Testing complex safety-critical systems in SOA context


Abstract—Due to its simplicity and ease of application, testing is the main technique by which complex safety-critical systems can be verified in order to find both omission and commission bugs. Strict requirements on such systems, joined to the necessity to re-execute the test set in the regression testing campaign, provokes a test case set and testing time explosion that can be tackled only by means of the use of parallel independent testing environments. Parallelism in such environments is not easy to accomplish due to the heterogeneity of processes, methodologies and tools. Service Oriented Architecture (SOA) is a key factor in the development of an organic modelling and execution methodology in order to build a heterogeneous and distributed environment that supports a system testing. In this paper we propose an adoption of a classical SOA reference architecture in order to address the build of such an environment for safety-critical control systems. Moreover we provide indications on the integration of SOA specific architecture components with existing centralized testing environments providing an example in signalling railway control systems.


I. INTRODUCTION

Safety-critical real-time systems require thorough Validation and Verification (V&V) activities [1], often regulated by international standards (e.g. CENELEC [2]-[4] for railway signalling systems), aiming to guarantee the correctness of the system and its compliance to functional and safety requirements. In V&V activities a relevant role is covered by the testing activity [5], due to its simplicity and ease of application.

Requirements for control applications are often very complex, demanding heterogeneous and highly distributed systems. So a very intensive testing activity is required, involving a high number of testing scenarios (usually more than ten thousand tests) and a high number of testing phases following the whole development cycle. For a safety-critical system, the number of test cases grows because of the need to find commission bugs that are not usually discovered by tests only based on functional requirements.

When the system is required to be both complex and safety-critical, the system testing becomes a difficult and time consuming process, which would be almost infeasible without advanced test specification methodologies and powerful execution and tools.

To be noticed that a safety-critical control system typically manages equipments which are capable to cause damages, and its malfunction may therefore lead to hazardous situations and possible accidents. System must therefore achieve the required reliability the first time it is put into operation. This gives few chances to test the system in its real operational environment without threat for human lives and properties. For these reasons, the testing of such systems is widely based on simulated environments. Simulated environments are also used to allow the test execution activity as soon as possible in early stages of the development cycle, when a prototype of the system is not yet available.

In our industrial experience, the test execution activity on railway interlocking control systems is supported by a powerful testing environment allowing:

1. tests definition by means of abstract (rules based) test scripts, automatically instantiated at runtime for each occurrence of the test reference scenario in the considered plant;
2. automatic test execution (in batch mode) of the defined abstract test scripts on both the target system and an its simulated version;
3. test logs generation.

In this way, the test specification activity can be performed just once for all railway interlocking systems using same functional requirements (the same safety logic). In fact, same abstract test scripts are configured, at runtime, depending on the particular rail yard, i.e. the same abstract test script is executed for each occurrence of the test reference scenario in

Manuscript received November 12, 2007. This work has partially been supported by AnsaldoSTS.

S. Marrone is with Second University of Naples and with AnsaldoSTS (phone: +39-081-2437057; stefano.marrone@unina2.it).

R. Donini, A. Orazzo and D. Papa are with AnsaldoSTS (phone: +39-081-2432366; donini.renato@asf.ansaldo.it, orazzo.antonio@asf.ansaldo.it, papa.domenico@asf.ansaldo.it).

N. Mazzocca is with University of Naples “Federico II” (n.mazzocca@unina.it).

S. Venticinque is with Second University of Naples (salvatore.venticinaque@unina2.it).
the considered rail yard.

At this moment, using the simulated environment and the described testing environment, abstract test scripts are automatically executed, at the same time, on different railway interlocking systems for different rail yards, via parallel independent replicas running on different workstations.

The high number of test cases produced by means of such an approach can be very time critical if not supported by an efficient management of computational resources in order to guarantee a correct and an efficient dispatching of tests on different workstations, a good load balancing algorithm and a test logs collection mechanism.

Distributed environments allow to overcome difficulties traditional testing environments are affected from. Service Oriented Architecture (SOA) platform was born to cope with these problems of distributed systems [9]. In fact main features of this architectural platform are the performance improvement, flexibility, scalability, ability to cope with heterogeneity and ease of load balancing.

Despite of generic features offered by several available distributed middlewares as High Level Architecture (HLA) [6], SOA adds other ones highlighting its main role in this topic.

SOA is an architectural style for building software applications that use services available in a network such as the web and it represents the widest accepted model to design geographical distributed systems. It promotes loose coupling between software components so that they can be reused [13]. Applications in SOA are therefore built basing on services. A service is associated to an implementation of a well-defined business functionality, and such services can then be consumed by clients in different applications or business processes. It will be clearer that such a feature is a key factor in our case study.

Nevertheless SOA has been used successfully in several information technology contexts, it is not immediate to understand the motivations for which such an approach can be used in critical systems testing processes. In fact we can naturally imagine that SOA features do not affect testing problems, but several motivations can be adducted to enforce this choose. Firstly, modern complex and critical systems are more rarely built by a single company: several such systems are in fact built by means of temporary aggregations of companies that share a common objective and collaborate each others with smaller subsystems in order to provide the whole system to the customer. We remind the case of European Railway Traffic Management System/European Train Control System (ERTMS/ETCS) railway signalling systems (for example Italian High Speed Lines) where the managerial and technical complexity of the project addresses the decomposition in subsystems and the assignment of such subsystems to several companies. For this kind of systems a simulation-based system testing was practically impossible due to the absence of a shared and interoperable simulation and testing environment. This aggregation is, in practical sense, a Virtual Organization (VO) [7] for which the benefits of the application of a SOA platform have already been shown [14]. This fact addresses the creation of distributed interoperable testing systems that are based on an universal reference architecture.

Moreover SOA architecture can dramatically increase the flexibility of testing environments and consequently reduce testing times due to the absence of manual reconfiguration of these environments even in a single-company context.

This paper is organized as follows: Section 2 sets the contribution of this paper in the context of related works; Section 3 provides a description of traditional functional system testing environments; Section 4 describes our SOA-based architecture for remote testing whereas Section 5 covers implementation issues by means of a simple example; conclusions and future works are contained in Section 6.

II. RELATED WORKS

Several middleware platforms have been proposed in literature in order to tackle this problem. The first consideration to do is the difference between the remote and distributed testing. In the first case, each test is performed by a single independent environment and, cause of the presence of several of such networked environments, testing could benefit by this kind of parallelism. The distributed environments are characterized by the presence of concurrency because a test can require the performance of more than one single environment and communication and synchronization between environments can be necessary.

SOA has just proved its applicability in industrial contexts and the benefits of such application have been already described. In [8] Komoda resumes pros and cons of introducing SOA platform in an industrial context. This paper is more focused on real-time control systems and not on the testing of such systems but, in our opinion, control and testing environments share several of such issues. Komoda highlights these key factors: response time (several SOA implementations suffer of lack of responsiveness), support of event-driven operations, complicated human interface support, reliability and test support functions.

Integration of existing applications in SOA context is a software engineering problem that has been already explored. Several works ([10] and [11] for example) have given a deep analysis of these problems defining several techniques to wrap legacy applications into a Web Service in order to reengineer such systems by means of SOA.

In the specific field of integration between software testing environments and SOA platforms there are few research works: Ding Z. et al [16] integrate software testing environments into the Grid. With respect to this approach we want to realize a more service-oriented framework that can address differences among service providers both in functional and in non-functional testing features.
Another category of works brings testing and simulation in distributed environments by means of a middleware. Such a middleware has to match requirements that distributed testing systems’ design addresses. Peng [12] described some of these requirements stating that a layered software is the key to combine both interoperability and scalability of these environments.

One of the most valuable milestone in this context remains the IEEE Std 1516 that illustrates the High Level Architecture (HLA) [6]. The aim of this reference architecture is to define a common framework in which existing simulators can work together over a distributed network by means of simulators wrapping into a newer interface. Inside this framework, simulators (“federates” in HLA jargon) can participate to a “federation” by means of publications of provided services. Another HLA component, the Run Time Infrastructure (RTI), is capable of a correct dispatching of messages among federates and other HLA facilities allow a federation to accomplish its objective with the respect of several non-functional requirements as real-time response, reliability independence from the location and scalability. Despite its ambitious objectives, HLA is used in a limited context as the military one. Other approaches to this problem, like SOA, are easier to be implemented and provided, often fulfilling non-functional requirements too.

Due to these considerations and to the characteristic of the objective of our testing environments we decided to apply SOA paradigm to distribute it over a network and for proper model testing process as services.

III. FUNCTIONAL TESTING PROCESS

A control system can be identified as safety-critical if its failure could result in one or more of the following consequences:

1) loss of human lives;
2) injury or illness;
3) serious environmental damages;
4) significant loss of, or damage to, properties;

In fact, such a system typically manages equipments which are capable of causing damages. Malfunction may therefore lead to hazardous situations and possible accidents.

Due to these delicate aspects, for each application field some standards about the development life cycle, including testing activities, are defined (e.g. CENELEC for railway applications [2]-[4]). The uncertainty associated with both the adequacy of the requirements specification and with the ability to translate these requirements into a final product implies that some forms of testing for the verification and validation of design at each development stage have to be considered.

In the aforementioned context functional testing covers a relevant role, aiming to guarantee the compliance of the system with its functional and safety requirements. The scope of this work is confined to such kind of tests.

The majority of industrial safety-critical control systems are very large and complex, therefore they often require a great amount of test cases to obtain an adequate confidence on their correctness. It is often difficult to obtain adequate realistic test cases to represent system real operation, firstly because the testing has to be as soon as possible performed, even if a system prototype is not yet available. The second difficulty stems from the fact that malfunctioning of such systems could cause a grave danger to human life and property. The system must therefore achieve the required reliability the first time it is put into operation. This gives few chances to test the system in its real operational environment without threat of danger.

System testing via environment simulation is an approach to overcome such difficulties. The greatest advantage of the method is that the system can be tested by means of highly realistic test cases, but without threat of danger to human life and property. Another attractive feature of the approach is that the system can be tested not only in the normal operation of the environment but also in adverse conditions representing the non-healthy operation of the environment. Thus the robustness and tolerance of the system to an unexpected system behaviour can be assessed. Such testing could be too hazardous to be carried out in the real environment, and could indeed be impossible because the design of the plant or equipment should have minimized the probability of the occurrence of hazard situations. For example, in nuclear installations it is normal for systems to be designed to protect the plant from very low frequency fault situations that may never occur during the life time of the system. But, for safety-critical systems testing in adverse conditions is essential. Thus, in these cases, simulation of the possible events is necessary if a realistic testing has to be performed. Environment simulators are generally used for the high level testing of a system, such as that performed during software and hardware integration or system integration. Last but not least, via simulated environment it is possible to speed up testing activities, using parallel independent simulated environments deployed on different workstations.

Our industrial experience is on railway interlocking control systems. Functional Testing, for such systems, is based on the Testing Environment briefly schematized in Fig. 1. It is composed by:

? a set of External System Simulators, totally or partially representing and simulating the environment interacting with the System Under Testing (SUT);
? a Tester linked to the Man Machine Interface (MMI) and to the External System Simulators.

In the scheme, the SUT is represented by the Computer Based Control System (CBCS), running the Control Software
and interacting with:

- Operators by means of the MMI;
- Physical entities (External Systems) of rail yards by means of sensors and actuators, represented as External Systems Interface in Fig. 1.

In the scheme the CBCS block can indifferently be the target control system or a simulated version reproducing its functional aspects. In this way, it is possible to execute tests on either fully or partially simulated environment.

An example of External System Simulator for railway interlocking applications is the Field Objects Simulator, simulating the behaviour of field objects (e.g. Points, Track Circuits, Coloured Signals, and so on).

The Tester is the block of the testing environment that coordinates other components (CBCS, External Systems Simulators) during the automatic execution of tests that are coded in Test Scripts.

As said before, different replicas of the testing environment can be deployed on different workstations, as depicted in Fig. 2. Each block represents a workstation running an instance of testing environment, executing in batch the whole test set, or part of it, on a particular rail yard interlocking system.

The whole set of testing workstations is manually managed by a Test Execution Manager. He usually has to dispatch the tests on the available workstations, trying to balance the load assigned to each of them, in order to reduce the tests execution time. In particular, having \( n \) replicas of the testing environment for the Rail yard \( i \) interlocking system, the whole test set is parallelized using such replicas in order to speed up as much as possible the tests execution activity.

Limits of such an approach appear immediately clear. First of all, the Test Execution Manager doesn’t continuously check, i.e. 24 hours a day, available resources. Moreover, the Test Execution Manager is usually not reactive regarding the resources’ load. For these reasons, there are limits in terms of:

- continuity of tests execution activity;
- optimal load balancing of available resources;
- efficient using of available resources.

Therefore, the use of simulated testing environments allows to increase the parallelism of the test execution but, nevertheless, the problem is still open and it can be resumed in these terms:

1) A huge amount of tests have to be automatically executed by the test environment;
2) Some workstations are available in our labs, each one supporting testing environment for different rail yards, with different data versions;
3) Some people manage workstations and manually allocate tests on them.

We have just presented a testing environment that can be positively affected by the passage from a local to a remote and parallel testing paradigm. In our industrial experience there are other railway signalling systems that would require a distributed testing approach, as the ERTMS/ETCS one [15] or as the evolution of the described one that takes into account interactions between adjacent plants. For sake of simplicity, we have started to consider only the case of a remote testing environment. A more proper distributed approach, extension of the proposed architecture, will be the
subject of further research.

IV. THE OVERALL ARCHITECTURE

In our intention, the proposed architecture would answer to the problems of the remote testing environment needs and would be the base of a future extension in the distributed solution direction. On the basis of already stated, it should be clear that a Service Oriented Architecture (SOA) well addresses these problems.

The main feature of SOA is the re-use and management of services, implemented by already available tools, in order to offer a high level service interface, uncoupled by the service implementation.

SOA uses the Publish-Find-Interact (PFI) paradigm as shown in Fig. 3. In this paradigm, service providers register their services in a public registry (Discovery Agency). This registry is used by the consumers to find services matching certain criteria. If the registry has such a service, it provides the consumer with a bind (contract) and an endpoint address for that service [13].

![Fig. 3: SOA Publish-Find-Interact Model [13].](image)

Referring to this architecture some natural associations with our problem domain appear natural: equipped workstations are in fact service providers whereas tests to be executed are services requested. All tests shall be managed by an automatic Testing Environment Interface (Test Manager) that can be considered the Service Requestor, as in Fig. 4.

![Fig. 4: Introduction of the Test Manager.](image)

To better allocate tests on a particular workstation, the Test Manager needs to know if that workstation can satisfy that request. To allow this agreement it is necessary that each service provider is able to promote its features. This is possible by means of a Service Registry and by means of a formal interface published on it by Service Providers.

Each test can be considered as the service to be delivered and each testing environment as a service provider. Referring to the Fig. 5 the test scripts are submitted (1) to the Test Script Manager (the Service Requestor) and before running a test script the client sends a request (2) to the Service Registry (Discovery Agency) within the test profile, that by means of a query operation looks for services that match test’s requirements and retrieves the service binds (3). Then the Test Manager refines the result set in order to select services that better fit the test profile (4).

The Test Manager needs to know the state of each server to be used and, for this reason, it books itself on these servers aiming to be dynamically informed about their status (busy, idle). Each server replies to the Test Manager’ booking request sending its own state (6); the reply is an acknowledge too. The server status shall be refreshed, in a fixed timeslot, at each status modification.

When at least one server is available to perform the required service the Test Manager exploits (7) the server to run the test (9) and to publish the results (11). Before the test running the server sends, as acknowledge, its busy state to the Test Manager (8), as well as the Test Manager is informed when the test is done by means of an idle message (10). Whenever all target servers are busy the Test Manager puts in a queue the request, then exploits the service as soon as possible. This policy guarantees a load balancing among servers and a full exploitation of them.
To clarify the previous concepts, we will deeper detail them in the following.

Each testing environment service will be published in a Universal Description Discovery and Integration (UDDI) register by means of an UDDI publishing operation. Associate to the registration will be available a service profile that contains the service features.

Testing phase starts with the interaction between client and Test Manager in order to evaluate the service providers that best fit in the user’s needs. This could be provided only after the passage of a test profile from the clients to the Test Manager. This profile contains the service the client wants both from a functional (requested signalling system, railway plant and version of the plant) and from a non-functional point of view (performance indexes and/or level of parallelism of the service provider). This information is used by the Test Manager that searches in the UDDI register for the free testing environment service with a service profile matching the test script profile.

For sake of clarity we introduce the concept of matching policy. A matching policy is a set of rules defining the criteria used to establish the fitness between the offered service profile and the asked test profile. The simplest possible matching policy for our goal is the exact matching of system under test, plant name and version of the plant without considering any non-functional service features. In other words a service can be used by a client only if both the signalling system, name of the plant and version of the plant correspond to the one requested by the client. The study of the mechanisms by which Test Manager and Client can share a matching policy is not in the scope of this paper; we make the hypothesis that Test Manager formally knows the matching policy a Client can ask.

Following the PFI paradigm, the Test Manager performs brokering services for the client in order to redirect the client’s request to the service which has been selected according to the previous matching phase. Test Manager is also responsible for the correct implementation of a load balancing policy.

The third and the fourth phase of a test execution process are respectively the request of the service by the client and the service response by the service providers whose service profile has been chosen by the Test Manager on the basis of the test profile. According to these phases the following issues must be given:

- service providers must be able to accept a test script (implementation of the test case has already stated in Section 3) and run it in the legacy software environment wrapped by the provider itself;
- results of the test execution has to be published on a web site in order to enable off-line log analysis. In this way also the customer could monitor the V&V activity progress;
- service providers must notify the end of such test execution in order to free providers’ resources or to let client to submit another test;
- service providers have to be able to manage testing fault conditions in order to allow Test Execution Manager to find the proper action in order to resolve these conditions.

Moreover Service Providers should publish a reset service in order to overcome a faulty condition that could bring the simulated plant into an unknown state and so that could restore initial plant state.

Another side-effect of the introduction of such an environment is the resolution of logistic problems by means of the physical distribution of resources among laboratories.

As stated in [9] SOA is not tied to any specific technology; this fact gives to SOA an universal applicability and the capability to be implemented in the way that best fits in performance and technological platforms’ requirements.

Notwithstanding, the most widespread technology adopted in SOA community is Web-Service that can be so considered the standard de-facto to build Service Oriented Architectures. They have been widely accepted because they grant interoperability exploiting standard transport protocols (HTTP, FTP, SMTP) and text based messages. Moreover the availability of several development frameworks aids to a rapid development approach and dramatically decrease the time-to-market for SOA. These are the main reasons we have chosen Web-Services as enabling technology for the implementation of our platform.

V. ON ENVIRONMENT IMPLEMENTATION

In order to better clarify the concepts introduced in the previous section, in this section it will be presented a
candidate simple example about the use of the proposed architecture.

Our example will describe the following items:

- service profile;
- test profile;
- matching policy;

whereas the other two

- service request;
- service response;

will not be described due to their simplicity.

For sake of brevity we will not formally describe the first three items by means of proper formalisms (e.g. Web-Service Description Language et al) but we will concentrate in the content of such items and use intuitive XML-based descriptions.

For instance an XML functional description of the Service Provider is showed in the following:

```
<ServiceDescriptor resID="serverID" servID="serviceID">
  <RailwaySystem name="sX">
    <Plant name="p">
      <Version value="v1"/>
      …
      <Version value="vN"/>
    </Plant>
  </RailwaySystem>
</ServiceDescriptor>
```

This means that a Service Provider, given a railway signalling system (sX) can provide a testing environment for plant p in several versions: v1, …, vN.

On the other hand one way or test profile specification could be the following:

```
<TestProfile clientID="clientID" profileID="profileID" matching="matchingID">
  <TestSet system="sX">
    <TestCase plant="p1" version="v1"/>
    …
    <TestCase plant="pI" version="vN"/>
  </TestSet>
</TestProfile>
```

Finally, we can specify a possible description of the matching policy:

```
<MatchingPolicy ID="matchingID">
  <Rule priority="0">
    <Equality TP_elem="system" SD_elem="system"/>
    <Equality TP_elem="plant" SD_elem="plant"/>
    <Equality TP_elem="version" SD_elem="version"/>
  </Rule>
  …
  </MatchingPolicy>
```

Some words have to be spent to comment the matching profile. In a matching policy several rules can be expressed: according to a specific matching profile, a test profile and a service descriptor can be matched if there is at least one rule of the matching policy that is satisfied. In order to express proper rule policies, rules can be ordered according to the priority field. Inside a rule, each matching condition must be satisfied in order to satisfy the rule. In this example we introduced only the Equality matching condition which means that an element of the test profile (TP_elem) must be equal to an element of the service descriptor (SD_elem). Other matching operators are possible in order to specify more complex rules and so more complex matching policies.

VI. CONCLUSIONS AND FUTURE WORKS

In this paper we have presented a novel approach for a remote and distributed testing based on Service Oriented Architecture. This approach has been adapted to fit to the needs of a real system testing environment for a safety-critical system. Starting from the stated considerations, a reference remote testing architecture upon SOA has been showed. This brought to the fact that the remote testing environments could be quickly built by means of easy integration among web-services and legacy environments and by means of legacy applications wrap.

Further research will quantitatively demonstrate the effectiveness of such approach by means of a performance analysis and numerical comparison between the novel and the traditional platforms on the basis of experimentations. Future works will be possible about the extension of this solution in order to cope with distributed problems and in order to increase internal parallelism of a single service provider (i.e. increasing of internal parallelism of test node). Moreover other works can focus attention on the formalization of the proposed SOA model (test profile, service descriptor and matching profile).

REFERENCES


[2] CENELEC, Railway applications The specification and demonstration of dependability, reliability, availability, maintainability and safety (RAMS), European Committee for Electrotechnical Standardisation, Brussels, EN 50126


[12] F. Peng, Design Considerations for a Distributed Test System, IEEE Autotestcon, 2005

http://www.w3.org/

