OSEK/VDX

Communication

Version 2.2.2

18th December 2000

Table of Contents

List of Figures

List of Tables

Index of services

Table of Abbreviations

– Interaction Layer –

API	Application Programming Interface
BS_Value	Block Size Value
CCC	Communication Conformance Class
DA	Dynamic Addressing
DM	Dynamic Message length
ECU	Electronic Control Unit
F_NORMAL	Normal addressing format
FIFO (queue)	First In First Out (queue)
I_CDM_RX_TO	Time Out of Communication Deadline Monitoring for Reception (periodical)
I_CDM_FRX_TO	Time Out of Communication Deadline Monitoring for First Reception (periodical)
I_CDM_TMD_TO	Time Out of Communication Deadline Monitoring- Direct Transmission Mode
I_CDM_TMM_TO	Time Out of Communication Deadline Monitoring- Mixed Transmission Mode
I_CDM_TMP_TO	Time Out of Communication Deadline Monitoring- Periodical Transmission Mode
I_n_FAILED_TMM	Number of FAILED Transmissions, Mixed Transmission Mode
I_n_FAILED_TMP	Number of FAILED Transmissions, Periodical Transmission Mode
I_TMM_TOF	Time Offset of Mixed Transmission Mode
I_TMM_TPD	Time Period of Mixed Transmission Mode
I_TMP_TOF	Time Offset of Periodical Transmission Mode
I_TMP_TPD	Time Period of Periodical Transmission Mode
NM	Network Management
RAM	Random Access Memory
SA	Static Addressing
SM	Static Message
ST_Value	Separation Time Value

– Network layer –

BS	Block Size
Consecutive Frame	Consecutive Frame Network Protocol Data Unit
CF_NPCI	Consecutive Frame Network Protocol Control Information
CF_NPDU	Consecutive Frame Network Protocol Data Unit
FC_NPCI	Flow Control Network Protocol Control Information
Flow Control Clear To Send	Flow Control Network Protocol Data Unit Clear To Send
Flow Control Wait	Flow Control Network Protocol Data Unit Wait
FC_NPDU_CTS	Flow Control Network Protocol Data Unit Clear To Send
FC_NPDU_WT	Flow Control Network Protocol Data Unit Wait
FC_NPDU	Flow Control Network Protocol Data Unit (either FC_NPDU_CTS or FC_NPDU_WT)
First Frame	First Frame Network Protocol Data Unit
FF_NPCI	First Frame Network Protocol Control Information
FF_NPDU	First Frame Network Protocol Data Unit
N_AI	Network Addressing field
N_Ar	Network time interval A on the receiver side
N_As	Network time interval A on the sender side
N_Br	Network time interval B on the receiver side
N_Bs	Network time interval B on the sender side
N_Cr	Network time interval B on the receiver side
N_Cs	Network time interval C on the sender side
NPCI	Network Protocol Control Information
NPDU	Network Protocol Data Unit
N_SA	Network Source Address
NSDU	Network Service Data Unit
N_TA	Network Target Address
N_USData.req	N_USData.request
N_USData.con	N_USData.confirmation
N_USData.ind	N_USData.indication
N_UUData.req	N_UUData.request
N_UUData.con	N_UUData.confirmation
N_UUData.ind	N_UUData.indication

N_WFTmax	Maximum number of *Flow Control Wait* transmission
Single Frame	Single Frame Network Protocol Data Unit
SF_NPCI	Single Frame Network Protocol Control Information
SF_NPDU	Single Frame Network Protocol Data Unit
STmin	Minimum separation time

– Data Link layer –

CAN	Controller Area Network
DLL	Data Link layer
DPDU	Data Link Protocol Data Unit
DSDU	Data Link Service Data Unit
D_UUData.req	D_UUData.request
D_UUData.con	D_UUData.confirmation
D_UUData.ind	D_UUData.indication

1 Introduction

The aim of OSEK communication (OSEK COM) is to agree on interfaces and protocols for in-vehicle communication. The term in-vehicle communication means both:

- communication between networked vehicle nodes (inter-ECU communication).
- communication within electronic control units (ECU-internal communication).

OSEK COM provides a standardized software communication interface that lowers the level of coupling between the application and the particular type of media used, hence facilitating portability of software across different communicating environments.

The definition of different conformance classes enables the integration of OSEK COM in systems featuring various levels of capabilities. The communication platform is scaleable to provide the required functionality.

Using the OSEK communication services for intertask communication presents the following advantages :

- This interface is independent of the bus system used hence allowing for application software asset re-use across different bus platforms.

- OSEK COM provides a clearly defined API communication interface : this enables provision of a largely re-usable communication kernel to support various types of applications. Thus, there is no need for the application programmer to redevelop the communication function.

- Portability of the application software is supported : if the OSEK communication services are used exclusively for inter-task communication, the application can be assigned to any ECU providing an OSEK compliant interface.

1.1 Requirements

The main requirements to be met by the OSEK communication specification are :

Portability of application software :

It is necessary that mechanisms to support inter-ECU communication are commonised with those that support ECU-internal communication in order to facilitate portability of application software across different configurations. Specific external communication services are defined to support applications that interface with the bus system under all circumstances (e.g. diagnostics).

Independence of underlying network and hardware :

The communication services and protocols are defined so that various bus systems and controllers can be accommodated with minimal configuration effort. In order to overcome the restrictions of the underlying network, regarding the maximum length of application messages, segmented as well as unsegmented communication need to be supported. The relationship between communication services and protocols need to be resolved in order to ensure stable services and effective protocol performance to support a wide range of applications.

Scalability :

Allowance must be made for the support of network nodes featuring different levels of functionalities and requiring different levels of functions.

Support for network management :

Provision is to be made such that OSEK network management communication is not in conflict with normal communication.

Interoperability of electronic equipment :

Provision of standardized communication protocols are to be made in order to allow for the interoperability of electronic equipment originating from various suppliers and connected to the same vehicle network.

1.2 Content and structure of the document

1.2.1 Communication protocol layers

This document introduces the communication requirements in two steps. Each protocol layer is introduced via an overview. The overview does not state any requirements. Instead, it aims at showing the scope of and introducing the concepts further developed in the communication layer that it introduces: this section may be enough for those who need a general understanding of the subject matter. This section does not attempt to demonstrate how the functions are performed in detail.

A requirement specification follows each overview: this section states the detailed functional and non-functional requirements of the specific protocol layer. This section is mainly addressed to OSEK implementers and those who will specify requirements for communicating applications.

1.2.2 System generation requirements

Requirements defining what can and what needs to be defined at system generation time are gathered in a unique section of the specification. These requirements are presented following a systematic approach (entity, entity attributes, and entity relationships). This section feeds the requirements for a portable system generation scheme that is further standardised by the OIL (OSEK Implementation Language) working group.

1.2.3 Communication conformance class

Requirements for the provision of a given set of different functions defined in this specification are presented at the end of the document.

1.2.4 Changes between OSEK COM 2.1 r1 and this specification

The previous specification has been revised against OSEK technical committee remarks. As a whole, efforts have been focused to improve the cohesion of the protocol layers (overview / specification section). In addition, the provisional specification of the network layer has been updated reflecting the results of the harmonisation process between OSEK and ISO (TC22/SC3/WG1/ TF2 Diagnostics on CAN).

The API section has been reviewed and extended with two services to support the transmission of variable length messages. In addition, “flag” mechanisms have been provided two interface services to access them in a portable way. Initialisation services have been repartionned to ease their interface with an OSEK operating system.

The communication conformance class has been revisited in order to allow for a smaller certifiable specification sub-set that support internal communication only. Given the demand to authorise queued message usage within small specification subset ; the queued message is now an optional features that can be provided if required.

Finally, system generation requirements have been developed to feed the OIL initiative.

1.3 Communication concept

The general positioning of OSEK COM within the OSEK architecture is represented in the figure below. It shows the positioning of OSEK COM with respect to the application, the OSEK network management, the underlying OSEK operating system (if used), and with the hardware communication bus.

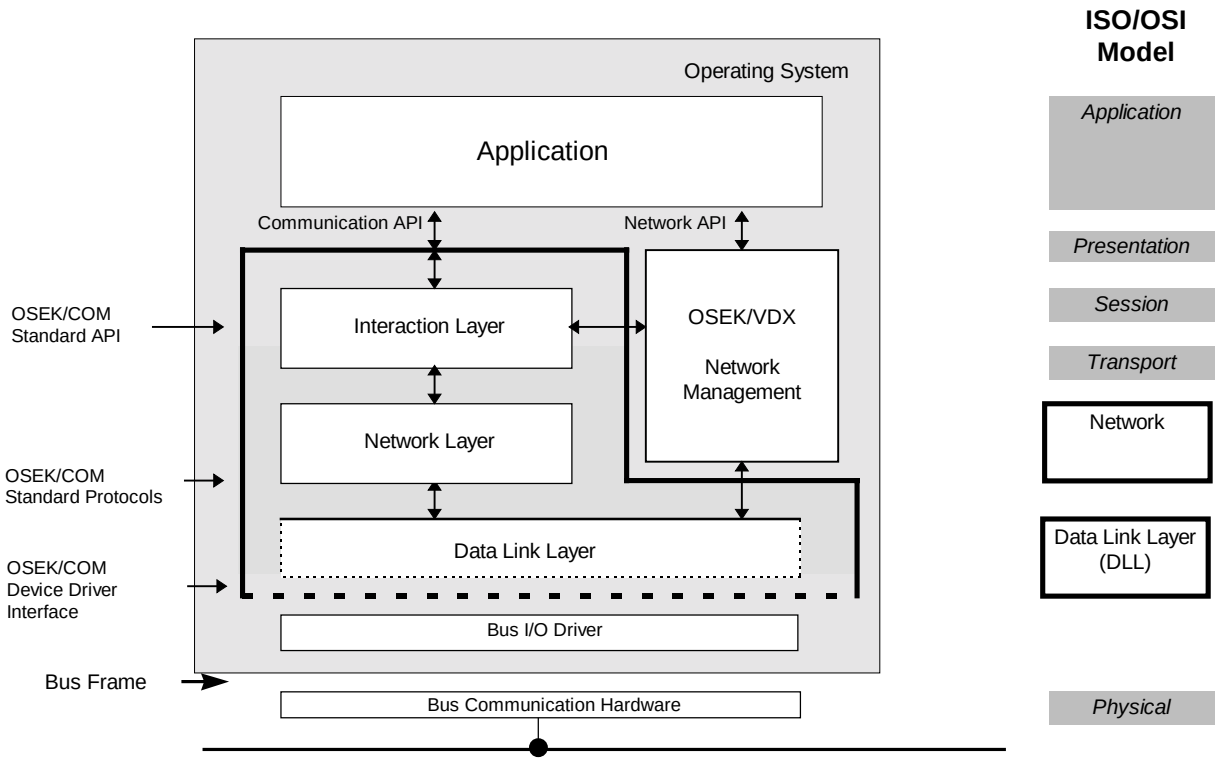

Figure 1-1: Layer model of OSEK COM

OSEK COM is based on the layer model[1] shown above, consisting of the Interaction Layer (INL), Network Layer (NWL) and the opened Data Link Layer (DLL). The physical layer is integrated into the communication controller's hardware and is not covered by the OSEK COM specification.

[1] The layer model is provided herein as a conceptual model that does not imply a particular implementation.

Presentation layer

This OSEK COM specification does not define a presentation layer. Data representation (big-endian, little-endian, etc.) has to be defined and maintained by the application designer in order to have a consistent and uniform data representation across the distributed architecture.

Interaction layer

The interaction layer provides the application program interface (API) of OSEK COM. It provides communication services for the transfer of application messages. For network communication, the interaction layer uses the services provided by the lower layers. ECU-internal communication is handled directly by the interaction layer without the involvement of underlying layers.

Network layer

The network layer provides services for the unacknowledged and segmented transfer of application messages. The network layer provides flow control mechanisms to enable interfacing of communication peers featuring different level of performance and capabilities. The network layer uses services provided by the Data Link Layer.

Data link layer interface

The data link layer interface provides services for the unacknowledged transfer of individual data packets over a network to the layers above. The size of these data packets (frames) depends on the underlying network and is not specified by OSEK COM. Additionally, the data link layer provides services for the network management module (e.g. configuration, initialisation, status request).

OSEK COM provides a rich set of communication facilities but it is likely that many applications will only require a subset of this functionality. In order to optimise the use of OSEK COM for particular applications there are four distinct conformance classes defined with increasing levels of functionality. The designer selects the conformance class used for an application at system generation time in order to optimise resource usage by excluding unnecessary features.

To differentiate the OSEK COM conformance classes from the OSEK OS Kernel conformance classes they are referred to as Communication Conformance Classes, or CCCs, with CCCA containing only the basic functionality and CCC2 being a full implementation. Application source code written for a lower conformance class is compatible with a higher one, but the reverse would not normally be true. The actual level of resource usage optimisation outside of the conformance classes is implementation specific.

2 Interaction layer

This chapter states the services and the functionality requirements of the interaction layer.

Requirements for the provision of implemented functionality of the interaction layer are stated in the communication conformance class section of the OSEK COM specification.

2.1 Interaction layer overview

2.1.1 Interaction layer operation

The main purpose of the interaction layer is to communicate data between application activities by sending messages. An activity can be either the sender or receiver of a particular message. Communication may be either internal or external.

It is necessary to have a clearly defined programming interface for communications to allow application portability and easy maintenance.

The interaction layer must ensure data integrity. To achieve this it is necessary to manage resource access to prevent conflicts. The interaction layer is also responsible for routing communications to the appropriate destination, be that internal or external to the originating processor.

Robust application design requires that the interaction layer also returns information allowing the progress of a message to be tracked.

2.1.2 Communication model

OSEK COM implements an asynchronous programming model. Communication functions are executed concurrently with the application, which allows data transmission to continue without blocking other processing. A call to a message send function will therefore return immediately after initiating a corresponding data transfer in the underlying communication layers. This implies that a service that sends a message is unable to return final transmission status because the transfer to the network may still be in progress. Therefore OSEK COM must provide a means for an application to detect completion and determine its status. This status can be communicated to the application by means of notification mechanisms defined in this specification.

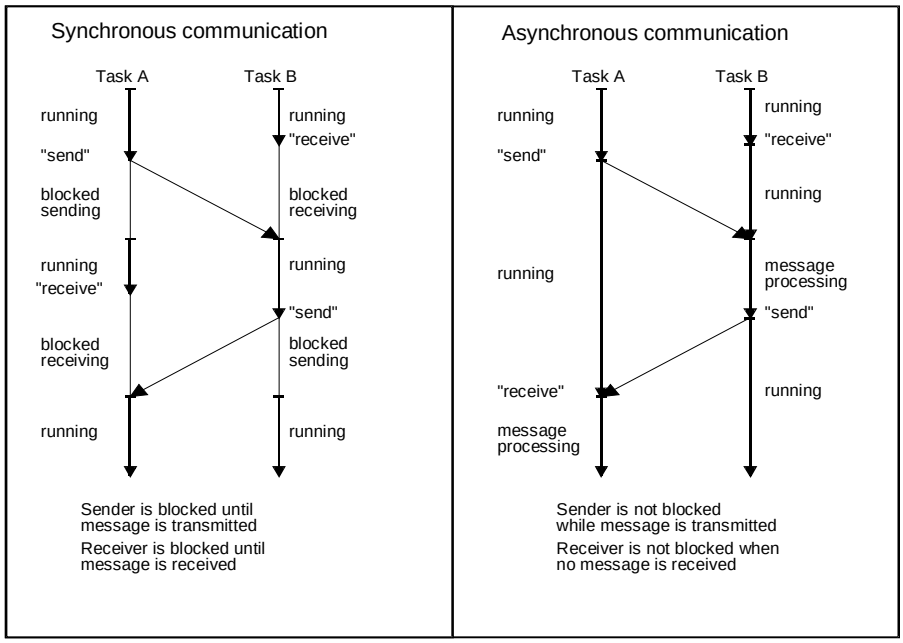

Figure 2-1: Synchronous vs. asynchronous communication schemes

2.1.3 Message concept

OSEK COM exchanges data using messages. A message is a container for application specific data, the format and use of which is not relevant to COM itself.

Conceptually, a message may only have a single sender in a system, but it may have any number of receivers. Where there is a single sender and a single receiver the messages is described as 1:1. A message with multiple receivers is described as 1:*n*.

A message type must be defined at system generation time; messages cannot be added or deleted at run-time.

2.1.4 Queued and Unqueued messages

To accommodate different application requirements, a particular message can be defined as either queued or unqueued. COM stores queued messages in a FIFO buffer and they are read by the application in the order they arrived. A particular queued message can therefore only be read once, as the read operation removes it from the queue.

By contrast, an unqueued message behaves more like a global variable and is not consumed by reading. An unqueued message returns the last received value each time it is read.

A queued message would normally be used to track changes of state within a system, where it is important that receiver maintains synchronisation of state information with the sender. It requires the receiver to service the queue on a regular basis to prevent an overflow.

Unqueued messages are appropriate for the transmission of a current value, where this value may be read multiple times by the receiver as required, or may even be ignored.

2.1.5 Messages copies

For normal message transmission, OSEK COM provides each sender and receiver with its own copy of a particular message. This allows each activity to manipulate the message data without interfering with other users and is described as a WithCopy access. Data consistency is guaranteed and a uniform data model is provided.

In order to optimise RAM usage it is also possible to access a message buffer using a WithoutCopy method. In WithoutCopy mode both the sender and any receivers may share the same physical message buffer. By using a WithoutCopy transfer OSEK COM cannot guarantee data consistency without the use of a special mechanism to prevent simultaneous access to the message buffer. Therefore syntax of a WithoutCopy access violates the uniform data model. However, if it is possible to determine during the application design phase that no conflicting message buffer accesses can occur then message protection is unnecessary and the uniform data model is preserved. Whether an access is with or without copy is determined for each message at system generation time on a per-task basis.

2.1.6 Direct and periodical transmission modes

There are two distinct modes for the transmission of an external message. Physical transmission either occurs as soon as the application sends a message or the message can be transmitted repeatedly at a pre-set interval. These transmission modes are called direct and periodical respectively. A message can be defined as having both direct and periodical transmission – this is referred to as a mixed transmission mode.

The transmission mode for a particular message is set at system generation time and cannot be altered at run-time.

A periodical message would typically be used for information that is not time sensitive. It also provides a ‘heart-beat’ for an ECU, which allows other ECUs to determine that it is functioning correctly. Periodical transmission implies that a message’s contents can be altered more than once between transmissions and these intermediate values will never be sent to the receiver. This can be useful for reducing loading on the transmission medium when updates are not time-critical.

Direct transmission is appropriate when the message’s application data must be sent quickly whenever an update occurs. For rarely updated messages the use of direct transmission can also minimise loading on the physical transmission medium, but there is a danger that a message may be lost through a system error or restart of the receiving ECU. In such circumstances a mixed mode transmission is desirable as it implies that the sender and receiver will become re-synchronised within a finite period of time.

Mixed transmission mode is further enhanced with a ‘relevant change’ mechanism. This can cause additional immediate transmissions of a message whenever the application data satisfies pre-defined conditions. Such behaviour might be useful when a parameter moves outside of a normal operating range and the receiver needs more immediate updating than the default periodical transmission interval allows.

2.1.7 Message addressing

The destination address of a particular message is either static or dynamic.

A statically addressed 1:1 message has a single destination defined at system generation time. This cannot be altered at run-time. A dynamically addressed 1:1 message must have a list of

possible destinations defined at system generation. Conceptually, the application can then select one, and only one, of these pre-defined addresses whenever the message is sent.

A statically addressed 1:*n* message has a list of recipients defined at system generation, each of which will receive the message whenever it is sent. There is no specific API to cope with dynamically addressed 1:*n* messages but they can be implemented at the application level by sending the same message to several dynamically selected addresses.

By using the API for sending a message with a dynamic address it is possibly for an application to circumvent the 1:1 and 1:*n* models in OSEK COM. This facility may be desirable when dynamic or end-of-line configuration is required for an ECU but it should be enforced wherever possibly when using system-wide CASE (Computer Aided Software Engineering) tools.

2.1.8 Message lengths

To accommodate different application requirements, a message can be defined with either static or dynamic message lengths. For normal message transmission OSEK messages are defined with a static message length at system generation. In some applications there may be a need to vary the length of the data in a message dynamically at run-time. The maximum length of a dynamic-length message must be defined at system generation.

2.1.9 Application programming interface

OSEK provides a clearly defined Application-Programming Interface (API) for initialisation, communication and controlling the communication. These API functions can be logically grouped into three types: initialisation, sending/receiving, and control.

2.1.9.1 Initialisation

- InitCOM and CloseCOM:

  These service are used to initialise and release the platform specific communication resources.

- StartCOM:

  This service initialises internal COM data areas, calls message initialisation routines and starts the OSEK communication module.

- StopCOM:

  This is used to terminate a session of OSEK COM, release resources where applicable.

- StopPeriodical and StartPeriodical :

  These services start or stop the periodical transmission of all messages using the periodical or the mixed transmission mode. It is sometimes useful to suspend periodical activity without necessarily closing down the whole of COM.

2.1.9.2 Sending/Receiving

- SendMessage and ReceiveMessage:

  These are used for sending and receiving normal messages with a Static Address (SA) and a Static Message length (SM).

- SendDynamicMessage and ReceiveDynamicMessage:

  Where a Dynamic Message length message (DM) is used with an SA message these alternative services are used in order to pass the actual length of the message.

- SendMessageTo and ReceiveMessageFrom:

  This last pair of services are used for DM messages with a Dynamic Address (DA).

2.1.9.3 Control

- GetMessageResource and ReleaseMessageResource :

  As described in the *With* and *WithoutCopy message* section, these services provide a locking mechanism to prevent conflicting access to a shared message buffer.

- GetMessageStatus:

  This service retrieves the status information for a message.

- ChangeProtocolParameter:

  This service modifies network layer parameters for a specific message.

2.1.10 Notifications

OSEK COM implements an asynchronous communication model. Therefore OSEK provides a notification mechanism to inform the application about the final transmission status of a previously invoked send or receive service.

2.1.11 Deadline monitoring

OSEK COM provides a mechanism for monitoring the transmission and reception timing of messages called communication deadline monitoring. This can be used to:

- verify on the sender side that transmission requests (periodical or not) are followed by transmissions on the network within a given period of time; and,
- verify on the receiver side that periodical messages are received within an allowed time period.

2.1.12 Portability support

OSEK COM implements a uniform data model, which means that the behaviour of tasks and functions appears to be that of a sequential single-threaded system. This enables easier software testing and the re-hosting of application software since possible interruption points need not be taken into consideration during the programming of a task or function.

The implementation of the uniform data model requires abstract interface operations to guarantee data consistency. In pre-emptive systems, local copies of message data might be needed to ensure data consistency when tasks pre-empt each other.

OSEK COM allows the user to configure where a message is physically buffered for every message and every task separately. A message may be buffered in shared memory or a task may receive its exclusive copy. This approach provides to the system designer the full control over and the most flexibility in the usage of the scarcest resource - RAM.

Portability is achieved by means of using only runtime services provided by OSEK COM to access message objects. Any other method of accessing a message or the lower network layers is out of the scope of and thus not supported by OSEK COM.

2.2 Interaction layer specification

2.2.1 Definitions

Application layer: identifies layer 7 of the OSI basic reference model. This layer is out of the scope of the OSEK COM specification.

Activity : identifies an OSEK task, function, ISR or Callback.

Application receiver: identifies an activity (e.g. OSEK task) of the *application layer* that uses the OSEK communication receiving services.

Application sender: identifies an activity (e.g. OSEK task) of the *application layer* that uses by the OSEK communication sending services.

Application receiver address: identifies one or multiple application receiver(s). The definition of the address format (semantic) is application specific.

Application sender address: identifies a particular application sender. The definition of the address format (semantic) is application specific.

Internal communication: identifies transfer of information taking place between *application senders* and *application receivers* and supported by OSEK COM services provided by the Interaction layer only (e.g. transfer of information between two activities on the same processor).

External communication: identifies transfer of information supported by OSEK COM communication services taking place between *application senders* and remote *application receivers* and using at least one communication physical layer that corresponds to layer one of the OSI basic reference model (e.g. transfer of information between two activities located in two different ECU).. External communication needs to support at least one communication physical layer that corresponds to layer one of the OSI basic reference model.

Internal application receiver: identifies an *application receiver* that communicates with the *application sender* through *internal communication.*

External application receiver: identifies an *application receiver* that communicates with the *application sender* through *external communication.*

Internal-External communication: identifies transfer of information from one *application sender* to multiple *application receivers* using both internal communication and external communication services.

2.2.2 Initialisation and shutdown

There are three pairs of services provided by OSEK COM for starting and stopping its activities :

1. InitCOM and CloseCOM are used for low-level initialisation of hardware;
2. StartCOM and StopCOM control resource usage once the OSEK OS kernel has started
3. StartPeriodical/StopPeriodical enable and disable periodical message transmission.

Each service is designed for use at particular times within an application.

InitCOM shall be called as part of the OSEK OS start-up in order to prepare hardware, interrupt tables and other low-level features. It may be called from within a hook routine (i.e. StartupHook), directly by the kernel or before the kernel itself is started. InitCOM assumes that the processor interrupts are masked to allow it to configure interrupt-generating hardware. InitCOM shall not alter the interrupt mask itself. InitCOM may be called at any time therefore it shall not make use of any OSEK/VDX system services. InitCOM shall be called before StartCOM or any other COM service is called.

Once the kernel has started an application may call StartCOM. This service is intended to allocate and initialise system resources used by the COM module. StartCOM is free to use kernel functions required (and may use these to alter interrupt settings if required). Until StartCOM has been called, no COM service apart from InitCOM and CloseCOM shall be called.

StartPeriodical and StopPeriodical shall be used to control the transmission of periodical or mixed-mode messages.

StopCOM is designed to allow an application to cease using the COM module in order to free up its resources. COM may subsequently be re-started using the StartCOM service. StopCOM may use any kernel or COM service, and would normally be called at task level. StopCOM will not allow message corruption, although incoming messages unread by an application will be inaccessible and thus lost.

CloseCOM is the opposite service to InitCOM. CloseCOM would normally be called as the OSEK OS is shutting down to return the hardware to a safe state. CloseCOM may be called from an OSEK OS hook routine (i.e. ShutdownHook). CloseCOM is unable to make any use of COM or kernel services, and shall not modify the system interrupt mask. Any data currently being transmitted or due for transmission when CloseCOM is called may be lost or corrupted unless StopCOM has previously been called to perform a graceful shutdown. CloseCOM shall leave the hardware in a suitable state for re-initialisation by InitCOM.

The following picture illustrates the sequence of operation to respectively initialise and shutdown COM associated with an OSEK operating system. The picture assumes that StartOS returns as an example ; StartOS does not necessarily return since its behaviour is implementation specific. Refer to the OSEK OS specification to peruse requirements applicable to mentioned hook routines.

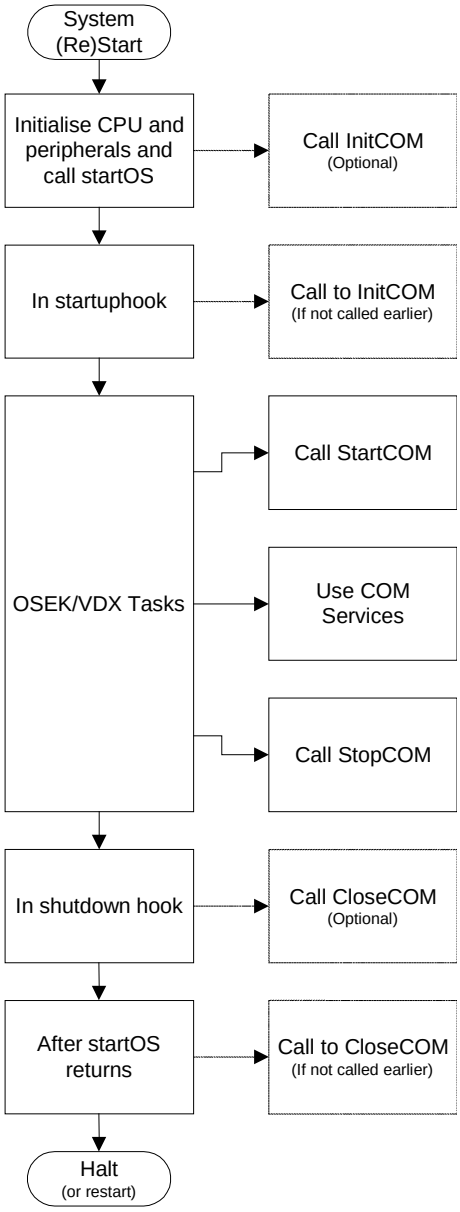

Figure 2-2: OSEK COM initialisation and shutdown services

2.2.3 Communication model

Communication services provide the means to transfer information between application senders and receivers.

Communication can take place between one application sender and one application receiver, referred to as 1:1, and one application sender to multiple application receivers, referred to as 1:*n*.

Internal communications can take place as 1:1 or 1:*n*. For external communication 1:1 communication is always possible, and 1:*n* is possible except in the case of USDT (Unacknowledged Segmented Data Transfer) multiple frame messages. External communication is supported by transferring information from one application sender to multiple application receivers (1:*n*) if UUDT (Unacknowledged Unsegmented Data Transfer) or USDT single frame messages are used for the transfer (see network layer chapter).

External communication is restricted to the transfer of information from one application sender to one application receiver (1:1) if USDT multiple frame messages are used for the transfer. (see network layer chapter).

2.2.4 Messages

2.2.4.1 Definition

A message is a data structure used to transfer information between application activities.

An OSEK message (also referred to as "message") identifies a generic item, handled by the interaction layer and used by the application layer to support the transfer of application information. OSEK COM has no knowledge of a message purpose, structure and content ; a message serves as a transportation unit that interacts with the application layer by means of OSEK application programming interface (API) communication services.

Two categories of messages are defined:

1. OSEK static length message: this category identifies a message whose length is set at system generation time. The length of such message cannot be changed at run-time.
2. OSEK dynamic length message: this category identifies a message whose length is set at run time. The maximum length is set at system generation time and cannot be changed at run-time.

Two types of messages are defined to support internal, external and internal-external communication:

1. Unqueued message.
2. Queued message.

2.2.4.1.1 Unqueued message

An unqueued message is a message (static or dynamic length message) that shall be overwritten whenever a new message data arrives.

The message shall not be consumed by any receive API service (e.g. ReceiveMessage).

Unqueued message can be defined on both the application receiver and sender sides, and used for internal, external, and internal-external communication.

Unqueued message can be transferred using any of the provided network layer protocols. (i.e. USDT, USDT).

2.2.4.1.2 Queued Message

A queued message is a static length message whose internal data structure is organised as a First-In/First-Out (FIFO) queue. Messages shall be delivered in the same order as they arrived. Queued messages are consumed by OSEK receive service (i.e. ReceiveMessage). The receive communication service shall be provided the oldest message of the queue to be returned to the application receiver.

If the queued message is full and a new message data arrives, this message data will be lost.

Queued messages are always accessed via message copies. (see message copy configuration chapter)

The queue (organised as a FIFO data structure) shall be implemented on the application receiver side only.

For internal communication, the message shall be of the type "Queued" for both the sender and receiver. For external and internal-external communication, it is not required that a message be accessed as a ‘queued’ message by the external receivers, the external receivers can process the message as an ‘unqueued’ message.

Notice that only one internal application receiver is permitted to access a specific queued message. This limitation results from the costly implementation model the support for multiple internal application receiver would require ; hence any queued message declared on the receiving end is accessible by a single internal application receiver only. A single FIFO data structure is therefore needed to be managed per receiving end.

The queue size for a particular message is specified on a per receiver-basis. The queue size value depends on system design constraints, e.g. message inter-arrival rate and reading rate.

No implementation requirements are specified by the OSEK COM specification to allow for various optimisation of implementation-specific detailed software architecture.

Queued messages can support internal , external and internal-external communication.

2.2.4.2 Message object

A message object identifies an implementation unit of a message. A message object encapsulates application data and message status information that shall be processed by the OSEK COM kernel as documented in this specification.

The detailed implementation format of a message object is implementation specific hence is not described within the OSEK COM specification.

2.2.4.3 Message accessor

A message accessor identifies a data structure that shall be used by the application to handle message data.

A message accessor shall be associated with each message handled by each application receiver and sender.

The type and length shall be compatible with the associated message.

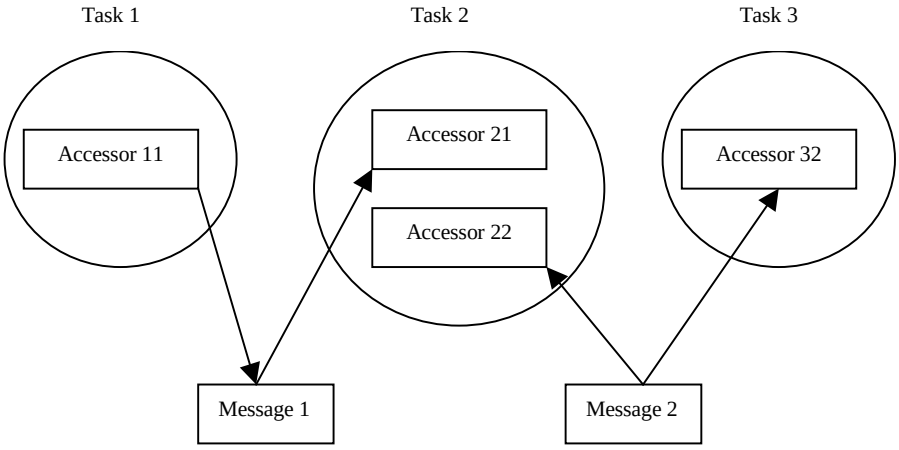

Figure 2-3: Message accessor

A message accessor shall be uniquely identified by means of an AccessName. There are no naming requirements for the AccessName.

2.2.4.4 Message copy configurations

Two configurations are supported by OSEK COM to send or receive a message : WithCopy or WithoutCopy. The WithCopy configuration requires the provision of a physical copy of the message data associated with the message accessor. The WithoutCopy requires no copy of the message data.

"WithCopy" :

> A message accessor shall be mapped onto a message copy that shall be used by the application receiver and sender. This local copy is provided for OSEK COM to ensure message consistency. The memory allocation strategy for message copies is implementation specific.

"WithoutCopy" :

> A message accessor shall be mapped onto a message object directly

The definition of message copy configuration shall be performed at system generation time.

2.2.5 Addressing schemes

Two addressing schemes are defined to support the exchange of OSEK static length and dynamic length messages: static addressing and dynamic addressing.

Assignment and definition of the addressing scheme applicable to a specific message shall be defined at system generation time.

2.2.5.1 Static addressing scheme

The static addressing scheme (SA) mandates the definition of application sender at system generation time and application receivers for a specific message.

A message shall be assigned to a unique set of sub-network frame addressing attributes (e.g. ISO 15765-2, F_NORMAL addressing format : CAN identifier).

Messages using the static addressing scheme shall be handled by means of the following communication API services :

1. SendMessage
2. ReceiveMessage
3. SendDynamicMessage
4. ReceiveDynamicMessage

2.2.5.2 Dynamic addressing scheme

On the application sender side, the addressing information (application receiver address) that identifies one or multiple application receiver(s) is selected by the application sender at run-time and shall be sent out with the message using the SendMessageTo communication service.

On the application receiver side, the addressing information that identifies the application sender shall be provided to the application by means of the ReceiveMessageFrom service.

Messages using the dynamic addressing scheme (DA) shall be handled by means of the following communication API services :

1. SendMessageTo
2. ReceiveMessageFrom

2.2.6 Data consistency model

In the WithCopy configuration, activities are provided with their own message copies, and can thus manipulate the message data (through an accessor) without interfering. Nevertheless, calls to API services may require exclusive access to the message object or to other internal resources. If an OSEK COM implementation can not ensure that such an access is safe, the service shall perform no-operation and return with special status code E_COM_LOCKED (refer to API description) : the message object is said to be *locked* with respect to that particular service. The application may then re-issue the request later if needed.

In the WithoutCopy configuration, several activities may share the same physical buffer (i.e. the message object) and OSEK COM can not ensure data integrity. The application itself is therefore responsible for avoiding conflicting access to a particular message object. OSEK COM provides a special mechanism that can help with that objective : message objects include a BUSY flag that can be tested and set, or cleared via the two dedicated API services GetMessageResource, ReleaseMessageResource. The interaction layer will not access a protected message object (BUSY flag set), unless explicitly requested via calls to API services. Data consistency is ensured if the application is designed so that activities always protect messages they need to access (including access via calls to API services).

For message objects that are used both in WithCopy and WithoutCopy configuration, transmission or reception services called from an activity using WithCopy configuration shall also test the protection state : if BUSY flag is set, the service shall perform no-operation and return E_COM_LOCKED ; the integrity of the message object is therefore preserved. Note that in either message configuration cases (ie WithCopy and WithoutCopy), the conditions for a message to be *locked* are implementation specific since these depends on the strategy to implement message objects and how these are accessed.

2.2.7 Message transmission

2.2.7.1 Transmission using WithCopy message accessor configuration

The provision of a message copy affects the operation of the interface interaction layer - application layer. The provision of a copy located within the application layer requires the transfer of information from the message copy to the interaction layer's message object.

The information encapsulated within a message copy shall be transferred to the appropriate message object upon call to a specific transmission API service.

This transfer of information between the application and interaction layers shall be applicable for internal, external and internal-external communication using any of the applicable transmission modes : i.e. Direct, Periodical, Mixed.

2.2.7.2 Transmission using WithoutCopy message accessor configuration

No transfer of message data can take place between the interaction and application layers.

An update of the message object shall be performed each time a message accessor associated with a specific message is assigned a particular value.

2.2.7.3 Transmission modes

OSEK COM supports three transmission modes allowing different schemes to transmit messages : Direct, Periodical and Mixed.

The Direct transmission mode supports internal, external and internal-external communication.

The Periodical and Mixed transmission modes supports external and internal-external communication.

Assignment and definition of the transmission mode applicable to a message shall be performed at system generation time.

2.2.7.3.1 Direct transmission mode

The application is in charge of requesting each transmission of a message to the interaction layer, using the SendMessage, SendDynamicMessage or SendMessageTo API service. In the case of external or internal-external communication, it shall be followed by a transmission request from the interaction layer to the network layer. Refer to network layer chapter for the definition of the sequencing of network layer services.

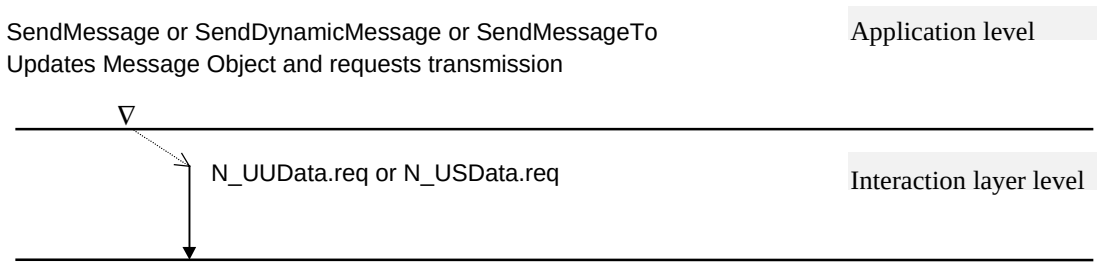

Figure 2-4: Direct transmission mode for external or internal-external communication (WithCopy configuration)

The Direct Transmission mode can be used with both UUDT and USDT protocols, with Static or Dynamic Length configuration and Static or Dynamic Addressing Scheme.

2.2.7.3.2 Periodical transmission mode

The interaction layer shall be able to issue periodical transmission requests of a message.

Each call to the API service SendMessage shall update the message object with the application data to be transmitted. The SendMessage service shall not issue any transmission request to the network layer.

The transmission is performed on a cyclic time basis by means of calling the N_UUData.request service of the network layer upon expiration of the periodical transmission mode time period (I_TMP_TPD).

If periodical message transmission is not possible due to message object being locked or BUSY then the transmission shall take place as soon as possible after it is released. Postponement of a periodical message transmission shall not affect the transmission time of subsequent messages.

The periodical transmission mode time period (I_TMP_TPD) shall be defined at system generation time on a per-message basis.

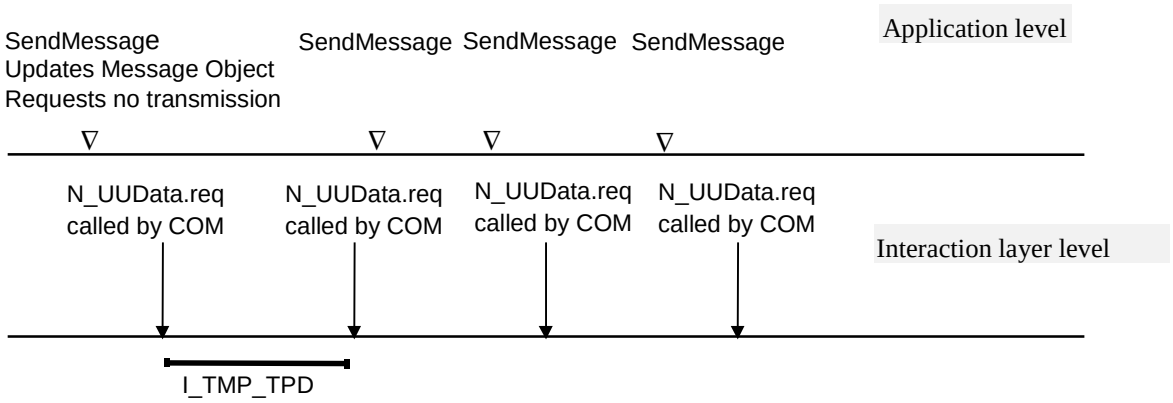

Figure 2-5: Periodical transmission mode

The Periodical Transmission Mode applies to the UUDT protocol only (Static Length, Static Addressing Scheme).

2.2.7.3.2.1 Activation / Deactivation of periodical transmission mechanism

This transmission mechanism shall be activated by a call to the StartPeriodical() API service. The StartPeriodical service shall start the periodical transmission mode time offset timer (I_TMP_TOF).

The first transmission request (N_UUData.request service) shall be issued upon expiration of the periodical transmission mode time offset (I_TMP_TOF).

StartPeriodical shall be called after the StartCOM API service has completed and once the message object is correctly initialised. The API service MessageInit() can be used to perform this initialisation.

The periodical transmission mechanism shall be stopped by means of the StopPeriodical API service.

The periodical transmission mode time offset (I_TMP_TOF) shall be defined at system generation time on a per-message basis.

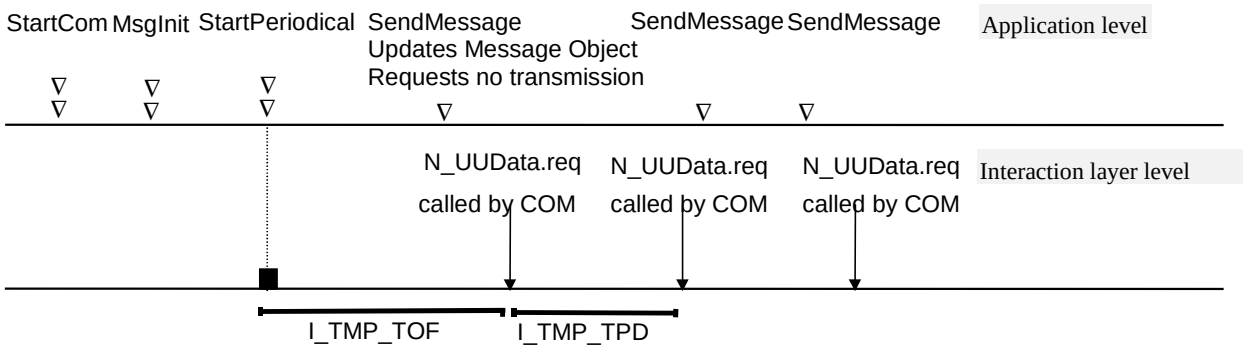

Figure 2-6: Activation/De-activation of Periodical transmission mode –

2.2.7.3.3 Mixed transmission mode

2.2.7.3.3.1 Definitions

Message Value: message value is the message data contained in the message object.

Old Message Value: old message value is the message value which was set by the last update.

Relevant change : A relevant change of the message value shall be detected when the message value matches a condition defined at system generation time

2.2.7.3.3.2 Mixed transmission mode mechanism

The interaction layer shall be able to issue itself periodical transmission requests of the message.

Each call to the API service SendMessage shall update the message object with the application data to be transmitted. The SendMessage service shall not issue any transmission request to the network layer.

The transmission is performed on a cyclic time basis by means of calling the N_UUData.request service of the network layer upon expiration of the mixed transmission mode time period (I_TMM_TPD).

Intermediate transmissions of the message shall be issued within the mixed transmission mode time-period (I_TMM_TPD) upon relevant changes (see system generation chapter) of the value of the message data . These intermediate transmissions do not modify the base cycle (i.e. I_TMM_TPD).

If periodical message transmission is not possible due to message object being locked or BUSY then the transmission shall take place as soon as possible after it is released. Postponement of a message transmission shall not affect the transmission time of subsequent messages.

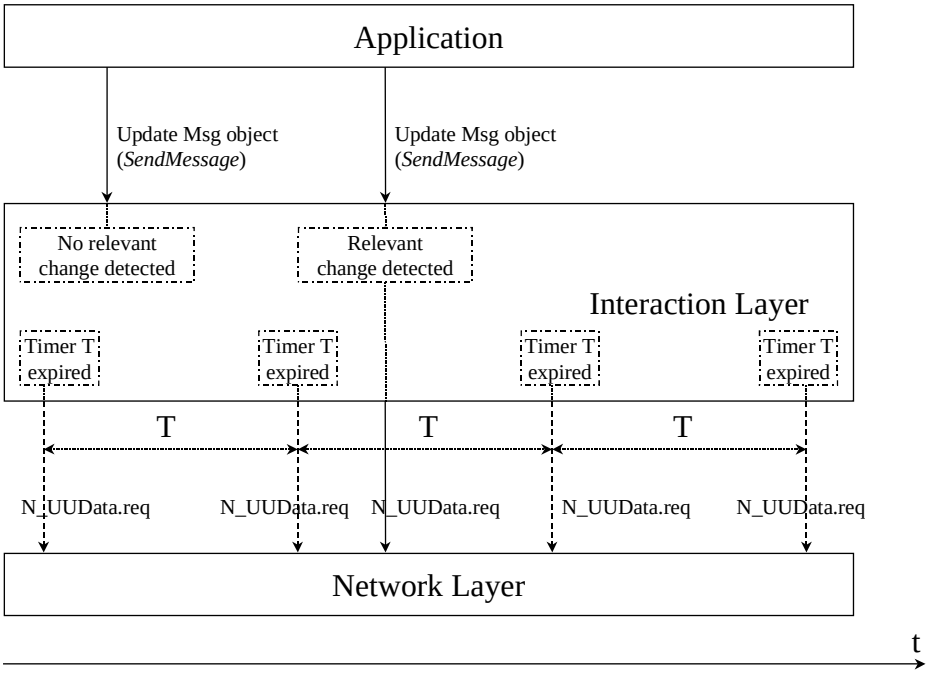

Figure 2-7: Mixed transmission mode

The relevant change is detected by the interaction layer through of an evaluation procedure "Test Message Value" which shall be executed upon each call to the SendMessage service.

The evaluation procedure "Test Message Value" shall be configured and generated at system generation time.

The mixed transmission mode time period (I_TMM_TPD) shall be defined at system generation time.

The Mixed Transmission Mode applies to the UUDT protocol only (Static Length, Static Addressing Scheme).

2.2.7.3.3.3 Activation / Deactivation of mixed transmission mechanism

The periodical transmissions shall be activated by a call to the StartPeriodical() API service. The StartPeriodical service shall start the mixed transmission mode time offset timer (I_TMM_TOF).

The first transmission request (N_UUData.request service) shall be issued upon expiration of the mixed transmission mode time offset (I_TMM_TOF).

StartPeriodical shall be called after the StartCOM API service has completed and once the message object is correctly initialised. The API service MessageInit() can be used to perform this initialisation.

The periodical transmissions shall be stopped by means of the StopPeriodical API service.

The mixed transmission mode time offset (I_TMM_TOF). shall be defined at system generation time.

2.2.8 Message reception

2.2.8.1 Reception using message accessor configuration WithCopy

The provision of a message copy affects the operation of the interface interaction layer - application layer. The provision of a copy located within the application layer requires the transfer of information between the interaction layer's message object and the message copy located within the application layer's.

The information encapsulated within a message object shall be transferred to the appropriate copy whose reference is provided to the reception API service.

This transfer of information between the interaction and application layers shall be applicable for internal, external and internal-external communication.

2.2.8.2 Reception using message accessor configuration WithoutCopy

No transfer of information can take place between the interaction and the application layers.

A reading of the message object shall be performed each time a message accessor associated with a specific message is used to supply its data content to another variable or as part of an operation, e.g. calculation.

2.2.8.3 Unqueued and queued messages

2.2.8.3.1 Queued Message

A queued message behaves like a FIFO.

When the FIFO is empty, the ReceiveMessage() API service is not able to provide any message data to the application ; the ReceiveMessage API service shall then return the status code E_COM_NOMSG.

If a new message data arrives from the Network Layer and the FIFO is not full, this new message is stored into the queue.

If a new message arrives from the Network Layer and the FIFO is full, this message is lost. The next call to the ReceiveMessage API service shall return the oldest message and the status code E_COM_LIMIT.

The pictures below illustrates the behaviour of a queued message.

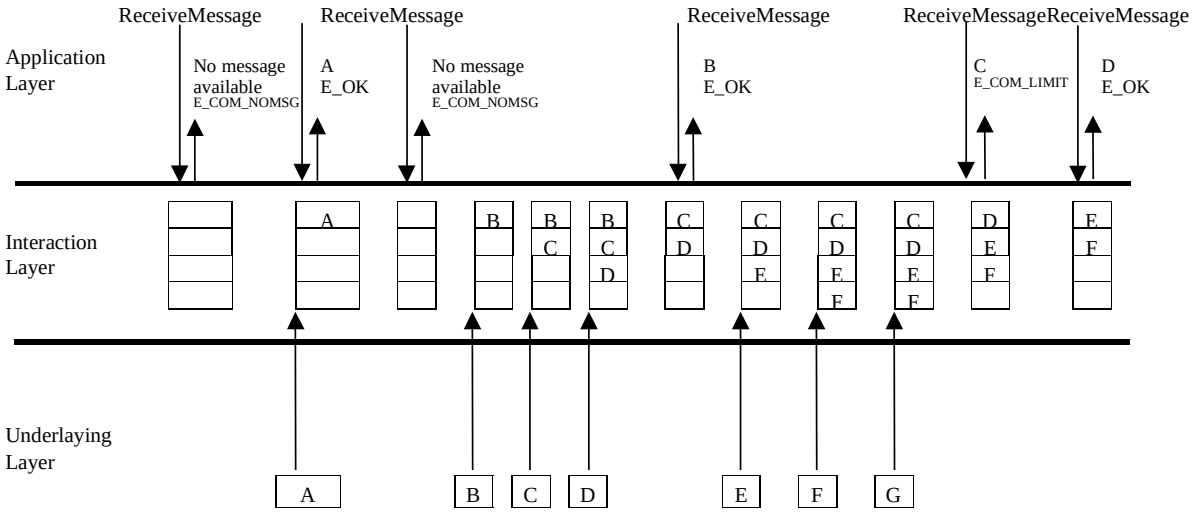

Figure 2-8: Behaviour of Queued Message

The picture below illustrates the behaviour of a queued message with a queue of length equal to 1 : in that case, once a message data has been stored in the queue, no new message data can be stored until the old message has been consumed by the ReceiveMessage API service.

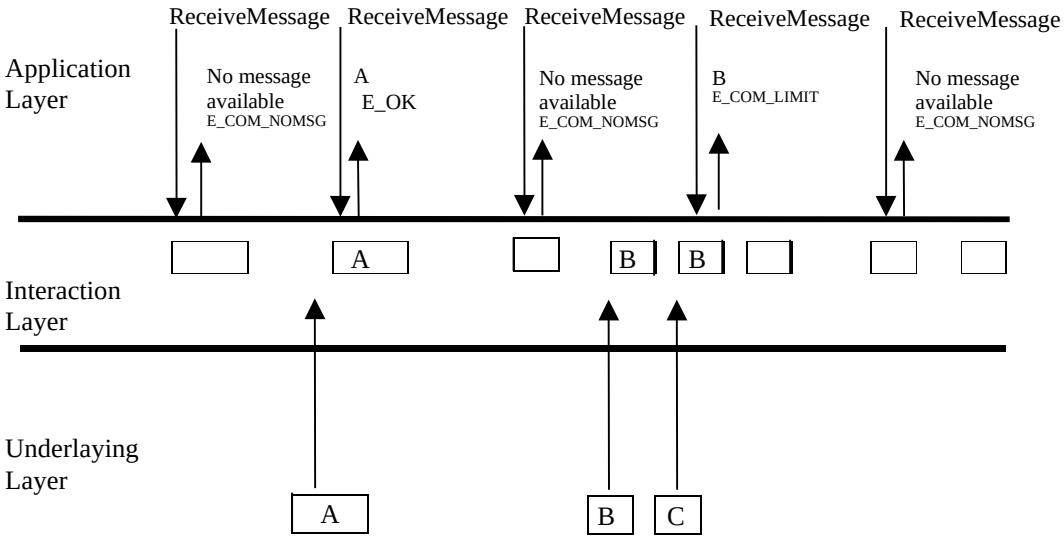

Figure 2-9: Behaviour of Queued Message with a queue length equal to 1

2.2.8.3.2 Unqueued Message

In the case of an unqueued message, no FIFO mechanism shall be implemented. The ReceiveMessage API service shall not consume the message : a message may be read multiple times by the application once it has been received by the Interaction layer.

If no message has been received then ReceiveMessage, ReceiveDynamicMessage or MessageReceiveFrom will return E_OK and the message value set at initialisation.

A message is overwritten by the arrival of a new message data, unless it is locked or BUSY (the new message data is discarded in this case).

The ReceiveMessage API service shall return E_OK even if no new message has been received since the last call of this service. In this particular case, the ReceiveMessage shall provide the same message data to the application than that returned by the last ReceiveMessage, ReceiveDynamicMessage or ReceiveMessageFrom call.

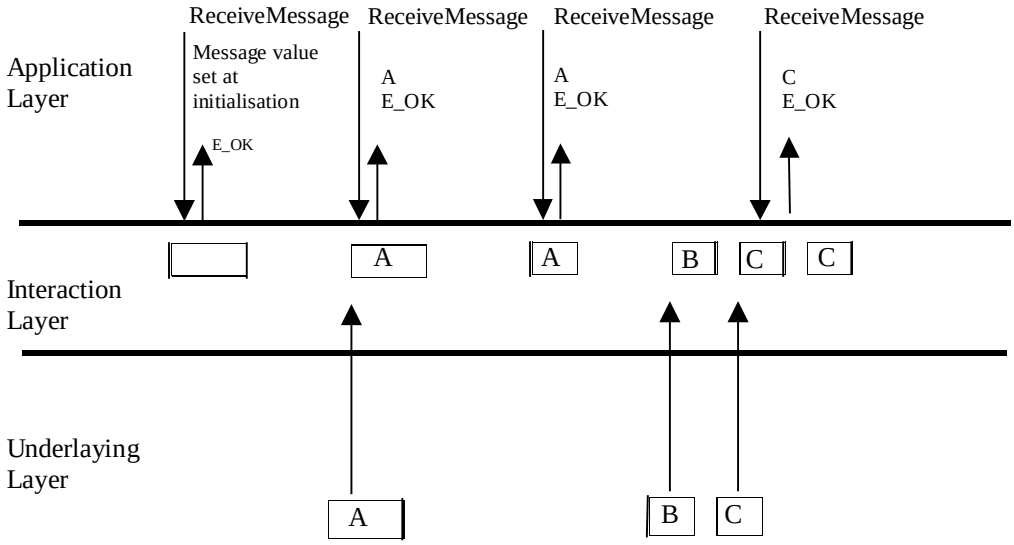

Figure 2-10: Behaviour of Unqueued message

Remark: the behaviour for an Unqueued Message is not the same than that of a Queued Message with a length of queue equal to 1.

2.2.8.3.3 Summary of Transmission modes

The following table summarises the applicability of interaction layer items to transmission modes. ✓ indicates that the transmission mode mentioned in the column header shall be capable of handling the definition item mentioned in the same row - second column.

Table 2-1: Transmission mode summary

		DIRECT	PERIODICAL	MIXED
Message	**Unqueued**	✓	✓	✓
	Queued	✓	✓	✓
Protocol	**UUDT**	✓	✓	✓
	USDT	✓		
Addressing Scheme	**Static**	✓	✓	✓
	Dynamic	✓		
Scope	**Internal**	✓		
	External	✓	✓	✓
	Int-Ext	✓	✓[2]	✓[3]

[2] A message can be transmitted periodically and received by an internal receiver.

[3] A message can be transmitted periodically and received by an internal receiver.

2.2.9 Communication deadline monitoring

This section of the specification defines :

1. mechanisms for monitoring transmission and reception of messages
2. notification mechanisms including services to interface with the OSEK Indirect Network Management (NM).

Communication deadline monitoring can be used:

- to verify on the sender side that transmission requests (periodical or not) are followed by transmissions on the network within a given time frame,
- to verify on the receiver side that periodical messages are received within the allowed time-frame.

Messages to be monitored and respective time-out values shall be defined at system generation time.

Notification mechanisms attached to monitored messages shall be defined at system generation time.

Communication Deadline monitoring is restricted to external and internal-external communication.

2.2.9.1 Definitions

Start Timer: start of measurement of elapsed time.

Timer Running: continuous measurement of elapsed time.

Cancel Timer: stop measurement of elapsed time

Time-out Interval : maximum time allowed for a particular monitored process

Time-out: elapsed time has exceeded the time-out interval .

2.2.9.2 Transmission monitoring

Transmission monitoring is available for any transmission mode.

2.2.9.2.1 Direct Transmission Mode

The Communication Deadline monitoring mechanism monitors that each call to SendMessage, SendDynamicMessage, or SendMessageTo is followed by a transmission on the media within a given time interval (I_CDM_TMD_TO).

The monitoring timer is started upon completion of the call to the SendMessage, SendDynamicMessage or SendMessageTo API service.

The timer is cancelled upon confirmation of the transmission by N_UUData.con or N_USData.con.

The corresponding monitored time interval (I_CDM_TMD_TO) value shall be defined at system generation time.

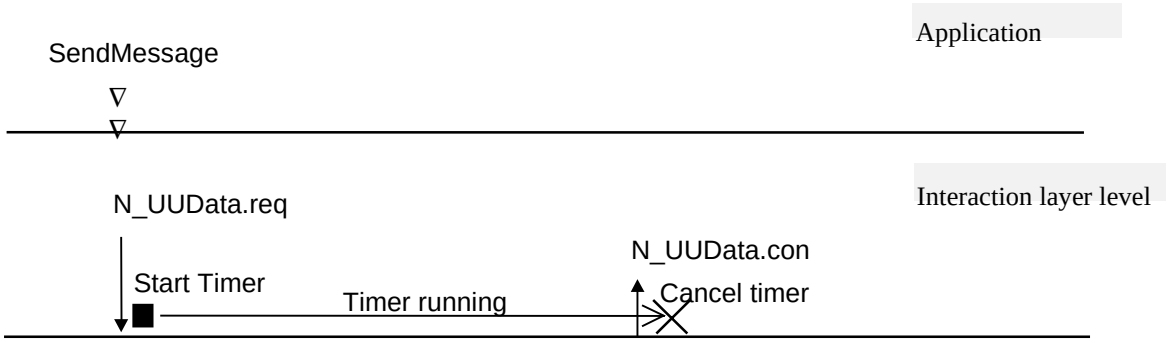

Figure 2-11: Direct transmission mode: example of a successful transmission in case of UUDT protocol

If the transmission does not occur, i.e. if there is no confirmation of the message, the time-out occurs and the application shall be notified using the appropriate notification mechanism.

The interaction layer does not send any additional transmission request to the underlying communication layer upon occurrence of the time-out. It is up to the application to decide of the appropriate actions to be taken.

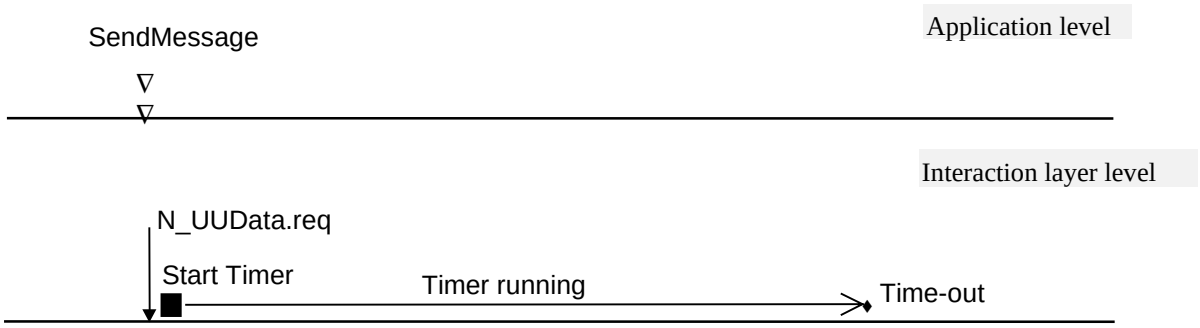

Figure 2-12: Direct transmission mode: example of a failed transmission in case of UUDT protocol

The successful transfer of the message or the occurrence of the time-out can be notified to the application .

2.2.9.2.2 Periodical Transmission Mode

The communication deadline monitoring mechanism monitors that at least one periodical message is transmitted within a given time interval. The time-out interval of the monitored time interval (I_CDM_TMP_TO) can be greater than the transmission period, depending on system design constraints. The time-out interval can be expressed as a number of failed transmissions I_n_FAILED_TMP multiplied by the time transmission period I_TMP_TPD. *The time-out interval of the monitored time interval shall be defined at system generation time.*

The monitoring timer shall be started after each periodical transmission request N_UUData.req if it is not currently running (i.e. if it is the first start of the timer or if the timer was previously cancelled).

The timer of the corresponding monitored time interval (I_CDM_TMP_TO) shall be cancelled by the confirmation of any transmission of the monitored message.

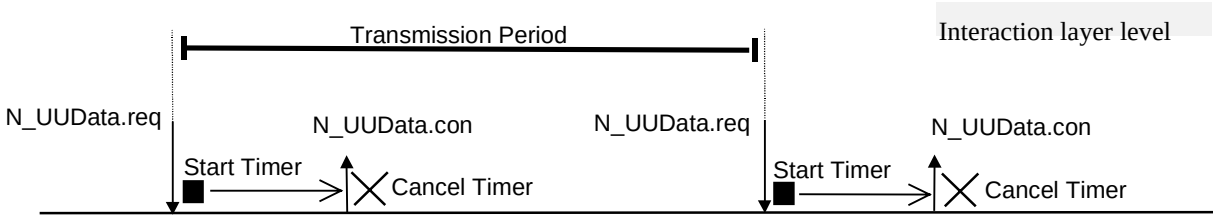

Figure 2-13: Periodical transmission mode: successful transmission

If no transmission occurs, i.e. if there is no message confirmation, the time-out occurs.

If the duration of the monitored time interval is equivalent to several transmission period I_n_FAILED_TMP* I_TMP_TPD with I_n_FAILED_TMP>1), the timer shall not be restarted after each transmission request N_UUData.req : the timer (I_CDM_TMP_TO) shall be only be restarted upon a N_UUData.req if the previous timer has expired.

The interaction layer shall not send any additional transmission request (N_UUData.request) to the underlying communication layer upon occurrence of the time-out. Transmission requests are still performed on the same cycle time basis.

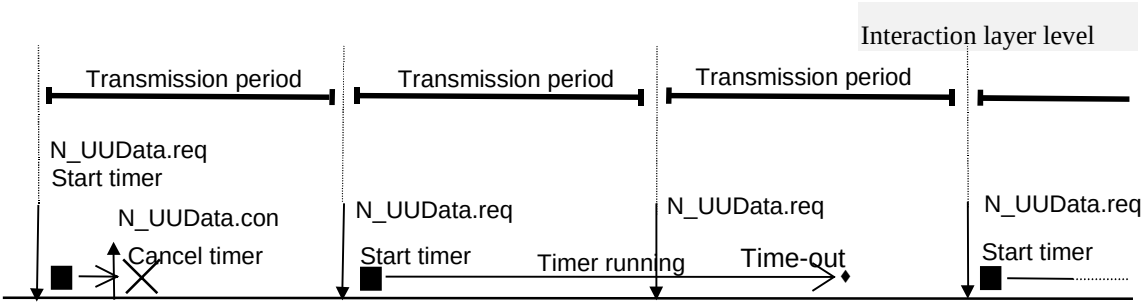

Figure 2-14: Periodical transmission mode: failed transmissions

The successful transfer of a message within the allowed time interval or the occurrence of the time out can be notified to the application or to the Indirect network management (see Network Management specification).

2.2.9.2.3 Mixed Transmission Mode

The Communication Deadline monitoring mechanism monitors that at least one message is transmitted within a given monitored time interval (I_CDM_TMM_TO). The time-out "period" can be expressed as a number of failed transmissions I_n_FAILED TMM multiplied by the time transmission period I_TMM_TPD.

The corresponding time-out "period" (I_CDM_TMM_TO) shall be defined at system generation time.

The timer (I_CDM_TMM_TO) is started after each transmission request N_UUData.req (periodical or issued by a relevant change in the message data value) it is not currently running (i.e. if it is the first start of the timer-out or if the timer was previously cancelled).

The timer is cancelled by the confirmation of any transmission (periodical or relevant change of message data).

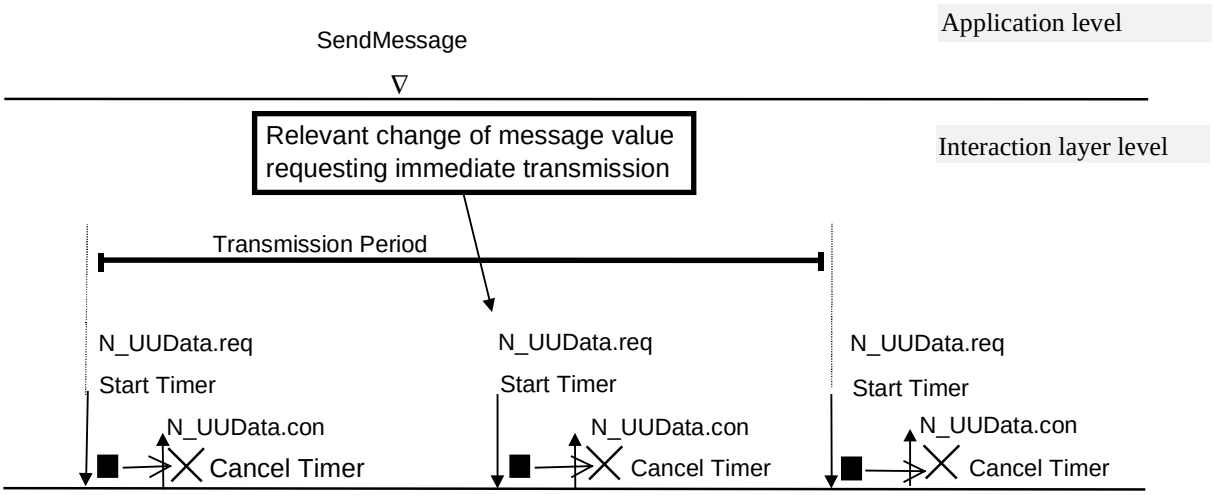

Figure 2-15: Mixed transmission mode: successful transmissions

If the duration of the monitored time interval is equivalent to several transmission period I_n_FAILED_TMM * I_TMM_TPD with I_n_FAILED_TMM>1), the timer shall not be restarted after each transmission request N_UUData.req : the timer shall be only restarted upon a N_UUData.req if the previous the timer has expired.

The interaction layer shall not send any additional transmission request (N_UUData.req) to the underlying communication layer upon occurrence of the time-out. Transmission requests are still performed on the same cycle time basis or upon a relevant change.

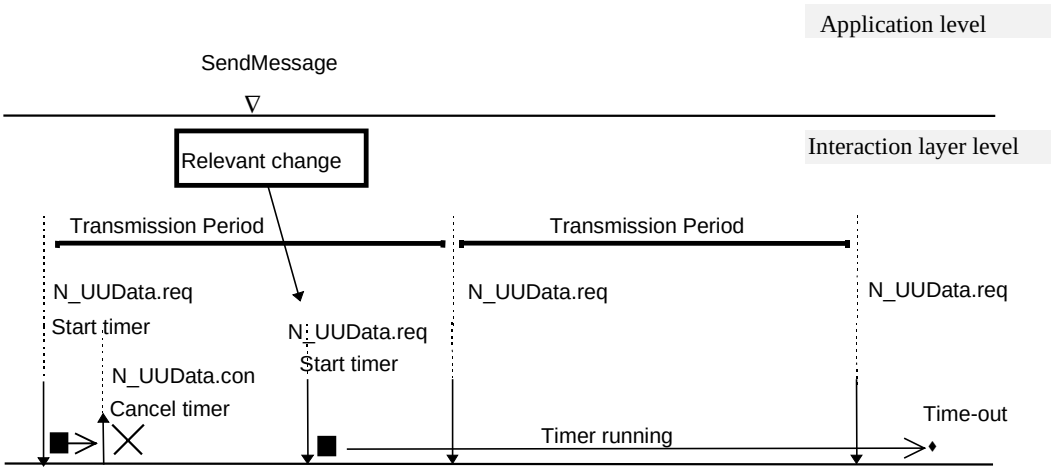

Figure 2-16: Mixed transmission mode: failed transmissions

The successful transfer of a message within the allowed time interval or the expiration of the time out can be notified to the application or to the Indirect Network management (see Network Management specification).

2.2.9.3 Periodical reception monitoring

The communication deadline monitoring mechanism monitors that a cyclic message is received within a given time interval (I_CDM_RX_TO).

The time-out interval of the monitored time interval (I_CDM_RX_TO) shall be defined at system generation time.

The monitoring timer is cancelled and restarted upon each new reception of the message by N_UUData.ind.

If there is no reception and the time-out occurs, the timer shall be immediately restarted.

First time-out

The timer of the first monitored time interval (I_CDM_RX_TO) shall be started once message object initialisation tasks are performed, i.e. after the MessageInit() API has completed.

Depending on system design constraints, a specific value (I_CMD_FRX_TO) can be chosen for the first the time-out interval. This value shall be defined at system generation time.

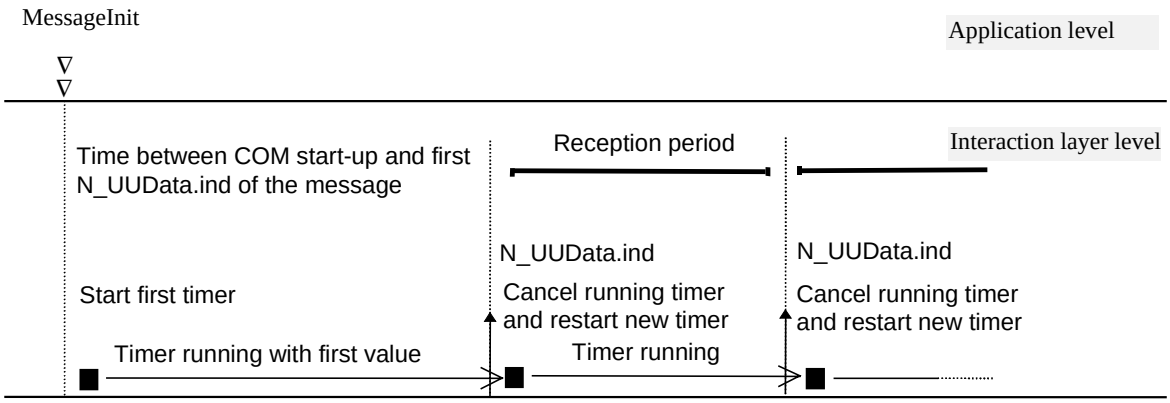

Figure 2-17: Periodical reception: correct and missing receptions

The use of this mechanism is not restricted to monitor the reception of messages transmitted using the periodical transmission mode. The expiration of the time out can be notified to the application or to the Indirect Network management (see Network Management specification).

2.2.10 Notification mechanisms

This section defines notification mechanisms for the application to determine the final status of a previously called send or receive operation.

The notification of the application is performed as soon as a specific event has occurred, e.g. the user does not need to call a specific OSEK COM API service in advance to ensure that the notification scheme is active.

The notification mechanisms are defined at system generation time and cannot be changed at system run-time.

2.2.10.1 Notification classes

OSEK COM supports the following notification classes:

1. **Notification Class 1** : Message Reception
   In case of internal, external or internal-external communication a message has been successfully received.

   In case of internal communication, the appropriate notification mechanism shall be set after the message transmission has been performed : the notification mechanism provides an indication of the current condition.
   In case of external or internal-external communication, the appropriate notification mechanism shall be set upon the occurrence of the network layer service primitives „indication“ and provided that the BUSY flag of the specific message does not indicate that it is in use :

   - For messages using UUDT, the applicable notification mechanism shall be set upon occurrence of N_UUData.indication with N_Result_UUDT = N_OK.

- For messages using USDT, the applicable notification mechanism shall be set upon occurrence of N_USData.indication with N_Result_USDT = N_OK.

2. **Notification Class 2** : Message Transmission
   In case of external or internal-external communication the interaction layer has been notified by the network layer that a message has been successfully transmitted.

   The appropriate notification mechanism shall be set upon the occurrence of the network layer service primitives „confirmation“ depending on the protocol used :

   - For messages using UUDT, the applicable notification mechanism shall be set upon occurrence of N_UUData.confirmation with N_Result_UUDT = N_OK.
   - For messages using USDT, the applicable notification mechanism shall be set upon occurrence of N_USData.confirmation with N_Result_USDT = N_OK.

3. **Notification Class 3** : Message Reception Error
   In case of external or internal-external communication a message reception error has been detected either by the deadline monitoring function or via an error code provided by the indication service primitive of the underlying layer :

   - For messages using UUDT, the applicable notification mechanism shall be set upon occurrence of N_UUData.indication with N_Result_UUDT different from N_OK.
   - For messages using USDT, the applicable notification mechanism shall be set upon occurrence of N_USData.indication with N_Result_USDT different from N_OK.

4. **Notification Class 4** : Message Transmission Error
   In case of external or internal-external communication a message transmission error has been detected either by the deadline monitoring function or via an error code provided by the confirmation service primitive of the underlying layer :

   - For messages using UUDT, the applicable notification mechanism shall be set upon occurrence of N_UUData.confirmation with N_Result_UUDT different from N_OK.
   - For messages using USDT, the applicable notification mechanism shall be set upon occurrence of N_USData.confirmation with N_Result_USDT different from N_OK.

5. **Notification Class 5:** USDT First Frame Indication
   In case of external communication, the applicable notification mechanism shall be set upon occurrence of N_USData_FF.indication service primitive.

2.2.10.2 Notification mechanisms requiring an operating system

The following notification mechanisms requiring the support of an OSEK operating system shall be provided:

1. Task
   The interaction layer activates a task of the application.

   The “Task” notification mechanism shall support notification classes 1 to 5.

2. Event

The interaction layer sets an event to a task of the application.

The “Event” notification mechanism shall support notification classes 1 to 5.

Only one type of notification mechanism can be defined for a given task and a given message, e.g. a given task A using a given message B can make use of either task activation or event setting.

2.2.10.3 Notification mechanisms not requiring an operating system

The following notification mechanisms which do not require the support by an underlying operating system shall be provided:

1. Callback routine
   The interaction layer calls a callback routine provided by the application.
   The “Callback” notification mechanism shall support notification classes 1 to 5.
2. Flag
   The interaction layer sets a flag which can be checked by the application on a cyclic basis by means of the ReadFlag() API service. Resetting of the flag is performed by the application by means of the ResetFlag() API service.
   The "Flag" notification mechanism shall support the notification classes 1 to 5.

The use of OSEK-COM API functions in callback routines are restricted as documented in the usage restriction chapter of the interaction layer.

2.2.10.4 Conditional notification

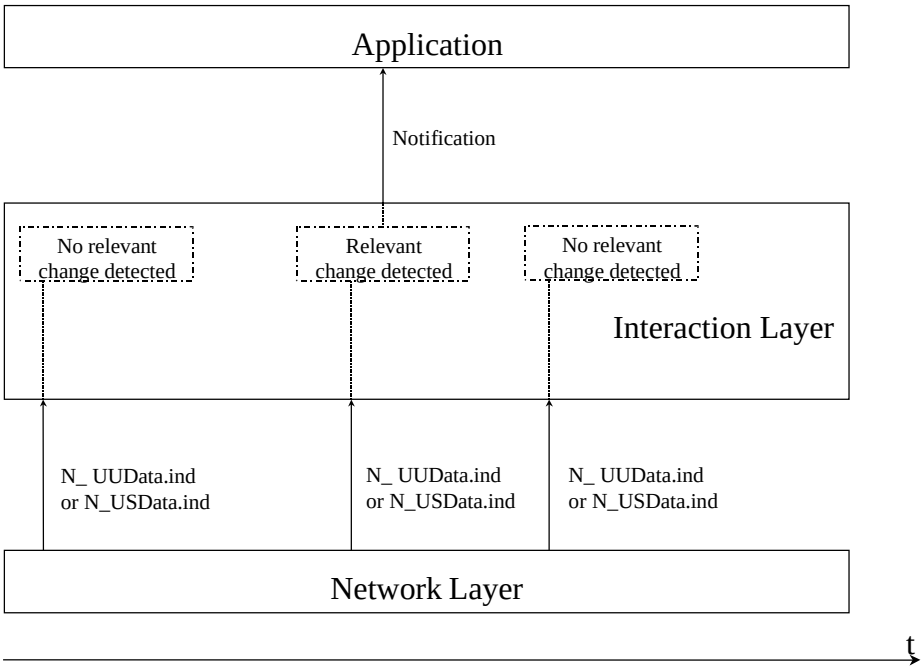

Figure 2-18: Conditional notification data flow

Conditional notification applies to notification class1 only. All notification mechanisms are applicable.

The interaction layer shall perform the following steps if a message arrives (s. flow chart below):

1. Check if conditional notification is enabled for this message
2. If yes, check if there is a relevant change in the message value
3. If there is a relevant change perform the notification mechanism

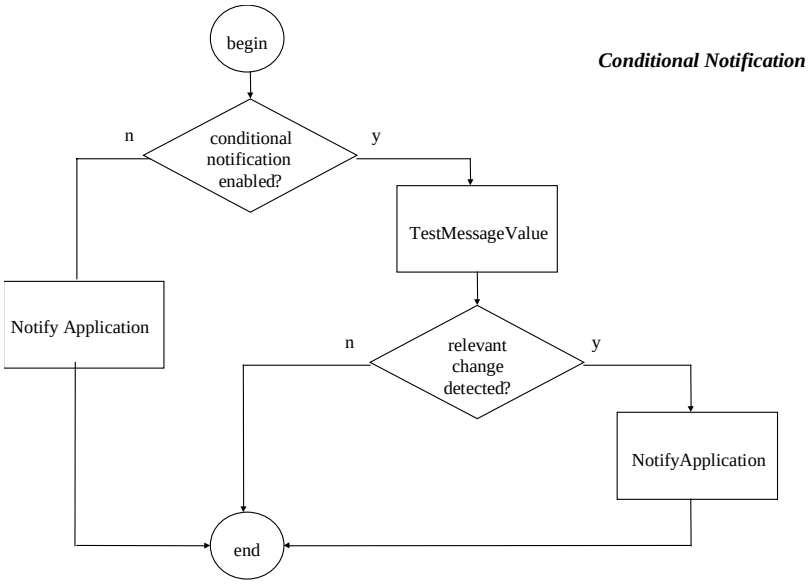

Figure 2-19: Conditional notification flow chart

Relevant changes shall be described at system generation time.

For a definition of relevant changes refer to section 2.7.3 (Mixed Transmission Mode).

2.2.10.5 Summary of notification classes and mechanisms

The following table summarizes the notification classes and mechanisms specified in the previous sections:

Table 2-2: Notification classes and mechanisms

Notification Classes		Conditional Notification	Notification mechanisms			
			task	event	callback	flag
1	Message Reception	✔	✔	✔	✔	✔
2	Message Transmission		✔	✔	✔	✔
3	Message Reception Error		✔	✔	✔	✔
4	Message Transmission Error		✔	✔	✔	✔
5	USDT First Frame Indication		✔	✔	✔	✔

The following table summarises the applicability of interaction layer items to notification mechanisms. ✓ indicates that the service mentioned in the column header shall be capable of handling the definition item mentioned in the same row - second column.

Table 2-3: Summary of notification classes and notification mechanisms

		Class 1	Conditional Class 1	Class 2	Class 3	Class 4	Class 5
Message	**Unqueued**	✓	✓	✓	✓	✓	✓
	Queued	✓	✓	✓	✓	✓	
Protocol	**UUDT**	✓	✓	✓	✓	✓	
	USDT	✓	✓	✓	✓	✓	✓
Scope	**Internal**	✓					
	External	✓	✓	✓	✓	✓	✓
	Int-Ext	✓	✓	✓	✓	✓	
Tx Mode	**Direct**			✓		✓	
	Periodical			✓		✓	
	Mixed			✓		✓	

2.2.11 Interface to OSEK Indirect Network Management

The following services have to be provided to the OSEK Indirect Network Management.

This services are used by OSEK COM to inform Indirect Network Management of the communication deadline monitoring results.

2.2.11.1 Messages transfer indication

Service name:	**`I_MessageTransfer`**
Syntax:	**internal service**
Indication:	I_MessageTransfer.ind (<Sender>)
Parameter(in):	
<Sender>	Uniquely identifies the sending entity.
Parameter(out):	none
Description:	OSEK COM informs OSEK Indirect NM via the service primitive I_MessageTransfer.ind (<Sender>) provided by OSEK NM that a monitored message has been received from a remote node or that a monitored message has been transmitted by the own node.

2.2.11.2 Time-out indication

Service name:	**`I_MessageTimeOut`**
Syntax:	**internal service**
Indication:	I_MessageTimeOut.ind (<Sender>)
Parameter(in):	
<Sender>	Uniquely identifies the sending entity.
Parameter(out):	none
Description:	OSEK COM informs OSEK Indirect NM via the service primitive I_MessageTimeOut.ind (<Sender>) provided by NM that a time-out has occurred for a monitored message.
Particularities:	none

2.2.12 Application programming interface

2.2.12.1 Service parameter type

This chapter describes API services in/out parameter types.

2.2.12.1.1 StatusType

Reference : Refer to the OSEK binding specification

Description :

Each call to an API service shall return a status code, giving thus an information on the completion or failure of the operation.

OSEK COM defines communication specific error codes. The system designer can also add implementation specific error codes.

The following naming convention shall apply :

All return values of OSEK COM API and Start Up services shall start with E_.

Return values of OSEK COM specific services shall begin with begin with E_COM_.

All implementation specific error codes shall start with E_COM_SYS_, e.g. E_COM_SYS_DISCONNECTED.

The communication specific error codes are given in the following table :

Table 2-4:Error codes defined by OSEK COM

StatusCode	Description
E_OK	No error, service call has succeeded.
E_COM_BUSY	Message in use by application task/function.
E_COM_ID	Invalid message name passed as parameter.
E_COM_LIMIT	Overflow of FIFO associated with queued messages.
E_COM_LOCKED	Rejected service call, message object is locked.
E_COM_NOMSG	No message available.
E_COM_RX_ON	On-going message reception.

There is only one error code used throughout the OSEK system (error code common to all specifications) : E_OK.

The implementation vendor has to ensure that the names and values of those errors do not overlap with the OSEK or OSEK COM specific error codes defined above.

To ensure portability of application code, the programmer has to use the symbolic names of the reserved StatusCodes as defined in the table above**.**

2.2.12.1.2 SymbolicName

Type :

Scalar

Range :

Set to support all OSEK messages of a specific application.

Description :

A *SymbolicName* is a type that shall provide a range of possible values for identification of all application specific OSEK message.

2.2.12.1.3 AccessNameRef

Type:

Reference

Range:

Set to support all OSEK messages of a specific application.

Description:

AccessNameRef is the address of the message data field:

In case of WithCopy configuration:

AccessNameRef points to the message copy associated with the message identified by the SymbolicName specified in the given API service.

In case of WithoutCopy configuration:

AccessNameRef points to the message object identified by the SymbolicName specified in the given API service.

2.2.12.1.4 DataLengthRef

Type:

Reference to scalar

Range :

DataLength(Ref) is a scalar reference to a variable that can hold a maximum value equal to 4095 bytes.

Description :

DataLengthRef range is set according to the maximum length known of an OSEK dynamic length message.

2.2.12.1.5 ParamValue

Type :

Scalar

Range :

0 to +127

Description :

ParamValue is a value type item whose range is set to an octet.

ParamValue is an unsigned type.

2.2.12.1.6 AddressRef

Type :

Scalar reference

Range :

0 to +127

Description :

AddressRef is a reference type to an item whose size is equal to an octet.

2.2.12.1.7 FlagValue

Type :

Enumeration

Range :

FALSE, TRUE

Description :

FlagValue is an enumeration ndicating the current state of a message flag.

2.2.12.1.8 FlagType

Type :

Scalar

Range :

Set to support all OSEK messagesflag of a specific application

Description :

AddressRef is a reference type to an item whose size is equal to an octet.

2.2.12.2 Configurations

Two configurations are supported:

Table 2-5: Configurations of the interaction layer

Configurations		Protocols		Supported configurations
Message Length	**A**ddressing scheme	UUDT	USDT	
Static	**S**tatic	✔	✔	SM/SA
Dynamic	**D**ynamic		✔	DM/DA
Static	**D**ynamic			Not supported
Dynamic	**S**tatic		✔	DM/SA

The purpose of "static message length & static addressing scheme" (SM/SA) is to support transfer of static length message (maximum message length of 4095 bytes).

The purpose of "dynamic message length & dynamic addressing scheme" (DM/DA) is to support transfer of dynamic length message (maximum message length of 4095 bytes) that can be attached to multiple set of data link layer attributes.

The purpose of "dynamic message length & static addressing scheme" (DM/SA) is to support the transfer of dynamic length message (maximum message length of 4095 bytes) attached to a single set of data link layer attributes, e.g a single CAN identifier.

2.2.12.3 Start-up services

2.2.12.3.1 InitCOM

Service name:	**InitCOM**
Syntax:	StatusType InitCOM (void)
Parameter(In):	none
Parameter(Out):	none
Description:	This service exists to initialise all hardware and low-level resources used by COM. It may be called before starting the OSEK/VDX kernel, or from within any kernel startup hook routines.
Caveats:	It is not possible to call InitCOM from an OSEK/VDX task level without first masking all interrupts. Calling InitCOM while COM is running will result in undefined behaviour.
Particularities:	The returned StatusType is handled by the application software and is outside the scope of OSEK COM.

Status:

Standard and Extended:

- A1 : The service shall return E_OK if the initialisation completed successfully.
- A2 : The service shall return an implementation or application specific error code if the initialisation did not complete successfully.

2.2.12.3.2 CloseCOM

Service name:	**CloseCOM**
Syntax:	StatusType CloseCOM (void)
Input Parameters:	none
Output Parameters:	none
Description:	This low-level service releases hardware resources used by the OSEK/VDX COM module. It will not release OS resources, which must be returned using StopCOM. CloseCOM must not be called from OSEK/VDX task level unless interrupts are masked. CloseCOM may be called from shutdown hooks or after the OSEK/VDX kernel has been stopped.
Caveats:	All COM operations will cease immediately and hardware will be de-initialised as appropriate. Data will be lost. It is legal to call CloseCOM without calling StopCOM, but OS resources allocated by StartCOM will not be released and COM will be left in an undefined state. Such action would normally only be taken in the event of a critical error, prior to a complete system re-initialisation.

Status :

Standard and Extended:

- A1: The service shall return E_Ok if OSEK COM was shut down successfully
- A2 : The service shall return an implementation or application specific error code if the initialisation did not complete successfully.

2.2.12.3.3 StartCOM

Service name:	**StartCOM**
Syntax:	StatusType StartCOM (void)
Parameter(In):	none
Parameter(Out):	none
Description:	The StartCOM service starts the OSEK communication module. This routine shall perform the initialisation of OSEK COM implementation specific internal states and variables. StartCOM may call the MessageInit function provided by the application programmer if the latter is used to initialise the application specific message objects. If StartCOM or one of the routines called fail then initialisation of the OSEK COM module shall abort and StartCOM shall return a status code as specified below. StartCOM must be called from within a task if an OSEK-compliant operating system is used.
Caveats:	InitCOM must be called to initialise hardware before StartCOM can be called. Failure to call InitCOM before StartCOM will result in undefined behaviour.

Status:

Standard and Extended:

- A1 : The service shall return E_OK if the initialisation completed successfully.
- A2 : The service shall return an implementation or application specific error code if the initialisation did not complete successfully.

2.2.12.3.4 StopCOM

Service name:	**StopCOM**
Syntax:	StatusType StopCOM (Scalar <ShutdownMode>)
Input Parameters:	ShutdownMode: **COM_SHUTDOWN_IMMEDIATE**

	The shutdown will occur immediately without waiting for pending operations to complete.
Output Parameters:	none
Description:	This services causes all OSEK COM activity to cease and all resources used by COM to be returned or left in an inactive state. All operations will cease immediately and hardware will be de-initialised as appropriate. By implication, data will be lost. StopCOM will not return until all pending COM operations have completed and their resources can be released. When StopCOM has completed successfully the system shall be left in a state in which StartCOM can be called to re-initialise OSEK COM.
Particularities :	None.

Status:

Standard and Extended:

- A1 : The service shall return E_OK if OSEK COM was shut down successfully
- A2: The service shall return E_COM_BUSY if OSEK COM could not shut down because an application (task) is currently using a resource owned by OSEK COM.

2.2.12.3.5 MessageInit

Service name:	**`MessageInit`**
Syntax:	StatusType MessageInit (void)
Parameter(In):	none
Parameter(Out):	none
Description:	This routine initialises all application specific message objects.
Particularities:	This function has to be provided by the application programmer and shall be called by the StartCOM routine only. Any other way to initialise application specific message object is allowed provided it does not hinder the functionality of the API services defined in this specification.

Status:

Standard and Extended:

- A1 : The service shall return E_OK if the initialisation of the application specific message object has completed successfully.
- A2 : The service shall return an implementation or application specific error code if the initialisation did not complete successfully.

2.2.12.3.6 StartPeriodical

Service name: **StartPeriodical**
Syntax: StatusType StartPeriodical (void)
Parameter (In): none
Parameter(Out): none
Description: This service shall initiate periodical transmission of messages. This service shall support the periodical transfer of messages using either the periodical or mixed transmission modes. This service shall also be capable to resume periodical transmission of messages if it had been stopped previously by means of StopPeriodical() service.
Particularities: none
Status:

Standard and Extended:

- A1 : The service shall return E_OK if the initiation of the periodical transmission of messages has completed successfully.
- A2 : The service shall return an implementation or application specific error code if the initialisation did not complete successfully.

2.2.12.3.7 StopPeriodical

Service name: **StopPeriodical**
Syntax: StatusType StopPeriodical (void)
Parameter (In): none
Parameter(Out): none
Description: This service shall stop periodical transmission of messages. This service shall stop the periodical transfer of all messages using either the periodical or mixed transmission modes.
Particularities: none
Status:

Standard and Extended:

- A1 : The service shall return E_OK if the initialisation completed successfully.
- A2 : The service shall return an implementation or application specific error code if the initialisation did not complete successfully.

2.2.12.4 Notification mechanism support services

2.2.12.4.1 ReadFlag

Service name:	**ReadFlag**
Syntax:	FlagValue ReadFlag (FlagType FlagName)
Parameter(In):	
FlagName	Message flag name
Parameter(Out):	
FlagValue	State of the flag <FlagName>
Description:	This service returns the value of the specified notification flag <FlagName>.
Particularities:	This service is provided so that appropriate mechanism can be implemented to ensure flag data consistency whilst enabling portable access to the message notification flag. The flag meaning depends to which notification class the specified flag <FlagName> is associated with, eg Flag associated with Notification class 1 and set at TRUE indicates that a message has arrived.

Status:

Standard and Extended::

- A1 : The service shall return TRUE if the conditions associated to the notification class to which the flag is associated are met.
- A2 : The service shall return FALSE if the conditions associated to the notification class to which the flag is associated are not met.

2.2.12.4.2 ResetFlag

Service name:	**ResetFlag**
Syntax:	StatusType ResetFlag (FlagType FlagName)
Parameter(In):	
FlagName	Message flag name
Parameter(Out):	none
Description:	This service set the specified notification flag <FlagName> to FALSE.
Particularities:	This service is provided so that appropriate mechanism can be implemented to ensure flag data consistency whilst enabling portable access to the message notification flag.

Status:

Standard and Extended:

- A1 : The service shall return E_OK if the flag reset completed successfully.
- A2 : The service shall return an implementation or application specific error code if the initialisation did not complete successfully.

2.2.12.5 Communication services

2.2.12.5.1 SendMessage

Service name: **SendMessage**

Syntax: StatusType SendMessage (
SymbolicName <Message>,
AccessNameRef <Data>
)

Parameter (In):

Message	Symbolic name of the message
Data	Reference to the message data field to be transmitted

Parameter (Out): none

Description: The service shall update the message object identified by <Message> depending on the message copy configuration and request transmission of the message object depending on the transmission mode specified. The service shall not verify whether the message object has been initialised.

1. In case of WithCopy :
   The service shall update the message object identified by <Message> with the message copy referenced by the <Data> parameter.
2. In case of WithoutCopy :
   No update of the message object shall be performed since no message copy is used to interface with the message object identified by <Message>.

Particularities: None

Status:

Standard:

- A1 : The service shall return E_COM_LOCKED whenever the message object identified by <Message> is locked. This condition has higher precedence over all other conditions defined in section A2 to A3 of this section.
- A2 : The service shall return E_COM_LOCKED if the copy configuration of <Data> is WithCopy and the message is set BUSY.
- A3 : The service shall return E_OK if the service operation has completed successfully.

Extended:

- A1 to A3 status codes defined under the "Standard" section shall be supported.
- A4 : The service shall return E_COM_ID if the parameter <Message> is invalid.

2.2.12.5.2 ReceiveMessage

Service name: **ReceiveMessage**

Syntax: StatusType ReceiveMessage (
SymbolicName <Message>,
AccessNameRef <Data>
)

Parameter (In):

Message: Symbolic name of the message

Parameter (Out):

Data: Reference to the message data field to store the received data

Description: The service shall deliver the message data associated with the message object <Message> depending on the message copy configuration.

1. In case of WithCopy :
   The service shall update the message referenced by <Data> with the message object identified by <Message>.
2. In case of WithoutCopy :
   The service shall return only a service status since the application accesses directly the message object.

The service shall return message data according to the behaviour specified in the message chapter of the interaction layer.

Particularities: None

Status:

Standard:

- A1 : The service shall return E_COM_LOCKED whenever the message object identified by <Message> is locked. This condition has higher precedence over all other conditions defined in section A2 to A5 of this section.
- A2 : The service shall return E_COM_LOCKED if the copy configuration of <Data> is WithCopy and the message is set BUSY.
- A3 : The service shall return E_OK if data of a queued or unqueued message identified by <Message> is available and returned to the application successfully.
- A4 : The service shall return E_COM_NOMSG if the queued message identified by <Message> is empty.
- A5 : The service shall return E_COM_LIMIT if an overflow of the FIFO of the queued message identified by <Message> occurred since the last call to “Service_Name” for that particular <Message>, E_COM_LIMIT indicates that at least one queued message has been lost and discarded since the FIFO was full upon the queued message arrival.

Extended:

- A1 to A5 status codes defined under the "Standard" section

shall be supported.

- A6 : The service shall return E_COM_ID if the parameter <Message> is invalid.

2.2.12.5.3 GetMessageResource

Service name: **GetMessageResource**

Syntax: StatusType GetMessageResource (
SymbolicName <Message>
)

Parameter (In):

Message — Symbolic name of the message object

Parameter (Out): none

Description: The service GetMessageResource shall set the message object <Message> status as busy.

Particularities: It is recommended that corresponding calls to Get- and ReleaseMessageResource should appear within the same function on the same function level. Before terminating the task or entering the wait state the corresponding service ReleaseMessageResource shall be called by the application layer.

This service can only be used to support the transfer of a message identified by <Message> whose copy configuration is "WithoutCopy".

Status:

Standard:

- A1 : The service shall return E_COM_LOCKED whenever the message identified by <Message> is locked. This condition has higher precedence over all other conditions defined in section A2 to A3 of this section.
- A2 : The service shall return E_OK if the message identified by <Message> has been set to BUSY successfully.
- A3 : The service shall return E_COM_BUSY if the message identified by <Message> is already set to BUSY.

Extended:

- A1 to A3 status codes defined under the "Standard" section shall be supported.
- A4 : The service shall return E_COM_ID if the <Message> parameter is invalid.

2.2.12.5.4 ReleaseMessageResource

Service name: **ReleaseMessageResource**

Syntax: StatusType ReleaseMessageResource (
SymbolicName <Message>
)

Parameter (In):

Message — Symbolic name of the message object

Parameter (Out): none

Description: The service ReleaseMessageResource shall unconditionally set the message object <Message> to NOT_BUSY.

Particularities: It is recommended that corresponding calls to Get- and ReleaseMessageResource appear within the same function on the same function level. Before terminating the task or entering the wait state the corresponding service ReleaseMessageResource shall be used.

This service can only be used to support the transfer of a message identified by <Message> whose copy configuration is "WithoutCopy".

Status:

Standard:

- A1 : The service shall return E _OK after the message has been set to NOT_BUSY.

Extended:

- A1 status code defined under the "Standard" section shall be supported.
- A2 : The service shall return E_COM_ID if the <Message> parameter is invalid.

2.2.12.5.5 SendMessageTo

Service name: **SendMessageTo**

Syntax: StatusType SendMessageTo (
SymbolicName <Message>,
AccessNameRef <Data>,
LengthRef <DataLength>,
AddressRef <Recipients>
)

Parameter (In):

Message — Symbolic name of the message.

Data — Reference to the message data field to be transmitted.

DataLength — Length of the message.

Recipient — Identifier of application receiver of the message.

Parameter (Out): none

Description: The service shall update the message object identified by <Message> depending on the message copy configuration and request transmission of the message object depending on the transmission mode specified. The service shall not verify whether the message object has been initialised. The length (DataLength) and identifier of the application receiver(s) of the message shall be transmitted to the underlying communication layer.

1. In case of WithCopy :
   The service shall update the message object identified by <Message> with the message copy referenced by the <Data> parameter.
2. In case of WithoutCopy :
   No update of the message object shall be performed since no message copy is used to interface with the message object identified by <Message>.

Particularities: This service shall interface with unqueued message only.

Status:

Standard:

- A1 : The service shall return E_COM_LOCKED whenever the message identified by <Message> is locked. This condition has higher precedence over all other conditions defined in section A2 to A3 this section.
- A2 : The service shall return E_COM_LOCKED if the copy configuration of <Data> is WithCopy and the message is set BUSY.
- A3 : The service shall return E_OK if the service operation has completed successfully.

Extended:

- A1 to A3 status codes defined under the "Standard" section shall be supported
- A4 : invalid parameter <Message>, E_COM_ID

2.2.12.5.6 ReceiveMessageFrom

Service name: **ReceiveMessageFrom**

Syntax:
```
StatusType ReceiveMessageFrom (
                    SymbolicName <Message>,
                    AccessNameRef <Data>,
                    LengthRef <DataLength>,
                    AddressRef <Sender>
                    )
```

Parameter (In):

Message Symbolic name of the message

Parameter (Out):

Data	Reference to the message data field to host the received data
DataLength	Reference to the length of the message
Sender	Reference to the identifier of the application sender of message

Description: The service shall deliver the message data associated with the message object <Message> depending on the message copy configuration. The length of the received message data (DataLength) and the identifier of the application sender (Sender) shall be delivered to the application.

1. In case of WithCopy :
   The service shall update the message referenced by <Data> with the message object identified by <Message>.
2. In case of WithoutCopy :
   The service shall return only a service status since the application accesses directly the message object.

The service shall return message data according to the behaviour specified in the message chapter of the interaction layer.

Particularities: This service shall interface with unqueued message only.

Status:

Standard:

- A1 : The service shall return E_COM_LOCKED whenever the message object identified by <Message> is locked. This condition has higher precedence over all other conditions defined in section A2 to A3 this section.
- A2 : The service shall return E_COM_LOCKED if the copy configuration of <Data> is WithCopy and the message is set BUSY.
- A3 : The service shall return E_OK if data of an unqueued message identified by <Message> is available and returned to the application successfully.

Extended:

- A1 to A3 status codes defined under the "Standard" section shall be supported.
- A4 : The service shall return E_COM_ID if the parameter <Message> is invalid.

2.2.12.5.7 SendDynamicMessage

Service name: **SendDynamicMessage**

Syntax: StatusType SendDynamicMessage (
SymbolicName <Message>,
AccessNameRef<Data>,
LengthRef <DataLength>
)

Parameter (In):

Message	Symbolic name of the message
Data	Reference to the message data field to be transmitted
DataLength	Length of the message

Parameter (Out): none

Description: The service shall update the message object identified by <Message> depending on the message copy configuration and request transmission of the message object depending on the transmission mode specified. The service shall not verify whether the message object has been initialised. The length (DataLength) of the message data shall be transmitted to the underlying communication layer.

1. In case of WithCopy :
   The service shall update the message object identified by <Message> with the message copy referenced by the <Data> parameter.
2. In case of WithoutCopy :
   No update of the message object shall be performed since no message copy is used to interface with the message object identified by <Message>.

Particularities: This service shall interface with unqueued message only.

Status:

Standard:

- A1 : The service shall return E_COM_LOCKED whenever the message identified by <Message> is locked. This condition has higher precedence over all other conditions defined in section A2 to A3 this section.
- A2 : The service shall return E_COM_LOCKED if the copy configuration of <Data> is WithCopy and the message is set BUSY.
- A3 : The service shall return E_OK if the service operation has completed successfully.

Extended:

- A1 to A3 status codes defined under the "Standard" section shall be supported
- A4 : The service shall return E_COM_ID if the parameter

<Message> is invalid.

2.2.12.5.8 ReceiveDynamicMessage

Service name: **ReceiveDynamicMessage**

Syntax: StatusType ReceiveDynamicMessage (
SymbolicName <Message>,
AccessNameRef <Data>,
LengthRef <DataLength>,
)

Parameter (In):

Message	Symbolic name of the message

Parameter (Out):

Data	Reference to the message data field to host the received data
DataLength	Reference to the length of the message

Description: The service shall deliver the message data associated with the message object <Message> depending on the message copy configuration. The length of the received message data (DataLength) and the identifier of the application sender (Sender) shall be delivered to the application.

1. In case of WithCopy :
   The service shall update the message referenced by <Data> with the message object identified by <Message>.
2. In case of WithoutCopy :
   The service shall return only a service status since the application accesses directly the message object.

The service shall return message data according to the behaviour specified in the message chapter of the interaction layer specification.

Particularities: This service shall interface with unqueued message only.

Status:

Standard:

- A1 : The service shall return E_COM_LOCKED whenever the message object identified by <Message> is locked. This condition has higher precedence over all other conditions defined in section A2 to A4 this section.
- A2 : The service shall return E_COM_LOCKED if the copy configuration of <Data> is WithCopy and the message is set BUSY.
- A3 : The service shall return E_OK if data of an unqueued message identified by <Message> is returned to the application successfully.

Extended:

- A1 to A3 status codes defined under the "Standard" section shall be supported.
- A4 : The service shall return E_COM_ID if the parameter <Message> is invalid.

2.2.12.5.9 GetMessageStatus

Service name: **GetMessageStatus**

Syntax: StatusType GetMessageStatus (
SymbolicName <Message>
)

Parameter (In):

Message: Symbolic name of the message object

Parameter (Out): none

Description: The service *GetMessageStatus* shall return the current status of the message object <Message>. If this service call fails, it shall return an implementation specific error code that shall be distinguishable from all other return values.

Particularities: none

Status:

Standard:

- A1 : The service shall return E_COM_LOCKED whenever the message identified by <Message> is locked. This condition has higher precedence over all other conditions defined in section A2 to A6 this section.
- A2 : The service shall return E_COM_BUSY if the message is currently set to BUSY.
- A3 : The service shall return E_COM_NOMSG if the FIFO of the queued message identified by <Message> is empty.
- A4 : The service shall return E_COM_LIMIT if an overflow of the FIFO of the queued message identified by <Message> occurred.
- A5 : The service shall return E_OK if none of the conditions specified in A1 to A6 of this section is applicable nor fulfilled and no indication of error is present.

Extended:

- A1 to A5 status codes defined under the "Standard" section shall be supported.
- A7 : The service shall return E_COM_ID if the <Message> parameter is invalid.

2.2.12.5.10 ChangeProtocolParameters

Service name: **ChangeProtocolParameters**

Syntax: StatusType ChangeProtocolParameters (
SymbolicName <Message>,
ParamValue <BS_Value>
ParamValue <ST_Value>
)

Parameter (In):

Message	Symbolic name of the message
BS_Value	value to be assigned to the BS parameter of the network layer.
ST_Value	value to be assigned to the STmin parameter of the network layer.

Parameter (Out): none

Description: This service shall modify the network layer parameter BS and STmin of the message identified by <Message>. This service shall use the network layer interface service N_ChangeParameter.request.

The API service parameter BS_Value shall be assigned to the interface network service parameter N_BS_Value.

The API service parameter ST_Value shall be assigned to the interface network service parameter N_ST_Value.

Status:

Standard:

- A1 : The service shall return E_OK if the parameters change completed successfully.
- A2 : The service shall return E_COM_RX_ON if the parameters change failed due to an on-going message reception.

Extended:

- A1 to A2 status codes defined under the "Standard" section shall be supported.
- A3 : The service shall return E_COM_ID if the parameter <Message> is invalid.

2.2.12.5.11 Summary of API services

The following table summarises the applicability of interaction layer items to API services. ✓ indicates that the service mentioned in the column header shall be capable of handling the definition item mentioned in the same row - second column.

Table 2-6: Summary of API communication services

		SendMessage	ReceiveMessage	SendDynamicMessage	ReceiveDynamicMessage	SendMessageTo	ReceiveMessageFrom
Message	**Unqueued**	✓	✓	✓	✓	✓	✓
	Queued	✓	✓				
Message length	**Static**	✓	✓				
	Dynamic			✓	✓	✓	✓
Addressing Scheme	**Static**	✓	✓	✓	✓		
	Dynamic					✓	✓
Protocol	**UUDT**	✓	✓				
	USDT	✓	✓	✓	✓	✓	✓
Scope	**Internal**	✓	✓				
	External	✓	✓	✓	✓	✓	✓
	Int-Ext	✓	✓				
Tx Mode	**Direct**	✓		✓		✓	
	Periodical	✓					
	Mixed	✓					

2.2.13 Usage of OSEK COM services

The OSEK COM specification does not make any restrictions regarding the usage of services provided by the OSEK operating system. The services that can be used by OSEK COM can be found in the OSEK Operating System specification.

The table below lists usage requirements of OSEK COM services within different type of OSEK activities (task, function, ISR or Callback).

Table 2-7: COM services available for TASK and ISR

Communication API Services		Activities			
		Task	Function	ISR	Callback
SendMessage	Unqueued	**Yes**	**Yes**	**Yes** WithCopy only	**Yes** WithCopy only
	Queued	**Yes**	**Yes**	**No**	**Yes** WithCopy only
ReceiveMessage	Unqueued	**Yes**	**Yes**	**Yes** WithCopy only	**Yes** WithCopy only
	Queued	**Yes**	**Yes**	**No**	**Yes** WithCopy only
GetMessageStatus		**Yes**	**Yes**	**Yes**	**Yes**
GetMessageResource		**Yes**	**Yes**	**No**	**No**
ReleaseMessageResource		**Yes**	**Yes**	**No**	**No**
SendDynamicMessage		**Yes**	**Yes**	**Yes** WithCopy only	**Yes** WithCopy only
ReceiveDynamicMessage		**Yes**	**Yes**	**Yes** WithCopy only	**Yes** WithCopy only
SendMessageTo		**Yes**	**Yes**	**Yes** WithCopy only	**Yes** WithCopy only
ReceiveMessageFrom		**Yes**	**Yes**	**Yes** WithCopy only	**Yes** WithCopy only

No OSEK COM runtime services are allowed to be called from within OSEK OS hook routines.

2.2.14 Mapping of interaction layer to network layer services

The interaction layer shall use the services provided by the OSEK network layer. This section documents the interface between the interaction layer and the network layer (see network layer chapter).

The message identifier <SymbolicName> is assigned to a specific <N_Handle>.

The application data stored in a message object <Data> is assigned to the network layer service parameter <N_User_Data>.

The length of a dynamic length message <DataLength> is assigned to the network layer service parameter <N_Length>.

The application receiver address <Recipients> is assigned to the network layer service parameter <N_TA>. The <N_TA> parameter shall only be used if interaction layer's Dynamic Addressing scheme (DA) is used.

The application sender address <sender> is assigned to the network layer service parameter <N_SA>. The <N_SA> parameter shall only be used if interaction layer's Dynamic Addressing scheme (DA) is used.

The value of the BS parameter <BS_Value> is assigned to the network layer service parameter <N_BS_Value>.

The value of the ST parameter <ST_Value> is assigned to the network layer service parameter <N_ST_Value>.

Table 2-8: Interaction layer / Network layer interface

Interaction layer / Network layer interface	
Interaction layer parameters	Network layer service parameters
SymbolicName	N_Handle
Recipients	N_TA
Sender	N_SA
BS_Value	N_BS_Value
ST_Value	N_ST_Value
Data	N_User_Data
Length	N_Length

3 Network layer

This chapter states the requirements of the network layer.

Requirements for the provision of network layer functionality are stated in the communication conformance class section of the OSEK COM specification.

3.1 Network layer overview

The network layer provides services to the interaction layer for the transfer of messages to the underlying data link layer. Two communication protocols are defined: Unacknowledged Unsegmented Data Transfer (UUDT) and Unacknowledged Segmented Data Transfer (USDT). The network layer provides flow control mechanisms to enable interfacing of communication entities featuring different levels of performance and capabilities. The network layer uses services provided by the data link layer.

3.1.1 Network Layer operation

The main purpose of the network layer is to transfer messages to and from the underlying data link layer. Messages within the data link layer must conform to the data link layer data unit size, so messages that are larger than the data link layer data unit size are split up by the network layer and transferred to the data link layer for transmission as multiple frame messages. Multiple frame messages that are received from the data link layer are reassembled by the network layer and presented to the interaction layer as a complete message. A flexible flow control scheme is implemented in the network layer to enable message receivers to regulate the rate of frame arrival within multiple frame messages.

3.1.2 Unacknowledged Unsegmented Data Transfer

Unacknowledged Unsegmented Data Transfer supports the transmission of single frame messages, i.e. messages whose data fits within the data link layer service data unit size. The receiver does not confirm the reception of the message to the sender.

3.1.2.1 UUDT Services

The transmission of a UUDT message is requested by calling the network layer service N_UUData.request. The result of the transmission request is indicated to the sender by the service N_UUData.confirmation and the reception of the single frame message is indicated to the receiver by the service N_UUData.indication.

3.1.2.2 UUDT Frame Format

Each transmitted frame consists of two parts: a network address field and a network data field.

3.1.2.3 UUDT Message Transmission

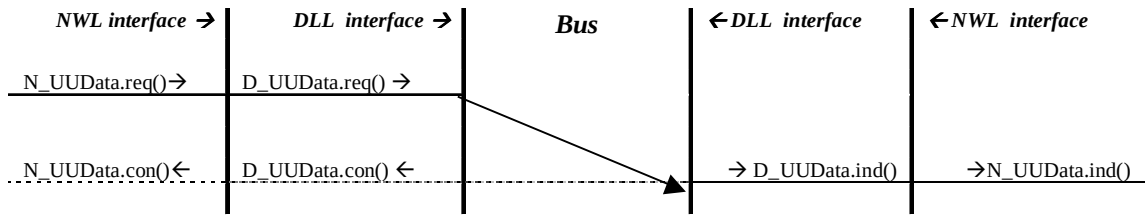

Figure 3-1: UUDT message transmission

The figure above illustrates the sequence of service calls for a UUDT message transmission, where the receiver is on a different ECU to the sender. A single frame is transmitted on the bus.

3.1.3 Unacknowledged Segmented Data Transfer

Unacknowledged Unsegmented Data Transfer supports the transmission of messages with up to 4095 bytes of data. If the size of the message is such that the data will fit within the data link layer data unit size, a single frame message transmission will result. This has a maximum of 15 bytes although the actual size may be less due to the data field size of the bus protocol frame. A USDT single frame message differs from a UUDT message due to the inclusion of a Network Protocol Control Information (NPCI) field. This results in one less byte being available for data. If the size of the message is such that the data will not fit within the data link layer data unit size, a multiple frame message transmission will occur. The receiver does not confirm the reception of the message to the sender.

3.1.3.1 USDT Services

The transmission of a UUDT message is requested by calling the network layer service N_USData.Request. The arrival of the first frame of the message is indicated to the receiver by the service N_USData_FF.indication. The final result of the transmission request is indicated to the sender by the service N_USData.confirmation and the reception of the complete message is indicated to the receiver by the service N_USData.indication. Specific internal parameters of the network layer may be changed by the service N_ChangeParameter.request and the result confirmed by the service N_ChangeParameter.confirmation.

3.1.3.2 USDT Frame Format

Each transmitted frame consists of three parts: a network addressing information field, a network protocol control information field and a network data field.

3.1.3.3 USDT Flow Control

In the case of USDT message transmission, the network layer transfers each frame of the message to the data link layer by a separate call to the data link layer services. The rate of transmission of the message consecutive frames is controlled by means of flow control frames

transmitted by the message receiver. The first flow control frame indicates the minimum separation time (STmin) of subsequent frames to be transmitted by the sender and the block size (BS). The Block Size is the number of Consecutive Frames that may be transmitted by the sender after which another Flow Control frame must be transmitted by the receiver. Only the first flow control frame contains valid values for STmin and BS. All flow control frames contain the network protocol control information Clear To Send or Wait. Clear To Send indicates that the sender may transmit the next Consecutive Frame. Wait tells the sender not to send any more frames to the receiver until a flow control Clear To Send frame is received. The receiver may transmit multiple flow control Wait frames.

3.1.3.4 USDT Dynamic Parameter Change

The network layer protocol parameters STmin and BS (Block Size) are specific to each message object which may be received and may be changed during run time. A change is only possible while the specific message is not being received, i.e. before the reception of a First Frame and after the reception of the entire message. The change is effected by the N_ChangeParameter.request service.

3.1.3.5 USDT Single Frame Message Transmission

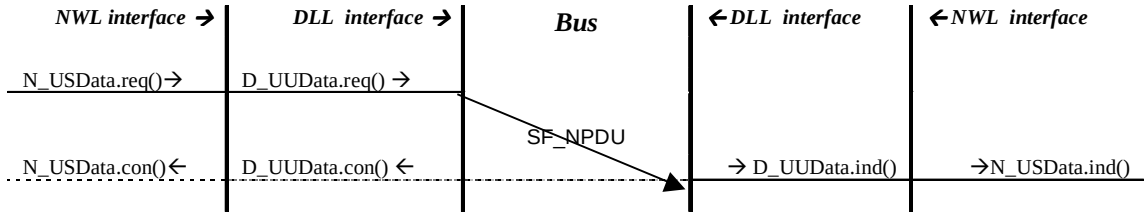

Figure 3-2: USDT single frame message transmission

The figure above illustrates the sequence of service calls for a USDT single frame message transmission where the receiver is on a different ECU to the sender. The value in brackets indicates the value of the network protocol control information field of the transmitted frame. A single frame is transmitted on the bus.

3.1.3.6 USDT Multiple Frame Message Transmission

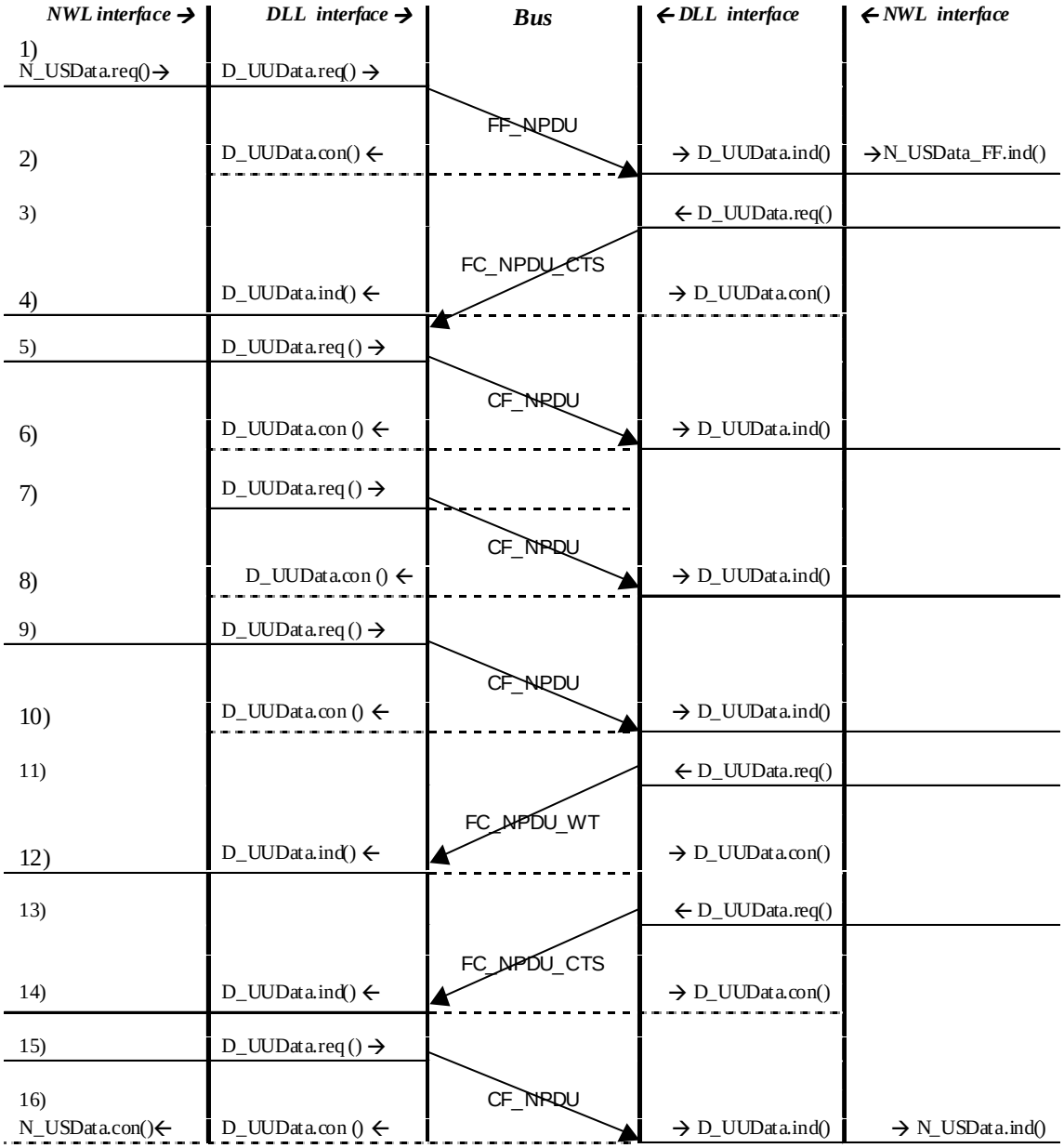

Figure 3-3: USDT multiple frame message transmission

The figure above illustrates an example of the sequence of service calls for a USDT multiple frame message transmission, where the receiver is on a different ECU to the sender. The value in brackets indicates the value of the network protocol control information field of the transmitted frame.

In this example, the following events occur:

1. The sender transmits a First Frame (FF_NPDU). This contains the message data length field in the network protocol control information.

2. The sender receives notification of successful transmission and the receiver receives notification of the arrival of the First Frame.
3. The receiver transmits a Flow Control frame (FC_NPDU_CTS). This contains the network protocol control information that the flow status is Clear To Send, the minimum separation time (STmin) of subsequent frames to be transmitted by the sender, and the block size (BS). In this example, the block size is 3.
4. The sender receives notification of the arrival of the Flow Control frame and the receiver receives notification of the successful transmission of the Flow Control frame.
5. The sender transmits the first Consecutive Frame (CF_NPDU) of the block. This frame contains a Sequence Number (SN) equal to one in the network protocol control information.
6. The sender receives notification of successful transmission and the receiver receives notification of the arrival of the Consecutive Frame.
7. The sender transmits the second Consecutive Frame (CF_NPDU) of the block. The transmission occurs at least STmin milliseconds after the transmission of the first Consecutive Frame. This frame contains a Sequence Number (SN) equal to two in the network protocol control information.
8. The sender receives notification of successful transmission and the receiver receives notification of the arrival of the Consecutive Frame.
9. The sender transmits the third Consecutive Frame (CF_NPDU) of the block. The transmission occurs at least STmin milliseconds after the transmission of the second Consecutive Frame. This frame contains a Sequence Number (SN) equal to three in the network protocol control information.
10. The sender receives notification of successful transmission and the receiver receives notification of the arrival of the Consecutive Frame.
11. The receiver now transmits a Flow Control frame (FC_NPDU_WT) as the three Consecutive Frames that make a block have now been received. In this example, the Flow Control frame contains the network protocol control information that the flow status is Wait. The sender must not send any further frames until it receives a Flow Control frame with a flow status of Clear To Send.
12. The sender receives notification of the arrival of the Flow Control frame and the receiver receives notification of the successful transmission of the Flow Control frame.
13. In this example, the receiver now transmits a Flow Control frame (FC_NPDU_CTS). This contains the network protocol control information that the flow status is Clear To Send.
14. The sender receives notification of the arrival of the Flow Control frame and the receiver receives notification of the successful transmission of the Flow Control frame.
15. The sender now transmits the first Consecutive Frame (CF_NPDU) of the second block. This frame contains a Sequence Number (SN) equal to four in the network protocol control information. In this example this frame contains the last byte of data.
16. The sender receives notification of successful transmission of the Consecutive Frame and the receiver receives notification that the complete message has been received.

3.1.4 Network layer timing constraints

If desired, timers may be enabled to monitor the time between specific events, such as the time to transmit a frame, or the time to receive the next Flow Control or Consecutive Frame. If a timeout occurs, i.e. the expected event did not occur within the specified period, the transmission or reception will be aborted and an error code returned to the interaction layer.

3.1.5 Interleaving of messages

The USDT protocol supports the interleaving of different messages. This means that multiple messages may be transmitted and received simultaneously, with the individual frames of the different messages being transmitted as determined by the bus access privilege of each frame.

3.2 Network layer specification

3.2.1 Definitions

Protocol : a set of defined rules to co-ordinate the exchange of information in a predefined way.

Protocol layer : a domain that defines a *protocol* in accordance with the OSI basic reference model standard.

Service data unit : an amount of information whose identity is preserved when transferred by a *protocol layer* and which is not interpreted by the said *protocol layer*.

Network layer : a specification of a *protocol layer* corresponding to the layer number three (3) of the OSI basic reference model.

Network service data unit : a *service data unit* that is defined for and belongs to the *Network layer*.

Protocol entity : an active and executable element of a specific *protocol layer* that performs the exchange of information in accordance with the *protocol* of that layer.

Network protocol entity : an active and executable element of the *Network layer* that performs the exchange of information in accordance with the *protocol* of that layer.

Network service user : a *protocol entity* that uses a service provided by a *network protocol entity*.

Protocol data unit : a unit of data specified in a *protocol layer* that is used to both support the exchange of information and co-ordinate the joint operation of the *protocol entities*.

Receiving network entity : a *protocol entity* of the *network layer* that receives information using the *network layer protocol*.

Sending network entity : a *protocol entity* of the *network layer* that sends information using the *network layer protocol*.

Network layer message : a *network service data unit* that is transferred, i.e. sent and received, by *network layer entities*.

Network User Data : the information transferred between *network protocol entities* on behalf of the *network service user* entities for whom the *network protocol entities* are providing services.

Multiple frame message : a *network layer message* that is segmented by a *sending network entity* and transmitted to a remote *receiving network entity* using services provided by the *data link layer*.

Single frame message : a *network layer message* that is be transmitted to a (or multiple) remote *receiving network entity(ies)* using services provided by the *data link layer*.

1:1 communication : communication supported by a *protocol* that enables the transmission of *network layer message* between exactly one *sending network entity* and exactly one *receiving network entity*.

1: N communication : communication supported by a *protocol* that enables the transmission of *network layer messages* between exactly one *sending network entity* and several *receiving network entities*.

3.2.2 Generality

The protocol layer defines two communication protocols :

1. Unacknowledged Unsegmented Data Transfer (UUDT) : this protocol supports the exchange of single frame message (only) with no network protocol control information (NPCI).
2. Unacknowledged Segmented Data Transfer (USDT) : this protocol supports the exchange of multiple and single frame messages. Network layer messages exchanged using the Unacknowledged Segmented Data Transfer (USDT) protocol are transferred with network protocol control information (NPCI).

Requirements for the provision of implemented communication protocols are stated in the communication conformance class section of the OSEK COM specification.

The mapping of service user elements (e.g. protocol data units) to the respective parameters of the interface services of the network layer is defined in the service user's protocol layer section of the OSEK COM specification.

3.2.3 Unacknowledged Unsegmented Data Transfer

3.2.3.1 Service data units

3.2.3.1.1 <N_Handle>

Type :

scalar

Range :

Application specific.

Description :

A N_Handle identifies a specific network layer message. A N_Handle is mono-directional, it shall either support the transmission or the reception of a network layer message. A N_Handle is associated with exactly one message of the interaction layer.

A N_Handle shall be mapped to a distinct and mono-directional D_Handle of the data link layer interface. A D_Handle that shall be used to support the transfer of network service data unit.

The picture below illustrates the relationship between a N_Handle with the interaction layer and the data link layer interface for a network message that is to be transmitted. A message of the interaction layer identified by a SymbolicName (i.e. SymbolicName_*1*) is associated with a mono-directional N_Handle (N_Handle_*1*) that support the

transmission of a network message relying on a single mono-directional data link handle (D_Handle_*11*).

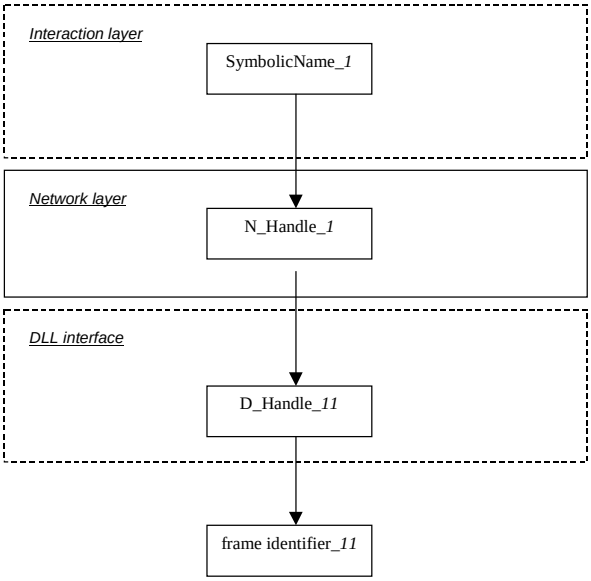

Figure 3-4: N_Handle (UUDT)

3.2.3.1.2 <N_TA>

Type :

scalar

Range :

(Decimal) : 0-255

Description :

The N_TA service parameter is optional and applicable as defined in the interaction layer. If used, this service data unit shall be assigned to the D_TA service data unit of the data link layer.

3.2.3.1.3 <N_SA>

Type :

scalar

Range :

(Decimal) : 0-255

Description :

The N_SA service parameter is optional and applicable as defined in the interaction layer. If used, the D_TA service data unit of the data link layer shall be assigned to the N_SA service data unit.

3.2.3.1.4 <N_User_Data>

Type :

string of bytes

Range :

The maximum length of this parameter shall be equal to the maximum length of the data link layer user data (D_User_Data).

Description :

N_User_Data is the network user data that shall be transferred by the sending network entity to the receiving network entity via a single call to the data link layer transmission request service (D_UUData.request).

3.2.3.2 Interface control information

3.2.3.2.1 <N_Result_UUDT>

Type :

enumeration

Range :

N_OK ,

N_NM

Description :

This parameter contains the status of the execution of the interface service N_UUData.request.

N_OK : this parameter indicates successful completion of the associated service called previously. This parameter can be issued to a service user on either both the sender or receiver side.

N_NM : this parameter shall be associated with a network layer message delivered by D_UUData.indication with D_Result_UUDT equal to D_NM.

3.2.3.3 Interface services

3.2.3.3.1 N_UUData.request

The service primitive requests transmission of the network user data <N_User_Data> identified by <N_Handle>.

N_UUData.request **(**
<N_Handle>
<N_User_Data>
)

3.2.3.3.2 N_UUData.confirmation

The service primitive confirms the completion (successful or not) of a N_UUData.request service for a specific <N_Handle>.

The parameter <N_Result_UUDT> provides the status of the executed service request.

N_UUData.confirmation **(**
<N_Handle>
<N_Result_UUDT>
)

3.2.3.3.3 N_UUData.indication

The service primitive delivers the user data <N_User_Data> identified by <N_Handle>.

The parameter <N_Result_UUDT> provides the status of the service indication.

N_UUData.indication **(**
<N_Handle>
<N_User_Data>
<N_Result_UUDT>
)

3.2.3.4 Protocol Data units

The network protocol data unit (NPDU) consists of various fields that enable each protocol entity to operate in conformance with a set of communication rules as defined in this document.

3.2.3.4.1 Network protocol data unit fields description

3.2.3.4.1.1 Network addressing information

The network addressing information (N_AI) field contains the N_Handle, N_TA (optionnal) and N_SA (optional) of a single frame message.

3.2.3.4.1.2 Network data field

The network data field (N_Data) contains the network user data to be exchanged.

The length of the network data field (N_Data) shall be equal to the length of the applicable data link layer user data (D_User_Data).

The length of the network data field (N_Data) shall be user defined at system generation time. The length of the network data field (N_Data) shall be referred to as N_Data_Length.

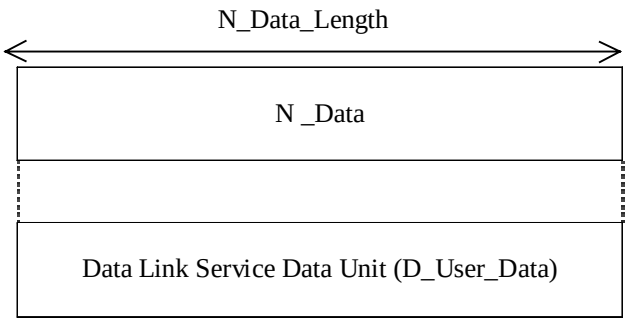

Figure 3-5: Network data field structure (UUDT)

3.2.3.4.2 Sequencing of interface services for single frame message transmission

Each network interface service interacts with the service user layer in a predefined way.

The request for transmission of a single frame message (N_UUData.request) shall be confirmed (N_UUData.confirmation) on the sending network entity.

A single frame message shall be delivered to the network service user (N_UUData.indication) on the receiving network entity.

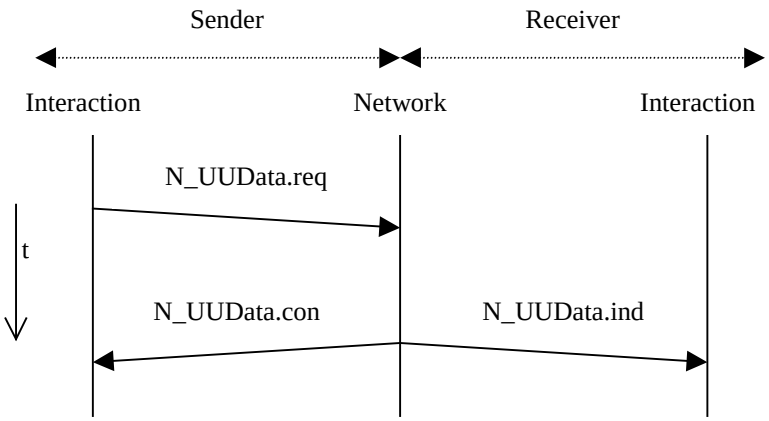

Figure 3-6: Single frame message transmission

3.2.3.5 Communication model

The transmission of a single frame message shall be supported by both 1:N and 1:1 communication models.

The communication model associated with a single frame message shall be defined at system generation time by specifying which network receiving entities shall receive a particular single frame message identified by its addressing information.

3.2.3.6 Mapping of the network layer to the Data link layer service

This chapter sets out the requirements for the mapping of the network protocol unacknowledged unsegmented data transfer (UUDT) parameters.

N_Handle is assigned to a specific D_Handle that shall be assigned to the corresponding data link layer service parameter, i.e. D_Handle.

If applicable, N_TA shall be assigned to D_TA service parameter.

N_Data shall be assigned to D_User_Data.

The following figure depicts the relationship between the network layer parameters and data link layer service parameters.

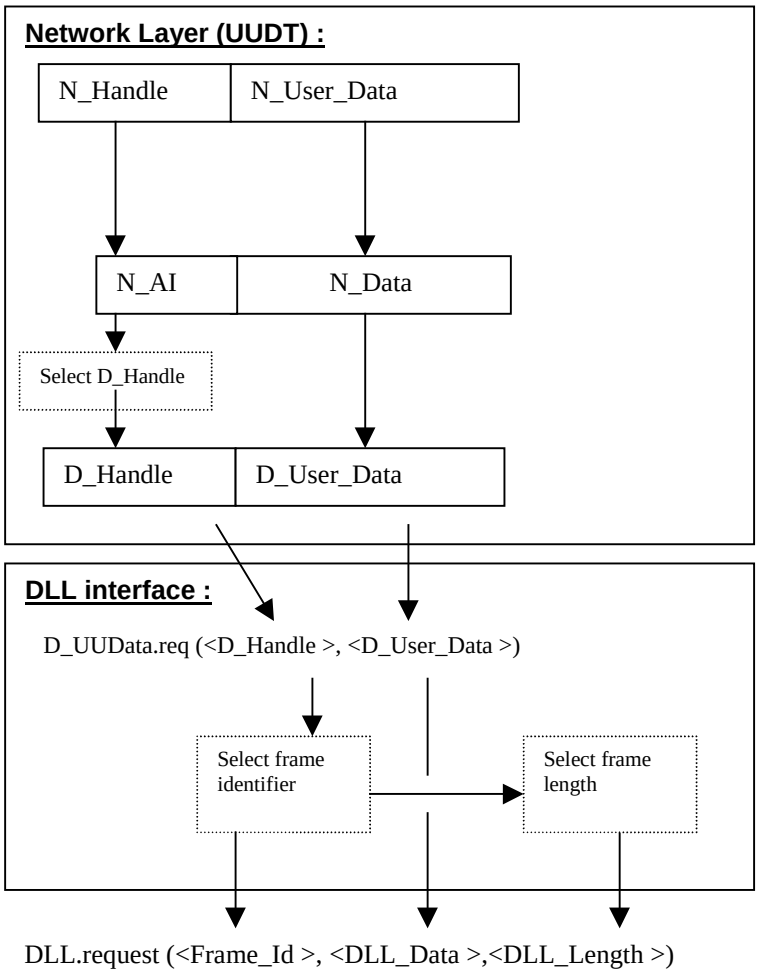

Figure 3-7: Mapping-out (UUDT)

3.2.3.7 Mapping of Data link service data units to network protocol data units

D_Handle is assigned to a specific N_Handle that shall be assigned to N_AI field of the network protocol data unit (NPDU).

If applicable, D_SA shall be mapped to N_SA.

D_User_Data shall be assigned to N_ Data.

D_Result_UUDT shall be assigned to N_Result_UUDT.

The following figure depicts the relationship between the data link layer service parameters and the network layer parameters.

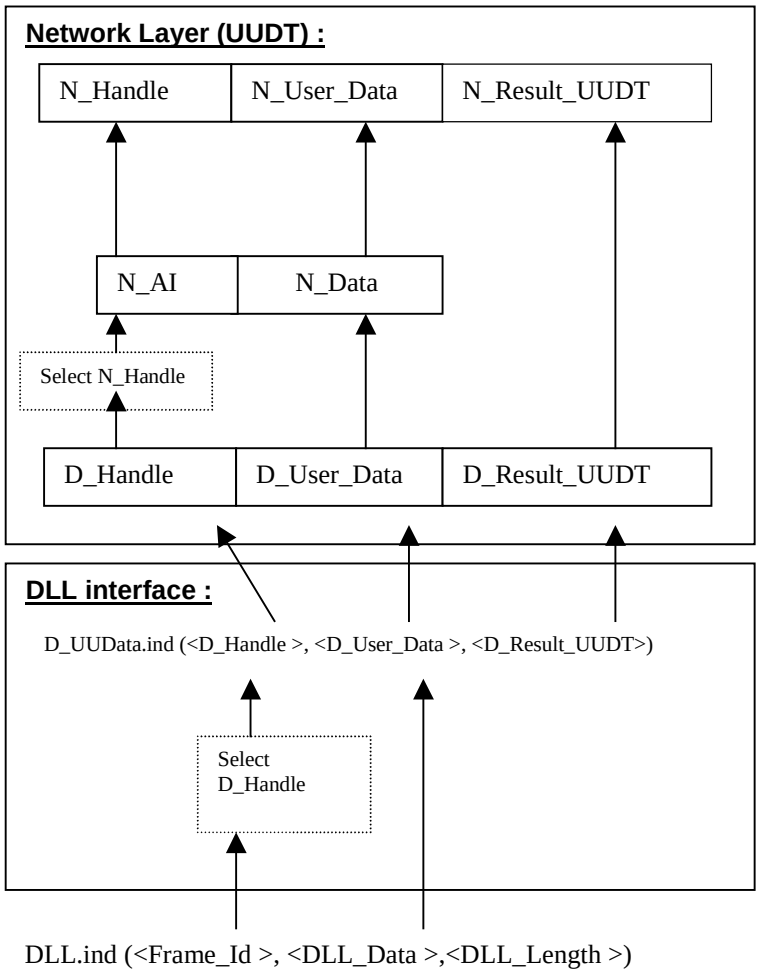

Figure 3-8: Mapping-in (UUDT)

3.2.4 Unacknowledged Segmented Data Transfer

3.2.4.1 Service data units

3.2.4.1.1 <N_Handle>

Type :

scalar

Range :

Application specific.

Description :

A N_Handle identifies a specific network layer message. A N_Handle is mono-directional, it shall either support the transmission or the reception of a network layer message. A N_Handle is associated with exactly one message of the interaction layer.

A N_Handle shall be mapped to two distinct and mono-directional D_Handle's of the

data link layer interface. A D_Handle shall be used to support the transfer of Single Frame (SF_NPDU), First Frame (FF_NPDU) and Consecutive Frame (CF_NPDU) network protocol data units. A second D_Handle shall be used to support the transfer of Flow Control (FC_NPDU) network protocol data units.

The picture below illustrates the relationship between a N_Handle with the interaction layer and the data link layer interface for a message that is to be transmitted. A message of the interaction layer identified by a SymbolicName (i.e. SymbolicName_*1*) is associated with a mono-directional N_Handle (N_Handle_*1*) that support the transmission of that message relying on two mono-directional data link handles (D_Handle_*11* and D_Handle_*12*).

D_Handle_*11* is associated with a bus frame identifier in order to support the transmission of Single Frame (SF_NPDU), First Frame (FF_NPDU) and Consecutive Frame (CF_NPDU) network protocol data units.

D_Handle_*12* is associated with a bus frame identifier in order to support the reception of Flow Control network protocol data units Frame (FC_NPDU).

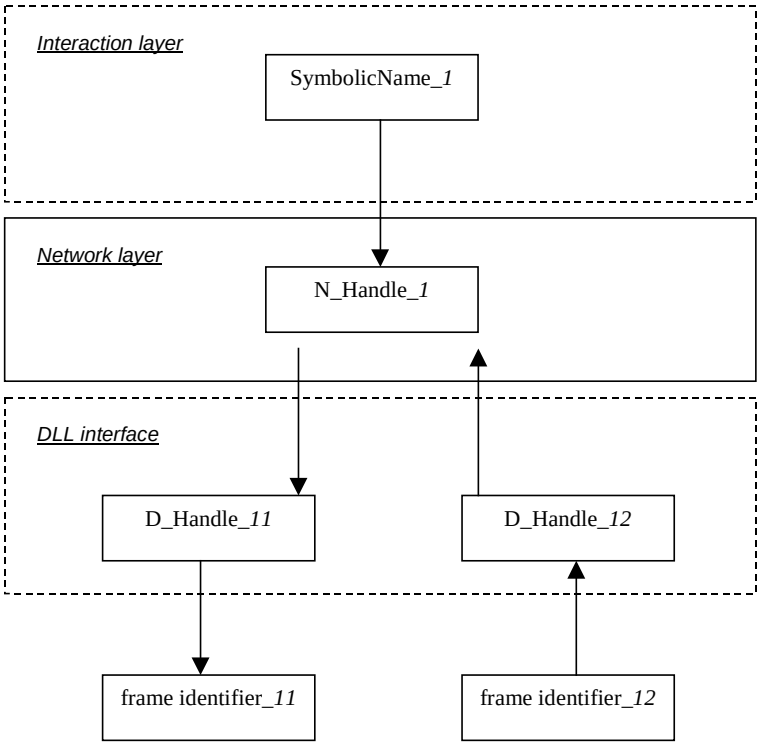

Figure 3-9: N_Handle (USDT)

3.2.4.1.2 <N_TA>

Type :

scalar

Range :

(Decimal) : 0-255

Description :

The N_TA service parameter is optional and applicable as defined in the interaction layer. If used, this service data unit shall be assigned to the D_TA service data unit of the data link layer.

3.2.4.1.3 <N_SA>

Type :

scalar

Range :

(Decimal) : 0-255

Description :

The N_SA service parameter is optional and applicable as defined in the interaction layer. If used, the D_TA service data unit of the data link layer shall be assigned to the N_SA service data unit.

3.2.4.1.4 <N_User_Data>

Type :

string of bytes

Range :

(Decimal) : Up to four thousand ninety five (4095) bytes maximum

Description :

N_User_Data is the network user data that shall be transferred by the sending network entity to the receiving network entity. This parameter includes all data the higher layer entities exchange.

3.2.4.2 Interface control information

3.2.4.2.1 <N_Length>

Type :

scalar

Range :

(Decimal) : 0-4095

Description :

This parameter indicates the length of the network user data (N_User_Data) to be transmitted/received by the network layer.

3.2.4.2.2 <N_BS_Value>

Type :

scalar

Range :

(Decimal) : 0-255

Description :

The N_BS_Value service parameter contains the value of the BS (block size) parameter that defines the number of consecutive frame network protocol data unit (CF_NPDU) that can be sent in a row.

This parameter value shall be assigned to the internal network layer parameter BS (block size) by means of the service N_ChangeParameter.request.

3.2.4.2.3 <N_ST_Value>

Type :

scalar

Range :

(Decimal) : 0-255

Description :

The N_ST_Value service parameter value shall be assigned to the internal network layer parameter STmin (Separation time) by means of the service N_ChangeParameter.request.

3.2.4.2.4 <N_Result_USDT>

Type :

enumeration

Range :

N_OK ,

N_TIMEOUT_A,

N_TIMEOUT_Bs,

N_TIMEOUT_Cr,

N_WRONG_SN ,

N_UNEXP_NPDU,

N_WFT_OVRN

Description :

This parameter contains the status relating to the outcome of the execution of the interface service N_USData.request and is provide to the network service user by N_USData.indication.

N_OK : this parameter shall be issued to the protocol user immediately after successful completion of the associated service. This parameter can be issued to a service user on both the sender and the receiver side. If a single frame message has been received then the only permitted value for N_Result_USDT shall be N_OK.

N_TIMEOUT_A : this parameter shall be issued to the service user upon occurrence of the N_Ar_max or N_As_max time-out's. This parameter can be issued to service user on both the sender and receiver side.

N_TIMEOUT_Bs : this parameter shall be issued to the service user upon occurrence of the N_Bs_max time-out. This parameter shall be issued to the service user on the sender side only.

N_TIMEOUT_Cr : this parameter shall be issued to the service user upon occurrence of the timer N_Cr_max time-out. This parameter shall be issued to the service user on the receiver side only.

N_WRONG_SN : this parameter shall be issued to the service user upon reception of an unexpected sequence number (NPCI.SN) value. This parameter shall be issued to the service user on the receiver side only.

N_UNEXP_NPDU : this parameter shall be issued to the service user upon reception of an unexpected protocol data unit. This parameter can be issued to the service user on both the sender and receiver side.

N_WFT_OVRN : this parameter shall be issued to the service user upon reception of a Flow Control network protocol data unit Wait (FC_NPDU_WT) that exceeds the maximum number of permitted reception of Flow Control network protocol data unit Wait (FC_NPDU_WT) in a row (N_WFTmax).

3.2.4.2.5 <N_Result_ChangeParameter>

Type :

enumeration

Range :

N_OK ,

N_RX_ON,

N_WRONG_PARAMETER,

N_WRONG_VALUE

Description :

This parameter contains the status relating to the outcome of the execution of the service N_ChangeParameter.request.

N_OK : this parameter means that the service execution has completed successfully.

This parameter shall be issued to the service user on both the receiver and sender side.

N_RX_ON : this parameter shall be issued to the service user to indicate that the service did not execute since a reception of the network layer message identified by <N_Handle> was taking place. This parameter can be issued to the service user on the receiver side only.

N_WRONG_VALUE : this parameter shall be issued to the service user to indicate that the service did not execute due to an out of range <N_ST_Value> or <N_BS_Value>. This parameter can be issued to the service user on both the receiver and sender side.

3.2.4.3 Interface services

3.2.4.3.1 N_USData.request

The service primitive requests transmission of <N_User_Data> identified by <N_Handle> with <N_Length> bytes.

N_USData.request (<N_Handle> <N_TA> <N_User_Data> <N_Length>)

3.2.4.3.2 N_USData.confirmation

The service primitive confirms the completion (successful or not) of a N_USData.request service for a specific <N_Handle>.

The parameter <N_Result_USDT> provides the status of the executed service request.

N_USData.confirmation (<N_Handle> <N_TA> <N_Result_USDT>)

3.2.4.3.3 N_USData.indication

The service primitive delivers the network user data <N_User_Data>, with length <N_Length>, identified by <N_Handle>.

The parameter <N_Result_USDT> provides the status of the service indication.

N_USData.indication (
<N_Handle>
<N_SA>
<N_User_Data>
<N_Length>
<N_Result_USDT>
)

3.2.4.3.4 N_USData_FF.indication

The service primitive indicates arrival of a First Frame network protocol data unit (FF_NPDU) of a multiple frame message identified by <N_Handle>.

This indication shall be performed upon reception of the First Frame network protocol data unit (FF_NPDU) of a multiple frame message.

N_USData_FF.indication (
<N_Handle>
<N_SA>
<N_Length>
)

3.2.4.3.5 N_ChangeParameter.request

The service primitive requests the change of the values of specific internal parameters of the local network protocol layer entity.

In case of reception, a change shall be possible before the indication of arrival of the First Frame network protocol data unit (FF_NPDU) and after the indication of the reception of a complete network layer message (N_User_Data) indicated by <N_USData.indication>

N_ChangeParameter.request (<N_Handle> <N_BS_Value> <N_ST_Value>)

3.2.4.3.6 N_ChangeParameter.confirmation

The service primitive confirms the completion (successful or not) of a N_ChangeParameter.request service applying to a network layer message identified by <N_Handle>.

N_ChangeParameter.confirmation (<N_Handle> <N_Result_ChangeParameter>)

3.2.4.3.7 Sequencing of interface services for multiple frame message transmission

Each network interface service interacts with the service user layer in a predefined way.

The request for transmission of a multiple frame message (N_USData.request) shall be confirmed (N_USData.confirmation) on the sending network entity.

The network service user shall be indicated upon arrival of a First Frame network protocol data unit (FF_NPDU) by means of the service N_USData_FF.indication on the receiving network entity. If a N_USData_FF.indication has been issued then the network service user shall expect a subsequent N_USData.indication belonging to the same multiple frame message.

A multiple frame message shall be delivered to the network service user (N_USData.indication) on the receiving network entity after the entire multiple frame message has arrived or upon time-out of N_Ar or N_Cr.

If a N_USData_FF.indication has been issued, then only one N_USData.indication shall be sent to the network service user regardless if the reception has terminated successfully or not. This N_USData.indication means that the reception has terminated with a status as reported in N_Result_USDT.

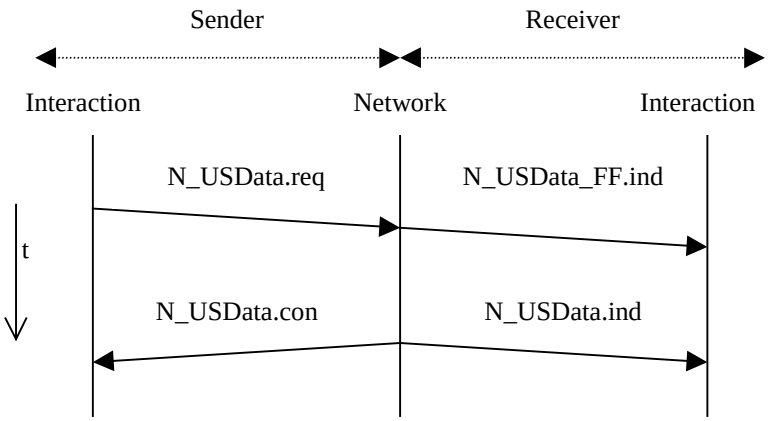

Figure 3-10: Multiple frame message transmission

3.2.4.3.8 Sequencing of interface services for single frame message transmission

Each network interface service interacts with the service user layer in a predefined way.

The request for transmission of an single frame message (N_USData.request) shall be confirmed (N_USData.confirmation) on the sending network entity.

An single frame message shall be delivered to the network service user (N_USData.indication) on the receiving network entity.

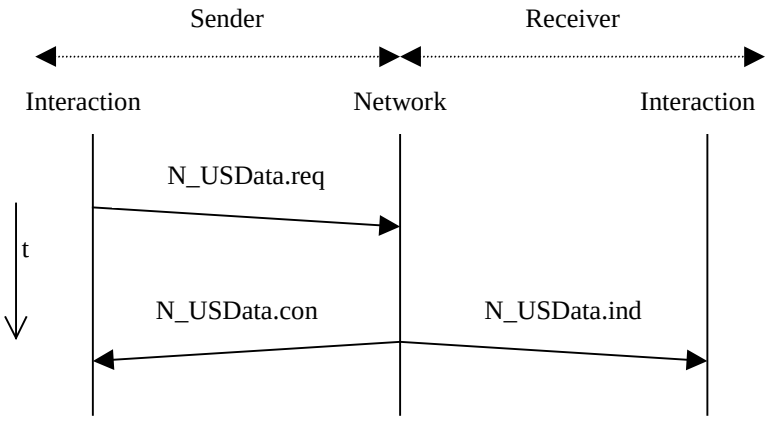

Figure 3-11: Single frame message transmission

3.2.4.4 Communication model

The transmission of a multiple frame message shall be supported by the 1:1 communication model only.

The transmission of a single frame message can be supported by both 1:N and 1:1 communication models.

The communication model associated with a single frame message shall be defined at system generation time by specifying which network receiving entities shall receive a particular single frame message identified by its addressing information.

3.2.4.5 Protocol Data units

The network protocol data unit (NPDU) consists of various fields that enable each protocol entity to operate in conformance with a set of communication rules as defined in this document.

3.2.4.5.1 Protocol data unit fields

All network protocol data unit (NPDU) consist of three (3) fields:

Table 3-1: NPDU format

Address Information	Protocol Control Information	Data Field
N_AI	NPCI	N_Data

3.2.4.5.1.1 Network addressing information

The network addressing information (N_AI) field contains the N_Handle, N_TA (optional) and N_SA (optional) of a multiple or single frame message.

3.2.4.5.1.2 Network Protocol Control Information

The network protocol control information (NPCI) identifies the type of network protocol data unit (NPDU) exchanged. This field is exchanged between sending/receiving network entities to co-ordinate their joint operation.

The following network protocol control information (NPCI) shall be supported:

1. Single Frame protocol control information (SF_NPCI)
2. First Frame protocol control information (FF_NPCI)
3. Consecutive Frame protocol control information (CF_NPCI)
4. Flow Control protocol information (FC_NPCI)

3.2.4.5.1.3 *Network data field*

The network data field (N_Data) is used to transmit the service user data provided by the network user data (N_User_Data) parameter of the N_USData.request service call. The network user data (N_User_Data), if needed, is segmented into smaller parts that each fit into the network data field (N_Data) before they are transmitted over the network.

The length of the network data field (N_Data) is depending on the type of network protocol data unit and the length of the underlying data link layer protocol data unit.

The length of the network data field (N_Data) shall be user defined at system generation time. The length of the network data field (N_Data) and all NCPI bytes , i.e. three (3) bytes, shall be referred to as N_Data_Length.

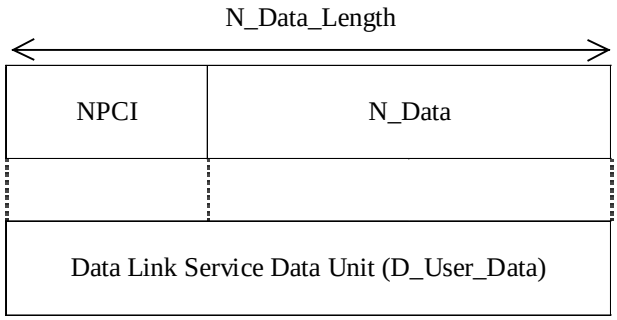

Figure 3-12: N_Data and NPCI fields to data link user data

3.2.4.5.2 Protocol data units specification

3.2.4.5.2.1 *Single Frame*

The Single Frame network protocol data unit (SF_NPDU) is identified by the Single Frame network protocol control information (SF_NPCI).

The Single Frame (SF_NPDU) consists of : a network addressing field (N_AI), network protocol control information (NPCI) field and a network data field (N_Data).

The Single Frame (SF_NPDU) shall be sent by the sending network entity and shall be received by one or multiple receiving network entities.

The Single Frame (SF_NPDU) shall be sent out to transfer a service data unit that can be transferred via a single service request to the data link layer (D_UUData.request). Single Frame's shall be sent to transfer single frame message only.

3.2.4.5.2.2 *First Frame*

The First Frame network protocol data unit (FF_NPDU) is identified by the First Frame network protocol control information (FF_NPCI).

The First Frame (FF_NPDU) consists of : a network addressing field (N_AI), network protocol control information (NPCI) field and a network data field (N_Data).

The First Frame (FF_NPDU) shall be sent by the sending network entity and received by a unique receiving network entity for the duration of the multiple frame message transmission.

The First Frame (FF_NPDU) identifies the first network protocol data unit (NPDU) of a multiple frame message transmitted by a sending network entity and received by a receiving network entity.

The receiving network entity shall start assembling the multiple frame message on receipt of a First Frame (FF_NPDU).

3.2.4.5.2.3 *Consecutive Frame*

The Consecutive Frame network protocol data unit (CF_NPDU) is identified by the Consecutive Frame network protocol control information (CF_NPCI).

The Consecutive Frame (CF_NPDU) transfers segments (N_Data) of the network user data (N_User_Data).

The Consecutive Frame (CF_NPDU) consists of : a network addressing field (N_AI), network protocol control information (NPCI) field and a network data field (N_Data).

The Consecutive Frame (CF_NPDU) shall be sent after reception of a Flow Control Clear To Send (FC_NPDU_CTS).

All network protocol data units (NPDU's) transmitted after the First Frame (FF_NPDU) shall be encoded as Consecutive Frames (CF_NPDU's).

The receiving entity shall pass the assembled network layer message to the service user of the receiving network entity after the last Consecutive Frame (CF_NPDU) has been received.

The Consecutive Frame (CF_NPDU) shall be sent by the sending network entity and received by a unique receiving network entity for the duration of the multiple frame message transmission.

3.2.4.5.2.4 Flow Control Clear To Send

The Flow Control network protocol data unit Clear To Send (FC_NPDU_CTS) is identified by the Flow Control (FlowStatus equal to Clear To Send) network protocol control information (FC_NPCI).

The Flow Control Clear To Send (FC_NPDU_CTS) consists of : a network addressing field (N_AI) and network protocol control information (NPCI) field.

The Flow Control Clear To Send (FC_NPDU_CTS) instructs a sending network entity to resume transmission of CF_NPDU's.

The Flow Control Clear To Send (FC_NPDU_CTS) shall be sent by the receiving network entity and shall be received by a unique sending network entity for the duration of the multiple frame message transmission.

The Flow Control Clear To Send (FC_NPDU_CTS) shall be sent by the receiving network entity to the sending network entity after correct reception of :

1. a First Frame (FF_NPDU)
2. the last Consecutive Frame (CF_NPDU) of a block of Consecutive Frames (CF_NPDU) if further Consecutive Frame (CF_NPDU) need(s) to be sent.

The sending network entity shall take into account the flow control parameters, i.e. STmin and BS, provided by the Flow Control Clear To Send (FC_NPDU_CTS) that follows the transmission of a First Frame (FF_NPDU) only. Flow control parameters provided by Flow Control Clear To Send (FC_NPDU_CTS) received after transmission of Consecutive Frame's (CF_NPDU's) shall be ignored by the sending network entity for the duration of the network message transmission.

3.2.4.5.2.5 Flow Control Wait

The Flow Control network protocol data unit Wait (FC_NPDU_WT) is identified by the Flow Control (FlowStatus equal to Wait) network protocol control information (FC_NPCI).

The Flow Control Wait (FC_NPDU_WT) consists of : a network addressing field (N_AI) and network protocol control information (NPCI) field.

The Flow Control Wait (FC_NPDU_WT) instructs a sending network entity to pause transmission of Consecutive Frame's (CF_NPDU's).

The Flow Control Wait (FC_NPDU_WT) shall be sent by the receiving network entity and shall be received by a unique sending network entity for the duration of the multiple frame message transmission.

The Flow Control Wait (FC_NPDU_WT) may be sent by the receiving network entity to the sending network entity after correct reception of :

1. a First Frame (FF_NPDU)
2. the last Consecutive Frame (CF_NPDU) of a block of Consecutive Frames (CF_NPDU) if further Consecutive Frame (CF_NPDU) need(s) to be sent in a new block.

3.2.4.5.2.5.1 Maximum number of Flow Control Wait network protocol data unit

The N_WFTmax parameter shall indicate how many Flow Control Wait (FC_NPDU_WT) can be transmitted by the receiving network entity in a row.

The N_WFTmax parameter upper limit shall be user defined at system generation time.

The N_WFTmax parameter shall only be used on the receiving network entity during message reception.

If N_WFTmax parameter value is set to zero (0) then flow control shall rely upon flow control clear to send (FC_NPDU_CTS) only. Flow Control Wait (FC_NPDU_WT) shall not be supported.

A N_USData.indication with N_Result_USDT set to N_WFT_OVRN shall be issued to the service user of the receiving network entity upon reception of a flow Control Wait (FC_NPDU_WT) that exceeds the maximum counter N_WFTmax.

3.2.4.6 Protocol Control Information

Each network protocol data unit (NPDU) is identified by means of a Network Protocol Control Information (NPCI) as illustrated below :

Table 3-2: Encoding of Network Protocol Control Information (NPCI) bytes

USDT NPCI encoding		**Network Protocol Control Information (NPCI)**									
		Byte #1								**Byte #2**	**Byte #3**
NPCI name	**Mnemonic**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
SingleFrame	SF_NPCI	0	0	0	0	SF_DL				N/A	N/A
FirstFrame	FF_NPCI	0	0	0	1	FF_DL					N/A
ConsecutiveFrame	CF_NPCI	0	0	1	0	SN				N/A	N/A
FlowControl	FC_NPCI	0	0	1	1	FS				BS	STmin
Reserved		$40 through $FF								reserved	reserved

3.2.4.6.1 Single Frame

The Single Frame protocol control information (SF_NPCI) shall be encoded by setting the upper nibble bits of SF_NPCI byte #1 to zero (0).

3.2.4.6.1.1 Single Frame .DataLength

Table 3-3: Definition of Network Protocol Control Information (NPCI) : SF_NPCI . DL

USDT NPCI encoding		Network Protocol Control Information (NPCI)									
		Byte #1								Byte #2	Byte #3
NPCI name	**Mnemonic**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
SingleFrame	SF_NPCI	0	0	0	0	**SF_DL**				N/A	N/A

The single frame message length is encoded out of the NPCI byte #1 low nibble value.

The single frame message length field (SF_DL) shall be assigned the value of the service parameter N_Length.

3.2.4.6.2 First Frame

The First Frame protocol control information (FF_NPCI) shall be encoded by setting the upper nibble bits of FF_NPCI byte #1 to zero (0) except the bit four (4) that shall be set to one (1).

3.2.4.6.2.1 First Frame .DataLength

Table 3-4: Definition of Network Protocol Control Information (NPCI) : FF_NPCI . DL

USDT NPCI encoding		Network Protocol Control Information (NPCI)									
		Byte #1								Byte #2	Byte #3
NPCI name	**Mnemonic**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
FirstFrame	FF_NPCI	0	0	0	1	**FF_DL**					N/A

The encoding of the multiple frame message length (N_Length) results into a twelve (12) bit length value where the least significant bit (LSB) is specified to be bit zero (0) of the NPCI byte #2 and the most significant bit (MSB) is bit three (3) of the NPCI byte #1 byte.

The maximum multiple frame message length supported is equal to four thousand and ninety five (4095) bytes of user data.

The multiple frame message length field (FF_DL) shall be assigned the value of the service parameter N_Length.

3.2.4.6.3 Flow Control

The purpose of Flow Control is to regulate the rate at which Consecutive Frame (CF_NPDU) are sent to the receiver. Two distinct types of Flow Control protocol data units are specified to support this function. The type is indicated by a field of the protocol control information called Flow Status (FS) as defined hereafter.

3.2.4.6.3.1 Flow Control .FlowStatus

Table 3-5: Definition of Network Protocol Control Information (NPCI) : FC_NPCI . FS

USDT NPCI encoding		Network Protocol Control Information (NPCI)									
		Byte #1								Byte #2	Byte #3
NPCI name	**Mnemonic**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
Flow Control	FC_NPCI	0	0	1	1	**FS**				BS	STmin

The Flow Status (FS) indicates whether the sending network entity can proceed the multiple frame message transmission.

The Flow Status (FS) shall be encoded in the low nibble of the NPCI-byte #1 byte of the Flow Control network protocol control information (FC_NPCI).

3.2.4.6.3.1.1 Flow Control .FlowStatus (ClearToSend)

Table 3-6: Definition of Network Protocol Control Information (NPCI) : FC_NPCI . FS(CTS)

USDT NPCI encoding		Network Protocol Control Information (NPCI)									
		Byte #1								Byte #2	Byte #3
NPCI name	**Mnemonic**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
Flow Control	FC_NPCI	0	0	1	1	**0**	**0**	**0**	**0**	BS	STmin

The Flow Control Clear To Send (FC_NPDU_CTS) shall be encoded by setting all bits of the FC_NPCI byte #1 to zero (0) except the bit five (5) and four (4) that shall be set to one (1).

The Flow control parameters BS and STmin shall be taken into account by the sending network entity upon reception of the Flow Control Clear To Send (FC_NPDU_CTS) that follows the First Frame (FF_NPCU) only.

3.2.4.6.3.1.2 Flow Control .FlowStatus (Wait)

Table 3-7: Definition of Network Protocol Control Information (NPCI) : FC_NPCI . FS(WT)

USDT NPCI encoding		Network Protocol Control Information (NPCI)									
		Byte #1								Byte #2	Byte #3
NPCI name	**Mnemonic**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
Flow Control	FC_NPCI	0	0	1	1	**0**	**0**	**0**	**1**	BS	STmin

The Flow Control Wait (FC_NPDU_WT) shall be encoded by setting all bits of the FC_NPCI byte #1 to "0" except the bit five (5), bit four (4) and bit zero (0) that shall be set to one (1).

The sending network entity shall not send further Consecutive Frame (CF_NPDU)) upon reception of a Flow Control Wait (FC_NPDU_WT) and until reception of a Flow Control Clear To Send (FC_NPDU_CTS).

3.2.4.6.3.2 Flow Control .BlockSize

Table 3-8: Definition of Network Protocol Control Information (NPCI) : FC_NPCI . BS

USDT NPCI encoding		Network Protocol Control Information (NPCI)									
		Byte #1								Byte #2	Byte #3
NPCI name	**Mnemonic**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
Flow Control	FC_NPCI	0	0	1	1	FS				**BS**	STmin

The Block Size (BS) parameter shall be encoded in byte#2 of the Flow Control network protocol information (FC_NPCI).

The Block Size (BS) parameter specifies transmission of BlockSize (BS) number of Consecutive Frame s (CF_NPDU's) in a row.

The Block Size (BS) parameter shall be within the range of zero (0) to two hundred fifty five (255).

The units of Block Size (BS) are absolute number of Consecutive Frame s (CF_NPDU's) per block : e.g. if BS is equal to twenty (20) (decimal) then the block shall consists of twenty (20) (decimal) Consecutive Frame's (CF_NPDU's).

If the Block Size (BS) parameter value is set to zero (0) then transmission of a single Flow Control Clear To Send (FC_NPDU_CTS) is permitted after reception of a First Frame (FF_NPDU) for the duration of the multiple frame message transfer. No further flow control using Flow Control protocol data unit (FC_NPDU_CTS or FC_NPDU_WT) shall be performed during the transmission of subsequent Consecutive Frame (CF_NPDU's) for the duration of the multiple frame message transmission.

If the Block Size (BS) parameter value is set to a value within the range of one (1) to two hundred fifty five (255) then flow control shall be performed accordingly. This range of

values shall be used to indicate to the sender the maximum number of Consecutive Frame's (CF_NPDU's) that can be received without an intermediate Flow Control (FC_NPDU) by the receiving network entity.

The Block Size (BS) shall be defined at system generation time on the receiving network entity side.

3.2.4.6.3.3 Flow Control .SeparationTime

Table 3-9: Definition of Network Protocol Control Information (NPCI) : FC_NPCI . STmin

USDT NPCI encoding		**Network Protocol Control Information (NPCI)**									
		Byte #1								**Byte #2**	**Byte #3**
NPCI name	**Mnemonic**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
Flow Control	FC_NPCI	0	0	1	1	FS				BS	**STmin**

The Separation Time (STmin) parameter shall be encoded in byte#3 of the Flow Control network protocol control information (FC_NPCI).

The SeparationTime (STmin) specifies the minimum time gap allowed between the reception of Consecutive Frame s (CF_NPDU's).

The SeparationTime (STmin) shall be within the range of zero (0) to two hundred fifty five (255).

This time is specified by the receiving entity and shall be kept by the sending network entity for the duration of a multiple frame message transmission.

The units of the Separation Time (STmin) are absolute milliseconds (ms) : e.g. if STmin is equal to 10 (decimal) then the minimum Separation Time authorised between Consecutive Frame s (CF_NPDU's) is equal to ten (10) milliseconds.

The measurement of the Separation Timer (Stmin) starts after completion of transmission of a Consecutive Frame (CF_NPDU) and ends at the request for the transmission of the next Consecutive Frame (CF_NPDU).

The Separation Time (STmin) shall be defined at system generation time on the receiving network entity side.

3.2.4.6.4 Consecutive Frame

The Consecutive Frame protocol control information (CF_NPCI) shall be encoded by setting all the upper nibble bits of the Consecutive Frame protocol control information (CF_NPCI) byte #1 to zero (0) except the bit five (5) that shall be set to one (1).

3.2.4.6.4.1 Consecutive Frame .SequenceNumber

Table 3-10: Definition of Network Protocol Control Information (NPCI) : CF_NPCI . SN

USDT NPCI encoding		Network Protocol Control Information (NPCI)									
		Byte #1								Byte #2	Byte #3
NPCI name	**Mnemonic**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
ConsecutiveFrame	CF_NPCI	0	0	1	0	**SN**				N/A	N/A

The SequenceNumber (SN) shall be encoded in the lower nibble bits of CF_NPCI byte #1.

The following rules apply to the SequenceNumber (SN):

1. The SequenceNumber (SN) shall start with zero (0) for all segmented messages. The First Frame shall be assigned the value zero (0). It does not include an explicit SequenceNumber in the N_PCI field but it shall be treated as the segment number zero (0).
2. The SequenceNumber (SN) of the first Consecutive Frame that immediately follows the First Frame shall be set to one (1).
3. The SequenceNumber (SN) shall be incremented by one (1) for each new Consecutive Frame (CF) that is transmitted during a multiple frame message transmission.
4. The SequenceNumber (SN) value shall not be affected by any Flow Control (FC_NPDU).
5. When the SequenceNumber (SN) reaches the value of fifteen (15), it shall wraparound and be set to zero (0) for the ConsecutiveFrame (CF).

This shall lead to the following sequence:

Table 3-11: Summary of SequenceNumber (SN) definition

NPDU	**FF_NPDU**	**CF_NPDU**	**CF_NPDU**	**CF_NPDU**	**CF_NPDU**	**CF_NPDU**	**CF_NPDU**	**CF_NPDU**
SN (hex)	0	1	…	E	F	0	1	…

3.2.4.7 Protocol layer timings

The following figure depicts the time intervals of the network layer specified in this section of the specification.

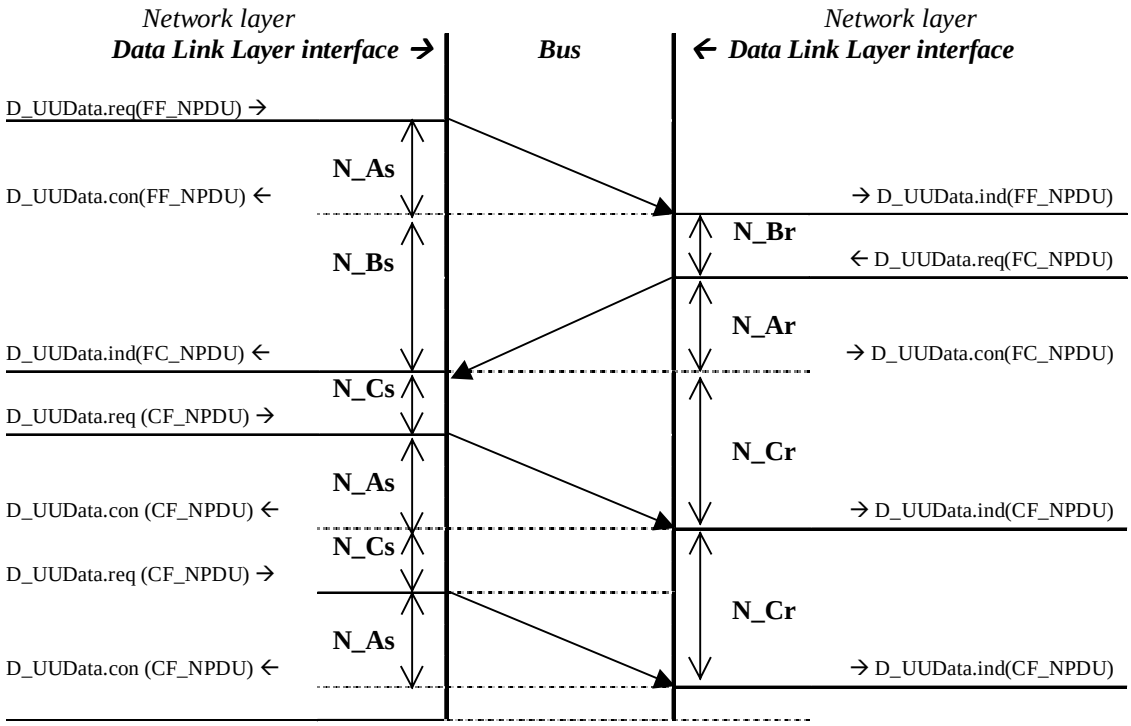

Figure 3-13: Placement of time intervals

3.2.4.7.1 N_As, N_As_max

N_As is the time to transmit a network protocol data unit (NPDU). N_As is defined on the sending network entity side between D_UUData.req(NPDU) and D_UUData.con(NPDU).

N_As_max is the maximum time allowed by a sending network entity for the transmission of a network protocol data unit (NPDU). For a successful transmission of network protocol data unit (NPDU) to occur it is necessary that N_As be smaller than N_As_max.

$$N\_As < N\_As\_\max$$

3.2.4.7.2 N_Ar, N_Ar_max

N_Ar is the time to transmit a network protocol data unit (NPDU). N_Ar is defined on the receiving network entity side between D_UUData.req(NPDU) and D_UUData.con(NPDU).

N_Ar_max is the maximum time allowed by a receiving network entity for the transmission of a network protocol data unit (NPDU). For a successful transmission of network protocol data unit (NPDU) to occur it is necessary that N_Ar be smaller than N_Ar_max.

$$N\_Ar < N\_Ar\_\max$$

3.2.4.7.3 N_Bs, N_Bs_max

N_Bs is the time to receive the next Flow Control (FC_NPDU). N_Bs is defined on the sending network entity side between D_UUData.con(FF_NPDU or CF_NPDU or FC_NPDU_WT) and D_UUData.ind(FC_NPDU).

N_Bs_max is the maximum time allowed by a sending network entity for the reception of the next Flow Control (FC_NPDU). For a successful reception of the next Flow Control (FC_NPDU) to occur it is necessary that N_Bs be smaller than N_Bs_max.

$$N\_Bs < N\_Bs\_\max$$

3.2.4.7.4 N_Br

N_Br is the time to transmit the next Flow Control (FC_NPDU). N_Br is defined on the receiving network entity side between D_UUData.ind(FF_NPDU or CF_NPDU or FC_NPDU_WT) and D_UUData.req (FC_NPDU). The sum of N_Br and N_Ar (FC_NPDU) times shall be smaller than N_Bs_max. It is recommended that the addition of N_Br and N_Ar(FC_NPDU) amounts at most to ninety per cent (90%) of N_Bs_max.

$$N\_Br + N\_Ar(FC\_NPDU) < N\_Bs\_\max * 0.9$$

3.2.4.7.5 N_Cs

N_Cs is the time to transmit the next Consecutive Frame (CF_NPDU). N_Cs is defined on the sending network entity side between either :

1. D_UUData.ind(FC_NPDU) and D_UUData.req(CF_NPDU)
2. D_UUData.con(CF_NPDU) and D_UUData.req(CF_NPDU)

The sum of N_Cs and N_As(CF_NPDU) shall be smaller than the corresponding N_Cr_max. It is recommended that the addition of N_Cs and N_As(CF_NPDU) amounts at most to ninety per cent (90%) of N_Cr_max.

$$N\_Cs + N\_As(CF\_NPDU) < N\_Cr\_\max * 0.9$$

3.2.4.7.6 N_Cr, N_Cr_max

N_Cr is the time to receive the next Consecutive Frame (CF_NPDU) after transmission of a Flow Control Clear To Send (FC_NPDU_CTS). N_Cr is defined on the receiving network entity side between either :

D_UUData.con(FC_NPDU) and D_UUData.ind(CF_NPDU)

D_UUData.ind(CF_NPDU) and D_UUData.ind(CF_NPDU)

N_Cr_max is the maximum time allowed by a receiving network entity for the reception of the next Consecutive Frame (CF_NPDU) after transmission of a Flow Control Clear To Send (FC_NPDU_CTS). For a successful reception of the next Consecutive Frame (CF_NPDU) to occur it is necessary that N_Cr be smaller than N_Cr_max.

$$N\_Cr < N\_Cr\_\max$$

3.2.4.7.7 Time intervals summary

The following table summarises the time intervals :

Table 3-12: Time intervals definition

Timing	Description	Data Link Layer service		Timeout	Performance
Parameter		**Start**	**End**	**(ms)**	**requirement (ms)**
N_As	Time for transmission of any NPDU on the sender side	D_UUDATAData.request	D_UUDATAData.confirm	N_As_max	N/A
N_Ar	Time for transmission of any NPDU on the receiver side	D_UUData.request	D_UUData.confirm	N_Ar_max	N/A
N_Bs	Time until reception of the next Flow Control.	D_UUData.confirm (FF), D_UUData.confirm (CF), D_UUData.indication (FC)	D_UUData.indication (FC)	N_Bs_max	N/A
N_Br	Time until transmission of the next Flow Control	D_UUData.indication (FF), D_UUData.confirm (FC), D_UUData.indication (CF)	D_UUData.request (FC)	N/A	(N_Br + N_Ar) < (0.9 * N_Bs_max)
N_Cs	Time until transmission of the next Consecutive Frame	D_UUData.indication (FC), D_UUData.confirm (CF)	D_UUData.request (CF)	N/A	(N_Cs + N_As) < (0.9 * N_Cr_max)
N_Cr	Time until reception of the next Consecutive Frame	D_UUData.confirm (FC), D_UUData.indication (CF)	D_UUData.indication (CF)	N_Cr_max	---

Note :

The small letters "s" and "r" are interpreted as follows : "s" = sender of the message and "r" = receiver of the message.

3.2.4.8 Wait frame handling

The following table defines conditions for the transmission of a Flow Control Wait protocol data unit by the receiving network entity, provided that N_WFTmax is greater than zero (0).

Table 3-13— Wait frame handling

Parameter	Conditions	Action
N_Br	N_Br has elapsed on the receiving network entity and the number of Flow Control transmitted in a row does not exceed N_WFTmax	Transmit Flow Control Wait protocol data unit

3.2.4.9 Network layer error handling

The network layer shall issue an appropriate service primitive to the network service user upon detection of an error condition. Each service primitive shall provide the N_Handle of the related network layer message and specific values to the N_Result_USDT parameters as defined in the service parameter section of this document.

3.2.4.9.1 Monitored time intervals expiry

This chapter specifies actions that shall be performed by the appropriate network entities upon expiry of the specific monitored time interval :

N_As_max :

N_USData.confirmation with N_Result_USDT equal to N_TIMEOUT_A shall be issued to the service user of the sending network entity upon occurrence of N_As_max time-out.

The sending network entity shall abort network layer message transmission upon occurrence of N_As_max time-out.

N_Ar_max :

N_USData.indication with N_Result_USDT set to N_TIMEOUT_A shall be issued to the service user of the receiving network entity upon occurrence of N_Ar_max time-out.

The receiving network entity shall abort the network layer message reception upon occurrence of N_Ar_max time-out.

N_Bs_max :

N_USData.confirmation with N_Result_USDT set to N_TIMEOUT_Bs shall be issued to the service user of the sending network entity upon occurrence of N_Bs_max time-out..

The sending network entity shall abort the network layer message transmission upon occurrence of N_Bs_max time-out.

N_Cr_max :

N_USData.indication with Result_USDT set to N_TIMEOUT_Cr shall be issued to the service user of the receiving network entity upon occurrence of N_Cr_max time-out.

The receiving network entity shall abort the network layer message reception upon occurrence of N_Cr_max time-out.

The following table summarises the above requirements :

Table 3-14: Error handling

Timeout	Cause	Action
N_As_max	Any NPDU not transmitted on time on the sender side	Abort network layer message transmission and issue N_USData.con (N_TIMEOUT_A)
N_Ar_max	Any NPDU not transmitted on time on the receiver side	Abort network layer message reception and issue N_USData.ind (N_TIMEOUT_A)
N_Bs_max	Flow Control lost (overwritten) on the sender side or preceding First Frame or Consecutive Frame lost (overwritten) on the receiver side.	Abort network layer message transmission and issue N_USData.con (N_TIMEOUT_Bs)
N_Cr_max	Consecutive Frane lost (overwritten) on the receiver side or preceding Flow Control lost (overwritten) on the sender side.	Abort network layer message reception and issue N_USData.ind (N_TIMEOUT_Cr)

3.2.4.9.2 Unexpected arrival of network protocol data unit

Depending on the network layer design decision to support full- or half- duplex communication, the interpretation of “unexpected” differs.

- Half-duplex : 1:1 communication is only possible in one direction at a time.
- Full-duplex : 1:1 communication is possible in both directions at a time.

In addition to the network layer design decision it has to be considered if a reception or transmission from and to a node, with the same address information (N_AI) as contained in the received unexpected NPDU, is in progress.

As a general rule, arrival of an unexpected network protocol data unit from any network entity shall be ignored, with the exception of single frames (SF N_PDU) and first frames (FF N_PDU).

The table below defines the network layer behaviour in case of the reception of unexpected frames, in consideration of the actual network layer internal status (NWL status) and the design decision to support half- or full-duplex communication. It has to be understood, that the received NPDU contains the same address information (N_AI) as the reception or transmission which may be in progress at the time the N_PDU is received.

Table 3-15— Handling of an unexpected arrival of a network layer NPDU

NWL status	Reception of SF_NPDU	FF_NPDU	CF_NPDU	FC_NPDU	Unknown NPDU
Transmit in progress	Full-duplex: Process the SF_NPDU as the start of a new reception	Full-duplex: Process the FF_N_PDU as the start of a new reception	Full-duplex: If a reception is in progress, see corresponding cell below in this table	If awaited, process the FC_N_PDU, otherwise ignore it.	Ignore
	Half-duplex: ignore	Half-duplex: ignore	Half-duplex: ignore		
Receive in progress	Terminate the current reception, report an N_USData.indication, with <N_Result_USDT> set to N_UNEXP_PDU, to the upper layer, and process the SF_NPDU as the start of a new reception	Terminate the current reception, report an N_USData.indication, with <N_Result_USDT> set to N_UNEXP_PDU, to the upper layer, and process the FF_NPDU as the start of a new reception	Process the CF_ NPDU in the on-going reception and perform the required checks (e.g. SN in right order)	Full-duplex: If a transmission is in progress, see corresponding cell above in this table	Ignore
				Half-duplex: Ignore	
Idle	Process the SF_NPDU as the start of a new reception	Process the FF_NPDU as the start of a new reception	Ignore	Ignore	Ignore

Note :

"Idle" means, that neither a transmission nor reception is in progress.

3.2.4.10 Wait frame error handling

The receiving network entity shall abort the message reception and issue to the higher layer a N_USData.indication with <N_Result_USDT> set to N_WFT_OVRN upon the (N_WFTmax + 1) occurrence in a row of N_Br_max.

The sending network entity shall consequently issue to the higher layer a N_USData.confirm with <N_Result_USDT> set to N_TIMEOUT_Bs due to the missing Flow Control Wait network protocol data unit.

3.2.4.11 Interleaving of network layer messages

The USDT protocol shall be capable to carry out parallel transmission of different network layer messages that are not mapped onto the same N_Handle.

This is necessary to ensure that the receiving peer is able to reassemble in a consistent manner the received network protocol data units. This scheme e.g. enables gateway operation that needs to handle different network layer message transmissions concurrently across distinct sub-networks.

3.2.4.12 Mapping of network layer to Data link layer protocol data units

This chapter sets out the requirements for the mapping of the network protocol unacknowledged segmented data transfer (USDT) parameters.

The N_Handle shall be utilised to select a specific D_Handle that shall be assigned to the corresponding service parameter.

The N_TA (if applicable) shall be assigned to the D_TA service parameter.

The network protocol control information (NPCI) and network Data field (D_Data) shall be assigned to the Data link user data (D_User_Data).

The following figure depicts the relationship between the network layer parameters and data link layer service parameters.

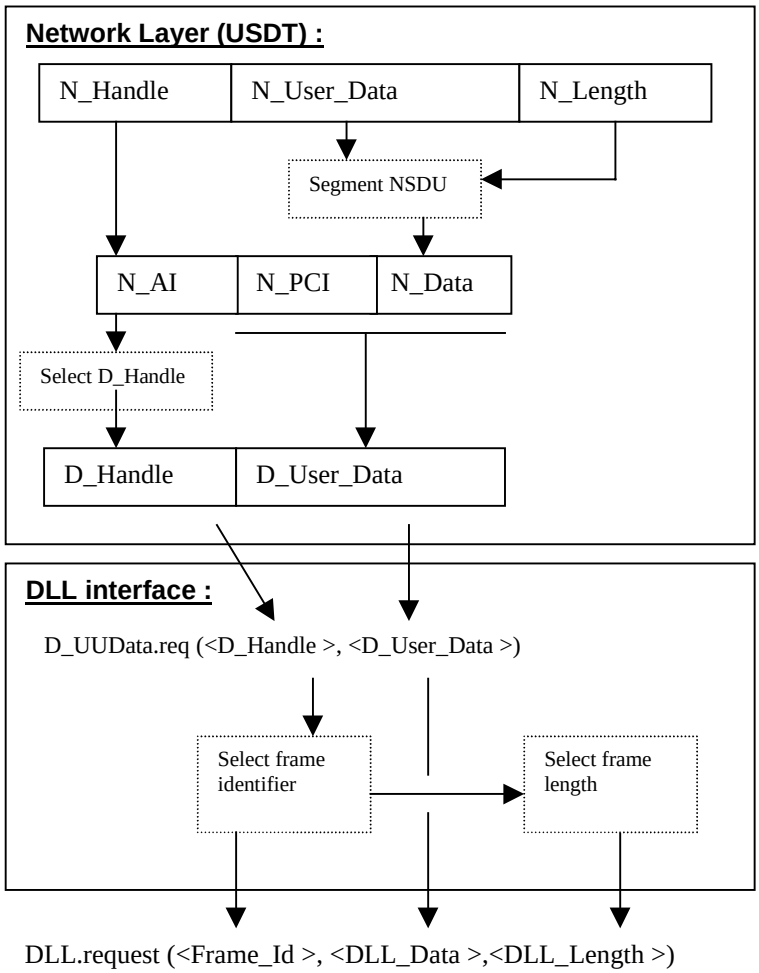

Figure 3-14: Mapping-out (USDT)

3.2.4.13 Mapping of data link service data units to network protocol data units

D_Handle is utilised to select a specific N_Handle that shall be assigned to the N_AI field of the network protocol data unit (NPDU).

D_SA (if applicable).shall be mapped to N_SA.

D_User_Data shall be re-assembled and delivered to the network service user N_User_Data.

D_Result_UUDT shall be processed by the network layer ultimately leading to N_Result_USDT.

The following figure depicts the relationship between the data link layer service parameters and the network layer parameters.

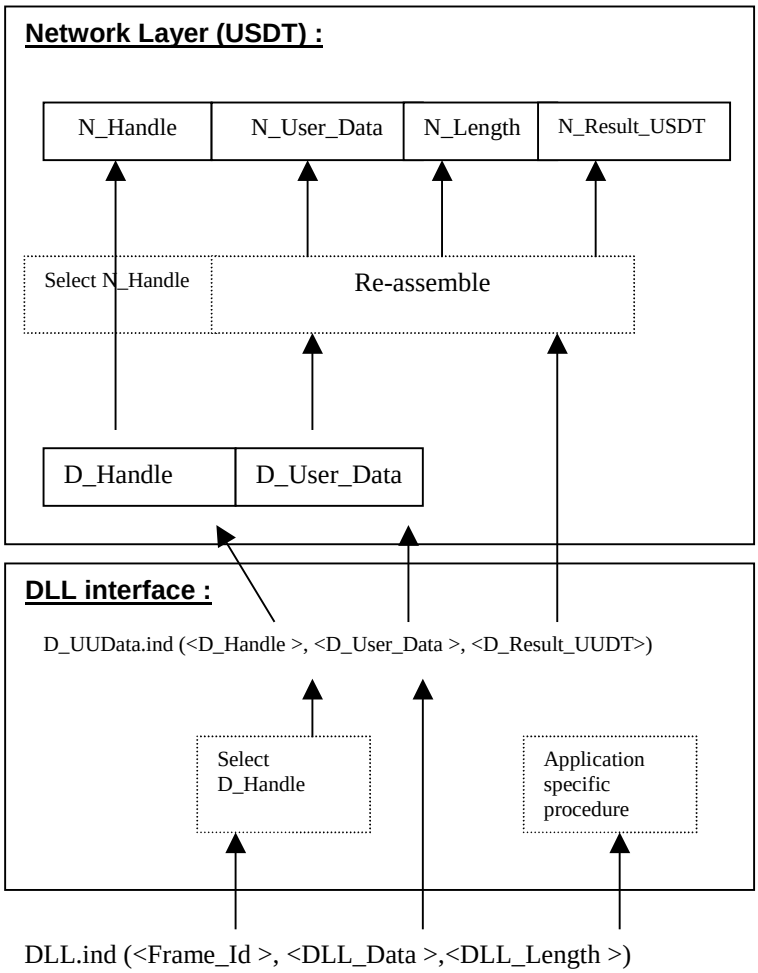

Figure 3-15: Mapping-in (USDT)

4 Data link layer interface

4.1 Data link layer overview

The data link layer interface provides abstract services for unacknowledged and unsegmented transfer of individual data frames over a network. The size of these data frames depends on the underlying network and is not specified by the OSEK COM specification. Additionally, the data link layer specification defines services that are referred to in the network management specification (e.g. configuration, initialisation, status request).

A distinction is made between the following types of data link layer services:

- Services provided to the upper OSEK COM layers only.
- Services provided to the OSEK network management only.
- Services provided to both of the above.

Note:

These services are internal services of the OSEK COM module : their interface (API) is not shown in full detail. This means that the name of these services and the name and usage of the parameters can differ from one OSEK COM implementation to another.

4.2 Data link layer specification

4.2.1 Definitions

Data link layer : a specific definition of a protocol layer corresponding to the layer number two (2) of the ISO/OSI basic reference model.

Data link layer frame : a bus protocol specific frame that is used to transfer information, e.g. CAN bus frame.

Data link layer frame identifier : a bus protocol specific identifier that is used to identify *data link layer frames, e.g. CAN bus frame identifier*.

Data link protocol data unit (DPDU) : a unit of data specified in the data link layer that is used to both support the exchange of information and co-ordinate the joint operation of the data link layer protocol. Data link protocol data units are encapsulated within *data link layer frames,* e.g. CAN frame.

Data link service data unit (DSDU) : an amount of information provided by the network layer, i.e. D_User_Data, whose identity is preserved and is not interpreted when transferred by the data link layer.

Addressing formats : a set of conventions that defines the positioning of the *data link protocol data unit* fields within the *data link layer frame*. These conventions are application specific and are consequently not specified by this specification.

Acceptance criteria : a set of criteria that are utilized to determine whether a received *data link protocol data unit* is to be forwarded to the network layer. These criteria and their respective implementation are application specific and are consequently not specified by this specification.

4.2.2 Services for the network layer

4.2.2.1 Transfer of data

Service name: **D_UUData**

Syntax: **internal service**

Request: D_UUData.req (<D_Handle>, <D_TA>, <D_User_Data>)

Confirmation: D_UUData.con (<D_Handle>, <D_TA>, <D_SA>, <D_Result_UUDT>)

Indication: D_UUData.ind (<D_Handle>, <D_SA>, <D_User_Data>, <D_Result_UUDT>)

Parameter:

<D_Handle> Reference to frame transfer path. A D_Handle is assigned to a single <N_Handle> service parameter. A D_Handle is monodirectional : the data link service data unit attached to a specific D_Handle can either be transmitted or received by the data link layer.

<D_TA> Target address (only used if Interaction layer's Dynamic Adressing scheme is used). This service parameter is optional and shall be defined if the network service parameter N_TA is utilised. The D_TA service parameter shall be utilised to support the selection of the data link layer frame identifier (e.g. CAN frame identifier) associated with the D_Handle. This specification does not define nor mandate any particular scheme for the definition or selection of data link layer frame identifier.

<D_SA> Source address (only used if Interaction layer's Dynamic Adressing scheme is used). This service parameter is optional and shall be defined if the network service parameter N_SA is utilised. The D_SA service parameter is retrieved from the received data link layer frame identifier (e.g. CAN frame identifier) in conformance with standardised or proprietary schemes. This specification does not define nor mandate any particular scheme for the definition or selection of data link layer frame identifier.

<D_User_Data>

<D_User_Data> is the data link user data that contains data received or to be transmitted by the data link layer. The length of <D_User_Data> is bus specific. The length depends on the type of underlying bus used and scheme adopted for the transfer of data.

<D_Result_UUDT>

Result of the requested service. The parameter <D_Result_UUDT> represents one of the following code :

- D_OK : this parameter shall be issued upon successful reception or transmission of a data link layer frame.
- D_NM (optional) : this parameter shall be supported only for data link layer frames that are monitored by the *indirect* OSEK network management only. This parameter shall be associated with a data link layer frame received that does not fulfil some or all acceptance criteria.

Description: The service primitive *D_UUData.req* sends the data <D_User_Data> over the network, using the physical (network) address associated to this frame transfer path referenced by the parameter <D_Handle> (and <D_TA> if applicable).

The service primitive *D_UUData.con* contains the confirmation regarding the execution of a previously called *D_UUData.req* service primitive.

The service primitive *D_UUData.ind* informs the receiver that a message corresponding to the frame transfer path referenced by <D_Handle> (and <D_SA> if applicable) has been received and provides the associated <D_User_Data>.

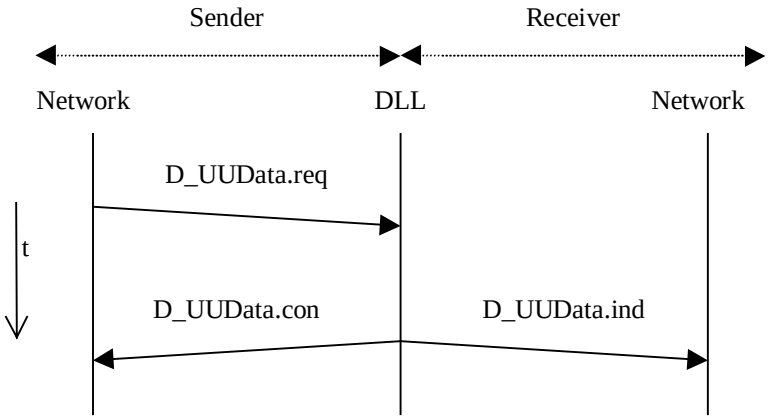

Figure 4-1: Sequencing of D_UUData service primitives

4.2.2.2 Handle status request

Service name: **`D_GetHandleStatus`**

Syntax: **internal service**

Parameter (In):

<D_Handle> Reference to frame transfer path. A D_Handle is assigned to a single <N_Handle> service parameter. A D_Handle is monodirectional : the data link service data unit attached to a specific D_Handle can either be transmitted or received by the data link layer.

Parameter(Out):

StatusData implementation specific status information

Description: The service *D_GetHandleStatus* provides the status information of the handle object referenced by <D_Handle>. <StatusData> contains implementation specific status information, e.g. overflow, transfer status and so on.

Particularities: none

4.2.3 Services for the network management

The following services are provided to the OSEK network management only and are not available to „normal“ applications.

4.2.3.1 Transfer of Network Management Messages

Service name: **D_WindowData**

Syntax: **internal service**

Request: D_WindowData.req (<D_NetId>,<D_NMPDU>, D_NMPDU_Length)

Indication: D_WindowData.ind (<D_NetId>, <D_NMPDU>, D_NMPDU_Length)

Confirmation: D_WindowData.con (<D_NetId>, <D_ Result_WindowData>)

Parameter:

<D_NetId> reference to a particular data link layer

<D_NMPDU> network management protocol data unit that shall be formatted as specified in the OSEK NM specification.

<D_NMPDU_Length>

length of network management protocol data unit (D_NMPDU)

<D_Result_WindowData>

local confirmation (implementation specific)

Description: The service primitive *D_WindowData.req* sends the given <D_NMPDU> over a specific sub-network, referenced by <NetId>, after using the mechanism described in the OSEK NM specification to encode the message.

The service primitive *D_WindowData.con* provides a local confirmation regarding the execution of the previously called *D_WindowData.req* service. The parameter < D_Result_WindowData> represents an implementation specific status information.

The service primitive D_WindowData.ind indicates the arrival of a network management message at the receivers station. It decodes the received message according to the mechanism given in the OSEK NM specification and provides the corresponding <D_NMPDU>.

Particularities: these service primitives are not affected by blocking the data link layer (see below).

4.2.3.2 Disabling the data link layer

Service name: **D_Offline**

Syntax: **internal service**

Parameter (in):

<D_NetId> Reference to a sub-network

Description: The service *D_Offline* disables (blocks of) the user communication on the data link layer. Thus the D_UUData.req service is disabled while the D_UUData.indication remains enabled.

This service does not affect network management communication enacted via the D_WindowData service primitives (see above).

The current state (Online/Offline) is part of the status of the data link layer, which is provided as result of the service *D_GetStatus* (see below).

Particularities: Affects D_UUData.req but not D_UUData.indication and D_WindowData service primitives.

4.2.3.3 Enabling the data link layer

Service name: **`D_Online`**

Syntax: **internal service**

Parameter (in):
<NetId> Reference to a sub-network

Description: The service *D_Online* enables the user communication on the data link layer. Thus the D_UUData.request service is re-enabled, other services are not affected.

Particularities: none

4.2.3.4 Layer status request

Service name: **`D_GetLayerStatus`**

Syntax: **internal service**

Parameter (In):
<D_NetId> reference to a particular data link layer

<D_Handler> specific algorithm to manage individual status of the hardware, e.g. acknowledges an interrupt or read the status of the protocol circuit.

Parameter (Out):
<D_GetLayerStatusInformation>
implementation (hardware) specific status information

Description: The service *D_GetLayerStatus* provides the status information of the data link layer :

- Interrupt acknowledge to the protocol circuit

- Get the status of the protocol circuit

Besides the states Online/Offline (set by the corresponding services described above), <Status> contains implementation specific status information of the communication hardware (e.g. BusOff).

Particularities: none

4.2.3.5 Error indication

Service name: **D_Status**

Syntax: **internal service**

Indication: D_Status.ind (<D_ErrorStatus>)

Parameter (out):
<D_ErrorStatus>

implementation specific error status

Description: The service *D_Status.ind* indicates to the network management that an event has occurred in the data link layer during communication. Such event may be recognised as “Errors” by the Network Management only.

Particularities: none

4.2.4 Services for the network layer and network management

4.2.4.1 Initialisation

Service name: **D_Init**

Syntax: **internal service**

Parameter (in):
<D_NetId> Reference to a sub-network

<D_Action> function to be performed

Parameter (out):
<D_Result_Init>

Result of the requested service

Description: The service primitive *D_Init* initialises the communication hardware. This routine is called once from the StartCOM function when the network is starting-up and subsequently, if the communication has to be re-initialised (i.e. after a hardware related communication error, e.g. BusOff), by the network management.

The possible <D_Action> are :

BUS_INIT : initialise the physical layer

BUS_SHUT_DOWN : shutdown physical layer communication

BUS_RESTART : restart physical layer communication

BUS_SLEEP : physical layer to enter into sleep mode

BUS_AWAKE : physical layer to leave sleep mode

Particularities: none

4.2.4.2 Addressing formats

It is the responsibility of the system designer to define the application specific addressing format that shall be utilised to transfer data link protocol data units (DPDU's) identified by the D_Handle.

5 System generation requirements

This section defines the system generation requirements necessary to support an OSEK COM implementation. These requirements are provided to support the development of the OIL COM specification that will be finally enable the declaration and spoecification of OSEK COM based applications. The requirements are summarized following each header using the following format :

```
ENTITY_NAME {                                        -- this is a comment
        Is :
                Attribute_name = attribute_type (allowed attribute value | ..)

        VERB :
                ENTITY_NAME Entity_Instance
                (minimum number of Entity_Instance ….. to maximum number of Entity_Instance)
                XOR, AND …

}(minimum number of Entity_Name instance … to maximum of Entity_Name instance per application)
```

1. The token ENTITY_NAME is an enumeration.
2. The token "Is :" introduces the ENTITY_NAME attributes.
3. The token "VERB :" introduces associations of ENTITY_NAME with other ENTITY_NAME's extracted from the above list.
4. The "minimum number of Entity instance" and "maximum number of Entity instance" are of type scalar.
5. The reserved token „XOR“ specifies that token „VERB“ applies either to the entity declared on the left side of the reserved token „XOR“ or on the right side of the reserved token „XOR“
6. The reserved token „XOR (Optional AND)“ specifies that both entities on left and right sides of that reserved token may be supported in the relationship if required.
7. The reserved token „--„ introduces a one line comment.
8. The reserved token „|„ separate allowed attribute values of a particular “Attribute_name”.
9. The reserved token „AND“ specifies that token „VERB“ applies either to both the entity declared on the left side of the reserved token „AND“ and that declared on the right side of the reserved token „AND“.

Note: The entity EVENT does not refer to the OIL object EVENT. It rather relates to a combination of the OIL objects EVENT and TASK.

5.1 Conformance class

This section defines the system generation requirements applicable to the conformance class.

CONFORMANCE_CLASS {
    Is :
    CCC = enumeration (A | B | 0 | 1 | 2)
} (1…1)

5.1.1 Entity requirements

An OSEK COM implementation shall conform to a specific conformance class as defined in the OSEK COM specification.

An OSEK COM implementation shall provide all features defined in the supported conformance class.

5.1.2 Entity attributes requirements

CCC :

1. *The type of "CCC" is enumeration.*
2. *The reserved values of the "CCC" enumeration type are "A", "B", "0", "1", "2".*
3. *A conformance class shall be identified with a single reserved value of the "CCC" attribute.*

5.2 Unqueued message

This section defines the system generation requirements applicable to unqueued message.

UNQUEUED_MESSAGE {

Is :

SymbolicName = `enumeration`

Length = `scalar`

Scope = `enumeration ( internal | external | internal_external)`

Is_Sent_by :

TASK TaskName

(0…1) XOR

ISR IsrName

(0…1) XOR

FUNCTION FunctionName

(0…1) XOR

CALLBACK CallbackName

(0…1)

DIRECT_TRANSMISSION_MODE DirectTransmissionModeName

(0…1) XOR

PERIODICAL_TRANSMISSION_MODE PeriodicalTransmissionModeName

(0…1) XOR

MIXED_TRANSMISSION_MODE MixedTransmissionModeName

(0…1)

TRANSMISSION_DEADLINE_MONITOR TransmissionMonitorName

(0…1)

Is_Received_by :

TASK TaskName

(0…MAX_Task)

ISR IsrName
(0…MAX_ISR)

FUNCTION FunctionName
(0…MAX_FUNCTION)

CALLBACK CallbackName
(0…MAX_CALLBACK)

RECEPTION_DEADLINE_MONITOR Scheme
(0…Max_RDM)

Is_Accessed_by :
ACCESSOR AccessName
(0…MAX_Accessor)

Activates_**Unconditionally**_On_**Successful**_Reception_Or_Transmission,
Activates_**Conditionnally**_On_**Successful**_Reception_Or_Transmission,
Activates_On_**Unsuccessful**_Reception_Or_Transmission,
Activates_On_**First_Frame_Indication** :

FLAG FlagName
(0…MAX_Flag) XOR (optional AND)

CALLBACK CallbackName
(0…MAX_Callback) XOR (optional AND)

EVENT EventName
(0…MAX_Event) XOR (optional AND)

TASK TaskName
(0…MAX_Task)

Is_Connected_to :
N_HANDLE N_HandleName
(0…1)
} (0…MAX_Unqueued_Message)

5.2.1 Entity requirements

The number of Unqueued messages per application is application specific. „MAX_Unqueued_Message“ shall document the maximum number of unqueued messages supported by an OSEK COM implementation.

5.2.2 Entity attributes requirements

5.2.2.1 SymbolicName :

1. *The symbolic name (SymbolicName) of an unqueued message shall be defined at system generation time.*
2. *The type of "SymbolicName" is enumeration.*

5.2.2.2 Length :

1. *The length (Message_Length) of an unqueued message identified by "SymbolicName" shall be defined at system generation time.*
2. *The type of "Message_Length" is scalar.*

5.2.2.3 Scope:

1. *The scope of an unqueued message (Scope) identified by "SymbolicName" shall be defined at system generation time.*
2. *The type of "Scope" is enumeration.*
3. *The reserved values of the "Scope" enumeration type are "internal", "external", "internal_external".*
4. *An unqueued message identified by "SymbolicName" shall be assigned a single reserved value of the "Scope" attribute.*

5.2.3 Entity association requirements

5.2.3.1 is_sent_by:

1. *An unqueued message identified by "SymbolicName“ shall be transmitted as a minimum by no application sender (TASK, ISR, FUNCTION, CALLBACK).*
2. *An unqueued message identified by "SymbolicName" shall be transmitted by as a maximum one application sender (TASK, ISR, FUNCTION, CALLBACK) identified byits name.*
3. *An unqueued message identified by "SymbolicName" shall be using one of the following transmission modes :*
   - *DirectTransmissionMode identified by "DirectTransmissionModeName", or*
   - *PeriodicalTransmissionMode identified by "PeriodicalTransmissionModeName", or*
   - *MixedTransmissionMode identified by "MixedTransmisionModeName"*

The PeriodicalTransmissionMode and MixedTransmissionMode are applicable if the "Scope" is "external" or "internal_external".

4. *An unqueued message identified by "SymbolicName" can use a Transmission_Deadline_Monitor identified by "TransmissionMonitorName" if the "Scope" is "external" or "internal_external".*
5. *An unqueued message identified by "SymbolicName" shall be using as a minimum no Transmission_Deadline_Monitor.*
6. *An unqueued message identified by "SymbolicName" shall be using as a maximum one Transmission_Deadline_Monitor identified by "TDM_Name".*

5.2.3.2 is_received_by:

1. *An unqueued message identified by "SymbolicName" shall be received as a minimum by no internal application receiver (TASK, ISR, FUNCTION, CALLBACK).*
2. *An unqueued message identified by "SymbolicName" shall be received as a maximum by "MAX_Task" internal application receiver's TASK identified by "TaskName" or by "MAX_ISR" internal application receiver's ISRs identified by "IsrName" or by "MAX_FUNCTION" internal application receiver's FUNCTIONS identified by "FunctionName" or by "MAX_CALLBACK" internal application receiver's CALLBACKs identified by "Callback"*
3. *An unqueued message identified by "SymbolicName" can use a Reception_Deadline_Monitor if the "Scope" is "external" or "internal_external".*
4. *An unqueued message identified by "SymbolicName" shall be using as a minimum no Reception_Deadline_Monitor .*
5. *An unqueued message identified by "SymbolicName" shall be using as a maximum “Max_RDM" one Reception_Deadline_Monitor identified by "RDM_Name".*

5.2.3.3 is_accessed_by:

1. *An unqueued message identified by "SymbolicName" shall be accessed as a minimum by one accessor.*
2. *An unqueued message identified by "SymbolicName" shall be accessed as a maximum by "MAX_Accessor" accessors identified by "AccessName"*

5.2.3.4 activates_unconditionally_on_successful_reception_or_transmission,

5.2.3.5 activates_on_unsuccessful_reception_or_transmission,

5.2.3.6 activates_on_first_frame_indication :

1. *An unqueued message identified by "SymbolicName" shall be assigned as a minimum to no FLAG, no CALLBACK , no EVENT or no TASK.*
2. *An unqueued message identified by "SymbolicName" shall be assigned as a maximum to „MAX_Flag“ FLAG identified by their respective "FlagName", „MAX_Callback“ CALLBACK identified by their respective "CallbackName",*

„MAX_Event“ EVENT identified by "EventName" or„MAX_Task“ TASK identified by their respective "TaskName".

5.2.3.7 activates_conditionally_on_successful_reception_or_transmission:

1. *An unqueued message identified by "SymbolicName" shall be assigned as a minimum to no FLAG, no CALLBACK , no EVENT or no TASK.*
2. *An unqueued message identified by "SymbolicName" shall be assigned as a maximum to „MAX_Flag“ FLAG identified by their respective "FlagName", „MAX_Callback“ CALLBACK identified by their respective "CallbackName", „MAX_Event“ EVENT identified by "EventName" or„MAX_Task“ TASK identified by their respective "TaskName".*
3. *A relevant change condition shall be maintained as described in „relevantChange“ of „Mixed Transmission Mode Specification“.*

5.2.3.8 is_connected_to_handle:

1. *An unqueued message identified by "SymbolicName" shall not be assigned to any network handle (N_Handle) if the "scope" is "internal".*
2. *An unqueued message identified by SymbolicName shall be assigned to one network handle (N_Handle) if the "scope" is either "external" or "internal_external".*

5.3 Queued message

This section defines the system generation applicable requirements to queued message.

QUEUED_MESSAGE {

Is :

SymbolicName = `enumeration`

Length = `scalar`

Scope = `enumeration ( internal | external | internal_external)`

Is_Sent_by :

TASK TaskName

(0…1) XOR

FUNCTION FunctionName

(0…1)

DIRECT_TRANSMISSION_MODE Mode

(0…1) XOR

PERIODICAL_TRANSMISSION_MODE Mode

(0…1) XOR

MIXED_TRANSMISSION_MODE Mode

(0…1)

TRANSMISSION_DEADLINE_MONITOR TransmissionMonitoringName

(0…1)

Is_Received_by :

TASK TaskName WITH FIFO_Depth = `scalar`

(0…1)

FUNCTION FunctionName WITH FIFO_Depth = `scalar`

(0…1)

RECEPTION_DEADLINE_MONITOR ReceptionMonitoringName

(0…MAX_RDM)

Uses :

ACCESSOR AccessName

(1…1)

Activates_**Unconditionally**_On_Successful_Reception_Or_Transmission,

Activates_**Conditionnally**_On_**Successful**_Reception_Or_Transmission,

Activates_On_**Unsuccessful**_Reception_Or_Transmission :

FLAG FlagName

(0…MAX_Flag) XOR (optional AND)

CALLBACK CallbackName

(0…1) XOR (optional AND)

EVENT EventName

(0…1) XOR (optional AND)

TASK TaskName

(0…1)

Connected_to :

N_HANDLE N_HandleName

(0…1)

} (0…MAX_QUEUED_MESSAGE)

5.3.1 Entity requirements

The number of queued messages per application is application specific. „MAX_Queued_Message“ shall document the maximum number of queued message supported by an OSEK COM implementation.

5.3.2 Entity attributes requirements

5.3.2.1 SymbolicName :

1. *The symbolic name (SymbolicName) of a queued message shall be defined at system generation time.*
2. *The type of "SymbolicName" is enumeration.*

5.3.2.2 Length :

1. *The length (Message_Length) of a queued message identified by "SymbolicName" shall be defined at system generation time.*
2. *The type of "Message_Length" is scalar.*

5.3.2.3 Scope :

1. *The scope of a queued message (Scope) identified by "SymbolicName" shall be defined at system generation time.*
2. *The type of "Scope" is enumeration.*
3. *The reserved values of "Scope" are : "internal", "external", "internal_external".*
4. *A queued message identified by "SymbolicName" shall be assigned a single reserved value of the "Scope" attribute.*

5.3.3 Entity association requirements

5.3.3.1 is_sent_by:

1. *A queued message identified by "SymbolicName" shall be transmitted as a minimum by no application sender (TASK, FUNCTION).*
2. *A queued message identified by "SymbolicName" shall be transmitted by as a maximum one application sender TASK identified by "TaskName" or FUNCTION identified by "FunctionName".*
3. *A queued message identified by "SymbolicName" shall be using one of the following transmission modes :*
   - *DirectTransmissionMode identified by "DirectTransmissionModeName", or*
   - *PeriodicalTransmissionMode identified by "PeriodicalTransmissionModeName", or*
   - *MixedTransmissionMode identified by "MixedTransmisionModeName"*

   The PeriodicalTransmissionMode and MixedTransmissionMode shall be applicable if the "Scope" is "external" or "internal_external".
4. *A queued message identified by "SymbolicName" can use a Transmission_Deadline_Monitor if the "Scope" is "external" or "internal_external".*
5. *A queued message identified by "SymbolicName" shall be using as a minimum no Transmission_Deadline_Monitor .*
6. *A queued message identified by "SymbolicName" shall be using as a maximum one Transmission_Deadline_Monitor identified by "TDM_Name".*

5.3.3.2 is_received_by:

1. *A queued message identified by "SymbolicName" shall be received as a minimum by no internal application receiver (TASK, FUNCTION).*
2. *A queued message identified by "SymbolicName" shall be received as a maximum*

by one application receiver's TASK identified by "TaskName" interfacing with a specific queued message with depth of the corresponding logical FIFO queue equal to "FIFO_Depth" or as a maximum by one application receiver's FUNCTIONs identified by "FunctionName" interfacing with a specific queued message with depth of the corresponding logical FIFO queue equal to "FIFO_Depth"

3. *A queued message identified by "SymbolicName" can use a Reception_Deadline_Monitor if the "Scope" is "external" or "internal_external".*
4. *A queued message identified by "SymbolicName" shall be using as a minimum no Reception_Deadline_Monitor.*
5. *A queued message identified by "SymbolicName" shall be using as a maximum "Max_RDM" Reception_Deadline_Monitor identified by "RDM_Name".*

5.3.3.3 is_accessed_by:

1. *A queued message identified by "SymbolicName" shall be accessed as a minimum by one accessor.*
2. *A queued message identified by "SymbolicName" shall be accessed as a maximum by one accessor's identified by "AccessName".*

5.3.3.4 activates_unconditionally_on_successful_reception_or_transmission,

5.3.3.5 activates_on_unsuccessful_reception_or_transmission :

1. *A queued message identified by "SymbolicName" shall be assigned as a minimum to no FLAG , no CALLBACK , no EVENT or no TASK.*
2. *A queued message identified by "SymbolicName" shall be assigned as a maximum to „MAX_Flag“ FLAG identified by their respective "FlagName", one CALLBACK identified by their respective "CallbackName", one EVENT identified by "EventName" or one TASK identified by their respective "TaskName".*

5.3.3.6 activates_conditionally_on_successful_reception_or_transmission :

1. *A queued message identified by "SymbolicName" shall be assigned as a minimum to no FLAG , no CALLBACK , no EVENT or no TASK.*
2. *A queued message identified by "SymbolicName" shall be assigned as a maximum to „MAX_Flag“ FLAG identified by their respective "FlagName", one CALLBACK identified by their respective "CallbackName", one EVENT identified by "EventName" or one TASK identified by their respective "TaskName".*
3. *A relevant change condition shall be maintained as described in „relevantChange“ of „Mixed Transmission Mode Specification“.*

5.3.3.7 is_connected_to_handle :

1. *A queued message identified by "SymbolicName" shall be assigned to no network handle (N_Handle) if the "scope" is "internal".*
2. *A queued message identified by SymbolicName shall be assigned to one network*

handle (N_Handle) if the "scope" is either "external" or "internal_external".

5.4 Message accessor

This section defines the system generation requirements applicable to message accessor.

ACCESSOR {

Is :

AccessName = `enumeration`

Copy = `enumeration (WithCopy | WithoutCopy)`

Accesses_message :

UNQUEUED | QUEUED _MESSAGE SymbolicName

(1…1)

Is_used_by_Task:

TASK TaskName

(1…1)

Is_used_by_Function:

FUNCTION FunctionName

(1…1)

Is_used_by_Callback:

CALLBACK CallbackName

(1…1)

Is_used_by_Isr:

ISR IsrName

(1…1)

} (0…MAX_ACCESSOR)

5.4.1 Entity requirements

The number of message accessor per application is application specific. „MAX_Accessor“ shall document the maximum number of message accessor supported by an OSEK COM implementation.

5.4.2 Entity attributes requirements

5.4.2.1 AccessName :

1. *The name of a message accessor (AccessName) shall be defined at system generation. The type of "AccessName" is enumeration.*

5.4.2.2 Copy :

1. *The specification of a message accessor copy (Copy) identified by "AccessName" shall be specified at system generation time.*
2. *The type of "Copy" is enumeration.*
3. *The reserved values of the "Copy" enumeration type are "WithCopy", "WithoutCopy".*
4. *A message accessor identified by "AccessName" shall be assigned a single reserved value of the "Copy" attribute.*
5. *If "Copy" is set to "WithCopy" then a copy shall be associated with the Accessor identified by "AccessName". If "Copy" is set to "WithoutCopy" then no copy shall be associated with Accessor identified by "AccessName".*
6. *"Copy" shall be set to "WithCopy" for Accessors identified by "AccessName" associated with queued messages.*

5.4.3 Entity association requirements

5.4.3.1 accesses_message :

1. *An accessor identified by "AccessName" shall be used to access a message (queued or unqueued) identified by "SymbolicName".*

5.4.3.2 is_used_by_task :

1. *An accessor identified by "AccessName" can be used by one task identified by "TaskName" only.*

5.4.3.3 is_used_by_function:

1. *An accessor identified by "AccessName" can be used by one function identified by "FunctionName" only.*

5.4.3.4 is_used_by_callback :

1. *An accessor identified by "AccessName" can be used by one Callback identified by "CallbackName" only.*

5.4.3.5 is_used_by_isr :

1. *An accessor identified by "AccessName" can be used by one interrupt sub routine (ISR) identified by "IsrName".*

5.5 Direct transmission mode specification

This section defines the system generation requirements applicable to direct transmission mode specification.

DIRECT_TRANSMISSION_MODE {
Is :
DirectTransmissionModeName = `enumeration`
Transmits :
QUEUED | UNQUEUED_MESSAGE SymbolicName
(1…1)
} (0…MAX_Direct_Transmission_Mode)

5.5.1 Entity requirements

The number of transmission mode specification per application is application specific. „MAX_Direct_Transmission_Mode“ shall document the maximum number of transmission mode specification supported by an OSEK COM implementation.

5.5.2 Entity attributes requirements

5.5.2.1 TransmissionModeName:

1. *The name of a transmission mode specification (DirectTransmissionModeName) shall be defined at system generation.*
2. *The type of "DirectTransmissionModeName" is enumeration.*

5.5.3 Entity association requirements

5.5.3.1 Transmits :

1. *A transmission mode specification identified by" DirectTransmissionModeName" shall specify as a minimum no message (queued or unqueued).*
2. *A transmission mode specification identified by "DirectTransmissionModeName" shall specify as a maximum "MAX_Unqueued_Message" or "MAX_Queued_Message" messages identified by their respective "SymbolicName".*

5.6 Periodical transmission mode specification

This section defines the system generation requirements applicable to periodical transmission mode specification.

PERIODICAL_TRANSMISSION_MODE {

Is :

PeriodicalTransmissionModeName = `enumeration`

I_TMP_TPD = `scalar`

I_TMP_TOFF = `scalar`

Transmits :

QUEUED | UNQUEUED_MESSAGE SymbolicName

(1…1)

} (0…MAX_Periodical_Transmission_Mode)

5.6.1 Entity requirements

The number of periodical transmission mode specification per application is application specific. „MAX_Periodical_Transmission_Mode“ shall document the maximum number of periodical transmission mode specification supported by an OSEK COM implementation.

5.6.2 Entity attributes requirements

5.6.2.1 TransmissionModeName :

1. *The name of a periodical transmission mode specification (PeriodicalTransmissionModeName) shall be defined at system generation.*
2. *The type of "PeriodicalTransmissionModeName" is enumeration.*

5.6.2.2 I_TMP_TPD attribute :

1. *The value of the periodical transmission mode time period (I_TMP_TPD) shall be defined at system generation.*
2. *The type of "I_TMP_TPD" is scalar; unit is millisecond.*

5.6.2.3 I_TMP_TOFF attribute :

1. *The value of the periodical transmission mode time offset (I_TMP_TOFF) shall be defined at system generation.*
2. *The type of "I_TMP_TOFF" is scalar; units is milliseconds.*

5.6.3 Entity association requirements

5.6.3.1 Transmits :

1. *A periodical transmission mode specification identified by "PeriodicalTransmissionModeName" shall specify as a minimum no message (queued or unqueued).*
2. *A periodical mode specification identified by "PeriodicalTransmissionModeName" shall specify as a maximum "MAX_Unqueued_Message" or "MAX_Queued_Message" messages identified by their respective "SymbolicName".*

5.7 Mixed transmission mode specification

This section defines the system generation requirements applicable to mixed transmission mode specification.

MIXED_TRANSMISSION_MODE {

Is :

MixedTransmissionModeName = enumeration

I_TMM_TPD = scalar

I_TMM_TOFF = scalar

RelevantChange specification

Transmits :

QUEUED | UNQUEUED_MESSAGE SymbolicName

(1…1)

} (0…MAX_Mixed_Transmission_Mode)

5.7.1 Entity requirements

The number of mixed transmission mode specification per application is application specific. „MAX_Mixed_Transmission_Mode“ shall document the maximum number of mixed transmission mode specification supported by an OSEK COM implementation.

5.7.2 Entity attributes requirements

5.7.2.1 TransmissionModeName :

1. *The name of a mixed transmission mode specification (MixedTransmissionModeName) shall be defined at system generation.*
2. *The type of "MixedTransmissionModeName" is enumeration.*

5.7.2.2 I_TMM_TPD :

1. *The value of the mixed transmission mode time period (I_TMM_TPD) shall be defined at system generation.*
2. *The type of "I_TMM_TPD" is scalar ; unit is millisecond.*

5.7.2.3 I_TMM_TOFF:

1. *The value of the mixed transmission mode time offset (I_TMM_TOFF) shall be defined at system generation.*

2. The type of "I_TMM_TOFF" is scalar ; unit is millisecond.

5.7.2.4 RelevantChange :

The RelevantChange attribute contains the specification that determines whether an intermediate transmission shall take place or not.

A relevant change of the message value is detected when the message value matches a specific condition.

A list of possible "relevant changes" is given below ("Value" is an abbreviation for message data value and "Oldvalue" is an abbreviation for old message data value) :

a) Value less than constant

A relevant change is detected if the message value is less than a constant which is defined at system generation time.

Example: temperature < MIN_TEMPERATURE

b) Value greater than constant

A relevant change is detected if the message value is greater than a constant which is defined at system generation time.

Example: temperature > MAX_TEMPERATURE

c) Value equal to constant

A relevant change is detected if the message value is equal to a constant which is defined at system generation time.

Example: temperature == REF_TEMPERATURE

d) (Value-Oldvalue) less than constant

A relevant change is detected if the change of the message value is less than a constant which is defined at system generation time.

Example: (temperature-temperature_old) < MIN_TEMPERATURE_CHANGE

e) (Value-Oldvalue) greater than constant

A relevant change is detected if the change of the message value is greater than a constant which is defined at system generation time.

Example: (temperature-temperature_old) > MAX_TEMPERATURE_CHANGE

f) (Value-Oldvalue) equal to constant

A relevant change is detected if the change of the message value is equal to a

constant which is defined at system generation time.

Example: (temperature-temperature_old) == REF_TEMPERATURE_CHANGE

g) always „true„

A relevant change is always detected regardless of the message value. Therefore each update of the message object issues an intermediate transmission.

OSEK COM also provides the means to implement user defined "relevant" changes.

5.7.3 Entity association requirements

5.7.3.1 transmits :

1. *A mixed transmission mode specification identified by "MixedTransmissionModeName" shall specify as a minimumno message (queued or unqueued).*
2. *A mixed mode specification identified by "MixedTransmissionModeName" shall specify as a maximummultiple "MAX_Unqueued_Message" or "MAX_Queued_Message" messages identified by "SymbolicName".*

5.8 Reception deadline monitoring specification

This section defines the system generation requirements applicable to reception deadline monitoring specification.

RECEPTION_MONITORING {

Is :

ReceptionMonitoringName = `enumeration`

Is_used_by_message :

UNQUEUED | QUEUED _MESSAGE SymbolicName

(1…MAX_Unqueued_Message | MAX_Queued_Message)

Activates:

TASK TaskName

(1…MAX_Task) XOR (optional AND)

EVENT EventName

(1…MAX_Event) XOR (optional AND)

CALLBACK CallbackName

(1…MAX_Callback) XOR (optional AND)

FLAG FlagName

(1…MAX_Flag)

NM NMName

} (0…MAX_Reception_Monitoring)

5.8.1 Entity requirements

The number of reception monitoring specification per application is application specific. „MAX_Reception_Monitoring“ shall document the maximum number of reception monitoring specification supported by an OSEK COM implementation.

5.8.2 Entity attributes requirements

5.8.2.1 ReceptionMonitoringName :

1. *The name of a reception monitoring specification (ReceptionMonitoringName) shall be defined at system generation.*
2. *The type of "ReceptionMonitoringName" is enumeration.*

5.8.3 Entity association requirements

5.8.3.1 is_used_by_message :

1. *A reception monitoring specification identified by " ReceptionMonitoringName " shall specify as a minimum one message (queued or unqueued) identified by "SymbolicName".*
2. *A reception monitoring specification identified by " ReceptionMonitoringName " shall specify as a maximum "MAX_Unqueued_Message" or "MAX_Queued_Message" messages identified by their respective "SymbolicName".*

5.8.3.2 activates :

1. *A reception monitoring specification identified by "ReceptionMonitoringName"can activate as a minimum no CALLBACK, no FLAG, no EVENT and no TASK.*
2. *A reception monitoring specification identified by "ReceptionMonitoringName"can activate as a maximum „MAX_Callback“ CALLBACK identified by their respective "CallbackName" or „MAX_Flag“ FLAG identified by „FlagName” or „MAX_Event“ EVENT identified by "EventName" and „MAX_Task“ TASK identified by their respective "TaskName".*
3. *The indirect Network Management shall be interfaced as a minimum by one reception monitoring specification identified by "ReceptionMonitoringName".*
4. *The indirect Network Management shall be interfaced as a maximum by multiple reception monitoring specification identified by "ReceptionMonitoringName".*
5. *At least one of the mechanisms defined in this section shall be utilized to process the events generated by the monitoring entity.*

5.9 Transmission deadline monitoring specification

This section defines the system generation requirements applicable to transmission deadline monitoring specification.

TRANSMISSION_MONITORING {

Is :

TransmissionMonitoringName = `enumeration`

Is_used_by_message :

UNQUEUED | QUEUED _MESSAGE SymbolicName

(1…MAX_Unqueued_Message | MAX_Queued_Message)

Activates :

TASK TaskName

(1…MAX_Task) XOR (optional AND)

EVENT EventName

(1…MAX_Event) XOR (optional AND)

CALLBACK CallbackName

(1…MAX_Callback) XOR (optional AND)

FLAG FlagName

(1...MAX_Flag)

NM NMName

} (0…MAX_Transmission_Monitoring)

5.9.1 Entity requirements

The number of transmission monitoring specification per application is application specific. „MAX_Transmission_Monitoring“ shall document the maximum number of transmission monitoring specification supported by an OSEK COM implementation.

5.9.2 Entity attributes requirements

5.9.2.1 TransmissionMonitoringName :

1. *The name of a transmission monitoring specification (TransmissionMonitoringName) shall be defined at system generation.*
2. *The type of "TransmissonMonitoringName" is enumeration.*

5.9.3 Entity association requirements

5.9.3.1 is_used_by_message :

1. *A transmission monitoring specification identified by "TransmissionMonitoringName" shall specify as a minimum one message (queued or unqueued) identified by "SymbolicName".*
2. *A reception monitoring specification identified by "TransmissionMonitoringName" shall specify as a maximum "MAX_Unqueued_Message" or "MAX_Queued_Message" messages identified by their respective "SymbolicName".*

5.9.3.2 activates :

1. *A transmission monitoring specification identified by "ReceptionMonitoringName"can activate as a minimum no FLAG , no CALLBACK , no EVENT and no TASK.*
2. *A transmission monitoring specification identified by "ReceptionMonitoringName"can activate as a maximum „MAX_Callback“ CALLBACK identified by their respective "CallbackName" or „MAX_Flag“ FLAG identified by „FlagName” or „MAX_Event“ EVENT identified by "EventName" and „MAX_Task“ TASK identified by their respective "TaskName".*
3. *The indirect Network Management shall be interfaced with as a minimum by one transmission monitoring specification identified by "TransmissionMonitoringName".*
4. *The indirect Network Management shall be interfaced with as a maximum by "MAX_Transmission_Monitoring" transmission monitoring specification identified by "TransmissionMonitoringName".*
5. *At least one of the mechanisms defined in this section shall be utilized to process the events generated by the monitoring entity.*

5.10 Task

This section defines the system generation requirements applicable to Task.

TASK {
Is :
ActivityName = `enumeration`

Is_Activated_**Unconditionally_On_Successful**_Reception_Or_Transmission,
Is_Activated_**Conditionnally_On_Successful**_Reception_Or_Transmission,
Is_Activated_On_**Unsuccessful**_Reception_Or_Transmission,
Is_Activated_On_**First_Frame_Indication** :
UNQUEUED SymbolicName
(1…MAX_Unqueued_Message)

QUEUED _MESSAGE SymbolicName
(1… MAX_Queued_Message)

Is_activated_**by_reception_deadline_monitor** :
RECEPTION_ MONITOR RDM_Name
(1…MAX_Reception_Monitor)

Is_activated_**by_transmission_deadline_monitor** :
TRANSMISSION_MONITOR TDM_Name
(1…MAX_Transmission_Monitor)
}(0…MAX_Task)

5.10.1 Entity requirements

The number of task per application is application specific. „MAX_Task“ shall document the maximum number of task supported by an OSEK COM implementation.

5.10.2 Entity attributes requirements

5.10.2.1 TaskName:

1. *The name of a Task (TaskName) shall be defined at system generation. The type of "TaskName" is enumeration.*

5.10.3 Entity association requirements

5.10.3.1 is_activated_unconditionally_on_successful_reception_or_transmission,

5.10.3.2 is_activated_on_unsuccessful_reception_or_transmission,

5.10.3.3 is_activated_on_first_frame_indication :

1. *A Task identified by "TaskName" shall be activated as a minimum by one message (queued or unqueued) identified by "SymbolicName".*
2. *A Task identified by "TaskName" shall be activated as a maximum by "MAX_Unqueued_Message" or "MAX_Queued_Message" messages identified by their respective "SymbolicName".*

5.10.3.4 is_activated_conditionnally_on_successful_reception_or_transmission :

1. *A Task identified by " TaskName " shall be activated as a minimum by one message (queued or unqueued) identified by "SymbolicName".*
2. *A Task identified by " TaskName " shall be activated as a maximum by "MAX_Message" messages (queued or unqueued) identified by their respective "SymbolicName".*
3. *A relevant change condition shall be maintained as described in „relevantChange“ of „Mixed Transmission Mode Specification“.*

5.10.3.5 is_activated_by_reception_deadline _monitor :

1. *A Task identified by "TaskName" shall be activated as a minimumby one Reception_Deadline_Monitor identified by "RDM_Name".*
2. *A Task identified by "TaskName" shall be activated as a maximumby "MAX_Reception_Monitor" Reception_Deadline_Monitor's identified by their respective "RDM_Name".*

5.10.3.6 is_activated_by_transmission_deadline _monitor :

1. *A Task identified by "TaskName" shall be activated as a minimumby one Transmission_Deadline_Monitor identified by "TDM_Name".*
2. *A Task identified by "TaskName" shall be activated as a maximumby "MAX_Transmission_Monitor" Transmission_Deadline_Monitor's identified by "TDM_Name".*

5.11 Function

This section defines the system generation requirements applicable to Function.

FUNCTION {

Is :

ActivityName = `enumeration`

Is_Activated_**Unconditionally_On_Successful**_Reception_Or_Transmission,

Is_Activated_**Conditionnally_On_Successful**_Reception_Or_Transmission,

Is_Activated_On_**Unsuccessful**_Reception_Or_Transmission,

Is_Activated_On_**First_Frame_Indication** :

UNQUEUED SymbolicName

(1…MAX_Unqueued_Message)

QUEUED _MESSAGE SymbolicName

(1… MAX_Queued_Message)

Is_activated_**by_reception_deadline_monitor** :

RECEPTION_ MONITOR RDM_Name

(1…MAX_Reception_Monitor)

Is_activated_**by_transmission_deadline_monitor** :

TRANSMISSION_MONITOR TDM_Name

(1…MAX_Transmission_Monitor)

}(0…MAX_Function)

5.11.1 Entity requirements

The number of Function per application is application specific. „MAX_Function“ shall document the maximum number of function supported by an OSEK COM implementation.

5.11.2 Entity attributes requirements

5.11.2.1 FunctionName:

2. *The name of a Function (FunctionName) shall be defined at system generation. The type of "FunctionName" is enumeration.*

5.11.3 Entity association requirements

5.11.3.1 is_activated_unconditionally_on_successful_reception_or_transmission,

5.11.3.2 is_activated_on_unsuccessful_reception_or_transmission,

5.11.3.3 is_activated_on_first_frame_indication :

1. *A Function identified by "FunctionName" shall be activated as a minimum by one message (queued or unqueued) identified by "SymbolicName".*
2. *A Function identified by "FunctionName" shall be activated as a maximum by "MAX_Unqueued_Message" or "MAX_Queued_Message" messages identified by their respective "SymbolicName".*

5.11.3.4 is_activated_conditionnally_on_successful_reception_or_transmission :

1. *A Function identified by " FunctionName " shall be activated as a minimum by one message (queued or unqueued) identified by "SymbolicName".*
2. *A Function identified by " FunctionName " shall be activated as a maximum by "MAX_Message" messages (queued or unqueued) identified by their respective "SymbolicName".*
3. *A relevant change condition shall be maintained as described in „relevantChange“ of „Mixed Transmission Mode Specification“.*

5.11.3.5 is_activated_by_reception_deadline _monitor :

1. *A Function identified by "FunctionName" shall be activated as a minimumby one Reception_Deadline_Monitor identified by "RDM_Name".*
2. *A Function identified by "FunctionName" shall be activated as a maximumby "MAX_Reception_Monitor" Reception_Deadline_Monitor's identified by their respective "RDM_Name".*

5.11.3.6 is_activated_by_transmission_deadline _monitor :

1. *A Function identified by "FunctionName" shall be activated as a minimumby one Transmission_Deadline_Monitor identified by "TDM_Name".*
2. *A Function identified by "FunctionName" shall be activated as a maximumby "MAX_Transmission_Monitor" Transmission_Deadline_Monitor's identified by "TDM_Name".*

5.12 Callback

This section defines the system generation requirements applicable to callback.

CALLBACK {
Is :
CallbackName = `enumeration`

Is_Activated_**Unconditionally_On_Successful**_Reception_Or_Transmission,
Is_Activated_**Conditionnally_On_Successful**_Reception_Or_Transmission,
Is_Activated_On_**Unsuccessful**_Reception_Or_Transmission,
Is_Activated_On_**First_Frame_Indication** :
UNQUEUED SymbolicName
(1…MAX_Unqueued_Message)

QUEUED _MESSAGE SymbolicName
(1… MAX_Queued_Message)

Is_activated_by_**reception_deadline_monitor** :
RECEPTION_MONITOR RDM_Name
(1…MAX_Reception_Monitor)

Is_activated_by_**transmission_deadline_monitor** :
TRANSMISSION_MONITOR TDM_Name
(1…MAX_Transmission_Monitor)

}(0…MAX_Callback)

5.12.1 Entity requirements

The number of callback per application is application specific. „MAX_Callback“ shall document the maximum number of callback supported by an OSEK COM implementation.

5.12.2 Entity attributes requirements

5.12.2.1 CallbackName:

1. *The name of a Callback (CallbackName) shall be defined at system generation.*

The type of "CallbackName" is enumeration.

2. *The following format of the callback prototype shall apply :*

void <CallbackRoutineName> (void);

Example for a callback routine:

```
void EngineTempIndication(void)
{
  // .... do application specific processing here
}
```

5.12.3 Entity association requirements

5.12.3.1 is_activated_unconditionally_on_successful_reception_or_transmission,

5.12.3.2 is_activated_on_unsuccessful_reception_or_transmission,

5.12.3.3 is_activated_on_first_frame_indication :

1. *1. A Callback identified by "CallbackName" shall be activated as a minimumby one message (queued or unqueued) identified by "SymbolicName".*
2. *2. A Callback identified by "CallbackName" shall be activated as a maximumby "MAX_Unqueued_Meesage" or "MAX_Queued-Message" messages identified by their respective "SymbolicName".*

5.12.3.4 is_activated_conditionnally_on_successful_reception_or_transmission :

1. *A Callback identified by "CallbackName" shall be activated as a minimumby one message (queued or unqueued) identified by "SymbolicName".*
2. *A Callback identified by "CallbackName" shall be activated as a maximumby "MAX_Unqueued_Meesage" or "MAX_Queued-Message" messages identified by their respective "SymbolicName".*
3. *A relevant change condition shall be maintained as described in „relevantChange“ of „Mixed Transmission Mode Specification“.*

5.12.3.5 is_activated_by_reception_deadline _monitor :

1. *A Callback identified by "CallbackName" shall be activated as a minimum by one Reception_Deadline_Monitor identified by "RDM_Name".*
2. *A Callback identified by "CallbackName" shall be activated as a maximum by "MAX_Reception_Monitor" Reception_Deadline_Monitor's identified by their respective "RDM_Name".*

5.12.3.6 is_activated_by_transmission_deadline _monitor:

1. *A Callback identified by "CallbackName" shall be activated as a minimum by one Transmission_Deadline_Monitor identified by "TDM_Name".*
2. *A Callback identified by "CallbackName" shall be activated as a maximum by "MAX_Transmission_Monitor" Transmission_Deadline_Monitor's identified by their respective "TDM_Name".*

5.13 Event

This section defines the system generation requirements applicable to event.

EVENT {
Is :
EventName = `enumeration`

Is_Activated_**Unconditionally**_**On**_**Successful**_Reception_Or_Transmission,
Is_Activated_**Conditionnally**_**On**_**Successful**_Reception_Or_Transmission,
Is_Activated_On_**Unsuccessful**_Reception_Or_Transmission,
Is_Activated_On_**First**_**Frame**_**Indication** :
UNQUEUED SymbolicName
(1…MAX_Unqueued_Message)

QUEUED _MESSAGE SymbolicName
(1… MAX_Queued_Message)

Is_activated_by_**reception**_**deadline**_**monitor** :
RECEPTION_MONITOR RDM_Name
(1…MAX_Reception_Monitor)

Is_activated_by_**transmission**_**deadline**_**monitor** :
TRANSMISSION_MONITOR TDM_Name
(1…MAX_Transmission_Monitor)

Is_received_by_Task :
TASK TaskName
(1…MAX_Task)
}(0…MAX_Event)

5.13.1 Entity requirements

The number of event per application is application specific. „MAX_Event“ shall document the maximum number of event supported by an OSEK COM implementation.

5.13.2 Entity attributes requirements

5.13.2.1 EventName:

1. *The name of an Event (EventName) shall be defined at system generation. The type of EventName is enumeration.*

5.13.3 Entity association requirements

5.13.3.1 is_activated_unconditionally_on_successful_reception_or_transmission,

5.13.3.2 is_activated_on_unsuccessful_reception_or_transmission,

5.13.3.3 is_activated_on_first_frame_indication :

1. *an event identified by "eventname" shall be activated as a minimum by one message (queued or unqueued) identified by "SymbolicName".*
2. *An Event identified by "EventName" shall be activated as a maximum by "MAX_Unqueued_Message" or "MAX_Queued_Message" messages identified by their respective "SymbolicName".*

5.13.3.4 is_activated_conditionally_on_successful_reception_or_transmission :

1. *An Event identified by " EventName " shall be activated as a minimum by one message (queued or unqueued) identified by "SymbolicName".*
2. *An Event identified by " EventName " shall be activated as a maximum by "MAX_Message" messages (queued or unqueued) identified by their respective "SymbolicName".*
3. *A relevant change condition shall be maintained as described in „relevantChange“ of „Mixed Transmission Mode Specification“.*

5.13.3.5 is_activated_by_reception_deadline _monitor :

1. *An Event identified by "EventName" shall be activated as a minimum by one Reception_Deadline_Monitor identified by "RDM_Name".*
2. *An Event identified by EventName shall be activated as a maximum by "MAX_Reception_Monitor" Reception_Deadline_Monitor identified by their respective "RDM_Name".*

5.13.3.6 is_activated_by_transmission_deadline _monitor:

1. *An Event identified by EventName shall be activated as a minimum by one Transmission_Deadline_Monitor identified by "TDM_Name".*
2. *An Event identified by EventName shall be activated as a maximum by "MAX_Transmission_Monitor" Transmission_Deadline_Monitors identified by their respective "TDM_Name".*

5.13.3.7 is_received by task :

1. *An Event identified by EventName can be received as a minimum by one Task identified by "TaskName".*
2. *An Event identified by EventName can be received as a maximum by "MAX_Task" Task identified by their respective "TaskName".*

5.14 Flag

This section defines the system generation requirements applicable to flag.

FLAG {
Is :
FlagName = enumeration

Is_Activated_**Unconditionally_On_Successful**_Reception_Or_Transmission,
Is_Activated_**Conditionnally_On_Successful**_Reception_Or_Transmission,
Is_Activated_On_**Unsuccessful**_Reception_Or_Transmission,
Is_Activated_On_**First_Frame_Indication** :
UNQUEUED SymbolicName
(1…MAX_Unqueued_Message)

QUEUED _MESSAGE SymbolicName
(1… MAX_Queued_Message)

Is_activated_by_**reception_deadline_monitor** :
RECEPTION_MONITOR RDM_Name
(1…MAX_Reception_Monitor)

Is_activated_by_**transmission_deadline_monitor** :
TRANSMISSION_MONITOR TDM_Name
(1…MAX_Transmission_Monitor)

} (0…MAX_Flag)

5.14.1 Entity requirements

The number of flag per application is application specific. „MAX_Flag“ shall document the maximum number of flag supported by an OSEK COM implementation.

5.14.2 Entity attributes requirements

5.14.2.1 FlagName:

1. *The name of a Flag (FlagName) shall be defined at system generation. The type of*

"FlagName" is enumeration.

2. *The following format of the flag prototype where "FlagType" is a type for Bit variable shall apply :*

   FlagType <FlagName>;

Example for the usage of a flag:

```
if(ARRIVED== ReadFlag(ACC_MSG_FLAG))
{
  ResetFlag(ACC_MSG_FLAG); / / reset flag
  / / ... do application specific processing here
}
```

5.14.3 Entity association requirements

5.14.3.1 is_activated_unconditionally_on_successful_reception_or_transmission,

5.14.3.2 is_activated_on_unsuccessful_reception_or_transmission,

5.14.3.3 is_activated_on_first_frame_indication :

1. *A Flag identified by "FlagName" shall be activated as a minimum by one message (queued or unqueued) identified by "SymbolicName".*
2. *A Flag identified by "FlagName" shall be activated as a maximum by "MAX_Message" messages (queued or unqueued) identified by their respective "SymbolicName".*

5.14.3.4 is_activated_conditionally_on_ successful_reception_or_transmission :

1. *A Flag identified by "FlagName" shall be activated as a minimum by one message (queued or unqueued) identified by "SymbolicName".*
2. *A Flag identified by "FlagName" shall be activated as a maximum by "MAX_Message" messages (queued or unqueued) identified by their respective "SymbolicName".*
3. *A relevant change condition shall be maintained as described in „relevantChange“ of „Mixed Transmission Mode Specification“.*

5.14.3.5 is_activated_by_reception_deadline _monitor :

1. *An Flag identified by "FlagName" shall be activated as a minimum by one Reception_Deadline_Monitor identified by "RDM_Name".*
2. *An Flag identified by FlagName shall be activated as a maximum by "MAX_Reception_Monitor" Reception_Deadline_Monitor identified by their respective "RDM_Name".*

5.14.3.6 is_activated_by_transmission_deadline _monitor:

1. *An Flag identified by FlagName shall be activated as a minimum by one Transmission_Deadline_Monitor identified by "TDM_Name".*
2. *An Flag identified by FlagName shall be activated as a maximum by "MAX_Transmission_Monitor" Transmission_Deadline_Monitors identified by their respective "TDM_Name".*

5.15 Network handle

This section defines the system generation requirements applicable to network handle.

N_HANDLE {

Is :

N_HandleName = `enumeration`

N_Direction = `enumeration (transmit | receive)`

N_Id = `enumeration`

Is_connected_to_message :

UNQUEUED | QUEUED _MESSAGE SymbolicName

(0…1)

Uses_protocol :

UUDT N_UUDT_Name | USDT N_USDT_Name

(0…1)

Is routed_by_address:

APPLICATION_ADDRESS N_Address

(0…MAX_Application_Address)

}(0…MAX_Network_Handle)

5.15.1 Entity requirements

The number of network handle mode per application is application specific. „MAX_Network_Handle“ shall document the maximum number of network handle supported by an OSEK COM implementation.

5.15.2 Entity attributes requirements

5.15.2.1 N_HandleName :

1. *The name of the network handle (N_HandleName) shall be defined at system generation.*
2. *The type of "N_HandleName " is enumeration.*

5.15.2.2 Direction :

1. *The direction of the information transfer (N_Direction) supported by the network handle (N_Protocol) shall be defined at system generation.*
2. *The type of "N_Direction" is enumeration.*
3. *The reserved values of the "N_ Direction" enumeration type are "Transmit", "Receive".*
4. *A network handle (N_Handle) identified by "N_HandleName " shall be assigned a single reserved value of the "N_Direction" attribute. A network handle is attached to a message that can either be transmitted or received.*

5.15.2.3 N_Id :

1. *The name of the specific underlying bus utilized (N_Id) to transmit the application data attached to the network handle (N_Protocol) shall be defined at system generation.*
2. *The type of "N_Id" is enumeration.*

5.15.3 Entity association requirements

5.15.3.1 is_connected_to_message:

1. *A network handle identified by" N_HandleName" shall connect to one message (queued or unqueued) identified by "SymbolicName".*

5.15.3.2 uses_protocol:

1. *A network handle identified by" N_HandleName" shall be supported either by a UUDT protocol entity identified by "N_UUDT_Name" or by a USDT protocol entity identified by "N_USDT_Name".*

5.15.3.3 is_routed by_address:

1. *A network handle identified by" N_HandleName" shall specify as a minimum no address.*
2. *A network handle identified by" N_HandleName" shall specify as a maximum (N) addresses identified by "N_Address".*

5.16 Application address

This section defines the system generation requirements applicable to a application address.

```
APPLICATION_ADDRESS {
        Is :
                N_Address = scalar
        Routes :
                N_HANDLE N_HandleName
                (1 …MAX_Network_Handle)

} (0…MAX_Application_Addresses)
```

5.16.1 Entity requirements

The number of addresses per application is application specific. „MAX_Application_Addresses“ shall document the maximum number of addresses supported by an OSEK COM implementation.

5.16.2 Entity attributes requirements

5.16.2.1 N_Address :

1. *The name of the network address (N_Address) shall be defined at system generation.*
2. *The type of "N_Address" is scalar.*

5.16.3 Entity association requirements

5.16.3.1 routes :

1. *A network address identified by" N_Address" shall support as a minimum one N_Handle identified by "N_HandleName".*
2. *A network address identified by" N_Address" shall support as a maximum (N) N_Handle identified by "N_HandleName".*

5.17 UUDT

This section defines the system generation requirements applicable to the unacknowledged unsegmented data transfer protocol.

UUDT {

Is :

N_UUDT_Name = `enumeration`

N_Data_Length = `scalar`

activates_on_N_UUData.confimation,

activates_on_N_UUData.indication :

TASK TaskName

(1…MAX_Task) XOR (optional AND)

EVENT EventName

(1…MAX_Event) XOR (optional AND)

CALLBACK CallbackName

(1…MAX_Callback) XOR (optional AND)

FLAG FlagName

(1...MAX_Flag) XOR (optional AND)

FUNCTION FunctionName

(1…MAX_Function) XOR (optional AND)

} (0…MAX_UUDT)

5.17.1 Entity requirements

This entity determines the type of protocol used and its corresponding parameters. „MAX_UUDT“ shall document the maximum number of UUDT protocol specifications supported by an OSEK COM implementation.

5.17.2 Entity attributes requirements

5.17.2.1 N_UUDT_Name :

1. *The name of the UUDT protocol specification (N_Address) shall be defined at system generation.*
2. *The type of "N_UUDT_Name" is enumeration.*

5.17.2.2 N_Data_Length :

1. *The length of the data that will be transmitted onto the communication media shall be defined at system generation.*
2. *The type of "N_Data_Length" is scalar.*

5.17.2.3 activates_on_N_UUData.confirmation,

5.17.2.4 activates_on_N_UUData.indication :

1. *The UUDT protocol entity identified by" N_USDT_Name " can activate as a minimum no FLAG , no CALLBACK , no EVENT, no TASK and no FUNCTION.*
2. *The UUDT protocol entity identified by" N_USDT_Name "can activate as a maximum „MAX_Callback“ CALLBACK identified by their respective "CallbackName" or „MAX_Flag“ FLAG identified by „FlagName” or „MAX_Event“ EVENT identified by "EventName" or „MAX_Task“ TASK identified by their respective "TaskName" and „MAX_Function“ FUNCTION identified by their respective "FunctionName".*

5.18 USDT

This section defines the system generation requirements applicable to the unacknowledged segmented data transfer protocol.

USDT {
Is :
N_USDT_Name = enumeration
N_Data_Length = scalar

BS = scalar
STmin = scalar

N_As_max = scalar
N_Ar_max = scalar
N_Bs_max = scalar
N_Cr_max = scalar

activates_on_N_USData.confirmation,
activates_on_N_USData.indication,
activates_on_N_USData_FF.indication :

TASK TaskName
(1…MAX_Task) XOR (optional AND)

EVENT EventName
(1…MAX_Event) XOR (optional AND)

CALLBACK CallbackName
(1…MAX_Callback) XOR (optional AND)

FLAG FlagName
(1...MAX_Flag) XOR (optional AND)

FUNCTION FunctionName
(1…MAX_Function)

`} (0…MAX_USDT)

5.18.1 Entity requirements

This entity determines the type of protocol used and its corresponding parameters. „MAX_USDT“ shall document the maximum number of USDT protocol specification supported by an OSEK COM implementation

5.18.2 Entity attributes requirements

5.18.2.1 N_USDT_Name :

1. *The name of the USDT protocol specification (N_Address) shall be defined at system generation.*
2. *The type of "N_USDT_Name" is enumeration.*

5.18.2.2 N_Data_Length :

1. *The length of the N_Data filed of the network protocol data unit that is transmitted onto the communication media shall be defined at system generation.*
2. *The type of "N_Data_Length" is scalar.*

5.18.2.3 BS :

1. *„BS“ shall be defined at system generation and assigned to the network protocol control information BS (FC_NPCI.BS).*
2. *The type of "BS" is scalar and shall be within the range of zero (0) to two hundred fifty five (255) decimal.*

5.18.2.4 STmin :

1. *„STmin“ shall be defined at system generation and assigned to the network protocol control information STmin (FC_NPCI.STmin).*
2. *The type of "STmin" is scalar and shall be within the range of zero (0) to two hundred fifty five (255) decimal.*

5.18.2.5 N_As_max, N_Ar_max, N_Bs_max, N_Cr_max :

1. *„N_As_max“, „N_Ar_max“, „N_Bs_max“, „N_Cr_max“,shall be defined at system generation and assigned to their corresponding network protocol timing parameter.*
2. *The type of „N_As_max“, „N_Ar_max“, „N_Bs_max“, „N_Cr_max“,is scalar (milliseconds) with a range that is application specific.*

5.18.2.6 activates_on_N_USData.confirmation,

5.18.2.7 activates_on_N_USData.indication,

5.18.2.8 activates_on_N_USData_FF.indication :

1. *The USDT protocol entity identified by" N_USDT_Name " can activate as a minimum no FLAG , no CALLBACK , no EVENT, no TASK and no FUNCTION.*
2. *The USDT protocol entity identified by" N_USDT_Name "can activate as a maximum „MAX_Callback“ CALLBACK identified by their respective "CallbackName" or „MAX_Flag“ FLAG identified by „FlagName” or „MAX_Event“ EVENT identified by "EventName" or „MAX_Task“ TASK identified by their respective "TaskName" and „MAX_Function“ FUNCTION identified by their respective "FunctionName".*

6 Conformance classes

Various application software requirements and specific system capabilities (e.g. communication hardware, processor, and memory) require different levels of communication software functionality.

OSEK COM defines these levels as „Communication Conformance Classes“ (CCCs). The main purpose of the conformance classes is to ensure that applications which have been for a particular conformance class are portable across different OSEK implementations and ECUs featuring that same or higher level of communication functionality. Hence different implementations of a same communication conformance class provide the same set of services and functionality to the application.

An OSEK COM implementation conforms to a communication conformance class only if it provides all the features defined for that conformance class. However, system generation needs only to link those system services that are required for a specific application. A specific communication conformance class is selected at system generation time[4] and cannot be changed during execution.

OSEK COM defines five communication conformance classes to provide support from ECU internal communication only (CCCA) up to inter-ECU external communication (CCC2).

CCCA :

CCCA defines the minimum features to support internal communication only, i.e. no support for external communication is available. Unqueued message shall be supported. Notification class 1 shall be supported with a maximum number of one notified consumer (task, callback or event only) per message identified by “SymbolicName”. No message status information shall be supported in order to allow for lean implementation of the communication kernel (note that CCCA requires that data consistency is ensured off line if WithoutCopy configuration is utilized since no message resource service is available). SendMessage and ReceiveMessage shall be supported.

CCCB :

CCCB defines features to support internal communication only, i.e. no support for external communication is available. All features of CCCA shall be supported with the following extension : full notification class 1, message status information, Queued messages , GetMessageStatus, GetMessageResource and ReleaseMessageResource services shall be supported.

CCC0 :

CCC0 defines minimum features to support internal, external and internal-external communication. All features of CCCB shall be supported with the exception of queued messages that are optional in CCC0. Operating system support is not required (but optional), notification 2 and UUDT shall be supported.

CCC1 :

[4] Besides, an OSEK COM implementation can be intrinsically compliant with one of the CCCs.

All features of CCC0 shall be supported with then following extensions : Notification class 3 and 4, Mixed and Periodical transmission concepts shall be supported.

CCC2 :

All features of CCC1 shall be supported with the following extensions : USDT protocol shall be supported in both SM/SA and DM/DA configurations. Queued messages shall be supported : a CCC2 implementation shall to be capable to support a FIFO depth of at least 8 messages.

Figure 6-1:Conformance classes summary

	Conformance Class				
Features	**CCCA**	**CCCB**	**CCC0**	**CCC1**	**CCC2**
Unqueued message	√	√	√	√	√
Direct Transmission mode	√	√	√	√	√
SendMessage	√	√	√	√	√
ReceiveMessage	√	√	√	√	√
Notification Class 1	√	√	√	√	√
Queued message		√	*	*	√
GetMessageResource		√	√	√	√
ReleaseMessageResource		√	√	√	√
GetMessageStatus		√	√	√	√
UUDT protocol			√	√	√
Notification Class 2			√	√	√
Periodical				√	√
Mixed				√	√

Notification Class 3				√	√
Notification Class 4				√	√
StartPeriodical				√	√
StopPeriodical				√	√
USDT protocol					√
Notification Class 5					√
SendDynamicMessage					√
ReceiveDynamicMessage					√
SendMessageTo					√
ReceiveMessageFrom					√
ChangeProtocolParameters					√

6.1 OSEK OS support

If an underlying OSEK OS is selected to support OSEK COM then *event setting*, *task activation* and *alarm activation* can be used in CCCA,B,0,1,2 depending on the operating system conformance class chosen to manage the application (see table below). Multiple requesting of tasks may only be required in CCC2, with FIFO size > 1.

Table 6-1: Event setting and task activation

Asynchronous mechanisms	**BCC1**[5]	**BCC2**	**ECC1**	**ECC2**
Activate task on message / on alarm	yes	Yes	yes	yes
Set event on message / on alarm	no	no	yes	yes
Multiple requesting of tasks (FIFO > 1) & Activate task	no	yes	no	only for BT

[5] BCC and ECC are abbreviations for the two types of the OSEK operating system conformance classes : Basic Conformance Class and Extended Conformance Class respectively (see OSEK/VDX OS specification)

7 Annex

7.1 CAN bus binding interface (normative)

7.1.1 Scope

This chapter specifies the binding of the OSEK COM data link layer interface (transfer of data services) to the CAN bus services as defined in the ISO 11898 international standard. This chapter does not target any particular implementation of the CAN standard as it refers to the ISO specification only.

7.1.2 D_UUData.req

This OSEK COM service is defined as follows :

Table 7-1 : D_UUData.req summary

Service name	Service parameter name
D_UUData.req	<D_Handle>
	<D_TA>
	<D_User_Data>

7.1.2.1 Service binding

D_UUData.req service shall be mapped to *L_Data.request* service. A call to *D_UUData.req* shall therefore result with a call to the *L_Data.request* service.

7.1.2.2 Parameters binding

The OSEK COM specification does not require a particular mapping of the D_UUData service parameters which are application specific.

Refer to section 7.2 for the definition of mappings developed to support diagnostic on CAN (ISO 15765-2).

7.1.3 D_UUData.con

This OSEK COM service is defined as follows :

Table 7-2 : D_UUData.con summary

Service name	Service parameter name
D_UUData.con	<D_Handle>
	D_TA
	D_SA
	<D_Result_UUDT>

7.1.3.1 Service binding

D_UUData.con service shall be mapped to *L_Data.confirm* service. A call to *D_UUData.con* shall therefore result with a call to *L_Data.confirm* service.

7.1.3.2 Parameters binding

The OSEK COM specification does not require a particular mapping of the D_UUData service parameters which are application specific.

Refer to section 7.2 for the definition of mappings developed to support diagnostic on CAN (ISO 15765-2).

7.1.4 D_UUData.ind

This service is defined as follows :

Table 7-3 : D_UUData.ind summary

Service name	Service parameter name
D_UUData.ind	<D_Handle>
	<D_SA>
	<D_User_Data>
	<D_Result_UUDT>

7.1.4.1 Service binding

D_UUData.ind service shall be mapped to *L_Data.indication* service. A call to *D_UUData.ind* shall therefore result with a call to *L_Data.indication* service.

7.1.4.2 Parameters binding

The OSEK COM specification does not require a particular mapping of the D_UUData service parameters which are application specific.

Refer to section 7.2 for the definition of mapping developed to support diagnostic on CAN (ISO 15765-2).

7.2 Use of ISO 15765-2 addressing formats (informative)

7.2.1 Scope and concepts

This section describes the mapping of the OSEK COM data link layer service parameters to support addressing formats defined in the ISO 15765-2 (diagnostics on CAN) international standard.

The addressing formats refers to the positioning of the network protocol data unit (including the addressing information) within a CAN bus frame. Several addressing formats are defined in ISO 15765-2 to support the transmission of message using the USDT protocol only.

Each addressing format requires a different number of CAN frame data bytes to encapsulate the addressing information associated with the data to be exchanged. Consequently, the number of data bytes transported within a single CAN frame depends on the type of addressing format chosen.

The following sections describe the mapping mechanisms for each addressing format based on the data Link Layer services and service parameters defined in ISO 11898.

7.2.2 CAN frame data length

This specification does not specify any requirements concerning the length of CAN data frame other than those implied by the size of the network layer protocol data units.

The DLC parameter of the CAN bus frame (as defined in ISO 11898) is set (by the sender) and read (by the receiver) to determine the number of data bytes per CAN frame to be processed by the network layer.

The DLC parameter can not be processed to determine the length of the message to be processed by the network layer : this information shall be extracted from the NPCI byte.

7.2.3 Normal addressing

For each combination of D_Handle, D_TA and D_SA (referenced below as *f*(D_TA/SA/Handle)) a unique CAN identifier is assigned.

D_User_Data is placed into the CAN frame data field. Note that the size of D_User_Data may vary depending on the type of NPDU transported.

Table 7-4:Mapping of NPDU parameters into CAN frame - NORMAL addressing

NPDU type	CAN Identifier	CAN frame data field							
		Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
SF_NPDU	*f*(D_TA/SA/Handle)	D_User_Data							
FF_NPDU	*f*(D_TA/SA/Handle)	D_User_Data							
CF_NPDU	*f*(D_TA/SA/Handle)	D_User_Data							
FC_NPDU	*f*(D_TA/SA/Handle)	D_User_Data			N/A				

7.2.3.1 Normal fixed addressing

Normal fixed addressing is a sub-format of normal addressing where the mapping of the address information (D_Handle, D_TA, D_SA) into the CAN identifier is further defined. In the general case of normal addressing, described above, the correspondence between the addressing information and the CAN identifier is left open.

For normal fixed addressing only 29 bit CAN identifiers are allowed. The following tables define the mapping of the address information into the CAN identifier, depending on the D_Handle (D_Handle), two type of target addresses are defined :

- Physical target address : the CAN frame is addressed to a particular and single node.
- Functional target address : the CAN frame is addressed to all nodes connected onto the CAN bus and shall be processed accordingly

D_User_Data is placed in the CAN frame data field. Note that the size of D_User_Data may vary depending on the type of NPDU transported.

Table 7-5:NORMAL FIXED addressing (physical address)

NPDU type	29 bit CAN Identifier bit position						CAN frame data field byte position							
	28 26	25	24	23 16	15 8	7 0	1	2	3	4	5	6	7	8
SF_NPDU	011 (bin)	0	0	21**8** (dec)	D_TA	D_SA	D_User_Data							
FF_NPDU	011 (bin)	0	0	21**8** (dec)	D_TA	D_SA	D_User_Data							
CF_NPDU	011 (bin)	0	0	21**8** (dec)	D_TA	D_SA	D_User_Data							
FC_NPDU	011 (bin)	0	0	21**8** (dec)	D_TA	D_SA	D_User_Data			N/A				

Table 7-6:NORMAL FIXED addressing, (functional)

NPDU type	29 bit CAN Identifier bit position						CAN frame data field byte position							
	28 26	25	24	23 16	15 8	7 0	1	2	3	4	5	6	7	8
SF_NPDU	011 (bin)	0	0	21**9** (dec)	D_TA	D_SA	D_User_Data							
FF_NPDU	011 (bin)	0	0	21**9** (dec)	D_TA	D_SA	D_User_Data							
CF_NPDU	011 (bin)	0	0	21**9** (dec)	D_TA	D_SA	D_User_Data							
FC_NPDU	011 (bin)	0	0	21**9** (dec)	D_TA	D_SA	D_User_Data			N/A				

7.2.4 Extended addressing

For each combination of D_SA, D_TA and D_Handle (referenced below as *f*(D_TA/SA/Handle)) a unique CAN identifier is assigned. D_TA is placed in the first data byte of the CAN frame data field.

D_User_Data is placed in the CAN frame data field. Note that the size of D_User_Data may vary depending on the type of NPDU transported. Padding requirement are vehicle manufacturer specific.

Table 7-7:Mapping of NPDU parameters into CAN frame - EXTENDED addressing

NPDU type	CAN Identifier	CAN frame data field byte position							
		Byte 1	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**
SF_NPDU	*f*(D_TA/SA/Handle)	D_TA	D_User_Data						
FF_NPDU	*f*(D_TA/SA/Handle)	D_TA	D_User_Data						
CF_NPDU	*f*(D_TA/SA/Handle)	D_TA	D_User_Data						
FC_NPDU	*f*(D_TA/SA/Handle)	D_TA	D_User_Data			N/A			

7.3 Use of ISO15765-2 addressing formats with SAE J1939 (informative)

7.3.1 Overview

This annex describes how to map D_Handle, D_SA and D_TA parameters into the CAN-frame when a data link layer according to SAE J1939 is used.

7.3.2 Rules

7.3.2.1 Normal fixed addressing

The table below shows the mapping of address information parameters into the CAN-frame when physical addressing as indicated by the D_Handle is used.

Table 7-8:Normal addressing, Physical addressed messages

J1939 name	P	R	DP	PF	PS	SA	Data field
Bits	3	1	1	8	8	8	64
Content	default 011 (bin)	0	0	218 (dec)	D_TA	D_SA	D_User_Data
CAN Id Bits	26 - 28	25	24	16 - 23	8 - 15	0 - 7	
CAN data byte							1 - 8
CAN Field	IDENTIFIER						Data

The table below shows the mapping of address information parameters into the CAN-frame when functional addressing as indicated by the D_Handle is used.

Table 7-9:Normal addressing, Functional addressed messages

J1939 name	P	R	DP	PF	PS	SA	Data field
Bits	3	1	1	8	8	8	64
Content	default 011 (bin)	0	0	219 (dec)	D_TA	D_SA	D_User_Data
CAN Id Bits	26 - 28	25	24	16 - 23	8 - 15	0 - 7	
CAN data byte							1 - 8
CAN Field	IDENTIFIER						Data

7.3.2.2 Priority (P)

The priority is user defined with a default value of three (3).

7.3.2.3 Reserved Bit (R)

The reserved bit shall be set to “0”.

7.3.2.4 Data Page (DP)

The data page bit shall be set to “0”.

7.3.2.5 Protocol Data Unit Format (PF)

The format is of the type PDU1, “Destination Specific”. Diagnostic messages shall use the following parameter group numbers (PGN):

— 55808 (dec) for physical addressing, which gives PF = 218 (dec),

— 56064 (dec) for functional addressing, which gives PF = 219 (dec).

7.3.2.6 PDU Specific (PS)

The PDU specific field shall contain the target address (D_TA).

7.3.2.7 Source Address (SA)

The source address field shall content the source address (D_SA).

7.3.2.8 Update rate

Update rate according to user requirements.

7.3.2.9 Data length

Data length shall be eight (8) bytes.

7.4 Format of service primitives (normative)

All network layer services have the same general structure. To define the services, three types of service primitives are specified :

- a service request primitive, used by higher communication layers or the application, to pass control information and data that shall be transmitted to the protocol layer;
- a service indication primitive, used by the network layer, to pass status information and received data to upper communication layers or the application;
- a service confirmation primitive used by the protocol layer to pass status information to higher communication layers or the application.

This service specification does not specify an application programming interface. It only specifies a set of service primitives that are independent of any implementation.

All network layer services have the same general format. Service primitives are written in the form :

service_name.primitive (

parameter A,

parameter B,

parameter C, …

)

where :

"**service_name**" is the name of the service,

"**primitive**" indicates the sort of service type. There are only four types of primitives : request, indication, Response and confirmation. The service primitives define how a service user co-operates with a service provider.

The following service primitives are distinguished:

request :

Using the service primitive *request* (service_name.request) a service user requests a service from the service provider.

indication :

Using the service primitive *indication* (service_name.indication), the service provider informs a service user about

1. an internal event of the network layer or
2. the service request of a peer protocol entity service user, e.g. reception of message in the data link layer: D_UUData.ind.

Response :

The service primitive *Response* (service_name.Response) is used by a service user in order to reply to a preceding "*indication*" from the service provider.

This service primitive is not used by the network layer.

confirmation :

With the service primitive *confirmation* (service_name.confirmation) the service provider informs the service user about the result of a preceding service request of the service user.

"

parameter A,

parameter B,

parameter C …"

is the N_SDU (Network layer Service Data Unit) as a list of values passed by the service primitive.

7.5 Definition of timing symbols (normative)

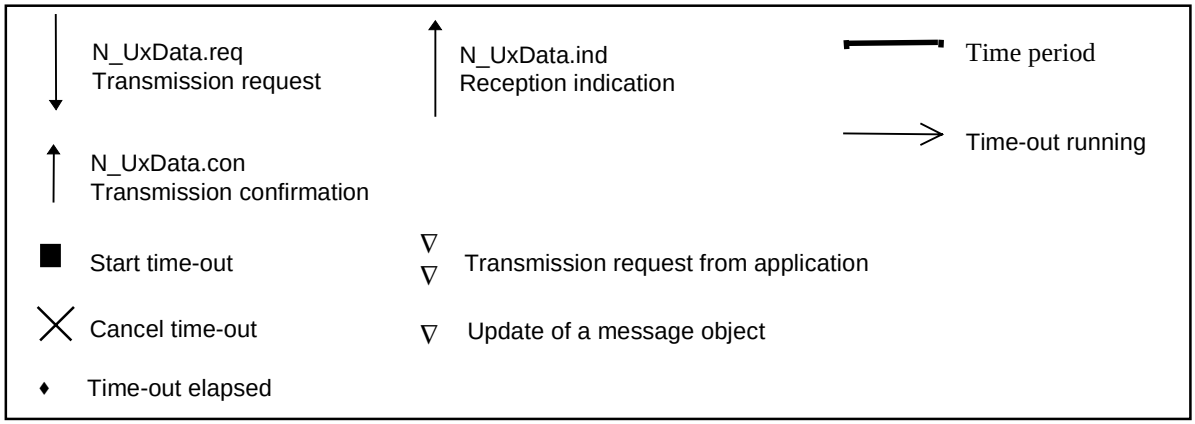

Figure 7-1: Legend of communication deadline monitoring

8 History

Version	Date	Authors	Remarks
1.00	**1995-09-11**		Initial release
		Jörg Graf	Adam Opel AG
		Ferdinand Lersch	BMW AG
		Karl Joachim Neumann	IIIT, University of Karlsruhe
		Willy Roche	Renault
		Hans-Jörg Mathony	Robert Bosch GmbH
		Jürgen Schiemann	Robert Bosch GmbH
		Uwe Zurmühl	Robert Bosch GmbH
		Oliver Friedrichsohn	Siemens AG
		Christoph Hoffmann	Volkswagen AG
2.00	**1997-09-30**		Release version 2.0
		Ferdinand Lersch	BMW AG
		Martin Huber	Daimler-Benz AG
		Helmar Kuder	Daimler-Benz AG
		Martin Reimann	Hella
		Dirk John	IIIT - University of Karlsruhe
		Ansgar Maisch	IIIT - University of Karlsruhe
		Thomas Pietsch	ITT Automotive
		Laurent Roy	LucasVarity
		Andrea Borin	Magneti Marelli
		Sven Larsson	Mecel, / Delco Electronics
		Ken Tindell	NRTT / Volvo
		Eric Farges	Renault
		Lise Massimelli	Renault
		Willy Roche	Renault
		Hans-Jörg Mathony	Robert Bosch GmbH
		Uwe Zurmühl	Robert Bosch GmbH
		Reinhard Laing	S&P MEDIA
		Patrick Palmieri	Siemens Automotive
		Paul Correia	Texas Instruments
		Dietmar Menden	UTA
2.0a	**1997-10-10**		
		Review according to OSEK COM, OSEK OS and MODISTARC remarks: - Requirements note - Chap 3.4 - remove sub-network remarks - Chap 3.5, 3.6 - remove multiple reading remarks - Table 2 - first column renaming - Table 3 - foot note - Table 4 -E_COM_PENDING - Chap 4.1- DataRefType - GetMessageStatus - DefineMessageAlarm - D_GetHandleStatus	
2.1 r1	**1998-06-17**		Release version 2.1
		Andrew Stirling	C&C
		Martin Huber	Daimler-Benz AG
		Helmar Kuder	Daimler-Benz AG
		Martin Reimann	Hella
		Dirk John	IIIT - University of Karlsruhe
		Laurent Roy	LucasVarity
		Lise Mathieu	Renault
		Stephane Korzin	Renault
		Jörg Jehlicka	Robert Bosch GmbH
		Patrick Palmieri	Siemens Automotive
		Gunnar Bennemann	S&P Media
		Fabrice Mendes	S&P Media
		Yves Blanpain	Texas Instruments

		Changes from 2.1 to 2.1r1 : Typing errors corrected
2.2 J	**2000-01-25**	Release version 2.2 draft J
		Andrew Stirling — Cambridge Consultants Frank Leonhardt — Hitachi Micro Systems Europe Dirk John — IIIT - University of Karlsruhe Carsten Thierer — IIIT - University of Karlsruhe Laurent Roy — LucasVarity Stuart Robb — Motorola Jurgen Hofmann — Porsche Lise Mathieu — Renault Stephane Korzin — Renault Jörg Jehlicka — Robert Bosch GmbH Hans-Åke Gustafsson — Stenkil Patrick Palmieri — Siemens Automotive Fabrice Mendes — Telelogic Jerome Charousset — Trialog Hartmut Hörner — Vector Informatik Michael Burke — Visteon
2.2-c-1	**2000-07-21**	Version 2.2 candidate release 1
		Generated from working group document “Cspec2.2 Draft N” with the following amendments: 1. OSEK TC comment : renaming of CCC3 into CCC2, p:17, 172, 173. 2. ISO comment : correct that N_OK can be generated on both the sender and receiver side in page 93, 94.
2.2	**2000-07-28**	Release Version 2.2
		Generated from 2.2-c-1 with no requirement change.
2.2.1	**2000-09-06**	Release Version 2.2.1
		Generated from 2.2 with correction of figure 6-1 (figure 6.1 in release version 2.2 has been corrupted during the generation of the pdf-file).
2.2.2-c-1	**2000-12-08**	Version 2.2.2 candidate release 1
		Generated from 2.2.1 with the following amendments and corrections raised by certification : 1. Remove statement requiring that E_COM_NOMSG shall be returned by the receive services if no unqueued message has been received aftre initialisation : revisit paragraph 3 of 2.2.8.3.2, remove E_COM_NOMSG in figure 2-10. 2. Add indication that status codes shall be supported if the osek com implementation is utilised in ‘extended’ mode : add ‘and extended’ in 2.2.12.3.1, 2.2.12.3.2, 2.2.12.3.3, 2.2.12.3.5, 2.2.12.3.6, 2.2.12.3.7, 2.2.12.4.1, 2.2.12.4.2.
2.2.2	**2000-12-18**	Release Version 2.2.2
		Generated from 2.2.2-c-1 with no requirement change.