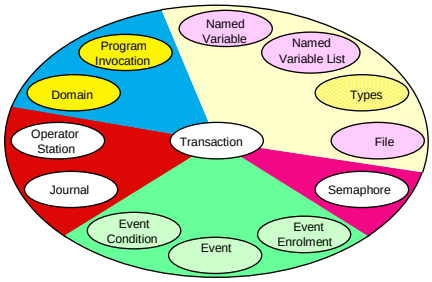


MMS - Manufacturing Message Specifications

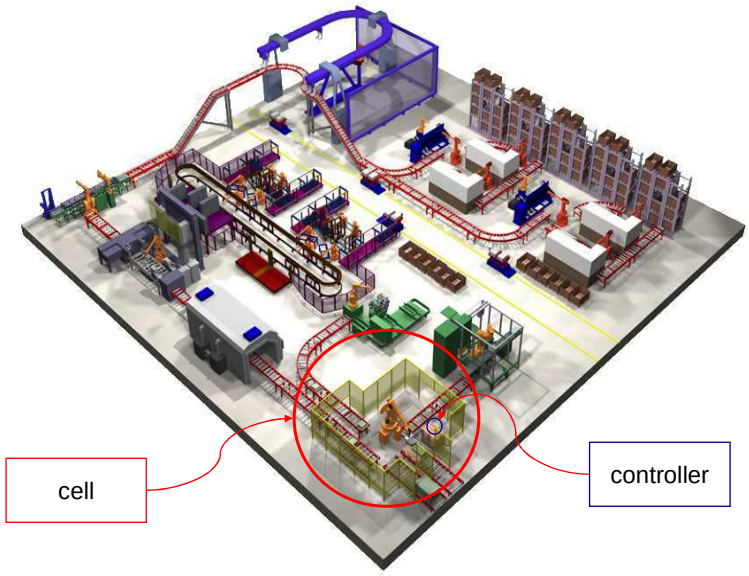
What MMS is

MMS (**M**anufacturing **M**essage **S**pecification) is an internationally standardized messaging system for exchanging real-time data and supervisory control information between networked devices and/or computer applications in a manner that is independent of:

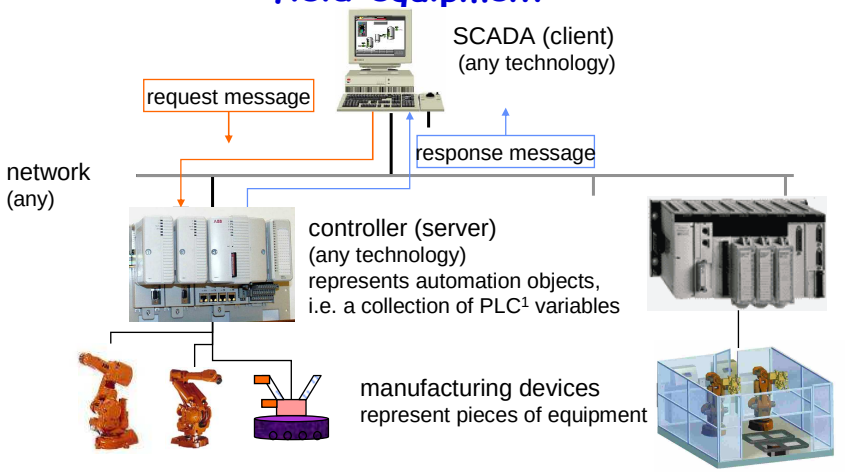
- the application function being performed or
- the developer of the device or application.

MMS is an international standard (ISO 9506) that has been developed and maintained by Technical Committee Number 184 (TC184), Industrial Automation, of the ISO

MMS Application domain



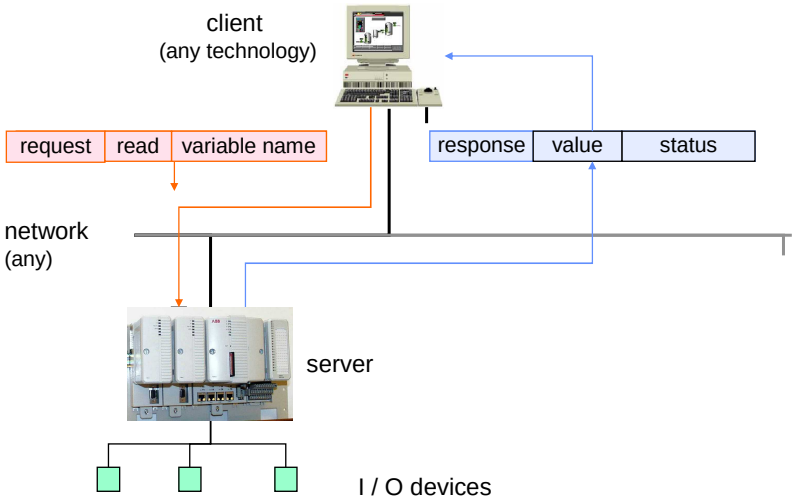
Interaction between Operator Workplace and field equipment



MMS: we want to access all controllers, regardless of the manufacturer, in the same way. The information exchange between the client and the server was performed through proprietary messages.

1: PLC= Programmable Logic Controller

The basic MMS idea: read a variable

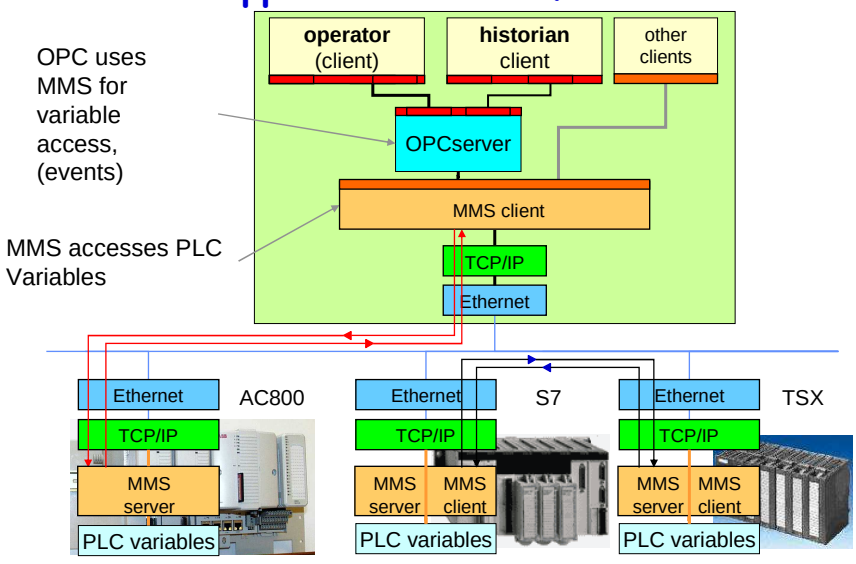


basic MMS idea: read and write equipment variables using standard messages.

Application: MMS for OPC

OPC uses MMS for variable access, (events)

MMS accesses PLC Variables



intention: any PLC should be accessed that way (MMS as universal server)

MMS - Manufacturing Message Specification history

Developed 1980 (!) for the MAP project (General Motor's flexible manufacturing initiative)

Originally unluckily tied to the OSI communication stack and Token Bus (IEEE 802.4)

Reputed for being heavy, complicated and costly due to poor implementations.

Boeing adopted MMS as TOPs (MMS on Ethernet) - a wise step.

Adopted by the automobile industry, aerospace industry, and PLC manufacturers: Siemens, Schneider, Daimler, ABB.

Standardized since 1990 as:

[1] ISO/IEC 9506-1 (2003): Industrial Automation systems - Manufacturing Message Specification - Part 1: Service Definition

[2] ISO/IEC 9506-2 (2003): Industrial Automation systems - Manufacturing Message Specification - Part 2: Protocol Specification

MMS - Concept

- MMS consists of two or more parts. Parts 1 and 2 define what is referred to as the "core" of MMS.
- Part 1 is the *service* specification. The service specification contains a definition of:
- 1) the Virtual Manufacturing Device (VMD),
- 2) the services (or messages) exchanged between nodes on a network, 3) the attributes and parameters associated with the VMD and services.
- Part 2 is the *protocol* specification which defines the rules of communication which includes:
- 1) the sequencing of messages across the network,
- 2) the format (or encoding) of the messages
- 3) the interaction of the MMS layer with the other layers of the OSI model.

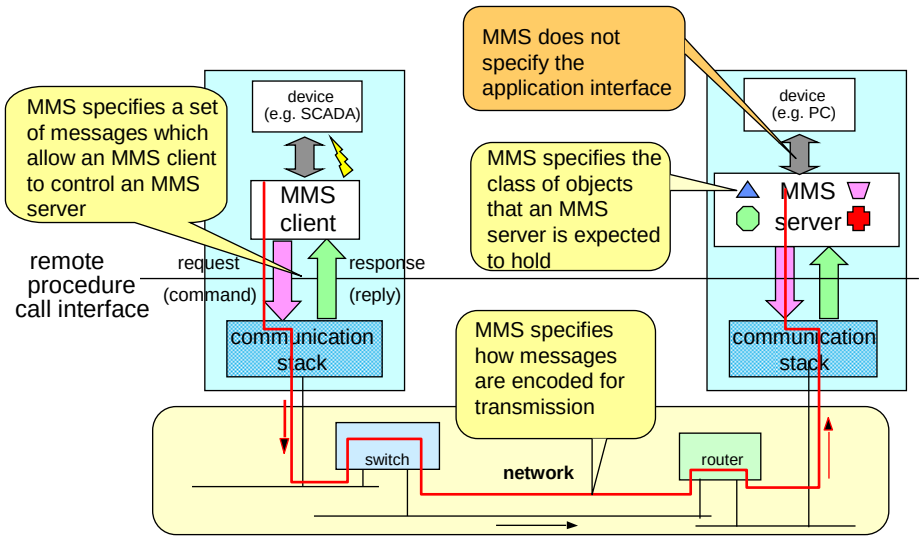
MMS - Concept

MMS (Manufacturing Message Specifications) defines:

- 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 client and a server station for the purpose of controlling these objects
- A set of encoding rules for these messages (how values and parameters are mapped to bits and bytes when transmitted)
- 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 - Communication model



Benefits provided by MMS

- MMS provides benefits by lowering the cost of building and using automated systems.
- MMS is appropriate for any application that requires a common communications mechanism for performing a diversity of communications functions related to real-time access and distribution of process data and supervisory control.
- It is important to evaluate the three major effects of using MMS that can contribute to cost savings:
- 1)Interoperability,
- 2) Independence
- 3) Access.

Benefits provided by MMS

- **Interoperability** is the ability of two or more networked applications to exchange useful supervisory control and process data information between them without the user of the applications having to create the communications environment.
- **Independence** allows interoperability to be achieved independent of:
- *The Developer of the Application.*
- *Network Connectivity.*
- *Function Performed.*
- **Data Access** is the ability of networked applications to obtain the information required by an application to provide a useful function.
- Although virtually any communications scheme can provide access to data at least in some minimal manner, they lack the other benefits of MMS, particularly Independence

MMS mapping to communication

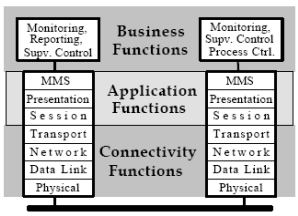
MMS **is not by itself a communication protocol**, it defines messages that have to be transported by an unspecified network

While many communications schemes only provide a mechanism for transmitting a sequence of bytes (a message) across a network, MMS does much more.

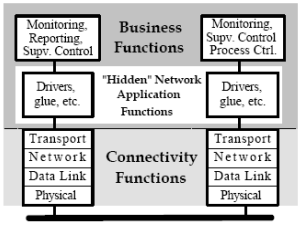
MMS also provides definition, structure, and meaning to the messages that significantly enhances the likelihood of two independently developed applications interoperating.

MMS has a set of features that facilitate the real-time distribution of data and supervisory control functions across a network in a client/server environment that can be as simple or sophisticated as the application warrants.

Justifying MMS



The MMS View of Network Applications

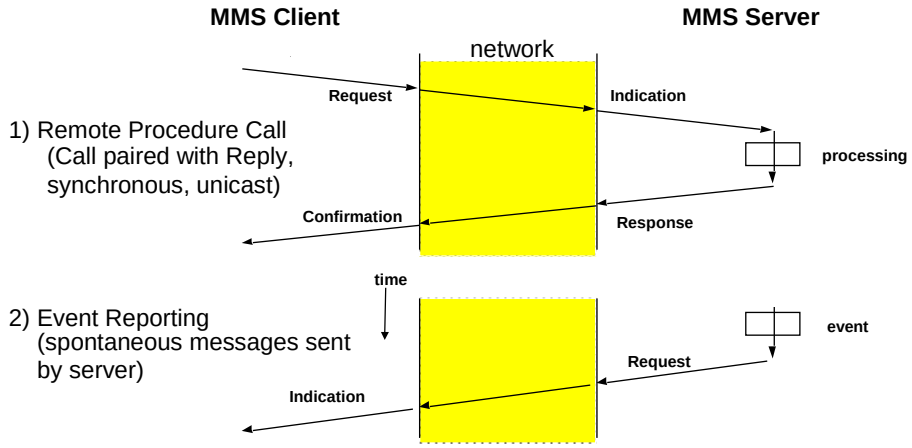


The Common View of Network Applications

MMS, provides *application* services to the business functions, not *connectivity* services.
The network cannot be considered simply as a mechanism to transfer messages (connectivity only). That view hides the value of the application functions because they become indistinguishable from the business applications which then must provide the network application functions.

MMS - Underlying Communication Principles

MMS is in principle independent from the communication stack.
MMS only requires that two types of communication services exist:

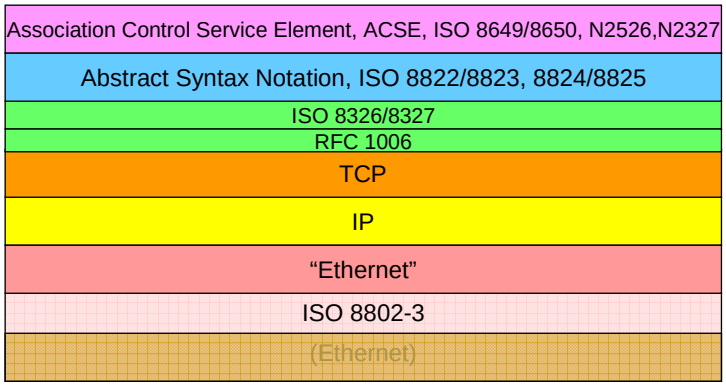


MMS - Original Communication Stack

Association Control Service Element, ACSE, ISO 8649/8650, N2526,N2327	"Application"
Abstract Syntax Notation, ISO 8822/8823, 8824/8825	Presentation
ISO 8326/8327	Session
ISO 8073 Class 4	Transport
ISO 8473 connectionless	Network
ISO 8802-2 Type 1	Link
ISO 8802-3 / ISO 8802-4	MAC
(Ethernet) / (token bus)	Physical

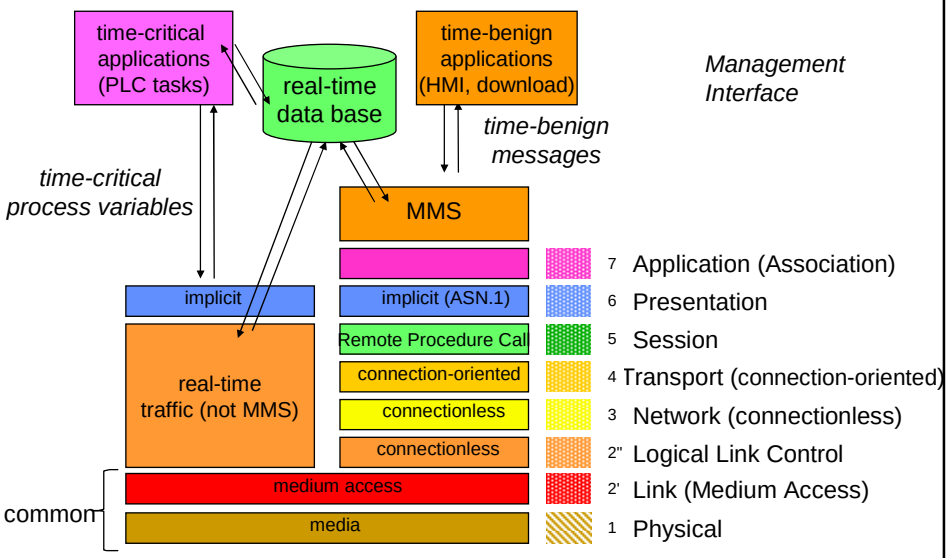
quite heavy… Boeing decided to drop ISO for TCP/IP, was not followed until 1999...

Current MMS Stack



much simpler...

MMS in the fieldbus stack

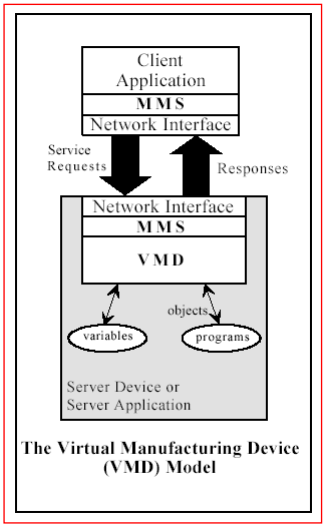


MMS is not for real-time communication, but it can access the real-time variables

MMS Objects

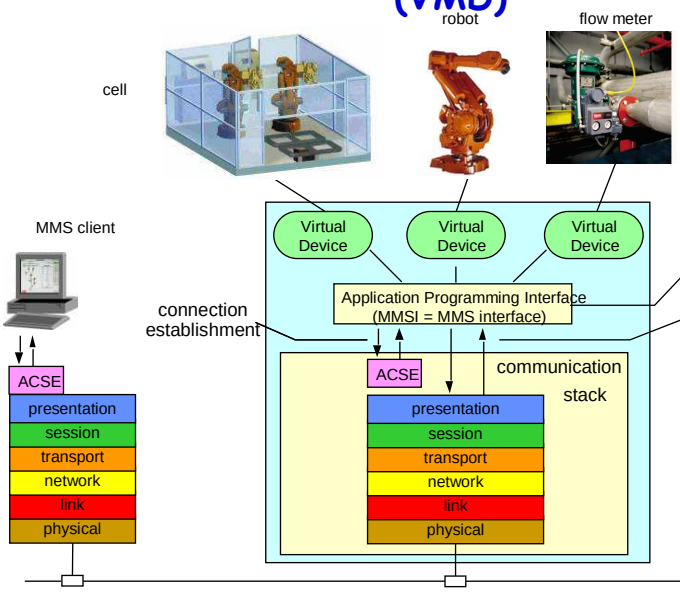
Each MMS server is expected to contain a number of standard objects

The key feature of MMS is the **V**irtual **M**anufacturing **D**evice (VMD) model.
The VMD model specifies how MMS devices, also called servers, behave as viewed from an external MMS client application point of view. MMS allows any application or device to provide both client and server functions simultaneously.



The Virtual Manufacturing Device (VMD) Model

MMS - Concept of Virtual Manufacturing Device (VMD)



A virtual device represents a (complex) piece of equipment

Application-specific object models

MMS messages

A physical device (PLC) may implement one or more virtual devices

Virtual Manufacturing Device

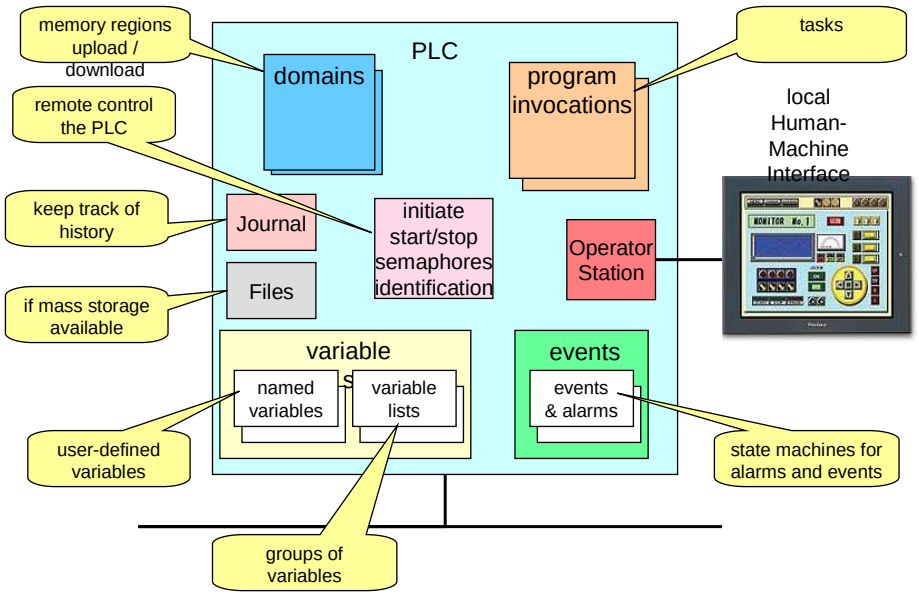
The VMD model defines:

- 1. Objects (e.g., variables) that are contained in the server.
- 2. The services that a client can use to access and manipulate these objects (e.g., read or write a
- variable).
- 3. The behavior of the server upon receipt of these service requests from clients.

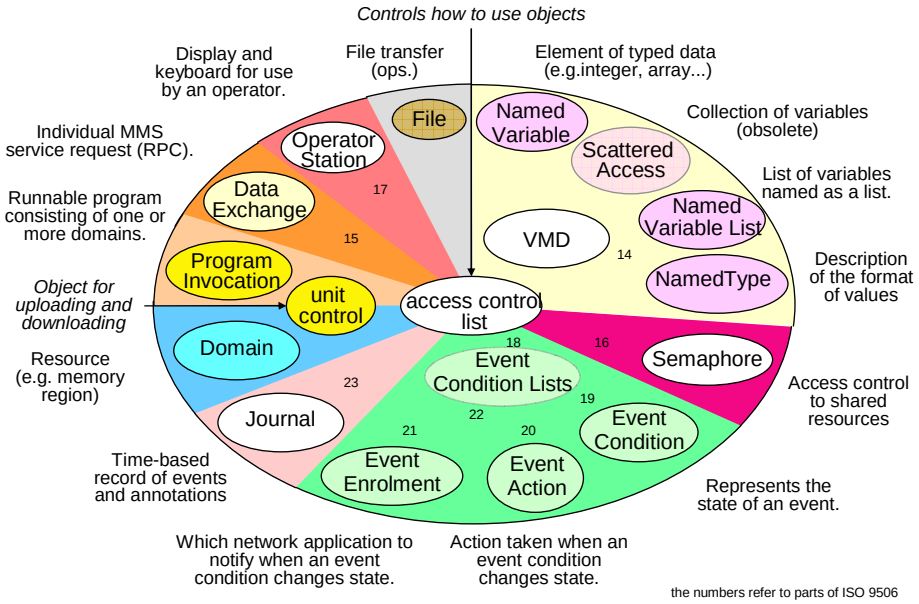
The VMD itself can be viewed as an object to which all other MMS objects are subordinate (variables, domains, etc. are contained within the VMD).

MMS provides services such as Status, UnsolicitedStatus, and Identify for obtaining information and status about the VMD. It also provides services like GetNameList and Rename for managing and obtaining information about objects defined in the VMD.

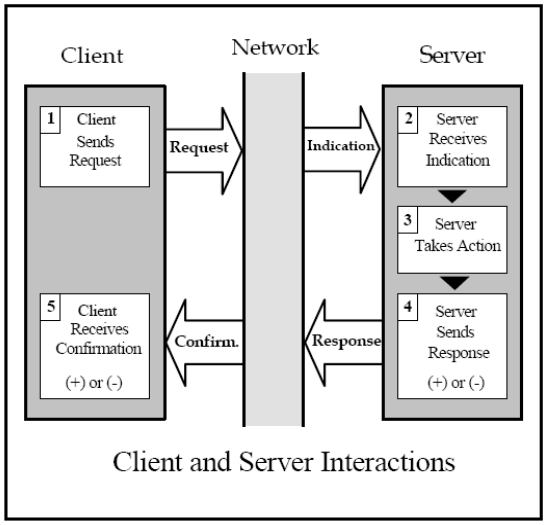
MMS - Objects in a PLC



MMS - Virtual Manufacturing Device (VMD) objects



Virtual Manufacturing Device



A key aspect of the VMD model is the client/server relationship between networked applications and/or devices.

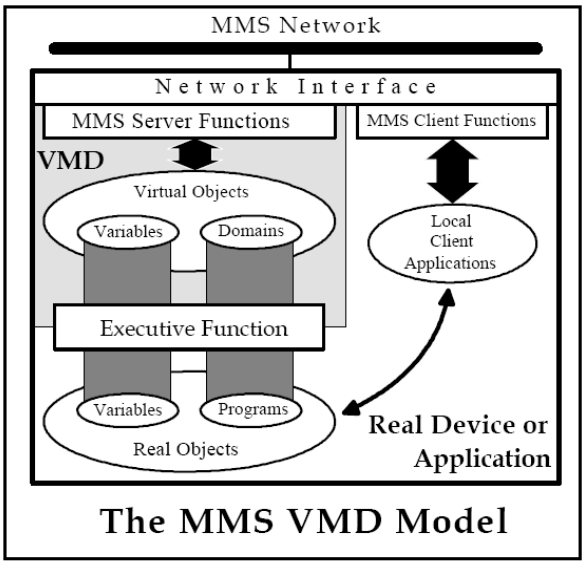
A server is a device or application that contains a VMD and its objects (e.g.,variables).

A client is a networked application (or device) that asks for data or an action from the server.

Real and "Virtual" Devices and Objects

- There is a distinction between a real device (e.g., a PLC) and the real objects contained in it (e.g.,variables) and the virtual device and objects defined by the VMD model.
- Real devices and objects **have peculiarities** (e.g. product features) associated with them that are unique to each brand of device or application.
- Virtual devices and objects conform to the VMD model and **are independent** of brand, language, operating system.
- Each developer of a MMS server device or MMS server application is responsible for "hiding" the details of their real devices and objects, by providing an executive function which translates the real devices and objects into the virtual ones defined by the VMD model.

Real and "Virtual" Devices and Objects



The MMS VMD Model

The executive function provides a *"mapping"* between the MMS defined virtual objects and the real device objects used by the real device. Applications local to the VMD, and the objects contained in them, are only accessible to a remote MMS client application if the executive function provides the mapping function for those objects and applications

Companion Standards

- In many cases, the relationship between the real objects and the virtual objects can be standardized for a particular class of device or application (e.g., a PLC). Developers and users of these real devices can define more precisely how MMS is applied to a particular class of device or application.
- The result is a ***companion standard*** *which* performs the following functions:
- Defines the mapping of real objects to the VMD model for a particular class of device.
- Provides additional definition of the behavioral characteristics of the VMD model for particular classes of devices.
- Defines additional network visible attributes for common objects.
- Defines additional objects and services that are unique to a particular class of device.

MMS – Object Name

All objects (except unnamed variables) are identified by an **object name**, that may be

- [0] VMD – specific: persistent, pre-loaded, all clients see the same
  "VMD status"
- [1] domain –specific: exists as long as the corresponding domain*
  "e.g. { "Domain1", "Valve3.Position"}
- [2] Application-Association specific: exists as long as the client remains connected, applies to non-persistent objects such as data sets that the client created
  "@/MyDataSet"

The identifier itself is a "visible string" (e.g. "Call.Robot1.Joint3.Pos").

Access to all objects can be controlled by a special object, the *Access Control List* that tells which client can delete or modify the object.

The service GetNameList retrieves the name and type of all named objects in the VMD.(this is the directory service)

* a domain is a (named) memory region that contains programs, variables, data

MMS - Data Types

MMS relies on the ASN.1 type (ISO 8823), but introduced new simple types:

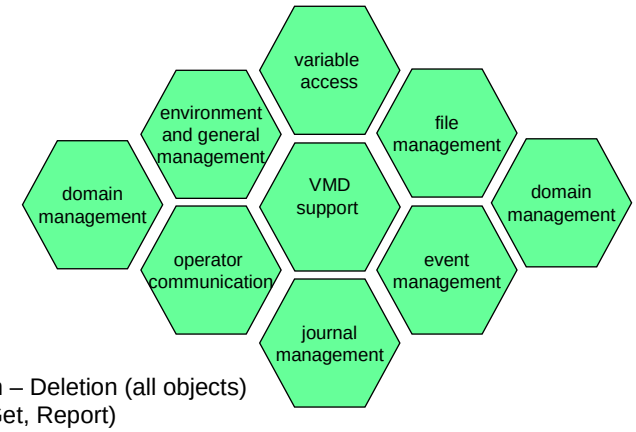
```
TimeOfDay ::= OCTET STRING (SIZE(4|6))        -- First four octets are the milliseconds
                                              since midnight for the current date.
Identifier ::= VisibleString                  -- up to 32 Uppercase and lowercase letters
                                              plus numbers, "$" and "_".
Integer8 ::= INTEGER(-128..127)               -- range -128 <= i <= 127
Integer16 ::= INTEGER(-32768..32767)          -- range -32,768 <= i <= 32,767
Integer32 ::= INTEGER(-2147483648..2147483647)   -- range -2**31 <= i <= 2**31 - 1
Unsigned8 ::= INTEGER(0..127)                 -- range 0 <= i <= 127
Unsigned16 ::= INTEGER(0..32767)              -- range 0 <= i <= 32767
Unsigned32 ::= INTEGER(0..2147483647)         -- range 0 <= i <= 2**31 – 1

FloatingPoint ::= OCTET STRING                -- according to IEEE 754 format

MMSString                                     Multilanguage string (VisibleString or ISO
                                              10646)
```

These types map directly to ASN.1 primitive types (they are a subrange of them), so there is no need to reserve additional primitive or constructed types for MMS.

MMS - Services (methods) on the objects



1. Creation – Deletion (all objects)
2. Read (Get, Report)
3. Modify (Alter)
4. Upload/Download (domains, files)
5. Operate (Start, Stop,…)

MMS - Initialisation

An MMS client establishes first an Association (connection) with an MMS Server

A server may sustain several simultaneous associations with different clients
(to synchronize access, MMS provides semaphores)

At initialisation time, the client lists the capabilities it expects and the server responds with the capabilities it offers.

The capabilities are defined by Conformance Building Block parameters.
e.g. cto ∈ CBB means that the server agreed to provide an Access Control List

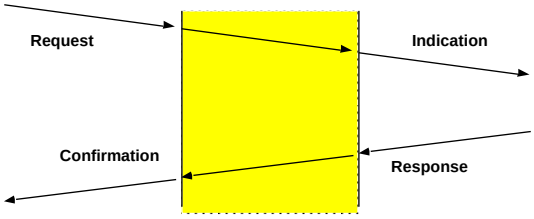
initialisation services:

Initiate	Status
ConcludeAbort	GetCapabilityList
Reject	GetNameList
Cancel	Rename
	Identify

MMS – Association establishment

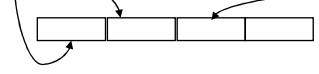
MMS Requester
(Requesting MMS-user)

MMS Responder
(Receiving MMS-user)



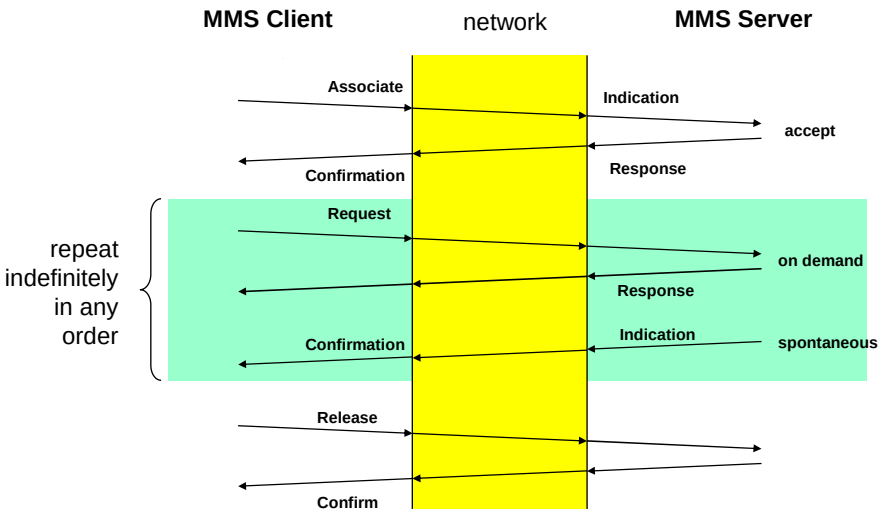

```
M_Associate_Req(
MMS_responder_address,
calling_application_reference,
called_application_reference,
communication_parameters,
authentication,..
```

```
M_Associate_Ind(
MMS_responder_address,
calling_application_reference,
called_application_reference,
communication_parameters,
authentication,..
```



this is no application interface, but a short way to describe the messages exchanged

MMS connection establishment and release



how the association is set up is beyond the scope of MMS (it depends on the stack)

MMS - Variables

Variables are the most important object type in MMS.

Through this service, a client can read and write local variables in a remote device.

Variables can be read or written as individual variables or better as lists.

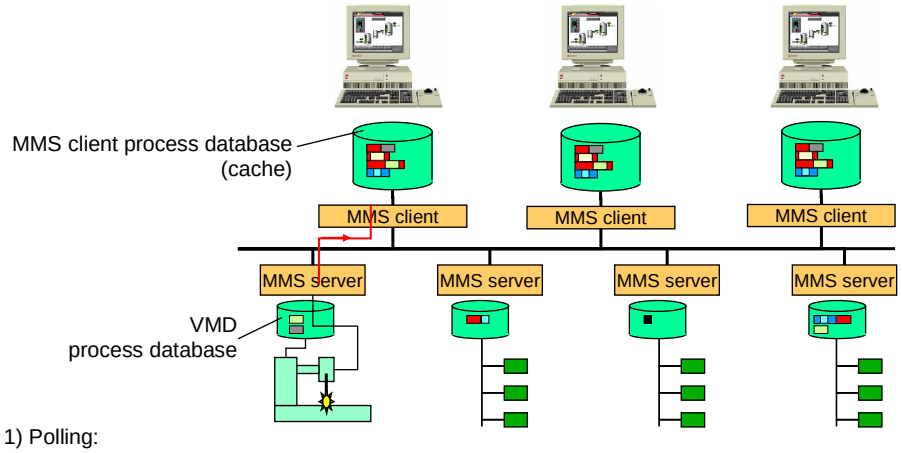
A Variable is characterized by:

- its Name

- its Data Type

- its attributes:
  - -MMS deletable (access to the variable can be deleted by the
  - -Access method (public,..)

MMS - Reading the variables



1) Polling:
   a) the bus scans periodically the variables and actualises the local databases
   b) the Operator Workstation polls cyclically the variables it is interested in
2) Events:
   a) the Controllers signal predefined events and broadcasts the corresponding values
   b) the Operator Workstation defines the relevant events and their destination(s)

MMS - Named and Unnamed Variables

Unnamed Variables
are identified by a **fixed physical address** in the VMD, expressed by either :

- numericAddress (an Unsigned32, e.g. 0xAF043BC0)
- symbolicAddress (a VisibleString, e.g. MW%1004)
- unconstrainedAddress (an OCTET STRING, e.g. 0x76AA)

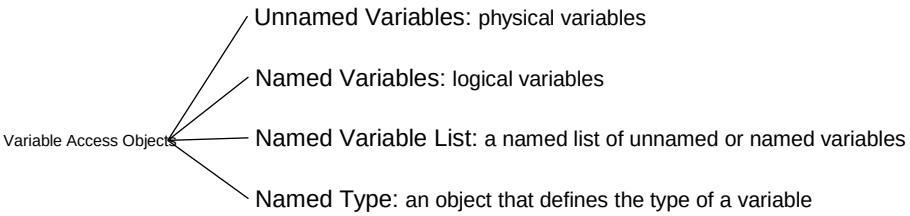
Named variables
are identified by an **object name:**
(a string of characters, VMD specific, domain specific or Association-specific)

MMS supports two ways of structuring the variables space:

1) use the identifier string, separated by "$" signs
(e.g. Cell4$Robot1$Motor3$TemperatureOil)

2) define a variable with a complex type

MMS Variables: individual or group access



Structure of the "Data" type

The values returned by an MMS Read are limited to the following types:

```
Data ::= CHOICE {
                        array       [1] IMPLICIT SEQUENCE OF Data, -- Nesting depth
                                    defined by nest ∈ CBB)
    structure           [2] IMPLICIT SEQUENCE OF Data, -- Possible if str1 ∈ CBB
    boolean             [3] IMPLICIT BOOLEAN,
    bit-string          [4] IMPLICIT BIT STRING,
    integer             [5] IMPLICIT INTEGER,
    unsigned            [6] IMPLICIT INTEGER,           -- Shall not be negative
    floating-point      [7] IMPLICIT FloatingPoint,
    real                [8] IMPLICIT REAL, -- obsolete
    octet-string        [9] IMPLICIT OCTET STRING,
    visible-string      [10] IMPLICIT VisibleString,
    generalized-time    [11] IMPLICIT GeneralizedTime,
    binary-time         [12] IMPLICIT TimeOfDay,
    bcd                 [13] IMPLICIT INTEGER,          -- Shall not be negative
    booleanArray        [14] IMPLICIT BIT STRING,
    objId               [15] IMPLICIT OBJECT IDENTIFIER,
    mMSString           [16] MMSString                  – Multilanguage string
    }
```

MMS Variable Lists

MMS provides services to build a Data Set, a group of variables that is to be transmitted as a whole.

This is generally done for each client specifically (Application-Association specific)

The client defines a list and populates it with the names of the variables and the transmission mode

MMS – Variable Access Services

Read	read a remote variable
Write	write a remote variable
InformationReport(optional)	spontaneous send the value to a client
GetVariableAccessAttributes	get the attributes of the variable
DefineNamedVariable	assigns named variable to an unnamed & type
DeleteVariableAccess	
DefineNamedVariableList	defines lists of variables
GetNamedVariableListAttributes	
(Read)	for individual variables or lists
(Write)	
(Information Report)	
DeleteNamedVariableList	
DefineNamedType	defines the types
GetNamedTypeAttributes	
DeleteNamedType	
DefineScatteredAccess	defines variables group treated as a whole
GetScatteredAccessAttributes	(obsolete, but useful)

MMS - Domains

•The MMS domain is a named MMS object that is a representation of some resource within the real device. This resource can be anything that is appropriately represented as a contiguous block of untyped data (referred to as *load data*).

•In many typical applications, domains are used to represent areas of memory in a device.

•Typically, a domain is loaded by segments of a size chosen by the receiver.

•When a domain is loaded, it may be saved to EPROM (typical PLC programming).

•Domains may be erased.

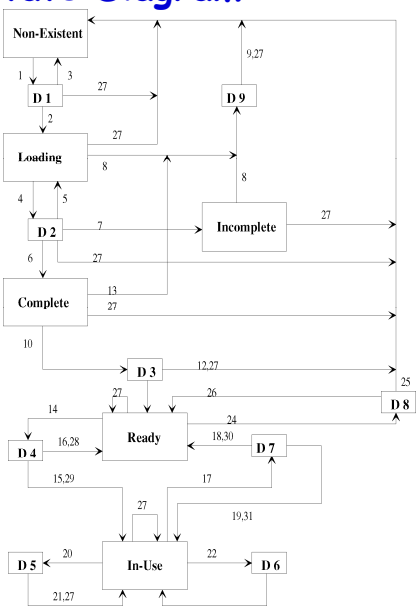
•Objects (Variables, Events, Program invocations,..) may be tied to a domain.

MMS - Domain State Diagram

Each domain is controlled by a state machine in MMS.

This is necessary since a domain is large and often needs to be loaded in several steps.

Also, it may be necessary to write the domain into a non-volatile memory and that needs a tighter control.



MMS - Operations on Domains

Operations on domains:

InitiateDownloadSequence	Download
DownloadSegment	
TerminateDownloadSequence	
RequestDomainDownload	
InitiateUploadSequence	Upload
UploadSegment	
TerminateUploadSequence	
RequestDomainUpload	
LoadDomainContent	
StoreDomainContent	e.g. to EPROM
DeleteDomain	erase
GetDomainAttributes	

Program Invocations

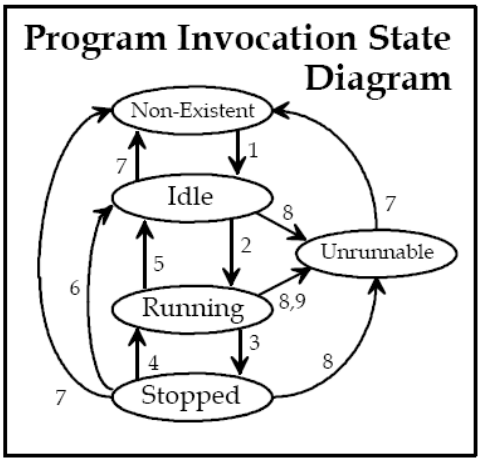
- It is through the manipulation of program invocations that a MMS client controls the execution of programs in a VMD.
- Program invocations can be started, stopped, or reset by MMS clients.
- A program invocation is an execution thread which consists of a collection of one or more domains.

```
CreateProgramInvocation
DeleteProgramInvocation
Start
Stop
Resume
Reset
Kill
GetProgramInvocationAttributes
Select
AlterProgramInvocationAttributes
ReconfigureProgramInvocation
```

Simple devices with simple execution structures may only support a single program invocation containing only one domain.

More sophisticated devices and applications may support multiple program invocations containing several domains.

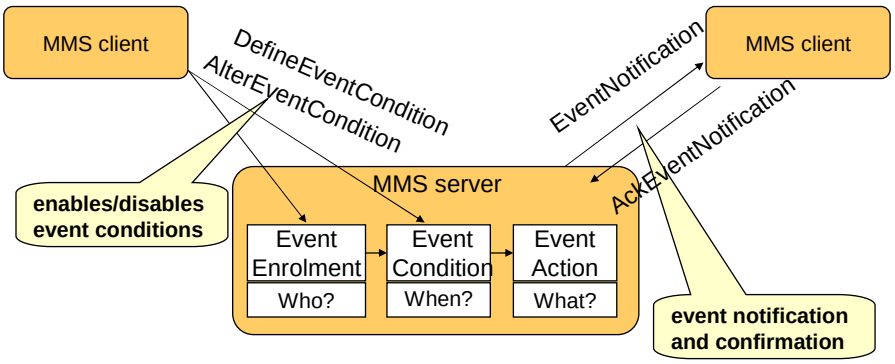
Program Invocations



MMS Clients use MMS services to cause state transitions in the program invocation as follows:
1. CreateProgramInvocation service request.
2. Start service request.
3. Stop service request or program stop.
4. Resume service request.
5. End of program and Reusable=TRUE.
6. Reset service request.
7. DeleteProgramInvocation service request.
8. Kill service request or error condition.
9. End of program and Reusable=FALSE.

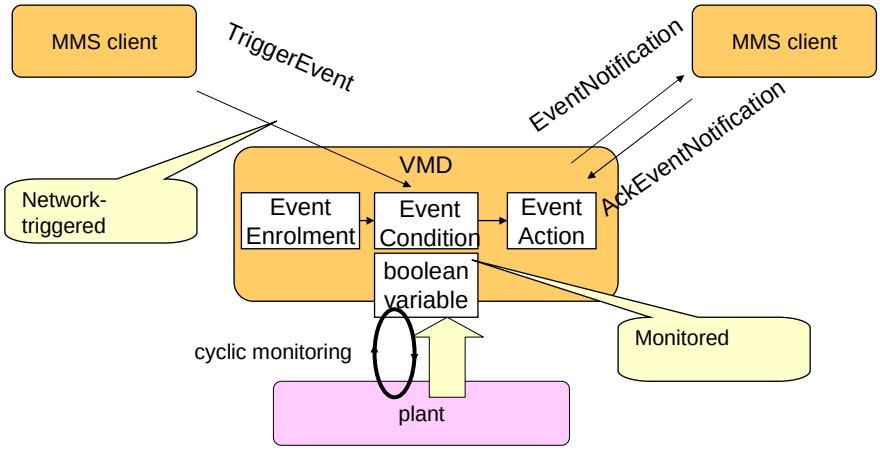
MMS - Event services

MMS provides services to:
- Event Condition (define the Boolean condition that triggers an event and its priority)
- Event Enrolment (define the MMS client(s) to notify when an event is triggered)
- Event Action (define the MMS confirmed service to be executed when the event occu



Events are the most complicated part of MMS

MMS - Event triggering



events are triggered by a change in a boolean variable in the server (monitored event) o
by an MMS client (trigger event) as an invitation procedure.

Event Services

The Event services are the most complicated part of MMS.
However, the event mechanism in a SCADA system is complex in nature.

Event Management

```
TriggerEvent
EventNotification
AcknowledgeEventNotification
GetAlarmSummary
GetAlarmEnrollmentSummary
```

Event Conditions

```
DefineEventCondition
DeleteEventCondition
GetEventConditionAttributes
ReportEventConditionStatus
AlterEventConditionMonitoring
```

Event Actions

```
DefineEventAction
DeleteEventAction
GetEventActionAttributes
ReportEventActionStatus
```

Event Conditions Lists

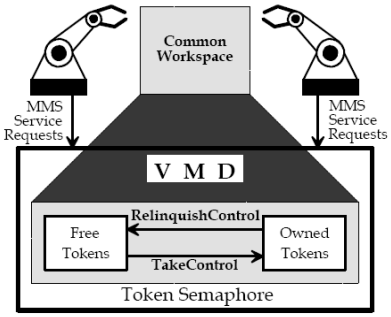
```
DefineEventConditionList
DeleteEventConditionList
AddEventConditionListReference
RemoveEventConditionListReference
GetEventConditionListAttributes
ReportEventConditionListStatus
AlterEventConditionListMonitoring
```

Event Enrollment

```
DefineEventEnrollment
DeleteEventEnrollment
GetEventEnrollmentAttributes
ReportEventEnrollmentStatus
AlterEventEnrollment service
```

The Semaphore Management Model

- In many real-time systems there is a need for a mechanism by which an application can control access to a system resource. An example might be a workspace that is physically accessible to several robots.



Token Semaphore Example

MMS defines two types of *semaphores* for these types of applications:

1) *Token Semaphores*

2) *Pool Semaphores*.

When a MMS client owns the token, it provides some level of access to the underlying resource.

Journals

- A MMS *journal* represents a log file that contains a collection of records (called *journal entries*) that are organized by time stamps.
- Journals are used to store time based records of tagged variable data, user generated comments, or a combination of events and tagged variable data.
- Journal entries contain a time stamp that indicates when the data in the entry was produced (not when the journal entry was made). This allows MMS journals to be used for applications where a sample of a manufactured product is taken at one time, analyzed in a laboratory off-line, and then at a later time placed into the journal.
- Each entry in a journal can be one of the following types: **Annotation, data, Event-data**
- The services available for MMS journals are as follows: **ReadJournal, WriteJournal, CreateJournal, DeleteJournal, InitializeJournal**.

Files

- MMS also provides a set of simple file transfer services for devices that have a local file store but do not support a full set of file services via some other means.
- For instance, many robot implementations of MMS use the file services for moving program (domain) files to the robot from a client application. The MMS file services support file transfer only, *not* file access.
- The services for files are:
- **FileOpen**
- **FileRead**
- **FileClose**
- **ObtainFile**
- **FileRename, FileDelete, FileDirectory**

MMS - Importance

MMS has been during its 15 years of existence a reference model for industry rather than an actual implementation.

Its high complexity makes it very general, but the requested bandwidth and computing power were out of reach until few years ago.

It is the base of the Utility Communication Architecture (UCA), an EPRI*-sponsored standardization of data exchange between control centers.

`http://www.epri.com/uca/iccp.html`

It is also the base of IEC 61850 **„Communication networks and systems in substations“**, which bases on TCP/IP/Ethernet

It gave rise to several other "simpler" models (DLMS, BacNet, FMS....)

For more information, see:

`http://lamspeople.epfl.ch/kirrmann/mms/`

`http://www.nettedautomation.com/qanda/mms/#OPC/MMS`

EPRI = USA electrical power research institute

Conclusion

Although MMS itself had little success (it is complicated), the concepts behind MMS have inspired numerous other standards.

Industrial Communication protocols require a large bandwidth and a lot of processing power at the servers, which is incompatible with low-cost, decentralized periphery.

While most field busses are able to connect relatively simple devices, the same is not true for MMS and its derivatives.