# Customized Test Data Generator for HL7v3 based Healthcare Information Systems

**Alexandru Egner*, Florica Moldoveanu*, Nicolae Goga**, Alin Moldoveanu*, Victor Asavei*, Anca Morar***

**University Politehnica of Bucharest, Romania (e-mail: alexandru.egner@cs.pub.ro, florica.moldoveanu@cs.pub.ro, victor.asavei@cs.pub.ro, anca.morar@cs.pub.ro),*
***University of Groningen, Netherlands (e-mail: n.goga@rug.nl)*

**Abstract:** Test effectiveness is a fundamental quality aspect of a test specification, which reflects its ability to demonstrate system quality levels and discover system faults. The effectiveness is tightly linked with the quality of the test data. The paper highlights specific challenges related to testing eHealth applications and emphasizes the difficulties in testing HL7v3 based applications. This paper presents a new approach on generating input test data sets: a highly customizable distance-based test data generator. The paper highlights the importance of having organized structures of test data and shows how the proposed test data generator uses adaptable distances to create clusters of test data. The generator is designed to create test data in a testing language independent format and provide means of conversion to the format used by the target testing language. A general architecture of this generator is presented, and implementation guidelines are proposed. The paper also presents the conclusions drawn from validating the generator in a real scenario.

*Keywords:* automatic testing, test data generation, HL7v3, TTCN-3, eHealth, clustering

## 1. INTRODUCTION

The interest in eHealth has increased greatly recently. The domain is rapidly evolving because of the impact it has on real life. Patient data is nowadays electronically stored and the communication between eHealth systems is faster and safer. Many protocols have been developed to expand the eHealth boundaries. Such protocol is Health Level 7 version 3 (HL7v3) (HL7, 2011), a messaging protocol defined to ensure interoperability between eHealth applications, providing standardized structures for medical data communication. In the context of a constantly growing interest in eHealth domain, testing eHealth applications plays an important role.

The quality of testing eHealth applications depends on the quality of the input data set. Having a quality input data set is an important challenge, especially when testing applications compliant to HL7v3, a protocol that allows a high level of flexibility in terms of the structure and content of the message. HL7v3 was designed to provide the possibility of defining custom message structures, which in time led to the development of many applications that communicate medical data in proprietary formatted messages. For this reason, defining a suitable set of test data for each application can be troublesome.

Having a quality input test data set is a prerequisite for testing in general, not an eHealth specific one. When testing HL7v3 based applications, however, there are many factors that make the defining of a quality input test data set a difficult challenge. For example, according to the authors' knowledge, there exists no customizable HL7v3 message generator that can be used by test designers to generate inputs for functional testing.

This paper proposes a method of generating custom input test data sets. The solution is generic, i.e. it may be used for generating all types of input test data, not just for eHealth applications. The method uses distances for generating different test data values. The advantages of using different distances in the generating process are discussed in this paper.

The case study chosen to validate the solution was part of the project Reliability Testing of Medical Systems (ReTeMeS) (RTMS, 2012). A customizable test input data generator was developed to address the need of a quality input test data set that was required for testing an HL7v3 based system.

This paper presents the method used in the generating process of the input test data set, the design and implementation of the customizable generator used in the case study, which was named Test Data Generator, and the conclusions drawn from using this generator.

## 2. TESTING HL7V3 BASED APPLICATIONS USING TTCN-3

HL7 is an ANSI accredited organization that operates in the healthcare domain and is involved in the development of international healthcare informatics interoperability standards. HL7v3 is a messaging standard developed by the HL7 organization, derived from the need for supporting interoperability between healthcare systems.

HL7v3 was developed to correct much of the problems of

HL7v2, however it is not yet used as a mainstream. One of the causes is that, even though it was designed to be forward compatible, full backward compatibility with the previous versions cannot be achieved. This is one of the main reasons why interoperability testing plays such an important role in the development of the standard. Recent progress in the direction of interoperability testing of HL7v3 (Namli *et al.*, 2009; Vega *et al.*, 2010) based applications has been made, one of the pioneers of this progress being the use of TTCN-3 and its correspondent test system.

Testing and Test Control Notation version 3 (TTCN-3) standardized test scripting language (ETSI, 2007(a)) is a textual language, which has the look and feel of a programming language, but has been designed for testing including artefacts required for testing.

A TTCN-3 based test specification is called Abstract Test Specification (ATS). It usually consists out of various TTCN-3 modules. A module is the top-level element of the TTCN-3 language, which is used to structure the test definitions of:

- Test data: the structure of test data is defined as types of messages and the specific test data is represented by instances of these types called templates;
- Test configurations: ports and test components are used to define the active entities of the test system and the interfaces to the System Under Test (SUT);
- Test behaviour: functions, altsteps, and testcases that implement the interactions between the test components and the SUT;
- Control: the global behaviour of the test system to control the flow of a test case execution.

In order to get an executable test system based on TTCN-3, an adaptation component (ETSI, 2007(b)) called Adapter and an encoding/decoding component called CoDec have to be provided, in addition to the Abstract Test Specification. The execution environment uses these components to adapt data and requests to the System Under Test (SUT) and its interfaces.

TTCN-3 has gained much popularity for testing HL7-based applications recently. This is the reason why TTCN-3 was chosen as the target testing language for demonstrating the capabilities of the Test Data Generator.

## 3. OVERVIEW OF THE INPUT DATA USED FOR TESTING HL7V3 BASED APPLICATIONS

It is widely accepted in the literature that dividing the input domain in equivalence classes reduces the cost of testing, without affecting its quality. The problem of creating classes of equivalence based on the input data domain of an SUT is addressed in many testing environments. Clustering applied to an input domain is a powerful method to considerably reduce the quantity of relevant input data. The main advantage of using clustering in testing is that it reduces the number of test cases run on a SUT. Any representative of a cluster triggers the same behaviour of the SUT as the rest of the members of the same cluster would. This eliminates redundant testing, significantly reducing the costs of testing.

While clustering has many advantages, there are still some drawbacks that prevent test designers to adopt it. In some cases the process of clustering is so expensive that is not worthed. Clustering HL7v3 messages is especially cumbersome, mostly due to the fact that there is no customizable HL7v3 message generator available. Without a message generator, clusters can be derived out of repositories of existing messages.

The Reference Information Model (RIM) is a fundamental model from which all HL7v3 messages are derived. To avoid overloading the RIM with a huge number of classes, HL7 defines generic healthcare information classes at the RIM level. RIM classes can be refined to more specific classes needed for a particular domain. The process of refinement leads to the development of huge number of classes for different domains. For this reason, many HL7v3 profiles have been developed from the need to formalize the messages, so that applications from a specific domain can better interoperate.

In this context, where applications usually communicate through different types of HL7v3 messages, with different structures, finding repositories of similar messages is complicated.

Considering the difficulties of finding such repositories of specific type of HL7v3 messages, the paper proposes a different approach. Instead of collecting similar HL7v3 messages, that adhere to the same message structure, into a test data repository, and then apply clustering algorithms to group the messages, the tester can directly generate clusters of equivalent messages. This way the tester can create clusters of HL7v3 messages at virtually no cost. The Test Data Generator is the proposed approach that addresses the issue of creating such clusters. The following sections describe the Test Data Generator and its immense potential.

The Test Data Generator can be used by test designers for creating custom types of input data. It can be used to generate messages that adhere to any type of HL7v3 compliant structure. The generator enables the possibility of choosing the target testing language and its correspondent test system, adapting the format of the generated test data to the language dependent format.

Due to the high level of customization it allows, specific messages can be created as part of a certain cluster. Generating clusters of HL7v3 messages is an important realization that can be integrated in many testing methodologies.

Clustering consists of classifying similar data into different groups (clusters). Data clustering is commonly used when data sampling and processing is needed (Jain *et al.*, 1999). In terms of software testing, a cluster should contain stimuli that are considered similar in terms of the SUT behaviour, i.e. produce equivalent results. There are many ways to generate clusters of test data. One important challenge is to allow testers a high level of flexibility for defining test inputs. The process should provide means of customizing what the tester defines as the most significant values of the generated HL7v3

messages (i.e. boundaries for integer ranges, dictionaries for charstring types, etc.). Another challenge is to enable test designers to automatically generate specific HL7v3 message clusters.

Data variance is investigated by inspecting the proximity of objects. The survey of (Jain *et al.*, 1999) presents how clustering facilitates the grouping of a given collection into meaningful similar data points. Measurement of the proximity (similarity) between data points is accomplished through well-defined partition clustering algorithms. Example of direct utility of this result is in statistical theory and machine learning, where the first step is pattern/data representation.

Cluster analysis is a method for finding groups of clusters in a population of objects. The goal of cluster analysis is to divide a population in such way that objects with similar attribute values are placed in the same cluster, the reunion of all clusters form the entire domain, and clusters are disjunctive. The similarity or dissimilarity is measured using dissimilarity metrics such as the Euclidean Distance (Gower, 1985). Another example is the semantic distance function on pairs of words or terms, entitled Google distance, which has been suggested in (Cilibrasi and Vitanyi, 2007).

Independent of a test specification language, the test data adequacy criterion remains among the most important factors for the effectiveness of a test (Sneed, 2004). Approaches to study the relevance of the selected test data include the notion of distance between programs, where programs are the tested systems. A test data set is considered to be adequate if it distinguishes the tested program from other programs, which are sufficiently far from it, i.e. produce different input-output behaviour. Close programs producing same results are considered the same (Davis and Weyuker, 1988). A similar concept of Adaptive Random Testing is provided in (Ciupa *et al.*, 2006) where random test inputs rely on the notion of distance between the test values.

All these approaches intend to study test data variance as a measure of test data values spread over the input domain. Given the very large number of possible inputs that could be feed to a program that is tested, the goal is to minimize the number of test cases in a test suite while maintaining test effectiveness as high as possible.

## 4. DESIGN OF THE TEST DATA GENERATOR

The Test Data Generator uses as input an existing HL7v3 message (i.e. an instance of a HL7 type) and generates messages that have a custom level of similarity with the original one. The level of similarity between the generated message and the original one is customizable. The input of the Test Data Generator can be both a message that adheres to a HL7v3 profile, or a message that has an user defined structure based on custom RIM refined classes.

The Test Data Generator is intended to allow generation of HL7v3 messages that can be used within different testing languages. This means that the format of the generated messages should match the testing language specific format. In order to achieve this, a clear distinction should be made between the process of generating HL7v3 messages, which is language independent, and the adaptation of the generated messages to a language specific format. This adaptation consists of a conversion from the base structure of the generated message to the specific testing language format. This separation proves useful when there is a need for changing the testing language that a particular test system uses. In this case, the generated messages are still compatible with the replacing testing language, provided that a suitable format converter is implemented.

In order to allow the separation of the two functionalities of the test data generating process, the HL7v3 messages should be generated in a simple, generic format that can easily be translated to other testing language dependent formats. Extensible Markup Language (XML) has been chosen as the base format for the messages. The generated HL7v3 messages that adhere to the XML format are grouped under the name raw test data.

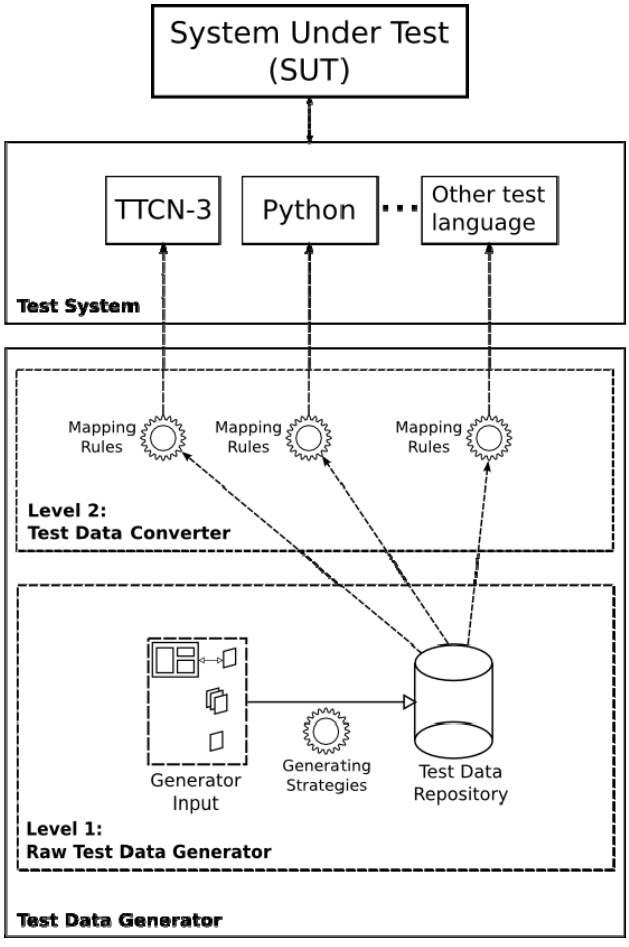


Fig. 1. Overview of the Test Data Generator architecture

The architecture of the Test Data Generator consists of two levels: the level of generating the raw test data, namely the Raw Test Data Generator (RTDG), and the level that represents the conversion of raw test data to different formats, namely the Test Data Converter (TDC).

Fig. 1 depicts the overview of the architectural design. These two levels are designed to allow test designers a great flexibility when choosing the target testing language and its correspondent test system.

The first level of the architecture is represented by the RTDG. This is the main component of the Test Data Generator and it is responsible for generating the raw test data. It allows two levels of customization that enable the test designer to build specific types of messages. RTDG can be used to generate messages one by one, with the help of an integrated Graphical User Interface (GUI), or as a batch process, in case a large repository of test data is needed.

The second level of the architecture describes TDC. This level consists of several independent modules whose responsibility is to convert the raw test data from the XML format to any test language specific format. This level represents the key element that links the generated raw test data with the test languages used by different test systems.

## 5. IMPLEMENTATION OF THE TEST DATA GENERATOR

### *5.1 Generating raw test data*

The Raw Test Data Generator (RTDG) is the first layer of the Test Data Generator and is responsible for generating custom Hl7v3 messages. Ensuring the independence of the generated input data from the testing system provides the testing team a great level of flexibility, but leads to the addition of an intermediate layer between the input data and the test system. However, as described in the Subsection 5.2, this translation is not difficult to realize.

RTDG provides two ways to customize the generating process of HL7v3 messages. First one applies to the message's field values. This type of customization is named low-level customization, since it does not provide means to customize the message as a whole. Instead, it affects only small parts of it. As opposed, the other type of customization, named high-level customization, enables the generation of certain type of messages, by constraining them to match a specific message template.

*The low level customization of the test data generator*

One of the fundamental concepts used in clustering theory is the distance. Distances are metrics that are used to determine the similarity of two values. In the clustering context, they can be used to validate the membership of a value to a certain cluster. The same concept is applied to the Test Data Generator. There are several distances used by the Test Data Generator for customizing the generating process. By gradually changing the distances, messages with various levels of similarity can be generated.

In order to generate messages in a specific cluster, the tester should first establish the cluster boundaries. This is done by determining the distance thresholds for which all the messages generated using distances in that interval are part of the same cluster. By changing the distance within the thresholds boundaries, a large set of HL7v3 messages that are members of the same cluster can be generated. Similarly, other clusters can be generated by defining new distance thresholds.

Before detailing how distances are used and how they affect the generated messages, the basic modules that form RTDG and their interactions are presented. RTDG uses three input files:

- An XSD file, the XML Schema that defines the HL7v3 types;
- A valid XML file containing a compliant HL7v3 message;
- A configuration file that constrains the generator to take into account different generating information, such as the use of mandatory field values or semantic contexts.

The output of RTDG is another XML file, containing the new HL7v3 message. RTDG is designed to provide a great level of flexibility in terms of the intended output. To adhere to the cluster generator's requirements described in the previous sections, it has to provide two functionalities. The first one is to generate diversified output that can be grouped in several different clusters. The second functionality is to generate HL7v3 messages that have a high degree of similarity that are part of the same cluster. Considering the two requirements, a set of modules that allow customization of the level of similarity between the input HL7v3 message and the generated one were developed.

Before detailing the technical realization of these modules, a brief description of the test data generation process is provided. Firstly, the input XML file representing the base HL7v3 message is parsed and converted to a Document Object Model (DOM) document. The resulting tree is traversed and for each leaf a list of ancestors is saved. This list of ancestors is used when creating an XPath expression to uniquely identify the node in the XML Schema that corresponds to the leaf. The type of the value contained in that leaf is then extracted from the identified node. Depending on that type, certain decisions are made, and the leaf is given a new value. In the last step, when all the leaves have been assigned the generated values, the modified DOM document is saved into an XML file, which represents the newly generated HL7v3 message.

The modules responsible for generating new values for the leaves form the core of RTDG. These modules are named low-level generators. Some of the low level generators are customizable, so that the generated test data matches the need. There are two types of such generators.

The first type consists of generators that create new values, independent of the original ones. This is the case of values that are, for example, of type boolean, or enumerated. In these cases, the new value is not generated considering the old one. These generators only ensure that the new value is a valid one. For example, in case of the enumerated type, a list of possible values is created based on the constraints specified in the XML Schema file. A value is then chosen from that list without taking into account the original one.

This way the generator ensures that all possible values are given equal chances. Another example is the string type for which constraints are defined in the XML Schema file, that restrict it to match a certain regular expression. For this particular case we use an external Java library, Xeger (XEGER, 2009), which allows us to generate values that match the specified regular expression. This type of generators does not allow any customization, being used in the creation of clearly defined or constrained values.

The second type consists of generators that apply a set of transformations to the original value. These are called distance-based generators. The distance is a metric for measuring the level of distinction between two values. The distance-based generators implement various types of distances for different value types. All these distances are customizable in order to allow for a specified level of similarity between values. For example, a string with no constraints defined in the input XML Schema file translates into a string of customizable Levenshtein distance (Yujian and Bo, 2007), while an integer translates into another integer of customizable Euclidean distance. The distance-based generators are the only low level generators that can be customized.

Both types of mentioned generators create new values according to a field type. Although they behave differently, their main scope is to generate field values rather than whole messages. This is the reason why the process they are involved in is called a low level generation. The low level customization refers to the ability to customize the distances used by the distance based generators, in order to allow for a more flexible output of the test data generating process.

Two of the distances that are used and can be adjusted by the low level customization are the Levenshtein distance and the Euclidean distance. They demonstrate that the test data generating process can be customized using distances.

The Levenshtein distance is a metric for measuring the amount of differences between two sequences. It is also called the edit distance. In the strings context, the Levenshtein distance is defined as the minimum number of edits needed to transform a string into another one. As opposed to the Hamming distance, the Levenshtein distance can also be applied in the case of strings of different length. The Levenshtein distance used by the distance-based generator can be adjusted in the process of generating test data through the means of the low level customization.

The Euclidean distance is a metric for measuring the distance between two different points. In the case of integer values, the Euclidean distance calculates as the absolute value of the subtraction of the two numbers. The distance-based generator that uses the Euclidean distance is responsible for generating values that have a customizable distance from the original one.

The customization consists of the possibility of adjusting the distances prior to the start of the generation process. Each distance can be independently adjusted. If the output is intended to have minor differences from the original input message, lower values should be assign to the distance based generators. By setting low distances, the tester generates messages that are part of the same cluster. Higher values assigned to the distance based generators determine major differences between the input and the output message. In this case, the generated message and the original one are not part of the same cluster.

The overview of the low-level generation process is depicted in Fig. 2. This figure shows how the new message is generated with the help of the low level generators. The three examples chosen represent the three most used low-level generators. The first two, which handle the generation of strings without constraints and integers, are distance-based generators. They can be customized to use different distances, according to the intended output message. The third one, which handles strings with constraints, is not customizable and does not generate values considering the original value. Fig. 2 shows how the generated message is formed on the same structure as the original one, the only difference being the values it contains.

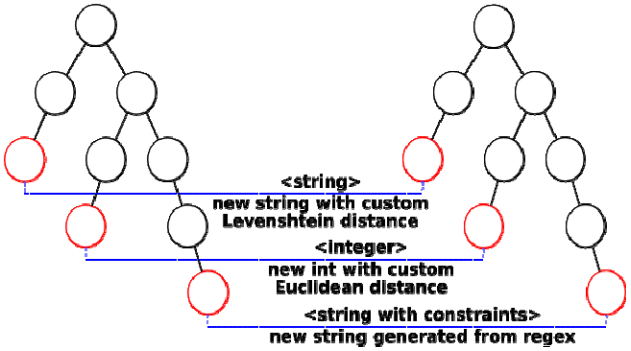


Fig. 2. Low-level generators

A comparison between snippets from the original message and the generated one is shown in

Fig. 3. This example illustrates how low level generators are used to create new messages.


Fig. 3. Example of generated values

Fig. 3 shows different types of situations encountered when generating values:

- The field type is a string, and its value is constrained to match a certain regular expression;
- The field value can be chosen from a list, no other value besides the ones in the list can be assigned to that field;
- The field value is fixed, no other value can be assigned to that field;
- The field type is a string and its value has no defined constraints.

The example illustrates three cases that appear during the generation process. The first case is represented by the fixed values, exemplified by the value INSTANCE of the determinerCode. The XML Schema defines several fixed values that should not be affected by the generating process. The second case is represented by values that have constraints, but are not fixed. These values are handled by the non-customizable low-level generators. The values are generated considering the constraints. These are the cases of the enumerated type and the string that is constrained to match a regular expression. The final case is the one of values that nave no defined constraints. These are handled by the distance-based generators. These are the only cases in which the new value is generated based on the original one. The example shows how a distance-based generator for strings, which uses the Levenshtein distance, generates new values. With the Levenshtein distance set to 3, the value QEDTest of the attribute extension becomes sg0QEDTest.

*The high level customization of the test data generator*

The fact that RTDG can be used to create test data repositories for different applications, using different message structures, is dependent on the level of customization provided by the generating process. In the previous section, only the low level customization, which is ensured by the distance-based generators, has been mentioned. This section reveals other components that enable further customization of the generating process, which form the high-level customization level.

As the paper presented the low level customization of the test data generating process, the distance-based generators were mentioned. Test designers should be offered a method to customize test data as a whole, as well. This process is called high-level customization. There are various ways in which high level customization can be used. One useful feature is the possibility of adding constraints to the existing ones defined in the XML Schema file on certain fields. This is useful when adjusting the normal behaviour of the low level customizations to specific scenarios. An example is the restriction of a plain string, with no constraints defined in the XML Schema file, to match a certain regular expression, or of an integer value to have specified boundaries. Another feature is the possibility of obstructing low-level customization on certain fields of the message. These operations overrule the low level customizations that normally apply to each value of the generated test data.

Fig. 4 presents the general architecture of RTDG, highlighting the two types of customization and the way they influence the test data generating process.

As presented in Fig. 2 and in Fig. 3, the generating process consists of modifying the values of an existing message according to certain rules. These rules are defined in two different contexts.

The first context is represented by the low level generators. At this level, two types of rules can be defined. The first type of rules is defined in the XML Schema. An example of such rule is that a field must have a fixed specific value. This type of rule cannot be overruled by external mean. This is shown as the lowest of the three lines, which represent the low level generators. The other type of rule that can be defined is represented by the ones used by the distance-based generators. These rules may be overridden and are shown as the upper two lines describing the low level generators.

The second context is the high level customization. At this level, rules can be defined in the configuration file, or by using the Graphical User Interface (GUI). There is no difference in terms of the behaviour of the generating process when defining rules in the configuration file or by using the GUI. These rules override the ones governing the low level customization.

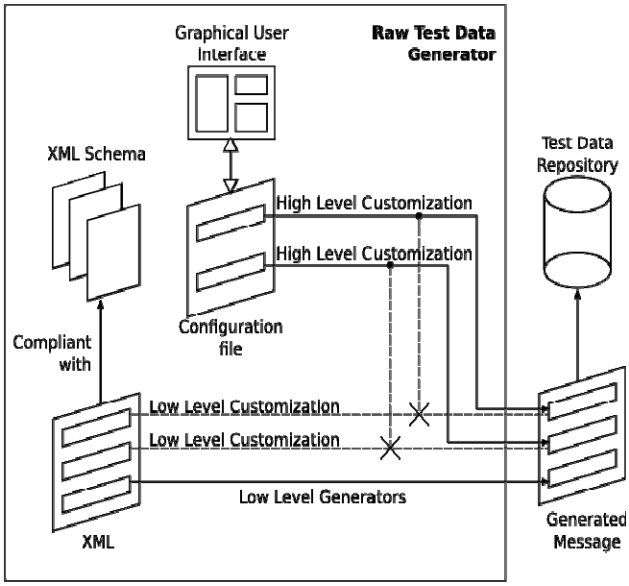


Fig. 4. Architecture of the Raw Test Data Generator (RTDG)

As shown in

Fig. 4, there are two components that handle the high level customization, which are tightly linked: the configuration file and the Graphical User Interface (GUI).

The use of an additional configuration file enables a method to further customize the test data generation process. The configuration file should allow the test designers to define specific message types and rules, suitable to a particular test purpose or test objective. The configuration file should consist of a set of constraints that apply to fields of the

message in the course of the generating process. The authors have identified several types of constraints that can be defined in the configuration file, such as:

- Fixed values for particular fields within the message;
- Values that are chosen from a fixed set;
- Values that match a certain regular expression;
- Values that are system dependent.

The constraints defined in the configuration file override the ones used in the low level customization. The values that have no constraints defined in the configuration file should be generated based on the low level customization. The high level customization is useful when test designers want to define clusters of messages. A snippet of how the configuration file may look like is shown in

Fig. 5.

```
<extension>
  <ancestors value="QUPC_IN043100UV01 receiver device id"/>
  <pattern value="[a-zA-Z][a-zA-Z0-9-_]*"/>
  <original value="qed"/>
</extension>
```

Fig. 5. Example of a string constraint defined in the configuration file

Fig. 5 shows how a constraint can be defined for a string. In this case, a value constraint is defined for node “id”. Since there can be several nodes with the name id, a list of ancestors for that node should be defined, in order to identify the correct one. This list contains all the nodes in the tree path, starting from the root node of the message. Next, the constraint is defined. In this case it is a regular expression that the new value of the id node should match. Finally, the original value is defined. This original value is not the same as the value of the field id within the HL7v3 message. Instead, it is used to overlap to that value, so that it provides another starting point to the distance based generators. In this particular case, where constraint is defined as a regular expression, there is no need for the original value, since the value is not created with the help of a distance-based generator. However, when defining other types of constraints, having the possibility to define another starting point for the generation process can be an useful asset.

The GUI component is the visual equivalent of the configuration file. The GUI has two main characteristics. The first one is to provide a mean for visualization of HL7v3 messages. This includes displaying information about the constraints defined in the XML Schema file for each field value of the message. The second one is to allow the definition of high-level constraints, in the same way as with the configuration file. An example of how a test designer would use this GUI is to load the input XML file, select the fields that he wants to customize and set new constraints, overlapping the original ones, and finally, generate the new message.

The configuration file has, as mentioned, a similar role to the generation of test data as the GUI. Their objective is the same, but they differ mainly in one important aspect. While the GUI is useful for test designers who want to visualize the input test data and define restrictions in an easier way based on it, it has the drawback of not being an automated process. As opposed, the use of a configuration file is better when batch processing, allowing for a larger set of test data to be defined. For those test designers who need both, the implementation of other components, such as converters between the two formats, or GUI generators based on the configuration file, the XML and the XML Schema, may prove useful.

### *5.2 Converting raw test data*

The second level of the Test Data Generator is the Test Data Converter (TDC), the logical level above RTDG, as shown in

Fig. 1. This level is responsible for linking the raw test data with test systems. Many of these test systems handle data in a proprietary format, in which case there should be a way to transform the generated raw test data into that specific format.

To validate this solution, Testing and Test Control Notation version 3 (TTCN-3) was selected as the target testing language. The paper illustrates two examples of TDCs. Both of them are used to adjust the test data to the TTCN-3 format. These examples should substantiate the use of the Test Data Generator.

#### *Test Data Converter for specific HL7v3 profiles*

This TDC is a simple GUI interface that allows for a quick transition between the XML format, in which HL7v3 messages are defined, and the TTCN-3 template format. Its purpose is to provide a mean for the test data to be used when defining test cases in the TTCN-3 environment. It is not meant for complete TTCN-3 test system solutions. It is helpful for the test designers who define the Abstract Test Specification.

The conversion of the test data from XML to TTCN-3 is not straightforward. Since TTCN-3 has no reflection mechanism, there is no immediate way of assigning values to the template’s fields at runtime. For this reason, an intermediate layer was added: a set of Java classes defining HL7v3 types. The process to convert test data between the two formats follows the next steps. Firstly an XML file is loaded. The file is checked if it contains a valid HL7v3 message. If it does, it is parsed and converted into a DOM document. Each node is later translated into a Java object, which is serialized in a specific format. The output is a TTCN-3 file containing all the information that was stored in the XML file.

Even though the test data is not directly converted to the TTCN-3 format and the process needs an intermediate layer, the concept is a valid one and can be implemented. Moreover, TDCs for other scripting languages, such as Python or Tcl, may prove easier to implement.

The drawback of using this type of TDC is that each change of scenario, such as changing the HL7v3 profile, or the target testing language, leads to developing a completely new

converter. The set of Java classes that were use to intermediate the conversion describe general HL7v3 types and specific HL7v3 messages, as defined by the Query for Existing Data (QED), which is a HL7v3 profile. The conversion of other type of HL7v3 messages assumes the existence of other Java classes that act as the intermediate layer.

*Test Data Converter that uses the TTCN-3 test system components*

This example shows another approach on the conversion of the HL7v3 messages from one format to other. This approach has the advantage of automating the conversion process, and is adaptable to changes to other HL7v3 profiles. The drawback is that it can only be used to convert test data to the TTCN-3 template format, and that the test designer should be familiar with the architecture of a TTCN-3 test system and its components.

Fig. 6 shows the architecture of a TTCN-3 test system. The Encoder/Decoder (CoDec) component, which is used in the conversion process, is detailed. The CoDec is the TTCN-3 test system component used for encoding or decoding the TTCN-3 templates into objects that are passed to the TTCN-3 Executable component. It assures the link between the TTCN-3 templates and the TTCN-3 test system. In this approach, the CoDec component is used to decode values into TTCN-3 templates.

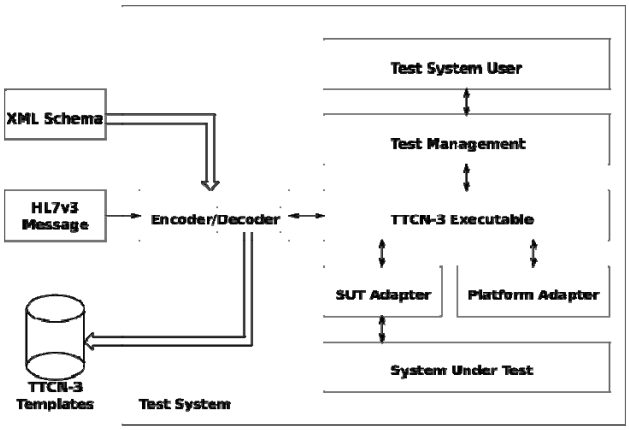


Fig. 6. TTCN-3 Template Generator

There are tools that automatically generate the CoDec component based on XML Schemas, such as TTWorkbench (TTE, 2009). The CoDec has two functionalities. The first one is encoding data, i.e. converting data from TTCN-3 templates to different types of objects that can be used by the TTCN-3 test system. The second one is decoding data, the inverse operation. The objective is to obtain TTCN-3 templates in this approach and for this, only the second functionality of the CoDec, the decoding of data, is considered.

The process of converting the generated raw test data into TTCN-3 templates follows the next steps. The raw test data, which is contained in XML files, is parsed and passed as input to the CoDec. The decode method of the CoDec is then called. The output is a TTCN-3 template that is serialized in a TTCN-3 file.

This approach is suited for test designers that choose TTCN-3 as the target testing language and are familiar to the TTCN-3 test system architecture.

With the two proof of concepts shown, the paper illustrated that generated raw test data can be converted to different types of formats, that correspond to different testing languages, in several ways.

## 6. CONCLUSIONS AND FUTURE WORK

This paper presented a new approach to generating repositories of test data for testing HL7v3 based applications. The importance of customizing the generating process was highlighted and a method to achieve the customization level was presented. An architecture for the Test Data Generator, consisting of two levels of customization that enable the test developer with a great flexibility when designing test inputs was proposed. The paper also illustrated how can the Test Data Generator be adjusted and used for generating clusters of test data. This paper discussed different ways in which the Test Data Generator can be used in conjunction with different testing language and their correspondent test systems. The case study validated the concept of using the Test Data Generator in the context of TTCN-3 test system.

Future research focuses on extending customization of the process by adding new distance based generators and defining metrics to measure distances between entire HL7v3 messages. Another research direction is to investigate the possibility of constraining field values and messages using the constraint programming paradigm.

## REFERENCES

Cilibrasi, R.L. and Vitanyi, P.M.B. (2007). The Google Similarity Distance. *IEEE Transactions on Knowledge and Data Engineering IEEE Educational Activities Department*, 19(3), 370–383. ISSN 1041-4347.

Ciupa, I., Leitner, A., Oriol, M., and Meyer, B. (2006). Object distance and its application to adaptive random testing of object-oriented programs. In *RT '06: Proceedings of the 1st international workshop on Randomtesting*, 55–63. ACM Press, New York, NY, USA.

Davis, M. and Weyuker, E. (1988). Metric space-based test-base adequacy criteria. *The Computer Journal*, 31(1), 17–24.

ETSI (2007a). European Telecommunications Standards Institute (ETSI). European Standard (ES) 201 873-1 V3.2.1 (2007-02): The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language. Sophia-Antipolis, France.

ETSI (2007b). European Telecommunications Standards Institute (ETSI). European Standard (ES) 201 873-5 V3.2.1 (2007-02): The Testing and Test Control

Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI). Sophia-Antipolis, France.

Gower, J. (1985). Properties of Euclidean and non-Euclidean distance matrices. *Linear Algebra and its Applications*, 67, 81–97.

HL7 - Health Level Seven Standards Developing Organizations (HL7 SDOs). Health Level Seven (HL7) Messaging Standards Standard, Version 3. www.hl7.org/v3ballot/html/index.htm. Accessed 1 Feb. 2011.

Jain, A.K., Murty, M.N., and Flynn, P.J. (1999). Data Clustering: A Review. *ACM Computing Surveys*, Vol. 31, No. 3, pp. 264–323.

Namli, T., Aluc, G., and Dogac, A. (2009). An interoperability test framework for HL7-based systems. *IEEE Transactions on Information Technology in Biomedicine*, 13(3), 389–399.

RTMS (2007-2009). ReTeMes Project: Reliability Testing Of Medical Systems, EUREKA European Project (E!4053 RETEMES). www.eurekanetwork.org/project/-/id/4053. Accessed 20 Jun. 2012.

Sneed, H.M. (2004). Measuring the Effectiveness of Software Testing. *Proceedings of SOQUA 2004 and TECOS 2004*, volume 58 of Lecture Notes in Informatics (LNI). Gesellschaft fur Informatik.

TTE (2009). Testing Technologies IST GmbH. TTworkbench Professional, an Eclipse based TTCN-3 IDE. www.testingtech.com/products/ttworkbench.php. Accessed 20 Jun. 2012.

Vega, D., Schieferdecker, I., and Din, G. (2010). Design of a Test Framework for Automated Interoperability Testing of Healthcare Information Systems. *In eTELEMED 2010: Proceedings of the Second International Conference on eHealth, Telemedicine, and Social Medicine*, St. Maarten, Netherlands Antilles, 123–130.

XEGER (2009). Xeger java library for generating random text from regular expressions. http://code.google.com/p/xeger. Accessed 20 Jun. 2012.

Yujian, L. and Bo, L. (2007). A Normalized Levenshtein Distance Metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6), 1091–1095.