:element name="StopDigitalOutput">
              <xs:complexType>
                <xs:attribute name="DigitalOutputName" type="xs:IDREF" />
                <xs:attribute name="Time" type="xs:time" />
              </xs:complexType>
            </xs:element>
            <!--Get Time of first positive transition from Node in File -
->
            <xs:element name="FirstUpOutputTransition">
              <xs:complexType>
                <xs:attribute name="DigitalOutputName" type="xs:IDREF" />
                <xs:attribute name="Time" type="xs:string" />
              </xs:complexType>
            </xs:element>
            <!--Get Time of first negative transition from Node in File -
->
            <xs:element name="FirstDownOutputTransition">
              <xs:complexType>
                <xs:attribute name="DigitalOutputName" type="xs:IDREF" />
                <xs:attribute name="Time" type="xs:string" />
              </xs:complexType>
            </xs:element>
            <!--Get Time of last positive transition from Node in File --
>
            <xs:element name="LastUpOutputTransition">
              <xs:complexType>
                <xs:attribute name="DigitalOutputName" type="xs:IDREF" />
                <xs:attribute name="Time" type="xs:string" />
              </xs:complexType>
            </xs:element>
            <!--Get Time of last negative transition from Node in File --
>
            <xs:element name="LastDownOutputTransition">
              <xs:complexType>
                <xs:attribute name="DigitalOutputName" type="xs:IDREF" />
                <xs:attribute name="Time" type="xs:string" />
              </xs:complexType>
            </xs:element>
            <!--=====================================================-->
            <!--Digital Input Simulator-->
            <!--=====================================================-->
            <!--Constructor of a digital input DigitalInputName connected
to Node -->
```

```
                <xs:element name="DigitalInput">
                  <xs:complexType>
                    <xs:attribute name="DigitalInputName" type="xs:ID" />
                    <xs:attribute name="Node" type="xs:string" />
                  </xs:complexType>
                </xs:element>
                <!--Record input sequence in File for Duration from Node -->
                <xs:element name="GetDigitalIinputSequence">
                  <xs:complexType>
                    <xs:attribute name="DigitalInputName" type="xs:IDREF" />
                    <xs:attribute name="Duration" type="xs:duration" />
                  </xs:complexType>
                </xs:element>
                <!--Read current Boolean input from Node -->
                <xs:element name="GetDigitalInput">
                  <xs:complexType>
                    <xs:attribute name="DigitalInputName" type="xs:IDREF" />
                    <xs:attribute name="Boolean" type="xs:boolean" />
                  </xs:complexType>
                </xs:element>
                <!--Get Time of first positive transition from Node in File -
->
                <xs:element name="FirstUpInputTransition">
                  <xs:complexType>
                    <xs:attribute name="DigitalInputName" type="xs:IDREF" />
                    <xs:attribute name="Time" type="xs:string" />
                  </xs:complexType>
                </xs:element>
                <!--Get Time of first negative transition from Node in File -
->
                <xs:element name="FirstDownInputTransition">
                  <xs:complexType>
                    <xs:attribute name="DigitalInputName" type="xs:IDREF" />
                    <xs:attribute name="Time" type="xs:string" />
                  </xs:complexType>
                </xs:element>
                <!--Get Time of last positive transition from Node in File --
>
                <xs:element name="LastUpInputTransition">
                  <xs:complexType>
                    <xs:attribute name="DigitalInputName" type="xs:IDREF" />
                    <xs:attribute name="Time" type="xs:string" />
                  </xs:complexType>
                </xs:element>
                <!--Get Time of last negative transition from Node in File --
>
                <xs:element name="LastDownInputTransition">
                  <xs:complexType>
                    <xs:attribute name="DigitalInputName" type="xs:IDREF" />
                    <xs:attribute name="Time" type="xs:string" />
                  </xs:complexType>
                </xs:element>
                <!--=====================================================-->
                <!--Voltage Output Simulator-->
                <!--=====================================================-->
                <!--Constructor of a voltage output object connected to Node
-->
                <xs:element name="VoltageOutput">
                  <xs:complexType>
```

```
      <xs:attribute name="VoltageOutputName" type="xs:ID" />
      <xs:attribute name="Node" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <!--Read a Comtrade File with Voltage output for Node -->
  <xs:element name="SetVoltageOutputSequence">
    <xs:complexType>
      <xs:attribute name="VoltageOutputName" type="xs:IDREF" />
      <xs:attribute name="File" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <!--Record Voltage output sequence for Duration from Node -->
  <xs:element name="GetVoltageOutputSequence">
    <xs:complexType>
      <xs:attribute name="VoltageOutputName" type="xs:IDREF" />
      <xs:attribute name="Duration" type="xs:duration" />
    </xs:complexType>
  </xs:element>
  <!--Set an AC Phasor voltage output to Node -->
  <xs:element name="SetACVoltageOutput">
    <xs:complexType>
      <xs:attribute name="VoltageOutputName" type="xs:IDREF" />
      <xs:attribute name="Module" type="xs:float" />
      <xs:attribute name="Angle" type="xs:float" />
    </xs:complexType>
  </xs:element>
  <!--Read current AC Phasor voltage output to Node -->
  <xs:element name="GetACVoltageOutput">
    <xs:complexType>
      <xs:attribute name="VoltageOutputName" type="xs:IDREF" />
      <xs:attribute name="Module" type="xs:float" />
      <xs:attribute name="Angle" type="xs:float" />
    </xs:complexType>
  </xs:element>
  <!--Set a DC Module voltage output to Node -->
  <xs:element name="SetDCVoltageOutput">
    <xs:complexType>
      <xs:attribute name="VoltageOutputName" type="xs:IDREF" />
      <xs:attribute name="Module" type="xs:float" />
    </xs:complexType>
  </xs:element>
  <!--Get current DC Module voltage output to Node -->
  <xs:element name="GetDCVoltageOutput">
    <xs:complexType>
      <xs:attribute name="VoltageOutputName" type="xs:IDREF" />
      <xs:attribute name="Module" type="xs:float" />
    </xs:complexType>
  </xs:element>
  <!--Start voltage output (set before) to Node and return Time
-->
  <xs:element name="StartVoltageOutput">
    <xs:complexType>
      <xs:attribute name="VoltageOutputName" type="xs:IDREF" />
      <xs:attribute name="Time" type="xs:time" />
    </xs:complexType>
  </xs:element>
  <!--Stop voltage output to Node and return Time -->
  <xs:element name="StopVoltageOutput">
    <xs:complexType>
```

```
      <xs:attribute name="VoltageOutputName" type="xs:IDREF" />
      <xs:attribute name="Time" type="xs:time" />
    </xs:complexType>
  </xs:element>
  <!--=====================================================-->
  <!--Current Output Simulator-->
  <!--=====================================================-->
  <!--Constructor of a Current output object connected to Node
-->
  <xs:element name="CurrentOutput">
    <xs:complexType>
      <xs:attribute name="CurrentOutputName" type="xs:ID" />
      <xs:attribute name="Node" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <!--Read a Comtrade File with Current output for Node -->
  <xs:element name="SetCurrentOutputSequence">
    <xs:complexType>
      <xs:attribute name="CurrentOutputName" type="xs:IDREF" />
      <xs:attribute name="File" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <!--Record Current output sequence for Duration from Node -->
  <xs:element name="GetCurrentOutputSequence">
    <xs:complexType>
      <xs:attribute name="CurrentOutputName" type="xs:IDREF" />
      <xs:attribute name="Duration" type="xs:duration" />
    </xs:complexType>
  </xs:element>
  <!--Set an AC Phasor Current output to Node -->
  <xs:element name="SetACCurrentOutput">
    <xs:complexType>
      <xs:attribute name="CurrentOutputName" type="xs:IDREF" />
      <xs:attribute name="Module" type="xs:float" />
      <xs:attribute name="Angle" type="xs:float" />
    </xs:complexType>
  </xs:element>
  <!--Read current AC Phasor Current output to Node -->
  <xs:element name="GetACCurrentOutput">
    <xs:complexType>
      <xs:attribute name="CurrentOutputName" type="xs:IDREF" />
      <xs:attribute name="Module" type="xs:float" />
      <xs:attribute name="Angle" type="xs:float" />
    </xs:complexType>
  </xs:element>
  <!--Set a DC Module Current output to Node -->
  <xs:element name="SetDCCurrentOutput">
    <xs:complexType>
      <xs:attribute name="CurrentOutputName" type="xs:IDREF" />
      <xs:attribute name="Module" type="xs:float" />
    </xs:complexType>
  </xs:element>
  <!--Get current DC Module Current output to Node -->
  <xs:element name="GetDCCurrentOutput">
    <xs:complexType>
      <xs:attribute name="CurrentOutputName" type="xs:IDREF" />
      <xs:attribute name="Module" type="xs:float" />
    </xs:complexType>
  </xs:element>
```

```
            <!--Start Current output (set before) to Node and return Time
-->
            <xs:element name="StartCurrentOutput">
              <xs:complexType>
                <xs:attribute name="CurrentOutputName" type="xs:IDREF" />
                <xs:attribute name="Time" type="xs:string" />
              </xs:complexType>
            </xs:element>
            <!--Stop Current output to Node and return Time -->
            <xs:element name="StopCurrentOutput">
              <xs:complexType>
                <xs:attribute name="CurrentOutputName" type="xs:IDREF" />
                <xs:attribute name="Time" type="xs:time" />
              </xs:complexType>
            </xs:element>
            <!--=======================================================-->
            <!--Network Simulator-->
            <!--=======================================================-->
            <!--Construct a network simulator object connected to Node --
>
            <xs:element name="NetworkSimulator">
              <xs:complexType>
                <xs:attribute name="NetworkSimulatorName" type="xs:ID" />
                <xs:attribute name="Node" type="xs:string" />
              </xs:complexType>
            </xs:element>
            <!--Read a File with network messages to and from Node -->
            <xs:element name="SetMessageSequence">
              <xs:complexType>
                <xs:attribute name="NetworkSimulatorName" type="xs:IDREF"
/>
                <xs:attribute name="File" type="xs:string" />
              </xs:complexType>
            </xs:element>
            <!--Record in File messages for Duration to and from Node -->
            <xs:element name="GetMessageSequence">
              <xs:complexType>
                <xs:attribute name="NetworkSimulatorName" type="xs:IDREF"
/>
                <xs:attribute name="Duration" type="xs:duration" />
              </xs:complexType>
            </xs:element>
            <!--Read a File at Interval with messages to and from Node --
>
            <xs:element name="RepeatMessageSequence ">
              <xs:complexType>
                <xs:attribute name="NetworkSimulatorName" type="xs:IDREF"
/>
                <xs:attribute name="File" type="xs:string" />
                <xs:attribute name="Interval" type="xs:duration" />
              </xs:complexType>
            </xs:element>
            <!--Start network messages to and from Node & return Time -->
            <xs:element name="StartNetworkSimulator">
              <xs:complexType>
                <xs:attribute name="NetworkSimulatorName" type="xs:IDREF"
/>
                <xs:attribute name="Time" type="xs:time" />
              </xs:complexType>
```

```
                </xs:element>
                <!--Stop network messages to and from Node & return Time -->
                <xs:element name="StopNetworkSimulator">
                  <xs:complexType>
                    <xs:attribute name="NetworkSimulatorName" type="xs:IDREF"
/>
                    <xs:attribute name="Time" type="xs:time" />
                  </xs:complexType>
                </xs:element>
                <!--Get Time of first Picom from FromNode to Node in File -->
                <xs:element name="FirstPICOMFrom ">
                  <xs:complexType>
                    <xs:attribute name="NetworkSimulatorName" type="xs:IDREF"
/>
                    <xs:attribute name="FromNode" type="xs:string" />
                    <xs:attribute name="Picom" type="xs:string" />
                    <xs:attribute name="Time" type="xs:string" />
                  </xs:complexType>
                </xs:element>
                <!--Get Time of last Picom from FromNode to Node in File -->
                <xs:element name="LastPICOMFrom">
                  <xs:complexType>
                    <xs:attribute name="NetworkSimulatorName" type="xs:IDREF"
/>
                    <xs:attribute name="FromNode" type="xs:string" />
                    <xs:attribute name="Picom" type="xs:string" />
                    <xs:attribute name="Time" type="xs:string" />
                  </xs:complexType>
                </xs:element>
                <!--Get Time of first Picom from Node to ToNode in File -->
                <xs:element name="FirstPICOMTo">
                  <xs:complexType>
                    <xs:attribute name="NetworkSimulatorName" type="xs:IDREF"
/>
                    <xs:attribute name="ToNode" type="xs:string" />
                    <xs:attribute name="Picom" type="xs:string" />
                    <xs:attribute name="Time" type="xs:string" />
                  </xs:complexType>
                </xs:element>
                <!--Get Time of last Picom from Node to ToNode in File -->
                <xs:element name="LastPICOMTo">
                  <xs:complexType>
                    <xs:attribute name="NetworkSimulatorName" type="xs:IDREF"
/>
                    <xs:attribute name="ToNode" type="xs:string" />
                    <xs:attribute name="Picom" type="xs:string" />
                    <xs:attribute name="Time" type="xs:string" />
                  </xs:complexType>
                </xs:element>
                <!--=====================================================-->
                <!--Operator Simulator-->
                <!--=====================================================-->
                <!--Constructor of an operator object at a station in Node --
>
                <xs:element name="Operator">
                  <xs:complexType>
                    <xs:attribute name="OperatorName" type="xs:ID" />
                    <xs:attribute name="Node" type="xs:string" />
                  </xs:complexType>
```

```
                </xs:element>
                <!--Perform and confirm manual Action at staton in Node -->
                <xs:element name="OperatorAct">
                  <xs:complexType>
                    <xs:attribute name="OperatorName" type="xs:IDREF" />
                    <xs:attribute name="Action" type="xs:string" />
                    <xs:attribute name="Verdict" type="xs:string" />
                  </xs:complexType>
                </xs:element>
                <!--Confirm automatic Action by SAS at station in Node -->
                <xs:element name="OperatorConfirm">
                  <xs:complexType>
                    <xs:attribute name="OperatorName" type="xs:IDREF" />
                    <xs:attribute name="Action" type="xs:string" />
                    <xs:attribute name="Verdict" type="xs:string" />
                  </xs:complexType>
                </xs:element>
                <!--=======================================================-->
                <!--Test Arbiter-->
                <!--=======================================================-->
                <!--Constructor of an arbiter object named TestArbiterName --
>
                <xs:element name="TestArbiter">
                  <xs:complexType>
                    <xs:attribute name="TestArbiterName" type="xs:ID" />
                  </xs:complexType>
                </xs:element>
                <!--Evaluate and return value of Expression as a Verdict -->
                <xs:element name="TestArbiterConfirm">
                  <xs:complexType>
                    <xs:attribute name="TestArbiterName" type="xs:IDREF" />
                    <xs:attribute name="Expression" type="xs:string" />
                    <xs:attribute name="Verdict" type="xs:string" />
                  </xs:complexType>
                </xs:element>
                <!--=======================================================-->
                <!--Test Timer-->
                <!--=======================================================-->
                <!--Constructor of a timer object named TestTimerName -->
                <xs:element name="TestTimer">
                  <xs:complexType>
                    <xs:attribute name="TestTimerName" type="xs:ID" />
                  </xs:complexType>
                </xs:element>
                <!--Start timer counting -->
                <xs:element name="Start">
                  <xs:complexType>
                    <xs:attribute name="TestTimerName" type="xs:IDREF" />
                  </xs:complexType>
                </xs:element>
                <!--Stop timer counting -->
                <xs:element name="Stop">
                  <xs:complexType>
                    <xs:attribute name="TestTimerName" type="xs:IDREF" />
                  </xs:complexType>
                </xs:element>
                <!--Get current time count -->
                <xs:element name="GetTime">
                  <xs:complexType>
```

```
                <xs:attribute name="TestTimerName" type="xs:IDREF" />
                <xs:attribute name="Time" type="xs:string" />
              </xs:complexType>
            </xs:element>
            <!--=====================================================-->
            <!--Test Scheduler-->
            <!--=====================================================-->
            <!--Imports SCL file for Substation Automation System -->
            <xs:element name="ReadSCL">
              <xs:complexType>
                <xs:attribute name="File" type="xs:string" />
              </xs:complexType>
            </xs:element>
            <!--Imports XML Script file for testing -->
            <xs:element name="ReadScript">
              <xs:complexType>
                <xs:attribute name="Script" type="xs:string" />
              </xs:complexType>
            </xs:element>
            <!--Start running test case named Name -->
            <xs:element name="StartTestCase">
              <xs:complexType>
                <xs:attribute name="Name" type="xs:string" />
              </xs:complexType>
            </xs:element>
            <!--Stop further script processing for Duration ms -->
            <xs:element name="Wait ">
              <xs:complexType>
                <xs:attribute name="Duration" type="xs:duration" />
              </xs:complexType>
            </xs:element>
          </xs:choice>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Appendice D. UML Overview

D.1 Introduction

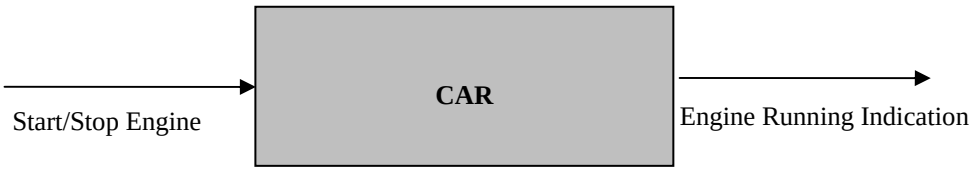
This Appendix is intended to introduce the reader to the concept of using different UML diagrams for describing different aspects of the same function, depending on the nature and requirements of the function. It is not meant to be the formal definition of which diagrams are used or the correct format of the diagrams.

D.2 Function Definition

As described in section 2.2, functions in an SAS involve processing of a set of input data to produce a set of output data. A function can be a simple process which is self-contained inside a single module, or it can be complex and involve several modules which communicate with each other to achieve the desired outputs of the function. UML diagrams are used to show the differing levels of complexity depending on the function being represented. This is done by different types of diagrams, each of which is designed to illustrate a specific aspect of the function. Not all diagrams are required for all functions.

The different uses of the diagrams can best be described by considering an everyday example familiar to many people, a person starting a car.

Expressed as a function, a car starting function has the following overview:



Picture 31 – Car Starting Funcition

To start the car the driver must activate the start engine input (usually an ignition key). The driver will maintain the activation of the start engine input until the engine is started, which is observed through some sort of indication – e.g. ignition light goes out – at which point the driver stops activating the start engine input. (We will assume that the car starts within a few seconds, there is adequate fuel and the battery does not lose its charge)

This function consists of a number of modules, such as the fuel system and the ignition system, working together and performing specific operations to achieve the desired output. Before we look at the types of UML diagram that may be used to describe the how these modules interact and what they do, we first need a way of describing the function itself. UML provides a suitable tool known as a USE CASE.

D.3 Use Case

A Use Case lists the functional steps that take place from the initial input to the final output. The steps make reference to modules within the function to show the overall interaction without itself describing how the modules actually relate to each other.

The Use Case below shows the car starting function.

Starting Car

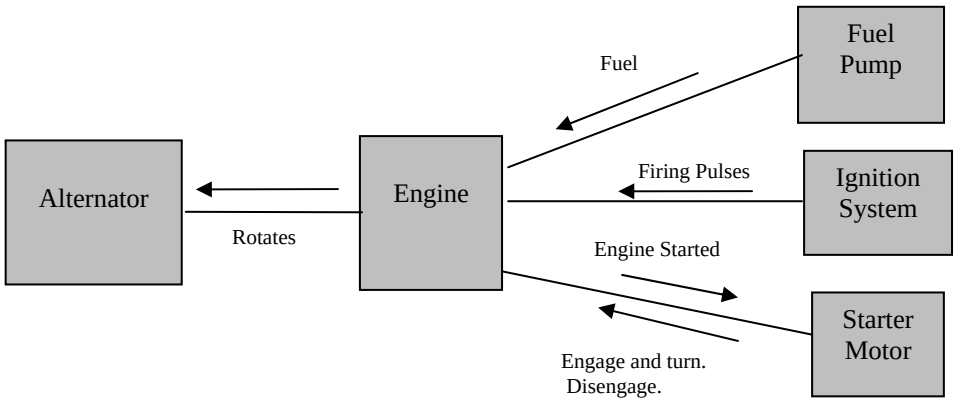
1. Driver turns ignition key and waits
2. Starter motor starts and engages with engine flywheel
3. Fuel pump starts
4. Ignition system starts
5. Engine starts
6. Alternator generates sufficient electricity to extinguish ignition light
7. Driver sees ignition light extinguish and releases the ignition key
8. Starter motor stops and disengages from flywheel

This use case shows that not only are there operations performed by the driver, but there are also operations performed by other parties – principally ‘Starter Motor’, ‘Fuel Pump’, ‘Engine’ and ‘Alternator’. All the parties are known as ACTORS in UML. The use case shows which actors are involved in a specific function and what each actor does in the execution of the function.

Note that use cases are not intended to identify all modules within the function, it is expected that these are already known, their role is to describe how the modules contribute to the achievement of the function. Note also that a system such as a car may have many other modules – braking, steering etc. – that do not play a part in this one specific function, and so are not shown in the use case.

D.4 Communication Diagram

Apart from the principle actor the other actors referenced in the use case are internal modules which collaborate to achieve the desired outcome. This collaboration is an essential aspect of the function’s operation and needs a method which describes it. UML provides a Communication Diagram for this purpose, as shown on Picture 32.

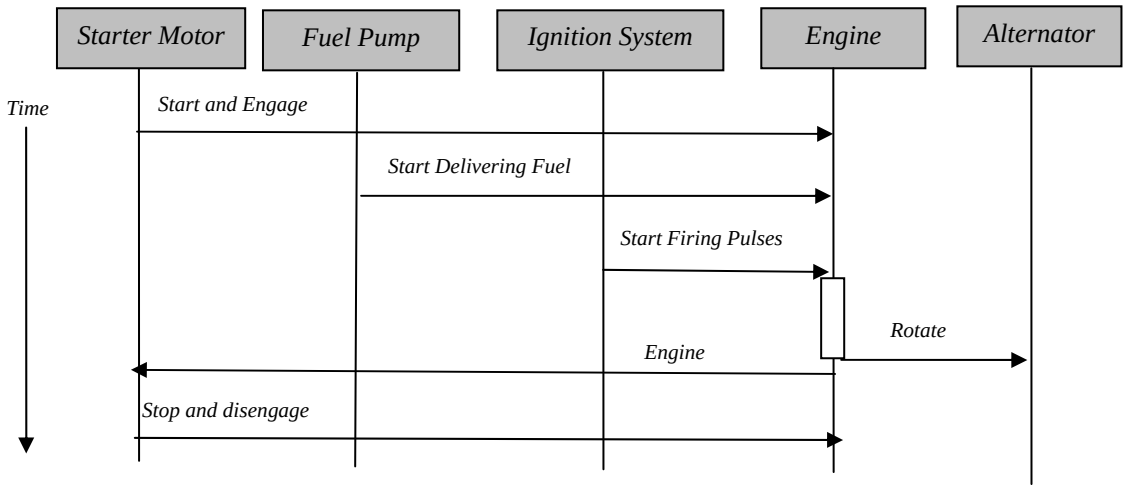


Picture 32 – Start Car Communication Diagram

This illustrates how the different modules interact and what type of interaction (or data transfer) occurs. For example, the interaction between the fuel pump and the engine is shown as the pump delivering fuel to the engine. The successful operation of the function depends on all the modules interacting in the way shown in the diagram. A failure of any one module, or any communication path, would prevent the successful completion of the function. Note that this diagram is designed to show that an interaction takes place between a number of modules, but does not easily show any information related to sequence of operations or timing requirements. This is especially noticeable in the communication between the engine and starter motor where three messages going in two directions are shown.

D.5 Sequential Diagram

To describe the sequential aspect of the function a UML Sequence Diagram is employed. This shows the same modules again interacting with the same communication mechanisms, but this time the diagram shows the order of operation.

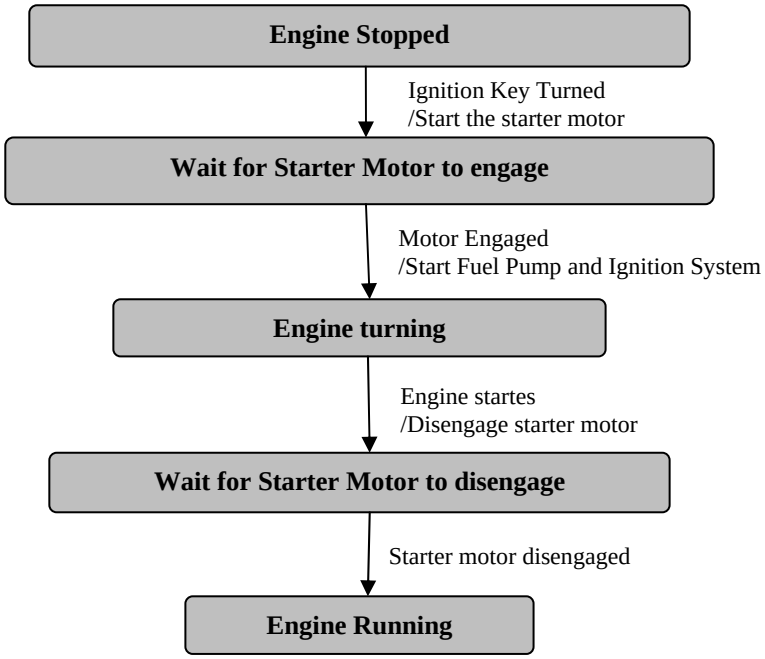


Picture 33 – Car Starting Sequence Diagram

The diagram illustrates the sequence that the different modules interact in, beginning with the starter motor starting and engaging, and ending with the starter motor disengaging after the engine has started.

D.6 State Diagram

Obviously the car can only be started if it isn't already running. It has to be in a stopped state in order to then be started, and the act of starting it moves it from stopped state to running state. Along the way there may be other interim states which make up the behaviour of the car starting function and which may be important to describe. This is done through a UML State Diagram which shows the states that the car is in and the events that must occur for it to move from one state to another (transitions).



Picture 34 – Car Starting State Diagram

The State Diagram illustrates another aspect of the car starting system in that there are times when the system is waiting for an event to occur, such as waiting for the starter motor to engage, because these are not instantaneous actions. During these times (states) other events could occur which could affect the function. For example, the driver could turn the ignition off while the system was still engaging the starter motor.

D.7 Other Functions

So far only starting the car has been examined as a function. Other functions would require additional diagrams to describe them. For example the function for stopping the car would need its own Use Case, Sequence Diagram and Communications Diagram to show its behaviour.

D.8 Conclusion

The reader should now have an appreciation of the different types of UML diagram that are available and in what manner they should be used. All the different UML diagrams interrelate with each other to show the complete functional behaviour and the processes and communication links that make up the total function. Other diagrams may also be used that are not described above, such as activity diagrams or class diagrams.

Appendice E. SCL Example Design Specification

The following is a listing of the SCL file that describes the SAS system used in this example. It is syntactaly correct, tested against the IEC 61850 XML Schema, but incomplete for some details unnecessary for this example.

```
<?xml version="1.0" encoding="utf-8"?>
<SCL xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.iec.ch/61850/2003/SCL&#xD;&#xA;SCL.xsd"
xmlns="http://www.iec.ch/61850/2003/SCL">
  <Header id="SCL Example T1-1" nameStructure="IEDName">
    <History />
  </Header>
  <Substation name="S12" desc="Baden">
    <LNode lnInst="1" lnClass="IHMI" iedName="A1KA1" ldInst="C1" />
    <VoltageLevel sxy:x="10" sxy:y="10" name="D1"
xmlns:sxy="http://www.iec.ch/61850/2003/SCLcoordinates">
      <PowerTransformer sxy:x="386" sxy:y="190" name="T1">
        <LNode lnInst="1" lnClass="PDIF" iedName="D1Q2BP1" ldInst="F1" />
        <LNode lnInst="1" lnClass="TCTR" iedName="D1Q2SB2" ldInst="C1" />
        <TransformerWinding name="W1">
          <Terminal connectivityNode="S12/D1/Q2/L2" substationName="S12"
voltageLevelName="D1" bayName="Q2" cNodeName="L2" />
        </TransformerWinding>
        <TransformerWinding name="W2">
          <Terminal connectivityNode="S12/E1/Q3/L2" substationName="S12"
voltageLevelName="E1" bayName="Q3" cNodeName="L2" />
        </TransformerWinding>
      </PowerTransformer>
      <Voltage unit="V" multiplier="k">220</Voltage>
      <Bay sxy:x="10" sxy:y="45" name="Q2">
        <ConductingEquipment sxy:x="376" sxy:y="15" name="QA1" type="CBR">
          <LNode lnInst="1" lnClass="CSWI" iedName="D1Q2SB2" ldInst="C1" />
          <LNode lnInst="1" lnClass="XCBR" iedName="D1Q2SB1" ldInst="C1" />
          <Terminal connectivityNode="S12/D1/Q1/B0" substationName="S12"
voltageLevelName="D1" bayName="Q1" cNodeName="B0" />
          <Terminal connectivityNode="S12/D1/Q2/L1" substationName="S12"
voltageLevelName="D1" bayName="Q2" cNodeName="L1" />
        </ConductingEquipment>
        <ConductingEquipment sxy:x="376" sxy:y="75" name="I1" type="CTR">
          <Terminal connectivityNode="S12/D1/Q2/L1" substationName="S12"
voltageLevelName="D1" bayName="Q2" cNodeName="L1" />
          <Terminal connectivityNode="S12/D1/Q2/L2" substationName="S12"
voltageLevelName="D1" bayName="Q2" cNodeName="L1" />
          <SubEquipment name="R" phase="A">
            <LNode lnInst="1" lnClass="TCTR " iedName="D1Q1BP1" ldInst="F1"
/>
          </SubEquipment>
          <SubEquipment name="S" phase="B">
            <LNode lnInst="2" lnClass="TCTR " iedName="D1Q1BP1" ldInst="F1"
/>
          </SubEquipment>
          <SubEquipment name="T" phase="C">
            <LNode lnInst="3" lnClass="TCTR " iedName="D1Q1BP1" ldInst="F1"
/>
          </SubEquipment>
```

```
          <SubEquipment name="I0" phase="N">
            <LNode lnInst="4" lnClass="TCTR " iedName="D1Q1BP1" ldInst="F1"
/>
          </SubEquipment>
        </ConductingEquipment>
        <ConnectivityNode name="L1" pathName="S12/D1/Q2/L1" />
        <ConnectivityNode name="L2" pathName="S12/D1/Q2/L2" />
      </Bay>
      <Bay sxy:x="10" sxy:y="10" name="Q1">
        <ConnectivityNode name="B0" pathName="S12/D1/Q1/B0" />
      </Bay>
    </VoltageLevel>
    <VoltageLevel sxy:x="10" sxy:y="280" name="E1"
xmlns:sxy="http://www.iec.ch/61850/2003/SCLcoordinates">
      <Voltage unit="V" multiplier="k">132</Voltage>
      <Bay sxy:x="10" sxy:y="10" name="Q3" desc="Turgi">
        <ConductingEquipment sxy:x="376" sxy:y="75" name="QA1" type="CBR">
          <LNode lnInst="1" lnClass="CSWI" iedName="D1Q3SB2" ldInst="C1" />
          <LNode lnInst="1" lnClass="XCBR" iedName="D1Q3SB1" ldInst="C1" />
          <Terminal connectivityNode="S12/E1/Q3/L0" substationName="S12"
voltageLevelName="E1" bayName="Q2" cNodeName="L0" />
          <Terminal connectivityNode="S12/E1/Q3/L1" substationName="S12"
voltageLevelName="E1" bayName="Q2" cNodeName="L1" />
        </ConductingEquipment>
        <ConductingEquipment sxy:x="376" sxy:y="135" name="QB1" type="DIS">
          <LNode lnInst="2" lnClass="XSWI" iedName="D1Q3SB3" ldInst="C1" />
          <Terminal connectivityNode="S12/E1/Q4/B1" substationName="S12"
voltageLevelName="E1" bayName="Q4" cNodeName="B1" />
          <Terminal connectivityNode="S12/E1/Q3/L0" substationName="S12"
voltageLevelName="E1" bayName="Q2" cNodeName="L0" />
        </ConductingEquipment>
        <ConductingEquipment sxy:x="376" sxy:y="15" name="I1" type="CTR">
          <Terminal connectivityNode="S12/E1/Q3/L1" substationName="S12"
voltageLevelName="E1" bayName="Q2" cNodeName="L1" />
          <Terminal connectivityNode="S12/E1/Q3/L2" substationName="S12"
voltageLevelName="E1" bayName="Q2" cNodeName="L2" />
          <SubEquipment name="R" phase="A">
            <LNode lnInst="1" lnClass="TCTR " iedName="D1Q1SB3" ldInst="F1"
/>
          </SubEquipment>
          <SubEquipment name="S" phase="B">
            <LNode lnInst="2" lnClass="TCTR " iedName="D1Q1SB3" ldInst="F1"
/>
          </SubEquipment>
          <SubEquipment name="T" phase="C">
            <LNode lnInst="3" lnClass="TCTR " iedName="D1Q1SB3" ldInst="F1"
/>
          </SubEquipment>
          <SubEquipment name="I0" phase="N">
            <LNode lnInst="4" lnClass="TCTR " iedName="D1Q1SB3" ldInst="F1"
/>
          </SubEquipment>
        </ConductingEquipment>
        <ConnectivityNode name="L0" pathName="S12/E1/Q3/L0" />
        <ConnectivityNode name="L1" pathName="S12/E1/Q3/L1" />
        <ConnectivityNode name="L2" pathName="S12/E1/Q3/L2" />
        <ConnectivityNode name="L3" pathName="S12/E1/Q3/L3" />
        <ConnectivityNode name="L4" pathName="S12/E1/Q3/L4" />
      </Bay>
```

```
    <Bay sxy:x="10" sxy:y="225" name="Q4">
      <ConnectivityNode name="B1" pathName="S12/E1/Q4/B1" />
    </Bay>
  </VoltageLevel>
</Substation>
<Communication>
  <SubNetwork name="W01" type="8-MMS">
    <Text>Station bus</Text>
    <BitRate unit="b/s" multiplier="M">10</BitRate>
    <ConnectedAP iedName="D1Q2SB1" apName="S1">
      <Address>
        <P xsi:type="tP_IP" type="IP">10.0.0.11</P>
        <P xsi:type="tP_IP-SUBNET" type="IP-SUBNET">255.255.255.0</P>
        <P xsi:type="tP_IP-GATEWAY" type="IP-GATEWAY">10.0.0.101</P>
        <P xsi:type="tP_OSI-TSEL" type="OSI-TSEL">00000001</P>
        <P xsi:type="tP_OSI-PSEL" type="OSI-PSEL">01</P>
        <P xsi:type="tP_OSI-SSEL" type="OSI-SSEL">01</P>
      </Address>
      <GSE ldInst="C1" cbName="XCBRQ2Status">
        <Address>
          <P type="MAC-Address">01-0C-CD-01-00-02</P>
          <P type="APPID">3001</P>
          <P type="VLAN-PRIORITY">4</P>
        </Address>
      </GSE>
      <PhysConn type="Plug">
        <P type="Type">FOC</P>
        <P type="Plug">ST</P>
      </PhysConn>
    </ConnectedAP>
    <ConnectedAP iedName="D1Q2SB2" apName="S1">
      <Address>
        <P xsi:type="tP_IP" type="IP">10.0.0.12</P>
        <P xsi:type="tP_IP-SUBNET" type="IP-SUBNET">255.255.255.0</P>
        <P xsi:type="tP_IP-GATEWAY" type="IP-GATEWAY">10.0.0.101</P>
        <P xsi:type="tP_OSI-TSEL" type="OSI-TSEL">00000001</P>
        <P xsi:type="tP_OSI-PSEL" type="OSI-PSEL">01</P>
        <P xsi:type="tP_OSI-SSEL" type="OSI-SSEL">01</P>
      </Address>
      <GSE ldInst="C1" cbName="CSWIQ2Status">
        <Address>
          <P type="MAC-Address">01-0C-CD-01-00-04</P>
          <P type="APPID">3001</P>
          <P type="VLAN-PRIORITY">4</P>
        </Address>
      </GSE>
      <PhysConn type="Plug">
        <P type="Type">FOC</P>
        <P type="Plug">ST</P>
      </PhysConn>
    </ConnectedAP>
    <ConnectedAP iedName="D1Q2SB3" apName="S1">
      <Address>
        <P type="IP">10.0.0.13</P>
        <P type="IP-SUBNET">255.255.255.0</P>
        <P type="IP-GATEWAY">10.0.0.101</P>
        <P type="OSI-TSEL">00000001</P>
        <P type="OSI-PSEL">01</P>
        <P type="OSI-SSEL">01</P>
```

```
  </Address>
  <SMV ldInst="C1" cbName="Amper">
    <Address>
      <P type="MAC-Address">01-0C-CD-04-00-04</P>
      <P type="APPID">4000</P>
      <P type="VLAN-ID">123</P>
      <P type="VLAN-PRIORITY">4</P>
    </Address>
  </SMV>
  <PhysConn type="Plug">
    <P type="Type">FOC</P>
    <P type="Plug">ST</P>
  </PhysConn>
</ConnectedAP>
<ConnectedAP iedName="D1Q2BP1" apName="S1">
  <Address>
    <P type="IP">10.0.0.14</P>
    <P type="IP-SUBNET">255.255.255.0</P>
    <P type="IP-GATEWAY">10.0.0.101</P>
    <P type="OSI-TSEL">00000001</P>
    <P type="OSI-PSEL">01</P>
    <P type="OSI-SSEL">01</P>
  </Address>
  <GSE ldInst="C1" cbName="Trip87">
    <Address>
      <P type="MAC-Address">01-0C-CD-01-00-02</P>
      <P type="APPID">3001</P>
      <P type="VLAN-PRIORITY">4</P>
      <P type="MAC-Address">01-0C-CD-01-00-06</P>
      <P type="APPID">3001</P>
      <P type="VLAN-PRIORITY">4</P>
    </Address>
  </GSE>
  <PhysConn type="Plug">
    <P type="Type">FOC</P>
    <P type="Plug">ST</P>
  </PhysConn>
</ConnectedAP>
<ConnectedAP iedName="E1Q3SB1" apName="S1">
  <Address>
    <P type="IP">10.0.0.15</P>
    <P type="IP-SUBNET">255.255.255.0</P>
    <P type="IP-GATEWAY">10.0.0.101</P>
    <P type="OSI-TSEL">00000001</P>
    <P type="OSI-PSEL">01</P>
    <P type="OSI-SSEL">01</P>
  </Address>
  <GSE ldInst="C1" cbName="XCBRQ3Status">
    <Address>
      <P type="MAC-Address">01-0C-CD-01-00-06</P>
      <P type="APPID">3001</P>
      <P type="VLAN-PRIORITY">4</P>
    </Address>
  </GSE>
  <PhysConn type="Plug">
    <P type="Type">FOC</P>
    <P type="Plug">ST</P>
  </PhysConn>
</ConnectedAP>
```

```
      <ConnectedAP iedName="E1Q3SB2" apName="S1">
        <Address>
          <P type="IP">10.0.0.16</P>
          <P type="IP-SUBNET">255.255.255.0</P>
          <P type="IP-GATEWAY">10.0.0.101</P>
          <P type="OSI-TSEL">00000001</P>
          <P type="OSI-PSEL">01</P>
          <P type="OSI-SSEL">01</P>
        </Address>
        <GSE ldInst="C1" cbName="CSWIQ3Status">
          <Address>
            <P type="MAC-Address">01-0C-CD-01-00-04</P>
            <P type="APPID">3001</P>
            <P type="VLAN-PRIORITY">4</P>
          </Address>
        </GSE>
        <PhysConn type="Plug">
          <P type="Type">FOC</P>
          <P type="Plug">ST</P>
        </PhysConn>
      </ConnectedAP>
      <ConnectedAP iedName="E1Q3SB3" apName="S1">
        <Address>
          <P type="IP">10.0.0.17</P>
          <P type="IP-SUBNET">255.255.255.0</P>
          <P type="IP-GATEWAY">10.0.0.101</P>
          <P type="OSI-TSEL">00000001</P>
          <P type="OSI-PSEL">01</P>
          <P type="OSI-SSEL">01</P>
        </Address>
<GSE ldInst="C1" cbName="XSWIQ3Status">
          <Address>
            <P type="MAC-Address">01-0C-CD-01-00-06</P>
            <P type="APPID">3001</P>
            <P type="VLAN-PRIORITY">4</P>
          </Address>
</GSE>
        <PhysConn type="Plug">
          <P type="Type">FOC</P>
          <P type="Plug">ST</P>
        </PhysConn>
      </ConnectedAP>
      <ConnectedAP iedName="A1KA1" apName="S1">
        <Address>
          <P type="IP">10.0.0.18</P>
          <P type="IP-SUBNET">255.255.255.0</P>
          <P type="IP-GATEWAY">10.0.0.101</P>
          <P type="OSI-TSEL">00000001</P>
          <P type="OSI-PSEL">01</P>
          <P type="OSI-SSEL">01</P>
        </Address>
        <PhysConn type="Plug">
          <P type="Type">FOC</P>
          <P type="Plug">ST</P>
        </PhysConn>
      </ConnectedAP>
    </SubNetwork>
  </Communication>
  <IED name="D1Q2SB1">
```

```
    <Services>
      <DynAssociation />
      <GetDirectory />
      <GetDataObjectDefinition />
      <GetDataSetValue />
      <DataSetDirectory />
      <ConfDataSet max="4" maxAttributes="50" />
      <ReadWrite />
      <ConfReportControl max="12" />
      <GetCBValues />
      <ConfLogControl max="1" />
      <ReportSettings cbName="Conf" datSet="Conf" rptID="Dyn" opt-
Fields="Conf" bufTime="Dyn" intgPd="Dyn" />
      <GSESettings cbName="Conf" datSet="Conf" appID="Conf" />
      <GOOSE max="2" />
      <FileHandling />
      <ConfLNs fixLnInst="true" />
    </Services>
    <AccessPoint name="S1">
      <Server>
        <Authentication />
        <LDevice inst="C1">
          <LN0 lnType="LN0" inst="">
            <DataSet name="XCBRQ2Pos">
              <FCDA ldInst="C1" prefix="" lnClass="XCBR" lnInst="1" do-
Name="Pos" fc="ST" />
            </DataSet>
            <ReportControl name="PosReport" datSet="Positions" rpt-
ID="E1Q1Switches" confRev="0">
              <TrgOps dchg="true" qchg="true" />
              <OptFields />
              <RptEnabled max="5">
                <ClientLN lnClass="IHMI" lnInst="1" iedName="A1KA1"
ldInst="LD1" />
              </RptEnabled>
            </ReportControl>
            <LogControl name="Log" datSet="XCBRQ2Pos" logName="C1">
              <TrgOps dchg="true" qchg="true" />
            </LogControl>
            <GSEControl name="XCBRQ2Status" datSet="XCBRQ2Pos" appID="Itl"
/>
          </LN0>
          <LN lnType="LPHDa" lnClass="LPHD" inst="1">
            <DOI name="Proxy">
              <DAI name="stVal">
                <Val>false</Val>
              </DAI>
            </DOI>
          </LN>
          <LN lnType="XCBRa" lnClass="XCBR" inst="1" />
        </LDevice>
      </Server>
    </AccessPoint>
  </IED>
  <IED name="D1Q2SB2">
    <Services>
      <DynAssociation />
      <GetDirectory />
      <GetDataObjectDefinition />
```

```
      <GetDataSetValue />
      <DataSetDirectory />
      <ConfDataSet max="4" maxAttributes="50" />
      <ReadWrite />
      <ConfReportControl max="12" />
      <GetCBValues />
      <ConfLogControl max="1" />
      <ReportSettings cbName="Conf" datSet="Conf" rptID="Dyn" opt-
Fields="Conf" bufTime="Dyn" intgPd="Dyn" />
      <GSESettings cbName="Conf" datSet="Conf" appID="Conf" />
      <GOOSE max="2" />
      <FileHandling />
      <ConfLNs fixLnInst="true" />
    </Services>
    <AccessPoint name="S1">
      <Server>
        <Authentication />
        <LDevice inst="C1">
          <LN0 lnType="LN0" inst="">
            <DataSet name="CSWIQ2Pos">
              <FCDA ldInst="C1" prefix="" lnClass="CSWI" lnInst="1" do-
Name="Pos" fc="ST" />
            </DataSet>
            <ReportControl name="PosReport" datSet="CSWIQ2Pos" rpt-
ID="E1Q1Switches" confRev="0">
              <TrgOps dchg="true" qchg="true" />
              <OptFields />
              <RptEnabled max="5">
                <ClientLN lnClass="IHMI" lnInst="1" iedName="A1KA1"
ldInst="LD1" />
              </RptEnabled>
            </ReportControl>
            <LogControl name="Log" datSet="CSWIQ2Pos" logName="C1">
              <TrgOps dchg="true" qchg="true" />
            </LogControl>
            <GSEControl name="CSWIQ2Status" datSet="CSWIQ2Pos" appID="Itl"
/>
          </LN0>
          <LN lnType="LPHDa" lnClass="LPHD" inst="1">
            <DOI name="Proxy">
              <DAI name="stVal">
                <Val>false</Val>
              </DAI>
            </DOI>
          </LN>
          <LN lnType="CSWIa" lnClass="CSWI" inst="2" />
        </LDevice>
      </Server>
    </AccessPoint>
  </IED>
  <IED name="D1Q2SB3">
    <Services>
      <DynAssociation />
      <GetDirectory />
      <GetDataObjectDefinition />
      <GetDataSetValue />
      <DataSetDirectory />
      <ConfDataSet max="4" maxAttributes="50" />
      <ReadWrite />
```

```
      <ConfReportControl max="12" />
      <GetCBValues />
      <ConfLogControl max="1" />
      <ReportSettings cbName="Conf" datSet="Conf" rptID="Dyn" opt-
Fields="Conf" bufTime="Dyn" intgPd="Dyn" />
      <GSESettings cbName="Conf" datSet="Conf" appID="Conf" />
      <GOOSE max="2" />
      <FileHandling />
      <ConfLNs fixLnInst="true" />
    </Services>
    <AccessPoint name="S1">
      <Server>
        <Authentication />
        <LDevice inst="C1">
          <LN0 lnType="LN0" inst="">
            <DataSet name="smv">
              <FCDA ldInst="C1" prefix="" lnClass="TCTR" lnInst="1" do-
Name="Amp" daName="instMag" fc="MX" />
              <FCDA ldInst="C1" prefix="" lnClass="TCTR" lnInst="2" do-
Name="Amp" daName="instMag" fc="MX" />
            </DataSet>
            <SampledValueControl name="Amper" datSet="smv" smvID="11"
smpRate="4800" nofASDU="5">
              <SmvOpts refreshTime="true" sampleSynchronized="true" sample-
Rate="true" />
            </SampledValueControl>
          </LN0>
          <LN lnType="TCTRa" lnClass="TCTR" inst="1" />
          <LN lnType="TCTRa" lnClass="TCTR" inst="2" />
        </LDevice>
      </Server>
    </AccessPoint>
  </IED>
  <IED name="D1Q2BP1">
    <Services>
      <DynAssociation />
      <GetDirectory />
      <GetDataObjectDefinition />
      <GetDataSetValue />
      <DataSetDirectory />
      <ConfDataSet max="4" maxAttributes="50" />
      <ReadWrite />
      <ConfReportControl max="12" />
      <GetCBValues />
      <ConfLogControl max="1" />
      <ReportSettings cbName="Conf" datSet="Conf" rptID="Dyn" opt-
Fields="Conf" bufTime="Dyn" intgPd="Dyn" />
      <GSESettings cbName="Conf" datSet="Conf" appID="Conf" />
      <GOOSE max="2" />
      <FileHandling />
      <ConfLNs fixLnInst="true" />
    </Services>
    <AccessPoint name="S1">
      <Server>
        <Authentication />
        <LDevice inst="C1">
          <LN0 lnType="LN0" inst="">
            <DataSet name="Trip87">
```

```
              <FCDA ldInst="C1" prefix="" lnClass="PDIF" lnInst="1" do-
Name="Op" fc="ST" />
            </DataSet>
            <ReportControl name="PosReport" datSet="Trip87" rpt-
ID="Diferential" confRev="0">
              <TrgOps dchg="true" qchg="true" />
              <OptFields />
              <RptEnabled max="5">
                <ClientLN lnClass="IHMI" lnInst="1" iedName="A1KA1"
ldInst="LD1" />
              </RptEnabled>
            </ReportControl>
            <LogControl name="Log" datSet="Trip87" logName="C1">
              <TrgOps dchg="true" qchg="true" />
            </LogControl>
            <GSEControl name="Trip87" datSet="Trip87" appID="Itl" />
          </LN0>
          <LN lnType="LPHDa" lnClass="LPHD" inst="1">
            <DOI name="Proxy">
              <DAI name="stVal">
                <Val>false</Val>
              </DAI>
            </DOI>
          </LN>
          <LN lnType="PDIFa" lnClass="PDIF" inst="1" />
        </LDevice>
      </Server>
    </AccessPoint>
  </IED>
  <IED name="E1Q3SB1">
    <Services>
      <DynAssociation />
      <GetDirectory />
      <GetDataObjectDefinition />
      <GetDataSetValue />
      <DataSetDirectory />
      <ConfDataSet max="4" maxAttributes="50" />
      <ReadWrite />
      <ConfReportControl max="12" />
      <GetCBValues />
      <ConfLogControl max="1" />
      <ReportSettings cbName="Conf" datSet="Conf" rptID="Dyn" opt-
Fields="Conf" bufTime="Dyn" intgPd="Dyn" />
      <GSESettings cbName="Conf" datSet="Conf" appID="Conf" />
      <GOOSE max="2" />
      <FileHandling />
      <ConfLNs fixLnInst="true" />
    </Services>
    <AccessPoint name="S1">
      <Server>
        <Authentication />
        <LDevice inst="C1">
          <LN0 lnType="LN0" inst="">
            <DataSet name="XCBRQ3Pos">
              <FCDA ldInst="C1" prefix="" lnClass="XCBR" lnInst="1" do-
Name="Pos" fc="ST" />
            </DataSet>
            <ReportControl name="PosReport" datSet="XCBRQ3Pos" rpt-
ID="E1Q1Switches" confRev="0">
```

```
              <TrgOps dchg="true" qchg="true" />
              <OptFields />
              <RptEnabled max="5">
                <ClientLN lnClass="IHMI" lnInst="1" iedName="A1KA1"
ldInst="LD1" />
              </RptEnabled>
            </ReportControl>
            <LogControl name="Log" datSet="XCBRQ3Pos" logName="C1">
              <TrgOps dchg="true" qchg="true" />
            </LogControl>
            <GSEControl name="XCBRQ3Status" datSet="XCBRQ3Pos" appID="Itl"
/>
          </LN0>
          <LN lnType="LPHDa" lnClass="LPHD" inst="1">
            <DOI name="Proxy">
              <DAI name="stVal">
                <Val>false</Val>
              </DAI>
            </DOI>
          </LN>
          <LN lnType="XCBRa" lnClass="XCBR" inst="1" />
        </LDevice>
      </Server>
    </AccessPoint>
  </IED>
  <IED name="E1Q3SB2">
    <Services>
      <DynAssociation />
      <GetDirectory />
      <GetDataObjectDefinition />
      <GetDataSetValue />
      <DataSetDirectory />
      <ConfDataSet max="4" maxAttributes="50" />
      <ReadWrite />
      <ConfReportControl max="12" />
      <GetCBValues />
      <ConfLogControl max="1" />
      <ReportSettings cbName="Conf" datSet="Conf" rptID="Dyn" opt-
Fields="Conf" bufTime="Dyn" intgPd="Dyn" />
      <GSESettings cbName="Conf" datSet="Conf" appID="Conf" />
      <GOOSE max="2" />
      <FileHandling />
      <ConfLNs fixLnInst="true" />
    </Services>
    <AccessPoint name="S1">
      <Server>
        <Authentication />
        <LDevice inst="C1">
          <LN0 lnType="LN0" inst="">
            <DataSet name="CSWIQ3Pos">
              <FCDA ldInst="C1" prefix="" lnClass="CSWI" lnInst="1" do-
Name="Pos" fc="ST" />
            </DataSet>
            <ReportControl name="PosReport" datSet="CSWIQ3Pos" rpt-
ID="E1Q1Switches" confRev="0">
              <TrgOps dchg="true" qchg="true" />
              <OptFields />
              <RptEnabled max="5">
```

```
                <ClientLN lnClass="IHMI" lnInst="1" iedName="A1KA1"
ldInst="LD1" />
              </RptEnabled>
            </ReportControl>
            <LogControl name="Log" datSet="CSWIQ3Pos" logName="C1">
              <TrgOps dchg="true" qchg="true" />
            </LogControl>
            <GSEControl name="CSWIQ3Status" datSet="CSWIQ3Pos" appID="Itl"
/>
          </LN0>
          <LN lnType="LPHDa" lnClass="LPHD" inst="1">
            <DOI name="Proxy">
              <DAI name="stVal">
                <Val>false</Val>
              </DAI>
            </DOI>
          </LN>
          <LN lnType="CSWIa" lnClass="CSWI" inst="2" />
        </LDevice>
      </Server>
    </AccessPoint>
  </IED>
  <IED name="E1Q3SB3">
    <Services>
      <DynAssociation />
      <GetDirectory />
      <GetDataObjectDefinition />
      <GetDataSetValue />
      <DataSetDirectory />
      <ConfDataSet max="4" maxAttributes="50" />
      <ReadWrite />
      <ConfReportControl max="12" />
      <GetCBValues />
      <ConfLogControl max="1" />
      <ReportSettings cbName="Conf" datSet="Conf" rptID="Dyn" opt-
Fields="Conf" bufTime="Dyn" intgPd="Dyn" />
      <GSESettings cbName="Conf" datSet="Conf" appID="Conf" />
      <GOOSE max="2" />
      <FileHandling />
      <ConfLNs fixLnInst="true" />
    </Services>
    <AccessPoint name="S1">
      <Server>
        <Authentication />
        <LDevice inst="C1">
          <LN0 lnType="LN0" inst="">
            <DataSet name="XSWIQ3Pos">
              <FCDA ldInst="C1" prefix="" lnClass="XSWI" lnInst="1" do-
Name="Pos" fc="ST" />
            </DataSet>
            <ReportControl name="PosReport" datSet="CSWIQ3Pos" rpt-
ID="E1Q1Switches" confRev="0">
              <TrgOps dchg="true" qchg="true" />
              <OptFields />
              <RptEnabled max="5">
                <ClientLN lnClass="IHMI" lnInst="1" iedName="A1KA1"
ldInst="LD1" />
              </RptEnabled>
            </ReportControl>
```

```
            <LogControl name="Log" datSet="XSWIQ3Pos" logName="C1">
              <TrgOps dchg="true" qchg="true" />
            </LogControl>
            <GSEControl name="XSWIQ3Status" datSet="XSWIQ3Pos" appID="Itl"
/>
          </LN0>
          <LN lnType="LPHDa" lnClass="LPHD" inst="1">
            <DOI name="Proxy">
              <DAI name="stVal">
                <Val>false</Val>
              </DAI>
            </DOI>
          </LN>
          <LN lnType="CSWIa" lnClass="CSWI" inst="2" />
        </LDevice>
      </Server>
    </AccessPoint>
  </IED>
  <IED name="A1KA1">
    <AccessPoint name="S1" />
  </IED>
  <DataTypeTemplates>
    <LNodeType id="LN0" lnClass="LLN0">
      <DO name="Mod" type="myMod" />
      <DO name="Health" type="myHealth" />
      <DO name="Beh" type="myBeh" />
      <DO name="NamPlt" type="myLPL" />
    </LNodeType>
    <LNodeType id="LPHDa" lnClass="LPHD">
      <DO name="Mod" type="myMod" />
      <DO name="Health" type="myHealth" />
      <DO name="Beh" type="myBeh" />
      <DO name="NamPlt" type="myLPL" />
      <DO name="PhyNam" type="myDPL" />
      <DO name="PhyHealth" type="myINS" />
      <DO name="Proxy" type="mySPS" />
    </LNodeType>
    <LNodeType id="CSWIa" lnClass="CSWI">
      <DO name="Mod" type="myMod" />
      <DO name="Health" type="myHealth" />
      <DO name="Beh" type="myBeh" />
      <DO name="Pos" type="myPos" />
      <DO name="GrpAl" type="mySPS" />
    </LNodeType>
    <LNodeType id="MMXUa" lnClass="MMXU">
      <DO name="Mod" type="myMod" />
      <DO name="Beh" type="myHealth" />
      <DO name="Health" type="myBeh" />
      <DO name="Amps" type="myMV" />
      <DO name="Volts" type="myMV" />
    </LNodeType>
    <LNodeType id="CILOa" lnClass="CILO">
      <DO name="Mod" type="myHealth" />
      <DO name="Beh" type="myBeh" />
      <DO name="Health" type="myINS" />
      <DO name="EnaOpen" type="mySPS" />
      <DO name="EnaClose" type="mySPS" />
    </LNodeType>
    <LNodeType id="TVTRa" lnClass="TVTR">
```

```
      <DO name="Mod" type="myMod" />
      <DO name="Health" type="myHealth" />
      <DO name="Beh" type="myBeh" />
      <DO name="Vol" type="mySAV" />
    </LNodeType>
    <LNodeType id="RSYNa" lnClass="RSYN">
      <DO name="Mod" type="myMod" />
      <DO name="Health" type="myHealth" />
      <DO name="Beh" type="myBeh" />
      <DO name="NamPlt" type="myLPL" />
      <DO name="Rel" type="mySPS" />
    </LNodeType>
    <DOType id="myMod" cdc="INC">
      <DA name="ctlVal" bType="Enum" type="Mod" fc="CO" />
      <DA name="stVal" bType="Enum" type="Mod" dchg="true" fc="ST" />
      <DA name="q" bType="Quality" dchg="true" fc="ST" />
      <DA name="t" bType="Timestamp" dchg="true" fc="ST" />
    </DOType>
    <DOType id="myHealth" cdc="INS">
      <DA name="stVal" bType="Enum" type="Health" dchg="true" fc="ST" />
    </DOType>
    <DOType id="myBeh" cdc="INS">
      <DA name="stVal" bType="Enum" type="Beh" dchg="true" fc="ST" />
    </DOType>
    <DOType id="myINS" cdc="INS">
      <DA name="stVal" bType="INT32" dchg="true" fc="ST" />
    </DOType>
    <DOType id="myLPL" cdc="LPL">
      <DA name="ldNs" bType="VisString255" fc="EX">
        <Val>IEC61850-7-4:2003</Val>
      </DA>
      <DA name="configRev" bType="VisString255" fc="DC">
        <Val>Rev 3.45</Val>
      </DA>
    </DOType>
    <DOType id="myDPL" cdc="DPL">
      <DA name="vendor" bType="VisString255" fc="DC">
        <Val>myVendorName</Val>
      </DA>
      <DA name="hwRev" bType="VisString255" fc="DC">
        <Val>Rev 1.23</Val>
      </DA>
    </DOType>
    <DOType id="myPos" cdc="DPC">
      <DA name="stVal" bType="Dbpos" type="Dbpos" dchg="true" fc="ST" />
      <DA name="q" bType="Quality" qchg="true" fc="ST" />
      <DA name="t" bType="Timestamp" fc="ST" />
      <DA name="ctlVal" bType="BOOL" fc="CO" />
    </DOType>
    <DOType id="mySPS" cdc="SPS">
      <DA name="stVal" bType="INT32" dchg="true" fc="ST" />
      <DA name="q" bType="Quality" qchg="true" fc="ST" />
      <DA name="t" bType="Timestamp" fc="ST" />
    </DOType>
    <DOType id="myMV" cdc="MV">
      <DA name="mag" bType="Struct" type="myAnalogValue" dchg="true"
fc="MX" />
      <DA name="q" bType="Quality" qchg="true" fc="MX" />
      <DA name="t" bType="Timestamp" fc="MX" />
```

```
      <DA name="sVC" bType="Struct" type="ScaledValueConfig" dchg="true"
fc="CF" />
    </DOType>
    <DOType id="myCMV" cdc="CMV">
      <DA name="cVal" bType="Struct" type="myVector" dchg="true" fc="MX" />
      <DA name="q" bType="Quality" qchg="true" fc="MX" />
      <DA name="t" bType="Timestamp" fc="MX" />
    </DOType>
    <DOType id="mySEQ" cdc="SEQ">
      <SDO name="c1" type="myCMV" />
      <SDO name="c2" type="myCMV" />
      <SDO name="c3" type="myCMV" />
      <DA name="seqT" bType="Enum" type="seqT" fc="MX" />
    </DOType>
    <DOType id="mySAV" cdc="SAV">
      <DA name="instMag" bType="Struct" type="myAnalogValue" fc="MX" />
      <DA name="q" bType="Quality" qchg="true" fc="MX" />
    </DOType>
    <DAType id="myAnalogValue">
      <BDA name="f" bType="FLOAT32" />
    </DAType>
    <DAType id="ScaledValueConfig">
      <BDA name="scaleFactor" bType="FLOAT32" />
      <BDA name="offset" bType="FLOAT32" />
    </DAType>
    <DAType id="myVector">
      <BDA name="mag" bType="Struct" type="myAnalogValue" />
      <BDA name="ang" bType="Struct" type="myAnalogValue" />
    </DAType>
    <EnumType id="ACDdir">
      <EnumVal ord="0">unknown</EnumVal>
      <EnumVal ord="1">forward</EnumVal>
      <EnumVal ord="2">backward</EnumVal>
      <EnumVal ord="3">both</EnumVal>
    </EnumType>
    <EnumType id="seqT">
      <EnumVal ord="0">pos-neg-zero</EnumVal>
      <EnumVal ord="1">dir-quad-zero</EnumVal>
    </EnumType>
    <EnumType id="Dbpos">
      <EnumVal ord="0">intermediate</EnumVal>
      <EnumVal ord="1">off</EnumVal>
      <EnumVal ord="2">on</EnumVal>
      <EnumVal ord="3">bad</EnumVal>
    </EnumType>
    <EnumType id="Tcmd">
      <EnumVal ord="0">stop</EnumVal>
      <EnumVal ord="1">lower</EnumVal>
      <EnumVal ord="2">higher</EnumVal>
      <EnumVal ord="3">reserved</EnumVal>
    </EnumType>
    <EnumType id="Beh">
      <EnumVal ord="1">on</EnumVal>
      <EnumVal ord="2">blocked</EnumVal>
      <EnumVal ord="3">test</EnumVal>
      <EnumVal ord="4">test/blocked</EnumVal>
      <EnumVal ord="5">off</EnumVal>
    </EnumType>
    <EnumType id="Mod">
```

```
      <EnumVal ord="1">on</EnumVal>
      <EnumVal ord="2">blocked</EnumVal>
      <EnumVal ord="3">test</EnumVal>
      <EnumVal ord="4">test/blocked</EnumVal>
      <EnumVal ord="5">off</EnumVal>
    </EnumType>
    <EnumType id="Health">
      <EnumVal ord="1">Ok</EnumVal>
      <EnumVal ord="2">Warning</EnumVal>
      <EnumVal ord="3">Alarm</EnumVal>
    </EnumType>
  </DataTypeTemplates>
</SCL>
```

Glossary

ACSI	Abstract Communication Service Interface
BRCB	Buffered Report Control Block
CDC	Common Data Class
GOOSE	Generic Object Oriented Substation Event
GSE	Generic Substation Event
GSSE	Generic Substation State Event
HMI	Human Machine Interface
IED	Intelligent Electronic Device
LD	Logical Device
LN	Logical Node
MCAA	Multicast Application Association
MMS	Manufacturer Message Specification
MSS	Main Success Scenario
OPC	OLE for Process Control
OSI	Open Systems Interconnection
PAS	Power Station System
PICOM	Piece of Information for COMmunication
PLC	Programmable Logic Controller
RCB	Report Control Block
SCADA	Supervisory Control and Data Acquisition
SCL	Substation Configuration Language
SCSM	Specific Communication Service Mapping
SGCB	Setting Group Control Block
SV	Sampled Values
TPAA	Two Party Application Association
UML	Unified Modeling Language
URCB	Un-buffered Report Control Block
XML	Extensible Mark-up Language

References

Standards

[1] IEC 61850-1: Communication networks and systems in substations – Part 1: Introduction and Overview, International Electro technical Commission, Switzerland.

[2] IEC TS 61850-2: Communication networks and systems in substations – Part 2: Glossary, International Electro technical Commission, And Switzerland.

[3] IEC 61850-3: Communication networks and systems in substations – Part 3: General requirements, International Electro technical Commission, Switzerland.

[4] IEC 61850-4: Communication networks and systems in substations – Part 4: System and project management, International Electro technical Commission, Switzerland.

[5] IEC 61850-5: Communication networks and systems in substations – Part 5: Communication requirements for functions and device models, International Electro technical Commission, Switzerland.

[6] IEC 61850-6: Communication networks and systems in substations – Part 6: Configuration description language for communication in electrical substations related to IEDs, International Electrotechnical Commission, Switzerland.

[7] IEC 61850-7-1: Communication networks and systems in substations – Part 7-1: Basic communication structure for substation and feeder equipment – Principles and models, International Electrotechnical Commission, Switzerland.

[8] IEC 61850-7-2: Communication networks and systems in substations – Part 7-2: Basic communication structure for substation and feeder equipment – Abstract communication service interface (ACSI), International Electrotechnical Commission, Switzerland.

[9] IEC 61850-7-3: Communication networks and systems in substations – Part 7-3: Basic communication structure for substation and feeder equipment – Common data classes, International Electrotechnical Commission, Switzerland.

[10] IEC 61850-7-4: Communication networks and systems in substations – Part 7-4: Basic communication structure for substation and feeder equipment – Compatible logical node classes and data classes, International Electrotechnical Commission, Switzerland.

[11] IEC 61850-7-410: Communication networks and systems for power utility automation – Part 7-410: Hydroelectric power plants – Communication for monitoring and control, International Electrotechnical Commission, Switzerland.

[12] IEC 61850-8-1: Communication networks and systems in substations – Part 8-1: Specific Communication Service Mapping (SCSM) – Mappings to MMS

(ISO 9506-1 and ISO 9506-2) and to ISO/IEC 8802-3, International Electrotechnical Commission, Switzerland.

[13] IEC 61850-9-1: Communication networks and systems in substations – Part 9-1: Specific Communication Service Mapping (SCSM) – Sampled values over serial unidirectional multidrop point to point link, International Electrotechnical Commission, Switzerland.

[14] IEC 61850-9-2: Communication networks and systems in substations – Part 9-2: Specific Communication Service Mapping (SCSM) – Sampled values over ISO/IEC 8802-3, International Electrotechnical Commission, Switzerland.

[15] IEC 61850-10: Communication networks and systems in substations – Part 10: Conformance Testing, International Electrotechnical Commission, Switzerland.

[16] IEC 60870-5-101: Telecontrol equipment and systems – Part 5-101: Transmission protocols – Companion standard for basic telecontrol tasks, International Electrotechnical Commission, Switzerland.

[17] IEC 60870-5-103: Telecontrol equipment and systems – Part 5-103: Transmission protocols – Companion standard for the informative interface of protection equipment, International Electrotechnical Commission, Switzerland.

[18] IEC 60870-5-104: Telecontrol equipment and systems – Part 5-104: Transmission protocols – Network access for IEC 60870-5-101 using standard transport profiles, International Electrotechnical Commission, Switzerland.

[19] IEC 60870-6: Telecontrol equipment and systems – Part 6: Telecontrol protocols compatible with ISO standards and ITU-T recommendations, International Electrotechnical Commission, Switzerland.

[20] IEC 61346: Industrial systems, installations and equipment and industrial products – Structuring principles and reference designations, International Electrotechnical Commission, Switzerland.

[21] IEEE Std C37.2:1996, IEEE Standard Electrical Power System Device Function Numbers and Contact Designations

[22] IEC 61882: Hazard and Operability Studies (HAZOP Studies) - Application Guide, International Electrotechnical Commission, Switzerland.

[23] IEC 61882: Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA), International Electrotechnical Commission, Switzerland.

[24] OMG, UML Testing Profile, The Object Management Group, available on www.omg.org, 2005.

[25] W3C, Extensible Markup Language (XML), The World Wide Web Consortium, available on www.w3.org, 2006.

Books

[1] Tsai, J. J. P., Bi, Y., Yang, S. J. H., Smith, R. A. W., Distributed Real-Time

Systems: Monitoring, Visualization, Debugging, and Analysis, John Wiley & Sons, Inc., New York, 1996.

[2] Bradley, N., The XML Schema Companion, Addison-Wesley, Boston, 2004.

[3] Fowler, M., UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition, Addison-Wesley Professional, New York, 2003.

[4] Douglass, B. P., Real Time UML: Advances in the UML for Real-Time Systems (3rd Edition), Addison-Wesley Professional, New York, 2004.

Papers

[5] K. P. Brand et al, First experiences with customer specifications of IEC 61850 based Substation Automation Systems, Cigré SC B5 Colloquium 2005, Paper 203.

[6] K. P. Brand et al, The Introduction of IEC 61850 and its Impact on Protection and Automation within Substation, Cigré Electra Magazine, n. 233, August 2007.