Overview and Introduction to the Manufacturing Message Specification (MMS)

Revision 2

Systems Integration Specialists Company, Inc.
6605 19½ Mile Road, Sterling Heights, MI 48314-1408, U.S.A.
Tel: (810) 254-0020, Fax: (810) 254-0053, BBS: (810)254-1578
E-Mial: support@sisconet.com, URL: http://www.sisconet.com

Printed in U.S.A.

OVERVIEW AND INTRODUCTION TO MMS

Revision 2

11/08/95

1. Overview of MMS

What is MMS?

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: 1) the application function being performed or 2) the developer of the device or application. MMS is an international standard (ISO 9506) that is developed and maintained by Technical Committee Number 184 (TC184), Industrial Automation, of the International Organization for Standardization (ISO).

The messaging services provided by MMS are generic enough to be appropriate for a wide variety of devices, applications, and industries. For instance, the MMS Read service allows an application or device to read a variable from another application or device. Whether the device is a **P**rogrammable **L**ogic **C**ontroller (PLC) or a robot, the MMS services and messages are identical. Similarly, applications as diverse as material handling, fault annunciation, energy management, electrical power distribution control, inventory control, and deep space antenna positioning in industries as varied as automotive, aerospace, petro-chemical, electric utility, office machinery, and space exploration have put MMS to useful work.

The History of MMS

In the early 1980s a group of numerical controller (NC) vendors, machine builders and users working under the auspices of committee IE31 of the **E**lectronic **I**ndustries **A**ssociation (EIA), had developed draft standard proposal #1393A titled "User Level Format and Protocol for Bidirectional Transfer of Digitally Encoded Information in a Manufacturing Environment". When the General Motors Corporation began its **M**anufacturing **A**utomation **P**rotocol (MAP) effort in 1980, they used the EIA-1393A draft standard proposal as the basis for a more generic messaging protocol that could be used for NCs, **p**rogrammable **l**ogic **c**ontrollers (PLC), robots and other intelligent devices commonly used in a manufacturing environments. The result was the **M**anufacturing **M**essage **F**ormat **S**tandard (MMFS). MMFS was used in the MAP Version 2 specifications published in 1984.

During the initial usage of MMFS, it became apparent that a more rigorous messaging standard was needed. MMFS allowed too many choices for device and application developers. This resulted in several mostly incompatible dialects of MMFS. Furthermore, MMFS did not provide sufficient functionality to be useful for the Process Control Systems (PCS) found in continuous processing industries. With the objective of developing a generic and non-industry specific messaging system for communications between intelligent manufacturing devices, the MMS effort was begun under the auspices of Technical Committee Number 184, Industrial Automation, of the International Organization for Standardization (ISO).

The result was a standard based upon the **O**pen **S**ystems **I**nterconnection (OSI) networking model called the **M**anufacturing **M**essage **S**pecification (MMS). A **D**raft **I**nternational **S**tandard (DIS) version of MMS was published in December 1986 as ISO DIS 9506.The DIS version of MMS (Version 0) overcame the problems with MMFS but had not yet been advanced to the status of an **I**nternational **S**tandard (IS). Faced with a publication deadline of November 1988, the MAP technical committees referenced the DIS version of MMS for the MAP V3.0 specification. In December 1988, the IS version of MMS (Version 1) was released as ISO 9506 parts 1 and 2. It was not until after the development of backwards compatibility agreements by the **N**ational **I**nstitute of **S**tandards and **T**echnology (NIST) that the IS version of MMS was referenced by the MAP V3.0 specifications[1].

[1] Backwards compatibility was a requirement for any changes to the MAP V3.0 specification due to the 6-year stability commitment made by the MAP Steering Committee in 1988.

The MMS Standard

The MMS standard (ISO 9506) is jointly managed by Technical Committee Number 184, Industrial Automation, of ISO and the International Electrotechnical Commission (IEC) and 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 **V**irtual **M**anufacturing **D**evice (VMD), 2) the services (or messages) exchanged between nodes on a network, and 3) the attributes and parameters associated with the VMD and services. Part 2 is the *protocol* specification. The protocol specification defines the rules of communication which includes 1) the sequencing of messages across the network, 2) the format (or encoding) of the messages, and 3) the interaction of the MMS layer with the other layers of the OSI model.

The protocol specification utilizes a presentation layer standard called the **A**bstract **S**yntax **N**otation Number One (ASN.1 **-** ISO 8824) to specify the format of the MMS messages.

MMS provides a rich set of services for peer-to-peer real-time communications over a network. MMS has been used as a communication protocol for many common industrial control devices like CNCs, PLCs, and robots. There are MMS applications in the electrical utility industry such as in **R**emote **T**erminal **U**nits (RTU), **E**nergy **M**anagement **S**ystems (EMS) and other **I**ntelligent **E**lectronic **D**evices (IED) like reclosers and switches. Most popular computing platforms have MMS connectivity available either from the computer manufacturer or via a third party. Some of the computer applications available include **A**pplication **P**rogramming **I**nterfaces (API), graphical monitoring systems, gateways, and drivers for spreadsheets, word processors, **A**pplication **E**nablers (A/Es) and **r**elational **d**ata **b**ase **m**anagement **s**ystems (RDBMS). MMS implementations support a variety of communications links including Ethernet, Token Bus, RS-232C, OSI, TCP/IP, MiniMAP, FAIS, and can connect to many more types of systems using networking bridges, routers, and gateways.

Benefits of MMS

MMS provides benefits by lowering the cost of building and using automated systems. In particular, 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. When looking at how the use of a common communications service like MMS can benefit a particular system, it is important to evaluate the three major effects of using MMS that can contribute to cost savings: 1) Interoperability, 2) Independence and 3) Access.

- **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. While many communication protocols can provide some level of interoperability, many of them are either too specific (to brand/type of application or device, network connectivity, or function performed **--** see Independence below) or not specific enough (provide too many choices for how a developer uses the network).

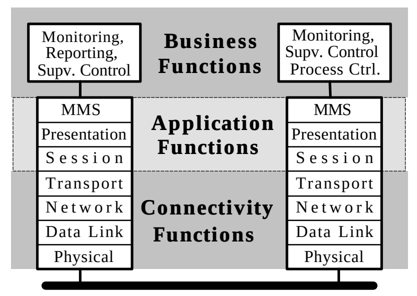
- **Independence** allows interoperability to be achieved independent of:
  - *The Developer of the Application*. Other communications schemes are usually specific to a particular brand (or even model in some cases) of application or device. MMS is defined by independent international standards bodies with participation from many leading industry experts and vendors.
  - *Network Connectivity*. MMS becomes **THE** interface to the network for applications, thereby isolating the application from most of the non-MMS aspects of the network and how the network transfers messages from one node to another.
  - *Function Performed*. MMS provides a common communications environment independent of the function performed. An inventory control application accesses production data contained in a control device in the *exact* same manner as an energy management system would read energy consumption data from the same device.
- **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 (see above).

MMS is rigorous enough to minimize the differences between applications performing similar or complimentary functions while still being generic enough for many different kinds of applications and devices. Communications schemes that are not specific enough can result in applications that all perform similar or complimentary functions in different ways. The result is applications that cannot communicate with each other because the developers all made different choices when implementing.

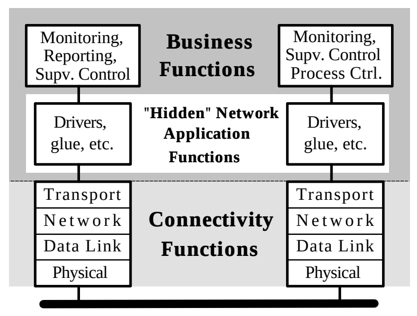
While many communications schemes only provide a mechanism for transmitting a sequence of bytes (a message) across a network, MMS does much more. MMS also provides definition, structure, and meaning to the messages that significantly enhances the likelihood of two independently developed applications 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 real challenge in trying to develop a business justification for MMS (or any network investment) is in assigning value to the benefits given a specific business goal. In order to do this properly, it is important to properly understand the relationship between the application functions, the connectivity functions, and the business functions of the network (see figure below). In some cases, the benefit of the common communications infrastructure MMS provides is only realized as the system is used, maintained, modified, and expanded over time. Therefore, a justification for such a system must look at *life cycle costs* versus just the purchase price.



The MMS View of Network Applications


Monitoring, Reporting, Supv. Control
Business Functions
Monitoring, Supv. Control Process Ctrl.
Drivers, glue, etc.
"Hidden" Network Application Functions
Transport
Network
Data Link
Physical
Connectivity Functions

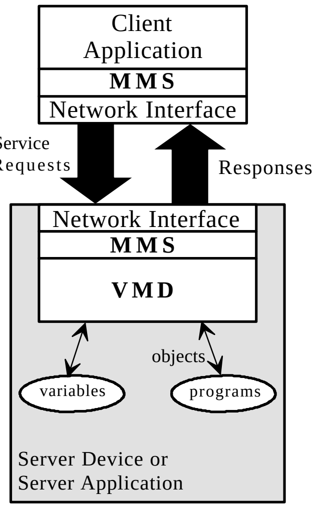
The Common View of Network Applications

In order to assign benefit to the use of MMS, it is important to first understand the value that MMS provides to applications. MMS, as an application layer protocol, provides *application* services to the business functions, not *connectivity* services. It is common to view the network 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. However, the costs are still there. Justifying MMS requires that the user recognize the value provided by the network application functions in facilitating interoperability, independence and access to data.

It is also important not to underestimate the cost associated with developing, maintaining, and expanding the application functions that have to be created if MMS is not used. A key element in assigning value is understanding that the business functions are what provide value to the enterprise. The cost of the custom network application functions directly reduces the amount of effort (i.e. cost) that can be placed on developing, maintaining, and expanding the business functions. With MMS, the communications infrastructure is built once and then re-used by all the business functions. Justifying MMS requires that the user recognize the value provided by the network application functions in facilitating interoperability, independence, and access to data.

MMS Application Services

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

The VMD model provides a consistent and well defined view to client applications of the objects contained in the VMD. Clients use MMS services to access and manipulate those objects. MMS requires that all servers behave according to the VMD model.

In general, 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 remainder of this overview of MMS will provide a summary of the objects defined by the VMD model and the MMS services provided to access and manipulate those objects. While the range of objects and services is broad, a given device or application need only implement whatever subset of these objects and services that are useful in that situation. A more detailed discussion of the MMS VMD model and the various MMS objects and their services will be presented following this overview.

VMD Object

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.

Variable and Type Objects

MMS provides a comprehensive and flexible framework for exchanging variable information over a network. The MMS variable access model includes capabilities for named, unnamed (addressed), and named lists of variables. MMS also allows the type description of the variables to be manipulated as a separate MMS object (named type object). MMS variables can be simple (e.g., integer, boolean, floating point, string) or complex (e.g., arrays and structures). The services available to access and manage MMS variable and type objects are very powerful and include services for:

- Read and Write services allow MMS client applications to access the contents of Named Variables, Unnamed Variables, and Named Variable List objects.
- The InformationReport service allows a server to report the contents of a variable to a remote client in an unsolicited manner.
- Define, delete, and get attribute services are available for both variables and types to allow clients to manage the variable access environment.
- Service options allow groups of variables to be accessed in a single MMS request, allow large arrays and complex structures to be partially accessed (alternate access), and allow clients to recast variable types and complex structures to suit their needs (access by description).

Program Control Objects (Domains and Program Invocations)

The VMD execution model defines two objects for controlling the execution of programs within the VMD. A MMS *domain* is defined as an object that represents a resource within the VMD (e.g., the memory in which a program is stored). A *program invocations* is defined as a group of domains, the execution of which can be controlled and monitored. Some of the features of the services the VMD execution model provides for MMS clients are:

- Services for commanding a VMD to upload/download their domains to/from a MMS client or file in a filestore system (either in the VMD or external to the VMD).
- Services for a VMD to request a domain upload/download from a client.
- Start, Stop, Reset, Resume, and Kill services for controlling the execution of program invocations.
- Services for deleting, creating, and obtaining the attributes of domains and program invocations.
- State changes in program invocations can be linked to MMS events.

Event Objects

The MMS event management model defines several named objects consisting of:

1. **event conditions:** An object that represents the state of an event (i.e., active or idle).
2. **event actions:** An object that consists of the action that should be taken by the VMD upon the occurrence of a change in state of the event condition.
3. **event enrollments:** An object that represents which MMS clients should be notified upon a change in state of an event condition.

The event management model provides a set of services for MMS clients:

- Services for notifying clients about events and for clients to acknowledge these notifications.

- Services for obtaining summaries of event conditions and event enrollments (called alarm summaries).
- Services for deleting, defining, obtaining the status and attributes of, and controlling event conditions, event actions, and event enrollments.

Semaphore Objects

MMS semaphores are named objects that can be used to control access to other resources and objects within the VMD. For instance, a VMD that controls access to a setpoint (a variable) for a control loop could use semaphores to only allow one client at a time to be able to change the setpoint (e.g., with the MMS Write service). The MMS semaphore model defines two kinds of semaphores. *Token semaphores* are used to represent a specific resource within the control of the VMD. *Pool semaphores* consist of one or more *named tokens* each representing a similar but distinct resource under the control of the VMD. MMS provides semaphore services which allow MMS clients to:

- Take control of and relinquish the control of semaphores.
- Define, delete, and get the attributes or status of semaphores.

Journal Objects

A MMS journal is a named object that represents a time based record, or log, of data. Each entry in a journal can contain the state of an event, the value of a variable, or character string data (called *annotation*) that the VMD, or a MMS client, enters into the journal. Services available allow journals to be created, read, deleted, and cleared (in whole or in part).

Operator Station Object

The operator station is an object that represents a means of communicating with the operator of the VMD via a keyboard and display. An Output service is available to display an alpha-numeric string on a text display. An Input service is available to obtain alpha-numeric keyboard input with and without a text prompt on the display.

Files

An annex of MMS that provides a simple set of services for transferring, renaming, and deleting files in a VMD. A FileDirectory service is provided to obtain a list of available files.

2. The Virtual Manufacturing Device (VMD) Model

The primary goal of MMS was to specify a standard communications mechanism for devices and computer applications that would achieve a high level of interoperability. In order to achieve this goal, it would be necessary for MMS to define much more than just the format of the messages to be exchanged. A common message format, or protocol, is ***only one*** aspect of achieving interoperability. In addition to protocol, the MMS standard also provides definitions for:

Objects. MMS defines a set of common objects (e.g., variables) and defines the network visible attributes of those objects (e.g., name, value, type).

Services. MMS defines a set of communications services (e.g., read, write) for accessing and managing these objects in a networked environment.

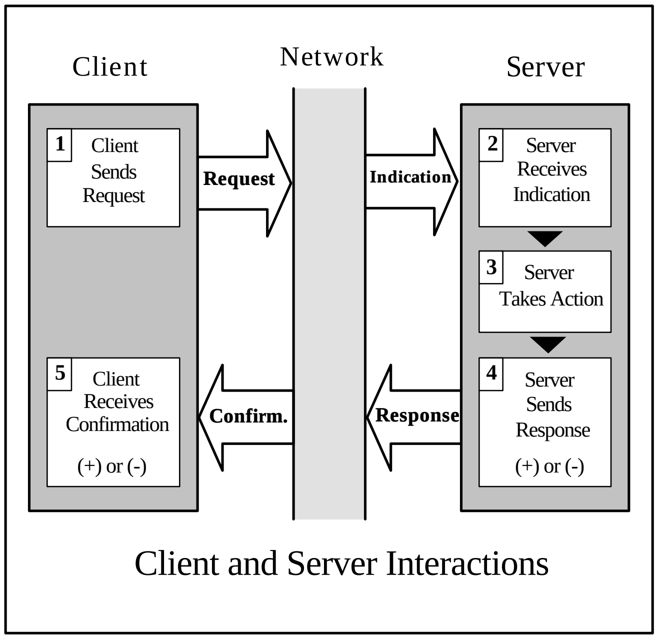
Behavior. MMS defines the network visible behavior that a device should exhibit when processing these services.

This definition of objects, services, and behavior comprises a comprehensive definition of how devices and applications communicate in which MMS calls the Virtual Manufacturing Device (VMD) model. The VMD model only specifies the *network visible* aspects of communications. The internal detail of how a real device implements the VMD model (i.e., the programming language, operating system, CPU type, input/output (I/O) systems) are not specified by MMS.By focusing only on the network visible aspects of a device, the VMD model is specific enough to provide a high level of interoperability. At the same time, the VMD model is still general enough to allow innovation in application/device implementation and making MMS suitable for applications across a range of industries and device types.

Client/Server Relationship

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. In a very general sense, a client is a network entity that issues MMS service requests to a server. A server is a network entity that responds to the MMS requests of a client (see figure below).While MMS defines the services for both clients and servers, the VMD model defines the network visible behavior of servers only.

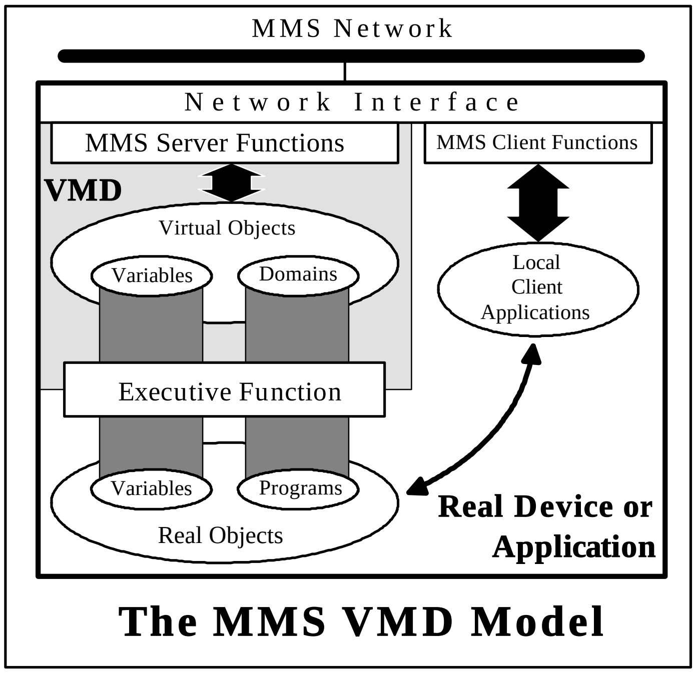
Many MMS applications and compatible devices provide both MMS client and server functions. The VMD model would only define the behavior of the server functions of those applications. Any MMS application or device that provides MMS server functions must follow the VMD model for all the network visible aspects of the server application or device. MMS clients are only required to conform to rules governing message format or construction and sequencing of messages (the protocol).


Client
Network
Server
1 Client Sends Request
Request
Indication
2 Server Receives Indication
3 Server Takes Action
4 Server Sends Response (+) or (-)
Response
Confirm.
5 Client Receives Confirmation (+) or (-)
Client and Server Interactions

Client/Server Interactions. MMS Clients and servers interact with each other by sending/receiving request, indication, response, and confirmation *Service Primitives* over the network. The diagram above depicts the interactions between a client and server for a MMS *Confirmed Service* where 1) the client sends a request, 2) the server receives an indication, 3) the server performs the desired action, 4) the server sends a positive (+) response if the action was successful or a negative (-) response if there was an error, and 5) the client receives the confirmation (+) or (-). An *Unconfirmed Service* is sent by the server and has only the request and indication service primitives (see InformationReport, and UnsolicitedStatus for examples of unconfirmed services)

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 (a.k.a. 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. The executive function translates the real devices and objects into the virtual ones defined by the VMD model when communicating with MMS client applications and devices.



Real and Virtual Objects. The executive function provides a translation, or *"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. Client applications local to the VMD may access and manipulate the real objects without using MMS.

Because MMS clients always interact with the virtual device and objects defined by the VMD model, the client applications are isolated from the specifics of the real devices and objects. A properly designed MMS client application can communicate with many different brands and types of devices in the same manner. This is because the details of the real devices and objects are hidden from the MMS client by the executive function in each VMD. This virtual approach to describing server behavior does not constrain the development of innovative devices and product features and improvements. The MMS VMD model places constraints only on the network visible aspects of the virtual devices and objects, not the real ones.

MMS Device and Object Modeling

The implementor of the executive function (the application or device developer) must decide how to "model" the real objects as virtual objects. The manner in which these objects are modeled is critical to achieving interoperability between clients and servers among many different developers. Inappropriate or incorrect modeling can lead to an implementation that is difficult to use or difficult to interoperate with.

For instance, take the situation of a PLC that contains a ladder program (a real object). The MMS implementor (the designer of the executive function) wishes to allow external client applications to copy the program from the PLC to another computer. For the purposes of this example, we will assume that the MMS VMD model gives the implementor the choice of modeling the ladder program object as a variable or domain object (both virtual objects). The choice of which virtual object to map to the real ladder program object is critical because MMS provides a set of services to manipulate variables that are quite different from the services used to manipulate domains. Variables can be read individually or in a list of typed data. Domains can be copied in whole only. If the ladder program is modeled as a MMS variable, it makes the task of performing a simple copying of the program complex. This is because the nature of the ladder program data (typically a large block of contiguous memory) would result in an extremely large variable that would be difficult to access easily. If the ladder program is modeled as a domain, there are specific MMS services provided for uploading and downloading the large blocks of untyped data typical of ladder programs. An incorrect modeling choice can make the real object difficult to access.

In some cases, it makes sense to represent the same real object with two different MMS objects. For instance, a large block of variables may also be modeled as a domain. This would provide the MMS client the choice of services to use when accessing the data. The variable services would give access to the individual elements in the block. The domain services would allow the entire block to be read/uploaded or written/downloaded as an element of a program invocation.

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 or PCS). 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*. In general, companion standards perform the following functions:

- Define the mapping of real objects to the VMD model for a particular class of device. For instance, the companion standard for process control systems would define the relationships between real objects such as setpoints, alarm limits and proportional-integral-derivative (PID) control loops to MMS domains, variables, and program invocations.
- Provide additional definition of the behavioral characteristics of the VMD model for particular classes of devices. For instance, the actions a PLC takes when receiving a MMS Start or Stop service request is very different from the actions taken by robots to these same service requests.
- Define additional network visible attributes for common objects. For example, the PLC companion standard defines additional status information for use in the MMS Status and UnsolicitedStatus services.
- Define additional objects and services that are unique to a particular class of device. The robot companion standard contains a definition of the VMD Stop service different from the core MMS Stop service.
- Define object naming and usage conventions for a particular class of device. For instance, standardized names and other naming conventions for common objects.

A companion standard, after proceeding through all the committees and work needed to become an ISO international standard, becomes a companion of the MMS standard as an additional part. The robot companion standard is ISO 9506 part 3, the NC companion standard will be ISO 9506 part 4, the PLC companion standard will be ISO 9506 part 5, and the PCS companion standard will be ISO 9506 part 6.

The reader should be aware that, for most systems, companion standards are not necessary even when using devices for which companion standards exist. The core MMS services defined in parts 1 and 2 of MMS provide sufficient standardization to achieve interoperability in most cases. Furthermore, aspects of the companion standards such as object naming, usage, and modeling conventions can be used in a core MMS application without having to implement the entire companion standard.

MMS Objects

MMS defines a variety of objects that are found in many typical devices and applications requiring real-time communications. A list of these objects is given below. For each object there are corresponding MMS services that let client applications access and manipulate those objects. Details for these object models and available services are only described for those objects supported by AX-S4 MMS.

- **VMD.** The device itself is an object.
- **Domain.** Represents a resource (e.g. a program) within the VMD.
- **Program Invocation.** A runnable program consisting of one or more domains.
- **Variable.** An element of typed data (e.g. integer, floating point, array, etc.).
- **Type.** A description of the format of a variable's data.
- **Named Variable List.** A list of variables that is named as a list.
- **Semaphore.** An object used to control access to a shared resource.
- **Operator Station.** A display and keyboard for use by an operator.
- **Event Condition.** An object that represents the state of an event.
- **Event Action.** Represents the action taken when an event condition changes state.
- **Event Enrollment.** Which network application to notify when an event condition changes state.
- **Journal.** A time based record of events and variables.
- **File.** A file in a filestore or fileserver.
- **Transaction.** Represents an individual MMS service request. Not a named object.

Object Attributes and Scope

Associated with each object are a set of attributes that describe that object. MMS objects have a name attribute and other attributes that vary from object to object. Variables have attributes such as, name, value, type. Other objects, program invocations for instance, have attributes like name and current state.

Subordinate objects exist only within the scope of another object. For instance, all other objects are subordinate to, or contained within, the VMD itself. Some objects, such as the operator station object, may be subordinate only to the VMD. Some objects may be contained within other objects, such as variables contained within a domain. This attribute of an object is called its *scope*. The object's scope also reflects the lifetime of an object. An object's scope may be defined to be:

- **VMD-Specific.** The object has meaning and exists across the entire VMD (is subordinate to the VMD). The object exists as long as the VMD exists.
- **Domain-Specific.** The object is defined to be subordinate to a particular domain. The object will exist only as long as the domain exists.
- **Application-Association-Specific.** Also referred to as AA-Specific. The object is defined by the client over a specific application association and can only be used by that specific client. The object exists as long as the association between the client and server exists.

The name of a MMS object must also reflect the scope of the object. For instance, the object name for a domain-specific variable must not only specify the name of the variable within that domain but also the name of the domain. Names of a given scope must be unique. For instance, the name of a variable specific to a given domain must be unique for all domain specific variables in that domain. Some objects, such as variables, are capable of being defined with any of the scopes described on the preceding page. Others, like semaphores for example, cannot be defined to be AA-specific. Still others, such as operator stations, are only defined as VMD-specific. When an object like a domain is deleted, all the objects subordinate to that domain must also be deleted.

The VMD Object

The VMD itself is also an object and has attributes associated with it. Some of the network visible attributes for a VMD are:

Capabilities

A capability of a VMD is a resource or capacity defined by the real device. There can be more than one capability to a VMD. The capabilities are represented by a sequence of character strings. The capabilities are defined by the implementor of the VMD and provides useful information about the real device or application.

Logical Status

Logical status refers to the status of the MMS communication system for the VMD which can be: STATE-CHANGES-ALLOWED, NO-STATE-CHANGES-ALLOWED or ONLY-SUPPORT-SERVICES-ALLOWED.

Physical Status

Physical status refers to the status of all the capabilities taken as a whole which can be equal to: OPERATIONAL, PARTIALLY-OPERATIONAL, INOPERABLE or NEEDS-COMMISSIONING.

VMD Support Services

Identify

This confirmed service allows the client to obtain information about the MMS implementation such as the vendor's name, model number, and revision level.

GetNameList

This confirmed service allows a client to obtain a list of named objects defined within the VMD.

GetCapabList

This service is used to obtain a list of capabilities of a VMD.

Rename

This service is used to rename a MMS object at the server.

Status

This confirmed service is used by a client to obtain the logical and physical status of the VMD. The Status and UnsolicitedStatus service also supports access to implementation specific status information (called *local detail*) defined by the implementor of the VMD.

UnsolicitedStatus

This unconfirmed service is used by a server (VMD) to report its status to a client unsolicited by the client.

3. The VMD Execution Model

The VMD model has a flexible execution model that provides a definition of how the execution of programs by the MMS server can be controlled. Central to this execution model are the definitions of the domain and program invocation objects.

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. For instance, a PLC's ladder program memory is typically represented as a domain. Some applications allow blocks of variable data to be represented as both domains and variables. MMS provides no definition for, and imposes no constraints on, the content of a domain. To do so would be equivalent to defining a "real" object (i.e., the ladder program). The content of the domain is left to the implementor of the VMD. In addition to the name, some of the attributes associated with MMS domains are:

Capabilities

Each domain can optionally have a list of capabilities associated with it that conveys information about memory allocation, input/output characteristics, and similar information about the real device. The capabilities of a domain are represented by a sequence of implementor defined character strings.

State

The state of a domain can be LOADING, COMPLETE, INCOMPLETE, READY, or IN-USE as well as other intermediate states.

Deletable

This attribute indicates whether the domain is deletable via the DeleteDomain service. A domain that can be downloaded is always deletable. Nondeletable domains are pre-existing and pre-defined by the VMD and cannot be downloaded.

Sharable

This attribute indicates if the domain can be shared by more than one program invocation (see the example batch controller on page 25).

MMS provides a set of services that allow domains to be uploaded from the device or downloaded to the device. The MMS domain services do not provide for partial uploads or downloads (except as potential error conditions). Nor do they provide access to any subordinate objects within the domain.

The set of services provided for domains is summarized below.

InitiateDownloadSequence, DownloadSegment, **TerminateDownloadSequence**

These services are used to download a domain. The InitiateDownloadSequence service commands the VMD to create a domain and prepare it to receive a download.

InitiateUploadSequence, UploadSegment, **TerminateUploadSequence**

These services are used to upload the contents of a domain to a MMS client.

DeleteDomain

This service is used by a client to delete an existing domain, usually before initiating a download sequence.

GetDomainAttributes

This service is used to obtain the attributes of a domain.

RequestDomainDownload, **RequestDomainUpload**

These services are used by a VMD to request that a client perform an upload or download of a domain in the VMD.

LoadDomainContent, StoreDomainContent

These services are used to tell a VMD to download (load) or upload (store) a domain from a file. The file may be local to the VMD or may be contained on an external file server.

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. 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.

As an example, consider how the MMS execution model could be applied to a personal computer (PC). When the PC powers up, it downloads a domain called the operating system into memory. When you type the name of the program you want to run and hit the <return> key, the computer downloads another domain (the executable program) from a file and then creates and runs a program invocation consisting of the program and the operating system. The program by itself cannot be executed until it is bound to the operating system by the act of creating the program invocation. In addition to the program invocation's name, the attributes of a program invocation are shown below.

State

The state of a program invocation can be NON-EXISTENT, IDLE, RUNNING, STOPPED, and UNRUNNABLE, as well as other intermediate states.

Deletable

Indicates if the program invocation is deletable via the DeleteProgramInvocation service.

Reusable

Reusable program invocations automatically reenter the IDLE state when the program invocation arrives at the end of the program. Otherwise, the program invocation in the RUNNING state must be Stopped then Reset in order to bring it back to the IDLE state.

Monitored

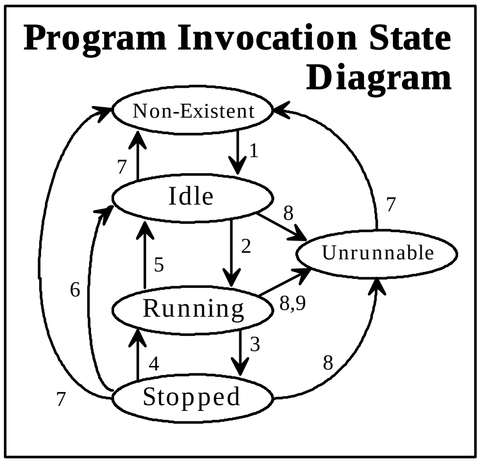
Monitored program invocations utilize the MMS event management services to inform the MMS client when the program invocation leaves the RUNNING state. Monitored program invocations have an event condition object defined with the same name as the program invocation.

List of Domains

The list of domains that comprise the program invocation.

Execution Argument

This is a character string passed to the program invocation using the Start or Resume service. The execution argument is used to pass data to the program invocation like parameters in a subroutine call.



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.

The MMS services for program invocations allow clients to control the execution of VMD programs and to manage program invocations as follows:

CreateProgramInvocation

This is used by a client to create a program invocation.

DeleteProgramInvocation

This service is used to delete a program invocation.

GetProgramInvocationAttributes

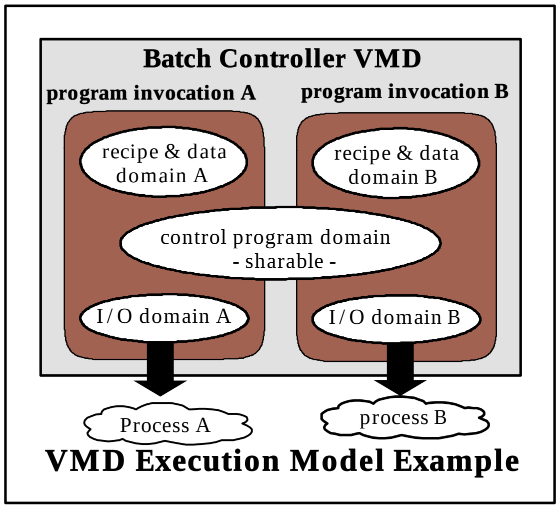
This service returns the attributes of the program invocation to the requesting client.

Start, Stop, Reset, Resume, Kill

These services are used by a client to cause the program invocation to change states (see diagram above).

Example Batch Controller

As an example of how the MMS execution model can be applied to a typical device, let us look at a VMD model for a simple batch controller. The diagram below depicts how the VMD model could be applied to define a set of objects (e.g., domains, program invocations, variables) appropriate for a batch controller. This model will provide clients an appropriate method of controlling the batch process using MMS services. In order to startup and control these two processes, a MMS client using this controller would perform the following actions:



Our example batch controller allows us to control two identical batch oriented processes. An I/O domain, specific to each process ties the data in the corresponding recipe/data domain to the process. The control program domain is sharable and contains the control algorithm. The MMS client can create a separate program invocation to control each process. This allows the client to control the recipe and perform supervisory control (start, stop, etc.) of both processes independently from each other.

- Initiate and complete a domain download sequence for each domain: recipe and data domains A and B, I/O domains A and B, and the control program domain.
- Create program invocation A consisting of I/O domain A, the control program domain, and recipe and data domain A.
- Start program invocation A.
- Create program invocation B consisting of I/O domain B, the control program domain, and recipe and data domain B.
- Start program invocation B.

The example above demonstrates the flexibility of the VMD execution model to accommodate a wide variety of real world situations. Further examples might be a loop controller where each loop is represented by a single domain and where the control loop algorithm (e.g., PID) is represented by a sharable separate domain. Process variables, setpoints, alarm thresholds, etc. could be represented by domain (control loop) specific variables. A program invocation would consist of the control loop domains and their algorithm domains needed to control the process.

4. The Variable Access Model

MMS Variables

A real variable is an element of *typed* data that is contained within a VMD. A MMS variable is a virtual object that represents a mechanism for MMS clients to access the real variable. The distinction between the real variable (which contains the value) and the virtual variable (which represents the access path to the variable) is important.

MMS defines two types of virtual objects for describing variable access:

- **Unnamed Variable Object.** An unnamed variable object describes the access to the real variable by using a device specific *address*. MMS includes unnamed variable objects primarily for compatibility with older devices that are not capable of supporting names. An unnamed variable object is a direct mapping to the underlying real variable that is located at the specified address. The attributes of the unnamed variable object are:

Address	This is the key attribute of the unnamed variable object.
MMS Deletable	This attribute is always FALSE for an unnamed variable object. Unnamed variable objects cannot be deleted because they are a direct representation of the "real" variable located at a specific address in the VMD.
Type Description	This attribute describes the type (format and range of values) of the variable.

- **Named Variable Object**. The named variable object describes the access to the real variable by using a MMS object name. MMS clients need only know the name of the object in order to access it. Remember that the name of a MMS variable also specifies the scope (see page 17) of the object. In addition to the name, a named variable object has the following attributes:

MMS Deletable	This attribute indicates if access to the variable can be deleted via the DeleteVariableAccess service.
Type Description	This attribute describes the type (format and range of values) of the variable.
Access Method	If the access method is PUBLIC, it means that the underlying address of the named variable object is visible to the MMS Client. In this case, the same variable can be accessed as an unnamed variable object.

Addresses

A MMS variable address can take on one of several forms. Which specific form that is used by a specific VMD, and the specific conventions used by that address form, is determined by the implementor of the VMD based upon what is most appropriate for that device.

The possible forms for a MMS variable address are:

- **Numeric.** A numeric address is represented by an unsigned integer number (e.g., 103).
- **Symbolic.** A symbolic address is presented by a character string (e.g., "R001" or "N7:0").
- **Unconstrained.** An unconstrained address is presented by a untyped string of bytes.

In general, it is recommended that applications utilize named variable objects instead of the addresses of unnamed variable objects wherever feasible. Address formats vary widely from device to device. Furthermore, in some computing environments, the addresses of variables can change from one run-time to the next. Names provide a more intuitive, descriptive, and device independent access method than addresses.

Named Variable Lists

MMS also defines a *named variable list* object that provides an access mechanism for grouping both named and unnamed variable objects into a single object for easier access. A named variable list is accessed by a MMS client by specifying the name (which also specifies its scope) of the named variable list. When the VMD receives the Read service request from a client, it reads all the individual objects in the list and returns their value within the individual elements of the named variable list.

Because the named variable list object contains independent subordinate objects, a positive confirmation to a Read request for a named variable list may indicate only partial success. The success/failure status of each individual element in the confirmation must be examined by the client to ensure that all of the underlying variable objects were accessed without error. In addition to its name and the list of underlying named and unnamed variable objects, named variable list objects also have a MMS deletable attribute that indicates whether or not the named variable list can be deleted via a DeleteNamedVariableList service request.

Named Type Object

The type of a variable indicates its format and the possible range of values that the variable can take. Examples of type descriptions include 16-bit signed integer, double precision floating point, 16 character string, etc. MMS allows the type of a variable to be either 1) *described* or 2) defined as a separate named object called a *named type*. A described type is not an object. It is a binary description of the type in a MMS service request (i.e., Read, Write, or InformationReport) that uses the Access by Description service option. The named type object allows the types of variables to be defined and managed separately. This can be particularly useful for systems that also use the DefineNamedVariable service to define names and types for unnamed variable objects. Other attributes of the named type object include:

MMS Deletable

This parameter indicates if the named type can be deleted using a DeleteNamedType service request.

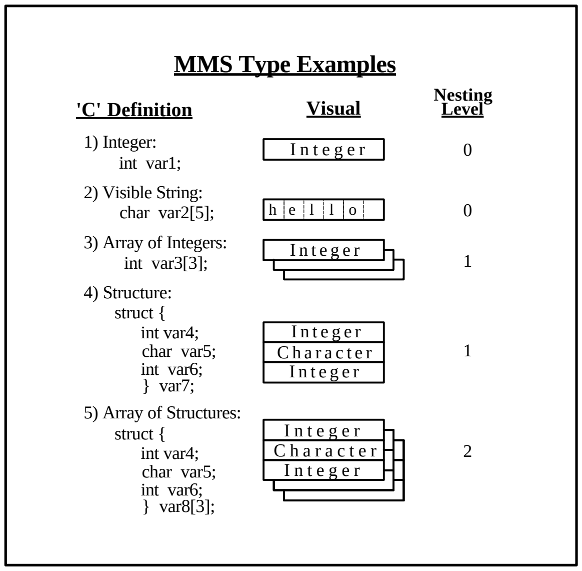
Type Description

This describes the type in terms of complexity (simple, array, or structured), format (e.g., integer, floating point), and the size or range of values (e.g., 16-bit, 8 characters)

Type Description

The specification for a MMS type description is very flexible and can describe virtually any data format in use today. The type description specifies the format of a variable and the range of values that the variable can represent. MMS defines three basic formats for types:

1) *simple*, 2) *array* and 3) *structured*:



Simple. Simple types are the most basic types and cannot be broken down into a smaller unit via MMS. The other type forms (arrays and structures) are constructed types that can eventually be broken down into simple types. Simple type descriptions generally consist of the *class* and *size* of the type. The size parameter is usually defined in terms of the number of bits or bytes that a variable of that type would comprise in memory.

The various classes of simple types defined by MMS consists of:

- BOOLEAN. Booleans are generally represented as a single byte with either a zero (FALSE) or non-zero (TRUE) value. There is no size parameter for Boolean types.
- BIT STRING. A Bit String is a sequence of bits. The size of a Bit String indicates the number of bits in the Bit String.
- BOOLEAN ARRAY. A Boolean Array is also a sequence of bits where each bit represents TRUE (1) or FALSE (0). It differs from an array of Booleans in that each element in a Boolean Array is represented by a single bit while each element in an array of Booleans is represented by a single byte. The size parameter specifies the number of Booleans (number of bits) in the Boolean Array.
- INTEGER. MMS Integer's are signed integers. The size parameter specifies the number of bits of the integer in 2's complement form.
- UNSIGNED. The Unsigned type is identical to the Integer type except that it is not allowed to take on a negative value. Because the most significant bit of an Integer is essentially a sign bit, an Unsigned with a size of 16 bits can only represent 15-bits of values or values of 0 through 32,767.
- FLOATING POINT. The MMS definition for floating point is modeled after the IEEE 754 standard but can accommodate any number of bits for the format and exponent width including the single and double precision floating point formats commonly in use today.

  - REAL. This type is a representation of floating point numbers conforming to the ISO 8824 standard.
  - OCTET STRING. An Octet String is a sequence of bytes (called an "octet" in ISO terminology) with no constraint on the value of the individual bytes. The size of an Octet String is the number of bytes in the string.
  - VISIBLE STRING. The Visible String type only allows each byte to contain a printable character. Although these character sets are defined by ISO 10646, they are compatible with the common ASCII character set. The size of the Visible String is the number of bytes in the string.
  - GENERALIZED TIME. This is a representation of time specified by ISO 8824. It provides millisecond resolution of date and time.
  - BINARY TIME. This is a time format specified by MMS and represents either 1) time of day by a value which is equal to the number of milliseconds from midnight or 2) date and time by a value that is equal to the time of day and the number of days since January 1, 1984.
  - BCD. Binary Coded Decimal format is where four-bits are used to hold a binary value of a single digit of zero to ten. The size parameter is the number of decimal digits that the BCD value can represent.
  - OBJECT IDENTIFIER. This is a special class of object defined by ISO 8824 that is used to define network objects.

- **Array.** An array type defines a variable that consists of a sequence of multiple identical (in format not value) elements. Each element in an array can also be an array or even a structured or simple variable. MMS allows for arbitrarily complex nesting of arrays and structures.
  The level of complexity that a VMD can support is defined by its *nesting level*. An array of simple variables (e.g., an array of Integers) requires a nesting level of 1. An array of an array or an array of structured variables, each consisting of simple variables, requires a nesting level of two.

- **Structures.** A structured type defines a variable that consists of a sequence of multiple, but not necessarily identical, elements. Each individual element in a structure can be of a simple type, an array, or another structure. MMS allows for arbitrarily complex nesting of structures and arrays. A structured variable consisting of individual simple elements requires a nesting level of 1. A structured variable consisting of one or more arrays of structures containing simple variables requires a nesting level of 3.

Variable Access Services

Read

The Read service is used by MMS clients to obtain the values of named, unnamed, or a named variable list objects (hereinafter "Variables").

Write

The Write service is used by MMS clients to change the values of Variables.

InformationReport

This service is used by a VMD to report the values of Variables to a MMS client in an unsolicited manner. It is roughly equivalent to sending a Read response to a client without the client having issued the Read request. The InformationReport service can be used to eliminate polling by client applications or as an alarm notification mechanism. The VMD could directly report changes in the value of Variables, alarm conditions, or even changes in state of the VMD or program invocations to clients by using the InformationReport service.

GetVariableAccessAttributes

This service is used by MMS clients to obtain the access attributes of a single named or unnamed variable object. The access attributes are the type (either a named type or type description), the MMS deletable attribute, and the address for unnamed variables. The address of named variables is optional and is only returned if the address is known and visible.

DefineNamedVariable

This service allows MMS clients to create a named variable object corresponding to a real VMD variable that can be represented by an unnamed variable object. Once defined, subsequent access to the unnamed variable object can be made via the named variable. This service also allows the MMS client to optionally specify a type definition for the named object that may be different from the type inherently defined for the unnamed variable object.

DeleteVariableAccess

This service is used to delete named variable objects where the MMS deletable attribute is TRUE. The service provides options for deleting only specific named variable objects or all named variable objects of a specified scope (VMD, domain, or AA-specific).

DefineNamedVariableList
GetNamedVariableListAttributes
DeleteNamedVariableList

These services are used to create, delete, and obtain the attributes (i.e. the list of underlying named and unnamed variable objects) of named variable list objects.

DefineNamedType

This service is used by a MMS client to create a new named type by specifying the type name (including scope), MMS deletable and type description attributes.

DeleteNamedType

This service is used to delete an existing named type where the MMS deletable attribute is TRUE. The service provides options for deleting only specific named type objects or all named type objects of a specified scope (VMD, domain, or AA-specific).

GetNamedTypeAttributes

This service is used by a MMS client to determine the MMS deletable and type description attributes of a specific named type object.

Variable Access Service Features

The Read, Write, and InformationReport services provide several features for accessing Variables. The use of these service features, as described below, by MMS clients can provide enhanced performance and very flexible access to MMS Variables.

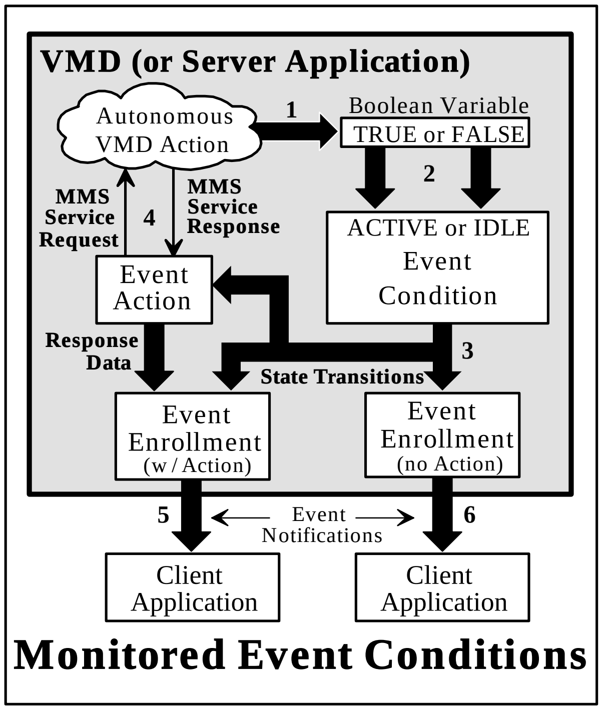
- **Access by Description** is supported for unnamed variable objects only. It allows the MMS client to describe the variable by specifying both an address **and** a type description. These variables are called *described variables*. The described type may be different from the type inherently defined for the unnamed variable object. This can be useful for accessing data in devices where the device's memory organization is simplistic. For example, many PLCs represent their data memory as a large block of 16-bit registers (essentially signed integers). Some applications may store ASCII string data in these registers. By using a described variable, a MMS client can have the data stored in these registers returned to it as a string instead of as a block of signed integers.

- **List of Variables** is a function that allows a list of named variable, unnamed variable, and named variable list objects to be accessed in a single MMS Read, Write, or InformationReport service. Care must be taken by the client to ensure that the resultant MMS service request message does not exceed the maximum message size (or *maximum segment size*) supported by the VMD. This option also requires that a client examine the entire response for success/failure for each individual element in the list of Variables.
- **Access Specification in Result** is an option for the Read service that allows a MMS client to request that the variable's access specification be returned in the Read response. The access specification would consist of the same information that would be returned by a GetVariableAccessAttributes service request.
- **Alternate Access** allows a MMS client to 1) partially access only specified elements contained in a larger arrayed and/or structured variable, and 2) rearrange the ordering of the elements contained in structured variables.

5. The Event Management Model

In a real sense, an event or an alarm is easy to define. Most people have an intuitive feel for what can comprise an event within their own area of expertise. For instance, in a process control application, it is common for a control system to generate an alarm when the process variable (e.g., temperature) exceeds a certain preset limit called the high alarm threshold. In a power distribution application, an alarm might be generated when the difference in the phase angle of the current and voltage waveforms of a power line exceeds a certain number of degrees.

The MMS event management model provides a framework for accessing and managing the network communication aspects of these kinds of events. This is accomplished by defining three named objects that represent 1) the state of an event (*event condition*), 2) who to notify about the occurrence of an event (*event enrollment*) and 3) the action that the VMD should take upon the occurrence of an event (*event action*).



A Monitored event condition has a Boolean variable associated with it that the VMD sets (1) via some form of local autonomous action. The VMD periodically evaluates this variable (2) to determine the state of the event condition. MMS Clients "enroll" to be notified of event condition state transitions (3) by defining an event enrollment. If an event action is defined for the event enrollment, the VMD obtains the response (4) to the event action (a MMS service request) and inserts the response data in the event notification (5) that is sent to the client. Event enrollments without an event action have their event notifications sent to the client (6) without any event action response data.

For many applications, the communication of alarms can be implemented by using MMS services other than the event management services. For instance, a simple system can notify a MMS client about the fact that a process variable has exceeded some preset limit by sending the process variable's value to a MMS client using the InformationReport service. Other schemes using other MMS services are also possible. When the application is more complex and requires a more rigorous definition of the event environment in order to ensure interoperability, the MMS event management model should be used.

Event Condition Object

A MMS event condition object is a named object that represents the current state of some real condition within the VMD. It is important to note that MMS does not define the VMD action (or programming) that causes a change in state of the event condition. In the process control example given above, an event condition might reflect an IDLE state when the process variable was not exceeding the value of the high alarm threshold and an ACTIVE state when the process variable did exceed the limit. MMS does not explicitly define the mapping between the high alarm limit and the state of the event condition. Even if the high alarm limit is represented by a MMS variable, MMS does not define the necessary configuration or programming needed to create the mapping between the high alarm limit and the state of the event condition.

From the MMS point of view, the change in state of the event condition is caused by some autonomous action on the part of the VMD that is not defined by MMS. The MMS event management model defines two classes of event conditions:

- **Network Triggered.** A network triggered event condition is triggered when a MMS client specifically triggers it using the TriggerEvent service request. Network triggered events do not have a state (their state is always DISABLED). They are useful for allowing a MMS client to control the execution of event actions and the notifications of event enrollments.
- **Monitored.** A monitored event condition has a state attribute that the VMD sets based upon some local autonomous action. Monitored event conditions can have a Boolean variable associated with them that is used by the VMD to evaluate the state. The VMD periodically evaluates this variable. If the variable is evaluated as TRUE, the VMD sets the event condition state to ACTIVE. When the Boolean variable is evaluated as FALSE, the VMD sets the event condition state to IDLE. Event conditions that are created as a result of a CreateProgramInvocation request with the Monitored attribute TRUE, are monitored event conditions but they do not have an associated Boolean variable.

In addition to the name of the event condition (an object name that also reflects its scope) and its class (Network Triggered or Monitored), MMS defines the following attributes for both network triggered **and** monitored event conditions:

MMS Deletable

This attribute indicates if the event condition can be deleted by using a DeleteEventCondition service request.

State

This attribute reflects the state of the event condition and can be IDLE, ACTIVE, or DISABLED. Network triggered events are always DISABLED.

Priority

This attribute reflects the relative priority of an event condition object with respect to all other defined event condition objects. Priority is a relative measure of the VMD's processing priority when it comes to evaluating the state of the event condition as well as the processing of event notification procedures that are invoked when the event condition changes state. A value of zero (0) is the highest priority. A value of 127 is the lowest priority. A value of 64 is the "normal" priority.

Severity

This attribute reflects the relative severity of an event condition object with respect to all other defined event condition objects. Severity is a relative measure of the effect that a change in state of the event condition can have on the VMD. A value of zero (0) is the highest severity. A value of 127 is the lowest. A value of 64 is for event conditions with "normal" severity.

Additionally, MMS also defines the following attributes for *Monitored* event conditions only:

Monitored Variable

This is a reference to the underlying Boolean variable whose value the VMD evaluates in determining the state of an event condition. It can be either a named or unnamed variable object. If it is a named object it must be a variable with the same name (and scope) of the event condition. If the event condition object is locally defined or it was defined via a CreateProgramInvocation request with the monitored attribute TRUE, then the value of the monitored variable reference would be equal to UNSPECIFIED. If the monitored variable is deleted, then the value of this reference would be UNDEFINED and the VMD should disable its event notification procedures for this event condition.

Enabled

This attribute reflects whether a change in the value of the monitored variable (or the state of the associated program invocation if applicable) should cause the VMD to process the event notification procedures for the event condition (TRUE) or not (FALSE). A client can disable an event condition by changing this attribute via an AlterEventConditionMonitoring service request.

Alarm Summary Reports

This attribute indicates whether (TRUE) or not (FALSE) the event condition should be included in alarm summary reports in response to a GetAlarmSummaryReport service request.

Evaluation Interval

This attribute specifies the maximum amount of time, in milliseconds, between successive evaluations of the event condition by the VMD. The VMD may optionally allow clients to change the evaluation interval.

Time of Last Transition to Active
Time of Last Transition to Idle

These attributes contain either the time of day or a time sequence number of the last state transitions of the event condition. If the event condition has never been in the IDLE or ACTIVE state, then the value of the corresponding attribute shall be UNDEFINED.

Event Condition Services

DefineEventCondition, DeleteEventCondition, GetEventConditionAttributes

These services are used by MMS clients to create event condition objects, to delete event condition objects (if the MMS Deletable attribute is TRUE), and to obtain the static attributes of an existing event condition object respectively.

ReportEventConditionStatus

This service allows a MMS client to obtain the dynamic status of the event condition object including its state, the number of event enrollments enrolled in the event condition, whether it is enabled or disabled, and the time of the last transitions to the active and idle states.

AlterEventConditionMonitoring

This service allows the MMS client to alter the priority, enable or disable the event condition, enable or disable alarm summary reports, and change the evaluation interval if the VMD allows the evaluation interval to be changed.

GetAlarmSummary

This service allows a MMS client to obtain event condition status and attribute information about groups of event conditions. The client can specify several filters for determining which event conditions to include in an alarm summary.

Event Actions

An event action is a named MMS object that represents the action that the VMD will take when the state of an event condition changes. An event action is optional. When omitted, the VMD would execute its event notification procedures without processing an event action. An event action, when used, is always defined as a confirmed MMS service request. The event action is attached or linked with an event condition when an event enrollment is defined. For example, an event action might be a MMS Read request. If this event action is attached to an event condition (by being referenced in an event enrollment), when the event condition changes state and is enabled, the VMD would execute this Read service request just as if it had been received from a client.

Except that the Read response (either positive or negative) is included in the EventNotification service request that is sent to the MMS client, defined for the event enrollment. A *confirmed* service request must be used (i.e., Start, Stop, Read). Unconfirmed services (e.g., InformationReport, UnsolicitedStatus, and EventNotification) and services that must be used in conjunction with other services (e.g., domain upload-download sequences), cannot be used as event actions. In addition to its name, an event action has the following attributes:

MMS Deletable

When TRUE, it indicates that the event action can be deleted via a DeleteEventAction service request.

Service Request

This attribute is the MMS confirmed service request that the VMD will process when the event condition that the event action is linked with changes state.

Event Action Services

DefineEventAction, DeleteEventAction, GetEventActionAttributes

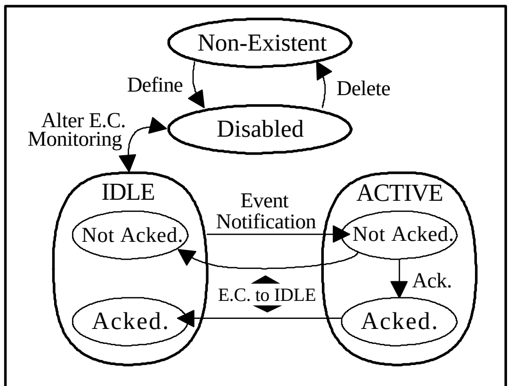
These services are used by MMS clients to create, delete (if the MMS Deletable attribute is TRUE), and to obtain the attributes of an event action object respectively.

ReportEventActionStatus

This service allows a MMS client to obtain a list of names of event enrollments that have referenced a given event action.

Event Enrollments

The event enrollment is a named MMS object that ties all the elements of the MMS event management model together. The event enrollment represents a request on the part of a MMS client to be notified about changes in state of an event condition.



Simplified Event Enrollment States

Event enrollments have major states (IDLE, ACTIVE) that reflect the event condition (E.C.). Minor states reflect the status of the event acknowledgment process.

When an event enrollment is defined, references are made to an event condition, an event action (optionally), and the MMS client to which EventNotification should be sent. In addition to its name, the attributes of an event enrollment are:

MMS Deletable

If TRUE, this attribute indicates that the event enrollment can be deleted via a DeleteEventEnrollment service request.

Event Condition

This attribute is the name of the event condition about which the event enrollment will be notified of changes in state.

Transitions

This attribute indicates the state transitions of the event condition for which the VMD should execute its event notification procedures as follows:

- DISABLED-TO-ACTIVE
- DISABLED-TO-IDLE
- IDLE-TO-ACTIVE
- IDLE-TO-DISABLED
- ACTIVE-TO-IDLE
- ACTIVE-TO-DISABLED
- ANY-TO-DELETED

Notification Lost

If this attribute is TRUE, it means that the VMD could not successfully complete its event notification procedures due either to 1) some local resource constraint or problem or 2) because the VMD could not establish an association to the client specified in the event enrollment definition. Further transitions of the event condition will be ignored for this event enrollment as long as these problems persist.

Event Action

This optional attribute is a reference to the event action that should be processed by the VMD for those state transitions of the event condition specified by the event enrollment's transitions attribute.

Client Application

This attribute is a reference to the MMS client to which the EventNotification service requests should be sent for those transitions of the event condition specified by the event enrollment's transitions attribute. This attribute should only be defined if the VMD supports third party services. This attribute is omitted if the duration of the event enrollment is CURRENT.

Duration

This attribute indicates the lifetime of the event enrollment. A duration of CURRENT means the event enrollment is only defined for the life of the association between the MMS client and the VMD (similar to an object with application association specific scope). If the association between the VMD and the client is lost and there is no client application reference attribute for the event enrollment (client application reference is omitted for event enrollments with duration = CURRENT), then the VMD is not capable of re-establishing an application association in order to send an EventNotification to the client. If the duration of the event enrollment is PERMANENT, then the application association between the VMD and the client can be terminated without affecting the event enrollment. In this case, when a specified state transition occurs, the VMD will automatically establish an application association with the specified client.

State

The state of an event enrollment indicates IDLE, ACTIVE, DISABLED, and a variety of other states that represent the status of the event notification procedures being executed by the VMD.

Alarm Acknowledgment Rules

This attribute specifies the rules of alarm acknowledgment that the VMD should enforce when determining the state of the event enrollment. If an acknowledgment to an EventNotification service request is required, the act of acknowledging (or not acknowledging) the Event Notification shall affect the state of the event enrollment.The various alarm acknowledgment rules are summarized as follows:

- NONE. No acknowledgments are required and they will not affect the state of an event enrollment. These types of event enrollments are not included in alarm enrollment summaries.
- SIMPLE. Acknowledgment will not be required, but acknowledgments of transitions to the ACTIVE state will affect the state of the event enrollment.
- ACK-ACTIVE. Acknowledgment of event condition transitions to the ACTIVE state will be required and will affect the state of the event enrollment. Acknowledgments of other transitions are optional and will not affect the state of the event enrollment.
- ACK-ALL. Acknowledgments are required for all transitions of the event condition to the ACTIVE or IDLE state and will affect the state of the event enrollment.

Time Active Acked, Time Idle Acked

These attributes reflect the time of the last acknowledgment of the Event Notification for state transitions in the event condition to the ACTIVE or IDLE state corresponding to the event enrollment.

Event Enrollment Services

DefineEventEnrollment, DeleteEventEnrollment, GetEventEnrollmentAttributes

These services are used by MMS clients to create, delete (if the MMS Deletable attribute is TRUE), and to obtain the static attributes of an event enrollment object.

ReportEventEnrollmentStatus

This service allows the client to obtain the dynamic attributes of an event enrollment including the notification lost, duration, alarm acknowledgment rule, and state attributes.

AlterEventEnrollment

This service allows the client to alter the transitions and alarm acknowledgment rule attributes of an event enrollment.

GetAlarmEnrollmentSummary

This service allows a MMS client to obtain event enrollment and event condition information about groups of event enrollments. The client can specify several filters for determining which event enrollments to include in an alarm enrollment summary.

Event Notification Services

MMS provides services for notifying clients of event condition transitions and acknowledging those event notifications as follows:

EventNotification

This is an *unconfirmed* service that is issued by the *VMD* to the MMS client to notify the client about event condition transitions that were specified in an event enrollment. There is no response from the client. The acknowledgment of the notification is handled separately. The EventNotification service would include a MMS confirmed service response (positive or negative) if an event action was defined for the event enrollment.

AcknowledgeEventNotification

This confirmed service is used by a MMS client to acknowledge an EventNotification sent to it by the VMD. The client specifies the event enrollment name, the acknowledgment state, and the transition time parameters that were in the EventNotification request being acknowledged.

TriggerEvent

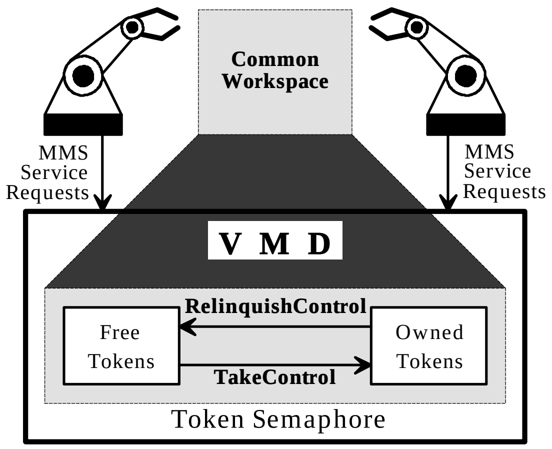
This service is used to trigger a Network Triggered event condition. It gives the client a mechanism by which it can invoke event action and event notification processing by the VMD. For instance, a client can define an event condition, event action, and event enrollments that refer to other MMS clients. When the defining client issues a TriggerEvent service request to the VMD, it will cause the VMD to execute a MMS service request (the event action) and send these results to other MMS clients via the EventNotification service.

6. 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. Some means to control which robot (or robots) can access the workspace is needed. MMS defines two types of *semaphores* for these types of applications: 1) *Token Semaphores* and 2) *Pool Semaphores*.

Token Semaphores

A token semaphore is a named MMS object that can be a representation of some resource, within the control of the VMD, to which access must be controlled. A token semaphore is modeled as a collection of tokens that MMS clients take and relinquish control of using MMS services. This allows both multiple or exclusive ownership of the semaphore.



Token Semaphore Example

A token semaphore is modeled as a collection of free tokens and owned tokens. When a robot (a MMS client in this example) wants to access the common workspace, it will issue a TakeControl request to the VMD controlling the workspace. If there is a free token available, the VMD will grant control by moving a token from the FREE state to the OWNED state and then responding positively to the Take Control request. If the other robot had already owned the token, then the VMD would respond negatively to the TakeControl request. The token representing the common workspace would remain under the control of the robot until control was released with a RelinquishControl request or upon a timeout by the VMD. The total number of tokens available indicates how many simultaneous owners can exist for the same semaphore.

When a MMS client owns the token, it provides some level of access to the underlying resource. An example might be where two users want to change a setpoint for the same control loop at the same time. These users could use a MMS token semaphore containing only one token to represent the control loop in order to coordinate their access to the setpoint. When the user "owns" the token, they can change the setpoint. The other would have to wait until ownership is relinquished.

A token semaphore can also be used for the sole purpose of coordinating the activities of two MMS clients without representing any real resource. This kind of "virtual" token semaphore looks and behaves the same except that they can be created and deleted by MMS clients using the DefineSemaphore service.

Because semaphores either represent a real resource or are used for the purpose of coordinating activities between two or more MMS clients, the scope of a semaphore cannot be AA-Specific. If an object's scope is AA-Specific, there can be only one client. Also, AA-Specific objects only exist as long as the application association exists while real resources must exist outside the scope of any given application association. In addition to its name, the token semaphore has the following attributes:

Deletable

If TRUE, it means that the semaphore does not represent any real resource within the VMD. Therefore, it can be deleted by a MMS client via the DeleteSemaphore service.

Number of Tokens

This attribute indicates the total number of tokens contained in the token semaphore.

Owned Tokens

This attribute indicates the number of tokens whose associated semaphore entry state is OWNED.

Hung Tokens

This attribute indicates the number of tokens whose associated semaphore entry state is HUNG.

Pool Semaphores

A pool semaphore is similar to a token semaphore except that the individual tokens are identifiable and have a name associated with them. These *named tokens* can optionally be specified by the MMS client when issuing TakeControl requests. The pool semaphore itself is a MMS object. The named tokens contained in the pool semaphore are not MMS objects. They are representations of a real resource in much the same way an unnamed variable object is. The name of the pool semaphore is separate from the names of the named tokens. Pool semaphores can only be used to represent some real resource within the VMD. Therefore, pool semaphores cannot be created or deleted using MMS service requests and cannot be AA-Specific in scope. In addition to the name of a pool semaphore, the following attributes also are defined by MMS:

Free Named Tokens

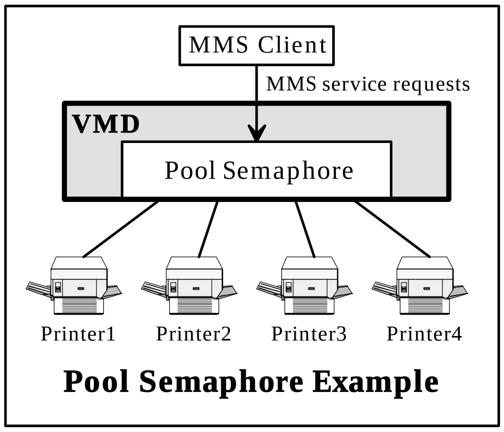
The named tokens for which no associated semaphore entry exists.

Owned Named Tokens

The named tokens for which the associated semaphore entry state is OWNED.

Hung Named Tokens

The named tokens for which the associated semaphore entry is HUNG.



A pool semaphore can be useful to control access to similar but distinguishable resources. In the example, each printer is represented as a separate named token. A MMS client can request control of a specific printer by issuing a TakeControl request specifying a named token. Alternately, if the client doe s not care which specific printer it is granted control of, it can issue the TakeControl request without specifying a named token.

Semaphore Entry

When a MMS client issues a TakeControl request for a given semaphore, the VMD creates an entry in an internal queue that is maintained for each semaphore. Each entry in this queue is called a *semaphore entry*. The attributes of a semaphore entry are visible to MMS clients and provide information about the internal semaphore processing queue in the VMD. The semaphore entry is not a MMS object. It only exists from the receipt of the TakeControl indication by the VMD until the token control of the semaphore is relinquished or if the VMD responds negatively to the TakeControl request. Several of the attributes of the semaphore entry are specified by the client in the TakeControl request. These attributes are:

Entry ID

This is a number assigned by the VMD to distinguish one semaphore entry from another. The Entry ID is unique for a given semaphore.

Named Token

Valid only for pool semaphores. It contains the named token that was optionally requested by the client in a TakeControl request. If the semaphore entry is in the OWNED or HUNG state, it is the named token that the VMD assigned as a result of a TakeControl request.

Application Reference

This is a reference to the MMS client application that issued the TakeControl request that created the semaphore entry.

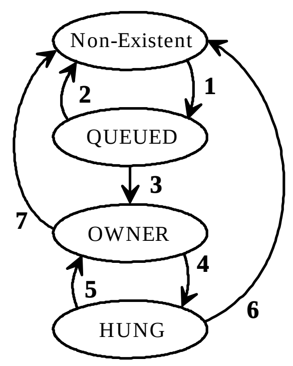
Priority

This attribute indicates the priority of the semaphore entry with respect to other semaphore entries. Priority is used to decide which semaphore entry in the QUEUED state will be granted a token (or named token) when multiple requests are outstanding. The value (0 = highest priority, 64 = normal priority, and 127 = lowest priority) is specified by the client in the TakeControl request.

Entry State

The entry state attribute represents the relationship between the MMS client and the semaphore by one of the following values:

- QUEUED. This means that a TakeControl request has been received but has not been responded to by the VMD. The client is waiting for control of the semaphore.
- OWNED. The VMD has responded positively to the TakeControl request and the client now owns the token (or named token).
- HUNG. This state means that the application association over which the MMS client issued the TakeControl request has been lost and the Relinquish if Connection Lost attribute is FALSE. A MMS client can take control of a semaphore entry in the HUNG state by issuing a TakeControl request with the *preempt* option TRUE and by specifying the MMS client to preempt (via the application reference attribute).



Semaphore Entry State Diagram

A semaphore entry is created each time a client attempts to take control of a semaphore. The semaphore entry reflects the state of the relationship between the client and the semaphore. State transitions can be caused by local autonomous action by the VMD or the following events:

1. TakeControl request received.
2. Timeout, Cancel request, or abort.
3. Semaphore available (token free).
4. Application association aborted and Rel.if.Conn.Lost = FALSE.
5. TakeControl with preempt.
6. Control timeout.
7. a) RelinquishControl request received, or b) control timeout, or c) application association aborted and Rel.if.Conn.Lost = TRUE.

Relinquish if Connection Lost

If this attribute is TRUE, the VMD will relinquish control of the semaphore if the application association for the MMS client that owned the token for this semaphore entry is lost or aborted. If FALSE, the semaphore entry will enter the HUNG state if the application association is lost or aborted.

Control Timeout

This attribute is specified by the client in the TakeControl request and indicates many milliseconds the client should be allowed to control the semaphore once control is granted. If the client has not relinquished control using a RelinquishControl request when the control timeout expires, the semaphore entry will be deleted and control of the semaphore will be relinquished. If the control timeout attribute is omitted in the TakeControl request, no control timeout will apply for that semaphore entry.

Abort on Timeout

This attribute is specified by the client in the TakeControl request and indicates if the VMD should abort the application association with the owner client upon a control timeout.

Acceptable Delay

This attribute is specified by the client in the TakeControl request and indicates how many milliseconds the client is willing to wait for control of the semaphore. If control is not granted during this time, the VMD will respond negatively to the TakeControl request. If the acceptable delay attribute is omitted from the TakeControl request, the client is willing to wait indefinitely.

Semaphore Services

TakeControl

Used by a client to request control of a semaphore.

RelinquishControl

Used by a client to release control over a semaphore that the client currently has control of.

DefineSemaphore, DeleteSemaphore

These services are used by clients to define and delete token semaphores that are used solely for coordinating the activities of two or more clients.

ReportSemaphoreStatus, ReportPoolSemaphoreStatus

These services are used to obtain the status (number of total, owned and hung tokens) of token and pool semaphores.

ReportSemaphoreEntryStatus

This service is used by a client to obtain the attributes of semaphore entries corresponding to a specified state.

7. Other MMS Objects

Operator Stations

An *operator station* is a MMS object. It can be used to represent character based input and output devices that may be attached to the VMD for the purpose of communicating with an operator local to the VMD. MMS defines three types of operator stations:

- ***Entry***. An entry only operator station consists of an input device only. This may be a keyboard or perhaps a bar code reader. The input data must be of the type Visible String consisting of alpha-numeric characters only.
- ***Display.*** A display only operator station consists of a character based output display that can display Visible String data (no graphics or control characters).
- ***Entry-Display.*** This type of operator station consists of both a entry station and a display station.

Because the operator station is a representation of a physical feature of the VMD, it exists beyond the scope of any domain or application association. Therefore, MMS clients access the operator station by name without scope. MMS allows for any number of operator stations for a given VMD. The services used by MMS clients to perform operator communications are:

Input

MMS clients use this service to obtain a single input string from an input device. The service has an option for displaying a sequence of prompts on the display if the operator station is an entry-display type.

Output

This service is used to display a sequence of output strings on the display of the operator station.

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. In this case, the journal entry time stamp would indicate when the sample was taken.

MMS clients read the journal entries by specifying the name of the journal (which can be VMD-Specific or AA-Specific only) and either 1) the date/time range of entries that the client wishes to read or 2) by referring to the entry ID of a particular entry. The entry ID is a unique binary identifier assigned by the VMD to the journal entry when it is placed into the journal. Each entry in a journal can be one of the following types:

Annotation

This type of entry contains a textual comment. This is typically used to enter a comment regarding some event or condition that had occurred in the system.

Data

This type of entry would contain a list of variable tags and the data associated with those tags at the time indicated by the time stamp. Each variable tag is a 32-character name that does not necessarily refer to a MMS variable (although it might).

Event-Data

This type of entry contains both variable tag data and event data. Each entry of this type would include the same list of variable tags and associated data as described above, along with a single event condition name and the state of that event condition at the time indicated by the time stamp.

The services available for MMS journals are as follows:

ReadJournal

This service is used by a client to read one or more entries from a journal.

WriteJournal

This service is used by a client to create new journal entries in a journal. A journal entry can also be created by local autonomous action by the VMD without a client using the WriteJournal service.

CreateJournal, DeleteJournal

These services are used by a client to create and delete (if the journal is deletable) journal objects. The CreateJournal service only creates the journal. It does not create any journal entries (see ReadJournal).

InitializeJournal

This service is used by a client to delete all or some of the journal entries that are in a journal. For instance, a client can use InitializeJournal to delete old journal entries that are no longer of any interest.

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. Although these file services are defined in an annex within the MMS standard, they are widely supported by most commercial MMS implementations. The services for files are described below:

FileOpen

This service is used by a client to tell the VMD to open a file and prepare it for a transfer.

FileRead

This service is used to obtain a segment of the file's data from a VMD. The client would continue to issue FileOpen requests until the VMD indicates that all the data in the file has been returned. The number of bytes returned in each FileRead response is determined solely by the VMD (and can vary from one FileRead response to the next) and is not controllable by the client.

FileClose

This service is used by a client to close a previously opened file. It is used after all the data from the file has been read or can be used to discontinue a file transfer before it is completed.

ObtainFile

This service is used by a client to tell the VMD to obtain a file. When a VMD receives an ObtainFile request it would issue FileOpen, FileRead(s) and FileClose service requests to the client application that issued the ObtainFile request. The client would then have to support the server functions of the FileOpen, FileRead, and FileClose services. A third party option is available (if supported by the VMD) to tell the VMD to obtain the file from another node on the network using some protocol (which may or may not be MMS).

FileRename, FileDelete, FileDirectory

These services are used to rename, delete, and obtain a directory of files on the VMD respectively.

8. MMS Context Management

MMS provides services for managing the context of communications between two MMS nodes on a network. These services are used to establish and terminate application associations and for handling protocol errors between two MMS nodes. The terms *association* and *connection* are sometimes used interchangeably. The node that initiates the association with another node is referred to as the *calling* node. The responding node is referred to as the *called* node.

In a MMS environment, two MMS applications establish an application association between themselves using the MMS Initiate service. This process of establishing an application association consists of an exchange of some parameters and a negotiation of other parameters. The exchanged parameters include information about restrictions that pertain to each node that are determined solely by that node (e.g. which MMS services are supported). The negotiated parameters are items where the called node either accepts the parameter proposed by the calling node or adjusts it downward as it requires (e.g., the maximum message size).

The calling application issues an Initiate service request that contains information about the calling node's restrictions and a proposed set of the negotiated parameters. The called node examines the negotiated parameters and adjusts them as necessary to meet its requirements. It then returns the results of this negotiation and the information about it's restrictions in the Initiate response. Once the calling node receives the Initiate confirmation, the application association is established and other MMS service requests can then be exchanged between the applications.

Once an application association is established, either node can assume the role of client or server independent of which node was the calling or called node. For any given set of MMS services, one application assumes the client role while the other assumes the role of server or VMD. Whether or not a particular MMS application is a client, server (VMD), or both is determined solely by the developer of the application.

Associations v.s. Connections

Although many people may refer to network connections and application associations interchangeably, there is a distinct difference. A connection is an attribute of the underlying network layers that represents a virtual circuit between two nodes. For instance, telephone networks require that two parties establish a connection between themselves (by dialing and answering) before they can communicate. An application association is an agreement between two networked applications governing their communications. It is analogous to the two parties agreeing to use a particular language and to not speak about religion or politics over the telephone. Application associations exist independent of any underlying network connections (or lack thereof).

In a connection oriented environment (e.g., OSI or TCP/IP[2]), the MMS Initiate service is used to signal to the lower layers that a connection must be established. The Initiate service request is carried by the network through the layers as each layer goes through its connection establishment procedure until the Initiate indication is received by the called node. The connection does not exist until after all the layers in both nodes have completed their connection establishment procedures and the calling node has received the Initiate confirmation. Because of this, the association and the connection occur simultaneously in a connection oriented environment.

[2] While there are profiles of OSI and TCP/IP network communications that do not require connections and can be used with MMS, most commercial MMS implementations for 7-layer LAN environments run over network stacks that require connections.

In a connectionless environment, it is not strictly necessary to send the Initiate request before two nodes can actually communicate. In an environment where the Initiate service request is not used before other service requests are issued by a MMS client to a VMD, each application must have all the knowledge regarding the other application's exchanged and negotiated parameters via some local means (e.g., a configuration file). This foreknowledge of the other MMS application's restrictions is the application association from a MMS perspective. Whether an Initiate service request is used or not, application associations between two MMS applications must exist before communications can take place. In some connectionless environments such as MiniMAP, MMS nodes still use the Initiate service to establish the application association before communicating.

Context Management Services

Initiate

This service is used to exchange and negotiate the parameters required for two MMS applications to have an application association.

Conclude

This service is used by a client to request that a previously existing application association be terminated in a graceful manner. The conclude service allows the server to decline to terminate the association due to ongoing activities such as a download sequence or file transfer.

Abort

This service is used to terminate an application association in an ungraceful way. The server does not have the opportunity to decline an abort. An abort may result in the loss of data.

Cancel

This service is used by a client to cancel an outstanding MMS service request (e.g., TakeControl) that has not yet been responded to by the server.

Reject

This service is used by either the client or server to notify the other MMS application that it had received an unsupported service request or a message that was not properly encoded.