OSEK/VDX

NM test procedure

Version 2.0

May 4th, 1999

What is OSEK/VDX?

OSEK/VDX is a joint project of the automotive industry. It aims at an industry standard for an open-ended architecture for distributed control units in vehicles.
A real-time operating system, software interfaces and functions for communication and network management tasks are thus jointly specified.
The term OSEK means ”Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug” (Open systems and the corresponding interfaces for automotive electronics).
The term VDX means „Vehicle Distributed eXecutive“. The functionality of OSEK operating system was harmonized with VDX. For simplicity OSEK will be used instead of OSEK/VDX in the document.

OSEK partners:

Adam Opel AG, BMW AG, Daimler-Benz AG, IIIT University of Karlsruhe, Mercedes-Benz AG, Robert Bosch GmbH, Siemens AG, Volkswagen AG.
GIE.RE. PSA-Renault (Groupement d’intérêt Economique de Recherches et d’Etudes PSA-Renault).

Motivation:

- High, recurring expenses in the development and variant management of non-application related aspects of control unit software.
- Incompatibility of control units made by different manufacturers due to different interfaces and protocols.

Goal:

Support of the portability and reusability of the application software by:

- Specification of interfaces which are abstract and as application-independent as possible, in the following areas: real-time operating system, communication and network management.
- Specification of a user interface independent of hardware and network.
- Efficient design of architecture: The functionality shall be configurable and scaleable, to enable optimal adjustment of the architecture to the application in question.
- Verification of functionality and implementation of prototypes in selected pilot projects.

Advantages:

- Clear savings in costs and development time.
- Enhanced quality of the control units software of various companies.
- Standardized interfacing features for control units with different architectural designs.
- Sequenced utilization of the intelligence (existing resources) distributed in the vehicle, to enhance the performance of the overall system without requiring additional hardware.
- Provides absolute independence with regards to individual implementation, as the specification does not prescribe implementation aspects.

OSEK conformance testing

OSEK conformance testing aims at checking conformance of products to OSEK specifications. Test suites are thus specified for implementations of OSEK operating system, communication and network management.

Work around OSEK conformance testing is supported by the MODISTARC project sponsored by the Commission of European Communities. The term MODISTARC means ”Methods and tools for the validation of OSEK/VDX based DISTributed ARChitectures”.

This document has been drafted by MODISTARC members:

Harald Heinecke	BMW AG
Wolfgang Kremer	BMW AG
Benoit Caillaud	INRIA
Dirk John	IIIT, Karlsruhe University
Yevgeny Shakuro	Motorola GmbH
Barbara Ziker	Motorola GmbH
Jean-Paul Cloup	Peugeot Citroën S.A.
Jean-Emmanuel Hanne	Peugeot Citroën S.A.
Samuel Boutin	Renault S.A.
Patrick Palmieri	Siemens Automotive SA
Didier Stunault	Thomson-CSF Detexis

TABLE OF CONTENTS

1. Introduction

1.1. Scope

This document specifies a test procedure for services and protocols of the OSEK NM as defined in specification document [5].

This document applies to conformance test suites for testing implementations which claim conformance to the OSEK NM specification. The test procedure consists of a list of test cases building the OSEK NM test suite. A test case consists of a sequence of statements corresponding to one or more test purposes specified in document [2].

As OSEK NM implementations can operate either the Direct OSEK NM or the Indirect OSEK NM, the NM test suite has been divided into two parts accordingly.

1.2. References

[1] OSEK/VDX Conformance Testing Methodology - Version 1.0. - 19 December 1997.

[2] OSEK/VDX - NM test plan - Version 1.0. - April 30th, 1998.

[3] OSEK/VDX Operating System - Version 2.0 - revision 1 - 15 October 1997.

[4] OSEK/VDX Communication - Version 2.1 - revision 1 - 17th June 1998.

[5] OSEK Network Management - Concept and Application Programming Interface- Version 2.50 - 31th of May 1998.

[6] ISO/IEC 9646-1 - Information technology, Open Systems Interconnection, Conformance testing methodology and framework, *part 1 : General Concepts*, 1992.

[7] ISO/IEC 9646-3 - Information technology, Open Systems Interconnection, Conformance testing, methodology and framework, *part 3 : The Tree and Tabular Combined Notation (TTCN),* 1992.

1.3. Abbreviations

API	Application Programming Interface
EUT	Equipment Under Test
ISO	International Standard Organization
IUT	Implementation Under Test
LT	Lower Tester
NM	Network Management
NMPDU	Network Management - Protocol Data Unit
OS	Operating System
PDU	Protocol Data Unit
PICS	Protocol Implementation Conformance Statement

PIXIT	Protocol Implementation eXtra Information for Testing
SDL	Specification and Description Language
TE	Test Equipment
TMP	Test Management Protocol
TM_PDU	Test Management - Protocol Data Unit
TTCN	Tree and Tabular Combined Notation
UT	Upper Tester
UUDT	Unsegmented Unacknowledged Data Transfer

2. Test environment

2.1. Test architecture

According to the methodology described in document [1], the test architecture for NM conformance is split into two parts:

- the Equipment Under Test (EUT) which encompasses the NM implementation to be tested, also called Implementation Under Test (IUT),
- the Test Equipment (TE) which implements the test suite and is connected to the Equipment Under Test via the network data bus.

The test suite makes up the Lower Tester (LT) which communicates through the Test Management Protocol (TMP) with its counterpart of the EUT called Upper Tester (UT). UT's role is on one hand to perform all actions requested by the LT and on the other hand to send back all the information collected at the NM API.

Any communication protocol can be used for exchanges between LT and UT provided it has been validated before. The UUDT protocol of OSEK COM is a possible choice as described in Figure 1 and Figure 2 below. Anyway, the TMP is expressed in terms of application messages called TM_PDUs (Test Management - Protocol Data Units). This way, its specification is independent of the underlying communication protocols.

To check direct NM conformance, the LT will also have to exchange NMPDUs with the IUT as illustrated in Figure 1:

- NMPDUs will be sent in order to simulate the NM activity of the other network nodes,
- NMPDUs will be received and analysed in order to determine whether or not the IUT behaviour conforms to the NM specification.

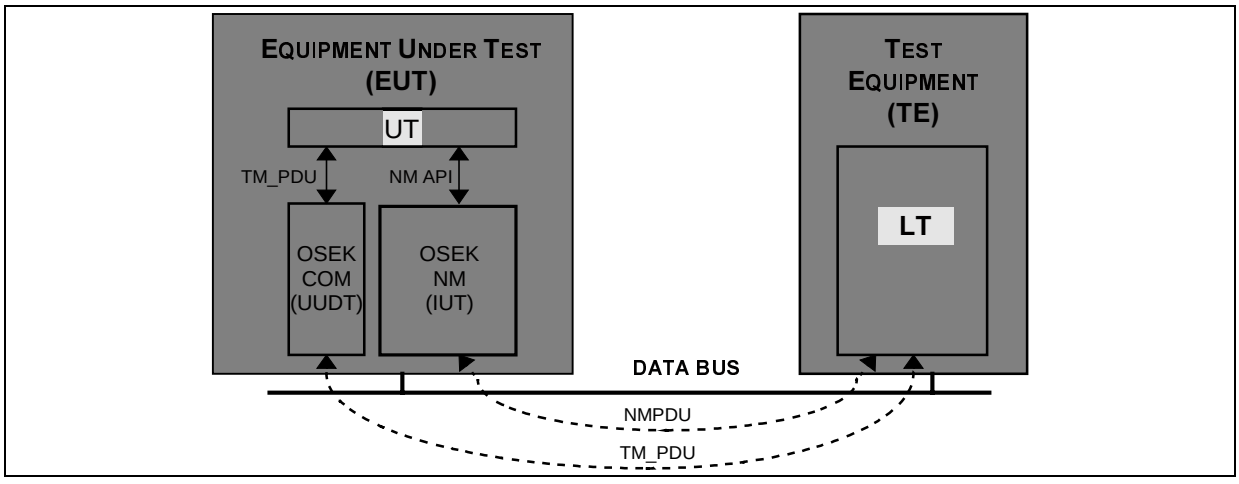

Figure 1 Test architecture for direct NM conformance

To check indirect NM conformance, the LT will have to send COM messages to the EUT in order to simulate the application data traffic on the network. Such messages will not be interpreted nor used by the UT. The purpose is to activate the COM/NM interface as required by indirect NM operation. The functionnality of this interface consists of:

- signalling reception of application messages monitored by the NM module,

- signalling expiry of message monitoring timers.

If the COM module is not OSEK and is unable to provide the required signalling, this functionnality has to be implemented inside the UT itself.

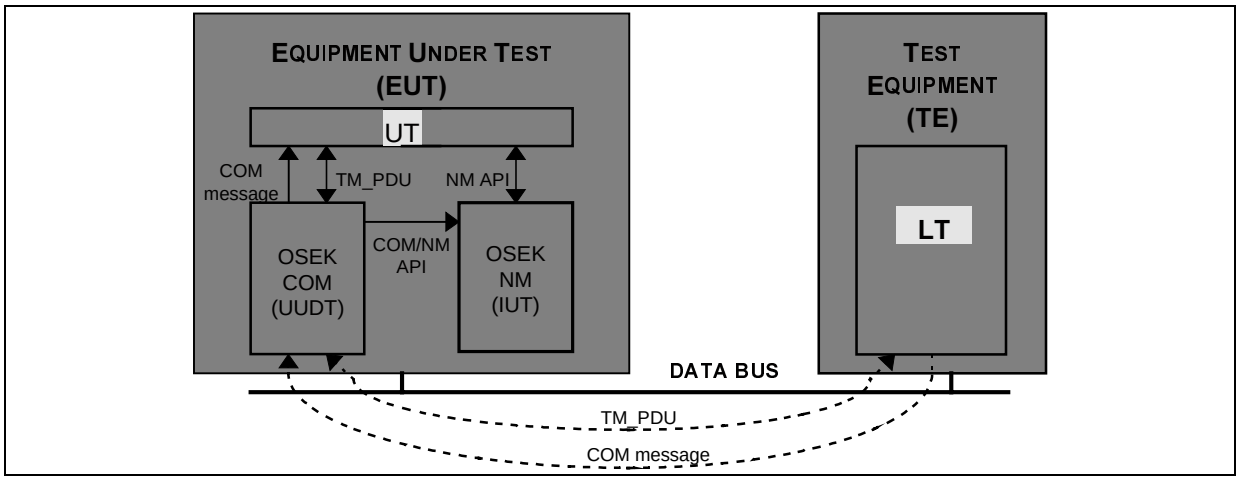

Figure 2 Test architecture for indirect NM conformance

Special TM_PDUs are specified to simulate network errors. They are not transmitted to the UT but interpreted by the lower communication layers which shall perform the requested actions. A possible approach is described in the next section.

2.2. Requirements

2.2.1. OS requirements

The test architecture for NM conformance includes a test application called UT and implemented in the same equipment unit as the IUT. UT implementation does not require special OS functionnality. The UT can be integrated in the same environment as the NM. Like the NM, it only needs task and alarm management services and it can be based on a non-OSEK OS providing equivalent functionnality.

The configuration of the UT can vary according to the NM configuration itself. For instance, one or more tasks will need to be implemented depending on the number of tasks that can be activated by the NM implementation. The configuration will also depend on the OS conformance class, the scheduling mechanisms and the inter-task communication (task activation or event setting).

Therefore, this document does not specify a configuration for the UT. It describes the operation of UT when it receives commands from the LT or information from the NM implementation, independently of the type and distribution of tasks and events.

2.2.2. Network perturbations

Since a large part of the NM specification is devoted to error recovery mechanisms, it is highly recommended that conformance test systems should be able to simulate network perturbations.

Indeed, the NM specification deals with the following perturbations:

- bus blocked (e.g. CAN BusOff),
- no transmission of NM frames.

Therefore, the NM test suite's specification assumes that such errors can be reported to the IUT at the Data Link interface. They could be generated either locally inside the TE or remotely from the EUT. Some TM_PDUs have been especially defined to control and manage error simulation from the LT. They are not transmitted to the UT:

- In the local option, they are interpreted by special test software implemented at the network driver interface inside the EUT. Such a software shall provide the IUT with the appropriate error reports.
- In the remote option, they are processed by special test software inside the TE which shall generate the requested Data Bus perturbations using some appropriate hardware.

The picture below illustrates these two options. It shows the location of the added "test software" and the path of error simulation TMPDUs in both configurations. In the remote option, the network perturbations are generated by a "Bus Manipulator" driven by the test software.

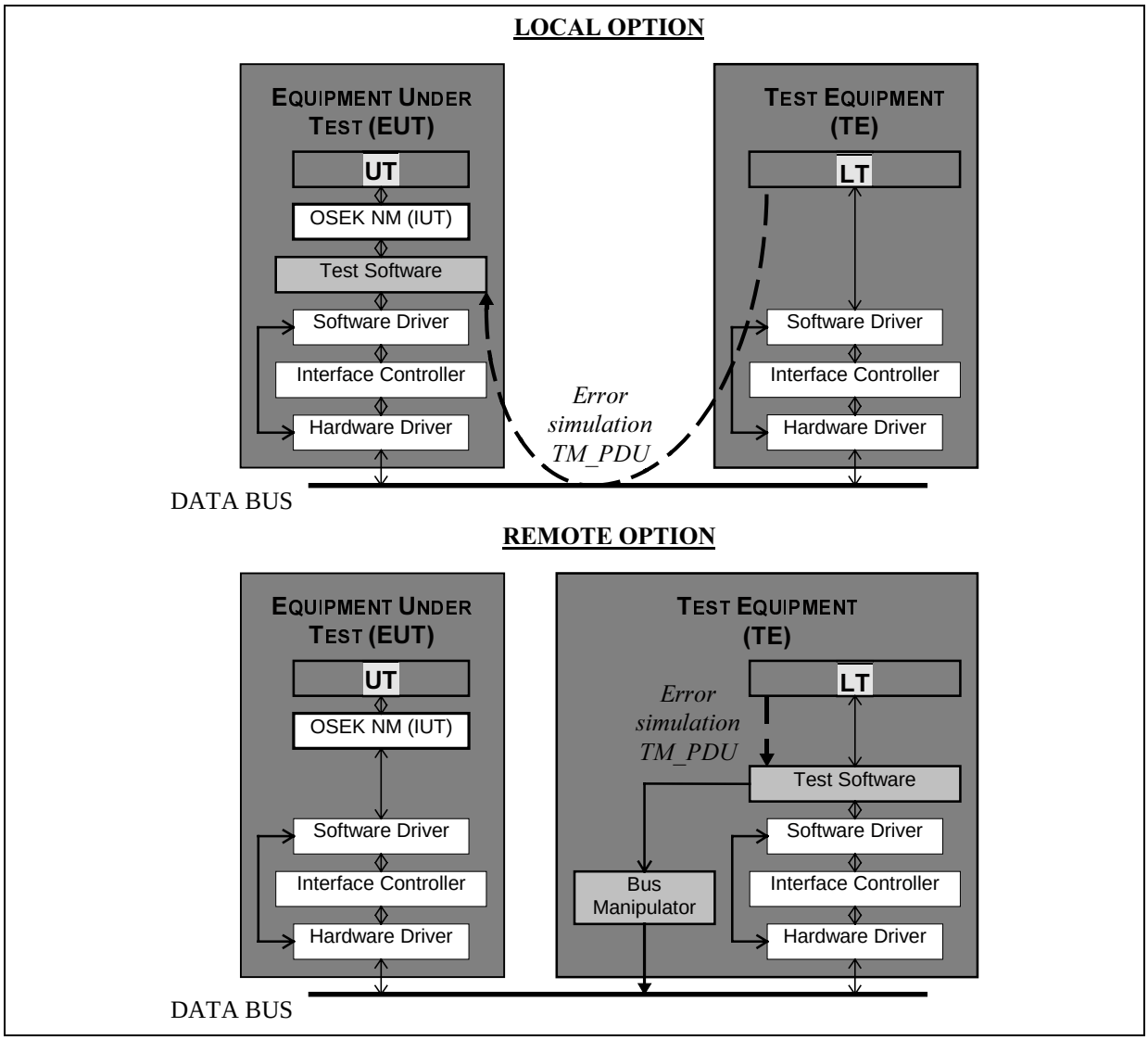

Figure 3 Architectures for error simulation

The local option is the more flexible and it allows to easily simulate all types of perturbations. But it requires modifications of the EUT software. The remote option requires additional hardware means and it may be more difficult to implement.

However, it should be pointed out that the local option only impacts software/hardware of the network interface. Whatever the selected error simulation technique, the UT, LT and IUT software need not be modified.

If error simulation is not possible, a reduced test suite can be executed. But the NM functionnality will not be completely checked.

2.2.3. COM requirements

The NM test suite's specification assumes that three COM messages are utilised for exchanges between the LT and the UT:

- Two messages are used by the TMP protocol to convey the TM_PDUs between the LT and the UT in either direction.
- The third message is used to test message transmission capability from the UT. According to the specification, the NM implementation must sometimes execute the D_Offline or the D_Online services to enable or to disable user's data transmission. The message is then used to verify whether or not those services were called as specified. Moreover, in Indirect NM, this message represents the user's message whose transmission is monitored by the Indirect NM.

2.2.4. Test tool adaptation

It is anticipated that some adaptation of the test tool will take place before running the test suites on a given IUT. Indeed, the NM specification does not define the format and the encoding rules of PDUs (direct NM) and of API parameters. The same approach will be followed in this document about the TMP specification. Only the structure of TMP messages is specified, not the data encoding formats.

Therefore, two different adaptations will be required:

- adaptation to comply with NMPDUs formats,
- adaptation to comply with API parameter formats.

3. Features and parameters

The NM specification defines optional features and allows different configurations of the specification parameters. Prior to any test suite execution, it is necessary to get a precise knowledge of what features and functions are supported and what parameter values or range of values are permissible. Such information has to be supplied by implementators in standard questionnaires defined hereafter. It will be then used to configure the test environment and to determine which tests can be executed.

Two questionnaires are to be provided. The first one is called PICS. It contains a statement of the capabilities and options which have been implemented. Each question pertains to one of the specification requirements, mandatory or optional. The PICS helps to determine whether all the mandatory features have been implemented and hence it allows a static evaluation of IUT conformance before test suite execution. The PICS is a fixed-format questionnaire in which the questions are simply answered Yes or No.

The second questionnaire is called PIXIT. It provides with additional information required to run the conformance tests. PIXIT questions ask for parameter values pertaining to the IUT and to the testing environment such as time-out values or addressing information. Anwers are used to parameterize the test suite and configure the LT and the UT.

3.1. Format of the questionnaires

The questionnaire tables consists of four columns for the PICS and five for the PIXIT:

- Item: specifies an identifier which can be used as a reference in other questions
- Service / protocol features or parameters: specifies the nature of requested information
- Status: gives a status of the feature/parameter in the specification (Mandatory, Optional)
- Support: indicates whether the feature/parameter has been implemented or not. This column is to be filled in by IUT implementators.
- Value: specifies the related parameter value (PIXIT only). This column is to be filled in by IUT implementators.

The questionnaires make use of the following symbols or abbreviations:

- Status column:
  - M Mandatory
  - O Optional
  - *pred:* Conditional expression where *pred* refers to the item that needs to be supported for the condition to apply. Conditions may contain logical expressions using the following symbols:
    - | logical OR,
    - . (dot) logical AND.
- Support column:
  - Yes feature/parameter supported
  - No feature/parameter not supported
  - N/A Not Applicable due to not matched condition

The support column does only propose answers meeting compliance requirements. For instance, if the feature or parameter is mandatory only a Yes answer is presented. Answering No means non-compliance. Doing this, static conformance analysis becomes straightforward.

Whenever a condition is specified in the status column, a "N/A" answer is proposed and should be ticked if the IUT does not match the condition. The condition defines what should be answered to some previous questions in order to keep the present statement meaningful. No condition is expressed when the statement is depending on previous answers relating to mandatory features (since such answers should normally be Yes).

3.2. Direct NM

3.2.1. PICS

The following questionnaires intend to provide a comprehensive list of direct NM features and options in order to determine the IUT capabilities with great accuracy. Protocol capabilities are listed before services features since the latter are directly connected to protocol implementation.

3.2.1.1. Overall capabilities

Item	Protocol Feature	Status	Support
	Operating modes supported:		
Ona	– Normal/Active mode	M	_Yes
Ola	– Limphome/Active mode	M	_Yes
Opa	– Passive Mode	O	_Yes _No
	– Network-wide BusSleep Mode		
Obr	• as receiver	O	_Yes _No
Obi	• as initiator	Obr:O	_Yes _No _N/A
	Miscelleanous:		
Rd	– Ring Data forwarding	O	_Yes _No
Dis	– Disabling/enabling user communication when entering/ leaving limphome state	M	_Yes

3.2.1.2. Network information supported

Item	Protocol Feature	Status	Support
	Network information managed by the IUT:		
Ni1	– Normal configuration	M	_Yes
Ni2	– Limphome configuration	M	_Yes
Ni3	– Position inside logical ring (predecessor, successor)	M	_Yes
Ni4	– Network status	O	_Yes _No
Ni5	– Ring data	Rd:M	_Yes _N/A

	If Network Status is implemented, is the following information available ? [1]				
Ns1	– Present network configuration stable/not stable	Ni4:O	_Yes	_No	_N/A
Ns2	– No error/error bus blocked	Ni4:O	_Yes	_No	_N/A
Ns3	– NMPassive/NMActive	Ni4:O	_Yes	_No	_N/A
Ns4	– NMOn/NMOff	Ni4:O	_Yes	_No	_N/A
Ns5	– no NMLimphome/NMLimphome	Ni4:O	_Yes	_No	_N/A
Ns6	– no NMBusSleep/NMBusSleep	P1[2]:O	_Yes	_No	_N/A
Ns7	– no NMTwbs/NMTwbs(Normal/Limphome)	P1[2]:O	_Yes	_No	_N/A
Ns8	– using of Ring Data allowed/not allowed	Ni4.Rd:O	_Yes	_No	_N/A
Ns9	– GotoMode(Awake)/GotoMode(BusSleep) called	Ni4.Obi:O	_Yes	_No	_N/A

[1] At least one of the following bits of information must be supported.
[2] P1 = Ni4.(Obr|Obi)

3.2.1.3. Protocol events

Item	Protocol Feature	Status	Support
	NMPDUs supported		
	– Ring message		
Rmr	• as receiver	M	_Yes
Rmt	• as transmitter	M	_Yes
	– Alive message		
Amr	• as receiver	M	_Yes
Amt	• as transmitter	M	_Yes
	– Limphome message		
Lmr	• as receiver	M	_Yes
Lmt	• as transmitter	M	_Yes
	Miscelleanous:		
Ev1	– Moving to Limphome when bus blocked	M	_Yes
Ev2	– Moving to Limphome when tx_limit exceeded	M	_Yes
Ev3	– Moving to Limphome when rx_limit exceeded	M	_Yes

3.2.1.4. NMPDU fields

Item	Protocol Feature	Status	Support	
	NMPDU fields supported			
Pf1	– Source	M	_Yes	
Pf2	– Destination	M	_Yes	
Pf3	– Code (ring, alive, limphome)	M	_Yes	
Pf4	– Sleep.ind	Obr|Obi:M	_Yes	_N/A
Pf5	– Sleep.ack (ring message only)	Obr|Obi:M	_Yes	_N/A
Pf6	– RingData (ring message only)	Rd:M	_Yes	_N/A

3.2.1.5. NM API capabilities

Item	Service Feature	Status	Support		
	NM API calls supported:				
Sv1	– InitConfig	O[(1)]	_Yes	_No	
Sv2	– GetConfig	M	_Yes		
Sv3	– CmpConfig	O[(2)]	_Yes	_No	
Sv4	– SelectDeltaConfig	O[(3)]	_Yes	_No	
Sv5	– StartNM	M	_Yes		
Sv6	– StopNM	M	_Yes		
Sv7	– GotoMode	Obi:M[(4)]	_Yes		_N/A
Sv8	– GetStatus	Ni4:O[(5)]	_Yes	_No	_N/A
Sv9	– CmpStatus	Ni4:O[(6)]	_Yes	_No	_N/A
Sva	– SelectDeltaStatus	Ni4:O[(7)]	_Yes	_No	_N/A
Svb	– SilentNM	Opa:M[(8)]	_Yes		_N/A
Svc	– TalkNM	Opa:M[(8)]	_Yes		_N/A
Svd	– TransmitRingData	Rd:O[(9)]	_Yes	_No	_N/A
Sve	– ReadRingData	Rd:O[(9)]	_Yes	_No	_N/A
	NM indication capabilities				
	– Can the NM indicate a normal configuration change				
Inct	• by task activation	O	_Yes	_No	
Ince	• by event setting	¬Inct:O	_Yes	_No	
	– Can the NM indicate a limphome configuration change				
Ilct	• by task activation	O	_Yes	_No	
Ilce	• by event setting	¬Ilct:O	_Yes	_No	_N/A
	– Can the NM indicate a network status change				
Inst	• by task activation	Ni4:O	_Yes	_No	_N/A
Inse	• by event setting	Ni4.¬Inst:O	_Yes	_No	_N/A
	– Can the NM indicate ring data reception				
Irdt	• by task activation	Rd:O	_Yes	_No	_N/A
Irde	• by event setting	Rd.¬Irdt:O	_Yes	_No	_N/A

[(1)] referred to as *InitConfig* option in NM test plan
[(2)] referred to as *CmpConfig* option in NM test plan
[(3)] referred to as *SelectConfig* option in NM test plan
[(4)] referred to as *BusSleep* option in NM test plan
[(5)] referred to as *NMStatus* option in NM test plan
[(6)] referred to as *CmpStatus* option in NM test plan
[(7)] referred to as *SelectStatus* option in NM test plan
[(8)] referred to as *Active/Passive* option in NM test plan
[(9)] referred to as *RingData* option in NM test plan

3.2.1.6. NM API parameters

Item	Service Feature	Status	Support	
	InitConfig parameters:			
Icp1	– NetId	Sv1:M	_Yes	_N/A

	GetConfig parameters:			
Gcp1	– NetId	M	_Yes	
Gcp2	– Config	M	_Yes	
Gcp3	– ConfigKind	M	_Yes	
	CmpConfig parameters:			
Ccp1	– NetId	Sv3:M	_Yes	_N/A
Ccp2	– TestConfig	Sv3:M	_Yes	_N/A
Ccp3	– RefConfig	Sv3:M	_Yes	_N/A
Ccp4	– CMask	Sv3:M	_Yes	_N/A
	SelectDeltaConfig parameters:			
Scp1	– NetId	Sv4:M	_Yes	_N/A
Scp2	– ConfigKind	Sv4:M	_Yes	_N/A
Scp3	– ConfigHandle	Sv4:M	_Yes	_N/A
Scp4	– CMaskHandle	Sv4:M	_Yes	_N/A
	StartNM parameters:			
Sap1	– NetId	M	_Yes	
	StopNM parameters:			
Sop1	– NetId	M	_Yes	
	GotoMode parameters:			
Gmp1	– NetId	M	_Yes	_N/A
Gmp2	– NewMode	M	_Yes	_N/A
	GetStatus parameters:			
Gsp1	– NetId	Sv8:M	_Yes	_N/A
Gsp2	– NetworkStatus	Sv8:M	_Yes	_N/A
	CmpStatus parameters:			
Csp1	– NetId	Sv9:M	_Yes	_N/A
Csp2	– TestStatus	Sv9:M	_Yes	_N/A
Csp3	– RefStatus	Sv9:M	_Yes	_N/A
Csp4	– SMask	Sv9:M	_Yes	_N/A
	SelectDeltaStatus parameters:			
Scp1	– NetId	Sva:M	_Yes	_N/A
Scp2	– StatusHandle	Sva:M	_Yes	_N/A
Scp3	– SMaskHandle	Sva:M	_Yes	_N/A
	SilentNM parameters:			
Sip1	– NetId	Svb:M	_Yes	_N/A
	TalkNM parameters:			
Tap1	– NetId	Svc:M	_Yes	_N/A
	ReadRingdata parameters:			
Rdp1	– NetId	Svd:M	_Yes	_N/A
Rdp2	– RingData	Svd:M	_Yes	_N/A
	TransmitRingdata parameters:			
Tdp1	– NetId	Sve:M	_Yes	_N/A
Tdp2	– RingData	Sve:M	_Yes	_N/A

3.2.1.7. NM API return codes

Note: There is no statement regarding CmpStatus and CmpConfig. The returned code (true/false) yields the result of comparison and should be considered as part of the respective procedure implementation.

Item	Service Feature	Status	Support
	Is E_OK return code supported by: (1)		
Eok1	– InitConfig	Sv1:O	_Yes _No _N/A
Eok2	– GetConfig	O	_Yes _No
Eok5	– StartNM	O	_Yes _No
Eok6	– StopNM	O	_Yes _No
Eok7	– GotoMode	Sv7:O	_Yes _No _N/A
Eok8	– GetStatus	Sv8:O	_Yes _No _N/A
Eokb	– SilentNM	Svb:O	_Yes _No _N/A
Eokc	– TalkNM	Svc:O	_Yes _No _N/A
Eokd	– TransmitRingData	Svd:M	_Yes _No _N/A
Eoke	– ReadRingData	Sve:M	_Yes _No _N/A
	Is E_notOK return code supported by: (1)		
Enokd	– TransmitRingData	Svd:M	_Yes _No _N/A
Enoke	– ReadRingData	Sve:M	_Yes _No _N/A

(1) referred to as *Status* option in NM test plan

3.2.2. PIXIT

The following questionnaires intend to provide actual values for implementation-dependent parameters stated in the NM specification. They also ask for some test parameters required to run the test cases. The values supplied by the IUT designer will be picked up to parameterize the test suite. It is understood here that some work is needed before to adapt the test environment to the actual implementation formats of NMPDU fields and API parameters (size, range of values...). There is no statement relating to such information in the questionnaires.

3.2.2.1. Protocol parameters

- Ring configuration

  To check the direct NM protocol, the LT needs to simulate other nodes of the logical ring. Therefore, the test user will be asked for four node addresses called SN1, SN2, PN1, PN2 and respecting the following sequence on the ring (NodeId represents the IUT node address):

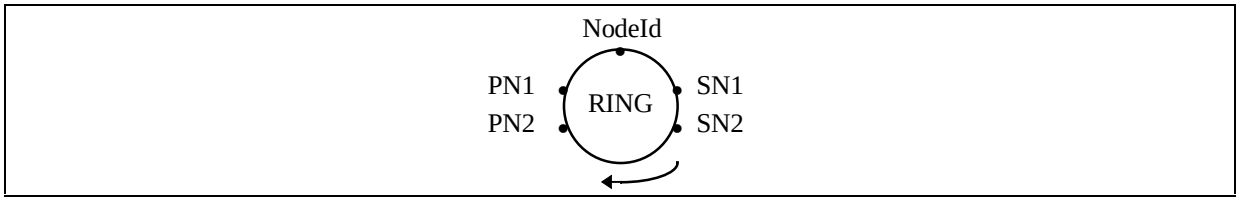

Figure 4 Logical ring configuration for the test suite

- Expiration window timers

  To check protocol timer implementation, a time window has to be defined where IUT outputs triggered by timer expiry can be accepted. For instance, to check an assertion such as "the NM transmits a ring message after T_{Typ}", the LT will firstly wait for T_{Typ} and verify that nothing has been received, secondly wait for the T_{Typ} window expiration and verify that the ring message has been received.

  A time window is therefore defined for each protocol timer.

Item	Protocol parameter	Status	Support	Value
	Ring configuration:			
Rc0	– Maximum number of nodes supported in network configuration	M	>= 2	
Rc1	– NodeId	M	_Yes	
Rc2	– SN1	M	_Yes	
Rc3	– PN1	Rc0>2:M	_Yes _N/A	
Rc4	– SN2	Rc0>3:M	_Yes _N/A	
Rc5	– PN2	Rc0>4:M	_Yes _N/A	
	Other parameters:			
Pp1	– rx_limit	M	_Yes	
Pp2	– tx_limit	M	_Yes	
Pp3	– reserved area of Opcode in NMPDUs	M	_Yes	
Pp4	– size of ring data in ring message	Rd:M	_Yes _N/A	
	Protocol timers:			
Wt1	– Ttyp	M	_Yes	
Wt2	– Tmax	M	_Yes	
Wt3	– Terror	M	_Yes	
Wt4	– Twaitbussleep	Obr|Obi:M	_Yes _N/A	
	Expiration window timers:			
Wt1	– TtypW	M	_Yes	
Wt2	– TmaxW	M	_Yes	
Wt3	– TerrorW	M	_Yes	
Wt4	– TwaitbussleepW	Obr|Obi:M	_Yes _N/A	

3.2.2.2. API parameters

- Indication of configuration change

  To check the functionnality of task activation or event setting on change of configuration, the Upper Tester must implement the associated tasks or events. The test user will be asked for the config handle and mask handle values processed by the IUT. And for each handle/mask association he must provide the list of nodes that must send an alive or ring message to generate an indication at the NM API.

- Indication of network status change

  To check the functionnality of task activation or event setting on change of network status, the Upper Tester must implement the associated tasks or events. The test user will be asked for the status handle and mask handle values processed by the IUT. And

for each handle/mask association he must provide the list of necessary status changes to generate an indication at the NM API.

- Indication of ring data reception

  To check the functionnality of task activation or event setting on ring data reception, the Upper Tester must implement the associated tasks or events.

Item	Service parameter	Status	Support		Value
Nid	NetId	M	_Yes		
	API return status:				
Rs1	– E_OK	M	_Yes		
Rs2	– E_notOK	Svd|Sve:M	_Yes	_N/A	
Rs3	– TRUE	Sv3|Sv9:M	_Yes	_N/A	
Rs4	– FALSE	Sv3|Sv9M	_Yes	_N/A	
	Task identifiers for NM indications:				
Ti1	– normal configuration change	Inct:M	_Yes	_N/A	
Ti2	– limphome configuration change	Ilct:M	_Yes	_N/A	
Ti3	– network status change	Inst:M	_Yes	_N/A	
Ti4	– ring data reception	Irdt:M	_Yes	_N/A	
	Event masks for NM indications:				
Em1	– normal configuration change	Ince:M	_Yes	_N/A	
Em2	– limphome configuration change	Ilce:M	_Yes	_N/A	
Em3	– network status change	Inse:M	_Yes	_N/A	
Em4	– ring data reception	Irde:M	_Yes	_N/A	
	Handles for config change indication:				
Hc1	– table of config handles	Inct|Ince|	_Yes	_N/A	
Hc2	– table of associated mask handles	Iect|Iece:	_Yes	_N/A	
Hc3	– table of node lists	M	_Yes	_N/A	
	Handles for status change indication:				
Hs1	– table of status handles	Ist|Ise:M	_Yes	_N/A	
Hs2	– table of associated mask handles	Ist|Ise:M	_Yes	_N/A	
Hs3	– table of lists of status changes	Ist|Ise:M	_Yes	_N/A	

3.2.2.3. Test suite parameters

- Test execution timers

  The following timers are defined to manage the test execution:

  Tresp: this timer is started when the LT is waiting for an NMPDU or a TMPDU from the EUT. If it expires, the test will conclude that no response is forthcoming.

  Twait: this timer is started when the LT must wait for a certain amount of time before sending the next NMPDU or TMPDU. This can happen when the LT has to send two PDUs consecutively and the IUT needs to terminate the first action before being able or entitled to accept the second PDU. The latter is sent after Twait expiry.

Item	Test suite parameter	Status	Support	Value
	Test execution timers:			
Tt1	– Tresp	M	_Yes	
Tt2	– Twait	M	_Yes	

3.3. Indirect NM

3.3.1. PICS

The following questionnaires intend to provide a comprehensive list of indirect NM features and options in order to determine the IUT capabilities with great accuracy. Protocol capabilities are listed before services features since the latter are directly connected to protocol implementation.

3.3.1.1. Overall capabilities

Item	Protocol Feature	Status	Support
	Node monitoring mechanism supported:		
Mtr	– Transmission monitoring	M	_Yes
Mre	– Reception monitoring	M	_Yes
Mni	– Network interface status monitoring	M	_Yes
	Time-out monitoring mechanism supported:		
Gt	– One global time-out	O1[(1)]	_Yes _No
Mt	– One monitoring time-out per message	O1[(1)]	_Yes _No
	Operating modes supported:		
On	– Normal mode	M	_Yes
Ol	– Limphome mode	M	_Yes
Obs	– BusSleep Mode	Mt:O	_Yes _No _N/A
	Miscelleanous:		
Dis	– Disabling/enabling user communication when entering/ leaving limphome state	M	_Yes

[(1)] O1: these two options are exclusive each other. One of them must be supported.

3.3.1.2. Network information supported

Item	Protocol Feature	Status	Support
	Network information managed by the IUT:		
Ni1	– Normal configuration	M	_Yes
Ni2	– Extended configuration	Mt:M	_Yes _N/A
Ni3	– Network status	O	_Yes _No
Ni4	– Extended network status	Mt:O	_Yes _No _N/A
	If Network Status is implemented, is the following information available ? [(1)]		

Ns1	– No error/error bus blocked	Ni3:O	_Yes	_No	_N/A
Ns2	– NMOn/NMOff	Ni3:O	_Yes	_No	_N/A
Ns3	– no NMLimphome/NMLimphome	Ni3:O	_Yes	_No	_N/A
Ns4	– no NMBusSleep/NMBusSleep	Ni3.Obs:O	_Yes	_No	_N/A
Ns5	– no NMWaitBusSleep/NMWaitBusSleep	Ni3.Obs:O	_Yes	_No	_N/A
	If Extended Network Status is implemented, is the following information available ?				
En1	– no error	Ni2:M	_Yes		_N/A
En2	– error, communication possible	Ni2:O	_Yes	_No	_N/A
En3	– error, communication not possible	Ni2:M	_Yes		_N/A

(1) At least one of the following bits of information must be supported.

3.3.1.3. Protocol events

Item	Protocol Feature	Status	Support
	Message monitoring		
Imt	– Indication of monitored message transmission	M	_Yes
Imr	– Indication of monitored message reception	M	_Yes
	Miscelleanous:		
Ev1	– Moving to Limphome when bus blocked	M	_Yes

3.3.1.4. NM API capabilities

Item	Service Feature	Status	Support		
	NM API calls supported:				
Sv1	– InitConfig	Mt:M	_Yes		_N/A
Sv2	– GetConfig	M	_Yes		
Sv3	– CmpConfig	O(1)	_Yes	_No	
Sv4	– SelectDeltaConfig	O(2)	_Yes	_No	
Sv5	– StartNM	M	_Yes		
Sv6	– StopNM	M	_Yes		
Sv7	– GotoMode	Obs:M(3)	_Yes		_N/A
Sv8	– GetStatus	Ni3:O(4)	_Yes	_No	_N/A
Sv9	– CmpStatus	Ni3:O(5)	_Yes	_No	_N/A
Sva	– SelectDeltaStatus	Ni3:O(6)	_Yes	_No	_N/A
	NM indication capabilities				
	– Can the NM indicate a configuration change				
Inct	• by task activation	O	_Yes	_No	
Ince	• by event setting	¬Inct:O	_Yes	_No	
	– Can the NM indicate an extended configuration change				
Iect	• by task activation	O	_Yes	_No	
Iece	• by event setting	¬Iect:O	_Yes	_No	_N/A
	– Can the NM indicate a network status change				
Ist	• by task activation	Ni3:O	_Yes	_No	_N/A

Ise	• by event setting	Ni3.¬Inst:O	_Yes _No _N/A

(1) referred to as *CmpConfig* option in NM test plan
(2) referred to as *SelectConfig* option in NM test plan
(3) referred to as *BusSleep* option in NM test plan
(4) referred to as *NMStatus* option in NM test plan
(5) referred to as *CmpStatus* option in NM test plan
(6) referred to as *SelectStatus* option in NM test plan

3.3.1.5. NM API parameters

Item	Service Feature	Status	Support	
Icp1	InitConfig parameters: – NetId	Sv1:M	_Yes	_N/A
Gcp1 Gcp2 Gcp3	GetConfig parameters: – NetId – Config – ConfigKind	 M M M	 _Yes _Yes _Yes	
Ccp1 Ccp2 Ccp3 Ccp4	CmpConfig parameters: – NetId – TestConfig – RefConfig – CMask	 Sv3:M Sv3:M Sv3:M Sv3:M	 _Yes _Yes _Yes _Yes	 _N/A _N/A _N/A _N/A
Scp1 Scp2 Scp3 Scp4	SelectDeltaConfig parameters: – NetId – ConfigKind – ConfigHandle – CMaskHandle	 Sv4:M Sv4:M Sv4:M Sv4:M	 _Yes _Yes _Yes _Yes	 _N/A _N/A _N/A _N/A
Sap1	StartNM parameters: – NetId	 M	 _Yes	
Sop1	StopNM parameters: – NetId	 M	 _Yes	
Gmp1 Gmp2	GotoMode parameters: – NetId – NewMode	 M M	 _Yes _Yes	 _N/A _N/A
Gsp1 Gsp2	GetStatus parameters: – NetId – NetworkStatus	 Sv8:M Sv8:M	 _Yes _Yes	 _N/A _N/A
Csp1 Csp2 Csp3 Csp4	CmpStatus parameters: – NetId – TestStatus – RefStatus – SMask	 Sv9:M Sv9:M Sv9:M Sv9:M	 _Yes _Yes _Yes _Yes	 _N/A _N/A _N/A _N/A
Scp1 Scp2	SelectDeltaStatus parameters: – NetId – StatusHandle	 Sva:M Sva:M	 _Yes _Yes	 _N/A _N/A

Scp3	– SMaskHandle	Sva:M	_Yes _N/A

3.3.1.6. NM API return codes

Note: There is no statement regarding CmpStatus and CmpConfig. The returned code (true/false) yields the result of comparison and should be considered as part of the respective procedure implementation.

Item	Service Feature	Status	Support
	Is E_OK return code supported by: (1)		
Eok1	– InitConfig	Sv1:O	_Yes _No _N/A
Eok2	– GetConfig	O	_Yes _No
Eok5	– StartNM	O	_Yes _No
Eok6	– StopNM	O	_Yes _No
Eok7	– GotoMode	Sv7:O	_Yes _No _N/A
Eok8	– GetStatus	Sv8:O	_Yes _No _N/A

(1) referred to as *Status* option in NM test plan

3.3.2. PIXIT

The following questionnaires intend to provide actual values for implementation-dependent parameters stated in the NM specification. They also ask for some test parameters required to run the test cases. The values supplied by the IUT designer will be picked up to parameterize the test suite. It is understood here that some work is needed before to adapt the test environment to the actual implementation formats of NMPDU fields and API parameters (size, range of values...). There is no statement relating to such information in the questionnaires.

3.3.2.1. Protocol parameters

- Network configuration

  To check the indirect NM protocol, the LT needs to simulate other network nodes. Therefore, the test user will be asked for two node addresses called MN1 and MN2 and during test suite execution the LT will send data messages with MN1 or MN2 as source addresses. In case of “one time-out per message” protocol version, the user must also specify the respective time-out values associated with MN1 and MN2 sources.

- Expiration window timers

  To check protocol timer implementation, a time window has to be defined where IUT outputs triggered by timer expiry can be accepted. For instance, to check an assertion such as "In NMLimphome state, application communication is enabled after T_{Error}", the test system will firstly wait for T_{Error} and verify that communication has not been enabled, secondly wait for the T_{Error} window expiration and verify that communication has been enabled.

  A time window is therefore defined for each protocol timer.

Item	Protocol parameter	Status	Support	Value
	Network configuration:			
Nc0	– Maximum number of nodes supported in network configuration	M	>= 2	
Nc1	– MN1	M	_Yes	
Nc2	– MN2	Nc0>2:M	_Yes _N/A	
	Other parameters:			
Pp1	– Max number of counter incr. for ON	Mt:M	_Yes _N/A	
Pp2	– Max number of counter incr. for MN1	Mt:M	_Yes _N/A	
Pp3	– Max number of counter incr. for MN2	Mt.Nc2:M	_Yes _N/A	
Pp4	– Max number of counter decr. for ON	Mt:M	_Yes _N/A	
Pp5	– Max number of counter decr. for MN1	Mt:M	_Yes _N/A	
	Protocol timers:			
Pt1	– Time-out for OBservation (TOB)	Gt:M	_Yes _N/A	
Pt2	– Terror	M	_Yes	
Pt3	– Twaitbussleep	Obs:M	_Yes _N/A	
Pt4	– TON (Time-out for node transmission)	Mt:M	_Yes _N/A	
Pt5	– TMN1 (Time-out for MN1)	Mt:M	_Yes _N/A	
Pt6	– TMN2 (Time-out for MN2)	Mt.Nc2:M	_Yes _N/A	
Pt7	– TMA (Time-out for all mute/absent)	Mt:M	_Yes _N/A	
	Expiration window timers:			
Wt1	– TOBW	Gt:M	_Yes _N/A	
Wt2	– TerrorW	M	_Yes	
Wt3	– TwaitbussleepW	Obs:M	_Yes _N/A	
Wt4	– TONW (transmission from own note)	Mt:M	_Yes _N/A	
Wt5	– TMNW (reception from remote nodes)	Mt:M	_Yes _N/A	

3.3.2.2. API parameters

- Indication of configuration change

  To check the functionnality of task activation or event setting on change of configuration, the Upper Tester must implement the associated tasks or events. The test user will be asked for the config handle and mask handle values processed by the IUT. And for each handle/mask association he must provide the list of nodes that must send an alive or ring message to generate an indication at the NM API.

- Indication of network status change

  To check the functionnality of task activation or event setting on change of network status, the Upper Tester must implement the associated tasks or events. The test user will be asked for the status handle and mask handle values processed by the IUT. And for each handle/mask association he must provide the list of necessary status changes to generate an indication at the NM API.

Item	Service parameter	Status	Support	Value
Nid	NetId	M	_Yes	
	API return status:			
Rs1	– E_OK	M	_Yes	

Rs2 Rs3	– TRUE – FALSE	Sv3|Sv9:M Sv3|Sv9M	_Yes _Yes	_N/A _N/A	
	Task identifiers for NM indications:				
Ti1 Ti2 Ti3	– normal configuration change – extended configuration change – network status change	Inct:M Iect:M Ist:M	_Yes _Yes _Yes	_N/A _N/A _N/A	
	Event masks for NM indications:				
Em1 Em2 Em3	– normal configuration change – extended configuration change – network status change	Ince:M Iece:M Ise:M	_Yes _Yes _Yes	_N/A _N/A _N/A	
	Handles for config change indication:				
Hc1 Hc2 Hc3	– table of config handles – table of associated mask handles – table of node lists	Inct|Ince| Iect|Iece: M	_Yes _Yes _Yes	_N/A _N/A _N/A	
	Handles for status change indication:				
Hs1 Hs2 Hs3	– table of status handles – table of associated mask handles – table of lists of status changes	Ist|Ise:M Ist|Ise:M Ist|Ise:M	_Yes _Yes _Yes	_N/A _N/A _N/A	

3.3.2.3. Test suite parameters

- Test execution timers

  The following timers are defined to manage the test execution:

  Tresp: this timer is started when the LT is waiting for an NMPDU or a TMPDU from the EUT. If it expires, the test will conclude that no response is forthcoming.

  Twait: this timer is started when the LT must wait for a certain amount of time before sending the next NMPDU or TMPDU. This can happen when the LT has to send two PDUs consecutively and the IUT needs to terminate the first action before being able or entitled to accept the second PDU. The latter is sent after Twait expiry.

Item	Test suite parameter	Status	Support	Value
	Test execution timers:			
Mt1 Mt2	– Tresp – Twait	M M	_Yes _Yes	

4. Test Management Protocol

4.1. Test scenarios

Figure 2 below describes the different communication scenarios between the UT and the LT. Dashed lines stand for messages that may be sent or not according to some status or configuration parameters.

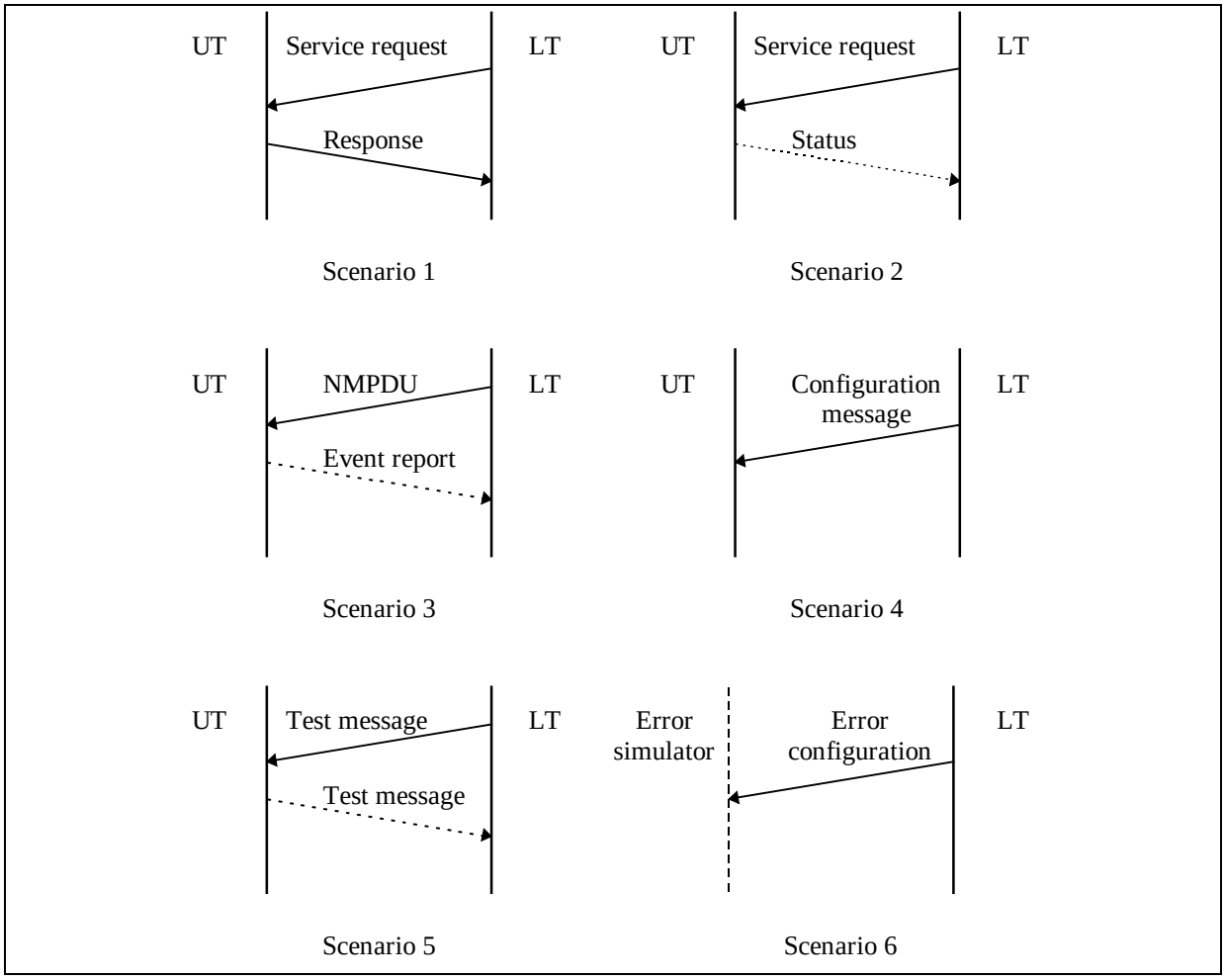

Figure 5 Test scenarios

Scenarios 1 and 2 are used to request the UT to call a service of the NM API. The *Service request* message conveys a service identifier and the associated parameters:

- Scenario 1 corresponds to GetConfig, CmpConfig, GetStatus and ReadRingData procedure calls. The result is the network config, the network status and the ring data respectively. It is sent back in the *Response* message.
- Scenario 2 corresponds to the other API calls. The only result is the API status. Depending on what is specified in the request message, it will be returned or not in the *Status* message.

In Scenario 3, the LT sends out an NMPDU which causes or not generation of an indication from the NM to the UT in order to inform of a configuration change, a status change or a ring

data reception. Depending on the UT configuration, the indication will be returned or not to the LT in the *Event report*.

Scenario 4 aims at configuring the UT behaviour. The *Configuration message* specifies which of the possible NM indications shall be returned to the LT.

Scenario 5 is used to verify whether or not user's communication has been enabled or disabled by the NM. On *Test message* reception, the UT shall try and send back the same message towards the LT.

Scenario 6 aims at configuring the network interface behaviour. The *Error configuration* message specifies which network perturbations are to be simulated.

4.2. Data Types

The test management protocol makes use of the following data types of the NM specification:

Data Types	Remark
NetIdType	Type for references to several communication networks
StatusType	Type of returned status information after a service call
ConfigKindName	Unique name defining the requested configuration. Legal names are: "Normal", "Normal_extended", "LimpHome".
ConfigHandleType	This data type represents a handle to reference values of the type ConfigRefType
NMModeName	Unique name defining the NM operational modes. Legal names are: "BusSleep" and "Awake"
NetworkStatusType	Type of Network Status
StatusHandleType	This data type represents a handle to reference values of the type StatusRefType
RingDataType	Type of the data field in the NMPDU

Table 1 Reused data types of NM specification

Data types specific to the test management protocol are defined below:

Name: **`TMPDUName`**

Description: Unique name defining the type of TMP message.

Values:

"apiCall":	to request the UT to call a procedure of the NM API
"apiStatus":	to carry out information returned by the last API call
"utConfig":	to configure UT's behaviour
"errorConfig":	to configure Network Interface's behaviour
"utEvent":	to report from NM indication through task activation or event signalling
"testMessage":	to request the UT to send back the test message

Name: **`DirNMAPIName`**

Description: Unique name defining the type of direct NM API.

Values: "initConfig", "getConfig", "cmpConfig", "selectConfig", "startNM", "stopNM", "gotoMode", "getStatus", "cmpStatus", "selectStatus", "silentNM", "talkNM", "readRingData" and "transmitRingData".

Name:	**StatusModeType**	
Description:	Unique name defining how the API return code will be dealt with.	
Values:	"never":	the return code is never returned to LT,
	"always":	the return code is always returned to LT,
	"ifError":	the return code is returned if different from E_OK,

Name:	**EventIdType**
Description:	This data type defines a mask for the NM indications being reported to the LT. NM indications correspond either to task activations or event signallings.
Values:	This data type includes one bit for each possible indication: One bit deals with NM signalling of Normal configuration change, One bit deals withNM signalling of Limphome configuration change (direct NM) or Extended configuration change (indirect NM), One bit deals with NM signalling of Network status change, One bit deals with NM signalling of Ring data reception (direct NM).

Name:	**DataProfileType**	
Description:	This data type describes a data profile.	
Values:	"allZero":	all bits of information are set to zero,
	"zeroOne":	bit setting is 0101..... ,
	"oneZero":	bit setting is 1010....,
	"allOne":	all bits of information are set to one,
	"badProfile"	none of the profiles above were received

Name:	**ConfigType**
Description:	This data type represents a network configuration.
Values:	Depends on NM implementation under test.

Name:	**NetErrorType**	
Description:	This data type specifies the network errors to be simulated.	
Values:	"noNetError":	no error simulation,
	"busBlocked":	simulation of bus blocked (e.g. CAN Bus Off),
	"noTransmission":	simulation of no Transmission (e.g. no frame acknowledgement at the data bus),
	"statusAfterOK"	to request the UT to get and save the network status,
	"bSleep":	to request Bus Sleep mode setting from the UT.

4.3. TMP messages from LT to UT

4.3.1. Common messages for Direct and Indirect NM

TMP messages are transmitted from LT to UT to request the UT to either:

- execute a service of the NM API,

- or configure UT's behaviour,
- or send a user message.

As long as the NM is not started, the network hardware inside the EUT is not initialised and the LT cannot communicate with the UT. It is therefore assumed that the UT will execute StartNM on its own. The CallStartNM message is only used to confirm that the NM should have been started. It can also request the status returned by StartNM.

Message Name: **`CallInitConfig`**

Scenario: 2 - Service request

Parameters:			
	TMPDUName	<pduCode>;	// "apiCall"
	DirNMAPIName	<dirNMAPI>;	// "initConfig"
	StatusModeType	<statusMode>;	
	NetIdType	<netId>;	

Purpose: This message requests the UT to execute "status = InitConfig(netId)".

Depending on both the returned status and the statusMode option, the UT will send back or not the status to the LT. Status transmission is done with the APIStatus message.

Message Name: **`CallGetConfig`**

Scenario: 1 - Service request

Parameters:			
	TMPDUName	<pduCode>;	// "apiCall"
	DirNMAPIName	<dirNMAPI>;	// "getConfig"
	NetIdType	<netId>;	
	ConfigKindName	<configKind>;	

Purpose: This message requests the UT to execute "status = GetConfig(netId, Config, configKind)" where Config refers to the local buffer containing the network configuration.

The UT shall then send back the configuration and the returned status to the LT using the NetConfigMsg message.

Message Name: **`CallCmpConfig`**

Scenario: 1 - Service request

Parameters:			
	TMPDUName	<pduCode>;	// "apiCall"
	DirNMAPIName	<dirNMAPI>;	// "cmpConfig"
	NetIdType	<netId>;	
	DataProfileType	<testConfig>;	
	DataProfileType	<refConfig>;	
	DataProfileType	<cMask>;	

Purpose: This message requests the UT to execute "result = CmpConfig(netId, TestConfig, RefConfig, CMask)" where TestConfig, RefConfig and CMask are initialised according to the data profiles specified by the corresponding parameters of the message.

The UT shall then return the result of CmpConfig to the LT using the APIStatus message.

Message Name: **CallSelConfig**

Scenario: 2 - Service request

Parameters:

TMPDUName	<pduCode>;	// "apiCall"
DirNMAPIName	<dirNMAPI>;	// "selectConfig"
StatusModeType	<statusMode>;	
NetIdType	<netId>;	
ConfigKindName	<configKind>;	
ConfigHandleType	<configHandle>;	
ConfigHandleType	<cMaskHandle>;	

Purpose: This message requests the UT to execute "result = SelectDeltaConfig(netId, configKind, configHandle, cMaskHandle)".

Depending on both the returned status and the statusMode option, the UT will send back or not the status to the LT. Status transmission is done with the APIStatus message.

Message Name: **CallStartNM**

Scenario: 2 - Service request

Parameters:

TMPDUName	<pduCode>;	// "apiCall"
DirNMAPIName	<dirNMAPI>;	// "startNM"
StatusModeType	<statusMode>;	
NetIdType	<netId>;	

Purpose: This message requests the UT to restart the NM and execute "status = StopNM(netId) ", then "status = StartNM(netId)".

Depending on both the returned status and the statusMode option, the UT will send back or not the status to the LT. Status transmission is done with the APIStatus message.

Message Name: **CallStopNM**

Scenario: 2 - Service request

Parameters:

TMPDUName	<pduCode>;	// "apiCall"
DirNMAPIName	<dirNMAPI>;	// "stopNM"
StatusModeType	<statusMode>;	
NetIdType	<netId>;	

Purpose: This message requests the UT to execute "status = StopNM(netId) ", then "status = StartNM(netId)" only if statusMode is set to "always".

The UT will send back the status returned by StopNM to the LT after execution of StartNM. Status transmission is done with the APIStatus message.

If statusMode is different from "always", nothing is done since the NM will be stopped and restarted on the next CallStartNM.

Message Name: **CallGotoMode**

Scenario: 2 - Service request

Parameters:

TMPDUName	<pduCode>;	// "apiCall"
DirNMAPIName	<dirNMAPI>;	// "gotoMode"
StatusModeType	<statusMode>;	
NetIdType	<netId>;	
NMModeName	<nmMode>;	

Purpose: This message requests the UT to execute "status = GotoMode(netId, nmMode)".

Depending on both the returned status and the statusMode option, the UT will send back or not the status to the LT. Status transmission is done with the APIStatus message.

Message Name: **CallGetStatus**

Scenario: 1 - Service request

Parameters:

TMPDUName	<pduCode>;	// "apiCall"
DirNMAPIName	<dirNMAPI>;	// "getStatus"
NetIdType	<netId>;	

Purpose: This message requests the UT to execute "status = GetStatus(netId, NetworkStatus)" where NetworkStatus is the network status returned by the API call.

The UT shall then send the network status and the API status to the LT using the NetStatusMsg message.

However, if the NM is in WaitBusSleep state, the network status is not sent immediately (since application communication is disabled). It will be returned on the next CallGetStatus. Network status is therefore not read again on that CallGetStatus.

The UT must also not call GetStatus and send a previously saved network status when CallGetStatus is received after TestMsg with netError set to "statusAfterOK" or "bSleep".

Message Name: **CallCmpStatus**

Scenario: 1 - Service request

Parameters:

TMPDUName	<pduCode>;	// "apiCall"
DirNMAPIName	<dirNMAPI>;	// "cmpStatus"
NetIdType	<netId>;	
NetworkStatusType	<testStatus>;	
NetworkStatusType	<refStatus>;	
NetworkStatusType	<sMask>;	

Purpose: This message requests the UT to execute "result = CmpStatus(netId, TestStatus, RefStatus, SMask)" where TestStatus, RefStatus, SMask are references to testStatus, refStatus and sMask parameters of the message.

The UT shall then return the result of CmpStatus to the LT using the APIStatus message.

Message Name: **CallSelStatus**

Scenario: 2 - Service request

Parameters:

TMPDUName	<pduCode>;	// "apiCall"
DirNMAPIName	<dirNMAPI>;	// "selectStatus"
StatusModeType	<statusMode>;	
NetIdType	<netId>;	
StatusHandleType	<statusHandle>;	
StatusHandleType	<sMaskHandle>;	

Purpose: This message requests the UT to execute "result = SelectDeltaStatus(netId, statusHandle, sMaskHandle)".

Depending on both the returned status and the statusMode option, the UT will send back or not the status to the LT. Status transmission is done with the APIStatus message.

Message Name: **CallConfigUT**

Scenario: 4 - Configuration message

Parameters:

TMPDUName	<pduCode>;	// "utConfig"
EventIdType	<eventId>;	

Purpose: This message defines which NM indications (task activation or event signalling) the UT shall report to the LT. Each bit of eventId specifies whether or not the corresponding indication is to be transmitted when occurring.

Default: no transmission.

NM indications are sent to the LT using the UTEvent message.

Message Name: **TestMsg**

Scenario: 5 - Test message

Parameters:

TMPDUName	<pduCode>;	// "testMessage"
NetErrorType	<netError>;	

Purpose: This message requests the UT to send back the same message to the LT.

The returned message will be received or not by the LT depending on the error type set in netError (see 4.4.2).

If netError is set to "statusAfterOK", no message is returned. The UT must execute "status = GetStatus(netId, NetworkStatus)". NetworkStatus value will be returned on the next CallGetStatus.

If netError is set to "bSleep", return of the message is delayed. The UT must execute the following sequence:
- GotoMode(BusSleep),
- wait TwaitBusSleep
- save the network status (GetStatus),
- GotoMode(Awake)
- send back TestMsg.

4.3.2. Specific messages of Direct NM

Additional TMP messages are transmitted from LT to UT to request the UT to execute specific services of the direct NM API.

Message Name: **`CallSilentNM`**

Scenario: 2 - Service request

Parameters:

TMPDUName	<pduCode>;	// "apiCall"
DirNMAPIName	<dirNMAPI>;	// "silentNM"
StatusModeType	<statusMode>;	
NetIdType	<netId>;	

Purpose: This message requests the UT to execute "status = SilentNM(netId)".

Depending on both the returned status and the statusMode option, the UT will send back or not the status to the LT. Status transmission is done with the APIStatus message.

Message Name: **`CallTalkNM`**

Scenario: 2 - Service request

Parameters:

TMPDUName	<pduCode>;	// "apiCall"
DirNMAPIName	<dirNMAPI>;	// "talkNM"
StatusModeType	<statusMode>;	
NetIdType	<netId>;	

Purpose: This message requests the UT to execute "status = TalkNM(netId)".

Depending on both the returned status and the statusMode option, the UT will send back or not the status to the LT. Status transmission is done with the APIStatus message.

Message Name: **`CallReadRD`**

Scenario: 1 - Service request

Parameters:

TMPDUName	<pduCode>;	// "apiCall"
DirNMAPIName	<dirNMAPI>;	// "readRingData"
NetIdType	<netId>;	

Purpose: This message requests the UT to execute "status = ReadRingData(netId, RingData)" where RingData represent the NMPDU data returned by the API call.

The UT shall then send the ring data and the API status to the LT using the RingDataMsg message.

Message Name: **`CallTransmitRD`**

Scenario: 2 - Service request

Parameters:

TMPDUName	<pduCode>;	// "apiCall"
DirNMAPIName	<dirNMAPI>;	// "transmitRingData"

	StatusModeType	<statusMode>;
	NetIdType	<netId>;
	DataProfileType	<dataProf>;

Purpose: This message requests the UT to execute "status = TransmitRingData(netId, RingData)" where RingData is initialised according to the data profile specified by the corresponding parameter of the message.

Depending on both the returned status and the statusMode option, the UT will send back or not the status to the LT. Status transmission is done with the APIStatus message.

4.3.3. Specific messages of Indirect NM

There is no specific API for indirect NM and therefore no specific command from LT to UT.

4.4. TMP messages from UT to LT

TMP messages are transmitted from UT to LT to inform the UT of the result of a service call or of an NM indication.

4.4.1. Common messages for Direct and Indirect NM

Message Name: **APIStatus**

Scenario: 1 - Response: after CallCmpConfig, CallCmpStatus
2 - Status: after CallInitConfig, CallSelConfig, CallStartNM, CallStopNM, CallGotoMode, CallSelStatus, CallSilentNM, CallTalkNM, CallTransmitRD,

Parameters:	TMPDUName	<pduCode>;	// "apiStatus"
	DirNMAPIName	<dirNMAPI>;	
	StatusType	<status>;	

Purpose: This message provides the LT with the status returned by the last service call. dirNMAPI defines the name of the service and can take any value except "getConfig", "getStatus" and "readRingData".

Message Name: **NetConfigMsg**

Scenario: 1 - Response: after CallGetConfig

Parameters:	TMPDUName	<pduCode>;	// "apiStatus"
	DirNMAPIName	<dirNMAPI>;	// "getConfig"
	StatusType	<status>;	
	ConfigType	<config>;	

Purpose: This message provides the LT with the network configuration and the API status returned by a GetConfig call.

Message Name:	**NetStatusMsg**		
Scenario:	1 - Response: after CallGetStatus		
Parameters:	TMPDUName	<pduCode>;	// "apiStatus"
	DirNMAPIName	<dirNMAPI>;	// "getStatus"
	StatusType	<status>;	
	NetworkStatusType	<networkStatus>;	
Purpose:	This message provides the LT with the network status and the API status returned by a GetStatus call.		

Message Name:	**UTEvent**		
Scenario:	3 - Indication		
Parameters:	TMPDUName	<pduCode>;	// "utEvent"
	EventIdType	<eventId>;	
Purpose:	This message informs the LT that an NM indication has just occurred. The corresponding information of eventId shall be set.		

Message Name:	**TestMsg**
Scenario:	5 - Test message
Parameters:	see TestMsg in § 4.3.1.
Purpose:	This message intends to check whether user's communication has been enabled or disabled by the NM. It is a copy of the test message received from the LT. It's not transmitted using the normal TMP message object but using the specific message object described in § 2.2.3. In Indirect NM, this message is also used by the NM to monitor the message transmission by the tested node.

4.4.2. Specific messages of Direct NM

Message Name:	**RingDataMsg**		
Scenario:	1 - Response: after CallReadRD		
Parameters:	TMPDUName	<pduCode>;	// "apiStatus"
	DirNMAPIName	<dirNMAPI>;	// "readRingData"
	StatusType	<status>;	
	DataProfileType	<dataProf>;	
Purpose:	This message provides the LT with the API status and the profile of the ring data returned by a ReadRingData call.		

4.4.3. Specific messages of Indirect NM

There is no specific message from LT to UT for Indirect NM.

4.5. TMP messages from LT to Network Interface

TMP messages are transmitted from LT to Network Interface to configure the network error simulation. They are common to direct and indirect NM.

Message Name: **`CallSetError`**

Parameters:	TMPDUName	<pduCode>;	// "errorConfig"
	NetErrorType	<netError>;	

Purpose: This message defines whether network errors shall be simulated or not, and if yes, it specifies the type of network error. This information is set by the "netError" parameter.

Default: no error simulation.

If bus blocked simulation is requested, the network interface shall behave so that a "bus blocked" indication is returned to the NM after each transmission attempt until simulation of this error is stopped by another CallSetError message with a different "netError" value.

If simulation of no message transmission is requested, the network interface shall behave so that a no transmission error is returned to the NM after each transmission attempt until error simulation is stopped by another CallSetError message with a different "netError" value.

The type of network error can also be configured with the TestMsg message (see 2.2.3).

5. Presentation of the NM test suites

The test suites for direct and indirect NM are specified in TTCN language [7]. The specification is supplied in Attachments 1 and 2 respectively.

To make the test cases independent of each other, the NM must be restarted before the test case execution. Therefore, the test cases start with CallStartNM and end with CallStopNM. They can be executed separately.

The NM test cases are derived from the test purposes of document [2 But the respective sequences of test cases and of test purposes are organised differently. The test purposes are listed according to the order of chapters and sections in the NM specification. On the contrary, the test cases are grouped in directories representing the main options of an implementation. Inside each directory, they are sequenced in a logical order to allow a progressive test of the associated functionnality.

Test case directories aim to represent respectively:

- the NM core functionnality that must be always implemented,
- the sleep mode,
- the passive mode (direct NM only),
- the API.

To facilitate cross-reference with the test plan, naming conventions have been defined. Test case names are derived from the location of the corresponding assertion in the test plan. Names consist of:

- a radix identifying the table of test assertion,
- the reference number of the assertion in the table. If the test case is linked to several assertions, the respective numbers are separated by "_". If several tests stem from the same assertion, the number is followed by a letter A, B, C...

Example: INI1_12 refers to assertions Nr 1 and 12 of the table "NM Init".

Attachment 1: Test suite for Direct NM

The correspondence between the test case names and the test plan is given in the following table:

Test case name	Test plan section
CONF... MOD... DATA...	Configuration management Operating modes and operating mode management Data field management
INI... NORM... LIMP... SLEEP...	NMInit and NMreset NMNormal NMLimpHome NMBusSleep

Table 2 Test case names of direct NM

Some test purposes are not referenced in the test suite. They are covered by test cases in other groups as described in the table below:

Test purpose	Covered by test case
CONF7	INI1B
CONF8	NORM7A
CONF9	INI1B
CONF11	CONF1_3A
MOD1	INI1_12
MOD4	INI1_12
MOD7	BSLEEP1
MOD8	INI9A
MOD22	MOD2
MOD27	NORM15A
MOD29	NORM15A
MOD31	NORM16

Test purposes regarding task activation or event setting on change of network status are not implemented in this version of the test suite (MOD24, MOD25, MOD26).

Encoding of network configurations:

In the test suite, network configurations are represented by a 6-bit field, e.g. '010110'B. Each bit provides with the status of a node in the ring configuration. From left to right, they stand for SN2, SN1, NodeId (own node), PN1, PN2, any other node. Bit value is 1 if the node is considered present and 0 if the node is considered absent.

Encoding of network status:

In the test suite, the network status is represented by a 9-bit field, e.g. '000010110'B. It is encoded according to the example given in the NM specification (Table 3 of [5]) Top to bottom bits in the table are encoded from right to left, i.e. the most right bit represents the information Present network configuration stable.

Encoding of NM/API events:

Events from NM/API are encoded in a 4 bit field, e.g. '0010'B, which represent from right to left:

- indication of normal configuration change,
- indication of limphome configuration change,
- indication of network status change,
- indication of ring data reception.

In CallConfigUT messages from LT to UT, bit value 1 means that the UT shall send an UTEvent message when the indication occurs. 0 means it shall not.

In UTEvent messages from UT to LT, bit value 1 means that the related indication has occurred. 0 means it has not.

Attachment 2: Test suite for Indirect NM

The Indirect NM specification includes two different protocol versions. Test cases which are common to both versions are grouped in directory BOTH. The others are specific to a given version, even if sometimes the test purpose is the same :

- the directory OGT contains the test cases for protocol "One global time-out",
- the directory OTM contains the test cases for protocol "One time-out per message".

Each directory in turn includes subdirectories defining respectively the test cases for the core functionnality, the limphome mode, the sleep mode (OTM only) and the API.

The correspondence between the test case names and the test plan is given in the following table:

Test case name	**Test plan section**
(directory BOTH) CONF... MOD...	 Configuration management Operating modes and operating mode management
(directory OGT) GINI... GNORM... GLIMP...	One global time-out TOB – Handling of StartNM and StopNM – NMNormal – NMLimpHome
(directory OTM) MINI... MNORM... MLIMP... MSLEEP... MCONF...	One time-out per message – Handling of StartNM, StopNM and InitConfig – NMNormal – NMLimpHome – NMBusSleep Configuration management

Table 3 Test case names of indirect NM

Some test purposes are not referenced in the test suite. They are covered by test cases in other groups as described in the table below:

Test purpose	**"One global time-out" Covered by test case:**	**"One time-out per message" Covered by test case:**
CONF6	GINI2	MINI2
CONF7	not applicable	MINI3
CONF8	GINI2	MINI2
CONF10	GCONF1_3	MCONF1_3
MOD4	GINI3_4	MINI5_6
MOD5	GINI3_4	MINI5_6

Test purposes regarding task activation or event setting on change of network status are not implemented in this version of the test suite (MOD7, MOD8, MOD9).

As the current NM specification does not provide an API for reading the extended network status, the related assertions are not tested:

- test purposes LIMP15, LIMP16, LIMP17 are not covered at all,
- test purposes INI6, NORM9, NORM10, LIMP12 are covered ony for the part reative to norma status.

Encoding of network configurations:

In the test suite, network configurations and extended network configurations are represented by a 3-bit field, e.g. '010'B. Each bit provides with the status of a node in the configuration. From left to right, they stand for MN2, MN1, NodeId (own node). Bit value is 1 if the node is considered (static) not mute or (static) present and 0 if the node is considered (static) mute or (static) absent.

Encoding of network status:

In the test suite, the network status is represented by a 5-bit field, e.g. '10110'B. It is encoded according to the example given in the NM specification (Table 8 of [5]). Top to bottom bits in the table are encoded from right to left, i.e. the most right bit represents the information No Error/Error, bus blocked.

Encoding of NM/API events:

Events from NM/API are encoded in a 4 bit field, e.g. '0010'B, which represent from right to left:

- indication of normal configuration change,
- indication of extended configuration change,
- indication of network status change,
- indication of extended network status change (not used).

In CallConfigUT messages from LT to UT, bit value 1 means that the UT shall send an UTEvent message when the indication occurs. 0 means it shall not.

In UTEvent messages from UT to LT, bit value 1 means that the related indication has occurred. 0 means it has not.