Smart Transducers Interface Specification

PTC/2002-09-11

Final Adopted Specification
August 2002

PATENT

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

NOTICE

The information contained in this document is subject to change without notice. The material in this document details an Object Management Group specification in accordance with the license and notices set forth on this page. This document does not represent a commitment to implement any portion of this specification in any company's products.

ISSUE REPORTING

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents & Specifications, Report a Bug/Issue.

Table of Contents

Preface

About the Object Management Group

The Object Management Group, Inc. (OMG) is an international organization supported by over 600 members, including information system vendors, software developers and users. Founded in 1989, the OMG promotes the theory and practice of object-oriented technology in software development. The organization's charter includes the establishment of industry guidelines and object management specifications to provide a common framework for application development. Primary goals are the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments. Conformance to these specifications will make it possible to develop a heterogeneous applications environment across all major hardware platforms and operating systems.

OMG's objectives are to foster the growth of object technology and influence its direction by establishing the Object Management Architecture (OMA). The OMA provides the conceptual infrastructure upon which all OMG specifications are based.

What is CORBA?

The Common Object Request Broker Architecture (CORBA), is the Object Management Group's answer to the need for interoperability among the rapidly proliferating number of hardware and software products available today. Simply stated, CORBA allows applications to communicate with one another no matter where they are located or who has designed them. CORBA 1.1 was introduced in 1991 by Object Management Group (OMG) and defined the Interface Definition Language (IDL) and the Application Programming Interfaces (API) that enable client/server object interaction within a specific implementation of an Object Request Broker (ORB). CORBA 2.0, adopted in December of 1994, defines true interoperability by specifying how ORBs from different vendors can interoperate.

OMG Documents

The OMG documentation is organized as follows:

OMG Modeling

- ***Unified Modeling Language (UML) Specification*** defines a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems.
- ***Meta-Object Facility (MOF) Specification*** defines a set of CORBA IDL interfaces that can be used to define and manipulate a set of interoperable metamodels and their corresponding models.
- ***OMG XML Metadata Interchange (XMI) Specification*** supports the interchange of any kind of metadata that can be expressed using the MOF specification, including both model and metamodel information.

Object Management Architecture Guide

This document defines the OMG's technical objectives and terminology and describes the conceptual models upon which OMG standards are based. It defines the umbrella architecture for the OMG standards. It also provides information about the policies and procedures of OMG, such as how standards are proposed, evaluated, and accepted.

CORBA: Common Object Request Broker Architecture and Specification

Contains the architecture and specifications for the Object Request Broker.

OMG Interface Definition Language (IDL) Mapping Specifications

These documents provide a standardized way to define the interfaces to CORBA objects. The IDL definition is the contract between the implementor of an object and the client. IDL is a strongly typed declarative language that is programming language-independent. Language mappings enable objects to be implemented and sent requests in the developer's programming language of choice in a style that is natural to that language. The OMG has an expanding set of language mappings, including Ada, C, C++, COBOL, IDL to Java, Java to IDL, Lisp, and Smalltalk.

CORBAservices

Object Services are general purpose services that are either fundamental for developing useful CORBA-based applications composed of distributed objects, or that provide a universal-application domain-independent basis for application interoperability.

These services are the basic building blocks for distributed object applications. Compliant objects can be combined in many different ways and put to many different uses in applications. They can be used to construct higher level facilities and object frameworks that can interoperate across multiple platform environments.

Adopted OMG Object Services are collectively called CORBAservices and include specifications such as *Collection*, *Concurrency*, *Event*, *Externalization*, *Naming*, *Licensing*, *Life Cycle*, *Notification*, *Persistent Object*, *Property*, *Query*, *Relationship*, *Security*, *Time*, *Trader*, and *Transaction*.

CORBAfacilities

Common Facilities are interfaces for horizontal end-user-oriented facilities applicable to most domains. Adopted OMG Common Facilities are collectively called CORBAfacilities and include specifications such as *Internationalization and Time*, and *Mobile Agent Facility.*

Object Frameworks and Domain Interfaces

Unlike the interfaces to individual parts of the OMA "plumbing" infrastructure, Object Frameworks are complete higher level components that provide functionality of direct interest to end-users in particular application or technology domains.

Domain Task Forces concentrate on Object Framework specifications that include Domain Interfaces for application domains such as Finance, Healthcare, Manufacturing, Telecoms, E-Commerce, and Transportation.

Currently, specifications are available in the following domains:

- *CORBA Business*: Comprised of specifications that relate to the OMG-compliant interfaces for business systems.
- *CORBA Finance*: Targets a vitally important vertical market: financial services and accounting. These important application areas are present in virtually all organizations: including all forms of monetary transactions, payroll, billing, and so forth.
- *CORBA Healthcare*: Comprised of specifications that relate to the healthcare industry and represents vendors, healthcare providers, payers, and end users.
- *CORBA Manufacturing*: Contains specifications that relate to the manufacturing industry. This group of specifications defines standardized object-oriented interfaces between related services and functions.
- *CORBA Telecoms*: Comprised of specifications that relate to the OMG-compliant interfaces for telecommunication systems.
- *CORBA Transportation*: Comprised of specifications that relate to the OMG-compliant interfaces for transportation systems.

Obtaining OMG Documents

The OMG collects information for each book in the documentation set by issuing Requests for Information, Requests for Proposals, and Requests for Comment and, with its membership, evaluating the responses. Specifications are adopted as standards only when representatives of the OMG membership accept them as such by vote. (The policies and procedures of the OMG are described in detail in the *Object Management Architecture Guide*.)

OMG formal documents are available from our web site in PostScript and PDF format. To obtain print-on-demand books in the documentation set or other OMG publications, contact the Object Management Group, Inc. at:

OMG Headquarters
250 First Avenue
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
pubs@omg.org
http://www.omg.org

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

IDL appears using this font.

Language Mapped code appears using this font.

Important Reminders appear using this font.

In various places a few issues are highlighted. These are mostly areas where we have discovered that some additional clarification may be needed.

A statement expressing a requirement will highlight its verb in bold, eg. shall or may not.

Acknowledgments

The following companies submitted and/or supported parts of this specification:

- TTTech Computertechnik A.G.
- VERTEL Corporation

This is a joint submission of TTTech Computertechnik A.G., Vertel Corporation (USA), and the Technische Universität Wien, which acted as a subcontractor to TTTech. This work has been supported by the European Research Project DSOS and the research project TTSB, conducted jointly between the Technische Universität Wien, Austria, the Universität Stuttgart, Germany and the Technische Universität Munich, Germany.

Introduction 1

1.1 Guide to the Specification

This document describes the specification for a set of smart transducer interfaces that supports the following properties

1. The provision of a standardized set of functions, or service to a user in order to operate, configure and diagnose a generic transducer device.
2. An encapsulation of the internal complexity of the generic smart-transducer hardware and software and the internal transducer failure modes to reduce the complexity at the system level.
3. Describe a canonical form of a communication service, or protocol with small delay and minimal jitter that is tailored to operate on low bandwidth channels given severely constrained environments.

1.2 Proof of Concept

The smart transducer interface described in this specification has been implemented on a number of different micro-controllers. The resource requirements on an 8 bit micro controller, including the communication protocol, are less than 4 kByte of ROM and 64 Bytes of RAM memory.

The specification is derived from experiences in prototyping design and implementation performed by TTTech Computertechnik AG and Vertel Corporation, and partners (through e*ORB real-time products used in Telematics, on-board vehicle systems, and the application to various prototype consumer electronics applications) in time triggered real-time vehicular systems, telematics, industrial automation and control.

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 (see Section 3.1, “Mandatory Requirements,” on page 3-1) 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.

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* ~~maintenance~~ *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 2-20 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 2-22. The UART transport protocol is specified in Section 2.8, “UART Transport Protocol,” on page 2-24.

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.

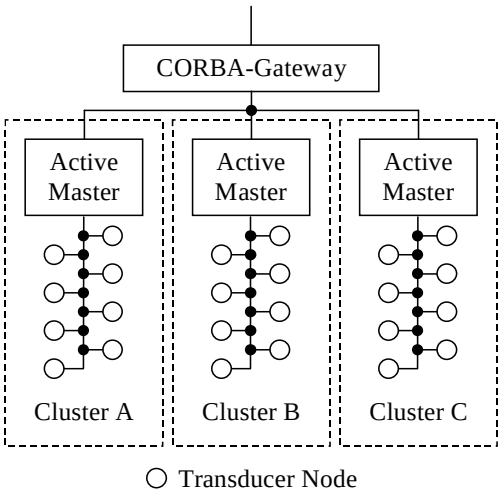

Figure 2-1 Transducer System with 3 clusters

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 ~~maintenance~~ 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.

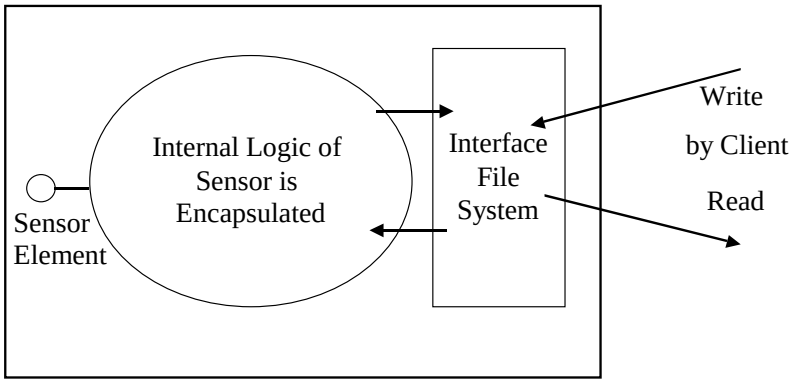

Figure 2-2 Interface File System in a Smart Transducer

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 - Kopetz, H. (1997). *"Real-Time Systems, Design Principles for Distributed Embedded Applications"*, ISBN: 0-7923-9894-7, Fourth printing 2001. Boston, Kluwer Academic Press.

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 ~~maintenance~~ 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 ~~maintenance~~Management Interface: The diagnostic and ~~maintenance~~ 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 2-24.

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 (see Section 3.5.5, "XML Description of a RODL," on page 3-8). 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 2-22 and Section 2.7.2, "Baptizing of Nodes," on page 2-23). 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.

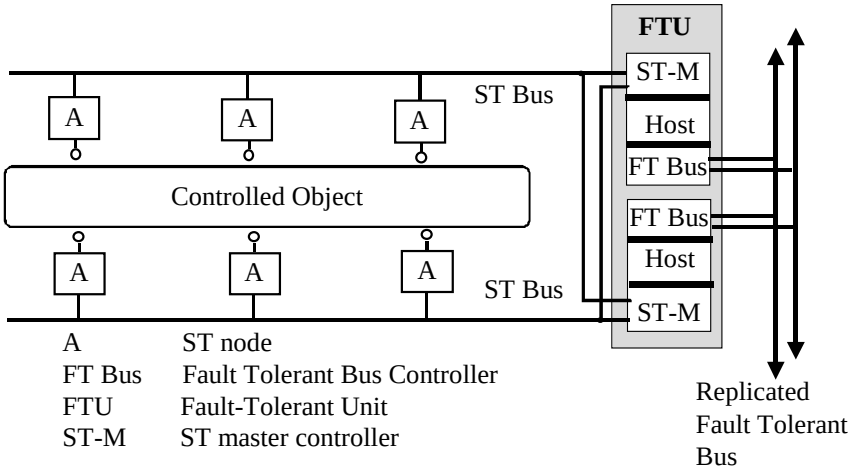

Figure 2-3 Fault-Tolerant Sensor Subsystem-

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, e.g., 0x00 is reserved for broadcast messages. A detailed description of these values is in Section 3.5, "Specification of Data Representation," on page 3-6.

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

~~Together with the 64 file names (6 bits), which an ST can hold, the file operation and the file name can be fitted into a single byte.~~

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 ~~three~~ two kinds of file accesses, called a *master-slave* (MS) round, a *multi-partner* (MP) ~~round and a *broadcast*~~ round. The MS rounds are used to implement the *diagnostic and* ~~*maintenance*~~ *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. ~~The *broadcast rounds* are used to implement operations that must be executed by all nodes of a cluster.~~

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~~+file name~~><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.

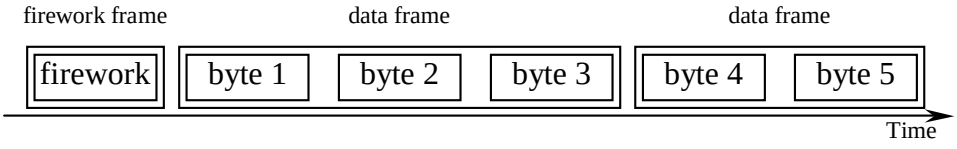

Figure 2-4 Structure of MP rounds

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.
4. Type of protection of the *data-frame* (none, four-bit or eight-bit checksum).

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

~~Between any two MP rounds there must be an interval of sufficient duration to execute one phase of an MS round, as depicted in Figure 2-5.~~ 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.

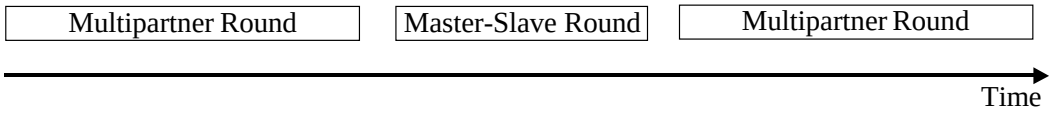

Figure 2-5 Interleaving of MP and MS rounds

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. ~~Each frame of an MP round can be protected by a four-bit or eight-bit checksum (coded in the RODL).~~
4. When designing MP rounds, ~~check~~ protected data bytes can be ~~included~~ used if required by the application scenario.
5. The data patterns in the firework byte have been carefully designed to provide a Hamming distance of 4.
6. 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 plus an offset of 2^{38} seconds~~1980. ~~The offset is introduced to be able to express~~ Thus instants before January 6, 1980 are expressed as ~~positive~~ 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 ~~MSA firework~~8-bit epoch counter is incremented by one. ~~To be able to~~ Thus each *slave* can distinguish between ~~time values belonging~~ 256 consecutive epochs. In order to allow a slave to (re)integrate to ~~successive rounds,~~ the system the master transmits ~~an 8~~ its 8-bit epoch-counter ~~at~~ with each MSA ~~round thus allowing each *slave* to distinguish between 256 consecutive epochs~~round. ~~To save bandwidth, this~~ 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.

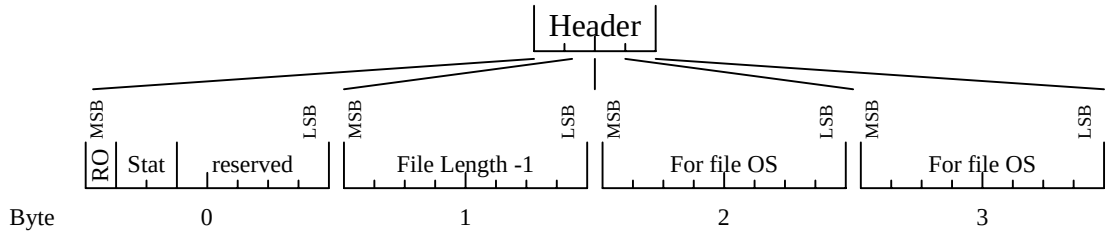

Figure 2-6 Addressing Space

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.

The standardized generic RODL format, expressed in XML, is contained in Section 3.5.5, “XML Description of a RODL,” on page 3-8. This generic RODL file format contains the name of the MP round (which corresponds to the RODL file name) and the information where the data that is involved in this round is located in the node-local IFS, what action to perform, and at which position of the MP round the data is placed.

A software development tool must transform this abstract RODL format into the concrete format required by a particular ST. This concrete format can then be downloaded into the ST by using the MS rounds.

2.4.2 The ~~System~~ Configuration File (file no. 0x08~~-0x0C and 0x3E, 0x3F~~)

~~There are 8 system files in an ST. No application may use any of the records of the system files for application specific functions.~~ The system file number 0x08 (~~configuration file~~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.

~~The configuration file (0x08) is mandatory for each ST node (master and slave). The layout of the Configuration file (0x08) is depicted below.~~

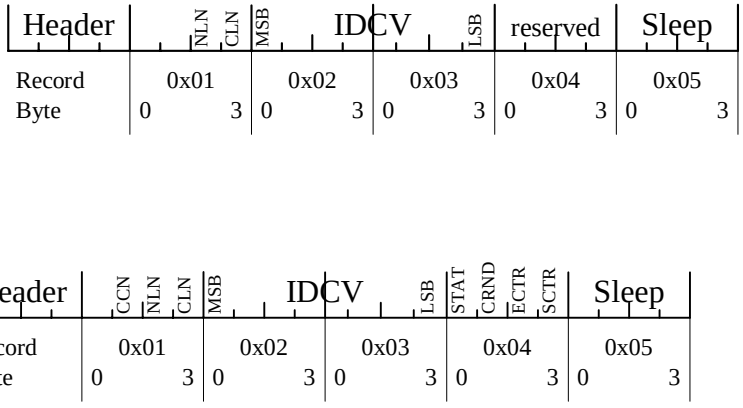

Figure 2-7 Layout of Configuration File

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.

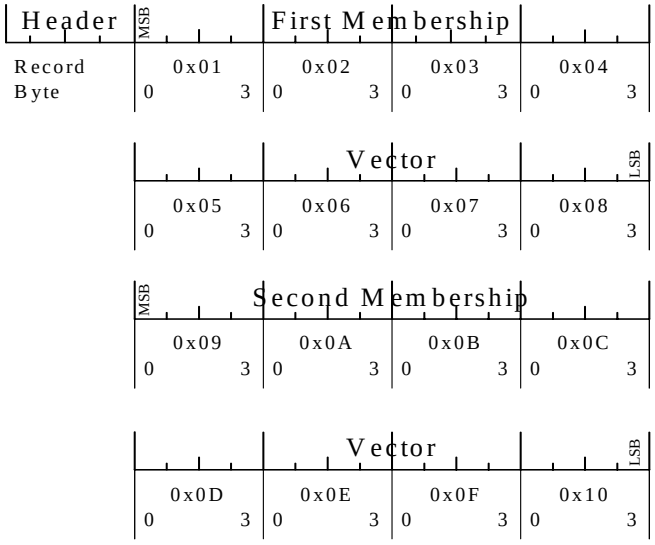

Figure 2-8 The Membership File

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.

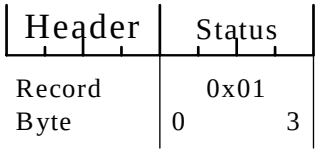

Figure 2-9 Status Record

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:

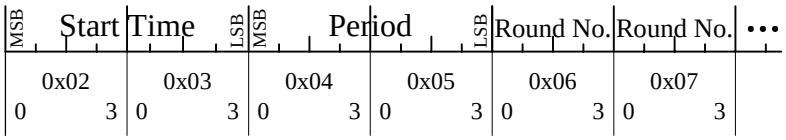

Figure 2-10 Section Two of ROSE File

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 Application Specific Files (0x0D - 0x3C)~~

~~There are 48 files available in each ST node for applications, to store sensor, actuator or diagnostic data. The first record (0x00) of every file must be a header record (as specified in 2.4). The programmer is free to design the layout of remaining records of the application specific files as desired.~~

2.4.6 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.

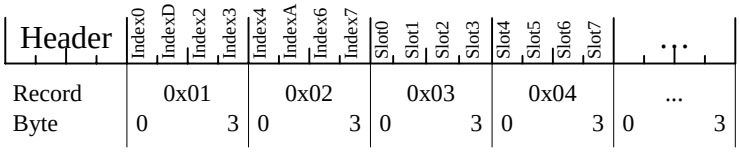

Figure 2-11 The Owner File

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.

2.4.7 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.

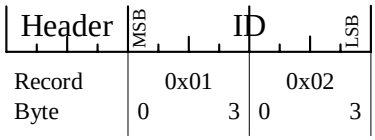

Figure 2-12 The Documentation File

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. The first four-byte field contains the hierarchical address consisting of *cluster name, logical name, file name,* and *record name* and thus identifies the name of the observation. The second 8-byte field contains

the time of the observation, expressed in the external time format described in Section 2.3.8, "Global Time," on page 2-13. The third four-byte field contains the attributes as described in Section 3.5, "Specification of Data Representation," on page 3-6. 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.

~~In the proposed CORBA *gateway* each RT observation (see Section 2.2.3) is represented by a structure consisting of 4 fields. The first four-byte field contains the hierarchical address consisting of *cluster name*, *node name*, *file name* and *record name* and thus identifies the name of the observation. The second 8-byte field contains the time of the observation, expressed in the external time format described in Section 2.3.8. The third four-byte field contains the attributes as described in Section 3.6. 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 ~~*Maintenance interfaces*~~Management Interface

The *diagnostic and* ~~*maintenance*~~ *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 is specific to an ST type and must be generated for a particular ST by an RODL generation tool from the abstract RODL specification contained in Section 3.5.5, "XML Description of a RODL," on page 3-8.

2.7 Special Services

2.7.1 Node Identification—Plug and Play

Each node has a universally unique *physical name*, stored in the ~~first two records~~ 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~~"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 ~~an MSA~~ 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.

~~The~~ Since the maximum length of ~~two MS~~ a MP round ~~and two MP rounds~~ is ~~172~~ 78 slots (fireworks plus up to 62 data bytes plus up to 15 slots for Inter Round Gap) and the ~~*variant slack* before~~ slot counter is reset to 0x00 with the ~~next MSA firework. Thus~~ beginning of each epoch (at the ~~time stamp~~ beginning of ~~a slave can be correctly resolved by~~ each round), the ~~master if the~~ slot ~~number do not exceed 255~~counter easily fits into an 8 bit register. ~~Otherwise,~~ Thus the pair epoch ~~number must~~ counter and slot counter may be ~~provided by the slave to identify the epoch~~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.

Requirements and IDL 3

This chapter attempts to prioritize key focal elements with regards to mandatory versus optional requirements from the smart transducers interface RFP (OMG Orbos/2000-12-13) based on current observed industrial practices. In addition a brief description of the initial proposed IDL interfaces is given with a detailed breakdown of the individual data and interfaces composed therein. It is the intent that these interfaces will undergo some refinement as the specification progresses.

~~*3.1 Mandatory versus Optional Requirements for Smart Transducers (ST)*~~

3.2 Mandatory Requirements

Every ST must support MS rounds in order that one can read the *physical name* ~~and the *logical name*~~ of an ST from the CORBA interface. This function is necessary to learn about the existence of an ST in a cluster.

Every ST must support at least reading of the first, second, and third (0x00, 0x01, and 0x02) record of the documentation file 0x3D as described in Section 2.4.6, “The Documentation File (file no. 0x3D),” on page 2-19 in order to allow reading the physical name.

Every master must provide the capability to translate the representation of the ~~external~~ internal time of its cluster to the representation of the ~~internal time of its cluster and vice versa~~external time.

Every CORBA gateway must provide the three interfaces: the RS interface, the DM interface, and the CP interface.

3.3 Optional Requirements

~~An implementor is free to select one or more additional functions from the following list of optional requirements. If such a function is implemented in an ST, is must be implemented according to the specification provided by this standard proposal.~~

In addition to the mandatory services an ST compliant to this specification may support an arbitrary set of the following optional services:

Fixed MP Round

A low-cost ST that is used in mass-market applications may contain only one or more *a-priori* preprogrammed RODLs in its ROM memory. Other low-cost nodes may not support any MP round.

Programmable MP Round

This function makes it possible to change the RODL within an ST on-line.

Baptizing of STs without an assigned Logical Name

This function enables the on-line assignment of a logical name to an ST that has no *a-priori* assigned logical name (the logical name 0xFF denotes an ST currently not integrated in the system).

Identification of STs (Plug and Play)

This function enables the on-line identification of a new node that has been detected to exist.

Sleep and Wakeup Service

This function enables a cluster to enter the sleep mode and to wake-up the cluster after a significant event has occurred.

First Membership~~ at Master~~

This function provides membership information about STs that participate in MP rounds.

Second Membership~~ at Master~~

This function provides membership information about currently available STs in a cluster.

Maintain Epoch Counter

This function enables a slave to maintain the Epoch Counter and thus provide timestamps within an ST.

Programmable ROSE file at Master

This function enables the on-line modification of the round sequence (ROSE) list at the master.

~~3.4 Mandatory versus Optional Interfaces at the CORBA Gateway~~

~~3.4.1 Mandatory Interfaces~~

~~The CORBA gateway must provide the tree interfaces, the RS interface, the DM interface and the CP interfaces.~~

~~3.4.2 Optional Interfaces~~

Convert External Time to Cluster-Internal Time

This function enables an ST to convert the external time into the cluster internal format.

External Clock Synchronization at Master or CORBA Gateway

The master or the CORBA gateway may provide an external clock synchronization interface in order to synchronize the actions of the master with an external time reference (e.g., GPS).

~~3.5 Proposed Compliance Points~~

~~All future extensions of ST services must be compatible with the services specified in this proposal.~~

Application Specific Extensions

An ST may support read-, write-, or execute-operations on application specific files in order to fulfill application specific purposes as long as record 0x00 is used as header record as described in Section 2.4, “Smart Transducer Filesystem in the ST,” on page 2-14.

3.6 Changes or Extensions required to adopted OMG Specifications

~~The external representation of time provided in this standard should be included in the CORBA time services.~~

A Smart Transducer object service may be used by an object to bootstrap itself into operation; as such, this specification mandates an additional ObjectId for use in the resolve_initial_references() operation defined in the ORB Initialization Specification, OMG Document 94-10-24.

The following ObjectId is reserved for finding an initial Smart Transducer object service:

SmartTransducer

No other extensions are proposed to OMG IDL, CORBA, and/or the OMG object model.

3.7 Complete IDL Definitions

This section contains the complete IDL definitions of the three CORBA Interfaces of a smart transducer system: the RS interface, the DM interface, and the CP interface.

The following code has been verified by using *e*ORB idlc* for 2 different languages (C++, Java), and orbit-idl (ORBit-devel-0.5.13-3).

```
//
// File: sti.idl
//

#ifndef _ST_INTERFACE_IDL_
#define _ST_INTERFACE_IDL_
#include <orb.idl>
#pragma prefix "omg.org"

module SmartTransducer {

    exception NoCluster {};
    exception NoNode {};
    exception NoFile {};
    exception NoRecord {};
    exception Damaged {};
    exception DataNotReady {};
    exception NoExecutable {};
    exception ReadOnly {};
    exception NoMessageReceived {};
    exception CommunicationError {};
    exception TimeError {};
    exception NotSupported {};
```

```
interface RShandle;
interface DMhandle;
interface CPhandle;

typedef octet ClusterID;
typedef octet logicalNameID;
typedef octet FileID;
typedef octet RecordID;
struct NameID {
    ClusterID cluster;
    logicalNameID logicalName;
    FileID file;
    RecordID record;
};

typedef unsigned long long TimeInstant;
typedef unsigned long long TimeDuration;

struct TakeInstant {
        TimeInstant Instant;
        TimeDuration Period;
    };
    struct DeliveryInstant {
        TimeInstant Instant;
        TimeDuration Period;
    };
struct Instants {
    TimeInstant instant;
    TimeDuration period;
};

typedef unsigned long long AttributesData;
struct AttributesData {
    octet ERR;
    octet CONF;
    octet PREC;
    unsigned short USER;
};

typedef unsigned long long RecordData;
```

```
typedef octet RecordData[4];

typedef any DeviceClusterDescriptor;

// Time-Triggered Real-Time Interface
interface RShandle {
    void Read( in NameID Name,
        out TimeInstant Time,
        inout AttributesData Attr,
        out RecordData Data
) raises (
            NoCluster,
            NoNode,
            NoFile,
            NoRecord,
            Damaged,
            DataNotReady,
            NoMessageReceived,
            CommunicationError,
            TimeError,
            NotSupported
        );
        void ReadDeliveryInstant
        void ReadDeliveryInstants(
            in NameID Name,
            out Instants Instant,
            out octet ERR
) raises (
            NoCluster,
            NoNode,
            NoFile,
            NoRecord,
            Damaged,
            DataNotReady,
            NoMessageReceived,
            CommunicationError,
            TimeError,
            NotSupported
        );
        void Write( in NameID Name,
            in TimeInstant Time,
            inout AttributesData Attr,
            in RecordData Data
        ) raises (
            NoCluster,
            NoNode,
            NoFile,
            NoRecord,
            ReadOnly,
            NoMessageReceived,
            CommunicationError,
```

```
                TimeError,
                NotSupported
            );
            void ReadTakeInstants(
            void ReadTakeInstant(
                in NameID Name,
            out TakeInstant Instant
                out Instants Instant,
                out octet ERR
) raises (
                NoCluster,
                NoNode,
                NoFile,
                NoRecord,
                Damaged,
                DataNotReady,
                NoMessageReceived,
                CommunicationError,
                TimeError,
                NotSupported
            );
        };

        // Diagnostic and maintenance interfaceManagement Interface
        interface DMhandle {
            void Read( in NameID Name,
                inout AttributesData Attr,
                out RecordData Data
            ) raises (
                NoCluster,
                NoNode,
                NoFile,
                NoRecord,
                Damaged,
                DataNotReady,
                NoMessageReceived,
                CommunicationError,
                TimeError,
                NotSupported
            );
            void Write( in NameID Name,
                inout AttributesData Attr,
                in RecordData Data
            ) raises (
                NoCluster,
                NoNode,
                NoFile,
                NoRecord,
                ReadOnly,
                NoMessageReceived,
                CommunicationError,
```

```
                TimeError,
                NotSupported
            );
            void Execute( in NameID Name,
                inout AttributesData Attr
            ) raises (
                NoCluster,
                NoNode,
                NoFile,
                NoRecord,
                Damaged,
                DataNotReady,
                NoExecutable,
                NoMessageReceived,
                CommunicationError,
                TimeError,
                NotSupported
            );
        };

        // Configuration and Planning Interface
        interface CPhandle {
            void Read( in NameID Name,
                inout AttributesData Attr,
                out RecordData Data
            ) raises (
                NoCluster,
                NoNode,
                NoFile,
                NoRecord,
                Damaged,
                DataNotReady,
                NoMessageReceived,
                CommunicationError,
                TimeError,
                NotSupported
            );
            void Write( in NameID Name,
                inout AttributesData Attr,
                in RecordData Data
            ) raises (
                NoCluster,
                NoNode,
                NoFile,
                NoRecord,
                ReadOnly,
                NoMessageReceived,
                CommunicationError,
                TimeError,
                NotSupported
```

```
        );
        void Execute( in NameID Name,
            inout AttributesData Attr
        ) raises (
            NoCluster,
            NoNode,
            NoFile,
            NoRecord,
            Damaged,
            DataNotReady,
            NoExecutable,
            NoMessageReceived,
            CommunicationError,
            TimeError,
            NotSupported
        );
    };

    // Smart Transducer Abstraction Handle
    struct STDevice {
        DeviceClusterDescriptor device;
        NameID name;
        RShandle rsh;
        DMhandle dmh;
        CPhandle cph;
    };

    typedef sequence<STDevice> devices;

    interface Current : CORBA::Current {
        attribute devices dev;
    };

};

#endif // _ST_INTERFACE_IDL_
```

3.8 Specification of Data Representation

octet ClusterID: This type may have values from 0 to 255 and is used to address a specific cluster (the ClusterID 0 is reserved for broadcast; the ClusterID 251-252 and 254 is reserved for future extensions; the ClusterID 253 is the gateway; the ClusterID 255 is reserved for integrating new clusters).

octet logicalNameID: This type may have values from 0 to 255 and is used to address the nodes (the logicalName 0 is reserved for broadcast; the logicalName 251-252 is reserved for future extensions; the logicalName 253 is the gateway; the logicalName 254 is the master; the logicalName 255 is reserved for integrating new nodes).

octet FileID: This type may have values from 0 to 63 and is used to address a specific file. The values from 64 to 255 are invalid!

octet RecordID: This type may have values from 0 to 255 and is used to address a specific record in a file.

struct NameID: This struct has the subfields cluster, logicalName, file, and record and is used for addressing a specific record of the IFS.

any DeviceClusterDescriptor: This is a CORBA any used specifically to allow for the incorporation of devices or legacy hardware that may only have generic MIB like identifiers, which may be placed in such an any by inserting a user definable IDL struct.

sequence<STDevice> devices: This sequence is provided to hold a set of devices that may be being accessed in many different ways from the same current logical locus of execution or current programming context.

~~unsigned~~ long long TimeInstant: This type is used for timestamps. The 40 upper bits represent the number of seconds (all 34841 years an overflow will occur) while the remaining 24 bits represent the fractions of a second, allowing an accuracy of 60 ns. In a system with external clock synchronization the 40 upper bits are initialized with the value ~~2^{38} (bit 62 of this 64 bit value is set)~~ 0 at 00:00:00 UTC on January 6, 1980, which is also the reference starting point (the epoch) for GPS-time. In this way every point in time ~~8710~~ 17420 years before and ~~26131~~ 17420 years after January 6, 1980 can be uniquely represented with an accuracy window of 60 ns. Stand-alone systems without external clock synchronization are set to 0 during initialization.

~~unsigned~~ long long TimeDuration: This type is used for durations that are represented in units of 2^{-24} seconds (about 60 ns).

~~unsigned long~~ typedef octet RecordData[4]: This type is used for representing a record and ~~has~~ consists of 4 octets with a valid range from 0 to ~~$2^{32}-1$~~255 each.

~~unsigned long~~ struct AttributesData: This type is used for representing some attributes ~~and has a valid range from 0 to $2^{32}-1$~~. The following subfields are defined:

- octet ERR ~~(Bit 0-3) of the attributes field~~ contains an error-code that is specified for the values from 0 to 12. All other values are reserved for future extensions. (See Section 3.8.1, “Error Codes ERR,” on page 3-11).
- ~~**CONF** (Bit 4-7) represents a confidence marker. (See Section 3.6.2)~~
- octet CONF represents a confidence-marker with a valid range from 0 to 15. (See Section 3.8.2, “Confidence Marker (CONF),” on page 3-12).
- octet PREC ~~(Bit 8-13)~~ represents the number of significant bits in the timestamp. Valid values are in the range from 0 to 63. (See Section 3.8.3, “Time Precision (PREC),” on page 3-12).
- ~~Bit 14-15 are reserved for future extensions.~~
- unsigned short USER ~~(Bit 16-31)~~ are application specific flags and may have a semantics specified by the application. The valid range is from 0 to 65535.

struct ~~DeliveryInstant~~Instants: The first value (subfield ~~Instant~~instant) informs about the next instant when the ~~named data item is delivered by~~ most recent of the ~~real-time transport system to the CORBA gateway~~denoted events will occur. The second value (subfield period) is the period of the named data item.

struct STDevice: This struct is the object-oriented abstraction representing an individual or multiple transducer devices. Its members are DeviceClusterDescriptor (optional), in addition it contains the NameID, and the three object reference pointers to the 3 interfaces on that particular transducer(s) cluster. The aim of this abstraction is to allow easy handling of an object representation of a cluster in programmatic terms. Multiples of these may easily be handled though the use of a number of Current interfaces that is used to track the locus of interaction with any or all of these transducers.

SmartTransducer::Current: This interface is used to semantically acquire and separate the interactions and read-write operations of any single individual transducer be being able to separate them at the programming level though a thread or task locus model as that of the CORBA Current interface. It has one attribute, which is a sequence of devices under the control of that context or thread in general programming terms. Implementations are free to not use a thread abstraction to represent this if so deemed appropriate.

3.8.1 Error Codes ERR

Errors are reported by setting an error-code ERR in the attributes field viz.

- Success (0): no error occurred.
- NoCluster (1): The specified cluster-ID doesn't exist.
- NoNode (2): The specified node-ID doesn't exist in the addressed cluster.
- NoFile (3): The specified file-ID doesn't exist in the node specified by the node-ID.
- NoRecord (4): The specified record-ID is beyond the length of the file.
- Damaged (5): The specified record is damaged and the error can't be corrected.
- DataNotReady (6): The addressed node is not able to determine the correct result within the given time, but is still operating.
- NoExecutable (7): An execute command to a record address that is not executable has been issued.
- ReadOnly (8): Write operation to a record marked as read-only.
- NoMessageReceived (9): No response from the addressed node.
- CommunicationError (10): The received data are not valid.
- TimeError (11): The clocks on the STs are out of sync.
- NotSupported (12): This function is not supported by this ST.

3.8.2 Confidence Marker (CONF)

The four-bit confidence marker CONF is a measurement for the quality of the sensor value, where 0 is defined to be no confidence and 15 is high confidence.

High precision sensors will yield better confidence than low cost sensors. When multiple sensors are used the quality of a measurement may be improved in conjunction with sensor-fusion (e.g., if all sensors provide the same consistent value the confidence-marker can be improved). When differing values are provided the value with best confidence may be preferred.

3.8.3 Time Precision (PREC)

The Precision represents the number of significant bits in the timestamp. This concludes in an error window of $2^{39-PREC}$ seconds. Valid values are from 0 (no precision; the timestamp might be a random value) to 63 (an error window of about 60 nanoseconds). Note that this parameter refers to precision within an ST system, not to the accuracy between the clocks within an ST system and the external time reference.

3.8.4 User Data (USER)

This field has no specific purpose and may be used as required by an application (e.g. additional data where more than 32 bits are required).

3.8.5 XML Description of a RODL

The following is an example XML code that describes RODL-file 7, which forces node 34 (0x22) to write the contents of file 17 (0x11), record 22 (0x16), bytes 0-3 to the communication channel in slot 12 (0x0C) - 15 (0x0F).

```
<?xml version="1.0" ?>
<rodl:rodl name="7"
   xmlns:rodl="http://www.ttpforum.org/2001/ROundDescriptorList"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.ttpforum.org/2001/
      ROundDescriptorList/RODLSchema.xsd">
   <rodl:logical name="34">
      <rodl:slot position="12">
         <rodl:operationCode>read</rodl:operationCode>
         <rodl:fileName>17</rodl:fileName>
         <rodl:recordNumber>22</rodl:recordNumber>
         <rodl:recordAlignment>0</rodl:recordAlignment>
         <rodl:messageLength>4</rodl:messageLength>
         <rodl:valid>true</rodl:valid>
      </rodl:slot>
   </rodl:node>
</rodl:rodl>
```