Cutting Time-to-Market by Adopting Automated Regression Testing in a Simulated Environment

Manuel Palmieri[1], Antonio Cicchetti[2], and Anders Öberg[3]

[1] Bombardier Transportation, Västerås, Sweden
manuel.palmieri@se.transport.bombardier.com
[2] Mälardalen University, Västerås, Sweden
antonio.cicchetti@mdh.se
[3] MAXIMATECC, Västerås, Sweden
anders.oberg@maximatecc.com

Abstract. Industrial system development is facing an ever growing complexity of the target applications together with market demands of reducing time and costs of products. This issue is even more relevant in safety critical domains, where the quality of the system has to be necessarily validated before any release can be placed on the market. Bombardier Transportation works in one of such domains, namely rail-equipment development, and has to deal with problems related to testing the developed applications in an efficient and effective way while trying to reduce costs and time-to-market.

This work illustrates the concrete solutions adopted by the company in order to effectively test their systems; in particular, they adopt automated regression testing and simulated environments to speed-up the process and alleviate the problems due to hardware costs and size as well as the non-reversibility of reality.

Keywords: Industrial System Testing, Automated Regression Testing, Simulation Environments, Rail Equipment Development.

1 Introduction

Nowadays' needs to increase software quality assurance and reduce development time is making manual testing not appropriate to meet market demands [10]. Moreover, companies that produce artefacts for safety-critical applications, notably automotive, aviation, medicine, nuclear engineering, transport, etc., have to face some additional issues. In particular, they have to deal with high costs for buying hardware units devoted to developers and testers; they have to face the impossibility to perform certain types of tests because of their dangerousness and destructiveness, and management of timing issues related to the execution of tests.

Bombardier Transportation is one of the world's largest companies in rail-equipment manufacturing and servicing industries with a wide product range

M.G. Merayo and E. Montes de Oca (Eds.): ICTSS 2014, LNCS 8763, pp. 129–144, 2014.

of passenger trains, locomotives, and boogies [11]. MAXIMATECC is a Swedish company whose purpose is to support manufacturers of industrial vehicles with solutions that deal with humans in control of vehicles working in critical environments [13]. Besides providing mechanical and electrical parts, Bombardier Transportation also produces software for its vehicle control systems. One of the most important parts of such software is the Train Control and Management System (TCMS), which is a high capacity, infrastructure backbone that allows easy integration of all controls and communications requiring functions on-board the train [11][12].

This article discusses about the exploitation of automated regression testing in a simulated environment by illustrating its concrete implementation in an industrial setting. The aim is to clarify the general needs, implementation solutions, advantages, and drawbacks of adopting such a testing methodology in industrial practice. To the best of our knowledge, despite the relevance of the problem, this is the first work that illustrates the issues and possible solutions due to the introduction of regression testing techniques and simulated environments in the railway industrial domain.

The paper is structured as follows: Sect. 2 describes the basic concepts that underpin the contribution of this work, while Sect. 3 illustrates the main characteristics of the TCMS and its complexity. Sect. 4 presents the Bombardier Transportation and MAXIMATECC framework that supports automated regression testing and its interconnection with the simulated environment, while Sect. 5 clarifies the fundamental steps the brought Bombardier Transportation and MAXIMATECC to obtain the desired support for simulation. The paper then shows concrete results of the comparison between manual versus automated testing procedures both in a real and in a simulated environment in Sect. 6. The work is concluded in Sect. 7 by drawing considerations on the lesson learned in the realization of the automated regression testing support including its interconnection with simulated environments, and outlines some future investigation directions.

2 Background

Software testing has been traditionally meant to be a manual activity, where people sitting in front of the computers test the software by trying various usage and input combinations. In the last decade, due to the need of increasing the effectiveness, efficiency and coverage of the testing, companies are investing more and more on the automation testing. [16]. In this work we refer to automated testing as the process to analyse software by using special tools to automatically control the execution of tests and detect differences between existing software conditions and expected results [1] [17]. Its introduction has given a significant boost to the testing area, by allowing to speed-up testing activities and hence the whole Software Development Life-Cycle (SDLC). Moreover, it has contributed to alleviate relevant disadvantages of manual testing by reducing test execution time, human mistakes in performing and evaluating tests, and limitations related to the dangerousness and time extension of tests [2] [16]. Besides, automated

testing has also favoured the use of regression testing, which was considered an expensive and time consuming testing technique. The combination of automated and regression testing has given life to a practice that is usually called automated regression testing [15].

Regression Testing is a particular type of software testing that prescribes retesting the software during its SDLC, with the aim to detect defects deriving from software changes. When a new version of the software is released, existing test suites have to be partly or completely rerun to verify and validate that software changes did not introduce new faults, and that the software behaviour is still matching the requirements. During the re-testing phase, developers and testers have to continuously revise and refine test cases with the aim to make them as much as possible accurate for the new software release [2].

The combination of automated and regression testing has the benefit that they compensate the disadvantages of one another. On the one hand, automated testing has the advantage to automatically perform tests avoiding human resources to step by step verify and validate the correctness of the software, but has the disadvantages of high initial costs, time-consuming setup and configuration operations needed to prepare the environment to run tests. On the other hand, regression testing has the advantage to have a short setup phase to run tests, but has the disadvantage that for each software change tests should be rerun and consequently very time-consuming if manually done. Therefore, benefits of automated regression testing become evident in long-term SDLCs, where the testing phase is quite long and there is the necessity to frequently run tests [2] [3].

Among possible implementations, automated regression testing in a simulated environment represents an important possibility for companies since it magnifies the advantages gained through automation [4]. In particular, it allows to improve accuracy, coverage, and reusability by setting and possibly enforcing certain system states that constitute the pre-conditions for a given test to be run. It is worth noting that in real environments it can be very difficult (and even impossible in some cases) to reproduce particular conditions due to timing issues as well as because of the potential destructiveness of the test. For the same reasons, simulated environments permit to save time and money (e.g by avoiding to buy multiple hardware components) in selecting the best alternative among several acceptable design solutions.

In the following, we first illustrate the TCMS system in order to let the reader grasp the complexity of a modern train application; then, we describe how Bombardier Transportation and MAXIMATECC provided an integrated solution for automated regression testing in a simulated environment.

3 The TCMS System

Bombardier Transportation produces software for its Vehicle Control Units (VCU) and Intelligent Display Units (IDU). One of the most important parts of the developed software is the TCMS, which is a high capacity, infrastructure backbone that allows easy integration of all controls and communications requiring functions

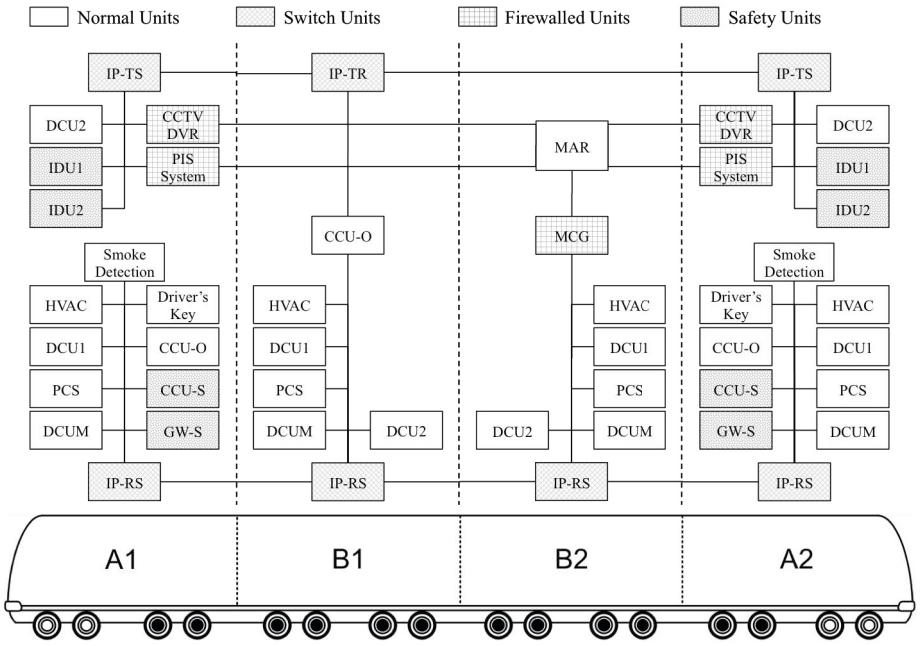

Fig. 1. General architecture of the TCMS system

on-board the train. It provides a high processing capability and bandwidth for real time exchange of data, both throughout the train and with the outside world, via up to date mobile communications [5].

TCMS interacts with all train subsystems, notably brakes, pantographs, doors, lights, toilets, and so forth. The architecture of TCMS consists of mainly three networks, namely Internet Protocol Train Communication (IPTCom), Multifunction Vehicle Bus (MVB), and Serial. In Fig. 1 is shown an example of an IPTCom network [1], where it is illustrated how involved sub-units communicate with each other. Even though the architecture in the picture is train specific, it is noteworthy that train architectures adhere to a generic design pattern that is detailed in the following.

Usually, each train consists of multiple cars, where the first and the last are a combination of control and passenger cars, whereas the ones in between are usually only passenger cars. As expected, control cars (which in this figure are named with A1 and A2) are the places where the driver can sit and where most of the train controls are contained. Moreover, they contain most of the TCMS units and those units that are classified as safety critical. Instead, B1 and B2 contain only units to control normal actions and reactions that are executed in all cars (e.g, activate driver's desk, raise pantograph, open battery contact, activate emergency brakes). A relevant characteristic of the TCMS architecture is its

[1] For the sake of space and readability, in this paper we only show part of the IPTCom network to let the reader grasp the complexity of the TCMS system.

redundancy: almost all units included in both control/passenger and passenger cars are replicated. This serves to increase the reliability of the system in case of malfunctions, damages, etc. of architecture units. Moreover, safety critical devices have the capacity to take decisions and act on the train in case of system malfunctions, driver absence, etc. These actions depend on the status of TCMS units, the status of the train and its location, the communication with the outside world, etc.

By looking deeper at the TCMS architecture example, it is possible to notice that most of the units contained in A1 are duplicated in A2 and the same happens for B1 and B2. In particular, B1 and B2 contain Heating, Ventilation and Air-Conditioning (HVAC), Driver Control Unit 1 (DCU1), Driver Control Unit 2 (DCU2), Passenger Counting System (PCS), and Drive Control Unit Motor (DCUM) converter units. A1 and A2 include a majority of the passenger car units, and additionally locate units such as Driver's Key, those classified safety critical such as IDUs, Central Computing Unit Safety (CCU-S), and GateWay Safe (GW-S), those protected by firewall (exclusion MCG) such as Closed Circuit TeleVision Digital Video Recorder (CCTV DVR) and Passenger Information System (PIS) System, and the Central Computing Unit Operation (CCU-O).

Since TCMS communicates with other systems (both inside and outsite the train), it needs to be protected with firewall units. In the IPTCom network of the TCMS architecture there are five devices that are protected by the firewall and they are CCTV DVR, PIS System and MCG as shown in Fig. 1. Moreover, TCMS is responsible to read Input and Output (IO) signals connected to the driver control and communicate their states to the dedicated hardware to process their values and perform corresponding actions. Additionally, it is also responsible to evaluate reactions, which can be generally described as consequences to actions, e.g., open doors, release brakes, activate heating, turn on lights [14].

Over the years, this system has become more and more complex and customer's requirements of shorter time-to-market became more stringent. Bombardier Transportation, with the aim to meet these requirements, keep its global leader position, improve its competitiveness and quality, and reduce the development time and costs, has revolutionized its development process by creating, *i)* an automated testing environment and, *ii)* a simulated environment able to replicate the behaviour of the real system, both in its software and hardware parts. The need to perform critical tests and regression testing has encouraged the use of automated testing versus manual testing. Furthermore, the decision to create a simulated environment has been mainly due to the impossibility to have the real train in the office because of space, costs, and because typically software development is distributed across different company sites.

4 Automated Regression Testing Support at Bombardier Transportation

This section describes how automated regression testing and simulated environment techniques are combined in the state of practice at Bombardier. In particular, we first introduce the overall environment that supports testing for both

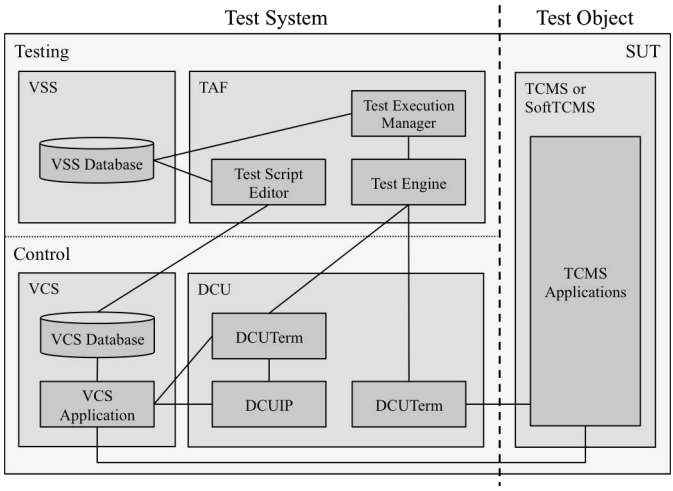

Fig. 2. General architecture of the test environment

real and simulated artefacts. Then, we illustrate how simulation mechanisms are integrated to test the System Under Testing (SUT).

4.1 Test Environment

Bombardier Transportation has created a testing environment with the aim to speed-up and automate as much as possible the testing process, and ease the adoption of regression testing. The result is a simple architecture, but at the same time complex in software components, consisting of two blocks, namely Test System and Test Object, as depicted in Fig. 2.

Test System consists of two sub-systems, i.e. Testing and Control, that provide automation support for the testing process and a controller tailored to the Test Object, respectively. The former includes the Test Automation Framework (TAF) and Visual Source Safe (VSS), whereas the latter encompasses a Vehicle Control Simulator (VCS) and Driver Control Unit (DCU).

TAF is a set of applications devoted to the specification of tests and their automated execution. In particular, the Test Script Editor (TSE) serves to create and edit test scripts and snippets [2] based on the definition of test specifications. TSE requires the communication with the VCS and VSS databases (as shown by means of interconnection lines in Fig. 2), in order to gather on one hand train signals and controls, and on the other hand test scripts, snippets, and sessions, respectively. Train signals and controls are categorized as *action*, *reaction*, and *neutral* instructions. Actions are the operations the SUT is subject to, reactions are the corresponding effects triggered in the SUT and observable in terms of changes to the system, and neutrals have no effects on the SUT, but serve to support the execution of test scripts, such as the execution of parallel reactions,

[2] Snippets allow the modularization of test scripts by grouping testing operations.

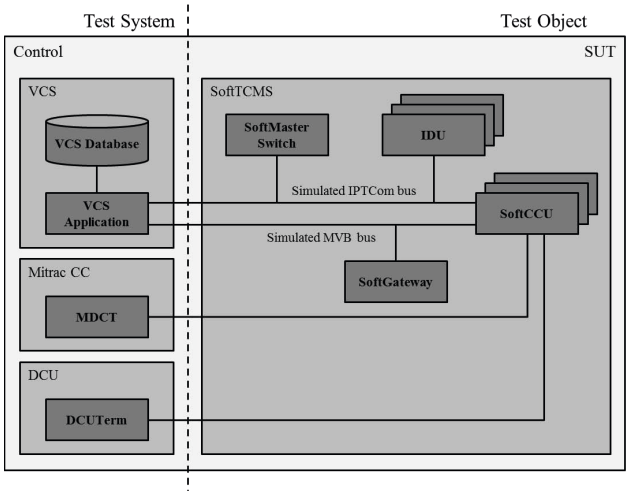

Fig. 3. General architecture of the simulated environment

the displaying of comments, etc. The Test Engine (TE) is in charge of running test scripts against the SUT. It consists of a GUI application that initializes the SUT and test scripts, and a library containing validation rules to compare testing against expected results. Therefore, TE takes care to perform actions against the SUT and verify the correctness of SUT reactions. In this respect, DCUTerm and DCUIP are exploited to access TCMS's built-in logging functionality for monitoring the status of signal values at millisecond resolution, and to generate test reports for each test script execution in various formats (i.e., PDF and XML documents), respectively. Finally, the Test Execution Manager (TEM) is used to create advanced testing workflows, notably collections of one or more test scripts scheduled by following a user's defined logic known as sessions. In the session the user can set a number of repetitions and conditional jumps for each test script, as well as define termination policies.

Test Object is the block that represents the SUT and can be either TCMS or SoftTCMS. TCMS differs from SoftTCMS since the latter is a simulated environment of the former. Test System, excluded the VCS, is the part of the Test Environment that is considered static, since all the included components are invariants of the testing support. Instead, the VCS and Test Object change depending on the project definition and can vary between the real and simulated environments, simply by selecting TCMS or SoftTCMS. In this manner the test object can be totally real, hybrid, or totally simulated.

4.2 Simulated SUT

The simulation of VCUs and the TCMS software loaded into them consists in creating several applications that replicate the behaviour of VCUs allowing the execution of the same TCMS software that is executed in real VCUs (called SoftTCMS in fig. 3). This provides the advantage to have the same application

in both simulated and real environments. In fact, the creation of train software applications is done by means of the same tool, namely MITRAC CC, regardless they are going to interact with real or simulated environments. This has the remarkable advantage that once specified, the application can be compiled to be executed on a general purpose operating system (OS), notably Microsoft Windows in this case, to be simulated, or can be compiled to be executed on a train-specific OS, to be run on real VCUs. Software systems are designed in terms of Programmable Logic Controller (PLC) applications written in the Function Block Diagram (FBD) language specified by the IEC 61131-3 standard [9]. Then, the phases to generate applications for both environments are two and they consist in the generation of the framework and in the compilation. The former addresses the generation of configuration files that are necessary to compile the final application toward the target OS. Once configuration files are ready, the applications for the simulated and real environments are compiled by using Microsoft Visual Studio and MITRAC CC, respectively. Moreover, a glue layer has been developed to make train OS applications compatible with Microsoft Windows OSs and hence properly interface them with the simulation layer.

The information about the status of the train is mainly provided to drivers by using IDUs, while the communication among VCUs, IDUs, etc. and their startup-setup is done by using IP Switches and Gateways. As a consequence, during the realization of the train also IDUs, IP-Switches and Gateways are subject to testing and need to be simulated. IDUs are simulated following a similar procedure of VCUs, whereas IP-Switches and Gateways are simulated by means of a single application that provides network services, such as DHCP, DNS, etc., as depicted in Fig. 3. The use of this application makes easier and faster the preparation of the simulated environment since there is no need to use real IP-Switches and Gateways.

Ultimately, with the aim to improve the debugging and testing experience, some additional tools have been developed, namely DCUTerm and the Mitrac Desktop Component Tester (MDCT). These tools allow debugging and testing whole as well as partial device applications, disclosing the opportunity to perform unit, integration, and system testing at software level.

To summarize, the development of the simulation environment and its supporting applications allowed the remote control and management of the real hardware, avoiding the uncomfortable need to have it on developer and tester's desks. In this respect, we discuss a sample testing scenario in the following, in order to illustrate a typical testing scenario in BT and clarify the potentials of exploiting simulated environments.

4.3 A Sample Testing Scenario

The example shown in Fig. 4 illustrates an excerpt of a test case that serves to verify the correctness of the closing of load shed contactors. It consists in verifying a series of requirements that are part of the Technical Requirements Specification, localised at system level. The approach that is used at this test

Action	Reaction
- Activate a cab. - Deactivate HV and remove external 3-phase. - Log battery voltage and current.	- Check that the battery chargers are indicated as not started on the IDU. - Check that battery voltage, current and temperature for each battery are shown on IDU. - Check that battery currents are negative as no charging is active. - Check that battery voltage for both battery buses are shown on IDU. - Check that event "Battery charger does not start" is not set, wait for 60 seconds.
Wait 5 minutes with battery discharging	- Check that the BC4 contactors are opened after 5 minutes with discharging current more than 10A. - Check that the battery discharging current is not less than 10A as then the limit for detecting discharging is not correct. - Check that no incorrect events are set when BC4 contactors are opened.
Wait total 30 minutes with battery discharging	- Check that the BC3 contactors are opened after 30 minutes with discharging current more than 10A. - Check that the battery discharging current is not less than 10A as then the limit for detecting discharging is not correct. - Check that no incorrect events are set when BC3 contactors are opened.

Fig. 4. A test case excerpt (from ComTS-IVVP-0181)

level is requirements based, which means that all test cases are created against specific requirements.

A typical test case is made up of a sequence of steps composed of actions and reactions (see Fig. 4). Moreover, all actions and reaction listed in the test cases are available in the TAF to design corresponding test scripts [3]. This means that each test script reflects exactly a test case and vice-versa. A further benefit that this approach gives to testers is the possibility to first create test scripts that can be immediately tested against the system and then create "official" test cases for documentation purposes.

The explanation of the whole test case goes beyond the purpose of this work; what is worth noting in Fig. 4 is the frequent exploitation of timed events, notably deadlines, delays, and so forth. The combined use of automation testing and simulation gives to testers the possibility to perform "short-cut" actions during tests in order to reduce the execution time. For example, when a test case introduces delays due to hardware parts, they can be easily reduced by performing additional actions against the system. The benefit of this approach is the possibility to cut the testing execution duration while preserving its reliability. In fact, if a tester has to check that the BC3 contactors are opened after 30 minutes with the discharging of the current of more than 10A, she/he can avoid waiting 30 minutes in the simulated environment by introducing an action that speeds up the discharging of the battery in the test script.

In the following, a deeper discussion of the simulation techniques is provided, in order to better illustrate available features and challenges faced in their realization.

[3] We refer to test cases for the manual testing and test scripts for automated testing.

5 Towards a Completely Simulated Train

The simulation of the TCMS has been a long-lasting and intense activity demanded an in-depth analysis and understanding of how to create an appropriate simulation and how to make the simulation as much as possible close to the real environment. The first simulation step has been addressed with the VCS, which is the control simulator of all TCMS subsystems, including physical and electrical models, driver desks (e.g. LEDs, buttons, hand levers, displays, etc.), and IO modules (i.e. sensors, actuators, etc.). The simulation of the VCS reproduces all possible train controls and signals by using graphical panels, with the purpose to enable users to control and manage the TCMS system by using GUI simulators instead of real hardware. Furthermore, the VCS simulator has been designed and developed to be reusable across different projects making minor adjustments. In this respect, a database has been created where information about train controls and signals are contained. The database is not common across train projects, rather it is project specific; however, it allows to decouple the VCS GUI from the corresponding signals exchanged when acting on the VCS, making the simulation support more scalable and re-usable.

After the realization of the VCS, Bombardier Transportation was able to run a hybrid system made up on the interaction between the VCS and VCUs. The non-simulated part of the hardware was still built up in rack cabinets and connected with the VCS through IO boards. Although this was a relevant step forward to speed up the testing, availabilities and costs to build up rack cabinets still represented a relevant limitation to perform parallel tests. Furthermore, the application update on TCMS devices still constituted a bottle-neck in the testing procedures. In fact, that is a complex and time-consuming process due to a careful (and often tricky) management of safety devices and because of the update speed. The application update requires a specific process that takes about 30 minutes for each safety device and only a bit less for non-safety devices. Furthermore, for safety devices if the update is not performed by strictly following a step-by-step well-defined procedure and respecting all its pre-conditions, it is very likely to get a failure during the update. This is mainly due to the fact that the update has to take into consideration a lot of information of the device, such as version match among software components, Cyclic Redundancy Check (CRC) of files, and so forth. As a consequence, by considering the complexity of the train architecture, the update of all installed devices is a procedure that takes several hours with a high probability of making mistakes during the update. Even worse, taking into account the typical change rate of VCU applications during the train development process, TCMS updates constituted one of the main time costs in debugging and testing activities.

As an additional remark, the difficulty or impossibility to debug VCU applications during their execution in the real environment contributed to make the work for developers and testers harder. In particular, because of timing problems when applications run asynchronously it is very difficult (and even not possible at all) to perform tests on time-dependent actions and reactions, especially when several events are expected to happen in parallel. As a consequence,

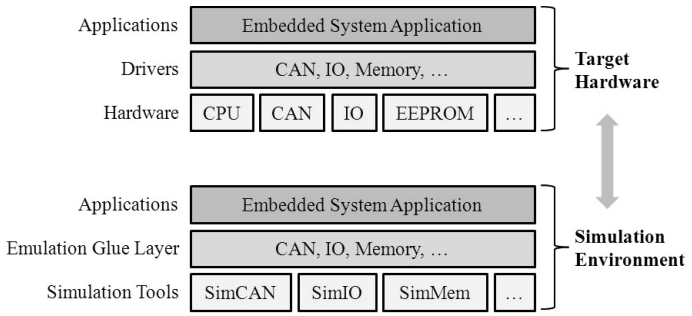

Fig. 5. Architecture of the Hardware Abstraction Layer [6]

Bombardier Transportation decided to simulate the remaining TCMS hardware units and communication buses with simulators compiled for Microsoft Windows OSs. Nowadays Bombardier Transportation is able to run the whole train system on a computer, which marked a major turning point in the context of application development and testing for TCMS. In today's practice, the main goal for this company is to set up a test system able to perform the automated regression testing on the realized simulated environment.

5.1 Architecture

In general, when a real environment is replicated with a simulated one, the main goal is to make the simulation as accurate as possible, in order to get reliable results and reduce efforts due to tuning the environment itself. In this respect Bombardier Transportation and MAXIMATECC adopted a well-defined layered architecture rather than simply developing simulators specific to each project, since empirical experience has shown its efficacy. In particular, MAXIMATECC proposed a Hardware Abstraction Layer (HAL), that is a simulation architecture where each layer and its components are simulated by preserving the same characteristics and intercommunications of the real environment. Fig. 5 depicts the core concept of the HAL: it mainly consists of three layers that represent different levels of abstraction, namely software, driver, and hardware.

The Applications layer comprises a single component that is the Embedded System Application. It represents the applications that are executed on the real environment as well as on the simulated environment. It is worth to recall here that this layer remains unchanged in both environments, making an application developed and tested with a simulated environment ready to be executed on the real environment. The layer of Drivers includes the Controller Area Network bus, IO interfaces, memory, and so forth. It is replaced in the simulated environment with the Emulation Glue Layer, which is a connection wrapper between Applications and Simulation Tools. Finally, the Hardware layer comprises the CPU, Control Area Network (CAN) connection, IO interfaces, Electrically Erasable Programmable Read-Only Memory (EEPROM), etc., and it is replicated component by component in the simulated environment with simulation tools.

5.2 SimTecc

One important issue to address when executing a real time system in a simulated environment is the timing. Usually, the execution of software simulators is faster than the real hardware, so it is tricky to get an accurate replica of the real environment timing behaviour in a simulated one. In general, a simulated test environment should have a predictable behaviour, independent of the computer performance, and exploiting breakpoints without affecting test results. Recent experiences in testing in simulated environments show empirical evidence that usually a small amount of bugs are related to timing issues. These bugs are typically more difficult to solve rather than logical bugs, therefore there exists a general pressure on tool providers to invest time and money for improving the time management accuracy.

In order to face the timing issues mentioned so far, MAXIMATECC has developed a platform that enables the simulation of hardware and buses on ordinary computers running Microsoft Windows OSs. This platform, called SimTecc, is mainly designed for testing distributed embedded systems [6]. MAXIMATECC addressed timing issues by developing a software component called TimeSync that operates as a clock and scheduler for simulated environments. The idea behind the development of this clock is to keep aligned the tick pace of all simulators and offer the possibility to *speed-up* the simulation according to the availability of operating system resources. To take advantage of TimeSync, all simulators present in the simulated environment use this software component by referring to its time instead of the one provided by the operating system. Fig. 6 shows an example of how two threads are scheduled in the real environment and in the simulated one by using the normal speed mode and full speed mode of TimeSync. In particular, in the simulation the tasks are not executed in parallel, rather the simulated clock is not advanced until all tasks scheduled for a certain point in time have been executed. In this way, the applications will behave as if the execution happened in parallel.

The use of the clock can be done in two different ways: using sleep functions or polling the current time. The main difference between them is that the former is a blocking action, whereas the latter is not. When a blocking function is called, a new target time is calculated by adding the sleep time to the target time. After the update of the time, it is work of the scheduler to compare the target times and perform the task with the lowest target time. Furthermore, before the task is executed a check against the system clock is done to decide if some idle time is required or if the task has to start immediately. This clock has the special benefit to operate in two different modes: normal speed and full speed. The former sets its tick pace as the one of the system; instead, the latter sets and changes continuously its tick pace in accordance to the availability of system resources. This means that if the system has free resources, TimeSync speeds up its tick pace with the aim to perform a faster simulation, whereas it slows down the tick pace if the system has limited resources. In this way it is possible to guarantee that the performance of the simulation and the corresponding test results are not affected by system slowdowns.

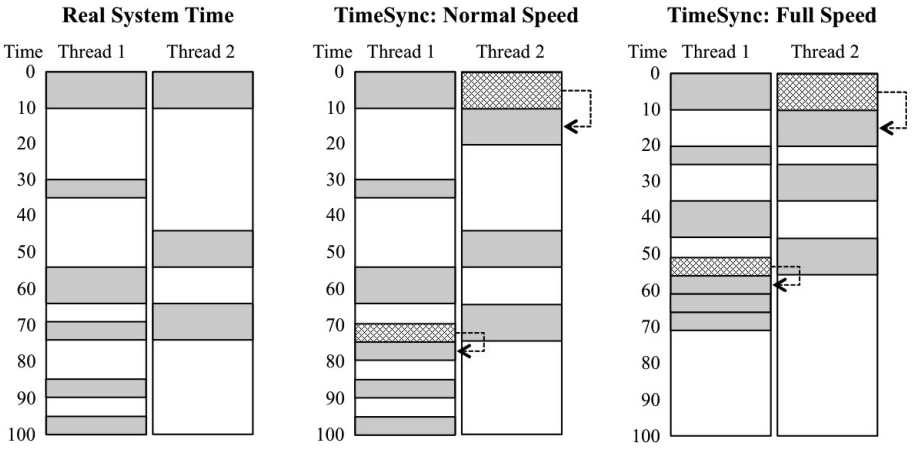

Fig. 6. Example of thread scheduling in the real and simulated environments

When automatic testing is performed in a simulated environment, there is no need for extra delays where no task is executing. So, the full speed mode is very useful to accelerate the testing by skipping the idle times and directly performing the next pending tasks. As a consequence, tasks that only need to check that a certain amount of time has elapsed, such as delays, can be realized as non-blocking functions. Another useful feature disclosed by TimeSync is the possibility to break and resume the simulation by stopping and resuming the clock and then providing a predictable behaviour when test applications measure the time. This feature allows developers and testers to break the execution of the simulation for debugging the code and then restart it without facing time-out issues. To summarize, TimeSync not only allows to speed-up testing and debugging phases by effectively exploiting resources available for the simulated environment, but also permits to perform actions that would not be possible in a real environment, notably stopping and resuming a testing procedure.

6 Performance Analysis

This section presents the results of a comparison done by performing selected test cases defined for a real train project. Each test can be decomposed in the following main phases: instrumentation, initialization, execution, and result storage. In Table 1 are shown testing times[4] devoted to manual testing and automated testing, both in the real (TCMS) and simulated environment (SoftTCMS). Each row represents a test case (the excerpt shown in Sect. 4.3 is ComTS-IVVP-0181 in the table) while the corresponding columns illustrate the time required to execute each test case both manually and automatically, and in both real and

[4] For the sake of space only the total testing time is shown, that is the time elapsed from instrumentation to result storage.

Table 1. Time costs of manual and automated testing performed on the TCMS and SoftTCMS

	Manual Testing		**Automated Testing**	
Test Cases	**TCMS**	**SoftTCMS**	**TCMS**	**SoftTCMS**
ComTS-IVVP-0179	00:10:00	00:08:48	00:03:14	00:02:04
ComTS-IVVP-0181	01:00:00	00:58:48	00:07:59	00:07:01
ComTS-IVVP-0183	00:10:00	00:08:48	00:05:41	00:04:32
ComTS-IVVP-0186	00:30:00	00:28:48	00:07:29	00:06:26
ComTS-IVVP-0188	00:10:00	00:08:48	00:03:19	00:02:16
ComTS-IVVP-0189	01:00:00	00:58:48	00:08:19	00:07:19
ComTS-IVVP-0194	01:00:00	00:58:48	00:02:11	00:01:05
ComTS-IVVP-0199	00:10:00	00:08:48	00:05:00	00:03:39
ComTS-IVVP-0205	00:30:00	00:28:48	00:06:08	00:02:57
ComTS-IVVP-0213	00:20:00	00:18:48	00:03:33	00:02:26
ComTS-IVVP-0214	00:10:00	00:08:48	00:03:21	00:02:15
ComTS-IVVP-0216	00:10:00	00:08:48	00:03:19	00:02:19
ComTS-IVVP-0225	00:05:00	00:03:48	00:02:21	00:01:11
Total	05:25:00	05:09:24	01:01:54	00:45:23

simulated environments. The times shown in the table, which are referred to the automated testing, are performed setting the TimeSync to normal speed.

By looking deeper at table data it is noticeable the significant difference between manual and automated testing, as well as between the operation on TCMS or SoftTCMS (see the row "Total"). In particular, in this case study the improvement achieved by using the automated testing is over 500%, which means that by following this approach the company is able to use less than 5 times the personnel involved in testing, or to quintuple the testing. Therefore, only by changing the testing process and keeping the same human resources, the software quality is expected to increase. Notably, in the current practice regression testing is a feasible activity that is performed each time changes are operated on the train application. The same practice could be hardly adopted without automated testing support since each re-testing would have required a couple of days to be completed. As a matter of fact, a quite widespread practice was to test changes only locally (e.g. the function subject to modifications) with the relevant risk of having a multitude of bugs appearing at system integration time.

Going down in detail and comparing TCMS versus SoftTCMS results, it is possible to notice that the difference between them is minimal, since they only differ for the re-initialization time that is typically faster when done virtually rather than on real hardware. Concretely, it takes about 1 minute and 52 seconds in the former case and only 45 seconds in the latter. Differently from manual testing, in automated testing some tests performed on the TMCS are twice slower than those performed on the SoftTCMS, even by setting normal speed for the TimeSync component (see Section 5.2) as in this case. This is due to the fact that the combination of automated testing and simulated environment tends to speed-up the testing, for example by shortening instrumentation phases.

Therefore, setting the TimeSync in full speed could have further improved test performances by a 200% factor, that is both manual testing and automated testing in the SoftTCMS would have taken half the time.

Finally, it is essential to remark that the comparison provided in the table above is only related to the time spent for a single test station. The benefits in terms of time and cost savings to reach the same coverage level are not taken into consideration at all. In fact, as aforementioned in this paper, the combination of the test system with SoftTCMS can be entirely executed on a computer, whereas the combination between the test system and TCMS needs a computer for running the test system and train hardware for running the TCMS. In this respect, Bombardier Transportation already equipped its testing laboratories with a dozen of simulation workstations, pursuing a drastic cut of testing costs.

7 Conclusions and Future Directions

Automated testing and simulated environments have been often considered premature for being used in industrial areas, especially when dealing with safety critical applications. However, nowadays' market expectations made their usage a way to pursue in order to reduce costs and time-to-market while still preserving (if not enhancing) the delivered products quality. As illustrated in this paper, automated testing and simulation environments are two orthogonal strategies that can support companies in matching market demands. However, they require a (possibly) long process in order to deeply analyse the domain and derive optimal automation and simulation solutions. In other words, the benefits granted by simulation and automation are the rewards gathered after a significant initial investment. On the other hand, when finally established those techniques allow companies to dramatically improve their testing performances, and as a consequence, their efficiency and efficacy in the application development. As a matter of fact, test automation permits companies to save relevant amounts of human resources that can be exploited in creating test cases and increasing the test coverage, for instance.

As future enhancements, Bombardier Trasportation and MAXIMATECC are investigating the extension of testing support to non-functional properties. Notably, they are working on the realization of fault injection [7] features to be added both in the automation testing and simulation tools in order to test fault-tolerance of system communications and applications. Moreover, in a long term view they envision the complete integration of Model-, Software-, and Hardware-in-the-Loop techniques to maximize the benefits of distributed development [8].

References

1. Kolawa, A., Huizinga, D.: Automated Defect Prevention: Best Practices in Software Management. Computer Society Press (January 2007) ISBN 0-470-04212-5
2. Jin, W., Orso, A., Xie, T.: Automated Behavioural Regression Testing. In: 3rd International Conference on Software Testing, Verification and Validation (2010)

3. Nguyen, H.Q., Hackett, M., Whitlock, B.K.: Global Software Test Automation: A Discussion of Software Testing for Executives. Happy About (July 2006) ISBN 1-60005-012-3
4. Rafi, D.M., Moses, K.R.K., Petersen, K., Mäntylä, M.V.: Benefits and Limitations of Automated Software Testing: Systematic Literature Review and Practitioner Survey. 10th International Workshop on Automation of Software Test (June 2012)
5. Bombardier Transportation, Train Control and Management System (July 2013), http://www.bombardier.com/en/transportation/products-services/propulsion-controls/products/train-control-and-management-system.html
6. MAXIMATECC, SimTecc Tutorial (2013)
7. Carreira, J.V., Costa, D., Silva, J.G.: Fault Injection Spot-Checks Computer System Dependability. IEEE Spectrum, 50–55 (August 1999)
8. Opal-Rt Technologies, About Hardware in the Loop and Hardware in the loop Simulation (July 2013), http://www.opal-rt.com/about-hardware-in-the-loop-and-hardware-in-the-loop-simulation
9. International Electrotechnical Commission, Programmable Controllers - Part 3: Programming Languages. IEC 61131-3, (February 2013)
10. Wong, B., Verner, J., Chulani, S., Boehm, B.: Third Workshop on Software Quality, ICSE 2005, pp. 688–689 (May 2005) ISBN 1-58113-963-2
11. Bombardier, About Us (November 2013), http://www.bombardier.com/en/about-us.html#1361982849731
12. Bombardier, MITRAC Game-Changing Electronics (November 2013), http://www.bombardier.com/content/dam/Websites/bombardiercom/supportingdocuments/BT/Bombardier-Transportation-MITRAC-Game-changing-Electronics.pdf
13. MAXIMATECC, About (November 2013), http://www.maximatecc.com/en-US/About.aspx
14. Neil, G.: On Board Train Control and Monitoring Systems. Electric Traction Systems, 211–241 (November 2006) ISBN 9780863417522
15. Zhu, F., Rayadurgam, S., Tsai, W.-T.: Automating regression testing for real-time software in a distributed environment. Object-Oriented Real-time Distributed Computing, 373–382 (April 1998) ISBN 0-8186-8430-5
16. Alegroth, E., Feldt, R., Olsson, H.H.: Transitioning Manual System Test Suites to Automated Testing: An Industrial Case Study. In: Software Testing, Verification and Validation (ICST), pp. 978–971 (March 2013) ISBN 978-1-4673-5961-0
17. SmartBear Why Automated Testing? (June 2014), http://support.smartbear.com/articles/testcomplete/manager-overview/