OSI Management, or X.700 as it is often referred to from the number of the relevant series of ITU-T recommendations, is a specialisation of the manager-agent model with the following specific features.

The Guidelines for the Definition of Managed Objects (GDMO) and the General Relationship Model (GRM) are used to specify formally managed objects while base ASN.1 is used to specify the values of attributes, the information in notifications and the arguments and results of actions. The naming architecture for managed objects is based on containment relationships, with distinguished names identifying object instances.

The management access mechanism is provided by the Common Management Information Service / Protocol (CMIS/P). This operates over a full OSI stack and offers reliable, connection-oriented end-to-end communication. It can be used to access more than one managed objects in an application in agent role through scoping and filtering and synchronisation (the OSI agent provides this facility).

OSI management was designed with the target to minimise management traffic as much as possible, as it is the same managed network that has to carry it. As such, it provides sophisticated facilities for event-driven management, primarily through event reporting and logging mechanisms but also through generic object management, metric monitoring and summarisation facilities.

Finally, generic functionality is provided through Systems Management Functions (SMFs). These define generic attributes, actions and notifications or generic object classes that serve common functions. The former may be reused by object definitions while the latter can be implemented by managed systems and provide useful functionality. They both make possible the provision of generic manager applications which rely only on those definitions.

OSI Management Overview

- Based on the Manager Agent Model
- GDMO / GRM / ASN.1 used to specify Managed Objects
- Naming architecture based on containment
- CMIS/P Provides the management access service
  - Reliable, connection-oriented service
  - Multiple managed object selection with scoping and filtering, synchronisation
- Sophisticated support for event-driven management
- System Management Functions standardise generic management functionality
The key concept about managed objects is that they represent the Management view of the resources that are to be managed. It is important to distinguish between the management view of a resource and the resource itself. A resource, say a protocol machine, may have a vast number of states, parameters, timers and so on necessary for its "normal" operation, i.e. as a protocol machine. However, it will in general be important to restrict the capability of managing systems to observing and/or modifying only a small subset of the complete set of parameters, etc., of the resource, either because they are not all of interest, or because modification of them "on line" might cause the machine to fail, or generate incorrect protocol, or misbehave in other ways.

Moreover, for monitoring or diagnostic purposes, or for consistency (and so help to human managers), it may be desirable to present for management purposes information that is not explicitly defined in the standard for the normal operation of the resource - e.g. counts of traffic, management thresholds, and so on.

So the definition of the managed object corresponding to a resource requires conscious choice of exactly what aspects of the operation of the resource are to be made visible to management, and in what form. Of course, a vital part of the definition is a full specification of the relationship between the management view on the one hand and the normal operation on the other.

To achieve the separation between the normal behaviour of the resource and the management view, the model represents all communications with an object for management purposes as consisting of management operations (on the object) or management notifications (issued by it), at the object boundary. What happens inside the boundary is the business of the object implementation, not of the Information Model.
Attributes are properties of managed objects. Attribute values are observable at the object boundary. They can reflect or determine object behaviour. Each attribute has an identifier and a value. The identifier is used to distinguish each attribute that belongs to an object. The value may be a simple number or a structure.

Operations can be used to read or modify attributes. The object enforces the constraints for modifying certain attributes or keeping the range of values within limits defined in the object definition.

Set-Valued attributes have a value which is a set of members of a given data type. The set size is variable: no values are repeated and order is not important. Additional operations can add or remove members from the set.

Group Attributes supply a means for referring to a collection of attributes within an object. Operations are applied to all members of the group in no particular order. Only operations which do not require specified values are allowed.
Guidelines for the Definition of Management Objects (GDMO)

- Content and structure of object class definitions:
  - Properties inherited from super-classes
  - Properties imported from other standards
  - Definition of further attributes, operations and notifications
  - Associations of attributed, operations and notifications with object behaviour
  - Assignment of Managed Object class name and attribute identifiers
  - Definition of possible containment relationships

The GDMO standard converts the architectural principles of the Information Model and the idea of importing definitions from other standards into formal techniques for defining managed object classes, i.e. for writing standards for specific managed object classes. Those responsible for writing such standards will include all the OSI layer groups of ISO, together with many ITU-T groups; in addition, it is hoped that the techniques developed for use within ISO and ITU-T will be used much more widely, for example by IEEE, the Network Management Forum and other suppliers and supplier groups, so that the object they define can be managed by OSI Management protocols and functions.

The Guidelines therefore give a recipe and a checklist for Managed Object Class definitions, including notational tools such as templates. They also provide for generic attribute class definition, so that other groups may if they wish define their own set of generic attributes separately from specific managed object class definitions. A layer group, for example, might wish to use a number of attributes in several object class definitions; separate definition of the common properties of these attributes would ensure consistency across the different managed object class definitions.
Background: The OSI ASN.1 Language

- Abstract Syntax Notation One (ASN.1) is a network data structuring language
- Supports both simple and constructed types
- Special type: Object identifier:
  - A Series of non-negative integers downs a global conceptual registration tree
- Used in both OSI and Internet Management

Some necessary background information concerns the OSI Abstract Syntax Notation One (ASN.1) language. This is an abstract “network” data structuring language that supports simple and constructed types. A particularly important ASN.1 type is the Object Identifier (OID), which expresses a sequence of non-negative integers on a global registration tree. OIDs are registered by standards bodies e.g. ISO, ITU-T, IETF and are used instead of friendly string names to avoid ambiguities. For example, the OSI objectClass attribute name is registered as {joint-iso-ccitt(2) ms(9) smi(3) part2(2) attribute(7) objectClass(65)}..

Two commonly used constructed types are SEQUENCE and SEQUENCE OF. A SEQUENCE type is equivalent to a structure in C/C++ or a record in Pascal while a SEQUENCE OF type is equivalent to a dynamic array.

ASN.1 is used in both the OSI management and SNMP frameworks to specify the management protocol packets and the structure of managed object information e.g. attributes, notifications, etc. Extensions of ASN.1 through its template facility are used to specify the O-O properties of managed objects. A similar abstract language used in OMG CORBA is the Interface Definition Language (IDL). This is used both as a data structuring tool and as the language to specify the O-O properties of object interfaces.
GDMO (Guidelines for the Definition of Managed Objects) is an abstract object-oriented specification language, based on ASN.1 templates. It specifies syntactic properties of objects, including inheritance and containment (naming) relationships. Behavioural aspects are specified in natural language. The use of SDL-92 (object-oriented SDL) is considered for the future. The General Relationship Model (GRM) is a more recent development and is a set of tools for the specification of relationship types between managed object classes. These include relationship, role and relationship binding templates.

GDMO managed object classes have attributes, accept operations and emit notifications. There are two types of operations, those that apply to attributes (Get, Set) and those that apply to the object instance (Action, Create, Delete). In fact, Create is addressed to the managed system (agent) which will create the object instance. Base ASN.1 is used to describe the attribute, action, notification and managed object specific error values.

GDMO, GRM and ASN.1 provide a very powerful fully object-oriented information framework. Mappings between GDMO and C++ have been investigated and there exist advanced OSI management platforms that offer tools (compilers) to automate code generation, leaving only behavioural aspects to be hand-coded.
An OSI Management Information Base (MIB) defines a set of Managed Object Classes (MOCs) and a schema which defines the possible containment relationships between instances of those classes. There may be many types of relationships between classes and their instances but containment is treated as a primary relationship and is used to yield unique names. The smallest reusable entity of management specification is not the object class, as is the case in other O-O frameworks, but the package. Object classes are characterised by one or more mandatory packages while they may also comprise conditional ones. An instance of a class must always contain the mandatory packages while it may or may not contain conditional ones. The latter depends upon conditions defined in the class specification.

Managing systems may request that particular conditional packages are present when they create a managed object instance. Conditional packages allow for dynamic (i.e. run-time) specialisation of an object instance while inheritance allows only for static (i.e. compile-time) specialisation through new classes (see next). The problem with conditional packages is that they do not have a direct counterpart in O-O programming frameworks e.g. C++, Java etc.

A package is a collection of attributes, actions, notifications and associated behaviour. Attributes are analogous to object types in SNMP. Like an object type, an attribute has an associated syntax. Unlike SNMP though, there are no restrictions on the syntax used. Support for arbitrary syntaxes provides a much more flexible scheme than that of SNMP. For example, it allows the definition of complex attributes such as a threshold, whose syntax can include fields to indicate whether the threshold is currently active or not and its current value. It is also more expensive to implement since support for encoding and decoding of completely arbitrary syntaxes must be provided.
OSI Managed Objects

- Actions Provide a flexible ‘remote method’ execution paradigm
- Exceptions may be associated to Actions and Attributes
- Behaviour is the semantics of Classes, Packages, Attributes, Actions and Notifications
- Specified in Natural Language only though some consideration has been given to use of SDL or Z

OSI managed object classes and packages may have associated specific actions that accept arguments and return results. Arbitrary ASN.1 syntaxes may be used, providing a fully flexible “remote method” execution paradigm. Exceptions with MOC-defined error information may be emitted as a result of an action. The same is also possible as a result of operations to attributes under conditions that signify an error, for which special information should be generated. Object classes and packages may also have associated notifications, specifying the condition under which they are emitted and their syntax. The latter may be again an arbitrary ASN.1 type. By behaviour, one means the semantics of classes, packages, attributes, actions and notifications and the way they relate as well as their relationship to the entity modelled through the class. Behaviour is specified in natural language only, as is also the case in SNMP.
OSI Management follows a fully O-O paradigm and makes use of concepts such as inheritance. Managed object classes may be specialised through subclasses that inherit and extend the characteristics of superclasses. The use of class inheritance together with and packages allow re-usability and extensibility of both specification and associated implementation if an object-oriented design and development methodology is used. Pursuing the previous example, a `tpProtocolEntity` object class may inherit from an abstract `protocolEntity` class that models generic properties of protocol entities, such as the operational state, the service access point through which services can be accessed, etc. An abstract `connection` class may model generic properties of connection-type entities such as the local and remote service access points, the connection state, creation and deletion notifications, etc.

All OSI MO classes inherit from the abstract class `top`, which contains self-describing information for an object instance e.g. the actual class, list of conditional packages etc. When an instance is created, the conditional packages may be requested to be present by initialising accordingly the value of the packages attribute, which has “set by create” properties.

Note that specification of manageable entities through generic (abstract) classes used only for inheritance and re-usability purposes may also result in generic managing functions by using polymorphism across management interfaces. For example, it is possible to provide a generic connection-monitor application that is developed knowing only the generic `connection` class. This may monitor connection objects in different contexts e.g. X.25, ATM, etc. without knowing the specific semantics of a context. That way, reusability is extended to managing functions as well to managed object classes and their implementations.
A managed object may be contained in another. This relationship is derived from the structure of the resources that are represented by the objects. For example, a network connection object will be contained in a network entity managed object: this represents the fact that the network connection is operated by the network entity in question, and so if the network entity fails, or is shut down, for example, the network connection will be affected, whereas network connections operated by other network entities will not be.

In contrast to inheritance, which is a relationship between object classes, containment is a relationship between object instances. It allows composite objects to be built up out of component objects. Its main use is to provide a means of naming managed objects.
OSI MO Containment (example)

The types of attributes, action and notifications arguments and replies and exception parameters are specified in ASN.1. Object Identifiers are associated with classes, packages, attributes, notifications and actions. They have nothing whatsoever to do with instance naming. Instead, managed object instances are named through a mechanism borrowed from the X.500 Directory. Managed object classes have many relationships but containment is treated as a primary relationship to yield unique names. Instances of managed object classes can be thought as logically containing other instances. As such, the full set of managed object instances available across a management interface are organised in a Management Information Tree (MIT), also referred to as the “containment hierarchy”. This requires that an attribute of each instance serves as the “naming attribute”. The tuple of the attribute and its value form a Relative Distinguished Name (RDN) e.g. 

connectionId=1234. This should be unique for all the object instances at the first level below a containing instance. If these instances belong to the same class, then it is the value of the naming attribute that distinguishes them (the “key”).

The containment schema is defined by name-bindings which specify the allowable classes in a superior/subordinate relationship and identify the naming attribute. Name bindings and naming attributes are typically defined for classes in the first level of the inheritance hierarchy, immediately under top so that they are “inherited” by specific derived classes. An example of a containment tree is shown above, modelling connections as contained by protocol entities, contained by layer subsystems, contained by a network element. A managed object name, also known as a Local Distinguished Name (LDN), consists of the sequence of all the relative names from the top of the tree down to the object e.g. 

{subsystemId=network, protocolEntityId=x25, connectionId=123}. OSI management names are assigned to objects at creation time last for the lifetime of the object. An OSI managed object has exactly one only name i.e. the naming architecture does not allow for multiple names.
The Common Management Information Service / Protocol (CMIS/P) provides distributed access to management information. It is a connection-oriented, reliable protocol that operates over a full OSI stack (it uses the Association Control and Remote Operations application service elements). Its operations are a superset of those available at the managed object boundary. Many objects may be addressed through one request by scoping and filtering. Operation success can be requested to be either atomic or best-effort while many linked replies may be returned to the manager. It offers very good bulk data transfer, discovery and atomicity facilities. In fact, scoping, filtering and synchronisation are offered by the agent part of the receiving application, all CMIS/P does is carry the relevant parameters.
CMIS services allow peer management processes to exchange management information. The CMIS services are:

- **M-CREATE**: issued by a manager to create a managed object.
- **M-DELETE**: issued by a manager to delete managed object(s).
- **M-SET**: issued by a manager to modify attributes of managed object(s).
- **M-GET**: issued by a manager to read attributes of managed object(s).
- **M-CANCEL-GET**: issued by a manager to terminate the return information from a previous M-GET operation.
- **M-ACTION**: issues by a manager to perform an action on managed object(s).
- **M-EVENT-REPORT**: issued by an agent to convey management information resulting from a notification emitted by a managed object, to a manager.

M-CREATE and M-GET are only available in confirmed mode, while the other can operate in confirmed or non-confirmed mode.
The CMIS service primitives are a superset of the operations available at the object boundary within agents, with additional features to allow for object discovery and bulk data retrieval, operations on multiple objects and a remote “retrieval interrupt” facility. The primitives available at the CMIS level are Get, Set, Action, Create, Delete, Event-report and Cancel-get. The Get, Set, Action and Delete operations may be performed on multiple objects by sending one CMIS request which expands within the agent based on scoping and filtering parameters. Since OSI managed objects are named according to containment relationships and organised in a management information tree, it is possible to send a CMIS request to a base object and select objects below that object through scoping. Objects of individual levels, until individual levels or the whole subtree may be selected. The selection may be further eliminated through a filtering parameter that specifies a predicate based on assertions on attribute values, combined by boolean operators. Scoping and filtering are very powerful and provide an object-oriented database type of functionality in OSI agents. This results in simplifying the logic of manager applications and reducing substantially management traffic.

When applying an operation to multiple objects through scoping and filtering, atomicity may be requested through a synchronisation parameter. The result/error for each managed object is passed back in a separate packet, which results in a series of linked replies and an empty terminator packet. A manager application may interrupt a series of linked replies through the Cancel-get facility. Finally, the Set, Action, Delete and Event-report operations may also be performed in an unconfirmed fashion. While this is typical for event reports (the underlying transport will guarantee their delivery in most cases) it is not so common for intrusive operations as the manager will not know if they succeeded or failed. Nevertheless, such a facility is provided and might be used when the network is congested or when the manager is not interested in the results/errors of the operation. The CMIS interactions between applications in manager and agent roles are depicted above (apart from Cancel-Get).
Management applications use application-layer services to communicate with peer applications

The Association Control Service Element (ACSE) is used to establish and terminate associations for the purpose of OSI systems management using the Systems Management Application Service Element (SMASE) and the Common Management Information Service element (CMISE). Association Control can also be used to negotiate optional CMIS functional units such as multiple object selection, multiple reply, filter and cancel-get.

CMIP is the protocol mapping (abstract syntax and elements of procedure) of CMISE on the Remote operation Service Element (ROSE) over a full OSI stack.

CMIP adds value to the definition of ROSE parameters, and hence a CMIP PDU is a specialised ROSE APDU. Thus, by definition, A CMIP protocol machines is a ROSE protocol machines (I.e. CMIP is not a user of ROSE).

There are other implementations of CMIP with variations on the lower parts of the stack, e.g. CMIP over TCP/IP (sometimes called CMOT), and over LLC (sometimes called CMOL). These two do not use the ACSE and lose the ability to do application context negotiation and association access control.

The different lower level stacks shown are taken from the ITU-T Q.811 recommendation associated with TMN.
CMIP Request/Responses

The requester of a CMIP operation is called an INVOKER (or invoking CMISE-service-user). The responder to a CMIP operation is called a PERFORMER (or performing CMISE-service-user). CMIP is unaware of the difference between “operations” and “notifications” and has no knowledge of the management role of either party in the management exchange (CMIS an CMIP do not use the manager-agent terminology).

All requests are transmitted in the RO-INVOKE APDU, and for the non-confirmed mode service, there is never a response (even if the request is in error).

For the confirmed mode services, normal responses are returned in the RO-RESULT APDU. CMIS errors that are detected by the performer are returned in the RO-ERROR APDU, and those detected by the CMIP protocol machine in the RO-REJECT APDU. There are no responses to RO-REJECT.

In the special case of multiple responses to a single request, the RO-INVOKE APDU is used to return each partial result, the sequence being terminated by an empty RO-RESULT APDU (to indicate the end of sequence)

There is a one-to-one correspondence between the ROIv and response (RORS, ROER or RORJ) APDUs, and they always contain the same identifier so that request and response can be matched. ROIv APDUs that are used to return partial results each have their own unique identifier, but also contain a copy of the identifier of the request, so that they can be correlated.
As OSI management was designed with minimal management traffic as target, it offers sophisticated facilities for event-based operation through event reporting and logging. Special support managed objects i.e. event forwarding discriminators and logs can be created and subsequently manipulated by managers to control the level of reporting and logging. This is possible to a very fine granularity, down to the level of any parameter included in the notification. These parameters are both general (class, name of emitting object, time and notification type), and specific to the notification (e.g. attribute that changed, previous and current value for an attribute value change notification).

Logging provides the facility to store the notification locally as a log record to be retrieved later. This provides a “standard” remote storage mechanism, giving enormous flexibility with respect to the manipulation of events, allowing later retrieval at less busy times, the examination of a critical event log in a system that misbehaved etc.
Distribution aspects in OSI management are supported by the OSI directory, which provides a federated hierarchical object-oriented database. The directory resides in many Directory Service Agents (DSAs) that administer parts of the global Directory Information Tree (DIT). Parts of the global MIT belong to different Autonomous Administrative Areas (AAAs) and start at Autonomous Administrative Points (AAPs). DSAs are accessed by applications in Directory User Agent (DUA) roles via the Directory Access Protocol (DAP) while DSAs communicate with each other via the Directory System Protocol (DSP). Accessing the local domain DSA is enough to search for information held anywhere in the global DIT. The above figure depicts the operational model of the directory and the global DIT. Directory Objects (DOs) are named using distinguished names that express containment relationships, in the same fashion as OSI managed objects. In fact, the directory naming architecture preceded that of OSI management and was essentially reused in the latter.

OSI management applications or, System Management Application Processes (SMAPs) in OSI parlance, are represented by directory objects. The latter contain System Management Application Entity (SMAE) objects associated with each interface of that process. SMAE DOs contain addressing information as well as information regarding other aspects of that interface, termed as Shared Management Knowledge (SMK). Since the same hierarchical naming architecture is used for both the OSI directory and management, the two name spaces can be unified. This can be achieved by considering a “logical” link between the topmost MIT object of an agent and the corresponding SMAP DO. Through this approach, the OSI directory and management name spaces are essentially unified.
The universal name space is shown above through the extended manager-agent model. The manager application may address objects through their global names, starting from the root of the directory tree e.g. \{c=GB, o=UCL, ou=CS, cn=ATM-NM-OS, networkId=ATM, logId=1, logRecordId=5\}. The underlying infrastructure will identify the directory portion of the name i.e. \{c=GB, o=UCL, ou=CS, cn=ATM-NM-OS\}, will locate the relevant DO and will retrieve attributes of the contained SMAE DO, including the OSI presentation address of the relevant interface. It will then connect to that interface and access the required managed object through its local name i.e. \{logId=1, logRecordId=5\}. Note that the networkId=ATM relative name of the topmost MIT object is not a part of the local name. Global names guarantee location transparency as they remain the same even if the application changes location. Only the presentation address attribute of the relevant SMAE DO needs to change. Note finally that applications in manager roles are also addressed through directory distinguished names for emission of event reports as the destination address in EFDs contains the directory name of the relevant manager SMAP.
Systems Management Functions

• Standardised generic management functionality offered by Management Systems
• May be reused in different agent MIBS
• Can be used to provide generic management applications (configuration, alarm, performance etc)

Systems Management Functions (SMFs) provide generic functionality that can be thought as a layer above CMIS/P. They define generic attributes, actions and notifications or generic object classes that serve common functions. The former may be reused by object definitions while the latter can be implemented by managed systems and provide reusable functionality.

The generic attributes, actions and notifications provide information commonality. The generic classes obviate the use of specific facilities dispersed in many other object classes. For example, the use of metric monitor facilities obviates the use of rates and tide-marks that apply to a counter or gauge within a specific class. The same applies to accounting, testing facilities etc.

SMFs also make possible the provision of generic manager applications which rely on those generic definitions. Examples are configuration and alarm managers, performance monitors, testers, event loggers etc.
A list of the first 16 SMFs is shown above. These are more or less complete or very close to completion. More are currently being worked on, including domains and policies, software and distributed time management, enhanced event control and many others.