TTCN-3 Codec Implementation for LTE RRM Conformance Test

Younan Duan, Zhizhong Ding*, Teng Huang, Dingliang Wang
Institute of Communications and Information Systems, Department of Communication Engineering, Hefei University of Technology, Hefei, 230009, China.

* Corresponding author. Tel.: 86-0551-62901552; email: zzding@hfut.edu.cn
Manuscript submitted September 22, 2014; accepted July 20, 2015.
doi: 10.17706/jsw.10.10.1176-1188

Abstract: Radio Resource Management (RRM) is a system level control of co-channel interference and other radio transmission characteristics in Long Term Evolution (LTE) systems. Testing and Test Control Notation (TTCN-3) is an international standardized abstract programming language for test development. Its abstraction, scalability and standardization of interfaces make TTCN-3 the best choice for implementing the RRM test suite. As an important entity in testing systems, the codec encodes TTCN-3 values into a bitstring according to the encoding rule, and decodes a bitstring according to decoding hypothesis. There are 13 kinds of message type definitions and seven encoding rules in the RRM conformance testing system, so the developers should implement multiple codecs according to different encoding rules. This paper mainly focuses on the common programming method for the codecs required by the RRM test suite. First, the significance of codecs in TTCN-3 testing systems is explained, then the TTCN-3 Control Interface is introduced, and finally the implementation of codecs for specific type definitions in the RRM conformance testing system is shown.

Key words: RRM, TTCN-3, TCI, codec, encode, decode.

1. Introduction

Radio Resource Management is a system level control of co-channel interference and other radio transmission characteristics in LTE wireless communication systems. It ensures the efficient use of the available radio resources and provides mechanisms that enable LTE air interface E-UTRAN to meet the related radio resource requirements [1].

TTCN-3 is an international standardized abstract programming language for test development, which is maintained by the European Telecommunications Standards Institute (ETSI). Its abstraction, scalability and standardization of interfaces make TTCN-3 the best choice for RRM conformance test [2].

During the development of a TTCN-3 testing system, some developers implement a single common codec for all the message types with the same encoding rule. This simple solution is accepted in some small-scale testing systems which do not strictly limit the encoding rules. However, as shown in section three, there are 13 kinds of message type definitions and seven encoding rules in the RRM conformance testing system. For example, the encoding rule named “SACCHSysInfoSparePadding Rule4” is required to limit the length of messages, and the padding bits are given, but the rule “Tabular notated” does not have such requirements. So a single common codec no longer sufficient. In fact, although these encoding rules correspond to different codecs, the implementation of these codecs still has some similarities, such as message field

arrangement and treatment of variable field length. So the implementation of these codecs is the key to the testing system.

This paper is mainly describing the implementation of codecs in the RRM conformance testing system. First, in section two, the structure of a test system is introduced, then an analysis of the TTCN-3 Control Interface for Codec (TCI-CD) is given. Section three explains the structure, the IDE and encoding rules of RRM conformance testing system. Finally in section four, the core issue of this paper, a common method for achieving the codecs is proposed. By using this method the codec named "SACCHSysInfoSparePadding Rule4" has been completed successfully, and its feasibility is proved by experimentation. Note that this paper only discusses the codecs for TTCN-3 types. The codecs for ASN.1 types are out of scope of this study.

2. Codec and TCI-CD Interface

To illustrate the role and significance of the codec, this section first introduces the general architecture of a TTCN-3 test system, then an explanation about TTCN-3 Control Interface (TCI) is given to make a further discussion on the codec.

2.1. Codec in TTCN-3 Testing System

Fig. 1 shows the general architecture of a TTCN-3 test system. All the entities cooperate with each other to implement the communication with System Under Test (SUT). The Test Management (TM) is responsible for the management of whole testing system. The test log is generated and maintained by the Test Logging (TL) [3]. The Component Handling (CH) is responsible for the management of components, especially in parallel test system. These four entities communicate with the Test Executable (TE) through the Test Control Interface (TCI), and TE finishes the execution of test suites descripted by TTCN-3 [3]. TTCN-3 data types are converted to bitstring and vice versa by the codec. To provide the communication between TE and SUT, The SUT Adapter (SA) is designed. The Platform Adapter (PA), which implements TTCN-3 external functions and provides a TTCN-3 test system with a single notion of time, communicates with TE through the TTCN-3 Runtime Interface (TRI) as well as the SA [4].

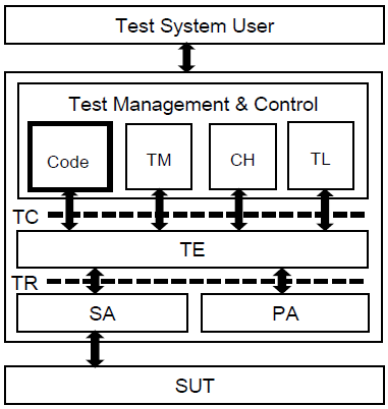


Fig. 1. General architecture of a TTCN-3 test system.

Now it is necessary to focus on the interaction of test system entities. Fig. 2 indicates the procedure of data sending and receiving. During the whole procedure, TE firstly uses triExecuteTestcase operation to initialize the connection between the SA and the SUT. Then the TE establishes a dynamic connection to the SUT by using triMap operation. After these two preparation steps, a message is waiting to be sent to the SUT: Firstly the message is delivered from the TE to the codec for encoding, and the encoded message is returned back. Secondly, the TE sends the encoded message to the SA by triSend operation and finally the SA sends it to the SUT. Now the timer starts and the TE waits for the answer from the SUT. Once the answer is received

at this time by the SA, it is immediately sent to the TE by triEnqueueMessage operation. TE then delivers it to the codec again for decoding, and the decoding result is returned back to the TE. At last TE stops the timer and unmaps the dynamic connection to end the procedure.

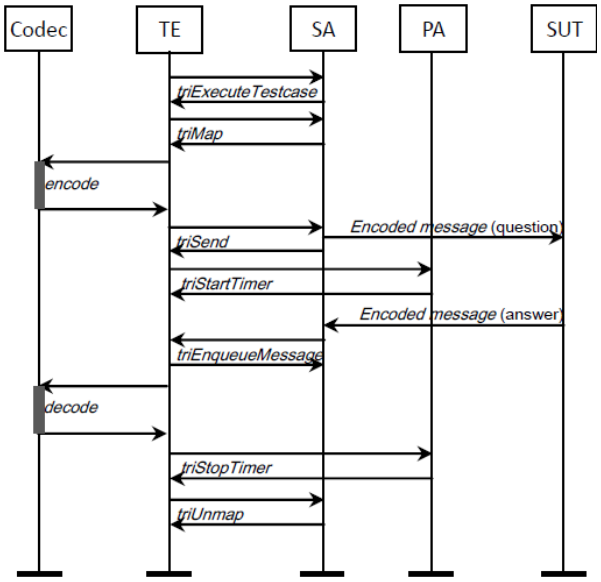


Fig. 2. Procedure of data sending and receiving.

From the above two figures, we have a preliminary understanding on the test system and codec. Because the abstract TTCN-3 values cannot be sent to the SUT directly, a codec implementation is needed to encode TTCN-3 values into a bitstring according to the encoding rule, and decode a bitstring according to decoding hypothesis [5]. During encoding, a TTCN-3 value has to be inspected and dissected to generate a series of binary value mandated by the encoding scheme. During decoding, a new TTCN-3 value has to be created according to the binary values received from the SUT and the receiving templates defined by TTCN-3 [6].

2.2. TCI Data

As previously mentioned, the TCI defines the interaction among the CH, the TM, the codec, the TL and the TE. In order to provide access to the codec for TE, the TCI defines a set of abstract data types which are used to determine

a) how TTCN-3 data is passed from a TE to the codec for encoding a TTCN-3 value to a bitstring, and

b) how data from the codec to the TE shall be decoded from a bitstring to a TTCN-3 value. [6]

In order to process the data for these abstract data types, a set of operations are defined by the codec. And functionality of each data type is defined by an associated set of operations and methods. Operations on these abstract data types return either a value of the same type or a basic type, such as integer.

An abstract TTCN-3 type and value representation includes an abstract data Type that represents all types in a module and the values of a given type. In fact, the testing system uses Type and its values for accessing, evaluating, and encoding or decoding data.

For the *Type*, a set of operations are defined to reference predefined and user-defined TTCN-3 data types, and these operations also create and maintain TTCN-3 values. For example, the *getName* returns the name of the type defined in the TE and the *getTypeEncoding* returns the type encoding attribute defined in the TTCN-3 module. For the TTCN-3 values, they are represented at the TCI via numerous abstract data types. Fig. 3 shows the hierarchy between the abstract data types for TTCN-3 values. All the abstract values, such as *IntegerValue*, *HexstringValue* and *RecordValue*, share the same base abstract data type Value. All

operations defined on this common base data type are implicitly defined also for the abstract value types derived from it. For example, the *getType* operation returns the type of the specified value which defined in the Value, and the *BitstringValue* type also shares this operation. In addition, *BitstringValue* has its own operations such as *getString* which returns the textual representation of it.

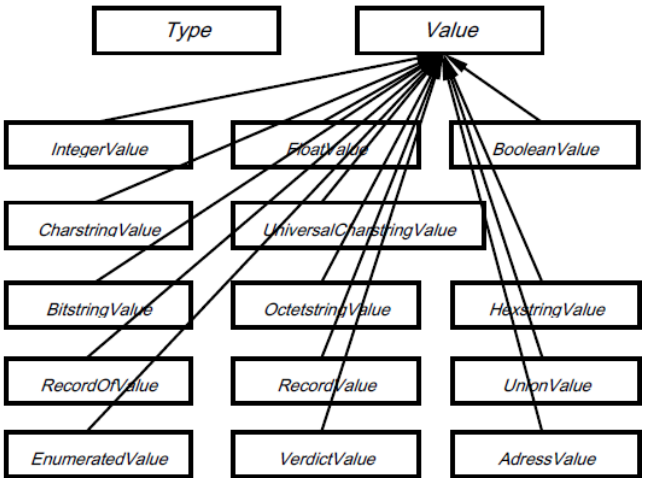


Fig. 3. Hierarchy between abstract data types [6].

2.3. TCI-CD Interface

In this part, we explain the relationship between the codec and TCI-CD Interface. We will see all what needs to be done is to implement encode and decode operations based on encoding rules.

The TCI-CD describes two parts: the operations that a TE is required to implement, and codec implementation that shall provide to the TE. The above operations are used to access codecs for encoding data to be sent, and decoding received data [6].

TCI-CD is divided into two types of interface: TCI-CD required and TCI-CD provided. The former specifies the operations that the CD requires from the TE, and the latter specifies the operations that the TM shall provide to the TE.

To be precise, TCI-CD provided includes the encode and decode operation. The *encode* shall be called whenever the TE has to encode a Value, and the *decode* whenever the TE has to decode an encoded value. In other words, the *encode* and *decode* operations are respectively the entrance of encoding and decoding process. And TCI-CD required includes a series of operations for codec to use, such as *getBitstring* which constructs and returns a basic TTCN-3 bitstring type.

For the construction of the encoded data, which is returned by the encode operation, the TCI only specifies the concrete value structure. However, the operations for the construction of such values are not specified by the TCI. Therefore the codec must implement such operations itself.

So the implementation process of codec is to use Type and Value interfaces and TCI-CD required to implement TCI-CD provided, i.e. *encode* and *decode* operations.

3. LTE RRM Conformance Testing

3.1. Structure of the Testing System

On the basis of the general architecture of a TTCN-3 test system, the structure of RRM conformance testing system is shown in Fig. 4. Host-PC includes the test control, TTCN-3 test suite and associated codecs. The System Simulator (SS) provides the simulation of eNBs and achieves the adaption to the SUT. In this system, the User Equipment (UE, i.e. LTE mobile device) acts as the SUT [7][8]. Host-PC connects the SS

through internal system interface, and the SS connects UE through LTE air interface.

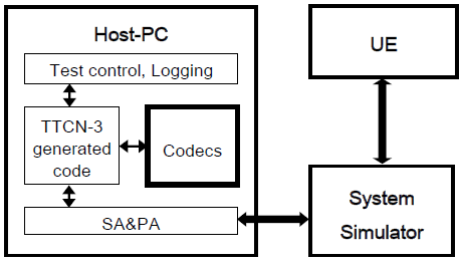


Fig. 4. Structure of RRM conformance testing system.

The part of Host-PC shown in Fig. 4 is what the developer of test suites should mainly focus on. The purpose of the test development is to implement the entities in the Host-PC and complete the interaction between them.

3.2. Brief Introduction to IDE

A suitable IDE is an important guarantee for such a complex task. Here we choose TT workbench 15 as the IDE of the whole project, which is launched by Testing Tech from Germany. TT workbench is under Eclipse framework and supported by Java virtual machine. Some entities, together with their interaction has been finished by inside codes such as CH, TL and SA. Therefore the developers just need to focus on TE and codecs. In addition, it supports ASN.1 and provides codecs for ASN.1 by plugins, so this paper only discusses the codecs for TTCN-3 types.

There are several codecs in TT workbench, including a general codec which translate all the TTCN-3 data into bitstring in a generic way. But these codecs do not apply to RRM conformance testing system, because they do not comply with the encoding rules mentioned in section three, part C. However, TT workench provides a certain convenience for the codec development, including reliability of data transmit between entities and exclusive operations for data. These features simplify high level development.

3.3. Requirements for Codecs

As shown in Table 1, there are 13 kinds of type definitions in the LTE RRM test suites, and each type definition has multiple message types. For the first two type definitions, the special ASN.1 encoding rules BER and PER are respectively needed, which are out of scope of this paper. The next six type definitions share the same encoding rule named Tabular notated, which concatenates all the present fields in the TTCN-3 template. For the last type definition in Table I, there is no specific encoding rule because the messages defined by these types do not need to be sent out of TE. Other type definitions have their own specific rules.

For example, type definition *SACCHSysInfo* Spare Padding includes 3 record types, and they need a special encoding rule named *SACCHSysInfoSparePadding* Rule4. It can be simplified as 3 points:

a) Concatenate all the present fields in the TTCN-3 template.

b) The encoded message has a maximum of 19 octets.

c) If the encoded message is shorter than 19 octets, it should be filled with “spare padding”- a repetition of octet ‘00101011’.

This rule can be found in [9], clause 6.10.2.9.5. In addition to the difference among message types, all the rules for TTCN-3 types listed in TABLE 1 have strong similarity. They just differ from each other on the length limit and requirements of specific fields. All these rules are explained in [9].

How to implement these codecs is the key to the testing system. In next section, we will be involved in

this core issue.

Table 1. Encoding Rules for Different Type Definitions

Type definitions	Encoding rules
ASN.1 types used for RRC signalling	ASN.1 PER
ASN.1 types used by NAS protocols	ASN.1 BER
NAS types SMS types DRB types DHCPv4 types ICMPv6 types GERAN types	Tabular notated
GPRS Padding	GPRSPadding Rule1
GSM Spare Padding	GSM Spare Padding Rule2
LowHigh Rule	LowHighRule3
SACCHSysInfo Spare Padding	SACCHSysInfoSparePadding Rule4
TTCN-3 types not used at the air interface: - Configuration of system simulator - Coordination between components - Types used internally in TTCN-3	No specific encoding required

4. The Implementation of Codec

In general, the implementation of codecs can be summarized as follows: first the encode operation finishes choosing the encoding method based on the message structure, then the specific encoding method is called to deal with its corresponding message structure, finally it returns the encoded bitstrings back to the encode operation, and the encode operation returns the bitstrings to TE. Decoding is a similar procedure [10].

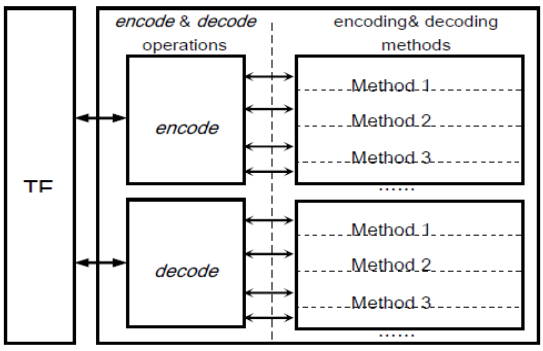


Fig. 5. Architecture of the codec programming.

As mentioned in section two, the implementation process of codec is to implement encode and decode operations. In fact, a codec often deals with multiple message structures. Different encoding and decoding methods are programmed for corresponding message structures. Therefore the encode operation not only returns the encoded data, but also finishes the selection among multiple encoding methods according to the message structure. Similarly, the decode operation not only attempts to construct a TTCN-3 value, but also finishes the selection among multiple decoding methods according to the expected message structure [11].

Fig. 5 summarizes the architecture of the codec programming. To achieve good encapsulation, we put encode operation on high level of codec programming and encoding methods on low level. When the encode operation is called by TE, it starts to choose and call the specific encoding method, and finally return the

encoded bitstring. Considering that different message structure may have the same field types, the developers can program subfunctions to encode different fields independently. For the integrity of the elaboration, this part is incorporated into the encoding method. It is the same in decoding procedure.

Before implementing the codecs, the programming language should be chosen. TTworkbench uses Java virtual machine to support TTCN-3 projects, so here the codecs are generated by Java code. A developer may choose ANSI C, C++, C# or XML as the language of codecs based on the requirement of different IDE, but the programming procedure is similar [12]. The detailed steps are introduced in next four parts.

4.1. Encode Operations

As the entrance of the entire encoding procedure, the *encode* operation not only returns the encoded data, but also finishes the selection among multiple encoding methods according to the message structure. Table 2 shows the feature of *encode* operation.

Table 2. The Feature of Encode Operation [6]

Signature	TriMessageType encode(in Value value)
In parameters	value(The value to be encoded)
Return Value	An encoded *TriMessage* for the specified encoding rule.
Constraint	This operation shall be called whenever the TE has to encode a Value.
Effetc	Returns an encoded TriMessage according to the encoding rules.

To achieve the first requirement, the *encode* operation returns an encoded *TriMessage* according to the encoding rule by using *return* statement.

To achieve the second requirement, we can use *if* statements. The type name will be obtained and checked to find the appropriate encoding methods. Using *Value.getType()* to return the type of the *Value*, and *Type.getName()* to get the name of the type defined in the TTCN-3 module, then compare it with the expected type name.

Considering the example used in section three, in "*SACCHSysInfoSparePadding Rule4*", 3 different record types are supported. They are:

- Systeminformationtype5
- Systeminformationtype5bis
- Systeminformationtype6

Each type has a specific encoding method. Here is the Java code:

```
public TriMessage encode(Value template) {
    try {
        if(template.getType().getName().equals("SYSTEMINFORMATIONTYPE5"))
            return encodeSysInfo5((RecordValue) template);
        else
            if(template.getType().getName().equals("SYSTEMINFORMATIONTYPE5bis"))
                return encodeSysInfo5bis((RecordValue) template);
            else
                if(template.getType().getName().equals("SYSTEMINFORMATIONTYPE6"))
                    return encodeSysInfo6((RecordValue) template);
                else
                    return null;
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

After entering the *encode* operation, the specific encoding method is determined.

4.2. Encoding Method

If it is a simple value without any hierarchies or branch structures, we just need to turn it into a bitstring. But if the value has structured types, such as record or set, we need to get every field, turn them into bitstrings one by one, and finally concatenate all the bitstrings.

The steps of encoding methods can be summarized as follows:

- Define *ByteArrayOutputStream out* to receive the encoding result. *ByteArrayOutputStream* type is used to cache data. Its buffer is automatically increased, and the data can be extracted when finishing writing. So we choose it to cache the encoded bits.
- Use *getField(fieldName)* operation to get the data in the field *fieldname.* This operation belongs to TCI *Value* interface.
- Convert the data to bytes. For different data types, the operations vary from each other. For example, for the string type, a *getByte()* operation is available. Or for the bitstring type, the bits need to be ordered in eight per group. Then the data is written into *out* by *write* operation.
- **(for the optional field and variable field length)** Use one bit as the flag if the field is optional. When the flag equals zero and one indicates that the field is omitted and has an exact value respectively. Write the flag into *out* in front of the field content.
- **(for the variable field length)** Similarly, we can write the length of the field between the flag and the field content. Here multiple consequent bits may be used. The number of bits depends on the length, and it cannot be changed once determined. Fig. 6 indicates the encoding field structure.

Fig. 6. Encoding field structure.

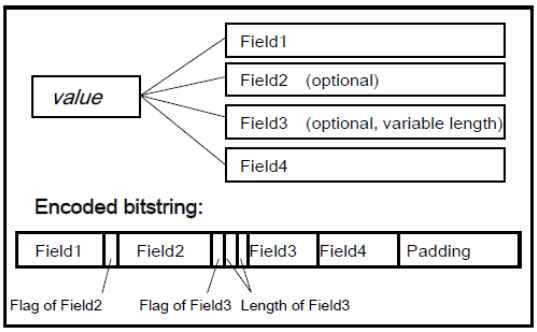


Fig. 7. Correspondence between TTCN-3 value and its encoded bitstring.

- **(for the length limit of message)** If the length of the encoded message does not reach the limit, fill the blank with a series of padding bits, otherwise generate warning information if the length exceeds.

Keep in mind that the smallest unit in Java stream is byte (1 byte=8 bits), so if the present bits are less than eight, it must waits for subsequent bits to reach 1 byte, then use *out.write()* operation to write them into out.

After all the fields are encoded, the out is converted into a byte array. Here we use *toByteArray()* operation. Then the byte array is converted into *TriMessage* and returned to *encode* operation by using *TriMessageImpl* operation. Note that this operation is not required in TCI standard. It is just supported by TTworkbench. Encoding process is then finished. Fig. 7 indicates the correspondence between TTCN-3 value and its encoded bitstring. The flowchart of whole encoding procedure is shown in Fig. 8.

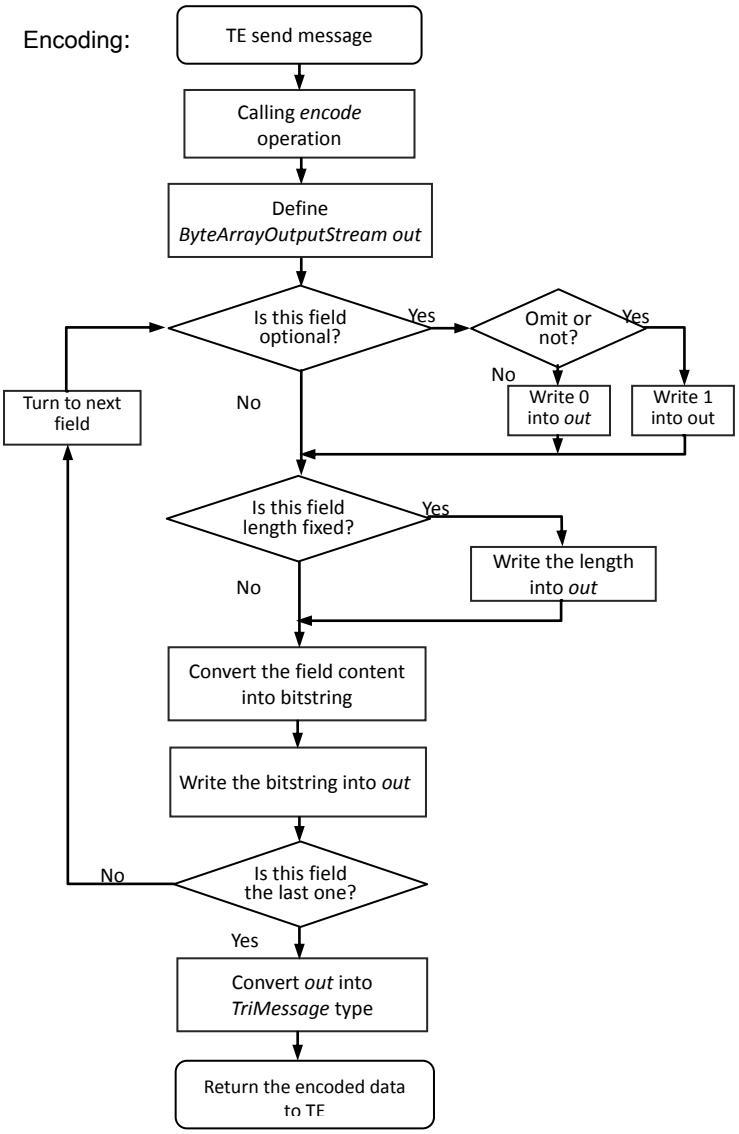


Fig. 8. Flowchart of encoding procedure.

4.3. Decode Operation

Similarly, as another entrance of the entire decoding procedure, the decode operation not only attempts to construct a TTCN-3 value based on the expected message type and the encoded data, but also finishes the selection among multiple encoding methods according to the message structure. Table 3 shows the feature of encode operation.

To fulfill the first requirement, the *decode* operation should return a decoded Value according to the encoding rules by using return statement.

To achieve the second requirement, we still use if statements. The type name of *decodingHypothesis* will be obtained and checked to find the appropriate encoding methods. Using *decodingHypothesis.getName()* to get the name of the type, then compare it with the expected type name.

Still in "*SACCHSysInfoSparePadding Rule4*", like *encode* operation, 3 different record types are supported and each type has a specific decoding method. Here is the Java code:

```
public Value decode(TriMessage message, Type decodingHypothesis) {
    try {
        if(decodingHypothesis.getName().equals("SYSTEMINFORMATIONTYPE5")) {
            decodeSysInfo5(message, decodingHypothesis);
            return result;
        }
```

```
        else
            if(decodingHypothesis.getName().equals("SYSTEMINFORMATI ONTYPE5bis")) {
                decodeSysInfo5bis(message, decodingHypothesis);
                return result;
            }
            else
                if(decodingHypothesis.getName().equals("SYSTEMINFORMATIONTYPE6")) {
                    decodeSysInfo6(message, decodingHypothesis);
                    return result;
                }
                    else
                        return null;
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

After entering the decode operation, the specific decoding method is determined.

Table 3. The Feature of Decode Operation [6]

Signature	Value decode(in TriMessageType message, in Type decodingHypothesis)	
In parameters	message	The message to be decoded.
	decodingHypothesis	The hypothesis the decoding can be based on.
Return Value	If the decoded value is of a compatible type as the decodingHypothesis, return the value. Else return null.	
Constraint	This operation shall be called whenever the TE has to decode an encoded value.	
Effetc	Decodes message according to the encoding rules and returns a TTCN-3 value.	

4.4. Decoding Method

Corresponds with the encoding method, if it is a simple decodingHypothesis without any hierarchy or branch structure, we just turn the bitstring into TTCN-3 value. If the hypothesis has structured types, we should set all the fields based on the decodingHypothesis, then fill them with decoded message. The detailed procedure can be summarized as follows:

- Load the encoding data as a byte array from TRI, and create a new instance named *result* from the *decodingHypothesis* by using *newInstance()* operation. For example, if the expected type is *record*, the statement is as follow:
- RecordValue *result = (RecordValue)decodingHypothesis.newInstance();*
- Create a new instance *fieldInstance* of the first field based on the *decodingHypothesis*. Read a certain length of bits from the encoding message, set them into *fieldInstance* and return it as a decoding field.
- **(for the optional field and variable field length)** Read one bit as the flag before decoding the content. The flag equals 0 or 1 indicates that the field is omitted or has an exact value.
- **(for the variable field length)** Similarly, if the field length is written before the field content, we should read the length bits to get the field length. The number of length bits are determined during encoding.
- Use *setField()* operation to set the field *fieldName* of the result based on *fieldInstance*. This operation belongs to TCI Value interface.

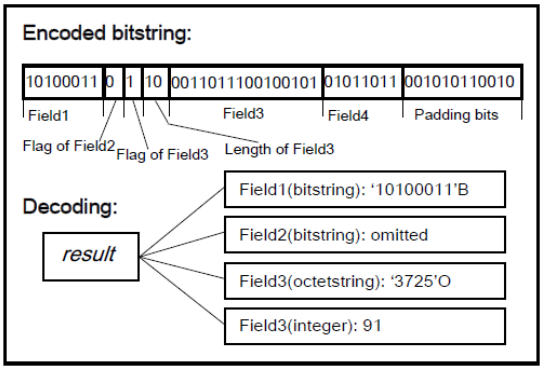


Fig. 9. Decoding result from encoded bitstring.

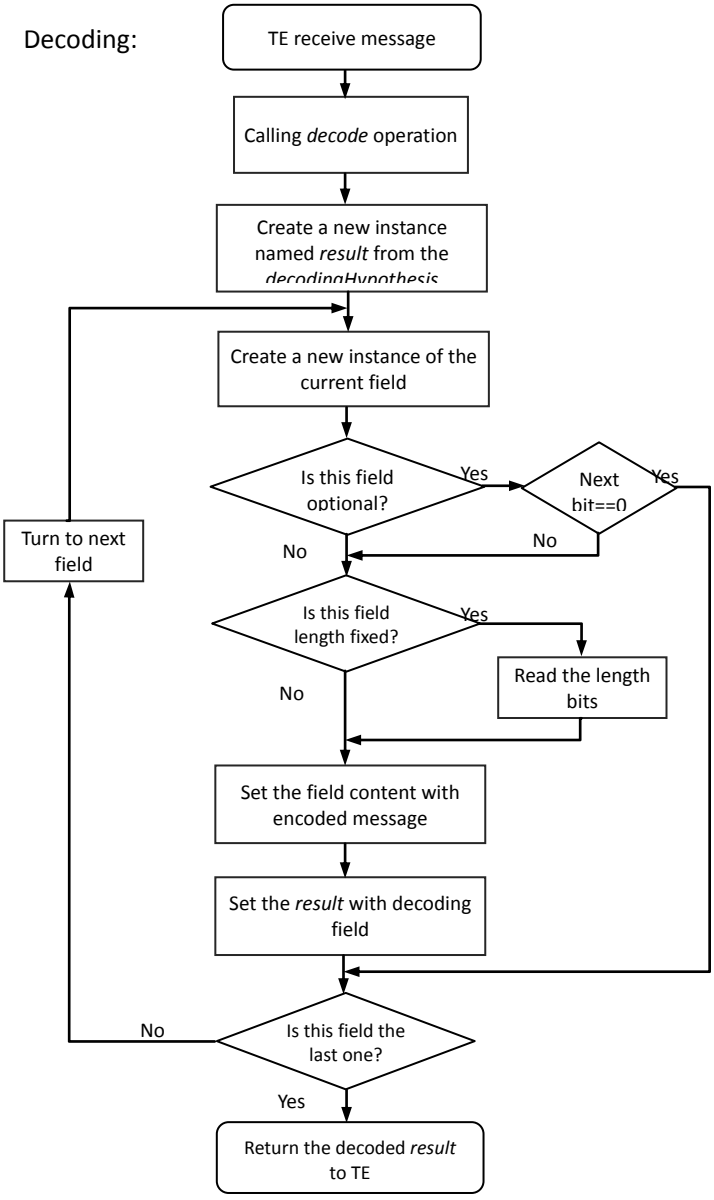


Fig. 10. Flowchart of decoding procedure.

Note that there is no specific step for decoding messages which have length limit. Because the message length has been fixed by encoding procedure. If the padding bits exist, we just ensure that all the fields are decoded correctly, then the remaining padding bits are discarded.

Repeat this procedure until all the fields in the *result* are set and then return the *result* to *decode* operation, which ends decoding process. Fig. 9 indicates the decoding result from an encoded bitstring. The flowchart of whole decoding procedure is shown in Fig. 10.

4.5. Validation Experiments

By using the above methods, the codec "SACCHSysInfoSparePadding Rule4" has been implemented successfully. To validate its feasibility and correctness, the developers design a loopback test system: the encoded message is sent to the computer itself, then the system decodes it and compare to the original message. By this way the system can tests encoding and decoding at one time.

The experimental results show that the encoding and decoding accuracy is 100%, and it works perfectly. Then the codec is added to RRM conformance testing system to complete corresponding function, and the encoding results match the standard requirements.

5. Conclusion

RRM ensures the efficient use of the available radio resources and also provides mechanisms that enable LTE air interface E-UTRAN to meet radio resource related requirements. According to the complexity of the LTE system, the TTCN-3 code for RRM conformance test describes thousands of message types and multi-level data structures which need various specific encoding rules.

This paper first introduces the structure of the RRM conformance testing system, then analyses the relationship between TCI and codec. Finally it proposes a common method for the codecs required by the RRM conformance testing system. By using these methods the codec named "SACCHSysInfoSparePadding Rule4" has been completed successfully.

The feasibility of this codec has been proved by a series of experiments. Next we will gradually implement all the other codecs for RRM conformance test, to boost the completion of the whole RRM conformance test suite.

Acknowledgment

The first author thanks Ying Chen and Yuchuan Liu for their assistance and appreciate the reviewers' valuable comments.

References

[1] Radio resource management. (Wikipedia). Retrieved October, 2013, http://en.wikipedia.org/wiki/Radio_resource_management

[2] WHY USE TTCN-3? Retrieved August 21, 2001, http://www.ttcn-3.org/index.php/about/why-use-ttcn3

[3] Colin, W., Thomas, D., Stephan, T., Stefan, K., Federico, E., & Stephan, S. (2005). *An Introduction to TTCN-3*. John Wiley & Sons Ltd, 23-24.

[4] 3GPP TS 36.523-3 V10.1.0 (2012-09): 3rd generation partnership project; Technical specification group radio access network; Evolved universal terrestrial radio access (E-UTRA) and evolved packet core (EPC); User equipment (UE) conformance specification; Part 3: Test suites (release 10).

[5] Hong, C. D., Limin, L., & Li, X. W. (2012). The application of TTCN-3 in the conformance testing of TD-LTE system. *Proceedings of the 2012 Second International Conference on Business Computing and Global Informatization* (pp. 750-753).

[6] ETSI ES 201 873-6 V4.5.1. (2013-04). Methods for testing and specification (MTS); The Testing and test control notation version 3; Part 6: TTCN-3 control interface (TCI), 42-45.

[7] 3GPP TS 36.300 V9.9.0. (2011-12). 3rd generation partnership project; Technical specification group

radio access network; Evolved universal terrestrial radio access (E-UTRA) and evolved universal terrestrial radio access network (E-UTRAN); Overall description; Stage 2 (release 9).

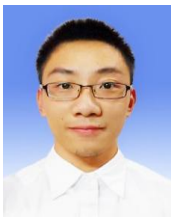
[8] 3GPP TS 36.521-3 V10.2.0. (2012-09). 3rd generation partnership project; Technical specification group radio access network; Evolved universal terrestrial radio access (E-UTRA); User equipment (UE) conformance specification; Radio transmission and reception; Part 3: Radio resource management (RRM) conformance testing(release 10)

[9] 3GPP TS 34.123-3 V10.4.0. (2013-06): 3rd generation partnership project; Technical specification group radio access network; User equipment (UE) conformance specification; Part 3: abstract test Suite (ATS) (release 10), 50-51.

[10] Li, W. H., Ma, Y., Huang, X. H. (2010). Common TTCN-3 codec to reduce test engineering cost. *Proceedings of International Conference on Artificial Intelligence Applications and Innovations*.

[11] ETSI ES 201 873-1 V4.5.1. (2013-02). Methods for testing and specification (MTS); The testing and test control notation version 3; Part 1: TTCN-3 core language.

[12] ETSI ES 201 873-5 V4.5.1. (2013-04). Methods for testing and specification (MTS); The testing and test control notation version 3; Part 5: TTCN-3 runtime interface (TRI).



Younan Duan was born in Bengbu, China, in 1990. He received his B.E degree in communication engineering from Huaibei Normal University in 2012, and received the master degree in signals and information processing from Hefei University of Technology, Anhui province, China, in 2015. His research interests include protocol conformance testing based on TTCN-3, and hardware implementation of 802.11p protocol.

Zhizhong Ding was born in Wuhu, China, in 1961. He received his B.E degree in radio communications from Nanjing University of Aeronautics and Astronautics, Nanjing, China, the master's degree in circuit and system from Hefei University of Technoloy, Hefei, China, and the Ph. D. in information and communication engineering from University of Science and Technology of China. His research interests include wireless communications, network communications and information theory.

He currently is a professor with the Department of Communication Engineering and with the Institute of Communications and Information Systems, Hefei University of Technology.

Teng Huang was born in Hefei, China in 1988. He received his B.E degree in communication engineering from Anqing Normal University in 2012, and the master degree in communication and information systems from Hefei University of Technology, Anhui province, China, in 2015. His research interests include protocol conformance testing and OFDM system.

Dingliang Wang was born in Lujiang in 1991. He received B.E degree in communication engineering from Hefei University of Technology in 2013, and now he is working toward master's degree in HFUT. His research interests include protocol conformance testing and wireless communication.