Manufacturing Message Specification – ISO 9506 (MMS)

Dipl.-Ing. Karlheinz Schwarz[1]
Schwarz Consulting Company (SCC)
Im Eichbaeumle 108
D-76139 Karlsruhe (Germany)
schwarz@scc-online.de
Tel: +49 721 68 48 44
Fax: +49 721 67 93 87

The international standard MMS (Manufacturing Message Specification) is an OSI application layer messaging protocol origionally designed for the remote control and monitoring of devices such as Remote Terminal Units (RTU), Programmable Logic Controllers (PLC), Numerical Controllers (NC), or Robot Controllers (RC). It provides a set of services allowing the remote manipulation of variables, programs, semaphores, events, journals, etc. MMS offers a wide range of services satisfying both simple and complex applications.

MMS is the basis of the international project *Utility Communication Architecture* (UCA™, IEEE TR 1550), IEC 60870-6-TASE.2 (*Inter control center communication*), IEC 61850 (*Communication networks and systems in substations*), and IEC 61400-25 (*Communications for monitoring and control of wind power plants*).

Key words:

[1] About the author: see annex

1 Introduction

The international standard MMS (Manufacturing Message Specification) is an OSI application layer messaging protocol origionally designed for the remote control and monitoring of devices such as Remote Terminal Units (RTU), Programmable Logic Controllers (PLC), Numerical Controllers (NC), or Robot Controllers (RC). It provides a set of services allowing the remote manipulation of variables, programs, semaphores, events, journals, etc. MMS offers a wide range of services satisfying both simple and complex applications.

MMS is the basis of the international project *Utility Communication Architecture* (UCA™, IEEE TR 1550), IEC 60870-6-TASE.2 (*Inter control center communication*), IEC 61850 (*Communication networks and systems in substations*), and IEC 61400-25 (*Communications for monitoring and control of wind power plants*).

For years the automation of technical processes has been marked by increasing requirements with regard to flexible functionalities for the transparent control and visualization of any kind of processes. The mere cyclic data exchange will more and more be replaced by systems that join together independent yet coordinated systems – like communication, processing, open and closed loop control, quality protection, monitoring, configuring and archiving systems – to a whole. These individual systems are interconnected and work together. As a common component, they require a suitable real-time communication system with adequate functions.

The MMS standard defines common functions for distributed automation systems. The expression "manufacturing", which stands for the first M in MMS, has been badly chosen. The MMS standard doesn't contain any manufacturing specific definitions. The application of MMS is as general as the application of a personal computer. MMS offers a platform for a variety of applications.

The first version of MMS documents were published in 1990 by ISO TC 184 (Industrial Automation) as an outcome of the GM initiative *Manufacturing Application Protocols* (MAP). The current version has been published in 2003:

- **Part 1: ISO 9506-1 Services** (2003): describes the services that are provided to remotely manipulate the MMS objects. For each service, a description is given of the parameters carried by the service primitives. The services are described in an abstract way which does not imply any particular implementation.

- **Part 2: ISO 9506-2 Protocol** (2003): specifies the MMS protocol in terms of messages. The messages are described with a notation ASN.1 which gives the syntax.

Today, MMS is being implemented – unlike the practice 15 years ago and unlike the supposition still partly found today – on all common communications networks which support a safe transport of data. These can be networks like TCP/IP or ISO/OSI on Ethernet, a fieldbus or simple point-to-point connections like HDLC, RS 485 or RS 232. MMS is independent of a seven layer stack. Since MMS was originally developed in the MAP environment, it was generally believed earlier that MMS could be used only in connection with MAP.

This paper introduces the basic concepts of MMS applied in the above mentioned standards.

2 What MMS defines

MMS defines the following aspects of the communication between two devices:

- A set of **standard objects** which must exist in every conformant device, on which operations can be executed (examples: read and write local variables, signal events...)
- A set of **standard messages** exchanged between a manager and an agent station for the purpose of controlling these objects
- A set of **encoding rules** for these messages (how values are mapped to bits and bytes)
- A set of **protocols** (rules for exchanging messages between devices)
- MMS does not specify application-specific operations (e.g. change motor speed). This is covered by application-specific, "companion standards", e.g. flexible manufacturing, drives, remote meter reading, ...)
- MMS does not define an API (application program interface)

3 Brief history

All activities with regard to communication standards for automation started in the early 80s:

- ISO OSI **Reference model** ISO 7498
- **Process Bus** in IEC TC 65 and ISA SP 50 (PROWAY A, B, C)
- IEEE 802.4 (**Token Passing**)
- **MAP** (Manufacturing Automation Protocols) in the USA based on IEEE 802.4
- IEEE 802.3 (**Ethernet**) – not accepted before mid of the 90s
- **Telecontrol** protocols IEC 60870 (IEC TC 57)
- **Fieldbus** (Profibus, IEC TC 65, FIP, ...)

Histroy of MAP:

1980 Manufacturing Automation Protocol (MAP) - General Motors

1985 Version MAP 2.1 using OSI Reference model and 10-Mbit/s-Broadband according to IEEE 802 .4 (Token-Bus)

1987 Version MAP 3.0 - MMS (Manufacturing Message Specification ISO/IEC 9506)

1986 Mini-MAP-Specification (3 layers)

1991 MMS for TOP-Specification (Technical and Office Protocol) ,

1991 IEEE 802 .4 with optical physical layer

1991 FDDI als Backbone für MAP und TOP ,

1993 IEEE 802 .3 (Ethernet-LAN) acceptec as optional Data Link

1993 Improved Mini-MAP-Version based on FAIS (Factory Automation Interconnection System)

1986 CNMA (Communications Network for Manufacturing Automation) apply MAP for European market

1990 Siemens defines MMS with private syntax (C-structures) called STF (SINEC Technological Functions)

1995 Siemens defines S7 services (derived from STF and FMS)

2003 Current edition of MMS (ISO 9506:2003)

4 The MMS client/server model

MMS describes the behavior of two communicating devices by the client/server model (see figure 1). The client can for example be an operating and monitoring system, a control center or another intelligent device. The server represents one or several real devices or whole systems. MMS uses an object-oriented modeling method with object classes (Named Variable, Domain, Named Variable List, Journal etc.), instances from the object classes and methods (services like read, write, information report, download, read Journal etc.).

The standard is comprehensive. This does not at all mean that a MMS implementation must be complex or complicated. If only a simple subset is used, then the implementation an also be simple. Meanwhile MMS implementations are available in the third generation. They allow the use of MMS both on PC platforms and embedded controllers.

The MMS server represents the objects which the MMS client can access. The VMD object (Virtual Manufacturing Device) represents the outermost "container" in which all other objects are contained.

Real devices can play both roles (client and server) simultaneously. A server in a control center for its part can be client with respect to a substation. MMS basically describes the behavior of the server. The server contains the MMS objects and it also executes services. MMS can be regarded as "server centric". In principle, in a system more devices are installed which function as server (for example controllers and field devices) than devices which perform the client role (e. g. PC and workstation).

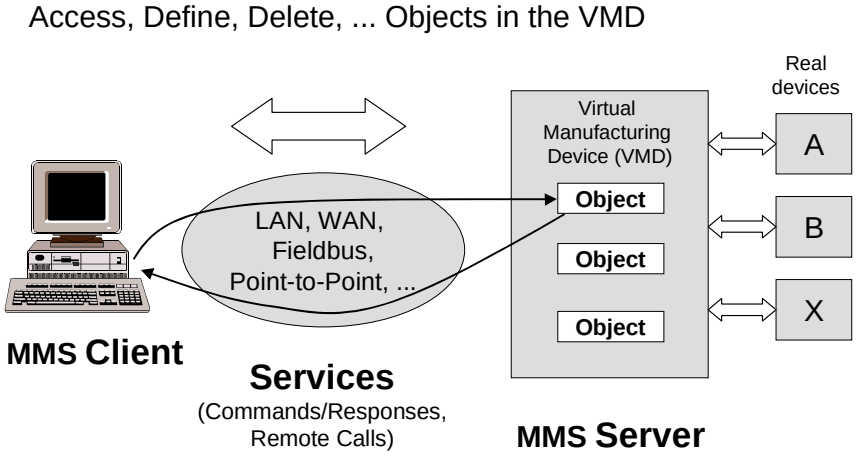

Figure 1 MMS Client/Server Model

The “calls” which the client sends to the server are described in the part ISO 9506-1 (services). These “calls” are processed and answered by the server. The services can also be referred to as remote calls, commands or methods. Using these services, the client can access the objects in the server. It can for example browse through the server, i.e. making visible all available objects and their definitions (configurations). The client can define, delete, change, or access objects via reading and writing.

A MMS server models real data (e. g. temperature measurement, counted measurand or other data of a device). These real data and their implementation are concealed or hidden by the server. MMS doesn't define any implementation details of the servers. It is only defined how the objects behave and represent themselves to the outside (from the point of view of the wire) and how a client can access them.

MMS provides the very common classes. The Named Variable for example allows to structure any information provided for access by an application. The content (the semantic of the exchanged information) of the Named Variables is outside of the MMS standard. Several other standards define common and domain specific information models.

IEC 61850 defines the semantic of many "points" in electric substations. For example: "Atlanta26/XCBR3.Pos.stVal" is the position of the third (3rd) circuit breaker in substation "Atlanta26". The names “XCBR”, “Pos”, and “stVal” are standardized names.

The coming standard IEC 61400-25 (communication for wind power plants) defines a comprehensive list of named "points" specific for wind turbines. Example: "Tower24/WROT.RotSpd.mag" is the (deadbanded) measured value of the rotor position of Tower24. "RotSpd.avgVal" is the average value (calculated based on a configuration attribute "avgPer". These information models are based on common data classes like Measured value, 3-phase value (delta and Y), single point status.

5 The virtual manufacturing device (VMD)

According to figure 2, the real data and devices are represented – in the direction of a client – by the VMD (Virtual Manufacturing Device). In this regard, the server represents a “standard driver” which maps the real world to a virtual one. The following definition helps to clarify the modeling in the form of a virtual device:

If it's there and you can see it	It's REAL
If it's there and you can't see it	It's TRANSPARENT
If it's not there and you can see it	It's VIRTUAL
If it's not there and you can't see it	It's GONE

Roy Wills

The VMD can represent, for example, a variable "Measurement3" whose value may not permanently exist in reality; only when the variable is being read, a measurand transducer will get started to determinate the value. All objects in a server can already be contained in a device before the delivery of a device. The objects are predefined in this case.

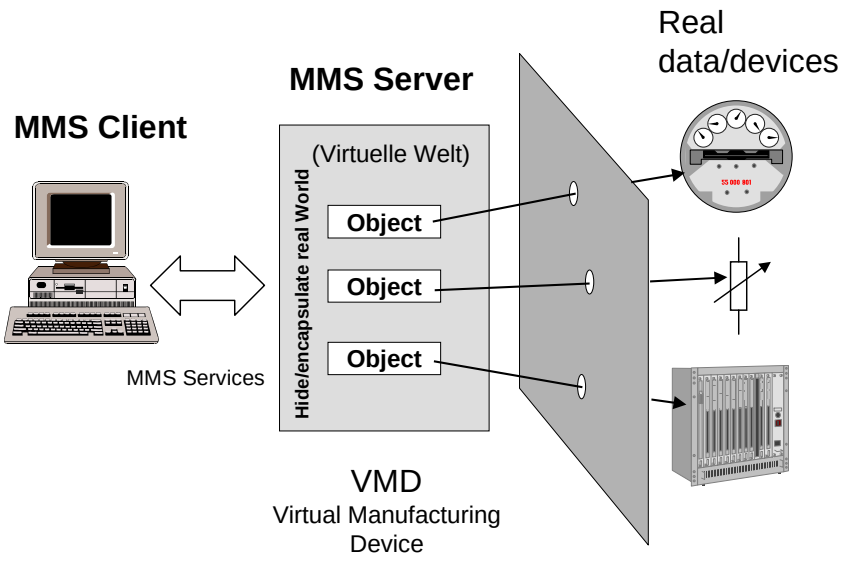

Figure 2 Hiding real devices in the Virtual Manufacturing Device (VMD)

Independent of the implementation of a VMD, data and the access to data are always treated in the same way. This is completely independent of the operating system, the programming language, and memory management. Like printer drivers for a standard operating system hide the various real printers, so a VMD also hides real devices. The server can be understood as a communication driver which hides the specifics of real devices. From the point of view of the client, only the server with its objects and its behavior is visible. The real device isn't visible directly.

MMS merely describes the server side of the communication (objects and services) and the messages which are exchanged between client and server.

The VMD describes a virtual manufacturing device (VMD) completely. This virtual device represents the behavior of a real device as far as it is visible "over the wire". It contains for example an identification of manufacturer, device type and version. The virtual device contains objects like variables, lists, programs and data areas, semaphores, events, journals etc.

The client can read the attributes of the VMD (see figure 4), i.e. it can browse through a device. If the client doesn't have any information about the device, it can view all the objects of the VMD and their attributes by means of the different Get services. With that the client can perform a first plausibility check on a just installed device by means of a Get(Object-Attribute)-Service. It learns whether the installed device is the ordered device with the right model number (Model Name) and the expected issue number (Revision). All other attributes can also be checked (for example variable names and types).

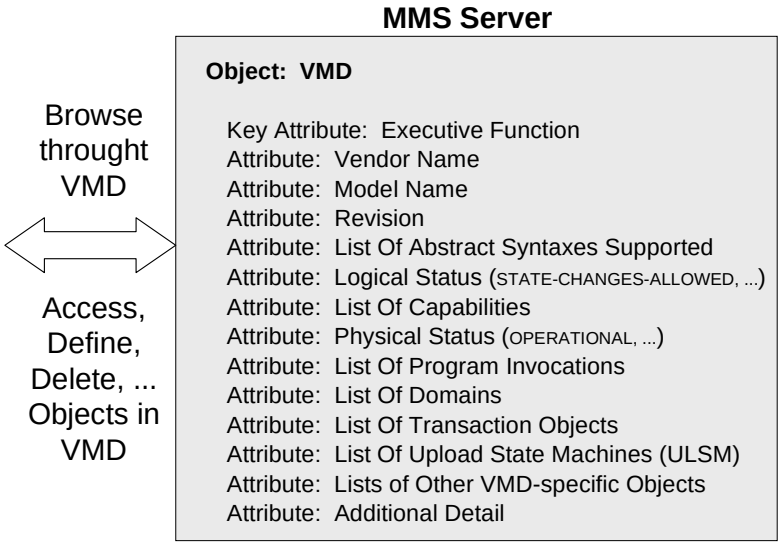

Figure 3 VMD Attributes

The attributes of all objects represent a self-description of the device. Since they are stored in the device itself, a VMD always has the currently valid and thus consistent configuration information of the respective device. This information can be requested online directly from the device. In this way, the client receives always up to date information.

MMS defines some 80 functions:

- browsing functions about the "contents" of the virtual device: "Which objects are available?",
- functions for reading, reporting and writing of arbitrarily structured variable values,
- functions for the transmission of data and programs, for the control of programs and many other functions.

The individual groups of the MMS services and objects are shown in figure 4. MMS describes such aspects of the real device that shall be open i.e. standardized. An open device must behave as described by the virtual device. How this behavior is achieved is not visible and also not relevant to the user that accesses the device externally. MMS doesn't define any local, specific interfaces in the real systems. The interfaces are independent of the functions which shall be used remotely. Interfaces in connection with MMS are always understood in the sense that MMS quasi represents an interface between the devices and not within the devices. This interface could be described as an external interface. Of course, interfaces are also needed for implementations of MMS functions in the individual real devices. These shall not and cannot be defined by a single standard. They are basically dependent on the real systems – and these vary to a great extent.

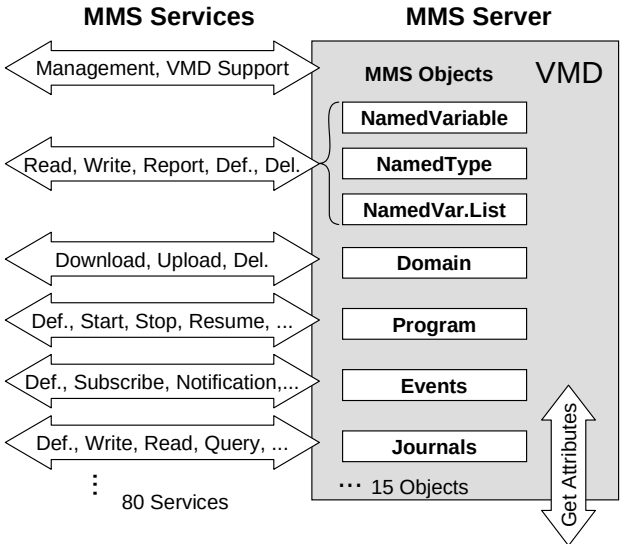

Figure 4 MMS Objects and Services

MMS models and services:

ISO 9506-1 (part 1) – Service Specification

Environment and General Management Services

Two applications that want to communicate with each other can set up, maintain and close a logical connection (initiate, conclude, abort).

VMD Support

The client can thereby query the status of a Virtual Manufacturing Device (VMD) or the status is reported (Unsolicited Status); the client can query the different lists of the objects (Get Name List), query the attributes of the VMD (Identify) or change the names of objects (Rename).

Domain Management

Using a simple flow control (Download, Upload, Delete Domains, ...), programs and data of arbitrary length can be transmitted between client and server and also a third station (and vice versa). In the case of simple devices, the receiver of the data stream determines the speed of the transmission.

Program Invocation Management

Services to create, start, stop and delete modularly structured programs by the client (Start, Stop, Resume, Kill, Delete, ...).

Variable Access

This service allows the client to read and write variables that are defined in the server or a server is enabled to report the contents to a client without being requested (Information Report). The structures of these data are simple (octet string) to arbitrarily complex (Structure of Array of ...). In addition, data types and arbitrary variables can be defined (Read, Write, Information Report, Define Variable, ...). The variables consitute the core functionality of every MMS application; therefore the variable access model will be explained in detail below.

Event Management

It allows an event-driven operation; i.e. a given service (e. g. Read) is only carried out if a given event has occurred in the server. An alarm strategy is integrated. Alarms will be reported to one or more clients if certain events occur. These have the possibility to acknowledge the alarms later (Define, Alter Event Condition Monitoring, Get Alarm Summary, Event Notification, Acknowledge Event Notification, ...). This model isn't explained further.

Semaphore Management

The synchronization of several clients and the coordinated access to the resources of real devices is carried out hereby (Define Semaphore, Take/Relinquish Control, ...). This model isn't explained further.

Operator Communication

Simple services for the communication with operating consoles integrated in the VMD (Input and Output). This model isn't explained further.

Journal Management

Several clients can enter data into journals (archives, logbooks) which are defined in the server. Then these data can selectively be retrieved through filters (Write Journal, Read Journal, ...). This model isn't explained further.

ISO 9506-2 (Part 2) – Protocol Specification

If a client invokes a service, then the server must be informed about the requested type of the service. For a Read service, e. g. the name of the variables must be sent to the server. This information which the server needs for the execution is exchanged in so-called Protocol Data Units (PDU). The set of all the PDU that can be exchanged between client and server constitute the MMS protocol.

In other words, the protocol specification – using the standards ISO 8824 (Abstract Syntax Notation One, ASN.1) and 8825 (ASN.1 Basic Encoding Rules, BER) – describes the abstract and concrete syntax of the functions defined in part 1. The syntax is explained below exemplarily.

6 Locality of the VMD

VMDs are virtual descriptions of real data and devices (e. g. protection devices, measurand transducers, wind turbine, and any other automation device or system). Regarding the implementation of a VMD, there are three very different possibilities where a VMD can be located (see figure 5):

1. In the end device: One or several VMD are in the real device which is represented by the VMD. The implementations of the VMD have direct access to the data in the device. The modeling can be carried out in such a way that each application field in the device is assigned to its own VMD. The individual VMDs are independent from each other.

2. In the gateway: One or several VMD are implemented in a separate computer (a so-called gateway or agent). In this case, all MMS objects that describe the access to real data in the devices are at a central location. While being accessed, the data of a VMD can be in the memory of the gateway – or they must be retrieved from the end-device only after the request. The modeling can be carried out in such a way that for each device or application a VMD of its own will be implemented. The VMD are independent from each other.

3. In a file: One or several VMD are implemented in a data base on a computer, on a FTP server or on a CD ROM (the possibilities under 1. and 2. are also valid here). Thus, all VMD and all included objects with all their configuration information can be entered directly into engineering systems. Such a CD ROM, which represents the device description, also could be used for example to provide a monitoring system with the configuration information: names, data types, object attributes etc. Before devices are delivered, the engineering tools can already process the accompanying device configuration information (Electronic Data Sheet)! The configuration information can also be read later online from the respective VMD via corresponding MMS requests.

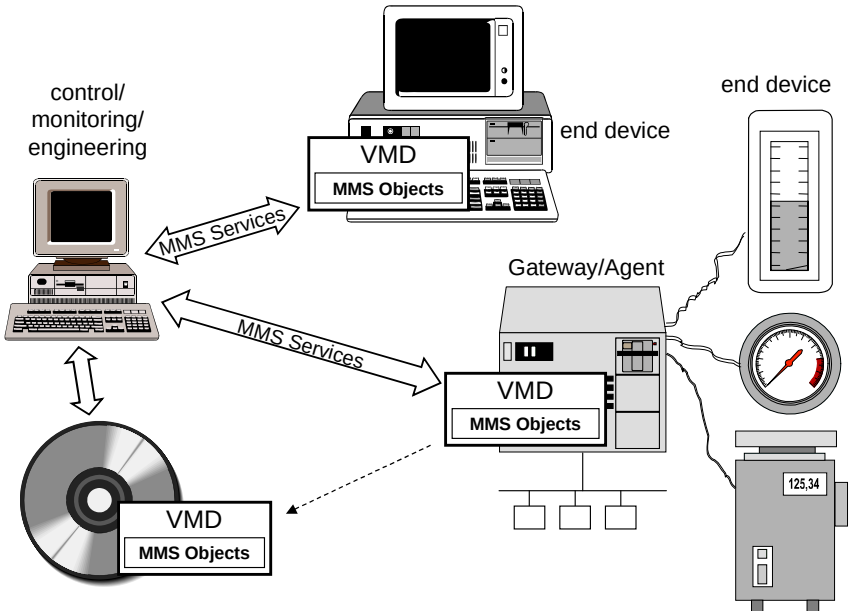

Figure 5 Location of VMDs

The VMD is independent of the location. This also allows for example – besides the support during configuration – that several VMD can be installed for testing purposes on another

computer than the final system (see figure 6). Thus, the VMD of several large robots can be tested in the laboratory or office. The VMD will be installed on one or several computers (the computers emulate the real robots). Using a suitable communication (for example intranet or also a simple RS 232 connection – available on every PC), the original client (a control system which controls and supervises the robots) can now access and test the VMD in the laboratory. This way, whole systems can be tested beforehand regarding the interaction of individual devices (for example monitoring and control system).

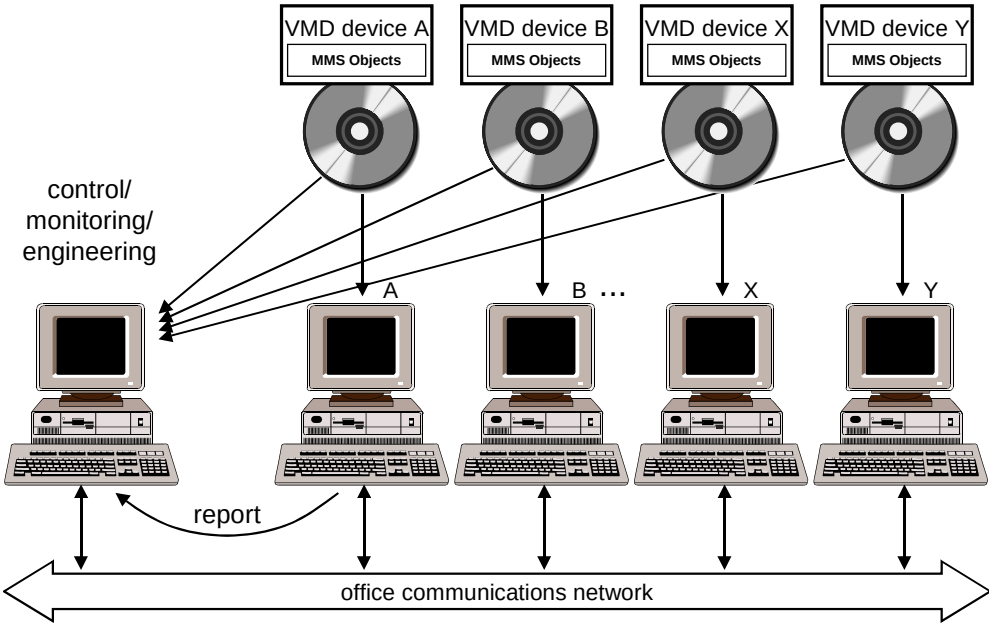

Figure 6 VMD testing using PC in an office environment

If the internet is used instead of the intranet, global access is possible to any VMD which is connected to the internet. The author himself tested the access from Germany to a VMD which was implemented on a PLC in the USA. That is to say, through standards like MMS and open transmission systems it has become possible to set up global communications networks for the real-time process data exchange.

The previous statements about the VMD are valid in full extent also for all standards that are based on MMS.

7 Interfaces

The increasing distribution of automation applications and the exploding amount of information require more and more and increasingly more complex interfaces for operation and monitoring. Complex interfaces turn into complicated interfaces very fast. Interfaces “cut” components in two pieces; through this, interactions between the emerged sub-components – which were hidden in one component before – become visible. An interface discloses which functions are carried out in the individual sub-components and how they act in combination.

Transmitter and receiver of information must likewise be able to understand these definitions. The request "Read T142" must be formulated "understandably", transmitted correctly and understood unambiguously (see figure 7). The semantic (named terms that represent something) of the services and the service parameters are defined in MMS. The content, e.g., of Named Variables is defined in domain-specific standards like IE 61850.

Interfaces occur in two forms:

- internal program-program interfaces or APIs and
- external interfaces over a network (WAN, LAN, fieldbus, ...).

Both interfaces affect each other. MMS defines an external interface. The necessity of complex interfaces (complex because of the necessary functionality, not because of an end in itself) is generally known and accepted. To keep the number of complex interfaces as small as possible, they are defined in standards or industry standards – mostly as open interfaces. Open interfaces are in the meantime integral component of every modern automation. In Mid 1997 it was explained in [22] that the trend in automation engineering obviously leads away from the proprietary solutions to open, standardized interfaces – i.e. to open systems.

Figure 7 Sender and receiver of information

The reason why open interfaces are complicated is not because they were standardized. Proprietary interfaces tend more towards being complicated or even very complicated. The major reasons for the latter observation are found in the permanent "improvement" of the interfaces which expresses itself in the quick changes of version and in the permanent development of new – apparently better – interfaces. Automation systems of one manufacturer often offer – for identical functions – a variety of complicated interfaces which are incompatible to each other.

At first, interfaces can be divided up into two classes (see figure 8): internal interfaces (for example in a computer) and external interfaces (over a communication network). The following consideration is strongly simplified because e. g. in reality both internally and externally several interfaces can lie one above the other. However, it nevertheless shows the differences in principle which must be paid attention to. MMS defines an external interface.

Many understand MMS in such a way that it offers – or at least also offers – an internal interface. This notion results in completely false ideas. Therefore, the following consideration is very helpful.

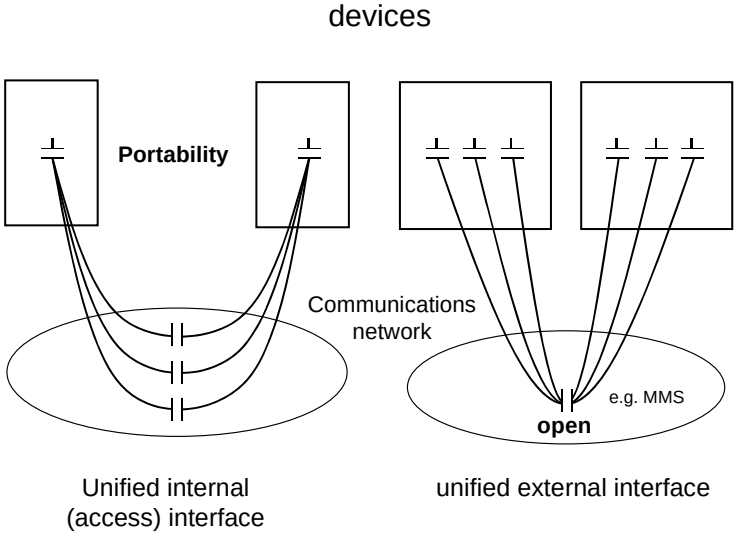

Figure 8 Internal and external interfaces

The left hand side of the figure shows the case with an uniform internal interface and varying external interfaces. This uniform internal interface allows many applications to access the same functions with the same parameters and perhaps the same programming language – independent of the external interface. Uniform internal interfaces basically allow the portability of the application programs over different external interfaces.

The right hand side of the figure shows the case with the external interface being uniform. The internal interfaces are various (since the programming languages or the operating systems for example are various). The uniform external interface is independent of the internal interface. The consequence of this is for example that devices whose local interfaces differ and are implemented in different environments can communicate together. Differences can result, for example, from an interface being integrated into the application in a certain device, but being available explicitly in another device. The essential feature of this uniform external interface is the interoperability of different devices. The ISO/OSI reference model is aimed at exactly this feature.

The (internal) MMS interface, for example in a client (perhaps: $READ (Par. 1, Par. 2, ... Par. N)), depends on manufacturer, operating system and programming language. MMS implementations are for example available for UNIX or Windows NT. On the one hand, this is a disadvantage because applications which want to access a MMS server would have to support – depending on environment – various real program interfaces. On the other hand, the MMS protocol is completely independent of the – fast changing – operating system platforms. Standardized external interfaces like MMS offer a high degree on stability, because in the first place the communication can hardly be changed arbitrarily by a manufacturer and in the second place several design cycles of devices can survive.

Precisely the stability of the communication, as it is defined in MMS, also offers a stable basis for the development of internal interfaces on the various platforms such as under Windows 95, NT or in UNIX environments.

Openness describes in the ISO/OSI world the interface on the "wire". The protocol of this external interface executes according to defined standardized rules. For an interaction of two components these rules have to be taken into account on the two sides; otherwise the two will not understand each other.

8 Environment and General Management Services

MMS uses a connection-oriented mode of operation. That is to say, before e. g. a computer can read a value from a PLC for the first time, a connection must be set up between the two.

MMS connections have particular quality features such as:

- Exclusive allocation of computer and memory resources to a connection. This is necessary to guarantee that all services (for example five Read, ...) that are allowed to be carried out simultaneously find sufficient resources on both sides of the connection,
- Flow control in order to avoid blockages and vain transmissions, if e. g. the receive buffers are full,
- Segmentation of long messages,
- Routing of messages over different networks,
- Supervision of the connection if no communication takes place,
- Acknowledgment of the transmitted data,
- Authentication, access protection (password) and encoding of the messages.

Connections are generally established once and then remain established as long as a device is connected (at least during permanently necessary communication). If, for example, a device is only seldom accessed by a diagnostics system, a connection then doesn't need to be established permanently (waste of resources). It suffices to establish a connection and, later, to close it to release the needed resources again. The connection can remain established for rare but time-critical transmissions. The subordinate layers supervise the connection permanently. Through this the interruption of a connection is quickly recognized.

The MMS services for the connection management are:

- **initiate** (connection set-up).
- **conclude** (orderly connection tear-down, waiting requests are still being answered).
- **abort** (abrupt connection tear-down, waiting requests are deleted).

Besides these services that are all mapped to the subordinate layers, there are two further services:

- **cancel** and
- **reject**.

After the MMS client has sent a Read request to the MMS server, for example, it may happen that the server leaves the service in its request queue and – for whatever reason – does not process it. Using the Service Cancel, the client can now delete the request in the server. On the other hand, it may occur that the server shall carry out a service with "forbidden" parameters. Using Reject it rejects the faulty request and reports this back to the client.

Although MMS was originally developed for ISO/OSI networks a number of implementations are available in the meantime that also use other networks such as the known TCP/IP network. From the point of view of MMS this is insignificant as long as the necessary quality of the connection is guaranteed.

9 VMD Support

The VMD object consists of 12 attributes. The Key Attribute identifies the "Executive Function". The Executive function corresponds directly with the entity of a VMD. A VMD is identified by a presentation address):

Object: VMD
- Key Attribute: Executive Function
- Attribute: Vendor Name
- Attribute: Model Name
- Attribute: Revision
- Attribute: Logical Status (STATE-CHANGES-ALLOWED, NO-STATE- CHANGES-ALLOWED, LIMITED-SERVICES-SUPPORTED)
- Attribute: List of Capabilities
- Attribute: Physical Status (OPERATIONAL, PARTIALLY-OPERATIONAL, INOPERABLE NEEDS-COMMISSIONING)
- Attribute: List of Program Invocations
- Attribute: List of Domains
- Attribute: List of Transaction Objects
- Attribute: List of Upload State Machines (ULSM)
- Attribute: List of Other VMD-specific Objects

The attributes Vendor Name, Model Name and Revision inform about the manufacturer and the device.

The Logical status defines, which services may be carried out. The status Limited-Services-Supported allows that only such services may be executed which have read access to the VMD.

The Physical Status indicates whether or not the device works in principle.

The two following services:

- **Unsolicited Status** and
- **Status**

are used to get the status unsolicited (Unsolicited status) or explicitly requested (Status). Thus, a client can recognize whether a given server – from the point of view of the communication – works at all.

The List of Capabilities offers clients and servers a possibility to define application-specific agreements in the form of features. The available memory of a device, for example, could be a capability. Through the Service Get Capability List the current value can be queried. The remaining attributes contain the lists of all the MMS objects available in a VMD. The VMD contains therefore an Object Dictionary in which all objects of a VMD are recorded.

The following three services complete the services of the VMD:

- **Identify** supplies the VMD attributes Vendor Name, Model Name and Revision. With that, a plausibility check can be carried out from the side of the client.
- **Get Name List** returns the names of all MMS objects. It can be selectively determined from which classes of objects (for example Named Variable or Event Condition) the names of the stored objects shall be queried. Let's assume a VMD is not yet known to the client till now (because it is, for example, a maintenance device), the client can then browse through the VMD and systematically query all names of the objects. Using the Get services which are defined for every object class (e. g. Get Variable Access Attributes), the client can get detailed knowledge about a given object (for example the Named Variable "T142").
- **Rename** allows a client to rename the name of an object.

10 Domain Management

Domains are to be viewed as containers which represent memory areas. Domain contents can be interchanged between different devices. The object type "domain" with its 12 attributes and 12 direct operations, which create, manipulate, ... delete a domain, are part of the model.

The abstract structure of the domain object consists of the following attributes:

Object: Domain
- Key Attribute: Domain Name
- Attribute: List of Capabilities
- Attribute: State (LOADING, COMPLETE, INCOMPLETE, READY, IN-USE)
- Constraint: State = (LOADING, COMPLETE, INCOMPLETE)
  - Attribute: Assigned Application Association
  - Attribute: MMS Deletable
  - Attribute: Sharable (TRUE, FALSE)
  - Attribute: Domain Content
  - Attribute : List of Subordinate Objects
- Constraint: State = (IN-USE)
  - Attribute: List of Program Invocation References
  - Attribute: Upload In Progress
  - Attribute: Additional Detail

The domain name is an identifier of a domain within a VMD.

Domain Content is a dummy for the information which is within a domain. The contents of the data to be transmitted can be coded transparently or according to certain rules agreed upon before. Using the MMS version (2003), the data stream can be coded per default in such a way that a VMD can be transmitted completely – including all MMS object definitions which it contains. This means on the one hand that the contents of a VMD can be loaded from a configuration tool into a device (or saved from a device), and on the other hand that the contents can be stored on a disk per default.

Using a visible string, List of Capabilities describes which resources are to be provided – by the real device – for the domain of a VMD.

MMS Deletable indicates whether or not this domain may be deleted by means of a MMS operation.

Sharable indicates whether or not a domain may be used by more than one program invocation.

List of Program Invocation lists those Program Invocation Objects that use this domain.

List of Subordinate Objects lists those MMS objects (no domains or program invocations) which are defined within this domain: objects which were (a) created by the domain loading, (b) created dynamically by a Program Invocation, (c) created dynamically by MMS operations or (d) created locally.

State describes one of the ten states in which a domain can be.

Upload in Progress indicates whether or not the Domain Content of this domain is being copied to the client at the moment.

MMS defines loading in two directions:

- data transmission from the client to the server (download) and

- data transmission from the server to the client (Upload).

Three phases can be distinguished during loading:

- open transmission,
- segmented transmission, controlled by the data sink, and
- close transmission.

Transmission during download and upload is initiated by the client respectively. If the server initiates, then it has the possibility to initiate the transmission indirectly (see figure 9). For this purpose, the server informs the client that it (the client) shall initiate the loading. Even a third station can initiate the transmission by informing the server which then informs the client.

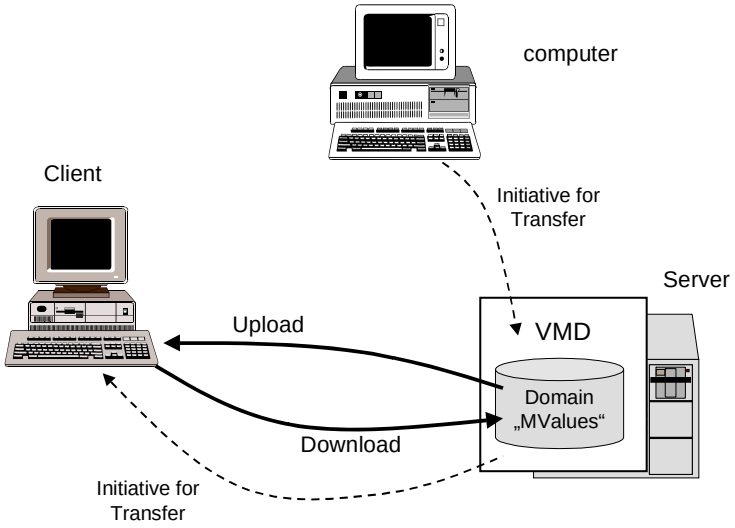

Figure 9 MMS domain transfer

What is the domain scope?

Further MMS objects can be defined within a domain: variable objects, event objects and semaphore objects. That is to say a domain forms a scope (validity range) in which named MMS objects are reversibly unambiguous.

MMS objects can be defined in three different scopes, as shown in figure 10. Objects with VMD specific scope (for example the variable "Status_125") can be addressed directly through this name by all clients. If an object has domain specific scope such as the object "Status_155", then it is identified by two identifiers:

Domain Identifier "Motor_2" and Object Identifier "Status_155".

A third scope is defined by the Application Association. The object "Status_277" is part of the corresponding connection. This object can only be accessed through this connection. When the connection is closed, all objects are deleted in this scope.

MMS objects can be organized using the different scopes. The object names (with or without domain scope) can be compounded from the following character set:

The identifiers can contain 1 to 32 characters and they must not start with a number.

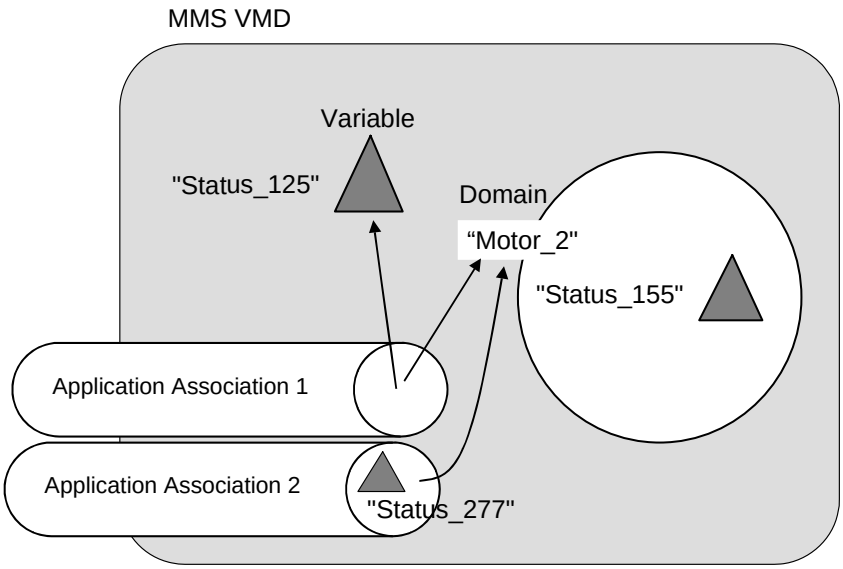

Figure 10 VMD- and Domain-Scope

The object names can be structured by agreement in a further standard or other specification. Many standards which reference MMS make much use of this possibility. This way, all named variables with the Prefix "RWE_" and similar prefixes, for example, could describe the membership of the data (in a trans-European information network) to a specific utility of an interconnected operation.

11 Program-Invocation-Management

A Program Invocation Object is a dynamic element which corresponds with the program executions in multi-tasking environments. Program Invocations are created by linking several domains. They are either predefined or created dynamically by MMS services or created locally.

A Program Invocation Object is defined by its name, its status (Idle, Starting, Running, Stopping, Stopped, Resuming, Unrunnable), the list of the domains to be used and nine operations.

Object: Program Invocation
- Key Attribute: Program Invocation Name
- Attribute: State (IDLE, STARTING, RUNNING, STOPPING, STOPPED, RESUMING, RESETTING, UNRUNNABLE)
- Attribute: List of Domain References
- Attribute: MMS Deletable(TRUE, FALSE)
- Attribute: Reusable (TRUE, FALSE)
- Attribute: Monitor (TRUE, FALSE)
- Constraint: Monitor = TRUE
  - Attribute: Event Condition Reference
  - Attribute: Event Action Reference
  - Attribute: Event Enrollment Reference
  - Attribute: Execution Argument
  - Attribute: Additional Detail

Program Invocations are structured flatly – though several Program Invocations can reference the same domains (Shared Domains). The contents of the individual domains are absolutely transparent both from the point of view of the domain and from the point of view of the Program Invocations. What is semantically connected with the Program Invocations is outside the scope of MMS. The user of the MMS objects must therefore define the contents; the semantics result from this context. If a Program Invocation connects two domains, then the domain contents must define what these domains shall do together – MMS actually only provides a wrapper.

The Program Invocation Name is a clear identifier of a Program Invocation within a VMD.

State describes the status in which a Program Invocation can be. Altogether seven states are defined.

List of Domains contains the names of the domains which are combined to a Program Invocation. This list also includes such domains which are created by the Program Invocation itself (this can be a domain into which some output is written).

MMS Deletable indicates whether or not this Program Invocation may be deleted by means of a MMS operation.

Reusable indicates whether or not a Program Invocation can be started again after the program execution. If it cannot be started again, then the program Invocation can only be deleted.

Monitor indicates whether or not the Program Invocation reports a transition to the client when exiting the Running status.

Start Argument contains an application specific character string which was transferred to a Program Invocation during the last start operation; this string e. g. could indicate which function has started the program last.

Additional Detail allows the companion standards to make application-specific definitions.

Program Invocation Services

Create Program Invocations: This service arranges an operational program, which consists of the indicated domains, in the server. After installation, the Program Invocation is in the status Idle from where it can be started. The monitor and the monitor type indicate, whether or not and how the program invocation shall be monitored.

Delete Program Invocation: Deletable Program Invocations are deleted through this service. Primarily, that is to say that the resources bound to a Program Invocation are released again.

Start: The start service causes the server to transfer the specified Program Invocation from the Idle into the Running state. Further information can be transferred to the VMD through a character string in the start argument. A further parameter (Start Detail) contains once again additional information which can be defined by companion standards.

Stop: The stop service changes a specified Program Invocation from the Running into the Stopped state.

Resume: The Resume service changes a specified Program Invocation from the Stopped into the Running state.

Reset: The Reset service changes a specified Program Invocation from the Running or Stopped into the Idle state.

Kill: The Kill service changes a specified Program Invocation from arbitrary states into the Unrunnable state.

Get Program Invocation Attribute: Through this service the client can read all attributes of a certain Program Invocation.

12 The MMS variable model

MMS Named Variables are addressed using identifiers made up of the domain name and the Named Variable name within the domain. Components of an MMS Named Variable may also be individually addressed using a scheme called Alternate Access. The Alternate Access address of a component consists of the domain name and the Named Variable name, along with a sequence of enclosing component names of the path down to the target component.

The Variable Access Services contain an extensive variable model which offers the user a variety of advanced services for the description and offers the access to arbitrary data of a distributed system.

A wide variety of process data are processed by automation systems. The data, their definition and representation are usually orientated at the technological requirements and at the available automation equipment. The methods the components employ for the representation of their data and the access to them correspond to the way of thinking of their imple-

menters. This has resulted in a wide variety of data representations and access procedures for one and the same technological datum in different components. If e. g. a certain temperature measurement shall be accessed to in different devices, then a huge quantity of internal details must generally be taken into account for every device (request, parameter, coding of the data, ...).

As shown in figure 11, the number of the protocols for the access of a client (on the left in the figure) to the data from n servers (S1 – Sn) can be reduced to a single protocol (on the right in the figure). Through this, the data rate required for the communication primarily in central devices can be reduced drastically.

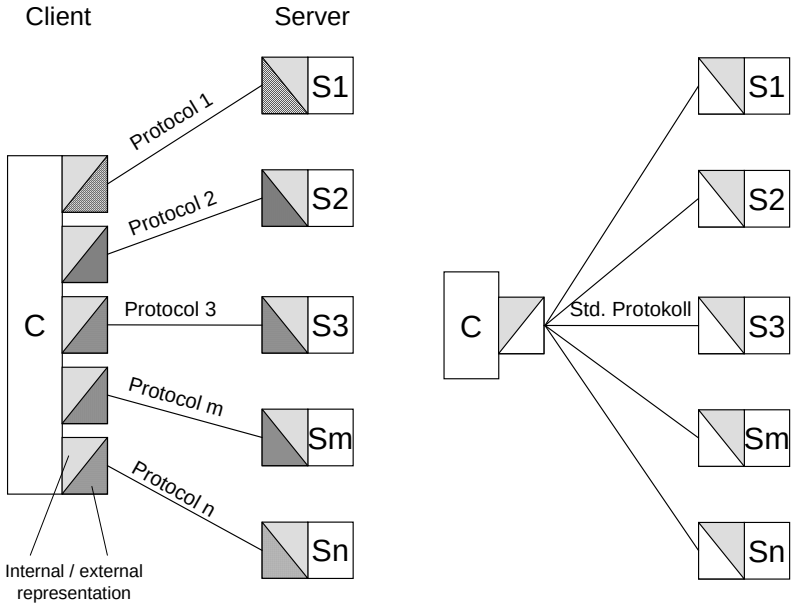

Figure 11 Unified Protocols

In programs variables are declared, i.e. they get a name, a type and a value. Described in a simplified way, both the name and the type are converted by the compiler into a memory location and into a reference which is only accessible to the compiled program. Without any further measures, the data of the variable aren't identifiable outside the program. It is concealed for the user of the program how a compiler carries out the translation into the representation of a certain real machine.

The data are stored in different ways depending on the processor; i.e. primarily that the data are stored in various memory locations. During the run-time of the program only this representation is available.

These data are not visible from the outside. They must first be made visible for the access from the outside. To enable this, an entity must be provided in the implementation of the application. It is insignificant here whether or not this entity is separated from or integrated into the program. This entity is acting for all data which shall be accessible from the outside.

This consideration is helpful to the explanation of the MMS variable model: What do protocols through which process data are accessed have in common? Figure 12 shows the characteristics in principle. On the right is the memory with the real process data which shall be

read. The client must be able to identify the data to be read (gray shade). For this purpose, pointer (start address) and length of the data must be known. By means of this information the data can be identified in the memory.

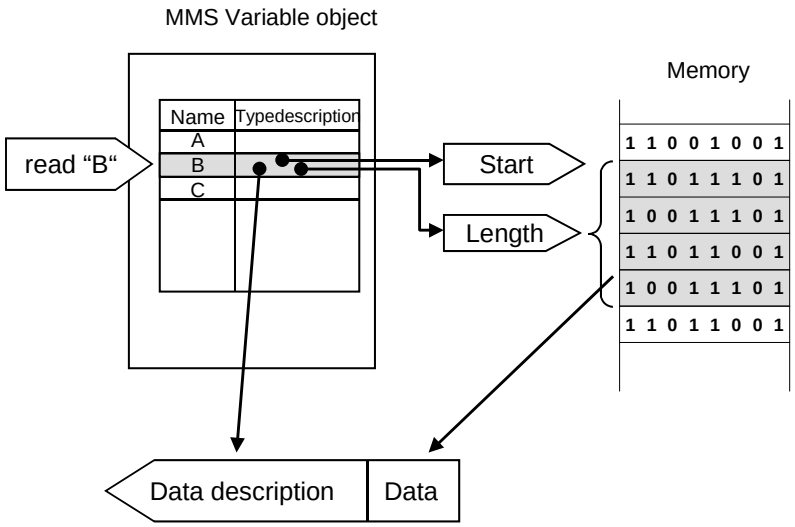

Figure 12 Data access principle

Yet how can a client know the pointer of the data and what the length of the data is? It could have this information somehow and indicate it when reading. Yet if the data should move some time, then the pointer of the data is not correct any more. There can also be the case that the data do not exist at all at the time of reading but must be calculated first. In this case there isn't any pointer. To avoid this, references to the data, which are mapped to the actual pointers (table or algorithm), are used in most cases. In our case, the reference "B" is mapped by table to the corresponding pointer and the length.

The pointer and the length are stored in the type description of the table. The pointer is a system-specific value which generally isn't visible on the outside. The length is dependent on the internal representation of the memory and on the type. An individual bit can for example be stored as an individual bit in the memory or also as a whole octet. However, this is not relevant from the point of view of the communication.

The data themselves and their description are important for the message response of the read service. The question of the external representation (is for example an individual bit encoded as a bit or as an octet?) is – unlike the internal representation – of special importance here! The various receivers of the data must be able to interpret the data unambiguously. For this purpose they need the representation which is a substantial component of the MMS variable model. The data description is therefore derived from the type description.

For the deeper understanding of the variable model three aspects have to be explained more exactly:

- objects and their functions,

- services (read, write, ...) which access the MMS variable objects,
- data description for the transmission of the data.

The object model of a MMS variable object is conceptually different from a variable according to a programming language. The MMS objects describe the access path to structured data. In this sense, they do not have any variable value.

Access paths

The access path represents an essential feature of the MMS variable model. Starting from a more complex hierarchical structure we will consider the concept. The abstract and extremely simplified example is deliberately chosen. Here, we are merely concerned about the principle.

Certain data of a machine shall be modeled using MMS methods. The machine has a tool magazine with n similar tools. A tool is represented – according to figure 13 – by three components (tool type, number of the blades and remaining use time).

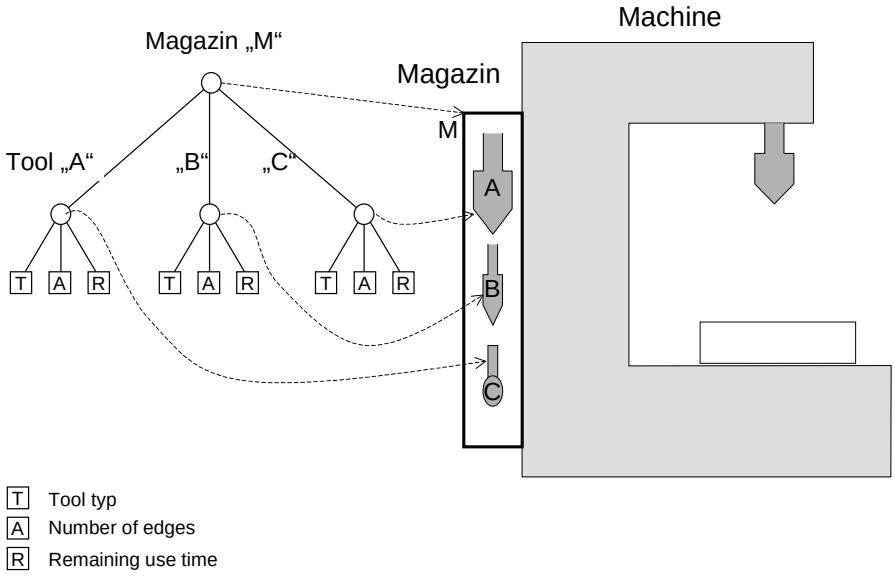

Figure 13 Data of a Machine

The machine with its tool magazine "M" is outlined in the figure on the right. The magazine contains three tools "A", "B" and "C". The appropriate data structure of the magazine is shown on the left. The structure is tree-like; the root "M" is drawn as the topmost small circle (node). "M" has three components (branches) "A", "B" and "C" which are also represented as circles. These components in turn also have three components (branches). In this case, the branches end in a leaf (represented in the form of a square). Leaves represent the end-points of the branches. For each tool, three leaves are shown: "T" (tool type), "A" (number of blades) and "R" (remaining use time). The leaves represent the real data. The nodes ordered hierarchically are merely introduced for reasons of clustering. Leaves can occur at all nodes. A leaf with the information "magazine full/not full" could for example be attached at the topmost node.

With this structure, the MMS features are explained more detailed. The most essential aspect is the definition of the access path. The access to the data (and their use) can be carried out according to various task definitions:

1. selecting all leaves for reading, writing ...,

2. selecting certain leaves for reading, writing ... and

3. selecting all leaves as component of a higher-level structure, for example "machine" with the components magazine "M" and drilling machine,

4. selecting certain leaves as component of a higher-level structure.

Examples of the cases 1. and 2. are shown in figure 14. The case that the complete structure is read is shown in the left top corner (the selected nodes, branches and leaves are represented in bold lines or squares). All nine data are transmitted as response (3 x (T+A+R)). At first, the representation during transmission is deliberately refrained from here and also in the following examples.

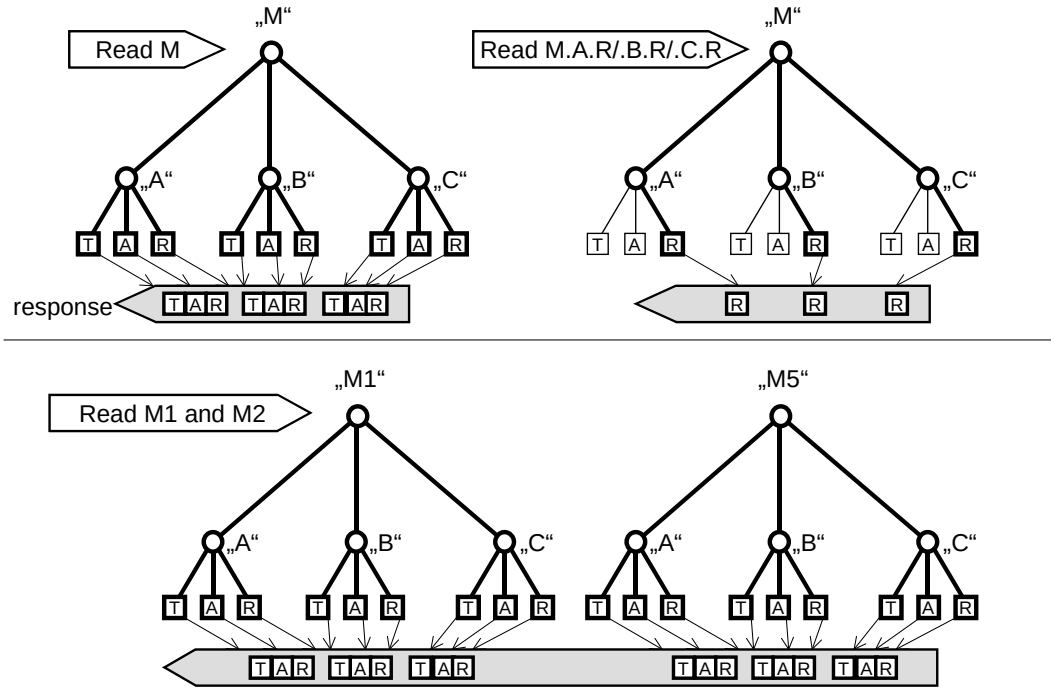

Figure 14 Access and partial access

Only a part of the data is read in the top right corner of the figure: Respectively only the leaf "R" of all three components "A", "B" and "C". The notation for the description of the subset M.A.R/.B.R/.C.R is chosen arbitrarily.

The subset M.A.R represents a (access) path which leads from a root to a leaf. It can also be said that one or several paths represent a part of a tree. The read message contains three paths which must be described in the request completely. A path, for example, can also end at "A". All three components of "A" will be transmitted in this case.

Besides the possibility to describe every conceivable subset, MMS also supports the possibility to read several objects "M1" and "M5" in a read request simultaneously (see lower half of figure). Of course, every object can only be partly read too (which is, however, not represented).

An example of case 4 is shown in figure 15 (case 4 has to be understood as the generalization of case 3). Here a new structure was defined using two sub-structures. The object "machine" contains only the "R" component of all six tools of the two magazines "M1" and "M5". The object "machine" shall not be mixed up with a list (Named Variable List). Read "machine" supplies the six individual "R" values.

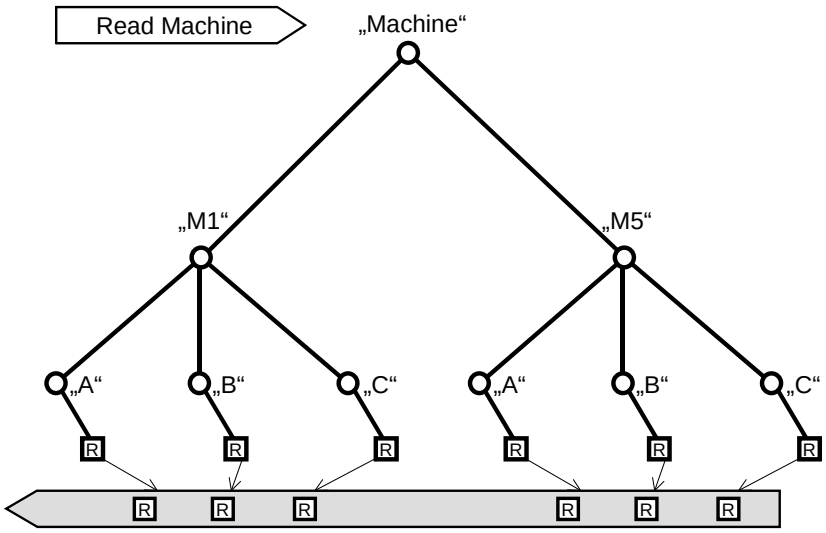

Figure 15 “Partial” trees

The component names, such as "A", "B" or "C", need to be unambiguous only below a node – and only at the next lower level. Thus, "R" can always stand for remaining life time. The position in the tree indicates the remaining life time of a particular tool. The new structure "machine" has all features which were described in the previous examples also for "M1" and "M5".

These features of the MMS variable objects can be applied to (1) the definition of new variable objects and (2) the access to existing variable objects. The second case is also interesting. As shown in figure 16, the same result as in figure 15 can be reached by enclosing the description (only the "R" components of the tools shall be read from the two objects "M1" and "M5") in the read request every time. The results (read answer) are absolutely identical in the two cases.

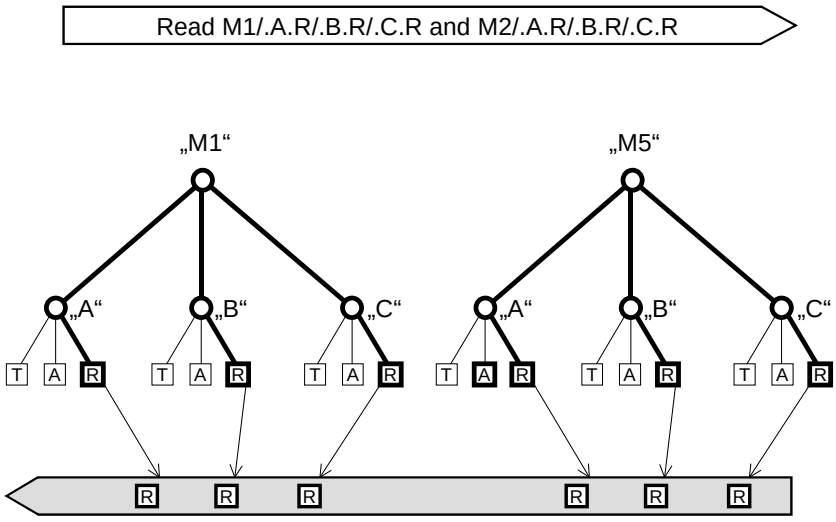

Figure 16 “Partial” trees used for read requests

Every possibility, to assemble hierarchies from other hierarchies (1), or to e. g. read parts of a tree during the access (2), has its useful application. The first case is important in order to avoid enclosing the complete path description every time when reading an extensive part of tree. The second case offers the possibility to construct complex structures based on standardized basic structures (for example the structure of "tool data" consisting of the components "T", "A" and "R") and to use them for the definition of new objects.

Summarizing, it can be stated that the access paths accomplish two tasks:

- description of a subset of nodes, branches and leaves of objects during reading, writing, ... and
- description of a subset of nodes, branches and leaves of objects during the definition of new objects.

In conclusion, this may again be expressed in the following way: Path descriptions describe the "way" or the "ways" to a single data (leaf) or to several data (leaves). A client can read the structure description (complete tree) through the MMS service Get Variable Access Attributes.

Another aspect is of special importance too. Till now, we have not considered the description of leaves. Every leaf has one of the following MMS basic data types:

- BOOLEAN,
- BIT,
- INTEGER,
- UNSIGNED,
- FLOATING POINT,
- REAL,
- OCTET STRING,
- VISIBLE,
- GENERALIZED TIME,
- BINARY,
- BCD.

Every node in a tree is either an array or a structure. Arrays have n elements (0 to n-1) of the same type (this can be a basic type, an array or a structure). When describing a part of a tree, any number of array elements can be selected (e. g. one element, two adjacent elements, two arbitrary elements, ...).

Structures consist of one or several components. Every component can be marked by its position and if necessary by an "Identifier" – the component name. This component name (e. g. "A") is used for the access to a component. Parts of trees can describe every subset of the structure.

The path description contains the following three elements:

- All of the possibilities of the description of the structures (individual and composite paths in the type description of the MMS variables) are defined in the form of an extensive abstract syntax.

- For every leaf of a structure (these are MMS basic data types), the value range (size) is also defined besides the class. The value range of the class INTEGER can contain one, two or more octets. The value range four (4) octets (often represented as Int32) e. g. indicates that the value cannot exceed these four octets. On the other hand, with ASN.1 BER coding (explained later) and the value range Int32, the decimal value "5" will be transmitted in only one octet (not in 4 octets!). That is to say, only the length needed for the current value will be transmitted.

- The aspect of the representation of the data and their structuring during transmission on the line (communication in the original meaning) is dealt with below in the context of the encoding of messages.

The path description is used in a quintuple way (see figure 17):

- during the access to and during the definition of variable objects,

- in the type description of variable objects,

- in the description of data during reading for example,

- during the definition of Named Type Objects (an object name of its own is assigned to the type description – i.e. to one or several paths – of those objects) and

- when reading the attributes of variables and Named Type Objects.

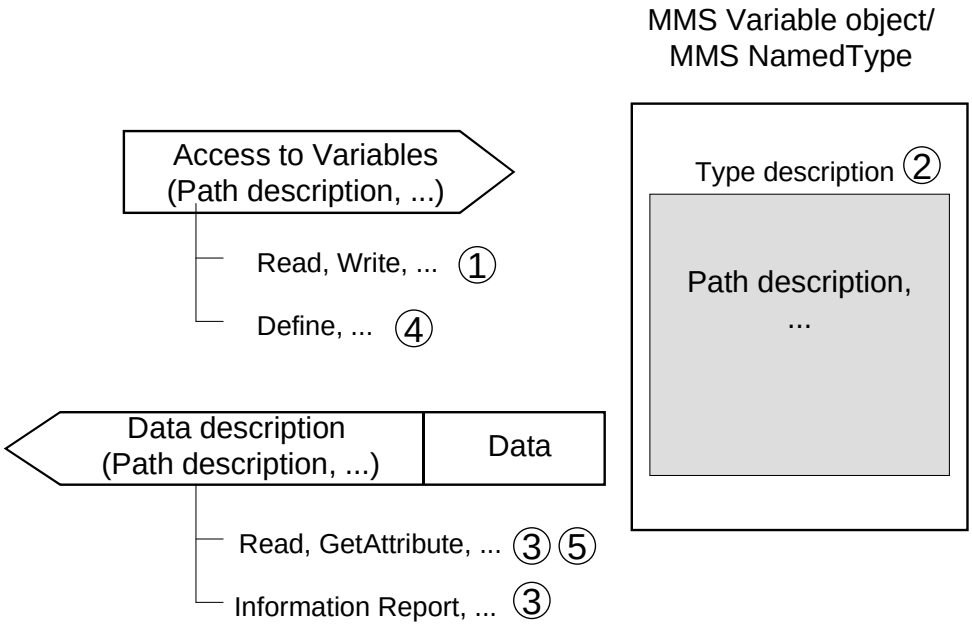

Figure 17 Application of path descriptions

Objects of the MMS variable model

The five objects of the MMS variable model are:

Description of simple or complex values:

- Unnamed Variable,
- Named Variable.

List of several unnamed variables or named variables:

- Named Variable List,
- Scattered Access (is not explained here).

Description of the structure by means of a user-defined name:

- Named Type.

The Unnamed Variable

The Unnamed Variable Object describes the assignment of an individual MMS variable to a real variable which is located at a definite address in the device. An Unnamed Variable Object can never be created or deleted. The Unnamed Variable has the attributes:

Object:Unnamed Variable

Key Attribute:	Address
Attribute:	MMS Deletable (FALSE)
Attribute:	Access Method (PUBLIC)
Attribute:	Type Description

Address: Address is used to reference the object. There are three different kinds:

a) Numeric Address (non-negative integer values),

b) Symbolic Address (character string),

c) Unconstrained Address (implementation specific format).

Even though in b) the address is represented by character strings, this kind of addressing has absolutely to be distinguished from the Object Name of a Named Variable (see domain scope and explanations below).

MMS Deletable: The attribute is always FALSE here.

Access method: The attribute is always PUBLIC here.

Type Description: The attribute points to the inherent abstract type of the subordinate real variable as it is seen by MMS. It specifies the class (bitstring, integer, floating-point etc.), the value range of the values and the group formation of the real variable (arrays of structures). The attribute Type Description is completely independent of the addressing.

Figure 18 represents the Unnamed Variable roughly sketched. The Unnamed Variable with the address "62" (MMSString) has three components with the names: Value, Quality and Time. These component names are only required if individual components (specifying the path, for example "62"/"Value") shall be accessed.

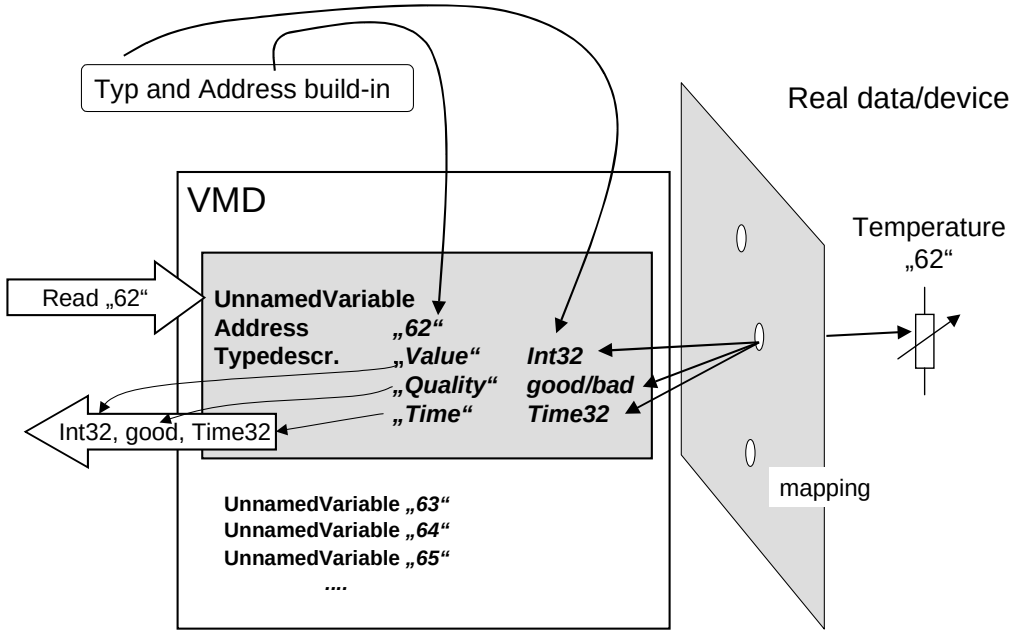

Figure 18 Unnamed Variable Object

MMS Address of the Unnamed Variable

The MMS Address is a system-specific reference which is used by the system for the internal addressing – it is quasi released for the access via MMS. There the address can assume one of three forms (here the ASN.1 notation is deliberately used for the first time):

```
Address ::= CHOICE {
    numericAddress          [0] IMPLICIT Unsigned32,
    symbolicAddress         [1] MMSString,
    unconstrainedAddress    [2] IMPLICIT OCTET STRING
    }
```

The definition above has to be read as follows: Address defines as (::=) a selection (keyword CHOICE) of three possibilities. The possibilities are numbered here from [0] to [2] to be able to distinguish them. The keyword IMPLICIT is discussed later.

The numeric address is defined as Unsigned32 (4 octet). Thus, the addresses can be defined as an index with a value range of up to 2**32. Since only the actual length (e. g. only one octet for the value 65) will be transmitted for an Unsigned32, the minimal length of the index, which can thus be used, is merely 1 octet. Already 255 objects (of arbitrary complexity!) can be addressed with one octet.

The symbolic address can transmit an arbitrarily long MMSString (for example "DB5_DW6").

The Unconstrained Address represents an arbitrarily long octet string (for example 24FE23F2A1hex). The meaning and the structure of these addresses is outside the scope of the standard.

These addresses can be used in MMS Unnamed Variable and Named Variable Objects and in the corresponding services. MMS can neither define nor change these addresses. The address offers a possibility to reference objects by short indexes.

The addresses can be structured arbitrarily. Unnamed Variables could, for example, contain measurements in the address range [1 000 to 1 999], status information in the address range [3 000 to 3 999], limit values in the address range [7 000 to 7 999] etc.

Services for the Unnamed Variable Object

Read: This service uses the V-Get function to transmit the current value of the real variable, which is described by the Unnamed Variable Object, from a server to a client. Variable Get represents the internal, system-specific function through which an implementation gets the actual data and provides them for the communication.

Write: This service uses the V-Put function to replace the current value of the real variable, which is described by the Unnamed Variable Object, by the enclosed value.

Information Report: As Read, though without prior request by the client. Only the Read.response is sent by the server to the client without being asked. The Information Re-

port corresponds to a spontaneous message. The application itself determines when the transmission is to be activated.

Get Variable Access Attributes: Through this operation, a client can query the attributes of a Unnamed Variable Object.

Explanation of the TypeDescription

Features of the structure description of MMS variable objects were explained in principle above. For those interested in the details, the formal definition of the MMS Type Specification is explained according to figure 19.

```
TypeSpecification ::= CHOICE {
        typeName            [0] ObjectName,
        array                   [1] IMPLICIT SEQUENCE {
                packed              [0] IMPLICIT BOOLEAN DEFAULT FALSE,
                numberOfElements    [1] IMPLICIT Unsigned32,
                elementType             [2] TypeSpecification },
        structure           [2] IMPLICIT SEQUENCE {
                packed              [0] IMPLICIT BOOLEAN DEFAULT FALSE,
                components          [1] IMPLICIT SEQUENCE OF SEQUENCE {
                        componentName       [0] IMPLICIT Identifier OPTIONAL,
                        componentType       [1] TypeSpecification } },

        Simple                  Size                        Class
        boolean             [3] IMPLICIT NULL,              - BOOLEAN
        bit-string          [4] IMPLICIT Integer32,         - BIT-STRING
        integer                 [5] IMPLICIT Unsigned8,         - INTEGER
        unsigned            [6] IMPLICIT Unsigned8,         - UNSIGNED
        floating-point      [7] IMPLICIT SEQUENCE {
                format-width        Unsigned8,      - # of bits in fraction plus sign
                exponent-width      Unsigned8       - size of exponent in bits },
        real                [8] IMPLICIT SEQUENCE {
                base                [0] IMPLICIT INTEGER(2|10),
                exponent            [1] IMPLICIT INTEGER,       - max number of octets
                mantissa            [2] IMPLICIT INTEGER        - max number of octets },
        octet-string        [9] IMPLICIT Integer32,         - OCTET-STRING
        visible-string      [10] IMPLICIT Integer32,        - VISIBLE-STRING
        generalized-time        [11] IMPLICIT NULL,             - GENERALIZED-TIME
        binary-time         [12] IMPLICIT BOOLEAN,    - BINARY-TIME
        bcd                     [13] IMPLICIT Unsigned8         - BCD
        objId               [15] IMPLICIT NULL}
```

Figure 19 MMS Typespecification

The description in ASN.1 was deliberately selected also here. The type specification is a CHOICE (selection) of 15 possibilities (Tags [0] to [13] and [15]). Tags are qualifications of the selected possibility. The first possibility is the specification of an ObjectName, a Named Type Object. If we remember that one Named Type Object describes one or several paths, then the use is obvious. The path description referenced by the name can be used to define a Named Variable Object. Or, if during reading the path must be specified, it can be referenced by a Named Type Object in the server.

Note that the ASN.1 definitions in MMS are comparable with XML schema. ASN.1 BER (Basic encoding rules) provide very efficient message encoding compared to XML documents. The

coming standard IEC 61400-25 provides applies ASN.1 as well as XML schema for the specification of messages.

The two next possibilities (array and structure) have a common feature. Both refer – through their element type or component type – back to the beginning of the complete definition (type specification). This recursive definition allows the definition of arbitrarily complex structures. Thus, an element of a structure can in turn be a structure or an array.

Arrays are defined by three features. Packed defines whether or not the data are stored optimized. Number-OfElements indicates the number of the elements of the array of equal type (Element Type).

The data of structures can also be saved as packed. Structures consist of a series of components (components [1] IMPLICIT SEQUENCE OF SEQUENCE). This series is marked by the keyword SEQUENCE OF ... which describes a repetition of the following definition. Next in the list is SEQUENCE {component Name und component Type} which describes the individual component. Since the SEQUENCE OF ... (repetition) can be arbitrarily long, the number of the components at a node is also arbitrary.

Then follow the "simple data types". They start at the tag [3]. The length of the types is typical for the simple data types. For example, integers of different lengths can be defined. The length (size) is defined as Unsigned8, which allows for an integer with the length of 255 octets.

It should be mentioned here that – in the ASN.1 description of the MMS syntax – expressions like integer (written in small letters) show that they are replaced by another definition (in this case by the Tag [5] with the IMPLICIT-Unsigned8 definition). Capital letters at the beginning indicate that the definition is terminated here – it isn't replaced any more. It is here a basic definition.

Figure 20 shows an example of an object defined in IEC 61850-7-4. The circuit breaker class is instantiated as ”XCBR1”. The hierarchical components of the object is mapped to MMS (according to IEC 61850-8-1). The circuit breaker is defined as a comprehensive MMS Named Variable.

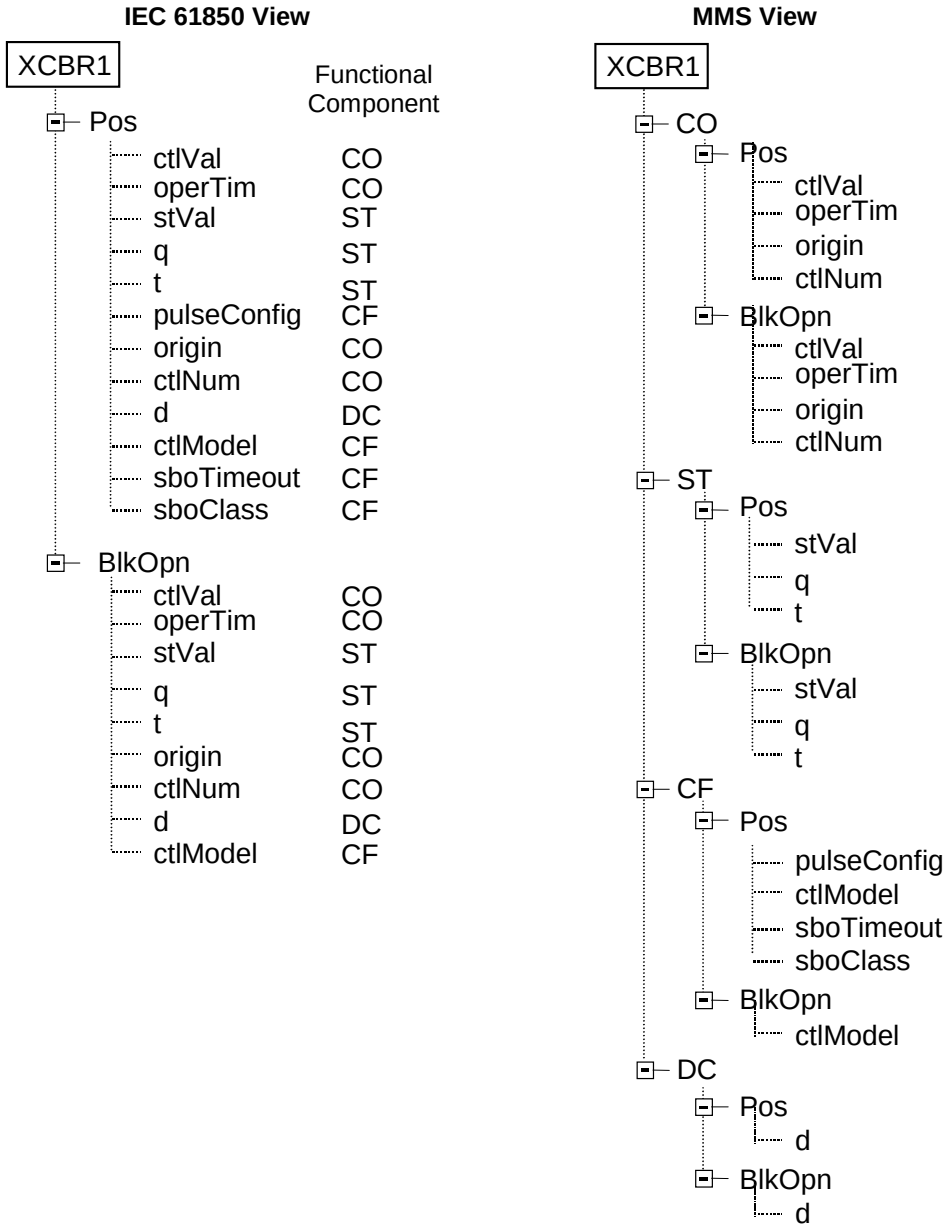

Figure 20 Example MMS Named Variable

The components of the hirarchical model can be accessed by the description of the Alternate Access: “XCBR1” -> component “ST” -> component “Pos” -> component “stVal”. Another possibility of mapping the hierarchy to a flat name is depicted in figure 21. Each path is defined as a character string.

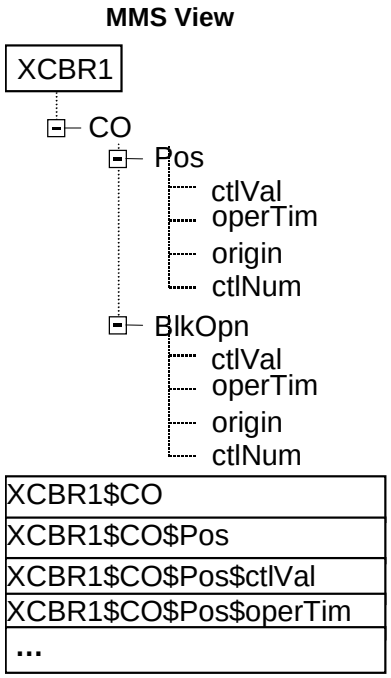

Figure 21 Example MMS Named Variable

The Named Variable

The Named Variable Object describes the assignment of a named MMS variable to a real variable. Only one Named Variable Object should be assigned to a real variable. The attributes of the object are as follows:

Object: Named Variable
- Key Attribute: Variable Name
- Attribute: MMS Deletable
- Attribute: Type Description
- Attribute: Access Method (PUBLIC, ...)
  - Constraint: Access Method = PUBLIC
  - Attribute: Address

Variable Name: The Variable Name unambiguously defines the Named Variable Object in a given scope (VMD specific, domain specific or Application Association specific). The Variable Name can be 32 characters long (plus 32 characters if the object has a domain scope).

MMS Deletable: This attribute shows whether or not the object may be deleted using a service.

Type Description: This attribute describes the abstract type of the subordinate real variable as it represents itself to the external user. This attribute is not inherently in the system unlike the Unnamed Variable Object, i.e. this Type Description can be defined from the outside.

Access method: This attribute contains the information which a device needs to identify the real variable. It contains values which are necessary and adequate to find the memory location. The contents lie outside MMS. A special method, the method PUBLIC, is standardized. The attribute Address is also available in the case of PUBLIC. This is the address which identifies an Unnamed Variable Object. Named Variables can thus be addressed by the name and the ADDRESS (see figure 22).

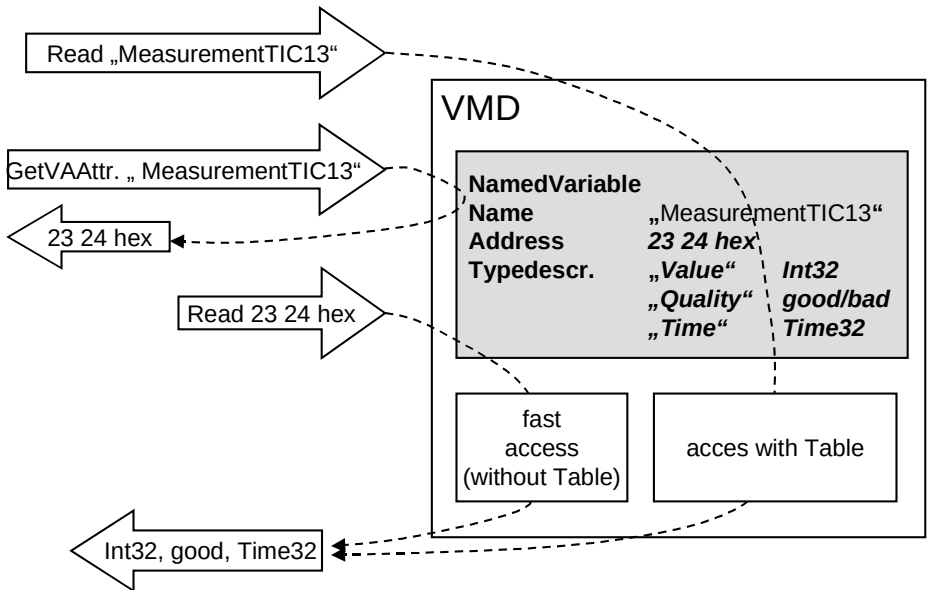

Figure 22 Address and Variable Name of Named Variable Objects

Address: See Unnamed Variable Object. Defining a Named Variable Object does not allocate any memory either, because the real variable must already exist; it is assigned to the Named Variable Object with the corresponding name.

Altogether six operations are defined on the object:

Read: The service uses the V-Get function to retrieve the current value of the real data which is described by the Object.

Write: The service uses the V-Put function to replace the current value of the real data, which is described by the Object, by the enclosed value.

Define Named Variable: This service creates a new Named Variable Object which is assigned to real data.

Get Variable Access Attribute: Through this operation, a client can query the attributes of a Named Variable Object.

Delete Variable Access: This service deletes a Named Variable Object if Attribute Deletable is (=TRUE).

Figure 23 shows the possibilities to reference a Named Variable Object by names, and if it is required, also by Address (optimal access reference of a given system). For a given name, a client can query the Address by means of the service Get Variable Access Attribute.

This possibility allows the access through technological names (MeasurementTIC13) or with the optimal (index-) address 23 24 hex.

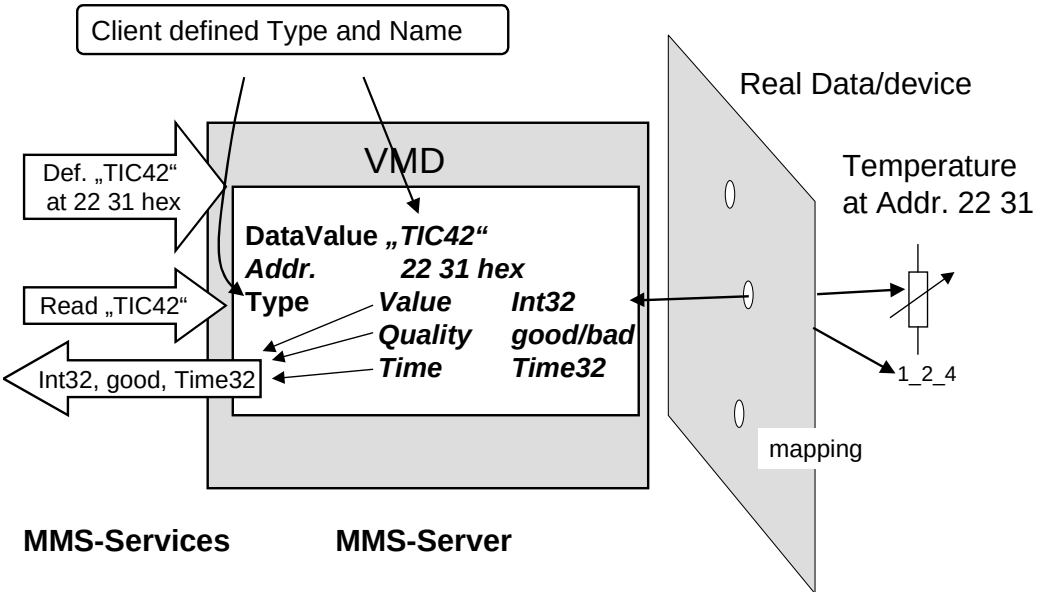

Figure 23 Client defined Named Variable Object

As shown in figure 23, an essential feature of the VMD is the possibility for the client application to define by request Named Variable Objects in the server via the communication. This includes the definition of the name, the type and the structure. The name by which the client would like to reference the Named Variable later is "TIC42" here. The first component "Value" is of the type Integer32, the second is "Quality" with the values "good" or "bad", the third is "Time" of the type "Time32". The type of the Data Value Object can be arbitrarily simple (flat) or complex (hierarchical). As a rule, the Data Value Objects are implicitly created by the local configuring or programming of the server (they are pre-defined).

The internal assignment of the variable to the real temperature measurement is made by a system-specific, optimal reference. This reference whose structure and contents are transparent must be known when defining the Named Variable, though. The reference can e. g. be a relative memory address (for example DB5 DW15 of a PLC). So a quick access to the data is allowed.

The Named Variable Object describes how data for the communication are modeled, accessed, encoded and transmitted. What is transmitted, is described independently of the function. From the point of view of the communication, it is not relevant where the data in the server actually come from or where in the client they actually go to and how they are managed – this is deliberately concealed.

Figure 24 shows the concrete encoding of the Information Report message. The message is encoded according to ASN.1 BER (the basic encoding rule for abstract syntax notation number one – ISO 8825).

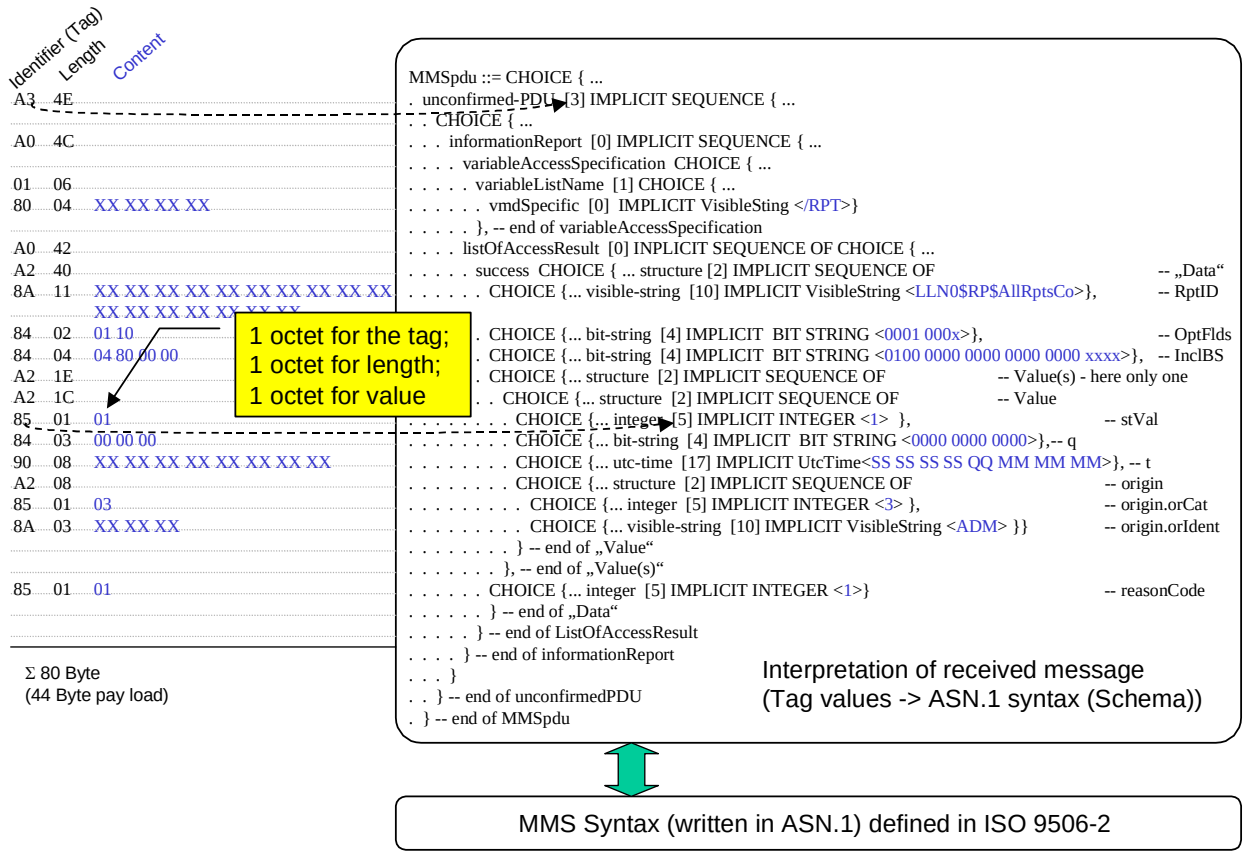

Figure 24 MMS Information Report (spontaneous message)

The encoding using XML would be several times longer as using ASN.1 BER. These Octets are packed into further messages that add lower layer-specific control and address information, e.g., the TCP header, IP header, Ethernet frames.

The receiving IED is able to interpret the Report message according to the identifier, lengths, names, and other values. The interpretation of the message requires the same stack, i.e., knowledge of all layers involved – including the definitions of IEC 61850-7-4, IEC 61850-7-3, IEC 61850-7-2, and IEC 61850-7-1.

The access to several variables

Named Variable List

The Named Variable List allows to define a single name for a group of references to arbitrary MMS Unnamed Variables and Named Variables. Thus, the Named Variable List offers a grouping for the frequently repeated access to several variables (see figure 25). Although

the simultaneous access to several MMS variables can be carried out also in a single service (Read or Write), the Named Variable List offers a substantial advantage.

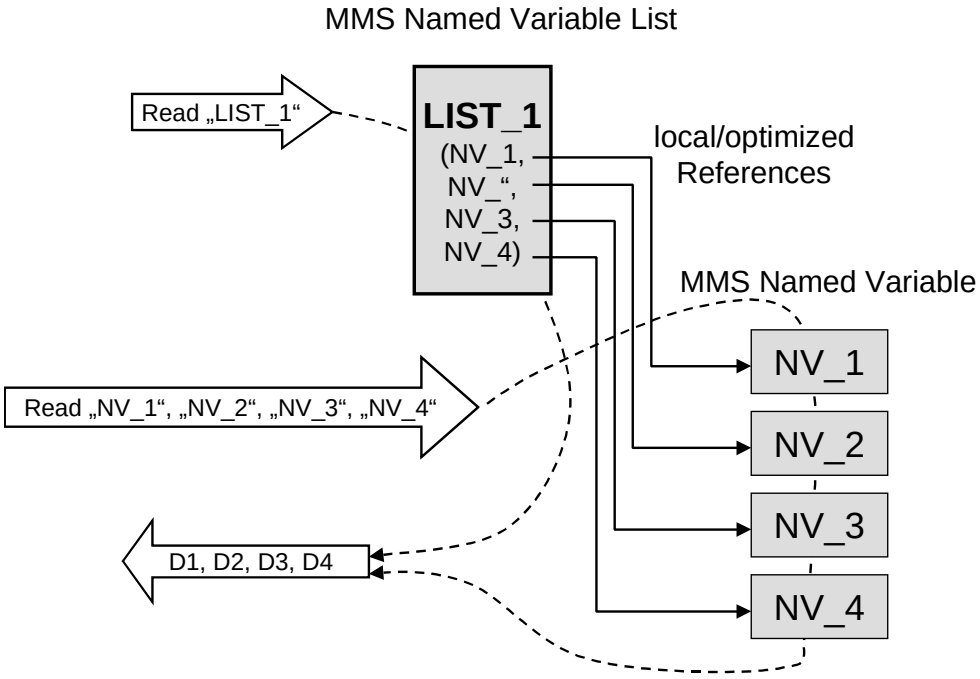

Figure 25 MMS Named Type and Named Variable

When reading several variables in a Read request, the individual names and the internal access parameters (pointers and lengths), corresponding to the names in the request, must be searched for in a server. This search can last for some time in the case of many names or a low processor performance. By using the Named Variable List Object, the search is not required – except for the search of a single name (the name of the Named Variable List Object) – if the references, for example, have been entered into the Named Variables on the list system-specifically and thus optimally. Once the name of the list has been found, the appropriate data can be provided quickly.

Thus, the Named Variable List Object provides optimal access features for the applications. This object class is used in the known applications of MMS very intensively.

The structure of the Named Variable List Object is as follows:

Object: Named Variable List
- Key Attribute: Variable List Name
- Attribute: MMS Deletable (TRUE, FALSE)
- Attribute: List of Variable
  - Attribute: Kind of Reference (NAMED, UNNAMED, SCATTERED)
  - Attribute: Reference
  - Attribute: Access Description

Variable List Name: The Variable List Name unambiguously identifies the Named Variable List Object in a given scope (VMD specific, domain specific or Application Association specific). See also MMS Object Names above.

MMS Deletable: This attribute shows whether or not the object may be deleted.

List of Variable: A list can contain an arbitrary number of objects (Unnamed Variable, Named Variable or Scattered-Access Object).

Kind of Reference: Lists can refer to three object classes: Named Variables, Unnamed Variables and Scattered Access; no Named Variable Lists can be included.

Reference: An optimal internal reference to the actual data is assigned to every element of the list. If a referenced object is not (any more) available, the entry into the list will indicate it. When accessing the list, for example by Read, no data but an error indication will be transmitted to the client for this element.

Access Description: In the same way as for the access of a variable – e. g. in the request during Read – could be defined that only parts (part of a tree) of a variable shall be read, can this also be applied to every element of the Named Variable List.

Services

Read: This service reads the data of all objects being part of the list (Unnamed Variable, Named Variable and Scattered Access Object). For objects which are not defined an error is reported in the corresponding place of the list of the returned values.

Write: This service writes the data from the write request into the objects being part of the list (Unnamed Variable, Named Variable and Scattered Access Object). For objects which are not defined an error is reported in the corresponding place of the list of the returned values.

Information Report: As the Read service; as if the read data were sent by the server to the client without prior request (Read Request) by the client, i.e. as if only a Read Response would be transmitted.

Define Named Variable List: Using this service a client can create a Named Variable List Object.

Get Named Variable List Attributes: This service queries the attributes of a Named Variable List Object.

Delete Named Variable List: This service deletes the specified Named Variable List Object.

The Named Type Object

The Named Type Object merely describes structures. The object model is very simple:

Object: Named Type

Key Attribute:	Type Name
Attribute:	MMS Deletable (TRUE, FALSE)
Attribute:	Type Description

Essential attribute is the Type Description, which was already discussed before for Named and Unnamed Variables. On the one hand, TASE.2 standard data structures can be specified by means of Named Types. This is the most frequent application of the Named Type Objects. On the other hand, Named Types can be used for the access to the server. The read request can refer to a Named Type Object. Or the Named Type Object will be used to define Named Variables.

Figure 26 describes the application of the Named Type Objects for the definition of a Named Variable. A variable will be created by the request Define Named Variable.

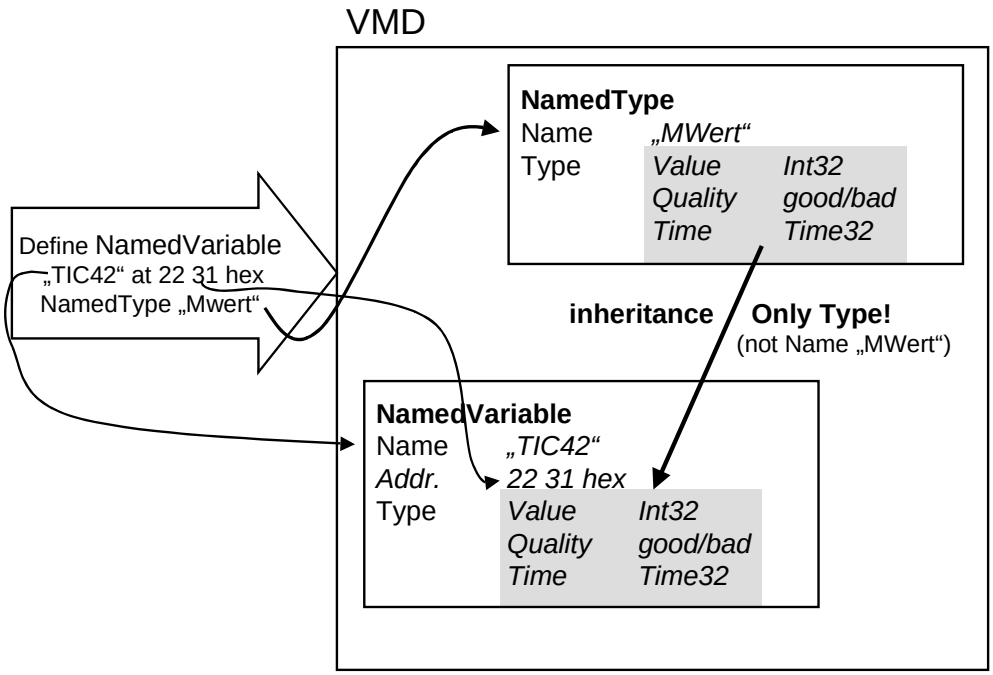

Figure 26 Inheritance of Type of the MMS NamedType Objects

It shall have the name "TIC42", the address 22 31 hex and the type that is defined in the Named Type Object "MWert". The variable inherits the type from the Named Type Object. The inheritance has the consequence that the variable will only have the type but not the name of the Named Type Object. This inheritance was defined so strictly in order to avoid that through deleting the Named Type the type of the variable would get undefined or that by subsequent definition of a differently structured Named Type with the old type name "MWert" the type of the variable would be changed (the Named Type Object and with that the new Type Description would be referenced by the old name).

Perhaps it is objected now that this strict inheritance has the consequence that also the type would have to be saved for each variable (even though many variables have the same

Type Description). Since these variables can internally be implemented in a system in whatever way the programmer likes it, they can refer through an internal index to a single (!) type description. He must only make sure that this type description is not deleted. If the accompanying Named Type Object gets deleted, then the referenced type description must remain preserved for these many variables.

The disadvantage that the name of the "structure mother" , i.e. the Named Type, is not known any more as an attribute of the variables has been eliminated in the MMS Revision.

Services

Define Named Type: This services creates a Named Type Object.

Get Named Type Attribute: This service delivers all attributes of a Named Type Object.

Read, Write, Define Named Variable, Define Scattered Access, Define Named Variable List, Define Named Type use the Type Description of the Named Type Object when carrying out their tasks.

13 Resume

MMS is a standard messaging specification (comparable to web services), widely implemented by industrial device manufacturers like ABB, Alstom, General Electric, Siemens. It solves problems of heterogeneity so often found in automation applications. MMS is the lingua franca of industrial devices.

MMS provides much more than TCP/IP which essentially offers a transfer stream of bytes. MMS transfers commands with parameters between machines.

MMS allows a user to concentrate on the applications, application data to be accessible – and not on communication problems, which are already solved. It prodes a basis for the definition of common and domain-specific semantic. Examples are the standards IEC 60870-6 TASE.2, IEC 61850, and IEC 61400-25.

14 Literature and web pages

[1] Manufacturing Message Specification (MMS) – Part 1 Service definition ISO International Standard ISO 9506-1, 2003

[2] Manufacturing Message Specification (MMS) – Part 2 Protocol Definition ISO International Standard ISO 9506-2, 2003

[3] Manufacturing Message Specification (MMS) – Part 3 Companion Standard for robitics ISO/IEC 9506-3, 1991

[4] Manufacturing Message Specification (MMS) – Part 4 Companion Standard for numerical control ISO/IEC 9506-4, 1992

[5] Manufacturing Message Specification (MMS) – Part 5 Companion Standard for Programmable controllers ISO/IEC CD 9506-5, 1993

[6] Manufacturing Message Specification (MMS) – Part 6 Companion Standard for Process control ISO/IEC 9506-6, 1993

[7] ESPRIT Consortium CCE-CNMA, Preston, UK (Editoren): MMS: A Communication Language of Manufacturing. Berlin: Springer-Verlag, 1995

[8] ESPRIT Consortium CCE-CNMA, Preston, UK (Editoren): CCE: An Integration Platform for Distributed Manufacturing Applications. Berlin: Springer-Verlag, 1995

[9] Inter-Control Center Communication, IEEE Transactions on Power Delivery, Vol. 12 (April 1997) No. 2, pp. 607 – 615

[10] Telecontrol equipment and systems – Part 6: Telecontrol protocols compatible with ISO standards and ITU-T recommendations – Section 503: Services and Protocol (ICCP Part 1) IEC 60870-6-503, 1997

[11] Telecontrol equipment and systems – Part 6: Telecontrol protocols compatible with ISO standards and ITU-T recommendations – Section 802: Object Models (ICCP Part 4) IEC 60870-6-802, 1997

[12] März, W.; Schwarz, K.: Powerful and open communication platforms for the operation of interconnected networks, ETG-Tage/IEEE PES Summer Meeting 1997, Proceedings, Berlin

[13] IEEE Technical Report 1550 (1999): Utility Communications Architecture, UCA; http://www.nettedautomation.com/standardization/IEEE_SCC36_UCA

[14] IEC 60870-6-TASE.2: Telecontrol application service element 2

[15] Becker, Gerhard; Gärtner, W.; Kimpel, T.; Link, V.; März, W.; Schmitz, W.; Schwarz, K.: Open Communication Platforms for Telecontrol Applications – Benefits from the New Standard IEC 60870-6 TASE.2 (ICCP), etz-Report 32, VDE-Verlag Berlin, 1999

[16] Wind Power Communication - Verification report and recommendation; Elforsk rapport 02:14; Stockholm April 2002;
www.nettedautomation.com/download/02_14_rapport.pdf

[17] IEC 61850-7-1, Communication networks and systems in substations – Part 7-1: Basic communication structure for substation and feeder equipment – Principles and models

[18] IEC 61850-7-2, Communication networks and systems in substations – Part 7-2: Basic communication structure for substation and feeder equipment – Abstract communication service interface (ACSI)

[19] IEC 61850-7-3, Communication networks and systems in substations – Part 7-3: Basic communication structure for substation and feeder equipment – Common data classes

[20] IEC 61850-7-4, Communication networks and systems in substations – Part 7-4: Basic communication structure for substation and feeder equipment – Compatible logical node classes and data classes

[21] IEC 61850-8-1, Communication networks and systems in substations – Part 8-1: Specific communication service mapping (SCSM) – Mappings to MMS (ISO/IEC 9506-1 and ISO/IEC 9506-2) and to ISO/IEC 8802-3

[22] IEC 61400-25, Wind turbines – Part 25: Communications for monitoring and control of wind power plants

Useful web pages that provide additional information:

http://www.scc-online.de

http://www.nettedautomation.com

http://www.nettedautomation.com/download/SCC-Profile-en_2008-01-16.pdf

IEC 61850 circuit breaker model:

http://www.nettedautomation.com/qanda/iec61850/information-service.html#

Dipl.-Ing. Karlheinz Schwarz (55) received his diploma (master degree) in Information Technol-ogy at the University of Siegen (Germany) in 1982. He is married and has four children and five grandchildren.

As a manager with Siemens Automation & Drives (communication systems) he represented the po-sitions of Siemens and the German national committee in the international standardization of MAP, MMS, MMS companion standards, Fieldbus, and other standardization projects from 1984 until 1997.

He is president of SCC (Schwarz Consulting Company), Karlsruhe (Germany) specializing in distributed automation systems. He is an independent consultant in the area of information modeling, systems and information integration, system and device engineering and configuration, open information exchange, and open communications since 1992. Mr. Schwarz has immense experience in the migration from proprietary or other solutions to standard compliant solutions.

He is involved in many standardization activities within IEC (TC 57, TC 65, and TC 88), ISO (TC 184), CENELEC (TC 65 CX), IEEE (SCC 36 "UCA", 802), and DIN since 1985. He is engaged in repre-senting main industry branches in the global standardization and providing consulting services to us-ers and vendors. Mr. Schwarz is a well-known authority in the application of mainstream information and communication technologies. He provides guidance in the migration from proprietary solutions to advanced seamless and standard-based solutions applicable in substations, and power generation units, and between these and with local, regional, and central SCADA systems. Specifically, his contri-butions to the publication of many standards are considered to be outstanding.

He has been awarded with the IEC 1906 Award in 2007 “For his strong involvement in the edition of the IEC 61850 series, its promotion inside and outside IEC, and specifically its adaptation for wind turbine plant control.

http://www.nettedautomation.com/download/IEC1906-Award.pdf