High Level Design Document
Enhance TTY for dynamic Switching between HCO/VCO

Department:	Berlin Wireless Centre
Creation Date:	03 February, 2005
Last Modified:	03 February, 2005, by Richa Arora
ID and Version:	
Status:	Draft

0 Document Controls

0.1 Document History

ID	Author	Date	Status
	Richa	03 February, 2005	Draft

0.2 References, Abbreviations, Terms

Codec	Coder/Decoder
DSP	Digital Signal Processing
HCO	Hearing Carry Over
MMI	Man Machine Interface
PS	Protocol Stack
TTY	Text Telephone
VCO	Voice Carry Over

[1] ACI- Application Control Interface, 8415_052.doc

[2] 3GPP TS 24.008

Table of Contents

1 Introduction

This document addresses the high level design description of the TTY sub feature “Dynamic Switching between HCO/VCO in TTY”

The sub feature is designed to provide the facility to user to switch between HCO/VCO during an active TTY call.

The user shall be able to change the TTY modes during an active TTY call. This feature shall enhance the current implementation of changing the TTY modes statically. Currently, the TTY mode has to be set before making a TTY call and these modes cannot be changed during an active TTY call. With this sub feature the user shall be able to change the TTY mode during an active call.

2 Required Functionality

The following functionalities shall be considered while implementing the dynamic switching of the TTY modes:-

1. Provide functionality in ACI to change the TTY mode during the active TTY call.
2. Provide functionality in Audio driver to modify the audio profile for switching the TTY mode.

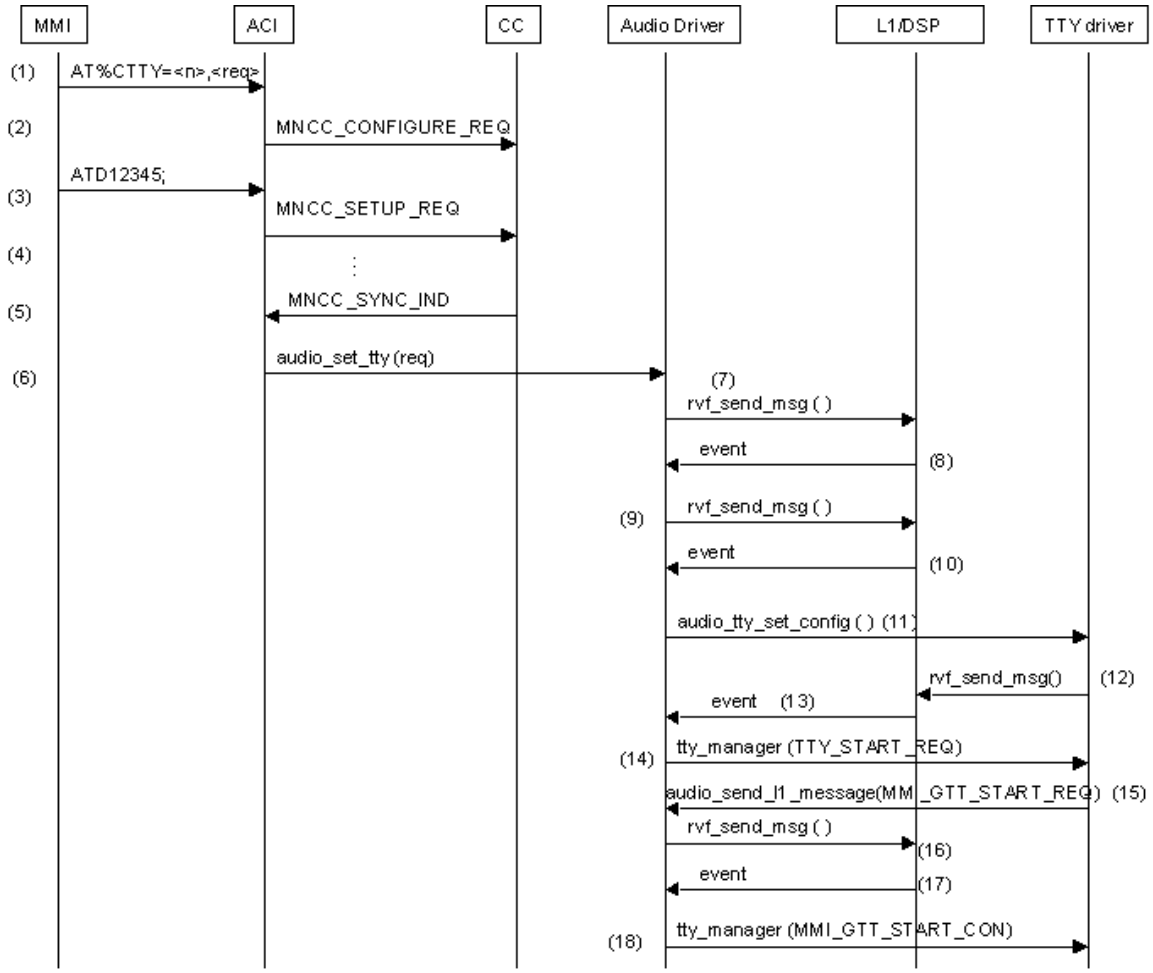
2.1 Derived Functionality

1. The AT command %CTTY shall be enhanced to support the dynamic switching of the TTY modes
2. The Audio driver shall be modified to independently change the audio profile without affecting the DSP codec
3. There shall be no change in the static switching of the TTY mode, which is done by sending MNCC_CONFIGURE_REQ by ACI to CC entity
4. There shall be no change in the way TTY is stopped when a TTY call is ended.
5. There shall be no change in current implementation of L1/DSP codec for dynamic switching of TTY modes
6. Since Windows MMI shall be used, hence no change in MMI shall be required

3 System Design

The current scenarios are explained as below:-

3.1 M0 call with TTY enabled



1. The TTY mode is set statically with %CTTY command. For values of <n> and <req> please refer to AT Command Interface document [1]. For this case we have assumed the value of <req> to be 1 or 2 or 3

2. ACI sends a MNCC_CONFIGURE_REQ to set the Bearer Capability Parameters according to 3GPP TS 24.008 [2] for a TTY call

3. A voice call is initiated.

4. ACI starts the call setup by sending MNCC_SETUP_REQ to CC.

5. After receiving MNCC_SYNC_IND from CC, ACI is informed about channel being allocated.

6. After channel being allocated, ACI initiates the TTY setup.

7. The audio driver sends a message with AUDIO_MODE_SAVE_REQ as the message identifier and "default" as message parameter to the L1/DSP.

8. An event is generated by DSP, on receipt of which the audio driver invokes the callback function of AUDIO_MODE_SAVE_REQ message that is audio_save_def_return().

9. The callback of AUDIO_MODE_SAVE_REQ message (audio_save_def_return()) sends a message with AUDIO_MODE_LOAD_REQ as the message identifier and the "requested TTY mode" as the message parameter.

10. An event is generated by DSP, on receipt of which the audio driver invokes the callback function of AUDIO_MODE_LOAD_REQ message that is audio_load_return().

11. The callback function of AUDIO_MODE_LOAD_REQ message, the audio driver sets the configuration for the TTY driver.

12. The TTY driver sends a message with TTY_START_REQ as message identifier to the L1/DSP.

13. An event is generated by DSP, on receipt of which the audio driver invokes TTY driver function tty_manager() with TTY_START_REQ as the message identifier.

14. The function tty_manager() is invoked with TTY_START_REQ as the message identifier by the audio driver, on receipt of which the TTY driver changes the state to TTY_WAIT_START_CON.

15. The TTY driver invokes the audio driver function audio_send_l1_message() with MMI_GTT_START_REQ as the message identifier and changes the state to TTY_WAIT_START_CON

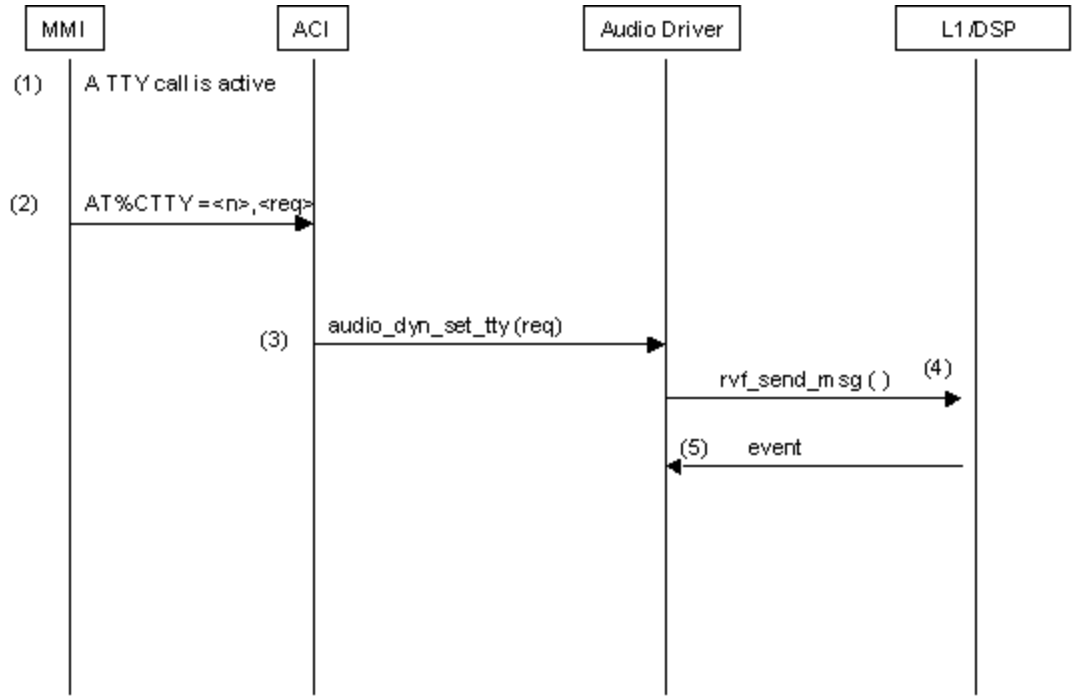
16. On receipt of MMI_GTT_START_REQ message the audio driver passes the message to the L1/DSP.

17. An event is generated by L1/DSP on completion of the MMI_GTT_START_REQ.

18. The audio driver on receipt of the above event, sends a MMI_GTT_START_CON message to the TTY driver and the state is changed to TTY_WAIT_STOP_COMMAND.

3.2 Dynamic Switching of TTY Mode

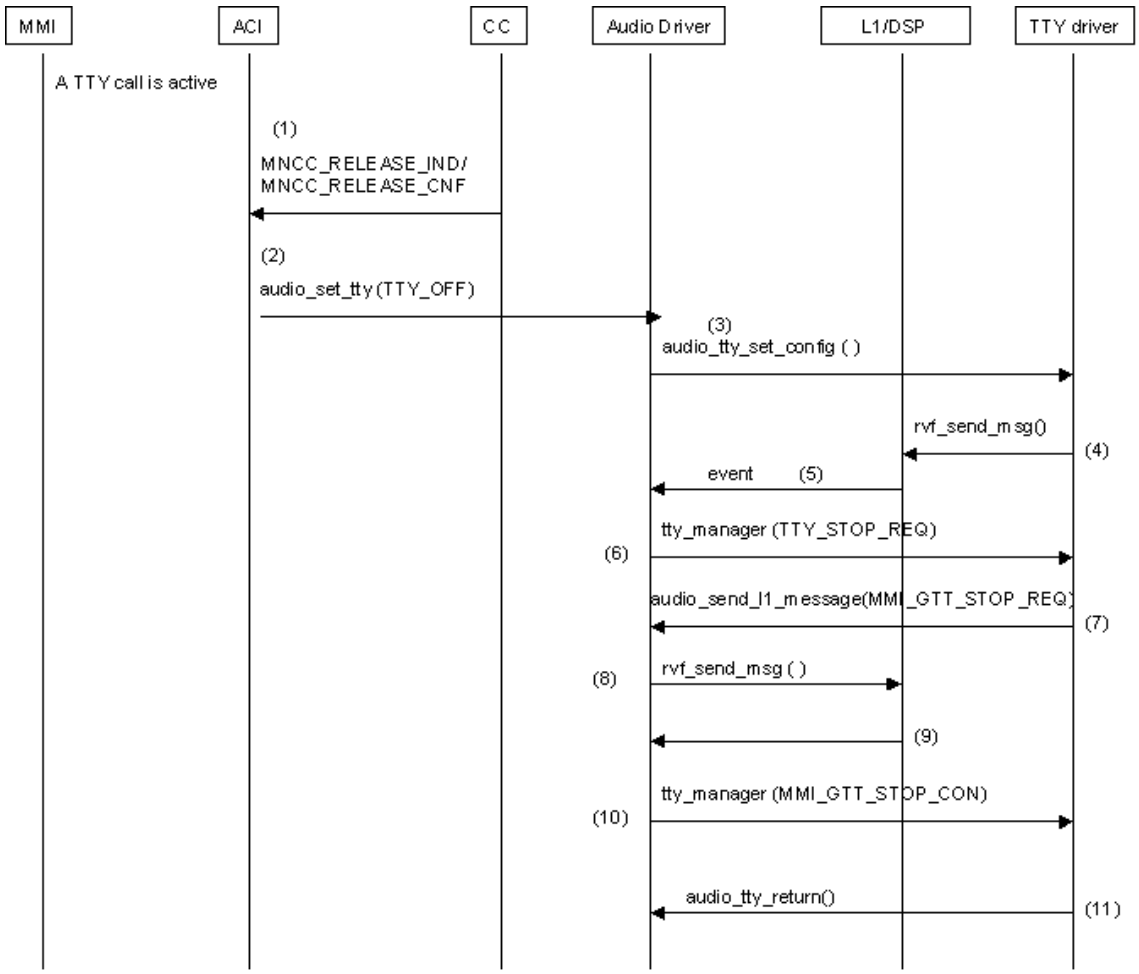
For the dynamic switching of the TTY mode, the scenario is explained as below, assuming that a TTY active call has been established as shown above in the Section 3.1.



1. A TTY call is successfully established as described in the Section 3.2
2. The TTY mode can be changed dynamically with %CTTY command.
3. ACI after verifying for an active TTY call and mode, shall invoke the audio driver function audio_dyn_set_tty() (to be added) with the required TTY mode.
4. The audio driver changes the audio profile by sending a message with AUDIO_MODE_LOAD_REQ as message identifier and the requested TTY mode.
5. As a result an event is generated by the DSP, on receipt of which the call back function of the AUDIO_MODE_LOAD_REQ message shall be invoked. This call back function shall take care of setting the necessary parameters of TTY mode and state in the audio driver.

3.3 Disabling of TTY

The current design of stopping the TTY is as shown below:-



1. An active TTY call is terminated.
2. ACI sends TTY_OFF mode to the audio driver.
3. The audio driver sends TTY_STOP_REQ mode to the TTY driver after receiving the TTY_OFF from ACI.
4. The TTY driver sends a message with message identifier as TTY_STOP_REQ to L1/DSP.
5. As a result, an event is generated by the L1/DSP.
6. After getting the TTY_STOP_REQ as message identifier in the event the audio driver calls the tty_manager () function of the TTY driver with the TTY_STOP_REQ as the message identifier.
7. The TTY driver invokes the audio driver function audio_send_l1_message() with MMI_GTT_STOP_REQ as the message identifier. The TTY driver changes the state to TTY_WAIT_STOP_CON.
8. On receipt of MMI_GTT_STOP_REQ message the audio driver passes the message to the L1/DSP.
9. An event is generated by L1/DSP as a result.
10. The audio driver on receipt of the above event sends a MMI_GTT_STOP_CON message to the TTY driver. The TTY driver changes the state to TTY_IDLE.
11. The call back function audio_tty_return() is invoked by the TTY driver with the return status code.

4 Requirements to PS entities

4.1 Requirements to ACI

4.1.1 Modification of %CTTY command handler

The current implementation of %CTTY supports the static switching of the TTY mode by sending a MNCC_CONFIGURE_REQ to CC entity.

This implementation shall be enhanced to provide switching of TTY mode if any TTY call is active. The command handler shall take care of both static and dynamic switching.

The command handler shall set the Bearer Capabilities by sending MNCC_CONFIGURE_REQ to CC if no TTY call is active(already implemented).

If any TTY call is active, then the command handler shall change the TTY mode as requested as described in Section 3.2

4.1.2 Modification of the Audio Driver

The current implementation of the audio driver takes care of switching the audio profiles along with switching of the DSP codec and starting of the TTY driver. This TTY driver in turn takes care of sending MMI_GTT_START_REQ/ MMI_GTT_STOP_REQ message identifier to L1/DSP.

For the requirement of dynamically changing the TTY mode, the audio driver shall be modified to change the audio profile without affecting the DSP codec or TTY driver.

To switch the TTY mode, the audio profile shall be modified appropriately by sending message as AUDIO_MODE_LOAD_REQ with message identifier to the audio entity as shown in the Section 3.2

There shall be no modification when the TTY is stopped through the audio driver when an active TTY call is disconnected as shown in Section 3.3.