IMPLEMENTING OSI MANAGEMENT

A TUTORIAL

for the 3rd International Symposium
on
Integrated Network Management

April 1993

George Pavlou
Senior Research Fellow
Dept. of Computer Science
University College London
Gower Street
London WC1E 6BT

e-mail: gpavlou@cs.ucl.ac.uk

OBJECTIVES

- **to show that the OSI management power can be efficiently exploited through well thought-out implementations**

- **to explain how this power can be harnessed and re-used by hiding complexity behind objet-oriented Application Program Interfaces (APIs)**

- **to decompose the whole implementation problem into smaller ones and solve each - a modular approach**

- **to highlight difficult issues and suggest workable solutions**

- **to suggest optimizations that will result in efficient implementations**

REQUIRED BACKGROUND

- **A basic understanding of the OSI Management architecture - an short overview is given here**

- **An understanding of modular and object-oriented design methodology - the basic principles are introduced here**

- **Some familiarity with C/C++ terminology and syntax - this is not essential**

TUTORIAL FORMAT

- **Informal - questions may be asked at any time**

TUTORIAL STRUCTURE

- **Introduction to OSI management**

- **Object-Oriented Design Methodology**

- **Communication Services**
  - the Common Management Information Service

- **General Infrastructure**
  - Support for Asynchronous Event-Driven Applications
  - Transparent Abstract Syntax Handling

- **Management Agent Infrastructure and Realization**
  - Real resource access policies
  - Managed Object Support
  - the Generic Managed System

- **Manager Infrastructure**
  - High-Level Methods for Remote MIB Access
  - Graphical User Interface Integration

- **Summary and Information**

INTRODUCTION TO

THE OSI MANAGEMENT MODEL

INTRODUCTION

- **OSI Management provides rich and powerful management mechanisms**

- **These can be used to provide powerful, extensible and scalable management solutions**

- **There has been a belief, amplified by the lack of implementations, that OSI management facilities are difficult to implement**

- **It will be shown that this is not true, at least not if a modular (object-oriented) approach is followed**

- **The solutions suggested will be based on object-oriented design methodology**

THE OSI MANAGEMENT MODEL

- **OSI Network Management follows an object-oriented model - physical or logical real resources are managed through abstractions of them known as Managed Objects (MOs)**

- **Management systems need also MOs that do not represent anything real but exist for the needs of the management system itself (control MOs)**

- **MOs are handled by applications in agent roles and are accessed by applications in manager roles in order to implement management policies**

- **The global collection of management information is termed the Management Information Base; each agent handles a part of it in its Management Information Tree**

- **Information in manager-agent interactions is conveyed through the management service / protocol CMIS/P**

- **Agent, managed system and managed node are synonymous; the same holds for manager, managing application, management station**

MANAGEMENT FUNCTIONAL AREAS

- **Fault management - generate notifications, maintain error logs and transmit error reports. perform tests**

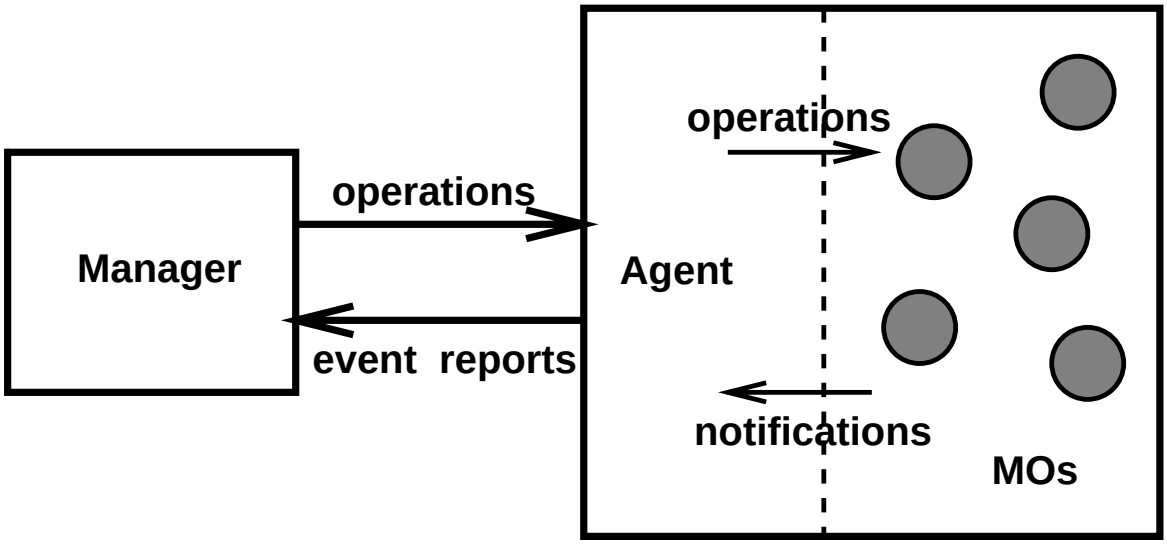
- **Configuration management - configuration of parameters, state and relationship information, software distribution, inventory**

- **Security management**
  a. **management of security - password control, cryptographic key distribution**
  b. **security of management - authentication, access control**

- **Accounting management - measurement and data collection, usage reporting**

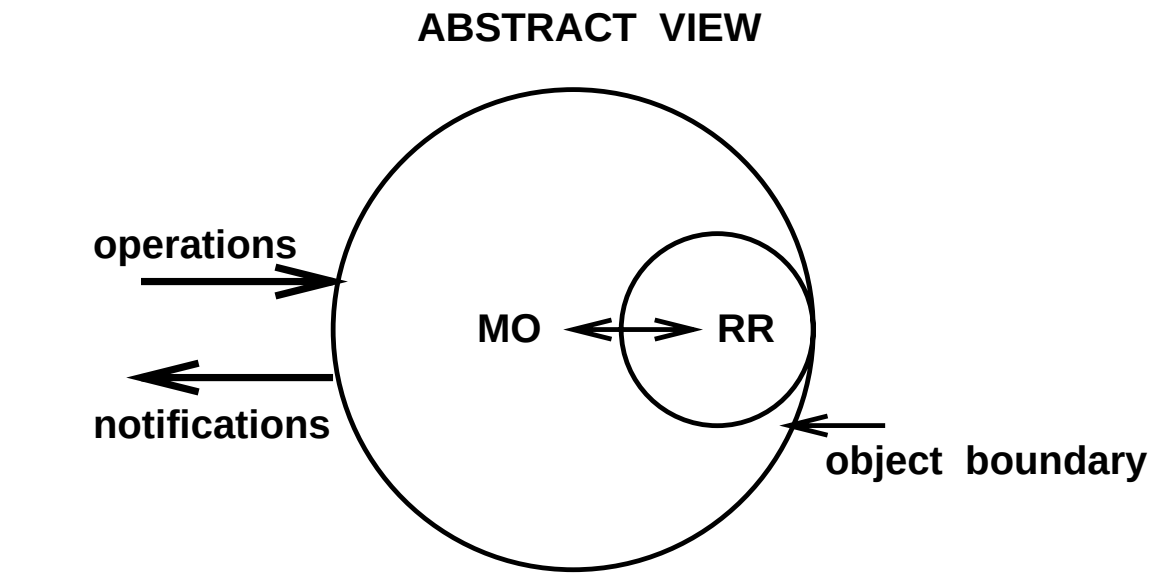
- **Performance management - workload monitoring, measurement summarization, capacity planning**

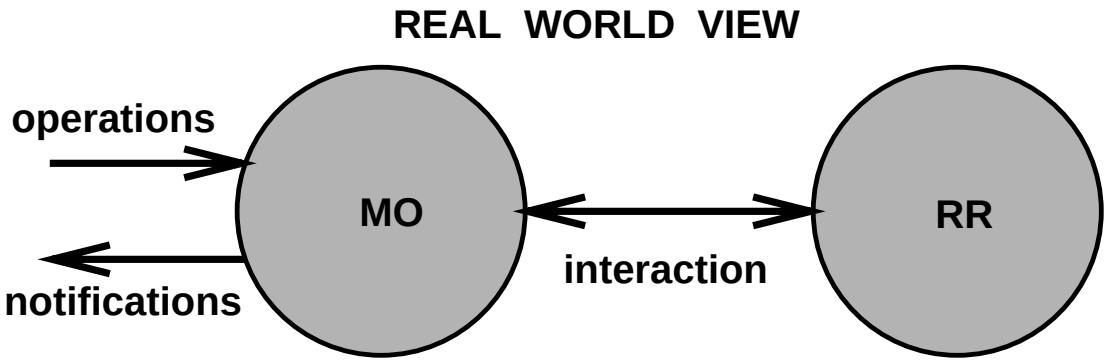
MANAGEMENT INTERACTIONS



MANAGEMENT PROTOCOL

THE MANAGED OBJECT ABSTRACTION



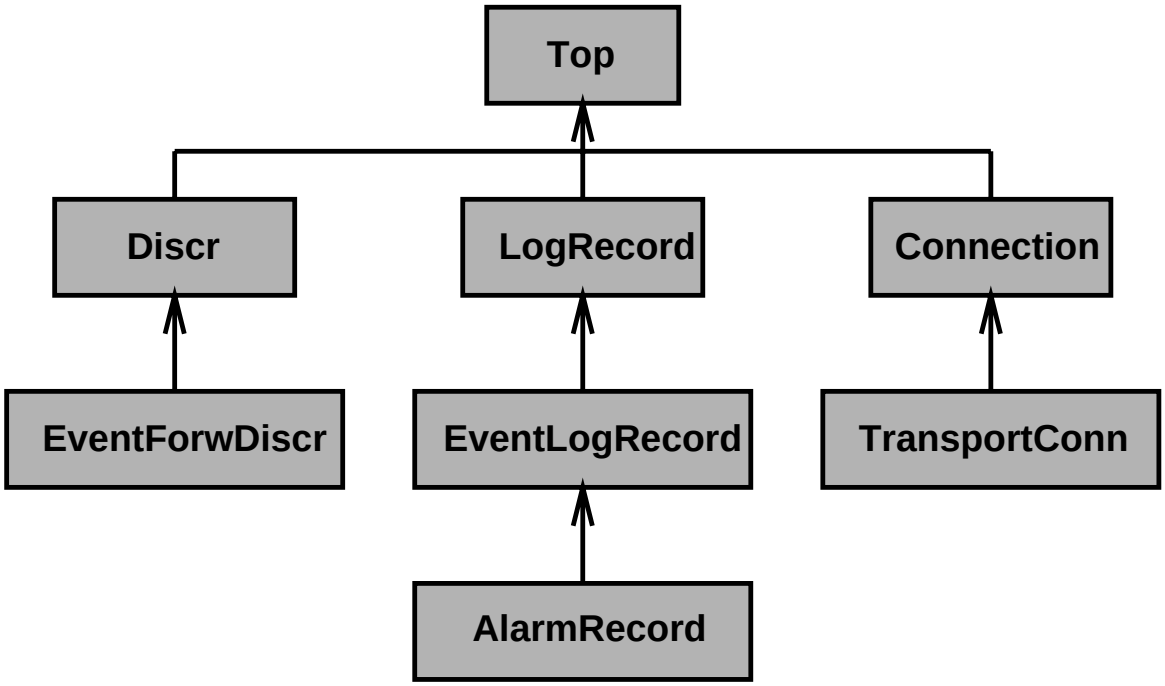


- ATTRIBUTES, ACTIONS, NOTIFICATIONS, BEHAVIOUR
- ENCAPSULATION

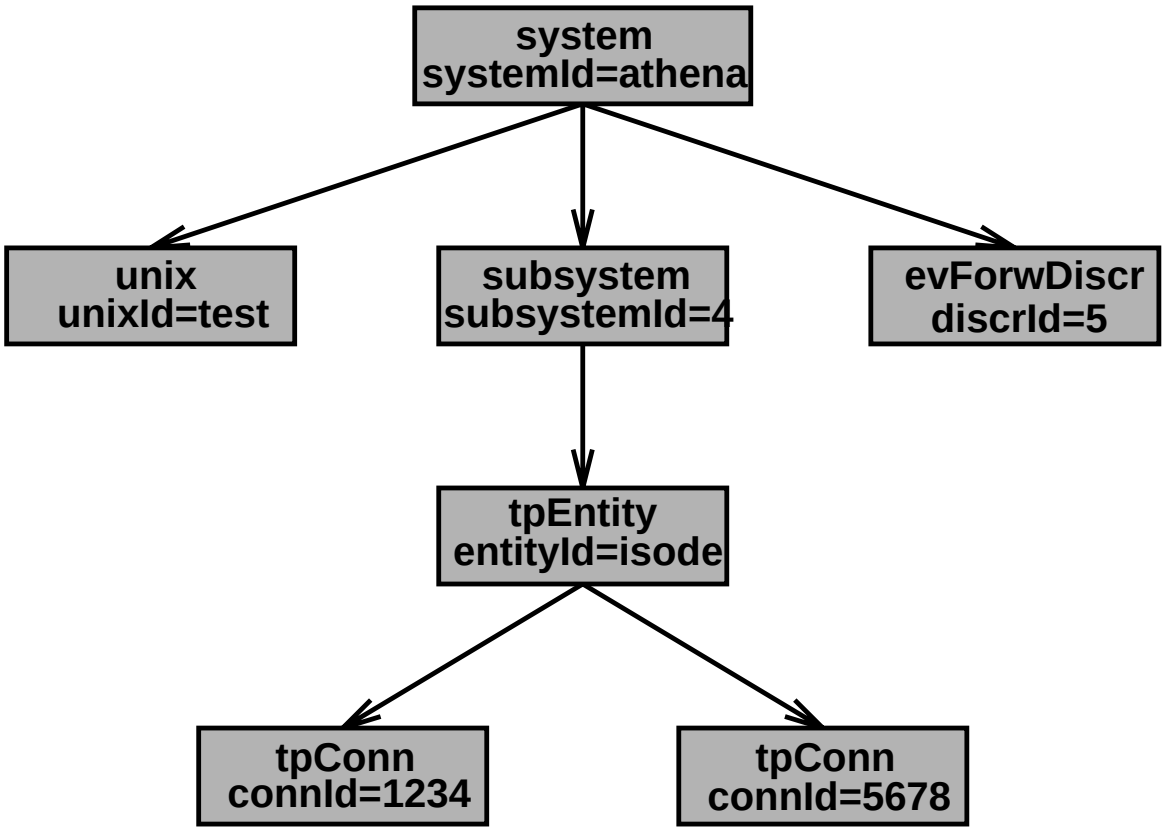
MANAGEMENT INFORMATION MODELING

- **There are two basic aspects in management information modeling**
  - **inheritance**
  - **containment**
- **Every class is derived from the Top one which contains self-describing information for an object instance**
- **All specified managed object classes form a global inheritance tree**
- **In all managed systems, managed object instances are organized in a Management Information Tree (MIT) according to containment relationships for the purpose of naming**
- **Properties of managed object classes are formally specified in a ASN.1 template language known as Guidelines for the Definition of Managed Objects (GDMO)**

AN EXAMPLE INHERITANCE TREE



AN EXAMPLE CONTAINMENT TREE (MIT)



example of a class and (local) instance (name)
class: tpConn
instance: { subsystemId=4 @ entityId=isode @ connId=5678 }

MANAGEMENT PROTOCOL OPERATIONS

- **The management protocol primitives express operations that convey the information of messages to managed objects**

  - **Get - read management information (attributes)**
  - **Set - write management information (attributes)**
  - **Action - perform an action, a general method on a managed object**
  - **Create - create a managed object**
  - **Delete - delete a managed object**
  - **Cancel-Get - cancel a previously issued get operation**
  - **Event-Report - send an event report**

SCOPING, FILTERING, SYNCHRONIZATION

- **Get, Set, Action and Delete may be performed on many objects through scoping; the operation is addressed on a "base" managed object and can be performed on objects of that subtree**

- **The selection of objects can be controlled through filtering on the managed object attributes; this may be used independently of scoping**

- **All or no operations can be requested to succeed by specifying the synchronization parameter to be atomic - the default is best effort**

EVENT REPORTING AND LOGGING

- **Event reporting is fundamental to the nature of the OSI model which supports an event-driven approach**
- **Logging is a complementary function and also essential**
- **MOs emit notifications; these may be converted to event reports and/or log records according to the event reporting and logging function**
- **Special managed objects of class Event Forwarding Discriminator and Log control this activity**
- **These may be created / deleted / manipulated through management to control the level of reporting / logging**

PROXY MANAGEMENT

- **Full OSI agents are impossible to run on small network elements**

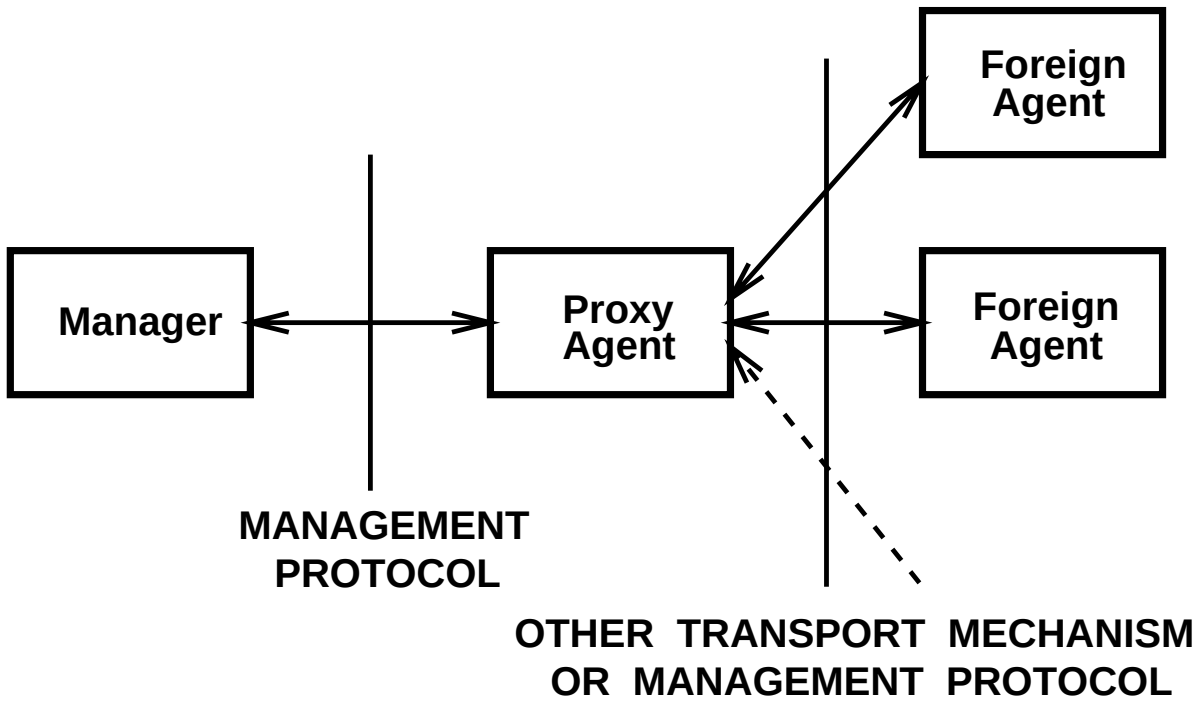
- **Moreover, a lot of network elements that support another management model / protocol exist already**

- **All these elements still need to be managed - this can be done through proxy agents that translate between the OSI and the "foreign" protocol**

- **The nature of the translation varies:**
  - **if the foreign agent supports the same protocol over a different stack e.g. CMOT, CMOL, the translation is simple**
  - **if it supports a different protocol e.g. SNMP or other, translation is difficult and some functionality may be lost**

- **CMIS to SNMP translation for existing SNMP MIBs that have been translated to equivalent OSI ones is deterministic and can be automated**

PROXY MANAGEMENT



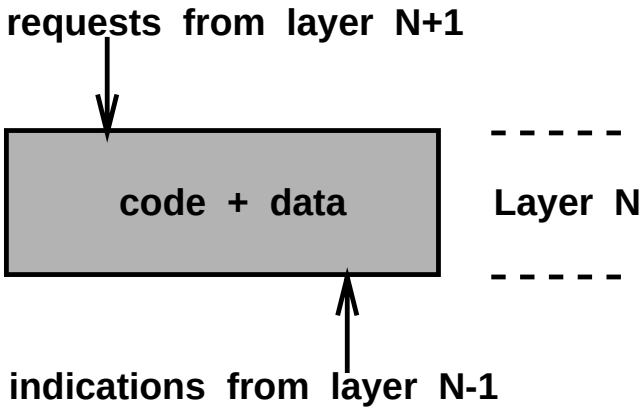
OBJECT-ORIENTED DESIGN METHODOLOGY

MODULAR DESIGN METHODOLOGY

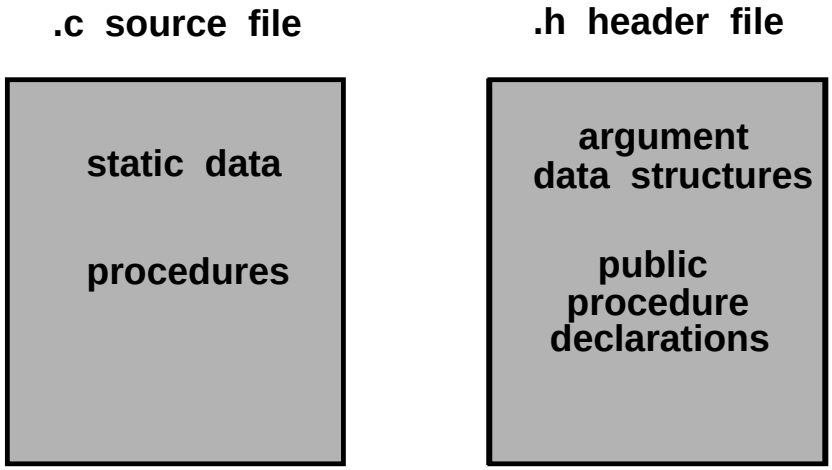
- **A Module is a piece of code with well defined functionality**
- **Its functionality is made available through well-defined entry-points, implemented as procedure calls**
- **Any internal data are "invisible" from outside the module**
- **Drawback: modules are single-instanced, functionality cannot be extended / modified from outside**
- **Widely used programming languages supporting modules: C (implicitly), ADA, Modula**

An Example of a Module

A whole protocol layer can be a module



A module in C



OBJECT-ORIENTED DESIGN METHODOLOGY

- **An Object is similar to a module, a collection of procedures ("methods") and data**

- **These are more tightly coupled and belong to the same structure ("class")**

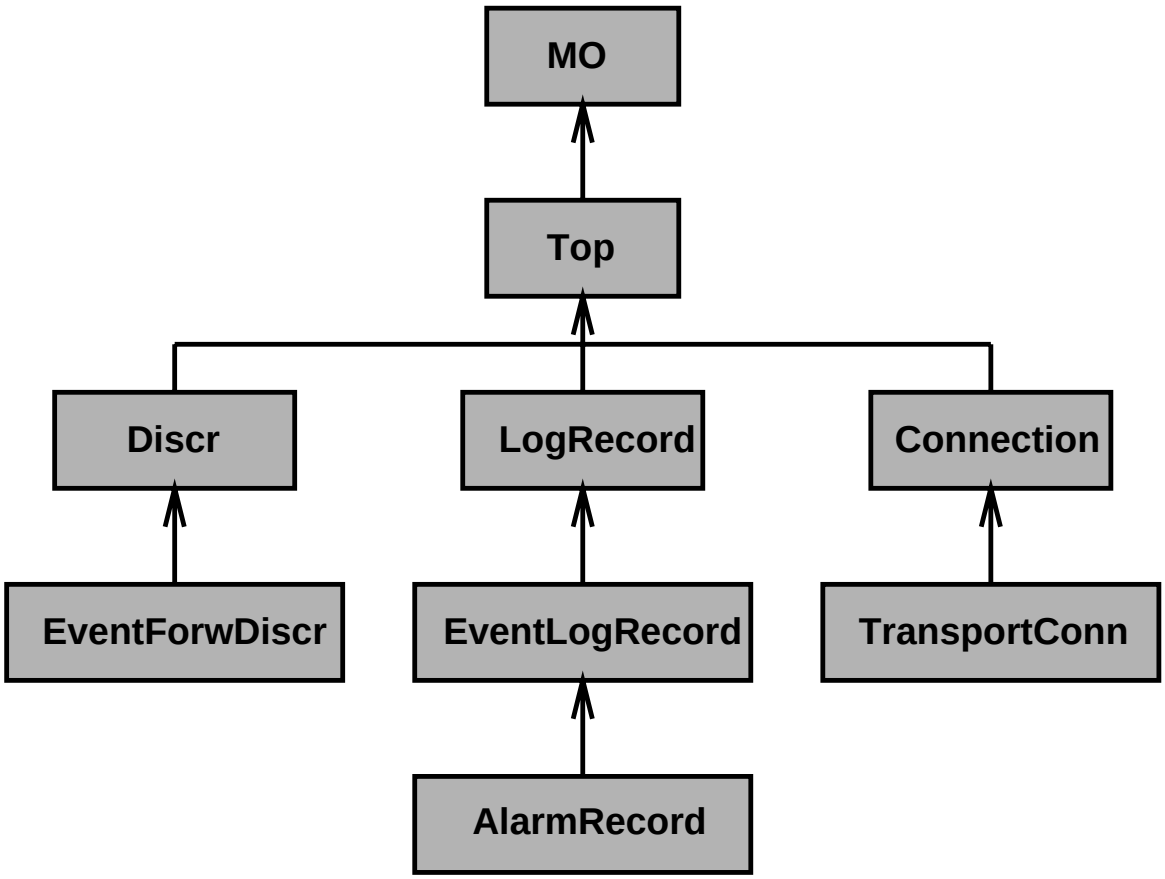
- **Many objects of a class may exist at any time ("instances")**

- **A class behavior can be modified, extended or even restricted by simply deriving another ("inheritance")**

- **Objects of different derived classes may be treated as objects of a common parent class ("polymorphism")**

- **Widely used object-oriented programming languages: C++, Objective-C, Eiffel, Smalltalk**

AN EXAMPLE INHERITANCE TREE



EXAMPLE OF POLYMORPHISM

- **The demonstrated inheritance hierarchy is related to an OSI management implementation - note the implementation specific "Managed Object" (MO) root**

- **Instances of class Event Forwarding Discriminator, Alarm Record and Transport Connection can be treated in the software just as managed objects i.e. MO instances**

- **Performing a "get attribute(s)" operation on a managed object may mean completely different things with respect to the real resource the managed object represents**

- **Through polymorphism, knowledge of the specific class is NOT needed in order to perform the operation as the MO class provides the stubs for all the management operations**

- **The "get attribute(s)" method is redefined by every specific class to respond appropriately but the interface remains the same ("virtual method")**

POLYMORPHISM THROUGH VIRTUAL METHODS

- **Common behavior to classes of an inheritance tree is expressed through "virtual methods" - these may be redefined in derived classes**

- **Inherited behavior may be modified, extended or restricted by simply changing the code of virtual methods**

- **When treating objects as if they were instances of a common "base" class, the correct virtual method according to the actual class is invoked.**

- **Polymorphism through virtual methods can be emulated in languages like C that enable function pointers to be members of structures**

- **An object-oriented language though will do the job for you, C++ is known to be efficient, even compared to C, and has additional useful features**

COMMUNICATION SERVICES

COMMUNICATION SERVICES

- **In the OSI Management model, communications services are provided by the Common Management Information Service (CMIS)**

- **This service is realized through the Common Management Information Protocol (CMIP) over an full or lightweight OSI stack**

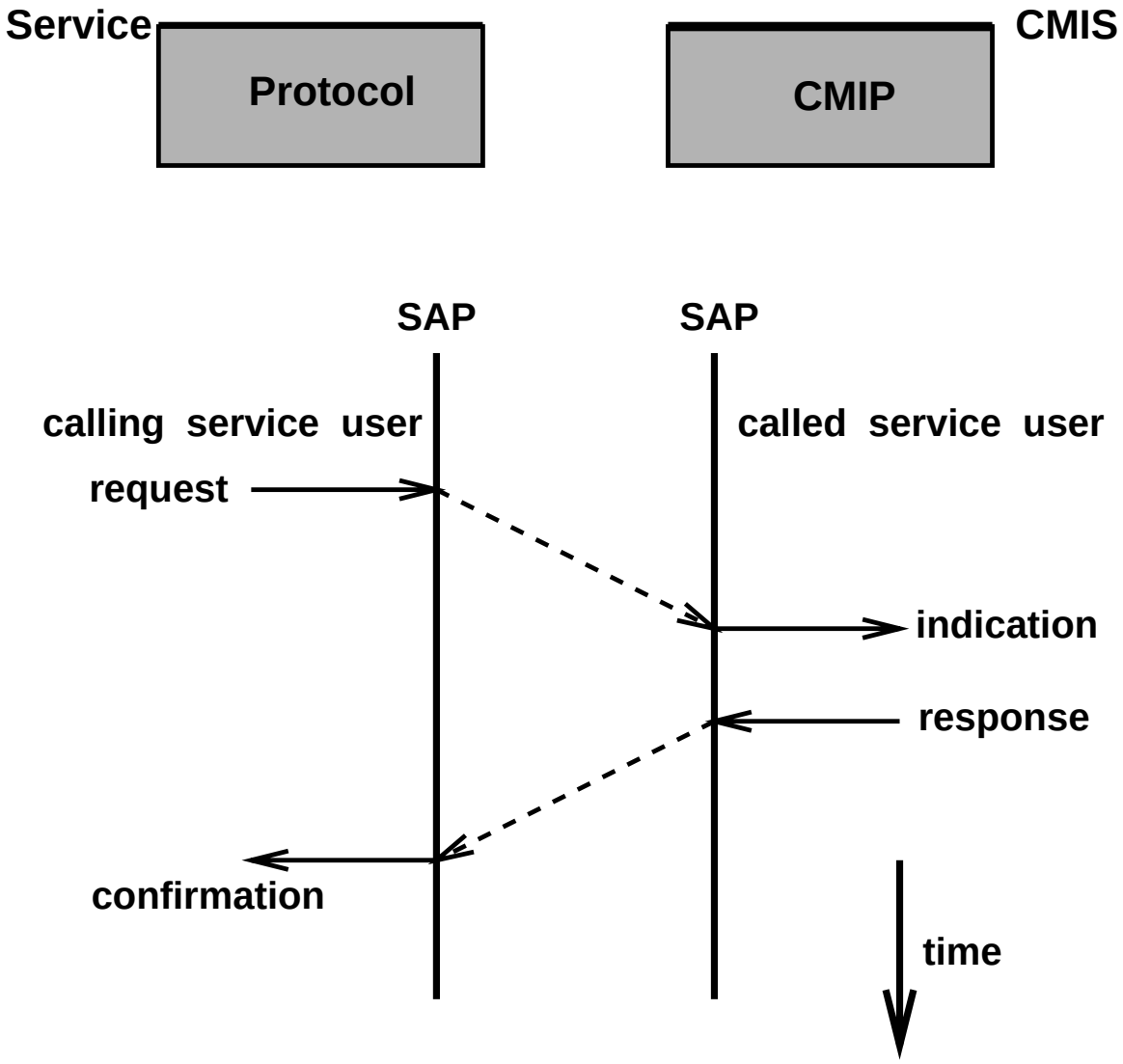
- **In the OSI world there are two mappings defined: a service over a full OSI stack (CMIP) and a connectionless one over Logical Link Control class 1 (LLC1)**

- **In the Internet world there is a third mapping to provide the service over TCP/UDP using a lightweight presentation protocol (CMOT)**

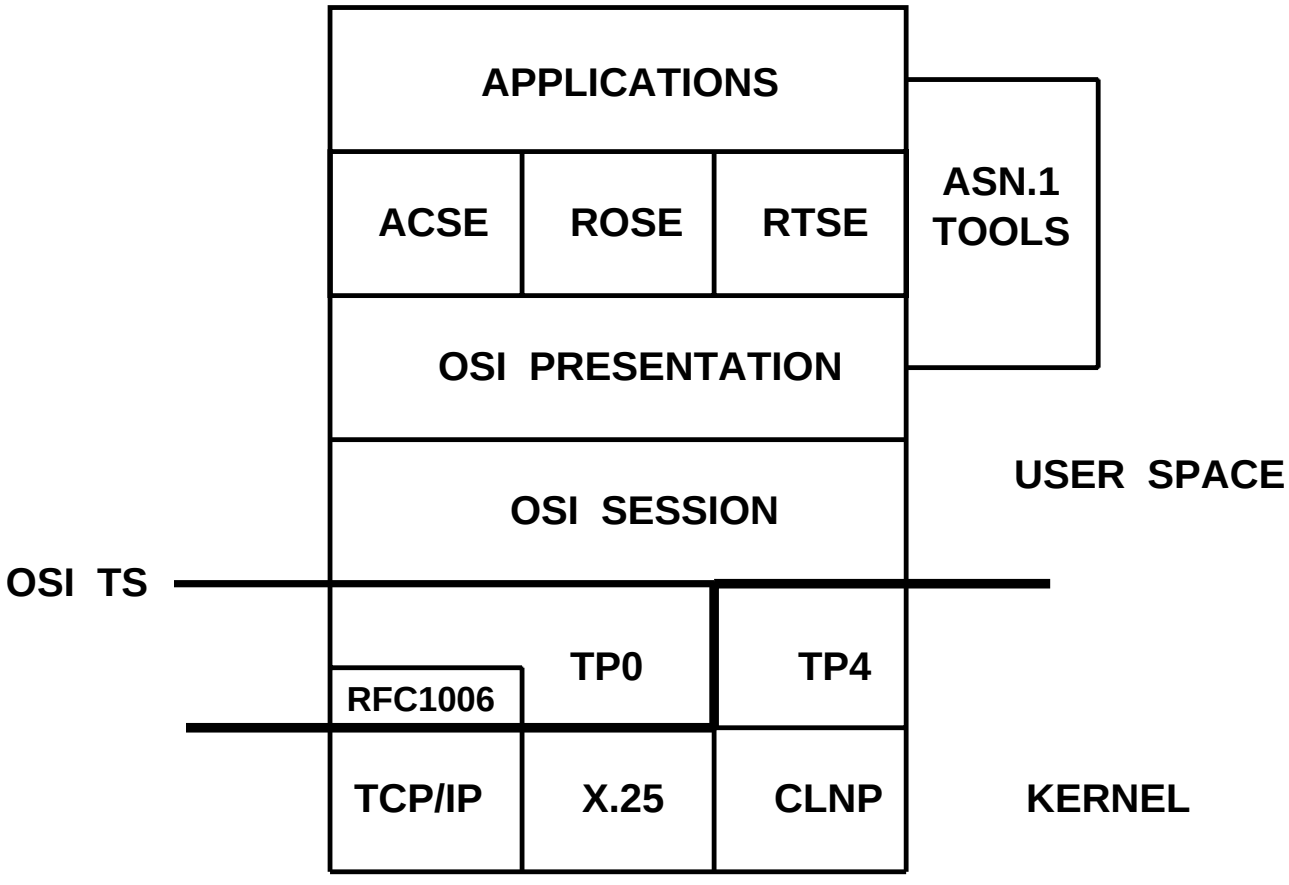
- **These do not interwork with each other; CMOT and CMIP applications are portable on each other's stack with the same API but will not work over the CMOL stack**

- **Proxy systems with full CMIP may be used for CMOL agents while management bridges may be constructed between CMIP and CMOT**

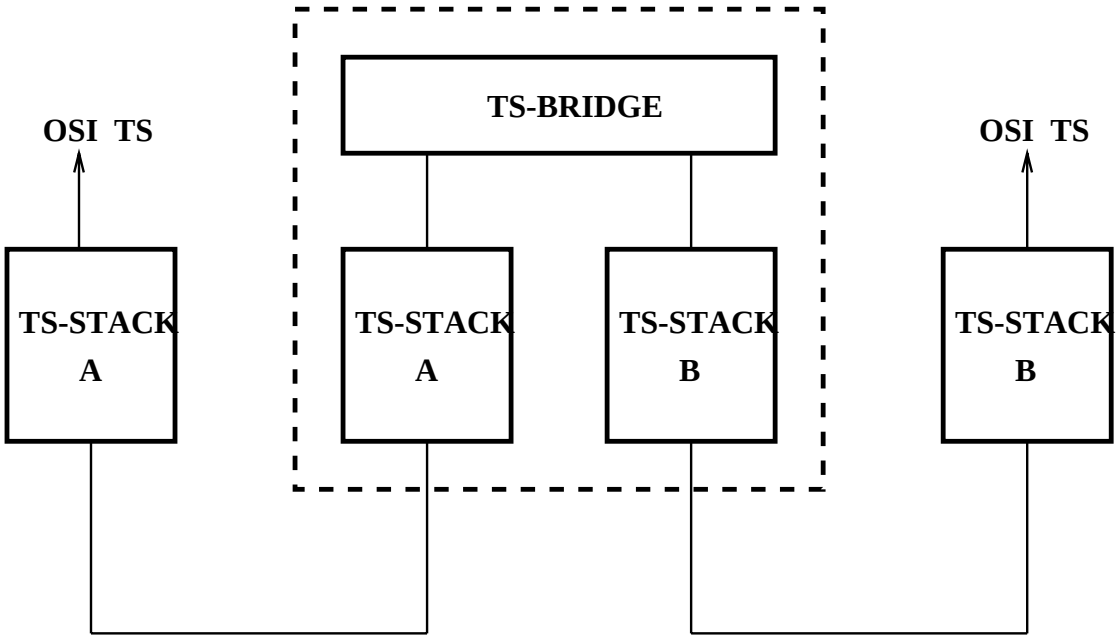
THE CONCEPT OF SERVICE-PROTOCOL



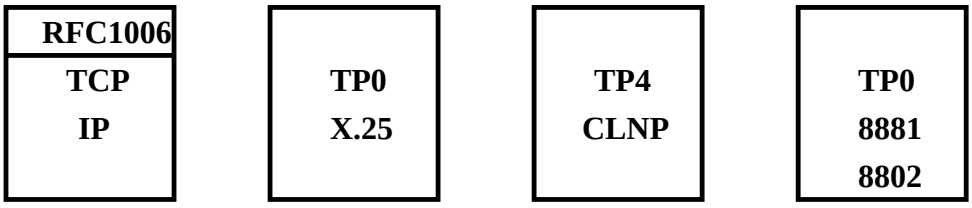
FULL OSI STACK



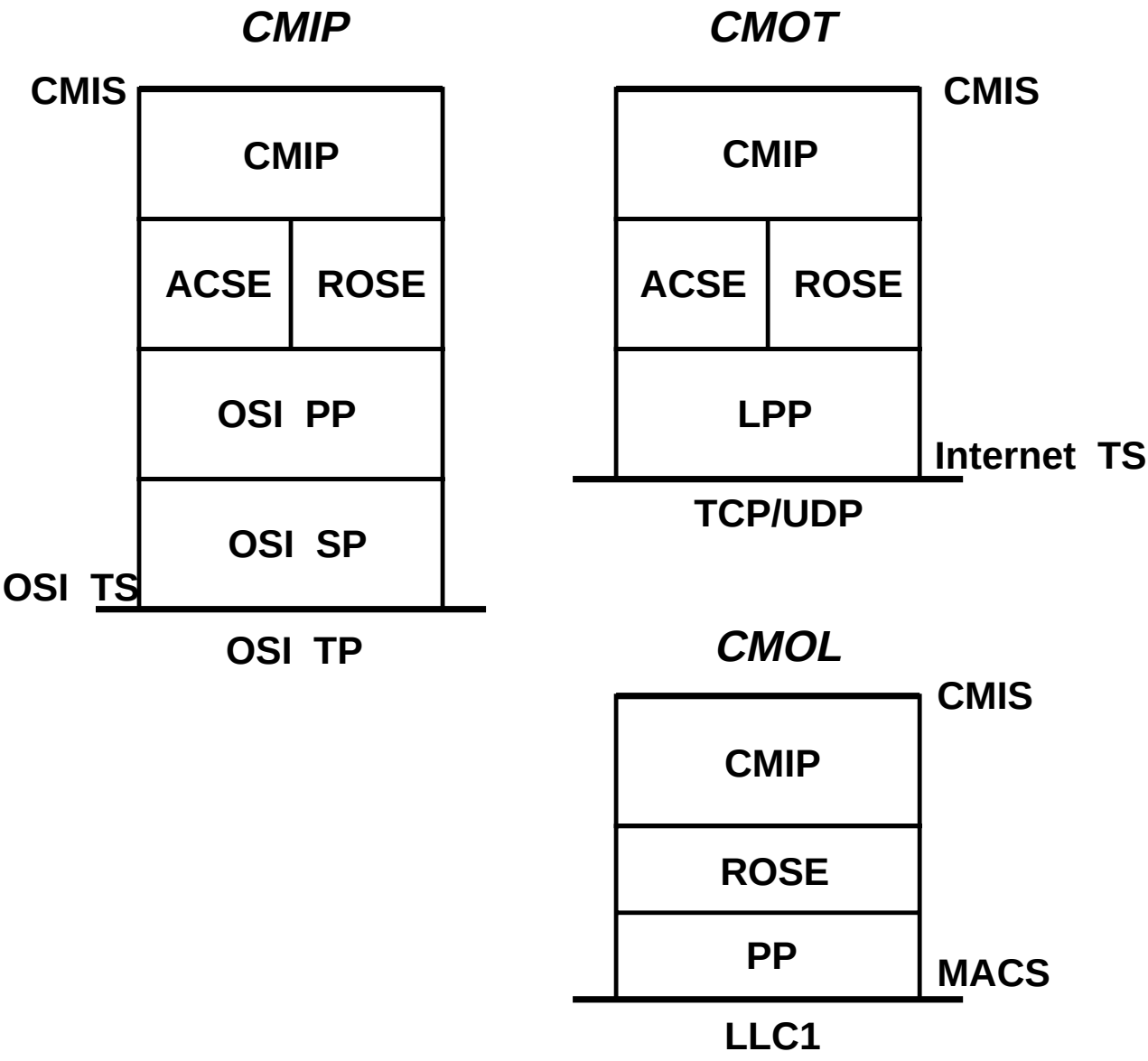
TRANSPORT - SERVICE BRIDGE



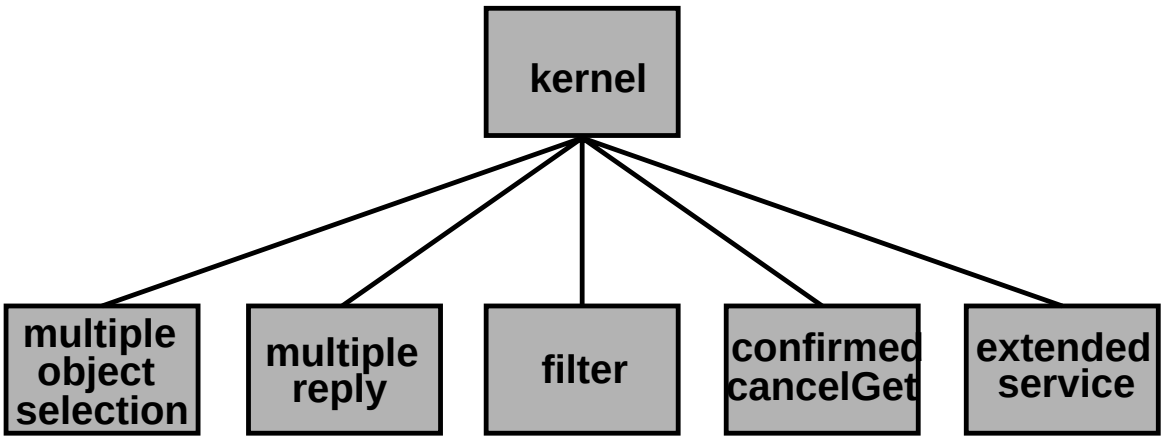
Example TS-Stacks



CMIP, CMOT & CMOL STACKS



CMIS FUNCTIONAL UNITS



multipleObjectSelection + multipleReply = scoping

extendedService: access to all PSAP services

CMIP STACK OPTIMIZATION

- **When the extended service functional unit is not provided, the OSI session layer services are not used (ACSE / ROSE do not use them)**

- **A minimal session layer can be provided with no functionality i.e. a simple pass through to the adjacent layer service - this may minimize drastically the program size**

- **If only the Basic Encoding Rules are to be used, the whole ASN.1 manipulation in upper layers can be have two only steps: internal representation <-> BER stream**

- **When using upper OSI layers over TCP/IP through the RFC 1006 method or the CMOT approach over TCP, the stream-oriented nature of TCP can be exploited for linked replies**

- **These can be deferred at the packetization level (RFC 1006 code) until the optimal subnet datagram size is all used**

MANAGEMENT SERVICE API IMPLEMENTATION MODELS

- **A typical way to provide a management service API is as a library of procedures implementing all the requests and responses**

- **In the case of CMIS, these should be Get, Set, Action, Create, Delete, CancelGet and their results (GetRes, SetRes etc.)**

- **Special parameters in the results can be used to indicate error conditions or linked replies**

- **For "reading" requests or indications, a common call can be provided to fill in a data structure, the latter being the union of all possible requests, results and errors**

- **Additional calls will be needed for establishing, releasing and aborting a management association; the association descriptor in most environments can simply be a file descriptor**

- **The interface can be both synchronous and asynchronous; most operating systems provide support for listening on a number of external data entry points (file descriptors)**

AN EXAMPLE API TO REALIZE A CMIS GET

```
int  M_Get (msd, invoke, objClass, objInst, scope, filter,
                                access, sync, nattrs, attrs, mi)
int  msd, invoke;
MIDentifier*  objClass;
MName*  objInst;
CMISScope*  scope;
CMISFilter*  filter;
External*  access;
CMISSync  sync;
int  nattrs;
MIDentifier  attrs [];
MSAPIndication*  mi;

int  M_GetRes (msd, invoke, linked, objClass, objInst, curTime,
                                nattrs, attrs, error, errorInfo, mi)
int  msd, invoke, linked;
MIDentifier*  objClass;
MName*  objInst;
char*  curTime;
int  nattrs;
CMISGetAttr  attrs [];
CMISErrors  error;
CMISErrorInfo*  errorInfo;
MSAPIndication*  mi;

int  M_WaitReq (msd, secs, mi)
int  msd, secs;
MSAPIndication*  mi;
```

THE USE OF ASN.1 THROUGH THE CMIS INTERFACE

- **The CMIP protocol layer may encode / decode most of the parameters in CMIS primitives; language data structures can be used through the interface**

- **The only parameters that need to be encoded / decoded by the service user are these are these whose type is unknown (ASN.1 ANY):**
  - **attribute, action and event report values**
  - **access control information**

- **ASN.1 compilers may be used to encode / decode both the CMIP Protocol Data Units (PDUs) and the above values when passed through the service interface**

THE XOM / XMP MANAGEMENT SERVICE API

- **There are many ways to implement a management service API**

- **The X/Open XOM/XMP API (OSI-Abstract-Data Manipulation / Management Protocols) is a common interface for both CMIS and the Internet SNMP**

- **As the latter is connectionless, the CMIS API has a connectionless orientation with a separate object providing association management**

- **The use of separate ASN.1 manipulation API (XOM) and a common CMIS/SNMP one results in a lot of work left to the service user - programming at this level can be tedious**

- **Applications using XOM/XMP will be portable across stacks providing this API; this is important as XOM/XMP tends to become the standard CMIS API**

GENERAL INFRASTRUCTURE

GENERAL INFRASTRUCTURE

- **This general infrastructure has two aspects:**
  - **support for polling in real-time**
  - **support for fully asynchronous event-driven applications**
  - **support for transparent abstract syntax handling**

- **The second aspect is not specific to management, it could be relevant to any real-time application with external communications needs**

- **The third aspect is again not specific to management, it could be relevant to any other OSI or other application that uses ASN.1**

POLLING SUPPORT

- **By polling support is meant the capability to request "wake-ups" in real-time**

- **Management applications need polling support for two reasons:**
  - **managers may need to poll agents periodically to detect changes in the absence of notifications**
  - **agents may need to poll "dump" real resources in order to support notifications**

- **This needs a common mechanism because of the way most operating systems support this facility i.e. only one wake-up may be pending for each process at any time**

COMMUNICATION NEEDS

- **Management applications have external communication needs:**
  - **managers need to talk to agents**
  - **agents need to talk to managers, "loosely coupled" real resources, systems for which they act as proxy and subordinate agents**
- **Activities triggered through these communications may coincide with internal polling requests**
- **There are two ways to organize internally a complex management application with respect to external communications:**
  - **service every external request immediately using a concurrent execution paradigm (threads, tasks)**
  - **serialize all external and internal requests and take each to completion**
- **The former is suitable for a synchronous while the latter should be used with an asynchronous remote execution paradigm**

CONCURRENT EXECUTION PARADIGM

- **Concurrent execution paradigms allow to service immediately every external request**

- **This is achieved through concurrent tasks or execution threads within the same process**

- **Every new activity triggered from an external or internal event constitutes a new task or execution thread**

- **The remote execution paradigm can be synchronous i.e. having remote procedure call semantics as this will not prevent other activities (remote operations take usually much more than local ones to complete)**

- **The problems with this approach are:**
  - **internal locking is needed to maintain information integrity and consistency**
  - **this may lead to deadlocks so deadlock avoidance mechanisms are needed**
  - **debugging real-time concurrent applications is very difficult**
  - **finally, there is no task/thread mechanism supported by commonly used operating systems =>**
    impact on portability

ASYNCHRONOUS EVENT-DRIVEN EXECUTION PARADIGM

- **In an asynchronous, event-driven execution paradigm, external or internal requests are serialized and taken to completion on a first-come first-served basis**

- **This is achieved through a central coordinating mechanism that undertakes all external listening and the scheduling/servicing of timer events**

- **The remote execution paradigm should be asynchronous in order not to delay other pending activities**

- **This approach does not need a task/thread facility but relies on a mechanism that enables to block on external communication endpoints and unblock only on external or timer events**

- **Such mechanisms are provided by most operating systems and this enhances portability**

- **The only problem with this approach is that asynchronous remote operations should be used for efficiency - this introduces the notion of state**

THE COORDINATOR - KNOWLEDGE SOURCE MODEL

- **A fully asynchronous event-driven scheme can be realized through the Coordinator - Knowledge Source abstraction**

- **The Coordinator is a central application object that receives all external events and schedules/services timer alarms**

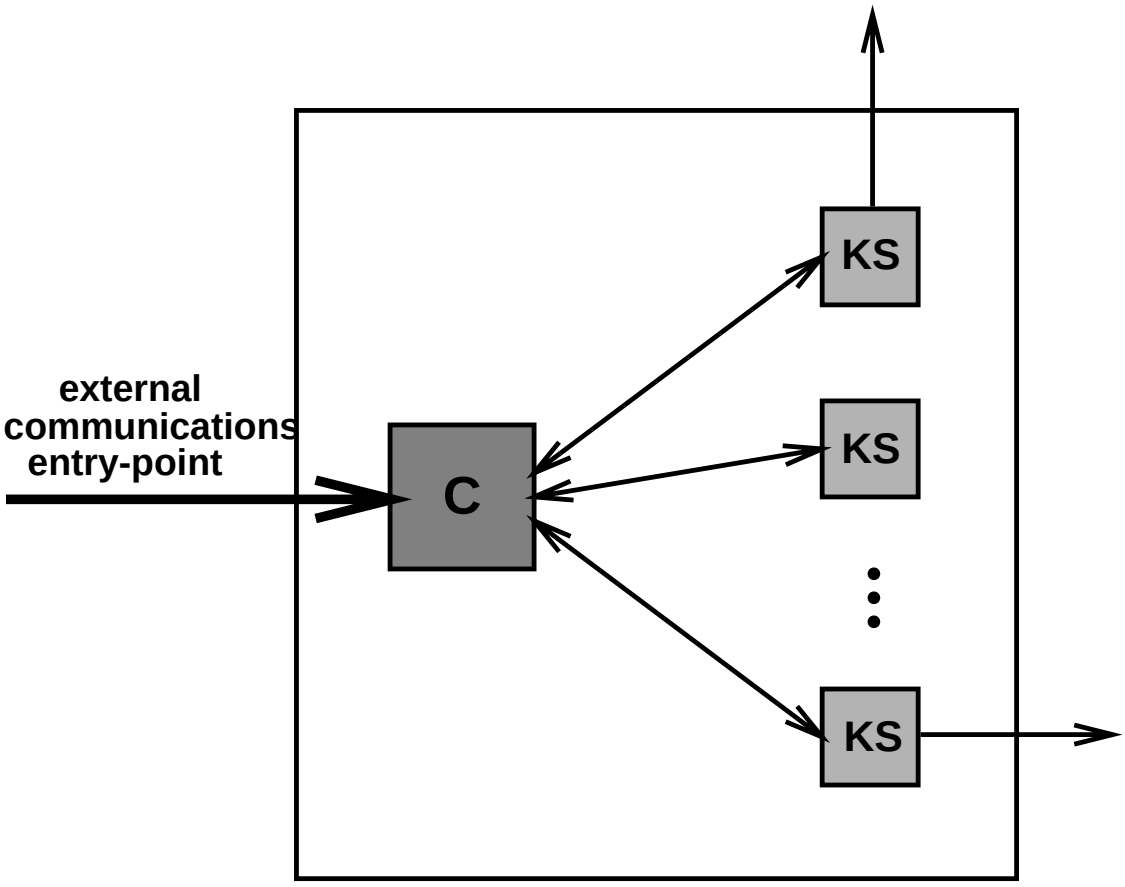
- **The Knowledge Source is a general application object that is capable of registering external communication endpoints on which is is told when events occur and of requesting "wake-ups" (timer alarms) in real-time**

- **The relationship of those is one to many: there is always one only Coordinator while there may be many Knowledge Sources in an application.**

- **The model assumes that other application objects should NOT do blocking listen on external communication endpoints as they will block the whole application**

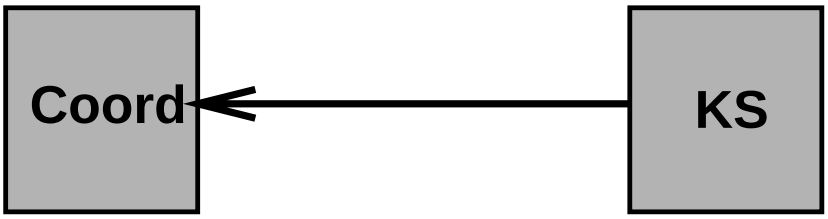
- **In such a scheme it is easy to introduce concurrent execution if desired as the Coordinator exercises centralized control**

THE COORDINATOR - KNOWLEDGE SOURCE MODEL



C: Coordinator
KS: Knowledge Source
→ Messages to other processes

THE KS -> COORDINATOR INTERFACE



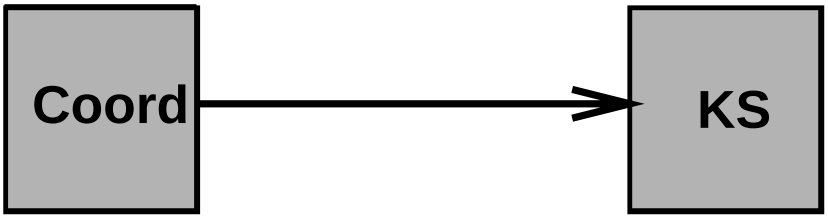
- **Register / deregister a communication endpoint:**

  - **coord -> registerCommEndpoint (ks, cepId)**
  - **coord -> deregisterCommEndpoint (ks, cepId)**

- **Schedule / cancel real-time wake-ups (possibly many for eack KS, distinguished through a "token"):**

  - **coord -> scheduleWakeUps (ks, token, interval, forNotifications)**
  - **coord -> cancelWakeUps (ks, token)**

THE COORDINATOR -> KS INTERFACE



- **Callbacks for activity (i.e. data waiting) on a communication endpoint, a timer event or application shutdown:**

  - **ks -> readCommEndpoint (cepId)**
  - **ks -> wakeUp (token)**
  - **ks -> shutdown (cepId)**

OPTIMIZING POLLING

- **In management agents, polling is mostly used to support notifications for "dump" real resources**
- **Notifications should only be generated if the event reporting and logging action is active i.e. if there are event discriminator and log managed objects in the local MIB**
- **These objects could notify the Coordinator of their presence through register/deregister methods**
- **The scheduleWakeUps call contains a boolean "forNotifications" argument which, if true, will result in waking-up the knowledge source only if needed**
- **That way, unnecessary processing and possibly bandwidth consumption are avoided**

THE COORDINATOR - KS INTERACTION

- **After all initialization in an application has finished, the Coordinator listens for all external or internal (timer events) activity**

- **In applications acting as agents, there is a special knowledge source that understands the CMIS management service - its communication endpoints are the application PSAP and established management association endpoints**

- **There may be other knowledge sources in agents for talking to "loosely coupled" resources or subordinate agents and in managers for talking to other agents**

- **During the lifetime of an application, more communication endpoints may be registered/deregistered and timer events scheduled/cancelled**

- **Before an application exits, knowledge sources with registered communication endpoints are notified in order to release them gracefully**

THE COORDINATOR MAIN LOOP

- **The Coordinator has methods to act as timer and terminate signal handlers and also a method serving as the central listening loop**
- **The logic of the latter is to serialize all activities due to external or internal events**
- **timer or terminate signals are disallowed when something else is in progress**
- **The main loop is shown next in C-like pseudo-code notation**

THE COORDINATOR MAIN LOOP

```
for ( ever )
    block at all external communication endpoints

    if ( unblocked because of timer event )
        continue

    for ( all the external communication endpoints )

        if ( there is data waiting on this endpoint )
            block signals
            notify the related knowledge source
            unblock signals
```

TRANSPARENT ASN.1 HANDLING

- **In OSI management, attribute, action and notification types are ASN.1 object identifiers**
- **Their values may be instances of any ASN.1 syntax according to the type**
- **ASN.1 compilers convert internal to external syntax representations and vice versa i.e. programming language structures <-> abstract syntax ones**
- **When programming management applications, it would be nice to avoid the explicit use of such conversions and program in terms of language structures**
- **These conversions can be handled transparently through a table-driven approach and a generic ASN.1 type ("attribute")**

SYNTAX MANIPULATION METHODS

- **Such tables can be parsed and build an internal representation that maps object identifiers to friendly names and syntax**

- **A set of methods for every syntax should be provided that will enable to manipulate instances of it generically:**
  - **an encode method, converting an internal to the abstract representation**
  - **a decode method, performing the opposite conversion**
  - **a parse method, converting a string to the internal representation**
  - **a print method, performing the opposite conversion**
  - **a free method, freeing an internal representation**
  - **a compare method, comparing two instances of the type**
  - **a get-next-element method, returning the next member of multi-valued syntaxes**

THE OID - SYNTAX TABLES

- **It is possible to deal with ASN.1 generically if a mapping is provided between attribute, action and notification identifiers and related ASN.1 syntax**

- **Two tables are needed:**
  - **a general object identifier table used to convert identifiers to friendly names (e.g. the last textual component)**
  - **an object identifier / syntax table used as above and also for a mapping to the syntax associated with the type**

- **The former should be used for general tags, classes, name bindings and packages (no associated syntax) while the latter for attributes, actions and notifications**

- **These are parsed and internal representations of the mappings are built, "pointing" also to the syntax manipulation methods**

HOW THE TABLES ARE USED

- **The tables are used first to convert object identifiers to user friendly names and vice-versa e.g.**

  2.9.3.2.3.13 <-> system

- **For every type with associated syntax, the syntax methods can be manipulated:**

  - **encode/decode for manipulating values to be sent or received through the network**
  - **parse for parsing string representation of values in manager GUIs**
  - **print for printing values in manager GUIs or elsewhere**
  - **free for freeing values when no longer needed**
  - **compare for filtering**
  - **get-next-element for add/remove and filtering of multi-valued attributes**

AN EXAMPLE OID TABLE

```
#format
#
# name:                          OID

joint:                           2

# general management tags

ms:                              joint.9
smi:                             ms.3
part2:                           smi.2

smi2ManagedObjectClass:          part2.3
smi2Package:                     part2.4
smi2Parameter:                   part2.5
smi2NameBinding:                 part2.6
smi2AttributeID:                 part2.7
smi2AttributeGroup:              part2.8
smi2Action:                      part2.9
smi2Notification:                part2.10

# managed object classes
top:                             smi2ManagedObjectClass.14
system:                          smi2ManagedObjectClass.13
discriminator:                   smi2ManagedObjectClass.3
eventForwardingDiscriminator:    smi2ManagedObjectClass.4
log:                             smi2ManagedObjectClass.6
logRecord:                       smi2ManagedObjectClass.7

# conditional packages
allomorphicPackage:              smi2Package.17

# name bindings
discriminator-system:            smi2NameBinding.1
log-system:                      smi2NameBinding.2
logRecord-log:                   smi2NameBinding.3
```

AN EXAMPLE OID / SYNTAX TABLE

```
#format
#
# name:                 OID                   :syntax

discriminatorConstruct: smi2AttributeID.56   :CMISFilter
administrativeState:    smi2AttributeID.31   :AdministrativeState
operationalState:       smi2AttributeID.35   :OperationalState
availabilityStatus:     smi2AttributeID.33   :AvailabilityStatus

objectClass:            smi2AttributeID.65   :OID
allomorphs:             smi2AttributeID.50   :OIDList
nameBinding:            smi2AttributeID.63   :OID
packages:               smi2AttributeID.66   :OIDList

discriminatorId:        smi2AttributeID.1    :Integer
destination:            smi2AttributeID.55   :DestinationAddress

logId:                  smi2AttributeID.2    :Integer
logFullAction:          smi2AttributeID.58   :LogFullAction
maxLogSize:             smi2AttributeID.62   :Integer
currentLogSize:         smi2AttributeID.54   :Integer
numberOfRecords:        smi2AttributeID.64   :Integer

logRecordId:            smi2AttributeID.3    :Integer
loggingTime:            smi2AttributeID.59   :UTCTime

managedObjectClass:     smi2NameBinding.60   :OID
managedObjectInstance:  smi2NameBinding.61   :DN
eventType:              smi2NameBinding.14   :OID
eventTime:              smi2NameBinding.13   :UTCTime
```

AUTOMATING SYNTAX METHOD GENERATION

- **Most of the ASN.1 syntax manipulation methods can be automatically generated using an ASN.1 compiler**
- **These can be encode, decode, free and get-next-element**
- **Pretty-printers and parsers can be automatically produced but as they are bound to produce ugly string representations, it is better to be hand-coded**
- **The compare method could be automatically generated in most cases but buried-in semantics may necessitate hand-coding**

THE GENERIC ATTRIBUTE

- **All the functionality provided by the syntax manipulation methods could be encapsulated by a generic ASN.1 object, to be called Attribute**

- **This can be instantiated with knowledge of its syntax, so it could automatically provide support for encoding/serializing, decoding, pretty-printing, filtering, set/add/remove**

- **This type of object can be used for representing attribute, action and notification values in both agents (MOs) and managers (access APIs)**

- **It may perform a table look-up at instantiation to get access to its syntax methods, or alternatively, specific objects for every syntax could be derived with built-in knowledge**

- **The latter can inherit behavior from the generic class and they can be automatically produced through an attribute compiler**

THE GENERIC ATTRIBUTE INTERFACE

- **The main methods of the generic attribute type which provide an interface for manipulating ASN.1 syntaxes generically are:**

  - **encodedValue = attr -> encode ()**
  - **decodedValue = attr -> decode (encodedValue)**
  - **stringValue = attr -> print ()**
  - **attr -> clear ()**
  - **copyValue = attr -> copy ()**
  - **value = attr -> get ()**
  - **result = attr -> set (newValue)**
  - **result = attr -> setDefault (dfltValue)**
  - **result = attr -> add (addValue)**
  - **result = attr -> remove (remValue)**
  - **result = attr -> filter (filterType, assertedValue)**
  - **attr = construct (syntaxName, initialValue)**
  - **attr = construct (typeOid, encodedValue)**

THE GENERIC ATTRIBUTE INTERFACE

- **The generic attribute type enables to program in terms of language structures while it also enables to encode and decode values**

- **The copy method can be realised by using encode/decode at a moderate performance cost - the provision of a syntax specific copy method is avoided**

- **The get and set methods could be redefined in derived attribute types specific to a resource**

- **There are two ways to construct an instance, using the syntax name and initial value or the type object identifier and encoded value**

- **All this functionality can be fully automated through ASN.1 compilers**

MANAGEMENT AGENT INFRASTRUCTURE

THE NATURE OF MANAGED REAL RESOURCES

- **The managed real resources may be either physical or logical**
- **They may reside in an operating systems kernel, in communication boards, in user-space processes or even in remote systems managed in a proxy fashion**
- **A two-way communication is needed between managed objects and the resources they encapsulate: access for monitoring and control and also asyncronous reporting of events (for "clever" resources)**
- **The managed object should always provide an up-to-date and consistent view of the corresponding resource through the management interface**

REAL RESOURCE ACCESS METHODS

- **For access purposes, real resources fall in the following two categories with respect to the corresponding managed objects / agent:**

  a. **"tightly coupled" - resources that reside in a common address space e.g. in local memory, shared memory, kernel's virtual memory etc.**

  b. **"loosely coupled" - resource that are not in a common address space e.g. in other user space processes, subordinate agents etc.**

- **Access to them from the managed object can be:**

  a. **"upon external management request" - the values are fetched only if requested through the management protocol**

  b. **"cache ahead" - the managed object polls the resource periodically and updates itself with fresh values**

  c. **"through asynchronous reports" - these are sent from the resource to the managed object according to activity**

ADVANTAGES / DISADVANTAGES

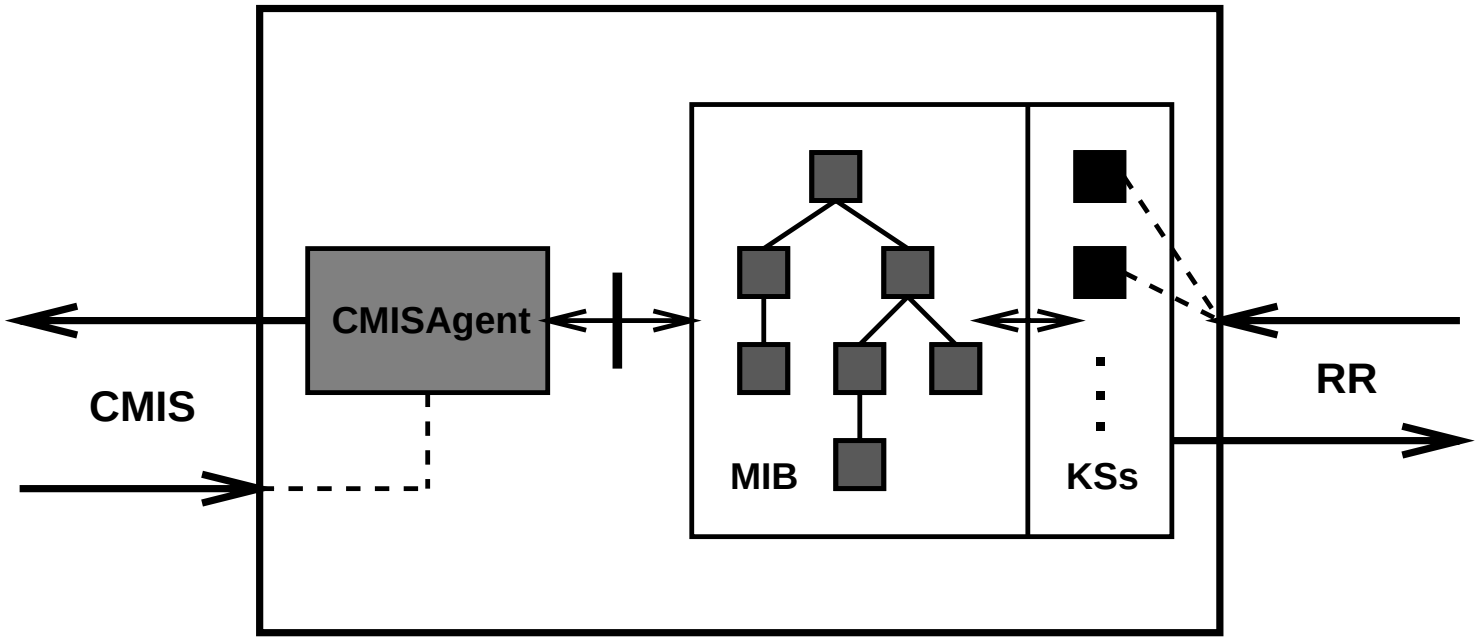
- **"upon external management request" - induces the least overheads, response time may be slow for loosely coupled resources and cannot support notifications for "dump" ones**

- **"cache ahead" - incurs processing even when no managers are interested, response time is fast but looses in information timeliness, necessary to support notifications for "dump" resources**

- **"through asynchronous reports" - again incurs processing even when no managers are interested, best if it can be tailored and useful mostly for notifications**

- **The OSI management model veers more towards the event-driven approach, so processing in agents/resources to support it is necessary**

AGENT APPLICATION DECOMPOSITION



CMIS: Front-end management interface

RR: Back-end loosely coupled real resource interface

KSs: Knowledge Sources

THE CMIS AGENT OBJECT

- **Every agent application needs an object that understands the management protocol, exercises access control, passes requests to the managed objects and returns replies**

- **In an OSI agent, such an object will handle association establishment and release, receive CMIS requests, address the right managed objects through scoping and filtering, exercise access control, handle synchronization and eventually return the results/errors**

- **It will also receive event reports from the notification function which it will forward to the specified destination after exercising access control**

- **This object is a specialized knowledge source whose initial endpoint is the application Presentation Service Access Point (PSAP) and subsequent ones relate to management associations**

OPERATION PROCEDURES

- **Get, Set, Action and Delete requests may involve scoping filtering and synchronization. The agent object does exactly the following:**

- **Checks the managed object class against the set of classes it knows - noSuchObjectClass is returned on error**

- **Checks the instance (managed object name) against the local containment tree - noSuchObjectInstance is returned upon error, a managed object pointer otherwise**

- **Checks the specified class is the instance class or any of its allomorphic classes - classInstanceConflict is returned upon error**

- **If scoping is other than baseObject, it applies scoping to find the scoped objects**

- **If a filter has been specified, this is applied to the selected object(s) to find those on which to perform the operation - access control rules apply with respect to read access to attributes**

OPERATION PROCEDURES (cont'd)

- **Access control rules apply for the operation:**
  - **If synchronization was requested to be atomic and access to at least one of the objects is denied, an empty reply is returned**
  - **Otherwise, for every object for which access is denied, a response with accessDenied error is returned**
  - **for Get and Set, this may be only a partial error in which case the operation should be performed for the attributes that access was granted**

- **If more than one objects have been selected and synchronization was requested to be atomic, there are two options:**
  - **a syncNotSupported error is returned if synchronization is not supported**
  - **if the operation cannot succeed for all the objects, an empty result is returned (see later on synchronization)**

- **Finally the requested operation is applied to the managed object(s) and a single reply or a series of linked replies followed by an empty reply is returned**

CREATE PROCEDURES

- **For a Create request, the agent may do the following:**
- **Check the class and return noSuchObjectClass if unknown and accessDenied if known but cannot be created through CMIS**
- **Apply access control rules and return accessDenied if the manager has no create rights**
- **If the object or the superior instance has been supplied, check that the object instance does not exist (possible duplicateInstance error) and/or the superior instance does exist (possible invalidObjectInstance error)**
- **If a reference instance has been supplied, check if it exists (possible noSuchReferenceObject error)**
- **Perform the Create operation by requesting the class object to create the instance (see later) and return the result/error**

EVENT REPORT PROCEDURES

- **As the real resources modeled by the managed objects in the agent operate, notifications may be generated**

- **These may be converted to event reports of log records according to the presence of event forwarding discriminators and/or logs in the local MIB**

- **Before a record is logged and/or before the agent is requested to forward an event report, access control rules apply**

- **If the agent is requested to forward an event report, it has to check first if it already has an association open to that destination [through the Application Entity Information (AEI) / Presentation Address]**

- **If it does, it should just send the event report on that association while if not, it should establish a new one to send the report**

- **If the confirmed service is requested, the agent should wait for an event report reply and possibly resend the event report after a time-out until it receives the reply**

CANCEL GET PROCEDURES

- **A manager may request the immediate termination of a series of linked replies through a Cancel-Get request referencing the invoke id of a previous Get request**

- **In a single-threaded execution paradigm, while servicing the Get request and before sending back each reply, the association should be checked for a pending Cancel-Get request**

- **If requests other than Cancel-Get are found, they should be queued for subsequent processing**

- **If a Cancel-Get request for the currently serviced or any of the queued Get requests is found, an operationCancelled error should terminate the Get request, followed by the Cancel-Get reply**

- **If the Cancel-Get request does not correspond to any Get request, a noSuchInvokeId error should be returned**

LOOSELY COUPLED RESOURCES

- **A request for an operation to a managed object modeling a loosely coupled resource with a "get-upon-management-request" scheme will result in interprocess or network communication**

- **The interface between the agent and managed object / real resource can be either synchronous or asynchronous**

- **In the case of a synchronous interface and a single-threaded execution paradigm the whole agent will block until the reply is received**

- **An asynchronous interface is needed to overcome this limitation, which means that both the managed object and the agent should maintain state information**

- **The agent needs in particular to maintain state of which operations are pending and introduce time-outs ("wake-ups") to avoid waiting for ever**

THE CONTAINMENT HIERARCHY

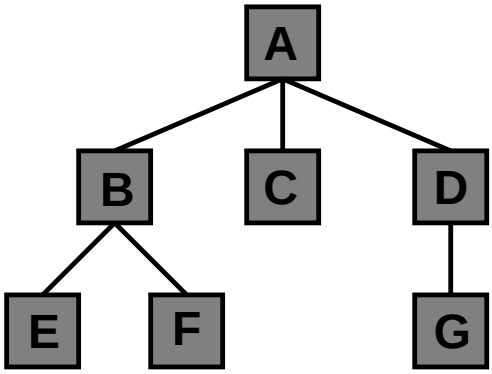
- **The containment tree can be represented internally as a physical binary tree of managed object instances**

- **Every object needs to keep sibling, subordinate and superior pointers**

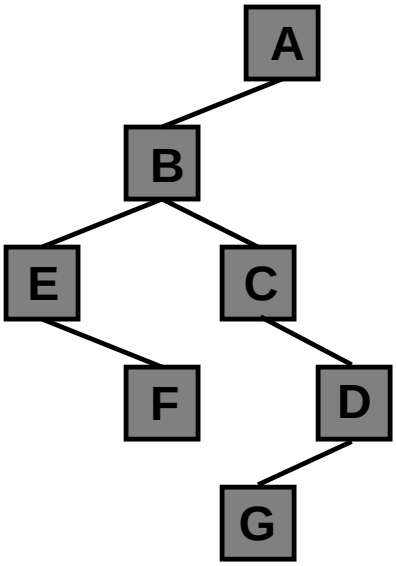
- **It should also keep its relative distinguished name in order to make possible the addressing of managed object through their distinguished name**

CONTAINMENT HIERARCHY

N-ARY ABSTRACT TREE



INTERNAL BINARY REPRESENTATION



MANAGED OBJECTS

- **Managed objects are represented internally as language objects that contain data (attributes) and offer a well-defined method interface**

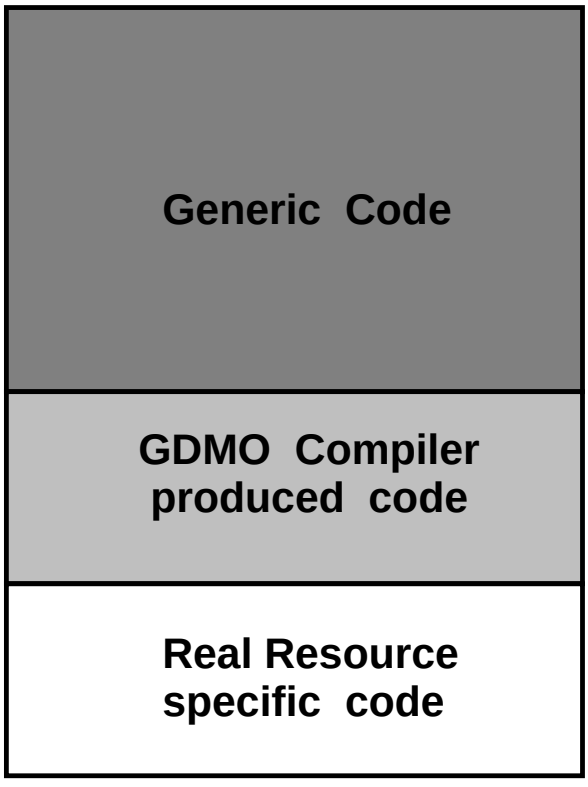
- **In fact, attributes are themselves language objects, subclasses of the general Attribute class**

- **The managed object code has three parts:**
  - **generic code that enables object and attribute addressing, scoping, filtering etc. and also keeps handles to attribute and class information**
  - **code specific to the managed object classes that constitute the instance; this may be produced by a GDMO compiler**
  - **code specific to the real functionality of the object i.e. code that interacts with the corresponding real resource or the rest of the MIB for control objects**

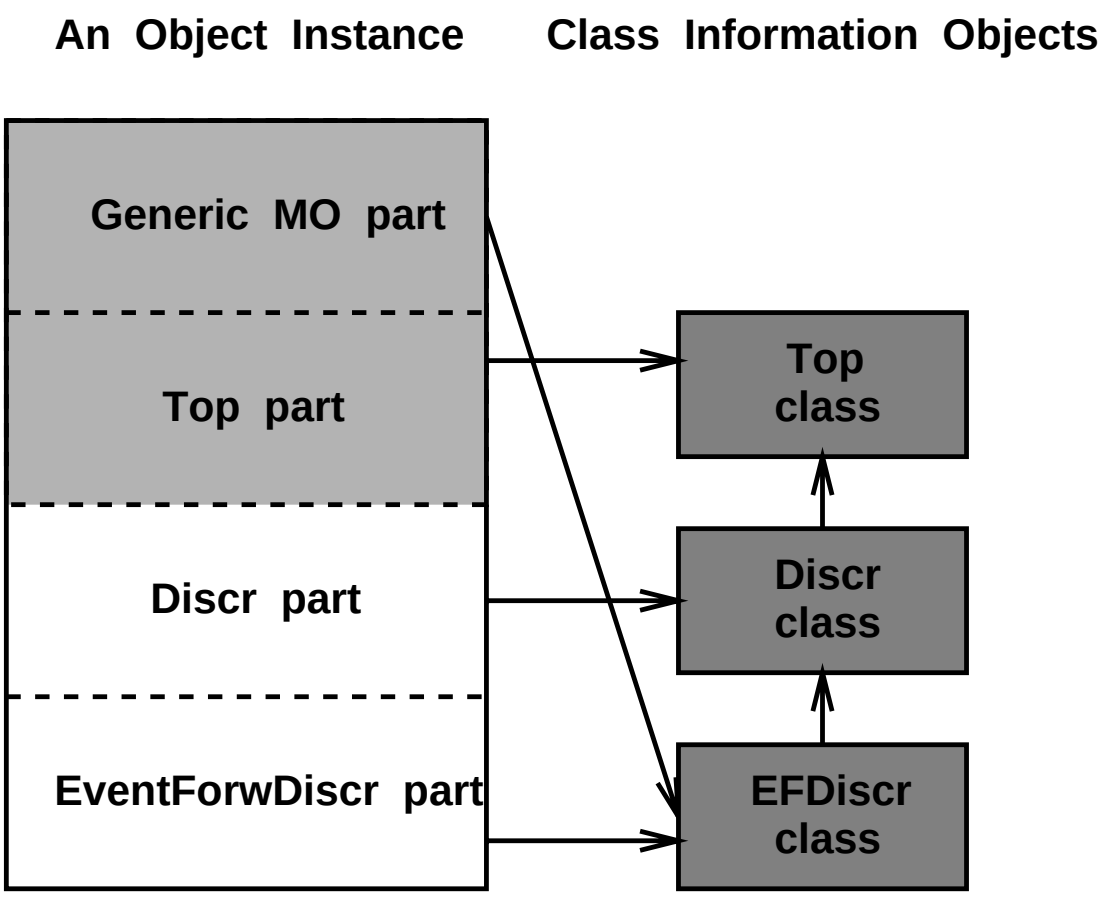
MANAGED OBJECTS (cont'd)

- **In every managed object there is both instance and class information**
- **Instance information is its data i.e. attributes and any other instance specific information**
- **The class information is common to all instances of a class and therefore it is internally represented as a separate object instance**
- **A managed object instance will contain code for at least two separate object classes (Top and the specific class realized)**
- **This code has access to the class object and instantiates the attributes of the basic and any conditional packages**
- **Handles to the class and attribute information are passed to the generic code which uses them to automate a lot of tasks**
- **An example of this internal organization is shown for an instance of the Event Forwarding Discriminator class**

A MANAGED OBJECT INSTANCE

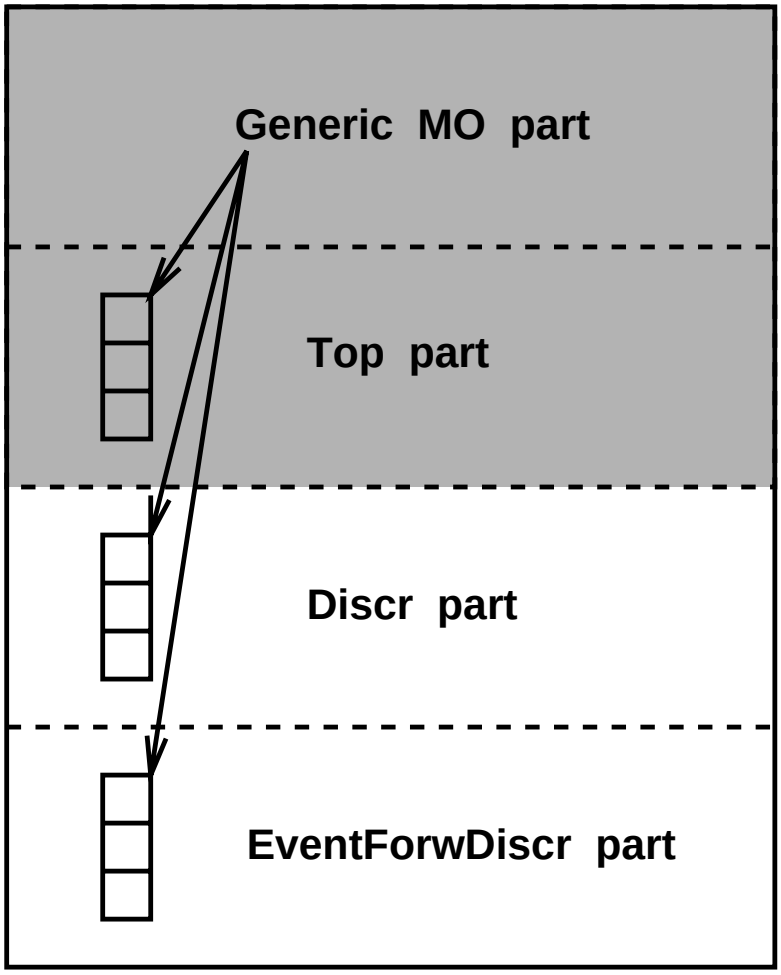


An EventForwardDiscriminator Example



An EventForwardDiscriminator Example (cont'd)

An Object Instance and its Attributes



MANAGED OBJECT CLASSES

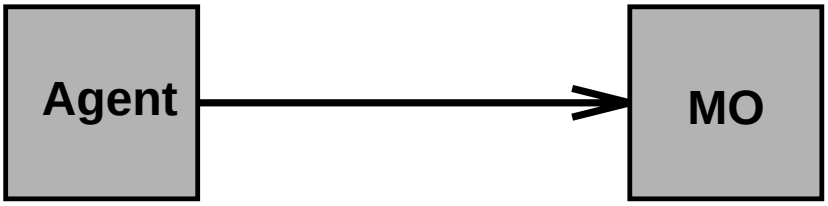
- **Every managed object class is represented internally as a separate object instance**
- **This is static i.e. it exists even if no managed object realizing the functionality of the class exists**
- **The agent object knows of all managed object classes supported in the managed system through a class table**
- **Class objects point to their parent in the inheritance hierarchy to form an internal representation of the inheritance tree**
- **This is in order to have access to full class information when the creation of an instance is requested through CMIS**

MANAGED OBJECT CLASSES (cont'd)

- **Class objects hold the following information about a class:**
  - **the class name/oid**
  - **name binding information**
  - **attribute and attribute group names/oids**
  - **information on which attributes are settable and the default value (if any)**
  - **notification names/oids and encoding information**
  - **action names/oids and decoding/encoding information**
  - **package information**
  - **allomorphic class information**
  - **name binding information**

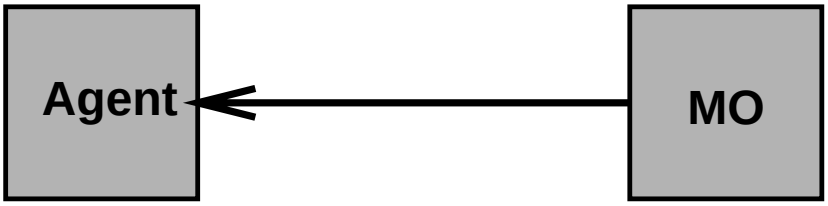
- **Most important, integer tags are held for attributes, actions and notifications; object identifiers in CMIS operations are automatically mapped on those for the resource-specific API**

THE AGENT - MANAGED OBJECT INTERFACE



- **mo = root_mo -> Find (objInst)**
- **result = mo -> CheckClass (objClass)**
- **moList = mo -> Scope (scope)**
- **result = mo -> Filter (filter)**
- **result = mo -> Get (class, attrIdList)**
- **result = mo -> Set (class, conf, attrId/value/modify list)**
- **result = mo-> Action (class, conf, actionType, actionInfo)**
- **result = mo-> Delete ( )**
- **result = classObj -> Create (superior, rdn, reference, initialAttrList)**

THE MANAGED OBJECT - AGENT INTERFACE



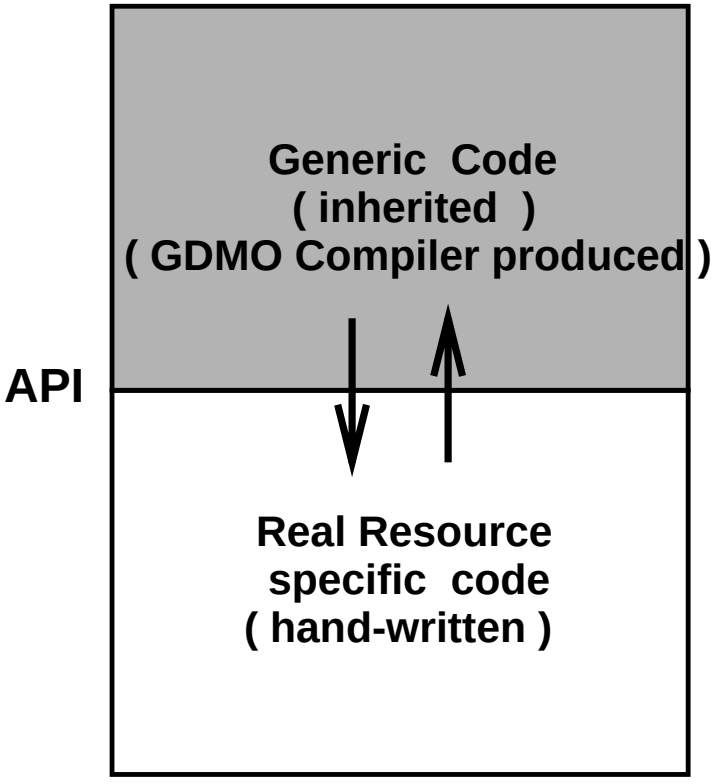
- **sendEventReport (objClass, objInst,**
  eventType, eventTime, eventInfo,
  destination, backUpDestinationList)
- **agent -> asynchronousOperationResult (operation)**

THE REAL RESOURCE ACCESS API

- **Every managed object instance has some code specific to real resource access**
- **There is an internal API through which this hand-written code is invoked**
- **This is a simple API as all the possible checks have already been done by the generic code**
- **In particular, attributes, actions and notifications are referred to through integer tags while only language data structures are passed across (instead of encoded ASN.1 values)**
- **That way, the amount of code that needs to be written by MIB implementors is absolutely minimized**

THE MANAGED OBJECT INTERNAL INTERFACE

A MANAGED OBJECT INSTANCE



THE MANAGED OBJECT INTERNAL INTERFACE

From the generic to the resource specific part:

- **get (attrIdList)**
- **set (attrId/value/modify list)**
- **result = action (actionType, actionInfo, actionResult)**
- **result = delete ( )**
- **result = buildEventReport (notificationId)**
- **refreshSubordinate (rdn)**
- **refreshSubordinates ( )**

From the resource specific to the generic part:

- **asynchronousOperationResult (operation)**
- **notification (notificationId)**

REFRESHING TRANSIENT MANAGED OBJECTS

- **Transient managed objects are these that can be created and deleted according to the operation of the underlying resource e.g. table entries, connections etc.**

- **If there are no notifications associated with their creation and deletion, the managed system does not have always an up-to-date image of them**

- **When such an object is referenced through its name in a management request or through scoping, the managed system should get an up-to-date image**

- **This can be achieved through a refresh method in the (non-transient) containing object**

- **Partitioning the knowledge that way accords to the nature of the classes in the containment relationship e.g. table and table entry, protocol machine and connection etc.**

- **It is noted that this may not be needed if notifications for creation and deletion are supported for those classes as the MIB image may then be up-to-date**

REAL RESOURCE INTERACTION

- **When a managed object needs to do periodic polling in order to support notifications or to implement a "cache-ahead" strategy, it can simply inherit the knowledge source functionality**

- **In the case of a "loosely-coupled" resource, it is likely that many managed objects will need to share the same communication channel**

- **In this case, a separate object i.e. a knowledge source is needed to handle that communication**

- **This involves initializing and terminating communication, understanding the protocol used, de-multiplexing messages and delivering them to the appropriate managed object**

- **This type of object constitutes the back-end of an MIB part, implementing the communication to the associated "loosely coupled" resource**

ADDRESSING A MANAGED OBJECT

- **In all the management requests through CMIS to the agent (with the exception of Create), a base managed is referenced through a distinguished name.**

- **This may be a global name i.e. from the Directory root to the object or a local name i.e. from the first object below system**

- **The validity of the non-local part of a global name can be checked against the value of the systemTitle system attribute**

- **In both cases, addressing the correct object in software is a matter of checking the components of the distinguished name against objects in the local containment tree**

- **A simple recursive algorith for doing this is shown next - note the refresh subordinate call to cater for transient objects**

THE ADDRESS METHOD

```
FindSubordinate ( rdn )

    refreshSubordinate ( rdn )

    for ( all the first level subordinates )
        if ( the rdn is equal to that of the subordinate)
            return subordinate

    return error


Find ( dn )

    if ( no dn )
        return this object

    subordinate = FindSubordinate ( rdn of the first
                                        dn component )
    if ( no such subordinate )
        return error

    return  subordinate -> Find ( next dn component )
```

CMIS SCOPING

- **The containment tree (Management Information Tree - MIT) is represented internally as a binary tree**
- **Evaluating scope is trivial and simply relegated to a pre- or post-order MIT search**
- **The only tricky aspect relates to non up-to-date instances of transient objects - the refresh subordinates method takes care of that**
- **A simple recursive algorithm for a pre-order search is shown next**

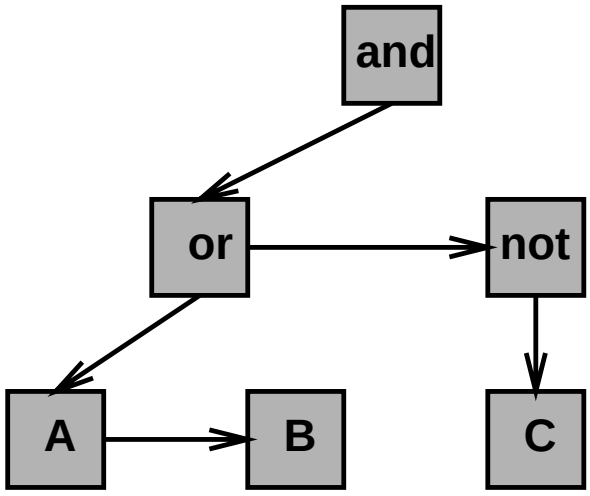
THE SCOPE METHOD

```
Scope ( scope, moList, level )

    switch ( scope )
        case wholeSubtree
            add this object to the list
            refreshSubordinates ( )

        case  individualLevel
            if ( level > level in scope )
                return // level exceeded
            if ( level = level in scope )
                add this object to the list
            else
                refreshSubordinates ( )

        case  baseToNthLevel
            if ( level > level in scope )
                return // level exceeded
            add this object to the list
            if ( level < level in scope )
                refreshSubordinates ( )

    if ( there is a subordinate )
        subordinate -> Scope ( scope, moList, level + 1 )

    if ( there is a sibling )
        sibling -> Scope ( scope, moList, level )
```

AN EXAMPLE FILTER

(A or B) and (not C)

Its internal representation



CMIS FILTERING

- **A CMIS filter is represented internally as a tree of objects that are either filter types (and, or, not) or filter items**
- **Evaluating a filter is trivial if the actual item eveluation is done by the attributes that have the syntax knowledge ("compare" method)**
- **Evaluation can be done through a simple recursive algorithm as shown next**
- **Before evaluating the filter, the object is told to refresh its attribute values**
- **If the filter evaluates to True and the subsequent operation is Get, the refreshed values should be used; this stateful approach is necessary for consistency and also also as an optimization**

THE FILTER METHOD

```
Boolean  Filter ( filter )

    switch ( filterType )
        case filterItem
            if ( the attribute in the assertion does not exist
                  or is not readable )
                return False
            if ( itemType = present )
                return True
            if ( attr -> decode ( assertedValue ) fails )
                return False

            status = attr -> filter ( itemType, decValue )
            free decValue and return status

        case filterAnd
            for ( all the filters in filter )
                if ( Filter ( thisFilter ) is False )
                    return False
            return True

        case filterOr
            for ( all the filters in filter )
                if ( Filter ( thisFilter ) is True )
                    return True
            return False

        case filterNot
            return Not ( filter in filter )
```

SYNCHRONIZATION

- **Implementing synchronization is very tricky**
- **There are two possible approaches:**
  - **a managed system internal two-phase commit with two passes**
  - **keeping previous values and restoring in case of error**
- **The second approach requires idempotency of operations but may still disrupt provided services**
- **The internal two phase-commit one is impossible to implement for managed objects representing resources in subordinate agents**

EVENT REPORTING AND LOGGING

- **The event reporting function can be implemented as part of the generic managed object**
- **Of course, it should make use of the necessary management control object classes (Discriminator,, Event Forwarding Discriminator, Log, Log Record etc.)**
- **A simple "notification" method call can be provided as a handle to trigger a notification**
- **This may be converted to an event report and/or logged**
- **Encoding the event report should be done only if a destination is found**
- **The ASN.1/BER representation is excellent support to implement easily managed object persistency (necessary for logs/log records and other objects)**

OTHER SYSTEMS MANAGEMENT FUNCTIONS

- **Any other systems management functions can be provided through the relevant management control managed objects**
- **Additional logic in the agent or the generic part will be needed to make use of them e.g. to support access control etc.**
- **For functionality that can be initiated by the specific managed object code, simple handles can be provided hiding realization details**
- **A good practice for the systems management functions is to be conditionally compiled / linked to reduce the size of agents**
- **The same holds for the CMIS functional units multiple object selection / reply (scoping), filter and cancel get**

THE GENERIC MANAGED SYSTEM

- **All the generic functionality described can be provided by a generic managed system platform which will provide a plug-in facility for new managed object classes**

- **The following should be provided:**
  - **the Coordinator / Knowledge Source and Generic Attribute infrastructure described**
  - **the CMIS agent object**
  - **the generic managed object and top classes**
  - **the SMI generic attribute types (counter, gauge, thresholds, tide-mark)**
  - **the DMI object classes for the systems management functions**
  - **functionality for the latter**
  - **a GDMO compiler to produce run-time support for managed objects, including specific new attribute types**

- **With this functionality in place, producing new agents means writing only the specific code for interaction with the underlying real resource**

MANAGER INFRASTRUCTURE

GENERIC MANAGER INFRASTRUCTURE

- **Managers may need to poll remote agents periodically and also to receive asynchronous event reports**
- **The general Coordinator - Knowledge source infrastructure may be used to provide this facility**
- **The centralized control of such a coordination mechanism may conflict with the requirements of a Graphical User Interface - an integrated approach is needed**
- **A raw procedural CMIS API, though adequate, requires explicit ASN.1 manipulation**
- **Higher-level object-oriented interfaces can be built using the general infrastructure described**
- **Two approaches, a "Remote" and a "Shadow MIB" are described**

INTEGRATING GRAPHICAL USER INTERFACES

- **Most manager applications with graphical user interfaces use simply synchronous management communications - in this case there is no problem**

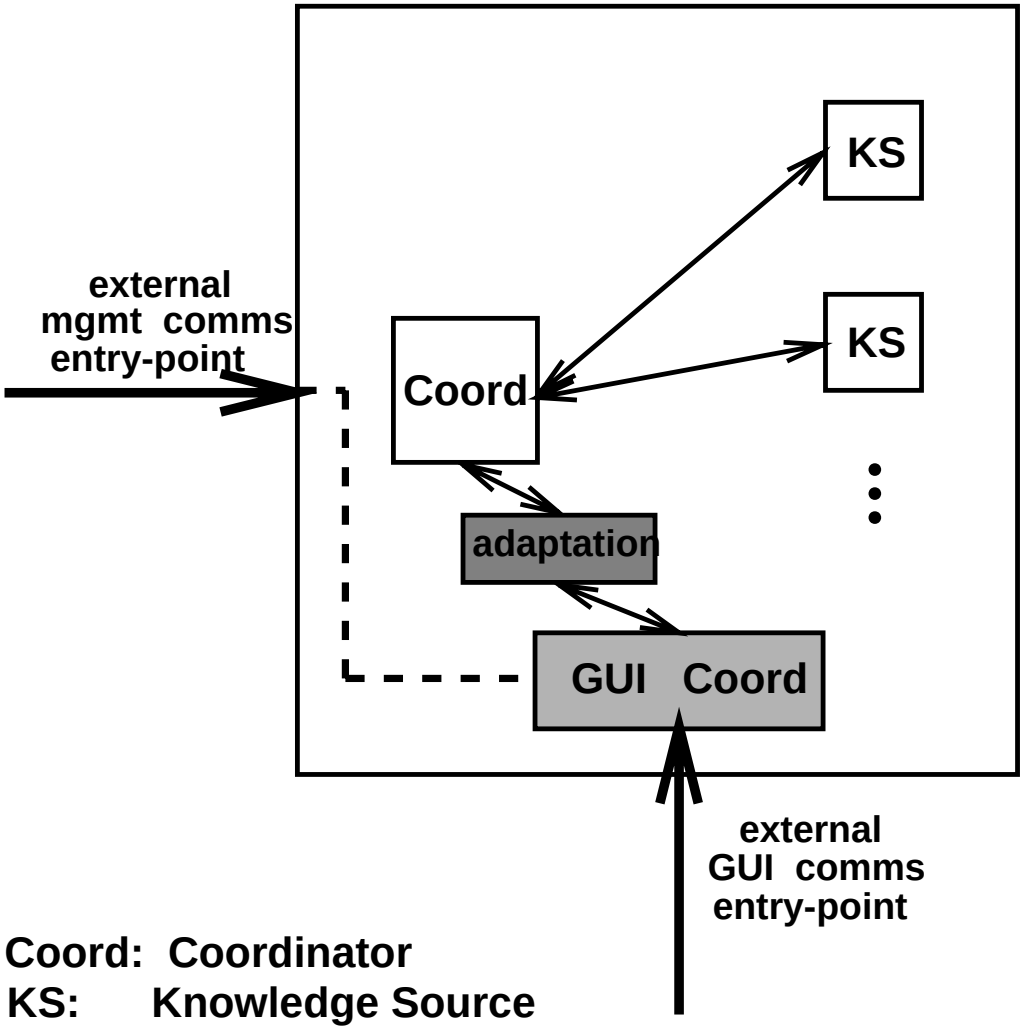
- **The problem begins when asynchronous operations are needed as most GUI mechanisms (X Windows etc.) use their own coordinating mechanism**

- **The solution is to make the management coordinating mechanism (the Coordinator object) able to export its communication enpoint mask**

- **In this case the GUI coordinating mechanism can be used as the central one with an additional adaptation object to pass events to management**

- **The same holds for the timer events in the case the "foreign" mechanism (GUI or other) needs to take control over these**

MANAGEMENT AND GUI INTEGRATION



Coord: Coordinator
KS: Knowledge Source

THE REMOTE MIB MODEL

- **The abstraction of a management association object can be provided through a RMIBAgent object**

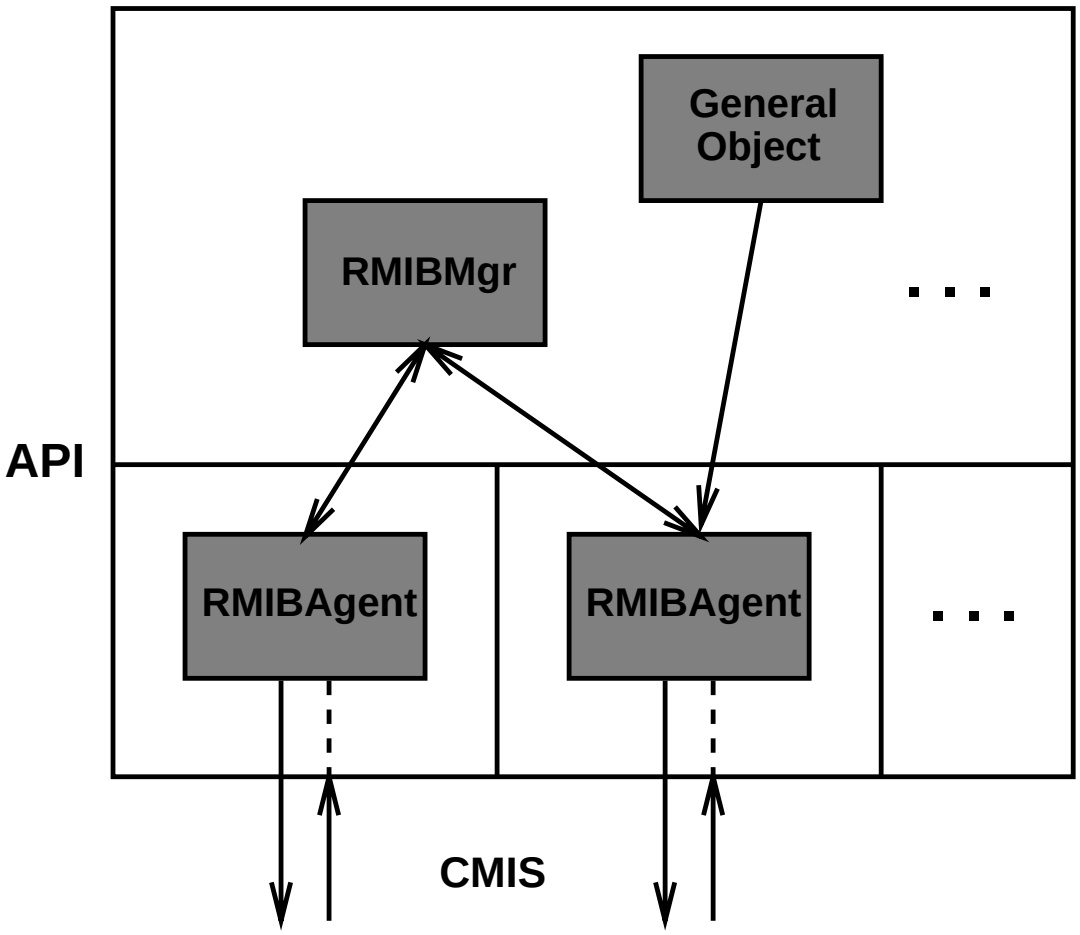
- **A manager object is also needed to provide callbacks for asynchronous results or event reports (RMIBMgr)**

- **The RMIBAgent can provide the following functionality:**
  - **handle association establishment and release**
  - **hide ASN.1 manipulation through the generic attribute**
  - **hide object identifiers through friendly names**
  - **hide distinguished name and CMIS filter complexity through a string-based notation**
  - **assemble linked replies**
  - **provide a high-level interface to event reporting, log-control or other functions**
  - **provide both a synchronous (RPC-semantics) and asynchronous service interface**

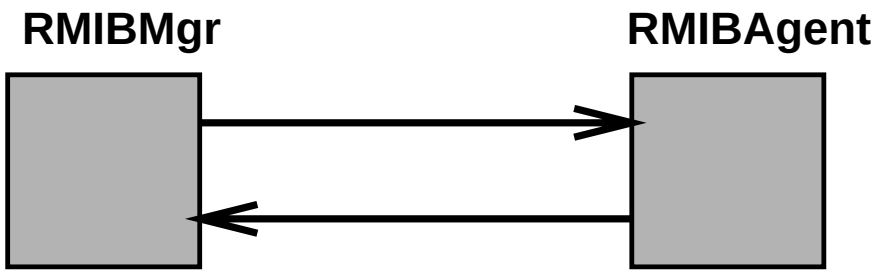
- **Any application object may use the synchronous interface while RMIBMgrs are needed for asynchronous callbacks**

- **A manager application needs at any time a number of RMIBAgents according to the number of remote MIBs (agents) it needs to access**

THE REMOTE MIB MODEL



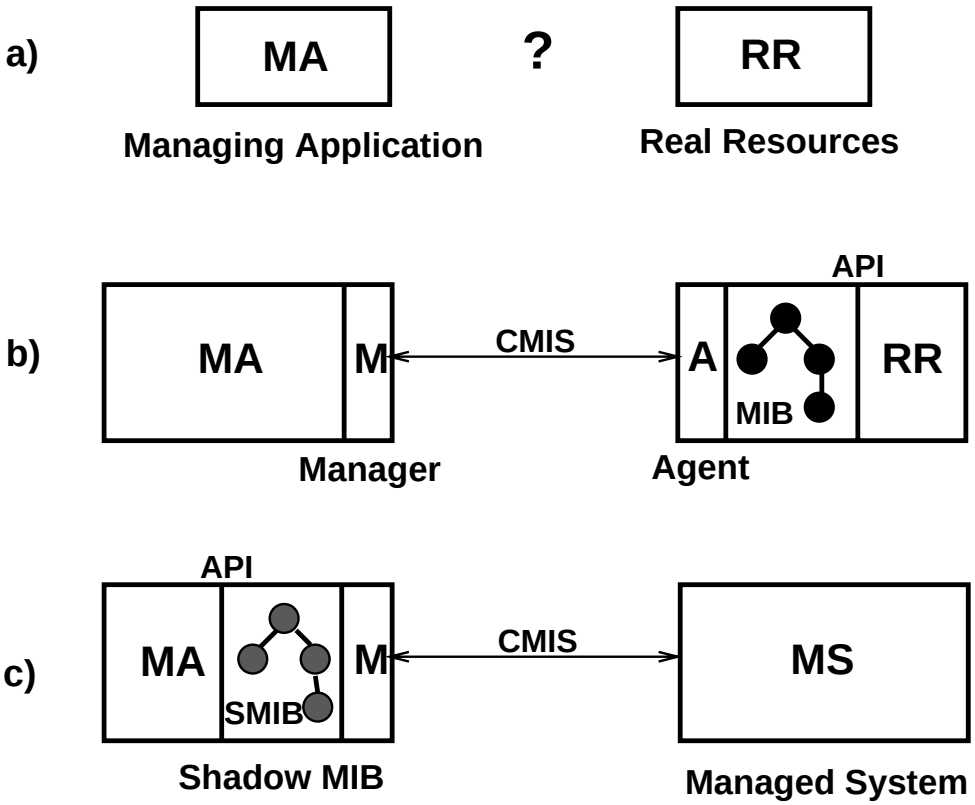
AN EXAMPLE EVENT REPORT INTERFACE



- **Requesting or stopping event reporting:**
  - **rmibAgent -> receiveEvent (eventType, objClass, objInst, manager)**
  - **rmibAgent -> receiveEvent (filterConstruct, manager)**
  - **rmibAgent -> stopReceiveEvent (eventType, objClass, objInst, manager)**
  - **rmibAgent -> stopReceiveEvent (filterConstruct, manager)**

- **Receiving event reports:**
  - **rmibManager -> eventReport (objClass, objInst, eventType, eventTime, eventInfo)**

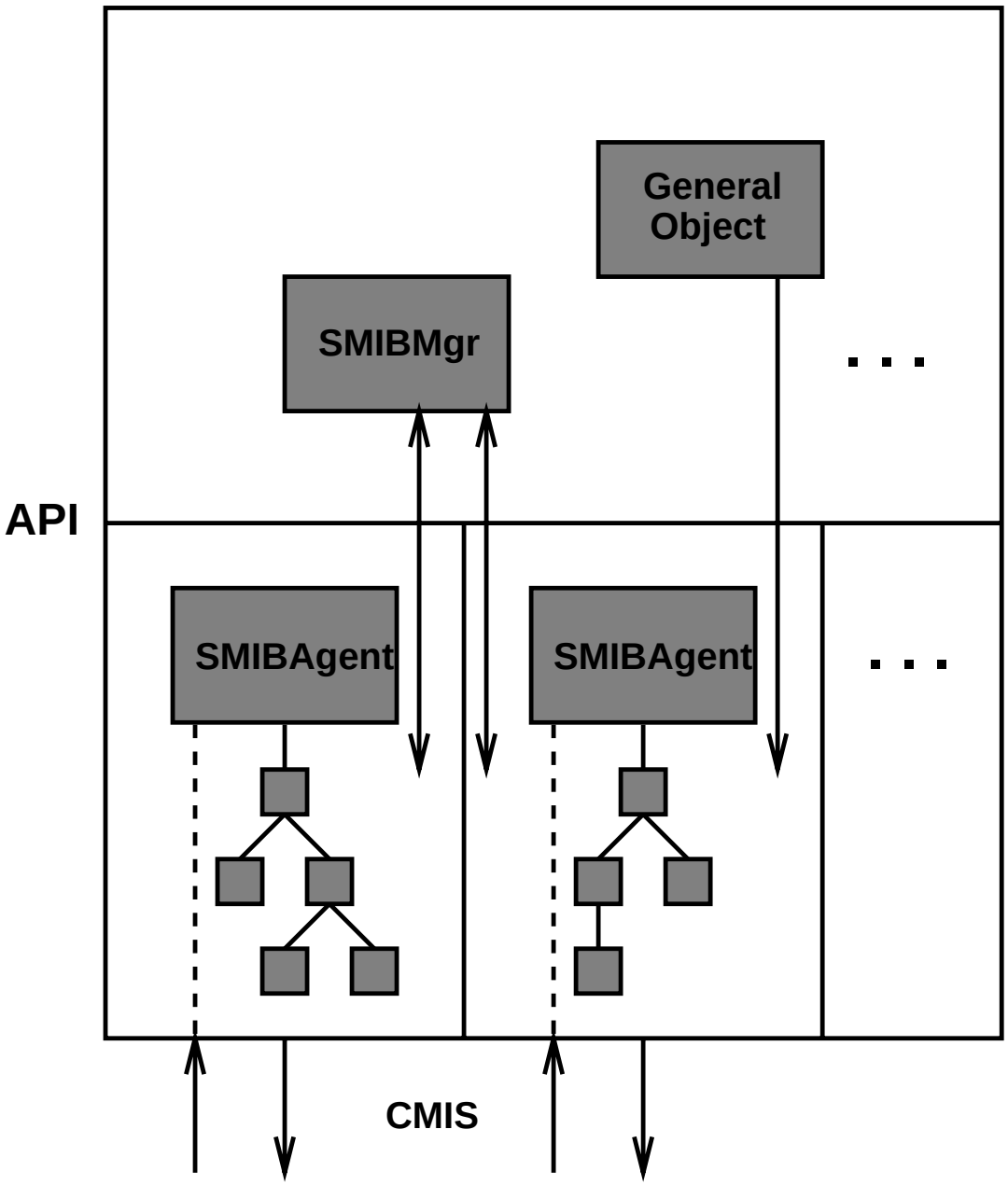
THE SHADOW MIB MODEL



THE SHADOW MIB MODEL

- **In the "Shadow MIB" model the abstraction of managed objects in the local address space is provided**
- **These are handled by a SMIBAgent while a SMIBMgr provides the interface for asynchronous callbacks**
- **Some of the CMIS power such as scoping and filtering is lost but it can be a good paradigm for rapid prototyping**
- **The local shadow managed objects can be updated through fetch-on-request, cache-ahead or event report schemes**
- **The use of a management protocol is almost completely hidden**
- **By adding location transparency through a "broker", the abstraction of a locally accessible global management information tree can be provided in true ODP fashion**

THE SHADOW MIB MODEL



THE GENERIC TMN APPLICATION

- **OSI Network Management provides the model for network management interactions but says nothing for the organization of network management systems**

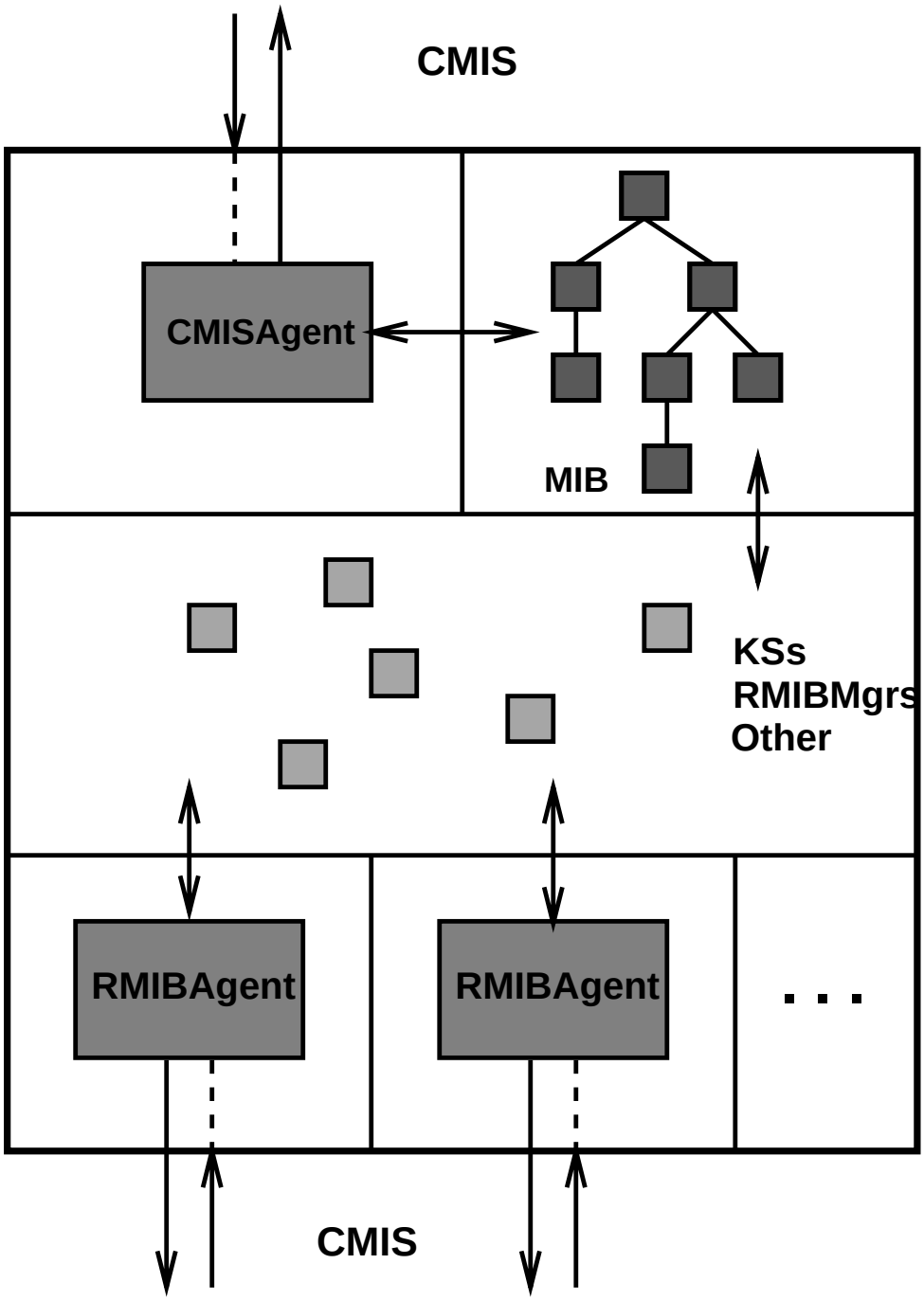
- **The CCITT Recommendation M.3010 is based on the principles of the OSI model to provide a model for higher-level management system organization**

- **In this management functionality is decomposed following a hierarchical layered approach as follows:**
  - **Network Element Management**
  - **Network Management**
  - **Service Management**

- **In this model, pure agents operate only at the lowest level of the hierarchy (network elements) with hybrid units in the other layers in hierarchical manager-agent relationships**

- **Following the object-oriented model described for decomposing OSI agents and managers, the structure of the generic TMN application is shown next**

THE GENERIC TMN APPLICATION



SUMMARY AND INFORMATION

SUMMARY

- **The OSI Management model offers powerful solutions that can be efficiently implemented**
- **It is though necessary to hide complex aspects behind well thought out APIs**
- **Object-oriented design methodology and implementation technology offer the solution to this problem**
- **Object-oriented management platforms can offer a very simple interface for MIB realization and access**
- **The same technology can be used to automate conversion between the OSI and Internet management approaches**

ACKNOWLEDGEMENTS

- **The research work presented has been carried out under the auspices of the RACE and EPSRIT European research programmes**

- **In particular, it has been carried out in the following projects:**

  - **RACE-II ICM - Integrated Communication Management**
  - **RACE-I NEMESYS - NEtwork Management using Expert SYStems**
  - **ESPRIT-III MIDAS - Management In a Distributed Application and Service Environment**
  - **ESPRIT-II PROOF - Primary Rate ISDN OSI Office Facilities**
  - **ESPRIT-I INCA - Integrated Network Communication Architecture**

THE OSI MANAGEMENT INFORMATION SERVICE (OSIMIS)

An openly available OSI management platform, implementing most of the infrastructure described and including applications

Uses the ISODE implementation of the OSI upper layers

Latest release 3.0 includes:

- **A full OSI CMIS/P based on the IS2 1991 standards version (in C) - interoperates with the Retix and NetView implementations**

- **General infrastructure for fully asynchronous event-driven applications and transparent ASN.1 manipulation - the Kernel (in C++)**

- **Generic Object-Oriented infrastructure to implement MIBs supporting ALL the functional units and the event report and log control functions - the Generic Managed System (GMS) (in C++)**

- **Generic Object-Oriented infrastructure to implement managers through the Remote MIB model (RMIB) (in C++)**

THE OSI MANAGEMENT INFORMATION SERVICE (cont'd)

- **A Generic X-Window based MIB browser application (in C++)**
- **Generic managers for all management operations (in C/C++)**
- **A full implementation of the OSI Internet MIB as in RFC 1214 in a non-proxy fashion (in C++)**
- **An experimental implementation of the ISO Transport Layer MIB (in C++)**

They will appear in the future:

- **A proxy implementation of the OSI Internet MIB**
- **A GDMO compiler**
- **Generic support for CMIS->SNMP conversion using the IIMC approach**
- **More systems management functions**

HOW TO GET AN OSIMIS COPY

- **There is the discussion forum <osimis@cs.ucl.ac.uk> to request information**

- **In order to subscribe send a request to <osimis-request@cs.ucl.ac.uk>**

- **If you simply want to get a copy:**

  - **use anonymous FTP to cs.ucl.ac.uk [128.16.5.31] or**

  - **use anonymous FTAM to 23421920030013 through IPSS or 20433450420113 through IXI with TSEL 259 (ASCII encoding) and**

  - **look in the directory osimis/**