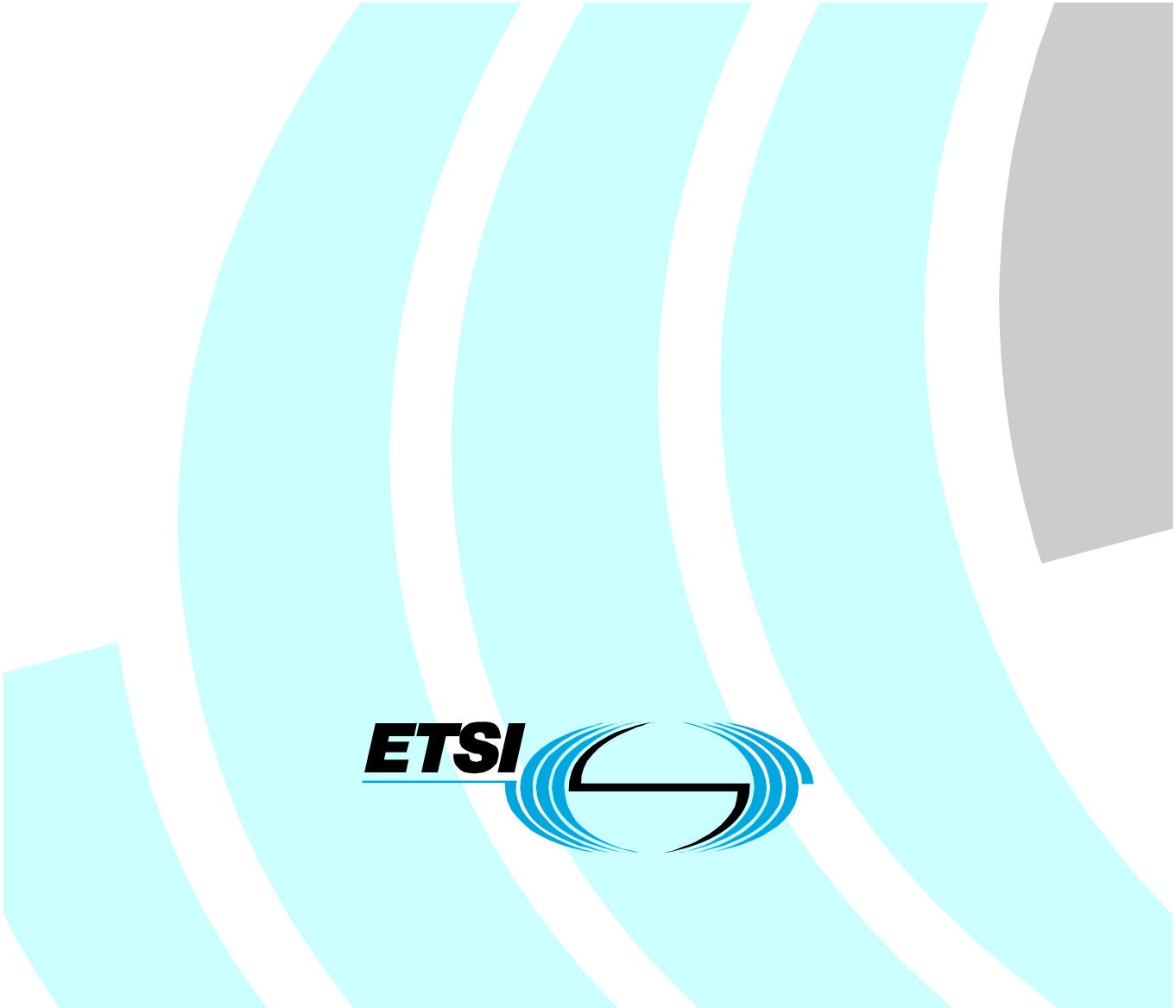
# ETSI TR 101 561 V1.1.1 (2011-06)

*Technical Report*

# IMS Network Testing (INT); Enhancement of Automated Interoperability Testing Framework in IMS core networks: Test adapter And codec design suited for TTCN-3 interoperability testing



Reference
DTR/INT-00035

Keywords
adaption, codec, IMS, interoperability, IPv6,
network

***ETSI***

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

# Contents

# Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://webapp.etsi.org/IPR/home.asp).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

# Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee IMS Network Testing (INT).

# 1 Scope

The purpose of the present document is to present and describe issues and design choices made while developing a generic test adapter and codec suited for TTCN-3 interoperability testing within both STF370 and STF407.

For further information, the reader is referred to EG 202 810 [i.3] for global view of methodology and framework for automated interoperability testing and TR 102 788 [i.4] for an overall view of the IMS interoperability test architecture which has served as the main source for design requirements.

The present document has been written with the assumption that the reader is well versed in C++ and TTCN-3 (ES 201 873-1 [i.2]) programming. Also good knowledge of the operation of ES 201 873-5 [i.7] and ES 201 873-6 [i.6] standards is assumed.

# 2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

## 2.1 Normative references

The following referenced documents are necessary for the application of the present document.

Not applicable.

## 2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] Official BOOST framework reference.

NOTE: Available at: http://www.boost.org/doc/libs/1_39_0/libs/libraries.htm.

[i.2] ETSI ES 201 873-1 (V3.4.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language".

[i.3] ETSI EG 202 810: "Methods for Testing and Specification (MTS); Automated Interoperability Testing; Methodology and Framework ".

[i.4] ETSI TR 102 788: "Methods for Testing and Specification (MTS); Automated Interoperability Testing; Specific Architectures".

[i.5] Official t3devket framework reference.

NOTE: Available at: http://www.irisa.fr/tipi/wiki/doku.php/t3devkit.

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

[i.7] ETSI ES 201 873-5: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI)".

[i.8] IETF RFC 4475: "Session Initiation Protocol (SIP) Torture Test Messages".

[i.9] IETF RFC 3261: "Session Initiation Protocol (SIP )".

[i.10] IETF RFC 3262: "Reliability of Provisional Responses in the Session Initiation Protocol (SIP)".

[i.11] IETF RFC 3265: "Session Initiation Protocol (SIP)-Specific Event Notification".

[i.12] IETF RFC 3313: "Private Session Initiation Protocol (SIP) Extensions for Media Authorization".

[i.13] IETF RFC 3323: "A Privacy Mechanism for the Session Initiation Protocol (SIP)".

[i.14] IETF RFC 3325: "Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks".

[i.15] IETF RFC 3326: "The Reason Header Field for the Session Initiation Protocol (SIP)".

[i.16] IETF RFC 3327: "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts".

[i.17] IETF RFC 3329: "Security Mechanism Agreement for the Session Initiation Protocol (SIP)".

[i.18] IETF RFC 3455: "Private Header (P-Header) Extensions to the Session Initiation Protocol (SIP) for the 3rd-Generation Partnership Project (3GPP)".

[i.19] IETF RFC 3515: "The Session Initiation Protocol (SIP) Refer Method".

[i.20] IETF RFC 3608: "Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration".

[i.21] IETF RFC 3841: "Caller Preferences for the Session Initiation Protocol (SIP)".

[i.22] IETF RFC 3891: "The Session Initiation Protocol (SIP) "Replaces" Header".

[i.23] IETF RFC 3892: "The Session Initiation Protocol (SIP) Referred-By Mechanism".

[i.24] IETF RFC 4028: "Session Timers in the Session Initiation Protocol (SIP)".

[i.25] IETF RFC 4244: "An Extension to the Session Initiation Protocol (SIP) for Request History Information".

[i.26] IETF RFC 5009: "Private Header (P-Header) Extension to the Session Initiation Protocol (SIP) for Authorization of Early Media".

[i.27] IETF RFC 2616: "Hypertext Transfer Protocol -- HTTP/1.1".

[i.28] IETF RFC 2617 HTTP: "Authentication: Basic and Digest Access Authentication".

# 3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

| | |
|---|---|
| DNS | Domain Name System (protocol) |
| EUT | Equipment Under Test |
| GUI | Graphical User Interface |
| HTTP | Hypertext Transfer Protocol |
| IP | Internet Protocol |
| IMS | IP Multimedia Subsystem |
| ISDN | Integrated Service Digital Network |
| OS | Operating System |
| PCAP | Packet Capture |
| SIP | Session Initiation Protocol |
| SSH | Secure Shell |
| TCI | TTCN-3 Control Interface |

| | |
|---|---|
| TCP | Transmission Control Protocol |
| TRI | TTCN-3 Runtime Interface |
| TTCN-3 | Testing and Test Control Notation 3 |
| TE | TTCN-3 Executable |

NOTE: As defined in ES 201 873-5 [i.7] and ES 201 873-6 [i.6].

| | |
|---|---|
| UE | IMS User Equipment |
| HTTPS | Hypertext Transfer Protocol Secure |
| IPv4 | Internet Protocol version 4 |
| LR | Leakage Ratio |
| NNI | Network to Network Interface |
| POSIX | Portable Operating System Interface |
| SDP | Service Delivery Platform |
| SUT | System Under Test |
| TSI | Time Slot Interchange |
| UNIX | Universal Network Information eXchange |
| XML | Extensible Markup Language |

# 4 Overview

The main purpose of the TTCN-3 interoperability test adapter is to implement the real test system interface as described in ES 201 873-1 [i.2], ES 201 873-6 [i.6] and ES 201 873-5 [i.7] of the TTCN-3 IMS interoperability test system described in TR 102 788 [i.4], i.e. the handling and transport of TTCN-3 messages send or received via abstract TTCN-3 test system interface ports and different EUTs. Nevertheless it has been attempted to keep the adapter design completely independent on IMS specific testing.

The adapter should be designed primarily to allow the use of the interoperability test system in the context of an interoperability event. It should however also be possible to use it in the context of a test bed. Note that these usage scenarios come along with a different set of constraints. For example, in the context of an interoperability event it is not until the day of the event that you know which products and version of these products will participate whereas in a test bed that information is better known. Therefore automation of equipment operation is much easier to realize in a test bed than for an interoperability event. In addition interoperability events a restricted to a limited amount of times (usually a week) whereas time is not so much a constraint in the scenario of a test bed - giving much room for adaptation updates.

The adapter conceptually splits into three parts:

1) an upper test adapter which provides an implementation of vendor specific operation of different EUTs involved in an interoperability test;

2) a lower test adapter which captures traffic and isolates requested payloads based on filter criteria specified by an interoperability test suite and forwards them as raw data to the test suite; and

3) a TTCN-3 platform adapter implementing timers.

# 5 Design proposals for receiving of traffic capture

This clause evaluates different design proposals for integrating the TTCN-3 interoperability codec and test adapter with physical traffic capture.

## 5.1 Design proposal for the codec

The following design considerations have been captured based on an analysis of the various protocols that need to be supported by the codec and the type definitions provided by TTCN-3 libraries.

### 5.1.1 Principles

The codec should be extensible at low cost. SIP is still evolving and new headers and messages may be added to the protocol. Therefore it is essential to be able to add new types in the codec without having to fully redesign it.

The codec should be as generic as possible and the chosen solution should not be operating system or hardware specific. Ideally, the codec would be useable on both Windows and UNIX-like operating systems.

In order to ease test system debugging and logs analysis, the codec should provide a mean to display raw messages, i.e. protocol messages exactly as they are received. This goal will be achieved by using the "payload" field in message templates.

Concerning the different kinds of SIP payloads that will be handled by the codec, it is important to note that the only supported payload will be SDP. All other payload types (among them, XML) will be processed as character strings.

UTF-8 is the standard encoding type for SIP messages. However, SIP test suite has been developed using `charstring` instead of `universal charstring`. The codec should somehow deal with this issue. One possible solution would be to replace Unicode characters by displayable characters and raise a warning.

### 5.1.2 Encoder

The TTCN-3 types in the SIP library provide an abstract representation of the SIP messages. Only the semantic information is represented, and all syntactic elements which do not carry any useful information are not represented. Among them:

- linear white space and new lines between the header fields;
- delimiters, e.g. `. : ; , ? & = @ a` as well as `"` for quoted string.

It is the responsibility of the encoder to add these structural characters when necessary, in order to build valid SIP messages. On the contrary, these elements will have to be removed by the decoder before assigning TTCN-3 values.

### 5.1.3 Decoder

Extra white space is semantically meaningless and can appear in many places inside a SIP message. The codec will have to be able to deal correctly with these white spaces and ignore them when necessary.

Additionally, SIP specification allows some header fields to appear multiple times, but with different encoding, i.e. one header field per header, all header fields in same header, or any variation of the prior. The codec will have to carefully process these headers and regroup them in a single TTCN-3 list value when decoding SIP messages, as the TTCN-3 typing is designed to handle multiple header fields in a list of header field values. The sequence in which these header fields appear must absolutely be preserved as it has a semantic importance.

SIP specification also gives the opportunity to use short names for some header identifiers. The codec will have to deal with these synonyms, which is not really complex but needs to be taken into account especially when dealing with the previous point.

Current TTCN-3 type system makes use of "record of" and "set of" structure. Unfortunately there are two ways of representing an empty optional "set of" or "record of": either the "record of" is omitted or it is present but contains zero elements. These two possibilities for representing the same data can have a real impact on matching process and therefore requires a design decision in decoding.

The decoder will support the test case writer and ensure or fill in correct values for the SIP Content-Length header.

Last but not least, it is necessary to find a way to handle an unsuccessful decoding of messages. The codec should produce a short report for each failed attempt showing the raw message as well as a short description of the error and its location in the message.

## 5.1.4 Discussion

### 5.1.4.1 Rejected solutions

#### 5.1.4.1.1 External SIP application

One possible solution could be to use an external SIP application to decode and encode SIP messages. For this approach it is necessary to develop a non-standardized interface with this application. In addition, it would make the codec and thus the complete test environment dependent on that external application. This situation is not desired especially when considering the questions of maintenance and extensibility. In addition, the codec would here be vulnerable to possible errors present in the external application, which would completely reduce the confidence one can have towards this test suite.

Another issue raised by this solution is the difficulty to debug the encoding/decoding of the messages. The codec provided by the existing implementations of the protocol (server, client, etc.) are designed to accept the message or drop it if it is not conformant, but they are not generally designed to help in debugging the messages received by another implementation (like giving a detailed account of the location of the error and its description).

#### 5.1.4.1.2 Wireshark

A second alternative would be to use a network protocol analyzer such as Wireshark and use its dissector library to decode SIP messages. This approach brings sensibly the same problems as the previous one:

- Possible bugs in external tool.
- Extensibility and maintenance dependent on external application.

In addition, wireshark can only decode the messages. An encoder has to be implemented separately which could be considered to render the overall solution more complex to maintain.

#### 5.1.4.1.3 Yacc parser

Another approach would be to use a parser generator such as Yacc. A parser typically receives a sequence of tokens from a lexer and tries to derive the unique sequence of grammar rules that can produce that token sequence. It would be then possible to create a parser in order to decode SIP messages.

However, parsers usually require the protocol grammar to be in a particular form (typically, Yacc requires LR(1) grammar). Unfortunately SIP grammar is not of type LR(1). In addition, even if it is LR(1), there is absolutely no guaranty that future adjunctions will not modify this property of the grammar.

In conclusion, this solution cannot be selected, as it would require rewriting SIP grammar to make it compatible with our needs, and to repeat this operation every time the protocol is extended, which does not match requirements concerning maintainability of the codec.

### 5.1.4.2 Selected solution: T3DEVKIT

T3DevKit [i.5] is a project developed by IRISA which aims to provide tools and libraries to help developing TTCN-3 codecs and adapters. It is composed of two main elements: T3DevLib, a library of C++ classes that handle generic codec & Adapter functionalities, and T3CDGen, a tool that can generate a codec implementation template from TTCN-3 source files. Protocol specific adaptations and codec behaviour are then customised by adding pieces of code called "codets". The global integration of T3DevKit in a TTCN-3 project is shown in figure 1.

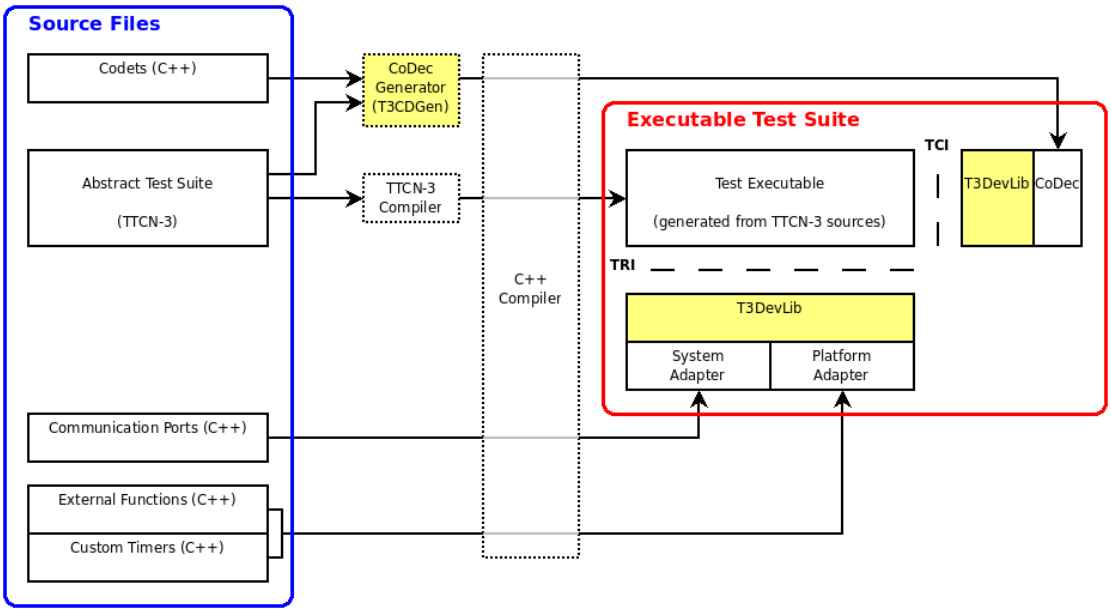



**Figure 1: T3DevKit integration (©IRISA http://t3devkit.gforge.inria.fr/doc/userref/)**

#### 5.1.4.2.1 Encoder

The encoder mainly appends the syntactic element that do not appear explicitly in the TTCN-3 message (but that can be derived from the structure of the message). This is done by writing the delimiter in question in the buffer before and/or after the adequate field.

Encoding of the "Content-Length" field of the SIP messages requires more processing. The length of the message body is not known in advance, the value provided by the TTCN-3 code may be incorrect or not be available. Instead it is possible to fill it with blank spaces and remembering the position of this value in the buffer. Then in the PreEncode() and PostEncode() codets of the message body, the current position in the buffer is saved and then used to compute and encode the value of the Content-Length field.

#### 5.1.4.2.2 Decoder

Implementing the decoder mostly consists of:

- validating the format of the message;
- skipping the syntactic elements that are not represented in the TTCN-3 structure;
- make predictions for the codec about the length of each fields (so that when the generated codec decodes a variable-length field (typically a charstring) it does not reads all the bytes until the end of the buffer, but stops at the real end of the buffer.

Since the format of the SIP and SDP message is mostly described by a BNF representation, it was decided to validate the input message and identify each field using regular expressions.

## 5.2 Design proposals for receiving of traffic capture

This clause evaluates different design proposals for integrating the IMS interoperability test adapter with physical traffic capture.

### 5.2.1 All traffic on single (mirrored) switch port

This design proposal assumes that all traffic produced in interoperability testing is mirrored by a switch on a single port. Multiple cascaded switches may be used to combine multiple monitoring ports into one physical port which is then connected to the computer running the test adapter via its network card.

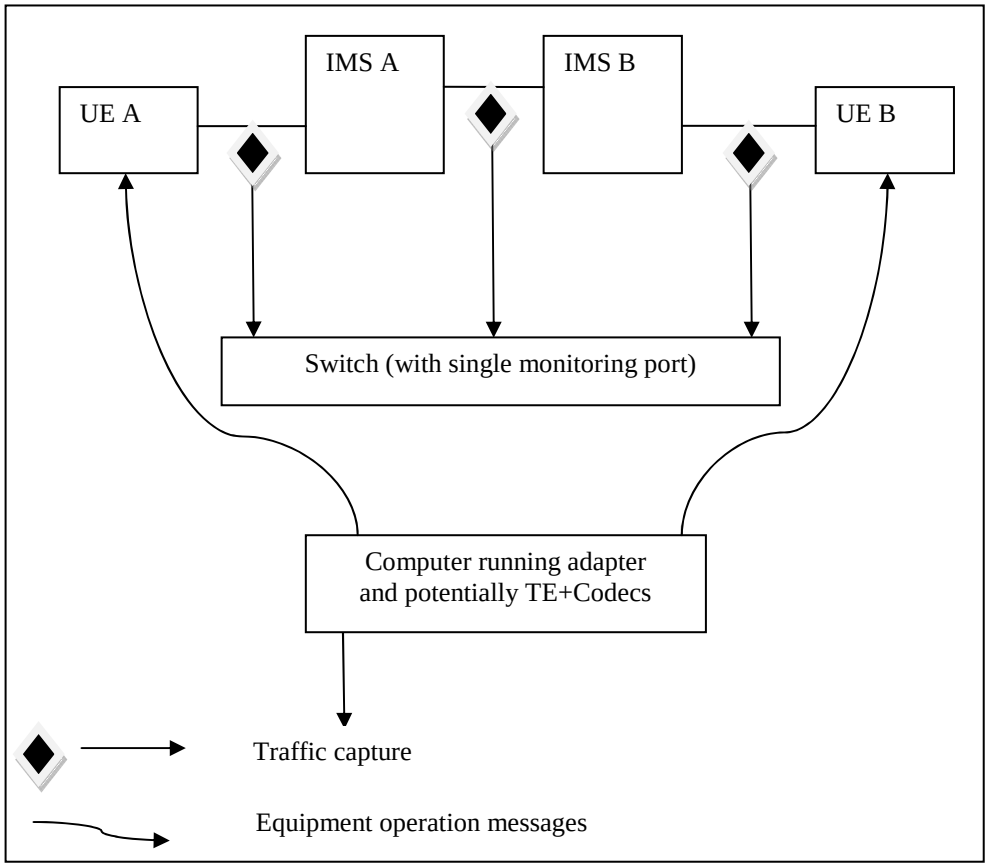



**Figure 2: Design with a switch providing single (mirrored) switch port for all traffic in context of IMS interoperability testing**

### 5.2.2 Traffic on multiple switch ports (no mirroring)

This solution requires that the computer running the test adapter has multiple network interface boards. However, this solution does not require the implementation of switches that support mirroring.

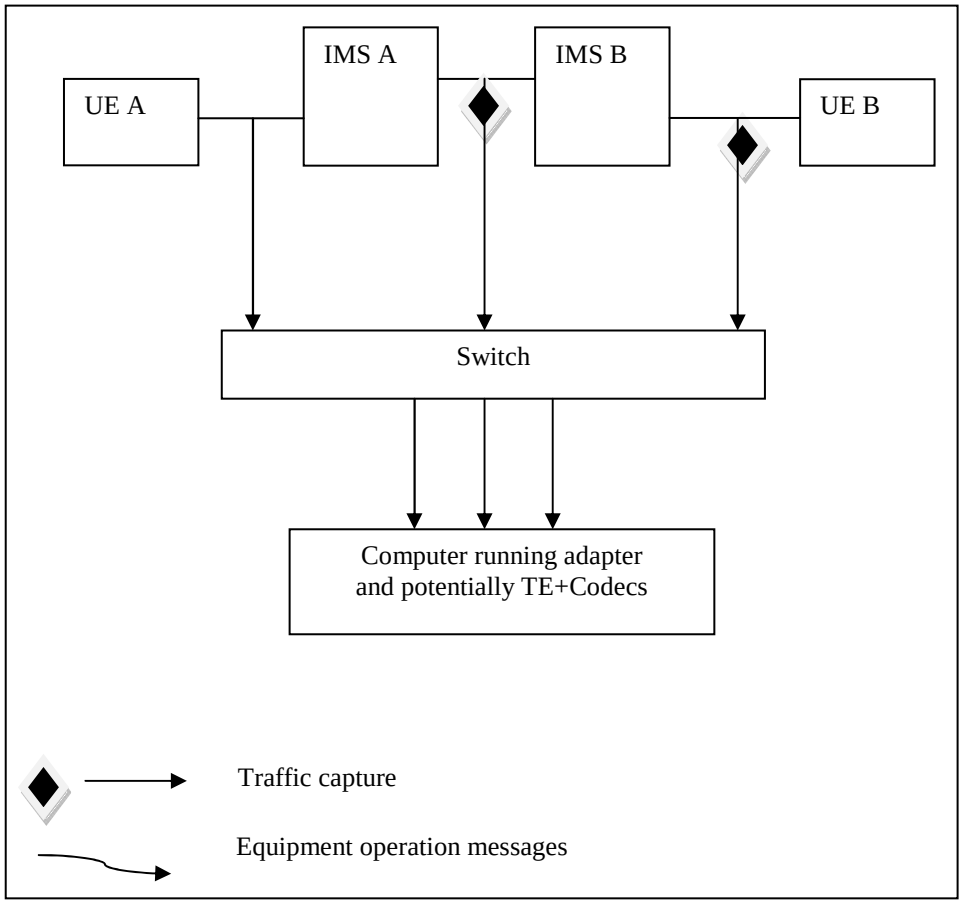



**Figure 3: Design with a switch providing multiple ports for traffic in context of IMS interoperability testing**

### 5.2.3 Discussion

The first proposal has the main advantage that it is easy to deploy. We can use essentially any laptop with, e.g. an Ethernet card. The main problem is that switches with mirroring capability may be hard to find.

The second solution will eventually impose some limitations on the number of network cards, i.e. interfaces that could monitored simultaneously. Note that a standard laptop usually only provides a single Ethernet card.

It was decided to use the first proposal. This however implies that the component(s), e.g. switches, used during testing for physically capturing traffic must provide the capability to monitor and mirror several network interfaces.

# 6 Software design

This clause introduces the requirements taken into account for the software design of both test adapter and codec.

## 6.1 Codec design

This clause introduces the software design solutions proposed to implement the Codec.

### 6.1.1 Regular expression

A utility class named "Regex" was developed. It is implemented based on the portable C++ BOOST [i.1] regex library and implements perl regular expressions. Moreover it is integrated partially with the T3DevKit API by providing functions to match the regex on a part of the encoding buffer, to report easily the length of a field, to move the position in the buffer at the beginning or the end of a matched group in the regex and to report detailed error messages in case of mismatch.

### 6.1.2 Template generation

TTCN-3 message templates are generated automatically by a script based on a TTCN-3 type module. The main difficulty is to generate a reasonable amount of templates: it is very easy to generate millions of template due to combinatory explosion. To avoid this problem, it is important to carefully define the rules that will be used for template generation. Ideally, these rules lead to the generation of a minimum number of templates by keeping only the most interesting ones, from a testing point of view.

#### 6.1.2.1 Generation rules

The following generic rules have been selected for this project: Port primitives are the starting point of the process. Complete templates are generated for each primitive, using a recursive strategy and by applying the following rules:

- While processing a `record`, generate two templates, one containing values for mandatory fields only, and one containing values for mandatory fields and optional fields. Additionally, generate a parameterised number of templates containing values for mandatory fields and randomly selected optional fields. By doing this, extreme case and some intermediate case templates are generated.

NOTE: With this method, each field is tested at least once.

- A set is processed like a record.
- While processing a `union`, generate one template per alternative.
- While processing a `record of`, generate different templates with different list sizes. The number of templates and the sizes of the list should be parameters.
- A `set of` is processed like a `record of`.
- While processing an `integer` field, generate different templates with different values (minimum value, maximum value, and optionally random values).
- While processing a string field (`octetstring`, `bitstring`, `charstring` ...), generate different templates with different values (short string, long string, and optionally random length strings).
- While processing a `boolean` field, generate two templates (`true` and `false`).
- While processing `enumerated` field, generate all possible templates, based value constraints.
- If a complex type as already been derived once in any of the already generated templates, it should not be derived again (it is very likely that the same Codec function will be used), and should trigger the generation of only one template.

In addition to these rules, the template generator should offer the possibility to use specific values for a particular field. This way, it is possible to test more intensively some important fields.

#### 6.1.2.2 Template generator design

The generator developed within STF370 is a Perl script which receives type and port definitions as input and produces three files: one containing all the generated message templates and sub-templates, a second file containing the generic test cases and a test control file.

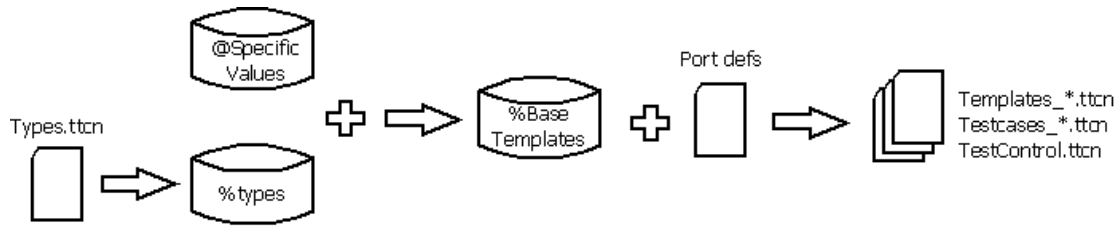



**Figure 4**

Before using the generator, some values need to be adjusted. First of all, the generic testcase pattern can be modified. It is stored in the global variable $testcaseTemplate. Default values are stored in the global hashmap %defaultValues and initialized through the function initializeDefaultValues(). This hashmap stores two kinds of values:

- default values used for basic types (`charstring`, `integer`, …);
- values for specific fields in a complex type (typically `record`). In this case, the syntax for hash entries is `Type<space>Fieldname`.

Table 1 shows some example entries:

**Table 1: Examples of default and specific values**

| Hash entry | Hash value | Comment |
|---|---|---|
| `Charstring` | `['"a"', '"abcde"', '"abcdefghij"']` | Three possible values for `charstrings` |
| `Integer` | `[1, 2, 3]` | Three possible values for `integers` |
| `Boolean` | `["true", "false"]` | Boolean can be `true or false` |
| `StatusLine sipVersion` | `['"SIP/2.0"']` | Field `sipVersion` in record `StatusLine` will be filled with value `"SIP/2.0"` |
| `DeltaSec` | `['"1"', '"123456"', '"123456789"']` | Three possible values for `DeltaSec` fields. In this case `Deltasec` is an alias of `charstring`. This entry will override default `charstring` values. |
| `SDP_contact addr_or_phone` | `['"test_email@etsi.org"', '"+33 4 - 9294 4200"']` | Field `addr_or_phone` in `SDP_contact` record will be filled with these two specific values. |

The generator is typically used as follow. The output files are created in the current directory.

```
$ cat compiledTypes.txt | SipCodecTestGenerator.pl
```

Among its known limitations, this Perl script does not feature a complete TTCN-3 parser; only types and port definitions can be analyzed and any other TTCN-3 instruction causes a failure. It is also important to notice that so far, the Perl script does not support comments either.

## 6.2 Test adapter design

This clause introduces the software design solutions proposed to implement the test adapter.

### 6.2.1 Requirements

The test adapter software will address the requirements listed in table 2 which are not normative requirements. They only indicate the functionalities within the objective of the test adapter software.

**Table 2: Test adapter software requirements**

| Requirements | Description |
|---|---|
| On/Off line mode | The adapter shall support test execution with traffic capture in real time (live) as well as from recorded traffic capture (offline). |
| Dynamical adapter configuration | The adapter shall be configurable from TTCN-3 code as much as possible. |
| Merge in off line mode | In off line mode, it should be possible to handle several recorded traffic capture and merge them into a single file, respecting time stamps. |
| Time stamp Offset | The adapter shall be able to start processing recorded traffic from a specified time stamp instead of the beginning of the file. |
| Filtering for specific protocols | The adapter should be flexible and allow isolation of protocols from traffic capture requested by TTCN-3 components. |
| Support of Ethernet capture based on PCAP format | The adapter shall at least support Ethernet traffic capture based on PCAP. |
| Support for SIP and DNS filtering | The adapter shall at least be able to filter SIP and DNS messages based on IPv4 address and port information. |
| Support for IPv4 based filtering | The adapter shall at least be able to filter traffic capture based on IPv4 address and port information for at least two end points. Each endpoint IP information may include multiple IP addresses and ports. |
| Support of IP and TCP fragmentation | The adapter shall be able to handle IP and TCP fragmentation. |
| Logging of messages sent to TE | The adapter shall provide a means to display messages exchanged with the TE as well as a time stamp. |
| OS independence | The adapter should not be operating system or hardware dependent. Ideally, the adapter would be useable under both Windows and UNIX-like operating systems. |
| Use of TRI C mapping | The adapter shall use the TRI C mapping in order to be reusable with the largest number of TTCN-3 tools. |
| Timers | The adapter shall implement TTCN-3 timer handling in real time. |
| Support for equipment operation GUI | The adapter shall support the conversion of equipment operation messages into interactions via a GUI for equipment operators. More specifically it should provide one GUI window per TTCN-3 equipment user component. Interactions shall be configurable as well as the computer where each GUI instances are supposed to run. |
| Extensibility | The adapter shall allow easy extension for filters for non SIP or DNS protocols. Similarly it shall support the integration of non-PCAP traffic capture tools. Also it should allow integration of vendor specific protocols for equipment operation. |

At the point of writing there was no requirement for external function implementations.

### 6.2.2 Software design

In order to fulfil the above requirements it was decided to design test adapter software architecture with the following main components:

- A Lower Test Adapter which provides traffic capture processing functionality which includes handling of IPv4 and TCP fragmentation, isolation of protocol messages, etc.
- A PCAP capture process which interacts with the Lower Test Adapter.
- A Upper Test Adapter which converts TTCN-3 equipment operation messages into EUT operator instructions and can process their feedback based on a terminal window.
- TRI implementation.
- Codecs for decoding of configuration message request and encoding responses in the adapter.
- Timer handling implementation.

The design decision was made to use the IRISA t3devkit framework (see reference [i.5]) to allow the implementation of the adapter in C++ in order to profit from object oriented programming benefits. The t3devkit maps the TRI C interface into a C++ interface.

Note that this C++ interface is not compliant to the standardized TRI C++ mapping.

Real-time timer handling is included as part of the t3devkit implementation.

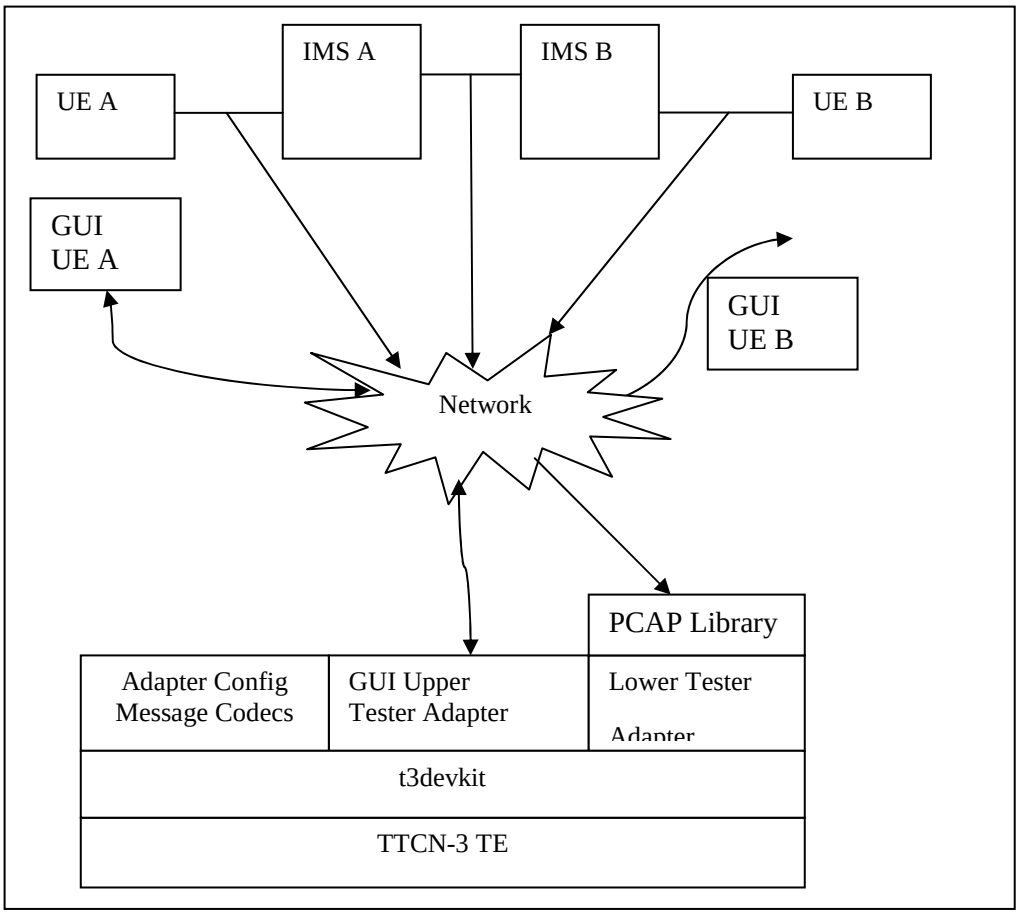



**Figure 5: Test Adapter Software Architecture**

Note that the network component includes any network related equipment including switches, etc. Also, the Platform adapter is not shown in figure 5.

Note that the GUI Upper Tester Adapter component could also be replaced with code that directly maps equipment operation messages to vendor specific primitives.

#### 6.2.2.1 Test adapter interfaces

The proposed test adapter has three types of interfaces:

- One with the TTCN-3 TE which implements part of the TRI interface.
- One with the traffic capture, i.e. the PCAP capture library.
- One with EUT operator, i.e. the GUI which interacts with the equipment operator.

#### 6.2.2.2 Test adapter configuration

In order to fulfil the dynamic adapter configuration requirements, the test adapter supports the following primitives:

- A general configuration primitive which is used to communicate parameters which are not specific to a specific monitored EUT interface. These parameters include an indication for live vs. offline capture mode, record captured traffic into file (only in live mode), Ethernet network interface card information, IP address of the PCAP capture process, (list of) traffic capture files (only in offline mode), timestamp offset (only in offline mode), etc.
- Start and stop capture primitives.
- Primitive to specify interface specific parameters for monitoring purposes. Using these parameters, the Lower Tester Adapter isolates protocol messages and dispatches the encoded messages properly to the TTCN-3 component that has requested the filter.

#### 6.2.2.3 Adapter configuration message encoding

When TTCN-3 TE and test adapter exchange messages via the TRI, the TRI requires that these messages have to be encoded. The following encoding rules are used to encode adapter configuration messages:

- The message type is encoded in the first octet except for capture messages which are pure raw data (see table 3 for details).
- Each information element of a message is encoded with < length><value> where < length> is always encoded on 2 octets.
- Text string values are kept as they are.
- Integer values are always encoded on 8 octets, using network byte order.
- Enumerated values are encoded in their integer representation using 1 octet.
- Lists of information elements are encoded using <number of parameters><{<length><value>}+ >, where <number of parameters> shall use 2 octets and <length> <value> are encoded as described above.
- Sequences of information elements simply encoded as a concatenation of encoded information elements; note that the position of list information elements is assumed to be known, i.e. hardcoded.
- Union elements are encoded using <alternative index> in a single octet; the index starts at zero and the alternative definition order is assumed to be the same as in the TTCN-3 types.
- Omitted information elements or values of length zero simply are encoded using <length> (or a <number of parameters> for the lists of information elements) set to '0000'H.

**Table 3: Adapter configuration and equipment operation messages and their message type encoding**

| Message type | Octet Value Encoding |
|---|---|
| GeneralConfigurationReq | 0x00 |
| GeneralConfigurationRsp | 0x01 |
| SetFilterReq | 0x02 |
| SetFilterRsp | 0x03 |
| StartTrafficCaptureReq | 0x04 |
| StartTrafficCaptureRsp | 0x05 |
| StopTrafficCaptureReq | 0x06 |
| StopTrafficCaptureRsp | 0x07 |
| EquipmentOperationReq | 0x08 |
| EquipmentOperationRsp | 0x08 |

- Figure 6 shows an example of an encoded GeneralConfigurationReq message in hexadecimal string format.
  - In the sequence '0000093132372E302E302E31'H:
    - '00'H is the message type (see Table 3), the message GeneralConfigurationReq.
    - '0009'H is the length of the captureProcessIpAddress information element value "127.0.0.1".
    - '3132372E302E302E31'H is the text string "127.0.0.1" itself.

```
0000093132372E302E302E31000000000000157D000001003A72706361703A2F2F5C4465766963655C4E50465F7B
46333031333632462D374444422D343237432D423436352D383243373845434433447347D0000020006557365722041000931302E372E362E31370000000000000157E000655736572204200093130 2E372E362E31370000000000000157E
```

**Figure 6: An example of encoded GeneralConfigurationReq message in hexadecimal**

- Figure 7 shows an example of an encoded equipment operation message in hexadecimal format.
  - In this sequence '000F55455F52454749535452415449 4F4E'H:
    - '08'H is the message type (see Table 3), the message EquipmentOperationReq
    - '000F'H is the length of the cmd information element value "UE_REGISTRATION"
    - 55455F52454749535452415449 4F4E is the text string "UE_REGISTRATION" itself
    - '0003'H is the number of parameters in the params list information element
    - The following octets are the parameters information element values "userSIP", "3344123432" and "123456" each preceded by their 2 octet length

```
08000F55455F52454749535452415449 4F4E0000000300000007757365725349500000000A3333343431323334333200000006313233343536
```

**Figure 7: An example of encoded UE_REGISTRATION equipment operation message**

### 6.2.2.4 Traffic capture

The adapter has been designed to allow also integration of non-PCAP based trace processing. In this case the PCAP library would be replaced by another traffic capture tool library. The integration of such other tools would however require a vendor specific implementation of the communication with the Lower Test Adapter and is beyond the scope of the present document.

### 6.2.2.5 Merge of multiple trace files

The test adapter assumes all traffic capture files to be merged are located in the same directory. In the case of the PCAP library implementation the merged file is generated during the execution of a test case in that same directory.

Note that this feature is only available in offline mode.

Note that should the test execution be repeated it is advised to use the merged file instead of the file list to reduce test execution time significantly!

#### 6.2.2.6 Possibilities for handling of equipment operation messages

There are different ways to handle the TTCN-3 equipment operation messages:

- Implement a human friendly GUI to guide the user to operate manually the EUT as specified in the command. Such an approach is required when the interoperability test is executed with real end user equipment, e.g. a mobile terminal with a IMS UE.
- Implement a software component which composes, sends, and receives encoded SIP messages and therefore acts like a replacement of the UE. This approach is only possible when the equipment is not a EUT as in the case of the UE in the IMS NNI interoperability testing.
- Implement software that is directly integrated with the equipment. This software is product specific in case the operation of the EUT is not standardized (which is usually the case). This allows *automatic* control of equipment and removes the need for a human equipment operator. This solution also requires a part integrated with the test system. The communication between the integrated software and the TTCN-3 test system can be achieved either via telnet, ssh, HTTP/HTTPS or TCP/IP connection and a port managed by XInetd.

### 6.2.3 Discussion

The human friendly GUI solution was selected in the test adapter software design. Due to the variety of different interfaces for the operation of EUTs and their predominately proprietary nature, it is very hard or even impossible to develop only one automatic mechanism to operate EUT or other equipment. In addition, the adapter was designed for IMS NNI interoperability testing where the test system will be used in the context of an interoperability event.

# 7 TTCN-3 message type definitions

This clause provides an overview of TTCN-3 port and message types used to communicate with the Adapter via the abstract TTCN-3 TSI. Note that the adapter is not directly dependent on the IMS interoperability test suite but rather the TTCN-3 interoperability library called LibIot.

## 7.1 TTCN-3 port description

Three types of TSI ports are defined in LibIot:

- Adapter port is used to receive and send general configuration messages, setting of test component specific filter criteria, and for controlling traffic capture.
- Monitor data port is used by TTCN-3 components to receive protocol messages from the lower test adapter.
- Equipment operation port is used by TTCN-3 components to send and receive equipment operation messages, e.g. to operate an IMS UE.

## 7.2 TTCN-3 messages description

Figure 8 shows the TTCN-3 message type for general adapter configuration. These types are defined in the LibIot_TypesAndValues TTCN-3 module.

```
  type record of charstring PhysicalInterfaceList;
  type record LiveCapture {
  PhysicalInterfaceList physicalInterfaces,
     RecordMode            recordMode
  }
  type enumerated RecordMode {
     e_norecord,
     e_record
  }
  type record of charstring FileList;
  type record MergeFileList {
  FileList    mergeFileList,
     charstring  mergeFilePath
  }
  type record CaptureSource {
     charstring singleFile, // e.g. PCAP file
     MergeFileList mergeFileList
  }
  type record OfflineCapture {
   UInt32         offset,
   CaptureSource  captureSource
  }
  type UInt16 PortNumber;
  type union CaptureMode {
  LiveCapture liveCpature,
  OfflineCapture offlineCapture
  }
  type record EutInterfaceInfo {
     charstring eut,
     IpAddress  ipAddress,
     PortNumber portNumber
}
Type record of EutInterfaceInfo EutInterfaceInfoList;
type record GeneralConfigurationReq {
     charstring  captureProcessIpAddress,
     PortNumber  captureProcessPort,
     CaptureMode captureMode,
     |EutInterfaceInfoList eutInfoList optional
}
type record Status {
  FncRetCode code, charstring reason optional
}
type record GeneralConfigurationRsp {
     Status status
}
```

**Figure 8: TTCN-3 types that define GeneralConfigurationReq/Rsp**

### 7.2.1 Record mode

This parameter is used to control the recording of traffic capture in a file in live capture mode. The name and location of the output file is selected based on the naming convention described in clause 6.2.2.3.

### 7.2.2 Merging of captured traffic files

If the mergeFileList is selected in the CaptureMode union and this field contains a list of the traffic capture file names in the mergeFileList field and a mergeFilePath field that contains a directory name where the merged file is to be stored. The traffic capture process will then perform, e.g. a PCAP merge operation and store the result at the specified location.

Note that the traffic capture component will provide the name of the merged file.

### 7.2.3 List of physical interfaces

In the live capture mode the physicalInterfaces field allows to specify a list of physical interfaces, Ethernet card information.

### 7.2.4 Example of setting general adapter configuration message values in TTCN-3

```
  group adapterGeneralConfiguration {
    /**
    *
    * @desc Maximum time limit used by trigger component for waiting for EUT response after command
has been sent
    */
    modulepar float PX_EUT_TRIGGER_RESPONSE := 5.0;
    /**
     * @desc
     *     in which mode the ATS should be executed. In realtime mode
     *     the ATS get messages form the EUT in realtime. IN offline mode the
     *     ATS gets messages form a trace file.
     */
    modulepar CaptureMode PX_IOT_EXECUTION_MODE := e_live/*e_offline*/;
    /**
     * @desc
     *     In case of offline mode, it defines the Pcap file to play.
     */
    modulepar charstring PX_IOT_EXECUTION_FILE := "TD_IMS_0001_19.pcap";
    /**
     * @desc
     *     Defines if the record traffic capture mode must be activated or not.
     */
    modulepar RecordMode PX_IOT_RECORD_MODE := e_norecord;
    /**
     * @desc
     *     Defines list of the files to merge.
     */
    modulepar charstring PX_IOT_FILE_MERGE_LIST :=
"TD_IMS_0001_11.pcap;TD_IMS_0001_19.pcap;TD_IMS_0001_20.pcap";
    /**
     * @desc
     *     Defines the location of the files to merge.
     */
    modulepar charstring PX_IOT_FILE_MERGE_PATH := "/tmp";
    /**
     * @desc
     *     Defines the time stamp offset to start playing record traffic capture file.
     */
    modulepar integer PX_IOT_TIMESTAMP_OFFSET := 1966;
    /**
     * @desc
     *     List of the network interfaces to monitor.
     *     Use ';' to separate the interfaces
     */
    modulepar charstring PX_IOT_IFACES := "rpcap://\Device\NPF_{F301362F-7DDB-427C-B465-
82C78ECD3D74};rpcap://\Device\NPF_{DummyIface}";
  } // group adapterGlobalConfiguration
```

**Figure 9: Example TTCN-3 parameter setting for general configuration message**

## 7.3 Setting of filter criteria

These messages are used by TTCN-3 test components to request their specific filtering of traffic capture. The adapter combines all filter criteria automatically when it receives a StartCaptureRequest.

```
    type UInt16 PortNumber;
type record of PortNumber PortNumberList;
type record IpInterfaceInfo {
   charstring domainName optional,
   IpAddress IpAddress,
   PortNumberList portNumbers
}
type record of IpInterfaceInfo IpInterfaceInfoList;
type union InterfaceInfo {
   IpInterfaceInfoList IpInterfaceInfo
}
type record (2..infinity) of InterfaceInfo InterfaceInfoList;
   type enumerated ProtocolFilter {
     e_sip,
     e_dns
   }
   type record SetFilterReq {
     ProtocolFilter    protocol,
     InterfaceInfoList interfaceInfos
   }
type record Status {
   FncRetCode code, charstring reason optional
   }
   type record SetFilterRsp
   {
     Status status
   }
```

**Figure 10: TTCN-3 types for Start/StopTrafficCaptureReq/Rsp**

## 7.4 Starting and stopping of traffic capture

These messages are used by the adapter process to send its filter to the TrafficCapture process and to command it to start or stop capturing traffic.

```
 type record StartTrafficCaptureReq   {   }
 type record Status {
FncRetCode code, charstring reason optional
 }
 type record StartTrafficCaptureRsp    {
   StatusCode result
 }
 type record StopTrafficCaptureReq   {   }
 type record StopTrafficCaptureRsp    {
   StatusCode result
 }
```

**Figure 11: TTCN-3 types for Start/StopTrafficCaptureReq/Rsp**

## 7.5 Equipment operation messages

These messages are used to request the operation of a EUT or other equipment during a test.

```
type record of charstring ParameterList;
type charstring EquipmentCommand;
type record EquipmentOperationReq {
   EquipmentCommand cmd, ParameterList params optional
}
type record Status {
   FncRetCode code, charstring reason optional
}

type record EquipmentOperationRsp {
   Status status
}
```

**Figure 12: TTCN-3 types for Start/StopTrafficCaptureReq/Rsp**

# 8 Test case suite for Adapter regression tests

This clause introduces the different tests cases developed to test the Adapter functionalities. The code below shows these test cases suite.

```
  control {
    execute(TC_GeneralConfigurationMessageOffLineMode());
    execute(TC_GeneralConfigurationMessageLiveMode());
    execute(TC_GeneralConfigurationMessageMerge());
    execute(TC_TriggerUERegister());
    execute(TC_TriggerUERegisterUEDeRegister());
    execute(TC_StartStopCapture());
    execute(TC_Monitoring());
    execute(TC_IMS_0001());
  }
```

**Figure 13: Test suite for Adapter regression testing**

This test suite covers both Lower and Upper test parts and the communications between the system adapter and the TrafficCapture process.

## 8.1 Description of TC_GeneralConfigurationMessageOffLineMode

This script tests a GeneralConfigureReq message to process a PCAP file (offline mode) without merge request.

Note that in this case, the UpperTest adapter part will ignore the equipment access message.

## 8.2 Description of TC_GeneralConfigurationMessageLiveMode

This script tests a GeneralConfigureReq message to process real time traffic capture.

Note that in this case, the UpperTest adapter part will use the human friendly application to guide the user for the actions to achieve.

## 8.3 Description of TC_GeneralConfigurationMessageMerge

This script tests a GeneralConfigureReq message to merge first a set of PCAP files and process the resulting file process real time traffic capture.

Note that in this case, the UpperTest adapter part will use the human friendly application to guide the user for the actions to achieve.

## 8.4 Description of TC_TriggerUERegister

This script tests a UE_REGISTER equipment access message in real time mode in order to use the human friendly application to guide the user for the actions to achieve.

# 9 Deployment diagram of the test adapter

In order to specify the most open and flexible software architecture as possible, the PCAP traffic capture component has been implemented as an independent process from the other adapter implementation, so that it can be executed by a remote computer if needed. In the default configuration the PCAP traffic capture process is assumed to be hosted on the same computer as the main adapter process.

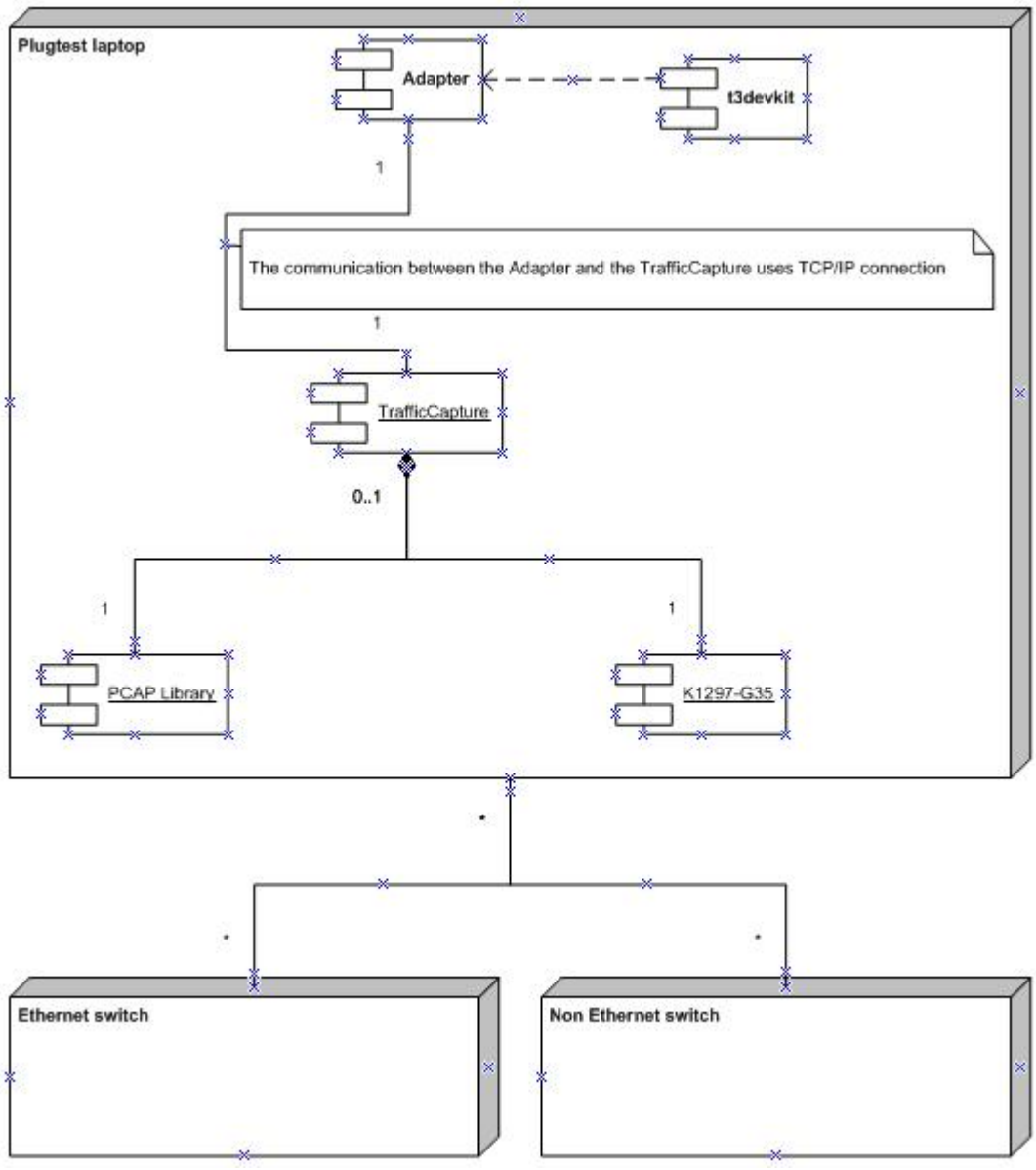



**Figure 14: Lower Test Adapter Deployment diagram**

Figure 14 shows a configuration where the same laptop is assumed to host both Adapter and Traffic Capture processes. This laptop could be connected either to an Ethernet switch or non Ethernet equipment or both.
A Tektronix K1297-G35 with one or more SS7 boards could be an example of a non-PCAP traffic capture tool.
Note that this adapter implementation does not include any K1297-G35 specific code and is just here as an example.

In this case, Adapter and TrafficCapture processes could communicate on local host mode, e.g. IP address could be 127.0.0.1:5501. The Traffic Capture process always acts as a server and the Adapter process as a client. Both use the port 5501.

For installation of the adapter, please refer to the Installation Procedure file described in tr_101561v010101p0.zip attached to the present document.

# 10 Interaction of Adapter and PCAP traffic capture processes

This clause introduces the different diagram sequences to describe the interaction of TTCN-3 scripts (i.e. TTCN-3 TE), Adapter and TrafficCapture processes.

## 10.1 Interactions between TTCN-3 script and Adapter

Figure 15 shows the TTCN-3 message exchanges between a TTCN-3 script and the Adapter.

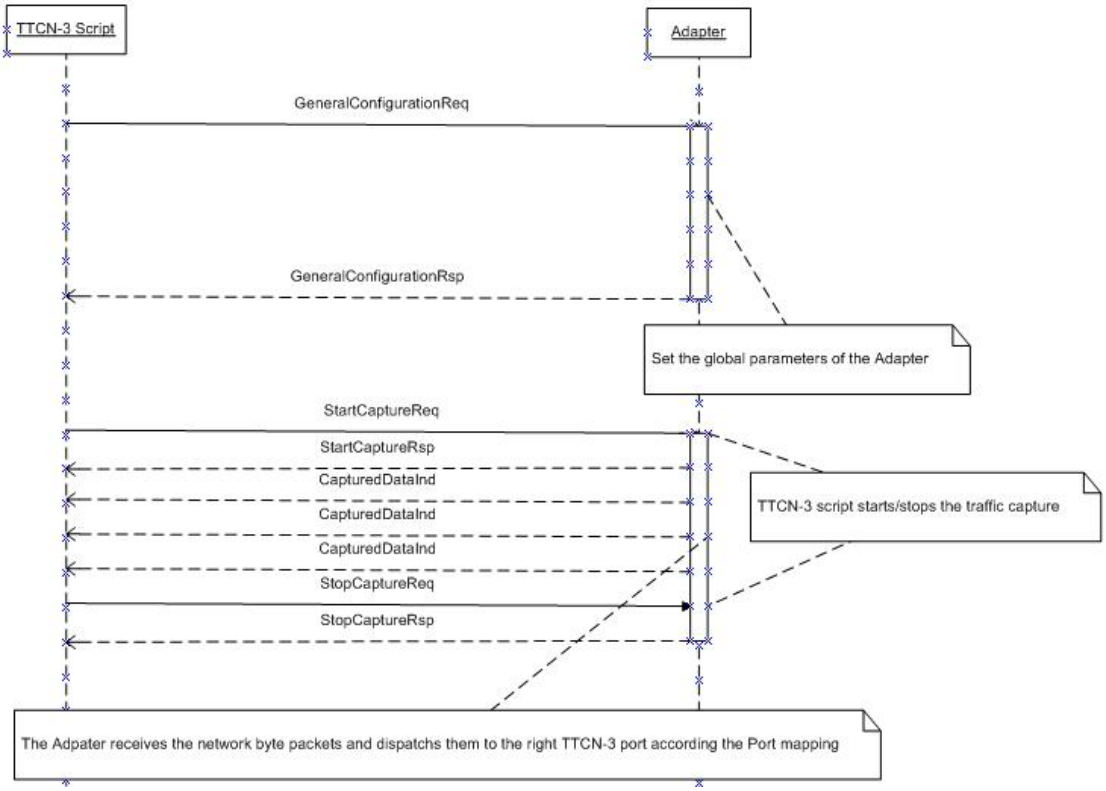



**Figure 15: Message exchanges between TTCN-3 script and the Adapter**

The GeneralConfigurationReq message is assumed by the adapter to be always sent prior to starting traffic capture. Parameters of this message are discussed in clause 6.2.2.2.

SetFilterReq can be sent at any time. Filters will be combined until the StartCaptureReq is received. Any SetFilterReq sent after a StartCaptureReq is ignored until a StopCaptureRequest is received.

Each captured byte packet message includes a complete captured protocol message. Note that these messages are pure data and are not considered as adapter configuration messages, i.e. they are not in any way encoded by the adapter.

## 10.2 Interaction between the Adapter and the traffic capture process

Figure 16 shows the message exchanges between Adapter and TrafficCapture processes. In order to be able to bind the listening socket the TrafficCapture process needs to be statically configured with an IP address and port number to listen to and started as a separate executable.

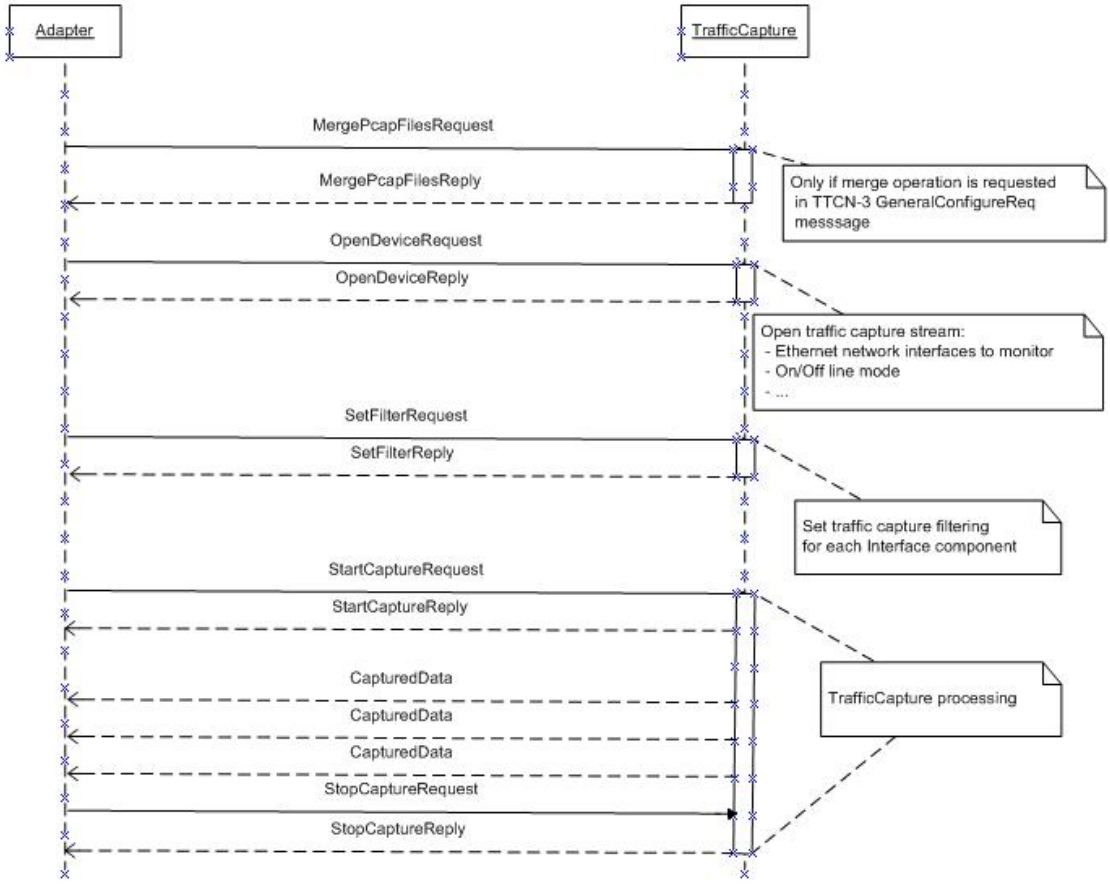



**Figure 16: Message exchanges between TTCN-3 script and the Adapter**

A MergeRequest may be sent to merge a list of files prior to the OpenDeviceRequest. The SetFilterRequest can be used after the OpenDeviceRequest to communicate filter criteria by the adapter (i.e. the combination of filters requested by all test components). The Captured byte packets are in case of the PCAP traffic capture individual Ethernet frames.

Note that the PCAP capture process does not guarantee the presence of complete, e.g. SIP message payloads, within a single Ethernet frame. Payloads may be distributed across multiple frame, e.g. due to IP and/or TCP fragmentation.

Note that all of this communication is transparent to the test system user.

# 11 Class diagram of the Adapter component

The Adapter component is built on two main classes:

1) The UpperTestAdapter class provides the implementation for the GUI upper test adapter.

2) The LowerTestAdapter class provides the implementation of captured traffic processing like isolation and dispatching of protocol messages to the correct TTCN-3 components.

Figure 17 shows the class architecture of the Adapter component.

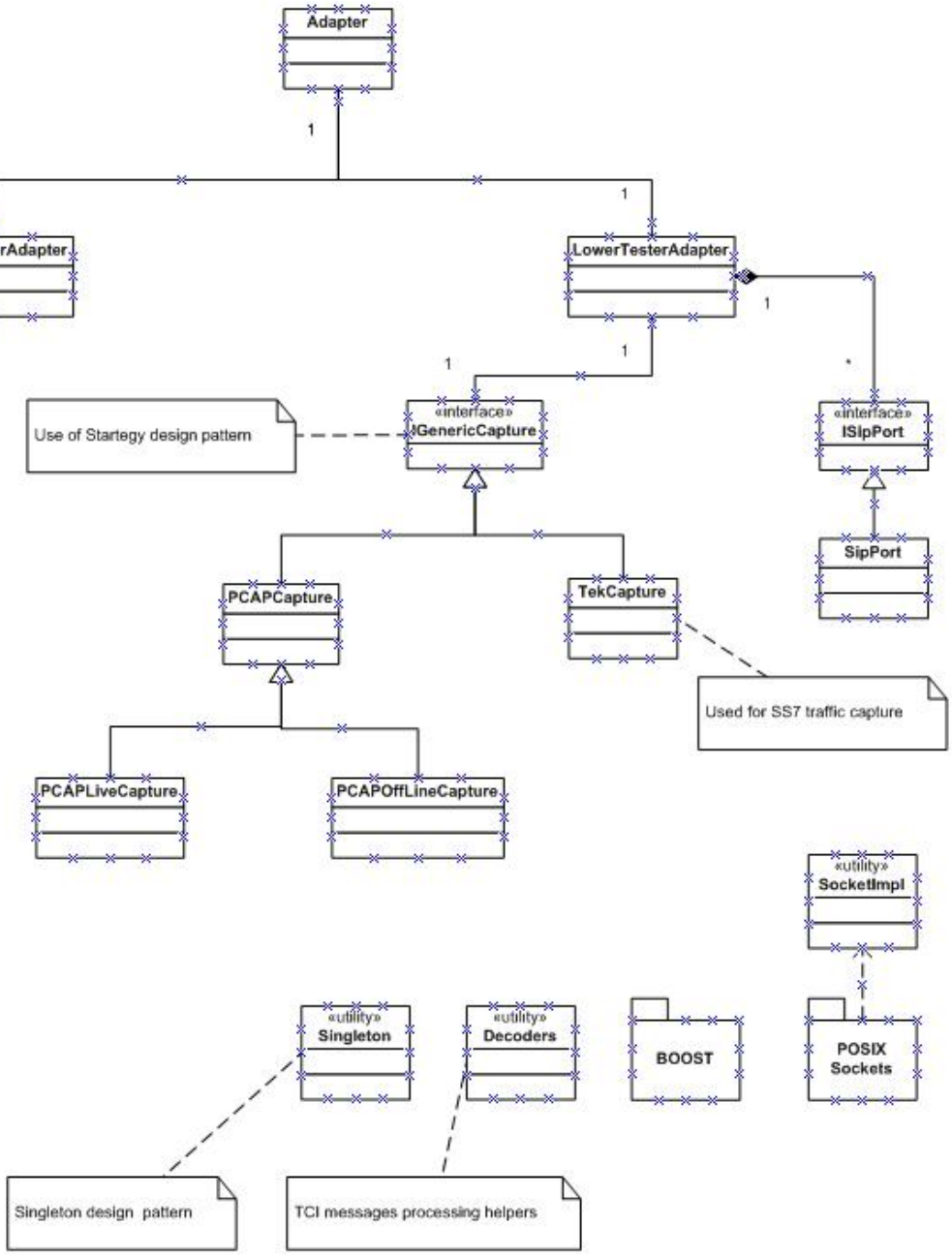



**Figure 17: Adapter class diagram**

## 11.1 UpperTestAdapter class description

Figure 18 describes the architecture of the Upper test adapter part. The main object is the unique instance of the class UpperTestAdapter. This class manages the equipment access messages with the test script.

These equipment messages are processed by the Processor class, one instance per equipment access port. Two kind of processing is currently available:

1) In offline mode, equipment access messages are ignored.

2) In online mode, equipment access messages are processed using the human friendly application.

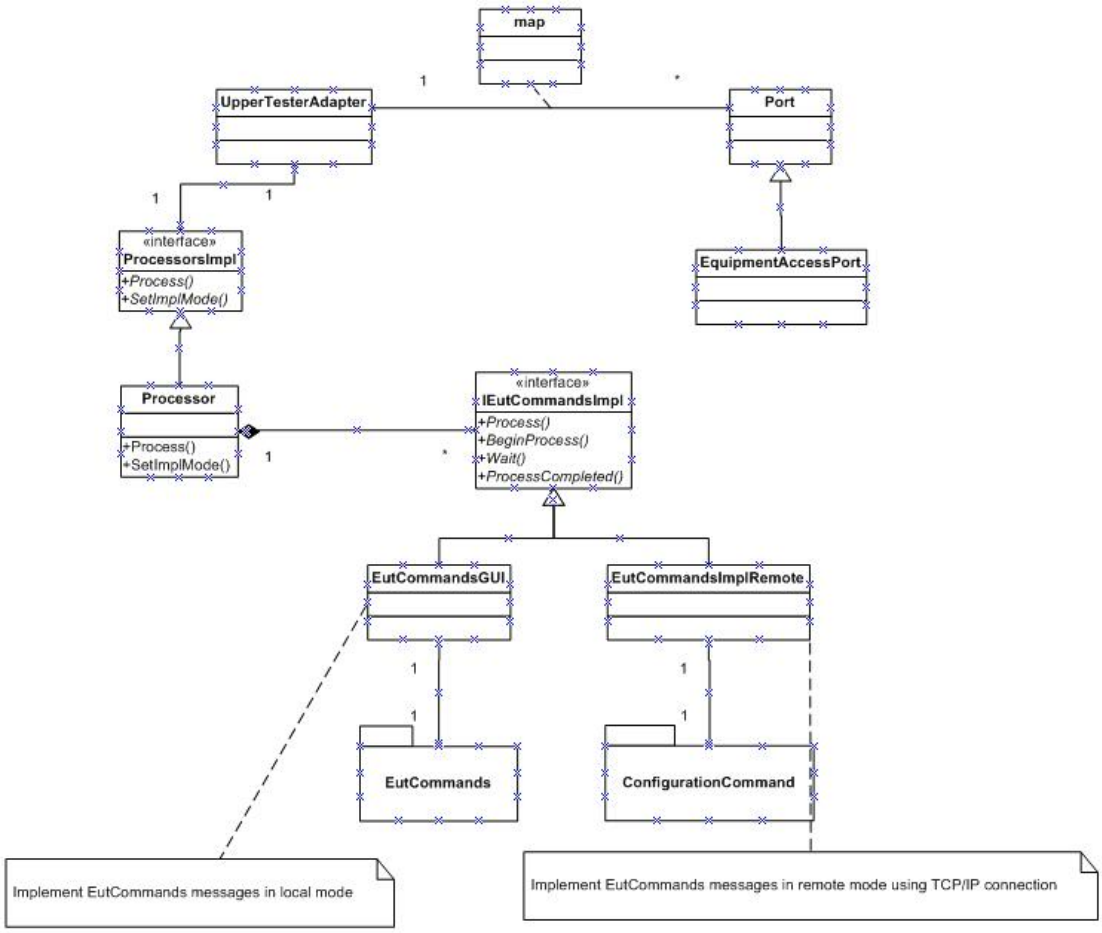



**Figure 18: Class diagram of the upper test adapter**

## 11.2 Helpers classes description

### 11.2.1 TTCN-3 messages decoding helpers

This class provides some helpers methods to decode messages into adapter internal data structures. Encoding rules and examples are shown in clause 6.2.2.3.

Note that the Codec for encoding configuration TTCN-3 messages like GeneralConfigurationReq or decoding messages like GeneralConfigurationRsp in the TTCN-3 TE are described into the Codec - Design document_draft.doc.

### 11.2.2 Socket implementation

The communication between the Adapter and the TrafficCapture processes uses a POSIX socket implementation. The class Socket provides a common implementation for all Adapter development.

### 11.2.3 Log framework

It provides a set of method for trace warning/error messages. It supports also trace levels (ERROR, INFO and DEBUG).

### 11.2.4 BOOST framework

Boost is a free library which is aimed at providing quality software components to developers, whilst using the styles of the Standard Template Library. Some of the components within the library may be put forward as future extensions to the Standard Library.

Please refer to the BOOST references in [i.1] for a full documentation of the Boost framework.

### 11.2.5 Common development rules

This clause provides a list of common usage in the Adapter development process:

- All the code shall be properly documented (principles, classes, methods, declarations, etc.).
- 'Doxygen' style comments are used for code documentation.
- For threading, boost with static method has been selected over class thread.

# 12 Implementation details

This clause introduces some development details of the different Adapter software components.

## 12.1 TTCN-3 messages execution

This clause shows the different implementations of the TTCN-3 messages supported by TTCN-3 components EutTrigger and EutConfiguration.

### 12.1.1 Automate equipment operation

This automation of equipment operation commands is vendor specific. However, a basic component, quickly customizable for each vendor, could be developed and integrated into the current software architecture.

### 12.1.2 Human friendly GUI

This kind of implementation is used when there is no way to automate equipment operation. In this case, equipment operation commands and parameters are presented by a graphical application to guide the user to achieve them. The messages to be displayed are stored in an XML file, one message per operation. This file can be upgraded in real time. This upgrade includes:

- Modifying existing messages.
- Adding new TTCN-3 messages support.

## 12.2 Class diagrams

Figure 19 describes the static architecture of the upper test adapter. It manages the equipment operation message port.

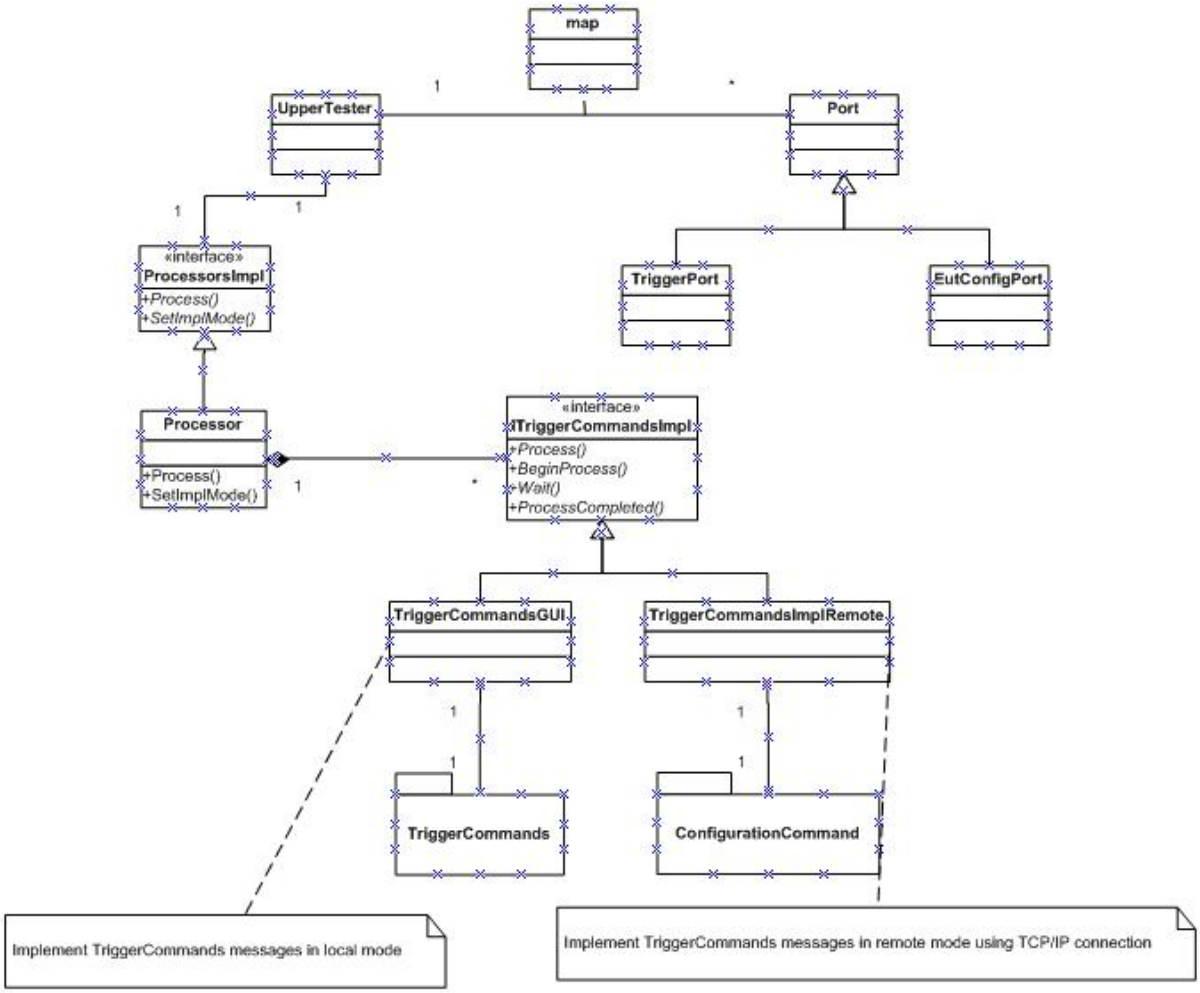



**Figure 19: Class diagram of the upper test adapter**

## 12.3 Human friendly GUI

The human friendly GUI application is a component of the system adapter. It provides configurable messages/actions to guide the user to process actions or configure equipments. The messages/actions are displayed according to trigger messages received from the system adapter.

The human friendly GUI application must be send before starting the test campaign; it acts like a server.

The following steps describe the behaviour:

1) The Upper test adapter receives a trigger message from the test script.

2) The Upper test adapter relays this trigger message to the human friendly GUI application and wait for the action result.

3) The Upper test adapter sends the action result to the test script.

The human friendly GUI application uses an XML configuration file to set the message and the action type for each trigger message: `<?xml version="1.0" encoding="utf-8" ?>`

```
<!-- Define default values for TTCN scrpts. These values will override values defined in the
TCs. -->
<UserGuideMessages>
  <Strategies>
    <Strategy name="Lannion" />
  </Strategies>
  <Lannion>
    <UE_REGISTRATION Text="Please register a SIP session with the following
parameters:\n\t{0}\n\t{1}\n\t{2}" Actions="OkCancel" />
    <UE_CHECK_IS_REGISTER Text="Please check that you are currently registered"
Actions="OkCancel" />
    <UE_CHECK_MESSAGE_RECEIPT Text="If you received the message '123', please click on button
Yes, otherwise, click on No" Actions="YesNo" />
    <UE_SEND_MESSAGE Text="Please send the message '123'" Actions="OkCancel" />
    <UE_DEREGISTRATION Text="Please de-register your current SIP session" Actions="OkCancel" />
  </Lannion>
</UserGuideMessages>
```

## 12.4 TrafficCapture component

TrafficCapture is a component of the system adapter which takes care of capturing traffic from a network adapter. For correct functionality, it requires a pcap driver to be installed. It works as a standalone process communicating with the LowerTest component using the TCP/IP protocol. In this communication, TrafficCapture works as a server.

### 12.4.1 Usage

When launched, the application starts listening on a specified port and waits for a connection attempt from the LowerTest component. The port number can be specified by the -p command line argument. If no port number is supplied this way, TrafficCapture uses port 5501.

After LowerTest becomes connected, it sends several requests to initiate traffic capture according to requirements specified in a TTCN-3 test case. TrafficCapture processes these request and replies to them returning a success code. If no errors occur during this procedure, TrafficCapture starts capturing frames and sending them to LowerTest for further processing.

During the test case, LowerTest can request to interrupt and resume capture. When the test case is over, LowerTest closes TCP/IP connection and TrafficCapture returns to the initial mode, waiting for new connection requests.

TrafficCapture TCP/IP interface does not contain any command for ending the application. It can be stopped manually by pressing <ctrl-c>.

For debugging purposes, the application output can be customised using the following command line arguments:

- Linfo: information messages are displayed.
- Lerr: errors are displayed.
- Lwarn: warnings are displayed.
- Ldebug: debugging information are displayed.
- Lcapt: capturing information are displayed.
- Lall: all messages are displayed.
- Lnone: no messages are displayed are displayed.

With the exception of last two switches, all other logging parameters can be combined.

## 12.4.2 Architecture

The core object of the application is a singleton TcpipServer instance. This instance opens a listening socket and accepts incoming connections. For all established connections, a separate ConnectionController is created.

The controller object runs in an own thread and processes incoming messages from the client. It passes the received binary data to a TrafficCaptureMessageFactory singleton. This factory object tries to convert the data to a message instance. All message instances generated by the factory are derived from a TrafficCaptureMessage class.

The generated message instance is later analysed by the controller and an appropriate action is concerning a capture device is taken (creating, starting, stopping, etc.).

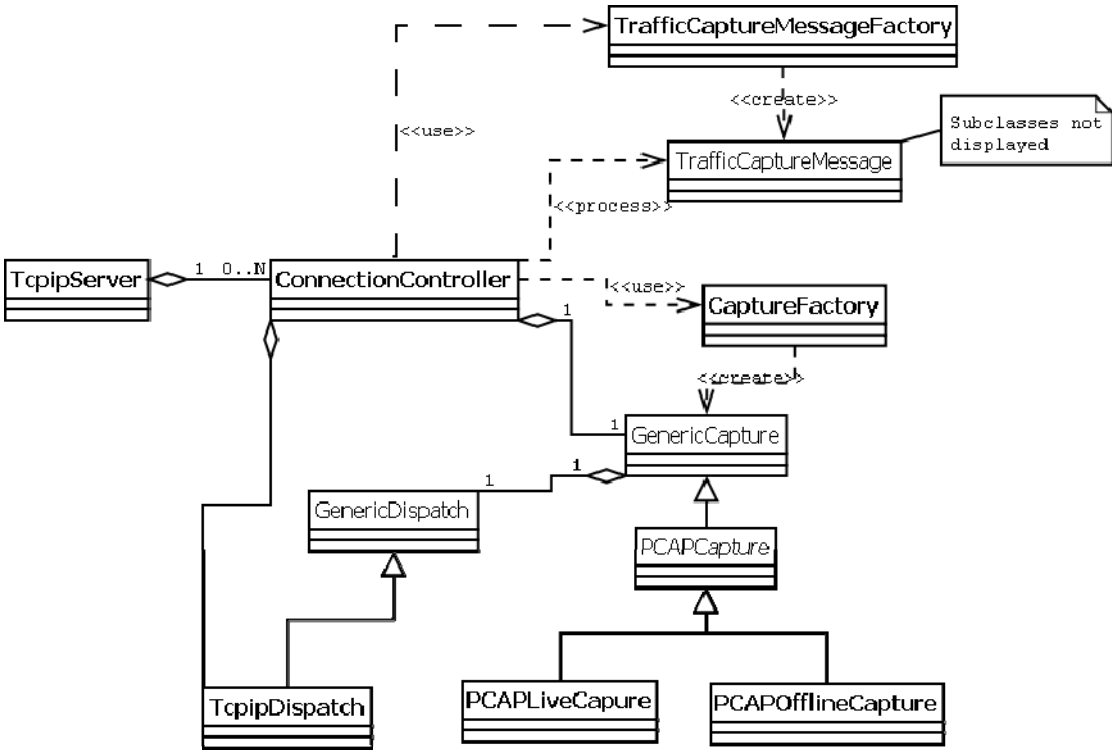



**Figure 20: TrafficCapture Class Diagram**

### 12.4.3 Functional Specification

All messages used in the communication with LowerTest are displayed in figure 21. This sequence diagram displays a typical scenario for a whole capture session. Individual use cases are described in detail in the following clauses.

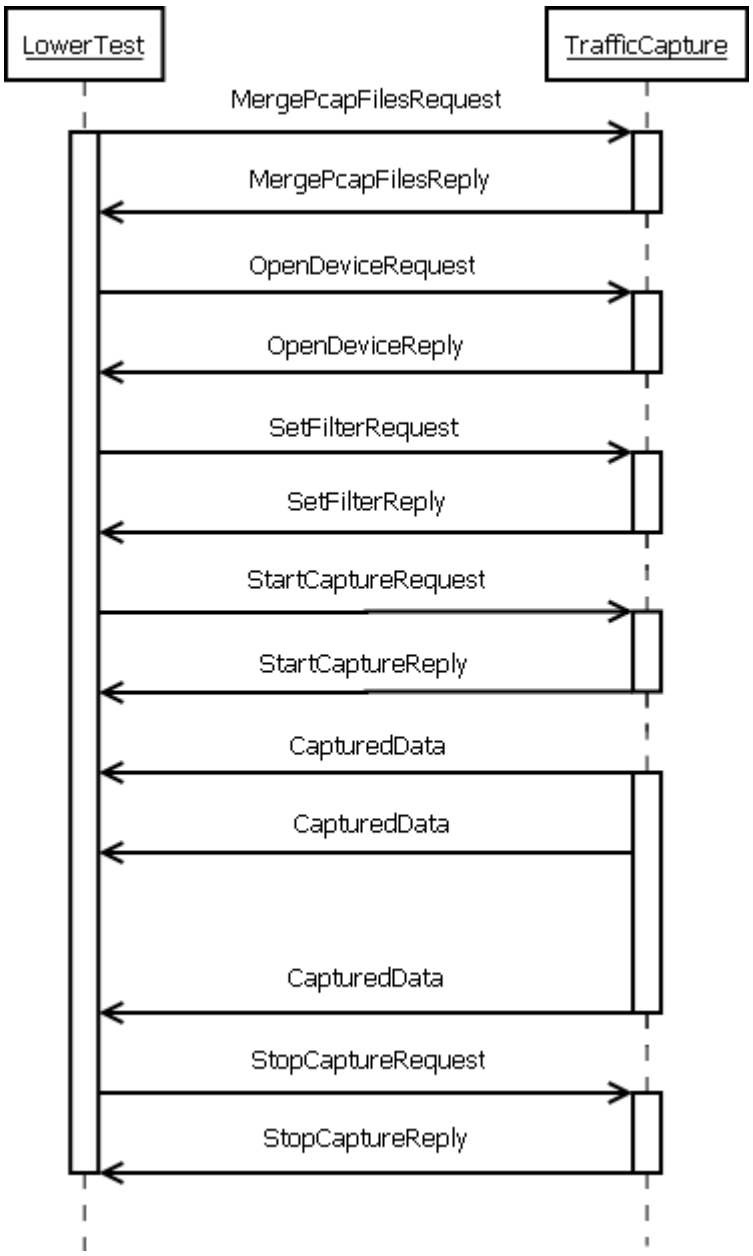



**Figure 21: Sequence diagram of typical TrafficCapture session**

#### 12.4.3.1 UC 01: File Merging

| | |
|---|---|
| Precondition | TCP/IP connection established. |
| Description | LowerTest sends a request (MergePcapFilesRequest) to merge two or more pcap files. TrafficCapture performs the operation, using an external tool - mergecap from the Wireshark package and send the result back to LowerTest (MergePcapFilesReply message). |
| Success | Merge file is created. LowerTest can get a path to the file from the reply message. |
| Exceptions | In case of any exception, the reply message success field is set to false and the path to the merge file is empty. Possible causes are as follows:<br>- Invalid path to the mergecap tool.<br>- Invalid path to the directory where the merge file should be created.<br>- Merge file already exists and it is not possible to overwrite it.<br>- Source files are not found. |

#### 12.4.3.2 UC 02: Opening Device

| | |
|---|---|
| Precondition | TCP/IP connection established<br>In case of offline approach using multiple files, the files must be merged (clause 12.4.3.2) |
| Description | LowerTest sends a request (OpenDeviceRequest) to prepare a capturing device. TrafficCapture creates the device using a factory approach and initialises it. The of operation result is sent back to LowerTest in a OpenDeviceReply message. |
| Success | Capturing device is ready and packet capturing can be started. |
| Exceptions | There are two different result codes indicating an error. If the result is a partial success, TrafficCapture detected an error, but the device is still capable of data capture at least from one source. If the result is a complete failure, capture cannot be started. The main causes of error are as follows:<br>1. Invalid format of parameter describing capturing device (incorrect network adapter, pcap file missing, etc.).<br>2. Pcap driver not installed.<br>3. Invalid/not supported device type requested. |

#### 12.4.3.3 UC 03: Setting Filter

| | |
|---|---|
| Precondition | Capturing device ready (clause 12.4.3.2). |
| Description | LowerTest sends a request (SetFilterRequest) to set a filter for the capturing device. TrafficCapture applies the filter to the device overwriting the previous filter and sends back the operation result in a SetFilterReply message. |
| Success | Filter applied to capturing device. |
| Exceptions | 1. Capturing device not initialised yet.<br>2. Invalid filter format. |

#### 12.4.3.4 UC 04: Starting Capture

| | |
|---|---|
| Precondition | Capturing device ready (clause 12.4.3.2). |
| Description | LowerTest sends a request (StartCaptureRequest) to start capture. TrafficCapture replies with StartCaptureReply and starts sending CapturedData indication messages to LowerTest. These messages contain captured frames. |
| Success | Captured packets are being sent to LowerTest. |
| Exceptions | 1. Capturing device not initialised yet. |

#### 12.4.3.5 UC 05: Stopping Capture

| | |
|---|---|
| Precondition | Packet capture started (clause 12.4.3.4). |
| Description | LowerTest sends a request (StopCaptureRequest) to stop frame capture. TrafficCapture stops sending CaptureData indications and replies with StopCaptureReply. |
| Success | Captured frames are no longer sent to LowerTest. Notice that the capturing device is still in initialised state, so it is possible to restart capture. |
| Exceptions | 1. Capturing device not initialised yet. |

### 12.4.4 Compilation

The application is written in C++. It can be compiled with VisualStudio or gcc (tested with cygwin and MinGW version). The application uses two external libraries: pcap and boost. In case of compilation for Windows platform, Winsock 2 library is required as well.

## 13 Test case suite for Codec: the torture tests

In the case of an encoder input data is controlled and cannot go beyond the TTCN-3 typing possibilities. The decoder however can receive a huge variety of encoded messages. Therefore it is essential to ensure that the decoder will be robust enough to deal with these messages and to decode them. To achieve this goal, this second validation approach makes use of torture messages defined in RFC 4475 [i.8]. Each of these messages presents some characteristics which would cause trouble to weak decoders.

### 13.1 Test System Architecture

These tests rely on the injection by the SUT adapter of encoded messages based on the testcase name and on timing after `TriMap()` call.

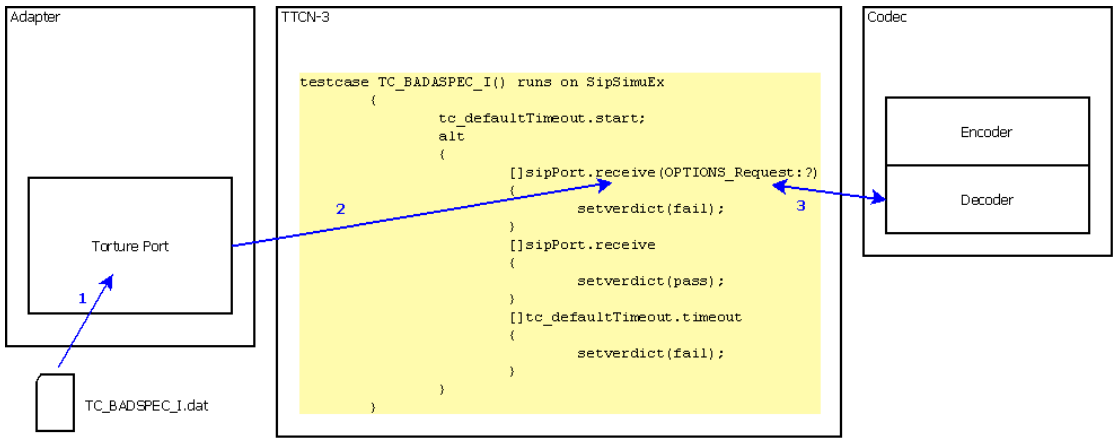



**Figure 22: Architecture**

```
bool SipTorturePort::Map (const PortId& connected_port_id)
{
        string filename ("data/");
        filename += GetTestcaseId().GetObjectName();
        filename += ".dat";
        cout << "Reading testcase data from " << filename << endl;

        ifstream in (filename.c_str(), ios_base::in | ios_base::binary);
        if (!in) {
                cerr << "Cannot open " << filename << endl;
                return false;
        }
        in.seekg (0, ios_base::end);
        streampos size = in.tellg();
        in.seekg (0, ios_base::beg);
        char* buff = new char[size];
        in.read (buff, size);
        EnqueueMsg (connected_port_id, MappedBitstring (buff, size*8));
        delete buff;
        return in.good();
}
```

All test cases look more or less the same since templates are only specified at the message type level, e.g. any INVITE_Request. The verdict is assigned depending on decoding result. In the following test cases example a SIP message is actually not valid and should therefore only set the verdict pass if it could not be decoded.

```
   testcase TC_BADASPEC_I() runs on SipSimuEx
   {
      tc_defaultTimeout.start;
      alt
      {
         []sipPort.receive(OPTIONS_Request:?)
         {
            setverdict(fail);
         }
         []sipPort.receive
         {
            setverdict(pass);
         }
         []tc_defaultTimeout.timeout
         {
            setverdict(fail);
         }
      }
   }
```

```
OPTIONS sip:user@example.org SIP/2.0N)
Via: SIP/2.0/UDP host4.example.com:5060;branch=z9hG4bKkdju43234
Max-Forwards: 70
From: "Bell, Alexander" <sip:a.g.bell@example.com>;tag=433423
To: "Watson, Thomas" < sip:t.watson@example.org >
Call-ID: badaspec.sdf0234n2nds0a099u23h3hnnw009cdkne3
Accept: application/sdp
CSeq: 3923239 OPTIONS
l: 0
```

# 14 Testing of Test Adapter

In order to validate the Adapter functionalities and to provide a tool for regression tests, the adapter development provides a test suite named TestExecution, written in TTCN-3.

This test suite covers the following functionalities:

- Merge PCAP file tests.
- General configuration message processing, including on-line vs. off-line mode).
- EUTs IP interface settings tests.
- PCAP Filtering tests.

- Start/Stop capture operations.
- Traffic capture monitoring.

# 15 SVN repositories

The test adapter sources were archived into the STF407 project's SVN repositories and are attached to the present document in tr_101561v010101p0.zip.

# 16 Development tools

The adapter project uses the external libraries described below:

- BOOST: two versions are used:
  - The version provided by CYGWIN (boost-1_33_1). It is used by the t3devkit toolkit. For more details, please refer to the installation procedure.
  - The latest version (currently boost_1_39_0), located here: http://www.boost.org/doc/libs/1_39_1/. It is used by the TrafficCapture component.
- WinPcap 4.0.2 developer's pack downloaded from http://www.winpcap.org/devel.htm. It is used by the TrafficCapture component.

# 17 RFCs covered by the codec

The codec should be able to encode and decode all SIP messages supported by the TTCN-3 SIP library type structure. Therefore it should cover the message formats described in the following RFCs:

- RFC 3261 [i.9]
- RFC 3262 [i.10]
- RFC 3265 [i.11]
- RFC 3313 [i.12]
- RFC 3323 [i.13]
- RFC 3325 [i.14]
- RFC 3326 [i.15]
- RFC 3327 [i.16]
- RFC 3329 [i.17]
- RFC 3455 [i.18]
- RFC 3515 [i.19]
- RFC 3608 [i.20]
- RFC 3841 [i.21]
- RFC 3891 [i.22]
- RFC 3892 [i.23]
- RFC 4028 [i.24]

- RFC 4244 [i.25]
- RFC 5009 [i.26]

Some SIP message constructs reuse some headers defined in the HTTP protocol. Therefore the following RFCs are partially supported:

- RFC 2616 [i.27]
- RFC 2617 [i.28]

# List of Tables

# List of Figures

# History

| Document history | | |
|---|---|---|
| V1.1.1 | June 2011 | Publication |
| | | |
| | | |
| | | |
| | | |