TTCN-3 IOT Adapter Design document

Contents

List of Tables

List of Figures

1 Introduction

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

For further information the reader is referred to [Fwk] for global view of methodology and framework for automated interoperability testing and [IMS arch] for an overall view of the IMS interoperability test architecture which has served as the main source for design requirements.

This document has been written with the assumption that the reader is well versed in C++ and TTCN-3 [core] programming. Also good knowledge of the operation of TRI [TRI] and TCI [TCI] standards is assumed.

2 Design Objective

The main purpose of the TTCN-3 interoperability test adapter is to implement the real test system interface [core, TRI] of the TTCN-3 IMS interoperability test system described in [IMS arch], 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 for 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.

3 Abbreviations

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

CN	Core Network
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
ISUP	ISDN User Part
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 (as defined in [TRI] and [TCI]
UE	IMS User Equipment

4 Design proposals for receiving of traffic capture

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

4.1 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.

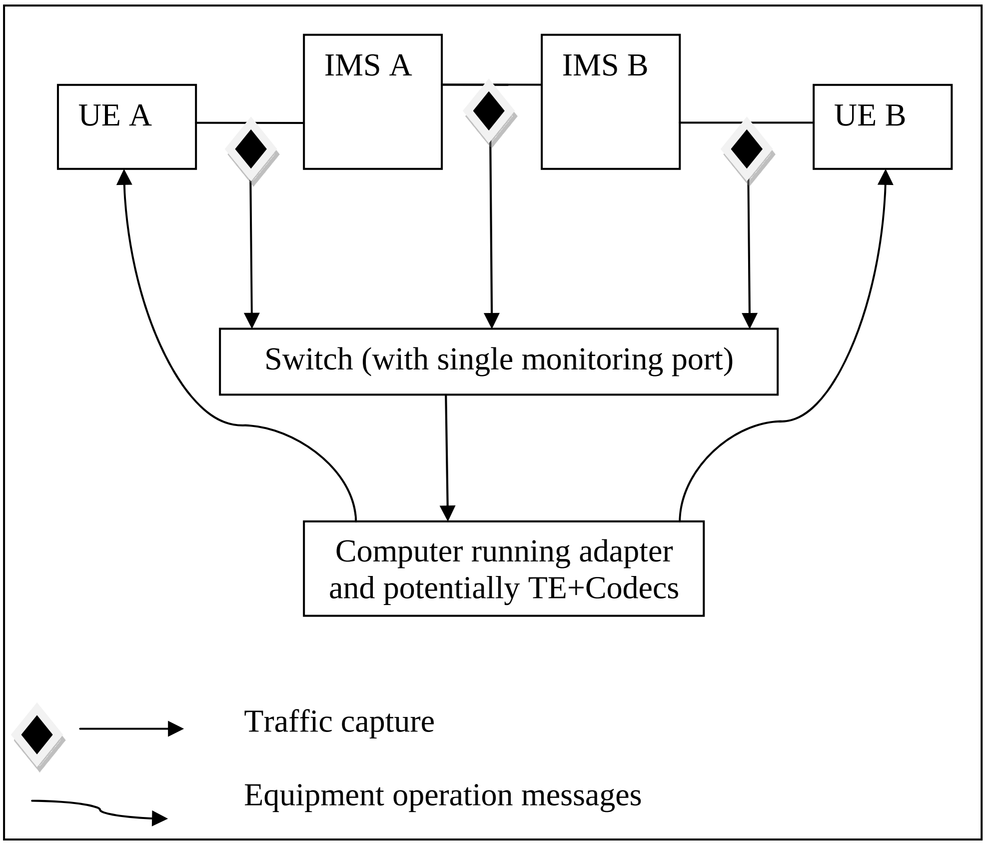


Figure 1: Design with a switch providing single (mirrored) switch port for all traffic in context of IMS interoperability testing

4.2 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.

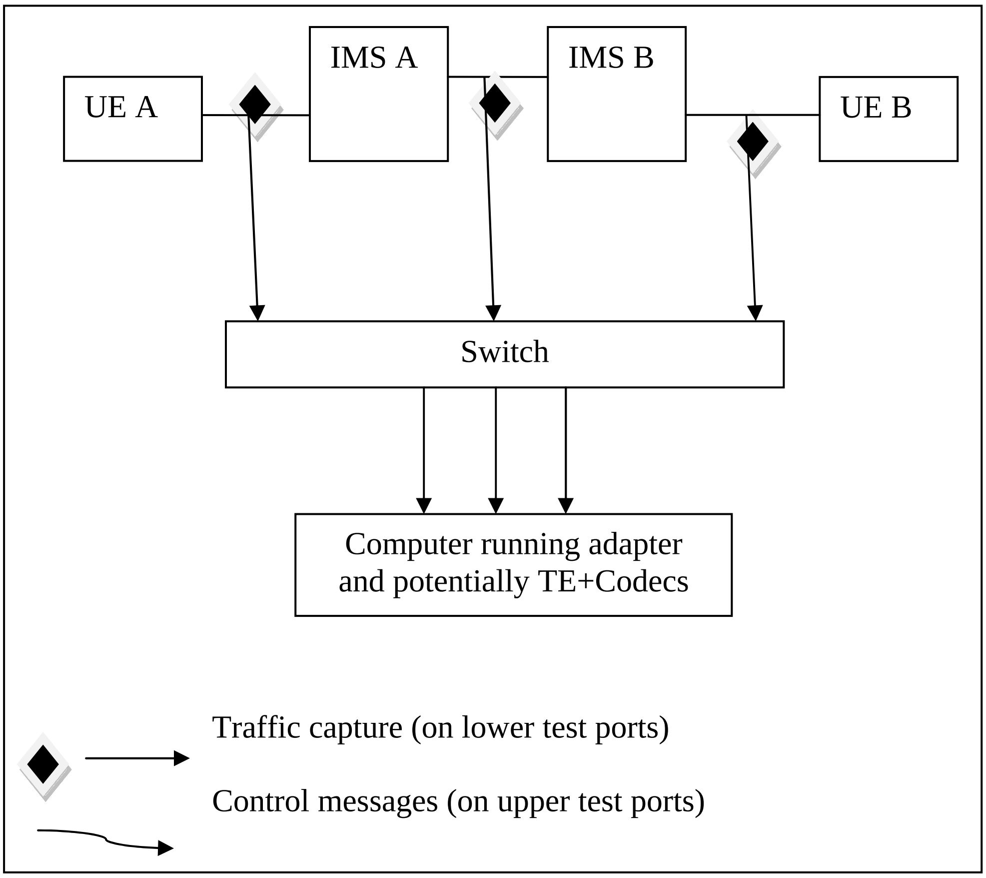


Figure 2: Design with a switch providing multiple ports for traffic in context of IMS interoperability testing

4.3 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.

5 Software design

This chapter introduces the requirements taken into account for the software design of the test adapter.

5.1 Requirements

The test adapter software shall address the following 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

Table 1: Test adapter software requirements

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

5.2 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 [t3devkit]) 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++. Note that there is this C++ is not compliant to the standardized TRI C++ mapping.

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

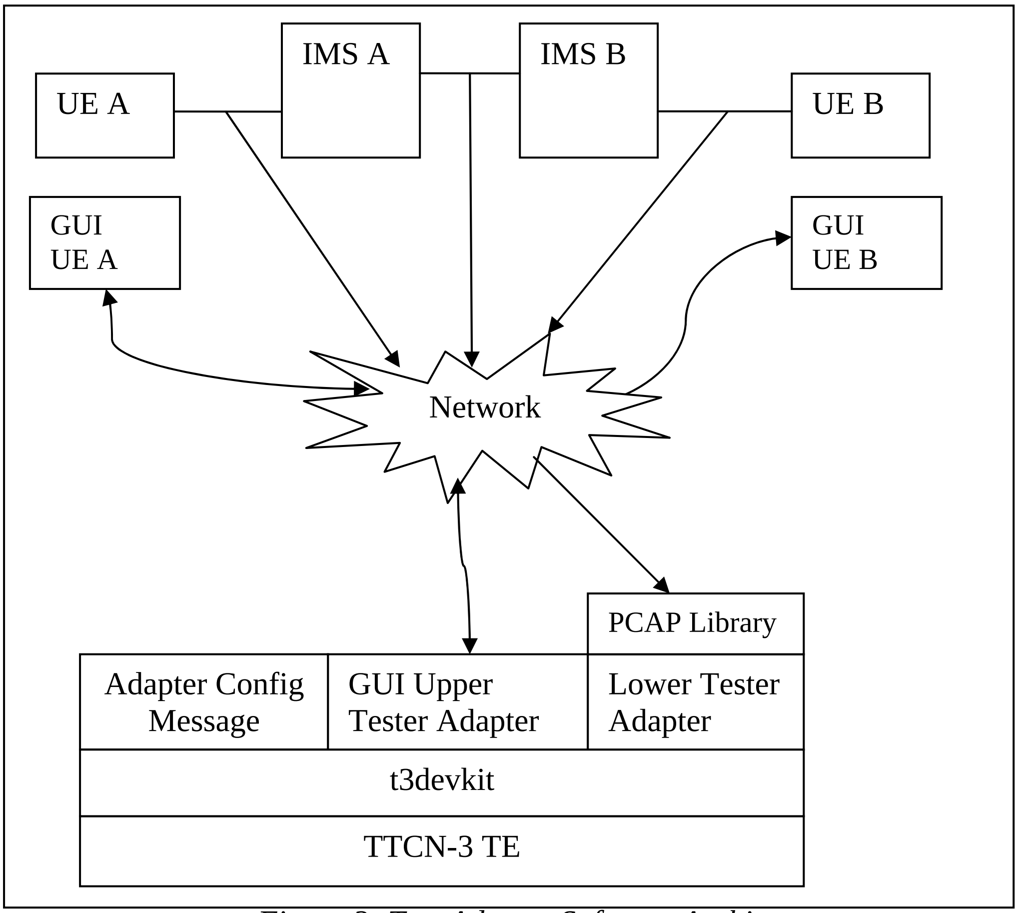


Figure 3: Test Adapter Software Architecture

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

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

5.2.1 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

5.2.2 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.

5.2.2.1 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 below table 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 defined in section X
- 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

Message type	Octet Value Encoding
GeneralConfigurationReq	0x00
GeneralConfigurationRsp	0x01
SetFilterReq	0x02
SetFilterRsp	0x03
StartTrafficCaptureReq	0x04
StartTrafficCaptureRsp	0x05
StopTrafficCaptureReq	0x06
StopTrafficCaptureRsp	0x07
EquipmentOperationReq	0x08
EquipmentOperationRsp	0x09

Table1. Adapter configuration and equipment operation messages and their message type encoding

- The Figure 4 shows an example of an encoded GeneralConfigurationReq message in hexadecimal string format
    - In the sequence ‘000000093132372E302E302E31’H:
        - ‘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

```
    000000093132372E302E302E3100000000000000157D0000000000000007AE0000
00000666696C7465720000005B72706361703A2F2F5C4465766963655C4E50465F
7B46333031333632462D374444422D343237432D423436352D38324337384543443
34437347D3B72706361703A2F2F5C4465766963655C4E50465F7B44756D6D7949
666163657D
```

Figure 4: An example of encoded GeneralConfigurationReq message in hexadecimal

- The Figure 5 shows an example of an encoded equipment operation message in hexadecimal format
    - In this sequence ‘000F55455F524547495354524154494F4E’H:
        - ‘000F’H is the length of the cmd information element value “UE_REGISTRATION”
        - 55455F524547495354524154494F4E 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

```
    0000000F55455F524547495354524154494F4E000000030000000775736572534
9500000000A3333343431323334333200000006313233343536
```

Figure 5: An example of encoded UE_REGISTRATION equipment operation message

5.2.3 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 this document.

5.2.3.1 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!

5.2.4 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 via achieved either via telnet, ssh, HTTP/HTTPS, or TCP/IP connection and a port managed by XInetd.

5.2.4.1 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.

6 TTCN-3 message type definitions

This chapter 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.

6.1 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

6.2 TTCN-3 messages description

The Figure 6 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 GeneralConfigurationReq {
      charstring  captureProcessIpAddress,
```

```
    PortNumber  captureProcessPort,
    CaptureMode captureMode
   }

  type record Status {
    FncRetCode code, charstring reason optional
   }

   type record GeneralConfigurationRsp
   {
    Status status
   }
```

Figure 6: TTCN-3 types that define GeneralConfigurationReq/Rsp

6.2.1 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 the chapter 5.2.2.1.

6.2.2 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.

6.2.3 List of physical interfaces

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

6.2.4 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_0020.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}";

        /**
         * @desc
         *     Traffic capture filtering.
```

```
  */
  modulepar charstring PX_IOT_TRAFFIC_CAPTURE_FILTERS := "(ip.proto == 0x11 && udp.port == 5060 && (ip.addr == 172.31.42.2 || ip.src == 172.31.42.3 || ip.src == 172.31.42.4 || ip.src == 172.31.42.5 || ip.src == 172.31.42.50))";

} // group adapterGlobalConfiguration
```

Figure 7: Example TTCN-3 parameter setting for general configuration message

6.3 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 whne 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 8 TTCN-3 types for Start/StopTrafficCaptureReq/Rsp

6.4 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 9 TTCN-3 types for Start/StopTrafficCaptureReq/Rsp

6.5 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 10 TTCN-3 types for Start/StopTrafficCaptureReq/Rsp

6.6 Test case suite for Adapter regression tests

This chapter 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_TOTO());
      execute(TC_IMS_0001());
    }
```

Figure 11: Test suite for Adapter regression testing

To be continued by Yann

7 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.

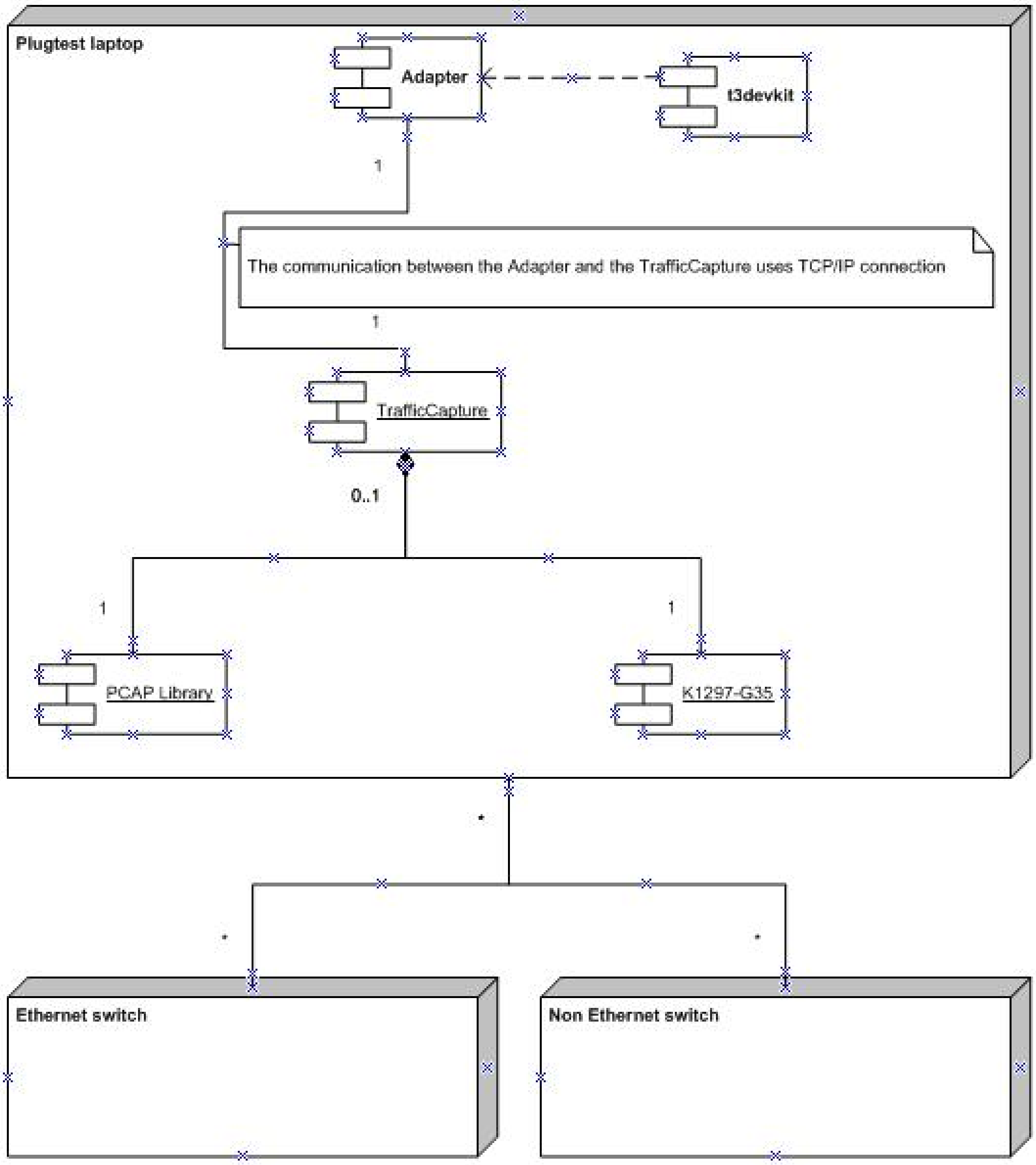


Figure 12: Lower Test Adapter Deployment diagram

The figure above 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 located here: H:\STF370\WP2 - IMS case study\adapter.

8 Interaction of Adapter and PCAP traffic capture processes

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

8.1 Interactions between TTCN-3 script and Adapter

The figure below shows the TTCN-3 message exchanges between a TTCN-3 script and the Adapter.

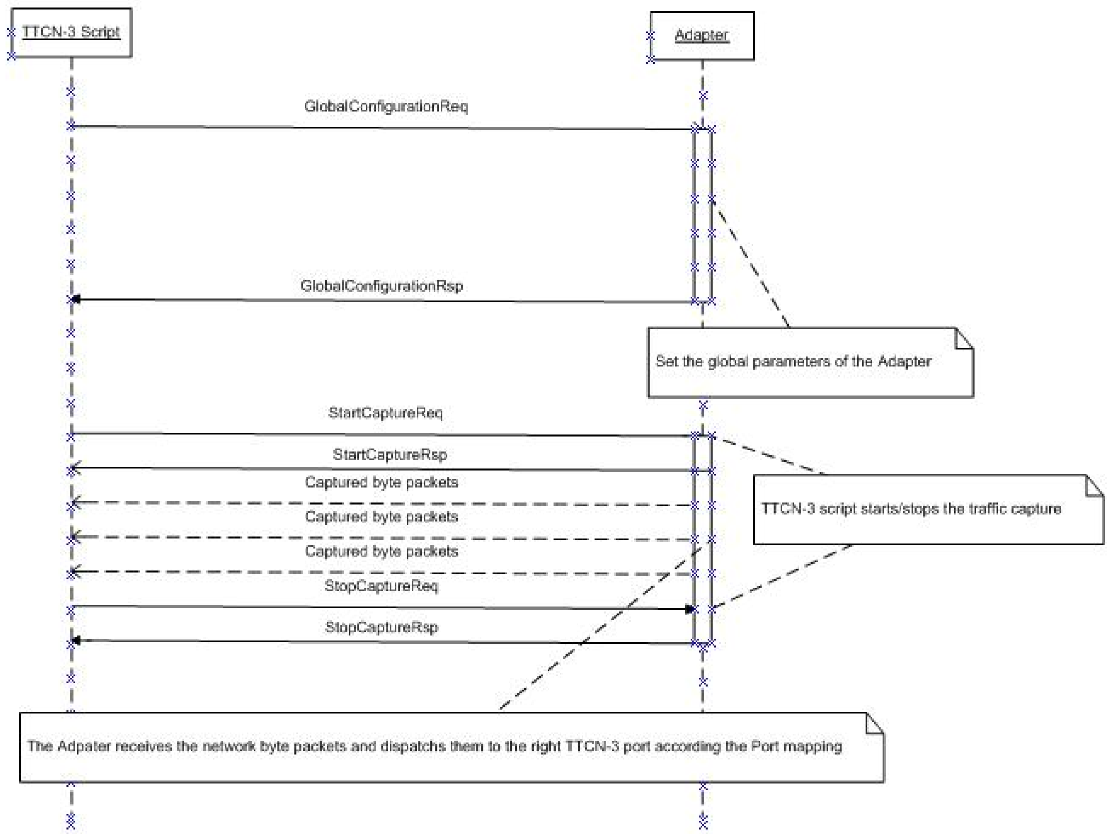


Figure 13: Message exchanges between TTCN-3 script and the Adapter

The GeneralConfigurationReq message is assumed by the adapter to be always sent prior to starting traffic capture. Parameters of this message are discussed in section 5.2.2.

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.

8.2 Interaction between the Adapter and the traffic capture process

The figure below 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.

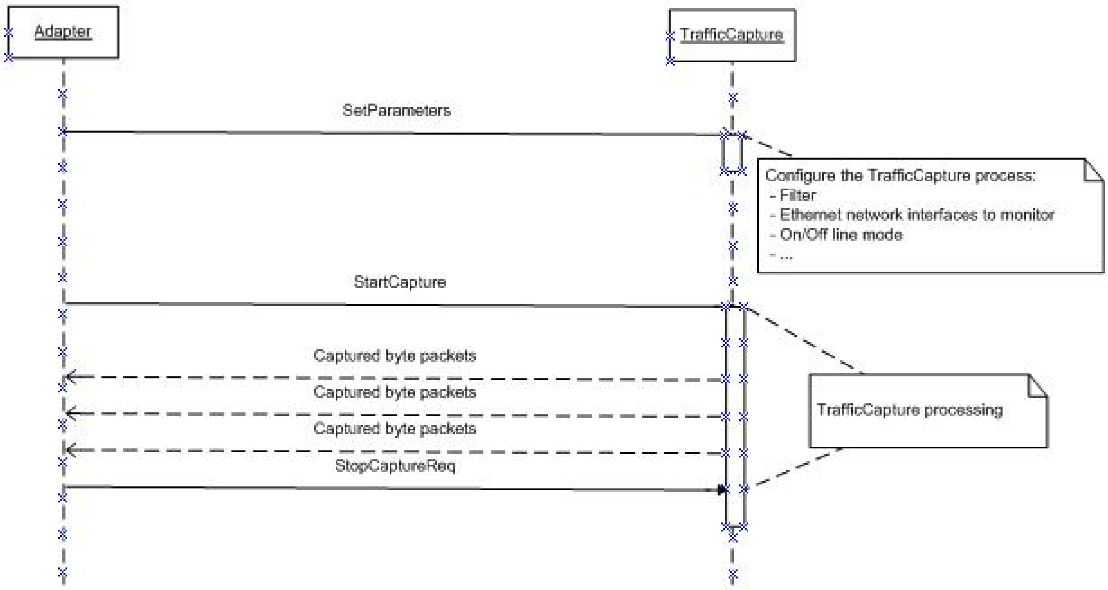


Figure 14: Message exchanges between TTCN-3 script and the Adapter

A MergeRequest may be sent to merge a list of files prior to the OpenDeviceRequest. The SetParameters 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 payolads, 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.

9 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

The Figure 15 shows the class architecture of the Adapter component.

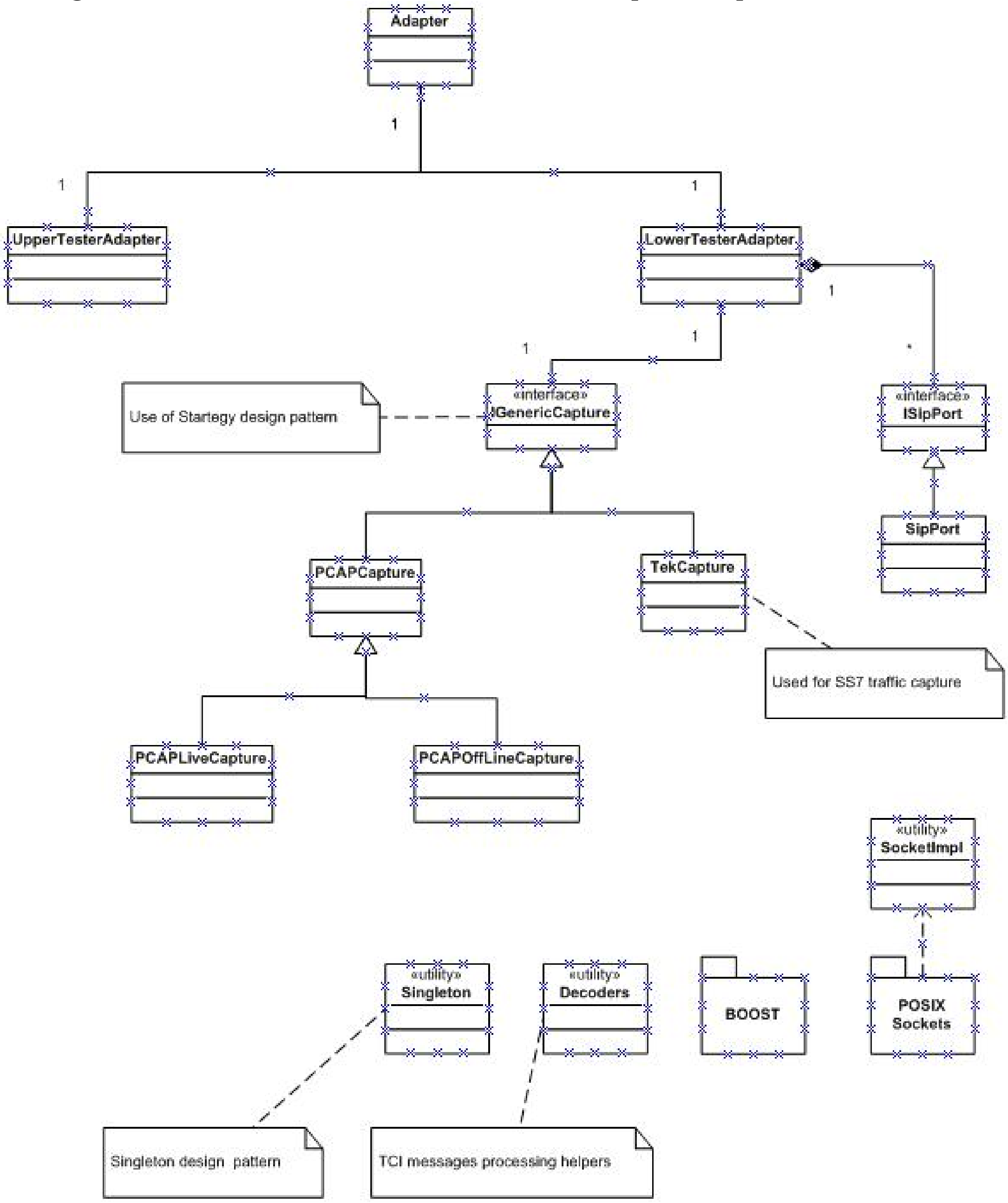


Figure 15: Adapter class diagram

9.1 LowerTestAdapter class description

TODO: To be continued by Alexendre

9.2 UpperTestAdapter class description

TODO: To be continued by Yann

9.3 Traffic capture related classes description

TODO: To be continued by Tomas

9.4 Helpers classes description

9.4.1 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 section 5.2.2.1.

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

9.4.2 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.

9.4.3 Log framework

TODO: To be continued

9.4.4 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 references [BOOST] for a full documentation of the Boost framework.

9.5 Common development rules

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

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

10 Implementation details

This chapter introduces the development details of the different Adapter software components.

10.1 LowerTest component

To be continued by Alexandre

10.2 UpperTest component

10.2.1 TTCN-3 messages execution

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

10.2.1.1 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.

10.2.1.2 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 a graphical application to guide a human equipment user in the operation of equipment. 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

10.2.2 Sequences diagrams

To be continued by yann

10.2.3 Class diagrams

The figure below describes the static architecture of the upper test adapter. It manages the equipment operation message port.

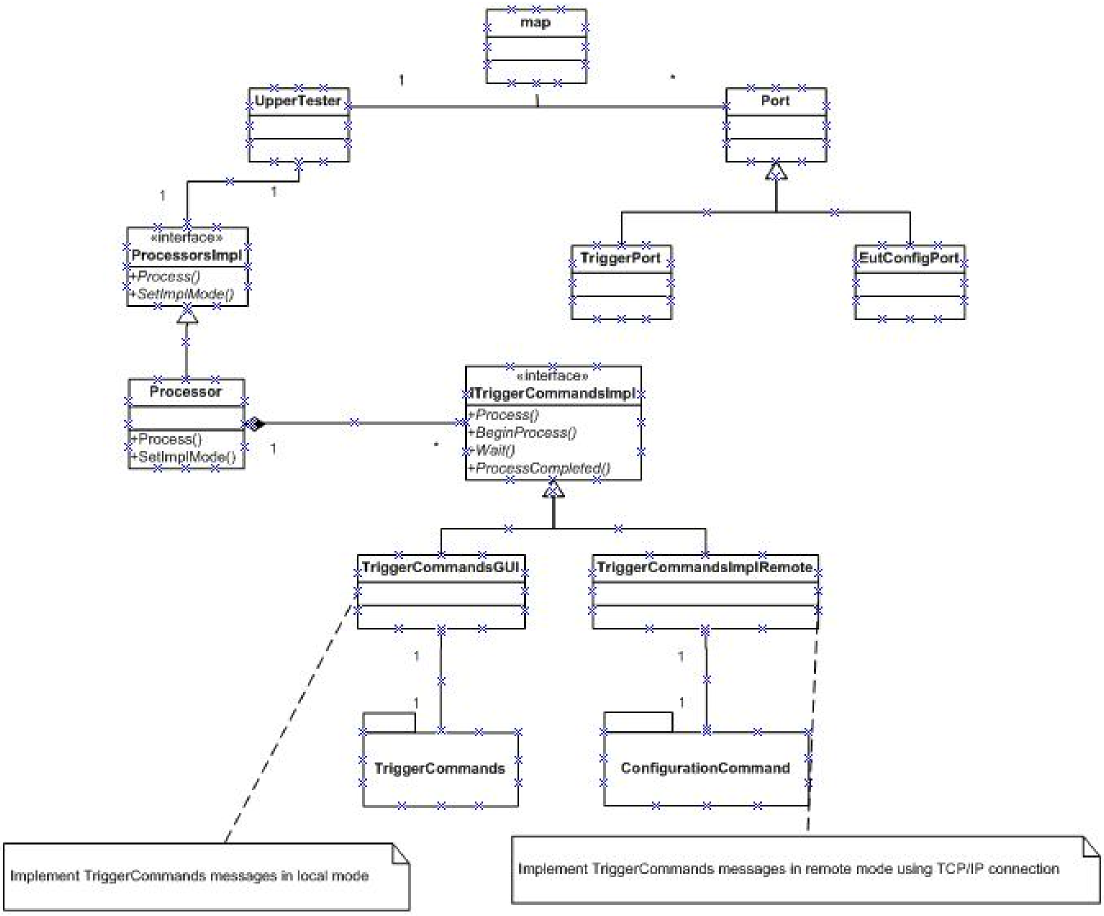


Figure 16: Class diagram of the upper test adapter

10.2.4 Human friendly GUI

To be continued by Yann

10.3 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.

10.3.1 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 doesn’t 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.

10.3.2 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.)

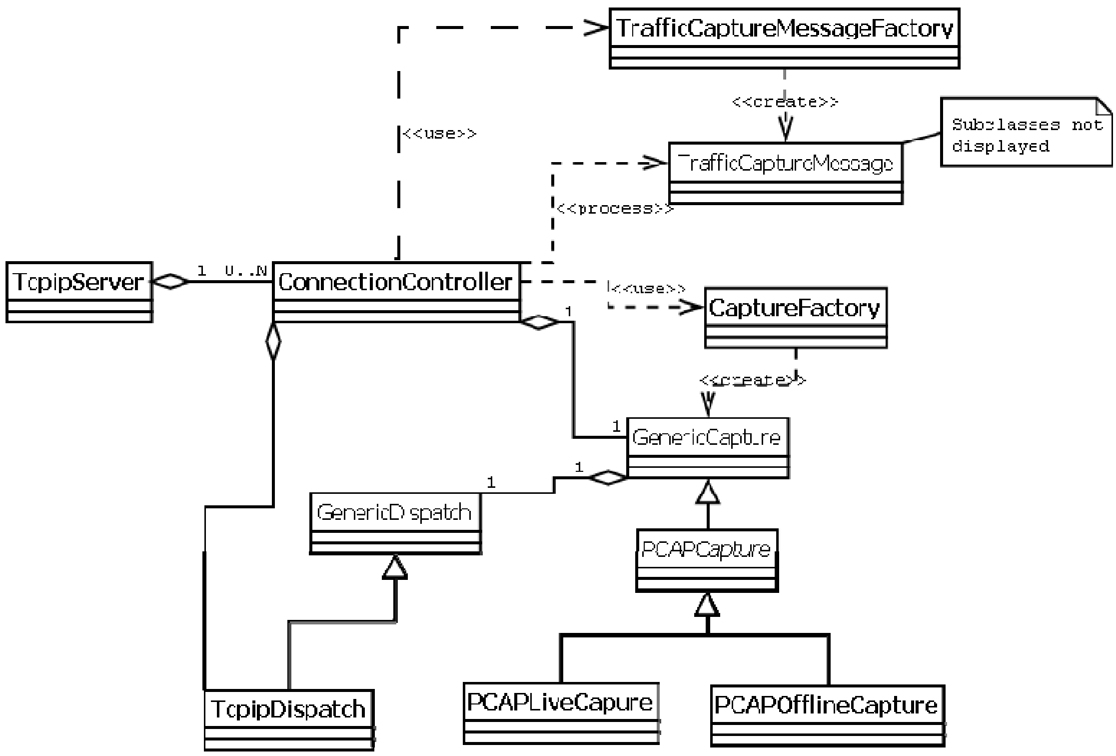


Figure 17: TrafficCapture Class Diagram

10.3.3 Functional Specification

All messages used in the communication with LowerTest are displayed in the Figure 18. This sequence diagram displays a typical scenario for a whole capture session. Individual use cases are described in detail in the following paragraphs.

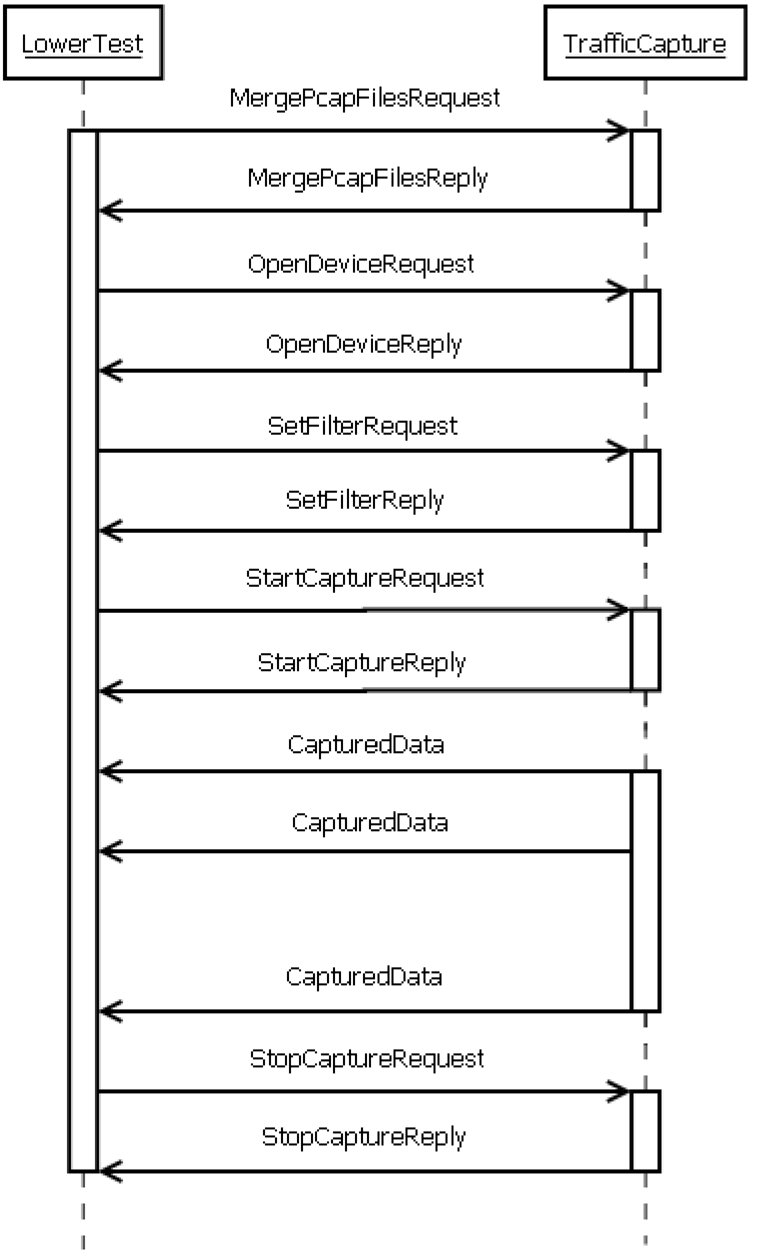
916 

Figure 18: Sequence diagram of typical TrafficCapture session

917 #### 10.3.3.1 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: 1. Invalid path to the mergecap tool

	2. Invalid path to the directory where the merge file should be created 3. Merge file already exists and it is not possible to overwrite it 4. Source files are not found

918

919 #### 10.3.3.2 UC 02: Opening Device

Precondition	TCP/IP connection established In case of offline approach using multiple files, the files must be merged (UC 01: File Merging)
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: 1. Invalid format of parameter describing capturing device (incorrect network adapter, pcap file missing etc.) 2. Pcap driver not installed 3. Invalid/not supported device type requested

920 #### 10.3.3.3 UC 03: Setting Filter

Precondition	Capturing device ready (UC 02: Opening Device)
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 2. Invalid filter format

921 #### 10.3.3.4 UC 04: Starting Capture

Precondition	Capturing device ready (UC 02: Opening Device)
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

922 #### 10.3.3.5 UC 05: Stopping Capture

Precondition	Packet capture started (UC 04: Starting Capture)
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

10.3.4 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.

11 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

Note that this test suite is located in to the directory “STF 370/adapter/validation”. The directory “STF 370/adapter/MM” provides the test suite solution for MMAGIC application.

12 SVN repositories

The adapter sources are archived into STF370 project, at the following location: svn+ssh://vcs.etsi.org/TTCN3/ATS/IMS_IOT/trunk.

Note that the adapter project depends also of t3devkit library, located here: svn+ssh://scm.gforge.inria.fr/svn/t3devkit/t3devkit/branches/stf370/t3devkit

13 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's 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's used by the TrafficCapture component
- WinPcap 4.0.2 developer's pack downloaded from http://www.winpcap.org/devel.htm. It's used by the TrafficCapture component

14 References

[Fwk] ETSI EG xxx xxx: "Methods for Testing and Specification (MTS); Automated Interoperability Testing; Methodology and Framework ".

[IMSarch] ETSI TR 1xx xxx, "Methods for Testing and Specification (MTS); Automated Interoperability Testing; Specific Architectures“.

[Core] ETSI ES 201 873-1 V3.4.1 (2008-09): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language".

[TRI] 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)"

[TCI] 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)"

[Codec] - Codec - Design document_draft.doc

[t3devkit] - Official t3devket framework reference http://www.irisa.fr/tipi/wiki/doku.php/t3devkit

[BOOST] – Official BOOST framework reference http://www.boost.org/doc/libs/1_39_0/libs/libraries.htm

15 Revision History

6 July 2009	0.1	Document creation / Initial draft
8 July 2009	0.2	First internal review
9 July 2009	0.3	Second internal review
22 July 2009	0.4	Document review by Stephan Schulz
29 July 2009	0.6	Major revision by Stephan Schulz