- ETS rules of development -

1. Origin and format of the Abstract Test Suites

The test suites implemented by ACACIA come from a standards institute like ETSI, ITU-T, and OSTC. Those organisations usually produce test specifications called Abstract Test Suites. They are described in TTCN and published in a machine processable (.mp) format (ETSI) that will become the basis of the future implementation produced by ACACIA. Sometimes, like for Q781 ITU-T test descriptions, only MSC charts are published. In this case the ACACIA team has to describe the tests in TTCN according to the MSC charts. The original TTCN description of the tests can be obtained from the original standards institute. In any case an HTML representation of the implemented TTCN ATS is provided in the Executable Test Suite package.

2. Implementation PROCESS

As we have seen before, the basis of the implementation is a machine processable file describing tests in TTCN and usually published by a Standards Organisation (SO). ACACIA's work consists of an implementation of this TTCN test description following three phases :

- Pre-processing phase
- Customisation phase
- Execution phase

2.1 Pre-processing phase

The file is translated automatically with the ACACIA TTCN compiler to produce C code. The ACACIA TTCN compiler supports all .mp files using the TTCN syntax described in : "ISO/IEC 9646-3 (IS) The Tree and Tabular Combined-Notation" (31 October 1996 Delivery 8.4)" document. If the compiler does not support some descriptions or if the published TTCN contains TTCN errors (indentation level, verdict on attachment tree…), the .mp file can be corrected at this stage. All the modifications made are referred to in a word or PDF document called xxx_xxref.doc (.pdf). The updated tables are included in this document and can be compared to the original description . mp file published by the SO., In the best case, they are just checked from a syntactical point of view. Consequently, depending on the power of the analyser used, syntactical errors can remain present.

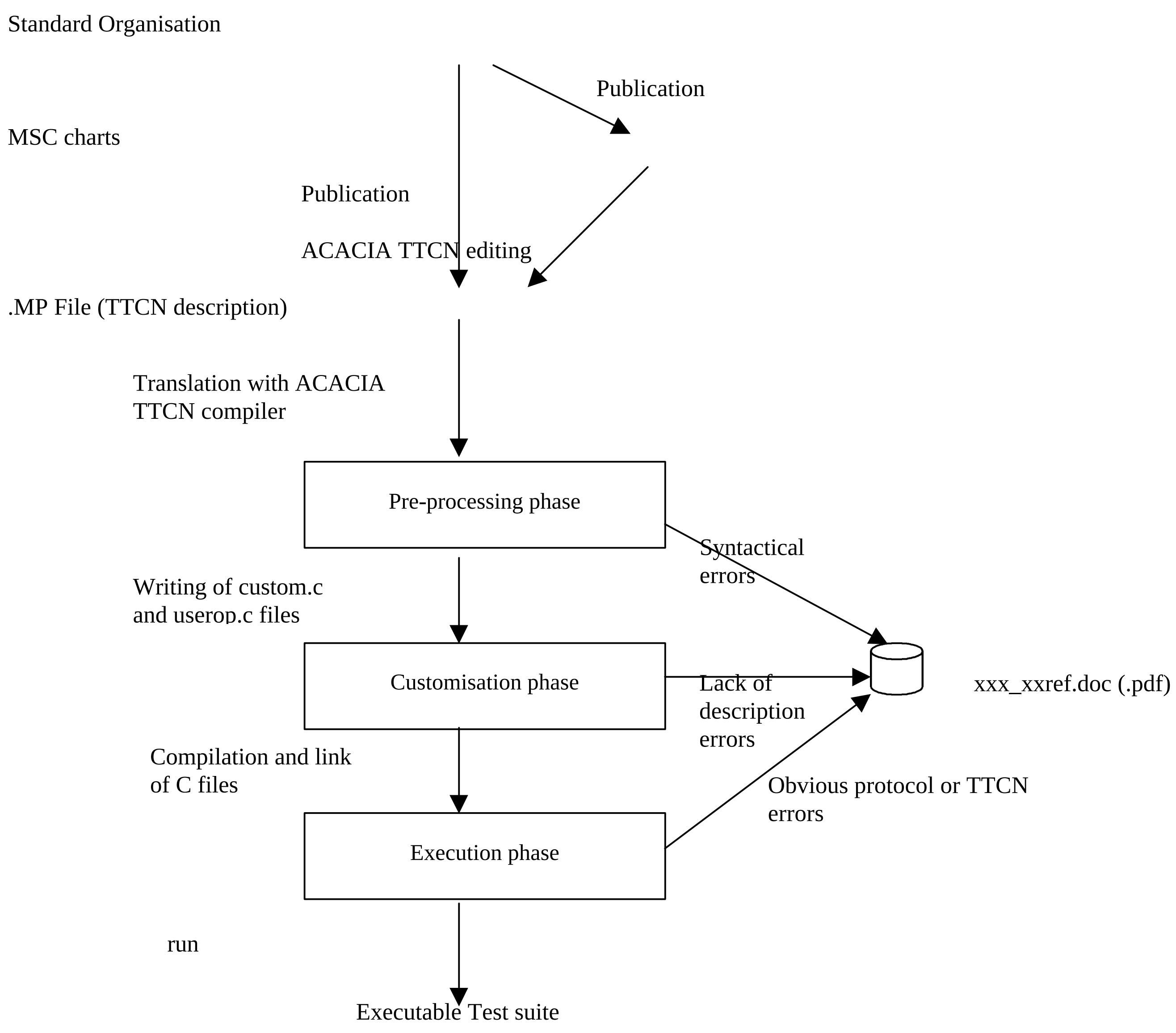
2.2 Customisation phase

The aim of this phase is to make the liaison between the C code, produced after translation, and the Clarinet books. For example, to make the association between the Abstract Service Primitive described in the TTCN and API functions of Clarinet book that will be used. The customisation phase issue is to get two .c code files : custom.c and the userop.c. The last one contains the C implementation of the operations, described in a natural language in TTCN, with the Clarinet API. The first one contains the liaison with the Clarinet book and some extra processing depending on the quality level of the TTCN ATS. It can be behaviour requested in the test purposes but not described in the TTCN, like the encoding of a PDU or some IEs of the PDU. Or the behaviour, of a lower level, that could disturb the execution of the test suite, but is not managed in it, but is absorbed or managed in the custom.c file. At this stage no modification is made to the .mp file except if some extra information has to be added to implement extra processing due to the lack of description in the test suite. In any case those modifications are recorded and justified in the xxx_xxref.doc (.pdf) document.

2.3 Execution phase

All the C code previously produced is compiled and linked in order to get an executable file of the test suite. Then, the test suite is run against an Implementation Under Test (IUT). This IUT can be an emulator from ACACIA or a simulated IUT written with clarinet APIs. This phase helps in detecting errors like obvious lower level protocol errors in the TTCN description, or dissension between test purposes and the TTCN description. On this last point, it is important to stress that the main base of a test is its test purpose described in a separate document in ETSI or at the top of the ITU-T MSC. The test purposes are included again in the TTCN tables but the based references are those described in the previous sentence. This phase also allows for discovery of TTCN execution error like received constraint used in sent event, non initialised variables….

2.4 Synthesis Diagram


Standard Organisation
Publication
MSC charts
Publication
ACACIA TTCN editing
.MP File (TTCN description)
Translation with ACACIA TTCN compiler
Pre-processing phase
Syntactical errors
Writing of custom.c and userop.c files
Customisation phase
Lack of description errors
xxx_xxref.doc (.pdf)
Compilation and link of C files
Obvious protocol or TTCN errors
Execution phase
run
Executable Test suite

3. Conformance to Published TTCN descriptions

3.1 Check with Documentation

As we have seen before, all modifications made to the .mp file published by the SO are recorded and justified in the xxx_xxref.doc (.pdf) document. To stay conformed to the published test description, ACACIA never modifies the interpretation of the Protocol to produce different test behavior. If our customers believe that an incorrect interpretation has been made, they have to send a contribution to the original Standard Organisation that has produced the test descriptions. On its side ACACIA, as member of ETSI for example, used to send contributions to those organisations to correct obvious TTCN mistakes (referred in the xxx_xxref.doc).

In the ETS package, a description in HTML of the implemented test suite is furnished (including all modifications). This can be consulted and compared to the original test description. All the modifications, due to TTCN errors or implementation constraints, are summarised in the xxx_xxref.doc available also in the ETS package. This word document gives the reference of the basis ATS used to make the ETS.

3.2 Check while Running the test suite

While running the test suite, each analysed received or sent event is printed in the traces as a TTCN event. The user can follow the branches of the tree and tabular notation used during the execution of the test. Those that match are summarised in the final table of the execution of the test. The path followed in the tree can be compared to the proposed tree in the published description. To consult the contents of constraints, the user can use the HTML link to open the TTCN tables associated with them.

The contents of sent message can be checked more precisely in HEXA or IDENTIFIED mode by changing the display option in the clarinet filter.

4. Conclusion

Because an SO like ETSI or ITU-T doesn’t propose any referenced model of an IUT to prove the conformance of an implementation, the validation can only be done by consulting and comparing with the published document and all documents associated with the implemented version (HTML test suite, xxx_xxref.doc).

5. ETS conformity: samples of information

5.1 ATS references

Hereafter is an example extracted from a 'Reference Note' showing specification of the protocol and the of the ATS source.

System under test	ISDN User part Version 2
Protocol specifications	ETSI Integrated Services Digital Network (ISDN); Signalling System No.7; ISDN User Part (ISUP) version 2 for the international interface; Part 33: Abstract Test Suite (ATS) and partial Protocol Implementation eXtra Information for Testing (PIXIT) proforma specification for basic services [ITU-T Recommendation Q.784.2 (1997), clauses 2 to 5 and annexes B to C, modified]
ATS specifications	ETSI: ETS 300 356-33
ATS file name	rdl.mp dated from 24/11/1997

5.2 ATS modifications

Hereafter is an example extracted from a 'Reference Note' explaining the modification applied to a TestCase which has an error.

Those constraint are used as sent event and the invoke id missing.

DeActDivinv1

ASN.1 Type Constraint Declaration	
Constraint Name: ASN.1 Type: Derivation Path: Comment:	**DeActDivinv1(inv_id: InvokeIDType;PROC: Procedure;basicservice:BasicService)** Component ASN1_Encoding: BER Send Component: DeactivationDiversion invoke component

ASN.1 Constraint Value

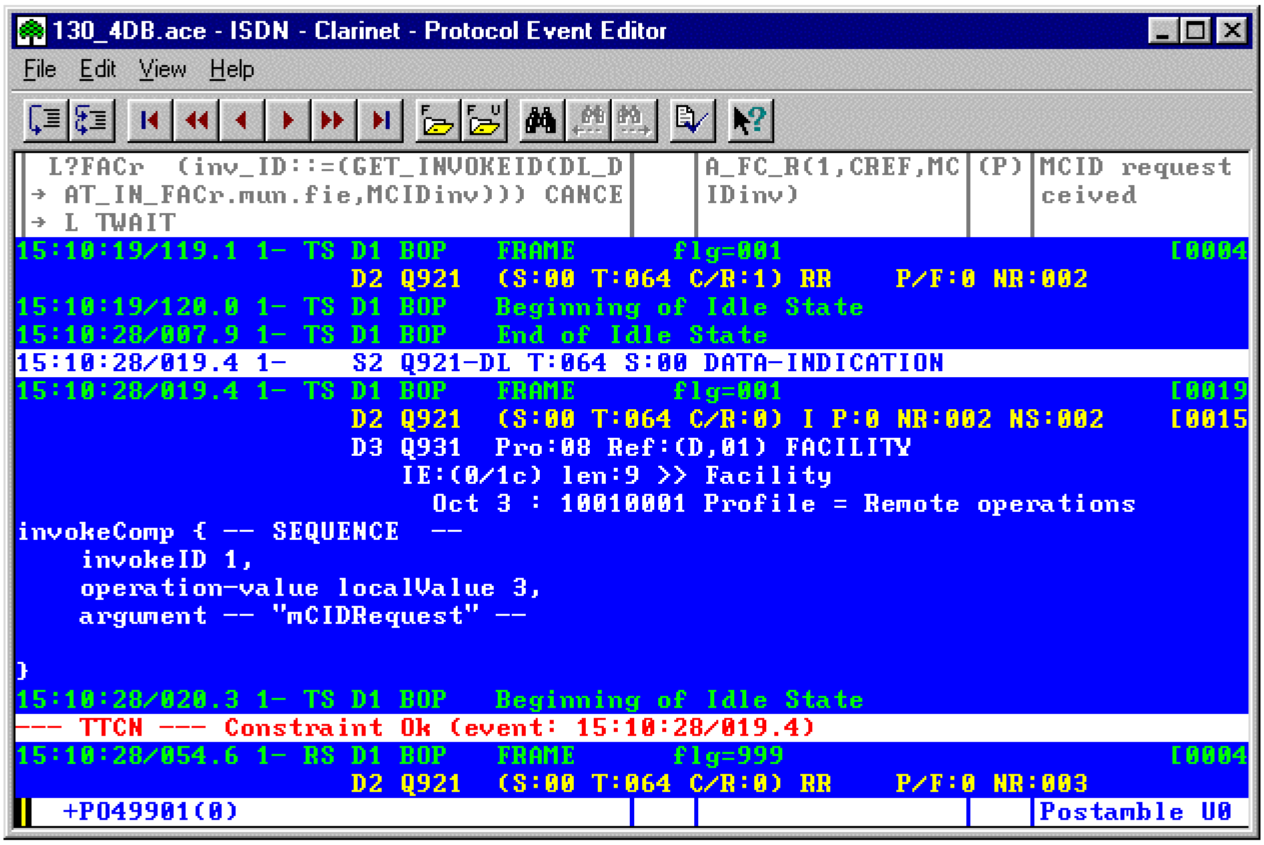
```
DeactivationDiversion_Components
  DeactivationDiversion_InvokeComp
   {invokeID            inv_id,
    operation_value      localValue 8,
    argument  {     procedure            PROC,
                           basicService        basicservice,
                           servedUserNr     allNumbers          NULL
                     }
   }
```

Detailed Comments:

      use with ActDivInv

5.3 Event Traces

The screen copy hereafter shows the running of an ISDN MCID TestCase in front of a Clarinet IUT invoking the MCID service.

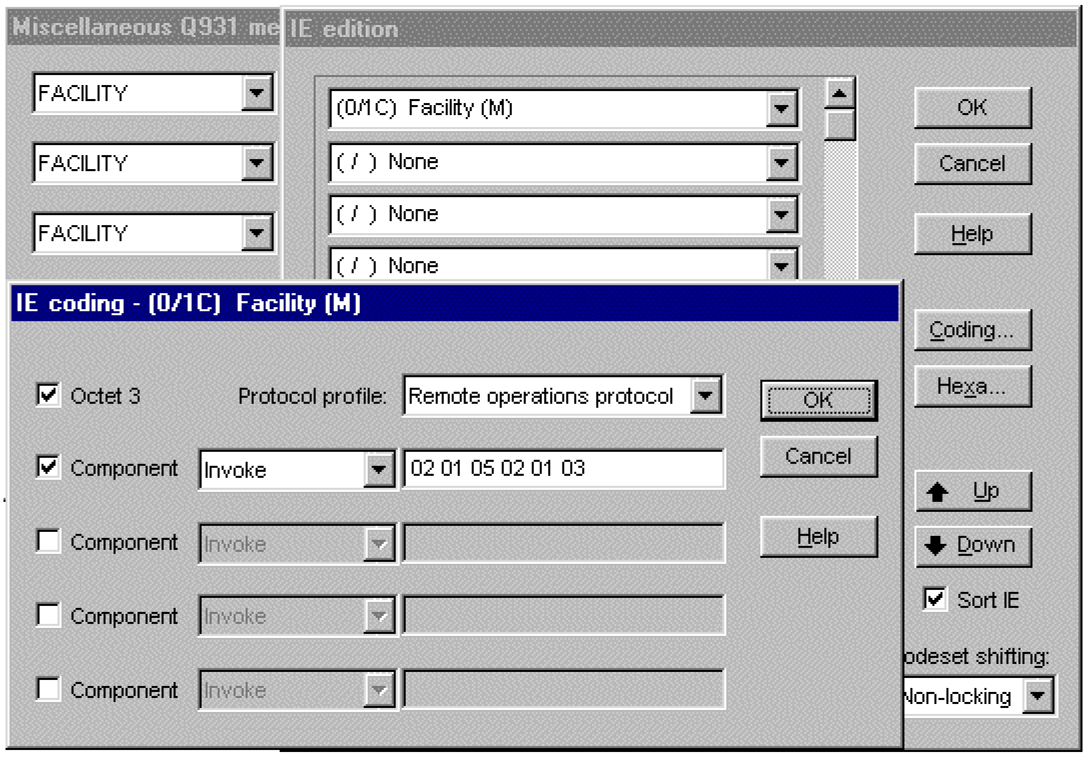


```
 L?FACr  (inv_ID::=(GET_INVOKEID(DL_D  |  A_FC_R(1,CREF,MC |(P)|MCID request
→ AT_IN_FACr.mun.fie,MCIDinv))) CANCE |  IDinv)           |   | ceived
→ L TWAIT
15:10:19/119.1 1- TS D1 BOP   FRAME       flg=001                        [0004
                    D2 Q921  (S:00 T:064 C/R:1) RR    P/F:0 NR:002
15:10:19/120.0 1- TS D1 BOP   Beginning of Idle State
15:10:28/007.9 1- TS D1 BOP   End of Idle State
15:10:28/019.4 1-     S2 Q921-DL T:064 S:00 DATA-INDICATION
15:10:28/019.4 1- TS D1 BOP   FRAME       flg=001                        [0019
                    D2 Q921  (S:00 T:064 C/R:0) I P:0 NR:002 NS:002     [0015
                    D3 Q931  Pro:08 Ref:(D,01) FACILITY
                       IE:(0/1c) len:9 >> Facility
                         Oct 3 : 10010001 Profile = Remote operations
invokeComp { -- SEQUENCE  --
    invokeID 1,
    operation-value localValue 3,
    argument -- "mCIDRequest" --

}
15:10:28/020.3 1- TS D1 BOP   Beginning of Idle State
--- TTCN --- Constraint Ok (event: 15:10:28/019.4)
15:10:28/054.6 1- RS D1 BOP   FRAME       flg=999                        [0004
                    D2 Q921  (S:00 T:064 C/R:0) RR    P/F:0 NR:003
  +P049901(0)                                        |Postamble U0
```

5.4 IUT profile

The screen copy hereafter shows the IUT with a Facility message defined to invoke the ISDN MCID service.



FACILITY	(0/1C) Facility (M)
FACILITY	(/) None
FACILITY	(/) None
	(/) None

Sort IE: ☑
odeset shifting: Non-locking

☑ Octet 3	Protocol profile:	Remote operations protocol
☑ Component	Invoke	02 01 05 02 01 03
☐ Component	Invoke	
☐ Component	Invoke	
☐ Component	Invoke	