Using TTCN-3 for testing the interoperability of HL7v3 based applications

Alexandru Egner[1], Florica Moldoveanu[1,2], Nicolae Goga[1,3]

[1] University Politehnica of Bucharest, Romania

[2] HL7 Romania Chair

[3] University of Groningen, Netherlands

Abstract

HL7v3 standard was designed to facilitate communication between all types of eHealth applications, regardless of the domain of activity. The flexibility of the HL7v3 messages led, however, to proprietary definitions of HL7v3 messages and structures. In order to unify the communication in different domains, profiles have been developed and standardized. In this context, testing plays an important role in assuring interoperability, as well. This paper presents a method to test the interoperability of HL7v3 applications, using the standardized TTCN-3 test scripting language and its corresponding TTCN-3 test system. The full testing process is described, highlighting all the components involved and providing guidelines for implementing them. This approach requires testers to be familiar to the TTCN-3 environment.

Keywords

HL7v3, TTCN-3, testing, QED, eHealth

Correspondence to:

Alexandru Egner
University Politehnica of Bucharest, Romania e-mail: alexandru.egner@cs.pub.ro

EJBI 2012; 8(4):28–32

1 Introduction

Testing the interoperability of HL7v3 based applications plays an important role in the growth of the eHealth community. HL7v3 is a very complex standard and its characteristics make testing a difficult task. This paper proposes a new approach to the interoperability testing of HL7v3 applications. The solution was validated in the context of a HL7v3 profile, Query for Existing Data (QED). However, it is adaptable to other profiles, as well.

The solution is based on the TTCN-3([1]) test scripting language and TTCN-3 test system. The most important advantage of this approach is that TTCN-3 is a standardized testing technology, which is reliable, very flexible and independent of the platform and the technology of the system under test. In addition, the TTCN-3 test system is portable and modularized. This paper presents the implementation details of the testing procedure, with highlight on adapting HL7v3 applications to the TTCN-3 test system.

2 Testing HL7v3 applications

HL7v3 was designed to facilitate communication between virtually any type of eHealth application, regardless of its corresponding healthcare domain. However, the way HL7v3 was designed caused sometimes interoperability issues. HL7v3 allows implementers to define custom message structures. This led to the development of many applications that communicate medical data in proprietary formatted messages. To overcome this, HL7v3 profiles have been developed and standardized for different healthcare domains. These profiles are the first step towards HL7v3 interoperable systems.

In this context, interoperability testing plays an important role in the growth of the HL7v3 community. However, testing the interoperability of HL7v3 based applications is especially difficult, because of the differences between message structures. Testing usually focuses on a specific HL7v3 message structure, which limits the applicability of the testing solution. This paper proposes, however, a generic approach to test the interoperability between

HL7v3 applications.

The solution is a testing framework based on the standardized TTCN-3 testing language and a TTCN-3 test system. The approach is validated on a specific HL7v3 profile, i.e. Query for Existing Data (QED). QED is an IHE profile [2] which allows systems to query data repositories for clinical information on vital signs, problems, medications, immunizations and diagnostic results.

2.1 TTCN-3 architecture

Testing and Test Control Notation version 3 (TTCN-3) is a strongly-typed scripting language, used for defining complex test specifications. TTCN-3 provides mechanisms to describe test behaviors by unambiguously defining the meaning of a test case pass or fail. TTCN-3 is a standardized testing language that has been used for more than 15 years in standardization and industry. It is very flexible, portable and well suited for conformance and interoperability testing. TTCN-3 test case specifications do not depend on the platform, architecture or technologies used by the System Under Test (SUT). It provides a built-in verdict mechanism that allows easy evaluation of the testing results. Moreover, TTCN-3 has a refined template matching mechanism that is very flexible and easy to manage.

With TTCN-3 testing language testers can define test cases and the order in which they are executed. However, to execute test cases, a TTCN-3 test system is needed. The TTCN-3 test system can be thought of as a set of interacting entities that implement specific test system functionalities. Figure 1 depicts the general architecture of a TTCN-3 test system, highlighting the main components and the relationship between them.

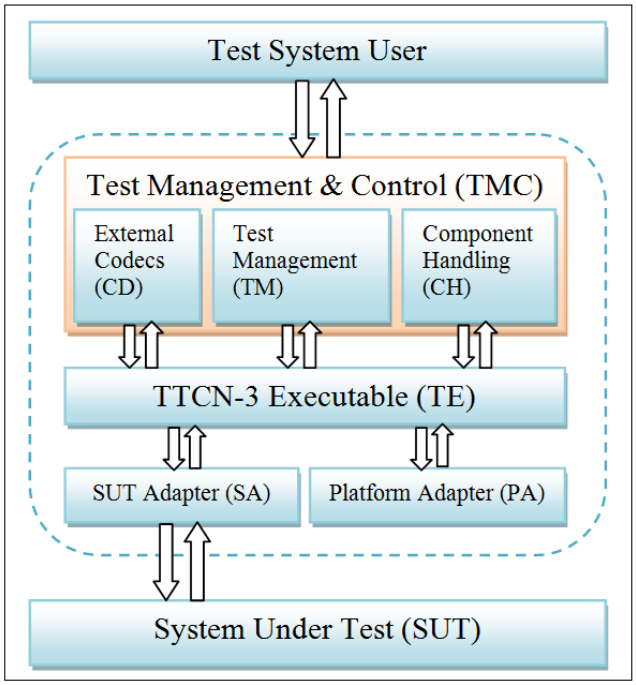

Figure 1: General architecture of a TTCN-3 test system

The central layer, TTCN-3 Executable (TE) handles the execution of TTCN-3 statements. TE depends on the services provided by the other two layers. Test Management Control (TMC) includes three entities: External Codecs (CD), Test Management (TM) and Component Handling (CH). CD is responsible for encoding and decoding data, TM represents the interface with the Test System User, and the CH is used for distributed execution of the test cases. Platform Adapter (PA) implements TTCN-3 external functions and provides timing mechanisms. SUT Adapter (SA) adapts the message/procedure based communication between the TTCN-3 test system and the SUT to the particular execution platform of the test system. CD and SA will be subsequently referred to as the Codec and the Adapter, respectively.

The majority of the TTCN-3 tools provide default implementation for the TM and CH. This is not the case of CD, SA or PA, since they cover aspects of the test system, which are either test suite or SUT specific.

2.2 The TTCN-3 type system

The TTCN-3 type system extends the basic constructs that usually have correspondents in programming languages with additional testing specific concepts, such as built-in data matching, distributed test system, or concurrent execution of test components. The TTCN-3 type system is very complex and includes also test verdicts, test system components, and even direct support for time. The core components of the TTCN-3 type system are the TTCN-3 records, TTCN-3 enumerated types, and TTCN-3 templates. These three components are used for storing the information contained in HL7v3 messages in TTCN-3 specific format.

TTCN-3 records are constructs used for grouping related fields in a single type. TTCN-3 records are used to store data in a structured way. Field names within a record must be unique and their types may be either built-in or a user-defined. TTCN-3 records are arguably the most used types of the TTCN-3 type system. TTCN-3 enumerated types are ideal for representing types that have small, finite sets of values. They are used to model types that take only a distinct named set of values, i.e. enumerations. TTCN-3 enumerated types are often used in HL7v3 to encode vocabularies. TTCN-3 templates are used for defining information exchanged between the test system and the SUT. While TTCN-3 types such as records and enumerated types define logical structures for storing information, templates contain the actual information. Subsequently, TTCN-3 records, TTCN-3 enumerated types and TTCN-3 templates will be referred as records, enums and templates, respectively.

When creating test cases, testers define two templates. The first one represents the input that is passed to the SUT, while the second one is the expected output. During test case execution, the TTCN-3 matching mechanism verifies if the expected output matches the one received from the SUT. Based on the similarity between the two,

a verdict is set to the test case, generally indicating if the test failed or passed. The input and expected output templates are defined in TTCN-3 specific format. Using these templates for testing SUTs require the existence of modules for converting data from TTCN-3 to SUT specific formats. For this conversion, two of the TTCN-3 test system components are used, namely the Codec and the Adapter.

An important aspect of testing HL7v3 applications using TTCN-3 is that templates are difficult to create. Their hierarchical structure can span on many levels, usually reaching more than twenty levels, in the case of QED messages, which makes manual definition and maintenance cumbersome.

3 Testing HL7v3 applications

In order to validate the suitability of testing HL7v3 applications using TTCN-3, a case study was considered. The SUT chosen is a mature application that uses the IHE QED profile. During the development of this solution, several decisions have been made to facilitate the communication with this SUT. However, as the paper describes further, the modularity of the TTCN-3 test system allows adapting this solution to testing any HL7v3 based application. The SUT is deployed as a web service, and the communication is performed through SOAP messages. The message flow for testing the interoperability of the HL7v3 based application is depicted in Figure 2.

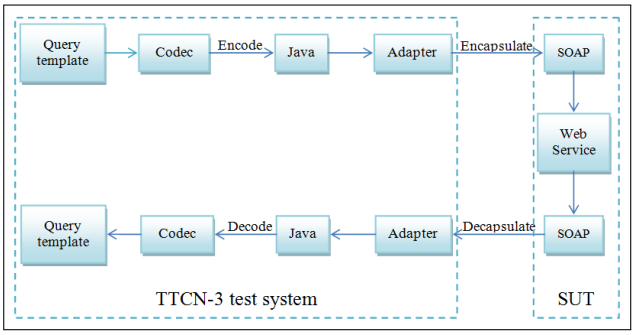

Figure 2: Testing HL7v3-based applications - message flow

When executing the test case, the template that describes the QED Query is sent to the Codec through the TTCN-3 Control Interface (TCI). The message is translated into a Java object and passed on to the Adapter through the TTCN-3 Runtime Interface (TRI). The Java object is then serialized and embedded into a SOAP message. After the connection between the Adapter and the SUT is established, the SOAP message is passed on to the SUT. If the query is valid, the web service replies with a QED Response in SOAP format. The Adapter converts the SOAP message into a Java object and forwards it to the Codec, where it is decoded into a TTCN-3 template. At this point, the TE evaluates the response from the SUT, setting the verdict of the test case.

3.1 Implementation of the Codec

The Codec (Coder/Decoder) is an important TTCN-3 test system component. It is responsible for interfacing the communication between TE and the Adapter. The Codec has two basic functions: encoding and decoding. TE interprets test cases and automatically converts templates representing QED Queries into Java objects, organized as structures. After the conversion, Java objects are sent to the Codec, via the TCI interface. The TCI [5] is composed of three interfaces that define the interaction between TE and TM, CD and CH.

In the encoding phase, the Codec translates the structure generated by TE into a Java HL7 object. In this way the Codec assures that the Adapter receives a set of input data that can easily be handled. The translation is performed at runtime, using Java Reflection. The structure is parsed, each composing element being translated into the corresponding HL7 Java object. These objects are then encapsulated into a Java-based query request.

The Codec is also responsible for sending this query request to the Adapter. TRI [6] defines the interaction between TE and the Adapter. There is an important constraint determined by the usage of TRI. The Java interface TriMessage has to be implemented by any class describing messages that are used in communication between the TE and the Adapter. This constrains messages to be formatted as byte arrays. Since none of the JAXB generated classes implement the Serializable interface, scripts had to be created to modify each class and add "implements Serializable" to their definition, so that requests can be serialized and sent to the Adapter within a TriMessage.

In the decoding phase, the Codec receives a TriMessage from the Adapter, containing the QED Response. The Codec deserializes the message, converts it into a Java structure and then forwards it to TE.

3.2 Implementation of the Adapter

The existence of the Adapter confers the TTCN-3 test system much flexibility. The Adapter is the TTCN-3 test system component responsible for establishing connections and handling communication with the SUT. The same test suite can be executed on SUTs with different platforms just by replacing this component.

The Adapter enables communication between TE and the SUT. It has two different functionalities: encapsulation of the query and extraction of the response from SOAP messages. Simple Object Access Protocol (SOAP) is a protocol specification for exchanging structured information. SOAP relies on XML as its message format.

In the encapsulation phase, the Adapter uses the TriMessage it receives from the Codec as input. The byte array containing the query is deserialized. Given the transparency of the test case to the Adapter, the conversion from Java to XML could only be done dynamically, at runtime, through Java Reflection. For this translation Java API for XML Processing (JAXP) was used. JAXP

[7] provides the capability of validating and parsing XML documents. It offers several parsing interfaces from which Document Object Model (DOM) parsing interface was chosen. DOM [8] enables parsing of XML documents and constructing complete in-memory representations of the documents.

DOM documents have tree-type structure. They are composed of a root element, which represents the XML document, and several nodes, representing XML elements. The translation of the Java message to XML was implemented as following the next steps:

Step 1: a DOM document is created based on the type of the query message;

Step 2: object's fields list is obtained using Java Reflection; each field represents a Java HL7 object;

Step 3: for each field a DOM element is generated and added to the root element's children list;

Step 4: another DOM element containing the preconditions is defined and added to the root's children list;

Step 5: the DOM document is serialized and the XML-formatted message is ready to be forwarded to the SUT.

DOM is used when extracting of the Java object from the XML-formatted response received from the SUT, as well. The transformation follows the next steps:

Step 1: the XML is deserialized into a DOM document;

Step 2: the root element of the DOM is used to generate a Java object representing the QED Response;

Step 3: the document is parsed and each node is translated into the corresponding HL7 Java object; these objects are set as fields of the Java-based QED Response object;

Step 4: after the parsing is finished, the Java QED Response object is serialized to a byte array and sent to the Codec via a TriMessage.

The Adapter is also responsible for handling the communication with the SUT. The communication protocol chosen for exchanging messages was SOAP. The web service used WSDL [9] to define the type of QED messages it can handle, such as Query, Continue or Cancel. In order for the communication to take place, the connection had to be established, and for that a Java client was needed. There are many tools that use the WSDL description to generate stubs and clients. Java API for XML Web Services – Reference Implementation (JAX-WS RI) was chosen. JAX-WS RI [10] was introduced in Java SE 5 to simplify the development and deployment of web service clients and endpoints. Once the client was created, methods for connecting, sending and receiving QED messages were available and the communication was possible.

3.3 Generating testing components

In terms of the TTCN-3 test system, when executing a test case, a template representing the QED Query is sent to the SUT and if the template representing the QED Response matches the expected response template, the verdict is set to pass. As SUTs usually can't handle TTCN-3 templates, they have to be converted to SUT compliant formats. Java was chosen as common language, for portability reasons and for the fact that it is the language in which TTCN-3 test system components are developed. Thus, the template representing the QED Query is encoded into a Java object that is passed to SUT, and the response from the SUT is decoded into a template storing QED Response.

The first attempt was to use a set of Java classes offered by HL7, i.e. the Java SIG Project (jsig) [3]. The main problem of jsig classes is the lack of a generic way to generate Java HL7 objects based on the TTCN-3 templates. Jsig classes offer no generic way of instantiating objects and setting field values at runtime. This was a major problem and another approach was needed.

The second approach was to use a set of XML Schemas that describe the HL7 data types and the QED queries. Java classes were generated based on the XSDs using Java Architecture for XML Binding (JAXB) [4]. Because of the constraints imposed by communication between TTCN-3 test system components, the classes were modified through some scripts, so that all implemented the Serializable interface.

The Java classes were used to generate the corresponding TTCN-3 records and enumerated. The generating tool that was developed and used has two components. The first one is responsible for instantiating the classes and extracting the relevant information at runtime, using Java Reflection. The second one is a TTCN-3 code generator which uses the information provided by the first component to create two TTCN-3 modules, one for the records and one for the enumerated.

After generating the records and enums, templates corresponding to HL7 data types could be defined. Finally, we were able to create query and response templates used in defining the test cases.

To summarize, when testing HL7v3 applications using TTCN-3, the tester defines two templates: the one representing the query and the one representing the expected response. When defining these templates, the TTCN-3 type system should contain types, i.e. records or enums, that describe basic HL7v3 structures, in a TTCN-3 format. Since these types are not part of the TTCN-3 type system, they had to be generated using the automated tools described earlier. These aspects allow testers to define test cases. However, when executing the test cases against a SUT, the TTCN-3 test system is used. In order to adapt the messages from the template format to an SUT known format, two components have to be implemented: the Codec and the Adapter.

4 Conclusions

There are many advantages which come with this approach, and probably the most important one is the technology used for testing. TTCN-3 is standardized, has been

validated as one of the best testing languages for protocol testing, it has a complex verdict assessment and its modularity makes it very flexible.

The proposed solution does not directly depend on the SUT, neither on its architecture, nor on the technology it uses. The Adapter component is responsible for linking the SUT with the test suite, which means that it is the only component that needs to be replaced when testing other systems. Another advantage is the automation. Test suites can be developed to thoroughly test several systems, without user intervention.

This approach has been validated on a mature system that is using a HL7v3 profile: QED. However, the authors highlight the adaptability of this solutions to other profiles, as well. Even though during the implementation phase many generation tools had to be developed, these tools can be used to other profiles, as well, since they are not profile-dependent.

This approach requires testers to be familiar to these technologies and to TTCN-3 standard. On the other hand, times and costs of the testing are reduced, since the testing process is completely automatized, and can be used with different SUTs, as well.

References

[1] Colin Willcock, Thomas Deiß, Stephan Tobies, Stefan Keil, Federico Engler, Stephan Schulz, "An introduction to TTCN-3" John Wiley & Sons, April 2005.

[2] http://wiki.ihe.net/index.php?title=PCC_TF-1/-QED#Appendix_E_-_WSDLs_for_QED (last access on December 30 2012)

[3] http://aurora.regenstrief.org/javasig (last access on December 30 2012)

[4] http://java.sun.com/developer/technicalArticles/-WebServices/jaxb/ (last access on December 30 2012)

[5] "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI)", ETSI ES 201 873-6 v3.2.1, February 2007.

[6] "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI)", ETSI ES 201 873-5 v3.2.1, February 2007.

[7] https://jaxp.dev.java.net/

[8] Peter-Paul Koch, "The Document Object Model: an Introduction", May 2001.

[9] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, Sanjiva Weerawarana, "Web Services Description Language (WSDL) Version 2.0 Part1: Core Language", June 2007.

[10] https://jax-ws.dev.java.net/

