Virtual Serial Device Application Note

80000nt10045a Rev. 0 – 2011-03-03

Making machines talk.

Disclaimer

The information contained in this document is the proprietary information of Telit Communications S.p.A. and its affiliates (“TELIT”).

The contents are confidential and any disclosure to persons other than the officers, employees, agents or subcontractors of the owner or licensee of this document, without the prior written consent of Telit, is strictly prohibited.

Telit makes every effort to ensure the quality of the information it makes available. Notwithstanding the foregoing, Telit does not make any warranty as to the information contained herein, and does not accept any liability for any injury, loss or damage of any kind incurred by use of or reliance upon the information.

Telit disclaims any and all responsibility for the application of the devices characterized in this document, and notes that the application of the device must comply with the safety standards of the applicable country, and where applicable, with the relevant wiring rules.

Telit reserves the right to make modifications, additions and deletions to this document due to typographical errors, inaccurate information, or improvements to programs and/or equipment at any time and without notice.

Such changes will, nevertheless be incorporated into new editions of this document.

APPLICABILITY TABLE

PRODUCT
GT863-PY
GT864-QUAD
GT864-PY
GM862-GPS
GC864-QUAD
GC864-DUAL
GC864-QUAD V2
GC864-DUAL V2
GE863-QUAD
GE863-GPS
GE863-SIM
GE863-PRO^3
GE864-QUAD
GE864-QUAD V2
GE864-DUAL V2
GE864-QUAD Automotive V2
GE864-QUAD Atex
GE865-QUAD
GL865-DUAL
GL865-QUAD

Contents

Figures

Tables

1. Introduction

1.1. Scope

Scope of this document is to provide a guideline showing how can be used together services implemented on TELIT module (e.g.: PYTHON, FOTA, ATRUN, SAT, EVMONI, CMUX) that share hardware/software communications resources in order to configure the TELIT module without resources conflict.

1.2. Audience

This document is intended for User Application designers who want to exploit at best the communication resources offered by the TELIT module without run up against contended resources among services.

1.3. Contact Information, Support

For general contact, technical support, to report documentation errors and to order manuals, contact Telit's Technical Support Center (TTSC) at:

TS-EMEA@telit.com
TS-NORTHAMERICA@telit.com
TS-LATINAMERICA@telit.com
TS-APAC@telit.com

Alternatively, use:

http://www.telit.com/en/products/technical-support-center/contact.php

For detailed information about where you can buy the Telit modules or for recommendations on accessories and components visit:

http://www.telit.com

To register for product news and announcements or for product questions contact Telit's Technical Support Center (TTSC).

Our aim is to make this guide as helpful as possible. Keep us informed of your comments and suggestions for improvements.

Telit appreciates feedback from the users of our information.

1.4. Related Documents

[1] Telit CMUX User Guide, 30268ST10299A
[2] Telit AT Commands Reference, 80000ST10025a
[3] Telit Modules Software User Guide, 1vv0300784
[4] Telit Easy Script Python, 80000ST10020a
[5] Telit AT Run and Event Monitor Services App. Note, 80000NT10043a
[6] Telit Event Monitor Application Note, 80000NT10028a
[7] Telit Running AT Remotely Application Note, 80000NT10029a
[8] Telit SIM Toolkit AT Application Note, 80000NT10030A
[9] Telit PFM Application Note, 80000NT10013a

1.5. Document History

Revision	Date	Changes
0	2011-03-03	First issue
/	/	/

1.6. Abbreviations and acronyms

CSD	Circuit Switched Data
DTE	Data Terminal Equipment
FOTA	Firmware Over The Air
NVM	Non Volatile Memory
PDP	Packet Data Protocol
PPP	Point to Point Protocol
RLP	Radio Link Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
VSD	Virtual Service Device

2. Virtual Serial Device

Virtual Serial Device, hereafter called VSD, is a piece of software designed to run on TELIT modules. It basically manages virtual connections among the physical serial ports, accessible to the user, and the services running on the module. To accomplish this activity, VSD supports several Access Points used as anchorage points for the logical connections. The following lists show the items involved in the connections management: Physical Serial Ports, Logical Access Points, AT Parser and Trace Utility, Services and Protocols. The VSD supports several configurations of these items that will be explained on this document.

Physical Serial Ports	Logical Access Points	AT Parsers, Trace Utility	Services	Protocols
ASC0	AT0	Instance #1	CMUX	CSD
ASC1	AT1	Instance #2	Python	TCP/IP
	AT2	Instance #3	SMS AT Run	Dial Up
	TRACE	Trace	Event Monitor	
	VHWDTE0		TCP AT Run	
	VHWDTE1		SAT	
	PYSER		FOTA	
	PYSER2			
	IPE GSM			
	VHWDTEGSM			
	VHWDTESMSAT			
	VHWDTETCPAT			
	VHWDTESAT			
	VHWDTEOTA			

Tab. 1: Services & other Items

In order to avoid resource conflicts when two or more Services are used on the module at the same time, it is suggest to start from a well defined configuration and from it select the Services that can work together without conflicts. Example of conflicts: two Services try to use the same access point or an AT command tries to “break” the already established connection between a physical port and an access point used by a Service.

It is advisable to recall the concept of instances, in this context, and their relationships with the Access Points: with the term “instance” is intended an AT Commands Parser: TELIT modules provide three logically independent AT Commands Parsers. Any instance matches an Access Point as showed on the following pages.

2.1. VSD Configuration at Power ON

Let's to start with the VSD configuration at the module power ON, see fig. 1. In this configuration VSD connects the physical port ASC0 to the AT0 AT command parser; AT0 parser is matching the instance # 1. Two more AT parser instances are provided by the module: instance # 2 and # 3. The user, by means of DTE equipment, enters AT commands; they are parsed by the ATO parser and executed by the module. VDS also connects physical port ASC1 to the Trace utility. The user, by means of DTE equipment running TELIT RTD application, sees the trace log. In this configuration the CMUX Standard Protocol [1] is not still used.

ⓘ Use AT+IPR command to set the ACS0 serial port speed. It supports the hardware flow control. At power ON, ACS0 serial port speed is factory setting as autobauding and the ASC1 serial port is used for debugging purposes, it doesn't support flow control.

Use the following AT commands to verify some connections.

AT#SELINT? check if AT Command interface is SELINT = 2
#SELINT: 2
OK

AT#SII?
#SII: 0 AT0 parser is active and connected to ASC0, see fig. 1
OK

AT#SII=1
OK AT0 parser is still active and connected to ASC0. Trace Service is disconnected from ASC1; AT1 parser (instance #2) is active and connected to ASC1, see fig. 2

ⓘ Set the right speed on ACS1 port. It does not use any flow control.

Yet, in this configuration, a generic User Application running on a user device, equipped with two physical serial ports, can send AT commands toward two AT parsers (instance #1, and instance# 2) at the same time.

If AT#SII=2 command is entered, AT2 parser (instance #2) will be use, see
fig. 3

Yet, let's suppose to enter the command:

AT#SII=0
OK the VDS assume again the configuration showed on
fig. 1.

Tab. 2 summarizes the AT#SII command behavior starting from a well defined VSD configuration in order to avoid possible resources conflicts with unpredictable results: power ON the module, see fig. 1, enter the AT commands one after another as indicated on the table and check the relating configuration.

Legend:

"ASCx" : physical serial port connected to the Access Point indicated on the column top;
"X" : Access Point unserviceable.

Power ON / AT#SII	VSD Access Points			
	AT0 Instance #1	AT1 Instance #2	AT2 Instance #3	Trace
Power ON	ASC0	X	X	ASC1
AT#SII=1	ASC0	ASC1	X	X
AT#SII=2	ASC0	X	ASC1	X
AT#SII=0	ASC0	X	X	ASC1

Tab. 2: AT#SII vs. Access Points

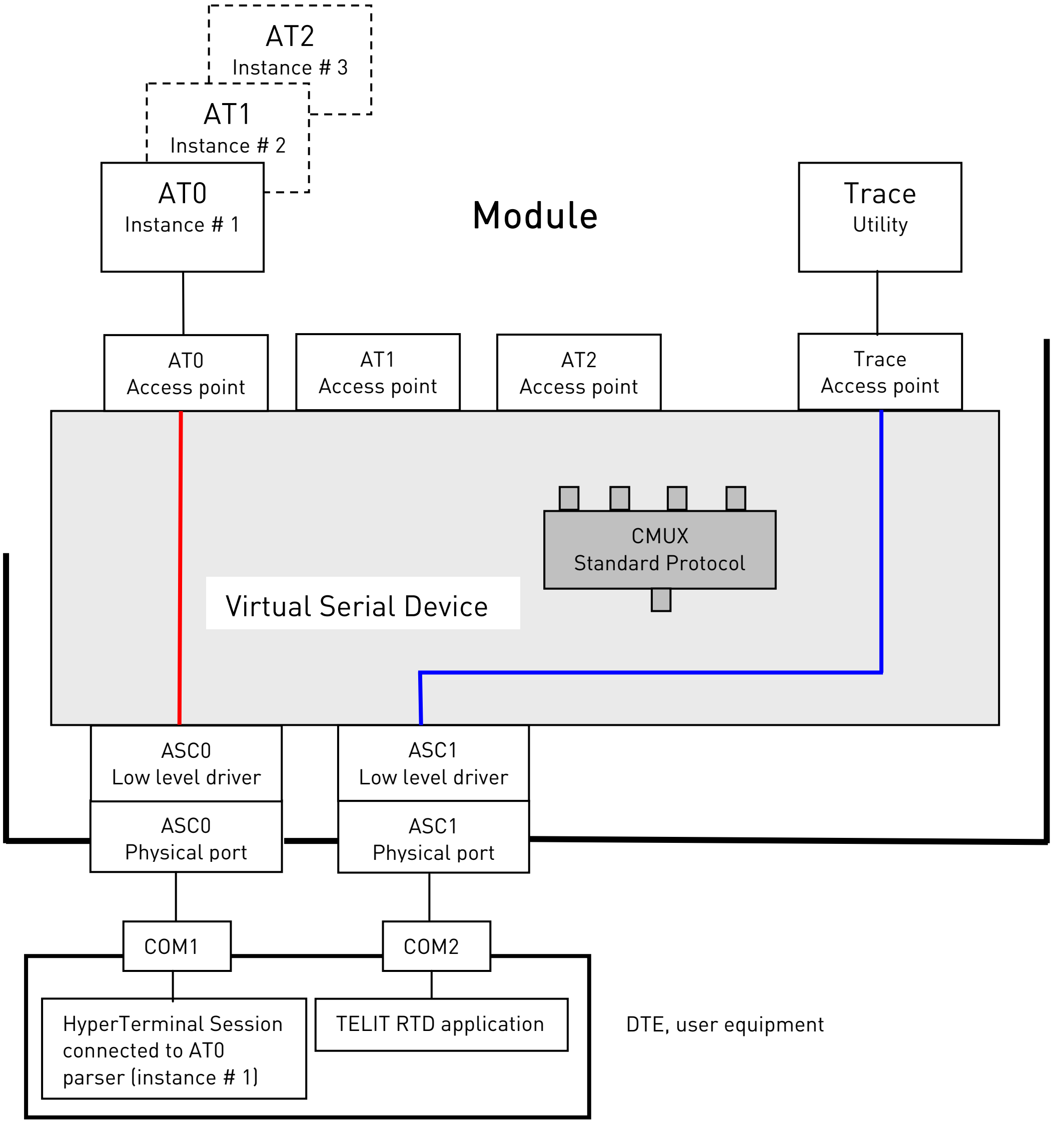

fig. 1: VSD configuration at Power ON

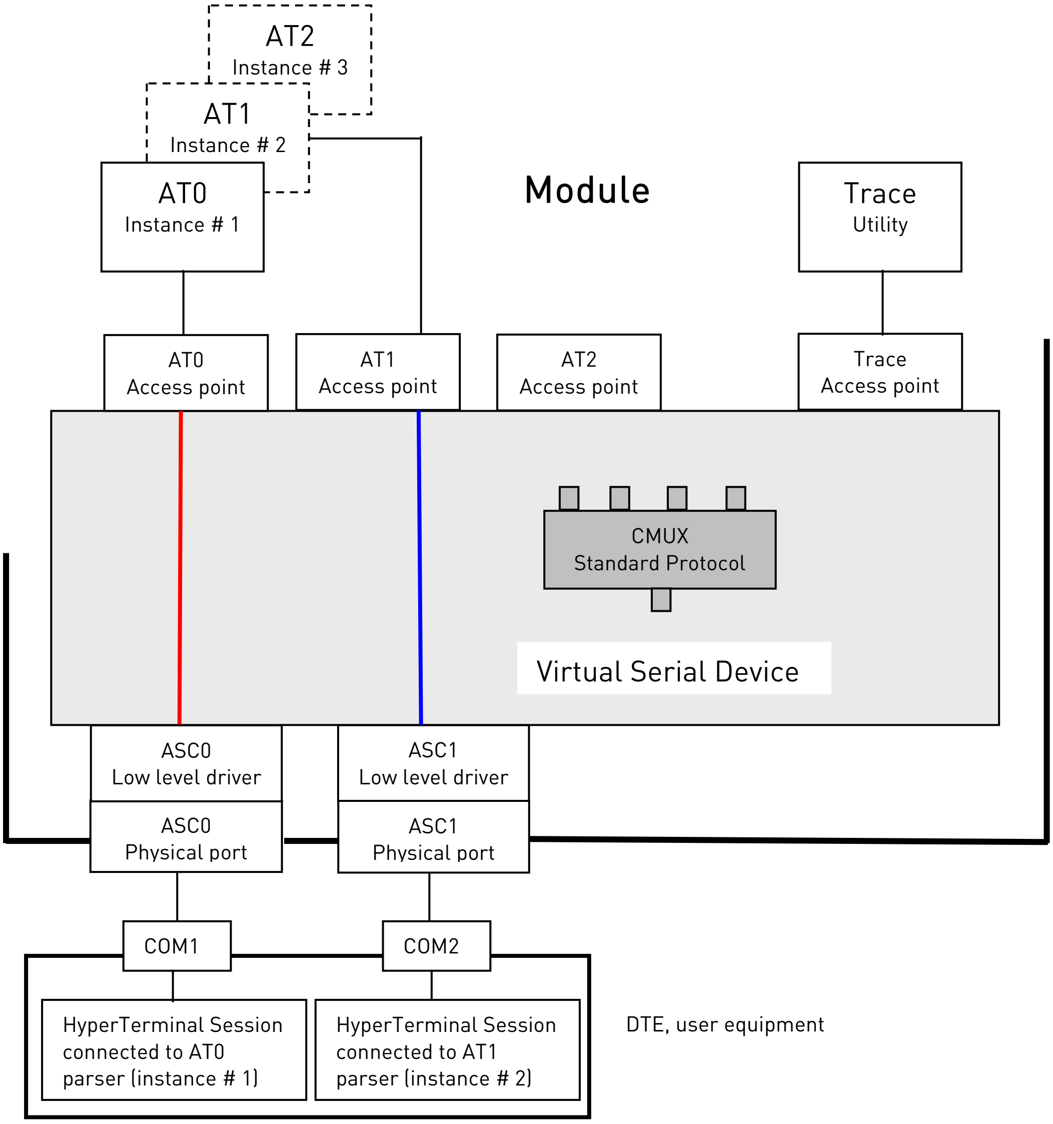

fig. 2: VDS configuration after AT#SII=1 command

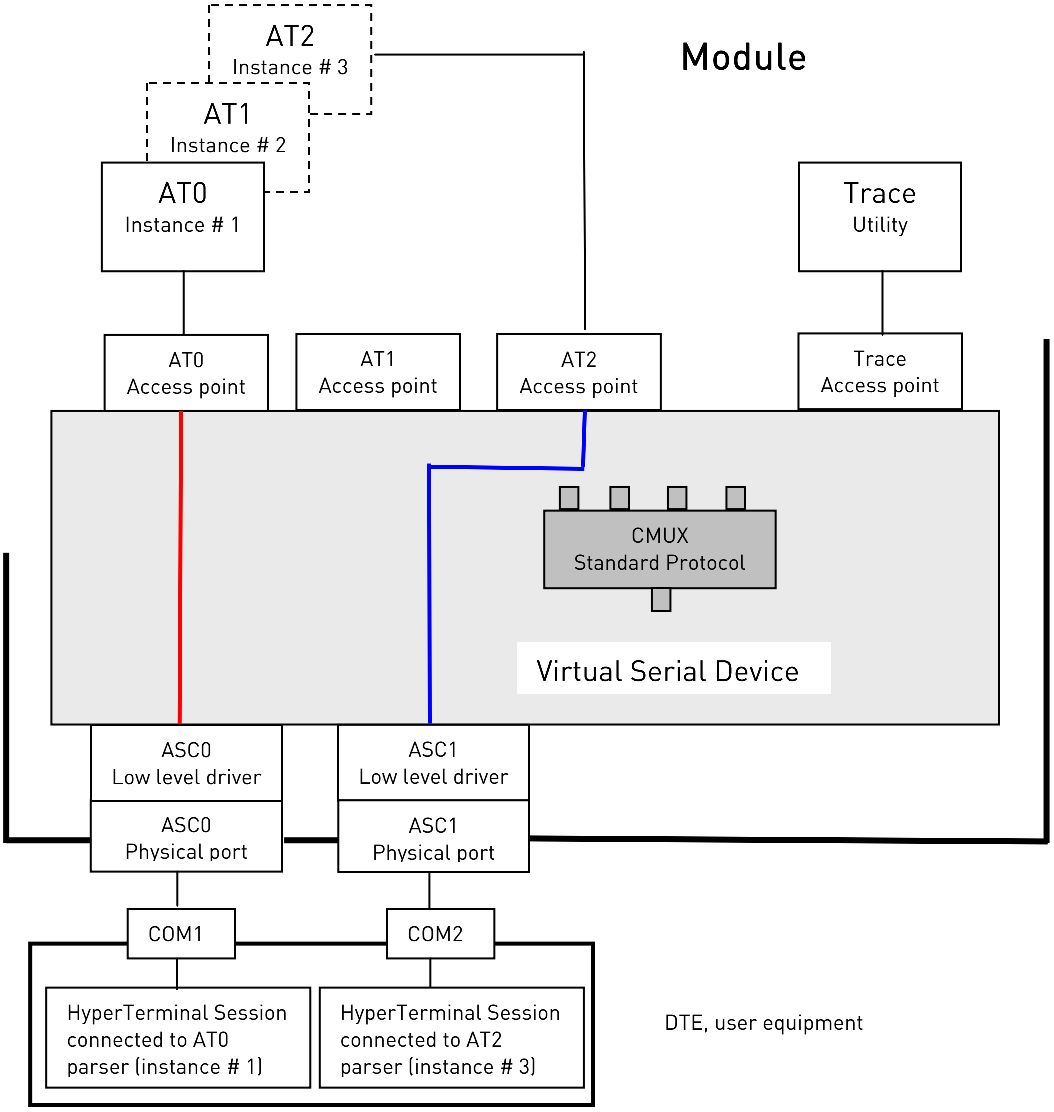

fig. 3: VDS configuration after AT#SII=2 command

2.2. CMUX Standard Protocol Service

The CMUX Standard Protocol Service [1], provided by the module, can be used by the user when he connects to the module equipment running an application using a protocol that can be automatically recognized by the module as a CMUX Standard protocol. To introduce the argument let's suppose to run on the equipment (DTE) the TELIT Serial Port MUX application [1] as showed by fig. 4. TELIT Serial Port MUX is a tool used to verify the protocol and show the serial connection capability implemented on the TELIT module.

A way to force the module from the configuration showed on fig. 1 to the configuration showed on fig. 4, without the use of the TELIT Serial Port MUX tool, is to enter the command AT+CMUX=0 [2]. After the execution of this command, the ASCO port expects on its line, for a defined time interval, the presence of the CMUX Standard Protocol running on the connected device.

⚠ When the CMUX Standard Protocol is used, don't enter AT+SII=0, 1, 2 command. In fact, AT#SII command steals CMUX's resource and CMUX Service steals AT#SII command's resource, serial lines malfunction will arise. To make virtual connections work again it is needed to disconnect/connect the user application (e.g.: Hyper Terminal sessions) from the MUX tool running on DTE.

Tab. 3 summarizes the CMUX behavior starting from a well defined VSD configuration in order to avoid possible resources conflicts with unpredictable results: power ON the module, see fig. 1, start CMUX.

Legend:

"ASC0/VCx": Virtual Connection that must be used to reach the Access Point indicated on the column top. The user can select one or more Access Points.

CMUX	VSD Access Points			
	AT0 Instance #1	AT1 Instance #2	AT2 Instance #3	Trace
CMUX ON	ASC0/VC1	ASC0/VC2	ASC0/VC3	ASC0/VC4

Tab. 3: CMUX vs. Access Points

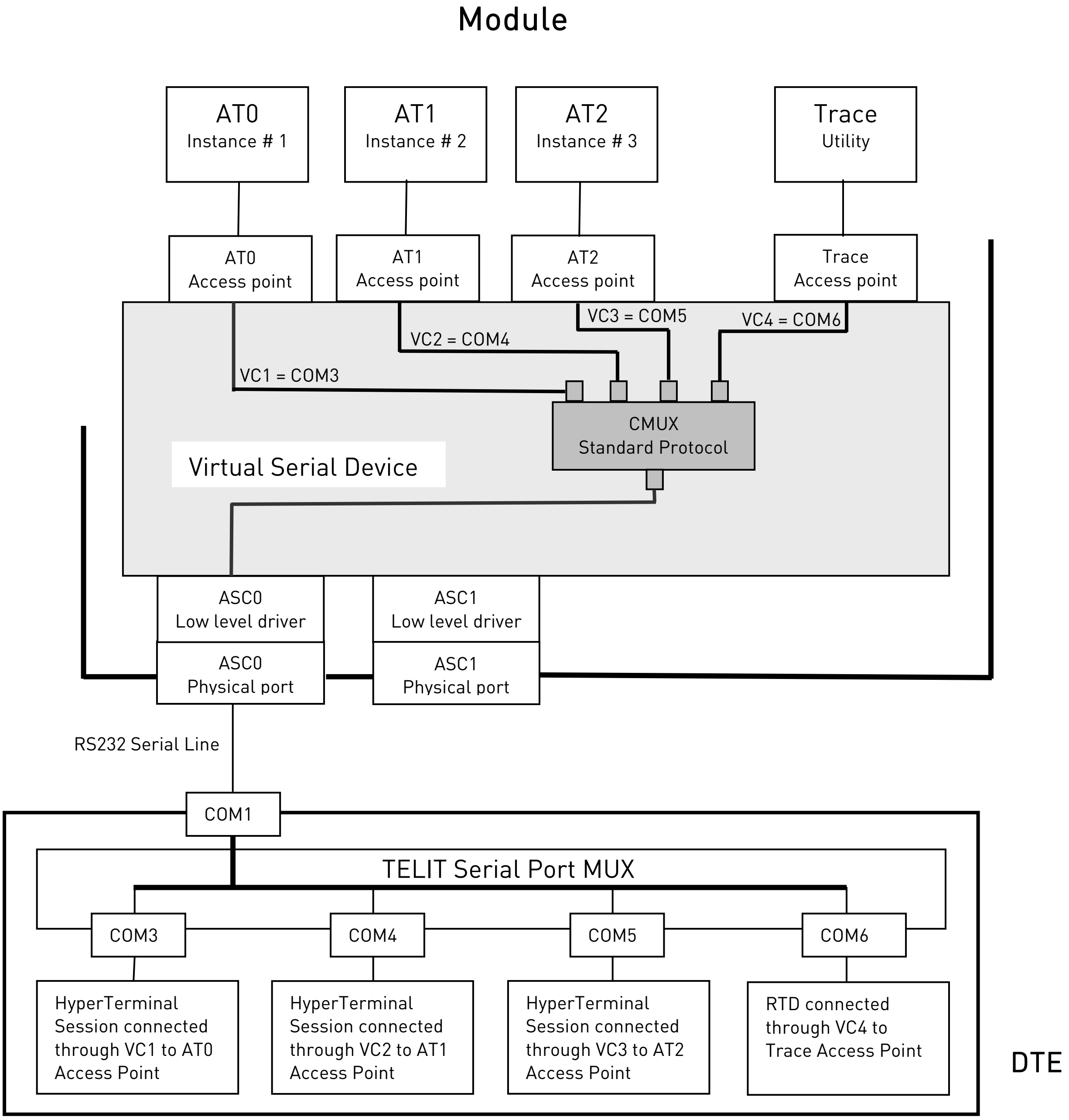

fig. 4: CMUX configuration

2.3. Python Service

Python programming language is provided by the TELIT module in order to offer to the user a tool to develop control scripts using the communication and hardware resources provided by the module. As showed on fig. 5 the VSD provides two access points called VHW DTE0 and VHW DTE1. MDM and MDM2 Python modules are logically connected respectively to VHW DTE0 and VHW DTE1 access points.
When the Python script runs the Python instruction ***import MDM***, the VSD disconnect the ASC0/AT0 logical connection and establishes the logical connection VHW DTE0/AT0, consequently the Python script can access AT0 parser. In the same way, ***import MDM2*** instruction forces the VSD to establish the logical connection VHW DTE1/AT1. From fig. 5 it is possible infer that ASC0 is disconnected and unutilized from external module side.

Python script can run another Python software module to use the ASCO port using the instruction ***import SER***. The fig. 6 shows the new connection: through the physical port ASC0 it is possible to be connected with the Python script.

The three Python software modules (MDM, MDM2 and SER) make use of three independent resources (ASC0, AT0 and AT1 Access Point), no resources contention can arise among them.

⚠

Refer to fig. 7: let's suppose that the user enters the command AT# STARTMODESCR = 2. After executing this command, the physical serial port ASC0 is logically connected to the access point AT2, consequently the Python script couldn't execute the instruction import SER, see paragraph 3. We are in presence of a resource conflict between the AT command and the Python import instruction.

ASC1 physical port supports the Trace Utility connection. By means of the TELIT RTD application it is possible to interpret and display the trace massages during debugging session. Moreover, by means of a Hyper Terminal it is possible to display the Python script ***print*** instructions to test the Python script; the ***print*** instructions are not packaged as RTD messages trace.

Python script can run one more software module to use the ASC1 port using the instruction ***import SER2***. The fig. 8 shows the new connection: through the physical port ASC1 it is possible to be connected with the Python script.

Tab. 4 summarizes the Python *import* instructions behavior starting from a well defined VSD configuration in order to avoid possible resources conflicts with unpredictable results: power ON the module, see fig. 1, execute the *import* instructions one after another as indicated on the table and check the relating configuration.

Legend:

“Python” : service acquires the Access Point indicated on the column top;
“/” : service does not use the Access Point. Access Point stays on its original status;
“ASCx” : physical serial port is connected to the Access Point.
“X” : Access Point is disconnected;

Python *import* instruction	VSD Access Points					
	AT0 Instance #1	AT1 Instance #2	AT2 Instance #3	PYSER	PYSER2	Trace
MDM	Python	/	/	/	/	/
MDM2	/	Python	/	/	/	/
SER	X	/	/	ASC0	/	/
SER2	/	/	/	/	ASC1	X

Tab. 4: Python vs. Access Points

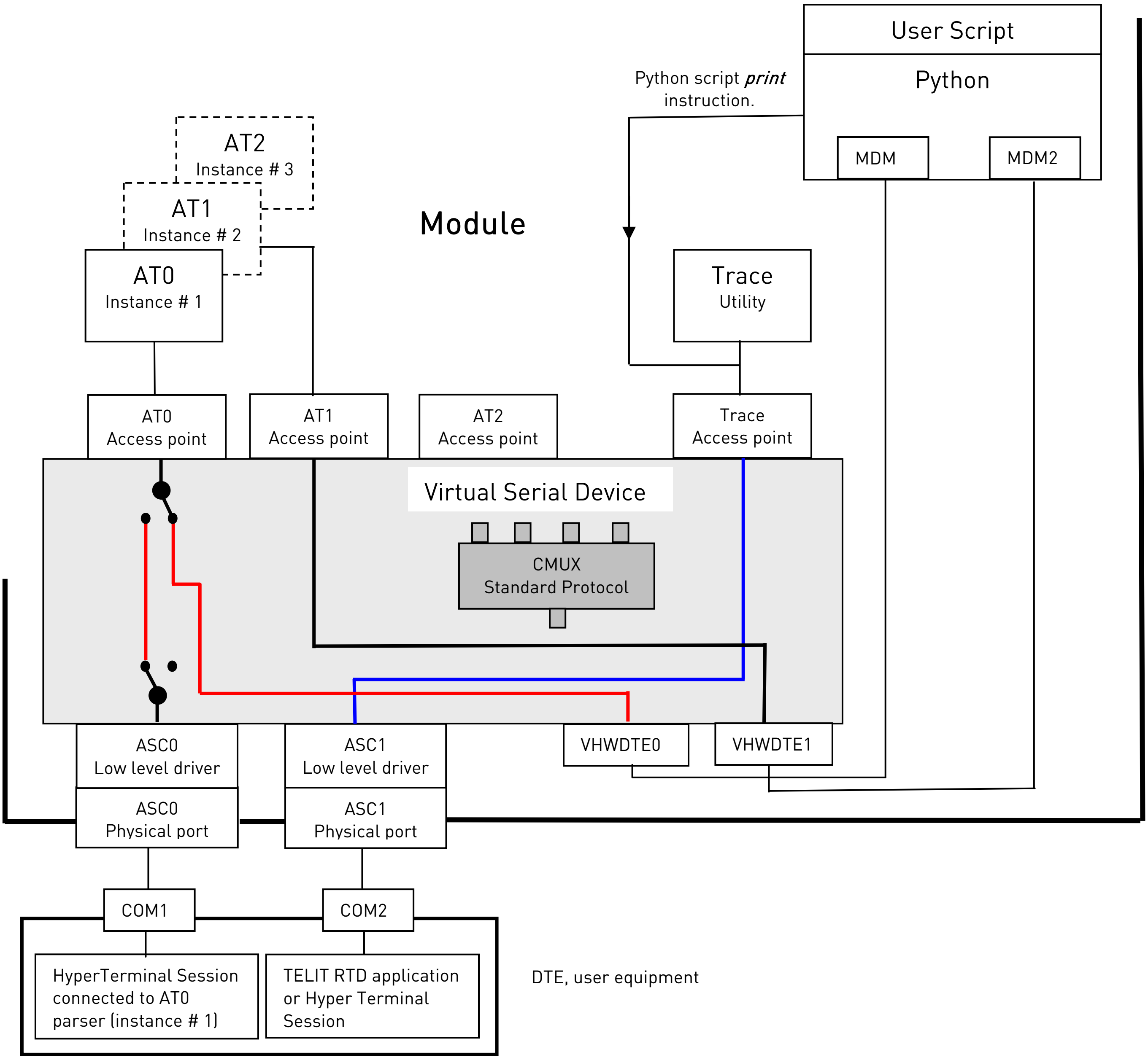

fig. 5: Python & MDM, MDM2 modules

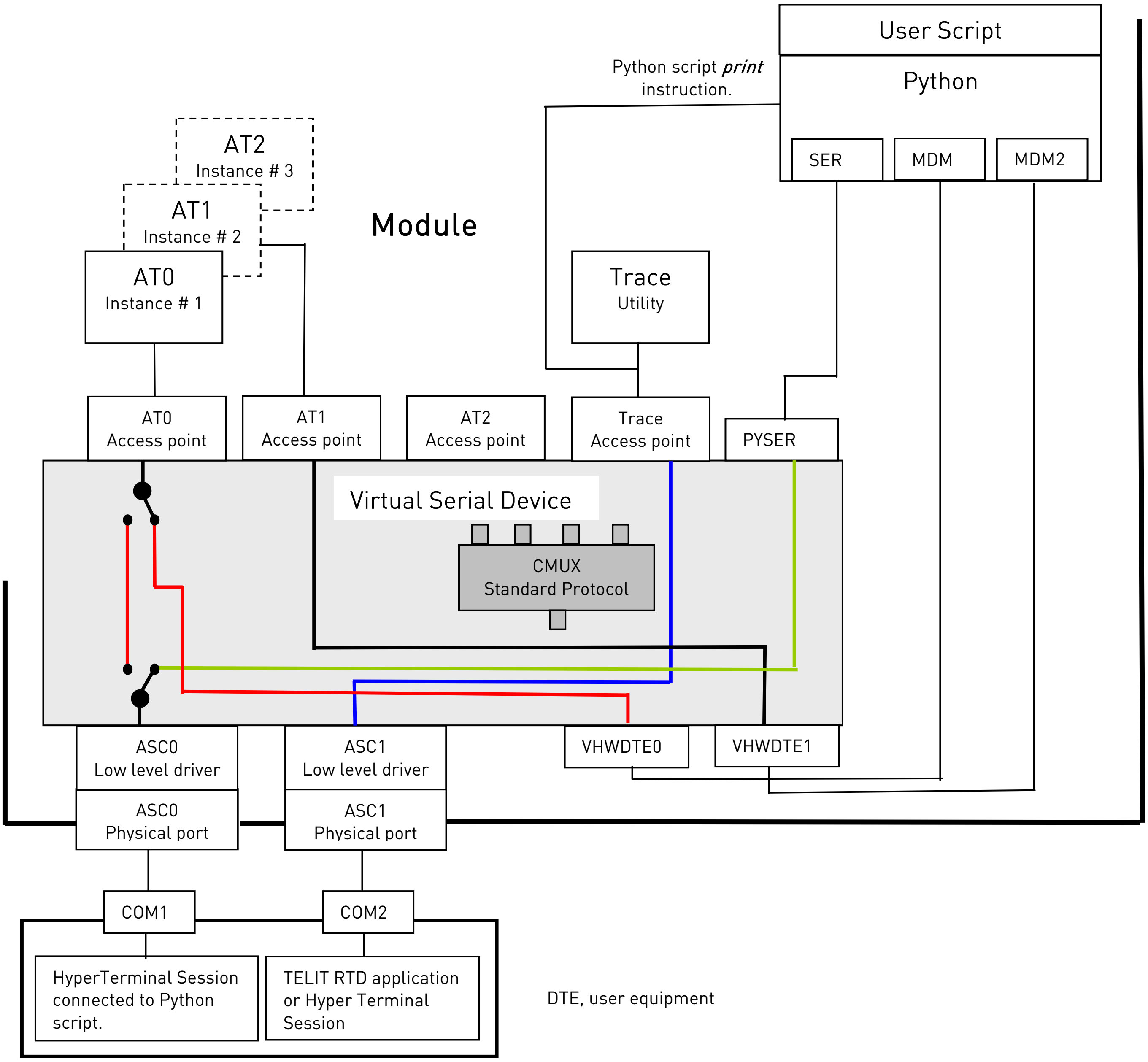

fig. 6: Python & MDM, MDM2, SER modules

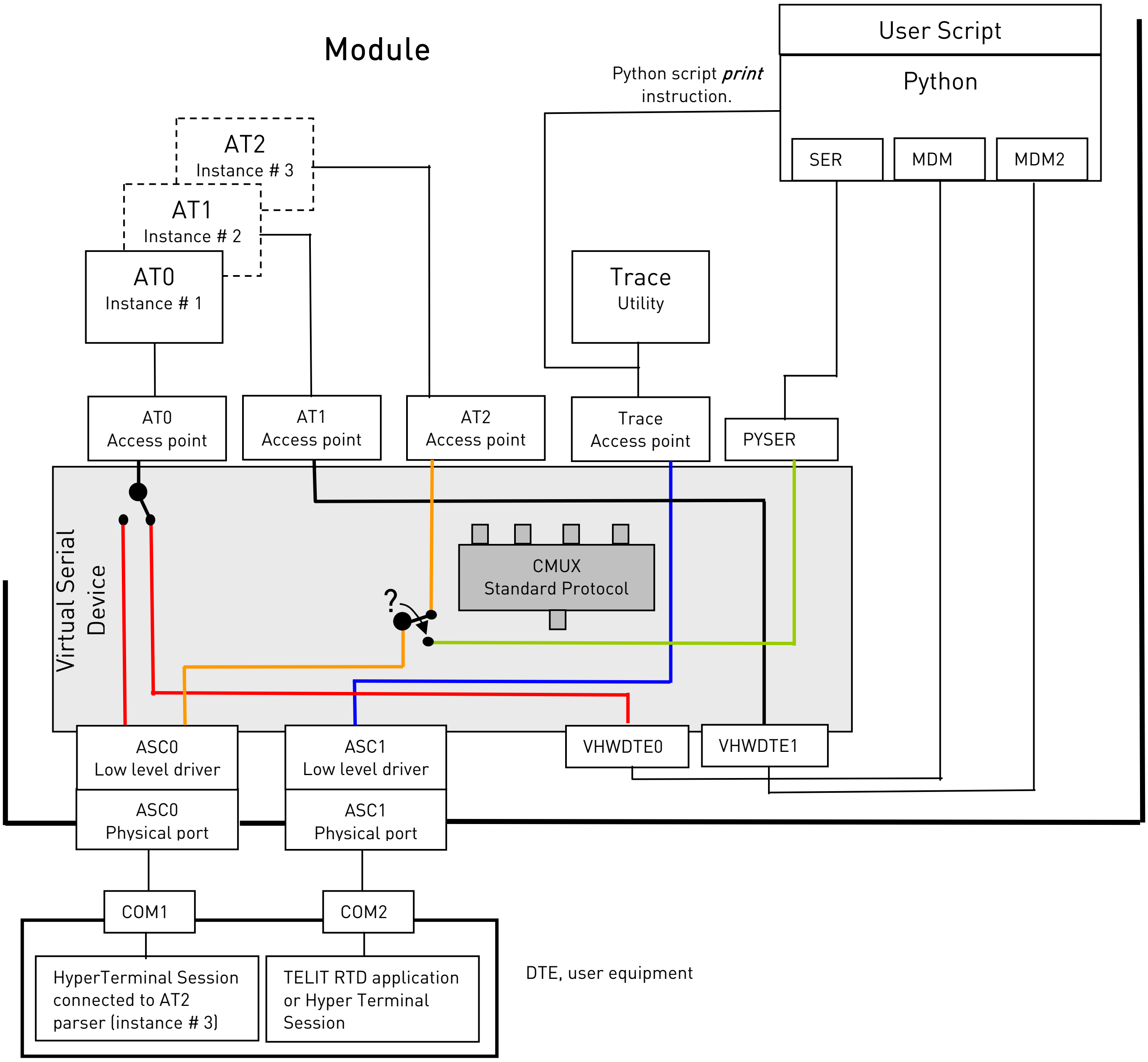

fig. 7: Python & AT#STARTMODESCR=2 conflict

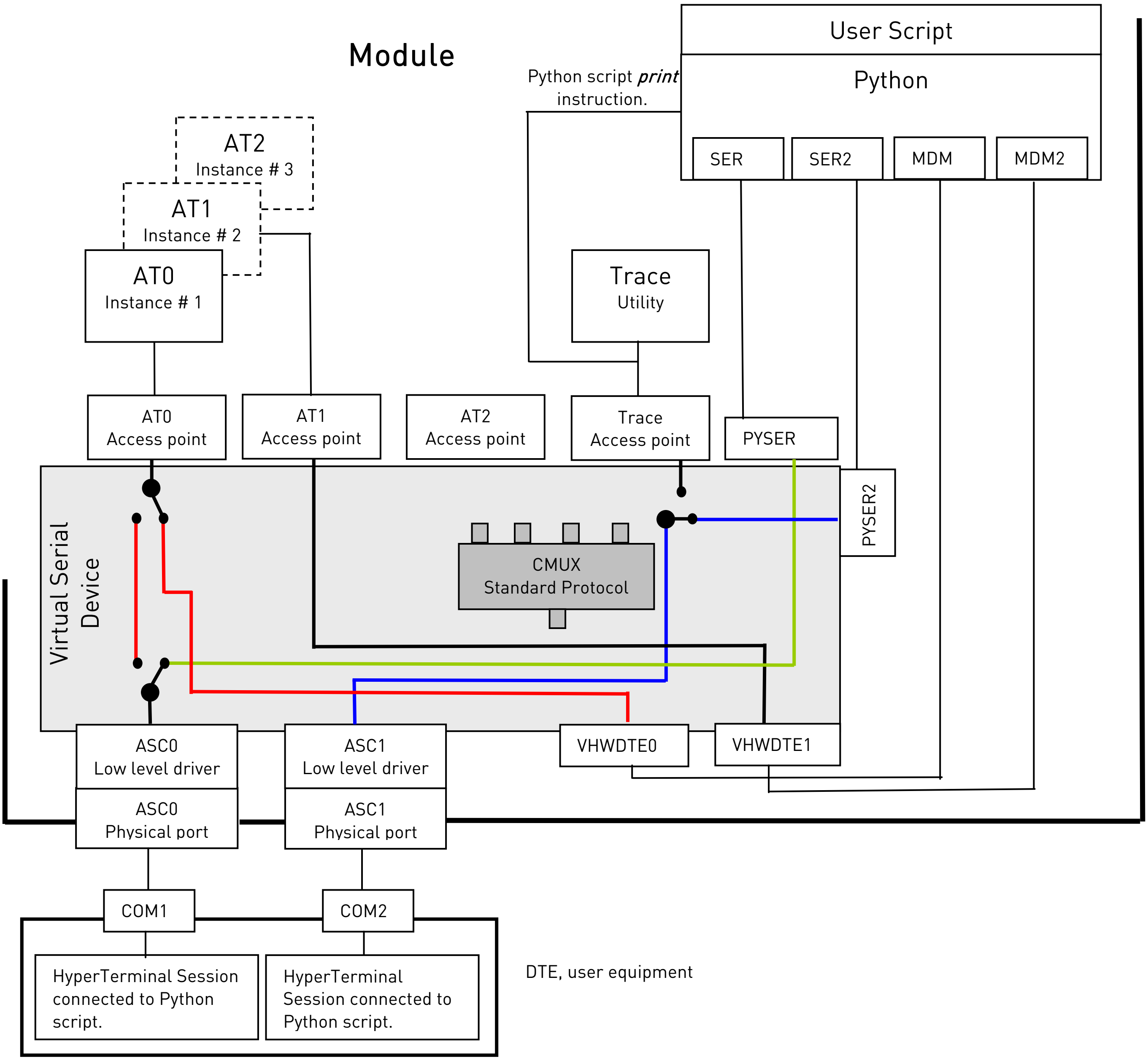

fig. 8: Python & MDM, MDM2, SER, SER2 modules

2.3.1. Python Service vs. GPS

TELIT products having the GPS module use the ASC1 port to connect the GSM side to the GPS side, consequently the ASC1 physical port is not more available to support the Trace Utility and the ***print*** instructions coming from Python Script. The CMUX service gives us the possibility to skip over the lack of serial port for the Trace Utility, see [4]. In order to describe the logic connection configurations adopted by the VSD to manage this occurrence, it is useful to consider two scenarios: LAB and Remote.

LAB Scenario

In LAB Scenario it is assumed that Trace Utility is needed for module and Python Script maintenance activities. To carry out these "debugging" actions a serial port is needed to access the Trace Utility and the ***print*** instructions of the Python Script. To create the physical access, follow these steps.

- Enter the command AT#CMUXSCR=1, save it on the NVM and power OFF the module;

- Power ON the module. The module checks the DTR control line of ASC0 serial port. If the DTR is asserted, it means that the connected DTE is ready to operate; in this case the module is controlled, for example, by a Hyper Terminal application. If DTR is not asserted the Python Script is started, it checks CMUXSCR, if its value is 1 CMUX service is started (AT+CMUX=0 command is simulated), the configuration is showed on fig. 9.

- After the initial CMUX configuration is established, the Python Script runs the following instructions if they are present, see fig. 10:
    - *import MDM* instruction, the connection 1 is broken and the connection 1a is created;
    - *import MDM2* instruction, the connection 2 is broken and the connection 2a is created;
    - *import SER* instruction, the connection 3a is created. No priority between *import MDM* and *import SER* instructions is required.

Tab. 5 summarizes the Python *import* instructions behavior starting from a well defined VSD configuration in order to avoid possible resources conflicts with unpredictable results: CMUX is installed, see fig. 9, execute the *import* instructions one after another as indicated on the table and check the relating configuration.

Legend:

“Python” : service acquires the Access Point indicated on the column top;
“ASCx” : physical serial port is connected to the Access Point.
“/” : service does not use the Access Point. Access Point stays on its original status;

Python *import* instruction	VSD Access Points				
	AT0 Instance #1	AT1 Instance #2	AT2 Instance #3	PYSER	Trace
MDM	Python	/	/	/	/
MDM2	/	Python	/	/	/
SER	/	/	/	ASC0/VC1	/

Tab. 5: Python vs. Access Points through CMUX

Another LAB solution

- It is assumed that factory setting uses AT#CMUXSCR=0, power ON the module;

- DTR of ASC0 serial port is asserted it means that the connected DTE is ready to operate; the Python Script is not automatically started;

- Enter AT+CMUX=0 to start the CMUX, fig. 9;

- Enter AT#EXECSRC to start the Python Script. It is suggested to use ASCO serial port and AT2 AT parser (instance # 3), this connection path remains always available, see fig. 10.

Remote Scenario

Remote Scenario identifies two configurations:

- Module connected to a DTE;
- Module in Stand Alone configuration.

In Remote Scenario it is assumed that no Trace Utility is needed for module and Python Script maintenance activities. In accordance with this observation, no serial port is required to access the Trace Utility and the *print* instructions of the Python Script; consequently CMUX Standard Protocol is not needed. To disable CMUX follow these steps:

- Enter the command AT#CMUXSCR=0, save it on the NVM and power OFF the module:

- Power ON the module. The module checks the DTR control line of ASC0 serial port. If the DTR is asserted, it means that the connected DTE is ready to operate; the Python Script is not started, it is responsibility of the DTE application to manage the module. If DTR is not asserted the script is started, it checks CMUXSCR, if its value is 0 CMUX service is not started.

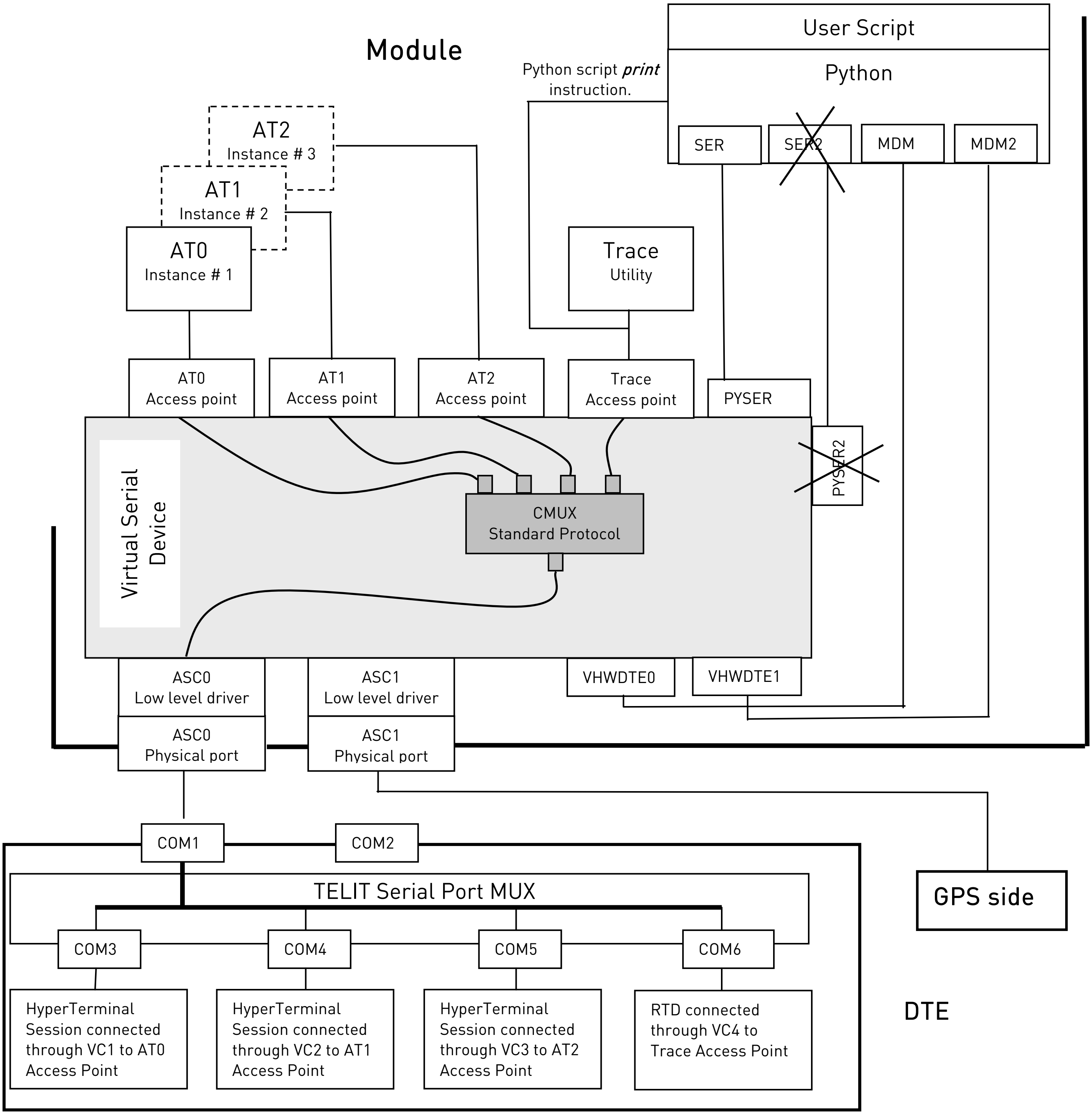

fig. 9: Initial CMUX configuration

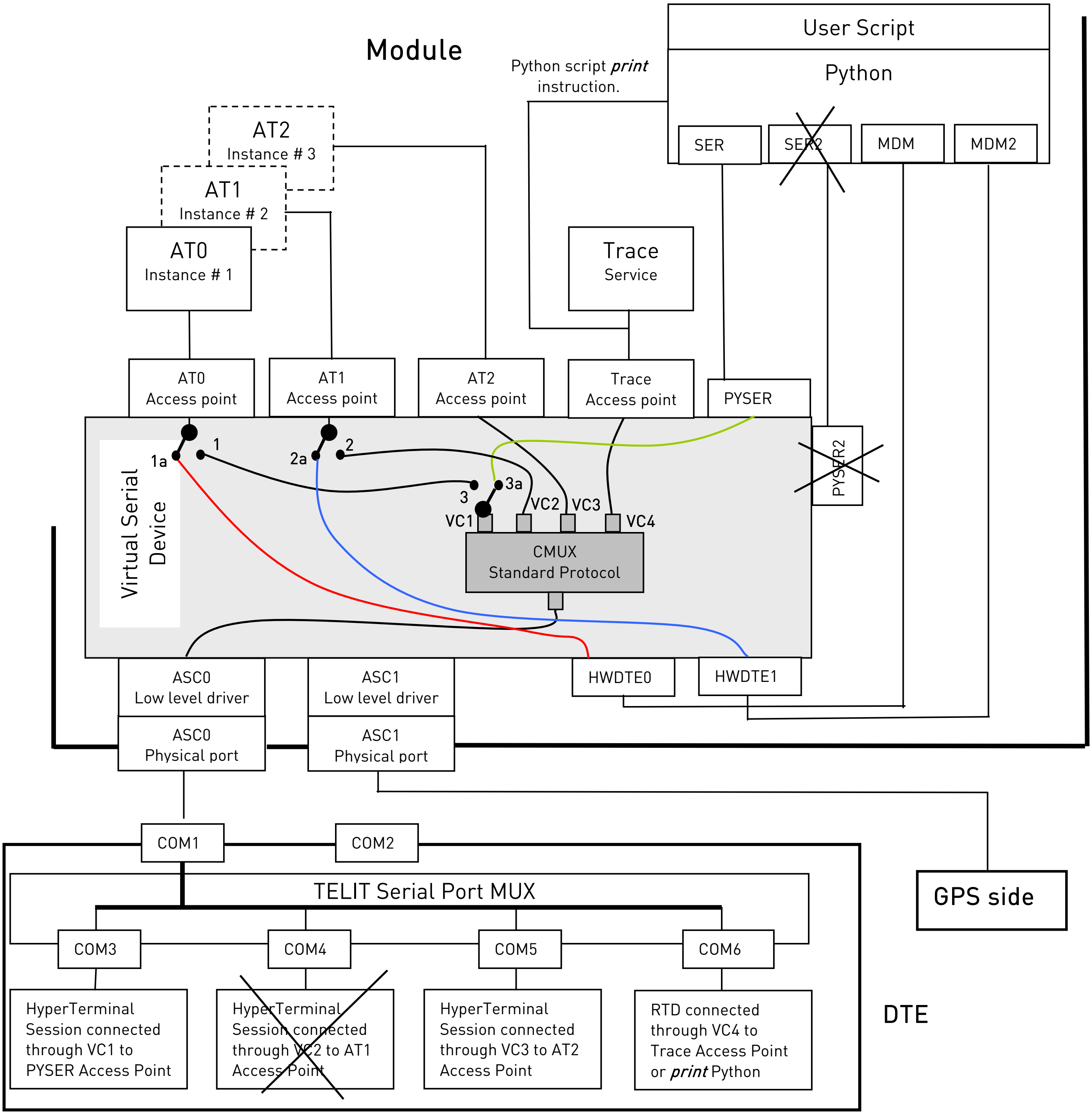

fig. 10: CMUX configuration and Python

2.4. CSD Protocol

Let's start with the VSD configuration at the module power ON, see fig. 1. In this configuration VSD connects the physical port ASC0 to the AT0 AT command parser through the AT0 Access Point; AT0 parser matches Instance # 1. The user, by means of DTE equipment, enters AT commands; they are parsed by the AT0 parser and executed by the module. VDS also connects physical port ASC1 to the Trace Utility through the Trace Access Point. The user can see trace logs if DTE runs the TELIT RTD application. In this configuration the CMUX Standard Protocol [1] is not used. To use the CSD protocol the following steps must be followed, refer to fig. 11:

- Enter the ATD<…> command to arrange the CSD connection. The logical connection 1 is active, the local module is in COMMAND mode;

- When the remote module responds successfully to the data calling, on locale DTE is displayed the CONNECT message. At this time the logical connection 1 is disconnected and the logical connection 1a is activated. The local and remote module can exchange data; they are in ON LINE mode. Any character that the user enters on DTE is sent to the remote module. To exit ON LINE mode and enter again COMMAND mode, the user must enter the escape sequence +++.

Tab. 6 summarizes the ATD<…> command behavior starting from a well defined VSD configuration in order to avoid possible resources conflicts with unpredictable results: power ON the module, see fig. 1, enter the ATD<…> instruction and check the relating configuration.

Legend:

"CSD/ASC0" : CSD service acquires the Access Point by means of ACS0 physical port;
"/" : service does not use the Access Point. Access Point stays on its original status;

ATD<…> command	VSD Access Points			
	AT0 Instance #1	AT1 Instance #2	AT2 Instance #3	Trace
ATD<…>	CSD/ASC0	/	/	/

Tab. 6: CSD vs. Access Points

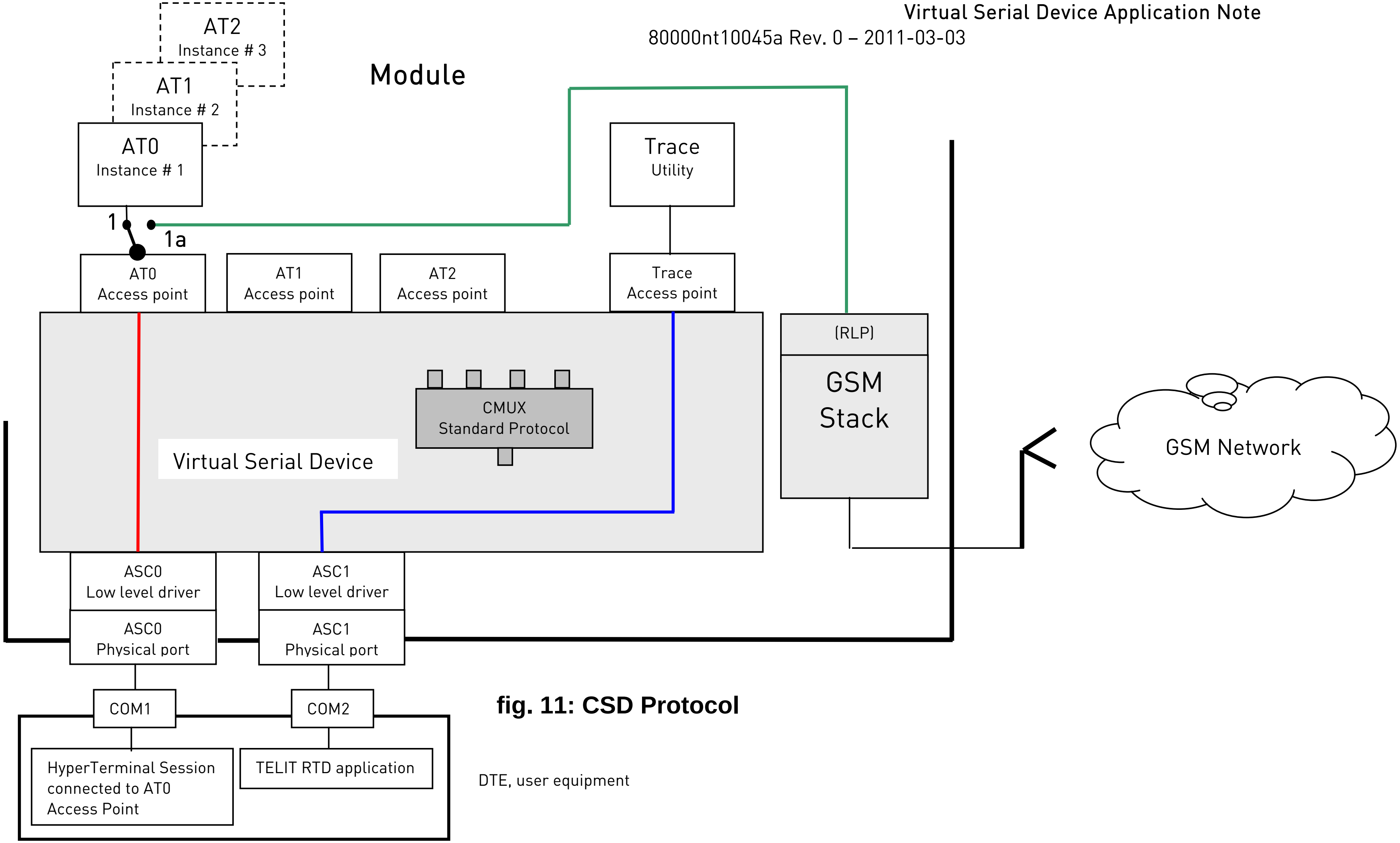

fig. 11: CSD Protocol

2.4.1. CSD and CMUX Service

To activate CMUX refer to paragraph 2.2. Yet, let's suppose to have activated CMUX and need to establish a CSD connection toward a remote module, follow these steps, refer to fig. 12:

- Select one of the three COMx: for example COM4 connected to the AT1 Access Point through Virtual Connection VC2 created by CMUX;

- Enter the ATD<...> command to establish the CSD data connection. The logical connection 1 is active, the local module is in COMMAND mode;

- When the remote module responds successfully to the data calling, on locale DTE is displayed the CONNECT message. At this time the logical connection 1 is disconnected and the logical connection 1a is activated. The local and remote module can exchange data; they are in ON LINE mode. Any character that the user enters on DTE is sent to the remote module. To exit ON LINE mode and enter again COMMAND mode, the user must enter the escape sequence +++.

⚠ Four communication serial lines are active by means of CMUX Service and can work together at the same time, but only one CSD connection at time can be established.

Tab. 7 summarizes the use of the CMUX starting from a well defined VSD configuration in order to avoid possible resources conflicts with unpredictable results: start CMUX, fig. 4, enter ATD<...> instruction by means of ASC0 on VC2 in order to create a CSD connection, then enter AT commands on VC1, and VC3 .

<u>Legend:</u>

“CSD/VCx” : CSD acquires the Access Point indicated on the column top;
“AT/VCx” : generic AT command acquires the Access Point;
“/” : Access Point stays on its original status;

ATD<...> commands	VSD Access Points			
	AT0 Instance #1	AT1 Instance #2	AT2 Instance #3	Trace
ATD<...>	AT/VC1	/	/	/
AT ...	/	CSD/VC2	/	/
AT...	/	/	AT/VC3	/

Tab. 7: CSD vs. Access Points through CMUX

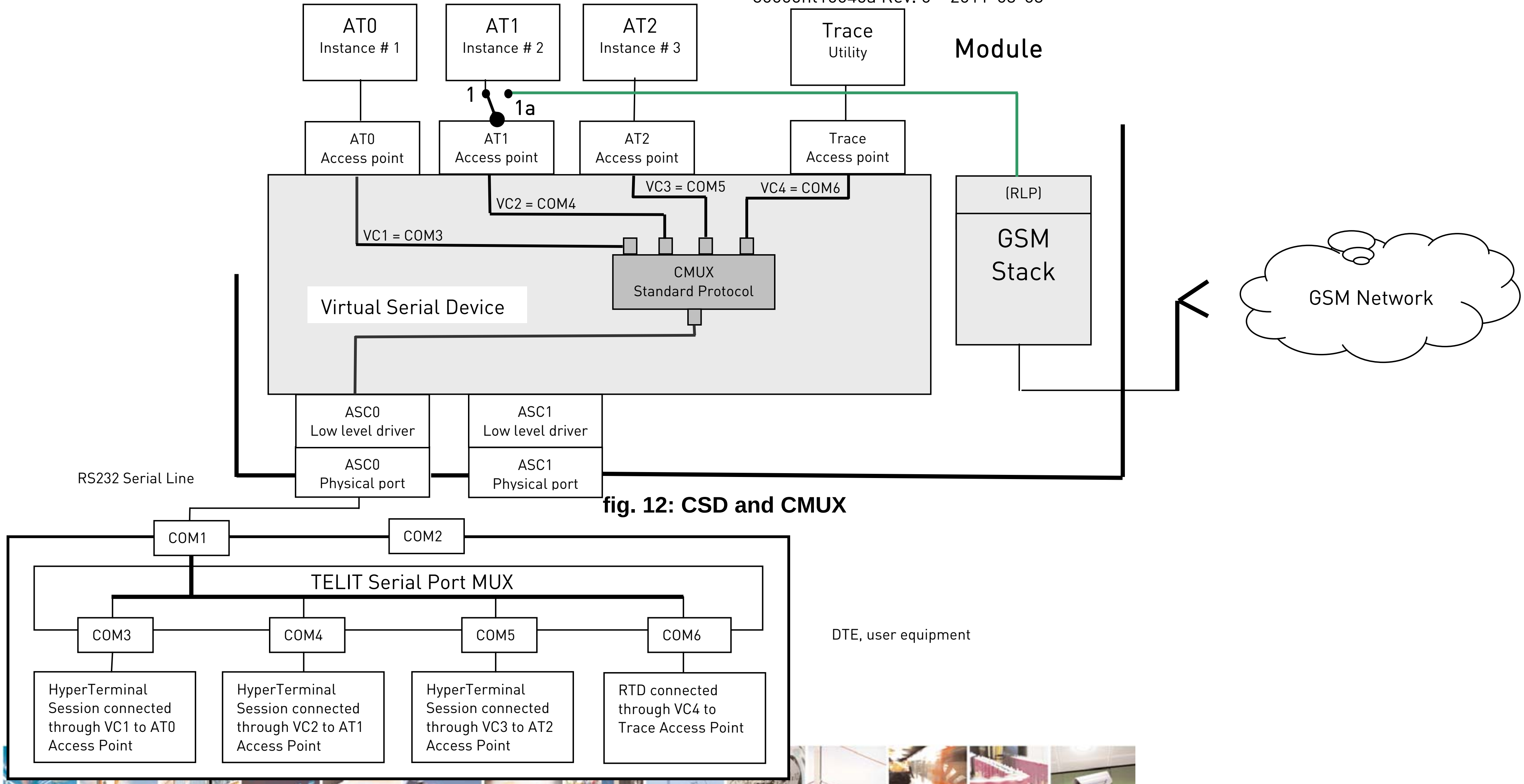

fig. 12: CSD and CMUX

2.4.2. CSD and Python Service

To activate Python refer to paragraph 2.3. With the activation of Python, fig. 13, logical connection 1 is broken and logical connection 1a is established. Let's suppose to establish a CSD connection, managed by the Python Script, toward a remote module. The script must follow these steps:

- The Python script runs the ATD<…> command to arrange the CSD connection. The connection 2 is active, the local module is in COMMAND mode;

- When the remote module successfully responds to the data calling, the script receives the CONNECT message. At this time the logical connection 2 is disconnected and the logical connection 2a is established. The local and remote module can exchange data; they are in ON LINE mode. The Python Script can send and receive characters to/from the remote module. To exit ON LINE mode and enter again COMMAND mode, the script must run the escape sequence **+++**.

⚠ ASC0 physical port is disconnected.

Tab. 8 summarizes the Python ***import*** instructions behavior starting from a well defined VSD configuration in order to avoid possible resource conflicts with unpredictable results: power ON the module, see fig. 1, execute the ***import*** instructions one after another as indicated on the table and check the relating configuration.

<u>Legend:</u>

"Python" : service acquires the Access Point indicated on the column top;
"/" : service does not use the Access Point. Access Point stays on its original status.

Python *import* instruction	VSD Access Points			
	AT0 Instance #1	AT1 Instance #2	AT2 Instance #3	Trace
MDM[1]	CSD/Python	/	/	/
MDM2	/	Python	/	/

Tab. 8: CSD and Python

[1] ASC0 is unserviceable.

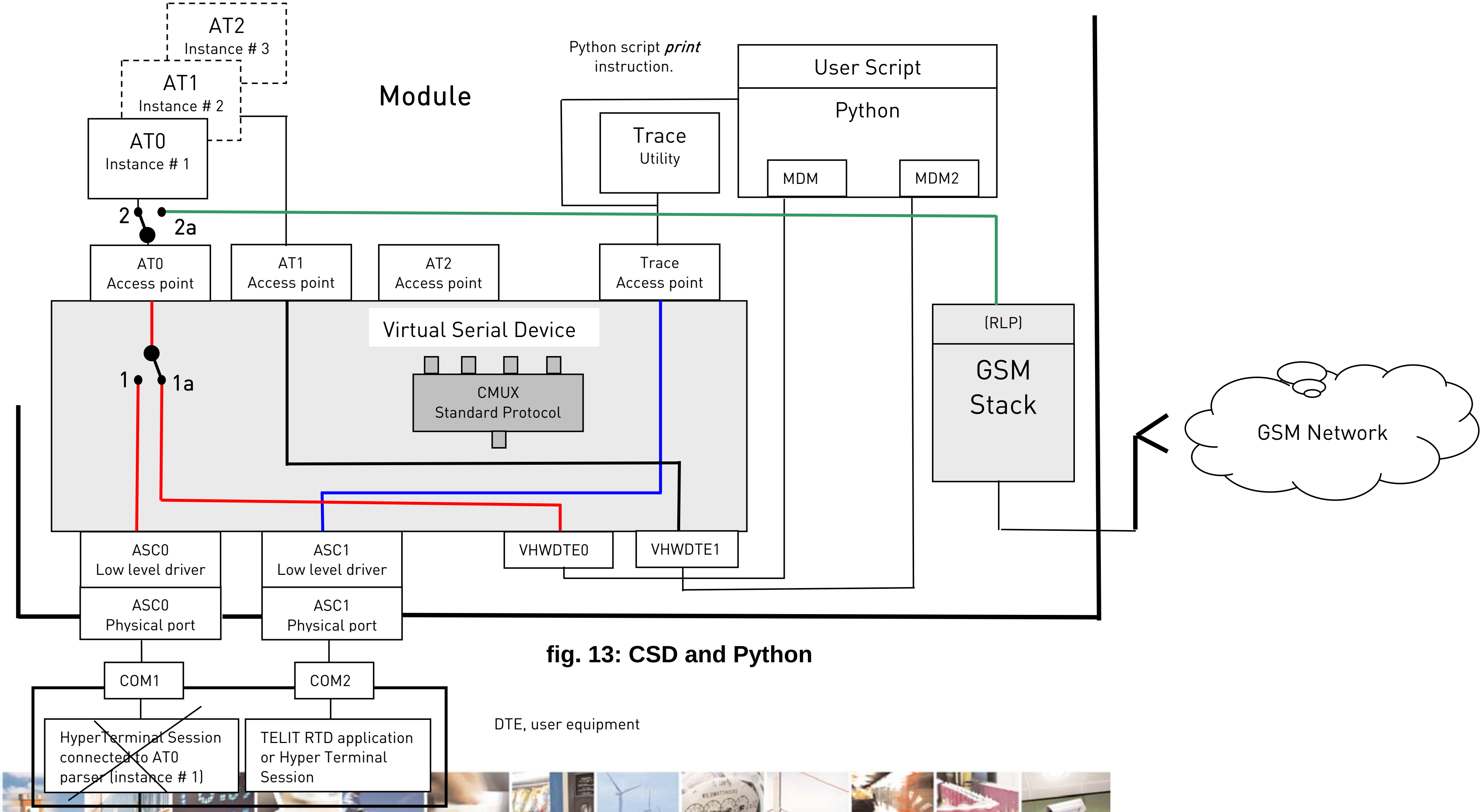

fig. 13: CSD and Python

2.5. TCPI/IP Protocol

2.5.1. TCP/IP on GPRS

Let's start with the VSD configuration at the module power ON, see fig. 1. In this configuration VSD connects the physical port ASC0 to the AT0 AT command parser through the AT0 Access Point; AT0 parser matches Instance # 1. The user, by means of DTE equipment, enters AT commands; they are parsed by the ATO parser and executed by the module. VDS also connects physical port ASC1 to the Trace Utility through the Trace Access Point. The user can see trace logs if DTE runs the TELIT RTD application. In this configuration the CMUX Standard Protocol [1] is not used. To use the TCP/IP protocol the steps showed hereupon must be followed, refer to fig. 14:

Enter:

- AT#SGACT=1,1 command to activate the already configured PDP context on GPRS network[2];

- AT#SD command to arrange the TCP/IP connection using the selected socket identifier. The logical connection 1 is active, the local module is in COMMAND mode.

When the remote module successfully responds to the open connection,

- on locale DTE is displayed the CONNECT message. At this time the logical connection 1 is disconnected and the connection 1a is activated: the local and remote module can exchange data; they are in ON LINE mode. Every character that the user enters on DTE is sent to the remote module. To exit ON LINE mode and enter again COMMAND mode, the user must enter the escape sequence +++, the TCP/IP connection remains "alive". To get again the connection, the user enters the command AT#SO indicating the socket identifier.

The above steps let's understand that the AT0 access point, at different times, can be connected to a maximum of six TCP/IP connections, all alive at the same time. ASC0 physical serial port is connected to AT0 access point.

AT#FTPOPEN command can be used to open an FTP connection toward an FTP server.

[2] Use AT#SGACT= 0,1 to use GSM network.

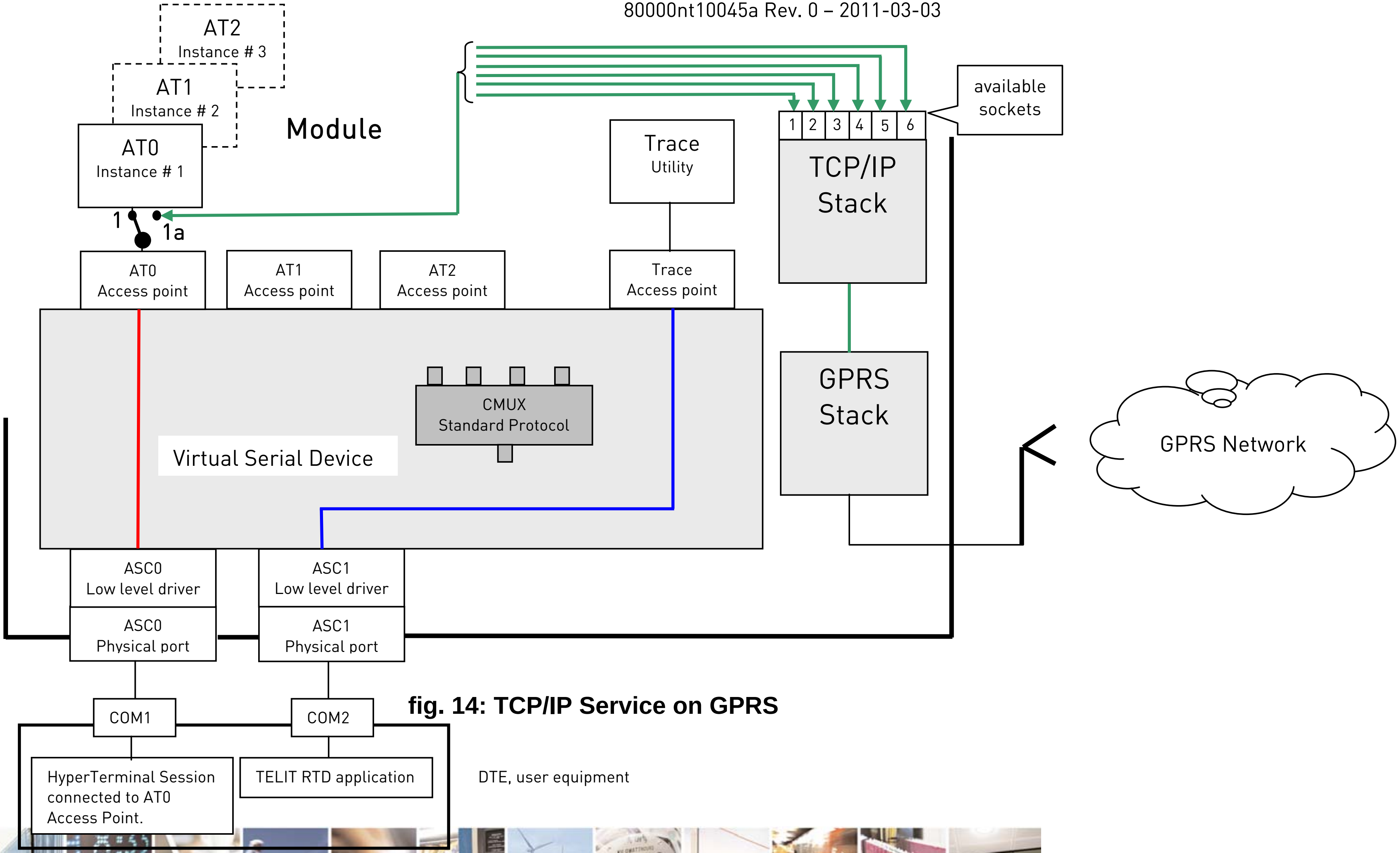

fig. 14: TCP/IP Service on GPRS

2.5.2. TCP/IP on Dial Up

To use the TCP/IP protocol through a dial up connection the following steps must be performed:

- Configure the PDP context in accordance with the SIM that is installed on the module. To carry out the configuration use, for example, an Hyper Terminal connected to the ASC0 serial port, the AT0 access point is connected to the instance #0 (1), see fig. 15. When the configuration is done, disconnect the Hyper Terminal from COM1;

- Use the tools provided by the Operating System (e.g.: Windows, Create a new Connection) to set up on the PC a dial up connection using the Standard 33600 bps Modem. During the dial up configuration process enter as phone number the string: *99#. When the connection is done, the AT0 Access Point is connected (1a) to the PPP stack;

- Yet, a user application can use the available sockets.

ℹ

In order to test the connections run an Internet Browser.

It is worth remind that in this configuration, the TCP/IP stack protocol is not running on TELIT module, but it is running on the PC as showed by fig. 15, fig. 16.

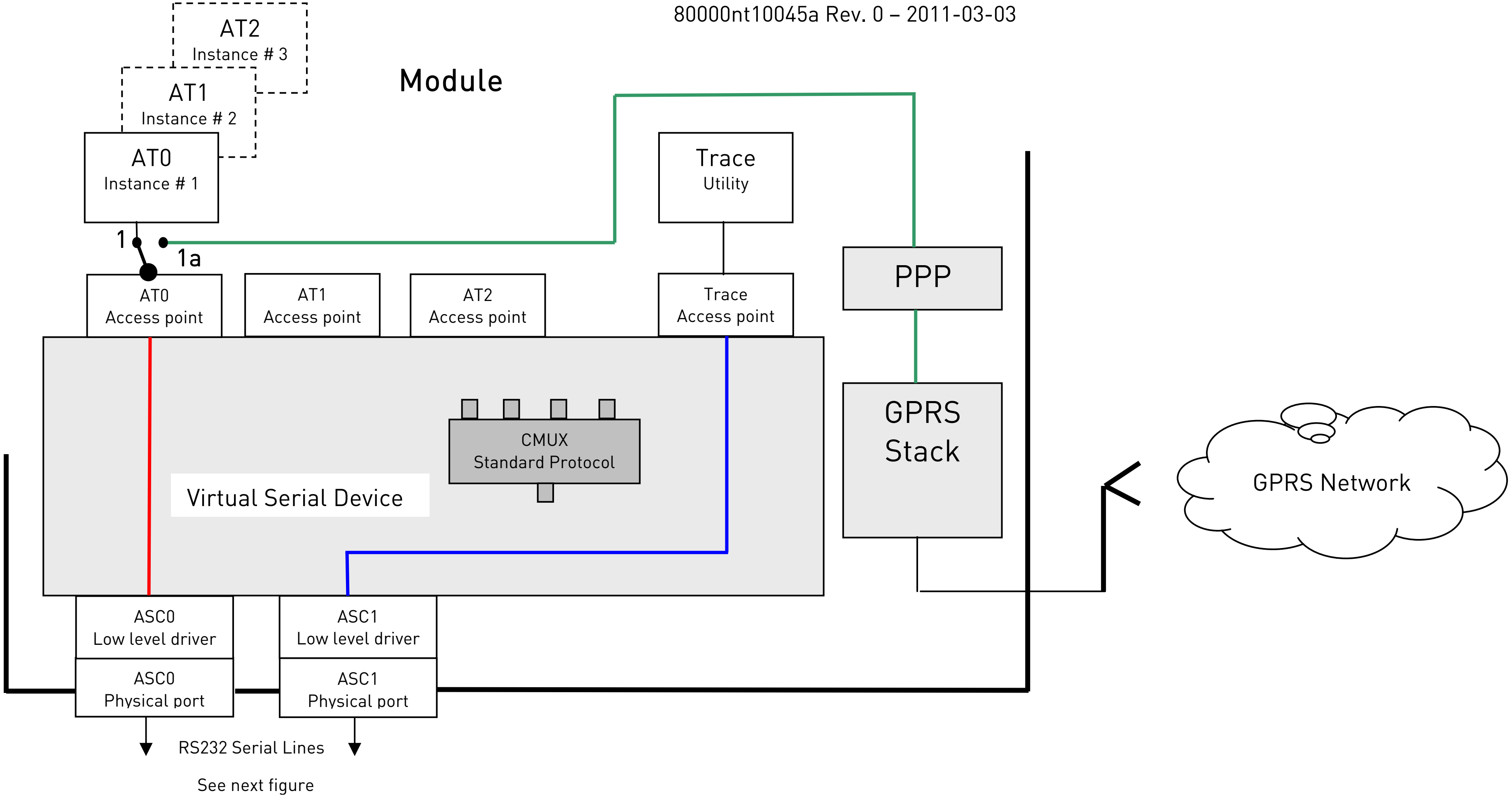

fig. 15: TCP/IP Protocol on Dial Up

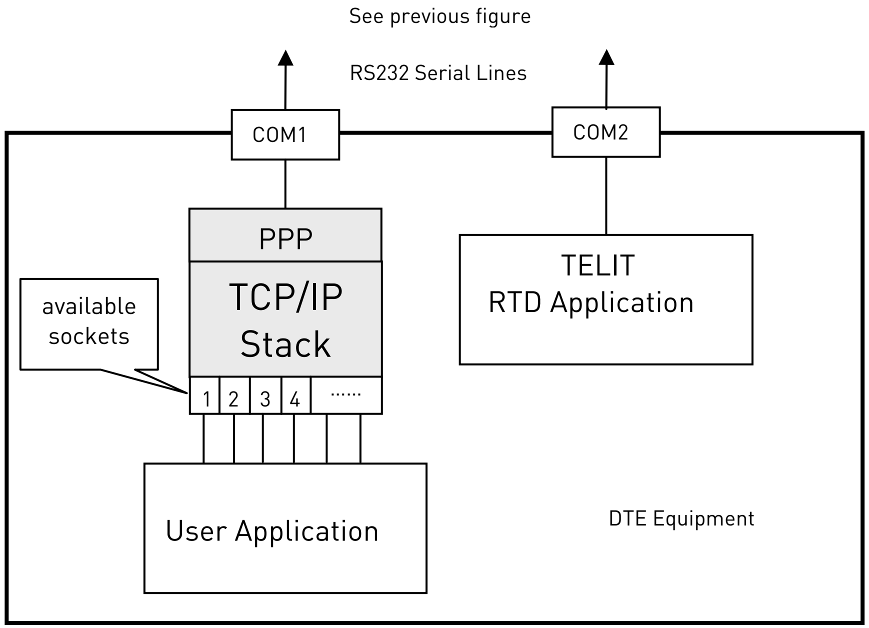

fig. 16: TCP/IP Protocol on Dial Up (con't)

2.5.3. TCP/IP on GSM

Let's start with the VSD configuration at the module power ON, see fig. 1. In this configuration VSD connects the physical port ASC0 to the AT0 AT command parser through the AT0 Access Point; AT0 parser matches Instance # 1. The user, by means of DTE equipment, enters AT commands; they are parsed by the ATO parser and executed by the module. VDS also connects physical port ASC1 to the Trace Utility through the Trace Access Point. The user can see trace logs if DTE runs the TELIT RTD application. In this configuration the CMUX Standard Protocol [1] is not used. To use the TCP/IP protocol on GSM the steps showed hereupon must be followed, refer to fig. 17:

Enter:

- AT#SGACT=0,1 command to activate GSM context [2]. The following path: PPP-TCP/IP stack / VHW DTEGSM / IPE GSM / GSM stack is established;

- AT#SD command to arrange the TCP/IP connection using the selected socket identifier. The logical connection 1 is active; the local module is in COMMAND mode.

When the remote module successfully responds to the open connection,

- on locale DTE is displayed the CONNECT message. At this time the logical connection 1 is disconnected and the connection 1a is activated: the local and remote module can exchange data; they are in ON LINE mode. Every character that the user enters on DTE is sent to the remote module. To exit ON LINE mode and enter again COMMAND mode, the user must enter the escape sequence +++, the TCP/IP connection remains "alive". To get again the connection, the user enters the command AT#SO indicating the socket identifier.

The above steps let's understand that the AT0 access point, at different times, can be connected to a maximum of six TCP/IP connections, all alive at the same time. ASC0 physical serial port is connected to AT0 access point.

AT#FTPOPEN command can be used to open an FTP connection toward an FTP server.

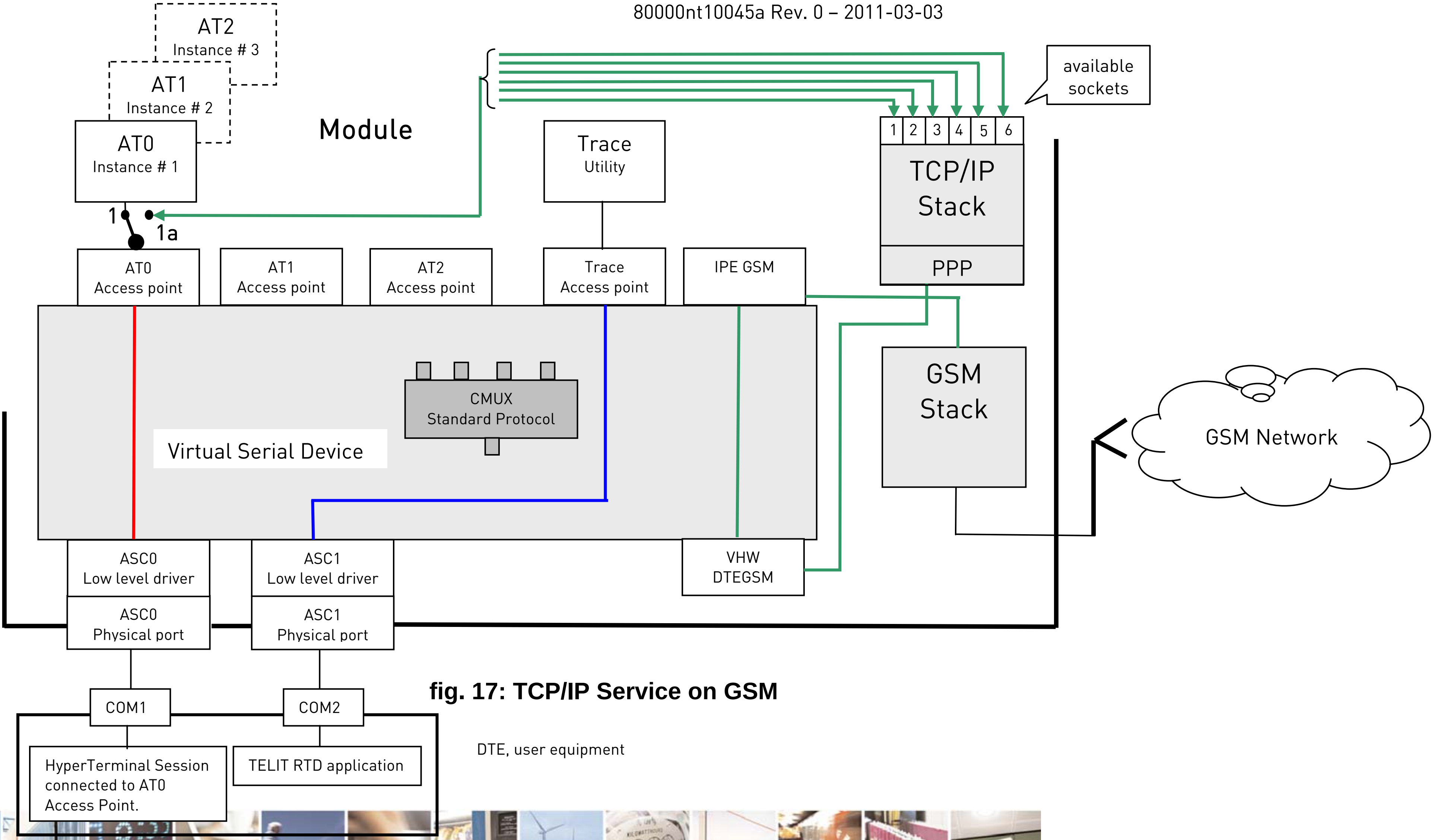

fig. 17: TCP/IP Service on GSM

2.5.4. TCP/IP and Python

Let's start with a well defined VSD configuration: the VSD configuration at the module power ON, see fig. 1. In this configuration VSD connects the physical port ASC0 to the AT0 AT command parser through the AT0 Access Point; AT0 parser matches Instance # 1. The user, by means of DTE equipment, enters AT commands; they are parsed by the AT0 parser and executed by the module. VDS also connects physical port ASC1 to the Trace Utility through the Trace Access Point. The user can see trace logs if DTE runs the TELIT RTD application. In this configuration the CMUX Standard Protocol [1] is not used.

Yet, let's suppose that we want a Python script uses the TCP/IP protocol to reach a remote server. To do that the Python script must follow the steps illustrated hereupon, refer to fig. 18.

The scrip runs:

- *import MDM* instruction, the connection 1 is broken and the connection 1a is created, the script is connected to the AT0 parser (instance #1);

- AT#SGACT=1,1 command to activate the already configured PDP context on GPRS network[3];

- AT#SD command to arrange the TCP/IP connection using the selected socket identifier. The logical connection 2 is active; the script is in COMMAND mode.

When the remote module successfully responds to the open connection,

- the script receives the CONNECT message. It means that the logical connection 2 is disconnected and the connection 2a is activated: the script and remote module can exchange data; they are in ON LINE mode. Every character that the script sends, it is received by the remote module. To exit ON LINE mode and enter again COMMAND mode, the script must send the escape sequence +++, the TCP/IP connection remains "alive". To get again the connection, the script runs the command AT#SO indicating the socket identifier.

The above steps let's understand that the AT0 access point, at different times, can be connected to a maximum of six TCP/IP connections, all alive at the same time. Python script is connected to AT0 access point.

AT#FTPOPEN command can be used to open an FTP connection toward an FTP server.

[3] Use AT#SGACT= 0,1 to use GSM network.

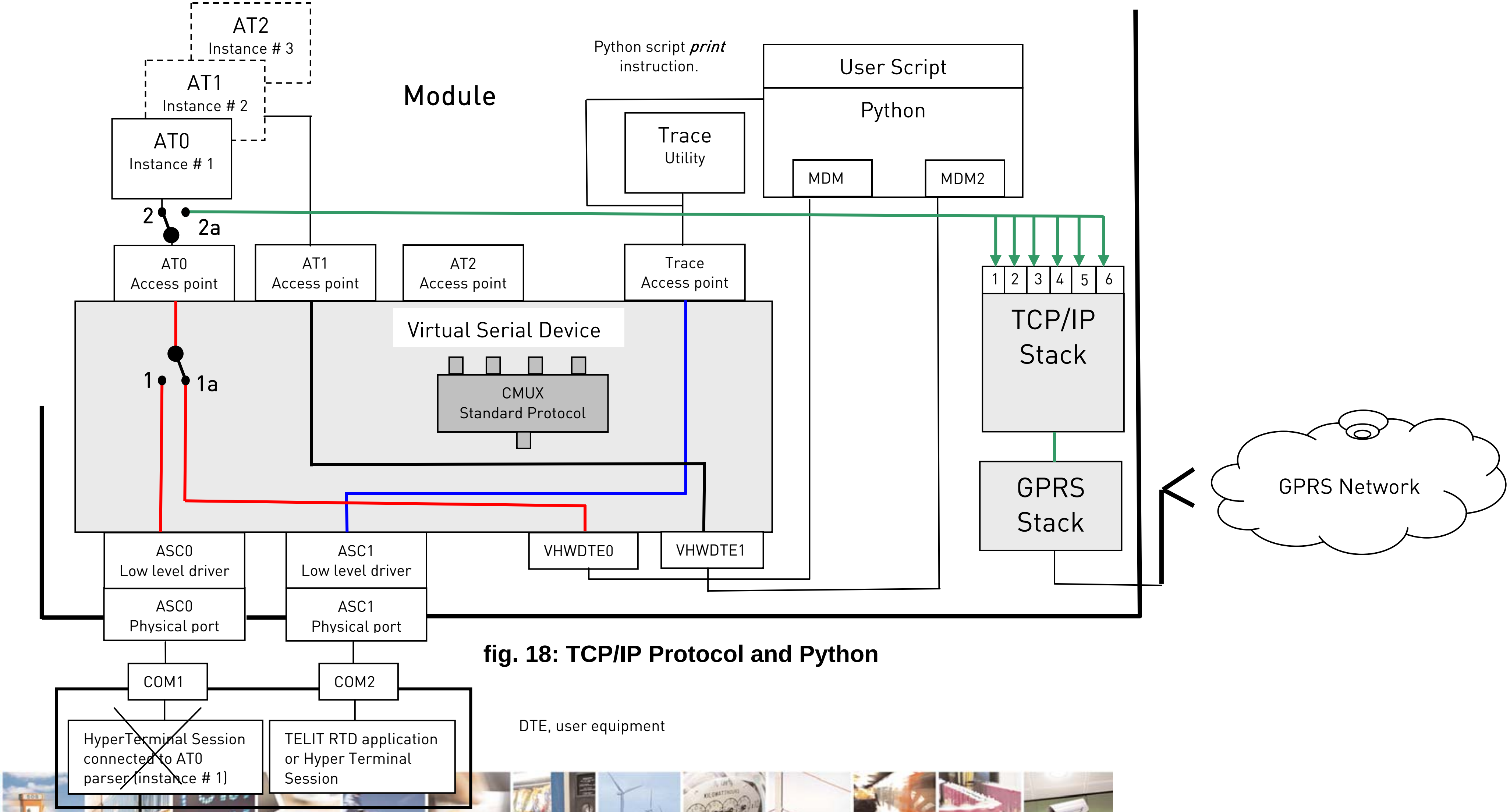

fig. 18: TCP/IP Protocol and Python

2.5.5. TCP/IP on GPRS and CMUX Service

To activate CMUX Service, refer to paragraph 2.2. Yet, let's suppose to have activated CMUX and need to establish six TCP/IP connections toward three servers at the same time. For example, any server has two connections. To accomplish this configuration follow these steps, and refer to fig. 19 and fig. 20:

- Select Hyper Terminal session connected through VC1 to AT0 parser;

- After socket configuration (e.g.: socket #1) and PDP context definition and activation, enter the AT#SD command to arrange the TCP/IP connection, the IP remote address and remote TCP port must be known. The local module, in COMMAND mode, is waiting to enter ON LINE mode;

- When the remote server successfully responds to the required connection, on locale DTE, by means of VC1 serial connection, is displayed the CONNECT message. At this time the logical connection 1 is disconnected and the logical connection 1a is activated. The local module and remote server can exchange data; they are in ON LINE mode. Every character that the user enters on DTE is sent to the remote server. To exit ON LINE mode and enter again COMMAND mode the user must enter the escape sequence **+++**.

- Repeat the steps to create a new TCP/IP connection using a new socket identifier.

These steps can be repeated for AT1 /VC2 and AT2/VC3 parsers using the available sockets.

Summarizing:

- Six TCP/IP connections are "alive" at the same time;
- Three Hyper Terminal sessions can be ON LINE at the same time, as required by the user;
- VC4 is available for an RTD session.

TELIT Serial Port MUX application can be substituted by any User Application provided with CMUX Standard Protocol [1].

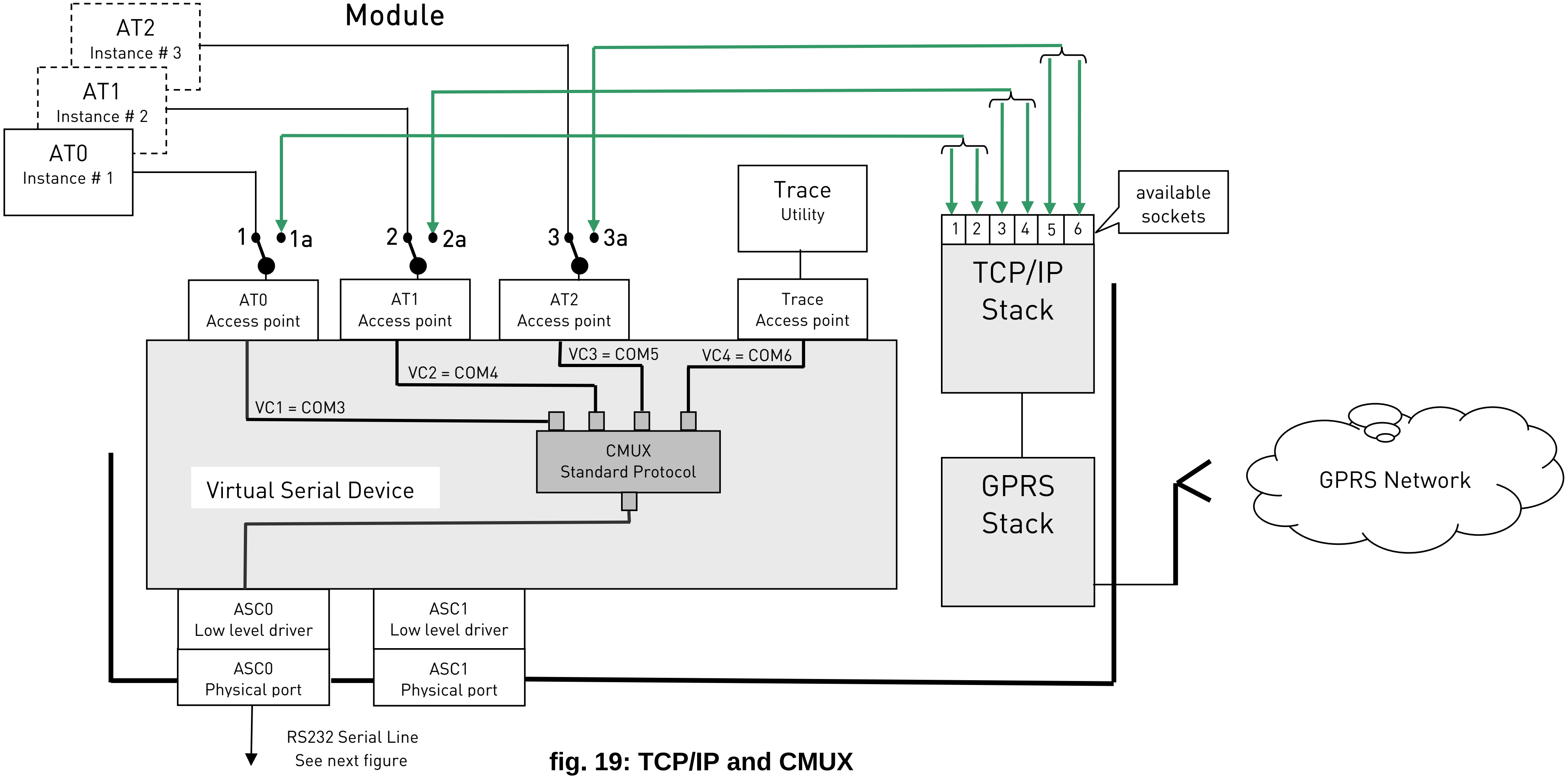

fig. 19: TCP/IP and CMUX

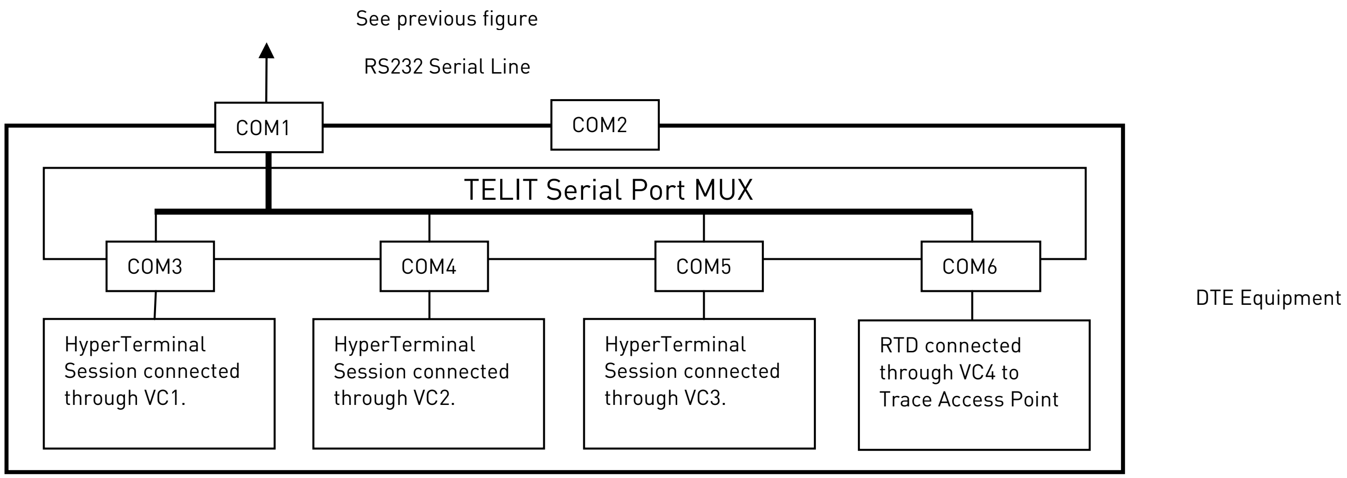
See previous figure
RS232 Serial Line
COM1
COM2
TELIT Serial Port MUX
COM3
COM4
COM5
COM6
DTE Equipment
HyperTerminal Session connected through VC1.
HyperTerminal Session connected through VC2.
HyperTerminal Session connected through VC3.
RTD connected through VC4 to Trace Access Point

fig. 20: TCP/IP and CMUX, (con't)

2.6. SMS, TCP AT Run and Event Monitor Services

Tab. 9 shows all the available connections (☺) between instances and Services: if one instance is used by TCP AT Run Service, that instance is not more available for the other two Services, while SMS AT Run and Event Monitor Services can use the same instance but not at the same time.

Services	Instances available for Services		
	#1	#2	#3
SMS AT Run	/	☺	☺
TCP AT Run	/	☺	☺
Event Monitor	/	☺	☺

Tab. 9: Instances vs. Services

The description of the AT commands used to configure the Services and how to use them is out of the scope of this guide, refer to [5], [6], [7].

2.6.1. SMS AT Run Service

SMS AT Run Service in a few words: after suitable SMS AT Run Service configuration, when an SMS is received it is analyzed. If it holds an AT command, the command is extracted and executed. In general, the AT command response is packaged into an SMS message and sent to the sender using the transport protocol provided by the SMS massage Service, see fig. 21.

SMS AT Run and Event Monitor Services are queued and both are using the VHWDTE SMSAT access point of VSD.

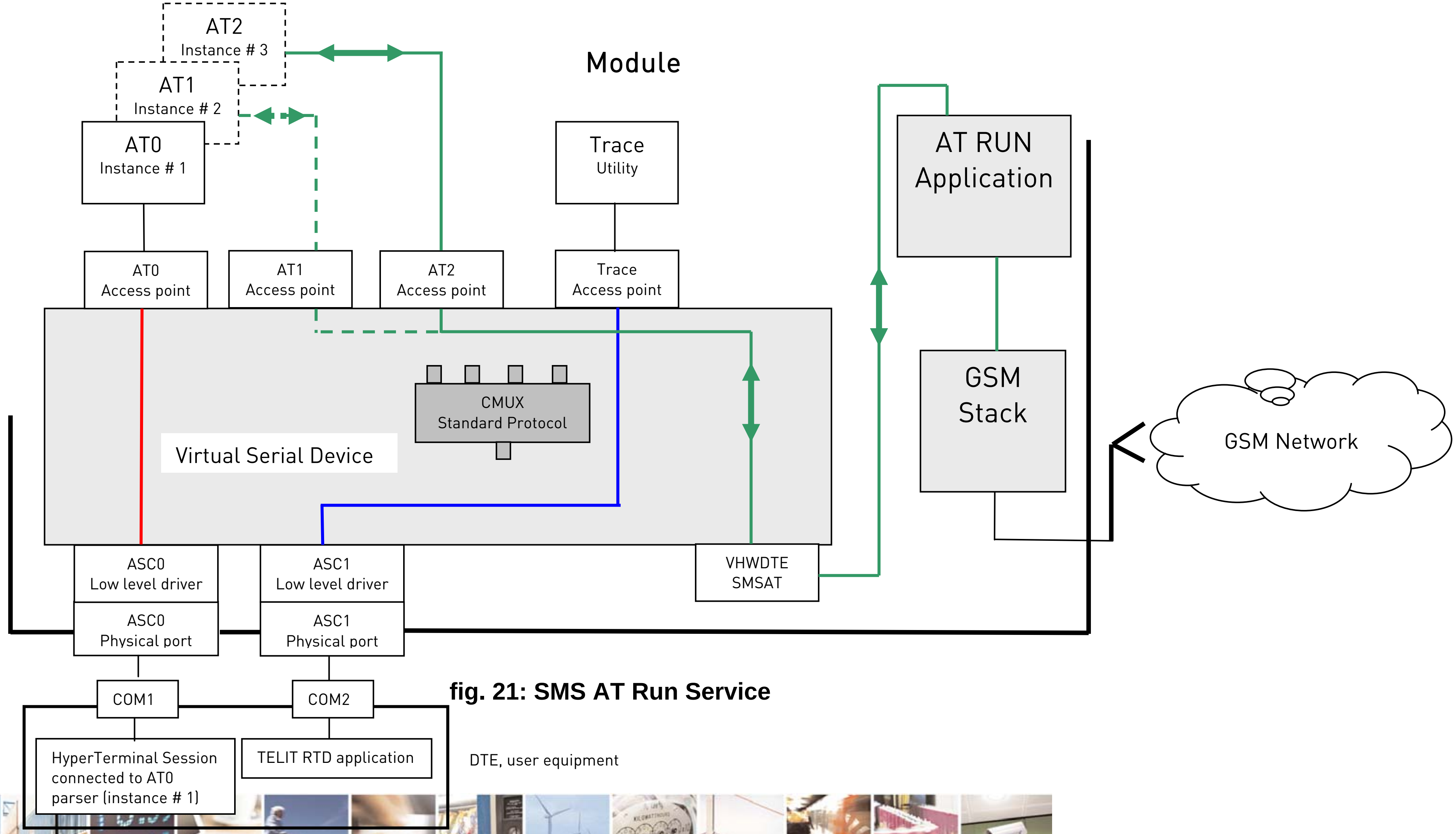

fig. 21: SMS AT Run Service

2.6.2. Event Monitor Service

Event Monitor Service in a few words: when the selected event, which can be picked up from an events list provided by the module, happens (e.g.: a GPIO status is changed) the AT command associated to the just happened event is executed, see fig. 22. Event Monitor Service needs to be configured.

ⓘ SMS AT Run and Event Monitor Services are queued and both are using the VHWDTE SMSAT access point of VSD.

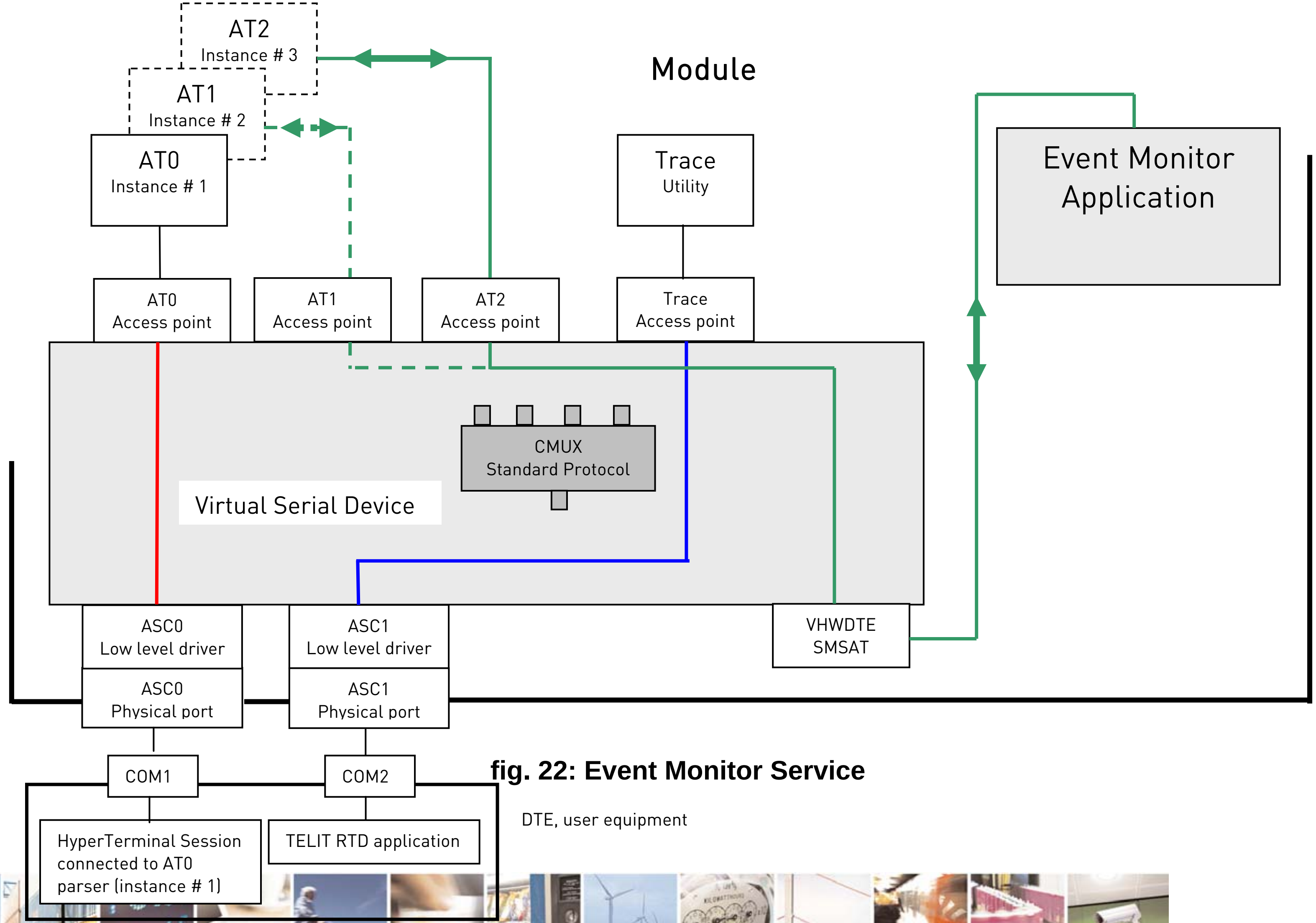

fig. 22: Event Monitor Service

2.6.3. TCP AT Run Service

SMS AT Run Service in a few words: let's suppose that the module is a remote module dedicated to control a physical quantity (e.g.: temperature). By means of another module, let's to call it local, it is possible to establish a TCP/IP connection between them. Yet, the local module can send AT command to the remote module by mean of the TCP/IP connection. The remote module executes the received AT commands and sends back the relative responses (temperature value) to the sender, see fig. 23. TCP AT Run Service needs to be configured. It is worth remind that the Service can use both contexts: GPRS or GSM, fig. 23 shows GPRS context.

TCP AT Run Service uses the dedicated VHWDTE TCPAT access point of VDS.

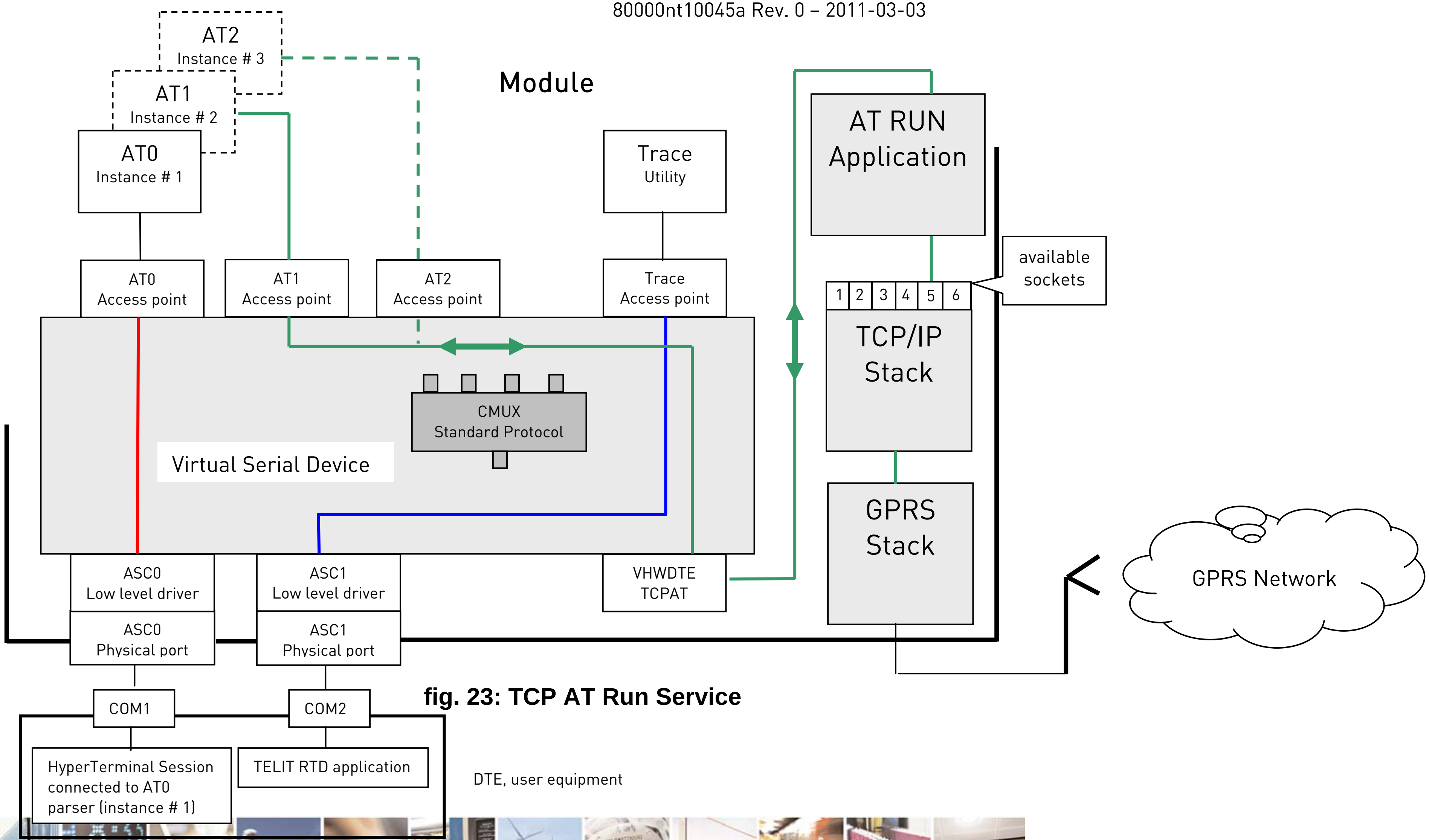

fig. 23: TCP AT Run Service

2.7. SAT Service

SAT Service in a few words: let's suppose that the module holds a SIM running a SAT application. SAT application sends, on its initiative, commands to the SAT Service running on the module. The SAT service analyzes the received commands, transforms them in AT commands and, by means of the connection managed by VSD, sends the AT commands to the AT2 parser (instance #3). The AT commands results are send back to the sender. As showed by Tab. 10 SAT Service can use only the instance # 3. SAT Service needs to be configured.

Services	Instances available for Service		
	#1	#2	#3
SAT	/	/	☺

Tab. 10: SAT Service Instance

The description of the AT commands used to configure SAT Service and how to use it is out of the scope of this guide, refer to [8].

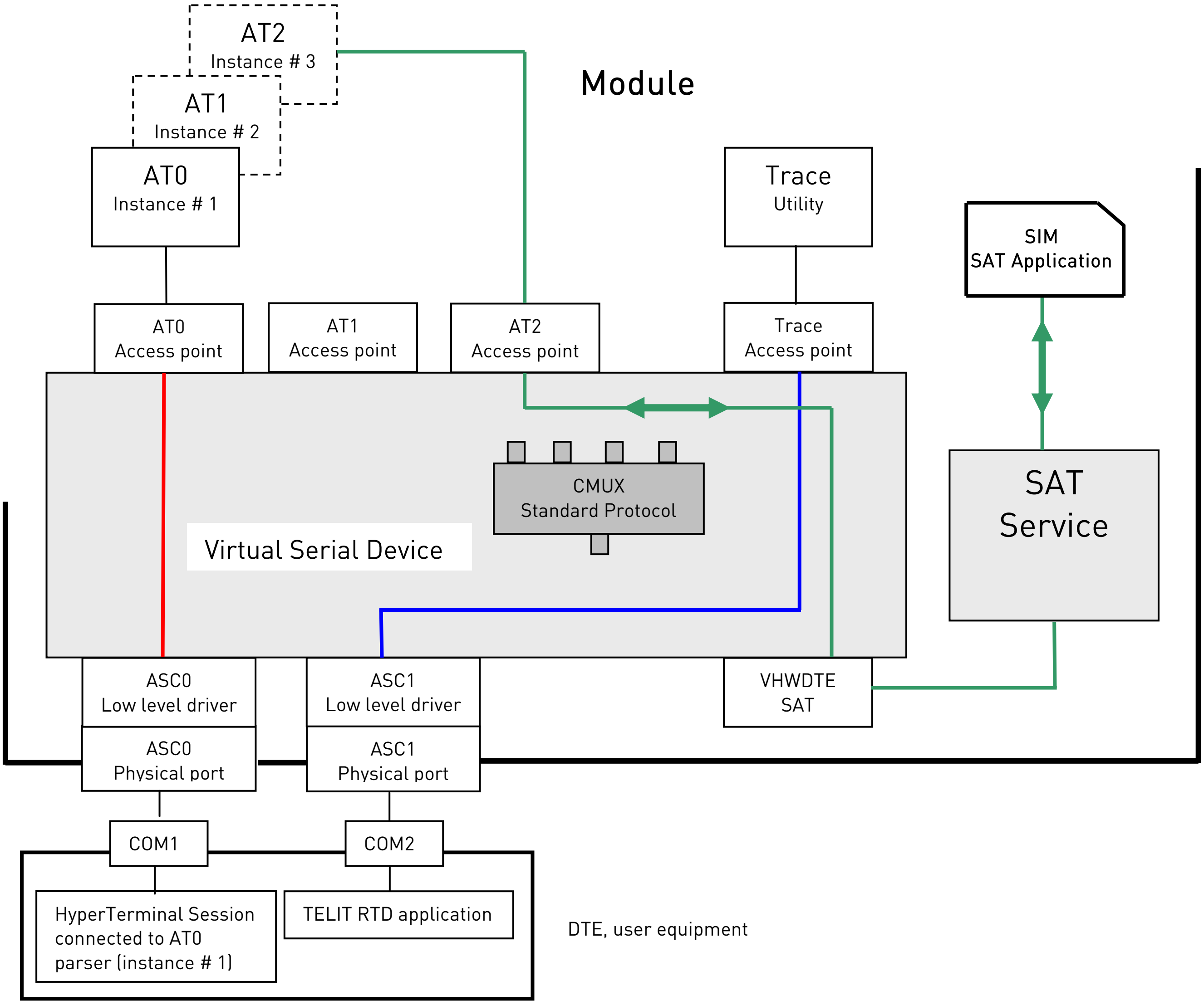

fig. 24: SAT Service

2.8. FOTA Service

FOTA Service in a few words: a FOTA server can force the module to enter the updating phase to upgrade the actual firmware. As showed by Tab. 11 FOTA Service can use only the instance # 3.

Services	Instances available for Service		
	#1	#2	#3
SAT	/	/	☺

Tab. 11: OTA Service Instance

The description of the OTA Service and how to use it is out of the scope of this guide [9].

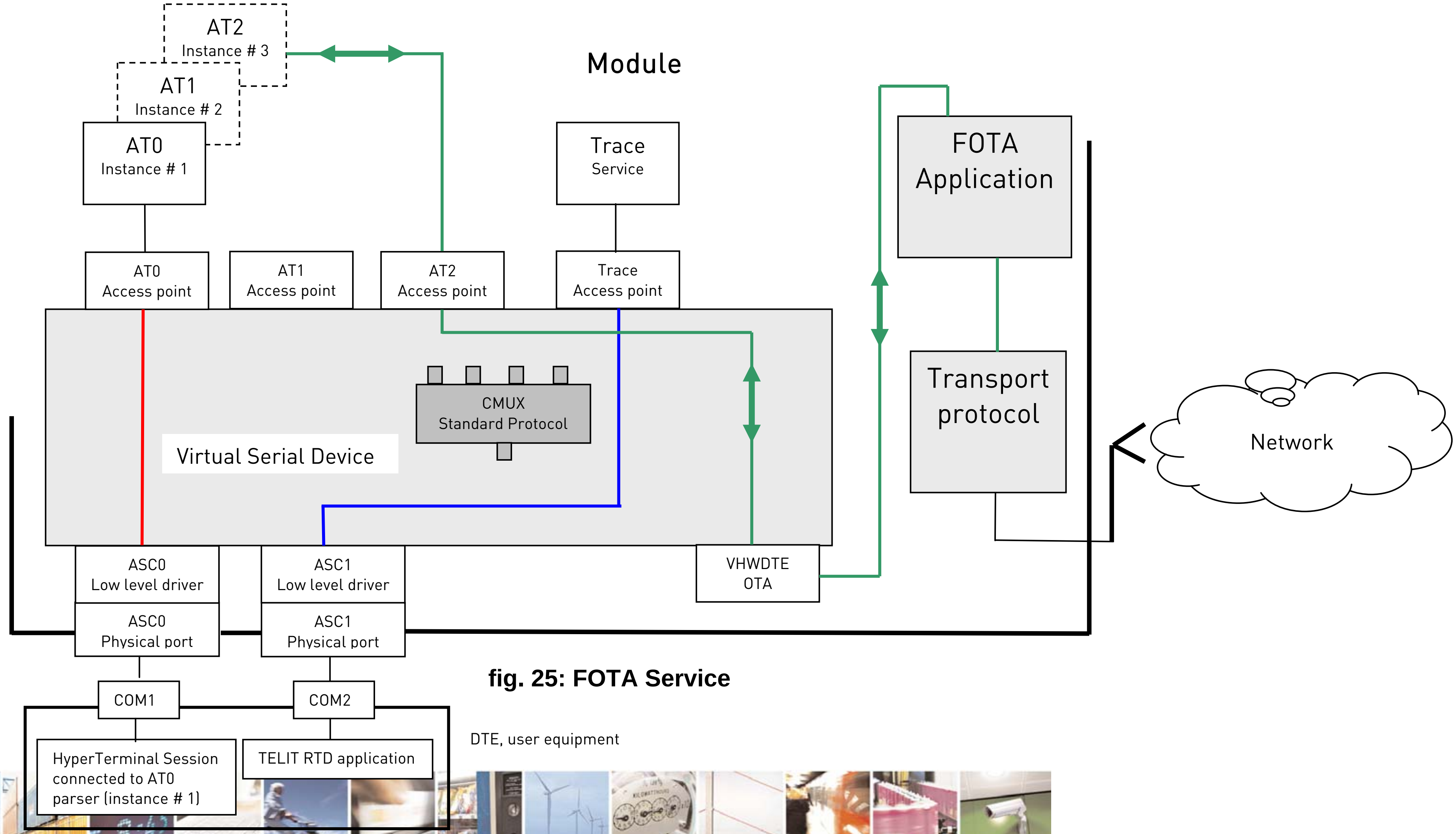

fig. 25: FOTA Service

3. Resource Sharing among Services

This chapter focuses on some examples illustrating the resource sharing among the Services. This description is carried out by means of tables to clearly show when two or more Services try to use the same resource and which one is involved in the contention. The examples don't exhaustively cover all the possible scenarios.

The general priority concept used by Services to acquire a resource should be the following: the generic Service can acquire a resource (e.g.: an instance) if the resource is not used by another Service. If the resource is busy the requesting Service can't acquire the resource, e.g.:

- CMUX Service can't "steal" resources if they are used by FOTA, EVMONI, SMSATRUN, TCPATRUN and PYTHON (MDM2) Services.

This is not always true, there are some exceptions hereupon showed:

- PYTHON Service doesn't respect the "general priority concept"; it "steals" the resources even if they are used by another Service.

- OTA Service doesn't respect the "general priority concept"; it "steals" the instance # 3 even if it is used by another Service.

There is a scenario where two Services can use the same resource:

- SMS AT Run and Event Monitor Services can use the same instance at the same time. The conflict is resolved by a queuing mechanism.

Legend for next tables:

“☺” : service can acquire the resource indicated on the column top;
“☺” : service has acquired the resource indicated on the column top;
“/” : service doesn’t use the resource; it stays on its original status;
“X” : service forces the resource to be unserviceable;
“X” : service has forced the resource into an unserviceable status;

Tab. 12 summarizes the Services object of the present document; TRACE Utility and AT Commands are added. For each Service are showed all the resources that the Service can acquire in order to evaluate possible resource conflicts when the user selects the Services to use together.

Services and AT Commands from serial ports	Resources						Refer to
	AT0 Instance #1	AT1 Instance #2	AT2 Instance #3	Trace Acc. P.	ASC0	ASC1	
PYTHON	☺	☺	/	X	☺	☺	fig. 8
TCP AT Run	/	☺	☺	/	/	/	fig. 23
SMS AT Run	/	☺	☺	/	/	/	fig. 21
Event Monitor	/	☺	☺	/	/	/	fig. 22
SAT	/	/	☺	/	/	/	fig. 24
FOTA	/	/	☺	/	/	/	fig. 25
CMUX	☺	☺	☺	☺	☺	/	fig. 4
GPS	/	/	/	/	/	☺	fig. 9
TRACE Utility + print/PY	/	/	/	☺	/	☺	fig. 1, fig. 5
AT Commands	☺	☺	☺	☺	☺	☺	fig. 1, fig. 2, fig. 3

Tab. 12: Services List

The following pages show some examples of TELIT module configurations using different Services at the same time.

Examples of resource sharing among Services without conflicts:

Refer to Tab. 13: AT Run and Event Monitor Services are running together. SMS AT Run and Event Monitor Services can use the AT2 Access Point at the same time. ASC0 serial port can be used to enter AT Commands, ASC1 serial port is used by the Trace Utility. No resource conflicts are present, see "☺".

Services and AT Commands from serial ports	Resources						Refer to
	AT0 Instance #1	AT1 Instance #2	AT2 Instance #3	Trace Acc. P.	ASC0	ASC1	
PYTHON	☺	☺	/	X	☺	☺	fig. 8
TCP AT Run	/	☺	☺	/	/	/	fig. 23
SMS AT Run	/	☺	☺	/	/	/	fig. 21
Event Monitor	/	☺	☺	/	/	/	fig. 22
SAT	/	/	☺	/	/	/	fig. 24
FOTA	/	/	☺	/	/	/	fig. 25
CMUX	☺	☺	☺	☺	☺	/	fig. 4
GPS	/	/	/	/	/	☺	fig. 9
TRACE Utility + print/PY	/	/	/	☺	/	☺	fig. 1, fig. 5
AT Commands	☺	☺	☺	☺	☺	☺	fig. 1, fig. 2, fig. 3

Tab. 13: AT Run, Event Monitor Services and other Services

Refer to Tab. 14: AT Run and Event Monitor Services are running together. SMS AT Run and Event Monitor Services can use the AT1 Access Point at the same time. ASC0 serial port can be used to enter AT Commands, ASC1 serial port is used by the Trace Utility. No resource conflicts are present, see "☺".

Services and AT Commands from serial ports	Resources						Refer to
	AT0 Instance #1	AT1 Instance #2	AT2 Instance #3	Trace Acc. P.	ASC0	ASC1	
PYTHON	☺	☺	/	X	☺	☺	fig. 8
TCP AT Run	/	☺	☺	/	/	/	fig. 23
SMS AT Run	/	☺	☺	/	/	/	fig. 21
Event Monitor	/	☺	☺	/	/	/	fig. 22
SAT	/	/	☺	/	/	/	fig. 24
FOTA	/	/	☺	/	/	/	fig. 25
CMUX	☺	☺	☺	☺	☺	/	fig. 4
GPS	/	/	/	/	/	☺	fig. 9
TRACE Utility + print/PY	/	/	/	☺	/	☺	fig. 1, fig. 5
AT Commands	☺	☺	☺	☺	☺	☺	fig. 1, fig. 2, fig. 3

Tab. 14: AT Run, Event Monitor Services and other Services (con't)

Refer to Tab. 15: Python, AT Run and Event Monitor Services are running together. SMS AT Run and Event Monitor Services can use the AT1 Access Point at the same time.

To avoid AT1 conflict, Python script mustn't run *import MDM2* instruction.

No resource conflicts are present, see "☺".

Services and AT Commands from serial ports	Resources						Refer to
	AT0 Instance #1	AT1 Instance #2	AT2 Instance #3	Trace Acc. P.	ASC0	ASC1	
PYTHON	☺	☺	/	X	☺	☺	fig. 8, no *MDM2*
TCP AT Run	/	☺	☺	/	/	/	fig. 23
SMS AT Run	/	☺	☺	/	/	/	fig. 21
Event Monitor	/	☺	☺	/	/	/	fig. 22
SAT	/	/	☺	/	/	/	fig. 24
FOTA	/	/	☺	/	/	/	fig. 25
CMUX	☺	☺	☺	☺	☺	/	fig. 4
GPS	/	/	/	/	/	☺	fig. 9
TRACE Utility + print/PY	/	/	/	☺	/	☺	fig. 1, fig. 5
AT Commands	☺	☺	☺	☺	☺	☺	fig. 1, fig. 2, fig. 3

Tab. 15: Python and other Services

Tab. 16 shows the resource sharing among Python, SMS AT Run, Event Monitor, CMUX and GPS Services. CMUX using ASC0/VC4/Trace Access Point enables to use the Trace Utility and the Python script *print* instruction. GPS uses ASC1 serial port.

No resource conflicts are present, see "☺".

Services and AT Commands from serial ports	Resources						Refer to
	AT0 Instance #1	AT1 Instance #2	AT2 Instance #3	Trace Acc. P.	ASC0	ASC1	
PYTHON	☺	☺	/	X	☺	☺	fig. 8, no *SER, SER2*
TCP AT Run	/	☺	☺	/	/	/	fig. 23
SMS AT Run	/	☺	☺	/	/	/	fig. 21
Event Monitor	/	☺	☺	/	/	/	fig. 22
SAT	/	/	☺	/	/	/	fig. 24
FOTA	/	/	☺	/	/	/	fig. 25
CMUX	☺	☺	☺	☺	☺	/	fig. 4
GPS	/	/	/	/	/	☺	fig. 9
TRACE Utility + print/PY	/	/	/	☺	/	☺	fig. 1, fig. 5
AT Commands	☺	☺	☺	☺	☺	☺	fig. 1, fig. 2, fig. 3

Tab. 16: CMUX and other Services

An example of resource conflict:

Let's suppose that the user enters the AT#STARTMODESCR=2 command and store it on NVM; the preliminary configuration is done, yet the module is powered off.

Power on the module, instance #3 and ASC0 are connected together, see paragraph 2.3. In this initial module configuration, the Python script can use only the following *import* instruction: *import MDM, import MDM2, and import SER2*. If *import SER* is used, the ASC0 resource conflict arises as showed by the red area in the following table. It is advisable to repeat that the Python script has the priority on the other services, so it steals the ASC0 resource breaking the instance #3/ASC0 connection created by the AT#STARTMODESCR=2 command, previously stored into NVM during the preliminary configuration phase and run on the power on.

Services and AT Commands from serial ports	Resources						Refer to
	AT0 Instance #1	AT1 Instance #2	AT2 Instance #3	Trace Acc. P.	ASC0	ASC1	
PYTHON	☺	☺	/	X	☺	☺	fig. 7, fig. 8
TCP AT Run	/	☺	☺	/	/	/	fig. 23
SMS AT Run	/	☺	☺	/	/	/	fig. 21
Event Monitor	/	☺	☺	/	/	/	fig. 22
SAT	/	/	☺	/	/	/	fig. 24
FOTA	/	/	☺	/	/	/	fig. 25
CMUX	☺	☺	☺	☺	☺	/	fig. 4
GPS	/	/	/	/	/	☺	fig. 9
TRACE Utility + print/PY	/	/	/	☺	/	☺	fig. 1, fig. 5
AT Commands	☺	☺	☺	☺	☺	☺	fig. 7

Tab. 17: Python & AT#STARTMODESCR=2 conflict