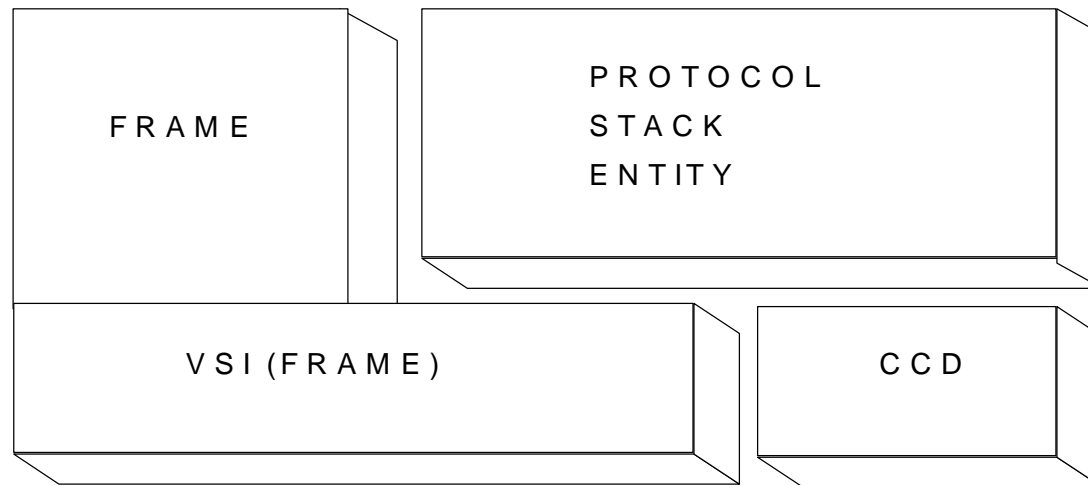


Condat

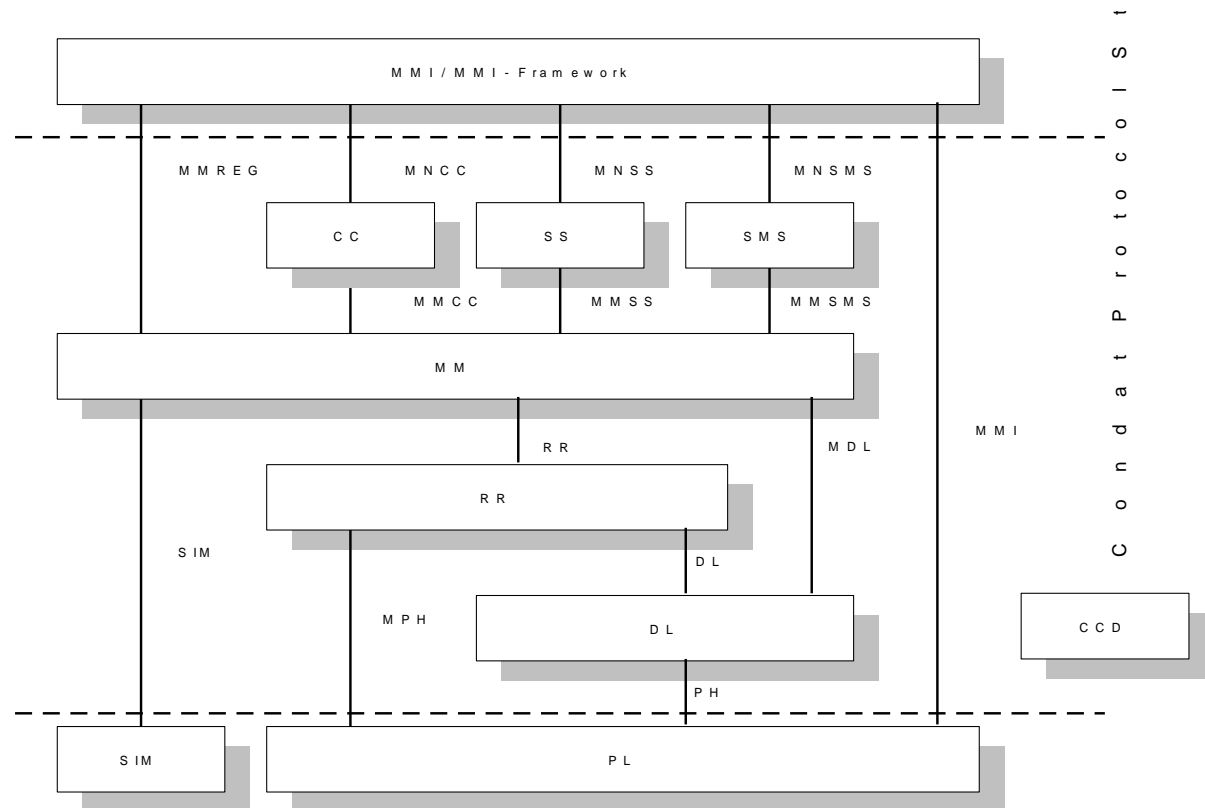
(G23) Coder Decoder

CCD

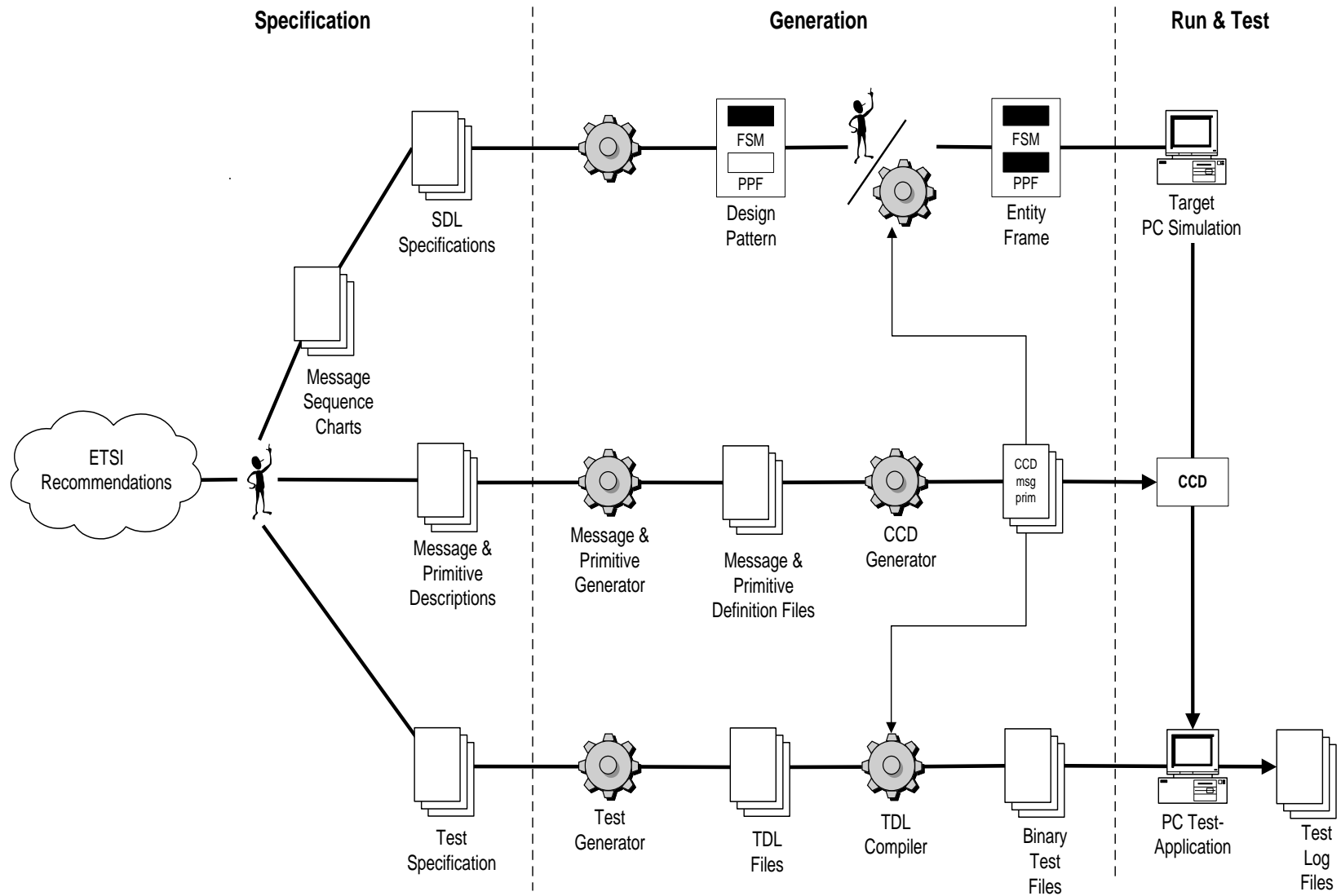
Structure of Protocol Stack Entity



Mobile-Station Protocol Architecture



Protocol Stack Development Method



CCD - GSM/GPRS/UMTS Coder Decoder System (I)



Subsystem

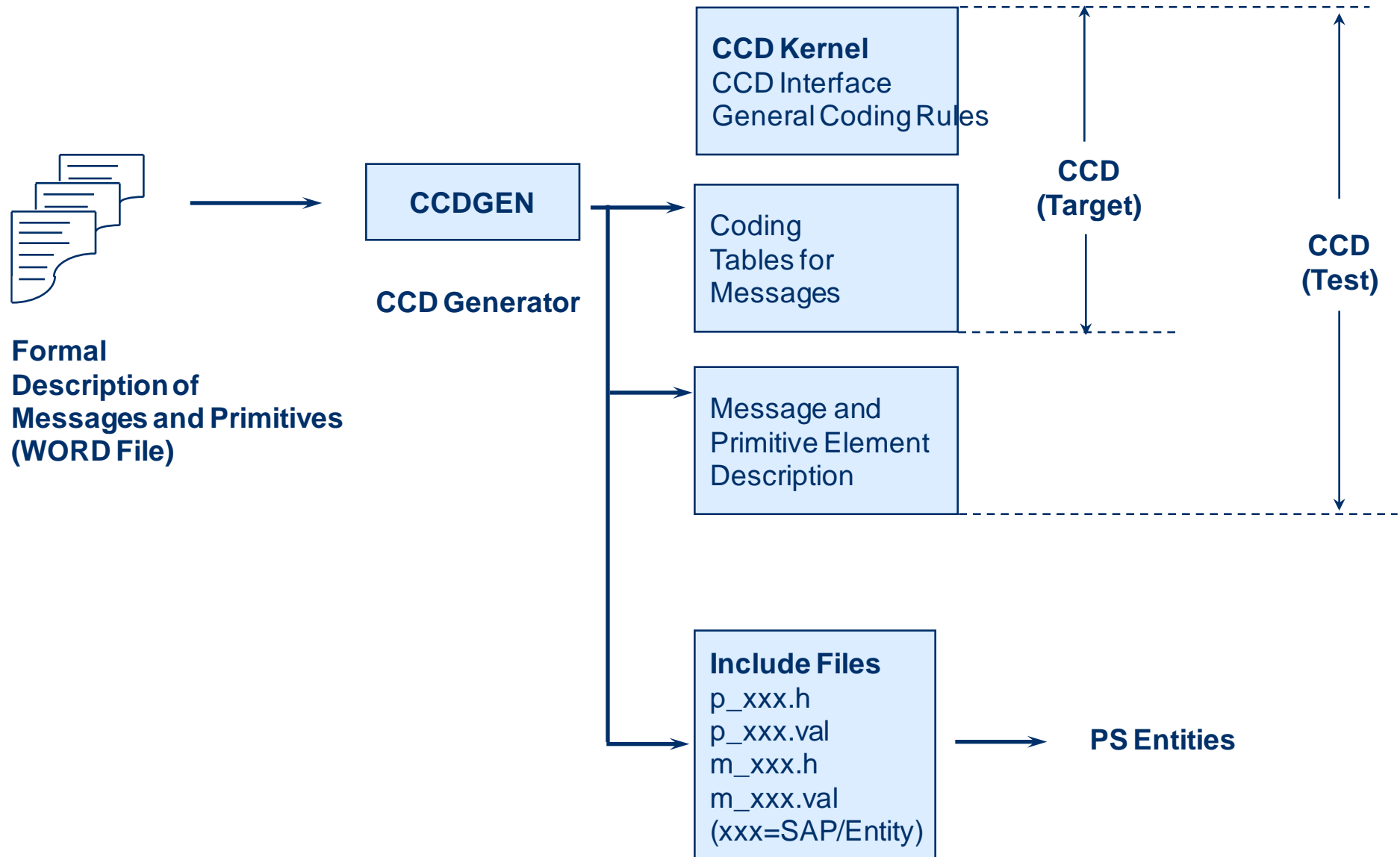
S:

- **CCDGEN**
- **CCDEDIT**
- **CCD**
- **CCD data base**
- **TAP (Test Application Process)**

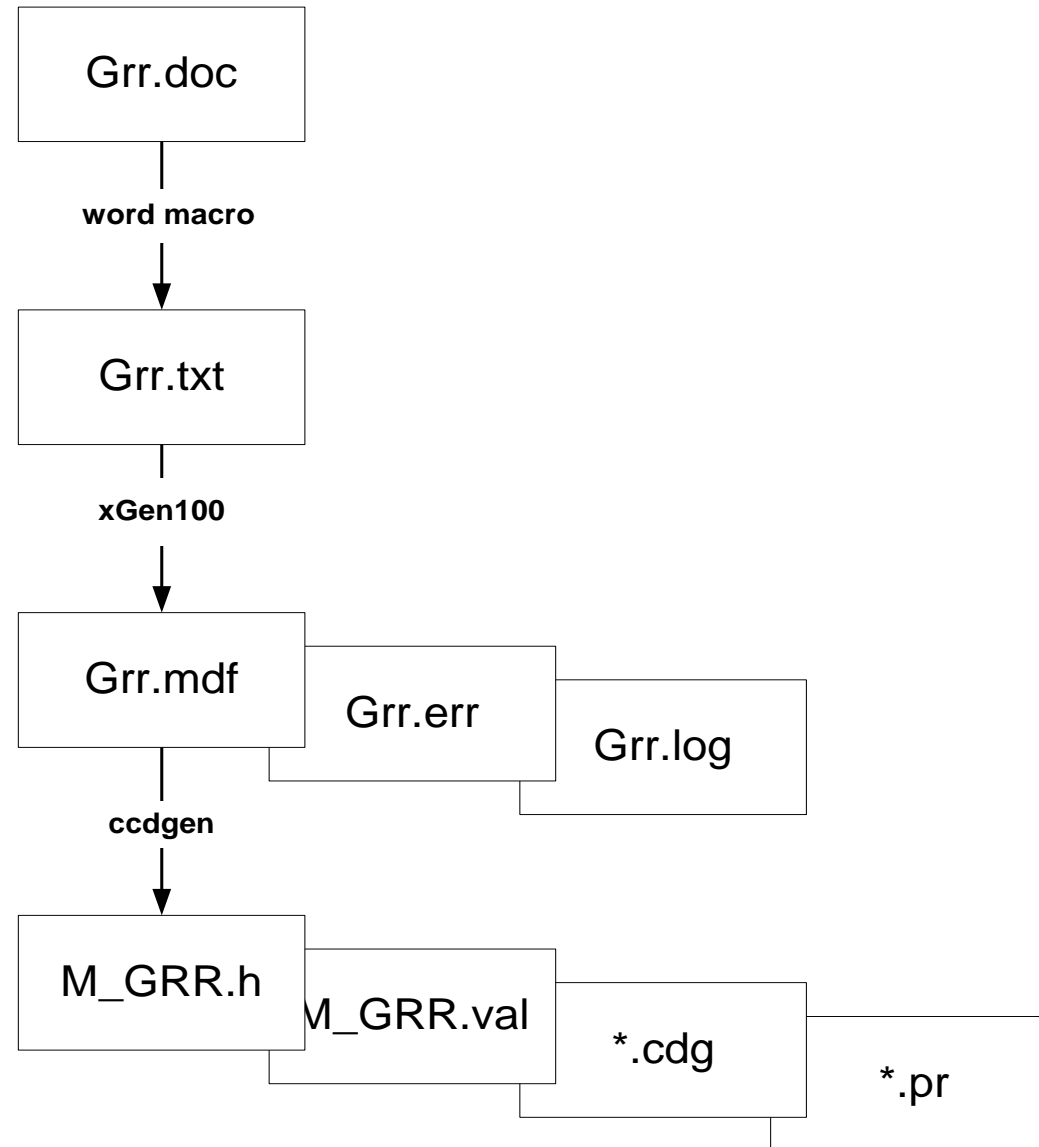
CCD data base:

- Data tables describe the structure of Air Interface Messages and Primitives, e.g. IE types, bit field sizes, offset values
- Test system : debug information and primitives
- Statically linked as object or library file to the application.
- Dynamically linked on win32 systems (ccd.dll)
- Tables reside in ...\CDGINC

CCD Data and Type Generation (I)



CCD Data and Type Generation (II)



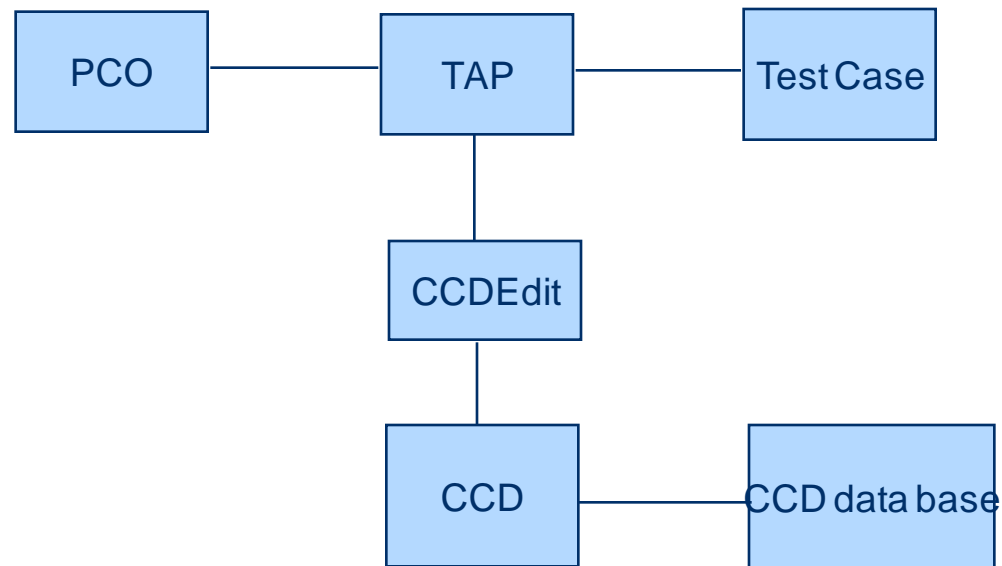
CCDGEN:

Creation of C include files and SDL files from Primitive Description Files (PDF) and Message Description Files (MDF).

```
...  
  
$(CDGINC) \mconst.cdg: $(MDFFILES)  
    $(TESTDRIVE)  
    cd $(MSGDIR)  
    -md $(CDGINC)  
    $(CCDGEN) -l -t -h -m64 -a2 -o$(CDGINC) $(**B)  
  
...  
  
$(CDGINC) \pconst.cdg: $(PDFFILES)  
    $(TESTDRIVE)  
    cd $(PRIMDIR)  
    -md $(CDGINC)  
    $(CCDGEN) -l -t -h -m64 -p -a0 -o$(CDGINC) $(**B)
```

TAP:

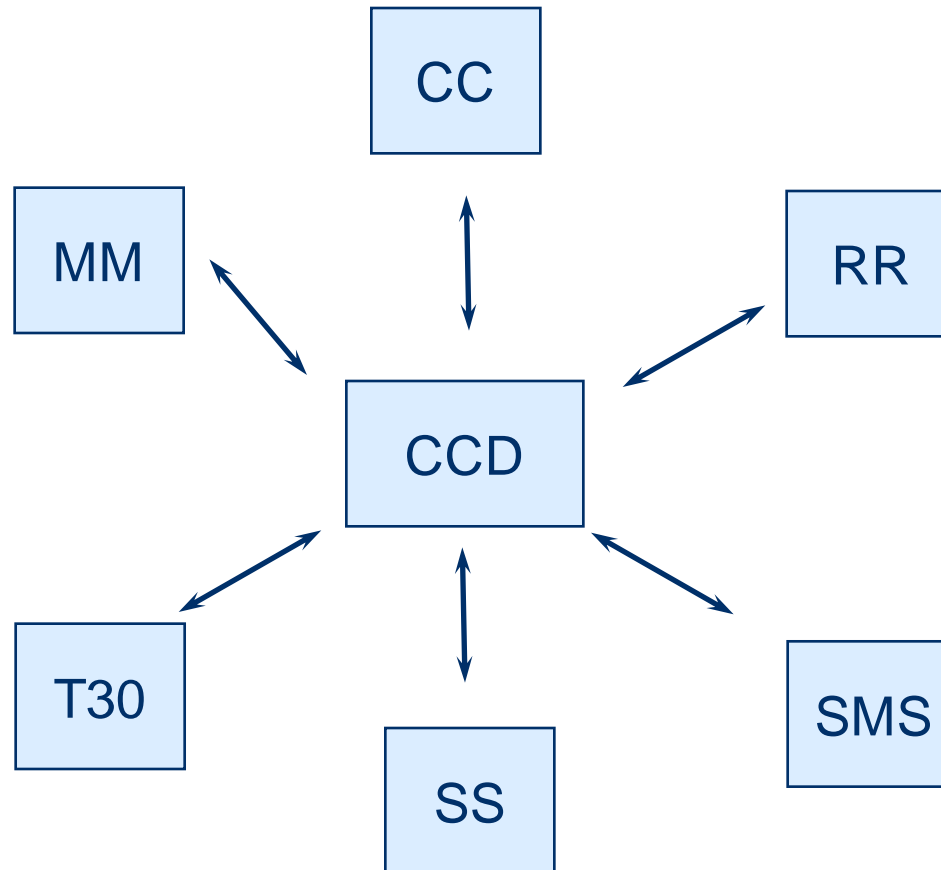
- Test Application Process
- Uses the CCD and CCDEDIT functionality



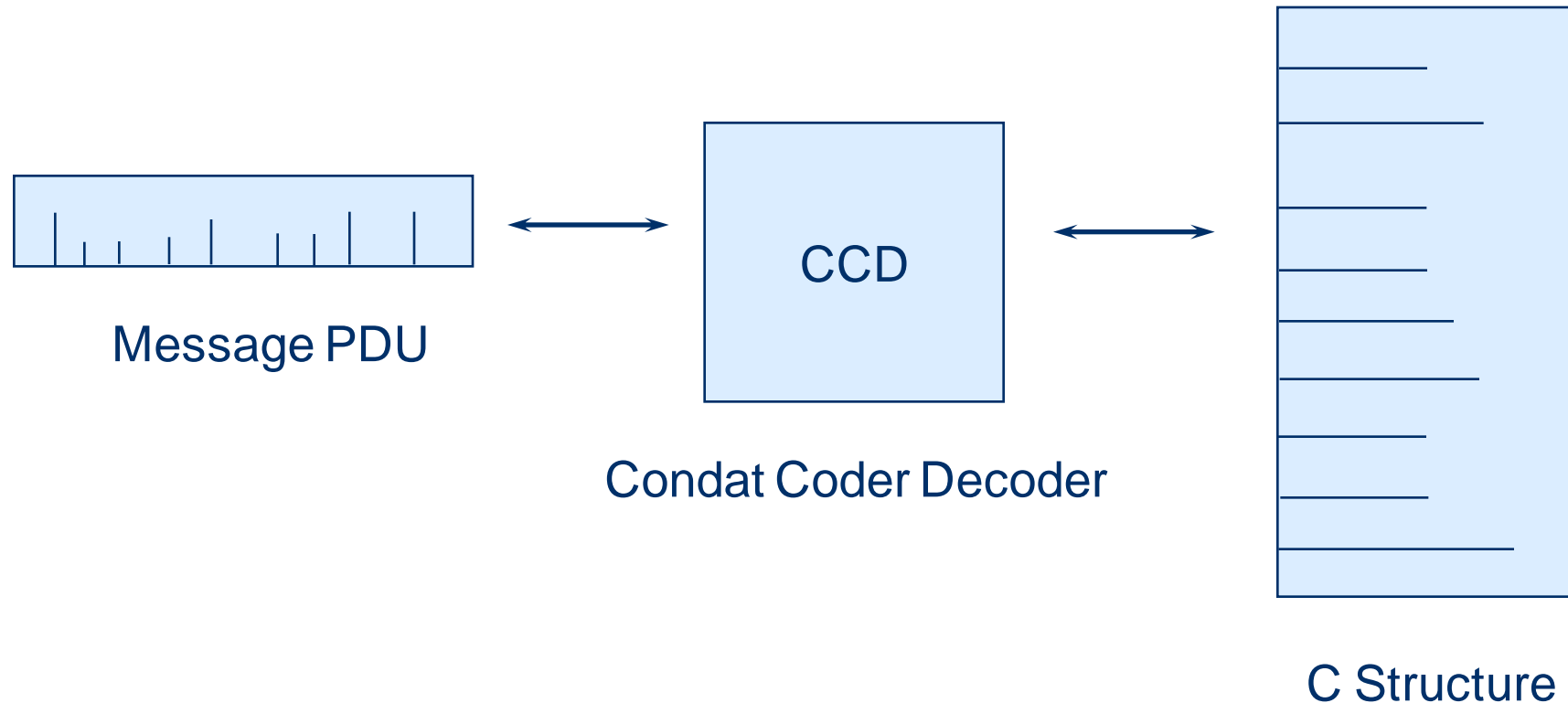
- CCDEDIT:**
- C function library providing Iterator functionality for the CCD data base.
 - Iterate thru the IEs of an Air Interface Message
 - Iterate thru the components of a Primitive
 - Especially used for test case definition and execution

```
RR_ESTABLISH_REQ      ESTCS_MOB_ORIG
                        estcs
                        sdu
                        {
                        component      MM
                        direction      UPLINK
                        pd              U_IMSI_DETACH_IND
                        ti              TI_0
                        mob_class_1    MOB_CLASS_1
                        mob_ident       MOB_IDENT_IMSI
                        }
```

CCD Usage Relation



GSM/GPRS/UMTS Coder Decoder Component (I)



- CCD:**
- C functions provided to convert Air Interface Messages into C structures and vice versa
 - Avoid bit operations in client code
 - Target independent

```
GLOBAL void for_setup (T_CC_DATA * cc_data, T_U_SETUP * setup)
{
    PALLOC_MSG (data, MMCC_DATA_REQ, U_SETUP);

    TRACE_FUNCTION ("for_setup()");

    ccd_codeMsg (CCDENT_CC,
                UPLINK,
                (T_MSGBUF *) &data->sdu,
                (UBYTE *) setup,
                NOT_PRESENT_8BIT);

    for_pd (data, cc_data);
}
```

CCD error handling:

```
#include ccdapi.h
...
USHORT parlist[6];
UBYTE first_err;

memset (parlist,0, sizeof (parlist));
first_err = ccd_getFirstError (CCDENT_CC, parlist);
/*
 * Error Handling
 */
switch (first_err)
{
    case ERR_COMPREH_REQUIRED: /* comprehension required */
        for_set_mandatory_error (cc_data, (UBYTE)parlist[0]);
        break;
    case ERR_MAND_ELEM_MISS: /* Mandatory elements missing */
        /*
         * Error handling is carried
         * out by checking the content.
         */
        break;
}
```

CCD - Decoding Example (I)

Example message: SAT Command in sat.doc

Definition:

long name	short name	ID	direction
SAT Command	stk_cmd	0b00000000	downlink

Elements:

ID	long name	short name	ref	ref [1]	pres	type	len
	Message Type	msg_type	6.12		M	V3	1
0xD0	Proactive SIM Command	pas_cmd	5.1	13.2	O	TLV5	2-258

CCD - Decoding Example (II)

Information Element pas_cmd in sat.doc

Definition:

long name	short name	Len
Proactive SIM Command	pas_cmd	2-255

Elements:

ID	long name	short name	ref	ref [1]	type	len
0x81	Command details	cmd_details	5.2	12.6	TLV5	5
0x82	Device identities	dev_ids	5.3	12.7	TLV5	4
	Command parameters	cmd_prms	6.10	12	V5	0-246

Message Describing in Intermediate Format MDF:

```
VAR cmd_prms      "Parameters"
  1952

COMP pas_cmd      "Proactive SIM Command" 0xD0
{
  GSM5_TLV        cmd_details      ; Command details
  GSM5_TLV        dev_ids          ; Device identities
  GSM5_V          cmd_prms         ; Command parameters
}

MSG stk_cmd       downlink 0b01000000 ; SAT Command
{
  GSM3_V          msg_type         ; Message Type
  GSM5_TLV        pas_cmd          ; Proactive SIM Command
}
```

C-Structure types to contain decoded stk_cmd:

```
typedef struct
{
    UBYTE      msg_type;    /*< 0: 1> Message type */
    UBYTE      v_pas_cmd;   /*< 1: 1> valid-flag      */
    T_pas_cmd  pas_cmd;     /*< 2:257> Proactive SIM Command */
} T_STK_CMD;
```

And for Target:

```
typedef struct
{
    UBYTE      msg_type; /*< 0: 1> Message type */
    UBYTE      _align0;  /*< 1: 1> alignment      */
    UBYTE      _align1;  /*< 2: 1> alignment      */
    UBYTE      v_pas_cmd; /*< 3: 1> valid-flag      */
    T_pas_cmd  pas_cmd;  /*< 4:272> Proactive SIM Command */
} T_STK_CMD;
```

Test function to decode an stk_cmd message:

```
char msgstr[] = "00 D0 3d
81 03 01 10 01
82 02 81 83
85 1f 43 6f 6e 66 69 72 6d 65 7a 20 6c 27 61 70 70 65 6c 0b 91 00 00
00 00 00 21 69 16 69 00 01 01 86 0b 90 00 00 00 00 00 21 69 16 69
71 88 04 80 50 00 00";
T_MSGBUF msgbuf; UBYTE target[4096];
while (strlen (msgstr)) {
    sscanf (msgstr, "%x", &byte);
    msgbuf.buf[i++] = (UBYTE) byte;
    msgstr +=3; }
msgbuf.l_buf = i*8;
msgbuf.o_buf = 0;
ccd_decodeMsg(..., &msg_buf, target, 0xff);
```

Debugger Output for C Structure after Decoding:

```
OUTPUT=(T_STK_CMD*) target;
|__msg_type          0x00
|__v_pas_cmd         0x01
|__pas_cmd
|    |__v_cmd_details 0x01
|    |__cmd_details
|    |    |__cmd_nr    0x01
|    |    |__cmd_typ   0x10
|    |    |__cmd_qlf   0x01
|    |__v_dev_ids     0x01
|    |__dev_ids
|    |    |__src_dev   0x81
|    |    |__des_dev   0x83
|    |__v_cmd_prms    0x01
|    |__cmd_prms
|    |    |__l_cmd_prms 0x01a0
|    |    |__o_cmd_prms 0x0000
|    |    |__b_cmd_prms 0x0012c686"Confirmez l'appel"
```

Set environment: `\gpf\initvars.bat`

Generate CCD data:

- For PC call `\g23m\condat\int\bin\makcdg.bat` and build the PS using MSDev
- For target call „perl `\g23m\g23.pl`“ which uses `\gpf\ccd\ccddata.mk`
- Both procedures use `\gpf\bin\xgen100.exe` and `\gpf\bin\ccdgen.exe`
- Outputs: `...\cdginc\`, `...\cdginc_fd_gp\`, `ccddata.lib` and `ccddata_dll.dll`

Build Test Application:

- go to \gpf\tap\ and call gnumake
- or call gnumake TAP_VERSION=OLD
- output will be e.g. tap2_gprs.exe
- under \gpf\bin\ or \gpf\tap\obj\

Build Test-DLL:

- call \gpf\bin\mktdc.bat
- it reads \g23m\condat\ms\doc\test\x.doc
- output is \g23m\condat\ms\tds\x\xy.tds
- and \g23m\condat\ms\test\test_x*.dll

Different variations : e.g. ccd_npc_tr_db.lib

- _tr, _db, _npc, _na7, _ar, _po, _vx
- Trace, Debug, Nucleus emulation on Win32, Nucleus on ARM7, GTI on ARM, pSOS, vxworks

Library is an object of ClearCase control:

- See also gpf_memo_cclabels.doc.
- CCD objects independent from ccd data tables in CCD_2.2.0 and onwards
- Read about versions: readme_ccd.txt
- Documentation: ccd_api.doc, faq_ccd.html and ccd_userguide.doc

Different cdginc files:

-> High potential for functioning problems

- Alignment bytes depend on processor type (check ccdgen call for -a0, -a1 or -a2)
- No CCD_SYMBOLS for target (no name column in mcomp.cdg)
- PS, TAP, PCO must use the same ccd data tables.

CCD - Miscellaneous (Instructions in ctrl column)

Instruction	meaning of instruction	example
[0..CONSTANT]	array of bytes (also USHORT in *.pdf)	[0..MAX_RFL_NUM_LIST]
[varname+number..CONSTANT]	array of bytes (also USHORT in *.pdf)	[rfl_cont_len+3..19]
[.CONSTANT]	array , dot marks a bitarray	(SETPOS){ident_type = ID_TYPE_TMSI} [.32]
BCDODD[numbers]	BCD numbers starting with digit1	
BCDEVEN[numbers]	BCD numbers starting with digit2	BCDEVEN[2] or BCDEVEN[0..20]
{ ... }	conditional	{flag=1 AND flag2=1 OR flag=0}
(...)	command sequence	(GETPOS,;,4,+,;,1,+,SETPOS)
GETPOS	get the bitstream pointer	(GETPOS,;,4,+,;,1,+,SETPOS)
SETPOS	set bit stream pointer	(SETPOS) {type_of_identity # ID_TYPE_NO_IDENTITY AND type_of_identity # ID_TYPE_TMSI} BCDODD [0..16]
KEEP,regNr	keep value of a variable in ccd register	(KEEP,1) see GRR.doc chapter 5.65
TAKE,regNr	Take the value of ccd register	[.(TAKE,1)+1..8] see GRR.doc chapter 5.136
MAX,regNr	Compare and keep the maximum in ccd register from a variable and ccd register	(MAX,2) see GRR.doc chapter 5.73 and 5.74
:	duplicate the element	(GETPOS,;,4,+,;,1,+,SETPOS)
^	swap the two elements	see CC.doc chapter 5.4 bearer capability
+ * -	first middle last octett	see CC.doc chapter 5.4 bearer capability
AND OR XOR	logical operations: AND, OR and XOR	{flag=1 AND flag2=1 OR flag=0}
= # < >	comparisions	(KEEP,1) {n_r_cells # 0}
(22) or (0)	Maximum length of spare padding bits	S_PADDING .00101011 (22) means if the message consists of less than 22 bytes then fill up with the bit pattern S_PADDING .00101011 (0) means if the message doesn't end on octett boundary then fill up to octett boundary with bit pattern eg.: xxxx1011

- **Dose CCD do memset() to 0?** Yes, using l_buf. So be careful with filling l_buf.

- **What are SHARED_CCD_BUF, CCD_START, CCD_END and _decodedMsg?** They help to use CCD internal allocated but shared memory. **Usage:**
CCD_START;

```
{
    UBYTE ccdRet;
    MCAST( com, COMPONENT );
    memset( com, 0, sizeof( T_COMPONENT ) );
    ccdRet = ccd_decodeMsg (CCDENT_FAC,
                          UPLINK,
                          (T_MSGBUF *) &mnss_facility_ind -> fac_inf,
                          (UBYTE *) _decodedMsg,
                          COMPONENT);
}
CCD_END;
```