Specification of the TTP/A-Protocol V2.00

Author:

Hermann Kopetz et al.
Real-Time Systems Group
University of Technology Vienna

September 2002

Table of Contents

Introduction *1*

This document describes the specification of the TTP/A fieldbus protocol. TTP/A stands for Time-Triggered Protocol for SAE Class A Applications (TTP/A). The protocol specification is part of the standard “Smart Transducers Interface Specification” [*OMG02*] issued by the Object Management Group.

The OMG document specifies a smart transducers interface and further describes a CORBA (Common Object Request Broker Architecture) interface that specifies a transparent client/server interaction with a fieldbus from a superordinated network. In contrast, this specification will describe the specification of the TTP/A protocol, which conforms to the smart transducers interface at fieldbus level. This document will be in full compliance with the “Smart Transducers Interface Specification” which is recommended for further reading.

1.1 Motivation and Objectives

A smart transducer (ST) may comprise a hardware or software device consisting of a small, compact unit containing a sensor or actuator element (possibly both), a microcontroller, a communication controller and the associated software for signal conditioning, calibration, diagnostics, and communication. The ST provides the intended services across interfaces to its clients. These interfaces are well specified in the value domain and in the temporal domain and only make those ST properties visible to the client that are required for the proper use of the ST. If the STs are in agreement with this specification, these interfaces have the same form and behavior for the wide array of differing sensor and actuator nodes in the various engineering disciplines. The internal structure and operation of these differing STs remain encapsulated within the ST and are not exposed at the interfaces that are accessible from the client. A user of an ST, which conforms to this standard, will thus have to cope only with one single generic ST interface for the multitude of existing and new sensor types. Many ST systems are designed for mass-market applications, where lowest manufacturing costs are absolutely essential. Therefore this specification has been designed to minimize the resource

requirements in the STs and thus supports very cost-effective implementations. A minimal ST fits into an 8-bit wide processor with on-chip oscillator and a minimum of less than 4 kByte of ROM and 64 bytes of RAM storage.

1.2 Structure of this Document

This chapter is organized as follows: Section 2.2, “Conceptual Model,” on page 2-2 presents the conceptual model that is the base of this specification. Section 2.3, “File Access Protocols,” on page 2-8 explains the design of the interface file system (IFS) and the file access protocols that are at the core of this specification. Section 2.4, “Smart Transducer Filesystem in the ST,” on page 2-14 is devoted to the IFS in the STs, while Section 2.5, “The Fireworks,” on page 2-19 describes the required framework. The interfaces are described in Section 2.6, “Description of CORBA Based Object Model and Interfaces,” on page 2-20. Special system services are treated in Section 2.7, “Special Services,” on page 2-21. The UART transport protocol is specified in Section 2.8, “UART Transport Protocol,” on page 2-23.

Smart Transducers Interface 2

2.1 Overview and Rationale

A smart transducer (ST) may comprise a hardware or software device consisting of a small, compact unit containing a sensor or actuator element (possibly both), a micro-controller, a communication controller and the associated software for signal conditioning, calibration, diagnostics, and communication. The ST provides the intended services across interfaces to its clients. These interfaces are well specified in the value domain and in the temporal domain and only make those ST properties visible to the client that are required for the proper use of the ST. If the STs are in agreement with this standard proposal, these interfaces have the same form and behavior for the wide array of differing sensor and actuator nodes in the various engineering disciplines. The internal structure and operation of these differing STs remain encapsulated within the ST and are not exposed at the interfaces that are accessible from the client. A user of an ST, which conforms to this standard, will thus have to cope only with one single generic ST interface for the multitude of existing and new sensor types.

Many ST systems are designed for mass-market applications, where lowest manufacturing costs are absolutely essential. Therefore this standard has been designed to minimize the resource requirements in the STs and thus supports very cost-effective implementations fulfilling the mandatory requirements only. A minimal ST fits into an 8-bit processor with an on-chip oscillator and a minimum of less than 4 kByte of ROM and 64 bytes of RAM storage.

Understandability and flexibility have been the driving forces behind this specification. The ST interface specification contained in this document provides a flexible capability to CORBA to access the *real-time service (RS)* interface, the *diagnostic and management (DM)* interface, and the *configuration and planning (CP)* interface of small STs in a distributed control system. By standardizing many different interfaces of STs, this specification contributes to a simplification of I/O programming and thus to software cost reduction of distributed control systems.

A distributed control system must support predictable performance in the temporal domain. Since many of the standard communication protocols, such as General Inter-ORB Protocol (GIOP), have not been designed for temporal predictability, this specification proposes a new time-triggered transport service within the distributed ST subsystem and an encapsulated gateway of this subsystem to the CORBA environment.

This chapter is organized as follows: Section 2.2 presents the conceptual model that is the base of this specification. Section 2.3 explains the design of the interface file system (IFS) and the file access protocols that are at the core of this specification. Section 2.4, “Smart Transducer Filesystem in the ST,” on page 2-14 is devoted to the IFS in the STs, while Section 2.5, “The Fireworks,” on page 2-19 describes the required framework. The CORBA interface is described in Section 2.6, “Description of CORBA Based Object Model and Interfaces,” on page 2-20. Special system services are treated in Section 2.7, “Special Services,” on page 2-21. The UART transport protocol is specified in Section 2.8, “UART Transport Protocol,” on page 2-23.

2.2 Conceptual Model

The following sections give a detailed description of the structure and concepts as they pertain to a smart transducer cluster.

2.2.1 Structure of a Smart Transducer System

A smart transducer (ST) system that can be accessed from a single CORBA gateway interface consists of up to 250 *clusters*. The *master* (an ST with extended features) of each cluster is connected to the CORBA *gateway* through a real-time communication network, which provides a synchronized time to each *master*. Each cluster can contain up to 250 STs that communicate via a cluster-wide broadcast communication channel. One *active* master controls the communication within one ST cluster (in the following sections the term *master* refers to the *active master* unless stated otherwise). Since the other STs are controlled by the *master*, we call them slave nodes also. Figure 2-1 depicts an ST system consisting of three clusters with one master each, and 8 slave nodes each.

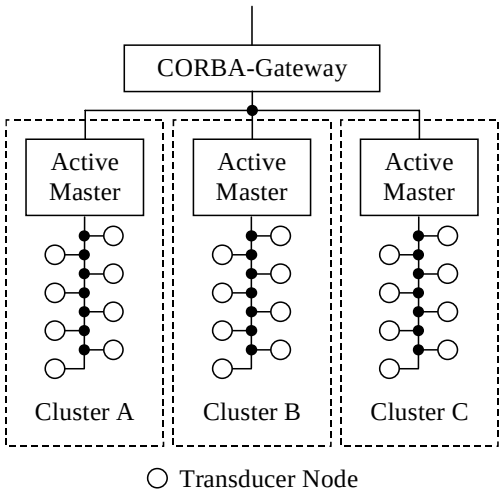

Figure 2-1 Transducer System with 3 clusters

During operation, every ST must have a cluster-unique *logical name*. Additionally, a *series number* that identifies the type of the transducer must be stored in each transducer. In most cases, an ST will also contain a *serial number* that is unique for each transducer type. If it contains a *serial number*, the concatenation *series number* and *serial number* determines the unique *physical name* of an ST that identifies an ST uniquely in the universe of STs. This *physical name* is used when assigning a logical name to an ST (this is called the *baptizing* of the ST and can be performed on line). If the plug and play capability is used, every ST in a cluster must have a unique *physical name*. In case there exists more than one ST with the same *physical name* in a cluster the baptize algorithm, which assigns an ST a logical name cannot be successful. (In such cases the *logical name* must be assigned out of system).

Every ST cluster has a *master* that controls the communication among the STs of a cluster. The interconnection between an ST system and the CORBA world is accomplished by one or more *gateway* nodes supporting three encapsulated CORBA interfaces: the *real-time service* (RS) interface, the *diagnostic and management* (DM) interface, and the *configuration and planning* (CP) interface. It is assumed that every ST contains a physical clock for measuring time. If required, the state of clocks in the STs can be related to an external time standard, such as GPS time.

2.2.2 The Interface File System

The information transfer between an ST and its client is achieved by sharing information that is contained in an encapsulated ST *internal interface file system* (IFS), as depicted in Figure 2-2. This IFS is at the core of the conceptual model, which is thus a data centric model.

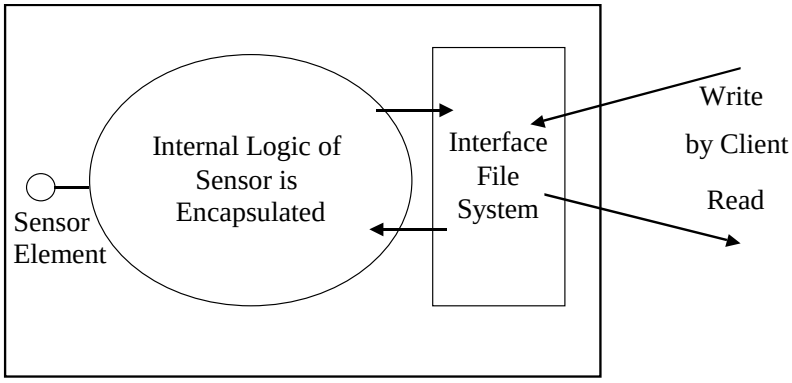

Figure 2-2 Interface File System in a Smart Transducer

An IFS file is an indexed sequential file with up to 256 records. A record has a fixed length of four bytes (32 bits). An IFS record is the smallest addressable unit within an ST system. Every record of every IFS file has a unique hierarchical address (which also serves as the *global name* of the record) consisting of the concatenation of the *cluster* name, the *logical name*, the *file name* and the *record name*. Since each name has a length of one single byte, the name of a record is thus also four bytes long and fits itself into a single record. There are three operations defined on a record: *read*, *write*, and *execute*. These operations are described in detail in Section 2.3, “File Access Protocols,” on page 2-8.

In very small STs, the IFS can degenerate to a few records of a few files. Such an ST will only support a limited functionality for a particular mass-market application. In order to become a viable standard for these mass-market applications, this specification suggests a set of services, that starts with a very limited minimum service level. This minimum service level must be provided by any conforming implementation. If more services than these minimum services are provided, this specification defines services that can be combined like building blocks in order to design an appropriate ST. If a building block is implemented, the ST must provide the full set of services of this building block. The specification of further building blocks will be the object of future standards.

2.2.3 Observations

Any property of a relevant *state variable* that is observed by an ST; for example, the temperature of a vessel, is called a *state attribute* and the corresponding information *state information*. An *observation* records the state of a state variable at a particular instant, the *point of observation*. An *observation* can be expressed by the atomic triple:

<Name of the observed state variable, observed value, time of observation>

Example: The following would be an observation: *"The temperature of vessel A was 75 degrees Celsius at 10:42 a.m."* This concept of an observation is the essential element for understanding the design of this specification.

An *observation* is an example of a *state information* data item. State information is *idempotent* and requires an *at-least-once* semantics when transmitted to a client. At the receiver, state information requires an *update-in-place* and a *non-consumable* read.

A sudden change of state that occurs at an instant is called an *event*. Information that describes an *event* is called *event information*. *Event information* is not *idempotent* and requires *exactly-once* semantics when transmitted to a consumer. At the receiver, *event information* must be *queued* and *consumed* on reading.

An observation is stored in a record of the IFS within an ST and is normally periodically updated by internal encapsulated processes of the ST. The *hierarchical address (global name)* of the selected record denotes the name of the *observed state* variable. The *observed value* is contained in the record and the *time of observation* is the time of updating the record by the internal process of the ST. If the value of an observation is longer than four bytes, then such an observation will be stored in multiple records of an IFS file.

In the ST model, the name of the *observed state* variable, the *global name*, serves a second purpose: it identifies the meta-data about the given ST (at a defined internet address outside the ST system) to explain the meaning of the data in the given ST implementation. Since STs are very resource constrained, the meta-data for the development is cleanly separated from the run-time system and kept in a comfortable development system. The *series number* (part of the *physical name*) that must be stored in every ST establishes the link between an ST type and its description.

At the encapsulated CORBA interface a complete *observation*; that is, the *name of the observed state variable* (4 bytes), the *time of update* (8 bytes) the *value* (4 bytes) and an *attribute field* (4 bytes) is presented in the CORBA interface in at least five consecutive four-byte records.

2.2.4 Distinction between Time Triggered and Event Triggered systems

For the reader, who is not familiar with the terms *time-triggered* and *event-triggered*, we include the following short explanation. A more detailed discussion can be found in the text [Kop97].

A *trigger* is an event that causes the start of some action; for example, the execution of a task or the transmission of a message. Depending on the triggering mechanism for the start of communication and processing activities in each node of a computer system, two distinctly different approaches to the design of real-time computer applications can be identified: the *event-triggered* (ET) and the *time-triggered* (TT) approach.

In the ET approach, all communication and processing activities are initiated whenever a significant change of state; that is, an event other than the regular event of a clock tick, is noted.

In the TT approach, all communication and processing activities are initiated at predetermined instants by the progression of time. While ET systems are flexible, TT systems are temporally predictable.

2.2.5 Interface Types

In the ST model we distinguish between three interface types of an ST, the *real-time service* (RS) interface, the *diagnostic and management* (DM) interface and the *configuration and planning* (CP) interface. All information that is exchanged across these interfaces is stored in files of the IFS. While the *real-time service* interface is time sensitive, the other two interfaces are not time sensitive.

Real-time Service Interface: The real-time service (RS) interface provides time sensitive information to its client. This information is normally used for control purposes (for example, periodic execution of a control loop), where the quality of control is degraded by jitter. Time critical information is therefore delivered periodically at the master with small known delay and minimal jitter. The temporally predictable real-time service interface is time-triggered. This implies that the jitter is determined by the precision of the clock synchronization, which is, even in the lowest cost implementations, below 100 µsec. In implementations supporting a higher bandwidth this precision can be improved to less than 1 µsec. To minimize the delay, the instant of update of the IFS file record that contains the real-time information can be synchronized *a-priori* with the instant of transmission-start of this information. In this case, the delay will be reduced to the duration of the interval required for the actual transmission.

Diagnostic and Management Interface: The diagnostic and management interface is used to monitor the ST, to parameterize the node, and to access the diagnostic information inside the ST.

Configuration and Planning Interface: The configuration and planning interface is used to configure a generic ST for a new application. This includes assigning a logical name to the ST and the assignment of the transmission slots in the time-triggered schedule for the real-time service (RS) interface.

2.2.6 The Transport Protocol

The ST system-internal transport protocol supports the *time-triggered* transport of data *frames* from one ST to all other STs of a cluster (broadcast transport service within a cluster). A *frame* consists of one or more bytes sent by an ST. Since the *instant* when a *frame* is sent is controlled — either directly or indirectly — by the *master*, it is assured that only one sender will access the communication channel at a particular instant. In case the communication is not successful, there is no automatic retransmission. The communication system is thus predictable with a known latency and minimal jitter. Different transport protocols, such as CAN or LIN, or the wireless IEEE 802.11, can be integrated within this standard. For low-cost STs, a single wire UART transport protocol that uses an ISO standardized physical layer is specified in Section 2.8, “UART Transport Protocol,” on page 2-23.

2.2.7 Metadata about a Smart Transducer

The structure and the meaning of the data items in the IFS files are only intelligible if some metadata about the particular IFS is known. Since an ST has only a very limited storage capacity, this metadata describing the semantics of the ST files resides outside the ST at a web site associated with each ST type. This metadata can be accessed via a *register service*. The metadata information is essential for the development of applications by a "human design process" or by an “automated design process.” In the beginning, this metadata will be described by an ad-hoc combination of "structured English" and XML metadata tags. If this specification is successful, the standardization of these metadata files by the OMG is an urgent topic in order to enable the development of effective design support tools.

The register service for the smart transducer systems has the following functions:

- Establishment of a link to the ST metadata. The *series number* (part of the *physical name*) in each ST, which indirectly defines the structure and contents of the IFS in an ST type, can be used to establish a link to a file at the ST vendor, which contains a metadata description of this ST.
- Namespace management of the physical names of STs. To avoid duplication of ST's *physical names*, each vendor is assigned a unique *series number* for each ST type and a defined partition of the namespace to assign unique *serial numbers* to each physical ST.
- Maintain ST yellow pages. The register service maintains a database of STs that are available on the open market. By querying this database, a novice user can find out which available ST meets his/her requirement and get a pointer to the web site of the supplier.

This ST specification provides mechanisms for the "plug and play" capability of ST systems (see Section 2.7.1, “Node Identification—Plug and Play,” on page 2-21 and Section 2.7.2, “Baptizing of Nodes,” on page 2-22). The *master* of a cluster can periodically query whether a new node has been connected to an existing cluster. Then the master can identify the *physical name* of this new node by executing a binary search algorithm. This search is performed simultaneously to the real-time operation of the other nodes of the ST cluster. As soon as the physical name of the new node has been identified, the master can access, via the register service, the metadata of the newly connected ST and can initiate a design process that integrates the new node into the running ST system.

2.2.8 Fault-tolerant Sensor Systems

Fault-tolerant ST systems can be constructed by the replication of ST and their clusters, and the connection of these replicated clusters to replicated *masters* that form fault-tolerant units. Since real-time applications often have FT mechanisms that are based on active replication, no distortion of the temporal properties of the service takes place in case of a failure of a unit. Figure 2-3 outlines an example of a fault-tolerant ST configuration.

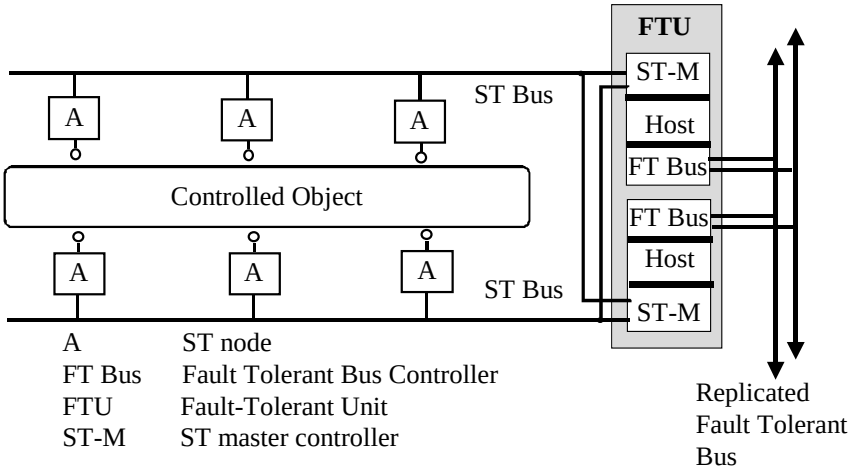

Figure 2-3 Fault-Tolerant Sensor Subsystem

A controlled object is observed by a plurality of replicated sensors that are connected to two distinct ST clusters. Replicated *masters* form a fault-tolerant unit with two access points controlling these two clusters. The upper *master* in Figure 2-3 controls the upper cluster. In the event that the upper *master* were to fail, the *lower* master, which would normally act as a standby *master* for the upper cluster would take control of both clusters. The same would apply for the lower *master* with respect to the *lower* cluster. Such a configuration will tolerate any single failure in any one of its constituent components without suffering any degradation of service.

2.3 File Access Protocols

2.3.1 File Structure and Naming

The Interface File System (IFS) is a hierarchical distributed file system that comprises a set of up to 64 index-sequential files in each node of the ST system. The structure of the IFS corresponds to the structure of the ST system, as outlined in Section 2.2, “Conceptual Model,” on page 2-2. An external client can access a record within an ST system, by the following structured address:

<cluster name, node name, file name, record name>

Since the ST system is optimized for eight bit node architectures, each name has a length of one byte. The maximum size of a distributed IFS is thus 2^{22} files, with 256 records of four bytes each. If the situation implies a restricted context of an address, then the address can be smaller. For example, inside a cluster the cluster name can be omitted and within a node, a record can be identified by two fields, the *file name* and the *record name*.

Some values for the cluster name and the logical node name are reserved for a special purpose: 0x00 is reserved for broadcast messages, 251-252 is reserved for future extensions, 253 is the gateway, 254 is the master, and 255 is reserved for integrating new nodes.

2.3.2 File Operations

The *master* of a cluster initiates a file operation by transmitting a special one-byte *frame*, called the *firework*. The *firework* informs all nodes that a new operation is starting and identifies the file-operation.

The file system supports three file operations: *read*, *write*, and *execute* a file record. Every file operation must be followed by the *global name* of the record. The *read* and *write* operations are executed atomically to *read* or *write* the named record.

When performing an *execute* operation, the *name* of the file record serves two purposes:

1. The concatenation of the *file-name* field (1 byte) and the *record-name* field (1 byte) denote the *type of operation* that is to be performed.
2. The *global record name* points to the *parameters* of this operation, which are contained in the *named* record.

This encoding technique improves the efficiency of operations in low-cost small bandwidth systems.

Example: If a temperature-sensor should start a new conversion executing a specific record may perform this. As soon as the conversion completes the result will be stored in this record.

Since there are only three file operations, the file operations code can be encoded in two bits as shown in Table 2-1 on page 2-9.

Op Code	Meaning in MP Round	Meaning in MS Round
00b	write from bus to IFS	write from bus to slave's IFS
01b	read to bus from IFS	read to bus from slave's IFS
10b	write to IFS and sync	forbidden
11b	execute	execute

Table 2-1 Description of OP-Codes

Since an ST can hold up to 64 files, the file name (6 upper bits) and the file operation (2 lower bits) can be fitted into a single byte.

In the ST system we distinguish between two kinds of file accesses, called a *master-slave* (MS) round, a *multi-partner* (MP) round. The MS rounds are used to implement the *diagnostic and management* (DM) interface and the *configuration and planning* (CP) interface. The periodic *multi-partner* (MP) rounds are used to implement the *real-time service* (RS) interface.

For operations that must be executed simultaneously by all nodes of a cluster it is possible to use a MS round with a logical name of 0x00 in order to perform a *broadcast* round.

2.3.3 Master-Slave (MS) Round

The master-slave (MS) round is used by the master of a cluster to read data from an IFS file record, to write data to an IFS file record, or to execute a selected IFS file record within the cluster.

An MS round consists of two phases, an address phase (MSA) and a data phase (MSD). During the address phase the *master* specifies (in a message to the slave node) which type of file operation is intended (*read*, *write*, or *execute*) and the address of the selected file record. The message in the address phase consists of the following six bytes:

<firework><epoch><logical name><file name | operation><record name><check byte>

Instead of the cluster name (which is required in the global IFS record address but not at the cluster level), an epoch counter that contains an identification of the current epoch of the cluster internal time base and is incremented each round is provided. In the subsequent MSD round the *master* sends a firework, which indicates that it is either transmitting the record data (if a file *write* operation is performed) or is waiting for the slave to transmit the requested record data (if a file *read* or *execute* operation is performed). The message in the data phase consists of six bytes:

<firework><data byte 0><data byte 1><data byte 2><data byte 3><check byte>

As mentioned before, the implementation must guarantee that the *record read* and *record write* operations are atomic at the record level. If atomicity is required beyond the record level, a concurrency control protocol must be implemented at the application level by designating one record as a concurrency control record. In order to avoid any delay of the writer, a non-blocking concurrency control protocol should be implemented.

The check byte in the MSA-Round and MSD-Round is calculated as a result of an exclusive-or operation of the preceding bytes (including the firework). The check byte is also used for transmitting inline error codes. In case of an error, all four data bytes of the MSD-Round are set to 0xFF and the check byte contains an error code in the lower nibble while the bits of the higher nibble are all set. Note that a message containing the value 0xFFFFFFFF differs significantly from an error message in the check byte.

Two optional *membership vectors* (bit fields) are defined (see Section 2.4.3). Every time the master receives from an ST a correct response within an MS round it sets the corresponding bit of the second membership vector. If none or a wrong answer is received, the respective bit is cleared.

If multiple MSA rounds are received the last one is chosen. If multiple MSD rounds are received the first one is chosen and the remaining are ignored. Thus the system provides additional resistance against unintended operations due to missed MSA or MSD rounds.

2.3.4 Multi-Partner (MP) Round

A *multi-partner* (MP) round is used to implement the *real-time service* (RS) with constant delay and minimal jitter. It is possible to define up to six different MP rounds at the same time; for example, to perform fast switches between different modes. MP rounds are periodic and optimized for high data efficiency. An MP round consists of a firework and subsequent data frames. A data frame is a sequence of one or more bytes originating from one ST *(master or slave)*. The sequence of *frames* of an MP round, depicted in Figure 2-4 is described in a *round-descriptor list* (RODL). The RODL is stored in a file of the IFS.

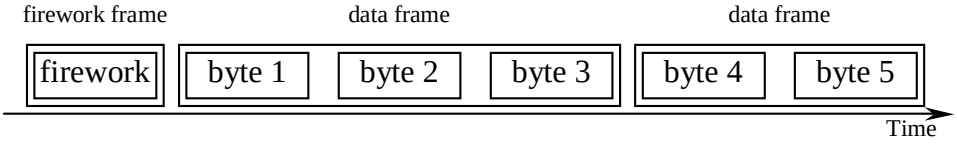

Figure 2-4 Structure of MP rounds

An MP round starts with a *firework*, sent by the *master*, followed by a sequence of data *frames*, either from the *master* or one of the *slaves*. The firework contains the name of the RODL file that must be executed in this round. The RODL file that is stored at the *slave* contains the following information:

1. Which byte numbers after the firework are assigned to this slave.
2. The type of operation (*read*, or *read-synchronize*, *write*, or *execute*) is requested.
3. Where to get or put these data bytes in the IFS.

The firework, which initiates a round, contains the name of the RODL to be executed. It is evident that the schedule described in the RODL referred to by the firework must be different in each slave. A global RODL can be considered as a distributed file system consisting of all ST-local RODLs.

Since Slot 0 is reserved for the firework and the last slot of a round is reserved for the Inter Round Gap (IRG), an MP round may be used to communicate up to 62 data bytes because a valid MP round is limited to 64 slots in total.

The *master* contains a special file, the ROund-SEquence (ROSE) file. A ROSE file contains the specification of the *instants* for a sequence of consecutively executable rounds and a *sequence period*, which determines after which duration this sequence, must be repeated.

Two optional *membership vectors* (bit fields) are defined (see Section 2.4.3). Every time the master receives from an ST a correct response within an MP round it sets the corresponding bit of the first membership vector. If none or a wrong answer is received, the respective bit is cleared.

The correct response of an ST in an MP round is considered a *life-sign* of the node. Based on this *life-sign* information, the *first membership vector* of active STs is maintained at the *master.* The *error detection latency* of the first membership vector is less than two *sequence periods.* A *second membership vector* is used to hold *life-sign* information of STs that are not members of the MP rounds currently issued by the ROSE file. This *second membership vector* is therefore updated via MS rounds. The *error detection latency* of this *second membership vector* is application specific.

2.3.5 Broadcast Round

A broadcast round has the same firework and the same layout as a master-slave round, but its address field always contains the *logical name* 0. The *logical name* 0 is a reserved *logical name* and addresses all baptized nodes (the *logical name* is not equal to 0xFF) in the cluster. The broadcast round consists also of two parts, but in contrast to a usual MS round, the slaves must not send an answer of an *execute* or *read* command, thus a read command is not feasible. An example for the use of a broadcast round is a *"sleep command"* that puts all nodes of a cluster into the *"sleep state"*.

2.3.6 Interleaving of Rounds

MP rounds and MS rounds are executed periodically. If there is no request from an external client for an MS round pending, the interval between the two MP rounds will be used to update the *second membership vector.* Interleaving diagnostic traffic with real-time traffic without disturbing the temporal characteristics of the real-time traffic as depicted in Figure 2-5 is recommended.

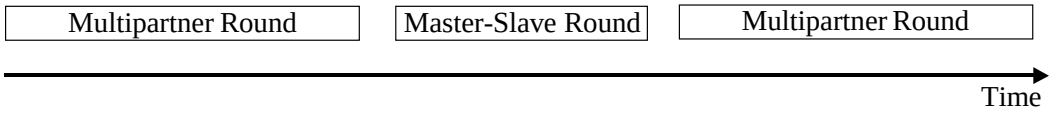

Figure 2-5 Interleaving of MP and MS rounds

It is possible to build a new MP round dynamically while the present MP round performs the *real-time service*. The master writes new configuration data dynamically into a new RODL by using MS rounds, and then, after the RODLs in all slave nodes have been updated, switches to the newly created RODL in order to execute the new MP round.

2.3.7 Data security

This specification includes a number of error detection mechanisms that help to detect the possible corruption of IFS file data in storage and during transport:

1. On transport, every byte contains a parity bit for error detection.
2. Every frame of an MS round is protected by a check byte (eight-bit checksum).
3. When designing MP rounds, protected data types can be used if required by the application scenario.
4. The data patterns in the firework byte have been carefully designed to provide a Hamming distance of 4.
5. An ST node can express the confidence in its sensor reading by assigning a value to a confidence marker. This confidence marker is an important input (and output) of sensor fusion algorithms.

2.3.8 Global Time

In a distributed transducer subsystem, a global notion of time must be available in every node of the system in order to coordinate the actions of the nodes in the temporal domain. Since the different nodes can have widely differing hardware characteristics, the precision, the granularities, and the horizon of the time representations in the differing nodes may vary in order to optimally match the time representation to the limited hardware capabilities of the nodes. In this specification we therefore distinguish between an external time representation at the CORBA interface and an internal time representation within a particular cluster. The *master* acts as a timeserver, transforms the external time representation to the internal time representation and vice versa, and provides a reference time for all nodes of a cluster. The *master*—or the CORBA *gateway*—can also implement an external clock synchronization; for example, with a GPS time receiver that provides a global accuracy in the sub microsecond range.

As an external time representation we specify a uniform eight-byte (64 bit) long time format based on GPS time, which allows to mark uniquely every instant within the time horizon of interest. It has a granularity of 2^{-24} seconds; that is, about 60 nano-seconds. This granularity has been chosen because it is possible to synchronize a site with a time signal from a GPS receiver within this accuracy. The duration between two external clock ticks must be an integer fraction of the physical second in order to facilitate the synchronization of the external clock with the GPS clock at the full-second instants. The external time representation has a horizon of 2^{40} seconds; that is, more than 10 000 years and thus will not wrap around in the foreseeable future. The epoch starts with the epoch of the GPS time; that is, January 6, 1980. Thus instants before January 6, 1980 are expressed as negative values. To express time and date in the conventional form, a Gregorian calendar function with the input (and output) of the long time representation must be implemented.

The smart transducer system can also be used with free running clocks; that is, without a GPS reference. In such a system we use the same external time representation as above, but initialize the time with 0 at startup of the CORBA *gateway.* In such a system it is still possible to measure durations, but a relationship of the transducer internal instants to an external time-reference cannot be established.

The CORBA *gateway* is connected to the *master* of each cluster through a real-time communication network, which can transport messages with constant delay and minimal jitter. It is thus possible to synchronize the clocks of the masters with the CORBA *gateway* clock.

In order to economize on the representation of the continuously flowing time in the *slaves*, only an interval of time around the current time “now”, can be expressed in the slaves in the internal time representation. This is in agreement with the strategy to reduce the memory requirement of a *slave* as far as possible. The epoch of the time scale at a slave (internal time representation) begins with the instant of the start of a firework. Every time a new round (MP, MSA, or MSD round) is started the 8-bit epoch counter is incremented by one. Thus each *slave* can distinguish between 256 consecutive epochs. In order to allow a slave to (re)integrate to the system the master transmits its 8-bit epoch-counter with each MSA round. This 8-bit epoch counter replaces the cluster name in the MS address round which is not needed any more at the addressed master.

The translation of the slave internal time representations of the transducer subsystem to the external time representation is in the responsibility of the *master* node. During the transmission of the data *frames* within a round, the *slaves* are periodically resynchronized with the reception of data bytes from a node with a trusted time base. The "trusted-time-base" slots of a multi-partner round are marked as "read synchronize" slots in the corresponding RODL. The integration and periodic resynchronization of local clocks of *slaves* with a maximum frequency deviation within 50% of the nominal frequency and a drift rate of up to 10^{-1} sec/sec is thus supported.

2.4 Smart Transducer Filesystem in the ST

The Interface File System (IFS) provides the common encapsulated addressing space for the exchange of information within a set of ST clusters and between a set of ST clusters and the CORBA *gateway.* The IFS of a single ST comprises 64 files of up to 256 four-byte records each. Except for a minimal documentation file, an ST must implement only those files that are required for its purpose.

The first record (rec. no. 0x00) of each file is the Header record, which contains file specific information.

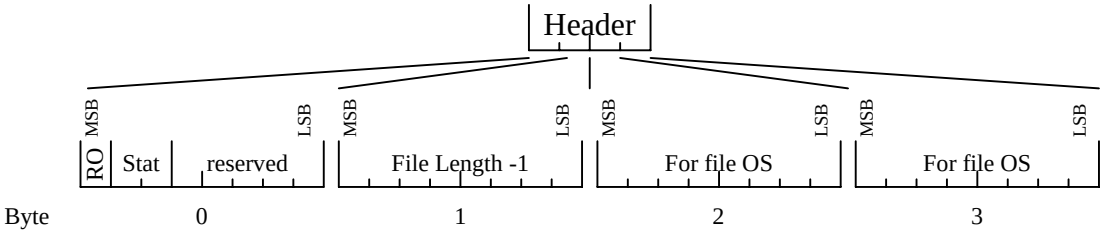

Figure 2-6 Addressing Space

RO: read-only bit. If set file is read-only.

Stat: 01_b file ok.

otherwise filedamaged

If a file is not implemented there is no need to implement the respective header record; the "NoFile"-error is returned instead. Unimplemented records in the middle of a file should return 0x00 in order to prevent holes in the IFS (the "NoRecord"-error is applicable for records beyond the length of a file only).

Two initial states are very likely in a memory element: all bits set or all bits cleared. Since a correct file header must contain at least one set bit and one cleared bit (in the stat field) such an initial state is recognized as a damaged file.

The namespace for files is subdivided into two parts:

System Files (file no. 0x00-0x0F and file no. 0x38-0x3F)

Application Specific Files (file no. 0x10-0x37)

The System Files are dedicated to special tasks that are further described in the following sections (system files not covered in these sections are reserved for future extensions). All the remaining files are Application Specific Files and may be freely used in any desired manner as long as the first record (rec. no. 0x00) contains the header record as specified above in order to be conformant to this specification.

2.4.1 The Round Descriptor Lists (RODLs) (file no. 0x00-0x07)

An ST can only participate in a multi-partner (MP) round if the ST has the information about the structure of this MP round stored in one of its six RODLs. Each RODL file contains the ST-local description of one MP round. The numbers of the RODL files are 0x00, 0x02, 0x03, 0x04, 0x06 and 0x07 (see Table 2-2 on page 2-20).

RODL 0x01 and 0x05 are reserved as internal buffer for implementing the MSD and MSA round.

For a detailed description of the internal RODL format see [*EHKLOT02*].

2.4.2 The Configuration File (file no. 0x08)

The system file number 0x08 (Configuration File) contains the current *logical name*, the Identifier Compare Value (IDCV) and the sleep record. It is necessary for each ST node (master and slave) that supports plug and play or the sleep function. The layout of the Configuration File (0x08) is depicted below.

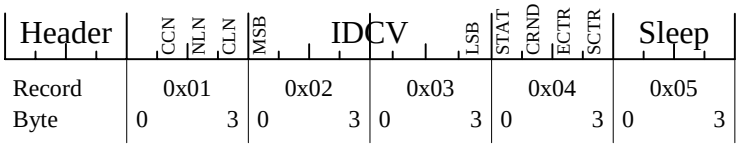

Figure 2-7 Layout of Configuration File

CCN: is the Currently assigned Cluster Name.

CLN: is the Currently assigned Logical Name.

NLN: is the New Logical Name used by the baptize algorithm.

IDCV: (optional) is needed by the baptize algorithm and stores the ID Compare Value.

STAT: is the current Status of the node.

CRND: is the number of the current round.

ECTR: is the current value of the Epoch-Counter.

SCTR: is the current value of the Slot-Counter.

2.4.3 The Membership File (file no. 0x09)

The Membership File contains two *membership vectors* of 256 bits (32 byte) each. The *logical name* of the ST is interpreted as an index to the 256 membership bits of the *membership vector.* The first *membership vector* contains all *slaves* that have sent a *life-sign* during the last sequence period. The *second membership vector* contains all *slaves*, which have responded correctly to the most recent MS operation (read or execute).

If there is no pending request by an external client, the master fills the empty MS slots by issuing a read operation to an ST. Eventually the master will have sent read operations to all addresses in the *(logical) name* space in order to update the *second membership vector.* Since the *second membership vector* is updated sporadically no guarantee about temporal accuracy of the *second membership vector* can be given. Refer to Figure 2-8.

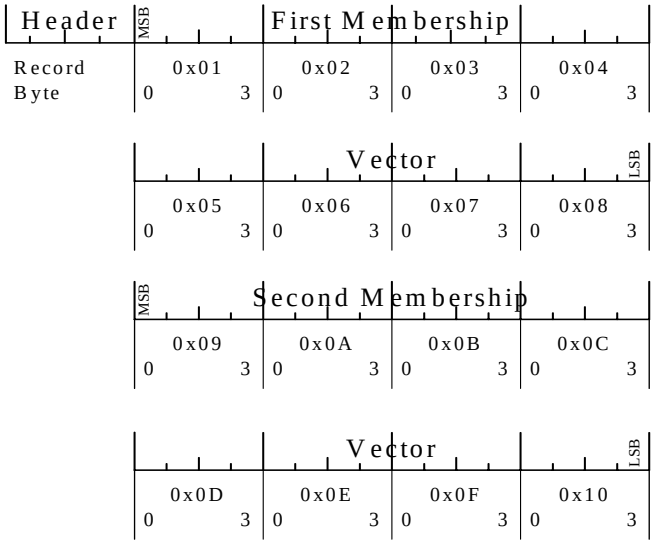

Figure 2-8 The Membership File

Example: The logical name 0x1F is assigned to the MSB (bit number 31) of record 0x08 for the *first membership vector* (respectively 0x10 in the *second membership vector*). If the respective bit is set, the node has been active in the last *sequence period.*

2.4.4 The Round Sequence (ROSE) File (file no. 0x0A)

A ROSE file makes sense for the master only and contains the specification of the start instants of a sequence of sequentially executable rounds and a *sequence period*, which determines after which duration this sequence must be repeated. The ROSE file consists of three sections. The first section is the status record. The second and third section each contain a sequence of MP round names. At any instant in time exactly one of the second or third section is active, while the other one is inactive. Modifications of the active section of the ROSE file are forbidden.

The Status record (0x01) describes which section of the ROSE file is currently active. It also contains the length of the second (and third) section of the ROSE file. Refer to Figure 2-9.

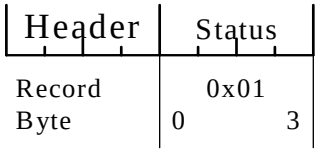

Figure 2-9 Status Record

Byte 0: 0 section two is active.
1 section three is active.
Byte 1: start record of section two
Byte 2: start record of section three

Section two of the ROSE file has the following format:

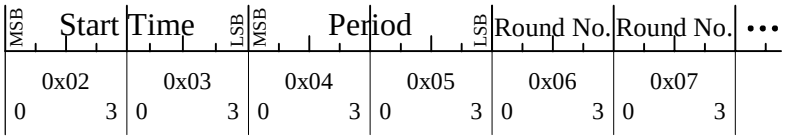

Figure 2-10 Section Two of ROSE File

This section contains the instant when the sequence should be started (start time) and the *sequence-period* (period).

Byte 0 of every following record contains in the three LSBs the name of the round to be issued. A set MSB of byte 0 signal that this is the last entry of a round-sequence. This end-of-round (EOR) bit must be cleared for all entries except the last.

Byte 1 contains the length of the inter-round gap (IRG). The IRG must be a positive integer multiple of one slot (13 bit cells) duration. Valid entries are 0x01, 0x02, …, 0x0F, referring to an IRG of the length of one slot, two slots, …, up to 15 slots.

The first entry must be an MSA entry (0x05, round number 5). All bits not specified above must be set to 0. Further every MSA round must have a complementary MSD round.

To change the active part of the ROSE file the address of the Status record (0x01) has to be part of an execute command. After finishing the current *round sequence* the master reads the other section of the ROSE file to do the new schedule.

2.4.5 The Owner File (file no. 0x0B)

The Owner File contains the “owner” (the logical name of the node which is allowed to write to the bus) of each slot of each round. This file must be consistent with the RODL files.

Rec. no. 0x01 and 0x02 are used as index containing the record no. of the first entry of each round. The remaining records contain a list of the logical names of the sending nodes of the respective round. The first entry of a round must always be in byte 0x00 of a record. See Figure 2-12.

The entry in the index for the MSA and MSD round (IndexA and IndexD) is unused but has been left in the index in order to allow consistent address calculations. These entries should be initialized with 0x00.

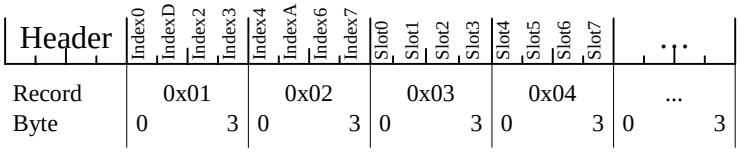

Figure 2-11 The Owner File

Unused slots are assigned to the logical name 0x00 which is reserved for broadcast and thus is not a valid logical name for an ST.

Since slot 0 is used for the fireworks byte the owner of slot 0 of each round is the master of the cluster.

2.4.6 The Documentation File (file no. 0x3D)

Every ST must contain at least the first, second, and third (0x00, 0x01, and 0x02) record of the documentation file 0x3D. This file is a read only file and contains the *physical name*, an eight-byte (64 bit) integer in record 0x01 and 0x02. The MSB is stored in the first byte. See Figure 2-12.

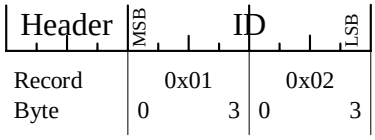

Figure 2-12 The Documentation File

With this information it is possible for the client to identify an ST and to access the documentation about the ST from the Internet. Implementations have the freedom to provide in the remaining records of the documentation file *read-only* documentation of this ST.

2.5 The Fireworks

The firework initiates the start of a round. The list of firework is depicted in Table 2-2. The fireworks (protected by a parity bit) have a Hamming distance of at least 4. To be able to distinguish between a firework and a normal data byte the parity for the fireworks have to be odd, while for normal data bytes even parity is used.

The MSA firework has been designed to generate a regular bit pattern for the start-up synchronization of ST nodes that contain an imprecise on-chip oscillator.

firework	Meaning	Description
0x78	RODL=0	multi-partner round 0
0x49	MSD	Master-Slave-Data
0xBA	RODL=2	multi-partner round 2
0x8B	RODL=3	multi-partner round 3
0x64	RODL=4	multi-partner round 4
0x55	MSA	Master-Slave-Address - Synchronize Round
0xA6	RODL=6	multi-partner round 6
0x97	RODL=7	multi-partner round 7

Table 2-2 Description of Firework-Bytes

2.6 Description of CORBA Based Object Model and Interfaces

2.6.1 Representation of Observed Transducer Data

In the proposed CORBA-*gateway* each RT observation (see Section 2.2.3) is represented by a structure consisting of 4 fields. For further details see [*OMG02*].

Finally, the fourth field contains the value of the RT entity. Normally, the fourth field will be 4 bytes long. If the value of an RT entity is longer, the value field will be extended accordingly to the application specific requirements.

2.6.2 Real-time Service (RS) Interface

The *real-time service* (RS) interface contains the RT images of the time-critical state variables of the ST system.

On input from the ST nodes to the CORBA gateway, these RT images (sensor values) are continuously updated from the addressed ST file records by the periodic MP rounds at *a-priori* known instants determined by the active ROSE/RODL files. Since the data at the RS interface is *state data*, a new version of a sensor value overwrites the old version. Fresh state data with known temporal characteristics is thus always available at the CORBA gateway and can be accessed by a CORBA method without any delay.

On output, the set-points for the actuators are periodically fetched at time instants computed *a-priori* and determined by the active ROSE/RODL file from the CORBA gateway and delivered to the addressed ST file record. Again *stateful data semantics* are assumed, viz., the available data value is not consumed on access. The real-time communication system that transports the RT observations is characterized by small

known delay and by a minimal jitter. Since the jitter is tightly controlled in the RS interface, the data can be used for time-sensitive real-time services; for example, distributed control loops.

2.6.3 Diagnostic and Management Interface

The *diagnostic and management* (DM) interface accesses application specific diagnostic and calibration data that is stored in application files at the ST nodes via *master-slave* (MS) rounds. Since in general the transport timing of MS rounds cannot be guaranteed, the DM interface should not be used for time-sensitive control data.

The representation of file specific information at the CORBA DM interface is the same as the representation of observations at the RS interface, as described above. Again data is treated as “state data”; that is, the contents of a new round overwrite the contents of the previous round.

2.6.4 Configuration and Planning Interface

The *configuration and planning* (CP) interface accesses the configuration data stored in the RODL files of the *slaves* and the ROSE file of the master by MS rounds.

The internal format of the RODL file (see [*EHKLOT02*]) is specific to an ST type and should be generated for a particular ST by an RODL generation tool from the abstract RODL specification.

2.7 Special Services

2.7.1 Node Identification—Plug and Play

Each node has a universally unique *physical name*, stored in the second and third (0x01 and 0x02) record of the documentation file. During operation, a node is not addressed by this unique *physical name*, but by a cluster unique 8-bit (one byte) *logical name* that is a shorter alias of the ST within a cluster. Additionally there are two *logical names* set aside for group addressing: the *logical name* 0x00 is reserved to address all STs of a cluster and the *logical name* 0xFF is reserved for addressing all *unbaptized* STs of a cluster. A node, newly connected to an ST cluster, must have either an *a-priori* assigned *logical name* or the special *logical name* 0xFF, which marks it as an *unbaptized* node. If the unique *physical name* of a new node is known, then a *baptize* operation; that is, the assignment of a l*ogical name* to this ST, can be started immediately, otherwise, the unique *physical name* must be retrieved by a special search algorithm. Node identification is an optional service.

If there are many *unbaptized* nodes connected to an ST cluster they all have the same *logical name* 0xFF at startup. Thus it is impossible to address exactly one *unbaptized* node by an MS round. In general, reading from multiple nodes via MS rounds is impossible. A special *execute* command to 0xFF will cause all *unbaptized* nodes to respond and thus inform the listener only about the existence of *unbaptized* nodes in the cluster. It is however possible to *write* to and *execute* at multiple nodes due to the

broadcast capability of the network. The two operations (write and execute) suffice to retrieve the *physical names* of the *unbaptized* nodes and thus solve the node identification problem.

The node identification uses a binary search algorithm over the entire code space of the unique node ids. It proceeds as follows:

1. An eight-byte Identifier Compare Value (IDCV) is written into system file "configuration" (0x08) of the IFS at each node with logical name 0xFF by a “broadcast” write.
2. A special record of system file "configuration" (0x08) is executed at all nodes with logical name 0xFF to compare its unique *physical name* with the IDCV.
3. Unbaptized Nodes with a unique *physical name* greater or equal to the IDCV respond, all other nodes stay silent.
4. As long as there exist some answering nodes the IDCV is raised and the algorithm is repeated. If no node answers, the IDCV must be lowered and the algorithm repeated.

This algorithm is repeated until exactly one IDCV is identified as an existing *physical name*.

An ST supporting the *baptize* feature must support an execute command to the first record of the IDCV (file: 0x08, record: 0x02). Executing this record the IDCV and the physical name (file: 0x3D, records: 0x01-0x02) is compared. If the *physical name* is greater or equal to the number stored in IDCV the node replies in the following *master-slave-data* (MSD) round with a single zero byte (the remaining slots of the MSD round remain empty), otherwise no answer is generated.

2.7.2 Baptizing of Nodes

If the *physical name* and the *logical name* are known, the *baptizing* operation proceeds as follows. The *master* writes (with a "broadcast" *logical name* 0xFF of the MS round) the *new logical name* and the *physical name* into special records of the system file "configuration" (0x08) of all nodes, which have not yet been baptized. It then performs an execute command on record (file: 0x08; rec: 0x01), which assigns the *new logical name* only to the node that has the same *physical name* in the documentation file as the *physical* name that has been previously written into the system file "configuration" (file: 0x08; rec: 0x02-0x03).

For baptizing, record number 0x01 of system file "configuration" (0x08) must have an execute operation assigned. Executing this record the *New Logical Name* (NLN) replaces the *logical name* if the IDCV (file: 0x08, record: 0x02-0x03) matches the physical name (file: 0x3D, records: 0x01-0x02).

The *logical name* 0xFF means that an ST is currently not integrated in the system. Such an ST must not answer any MS or MP request except the two MS-execute commands required for the baptize algorithm (MS-execute of file: 0x08, record: 0x01 and MS-execute of file: 0x08, record: 0x02). MS-write operations have to be performed without respect to the *logical name*.

Extremely low-cost nodes produced in large quantities for a particular application (e.g., in consumer electronics, or automotive applications) may not include the functionality for baptizing. These nodes may have an *a-priori* assigned *logical name*. This can be done outside the system context (e.g., during manufacturing). Since all logical names of an ST cluster must be different, only one node with a particular hard-coded *logical name* may be part of an ST cluster.

2.7.3 Wakeup and Sleep Service

A node can be forced into *sleep mode* by executing a special record in the system file "configuration" (0x08) of the IFS. Since each node can be accessed by the broadcast *logical name* (0x00) it is possible to force the entire ST cluster into sleep mode with a single sleep command. During sleep, a node is in a *save-power* state and has only very limited functional capabilities. In the sleep state, there is no activity on the bus. Wakeup occurs if a sleeping node detects activity on the bus or is woken up by a node specific local event.

Executing the Sleep record (file: 0x08, record: 0x05) sends the node to sleep mode.

Since a slave node is not resynchronized while sleeping, the *master* has to be aware that the clock of a node that just woke up may be unsynchronized. After receiving a wake up signal on the bus the *master* initiates an MSA round to synchronize and wake up all nodes within the cluster.

2.8 UART Transport Protocol

The predictable ST transport service can be implemented by a byte oriented UART protocol on a broadcast communication channel. Any communication is initiated by sending a firework from the *master* according to the time-triggered schedule stored in the active ROSE file. There are no collisions on the communication channel.

2.8.1 Bus Access

Whenever the time reaches an instant stored in the active ROSE file of a *master*, it will output the specified firework on the communication channel. In the UART protocol, a slot for the transmission of one byte has a length of 13 bits, composed as follows:

<start bit; eight data bits; parity bit; stop bit; inter-frame gap of two bits>

All bytes sent by the *master* must start precisely at the *a-priori* specified instants (*start-instant* of the round plus an amount - *bytecount* x *bitduration* x 13).

In the UART implementation, the rounds have the following structure:

Master Slave Address (MSA) round (six bytes):

<firework, epoch, logical name, file-name and command, record name, check byte>

The check byte is calculated by an exclusive OR over the first five bytes of the MSA round.

Master Slave Data (MSD) round (six bytes):

<firework, data byte 1, data byte 2, data byte 3, data byte 4, check byte>

Byte 0 (the most significant byte) of the record is transmitted first. The check byte is calculated by an exclusive OR over the first five bytes of the MSD round.

Multi-partner round (up to 64 bytes, according to RODL specification):

<firework for selected RODL, data byte 1, data byte 2, . . . data byte n>

2.8.2 Timing

Whenever the master sends a firework for a new round, a new epoch is started at the slaves. The starting instant of this new epoch is the first (falling) edge of the start-bit of the firework. In an MSA round the master provides the number of the current epoch in the byte following the firework. The slave measure time by counting the slots (or fractions thereof) after the epoch (internal time representation).

The sequence of rounds between the *start instant* of an MSA round and the *start instant* of the next MSA round is the *period of the schedule* contained in the RODL file. The duration of the inter-round gap between the last round of a period and the first round of the next period may be used to synchronize the start of the next MSA round with the external time (*variant slack*). All other inter-round gaps within a period must last a positive integer-multiple of precisely 13 bit lengths.

Since the maximum length of a MP round is 78 slots (fireworks plus up to 62 data bytes plus up to 15 slots for Inter Round Gap) and the slot counter is reset to 0x00 with the beginning of each epoch (at the beginning of each round), the slot counter easily fits into an 8 bit register. Thus the pair epoch counter and slot counter may be used as timestamp on ST level. Since every cluster can have differing transmission speeds (and time formats) the *master* must transform the internal time representation to the external time representation.

2.8.3 Start-up Synchronization and Re-synchronization

A node with an imprecise oscillator (e.g., RC on chip oscillator) must adjust its clock after startup and periodic during operation. For this purposes the firework 0x55 was chosen to be the fireworks of the MSA round. This firework has a very regular bit pattern. Further a read-synchronize command is defined during MP rounds. This command can be used to resynchronize the node's clock on reading a message originating from a node with a highly reliable oscillator.

2.8.4 Physical Layer

The UART protocol can be based on different physical layers. The two major requirements are:

- It must be possible to transport UART messages via the bus.

- In the case where a baptize service is available, concurrent write operations (all *slaves* write the same value at almost the same instant) to the bus must be supported and the *master* must be able to detect in such a case that there is some traffic on the bus. It is not a requirement that the *master* can read this data correctly.

  For example, the ISO9141 (ISO K Line), the RS485 and several other bus standards fulfill these requirements.

Conclusion 1

The TTP/A protocol has been designed to meet the following requirements:

1.1 Low Cost

A field bus will only be accepted in the market place, if its introduction reduces the overall system cost and, at the same time, increases the system dependability. The TTP/A field bus provides the following potentials for cost reduction at the system level:

Software cost: Standardized interfaces between system nodes and transducer nodes will lead to a significant cost reduction in the development process. The validation effort for the implementation of new applications will also decrease.

Transducer nodes: The TTP/A field bus is based on an universal asynchronous receiver transmitter (UART). This is a widely available component of the shelf (COTS) interface standard. Nearly all low-cost microcontrollers that are available on the market can be used as transducer nodes. The UART can be implemented either in hardware or in software.

System nodes: Special I/O interfaces of the system nodes are replaced by a UART. This reduces the pin count of a system node and makes a particular system node design more versatile.

Cabling: Since point-to-point cabling between the controller and each sensor (respectively actuator) is replaced by a bus where all the transducers and system nodes are plugged in, the wire length and the number of cabling connections is remarkably reduced.

1.2 Minimal Jitter

TTP/A provides deterministic communication with minimal and a priori known jitter. This increases the quality of control in control loops that are closed via the TTP/A field bus.

1.3 Autonomy of Transducer Subsystems

A TTP/A fieldbus subsystem operates autonomously and does not require any host CPU services for its operation. This autonomy of the field-bus subsystem reduces the complexity and simplifies the certification at the system level.

1.4 Architecture Conformance

From the point of view of system nodes, the interface to transducers connected, via the TTP/A field bus, is identical to remote transducers, which are relayed by the Time-Triggered Protocol for Class C Applications (TTP/C) from another system node. This unification of interfaces simplifies the system wide sharing of sensor information.

To our knowledge, none of the widely discussed field-bus protocols meets all of the above requirements. Since the smart transducers interface provided by TTP/A conforms to a world-wide standard it can be considered an interesting option for various sensor network applications. The interface provides many features that are required by fieldbus applications for automotive or automation industries. Supported features are the real-time capability, the encapsulation of the node's internals, and a universal address space with the interface file system. TTP/A can be implemented on low-cost Commercial-off-the-Shelf (COTS) hardware and supports various bus media types.

References 2

[OMG02] OMG. Smart Transducers Interface. Specifi*cation ptc/2002-05-01, Object Management Group, May 2002. Available at http://www.omg.org.*

[Kop97] Hermann Kopetz. Real-Time Systems, Design Principles for Distributed Embedded Aplications. Boston, Kluwer Academic Press, 1997.

[EHKLOT02] Wilfried Elmenreich, Wolfgang Haidinger, Raimund Kirner, Thomas Losert, Roman Obermaisser, Christian Trödhandl. TTP/A Smart Transducer Programming - A Beginner's Guide. Research Report 33/2002, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, 2002.