OSEK/VDX

Fault-Tolerant Communication

Version 1.0

July 24th 2001

Preface

OSEK/VDX is a joint project of the automotive industry. It aims at an industry standard for an open-ended architecture for distributed control units in vehicles.

For detailed information about OSEK project goals and partners, please refer to the “OSEK Binding Specification”.

This document describes the concept of a fault-tolerant communication layer. It is not a product description which relates to a specific implementation. This document also specifies the fault-tolerant communication layer - Application Program Interface.

General conventions, explanations of terms and abbreviations have been compiled in the additional inter-project "OSEK Overall Glossary". Regarding implementation and system generation aspects please refer to the "OSEK Implementation Language" (OIL) specification.

Table of Contents

1 Introduction

The specification of the fault-tolerant communication layer (FTCom layer) is to represent a uniform functioning environment which supports efficient utilisation of resources for automotive control unit application software.

1.1 System Philosophy

The objective of the OSEKtime working group is to specify a fault-tolerant real-time operating system with a fault-tolerant communication layer as a standardised run-time environment for highly dependable real-time software in automotive electronic control units. The OSEKtime system must implement the following properties:

- predictability (deterministic, a priori known behaviour even under defined peak load and fault conditions),
- clear, modular concept as a basis for certification,
- dependability (reliable operation through fault detection and fault tolerance),
- support for modular development and integration without side-effects (composability), and
- compatibility to OSEK/VDX OS.

The OSEKtime operating system core offers all basic services for real-time applications, i.e., interrupt handling, dispatching, system time and clock synchronisation, local message handling, and error detection mechanisms.

All services of OSEKtime are hidden behind a well-defined API. The application interfaces to the OS and the communication layer only via this API.

For a particular application the OSEKtime operating system can be configured such that it only comprises the services required for this application (the OSEKtime operating system is described in the OS specification).

OSEKtime also comprises a fault-tolerant communication layer that supports real-time communication protocols and systems. The layer offers a standardised interface to the following communication services and features: a global message handling service (comprising replication and agreement support, and transparent access to the communication system), start-up and reintegration support, and an external clock synchronisation service.

1.2 Purpose of this Document

The following description is to be regarded as a generic description which is mandatory for any implementation of the OSEKtime FTCom layer. This concerns the general description of strategy and functionality, the interface of the function calls, the meaning and declaration of the parameters and the possible error codes.

The specification leaves a certain amount of flexibility. On the one hand, the description is generic enough for future upgrades, on the other hand, there is some explicitly specified implementation-specific scope in the description.

It is assumed that the description of the OSEKtime FTCom layer is to be updated in the future, and will be adapted to extended requirements. Therefore, each implementation must specify which officially authorised version of the OSEKtime FTCom description has been used as a reference description.

Because this description is mandatory, definitions have only been made where the general system strategy is concerned. In all other respects, it is up to the system implementation to determine the optimal adaptation to a specific hardware type.

1.3 Structure of this Document

In the following text, the essential specification chapters are described briefly:

Chapter 2, Summary

This chapter provides a brief introduction to the OSEKtime FTCom layer, gives a survey about the interactions between OSEKtime layers and assumptions on the communication protocol.

Chapter 3, Message Handling

This chapter describes the message handling.

Chapter 4, Other FTCom Functions

This chapter describes the recommended practice for implementing time services, external clock synchronisation, membership service, lifesign update and start-up.

Chapter 5, Inter-task Communication

This chapter contains a description of the inter-task communication.

Chapter 6, Specification of FTCom System Services

This chapter contains a description of the FTCom layer API.

Chapter 7, Hints

This chapter describes recommendations which are not part of the specification.

Chapter 8, Index

List of all FTCom system services, figures and tables.

Chapter 9, History

List of all versions.

2 Summary

The fault-tolerant communication layer (FTCom layer) is responsible for the interaction between the communication controller hardware and the application software. It provides the necessary services to support fault-tolerant highly dependable real-time distributed applications (e.g. start-up of the system, message handling, state message interface).

The OSEKtime FTCom layer is built in accordance with the user's configuration instructions at system generation time.

2.1 Architecture of a OSEKtime System

In a time-triggered system the application software uses the interface provided by the operating system and by the fault-tolerance layer. The operating system is responsible for the on-line management of the CPUs resources, management of time and task scheduling. The FTCom layer is responsible for the communication between nodes, error detection and fault-tolerance functionality within the domain of the communication subsystem.

Figure 2-1 shows the architecture of a OSEKtime system. Application software and FTCom Layer are executed under control of the operating system. OSEK/VDX Network Management (NM) describes node-related (local) and network-related (global) management methods. The global NM component is optional and described in the OSEK/VDX NM specification.

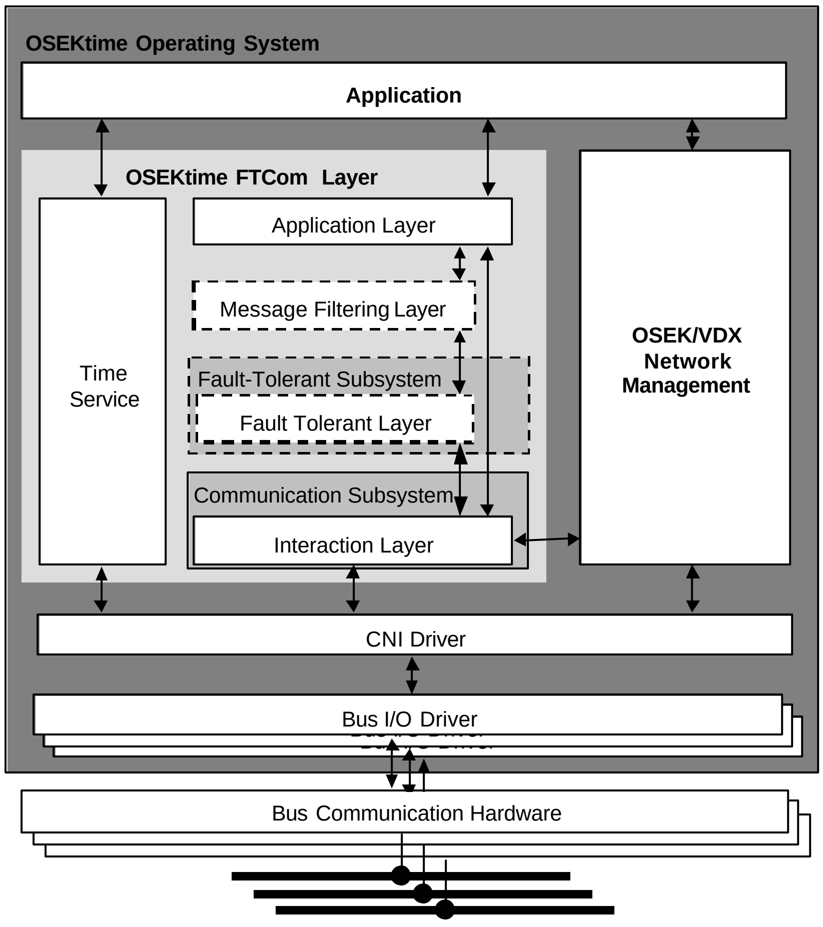

Figure 2-1: Architecture of a OSEKtime system

Services of the FTCom Layer

The Services of the FTCom layer are listed below:

- Global message handling
  - Replication and agreement
  - Message filtering
  - Communication controller communication network interface (CNI) access via CNI driver (incl. connections to multiple communication media, e.g., gateways)
- Start-up
- Time service and optional external clock synchronisation

Layered Model of OSEKtime FTCom Architecture

The layered model of OSEKtime FTCom architecture is shown in Figure 2-2. The OSEKtime FTCom system is divided into two subsystems:

- Firstly the Fault Tolerant Subsystem that contains fault tolerant mechanisms; and
- secondly, the Communication Subsystem that is responsible for the communication between distributed components.

FTCom is also divided into layers:

- Application Layer:
  - Provides an Application Programming Interface (API).
- Message Filtering Layer:
  - Provides mechanisms for message filtering.
- Fault Tolerant Layer:
  - Provides services required to support the fault-tolerant functionality:
    - Provides judgement mechanisms for message instance management.
    - Supports a message status information.
- Interaction Layer:
  - Provides services for the transfer of message instances via network:
    - Resolves issues connected with the presentation of a message instance on different hosts (e.g. different byte ordering).
    - Provides a message instance packing/unpacking service.
    - Supports a message instance status information.

The CNI Driver is not part of FTCom. It provides services for the transfer of FTCom frames via network:

- Resolves FTCom CNI frames presentation issues.
- Supports a FTCom frame status information.
- Deals with a specific CNI access scheme of a particular implementation of the communication hardware.

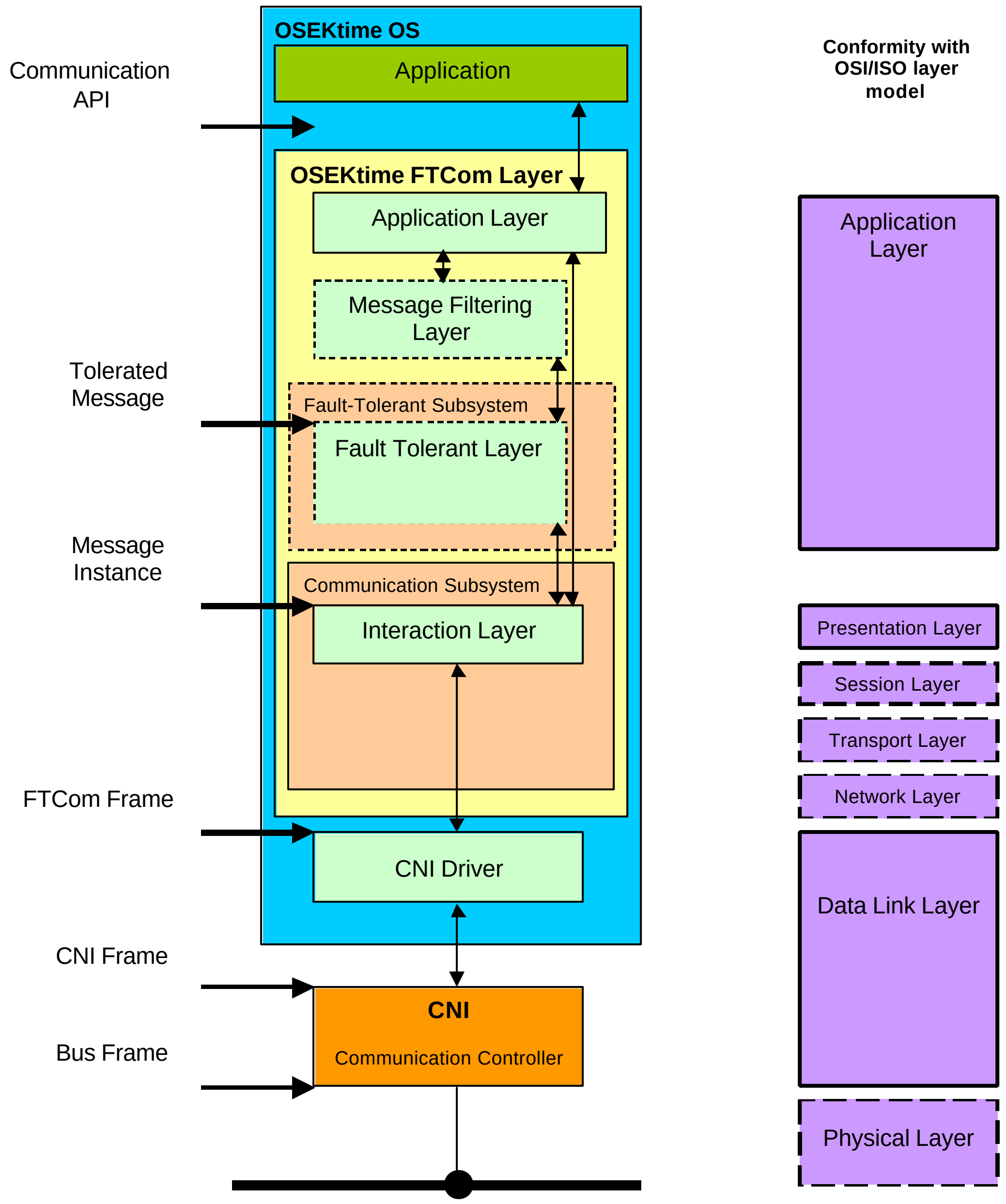

Figure 2-2: Layered model of OSEKtime FTCom architecture

2.2 Constraints on the FTCom and the underlying Communication Controller

Constraints on the FTCom and the underlying communication controller are:

- The fundamental basis for real-time and time-triggered systems is a ***globally synchronised clock*** with sufficient accuracy. The globally synchronised clock must be accessible and it must provide means to generate programmable time-interrupts.
- ***Error detection*** must be supported in the event of data corruption. In addition the communication protocol must support the detection of missing, late or early messages at the receiver(s) and the senders.
- ***Time-triggered, periodic frame transmission*** is assumed for all messages handled by the FTCom layer. Other types of transmission must be handled implementation specific.
- ***Defined Worst Case Start-up Time:*** The communication system must have a deterministic worst-case start-up time.

2.3 Message Exchange Interface

The FTCom layer is based on a state message interface: the send operation overwrites the last recent valid message value, while read operations get the most recent value.

The API calls *"ttReceiveMessage"*, *"ttSendMessage", and "ttInvalidateMessage"* (definition in section 6.3) are mandatory and the standard way to consistently exchange data between application and the FTCom layer. No other message access is allowed for the user (programmer). Every call causes a new consistent access of the FTCom interface.

3 Message Handling

The communication controller transmits *frames* typically consisting of a start-of-frame field, a header, a data field, and a CRC checksum on the communication media. Each frame can hold one or more application level *messages* in its data field. On the other hand, a message can be transmitted redundantly in more than one frame on the communication media. It is the main task of the FTCom layer to handle this relationship and the transport of messages between the application tasks and the communication network interface of the communication controller. The layout of a frame is user specified.

It might not be possible for the application tasks to use application messages in the representation as they are transmitted on the communication media and as they are also stored in the CNI:

- They are densely packed (i.e., not byte-aligned) to save communication media bandwidth,
- their byte order might be different from that of the receiver, and
- messages might be transmitted redundantly, so that selection of one message or voting on a set of messages becomes necessary.

Therefore, each message sent or received by a node is stored exactly once and in the local CPU's representation in a dedicated memory area under control of the host CPU. This memory area is called the FT-CNI. From there it can be accessed by the application tasks. Consequently, there are two representations of messages:

- Firstly, a message is represented in the FT-CNI. This representation should match the requirements of the host CPU and is based on the state message concept. For example, on a 16 Bit CPU it will be optimal to represent a 10 bit analogue conversion result by a 16 bit word.
- Secondly, a message is represented in frames as handled by the communication controller. This representation should match the properties of the communication controller. For example, to utilise communication bandwidth it is ideal to transmit only 10 bits of information for a 10 bit analogue conversion result.

Furthermore the FTCom layer provides a systematic approach to apply different filter algorithms on messages transferred from the CNI to the FT-CNI and vice versa.

The transport between the CNI and the FT-CNI is handled by *message copy tasks* that are invoked after reception of a frame and before sending a frame, respectively. Ideally, they are part of the time-triggered task schedule. From what was said above it follows that the main job of a message copy task is (1) to do message alignment, (2) to convert between communication media and local byte order (endianness), (3) to select or vote on redundant messages, and (4) to filter messages.

Figure 3-1 shows the relationship between the CNI, the message copy tasks of the FTCom layer, the FT-CNI, and the application tasks. The CNI holds the data fields of all frames as they are transmitted on the communication media. The message copy tasks of the FTCom layer disassemble the received frames and assemble the frames to be sent, and copy the messages to and from the FT-CNI, where they can be accessed by the application tasks.

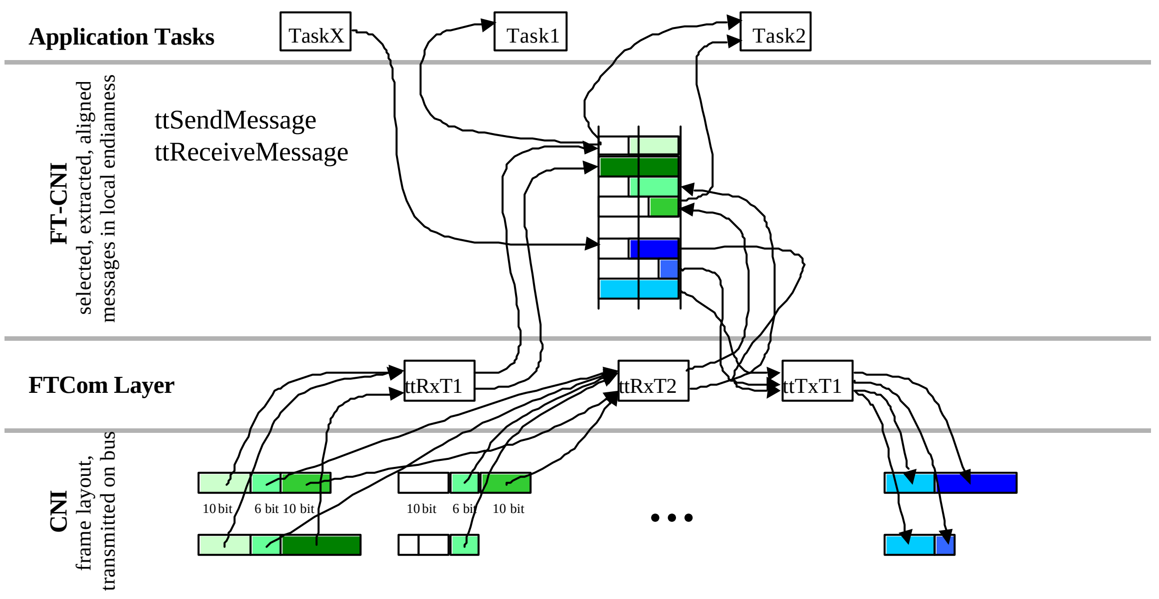

Figure 3-1: CNI, Message Copy Tasks, FT-CNI, and Application Tasks

3.1 Messages and Message Instances

In the description above the term “message” was used for all entities, whether they reside in the FT-CNI, the CNI or are transmitted on the communication media. To be more precise in the remainder of this specification the following notions of a message will be distinguished:

Message:	A block of application data (signals) stored in the FT-CNI. Messages, having the same name, can be sent by different nodes.
Message Instance:	One copy of a message stored in the CNI (transmitted on the communication system) at the sender. At the receiver these message instances may be used to generate a new single message, e.g., by using predefined agreement algorithms (RDAs).

3.2 Message Copy Functions

There are two types of message copy functions: the functions for receiving messages are different to the functions invoked before sending a message.

3.2.1 Receiving Messages

The message copy function for receiving messages has to perform the following actions:

- It first has to read all relevant frames from the CNI and do byte order (endianness) conversion, if necessary. “Relevant frames” means all frames that contain an instance of any message handled by this message copy task.
- Evaluate frame status fields and discard all frames with an invalid status.
- For each message, a copy must be created from a valid frame by aligning the relevant portion of the frame data field to suitable boundaries for the used CPU, and - if necessary - masking out all parts of other messages.

- This copy must be written to the FT-CNI.

3.2.2 Sending Messages

The message copy functions for assembling messages to be sent on the communication media must do the following:

- It must read all messages to be transmitted from the FT-CNI.
- For each frame, it must then align the message instances to their position in the frame data field, and then assemble the frame.
- The byte order (endianness) must be converted to the communication media byte order, if necessary.
- The function must then copy the assembled frame data field to the CNI.
- In case of an event-driven communication system, the transmission of a frame is suppressed if all message instances of a frame have been invalidated by the application (i.e., contain an invalid send status (see Section 3.6)).

3.3 Message Frame Mapping

The communication controller transmits frames up to a certain length. One frame may contain one or more message instances. In order to support fault-tolerance one message is carried by one or more frames (i.e., one instance of the message per frame).

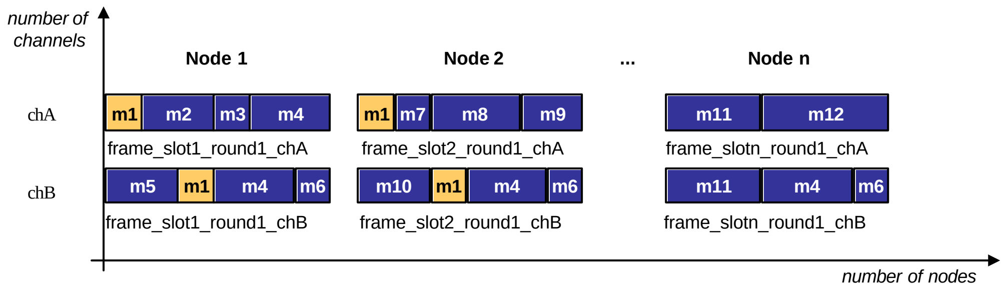

Figure 3-2: Example frame layout for a two-channel system

Figure 3-2 shows a configuration of a system with two channels (chA and chB). Each frame is named based on the slot and round number. The example shows a message m1 which is transmitted by two nodes in slot 1 and slot 2 on two channels. Therefore message m1 is mapped to four frames in one round.

The message frame mapping is static and is defined offline. The mapping between messages and frames adheres to the following rules:

- One message is carried by at least one frame.
- One frame carries 0 ... max_frame_size[1] message instances.

[1] In units of bits

- One message is carried at most once in a frame (i.e., one frame does not contain more than one instance of the same message).

Remark: It is possible that a frame is completely or partially empty and thus reserves space for future usage.

3.4 Packing/Unpacking Messages

If cost constraints require an optimal use of communication bandwidth, it is necessary to pack messages into frames with bit granularities. On the other hand, if communication bandwidth is not an issue, application messages can be transmitted unpacked.

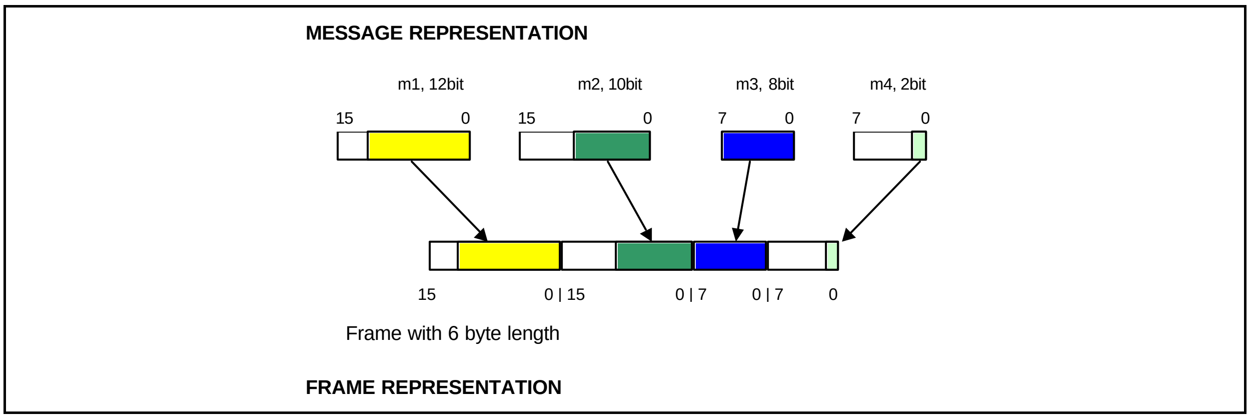

Figure 3-3: Example of direct message to frame mapping

For example, a 10 bit analogue/digital conversion result or status bits could be represented in a frame only by the necessary number of bits or by a full 16 bit value. The communication layer should provide the unpacked messages aligned with the CPUs word length (byte, word, long word) to optimise access independent of the message length.

At the frame level there are three types of message representation supported. A direct unpacked representation, a standard packed linear representation and an alternate packed representation (see Figure 3-3, Figure 3-4 and Figure 3-5).

Below, for both packed representations it is shown in which way four 16 bit word aligned messages are packed into a frame. The way a message is packed into frames is defined at system configuration phase.

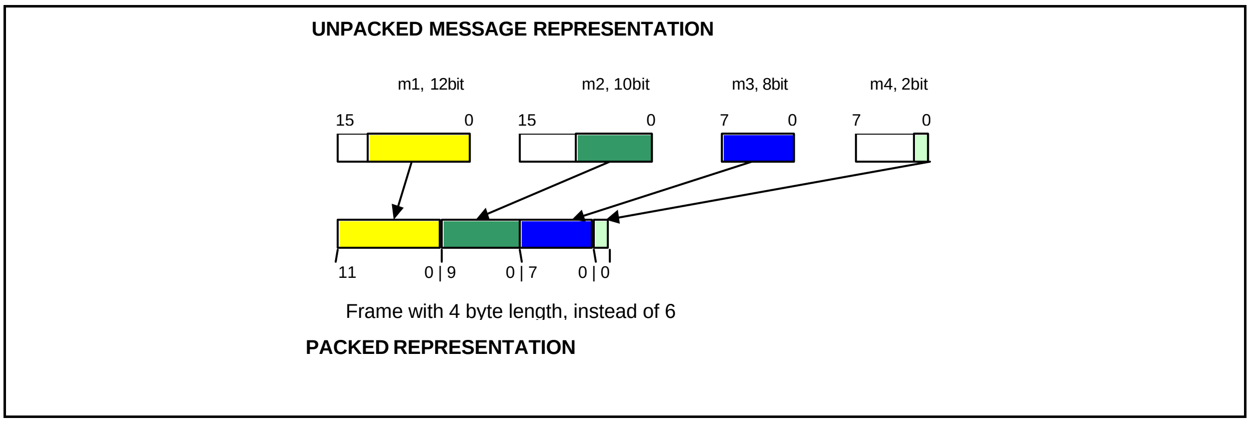

Figure 3-4: Example for standard message to frame mapping

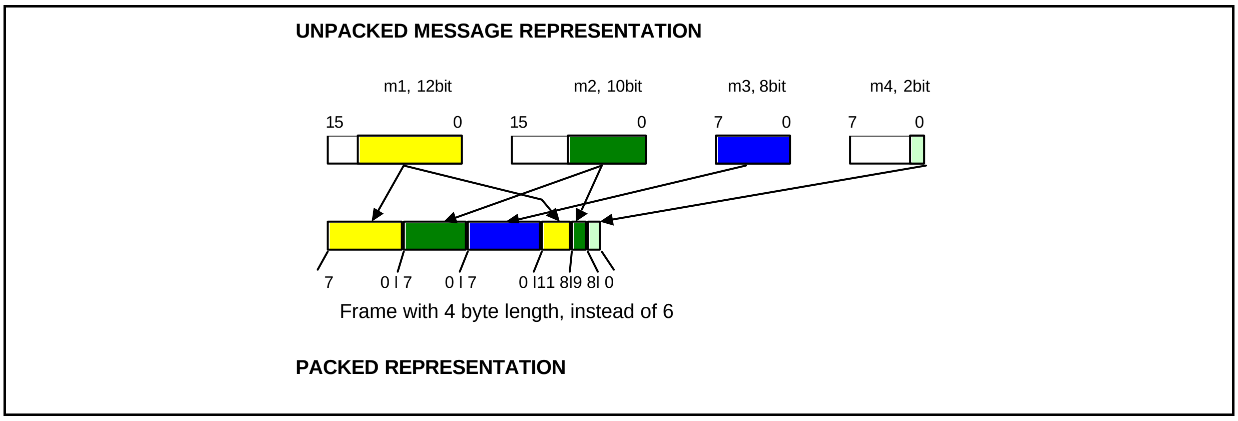

Figure 3-5: Example for alternate message to frame mapping

The standard message to frame mapping must be supported; the alternate message to frame mapping is optional.

For messages with bit granularities the mapping has the following properties:

- One message maps to at least one frame representation
- One frame representation consists of at least one bit array

3.5 Byte Order

In heterogeneous clusters with different CPUs and different interoperable communication controllers it is important to consider the byte order of the CPU (e.g., big or little endian) and on the communication media. The FTCom layer is responsible for the byte order conversion between the local CPU and the communication media.

3.6 Message Send Status

The sender must have a mechanism to present the validity state of a data value (for instance a sampled sensor value) to all nodes in the network. This *can* be realised, if the sender of a message can mark this value as invalid in the FT-CNI by a send status bit. The send status bit mechanism is optional, since a message can be marked as invalid by other means as well (e.g., by assigning a predefined invalid value by the application). If the send status bit is present and cleared, this marks the message as invalid. The send status bit will be copied by the FTCom copy task into all frames transmitting an instance of message. This allows the sender FTCom task to collect multiple message instances and pack them into a frame even if some of the associated messages are invalid.

If all message instances of a given frame are marked as invalid, the transmission of the frame is suppressed in case of an event-driven communication system.

To mark a message as invalid and send the message the function call *ttInvalidateMessage* is used (notice: *ttSendMessage* is not called in this case!). At the receiver side the function call *ttReceiveMessage* of an invalidated message returns the error code TT_E_FTCOM_MSG_INVALIDATED. If an invalidated message has been received the current instance of the message in the FT-CNI represents the last message value, which has been passed to the application.

To mark a message as invalid different configurations are possible, e.g. invalidate value, invalidate flag, etc.

3.7 Notification Mechanism

The following notification mechanism, which does not require the support by an underlying operating system, will be provided. The interaction layer sets a flag after the communication controller has consumed the message (i.e., the flag indicates that new data can be written to the communication controller's transmit buffer associated with the message without causing an unsent instance of the message to be overwritten).

In case of replicated messages, the flag indicates that all *local* instances (i.e., instances transmitted by the respective node) have been consumed by the node's communication controller(s).

The current value of the flag can be checked by the application by means of the *ttReadFlag* API service. The resetting of the flag is implicitly performed by the *ttSendMessage* API service.

3.8 Replication/Redundancy

The communication layer has to support fault-tolerant data transmission between nodes. Fault-tolerance is based on redundant communication channels and replicated nodes. Therefore, a message is transmitted over redundant channels by replicated nodes. Based on its configuration data, for receiving a message the communication layer has the information where to pick up the message information. It evaluates the receive status of each message instance and presents one copy to the application software. On the contrary, for sending a message, data is picked up from the application software and copied into all relevant frames. These activities are carried out by dedicated communication layer tasks that are executed by the operating system.

If a message is sent by more than one node then the FTCom layer must take care to ensure that only consistent data (for instance data which is sampled at the same point in time) is used. For replicated nodes messages consistency requires that the instances of the message are only accessed once all instances have been updated with logically corresponding values, for example values that are sampled at the same point in time (see Figure 3-6).

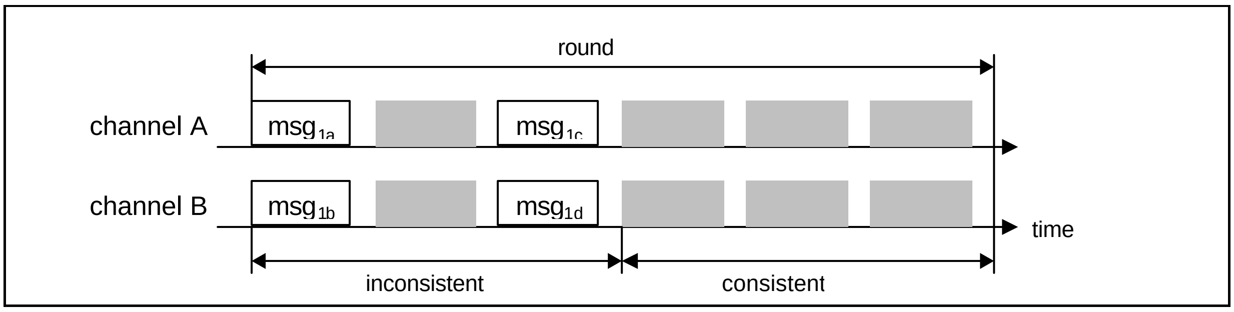

Figure 3-6: Consistency of replicated messages

3.9 Replica Determinate Agreement (RDA)

Optionally, the communication layer can support the application software by providing predefined agreement algorithms and a framework for user defined agreement algorithms. The agreement algorithms are responsible for how to represent messages to the application software from a set of redundant and replicated message instances. Based on the failure mode assumption an appropriate agreement algorithm can be selected.

For most replicated messages encountered in distributed applications, only a few RDAs are of importance, e.g., “pick any” for fail-silent replica-determinate messages, and “average” for values from redundant sensors. But in some applications, special RDA functions become necessary and need to be implemented in a systematic way. Therefore a generic way to describe the calculation of an RDA is required. Such a generic way is described by the following four steps:

1. Declaration

The counters, variables, and arrays required for the other steps are defined here. For this step, the number of instances of the message needs to be known in case an array for all instances is defined (e.g., for diagnosis purposes, or some RDAs like “majority vote”).

2. Initialisation

This step is executed at the beginning of the agreement of the message, i.e., before the first raw value of the message is processed. The counters and buffers are initialised with their initial values.

3. Next Value

This step is executed once for each instance that is correctly received. Instances that fail to be received correctly (e.g., because the sender failed to send, or because the transmission carrying the value was mutilated and resulted in a CRC error) are not processed in a “next value” step.

The number of “next value” steps therefore depends on the number of correctly received instances and is bounded by the replication degree of the message. In the extreme case, no “next value” step is executed between the “initialisation” step and the “finish computation” step.

4. Finish Computation

This step is executed at the end of the message retransmission interval, i.e., after the last instances of the message is processed. This step generates the final result of the RDA. If the agreement fails (either because no instances were received, or because the raw values received do not allow a result (e.g., a “majority vote” over only two different values)) the status of the agreement will be set to TT_E_FTCOM_RDA_FAILED.

3.9.1 Message RDA Status

The FTCom layer provides status information on the correctness of received messages to the application. The function call *ttReceiveMessage* returns the error code TT_E_FTCOM_RDA_FAILED if the RDA mechanism was not successful. The status becomes valid if all of the following conditions are true:

- at least one of the frames carrying an instance the message is valid
- the RDA (if applicable) did yield a valid result.

3.9.2 Example: RDA “average”

Declaration:

```
int counter; int sum;
```

Initialisation:

```
counter  = 0;
sum      = 0;
```

Next Value:

```
counter  = counter + 1;
sum      = sum + value;
```

Finish Computation:

```
if counter > 0 :
    result = sum / counter;
    RDA status is VALID
else :
    RDA status is INVALID
```

3.9.3 Example: RDA “majority vote”

Declaration:

```
int counter; int values[];
```

Initialisation:

```
counter = POSITION_ONE;
```

Next Value:

```
values[counter] = value;
counter = counter + 1;
```

Finish Computation:

```
      if counter > POSITION_ONE:
            operating on values[POSITION_ONE .. counter-1] do:
                  sort values;
                  find largest group of identical values;
                  find second largest group of identical values;
                  if size of largest group is greater than
                        size of second largest group
                        or there is only one group of values :
                        result = value of largest group;
                                   RDA status is VALID else (the two largest
            groups are of equal size):
                        result = NO_RESULT;
                     RDA status is INVALID
else:
                      RDA status is INVALID
```

3.10 Message Filter

The FTCom layer provides optional filter algorithms to support the user with data handling. These algorithms could be used both with the sending and with the receiving of predefined messages. The internal structure of FTCom can be seen in Figure 2-2.

3.10.1 Message Filter Function

The message filter is an offline configurable function layer, which filters messages out according to specific algorithms. For each message a different filtering condition can be defined through a dedicated algorithm.

While sending messages the message filter will pass the current message value to the interaction layer whenever the appropriate filtering condition is met (see Figure 3-7 A). All other message values will be filtered out. When this occurs, the message is marked as invalidated.

While receiving the messages, only the message values which meet the algorithms will be passed to the application as such the FT-CNI will be updated (see Figure 3-7 B). In parallel a status for the application will be provided by the message filter, which indicates whether the last value has been filtered out, or passed. If the value has been filtered out the current instance of the message in the FT-CNI represents the last message value, which has passed the message filter.

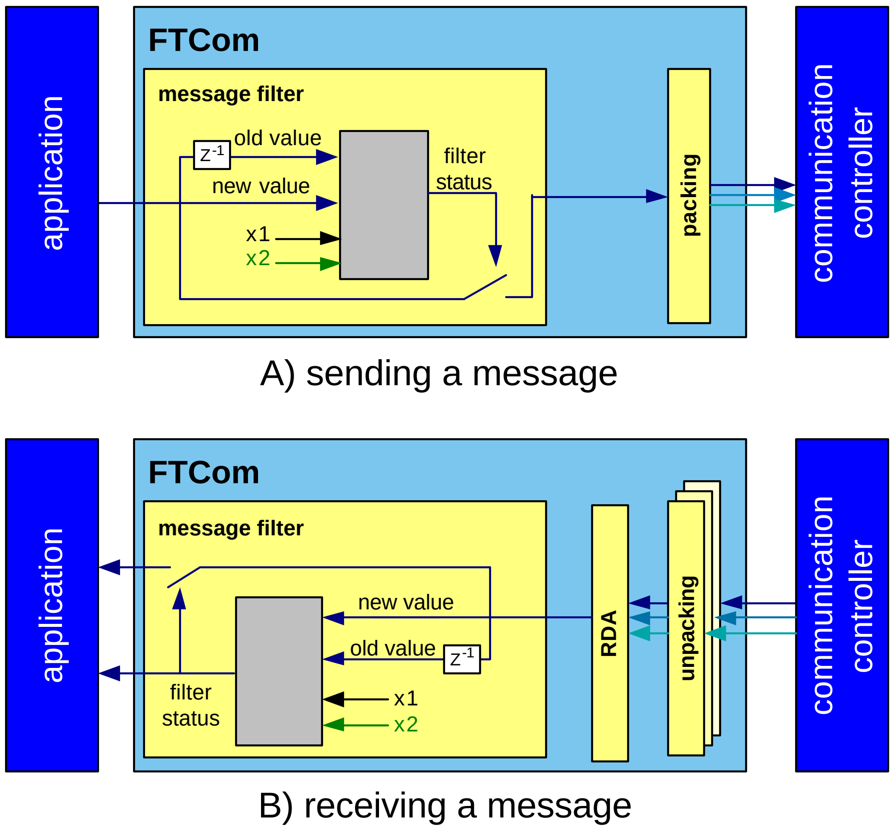

Figure 3-7: Message filter

For message filtering a set of 14 generic algorithms as well as a framework for user defined algorithms is provided. The generic algorithms are all optional.

The following attributes are used by the 14 generic algorithms (see Table 3-1):

new_value:	current value of the message
old_value:	last value of the message
x1, *x2*:	two constant values, which can be defined in offline tools to configure the message filter

Algorithm	Description
True	Passing messages in any case without using the message filter
False	Disabling of the appropriate messages
(new_value&x1) == x2	Passing messages whose masked value is equal to a specific value
(new_value&x1) != x2	Passing messages whose masked value is not equal to a specific value
new_value == old_value	Passing messages which have not changed
new_value != old_value	Passing messages which have changed
(new_value&x1) == (old_value&x1)	Passing messages where the masked value has not changed
(new_value&x1) != (old_value&x1)	Passing messages where the masked value has changed
x1 <= new_value <= x2	Passing messages if its value is within a predefined boundary
(x1 > new_value) OR (new_value > x2)	Passing messages if its value is outside a predefined boundary
new_value > old_value	Passing messages if its value has increased
new_value <= old_value	Passing messages if its value has not increased
new_value < old_value	Passing messages if its value has decreased
new_value >= old_value	Passing messages if its value has not decreased

Table 3-1: Basic algorithms of the message filter

If the attribute message filter is *True* for any particular message no filter algorithm is included in the runtime system for the particular message.

3.10.2 Message Filter Status

The FTCom layer provides information on the filter status of received messages to the application. Therefore the service call *ttReceiveMessage* returns the error code TT_E_FTCOM_MSG_NOT_RECEIVED, if the last value of a message has been filtered out (the received message has not been forwarded by the message filter to the application during the last execution).

3.11 Overview Message Handling API

The FTCom layer provides status information on the validity of received messages to the application. To get an overview on message handling at the sender and at the receiver see Figure 3-8. The function call *ttReceiveMessage* returns the status of a received message, depending on its configuration. *ttReceiveMessage* returns only one status code, therefore the error codes are prioritised in the following way:

1. TT_E_FTCOM_MSG_NOT_RECEIVED, no frame of the message has been received or the value of the message has not been forwarded by the message filter of the receiver during the last execution (only relevant if message filtering is configured).
2. TT_E_FTCOM_RDA_FAILED, message instance(s) have been received but the RDA calculation has no valid result (only relevant if RDA is configured).

If an invalidated message is transmitted, the function call *ttReceiveMessage* returns the error code TT_E_FTCOM_MSG_INVALIDATED.

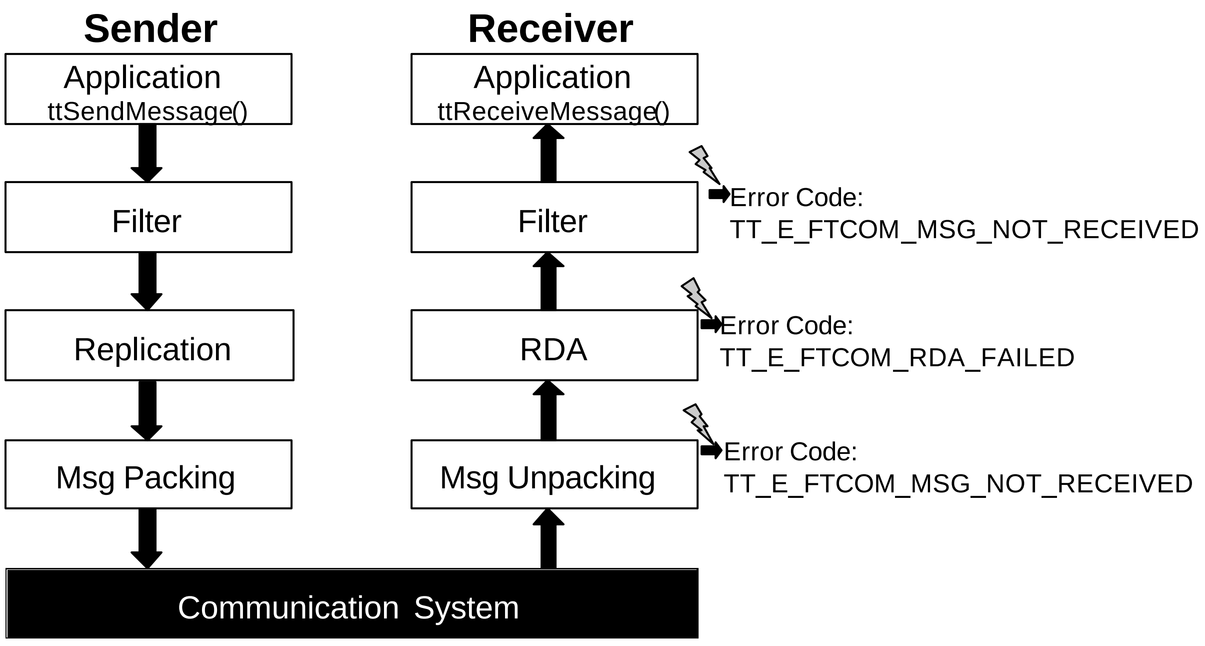

Figure 3-8: Overview of Message Handling API

Figure 3-9 shows the different ways of how the different layers of FTCom can be used during sending and receiving of messages. In FTCom the use of the Fault Tolerant Layer and the Filter layer is optional. Due to runtime and code size constraints it could be more efficient not to call these layers if they are not configured. The Fault Tolerant and Filter layer can also be used for internal communication (left side of Figure 3-9). A voting of mixed external and internal messages is possible as well.

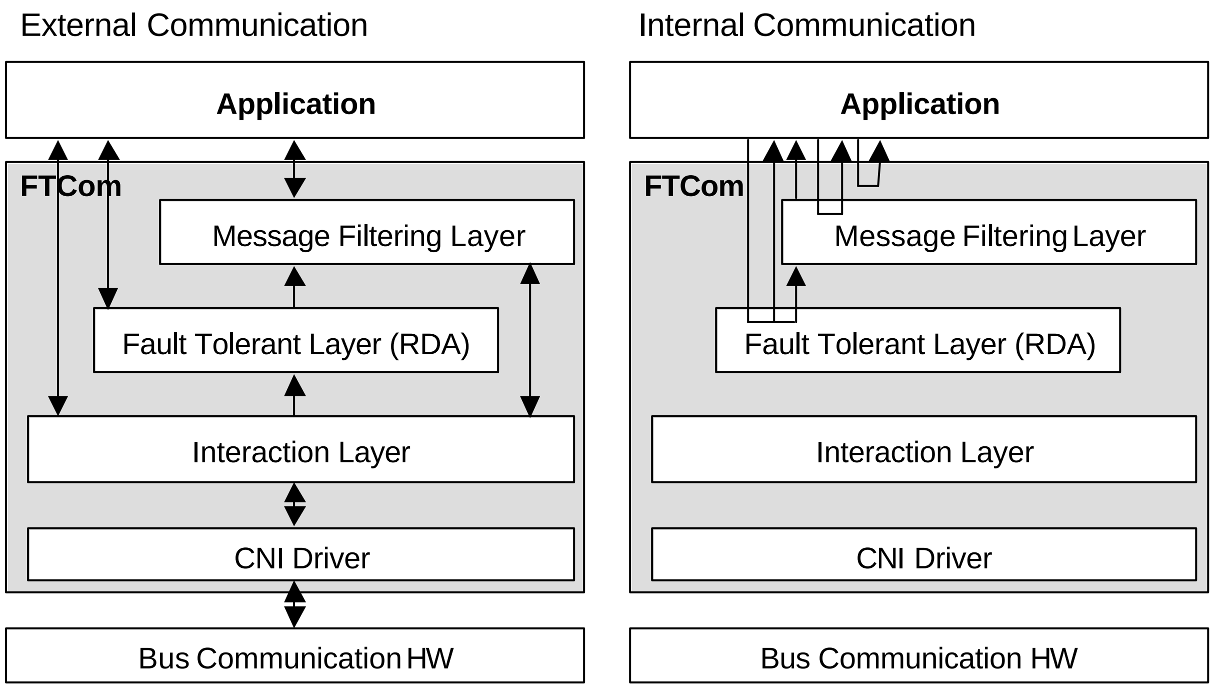

Figure 3-9: Communication paths

4 Other FTCom Functions

4.1 Time and Synchronisation Services

One of the assumptions on the underlying communication system is that a globally synchronised clock is provided. Time service is a function which depends on the used communication protocol and can only be implemented with detailed knowledge of the communication protocol. However a generic API call has to be provided by the FTCom layer (see chapter 0).

4.1.1 Assumptions

Several assumptions can be made concerning the underlying communication system and the time-triggered application:

- Communication on the communication media is structured in communication rounds which consist of several communication slots. Within each slot one communication frame is transmitted which contains one or more message instances.
- Application tasks are running synchronous to communication slots to receive and send messages with deterministic latency.
- The dispatcher round is a multiple of the communication round. A dispatcher table that is shorter than a communication round (e.g., half as long) can be replaced by a dispatcher table of equal duration by means of multiple task scheduling.
- If the dispatcher round is larger than the communication round it's necessary to distinguish between the communication rounds to synchronise applications running on different ECUs. For example, if an application is running on four ECUs, which read a message every second communication round and as a result drive four actuators it's obvious that the reading and processing of the message must happen in the same communication round.

4.1.2 Requirements

FTCom provides the so-called Synchronisation Layer to the OS, enabling it to synchronise the start of the dispatcher table to a special point in time (phase) in dedicated communication rounds. In order to conceal the knowledge about the communication system from the OS, FTCom needs some information about the application (together with the information about the communication system):

- The dispatcher round
- The phase (offset)
- The application is synchronised to which communication rounds
- The length of a communication round

FTCom passes the synchronisation information to the application on demand by the global time. Two services are therefore specified (see chapter 6.6, Time Service for details):

- *ttGetGlobalTime* which returns the current global time
- *ttGetSyncTimes* which returns the current global time and the global time at the expected start of the last dispatcher table.

The following definitions are used (see Figure 4-1):

Dispatcher Table:	offline generated time table where the OSEKtime dispatcher invocation events are defined
Dispatcher Round:	length of the Dispatcher Table
Communication Round:	length of the periodic transmission pattern on the communication subsystem
Ground State:	no task except the idle task is running and no message transmission (external and internal) is in progress (RDA, filter or copy task)

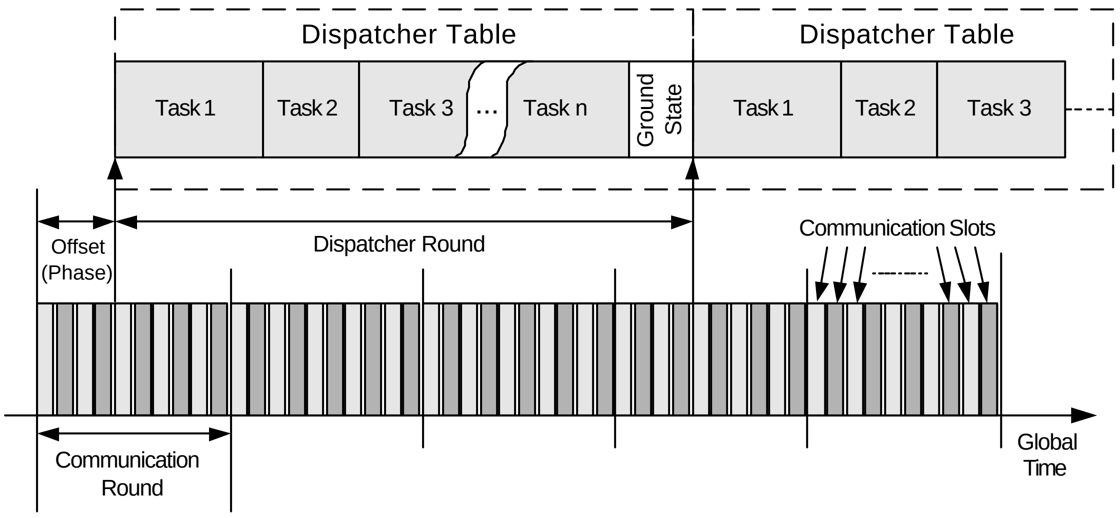

Figure 4-1: Dispatcher and communication rounds

4.2 External Clock Synchronisation

To facilitate the synchronisation of the globally synchronised clock to an external clock source, e.g., a GPS receiver, an external clock synchronisation service must be provided. This is not part of the standardised FTCom layer. The following describes the recommended practice for implementing an external clock synchronisation.

This service has two parts:

(1) Generate a correction value for the use by the communication system.

(2) Forward the correction value to the communication protocol.

4.2.1 Generation of External Correction Value

In a cluster with external clock synchronisation, there is always at least one node interfacing to an external time source. A node connected to such an external time source periodically sends out a time message containing a correction value for the complete cluster. All other nodes must receive this message and write the contents to a dedicated field in the communication controller.

The routine *ttExtClockSync* is used to generate the correction value in the nodes that have access to an external clock. It interfaces to the external periphery delivering a clock value, and executes the external clock synchronisation algorithm. The routine by default returns zero as a correction value. If specified by the user or by a FTCom layer tool, it returns the result of the user defined clock synchronisation algorithm. The routine must be invoked periodically, and is thus part of the time-triggered task schedule.

The external rate correction value must be sent to all other nodes in the cluster. Therefore, the routine generates a message. The message schedule on the communication media must accommodate for the time message: either an extra frame is sent, or the time message is contained in a frame together with other application data.

4.2.2 Write Correction Value to Communication Controller

The correction value contained in the last received time message must be written to the communication controller. A routine *ttSetExtSync* reads the time message and writes it the communication controller. This routine is periodically invoked and therefore part of the time-triggered task schedule.

4.3 Node Membership Service (optional)

A *membership service* is the consistent provision of information on the activity status of all communication partners. The FTCom layer optional provides a system call to find out the membership status of every node via its node id. If the underlying communication protocol comprises a membership service, this information should be used. Otherwise, the FTCom layer should ensure that the membership information on the nodes that is provided to the application is consistent (e.g., by implementing such a protocol in software, or by using other available information of the communication protocol).

4.4 Lifesign Update

To facilitate prompt error detection, a communication controller implementing a particular protocol may require the CPU to periodically update a defined register with a certain value (similar to a watchdog). This is called a *lifesign* mechanism. Details of if and how to update a lifesign and the frequency of the update operation depend on the actual communication protocol that is used.

The FTCom layer provides a system call to perform this regular lifesign update, which may be generated by an FTCom off-line design tool. The tool can also automatically schedule the system call, so that no user action is required for this service. To allow manual invocation as well, the system call is also included in the API description.

4.5 Start-up

The start-up of the distributed system is a function that depends on the used communication protocol and can only be defined with detailed knowledge of the communication protocol. A communication protocol specific API description needs to be defined.

5 Inter-task Communication

The OSEKtime FTCom layer provides services for the local communication of tasks located on the same ECU. These services should be used for all data exchanges between tasks. Message filtering and RDA are not required for local inter-task communication.

In a mixed OSEK and OSEKtime system three cases have to be distinguished:

1. Communication between an OSEK task and another OSEK task.
2. Communication between an OSEK task and an OSEKtime task.
3. Communication between an OSEKtime task and another OSEKtime task.

5.1 Communication between OSEK Tasks

This case is not part of the OSEKtime specification as the normal OSEK/VDX OS and COM communication mechanisms apply.

5.2 Communication between OSEK and OSEKtime Tasks

For communication between OSEK tasks on the one hand and OSEKtime tasks on the other hand the OSEKtime inter-task communication services are used. These services implement local message handling, i.e., the only way for communication with and between OSEKtime tasks is the use of messages.

The following API calls are used for the message handling:

- *ttSendMessage*
- *ttReceiveMessage*
- *ttInvalidateMessage*

Each call of *ttReceiveMessage* returns a new consistent copy of the message.

Each call of *ttSendMessage* updates the message.

Each call of *ttInvalidateMessage* invalidates the send status of the message.

The difference between local and global communication is transparent to the task.

5.3 Communication between OSEKtime Tasks

For communication among OSEKtime tasks the same mechanisms and service calls as for communication between OSEK/VDX tasks and OSEKtime tasks are used.

6 Specification of FTCom System Services

This chapter is structured according to the original OSEK specification. Sections 6.3 to 6.7 include a classification of OSEKtime FTCom system services.

Type of Calls

The system service interface is ISO/ANSI-C. The system service interface is ISO/ANSI-C. Its implementation is normally a function call, but may also be solved differently, as required by the implementation - for example by macros of the C pre-processor. A specific type of implementation cannot be assumed.

Structure of the Description

The FTCom system services are arranged in logical groups. A coherent description is provided for all services. The description of each logical group starts with data type definitions and a description of constants. A description of the group-specific system services follows.

Service Description

A service description contains the following fields:

Syntax:	Interface in C-like syntax.
Parameter (In):	List of all input parameters.
Parameter (Out):	List of all output parameters.
Description:	Explanation of the functionality of the operating system service.
Particularities:	Explanation of restrictions relating to the utilisation of the operating system service.
Status:	List of possible return values.
Standard:	List of return values provided in the operating system's stan-dard version.
Extended:	List of additional return values in the operating system's ex-tended version.

Most system services return a status to the user. No error hook is called if an error occurs. The return status is TT_E_FTCOM_OK if it was possible to execute the system service without any restrictions. If the system recognises an exceptional condition, which restricts execution of the system service, a different status is returned.

All return values of a system service are listed under the individual descriptions. The return status distinguishes between the ”standard” and ”extended” status. The ”standard” version fulfils the requirements of a debugged application system as described before. The "extended" version is considered to support testing of not yet fully debugged applications. It comprises extended error checking compared to the standard version.

The specification of services uses the following naming conventions for data types:

...Type:	describes the values of individual data (including pointers).
...RefType:	describes a pointer to the ...Type (for call by reference).

6.1 Common Data Types

ttStatusType

This data type is used for all status information the API services offer. Naming convention: all errors for API services start with E_. Those reserved for the OSEKtime operating system and for the OSEKtime Fault-Tolerant communication layer will begin with:

- TT_E_FTCOM_

The normal return value is TT_E_FTCOM_OK which is associated with the value of E_OK.

The following error values are defined:

All errors of API services:

- TT_E_FTCOM_ACCESS
- TT_E_FTCOM_ID
- TT_E_FTCOM_NOFUNC
- TT_E_FTCOM_VALUE
- TT_E_FTCOM_RDA_FAILED
- TT_E_FTCOM_MSG_NOT RECEIVED
- TT_E_FTCOM_MSG_INVALIDATED

The following sections contain a generic (protocol independent) description of the FTCom layer API.

6.2 Naming Conventions

6.2.1 General Naming Conventions

The following prefixes are used for all OSEKtime FTCom constructional elements, data types, constants, error codes and system services:

- “tt” prefix is used for constructional elements, data types and system services;
- “TT_E_FTCOM_” prefix is used for error codes;
- “TT” prefix is used for constants.

This is to ensure that no name clashes occur.

6.3 Message Handling

6.3.1 Data Types

ttStatusType

This data type is identical with StatusType out of the binding specification.

ttMsgIdType

This data type defines the data type for an identifier of a message.

ttAccessNameType

This data type defines the data type for references to the message body (data).

ttAccessNameRefType

This data type defines the reference to a variable of type ttAccessNameType.

6.3.2 Constants

TT_E_FTCOM_RDA_FAILED	constant of data type ttStatusType, RDA did not calculate a valid result
TT_E_FTCOM_MSG_NOT RECEIVED	constant of data type ttStatusType, no frame of the message has been received or the message has not been forwarded by the receiver's message filter during the last execution
TT_E_FTCOM_MSG_INVALIDATED	constant of data type ttStatusType, message was invalidated by sender or the message has not been forwarded by the sender's message filter during the last execution

6.3.3 ttSendMessage

Syntax:	ttStatusType	ttSendMessage (	
		ttMsgIdType <Message>, ttAccessNameRefType <Data>)	
Parameter (In):	Message	-	message identification
	Data	-	reference to message contents
Parameter (Out):	None		
Description:	ttSendMessage is called by the user out of a task body or an user ISR and copies the data <Data> of the message <Message> from the task local memory to a publicly accessible copy of the message (FT-CNI for non local messages). The message will always be marked as valid. ttSendMessage also reset the flag, which is associated with the given message.		
Particularities:	To be called by the user out of task body or from user ISRs.		

Status:	
Standard:	No error, TT_E_FTCOM_OK
Extended:	<Message> is invalid, TT_E_FTCOM_ID.
	<Data> is invalid or access denied, TT_E_FTCOM_ACCESS.

6.3.4 ttReceiveMessage

Syntax:	ttStatusType	ttReceiveMessage (ttMsgIdType <Message>, ttAccessNameRefType <Data>)
Parameter (In):	Message	- message identification
Parameter (Out):	Data	- reference to message contents

Description: ttReceiveMessage is called by the user out of a task body or an user ISR and copies the data <Data> of the message <Message> from a publicly accessible copy of the message (FT-CNI for non local messages) to the task local memory.

In case ttReceiveMessage return a status different from TT_E_FTCOM_OK, the contents of task local memory pointed to by <Data> are not modified.

Particularities: To be called by the user out of task body or from user ISRs.

Status:	
Standard:	No error, TT_E_FTCOM_OK
	TT_E_FTCOM_RDA_FAILED RDA did not calculate a valid result
	TT_E_FTCOM_MSG_NOT_RECEIVED no frame containing an instance of the message has been received or the value of the message has not been forwarded by the receiver’s message filter during the last execution
	TT_E_FTCOM_MSG_INVALIDATED message was invalidated by sender or the value of the message has not been forwarded by the sender’s message filter during the last execution
Extended:	<Message> is invalid, TT_E_FTCOM_ID.
	<Data> is invalid or access denied, TT_E_FTCOM_ACCESS.

6.3.5 ttInvalidateMessage

Syntax:	ttStatusType	ttInvalidateMessage (ttMsgIdType <Message>)
Parameter (In):	Message	- message identification
Parameter (Out):	none	
Description:	ttInvalidateMessage invalidates the message <Message> in the FT-CNI by setting the message status to invalidated message.	
Particularities:	To be called by the user out of task body or from user ISRs.	
Status:		
Standard:	No error, TT_E_FTCOM_OK	
Extended:	<Message> is invalid, TT_E_FTCOM_ID. The service is not specified for that <Message>, TT_E_FTCOM_NOFUNC. An instance of <Message> was the input of the function, e.g. <A''> instead of <A>, TT_E_FTCOM_ACCESS.	

6.3.6 Differences between OSEKtime and OSEK/VDX Message Management

This section lists the differences between the OSEKtime and the OSEK/VDX message management API, in order to avoid misinterpretations.

- The message copy attribute (WithCopy/WithoutCopy) is not a user-level configuration attribute because it is up to offline tools to optimise the message access scheme. Furthermore, the OSEK/VDX resource mechanism protecting messages without copy (GetMessageResource() / ReleaseMessageResource() services) is not applicable for OSEKtime.
- The E_COM_LOCKED error code is not supported because the message service call should be completed in any case in order to avoid a blocking problem. For example, the following construction is forbidden:
  ```
  while ( ttSendMessage (...) != TT_E_FTCOM_OK );
  ```
- Message data consistency should be guaranteed by the system. For example, a two message buffer/semaphore implementation concept may be used.
- Each message should have one sender and a number of receivers.

6.4 Membership Service

6.4.1 Data Types

ttNodeIdType

This data type defines the data type for an identifier of a node.

ttNodeMembershipType

This data type defines the data type for the node membership.

ttNodeMembershipRefType

This data type defines the reference to a variable of type ttNodeMembershipType.

6.4.2 Constants

TT_NODE_ACTIVE	constant of data type ttNodeMembershipType for active node
TT_NODE_INACTIVE	constant of data type ttNodeMembershipType for inactive node

6.4.3 ttGetNodeMembership

Syntax:	ttStatusType ttGetNodeMembership (ttNodeIdType <NodeID>, ttNodeMembershipRefType <NodeMembership>)		
Parameter (In):	NodeID	-	Identification of the node whose membership is queried.
	NodeMembership	-	Reference to the NodeMembership variable.
Parameter (Out):	none		
Description:	ttGetNodeMembership is called by the user out of a task body or an user ISR and returns the node membership information of the node <NodeID>.		
Particularities:	To be called by the user out of task body or from user ISRs. This service is optional.		
Status:			
Standard:	No error, TT_E_FTCOM_OK		
Extended:	none.		

6.5 Notification mechanism

6.5.1 Data Types

ttFlagIdType

This data type defines the data type for the identifier of a flag.

ttFlagStatusType

This data type defines the data type for the notification flag.

ttFlagStatusRefType

This data type defines the reference to a variable of type ttFlagStatusType.

6.5.2 Constants

TT_FLAG_SET	constant of data type ttFlagStatusType for set flags
TT_FLAG_CLEARED	constant of data type ttFlagStatusType for cleared flags

6.5.3 ttReadFlag

Syntax:	ttStatusType	ttReadFlag (ttFlagIdType <Flag>, ttFlagStatusRefType <Status>)
Parameter (In):	Flag	- identification of the flag <Flag>
Parameter (Out):	Status	- reference to the flag status variable
Description:	ttReadFlag returns the status of the flag <Flag>.	
Particularities:	To be called by the user out of task body or from user ISRs.	
Status:		
Standard:	No error, TT_E_FTCOM_OK.	
Extended:	<Flag> is invalid, TT_E_FTCOM_ID.	

6.6 Time Service

6.6.1 Data Types

ttTimeSourceIdType

This data type defines the data type for the identifier of a time source (e.g. global time base of a specific communication controller).

ttTickType

This data type defines the data type for the count value (count value in ticks).

ttTickRefType

This data type defines the reference to a variable of type ttTickType.

ttSyncStatusType

This data type defines the data type for the synchronisation status.

ttSyncStatusRefType

This data type defines the reference to a variable of type ttSyncStatusType.

6.6.2 Constants

TT_SYNCHRONOUS	network-wide synchronised time is available
TT_ASYNCHRONOUS	network-wide synchronised time is unavailable
TT_DEF_TIMESOURCE	default time source specified offline

6.6.3 ttGetGlobalTime

Syntax:	ttStatusType	ttGetGlobalTime (ttTimeSourceIdType <TimeSource>, ttTickRefType <GlobalTime>)
Parameter (In):	TimeSource	- time source identification (TT_DEF_TIMESOURCE for default time source)
Parameter (Out):	GlobalTime	- reference to current value of the network-wide synchronised time.
Description:	This service returns the current synchronised time of the dedicated time source <TimeSource> (see OSEKtime OS specification for more details on the clock synchronisation).	
Particularities:	To be called by the user out of task body or an ISR or by the OS.	
Status:		
Standard:	No error, TT_E_FTCOM_OK	
Extended:	TT_E_FTCOM_VALUE if GlobalTime is not available, TT_E_FTCOM_ID if <TimeSource> is invalid.	

6.6.4 ttGetComSyncStatus

Syntax:	ttStatusType	ttGetComSyncStatus (ttTimeSourceIdType <TimeSource>, ttSyncStatusRefType <SyncStatus>)
Parameter (In):	TimeSource	- time source identification (TT_DEF_TIMESOURCE for default time source)
Parameter (Out):	SyncStatus	- reference to the current synchronisation status.
Description:	This service indicates whether the global time of the dedicated time source <TimeSource> is available (TT_SYNCHRONOUS) or not (TT_ASYNCHRONOUS).	
Particularities:	To be called by the user out of task body or an ISR or by the OS.	
Status:		
Standard:	No error, TT_E_FTCOM_OK	
Extended:	TT_E_FTCOM_ID if <TimeSource> is invalid.	

6.6.5 ttGetSyncTimes

Syntax:	ttStatusType	ttGetSyncTimes (ttTimeSourceIdType <TimeSource>, ttTickRefType <GlobalTime>, ttTickRefType <ScheduleTime>)
Parameter (In):	TimeSource	- time source identification (TT_DEF_TIMESOURCE for default time source)
Parameter (Out):	GlobalTime	- reference to current value of the network-wide synchronised time.
	ScheduleTime	- reference to value of the global time at the start of the last dispatching table.
Description:	This service returns the current time of the dedicated time source <TimeSource> (see OSEKtime OS specification for more details on the clock synchronisation) and the time at which the start of the last dispatching table was scheduled.	
Particularities:	To be called by the OS.	
Status:		
Standard:	No error, TT_E_FTCOM_OK	
Extended:	TT_E_FTCOM_VALUE if GlobalTime is not available, TT_E_FTCOM_ID if <TimeSource> is invalid.	

6.7 External Clock Synchronisation

The following two tasks are a minimum set of functions to implement external clock synchronisation. Actual implementations might include extensions to this description, depending on specifics of the external clocks used.

6.7.1 ttExtClockSync

Syntax:	ttStatusType	ttExtClockSync (ttTimeSourceIdType <TimeSource>)
Parameter (In):	TimeSource	- time source identification
Parameter (Out):	none	
Description:	ttExtClockSync interfaces to an external clock hardware <TimeSource> and performs the external clock synchronisation algorithm according to the value read from this clock and the global time in the cluster. It generates the time message containing the correction value.	
Particularities:	To be called from a periodic task of the FTCom schedule.	
Status:		
Standard:	No error, TT_E_FTCOM_OK	
Extended:	TT_E_FTCOM_ACCESS when called from a user taskTT_NO_FUNC if no external clock hardware is available, TT_E_FTCOM_ID if <TimeSource> is invalid.	

6.7.2 ttSetExtSync

Syntax:	ttStatusType	ttSetExtSync (ttTimeSourceIdType <TimeSource>)
Parameter (In):	TimeSource	- time source identification
Parameter (Out):	none	
Description:	ttSetExtSync reads a time message out of the CNI and writes a correction value to an appropriate CNI External Rate Correction Field.	
Particularities:	To be called from a periodic task of the FTCom schedule.	
Status:		
Standard:	No error, TT_E_FTCOM_OK	
Extended:	TT_E_FTCOM_ACCESS when called from a user task, TT_E_FTCOM_ID when <TimeSource> is invalid	

7 Hints

Following topics are not part of the specification but are recommendations.

7.1 Optional Properties of the FTCom and the underlying Communication Controller

Optional properties of the FTCom and the underlying communication controller are:

- ***Atomic Frame Transmission*** should be guaranteed by the communication protocol. The FTCom layer should provide atomic message transmission.
- The communication system may support ***external clock synchronisation*** by periodically transmitting time messages from a node connected to an external time source to all other nodes. The time messages must contain at least a correction value to adjust the system time to the external time source.
- The communication system may support ***redundancy***. This may range from the weakest form of redundancy, time redundancy over a single channel, to multiple transmission channels. For redundant channels replica determinism must be supported, i.e., messages sent over two channels must arrive in a deterministic order.

8 Index

8.1 List of Services, Data Types and Constants

8.2 List of Figures

8.3 List of Tables

9 History

Version	Date	Remarks	
1.0	July 24th 2001	Authors:	
		Anton Schedl	BMW
		Elmar Dilger	Bosch
		Thomas Führer	Bosch
		Bernd Hedenetz	DaimlerChrysler
		Jens Ruh	DaimlerChrysler
		Matthias Kühlewein	DaimlerChrysler
		Emmerich Fuchs	DeComSys
		Thomas M. Galla	DeComSys
		Yaroslav Domaratsky	Motorola
		Andreas Krüger	Motorola, since 04/01 Audi
		Patrick Pelcat	PSA Peugeot Citroën
		Michel Taï-Leung	Renault
		Martin Glück	TTTech
		Stefan Poledna	TTTech
		Thomas Ringler	University of Stuttgart
		Brian Nash	Wind River
		Tim Curtis	Wind River