SAFETY TESTING FOR LHC ACCESS SYSTEM

F. Valentini, T. Ladzinski, P. Ninin, L.Scibile CERN, Geneva, Switzerland

Abstract

In the domain of *Safety Real-Time Systems* the problem of testing represents always a big effort in terms of time, costs and efficiency to guarantee an adequate coverage degree. Exhaustive tests may, in fact, not be practicable for large and distributed systems. This paper describes the testing process followed during the validation of the CERN's LHC Access System [1], responsible for monitoring and preventing physical risks for the personnel accessing the underground areas. In the paper we also present a novel strategy for the testing problem, intended to drastically reduce the time for the test patterns generation and execution. In particular, we propose a methodology for *black-box* testing that relies on the application of *Model Checking* techniques. Model Checking is a formal method from computer science, commonly adopted to prove correctness of system's models through an automatic system's state space exploration against some property formulas.

INTRODUCTION

In environments where safety of human life represents an operational constraint, exhaustive testing is an ideal methodology to ensure the correct system behaviour even in extremely rare conditions. However, the high cost in terms of time and resources makes such approach effectively not practicable.

Referring to a *black-box* testing scenario, we adopted a test selection and an adequacy criterion [4] leading the whole validation process of the LHC Access Control and Safety System. The implementation of a dedicated test hardware platform was then necessary in order to validate the entire control software distributed among 10 *Fail-Safe* Programmable Logic Controllers (PLC).

The LHC Access Control and Safety System is today responsible for all aspects related to the LHC tunnel accesses: from the verification of all needed privileges, to the detection and prevention of any dangerous situation for the accessing personnel. Particular care has been dedicated to the *safety* aspects connected to the operation of the LHC accelerator where the risk [3] covered by the Access System project concerns the exposition to radiation in the LHC tunnel and experimental areas.

The total area supervised by the system includes the whole 27 Km ring tunnel, divided in 9 *sites* in turn partitioned in more *zones*, classified by the same risk index. A large number of system component's states are continuously monitored by the control software in order to ensure the persistency of the '<u>Safe for Access</u>' conditions, during periods where access is granted, while the '<u>Safe for Beam</u>' conditions have to hold during any LHC operation. Principal types of system's Elements Important for Safety [EIS] are:

- *Access Points* – allowing accesses between zones at different risk and from/to the exterior;
- *Sector Doors* – allowing accesses between different sectors of a same zone;
- *End Zone Doors* – allowing emergency evacuations from a zone;
- *Accelerator Safety Elements* – special components able to block the circulating beams and to prevent any further injection;

In the first part of the article we describe the LHC Access System simulation platform, designed to faithfully reproduce the real system's architecture. Subsequently, we introduce an innovative testing methodology, relying on the application of Formal Methods from computer science, intended to automatically generate test cases from requirements.

Testers using this approach concentrate their efforts on system data model design and on the application of specific generating algorithms rather than hand-drawing individual tests. The model is basically a description of the system behaviour in terms of its visible actions, given by a formal language. The test case generation relies instead on Model Checking algorithms.

Model Checking is a technique developed for verifying finite-state systems. Given a system model, normally expressed in terms of a *Labelled Transition System* (LTS), and a system property written as a temporal logic formula [5], the model checker can determine whether the model satisfies the property formula, through an automatic state space exploration. In addition to being fully automatic, an important feature of model checkers is that a counterexample can be supplied to demonstrate whenever the model fails to satisfy the specified property. A counterexample is a path, a sequence of system actions, in the execution of the model leading to a particular system's state where the given property formula is violated.

We show how a system property formula can be turned into a *test case* and its violating paths used as executable tests.

TEST PLATFORM ARCHITECTURE

The aim of the simulation and test platform, shown in figure 1, is to reproduce the real environment where the Safety PLCs and all supervising/control software is going to operate.

Safety software for each LHC site, controlled by a dedicated fails-safe PLC, has been independently tested on the platform before being put in operation. This form of validation allowed to minimize the final test campaign conducted in the field, particularly time consuming due the number and the distribution of the controlled elements, installed throughout the 27 Km of LHC's ring.

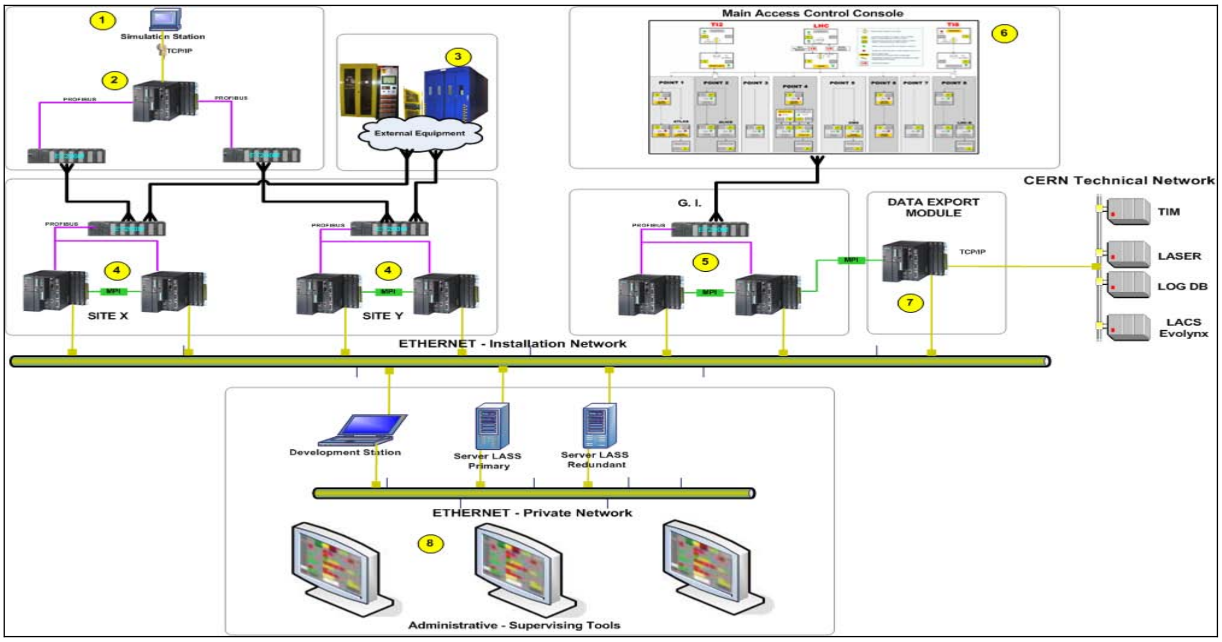

Figure 1: Architecture of the LHC Access System Test Platform

Emulation Station

The core of the platform is the *Emulation Station* (**1**) which consists in a standard PC containing the list of all controlled equipment. A specific software allows sending EIS emulated inputs/outputs via Ethernet connection to a non safety Siemens PLC (**2**), used as a gateway to the *fail-safe* PLCs under test (**4**).

External Equipment

This platform offers the possibility to plug real equipment (**3**), as Sector Doors or Access Points, to the safety PLCs under test.

Site's Safety PLC Validation

The access safety conditions to the entire LHC tunnel are controlled independently in each site by 2 redundant *fail-safe* PLCs synchronized by a central global PLC (**5**) and communicating on the same Ethernet network. The Test Platform allows simulating and testing two sites at the time (**4**), both receiving inputs from the gateway PLC (**2**) and from the global PLC (**5**). The sites under test consist in two couple of redundant and *fail-safe* Siemens PLCs.

Central Access Control Panel

The *Main Access Control Console* (**6**), used to open or close the access at each LHC site and to start the accelerator is also reproduced in this Test Platform. The Access Console communicates directly, via direct cable connections, with the Global Interlock (GI) Controller (**5**) which also consists in a Siemens fail-safe and redundant PLC. As well as receiving commands from the Access Main Console, the GI constantly coordinates and monitors the activity of each local site PLCs in order to be able to evaluate, at any time, the '*Safe for Access*' and the '*Safe for Beam*' conditions.

Supervising Consoles

All *Supervising Consoles* (**8**), used by the control room operators to graphically display the equipment states and to change the access modes (general, restricted, closed, patrol), are also reproduced in the Test Platform.

The test platform was used to validate all system *safety functions* as they are described in the project's functional specification document. Every safety functionality was tested with a specific scenario where all relevant system inputs were exercised in order to verify the correctness of the safety function response. All test input signals were emulated exhaustively, one by one, setting values manually on the *emulation station* (**2**), but without consider all possible *interleaving* that concurrent events could generate.
The approach, under study, presented in the next chapter is intended to treat concurrency tests, allowing the automatic executions of all possible input interleaving for a given safety function.

AUTOMATIC TESTING

We present a technique belonging to the so called Model-Based approach intended to automatically derive test patterns from a formal model of the system under validation. Tests derived in this way are normally called "*abstract*" because explicit operations of data mapping are necessary to obtain concrete executable tests.
We assume that the test strategy is oriented to the validation of each **system's requirement** trough

appropriate test cases according to the well known *Specification-Based* testing criterion [4].

Our approach consists in automatically deriving test sequences for a particular Test Case defined for the system under test. In practical use, the system is modelled with the *Labelled Transition System* (LTS) formalism, while each Test Case is described by a logic formula, expressed with *Linear Temporal Logic* (LTL) algebra and representing a system property. After this point, thanks to the Model Checking capability to exhibit counter-examples in correspondence of any system property violation, the whole process of the test input pattern generation becomes completely automatic.

System Modelling

The first step is to model each system requirement independently using the LTS formalism. Requirements are hence used as a criterion to partition the global system state space, which is usually impossible to represent completely in a single monolithic model.

The requirement modelling task must follow some basic guidelines:

- Models must be *closed* to the requirement, describing only the system's visible actions directly related to the requirement under validation;
- Models must keep a high level of adherence with part of the real system functioning. Any form of synthesis or abstraction should be avoided here.

The modelling phase is hence aimed at producing independent and precise models of the real system behaviour, representing a formal description of each system's requirement.

Test Case Definition

A Test Case covers a particular system behaviour (functionality) we are interested to verify, by assigning for example different values to a particular subset of the system variables.

The Linear Temporal Logic (LTL) is the formalism we use to conveniently express all possible Test Cases. In LTL it is possible to encode formulae about the future of system's events such that a condition will eventually be true, that a condition will be true until another fact becomes true, etc.

We used the capability of LTL formulae operators to reason about future, past and to establish timing relations between events in the system in order to formally represent each test case, which will be then seen as a system *safety* property (φ).

In the case of study presented in this paper, a legitimate Test Case, used to test every event driving to an *unsafe* state of a given sector X in RESTRICTED mode, will be formulated with the following system safety property:

"*The sector X can NEVER become unsafe IF the access level is set to Restricted*"

which in LTL language, using the **Until** operator, would be formalized:

$$\varphi = (\text{sector X} = \textit{safe})\ \mathbf{U}\ (\text{access} \neq \textit{Restricted})$$

to express the fact that any event making the sector X unsafe when the access level is set to Restricted represents a safety system violation and it is a significant test.

Test Input Pattern Extraction

Previously we described how model checking is used to check if a given property (φ) is valid in a model (M):

$$\mathbf{M} \models \varphi$$

then if the property is not hold by the model, a counterexample path:

$$\text{Act1} \rightarrow \text{Act2} \rightarrow \ldots\ldots \text{ActN} \rightarrow \mathbf{Error}$$

leading to the violation of (φ) is shown.

Each property formula violation trace, obtained via the Model Checking proofing capability, represents one possible executable path on the LTS system model and can serve as *abstract test*. If the system property (φ) is correctly specified, then all LTS executable paths obtained in this way are exactly all the existing valid tests for the considered Test Case.

CONCLUSION

In this paper we showed the importance of having an appropriate test platform, in order to reproduce real cases for *safety* PLC based software.

The need to approach this problem in the most efficient and safe possible way, forced us to realize a complex dedicated test hardware, but also to adopt a punctual test methodology in order to be able to estimate a quality measure for the executed tests.

After this experience, our conviction is that the automation of the testing process is an important key to save time, to obtain a much higher degree of coverage and to treat concurrency aspects of the system.

In the second part of the article we presented an approach to easily obtain a general purpose test input pattern generator, based on Model Checking techniques, which will shortly be implemented for the future validations of LHC access safety software upgrades.

REFERENCES

[1] P. Ninin, T. Ladzinski *et* al. "*LHC Access System: From Design to Operation*". (2008) This conference proceedings.

[2] P. Ninin and L. Scibile, "*Access Project Concept Architecture*", EDMS 477152. (2005) CERN.

[3] P. Ninin, S. Grau, G. Roy "*Synthèse de l'analyse préliminaire des risques*", EDMS 479041. (2005) CERN.

[4] H. Zhu, P. Hall, J. May, "*Software Unit Test Coverage and Adequacy*". ACM Computing Surveys, Vol. 29, (1997) UK.

[5] E. Clarke, O. Grumberg, D. Peled, "*Model Checking*". MIT Press, (1999) UK.

[6] M. Broy, B. Jonsson, J. Katoen, M. Leucker, A. Pretschner "*Model-Based Testing of Reactive Systems*". Springer, (1998) US.