Design Specification

SMBS

(CHANGES ON MFWA & MFW-SMS)

Document Number:	Document number to be assigned
Version:	0.1
Status:	Draft
Approval Authority:	
Creation Date:	2003-Apr-01
Last changed:	2015-Mar-08
File Name:	My_SMBS_DesignSpec.doc

Important Notice

Change History

Date	Changed by	Approved by	Version	Status	Notes
2003-Apr-01	Thomas Schott (create)		0.1	Draft	1

Notes:

1. Initial version

Table of Contents

List of Figures and Tables

List of References

[ISO 9000:2000]	International Organization for Standardization. Quality management systems - Fundamentals and vocabulary. December 2000

1 Introduction

This document describes the detailed design of the SMBS-entity.

It may contains the required changes on available MFW-SMS and MFW-Adapter regarding insertion of SMBS in the system also.

The SMBS has interfaces between other entities. These interfaces also be described here.

Some MSC describes the communication of SMBS together with the connected entities.

Finally, a short description regarding to test also be available. The test can only be done for concatenated SMS at the target, because of missing client for the SMBS.

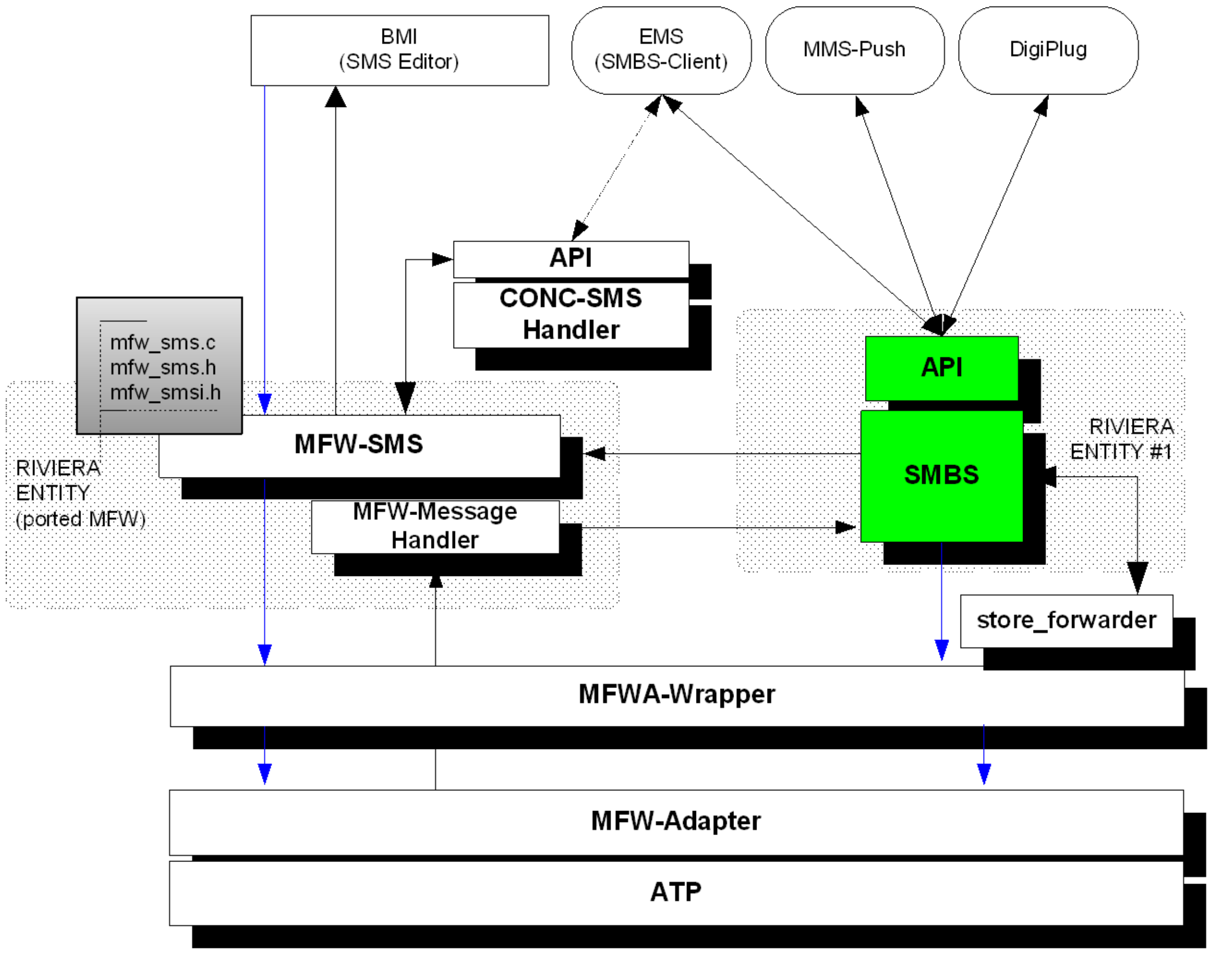
2 Overview

As described in the Technical Documentation the Short Message Bearer Service provides a way to distribute all SMS, which received from ACI.

The SMBS structure allows to subscribe clients, which handles these SMS based messages. By this way, it follows the client-server theory. A standard API provides the needed interface toward these clients to handle all possibly necessary requests from the client.

The concatenation of SMS will be handled in the client itself or may it can be use a concatenation handler service. This concatenation handler service is out of scope here (see design specification 'Concatenation Handler'). The SMBS itself does not be able to disassemble concatenated messages.

2.1 Application Interface



a) receiving / incoming SMS

Mainly, together with the MFW-Message Handler the MFW-Adapter provides the task decoupling between the GPF and Riviera task to prevent unexpected “hang ups” and provide synchronous returns to ACI. All messages will be stored in a queue called “postbox”.
The MFW-message handler distributes the incoming messages to to the SMBS for further processing. The SMBS will try out the SMS to MFW-SMS and/or the conc. SMS clients.

b) sending / outgoing SMS

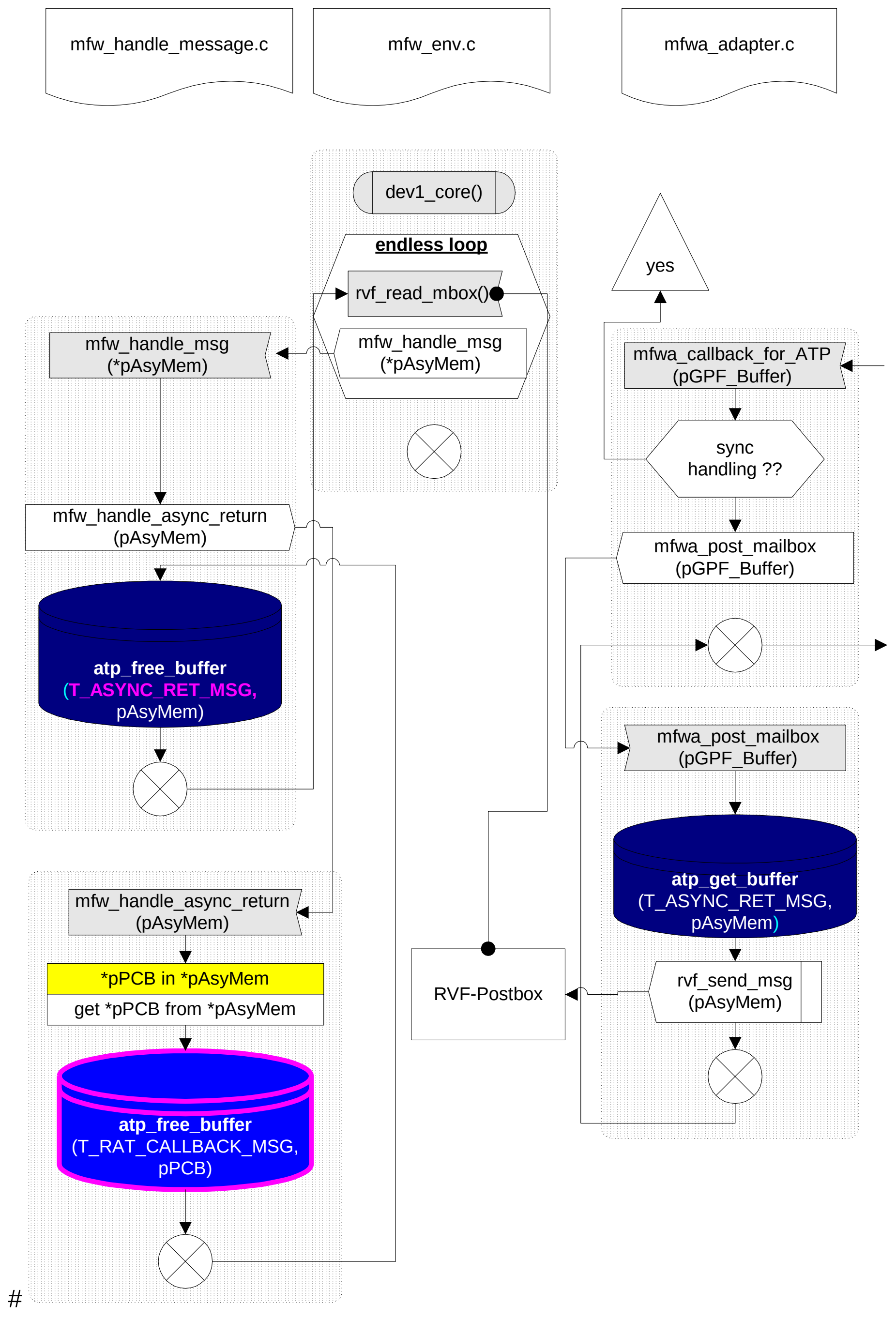
The MFW-Adapter wrapper is the responsible to adapt all ‘sAT…’, ‘qAT…’ an d ‘tAT..’ remote procedure calls toward the ATP/ACI. The task decoupling for this case will be done by ACI-Adapter.
A task protection mechanism provides that only one downward AT-command can be processed by ACI. Currently, none of clients which was subscribed onto SMBS can send a SMS. The sending mechanism will not be support in the SMBS yet.

3 Detailed description

3.1 Asynchronous handling

MFW-Message Handler and MFWA-Adapter will do the entire asynchronous handling.

Based on the current state, MFW handle different kind of messages at a different way. For instance all SMS related messages would be forward to the SMBS.


mfw_handle_message.c
mfw_env.c
mfwa_adapter.c
dev1_core()
endless loop
rvf_read_mbox()
mfw_handle_msg (*pAsyMem)
yes
mfw_handle_msg (*pAsyMem)
mfwa_callback_for_ATP (pGPF_Buffer)
sync handling ??
mfw_handle_async_return (pAsyMem)
mfwa_post_mailbox (pGPF_Buffer)
atp_free_buffer (T_ASYNC_RET_MSG, pAsyMem)
mfwa_post_mailbox (pGPF_Buffer)
atp_get_buffer (T_ASYNC_RET_MSG, pAsyMem)
mfw_handle_async_return (pAsyMem)
*pPCB in *pAsyMem
get *pPCB from *pAsyMem
RVF-Postbox
rvf_send_msg (pAsyMem)
atp_free_buffer (T_RAT_CALLBACK_MSG, pPCB)
#

3.1.1 SMS related asynchronous messages

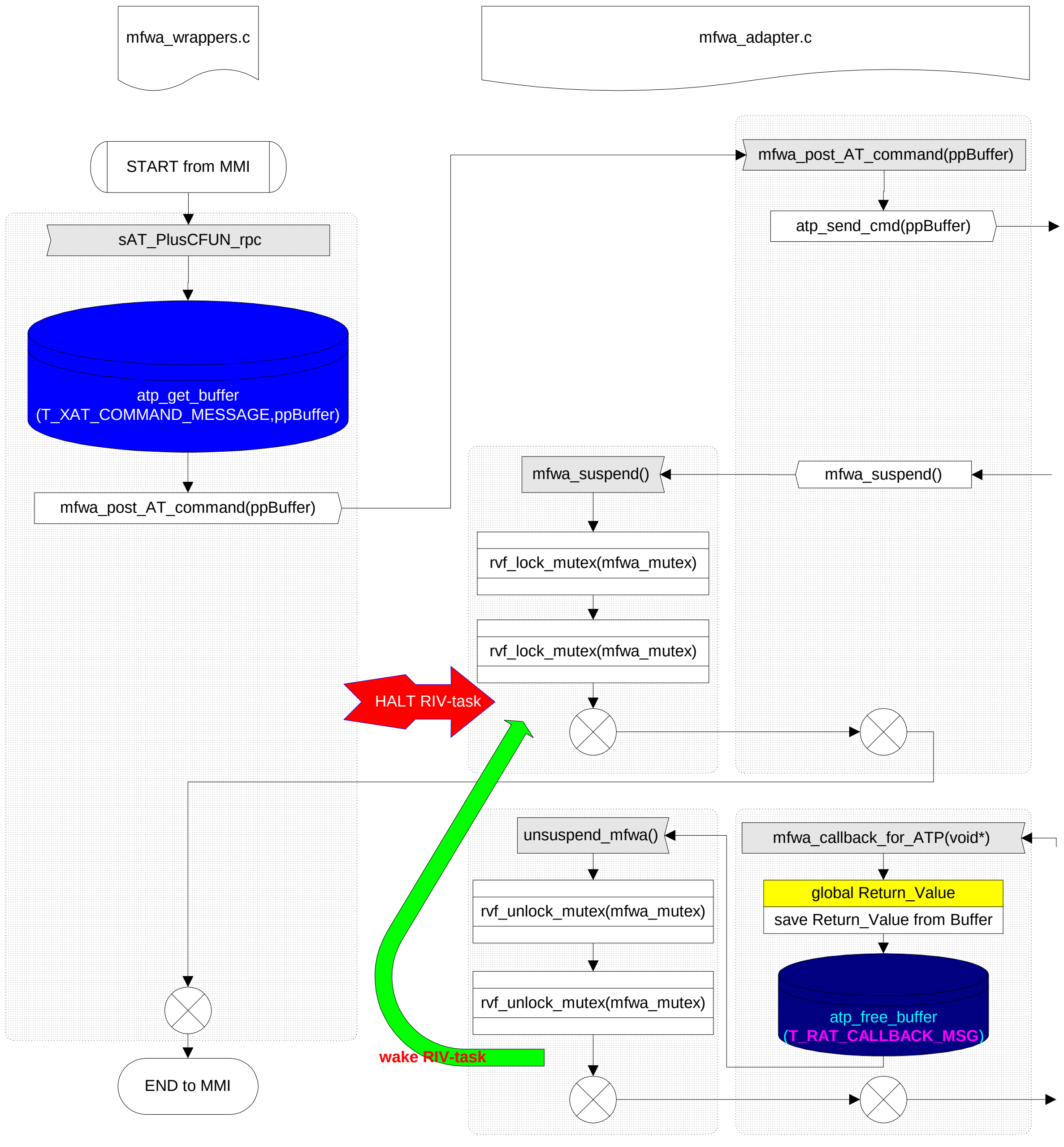
Till then, all SMS related messages were done by the SMS part of MFW. Some of these functions have to be developing at the SMBS side also. At the SMBS side all rAT…-commands will have ending by "_smbs" instead of "_mfw". All commands will not be called from Message-Handler anymore to MFW. The Message Handler will call the according function in the SMBS.

asynchronous message	parameter	meaning	available on SMBS - API	changes in MFW-SMS ??
rAT_PlusCDS_mfw	T_ACI_CDS_SM *st	Receive a status report message	No	no changes on MFW
rAT_PlusCMGC_mfw	UBYTE mr	a mobile originated short message command was sent successful	No	no changes on MFW
rAT_PlusCMGD_mfw	None	A SMS was wipe out successful from SIM	Yes	Adding of store type were necessary
rAT_PlusCMGL_mfw	T_ACI_CMGL_SM * smLst	Response of read SMS list from SIM or ME	Implement on SMBS: - answer to +CMGL=<…> - informs all clients about stored SMS on the SIM or ME	None
rAT_PlusCMGR_mfw	T_ACI_CMGL_SM *sm T_ACI_CMGR_CBM *cbm	Response of read a SMS from ACI or ME	Implement on SMBS: Receive all SMS to try distribute to a client or MFW-SMS	None
rAT_PlusCMGS_mfw	UBYTE mr UBYTE numSeg	Response of a successful sending SMS	No, because its not foreseen that a client can be send SMS's	None
rAT_PlusCMGW_mfw	UBYTE index UBYTE numSeg	Response that a SMS was successful saved on ME or SIM	No, because its not foreseen that a client can be stored a SMS to memory	None
rAT_PlusCMS_mfw	Command-ID Error-reason SMS Error-reason conc.SMS	Report the error result on failures of the last executed functions	No ???	None
rAT_PlusCMSS_mfw	UBYTE mr UBYTE numSeg	Response of a successful sending of a stored SMS	No, because its not foreseen that a client can be send SMS's	None
rAT_PlusCMT_mfw	T_ACI_CMGL_SM * sm	Receive a **new** SMS	Implement on SMBS: Receive this SMS to distribute to a client or MFW-SMS	None
rAT_PlusCMTI_mfw	T_ACI_SMS_STOR mem UBYTE index T_ACI_CMGL_SM *sm	Receive a **indexed** SMS	Implement on SMBS: Receive this SMS to distribute to a client or MFW-SMS	None
rAT_PlusCPMS_mfw	T_ACI_SMS_STOR_OCC *mem1 T_ACI_SMS_STOR_OCC *mem2 T_ACI_SMS_STOR_OCC *mem3	Call back for set the preferred SMS memory	No, because its not foreseen that a client can be set the preferred memory	None
rAT_sms_ready_mfw	none	finished initialize from the SIM card	Yes (indicates the client "this SMS is	Yes, (Implement a additional

asynchronous message	parameter	meaning	available on SMBS - API	changes in MFW-SMS ??
		after switch on or change of SiM card	old/new" in case of receiving a SMS)	BIT in SMS-structure)
rAT_SignalSMS_mfw	state	Informs the caller about the state of initialisation	Yes (indicates the client "this SMS is old/new" in case of receiving a SMS)	Yes, (Implement a additional BIT in SMS-structure)

3.2 synchronous handling

The MFWA-Wrapper is the interface substitution. It illuminates the ACI-interface with remote procedure calls. The MFWA-Adapter provide the task-decoupling mechanism between GPF and RIVIERA Task. Also in synchronous case, the MFWA-Adapter does not reentrance-able. That means the RIVIERA-task is still waiting for an (synchronous) answer from ACI.


mfwa_wrappers.c
mfwa_adapter.c
START from MMI
sAT_PlusCFUN_rpc
atp_get_buffer
(T_XAT_COMMAND_MESSAGE,ppBuffer)
mfwa_post_AT_command(ppBuffer)
mfwa_post_AT_command(ppBuffer)
atp_send_cmd(ppBuffer)
mfwa_suspend()
mfwa_suspend()
rvf_lock_mutex(mfwa_mutex)
rvf_lock_mutex(mfwa_mutex)
HALT RIV-task
unsuspend_mfwa()
mfwa_callback_for_ATP(void*)
rvf_unlock_mutex(mfwa_mutex)
global Return_Value
save Return_Value from Buffer
rvf_unlock_mutex(mfwa_mutex)
atp_free_buffer
(T_RAT_CALLBACK_MSG)
wake RIV-task
END to MMI

3.2.1 SMS related synchronous messages

Currently it is not foreseen that the clients can be support the entire catalogue is ‘sAT_…’-commands. Only the ‘sAT_…’-requests for delete of a SMS and list all SMS’s located on the SIM will be supported for this moment. Some other ‘sAT_…’-commands hast to be adapt by adding of an new parameter called ‘**T_UDH_DATA *sUDH’** to inform the ACI about the user data header which is located in the concatenated SMS.

asynchronous message	parameter	meaning	available on SMBS - API	changes in MFW-SMS ??
sAT_PlusCMGD_rpc	T_ACI_CMD_SRC srcId UBYTE index	Request to wipe aou a SMS from ME or SIM	implement on SMBS	no changes on MFW
sAT_PlusCMGL_rpc	T_ACI_CMD_SRC srcId T_ACI_SMS_STAT state SHORT startIdx T_ACI_SMS_READ rdMode	Request a list of all available SMS on the SIM or ME	none	No changes on MFW
sAT_PlusCMGS_rpc	T_ACI_CMD_SRC srcId CHAR *da T_ACI_TOA *toda T_SM_DATA_EXT *src_data CHAR *sca T_ACI_TOA *tosca SHORT isReply **T_UDH_DATA *sUDH**	Send of an (conc.) SMS	none	Yes, insert new UDH-Information structure
sAT_PlusCMGD_rpc	T_ACI_CMD_SRC srcId UBYTE index **T_UDH_DATA *sUDH**	Delete request for a (conc.) SMS	none	Yes, insert new UDH-Information structure
sAT_PlusCMGW_rpc	T_ACI_CMD_SRC srcId SHORT index CHAR *address T_ACI_TOA *toa T_ACI_SMS_STAT stat UBYTE msg_ref T_SM_DATA_EXT *src_data CHAR *sca T_ACI_TOA *tosca SHORT isReply **T_UDH_DATA *sUDH**	Write message to memory	none	Yes, insert new UDH-Information structure
sAT_PlusCMGC_rpc	T_ACI_CMD_SRC srcId SHORT fo SHORT ct SHORT pid SHORT mn	Send command	none	Yes, insert new UDH-Information structure

asynchronous message	parameter	meaning	available on SMBS - API	changes in MFW-SMS ??
	CHAR* da T_ACI_TOA* toda T_ACI_CMD_DATA* data **T_UDH_DATA *sUDH**			
sAT_PlusCMSS_rpc	T_ACI_CMD_SRC srcId UBYTE index CHAR* da T_ACI_TOA* toda	sending a short message from mem-ory	None	Yes, insert new UDH-Information structure

3.3 SMBS as RIVIERA entity

3.3.1 RIVIERA type

The SMBS will be develop as the RIVIERA-entity #1. This means:

- No timer can be supported
- No task is running in the SMBS
- SMBS is not be able to handle with RIVIERA-Messages by using *rvm_send_msg()*

3.3.2 File naming

According to the ‘RIVIERA development guide’ this entity will comprise following files:

smbs_api.h	➔ files gathering all the definition / prototyping needed to use the service of the SW entity
smbs_api.c	➔ files gathering all the code needed to use the service of the SW entity
smbs_env.c	➔ coding of the Riviera Generic Functions (except handle message and handle timer)
smbs_env.h	➔ definition related to _env.h
smbs_i.h	➔ internal definitions of SMBS
smbs_handle_message.c	➔ coding of the entire behavior function

makefile_changes: ????

3.4 Interface description

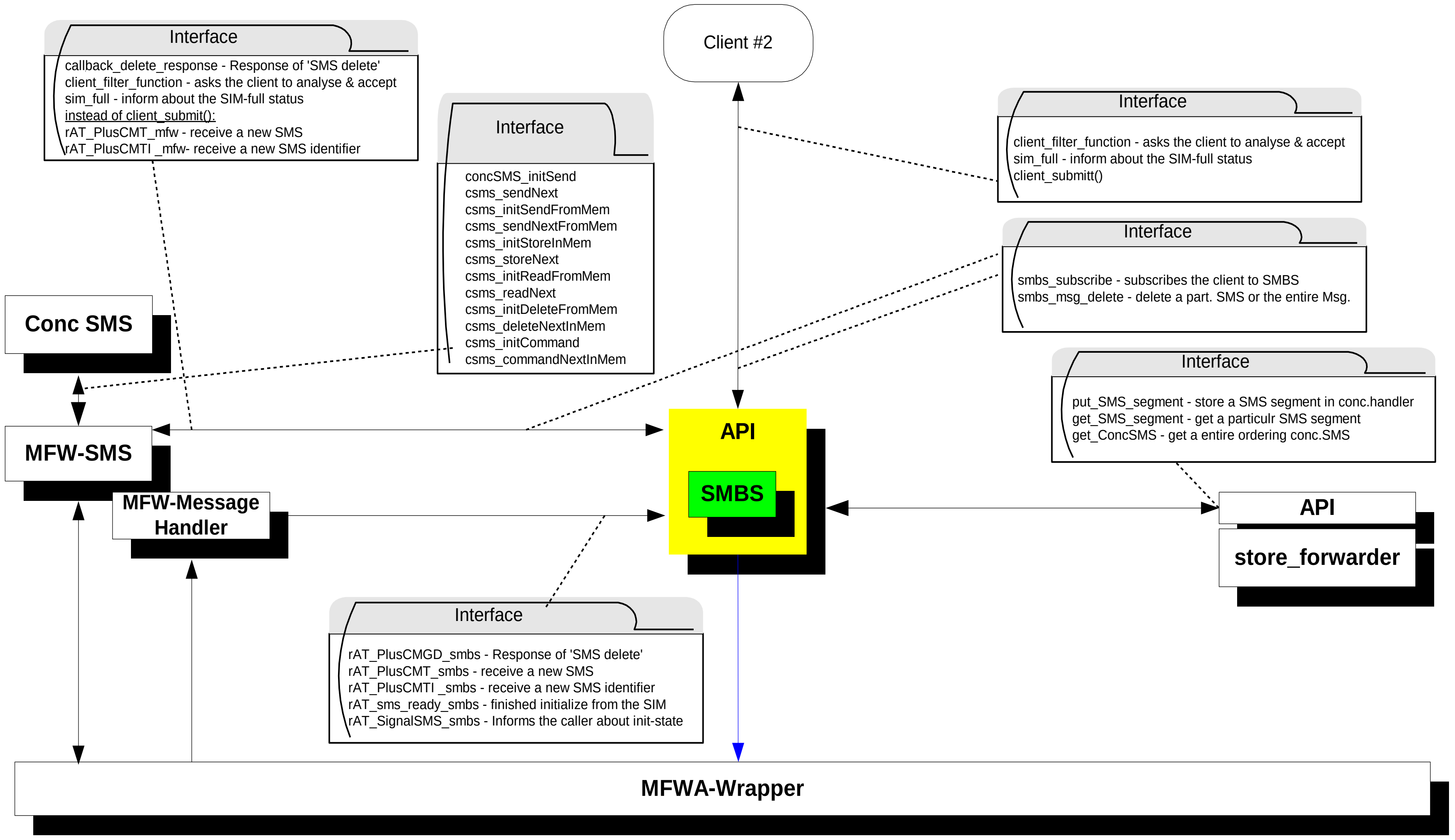
This chapter describes the interfaces between the clients and the concatenation handler.

3.4.1 Standard Interface

The standard RIVIERA interfaces for RIVIERA entity #1 will be supported. These functions will be called from the RIVIERA-Manager:

- smbs_start()	→ instantiate the SMBS
- smbs_init	→ initialize the SMBS
- smbs_stop()	→ halt of the SMBS
- smbs_kill()	→ remove the SMBS
- smbs_set_info()	→ set some pieces of information regarding callbacks, error handling..
- smbs_get_info()	→ retrieve the information which was set at the smbs_set_info()

3.4.2 Specified interface


Interface
callback_delete_response - Response of 'SMS delete'
client_filter_function - asks the client to analyse & accept
sim_full - inform about the SIM-full status
instead of client_submit():
rAT_PlusCMT_mfw - receive a new SMS
rAT_PlusCMTI _mfw- receive a new SMS identifier
Interface
concSMS_initSend
csms_sendNext
csms_initSendFromMem
csms_sendNextFromMem
csms_initStoreInMem
csms_storeNext
csms_initReadFromMem
csms_readNext
csms_initDeleteFromMem
csms_deleteNextInMem
csms_initCommand
csms_commandNextInMem
Client #2
Interface
client_filter_function - asks the client to analyse & accept
sim_full - inform about the SIM-full status
client_submitt()
Interface
smbs_subscribe - subscribes the client to SMBS
smbs_msg_delete - delete a part. SMS or the entire Msg.
Interface
put_SMS_segment - store a SMS segment in conc.handler
get_SMS_segment - get a particulr SMS segment
get_ConcSMS - get a entire ordering conc.SMS
Conc SMS
MFW-SMS
MFW-Message Handler
API
SMBS
API
store_forwarder
Interface
rAT_PlusCMGD_smbs - Response of 'SMS delete'
rAT_PlusCMT_smbs - receive a new SMS
rAT_PlusCMTI _smbs - receive a new SMS identifier
rAT_sms_ready_smbs - finished initialize from the SIM
rAT_SignalSMS_smbs - Informs the caller about init-state
MFWA-Wrapper

3.4.3 Interface to MFW-SMS and client(s)

3.4.3.1 client_filter_function

```
T_SMBS_RET  client_filter_function (T_SMBS_MSG_FILTER info)
```

Description

This filter function asks the client to analyse and accept the SMS based message provided in the info parameter.

Parameters

- T_SMBS_MSG_FILTER info

The SMS will be delivered in this structure. It is similar to the same format as it arrives to SMBS from ACI, except the additional BYTE to indicate arriving SMS while switch-on operation.
In case of delivering concatenated SMS, the plain user data will be consist at the 'data'-field. Only the complete user data header of the first SMS segment will be stored in the user data header field.
This parameter is an output value of the Concatenation Handler.

```
#define T_SMBS_MSG_FILTER T_SMBS_MT

typedef struct
{
  UBYTE              mt_item_nb; /* always set to 1 for TCS 2.1.1; for future concatenation use */
  T_SMBS_MT_ITEM     mt_item[MAX_MSG_LEN/BASIC_MAX_MSG_LEN]; /* only first item is valid */
}T_SMBS_MT;                                                  /* for  TCS 2.1.1 */

typedef struct
{
  UBYTE              mem;                      /* memory of message            */
  UBYTE              index;                    /* index of message             */
  T_SMBS_ADDR        orig_addr;                /* call party address           */
  CHAR               sc_addr[MAX_LEN];         /* service centre address       */
  UBYTE              prot_id;                  /* protocol identifier          */
  SHORT              dcs;                      /* data coding scheme           */
  UBYTE              rp;                       /* reply path                   */
  UBYTE              ti;                       /* type indicator               */
  T_SMBS_SCTP        sctp;                     /* service centre timestamp     */
  USHORT             msg_len;                  /* length of short message      */
  CHAR               sms_msg[BASIC_MAX_MSG_LEN];      /* short message not decoded,
refer to dcs         */
  UBYTE              udh_len;                  /* length of user data header = 0 if
UDHI = 0      */
  UBYTE              udh[BASIC_MAX_MSG_LEN];   /* user data header             */
  UBYTE              startup_phase;            /* true(<>0) at phone power up  */
}T_SMBS_MT_ITEM;
```

Types:

```
/* subscribe contants */
typedef enum
{
  SMBS_UNKNOWN = 0,
  SMBS_DIGIPLUG,
  SMBS_MMS_PUSH
} E_SMBS_TYPE;
```

```
/* asynchronous events (msgid) contants SMBS_USE_ID
to be defined */
#define SMBS_MESSAGE_OFFSET
BUILD_MESSAGE_OFFSET(SMBS_USE_ID)
/* Message IDs */
#define SMBS_UNKNOWN_MESSAGE
(SMBS_MESSAGE_OFFSET | 0x000)
#define SMBS_MT_MESSAGE
(SMBS_MESSAGE_OFFSET | 0x001)
```

```
#define SMBS_OK_MESSAGE
(SMBS_MESSAGE_OFFSET | 0x100)
#define SMBS_ERROR_MESSAGE
(SMBS_MESSAGE_OFFSET | 0x101)

/* deliver constants */
#define MAX_SCTP_DIGITS    2
#define MAX_LEN            21
#define SMBS_NUM_LEN       41
#define SMBS_TAG_LEN       21
#define MAX_MSG_LEN        1601 //Max
size for a SMS, configurable
#define BASIC_MAX_MSG_LEN 160
#define MAX_MSG_LEN_ARRAY
(MAX_MSG_LEN+1) //Array size for SMS

typedef INT8 T_SMBS_RET;

/* subscription types */
typedef T_SMBS_RET (*T_SMBS_FFP)
(T_SMBS_MSG_FILTER); /* SMBS filter func-
tion pointer */
typedef struct
{
  T_SMBS_FFP filter_function;
  E_SMBS_TYPE smbs_type;
}T_SMBS_SUBSCRIBE;


/* deliver types */
typedef struct
{
  CHAR  tag[SMBS_TAG_LEN];
  UBYTE len; /* length of name */
  UBYTE dummy1;
  UBYTE dummy2;
  UBYTE dummy3;
  CHAR  number[SMBS_NUM_LEN];
} T_SMBS_ADDR;
```

```
typedef struct
{
  UBYTE year     [MAX_SCTP_DIGITS];
  UBYTE month    [MAX_SCTP_DIGITS];
  UBYTE day      [MAX_SCTP_DIGITS];
  UBYTE hour     [MAX_SCTP_DIGITS];
  UBYTE minute   [MAX_SCTP_DIGITS];
  UBYTE second   [MAX_SCTP_DIGITS];
  SHORT timezone;
} T_SMBS_SCTP;

/* message delete types */
typedef struct
{
  UBYTE              mem; /* memory of msg */
  UBYTE              index;
}T_SMBS_MSG_DELETE;

typedef union
{
  T_SMBS_MT          sms_mt;
  T_SMBS_MSG_DELETE sms_del;
} T_SMBS_MSG_DELIVER;
typedef struct
{
  T_RV_HDR r_hdr;
  T_SMBS_MSG_DELIVER body;
}T_SMBS_MSG;

/* filtering types */
typedef struct
{
  T_SMBS_MT   preview;
}T_SMBS_MSG_FILTER;
```

Immediate Return

- **T_SMBS_RET**

The possible values are:

Value	Id	Definition
0	SMBS_OK	The SMS is accepted.
-3	SMBS_FORMAT_ERROR	The SMS has an unrecognized format

While called, the Client filter_function shall not be blocking (does not really need to be re-entrant since only called by SMBS entity) and shall exit as soon as the analysis is performed. SMS based message copy will have to be performed after having received the message through the return path mechanism.

Event Return

N/A

3.4.3.2 smbs_subscribe

```
T_SMBS_RET  smbs_subscribe (T_SMBS_SUBSCRIBE info, T_RV_RETURN return_path)
```

Description

This function is called to subscribe to the Riviera Short Message Bearer server. This function may be called several times. A NULL return_path is equivalent to an unsubscribe request.

Parameters

- **T_SMBS_SUBSCRIBE**

Below the detail of each parameters:

The T_FILTER_FUNCTION_POINTER describes the callback pointer to the filter function. This function has to be called from the SMSB of a SMS based messages or a plain SMS will arrive to the client. With the E_SMBS_TYPE the client will tell the SMBS the service to differentiate the clients.

```
typedef struct
{
  T_FILTER_FUNCTION_POINTER *pfFilterFunction;
  E_SMBS_TYPE                uiSMBS_Type;
} T_SMBS_SUBSCRIBE;
```

- meaning of bSIM_FullMessage:

value	Id	Definition
FALSE	-	The client don't wants to be inform about a SIM-Full occurrence
TRUE	-	The client wants to be inform about a SIM-Full occurrence

- **T_RV_RETURN**

Specifies the standard RIVIERA return path used for unsolicited SMS based messages notifications such as answers of sAT-commads. The client will be deciding whether this is assigned to itself or possibly for a other client.

Immediate Return

- **T_SMBS_RET**

The possible values are:

value	Id	Definition
0	SMBS_OK	The API function was successfully executed.
-2	SMBS_TYPE_ALREADY_SUBSCRIBED	A subscription for the same SMBS message type has already been received and accepted. Unsubscribe first.

Event Return

- **SMBS_MT_MESSAGE**

The SMS based message is received with the `T_SMBS_MSG` type. The `MsgId` parameter of the Riviera header is set to `SMBS_MT_MESSAGE`. The message body union will provide the `T_SMBS_MT` field. This field will contain the SMS based message. The nested couple `(mem,index)` of parameters will be the SMS based message reference to use for deleting it.

The possible values of *msgid* are:

value	Id	Definition
Bit field	SMBS_MT_MESSAGE	Mobile Terminated SMS based message indication

Memory usage: when the return path provides an addr_id as reply method, then a Riviera buffer is created from the SMBS bank. This self containing piece of memory (header, parameters and nested pointers if any) has to be deleted by the Client as soon as possible through the rvf_free_buf() function or has to be moved into the Client's bank through the rvf_count_buf() function. When a function callback is provided, the message will be accessible only within the scope of the function call.

Initialization: once the SMBS SW entity initialization is complete, the Client will receive all the SMS based messages still stored in the ME or SIM memory and ready to be deliver upon subscription. It provides a way to synchronize the SMBS SW entity and its Clients. Afterwards, the Clients will be able to delete old and unwanted messages.
The startup_phase flag provides the information about the SMBS processing phase (power up or after power up). However, this flag doesn't reflect the Read/Unread status of an SMS based message. Indeed, one SMS based message can be received by a client during the startup phase though this one was not known by the Client from last subscription session (received during power down/battery out phase)

Current restriction of use

none

smbs_msg_delete

```
T_SMBS_RET  smbs_msg_delete (T_SMBS_MSG_DELETE sInfo, T_RV_RETURN_PATH
            pfReturnPath)
```

Description

This function is called to delete the SMS parts of the received SMS based message. Only the SMS based message identifier is provided. The return path implementation is optional: a NULL value means no deletion confirmation will be sent.

Parameters

- **T_SMBS_MSG_DELETE sInfo**

Below the detail of each parameters:

The following structure described in which memory area the SMS-segment hast to be deleted.

```
typedef struct
{
  UBYTE      uiIndex;      // Index of to be deleted SMS-segment
  T_MEM_TYPE uiMemoryType; // the memory type
} T_SMBS_MSG_DELETE;
```

- **T_RV_RETURN_PATH pfReturnPath**

Specifies the return path used for SMS based messages deletion confirmations.

Immediate Return

- **T_SMBS_RET**

The possible values are:

Value	Id	Definition
0	SMBS_OK	The SMS based message deletion is in progress
-4	SMBS_INDEX_ERROR	The SMS based message index is unknown

Event Return

- **SMBS_OK_MESSAGE**

The SMS based message is received with the `T_SMBS_MSG` type. The `MsgId` parameter of the Riviera header is set to `SMBS_OK_MESSAGE`. The message body union will provide the `UBYTE index` field. This `index` parameter will be the SMS based message reference having successfully been deleted

The possible values of *msgid* are:

value	Id	Definition
Bit field	SMBS_OK_MESSAGE	Generic successful execution report
Bit field	SMBS_ERROR_MESSAGE	Generic failing execution report

Memory usage: when the return path provides an addr_id as reply method, then a Riviera buffer is created from the SMBS bank. This self containing piece of memory (header, parameters and nested pointers if any) has to be deleted by the Client as soon as possible through the rvf_free_buf() function or has to be moved into the Client's bank through the rvf_count_buf() function. When a function call-back is provided, the message will be accessible only within the scope of the function call.

Current restriction of use

None

sim_full

```
T_SMBS_RET  sim_full ()
```

Description

This function is called to inform each client if needed about the appearance of a SIM-full.

Parameters

none

Immediate Return

- **T_SMBS_RET**

The possible values are:

Value	Id	Definition
0	SMBS_OK	All clear
-2	SMBS_ERROR	Someone error occurs

Event Return

none

Current restriction of use

none

3.4.4 Interface to MFW-Message Handler

The SMBS provides a interface to the MFW-Message Handler regarding
- receiving of new SMS
- responses to delete SMS requests
- status messages about to inform the client about the init process

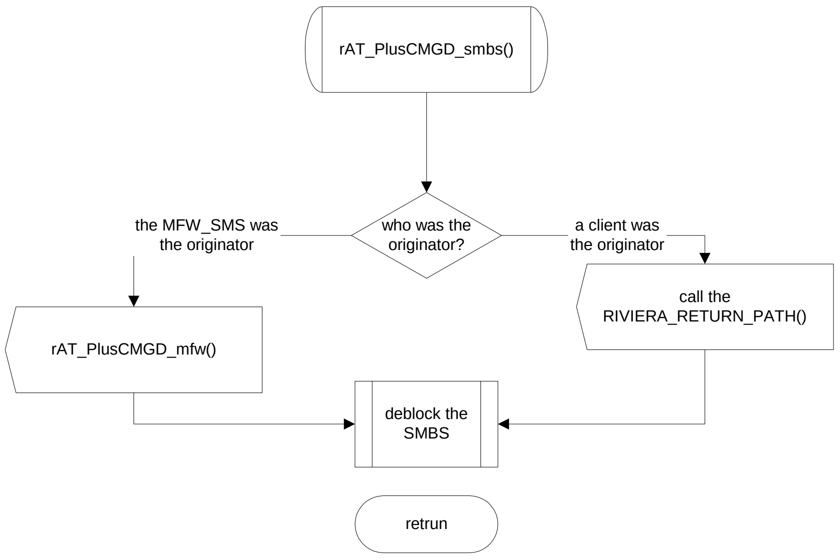
All messages are similar to the former, legacy messages which also available at the MFW-SMS.
For detailed description. please see to the MFW-SMS entity description.

```
GLOBAL void rAT_PlusCMGD_smbs()
```

- Response to 'SMS delete' – request

Behavior

The SMBS decides who the originator of this ‘delete_response’ was. If it’s a client then send this response to a client by using the return-path which was keep after the delete request. If it is the MFW-SMS then relay this message to the MFW-SMS by calling the ‘GLOBAL void rAT_PlusCMGD_mfw()’ function.


rAT_PlusCMGD_smbs()
the MFW_SMS was the originator
who was the originator?
a client was the originator
call the RIVIERA_RETURN_PATH()
rAT_PlusCMGD_mfw()
deblock the SMBS
retrun

The following function does not be implement at first stage, because the answer of an SMS read-request from SIM will be let forward to MFW-SMS:

```
GLOBAL void rAT_PlusCMGR_smbs(T_ACI_CMGL_SM  *sm, T_ACI_CMGR_CBM *cbm)
```

- Response of 'read SMS'

Behavior

After receiving of a new message the SMBS will be block itself, so all further incoming messages will be refused. The SMBS will try to foist the message to any client or MFW-SMS by using the client filter_function(). If a client this message accept, it will be send this to it. If no client wants this message it will be stored into the 'store_and_forwarder'-handler. Further see MSC in 4.3.2.

```
GLOBAL void rAT_PlusCMT_smbs(T_ACI_CMGL_SM*  sm)
```

- receive a new SMS

Behavior

see 'rAT_PlusCMGR_smbs' above.

```
GLOBAL void rAT_PlusCMTI_smbs(T_ACI_SMS_STOR mem, UBYTE index,
                              T_ACI_CMGL_SM *sm)
```

- receive a new SMS identifier

Behavior

see 'rAT_PlusCMGR_smbs'

```
GLOBAL void rAT_sms_ready_smbs(void)
```

- Finished initialize from the SIM

Behavior

After receiving this message the information will be keep in a SMBS-variable. This variable will use to inform the client about the MS is in switch-on or not.

```
GLOBAL void rAT_SignalSMS_smbs(void)
```

- Informs the caller about init-state

Behavior

see 'rAT_sms_ready_smbs'

3.4.5 Interface to ‘store and forwarder’ (SANDF)

The interface to the store and forwarder handler (SANDF) keeps the function calls to store particular SMS-segment’s to and to pick up entire concatenated SMS as chain-list.

3.4.5.1 sandf_store_SMS_Segment

```
CONC_RET sandf_store_SMS_Segment(T_SMBS_MSG_CHAIN sSMS_chain)
```

Description

This function stores an SMS-segment inside the store and forwarder handler. The segment will be stored inside the ‘store and forwarder handler’ and it includes the user data header of each segment also.

Parameters

See client_filter_function

Immediate Return

- **SANDF_RET**

```
typedef struct
{
  UBYTE uiIndex;    // index inside the concatenation handler
  BOOL  bComplete; // ‘True’ → the Conc SMS is now complete
} SANDF_RET;
```

The possible values for uiIndex in SANDF_RET are:

Value	Id	Definition
-1	SANDF_ERROR	A error was occurred while storing the SMS-segment
>0	Specifies the Index in store and forwarder Handler	The Index

The possible values for bComplete in CONC_RET are:

Value	Id	Definition
0	FALSE	After storing this segment, the corresponding SMS-based message are now complete
1	TRUE	After storing this segment, the corresponding SMS-based message are not complete yet

Event Return

none

Current restriction of use

none

3.4.5.2 sandf_pick_concSMS

```
T_SMBS_MSG_CHAIN sandf_pick_concSMS(UBYTE uiIndex)
```

Description

This function pick a SMS-segment located inside the 'store and forwarder'-handler.

Parameters

- **UBYTE uiIndex**

This parameter describes the internal Index to locate the requested entire conc. SMS inside the 'store and forwarder'-handler.

Immediate Return

- **T_SMBS_MSG_CHAIN**

`See` client_filter_function

The possible values for **T_SMBS_MSG_CHAIN** are:

Value	Id	Definition
NULL	SANDF_ERROR	A error was occurred while pick the conc.SMS - perhaps, it is not a entire conc.SMS (some elements left)
valid pointer	Pointer to struct	This *struct is the first element of the entire conc.SMS

Event Return

none

Current restriction of use

none

3.5 Changes on existing code

3.5.1 Changes on MFW-Message Handler

The MFW-Message Handler (file ‘mfw_handle_message.c’) was designed to distribute and adapt the asynchronous messages to the MFW.

Now, some message must be redirect to SMBS in the MFW-Message Handler function `‘handle_async_return()’`. At the first step, only the urgent needed messages will be redirect to SMBS.

- rAT_PlusCMT →change macro `RAT_CASE_PLUS (CMT, ( data_p->sm ))`
- rAT_PlusCMTI →change macro `RAT_CASE_PLUS (CMTI, ( … ))`
- rAT_PlusCMGD →change macro `RAT_CASE_PLUS (CMGD, ( …… ))`

Here the corresponding functions in the SMBS which must be called in the SMBS:

- `void rAT_PlusCMT_smbs(T_ACI_CMGL_SM*  sm);`
- `void rAT_PlusCMTI_smbs(T_ACI_SMS_STOR mem, UBYTE index, T_ACI_CMGL_SM* sm);`
- `void rAT_PlusCMGD_smbs();`

Additionally the messages to indicate about ‘switch-on’ and ‘SIM-full’ must also be called by MFW-Message Handler. Unfortunately a SIM-full indication from ACI is still missing, so this message has to be implement later.

3.5.2 Changes on MFW-SMS

3.5.2.1 change init() – function:

In the MFW-SMS must be change the sms_init() function. Here it must be call the subscription function to the SMBS. Also a so called ‘return_path’ as a function-pointer and the corresponding return-function must be implement at The MFW.

```
void MFW_SMBS_callback_func(void *)
{
      // what should I do here ??
}
```

The return_path is defined as follows:

```
typedef struct
{
      T_RVF_ADDR_ID     addr_id;
      void              (*callback_func)(void *);
} T_RV_RETURN_PATH;

void sms_init(void)
{
  T_SMBS_SUBSCRIBE sInfo;
  T_RV_RETURN return_path;

  sInfo.pfFilterFunction = (*client_filter_function) (T_SMBS_MSG_FILTER info);
  return_path.addr_id = MFW_USE_ID;
  return_path.callback_func = MFW_SMBS_callback_func;
  smbs_subscribe (T_SMBS_SUBSCRIBE sInfo, T_RV_RETURN return_path)
  :
  :
}
```

3.5.2.2 insert the filter-function:

At the MFW-SMS it must be implement the filter-function also. The SMBS asks the MFW-SMS whether this SMS is for the MFW-SMS or not.

```
client_filter_function) (T_SMBS_MSG_FILTER info)
{

}
```

3.5.2.3 insert smbs_msg_delete – function:

If the MFW-SMS wants to delete a SMS from the SIM the smbs_msg_delete() – function at the SMBS must be called instead of calling the sAT_CMGD. After receive the delete request in SMBS the SMBS will be relay it to the MFW-Adapter.

Current/New Implementation at MFW-SMS:

```
T_MFW sms_msg_delete(UBYTE index)
{
  TRACE_FUNCTION ("sms_msg_delete()");

  coll_sms_delete = MFW_SMS_WAIT;

  if(sAT_PlusCMGD_rpc(CMD_SRC_LCL, index) NEQ AT_EXCT)<-replace by smbs_msg_delete
    return MFW_SMS_FAIL; /*MC*/
  deleting_index = index;
  return MFW_SMS_OK;
}
```

3.5.3 Interface MFW-SMS ←→ concatenation Handler

3.5.3.1 Introduction

It is necessary to shift the concatenation handler into the MFW because the MFW is now a RIVIERA - entity.

<u>Consequences:</u>

- the ACI don't receive concatenated SMS no more
- the ACI receive parts of SMS-segments now with a user data header

3.5.3.2 Send a conc. SMS (not from memory)

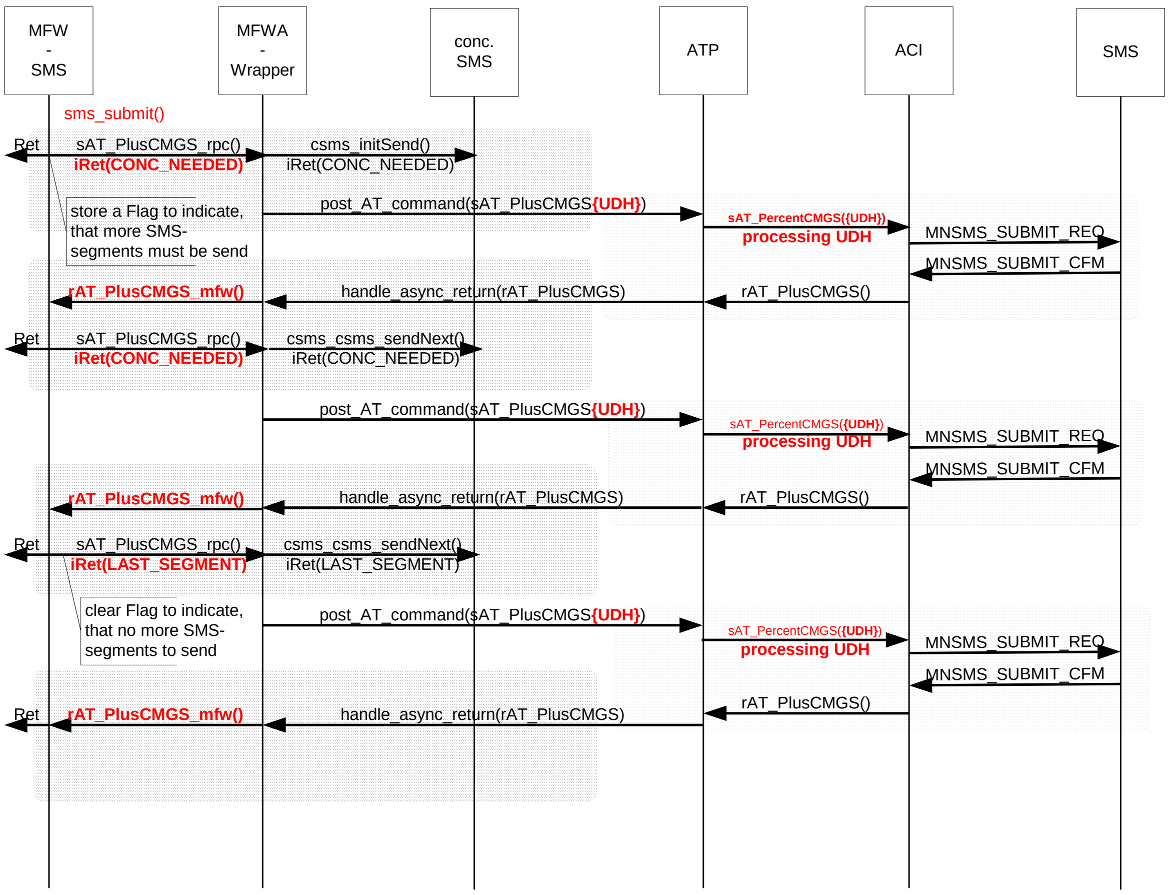
This function initializes shared parameter for CMGS and makes a first splitting of the SMS.

```
EXTERN T_CONC_INIT_RETURN mfw_concsms_init_send ( T_ACI_SM_DATA*  tar_data,
                                                  T_ACI_UDH_DATA* udh,
                                                  T_ACI_CMD_SRC   srcId,
                                                  CHAR*           da,
                                                  T_ACI_TOA*      toda,
                                                  T_SM_DATA_EXT*  src_data,
                                                  CHAR*           sca,
                                                  T_ACI_TOA*      tosca,
                                                  SHORT           isReply,
                                                  UBYTE           alphabet );
```

This function sends the next segment of the SMS.

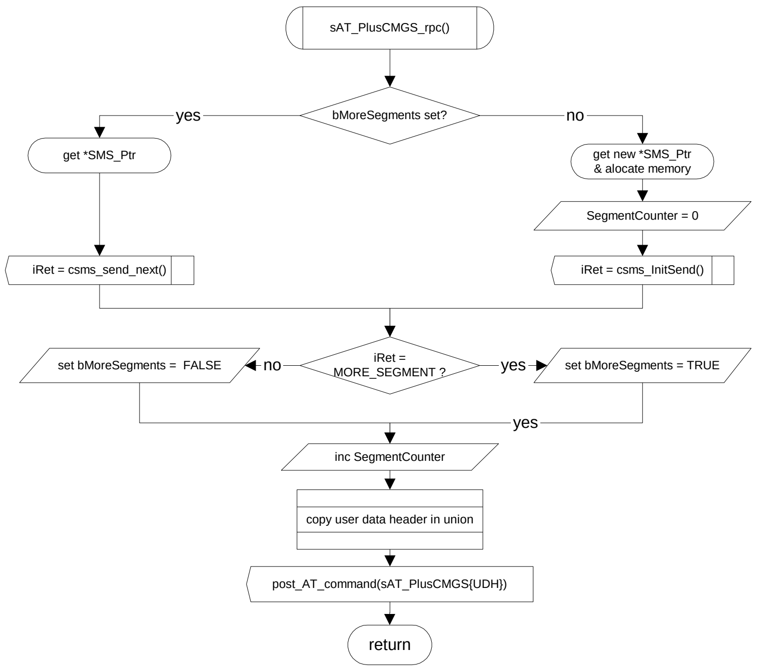
```
EXTERN BOOL mfw_concsms_send_next ( T_ACI_SM_DATA* data,
                                    T_ACI_UDH_DATA* udh,
                                    UBYTE mr,
                                    UBYTE numSeg);
```

3.5.3.2.1 MFW ←→ conc.SMS ←→ ACI while sending of an conc. SMS

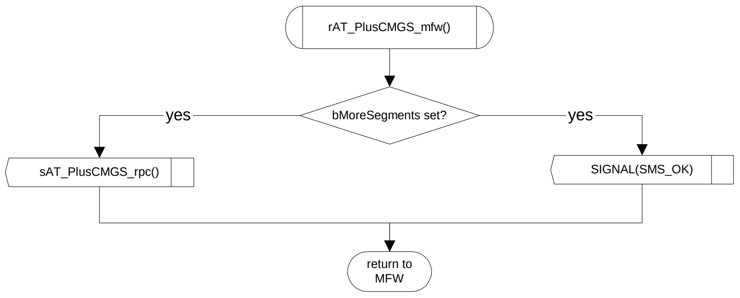


following existing functions must be adapted: (see above in MSC in red color)

MFW: - sAT_PlusCMGS_rpc



- rAT_PlusCMGS_mfw



ACI: - new function `‘sAT_PercentCMGS’` similar to `‘sAT_PlusCMGS`. Changes are strike out and marked in pink.

```
GLOBAL T_ACI_RETURN sAT_PercentCMGS ( T_ACI_CMD_SRC  srcId,
                                      CHAR*          da,
                                      T_ACI_TOA*     toda,
                                      T_SM_DATA_EXT* src_data,
                                      CHAR*          sca,
                                      T_ACI_TOA*     tosca,
                                      SHORT          isReply,
                                      T_ACI_UDH_DATA *udh )
{
  T_CONC_INIT_RETURN ret;
  ~~T_ACI_UDH_DATA udh;~~

  T_ACI_SM_DATA tar_data;
  UBYTE alphabet;

  TRACE_FUNCTION ("sAT_PlusCMGS ()");


  alphabet = cmhSMS_getAlphabetPp ( smsShrdPrm.pSetPrm[srcId]->dcs );

  ~~ret=concSMS_initSend(&tar_data, &udh, srcId, da, toda, src_data, sca,~~
                      ~~tosca, isReply, alphabet);~~

  if (~~ret EQ CONC_NEEDED~~ udh != NULL)
  {
    SET_CONC;
    return sAT_PlusCMGS_Gl(srcId, da, toda, &tar_data, &udh, sca, tosca,
                           isReply, rAT_PlusCMGS, rAT_PlusCMGS);
  }
  ~~if (ret EQ CONC_NOT_NEEDED)~~
  ~~{~~
    ~~return sAT_PlusCMGS_Gl(srcId, da, toda, &tar_data, NULL, sca, tosca,~~
                           ~~isReply, rAT_PlusCMGS, rAT_PlusCMS);~~
  ~~}~~
}
```

3.5.3.3 Send a SMS from memory

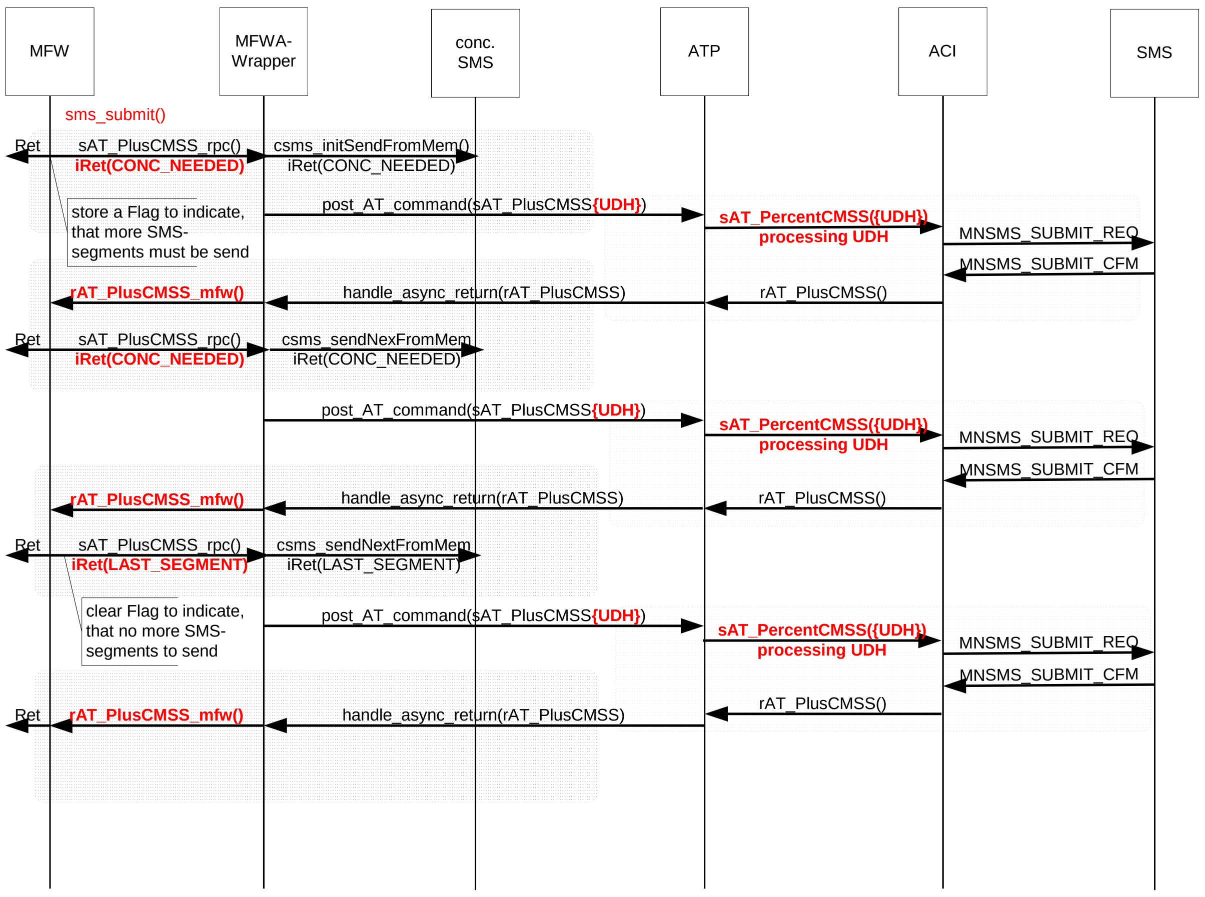
This function initializes shared parameter for CMSS.

```
EXTERN T_CONC_INIT_RETURN mfw_concsms_init_send_from_mem ( T_ACI_CMD_SRC srcId,
                                                           UBYTE         *index,
                                                           CHAR*         da,
                                                           T_ACI_TOA*    toda );
```

This function sends from memory the next segment of the SMS.

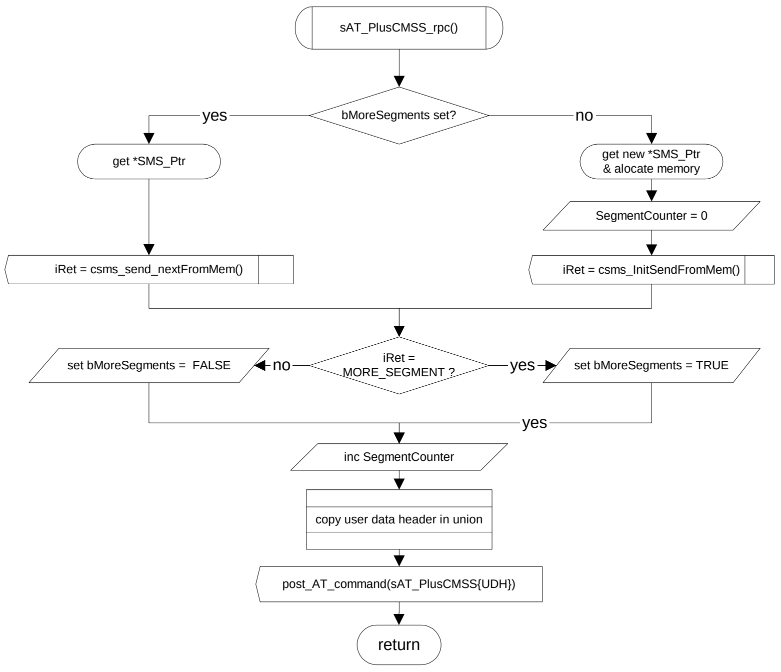
```
EXTERN BOOL mfw_concsms_send_next_from_mem (UBYTE mr, UBYTE numSeg, UBYTE* index);
```

3.5.3.3.1 MFW ←→ conc.SMS ←→ ACI while sending of an conc. SMS from memory

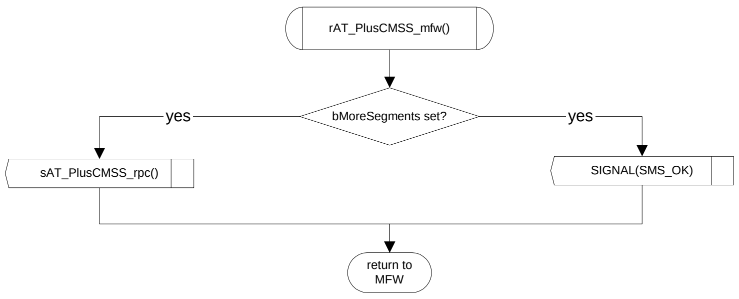


<u>following existing functions must be adapted:</u> (see above in MSC in red color)

<u>MFW:</u> - sAT_PlusCMSS_rpc


sAT_PlusCMSS_rpc()
bMoreSegments set?
yes
no
get *SMS_Ptr
get new *SMS_Ptr & alocate memory
SegmentCounter = 0
iRet = csms_send_nextFromMem()
iRet = csms_InitSendFromMem()
iRet = MORE_SEGMENT ?
no
yes
set bMoreSegments = FALSE
set bMoreSegments = TRUE
yes
inc SegmentCounter
copy user data header in union
post_AT_command(sAT_PlusCMSS{UDH})
return

- rAT_PlusCMSS_mfw



-

<u>new ACI function:</u> - new function `'sAT_PercentCMSS'` similar to `'sAT_PlusCMSS.`

For details see 3.5.3.2.1 MFW ←→ conc.SMS ←→ ACI while sending of an conc. SMS

3.5.3.4 Write SMS in memory

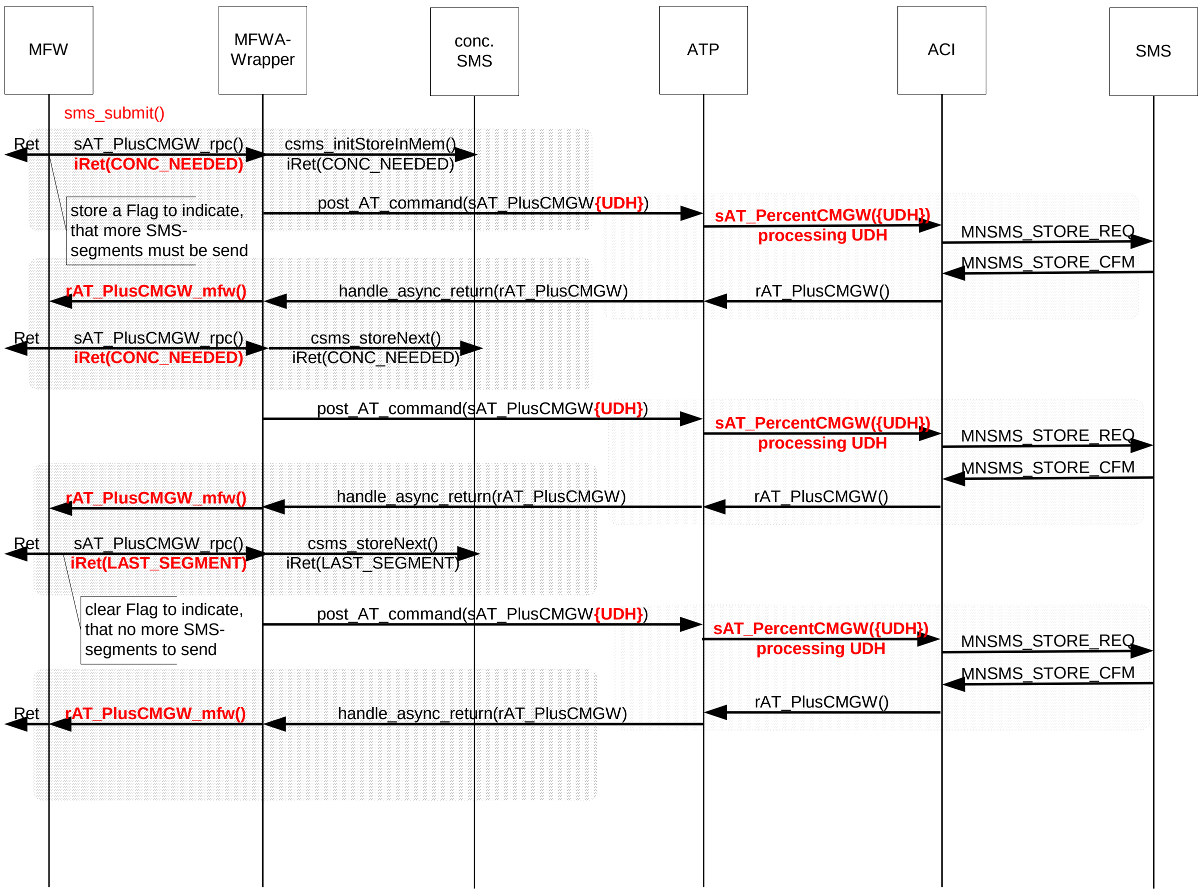
This function initializes shared parameter for CMGW, which writes SMS in memory.

```
EXTERN T_CONC_INIT_RETURN mfw_concsms_init_store_in_mem ( T_ACI_SM_DATA* tar_data,
                                                          T_ACI_UDH_DATA* udh,
                                                          T_ACI_CMD_SRC  srcId,
                                                          SHORT          index,
                                                          CHAR*          address,
                                                          T_ACI_TOA*     toa,
                                                          T_ACI_SMS_STAT stat,
                                                          UBYTE          msg_ref,
                                                          T_SM_DATA_EXT* src_data,
                                                          CHAR*          sca,
                                                          T_ACI_TOA*     tosca,
                                                          SHORT          isReply,
                                                          UBYTE          alphabet );
```

This function stores (writes) the next segment of the SMS in memory.

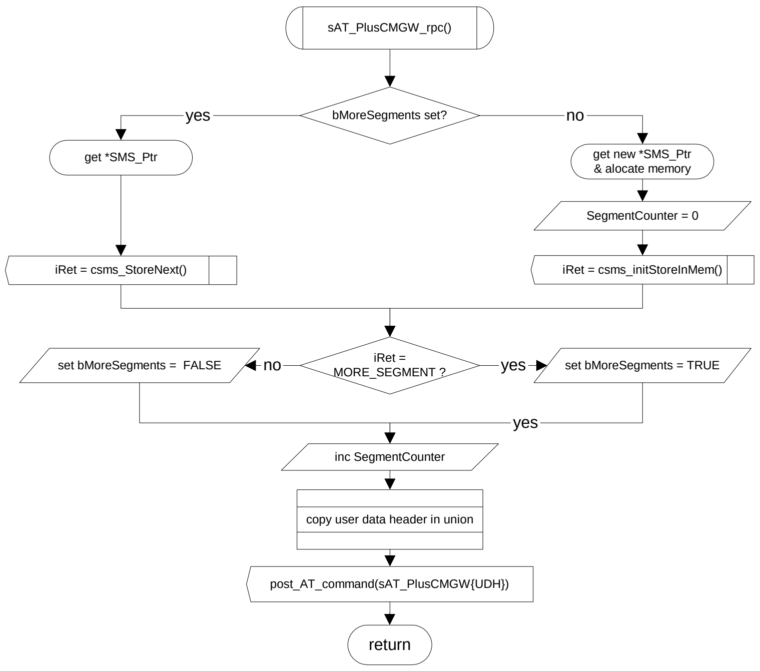
```
EXTERN BOOL mfw_concsms_store_next_in_mem ( T_ACI_SM_DATA* data,
                                            T_ACI_UDH_DATA* udh,
                                            UBYTE index,
                                            UBYTE numSeg );
```

3.5.3.4.1 MFW ←→ conc.SMS ←→ ACI while storing of an conc. SMS from memory

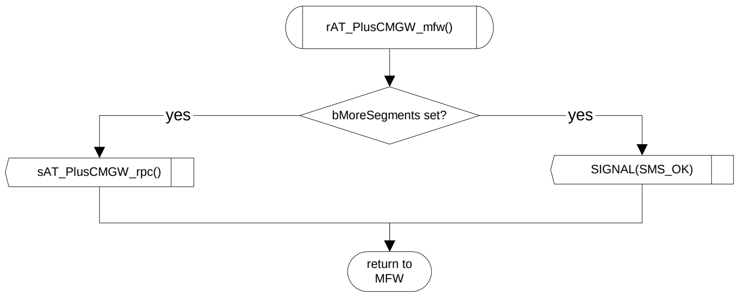


<u>following existing functions must be adapted:</u> (see above in MSC in red color)

<u>MFW:</u> - sAT_PlusCMGW_rpc



- rAT_PlusCMGW_mfw



-

<u>new ACI function:</u> - new function `'sAT_PercentCMGW'` similar to `'sAT_PlusCMGW`. Changes are strike out and marked in pink.

For details see 3.5.3.2.1 MFW ←→ conc.SMS ←→ ACI while sending of an conc. SMS

3.5.3.5 Read SMS from memory

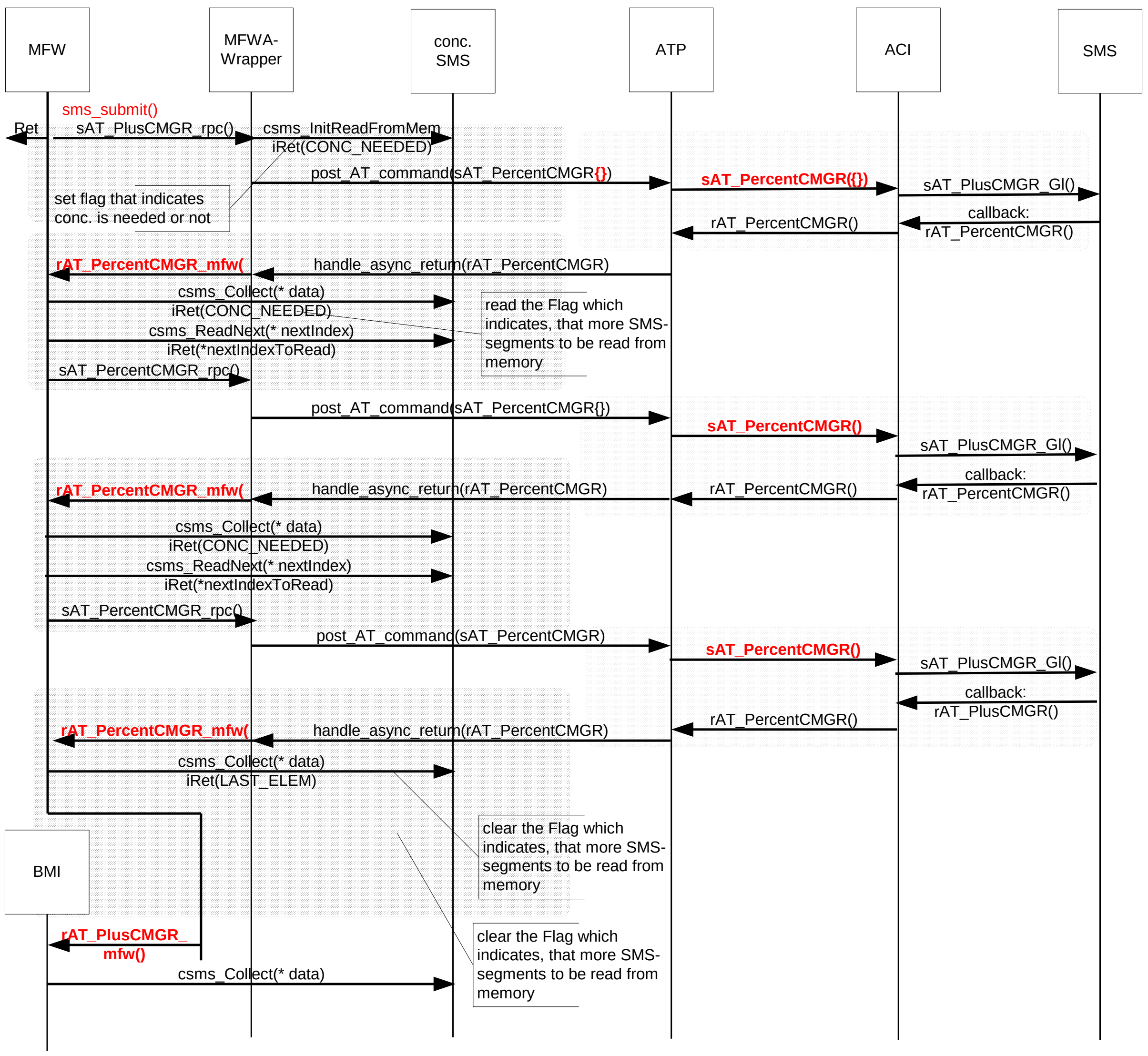
This function initializes shared parameter for CMGR, which reads SMS from memory.

```
EXTERN T_CONC_INIT_RETURN mfw_concsms_init_read_from_mem ( T_ACI_CMD_SRC  srcId,
                                                           UBYTE          index,
                                                           T_ACI_SMS_READ rdMode );
```

This function reads the next segment of the SMS from memory.

```
EXTERN BOOL mfw_concsms_read_next_from_mem (UBYTE* index);
```

3.5.3.5.1 MFW ←→ conc.SMS ←→ ACI while reading of an conc. SMS from memory


MFW
MFWA-Wrapper
conc. SMS
ATP
ACI
SMS
sms_submit()
Ret
sAT_PlusCMGR_rpc()
csms_InitReadFromMem
iRet(CONC_NEEDED)
post_AT_command(sAT_PercentCMGR{})
sAT_PercentCMGR({})
sAT_PlusCMGR_Gl()
set flag that indicates conc. is needed or not
callback: rAT_PercentCMGR()
rAT_PercentCMGR()
rAT_PercentCMGR_mfw(
handle_async_return(rAT_PercentCMGR)
csms_Collect(* data)
iRet(CONC_NEEDED)
csms_ReadNext(* nextIndex)
iRet(*nextIndexToRead)
read the Flag which indicates, that more SMS-segments to be read from memory
sAT_PercentCMGR_rpc()
post_AT_command(sAT_PercentCMGR{})
sAT_PercentCMGR()
sAT_PlusCMGR_Gl()
callback: rAT_PercentCMGR()
rAT_PercentCMGR_mfw(
handle_async_return(rAT_PercentCMGR)
rAT_PercentCMGR()
csms_Collect(* data)
iRet(CONC_NEEDED)
csms_ReadNext(* nextIndex)
iRet(*nextIndexToRead)
sAT_PercentCMGR_rpc()
post_AT_command(sAT_PercentCMGR)
sAT_PercentCMGR()
sAT_PlusCMGR_Gl()
callback: rAT_PlusCMGR()
rAT_PercentCMGR()
rAT_PercentCMGR_mfw(
handle_async_return(rAT_PercentCMGR)
csms_Collect(* data)
iRet(LAST_ELEM)
clear the Flag which indicates, that more SMS-segments to be read from memory
BMI
rAT_PlusCMGR_mfw()
csms_Collect(* data)
clear the Flag which indicates, that more SMS-segments to be read from memory

<u>following existing functions must be adapted:</u> (see above in MSC in red color)

<u>MFW:</u> - sAT_PlusCMGR_rpc

Zeichnung fehlt noch

- rAT_PlusCMGR_mfw

Zeichnung fehlt noch

<u>new ACI function:</u>

- new function ‘`sAT_PercentCMGR`’ similar to ‘`sAT_PlusCMGR`. Changes are strike out and marked in pink.

```
GLOBAL T_ACI_RETURN sAT_PercentCMGR ( T_ACI_CMD_SRC  srcId,
                                      UBYTE          index,
                                      T_ACI_SMS_READ rdMode )
{
  T_CONC_INIT_RETURN ret;

:
:

  ~~if (ret EQ CONC_NEEDED)~~
  ~~{~~
    ~~SET_CONC;~~
    ~~TRACE_EVENT("sAT_PlusCMGR: CONC_NEEDED");~~
    return sAT_PlusCMGR_Gl(srcId, index, rdMode, rAT_PlusCMGR);
  ~~}~~
  ~~else if (ret EQ CONC_NOT_NEEDED)~~
  ~~{~~
    ~~TRACE_EVENT("sAT_PlusCMGR: CONC_NOT_NEEDED");~~
    ~~return sAT_PlusCMGR_Gl(srcId, index, rdMode, rAT_PlusCMGR);~~
  ~~}~~
  ~~else~~
  ~~{~~
    ~~TRACE_EVENT("ERROR: sAT_PlusCMGR: CMS_ERR_InValMemIdx");~~
    ~~ACI_ERR_DESC( ACI_ERR_CLASS_Cms, CMS_ERR_InValMemIdx );~~
    ~~return ( AT_FAIL );~~
  ~~}~~
}
```

3.5.3.6 Delete SMS from memory

This function initializes shared parameter for CMGD, which deletes a SMS from memory.

```
EXTERN T_CONC_INIT_RETURN mfw_concsms_init_delete_from_mem ( T_ACI_CMD_SRC  srcId,
                                                              UBYTE          index );
```

This function deletes the next segment of the SMS in memory.

```
EXTERN BOOL mfw_concsms_delete_next_in_mem (UBYTE* index);
```

3.5.3.7 Functions for CMGC

This function initializes shared parameter for CMGC.

```
EXTERN T_CONC_INIT_RETURN mfw_concsms_init_command ( T_ACI_CMD_SRC   srcId,
                                                     SHORT           fo,
                                                     SHORT           ct,
                                                     SHORT           pid,
                                                     SHORT           mn,
                                                     CHAR*           da,
                                                     T_ACI_TOA*      toda,
                                                     T_ACI_CMD_DATA* data );
```

```
EXTERN BOOL mfw_concsms_command_next_in_mem (UBYTE mr, UBYTE* mn);
```

Not finish yet

3.5.4 changes on MFW-Wrapper

Cause of moving the actual implementation of the concatenation handler above to the ATP some sAT…-Functions must be changed.
At former time the user data header for each SMS-segment was generated below to ATP. Now the UDH will be create above. That means, in contrast to the former functions it has to be extend by an additional parameter which includes the user data header:

- **for store and submit**

```
sAT_PlusCMSS_rpc ( T_ACI_CMD_SRC srcId,
                   UBYTE index,
                   CHAR* da,
                   T_ACI_TOA* toda,
                   T_ACI_UDH_DATA *udh )
```

- **for SMS command**

```
T_ACI_RETURN sAT_PlusCMGC_rpc ( T_ACI_CMD_SRC srcId,
                                SHORT fo,
                                SHORT ct,
                                SHORT pid,
                                SHORT mn,
                                CHAR* da,
                                T_ACI_TOA* toda,
                                T_ACI_CMD_DATA* data,
                                T_ACI_UDH_DATA *udh )
```

- **for store a SMS**

```
T_ACI_RETURN sAT_PlusCMGW_rpc ( T_ACI_CMD_SRC srcId,
                                SHORT index,
                                CHAR* address,
                                T_ACI_TOA* toa,
                                T_ACI_SMS_STAT stat,
                                UBYTE msg_ref,
                                T_SM_DATA_EXT* src_data,
                                CHAR* sca,
                                T_ACI_TOA* tosca,
                                SHORT isReply,
                                T_ACI_UDH_DATA *udh )
```

- **for delete a SMS ??**

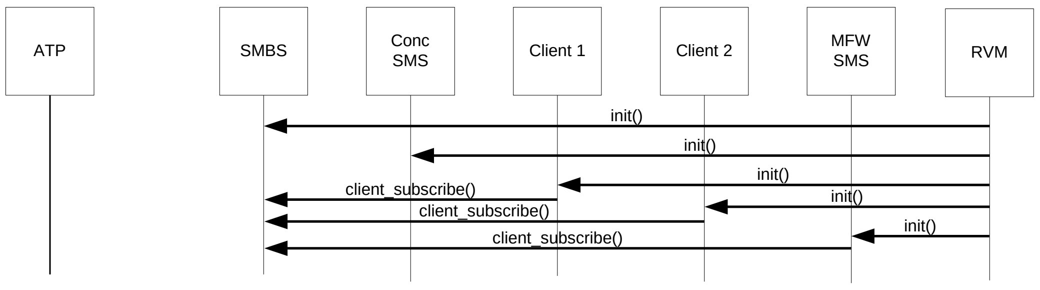
```
T_ACI_RETURN sAT_PlusCMGD_rpc ( T_ACI_CMD_SRC srcId,
                                UBYTE index,
                                T_ACI_UDH_DATA *udh )
```

- **for send a SMS**

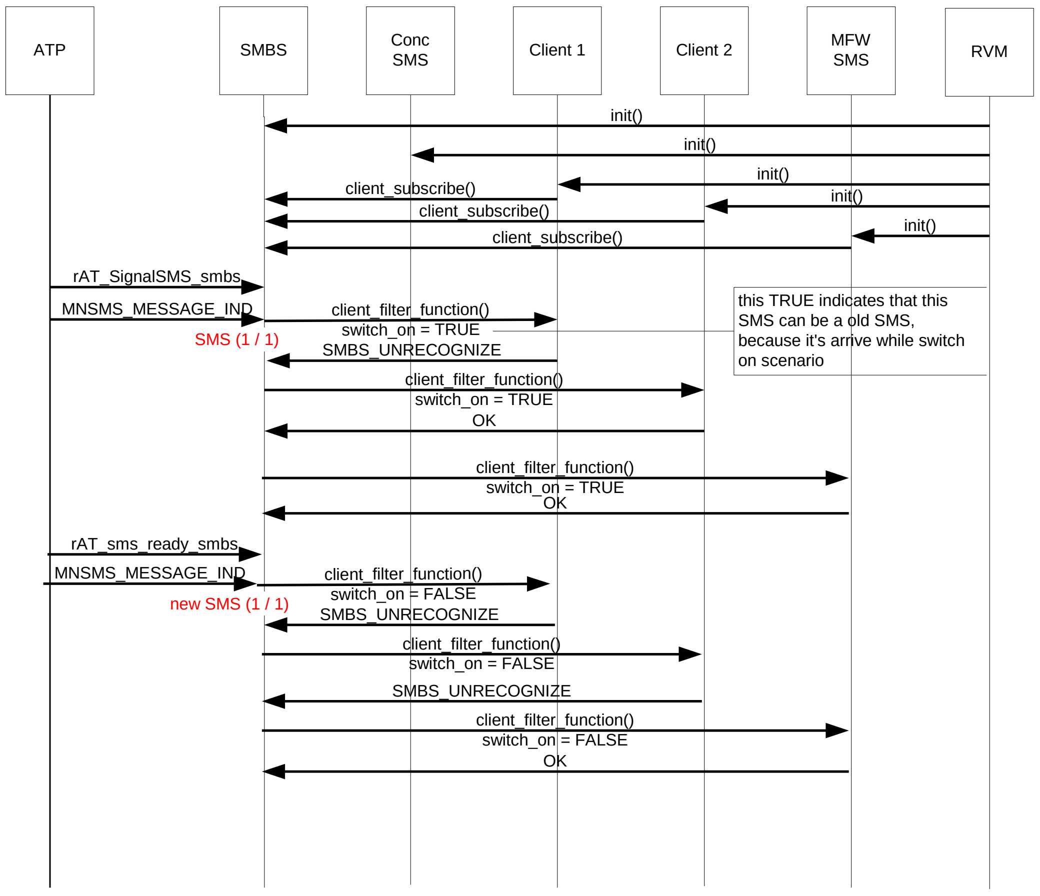
```
T_ACI_RETURN sAT_PlusCMGS_rpc ( T_ACI_CMD_SRC srcId,
                                CHAR* da,
                                T_ACI_TOA* toda,
                                T_SM_DATA_EXT* src_data,
                                CHAR* sca,
                                T_ACI_TOA* tosca,
                                SHORT isReply,
                                T_ACI_UDH_DATA *udh )
```

4 Scenarios

4.1 Subscription



4.2 Switch on scenario



While switch-on, each client must be subscribed to the SMBS. The subscription of the client will be start while executing the init()-function of each client.

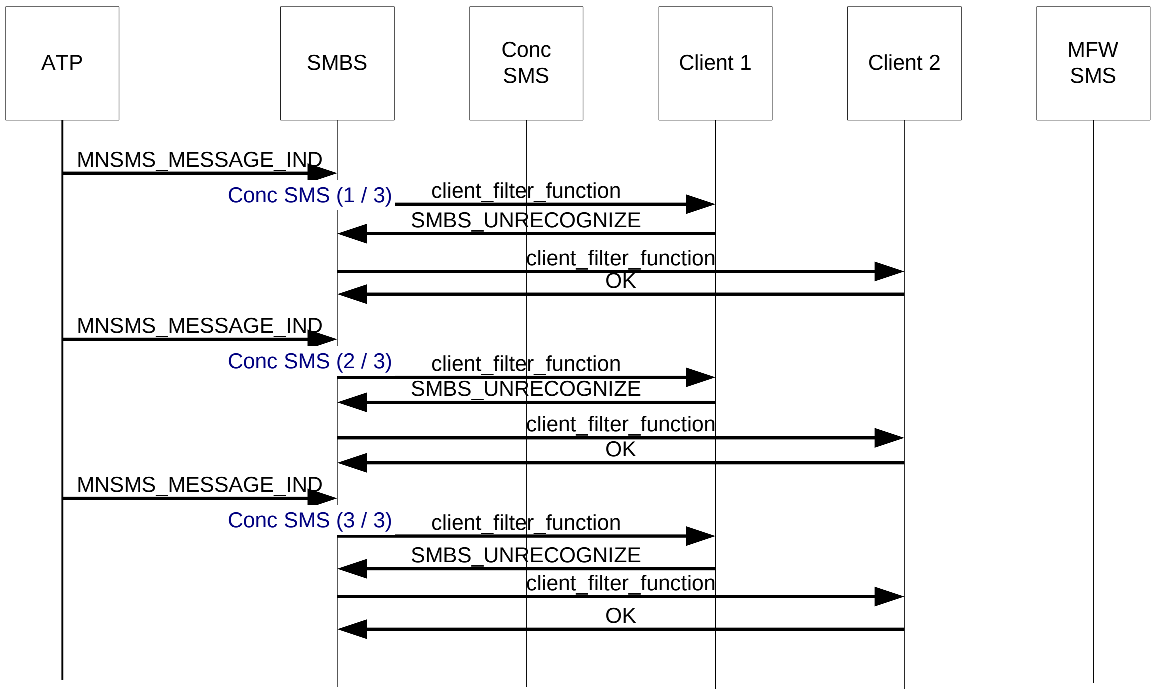
- Each client subscribes itself by using the client_subscribe function
- When the first segment of the concatenated SMS arrives, ATP will send this signal to the SMBS SWE. On reception of this event, SMBS will now send this event to the filter function of every single registered Client and waits for the response.

- Client 1 receives this message and now checks the type of SMS in its filter function. The filter function of the first client doesn't recognize the SMS type and send a reject back to the SMBS.
- Client 2 receives this message and now checks the type of SMS in its filter function. The filter function of the second client recognizes the SMS type as a single segment of a concatenated SMS and sends a OK back to the SMBS as the client is possible to de-concatenate this SMS and doesn't need the ConcSMS SWE.
  NOTE: The second client now has the responsibility for this segment. If for example this client organizes SMS in its own memory, it has to update the SMS SIM database as well in case this SMS will, for example, be deleted in the client. On the other hand the client can delete the SMS in the SIM database straight away if it stored successful in its own database. This is totally up to the implementation of the client.

- When SMBS receives an OK from a clients filter function it stops the distribution of the signal to further registered clients.
- When the second segment …(see above)
- When the last segment …(see above)

4.3 Receiving normal/concatenated SMS

4.3.1 Concatenated SMS – Client concatenates

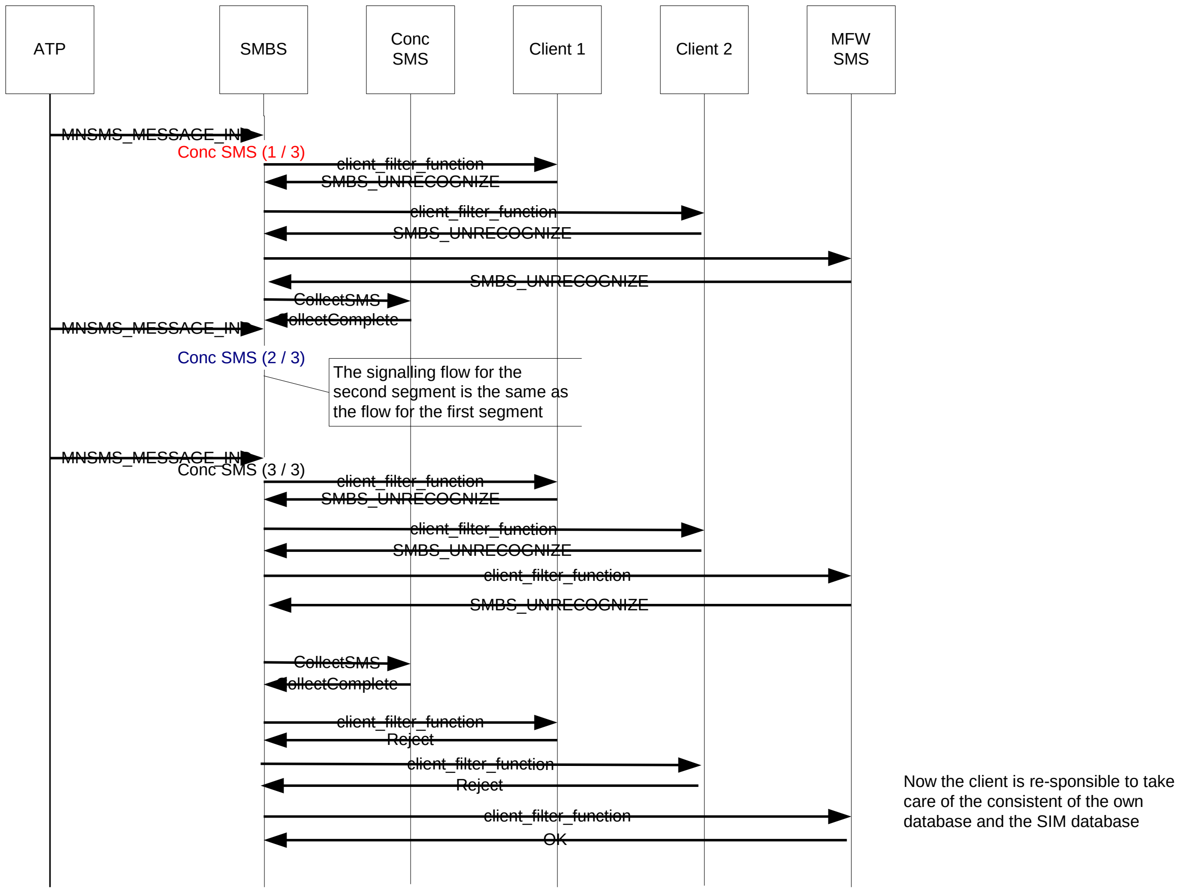


This example describes the sequence flow of a concatenated SMS existing out of 3 segments. Furthermore we consider clients having the possibility to concatenate/deconcatenate messages, and are **NOT** dependent on the Conc SMS SWE to do this.

- When the first segment of the concatenated SMS arrives, ATP will send this signal to the SMBS SWE. On reception of this event, SMBS will now send this event to the filter function of every single registered Client and waits for the response.
- Client 1 receives this message and now checks the type of SMS in its filter function. The filter function of the first client doesn't recognize the SMS type and send a reject back to the SMBS.
- Client 2 receives this message and now checks the type of SMS in its filter function. The filter function of the second client recognizes the SMS type as a single segment of a concatenated SMS and sends a OK back to the SMBS as the client is possible to de-concatenate this SMS and doesn't need the ConcSMS SWE.
  NOTE: The second client now has the responsibility for this segment. If for example this client organizes SMS in its own memory, it has to update the SMS SIM database as well in case this SMS will, for example, be deleted in the client. On the other hand the client can delete the SMS in the SIM database straight away if it stored successful in its own database. This is totally up to the implementation of the client.

- When SMBS receives an OK from a clients filter function it stops the distribution of the signal to further registered clients.
- When the second segment …(see above)
- When the last segment …(see above)

4.3.2 Concetanated SMS for MFW-SMS (cannot concatenate)

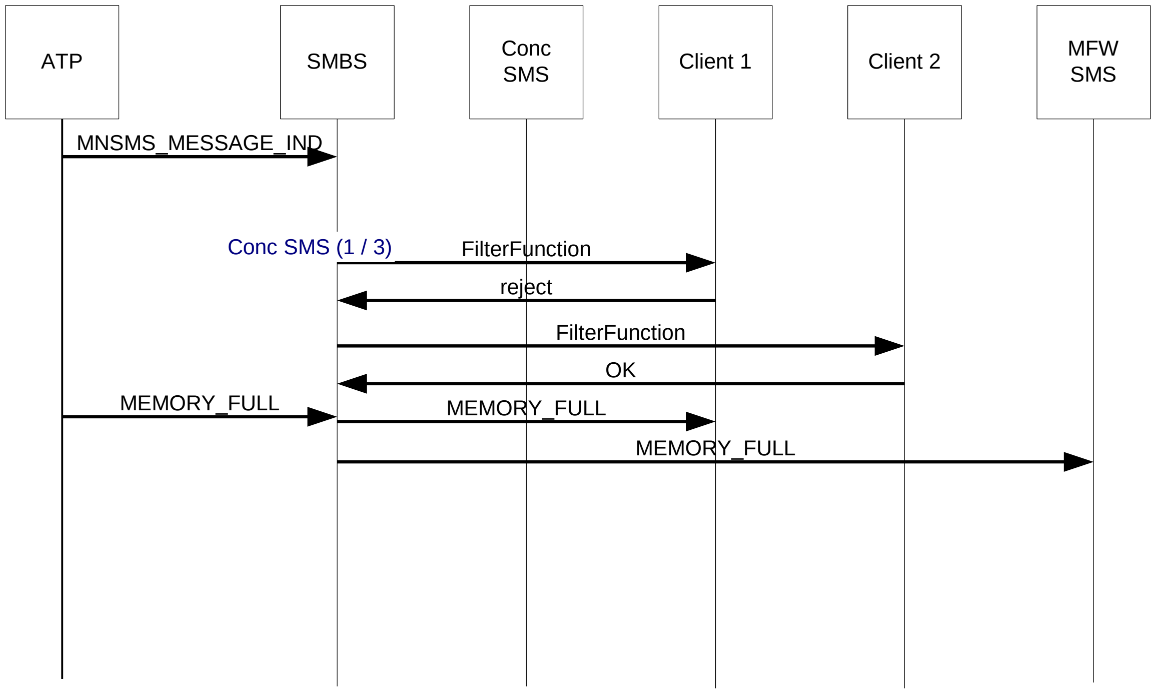


This example describes the sequence flow of a concatenated SMS existing out of 3 segments. Furthermore we consider very basic clients, having no possibility to concatenate/deconcatenate messages, and depend on the Conc SMS SWE to do this.

- When the first segment of the concatenated SMS arrives, ATP will send this signal to the SMBS SWE. On reception of this event, SMBS will now send this event to the filter function of every single registered Client and waits for the response.
- Client 1 receives this message and now checks the type of SMS in its filter function. The filter function of the first client doesn't recognize the SMS type and send a reject back to the SMBS.
- Client 2 receives this message …. Same as previous.
- If no registered client recognizes this signal by its filter function, the signal will be sent to MFW SMS by default.
- MFW SMS now recognizes the SMS as a concatenated SMS, but is only interested in a complete SMS and not in single segments and informs the SMBS to collect this message. If there is the customer requirement to inform the user of every single segment, than this can be handled as well.
- The SMBS now sends this segment to the ConcSMS module, which handles the concatenation/deconcatenation.
- SMBS is now ready to handle the second segment of the conc SMS, which now shows the same behavior than previously described first segment.

When the last (Third) segment arrives, the distribution of this segment to the clients is the same as before. When the ConcSMS receives this last segment it informs SMBS about, that there is one deconcatenated SMS waiting to be handled. This happens by returning with CollectComplete. SMBS now distributes this complete SMS to all clients, but only the MFW-SMS will handle this SMS and will display the content of this message on the display.

4.3.3 SIM – full scenario



This example describes the sequence flow of a concatenated SMS existing out of several segments. Furthermore we consider clients having the possibility to concatenate/deconcatenate messages, and are **NOT** dependent on the Conc SMS SWE to do this. In this example we will distribute the information that storage has reached its maximum to every client, which has registered to this information.
In this example we say that the first client is registered to this information.

- When the first segment of the concatenated SMS arrives, ATP will send this signal to the SMBS SWE. On reception of this event, SMBS will now send this event to the filter function of every single registered Client and waits for the response.
- Client 1 receives this message and now checks the type of SMS in its filter function. The filter function of the first client doesn't recognize the SMS type and send a reject back to the SMBS.
- Client 2 receives this message and now checks the type of SMS in its filter function. The filter function of the second client recognizes the SMS type as a single segment of a concatenated SMS and sends a OK back to the SMBS as the client is possible to deconcatenate this SMS and doesn't need the ConcSMS SWE.
- On reception of the OK, SMBS will send the information that the maximum storage has been reached to the first client and MFW, which is the only client who has registered to this information. A use of this information is, that the client can now evaluate this information and inform the user about the fact that the storage is full and no further SMS can arrive to the mobile phone as there is no way to store this SMS. In this way the user is aware of this and can act accordingly and can delete messages.

5 Test

Because of missing message client's it's currently not possible to test the special functions of SMBS.

It might possible to test the old behavior, such as:

- sending and receiving of normal Text-SMS
- sending and receiving of normal concatenated Text-SMS
- SIM-full behavior
- correct behavior while switch-on (distribute SMS, correct subscription)

by using of another Handheld in opposite to an D-Sample.

With this test it can be perform:

- test of redirect mechanism new implemented the MFW-Message Handler
- test of subscription-function in SMBS

Currently, it's not possible to test:

- the filtering behavior and thus the interaction between Message-client and SMBS
- the interaction between SMBS and ported concatenation handler

Appendices

A. Acronyms

ATP	Agnostic Transport Protocoll
CFM	Confirm
CM	Call Managment
G23M	GSM Protocol Stack
MMI	Man Machine Interface
MSC	Message Sequence Chart
MT	Mobile Terminated
RGUI	Riviera Graphical User Interface
SMS	Short Message Service
SWE	Software Entity
UI	User Interface

B. Glossary

International Mobile Telecommunication 2000 (IMT-2000/ITU-2000)	Formerly referred to as FPLMTS (Future Public Land-Mobile Telephone System), this is the ITU's specification/family of standards for 3G. This initiative provides a global infrastructure through both satellite and terrestrial systems, for fixed and mobile phone users. The family of standards is a framework comprising a mix/blend of systems providing global roaming. <URL: http://www.imt-2000.org/>