Towards a TTCN-3 Test System for Runtime Testing of Adaptable and Distributed Systems

Mariam Lahami, Fairouz Fakhfakh, Moez Krichen, Mohamed Jmaiel

Towards a TTCN-3 Test System for Runtime Testing of Adaptable and Distributed Systems

Mariam Lahami, Fairouz Fakhfakh, Moez Krichen, and Mohamed Jmaiel

Research Unit of Development and Control of Distributed Applications
National School of Engineering of Sfax, University of Sfax
Sokra road km 4, PB 1173 Sfax, Tunisia
{mariam.lahami,fairouz.fakhfakh,moez.krichen}@redcad.org,
mohamed.jmaiel@enis.rnu.tn
http://www.redcad.org

Abstract. Today, adaptable and distributed component based systems need to be checked and validated in order to ensure their correctness and trustworthiness when dynamic changes occur. Traditional testing techniques can not be used since they are applied during the development phase. Therefore, runtime testing is emerging as a novel solution for the validation of highly dynamic systems at runtime. In this paper, we illustrate how a platform independent test system based on the TTCN-3 standard can be used to execute runtime tests. The proposed test system is called TT4RT: TTCN-3 test system for Runtime Testing. A case study in the telemedicine field is used as an illustration to show the relevance of the proposed test system.

1 Introduction

Nowadays, a relevant issue in the software engineering research area consists in delivering software systems able to change their configuration dynamically in order to achieve new requirements and avoid failures without service interrupting. Therefore, they evolve continuously by integrating new components, deleting faulty or unneeded ones and substituting old components by new versions at runtime. Dealing with such reconfiguration actions, the possibility of unexpected errors (components failure, connections going down, etc.) during the reconfiguration process is unavoidable.

Accordingly, a validation technique, such as testing, has to be applied in order to detect as soon as possible such inconsistencies and to check functional and non-functional requirements after each dynamic reconfiguration. Nevertheless, traditional testing techniques cannot be done for these highly evolvable systems since they are applied during the development phase.

For this reason, a recent branch of work has demonstrated the interest of using the *Runtime Testing* as a new solution for the validation of the above systems [1–8]. They have focused on building specific test infrastructures, for instance based on the JUnit Framework. None of them have used a generic test standard like TTCN-3 for the specification or the execution of runtime tests.

Furthermore, they are using at most one technique to isolate runtime tests in the aim of reducing the interference between business and test data. To the best of our knowledge, only one approach presented in [9] uses TTCN-3 standard for online validation and testing of internet services. However, this work did not deal with test isolation issues when testing is applied in the production phase.

This paper makes a contribution in these directions by proposing a TTCN-3 test system for Runtime Testing (TT4RT). The key idea is to extend the reference architecture of the standardized TTCN-3 test system by a new module supporting different test isolation techniques. The latter is a fundamental issue that has to be tackled while executing runtime tests either components under test are testable or not testable. As illustrative example, we describe a case study in telemedicine area called Teleservices and Remote Medical Care System (TRMCS).

The remaining of this paper is structured as follows. Section 2 introduces the runtime testing approach and its challenges that we are facing.The TTCN-3 standard and its key elements are introduced in Section 3. The proposed approach is illustrated in section 4. Section 5 introduces the case study. Some scenarios are illustrated in Section 6. A brief description of related work is addressed in section 7. Finally, section 8 concludes the paper and draws some future work.

2 Runtime Testing Of Dynamic and Distributed Component based systems

Runtime testing is a novel solution for validating highly dynamic systems. It is defined in [10] as any testing method that has to be carried out in the final execution environment of a system while it is performing its normal work. It can be performed first at deployment-time and second at service-time. The deployment-time testing serves to validate and verify the assembled system in its runtime environment while it is deployed for the first time. For systems whose architecture remains constant after initial installation, there is obviously no need to retest the system when it has been placed in-service. On the contrary, if the execution environment or the system behavior and architecture have changed, service-time testing will be a necessity to verify and validate the new system in the new situation [10].

As previously mentioned in the definition of runtime testing, any test method can be applied at runtime such unit testing, integration testing, conformance testing, etc. In our work, we support unit testing as well as integration testing. On the first hand, unit testing is used to validate that the component behavior still conforms to its specification while it is running in isolation in the execution environment. On the other hand, integration testing is used at runtime to validate that the affected component compositions by the reconfiguration action still behave as intended. In order to minimize the number of testers to be deployed and the number of test cases to be re-executed, we apply unit and integration tests only on the affected parts of the system under test by a reconfiguration action. Consequently, the testing effort, cost and time will be reduced [11].

However, other challenges still persist such as test processes interference with the business processes of the running system due to their parallel execution. The best way to resolve such problem is the application of test isolation mechanisms widely discussed in [2, 4] (such as cloning components, adding a test interface, tagging test data, blocking components during test, etc). This challenging issue is resolved in our approach by supporting the well known test isolation mechanisms in the literature. It will be discussed in depth in following sections.

3 TTCN-3 Overview

TTCN-3 (Testing and Test Control Notation Language Version 3) is a test specification language used to define test procedures for reactive black-box testing of distributed systems [12]. This test standard has been widely used in the protocol testing field and is newly addressing other kinds of applications such as service-oriented or CORBA-based systems. It is also suitable for various types of tests such as conformance, robustness, regression and functional testing.

TTCN-3 allows the specification of dynamic and concurrent test systems. In fact, it offers a test configuration system made of two kinds of test components: Main Test Component (MTC) and Parallel Test Component (PTC). For each test case, an MTC is created. PTCs can be created dynamically at any time during the execution of test case. Thus, test system can use any number of test components to realize test procedures in parallel. Communications between the test system and the SUT are established through ports.

The structure of TTCN-3 test system is depicted in Figure 1. It is made up of a set of interacting entities where each one corresponds to a specific functionality involved in the test system implementation. These entities interact together through two major interfaces: the TTCN-3 Control Interface (TCI) [13] and the TTCN-3 Runtime Interface (TRI) [14]. They are briefly described [15] as follows:

- The Test Management (TM) Entity manages the test execution.
- The Test Logging (TL) Entity is responsible for maintaining the test logs.
- The TTCN-3 Executable (TE) Entity executes the compiled TTCN-3 code.
- The Coding/Decoding (CD) Entity encodes and decodes test data types and values.
- The Component Handling (CH) Entity handles the communication between test components.
- The SUT Adapter (SA) Entity implements communication between SUT and test system.
- The Platform Adapter (PA) Entity implements timers and external functions.

TTCN-3 is used in our context to define abstract test suites following the TTCN-3 notation. In this case, test suites are specified at an abstract layer, Abstract Test Suites (ATS). This feature helps to separate test design from test implementation and makes the ATS language platform independent. Furthermore, it increases the reusability of the elaborated test cases. By doing this,

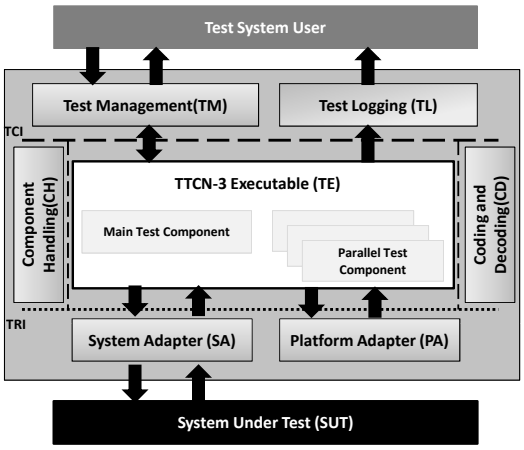

Fig. 1. TTCN-3 reference architecture.

we can address complexity of testing evolvable systems which are also heterogeneous in structure and technologies. Hence, different network and platforms technologies can communicate easily with the TTCN-3 test system through the adaptation layer [16]. The latter comprises three parts of the reference architecture that are Coding-Decoding entity, Test Adapter entity and Platform Adapter entity. These entities provide means to adapt the communication and the time handling between the SUT and test system in a loose coupling manner.

Due to all these features: a standardized, abstract and platform independent test-language and offering a flexible adaptation layer with the aim of facilitating interaction with the SUT, TTCN-3 was adopted in our work and also enhanced to support runtime testing.

4 The proposed Approach: TT4RT

Our main objective is to design and build a test system that handles complexity of testing evolvable and heterogenous (both in structure and technologies) systems. Therefore, we have enhanced the TTCN-3 test system by adding two layers as depicted in Figure 2: a *Test Management Layer* and a *Test Isolation Layer*. The main purposes of these layers are described in the following:

Test Management Layer. It intends to control the execution of runtime tests. It includes a GUI component called *TTmanGUI*. The latter is responsible mainly for starting and stopping test cases. The TTmanGUI interacts with the TM entity offered by the classical TTCN-3 test system in order to achieve the test execution management and also with test isolation layer in order to prepare the test environment.

We have to mention that this layer has as input an XML file which contains the components under test, the test components to deploy and their deployment

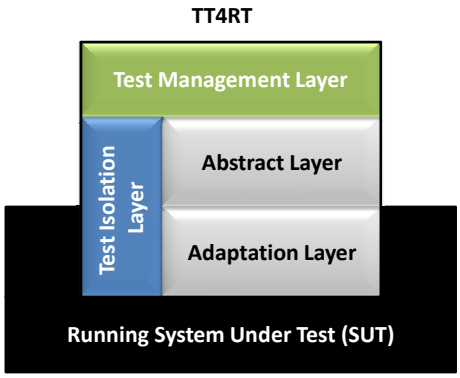

Fig. 2. Supported layers of TT4RT.

hosts as well as the test cases to execute. This file is called *Resource Aware Test Plan* since the assignment of the test components to execution nodes must fit some resource constraints [1]. The structure of this file will be introduced later.

Test isolation Layer. It aims to reduce the interference risk between test data and business data when testing is performed at runtime. It includes a component which is able to choose the most adequate test isolation technique for each component under test. This choice is suggested by using a policy called *Test Isolation Policy*. For each test request, the proposed policy is executed in order to generate the test isolation technique to apply. Our test system supports four test isolation techniques: duplication, blocking, tagging and built-in tests.

For reasons of space, these techniques are briefly introduced through some examples. For instance, if a component is not testable and it is under some timing constraints then it will be automatically duplicated. In this case, the test processes are executed in the duplicate with the aim of not disturbing the execution of the original component. Unless the component is under some timing constraints, it can be blocked until the test processes are achieved. Also, some components can be provided with some capabilities such as testability through a test interface or test awareness through a flag which lets the component under test differentiate between the test data and business data. The proposed policy treats all these conditions and produces the best solution when a test request is triggered.

Abstract Layer. As we have explained above, this layer is offered by the classical TTCN-3 test system in order to build abstract test suites. This feature makes runtime tests independent of the test execution environment and enhances their reusability and extensibility. All the specified test cases are compiled and stored in a repository in order to make them executable.

Adaptation Layer. It includes the implemented Coding/Decoding and System Adapter entities which facilitate the communication between TT4RT and the SUT in production phase.

[1] This assignment problem is not considered in this current work

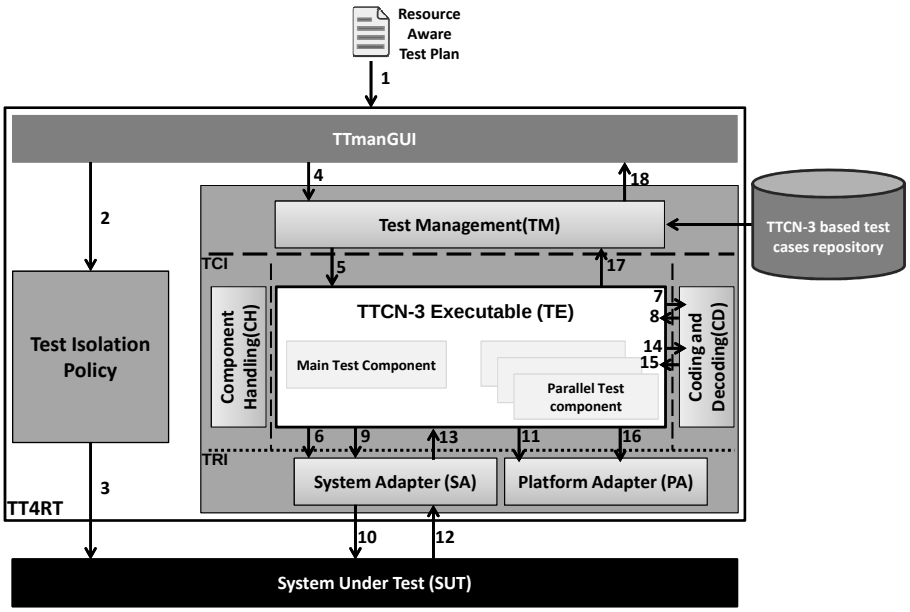

Fig. 3. The proposed workflow of the TT4RT system.

In order to detail the internal interactions in TT4RT system, a workflow illustrated by the Figure 3 is given:

- When a reconfiguration action is triggered, the test plan that describes the affected parts of the SUT by this dynamic change and the test configuration used to validate it, is generated. This plan is considered as an input to the TT4RT system (Step **1**).
- Test isolation policy is called for each component under test in order to choose the best test isolation technique (Step **2**).
- The appropriate test isolation technique is then used to prepare the test environment (Step **3**).
- After preparing the test environment, the test system user initiates the test execution through the TTmanGUI and by calling the adequate method in the TM entity *TciStartTestCase* (Steps **4-5**).
- Once the test process is started, the TE entity creates the involved test components and informs the SA entity that the test case has been started with the aim of allowing the SA entity to prepare its communication facilities. This is done through the call of *triExecuteTestcase* method (Step **6**).
- Next, TE invokes the CD entity in order to encode the test data from a structured TTCN-3 value into a form that will be accepted by the SUT. This is done through the call of *encode* method (Step **7**).
- The encoded test data is passed back to the TE entity as a binary string and forwarded to the SUT via the SA entity with the *triSend* method (Steps **8-9-10**).

- After the test data is sent, a timer can be started. To achieve this, the TE invokes the *triStartTimer* method on the PA entity (Step **11**).
- The SUT returns its response to the SA entity. The given response is an encoded value that has to be decoded in order to be understandable by the TTCN-3 test system (Step **12**).
- For doing this, the SA entity forwards the encoded test data to the TE entity through the method *triEnqueueMsg* (Step **13**).
- The TE entity transmits the encoded response to the CD entity with the intention of decoding it into a structured TTCN-3 value (this is done through the call of *decode* method) (Step **14**).
- The decoded response is passed back to the TE that stops the running timer by invoking the *triStopTimer* method on the PA and finally computes the global verdict (Steps **15-16-17**).
- At last, the test system user is notified by the generated verdict (pass, fail or inconclusive) by the TTmanGUI (Step **18**).

The gains of this design are the conformance to the TTCN-3 standard, generality and platform-independency (applicable to every component based or service oriented systems), reusability and extensibility (compiled code TTCN-3 is stored as jar files in a repository and can be loaded at any time and also updated dynamically without restarting TT4RT system). Furthermore, TT4RT can be used either at deployment time or at service time to validate the SUT. Instead the classical TTCN-3 test systems which consider the SUT as a black-box, TT4RT system treats the SUT as a grey-box (the SUT is composed of a collection of interacting components and compositions under test (CUTs)). This fact can help to localize easily the faulty component or composition and to proceed enhancement of the quality and reliability of the SUT.

5 Case Study

To illustrate our approach, we choose a case study in the telemedicine field. In fact, telemedicine has become an important research issue. It merges telecommunication and information technologies in order to provide remotely clinical health care. It facilitates communications between patients who suffer from chronic health problems and medical staff. In addition, it improves the access to medical services as well as the transmission of patient data (e.g monitored vital signs, laboratory tests, etc.) especially when critical events or emergency situations occur.

As widely described in the literature [17–19], telemedicine applications have to evolve dynamically in order to fulfill new requirements such as adding new health care services, updating the existing one in order to support improvements in wireless and mobile technologies, etc. This adaptability is essential to ensure that these applications remain within the functional requirements defined by application designers, as well as maintain their performance, security and safety properties. Furthermore, the execution environment of such applications is distinguished by its hardware heterogeneity (for instance PDA, PC and sensors) and

the use of large range of wireless networking solutions like wireless LANs, ad-hoc wireless networks and cellular/GSM/3G infrastructure-oriented networks.

Due to these dynamic variabilities, medical errors and degradation of QoS parameters can occur. Therefore, runtime testing is required to validate dynamic system changes. Thus, this validation technique can improve health care quality and lead to the early detection and repair of medical devices malfunctions. In the following subsections, we present the architecture of the studied telemedicine application and also its implementation.

5.1 Architecture of TRMCS Case Study

The adopted telemedicine application is called Teleservices and Remote Medical Care System (TRMCS). The main behaviors and structure of such system are inspired from [20]. As depicted in Figure 4, TRMCS system provides monitoring and assistance to patients suffering from chronic health problems. The interacting actors in the system are :

- Medical staff which is composed of physicians, nurses, etc. These health care providers can be located in their own office, hospitals or even an ambulance car.
- One or more patients who are located at their home and are equipped with wearable devices that can sense one or more vital signs such as blood pressure, respiration rate, pulse rate, oxygen saturation and body core temperature.

The wearable medical sensors measure and transmit biomedical data to local as well as remote medical data centers. They should operate autonomously and have to send alert signals when emergency problems arise.

The main functionalities that the TRMCS system supports are:

- The acquisition of biomedical data from patients equipped with wearable medical sensors.
- The processing of monitored data by generating reports.
- The transmission of the monitored data, medical images, laboratory tests to a local as well as remote medical data centers for storage.
- The analysis of monitored data by sending emergency signals when critical events are triggered or threshold conditions are reached [2].

The latter functionality is highlighted and used as proof-of-concept. Within the following studied scenario, the ability of TT4RT system to detect reconfiguration faults is demonstrated.

Studied scenario. The initial architecture of the studied scenario is outlined in the Figure 5. Each patient sends different kind of help requests to different help centers such as doctor's office, nursery, hospital and ambulatory. This help request can be issued by the patient through a user GUI or raised automatically by the monitoring system. In this current implementation, we support three kinds of help requests: generating call, SMS or alarm signal.

[2] For instance, when the heart rate exceeds a certain level of tolerance.

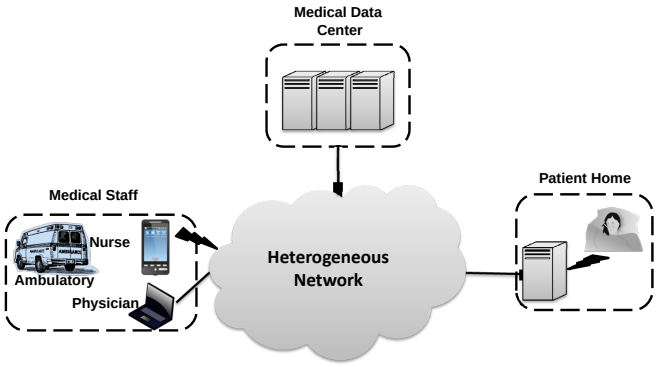

Fig. 4. Global view of Teleservices and Remote Medical Care System.

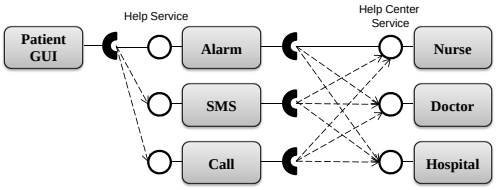

Fig. 5. The initial configuration of the studied scenario.

Reconfiguration scenario. It comes a moment when this system is changed to fulfill new requirements. For instance, the *Alarm* component is changed by a new version with the aim of increasing SUT performance and responsiveness. The new version sends the help request to the help center in a duration that does not exceed 15 time units instead of 30 time units for the old version. Once this reconfiguration is achieved, the new component and all the affected parts of the system by this modification have to be tested.

5.2 Implementation of TRMCS Case Study

In the literature, we have found some research works that use the Open Service Gateway initiative (OSGi) platform[3] to implement such case study [21, 17]. We follow the same technology choice due its dynamism (a powerful Framework to create highly dynamic applications). Thus, we implement the studied scenario using the OSGi Framework, especially OSGi iPOJO model[4]. Under OSGi architecture, software components are encapsulated into bundles which is a java archive file that contains packages and service prerequisites. These bundles are loaded and run automatically by the Apache Felix[5] 1.8 implementation.

[3] OSGi Alliance, http://www.osgi.org/markets/index.asp
[4] http://felix.apache.org/site/apache-felix-ipojo.html
[5] http://felix.apache.org/site/index.html

Without loss of generality, the proposed TT4RT system is used to validate this service oriented application[6] and to detect some previously seeded faults as outlined in the section below.

6 Validation of TRMCS by using TT4RT System

The key concepts presented so far have been used in order to keep the system quality at the same level after a dynamic update has taken place. Thus, we have applied some runtime tests to the evolvable sub-system while substituting the *Alarm* component by another version dynamically. The affected components and compositions by this modification, their testability options, the test cases to execute are specified in the generated Resource Aware Test Plan as outlined in the Figure 6.

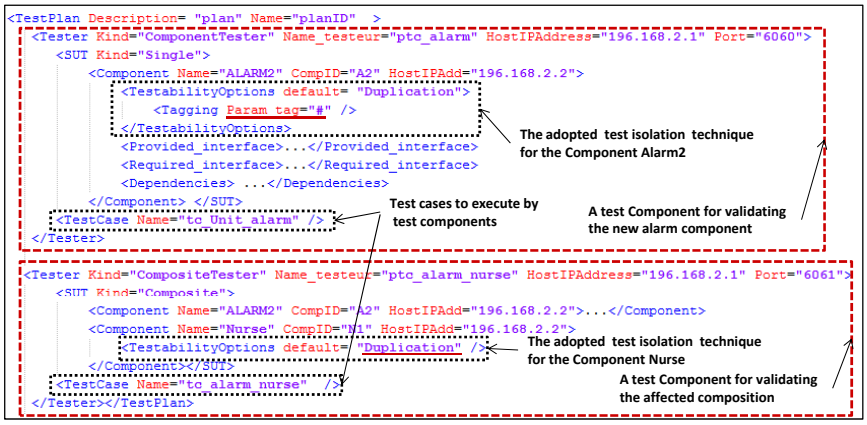

Fig. 6. The main features included in the Resource Aware Test Plan XML file.

Furthermore, We have to mention that some faults have been seeded in the new configuration in order to assess the capabilities of TT4RT system to find these reconfiguration faults. It is worth noting that some inconsistencies are automatically detected by the OSGi Framework, for instance a required service crashing. Nevertheless, TT4RT is still required to detect other kinds of faults such erroneous results provided by the new service, incompatibilities between compositions, degradation of quality of service, etc.

6.1 Specifying the abstract test cases

Different test cases specified following the TTCN-3 notation are available to detect reconfiguration faults. In fact, TTCN-3 standard is used to define not only

[6] It is worthy to note that our TT4RT system can be used to validate either object or component based applications.

the behavior of each test component but also the dynamic and concurrent test configuration of each test request. First of all, the Listing 1 bellow outlines the adopted test configuration and highlights the different test components involved in this testing process.

```
1  testcase tc_substitute_alarm() runs on mtcType system systemType {
2   var ptcType ptc_alarm,ptc_alarm_nurse,ptc_alarm_hospital,ptc_alarm_doctor;
3      ptc_alarm := ptcType.create("ptc_alarm");
4      map(ptc_alarm:ptcPort, system:systemPort);
5      ptc_alarm.start(ptcBehaviour_alarm());
6      ptc_alarm.done;
7
8      ptc_alarm_nurse := ptcType.create("ptc_alarm_nurse");
9      map(ptc_alarm_nurse:ptcPort, system:systemPort1);
10     ptc_alarm_nurse.start(ptcBehaviour_alarm_nurse());
11     ptc_alarm_nurse.done;
12
13     ptc_alarm_doctor := ptcType.create("ptc_alarm_doctor");
14     map(ptc_alarm_doctor:ptcPort, system:systemPort2);
15     ptc_alarm_doctor.start(ptcBehaviour_alarm_doctor());
16     ptc_alarm_doctor.done;
17
18     ptc_alarm_hospital := ptcType.create("ptc_alarm_hospital");
19     map(ptc_alarm_hospital:ptcPort, system:systemPort3);
20     ptc_alarm_hospital.start(ptcBehaviour_alarm_hospital());
21     ptc_alarm_hospital.done;
22 }
```

Listing 1. The test configuration.

The global test process is managed by the MTC component as defined in line 1. This MTC component is responsible for dynamically creating a PTC checking the new *Alarm* component (see line 3) and also three others PTCs for validation the communication between the affected compositions (alarm-nurse, alarm-hospital, alarm-doctor) as indicated respectively in line 8, 13 and 18. To start the execution of these test components, the *map*[7] and *start* methods are used and the adequate function is called (see for example line 4 and 5).

```
1  function ptcBehaviour_alarm() runs on ptcType {
2     timer localtimer := 15.0;
3     ptcPort.send(msg_to_alarm);
4     localtimer.start;
5      alt {
6            [] ptcPort.receive("Service invoked Successfully")
7            {setverdict (pass, "Test service alarm successfully");}
8            [] ptcPort.receive
9            {setverdict (fail, "Something else received");}
10           []  localtimer.timeout
11           { setverdict (fail, "Timeout");}
12            localtimer.stop;
13       }}
```

Listing 2. An example of a PTC behavior.

We have to mention that the instantiation of test components and communication links are done dynamically and the execution of their behaviors is done in a parallel manner. For instance, the next listing shows the behavior of one

[7] This method aims for connecting a port of a PTC to a port of SUT.

PTC validating the new *Alarm* component (*ptcBehaviour_alarm()*). As depicted in the Listing 2, a timer is defined in line 2 and started in line 4 when testing data are sent (see line 3). It is used to validate the timing behavior of the new *Alarm* component that transmits the help request in a period of time smaller than 15 time units. If this deadline is not respected by the new version (see line 10) a fail verdict is generated as indicated in line 11. Otherwise, the functional behavior is validated and accordingly a partial verdict is computed (see line 7 and 9).

For editing and compiling the specified tests, we have used respectively the CL Editor (TTCN-3 Core Language Editor) and the TThree Compiler that are included in the TTworkbench basic tool [8]. The generated Jars are stored in the repository for further use and can be dynamically loaded during the execution when runtime testing is required to validate dynamic changes.

6.2 Preparing the test execution environment

Before executing the compiled tests, the test isolation policy is called in order to choose for each component under test the suitable test isolation technique. The testability options of each component involved in the testing process are specified in the resource aware test plan as depicted in Figure 6. In the current scenario, the *Alarm* component is test aware. It differentiates between test data and business data by using a test tag as illustrated in the test plan. The *Nurse* component is not testable. Thus, we use the default test isolation technique mentioned in the test plan file which is the duplication technique. In this case, a new component is created which handles the test request. Such solution aims not to disturb the original *Nurse* component. The same work is done for *Doctor* and *Hospital* components.

6.3 Executing runtime tests

Once the Resource Aware Test Plan is loaded and the test process is started through the TTmanGUI, the test environment is built and the test components are created dynamically. The Figure 7 highlights the main interaction between some components under test and the corresponding test components. For instance, it shows that the test component *PTC_alarm* is created in order to validate the *Alarm2* component. The latter is instructed to use both the testing data and business data. Thus, *PTC_alarm* sends inputs data which is generated with the test tag and receive outputs in order to verify that the specified timing constraint is respected by the new component. Furthermore, affected compositions are also checked by test components like *PTC_alarm_nurse*. Due to the non testability of the *Nurse* component, it has been duplicated and the duplicate component is used while testing the composition under test behavior.

We follow the same principles for the rest of the components under test as specified in the resource aware test plan. Once the PTCs components terminate

[8] http://www.testingtech.com/products/ttworkbench.php

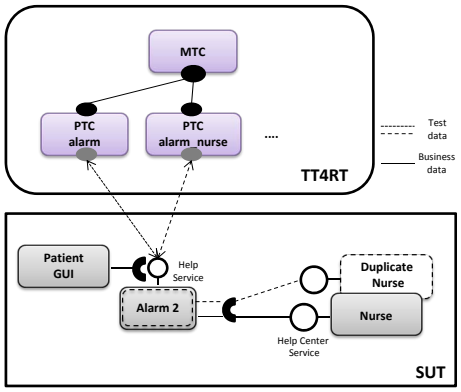

Fig. 7. Test components interaction with the affected components under test.

their specified behaviors, they are removed from the test configuration. The MTC component computes the global verdict which is finally displayed to the test system user through the TTmanGUI. The latter has been implemented using the Java language and the swing package. It has been packaged as an OSGi bundle. The Figure 8 shows the proposed graphical user interface also the final verdict of the executed runtime tests.

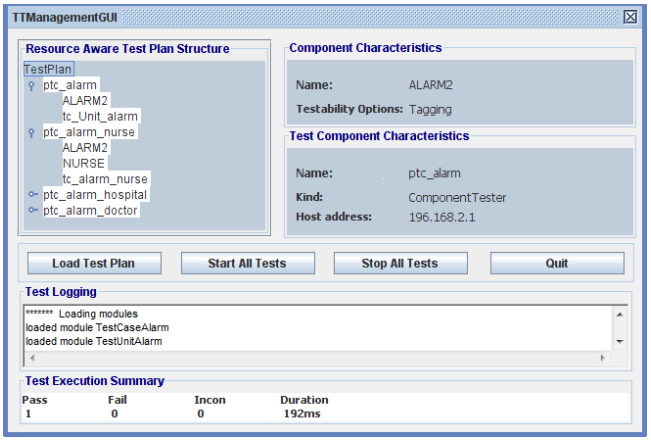

Fig. 8. Screenshot of the Prototype TTmanGUI of the TT4RT system.

7 Related Work

Recent research activities have been proposed to deal with runtime testing in dynamic environments. They aim to ensure the correctness of the running system

after reconfiguration. In fact, we distinguish approaches dealing with ubiquitous software systems [1], CBA systems [3, 5, 4], SOA systems [6, 7], publish/subscribe systems [2] and autonomic systems [8].

Each of them proposes a test system tightly coupled with the system under test (SUT). In addition, they did not concentrate on proposing a generic and platform independent test architecture that evolves when the system under test evolves too. Moreover, they are based on only one technique to isolate runtime tests in the aim of reducing the interference between business and test data except [8] which supports two kinds of test isolation techniques. These approaches mostly used a specific language framework such as Junit to write and execute tests. None of them have used a generic testing language such as TTCN-3.

There have been many efforts on proposing test systems based on the TTCN-3 standard. We distinguish research for testing protocol based applications [22, 15], Web services [23, 16], Web applications [24–26] and also real time and embedded systems [27, 28]. To the best of our knowledge, [9] is the only previous paper presenting ideas on using TTCN-3 standard for online validation and testing of internet services. However, this work did not deal with test isolation issues when testing is applied in the production phase.

Unlike these approaches, our work aims at proposing a generic and platform independent test system based on the TTCN-3 standard to execute runtime tests. The proposed test system supports different test isolation mechanisms in order to support testing different kinds of components: test sensitive, test aware or even non testable components. Such test system has an important impact on reducing the risk of interference between test behaviors and business behaviors as well as avoiding overheads and burdens.

8 Conclusion

The work presented in this paper focuses on the use of TTCN-3 standard for executing runtime tests to reveal component based system inconsistencies and faults. Our main contribution consists in adding a test isolation layer in the classic TTCN-3 test system in order to reduce test data interference with business data at runtime. Furthermore, we add a test management layer that facilitates the interaction between a test system user and the TT4RT system through a graphical interface. We illustrated the proposed approach by implementing a prototype for validating OSGi bundles in the context of telemedicine applications. In addition, we shortly presented the technical solutions that we employed for the current implementation of our TT4RT system.

Nevertheless, distributing test configurations in different nodes remains unsolved. Hence, we are exploring solutions in this area to distribute efficiently test components with fitting some resource and connectivity constraints. Besides, this work does not deal with test cases generation. All the executed tests are specified manually. Therefore, we aim to investigate effort in automating TTCN-3 test cases generation, especially when behavior adaptation occurs. An-

other area to explore is the optimization of test cases selection by re-testing only the affected parts of the system due to a reconfiguration action.

References

1. Merdes, M., Malaka, R., Suliman, D., Paech, B., Brenner, D., Atkinson, C.: Ubiquitous RATs: how resource-aware run-time tests can improve ubiquitous software systems. In: SEM '06: Proceedings of the 6th international workshop on Software engineering and middleware, New York, NY, USA, ACM (2006) 55–62
2. Piel, É., González-Sanchez, A., Groß, H.G.: Automating integration testing of large-scale publish/subscribe systems. In Hinze, A., Buchmann, A.P., eds.: Principles and Applications of Distributed Event-Based Systems. IGI Global (2010) 140–163
3. Piel, É., González-Sanchez, A.: Data-flow integration testing adapted to runtime evolution in component-based systems. In: Proceedings of the 2009 ESEC/FSE workshop on Software integration and evolution @ runtime, New York, USA, Association for Computing Machinery (2009) 3–10
4. Gonzalez, A., Piel, E., Gross, H.G.: Architecture support for runtime integration and verification of component-based systems of systems. In Mauro Caporuscio, Antinisca Di Marco, L.M.H.M.A.P.O.S., ed.: Automated Software Engineering - Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on, IEEE Computer Society (sep 2008) 41–48
5. Niebuhr, D., Rausch, A.: Guaranteeing correctness of component bindings in dynamic adaptive systems based on runtime testing. In: SIPE 09: Proceedings of the 4th international workshop on Services integration in pervasive environments, New York, NY, USA, ACM (2009) 7–12
6. Bai, X., Xu, D., Dai, G., Tsai, W.T., Chen, Y.: Dynamic reconfigurable testing of service-oriented architecture. Volume 1. (july 2007) 368 –378
7. Greiler, M., Gross, H.G., van Deursen, A.: Evaluation of Online Testing for Services A Case Study. In: 2nd International Workshop on Principles of Engineering Service-Oriented System, ACM (2010) 36–42
8. King, T.M., Allen, A.A., Cruz, R., Clarke, P.J.: Safe runtime validation of behavioral adaptations in autonomic software. In Calero, J.M.A., Yang, L.T., Mármol, F.G., García-Villalba, L.J., Li, X.A., Wang, Y., eds.: ATC. Volume 6906 of Lecture Notes in Computer Science., Springer (2011) 31–46
9. Deussen, P.H., Din, G., Schieferdecker, I.: A TTCN-3 Based Online Test and Validation Platform for Internet Services. In: Proceedings of the The Sixth International Symposium on Autonomous Decentralized Systems (ISADS'03), Washington, DC, USA, IEEE Computer Society (2003)
10. Brenner, D., Atkinson, C., Malaka, R., Merdes, M., Paech, B., Suliman, D.: Reducing verification effort in component-based software engineering through built-in testing. Information Systems Frontiers **9**(2-3) (2007) 151–162
11. Lahami, M., Krichen, M., Jmaiel, M.: A distributed test architecture for adaptable and distributed real-time systems. In the Journal of New technologies of Information (RNTI), CAL'2011 (2012)
12. ETSI: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language
13. ETSI: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI)

14. ETSI: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI)
15. Schulz, S., Vassiliou-Gioles, T.: Implementation of TTCN-3 Test Systems using the TRI. In: Proceedings of the IFIP 14th International Conference on Testing Communicating Systems XIV, Deventer, The Netherlands, The Netherlands, Kluwer, B.V. (2002) 425–442
16. Rentea, C., Schieferdecker, I., Cristea, V.: Ensuring quality of web applications by client-side testing using ttcn-3. In: TestCom/Fates. (2009)
17. Chen, I.Y., Tsai, C.H.: Pervasive Digital Monitoring and Transmission of Pre-Care Patient Biostatics with an OSGi, MOM and SOA Based Remote Health Care System. In: Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom). (2008) 704–709
18. Varshney, U.: Pervasive healthcare and wireless health monitoring. Mob. Netw. Appl. **12**(2-3) (2007) 113–127
19. André, F., Segarra, M.T., Zouari, M.: Distributed Dynamic Self-adaptation of Data Management in Telemedicine Applications. In: Proceedings of the 7th International Conference on Smart Homes and Health Telematics: Ambient Assistive Health and Wellness Management in the Heart of the City. ICOST '09, Berlin, Heidelberg, Springer-Verlag (2009) 303–306
20. Inverardi, P., Muccini, H.: Software Architectures and Coordination Models. J. Supercomput. **24**(2) (February 2003) 141–149
21. Chen, I.Y., Huang, C.C.: A Service Oriented Agent Architecture To Support Telecardiology Services On Demand. Journal of Medical and Biological Engineering (2005)
22. Schieferdecker, I., Vassiliou-Gioles, T.: Realizing distributed ttcn-3 test systems with tci. In: Proceedings of the 15th IFIP international conference on Testing of communicating systems, Berlin, Heidelberg, Springer-Verlag (2003)
23. Schieferdecker, I., Din, G., Apostolidis, D.: Distributed functional and load tests for web services. STTT **7** (2005) 351–360
24. Stepien, B., Peyton, L., Xiong, P.: Framework Testing of Web applications using TTCN-3. Int. J. Softw. Tools Technol. Transf. **10**(4) (2008) 371–381
25. Ying Li, Q.L.: Research on Web application software load test using Technology of TTCN-3. American Journal of Engineering and Technologu Research **11** (2011) 3686–3690
26. Din, G., Tolea, S., Schieferdecker, I.: Distributed load tests with ttcn-3. In: TestCom. (2006) 177–196
27. Okika, J.C., Ravn, A.P., Liu, Z., Siddalingaiah, L.: Developing a ttcn-3 test harness for legacy software. In: Proceedings of the 2006 international workshop on Automation of software test, New York, NY, USA, ACM (2006) 104–110
28. Serbanescu, D.A., Molovata, V., Din, G., Schieferdecker, I., Radusch, I.: Real-time testing with ttcn-3. In: TestCom/Fates. (2008) 283–301