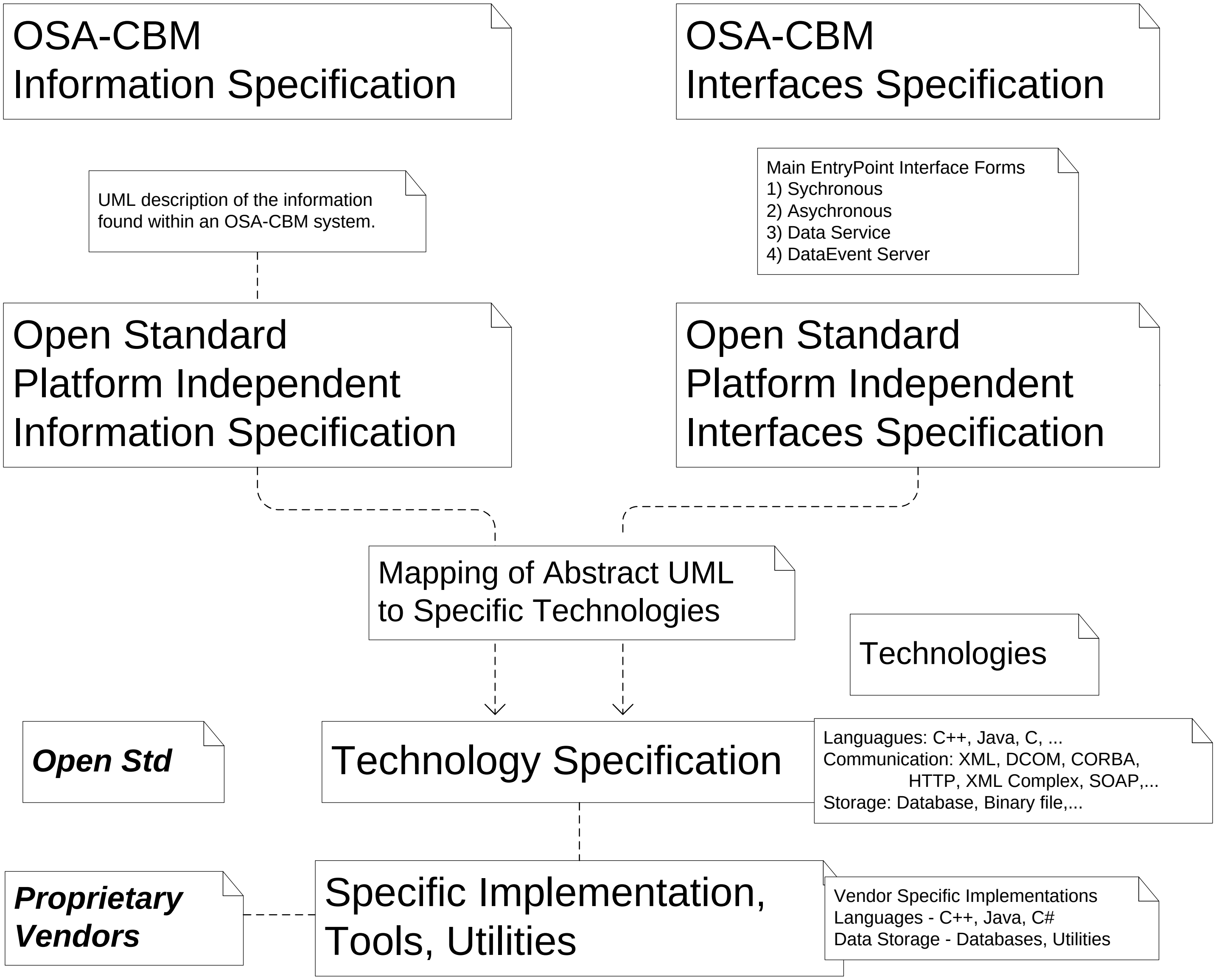
OSA-CBM stands for Open System Architecture for Condition Based Maintenance.
This specification is offered by the MIMOSA organization. Information on this organization can be found at www.MIMOSA.org.

Usage of this specification may only be done under the MIMOSA licensing agreement. It is open to the public usage only in accordance with the non-members' licensing right. It is open to MIMOSA members' usage in accordance with the members' licensing rights as held from 2002 and later. THIS WORK PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, see applicable license for complete details.

The version of OSA-CBM 3.1 corresponds to the OSA-EAI version 3.1.
This is the version reference in ISO 13374 which is a CBM standard.

This OSA-CBM is based on the work supported by the Office of Naval Research under Agreement No N00014-99-3-0011 OSA-CBM Boeing DUST.
This version has been modified from the original OSA-CBM 1.0 DUST program specification within the MIMOSA organizational process.
These modifications improved the specification in capability and compatibility to the MIMOSA OSA-EAI specification and its advancements.



OSA-CBM
Information Specification

OSA-CBM
Interfaces Specification

UML description of the information
found within an OSA-CBM system.

Main EntryPoint Interface Forms
1) Sychronous
2) Asychronous
3) Data Service
4) DataEvent Server

Open Standard
Platform Independent
Information Specification

Open Standard
Platform Independent
Interfaces Specification

Mapping of Abstract UML
to Specific Technologies

Technologies

Open Std

Technology Specification

Languagues: C++, Java, C, ...
Communication: XML, DCOM, CORBA,
HTTP, XML Complex, SOAP,...
Storage: Database, Binary file,...

Proprietary Vendors

Specific Implementation,
Tools, Utilities

Vendor Specific Implementations
Languages - C++, Java, C#
Data Storage - Databases, Utilities

This specification is designed for multi-technological implementation.
From this point the UML needs specific mappings into programming languages, network protocols, and database storage (e.g. MIMOSA OSA-EAI CRIS).
This document covers the abstract UML description of the specification.

This architecture splits the information specification, which defines the information that can be moved around in a CBM system, from the interfaces that can be used to move that information. This separates the information that is moved, stored, and processed from the mechanism that accomplishes these tasks.

An implementation of this technology will select applicable interface(s) and merge the information specification into a complete package.
The information specification and interface specifications as they are created will be found in other documentation.
Specific technological implementations may be vendor IP supplied tools and utilities. Such vendors are encouraged to become MIMOSA members.

Technology Specification

This document covers the OSA-CBM abstract UML specification.
It defines the core specification of the information found in a CBM system.
The Interface specification offers ways to move this information around.

A mapping effort is required to convert this specification into a technology representation that is verifiable. For example, a mapping of the UML Information Specification to XML will result in an XML schema that specifically defines the XML form of the data. The XML schema wil then be used to validate a system that is required to output OSA-CBM XML.

Notes on Compliance

Information Specification verses Interface Specification

The information specification describes the type of information found in a CBM system.
(OSA-CBM was developed in close connection with the MIMOSA OSA-EAI CRIS 3.0.)
The interface specification describes methods of moving this information around.
One development difficulty was that different technologies have different ways of achieving this goal. (For example, there are already existing standard ways of moving XML around.)

The initial goal for the OSA-CBM 3.1 technology mapping is an XML schema for the information content. Application developers may choose an existing technology for moving the XML around.

Compliance will be based on XML conforming to the standard schema.

Interfaces

Interface Types

1) Synchronous
2) Asychronous
3) Data Service
4) DataEvent Server

Why so many interface types? OSA-CBM is a specification that covers a broad technological base. The main aspect of value for OSA-CBM is its information specification. That information specification was designed with concepts in mind as how to map it to programming languages, transfer protocols, and data storage devices.

There are several interface types that are required for wide standard applicability. Each technological implementation will likely not implement every interface. Rather, the technology of choice will typically select the interface(s) by logical choice.

Example: A Web server returning XML over HTTP is Synchronous - Stateless.

Definitions

Interface
An Interface describes how information will be moved.
A request is made to get information from an object.
A notify is made to input information into an object

EntryPoint - the interface presented by an object to the outside world.
It provides direct access to the top level classes.
(For example, the DataEventSet and Configuration classes.)

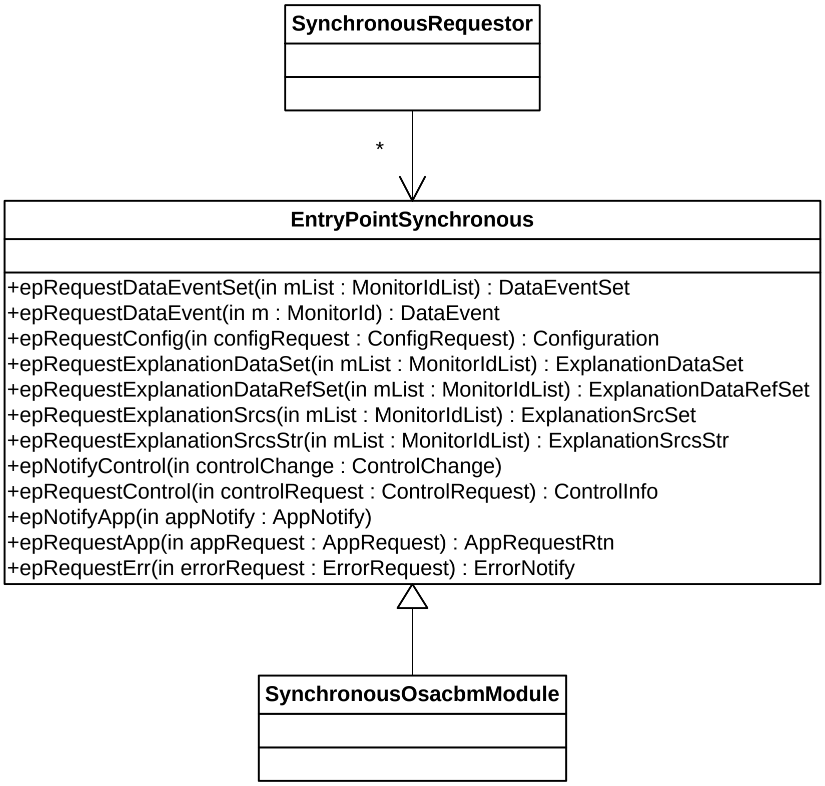
EntryPointSink - Asychronous data return path for requested information.

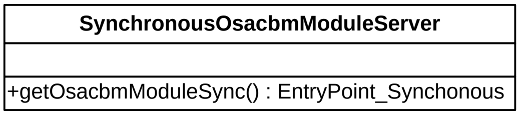
Synchronous Interface - Information is returned by the Request method.
Asynchronous Interface - Information is returned as avaliable via EntryPointSink.

Synchronous Interface

The synchronous interface returns data with the call.
It models the Web XML over HTTP fetch methodology

Example:
newData = moduleEPptr->requestDataEventSet( );


SynchronousRequestor
*
EntryPointSynchronous
+epRequestDataEventSet(in mList : MonitorIdList) : DataEventSet
+epRequestDataEvent(in m : MonitorId) : DataEvent
+epRequestConfig(in configRequest : ConfigRequest) : Configuration
+epRequestExplanationDataSet(in mList : MonitorIdList) : ExplanationDataSet
+epRequestExplanationDataRefSet(in mList : MonitorIdList) : ExplanationDataRefSet
+epRequestExplanationSrcs(in mList : MonitorIdList) : ExplanationSrcSet
+epRequestExplanationSrcsStr(in mList : MonitorIdList) : ExplanationSrcsStr
+epNotifyControl(in controlChange : ControlChange)
+epRequestControl(in controlRequest : ControlRequest) : ControlInfo
+epNotifyApp(in appNotify : AppNotify)
+epRequestApp(in appRequest : AppRequest) : AppRequestRtn
+epRequestErr(in errorRequest : ErrorRequest) : ErrorNotify
SynchronousOsacbmModule

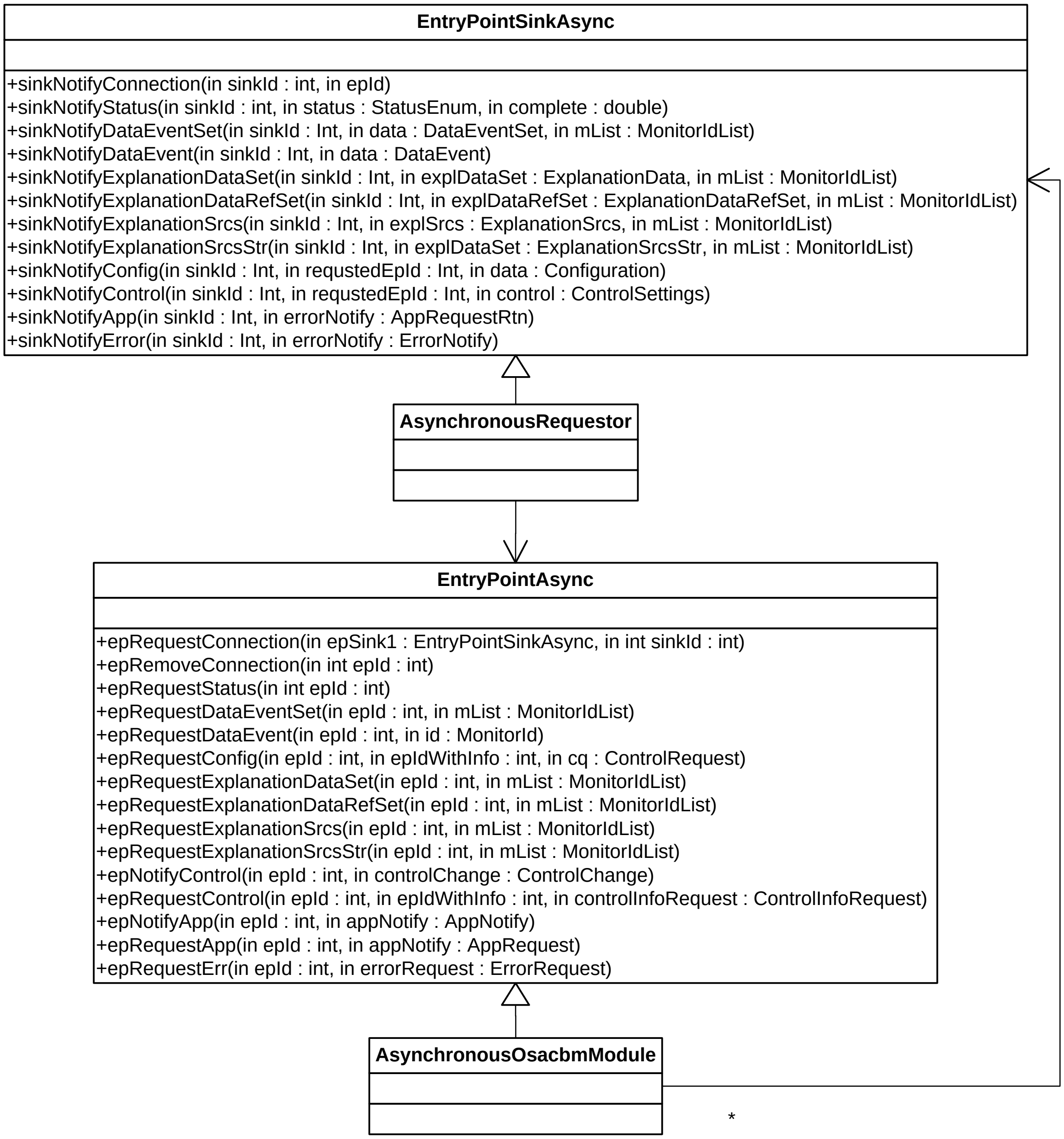

SynchronousOsacbmModuleServer
+getOsacbmModuleSync() : EntryPoint_Synchonous

The SynchronousOsacbmModuleServer interface is the suggested method
for creating many Synchonous servers for a specific Osacbm Module.

Asynchronous Connected Module Interface

The Asynchronous Interface:
1) Allows for any number of higher modules.
2) Two-way connection is established and maintained for duration of need.
The sinkId and epId are used for specific sink and entry point identification.
The two-way connection has several feature advantages:
2.1) It is typically faster in usage since the overhead of connection occurs only once.
2.2) It allows for three different modes of communication.
2.2.1) Return on request - main OSA-CBM 1.0 style of communication.
2.2.2) Return when threshold is exceeded. The connection can be setup for notification only on threshold crossing.
2.2.3) Push All - The lower module pushes data to the higher connected module for every frame without the need for a request beforehand.
3) Example method call interplay with the connection oriented Asynchronous Interface:

```
   // A higher module requests data from lower module that it previously established a connection with
   lpEp_lowerModuleWithData->requestDataEventSet( lpRequestingModuleEPSinkptr );
  // The lower module returns data when it is ready
  lpSink_higherModule->notifyDataEventSet( data );
```



EntryPointSinkAsync

```
+sinkNotifyConnection(in sinkId : int, in epId)
+sinkNotifyStatus(in sinkId : int, in status : StatusEnum, in complete : double)
+sinkNotifyDataEventSet(in sinkId : Int, in data : DataEventSet, in mList : MonitorIdList)
+sinkNotifyDataEvent(in sinkId : Int, in data : DataEvent)
+sinkNotifyExplanationDataSet(in sinkId : Int, in explDataSet : ExplanationData, in mList : MonitorIdList)
+sinkNotifyExplanationDataRefSet(in sinkId : Int, in explDataRefSet : ExplanationDataRefSet, in mList : MonitorIdList)
+sinkNotifyExplanationSrcs(in sinkId : Int, in explSrcs : ExplanationSrcs, in mList : MonitorIdList)
+sinkNotifyExplanationSrcsStr(in sinkId : Int, in explDataSet : ExplanationSrcsStr, in mList : MonitorIdList)
+sinkNotifyConfig(in sinkId : Int, in requstedEpId : Int, in data : Configuration)
+sinkNotifyControl(in sinkId : Int, in requstedEpId : Int, in control : ControlSettings)
+sinkNotifyApp(in sinkId : Int, in errorNotify : AppRequestRtn)
+sinkNotifyError(in sinkId : Int, in errorNotify : ErrorNotify)
```

AsynchronousRequestor

EntryPointAsync

```
+epRequestConnection(in epSink1 : EntryPointSinkAsync, in int sinkId : int)
+epRemoveConnection(in int epId : int)
+epRequestStatus(in int epId : int)
+epRequestDataEventSet(in epId : int, in mList : MonitorIdList)
+epRequestDataEvent(in epId : int, in id : MonitorId)
+epRequestConfig(in epId : int, in epIdWithInfo : int, in cq : ControlRequest)
+epRequestExplanationDataSet(in epId : int, in mList : MonitorIdList)
+epRequestExplanationDataRefSet(in epId : int, in mList : MonitorIdList)
+epRequestExplanationSrcs(in epId : int, in mList : MonitorIdList)
+epRequestExplanationSrcsStr(in epId : int, in mList : MonitorIdList)
+epNotifyControl(in epId : int, in controlChange : ControlChange)
+epRequestControl(in epId : int, in epIdWithInfo : int, in controlInfoRequest : ControlInfoRequest)
+epNotifyApp(in epId : int, in appNotify : AppNotify)
+epRequestApp(in epId : int, in appNotify : AppRequest)
+epRequestErr(in epId : int, in errorRequest : ErrorRequest)
```

AsynchronousOsacbmModule

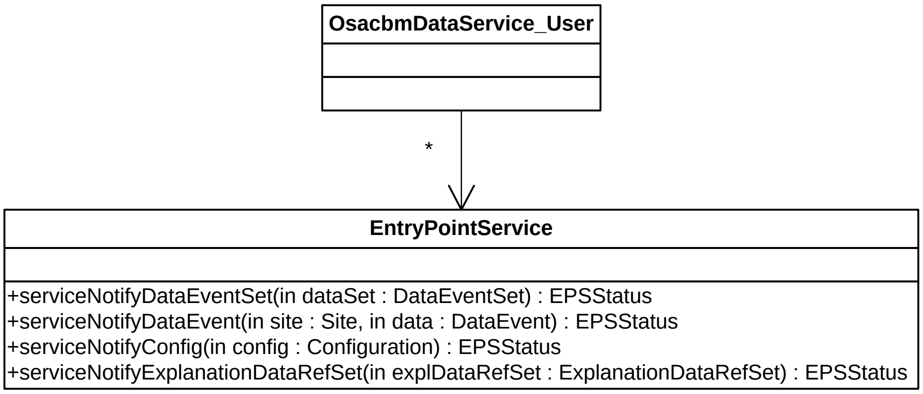
Data Service

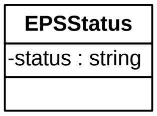
EntryPointService is a one way data input device.

An EntryPointService is a well known location service of a well known function.

Two possible uses would be:
1) data storage utility
2) maintenance advisory receiver service

The first four methods for DataEventSet, DataEvent, Config, and Explanation are the main ones expected to be used.


OsacbmDataService_User
*
EntryPointService
+serviceNotifyDataEventSet(in dataSet : DataEventSet) : EPSStatus
+serviceNotifyDataEvent(in site : Site, in data : DataEvent) : EPSStatus
+serviceNotifyConfig(in config : Configuration) : EPSStatus
+serviceNotifyExplanationDataRefSet(in explDataRefSet : ExplanationDataRefSet) : EPSStatus



EPSStatus is a return indicator of how an input message was received.

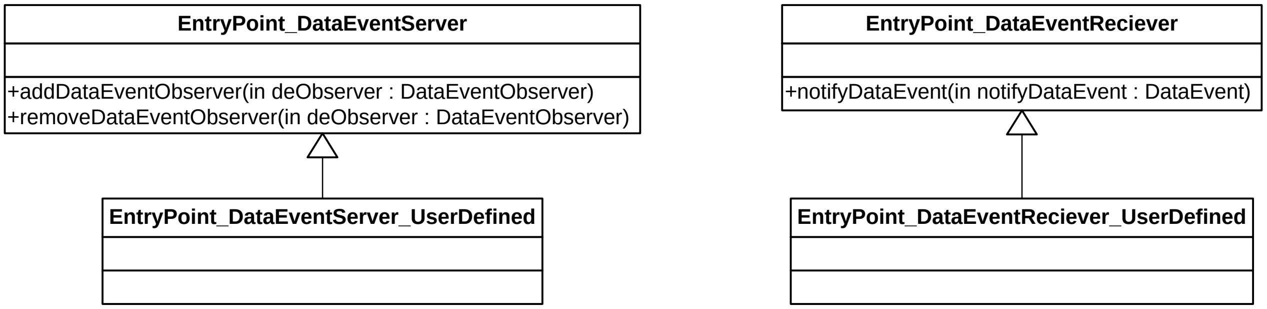
DataEvent Server

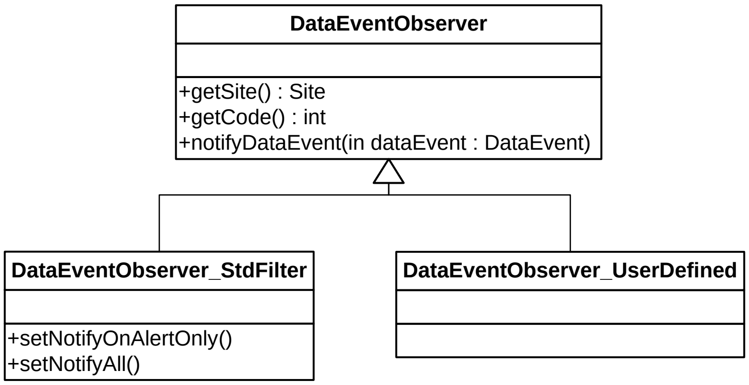
In many systems, signals are moved individually.
This page describes an interface mechanism for handling an individual DataEvent in a very simplified interface.

All UserDefined classes are abstract representations of some class that it written with any desired class name with specifically desired functionality for special handling of signals.

Serving module interface

Receiving module interface


EntryPoint_DataEventServer
+addDataEventObserver(in deObserver : DataEventObserver)
+removeDataEventObserver(in deObserver : DataEventObserver)
EntryPoint_DataEventServer_UserDefined
EntryPoint_DataEventReciever
+notifyDataEvent(in notifyDataEvent : DataEvent)
EntryPoint_DataEventReciever_UserDefined


DataEventObserver
+getSite() : Site
+getCode() : int
+notifyDataEvent(in dataEvent : DataEvent)
DataEventObserver_StdFilter
+setNotifyOnAlertOnly()
+setNotifyAll()
DataEventObserver_UserDefined

A module that provides DataEvents will inherit from the EntryPoint_DataEventServer.

A module that is to recieve DataEvents will inherit from EntryPoint_DataEventReciever.

The DataEventObserver will contain a reference to the EntryPoint_DataEventReciever and have the notification called by the EntryPoint_DataEventServer when a new DataEvent is ready.

Is is possible to put filtering of events into the DataEventObserver class via a child class. DataEventObserver_StdFilter is designed to all for filtering of DataEvents with alerts only.

Information Specification

The Information Specification describes in UML the information found in a CBM system. This specification was developed in conjunction with OSA-EAI CRIS.

There are six main categories of information:

Dynamic Data	(on platform)
Configuration Data	(not typical for on platform)
Explanation Data	(on platform optional)
Control Data	(simple user option)
App Data	(simple user option)
Error Data	(simple user option)

Each of these is individually addressable in the interface.

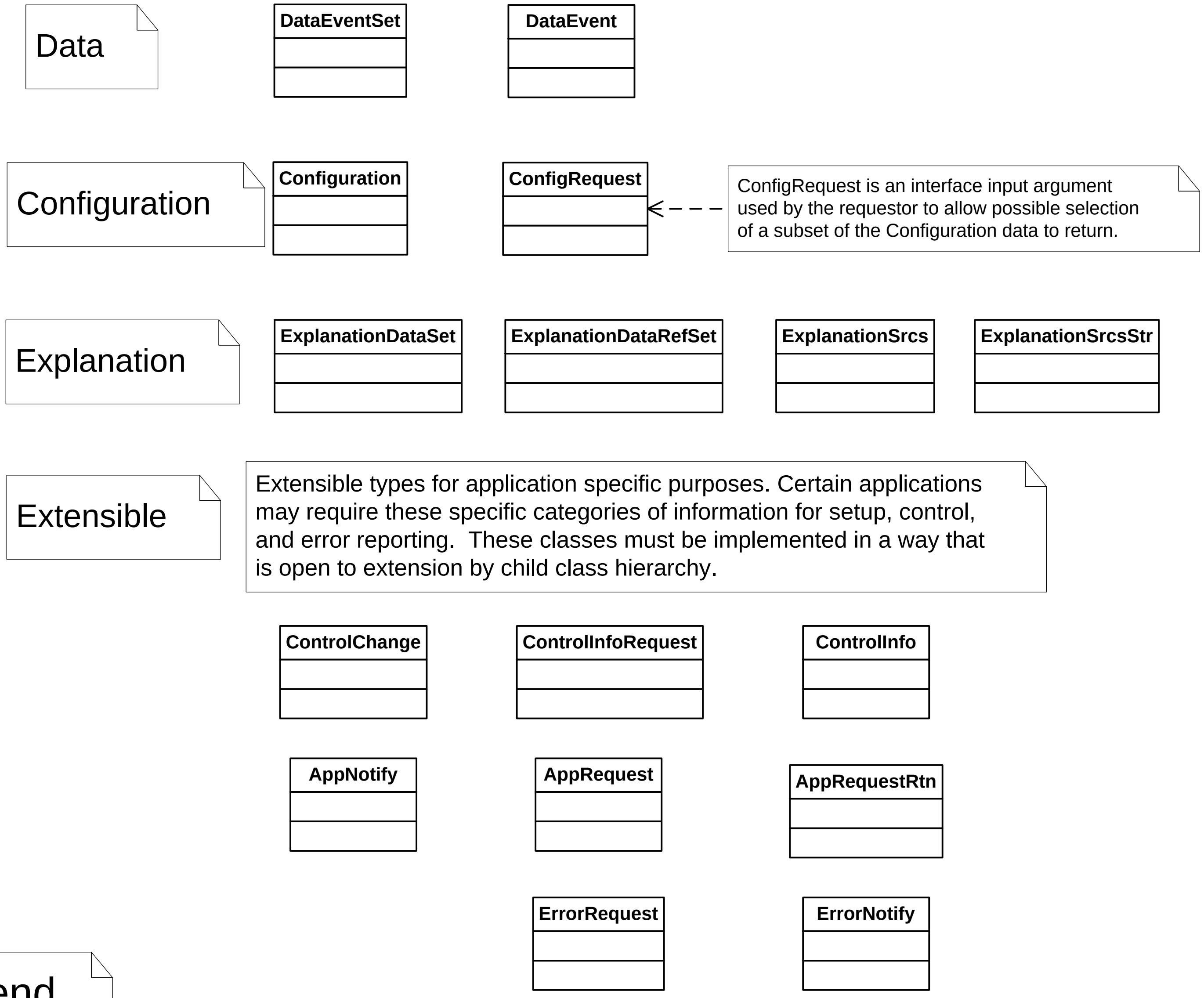
There is a request made for a DataEventSet and a DataEventSet is returned. A request is made for Configuration Data and Configuration Data is returned.

It is suggested that only Dynamic Data is required to be used in embedded systems such as a small platform IVHM system where configuration is static and engineering units are built in. The addition of configuration data especially forces users to put into these systems information not typically found there. That adds development time for something of very limited or no use within its present realm.

For such systems, MIMOSA servers may exist at servicing locations which contain the configuration information.

Information Specification - EntryPoint Classes

The EntryPoint interface provides direct access to the following classes.
The remaining thrust of this document describes their details in UML form.



ConfigRequest is an interface input argument used by the requestor to allow possible selection of a subset of the Configuration data to return.

Extensible types for application specific purposes. Certain applications may require these specific categories of information for setup, control, and error reporting. These classes must be implemented in a way that is open to extension by child class hierarchy.

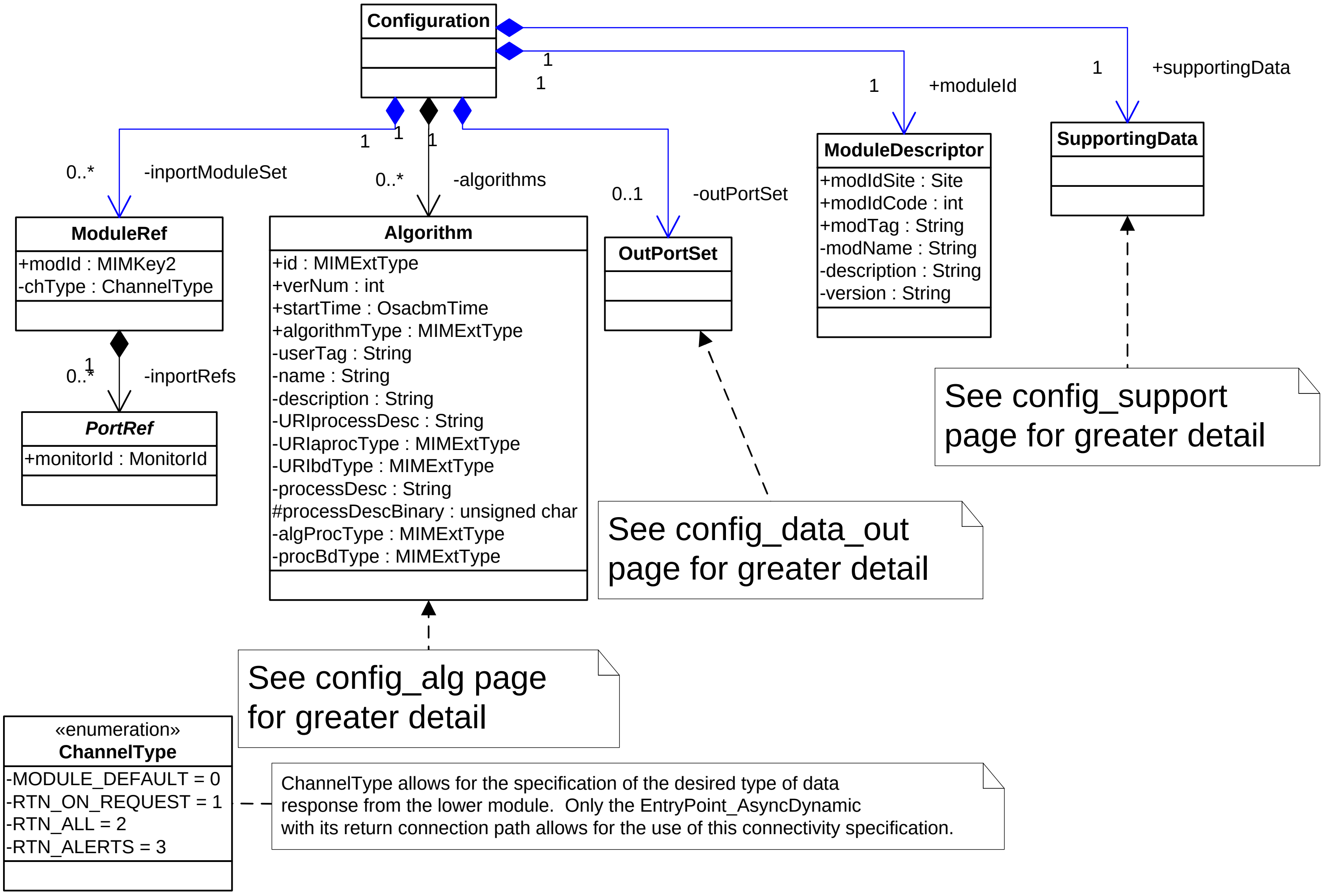
Legend

— organizational	— dynamic data	— general
— configuration	— explanation	— control
- (dash) is for an optional parameter	+ (plus) is for a non-optional parameter	# (pound) is for array parameter

The Legend explains some of the OSA-CBM specific nomenclature.
Note especially the '-', '+', and '#' used to indicate parameter optionality and count.

Configuration

Configuration gives information about an OSA-CBM module's input sources,
description of algorithms used for processing input data, a list of outputs,
and various output specifics such as engineering units, thresholds for alerts, etc.



See config_support page for greater detail

See config_data_out page for greater detail

See config_alg page for greater detail

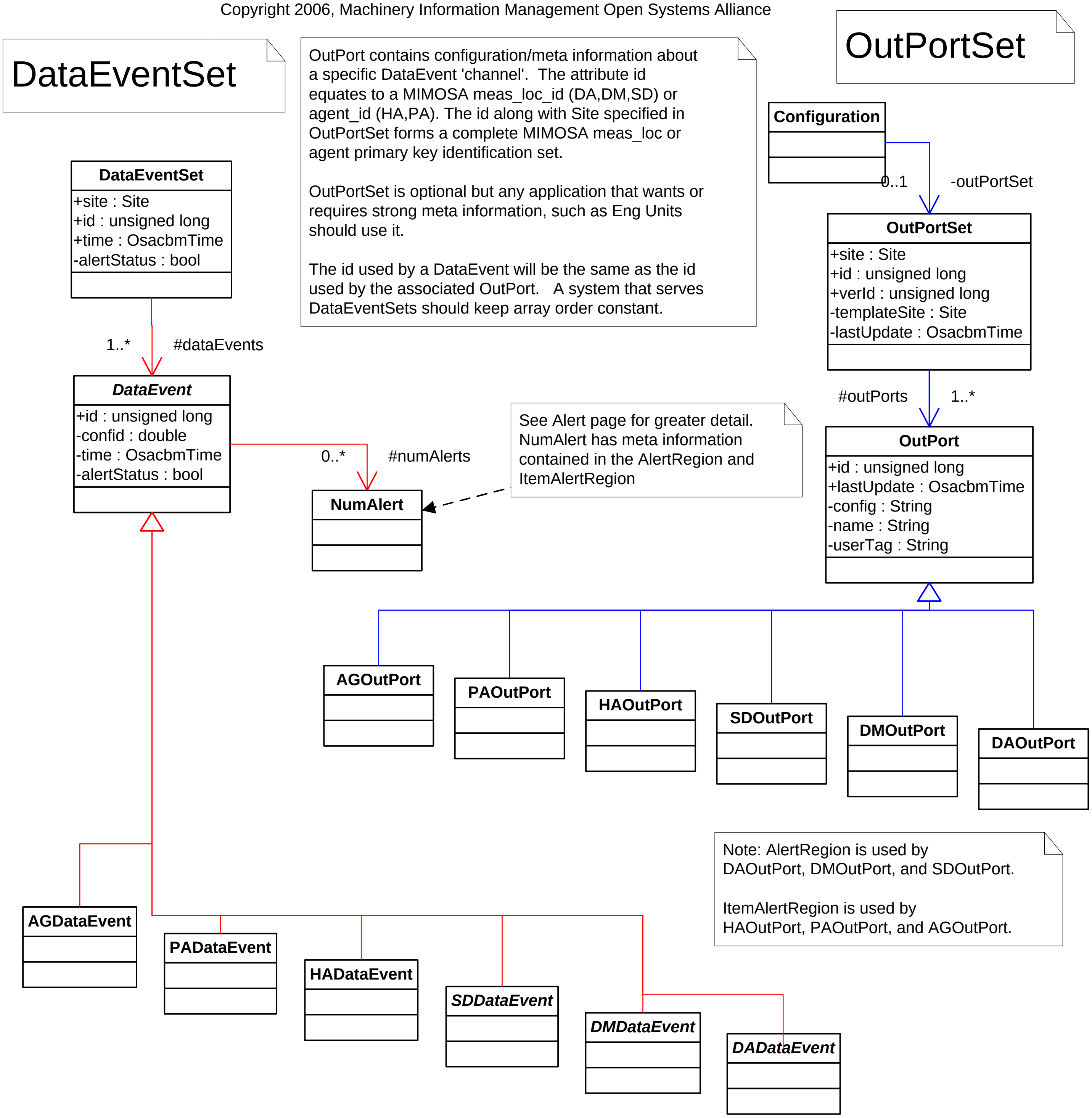
ChannelType allows for the specification of the desired type of data
response from the lower module. Only the EntryPoint_AsyncDynamic
with its return connection path allows for the use of this connectivity specification.

InportModuleSet gives information about where a module gets data from.

Algorithm describes the process used to generate a DataEvent.

OutPortSet lists each OutPort. An OutPort is a 'data channel' and the OutPort class gives specific configuration data for that data channel.

Supporting data gives additional information about MIMOSA MIMKey or primary key references which may be used elsewhere in this architecture.



OutPort contains configuration/meta information about a specific DataEvent 'channel'. The attribute id equates to a MIMOSA meas_loc_id (DA,DM,SD) or agent_id (HA,PA). The id along with Site specified in OutPortSet forms a complete MIMOSA meas_loc or agent primary key identification set.

OutPortSet is optional but any application that wants or requires strong meta information, such as Eng Units should use it.

The id used by a DataEvent will be the same as the id used by the associated OutPort. A system that serves DataEventSets should keep array order constant.

See Alert page for greater detail. NumAlert has meta information contained in the AlertRegion and ItemAlertRegion

Note: AlertRegion is used by DAOutPort, DMOutPort, and SDOutPort.

ItemAlertRegion is used by HAOutPort, PAOutPort, and AGOutPort.

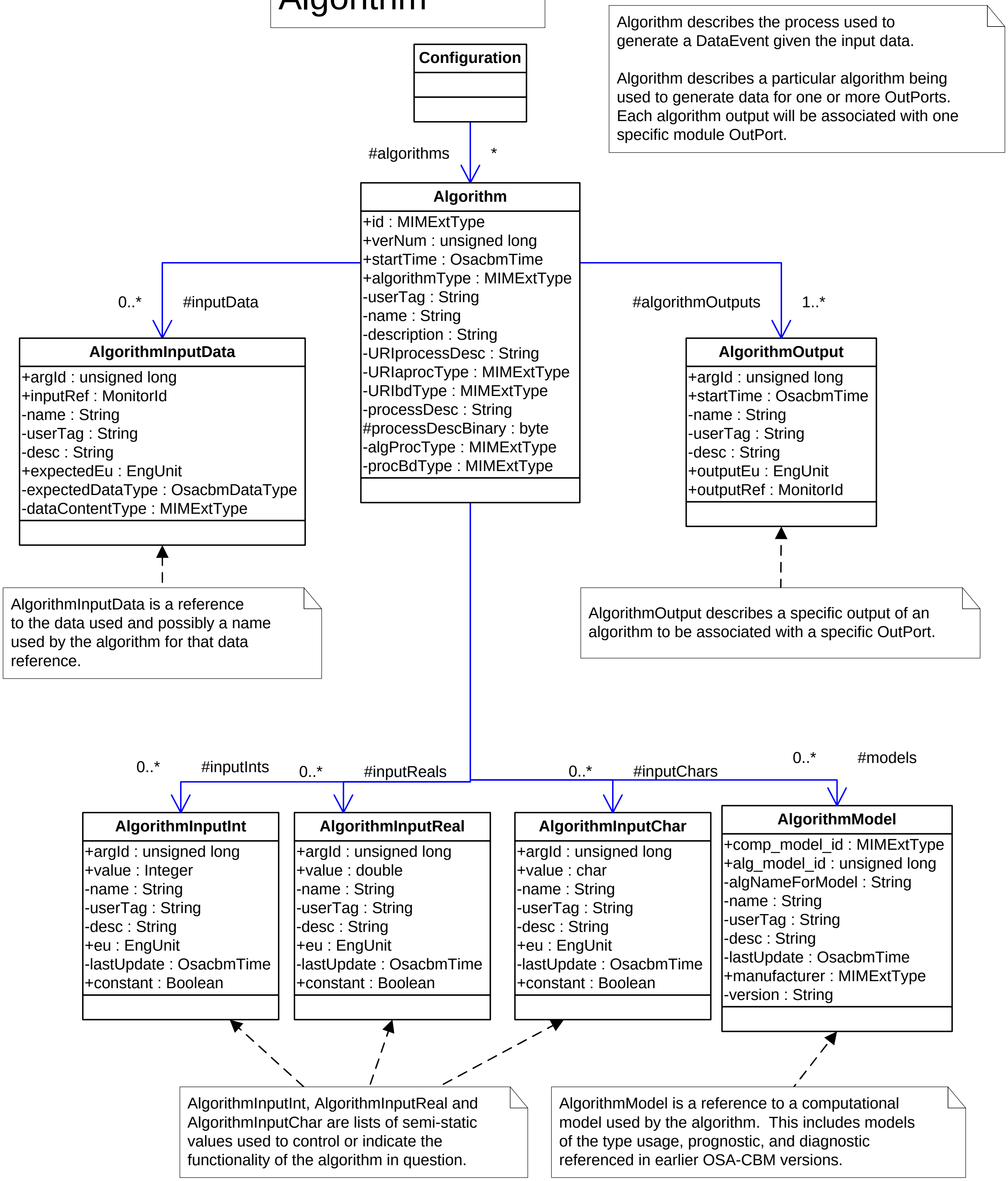
DataEvent contains the data for one OutPort data generation event.

The DataEvent child hierarchy below it is associated with a particular layer in the OSA-CBM architecture. Those classes have child classes below them describing a particular data type.

OutPort contains the configuration information specific to one output channel of a module. The OutPort child hierarchy associates to a particular layer in the OSA-CBM architecture.

Time on DataEvent is optional. If it is used it is meant to override the time from DataEventSet. It is also used with single DataEvent fetching.

Configuration Algorithm



Algorithm describes the process used to generate a DataEvent given the input data.

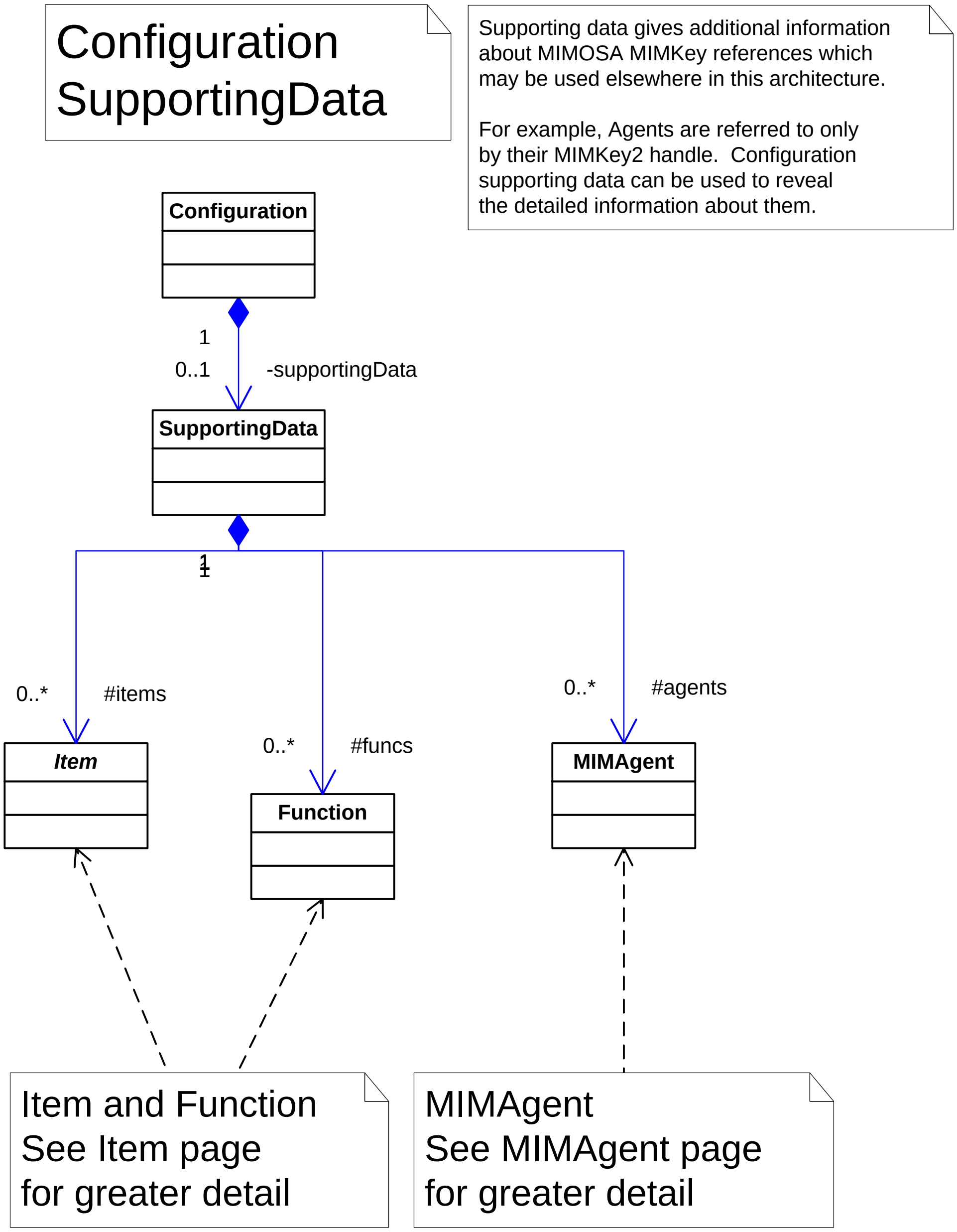
Algorithm describes a particular algorithm being used to generate data for one or more OutPorts. Each algorithm output will be associated with one specific module OutPort.

AlgorithmInputData is a reference to the data used and possibly a name used by the algorithm for that data reference.

AlgorithmOutput describes a specific output of an algorithm to be associated with a specific OutPort.

AlgorithmInputInt, AlgorithmInputReal and AlgorithmInputChar are lists of semi-static values used to control or indicate the functionality of the algorithm in question.

AlgorithmModel is a reference to a computational model used by the algorithm. This includes models of the type usage, prognostic, and diagnostic referenced in earlier OSA-CBM versions.



Configuration SupportingData

Supporting data gives additional information about MIMOSA MIMKey references which may be used elsewhere in this architecture.

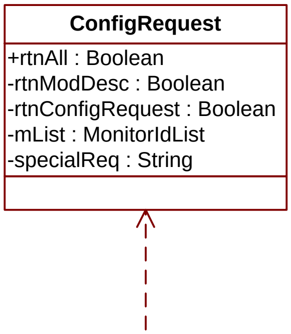
For example, Agents are referred to only by their MIMKey2 handle. Configuration supporting data can be used to reveal the detailed information about them.

Item and Function See Item page for greater detail

MIMAgent See MIMAgent page for greater detail

ConfigRequest

ConfigurationRequest is used in the entry point interface method to select possible subsets of the Configuration data to reduce the size of the returned request.



An OSA-CBM module, especially at the HA or PA level, may have a large list of supported components, like Item (i.e. Assets and Segments), Outports, algorithms, agents, etc.

The ConfigRequest is for future capability to be able to request a subset of that data.

Alternatively, this could just be a database interaction and not part of the OSA-CBM specification.

Parmeters:

rtnAll - overriding parameter to state all configuration information is desired.

rtnModDesc - indicates whether the module description is desired as part of the return.

rtnConfigRequest - indicates if the ConfigRequest is desired as part of the return.
This is used as confirmation that the request was properly received.

monitorIdList - list of data channels or agent/monitored components in the configuration subset.

specialReq - is for future extensible detailed subset request.

Explanation

Explanation is the Data or a Reference to the Data used by a module to produce an output.
The OutPort algorithm is a description of that process.

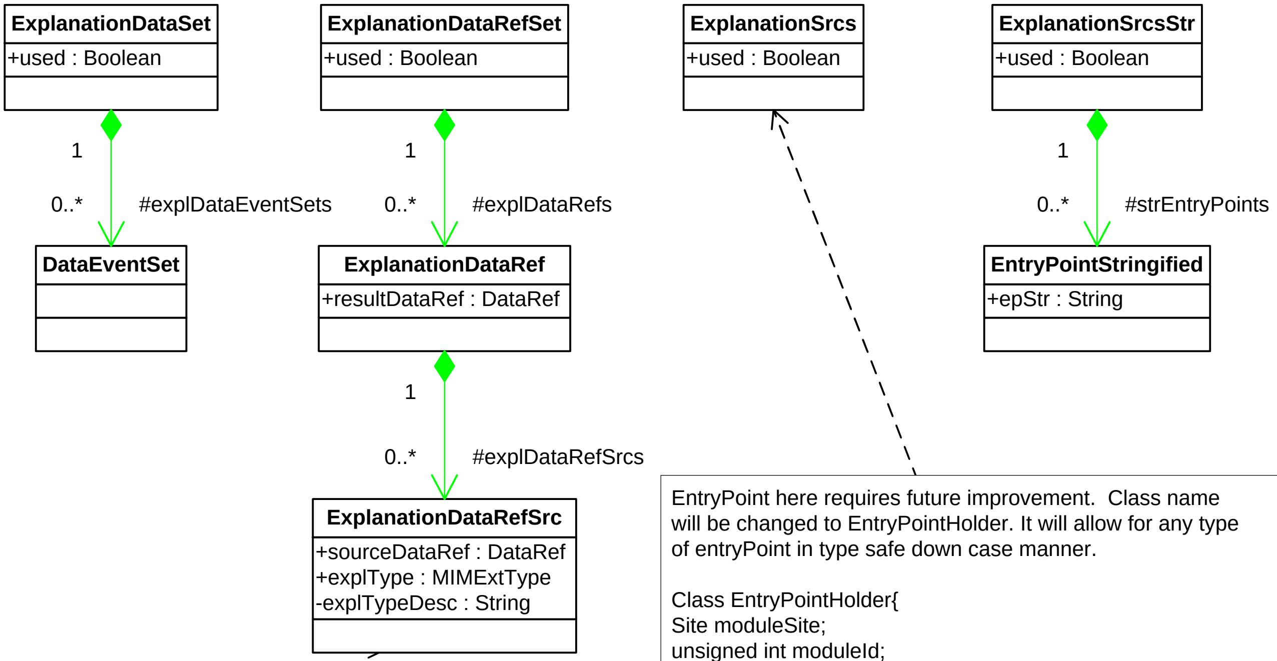
Explanation consists of four possible necessary forms depending upon the application.

The first is simply the data used for a calculation.

The second is more of a handle / timestamp type of reference to the data used. This is used when the data comes from a well known location or it is known to be stored somewhere. The main example is using data stored in a database.

The final two forms are two different ways of giving direct access to the modules supplying the data. One is a set of direct pointers to modules. The other form is a "stringified" form of a pointer that will allow a user to construct a pointer to the module.

Explanation Forms



EntryPoint here requires future improvement. Class name will be changed to EntryPointHolder. It will allow for any type of entryPoint in type safe down case manner.

```
Class EntryPointHolder{
Site moduleSite;
unsigned int moduleId;
EntryPointType epType; // MIMNonExt
};

Class EntryPointHolder_Type1:public EntryPointHolder{
EntryPointHolder_Type1& ep;
};
```

There will be many types of standard MIMOSA OSA-CBM explanation types.

Note on "used" boolean

Note, if a form is not used, then just set the boolean 'used' to false and return an empty set.

Extensible Components

These are called the extensible components because they are very application specific.

A specific use may be designed with UML and have a XML mapping.
A specific language implementation may use the UML class form.
Any serial communication needs, like HTTP, DCOM, or CORBA over Ethernet, may then convert the UML into the XML form and use the standard interface without the need to develop another communication interface.

Note once again, these are application specific!
It is fine if a small embedded system does not want to use them or wants to use them in a very narrowly defined way.

Control Specific

Control is the concept of being able to change module parameters on the fly.

One major use would be to be able to change a threshold alert monitor's threshold settings on the fly to adjust to present operating conditions.

Application Specific

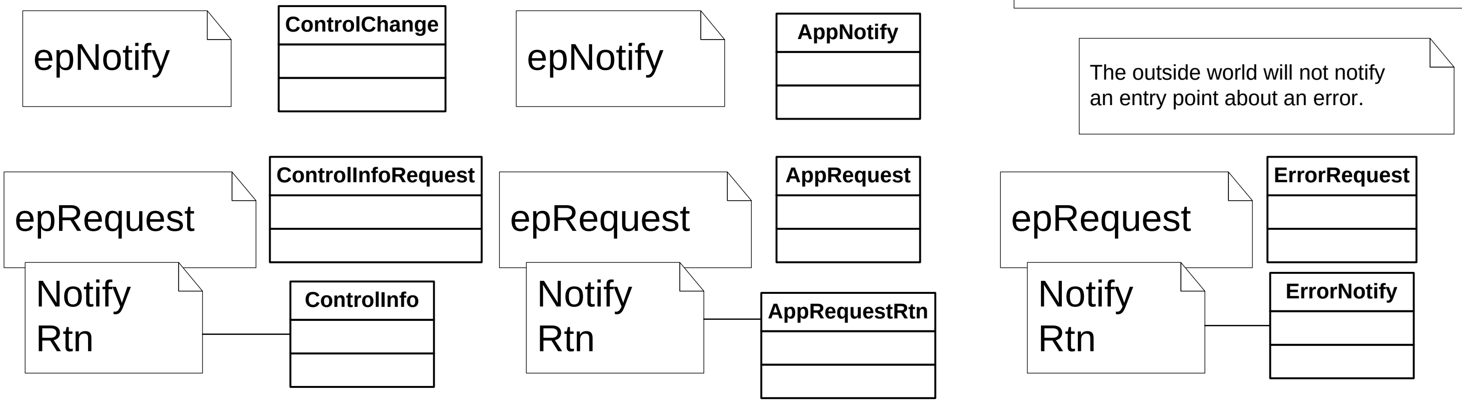
Application specific is the concept of being able to interact with a module in an application specific way.

One possible use might be to request extra non-standard information about a module.

Error Specific

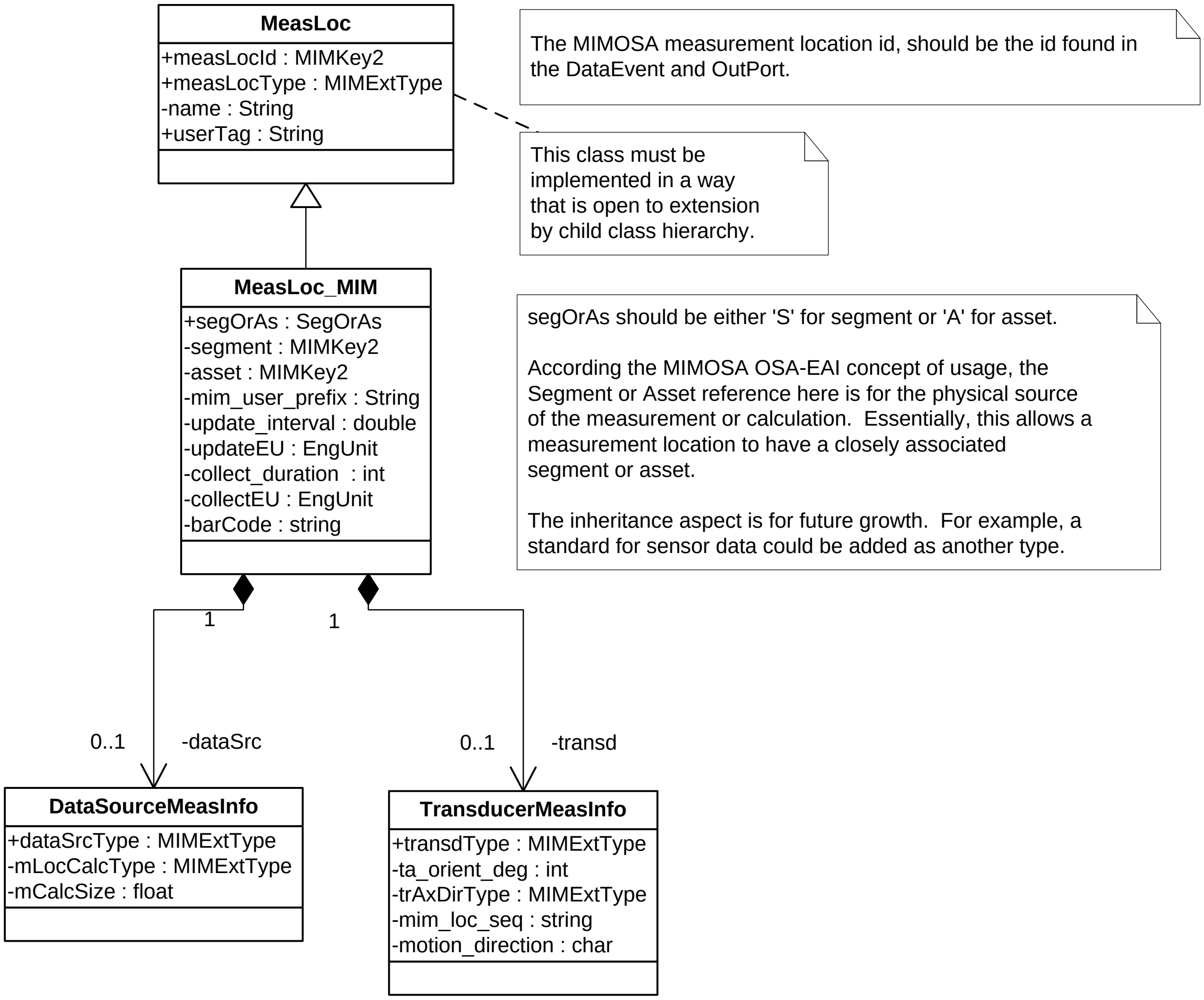
Error specific is the concept of indicating an error condition. Errors are application specific. However as the standard progresses, specific activities may start to standardize errors.
For example, invalid web requests using XML over HTTP will have a standard return response.

Note: Connected state configuration will allow for unsolicited error notification.



The outside world will not notify an entry point about an error.

Measurement Location (DA, DM, SD)



The MIMOSA measurement location id, should be the id found in the DataEvent and OutPort.

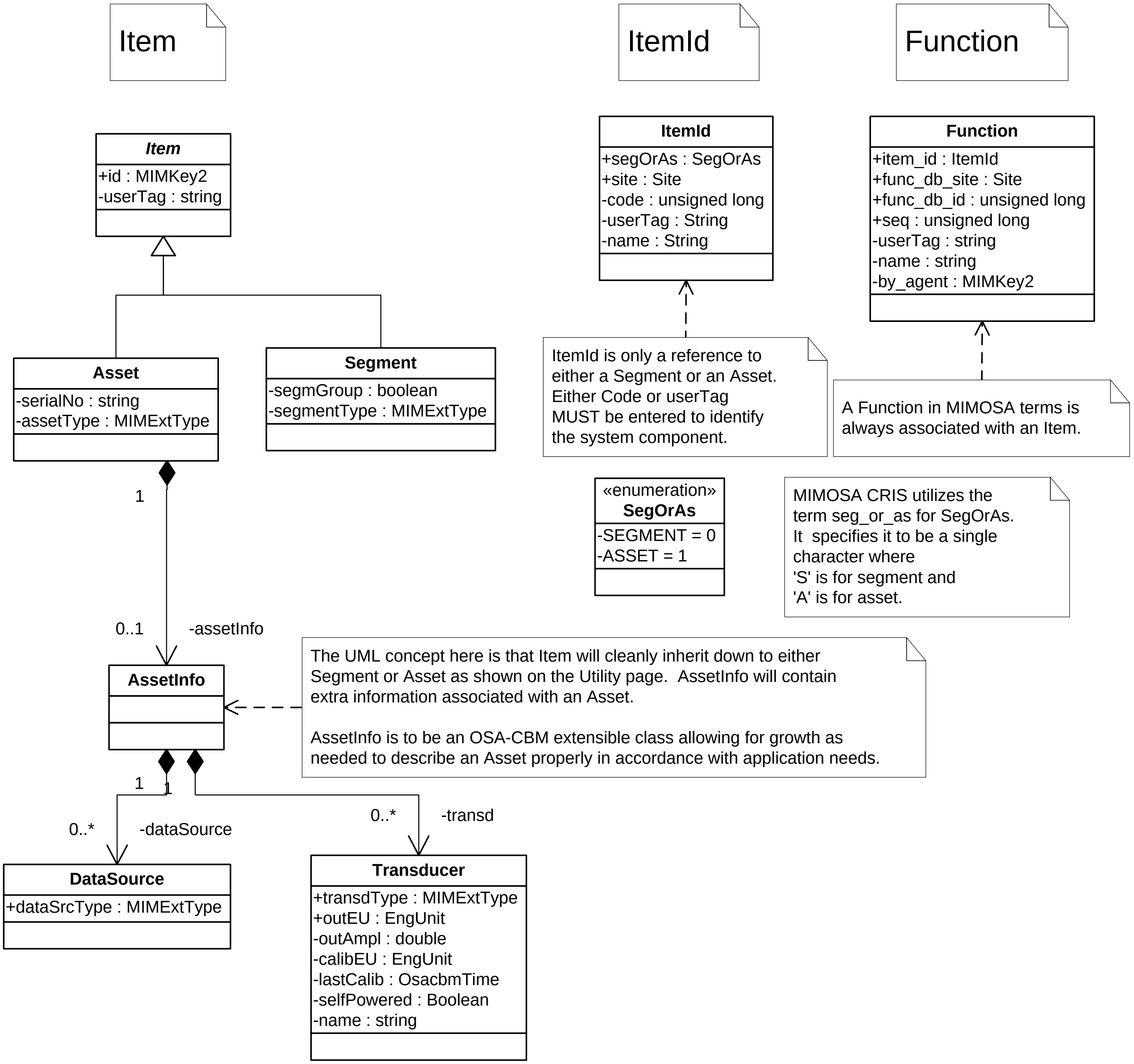
This class must be implemented in a way that is open to extension by child class hierarchy.

segOrAs should be either 'S' for segment or 'A' for asset.

According the MIMOSA OSA-EAI concept of usage, the Segment or Asset reference here is for the physical source of the measurement or calculation. Essentially, this allows a measurement location to have a closely associated segment or asset.

The inheritance aspect is for future growth. For example, a standard for sensor data could be added as another type.

The Item page details the Item and Function classes
that may be used or referenced elsewhere in this document.



ItemId is only a reference to either a Segment or an Asset. Either Code or userTag MUST be entered to identify the system component.

A Function in MIMOSA terms is always associated with an Item.

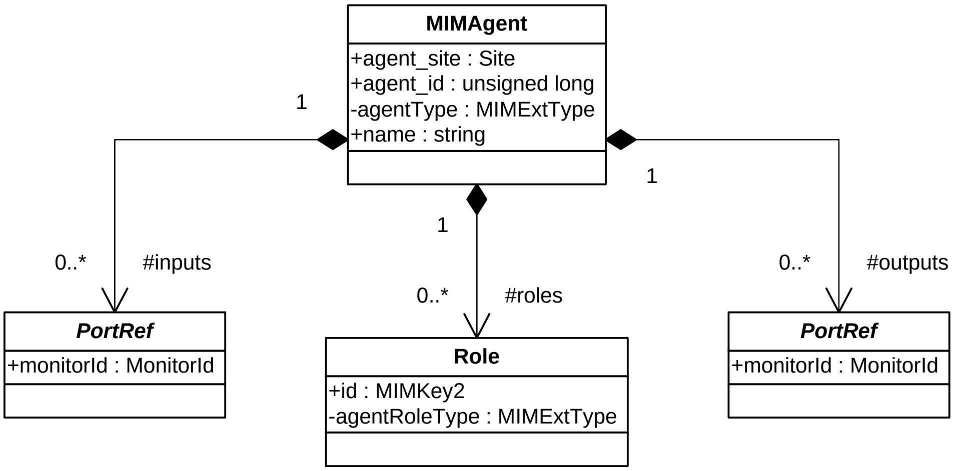
MIMOSA CRIS utilizes the term seg_or_as for SegOrAs. It specifies it to be a single character where 'S' is for segment and 'A' is for asset.

The UML concept here is that Item will cleanly inherit down to either Segment or Asset as shown on the Utility page. AssetInfo will contain extra information associated with an Asset.

AssetInfo is to be an OSA-CBM extensible class allowing for growth as needed to describe an Asset properly in accordance with application needs.

This page details the MIMAgent class
that is referenced elsewhere in this document.

MIMAgent and Roles

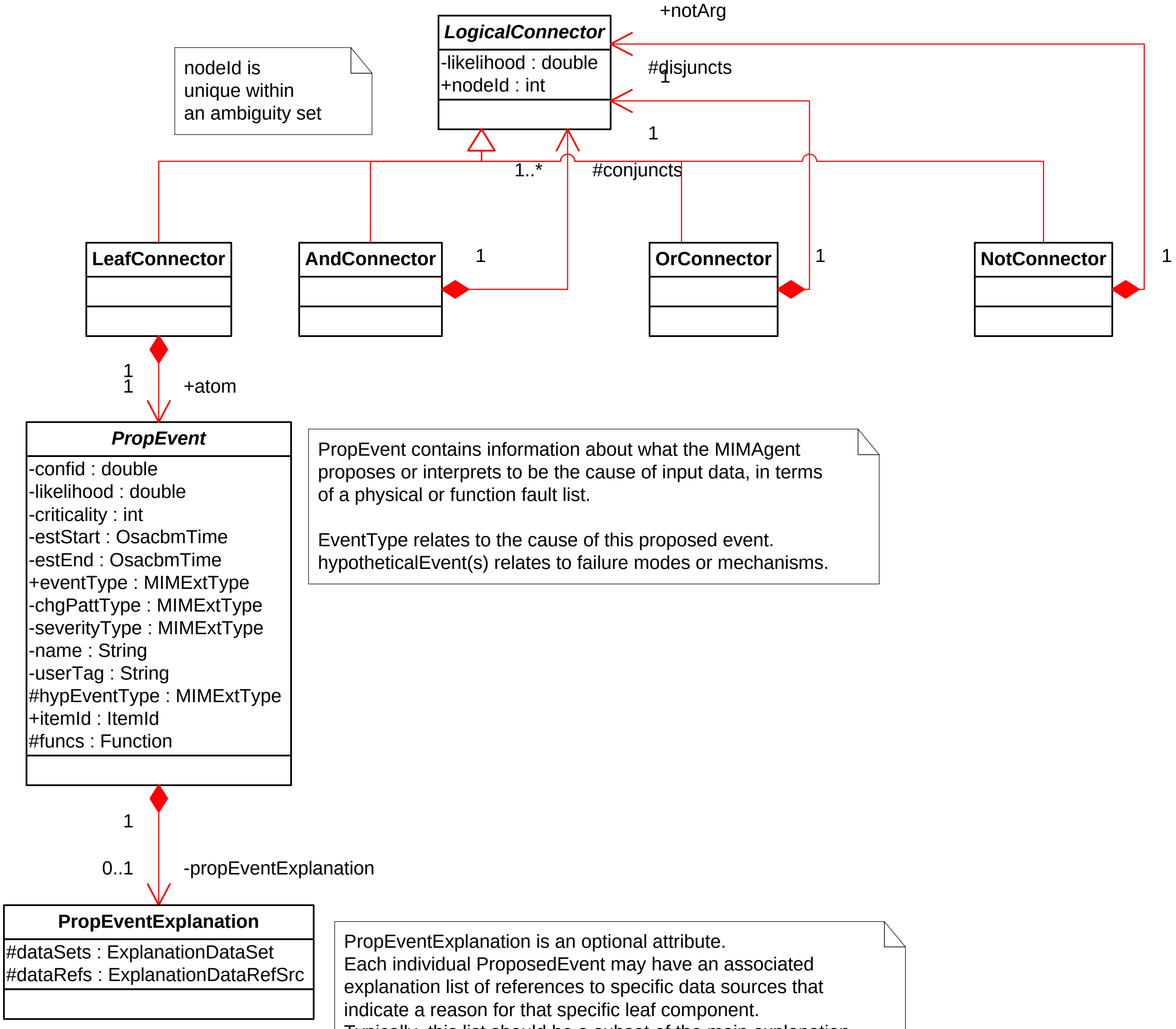


Proposed Event for Failure Descriptions (HA,PA)

The LogicalConnector provides for any style of ambiguity group by using combinations of the AndConnector, OrConnector, and NotConnector classes.

The LeafConnector class gives information about the proposed event fault.

A single LeafConnector without using the And, Or, and Not Connectors is the simplest form used to describe a single determined fault.

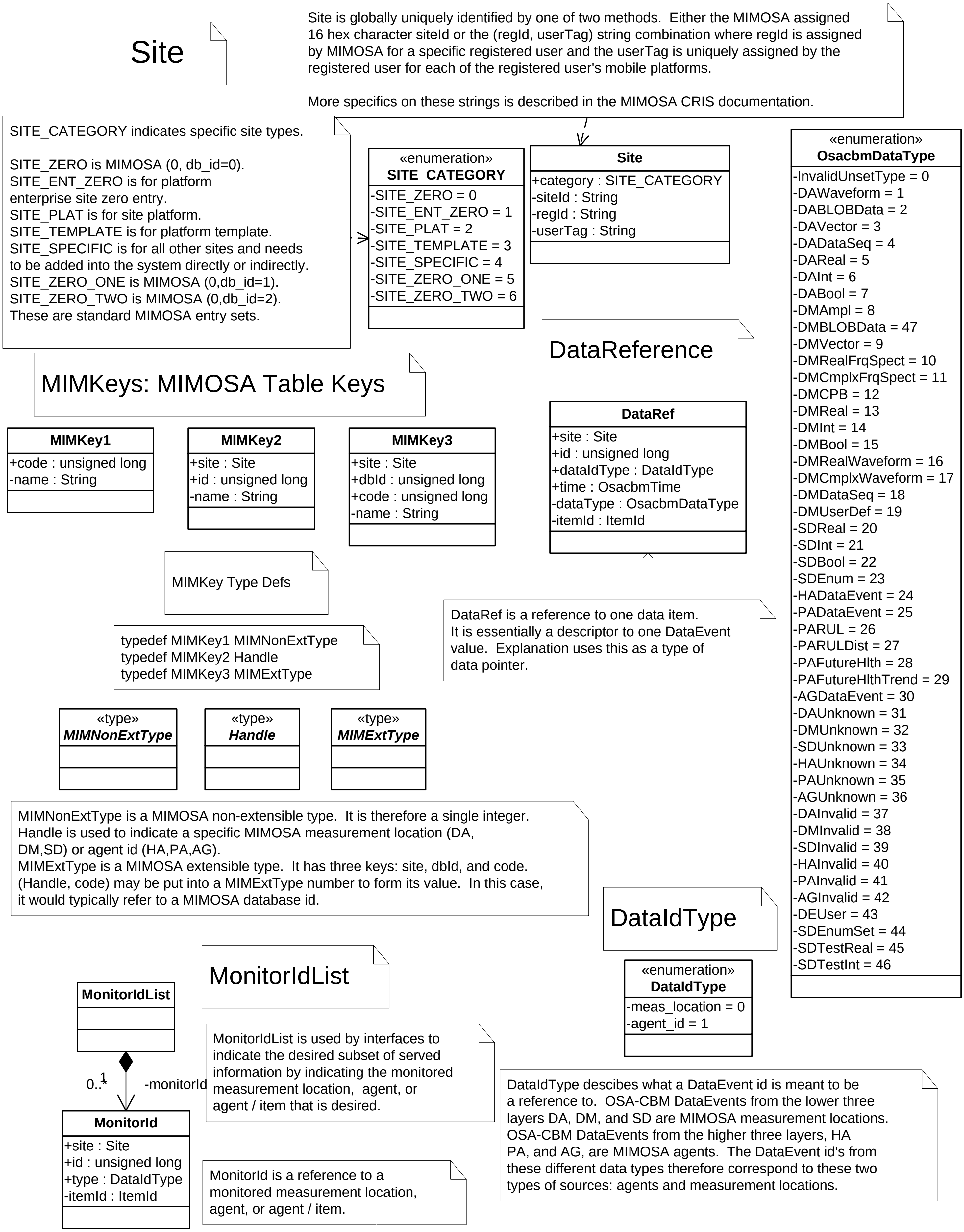


nodeId is
unique within
an ambiguity set

PropEvent contains information about what the MIMAgent proposes or interprets to be the cause of input data, in terms of a physical or function fault list.

EventType relates to the cause of this proposed event.
hypotheticalEvent(s) relates to failure modes or mechanisms.

PropEventExplanation is an optional attribute.
Each individual ProposedEvent may have an associated explanation list of references to specific data sources that indicate a reason for that specific leaf component.
Typically, this list should be a subset of the main explanation data set for the overall ambiguity group description.



Site

Site is globally uniquely identified by one of two methods. Either the MIMOSA assigned 16 hex character siteId or the (regId, userTag) string combination where regId is assigned by MIMOSA for a specific registered user and the userTag is uniquely assigned by the registered user for each of the registered user's mobile platforms.

More specifics on these strings is described in the MIMOSA CRIS documentation.

SITE_CATEGORY indicates specific site types.

SITE_ZERO is MIMOSA (0, db_id=0).
SITE_ENT_ZERO is for platform enterprise site zero entry.
SITE_PLAT is for site platform.
SITE_TEMPLATE is for platform template.
SITE_SPECIFIC is for all other sites and needs to be added into the system directly or indirectly.
SITE_ZERO_ONE is MIMOSA (0,db_id=1).
SITE_ZERO_TWO is MIMOSA (0,db_id=2).
These are standard MIMOSA entry sets.

«enumeration»
SITE_CATEGORY
-SITE_ZERO = 0
-SITE_ENT_ZERO = 1
-SITE_PLAT = 2
-SITE_TEMPLATE = 3
-SITE_SPECIFIC = 4
-SITE_ZERO_ONE = 5
-SITE_ZERO_TWO = 6

Site
+category : SITE_CATEGORY
-siteId : String
-regId : String
-userTag : String

«enumeration»
OsacbmDataType
-InvalidUnsetType = 0
-DAWaveform = 1
-DABLOBData = 2
-DAVector = 3
-DADataSeq = 4
-DAReal = 5
-DAInt = 6
-DABool = 7
-DMAmpl = 8
-DMBLOBData = 47
-DMVector = 9
-DMRealFrqSpect = 10
-DMCmplxFrqSpect = 11
-DMCPB = 12
-DMReal = 13
-DMInt = 14
-DMBool = 15
-DMRealWaveform = 16
-DMCmplxWaveform = 17
-DMDataSeq = 18
-DMUserDef = 19
-SDReal = 20
-SDInt = 21
-SDBool = 22
-SDEnum = 23
-HADataEvent = 24
-PADataEvent = 25
-PARUL = 26
-PARULDist = 27
-PAFutureHlth = 28
-PAFutureHlthTrend = 29
-AGDataEvent = 30
-DAUnknown = 31
-DMUnknown = 32
-SDUnknown = 33
-HAUnknown = 34
-PAUnknown = 35
-AGUnknown = 36
-DAInvalid = 37
-DMInvalid = 38
-SDInvalid = 39
-HAInvalid = 40
-PAInvalid = 41
-AGInvalid = 42
-DEUser = 43
-SDEnumSet = 44
-SDTestReal = 45
-SDTestInt = 46

MIMKeys: MIMOSA Table Keys

MIMKey1
+code : unsigned long
-name : String

MIMKey2
+site : Site
+id : unsigned long
-name : String

MIMKey3
+site : Site
+dbId : unsigned long
+code : unsigned long
-name : String

MIMKey Type Defs

typedef MIMKey1 MIMNonExtType
typedef MIMKey2 Handle
typedef MIMKey3 MIMExtType

«type»
MIMNonExtType

«type»
Handle

«type»
MIMExtType

MIMNonExtType is a MIMOSA non-extensible type. It is therefore a single integer.
Handle is used to indicate a specific MIMOSA measurement location (DA, DM,SD) or agent id (HA,PA,AG).
MIMExtType is a MIMOSA extensible type. It has three keys: site, dbId, and code.
(Handle, code) may be put into a MIMExtType number to form its value. In this case, it would typically refer to a MIMOSA database id.

DataReference

DataRef
+site : Site
+id : unsigned long
+dataIdType : DataIdType
+time : OsacbmTime
-dataType : OsacbmDataType
-itemId : ItemId

DataRef is a reference to one data item. It is essentially a descriptor to one DataEvent value. Explanation uses this as a type of data pointer.

DataIdType

«enumeration»
DataIdType
-meas_location = 0
-agent_id = 1

DataIdType descibes what a DataEvent id is meant to be a reference to. OSA-CBM DataEvents from the lower three layers DA, DM, and SD are MIMOSA measurement locations. OSA-CBM DataEvents from the higher three layers, HA PA, and AG, are MIMOSA agents. The DataEvent id's from these different data types therefore correspond to these two types of sources: agents and measurement locations.

MonitorIdList

MonitorIdList

MonitorId
+site : Site
+id : unsigned long
+type : DataIdType
-itemId : ItemId

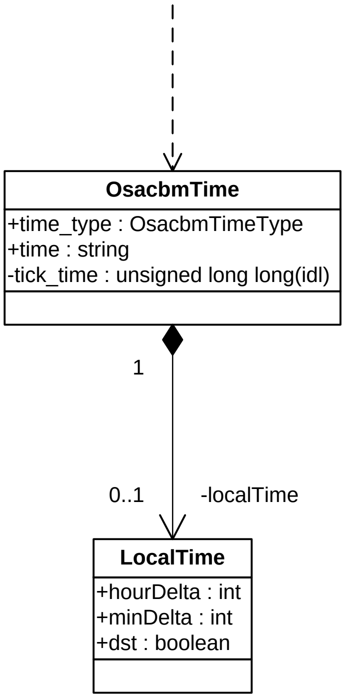
MonitorIdList is used by interfaces to indicate the desired subset of served information by indicating the monitored measurement location, agent, or agent / item that is desired.

MonitorId is a reference to a monitored measurement location, agent, or agent / item.

The Utility page details some classes that may be used or referenced elsewhere in this document.

Time

OSA-CBM uses the name OsacbmTime to eliminate name-clashing with other technologies.



Time has been expanded to have a few different internal content form types. This is to allow the simplest most direct method of handling time to be incorporated in an embedded program.

The MIMOSA type should be transmitted as a string conforming to ISO 8601. See description at side.

Tick time is presently defined for microseconds, with the data type long long. It is the number of ticks since start up of the program.

Posix is a Unix type time, which is also fetched in terms of the data type long long.

Methods to access specific time portions is highly desirable for any implementation.
Any language implementation should have methods:
uint getYear()
uint getMonth()
...
ushort getHour()
...
ushort getMicrosec()
ushort getNanosec()

methods should interpret Tick or Posix accordingly.

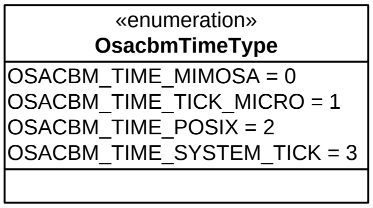
All rtn types are integer EXCEPT getTick which is an unsigned long long 64 bit int.

Date/time in ISO 8601 variable length character form:
YYYY-MM-DDThh:mm:ss.fffffffff
example 2006-05-31T14:30:33.123

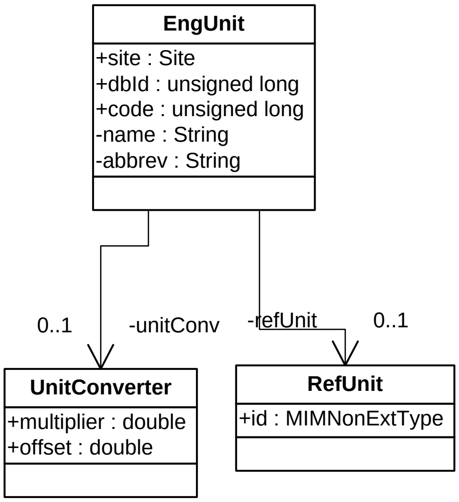
where:
YYYY = four-digit year
MM = two-digit month (01=January, etc.)
DD = two-digit day of month (01 through 31)
hh = two digits of hour (00 through 23) (am/pm NOT allowed)
T = literal "T" character
mm = two digits of minute (00 through 59)
ss = two digits of second (00 through 59)
fffffffff = represents a decimal fraction of a second to the billionth of a second

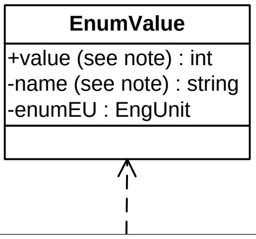
Year, month, and day must be specified. Additional timestamp content should be provided, if known. Zeros will be assumed for the omitted values. Negative DATETIME is not supported. All suffixes after the 29th character provided in the ISO 8601 specification, such as "Z" (representing Coordinated Universal Time (UTC), are not necessary since the CRIS specification explicitly manages local offset hours and minutes as distinct columns associated with the UTC (referred to in the CRIS specification prefixed with "GMT") column.

Note that the actual difference between the new DATETIME(10:29) data type and the CRIS V2.1 fixed-length STRING(29) form is the separator between date and time information is now a literal T instead of a blank space, the separator for the billionths of seconds is now a dot (.) instead of a dash (-), and trailing items after the year-month-day fields may be omitted.



EngUnit and Enum Type

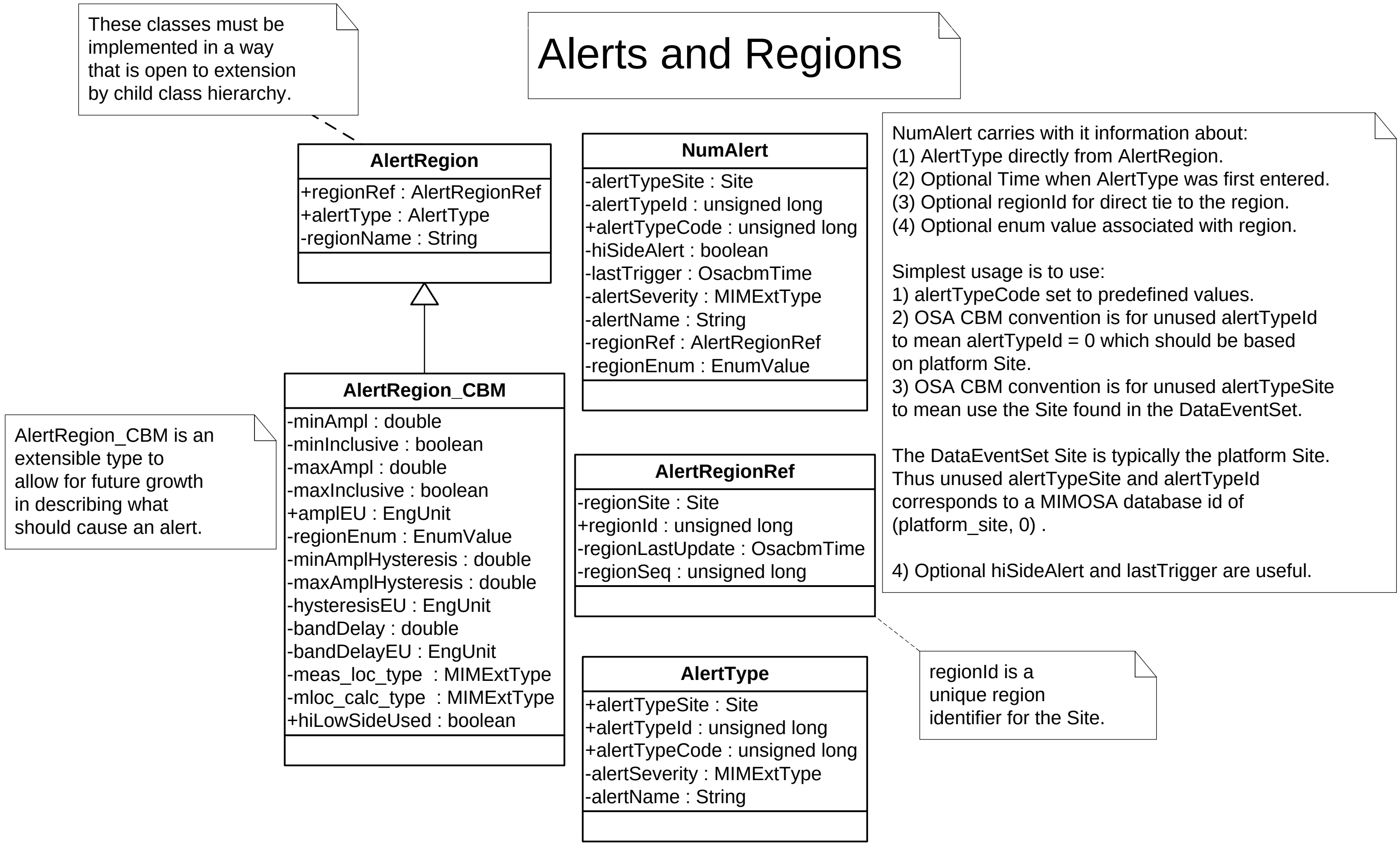




Enum value is uniquely identified by Eng Unit plus value.
Name may be transmitted optionally.

OutPort should have a corresponding EngUnit for transmitted values. Therefore, in a DataEvent transmission the DataEvent id can link to EngUnit from the OutPort and only a value may be needed in the actual DataEvent if shortness of expression is desired.

Alerts and Regions



These classes must be implemented in a way that is open to extension by child class hierarchy.

AlertRegion
+regionRef : AlertRegionRef
+alertType : AlertType
-regionName : String

AlertRegion_CBM
-minAmpl : double
-minInclusive : boolean
-maxAmpl : double
-maxInclusive : boolean
+amplEU : EngUnit
-regionEnum : EnumValue
-minAmplHysteresis : double
-maxAmplHysteresis : double
-hysteresisEU : EngUnit
-bandDelay : double
-bandDelayEU : EngUnit
-meas_loc_type : MIMExtType
-mloc_calc_type : MIMExtType
+hiLowSideUsed : boolean

AlertRegion_CBM is an extensible type to allow for future growth in describing what should cause an alert.

NumAlert
-alertTypeSite : Site
-alertTypeId : unsigned long
+alertTypeCode : unsigned long
-hiSideAlert : boolean
-lastTrigger : OsacbmTime
-alertSeverity : MIMExtType
-alertName : String
-regionRef : AlertRegionRef
-regionEnum : EnumValue

NumAlert carries with it information about:
(1) AlertType directly from AlertRegion.
(2) Optional Time when AlertType was first entered.
(3) Optional regionId for direct tie to the region.
(4) Optional enum value associated with region.

Simplest usage is to use:
1) alertTypeCode set to predefined values.
2) OSA CBM convention is for unused alertTypeId to mean alertTypeId = 0 which should be based on platform Site.
3) OSA CBM convention is for unused alertTypeSite to mean use the Site found in the DataEventSet.

The DataEventSet Site is typically the platform Site. Thus unused alertTypeSite and alertTypeId corresponds to a MIMOSA database id of (platform_site, 0) .

4) Optional hiSideAlert and lastTrigger are useful.

AlertRegionRef
-regionSite : Site
+regionId : unsigned long
-regionLastUpdate : OsacbmTime
-regionSeq : unsigned long

regionId is a unique region identifier for the Site.

AlertType
+alertTypeSite : Site
+alertTypeId : unsigned long
+alertTypeCode : unsigned long
-alertSeverity : MIMExtType
-alertName : String

Usage Concept

A set of AlertRegions will be associated with a specific OutPort. The value associated with the AlertRegion set is the one contained in the DataEvent being output.

When the value contained in a DataEvent activates an AlertRegion the DataEvent will contain a NumAlert associated to the AlertRegion along with the time of occurance stored in the lastTrigger. In a condition-based monitoring system, the following DataEvents from that OutPort will have new values and new times. However, the NumAlert will remain the same while in that region. This includes the lastTrigger time which may be used as an indicator of how long a particular region has been in effect.

When a Region is first entered it gives the specific DataEvent an "Alert Status". For CBM modules supporting "Alert Status" functionality, the output can be suppressed to output only when an alert trigger occurs. This mechanism requires one of the connected type interfaces.

(RegionId, OutPort handle) can be the identifier used by a higher module to control threshold levels via a user defined ControlVector. Future versions of the standard will begin to create a standard UML/XML form for this control.

Hysteresis BandDelay

Hyterisis and BandDelay are used to reduce threshold nuisance crossings.
Region is activated when:
1) Threshold amplitude is crossed when no BandDelay and no Hysteresis.
2) Threshold amplitude is crossed for more than BandDelay time.
3) Threshold amplitude is crossed by more than Hysteresis value.
3) Threshold amplitude is crossed by more than Hysteresis value for BandDelay.

Hi-Low

hiLowSideUsed default is false. MIMOSA OSA-EAI does not presently have it and therefore it would be false for OSA-EAI apps.

The simplest method of use is to have a single set of alertRegions for an output and a direct set of alert types for those regions. The system should be set up so that AlertTypes and AlertRegions are all unique to the system and within the system. Then only a single int is needed to identify a code, the RegionId and the AlertTypeCode respectively.
The Alert classes are based on the MIMOSA OSA EAI CRIS. Substitute the term Alert for Alarm. The OSA-CBM version has a few extra parameters like those for hysterisis and hiSideAlert indicator.
The main principles for optional arguments are:
For those terms that are primary keys in CRIS: if they are not used, then they should be elsewhere in the information schema, i.e. if Site is not specified use the site found in DataEventSet. For those terms that are not primary keys, like name, they may simply be expected to be found in a database. In short, assume that they are not needed for an operational monitoring system and would only make the system less efficient.

Data class for user definable types

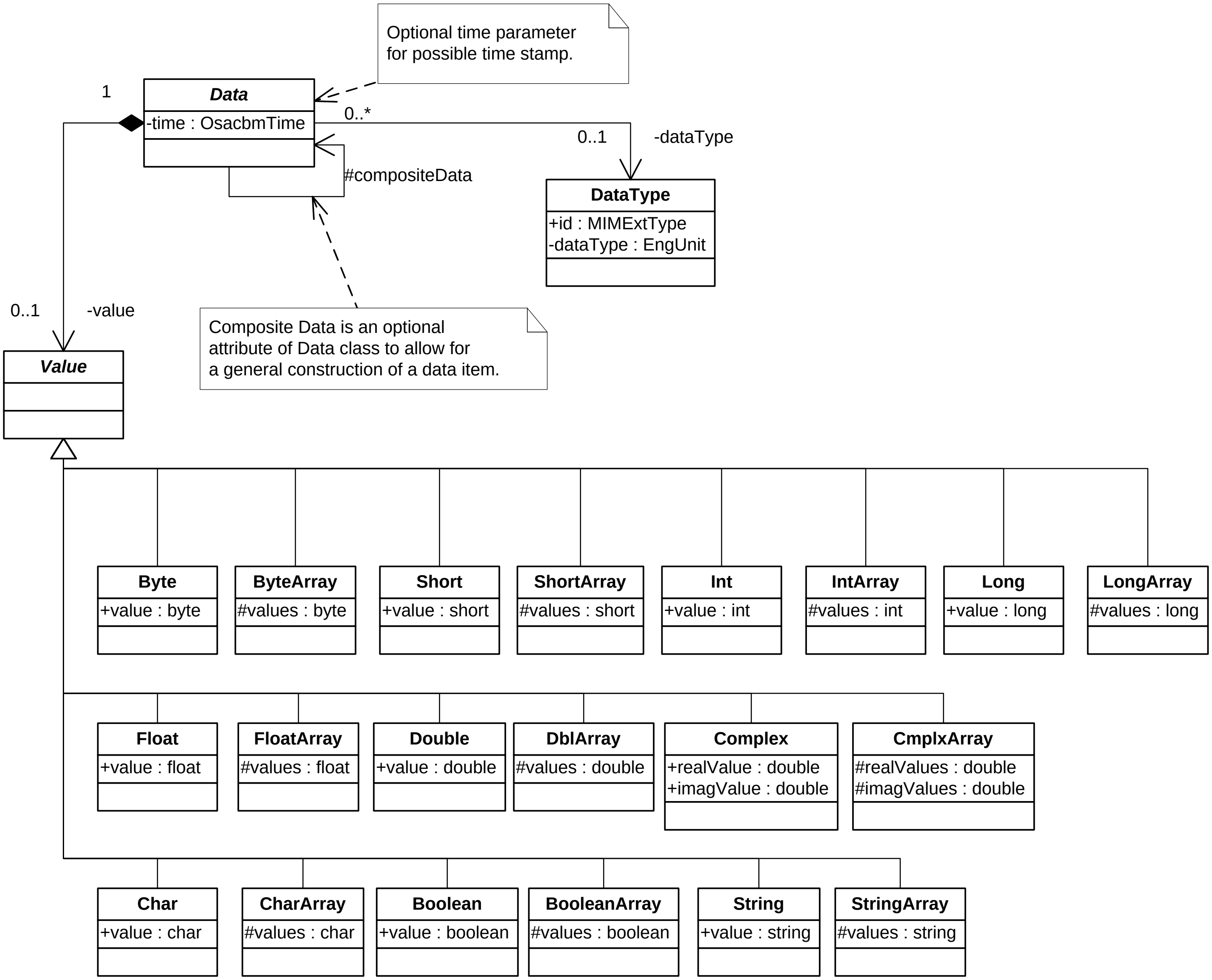
Data is mainly for user-definable types not in the OSA-CBM specification. This class set should be used mainly as a last resort. Please report any required use to the OSA-CBM technical subcommittee.

It is preferred that if there exists an information class that can contain your data in OSA-CBM then that should be used.

It is under consideration to have this removed if no valid use can be found.

The BLOB class is a more standardized and supported approach to sending data types that are not directly in the OSA-CBM specification.

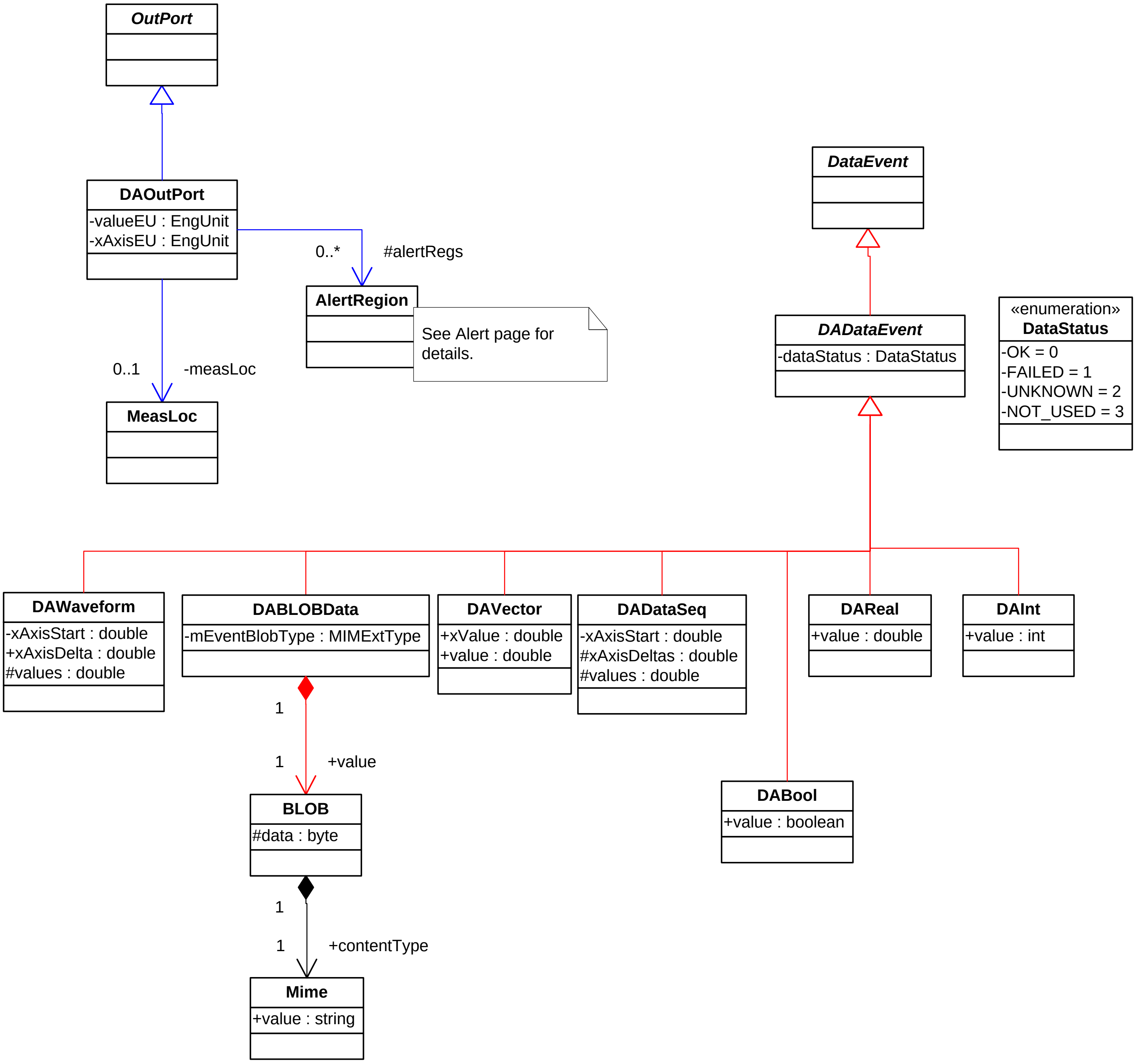
Additionally new types, such as multidimensional arrays, like wavelets can be added to the OSA-CBM specification fairly quickly.

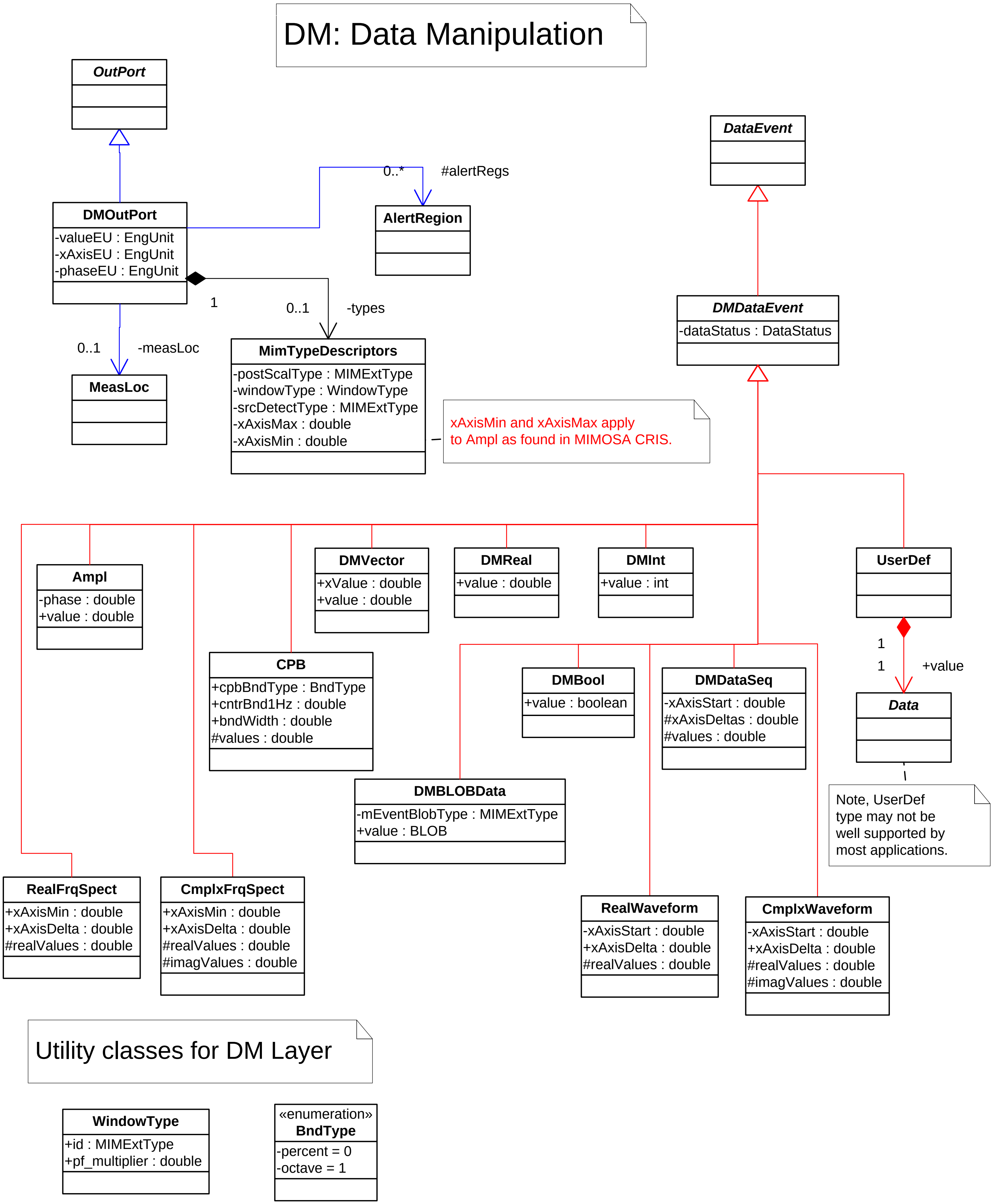


Optional time parameter for possible time stamp.

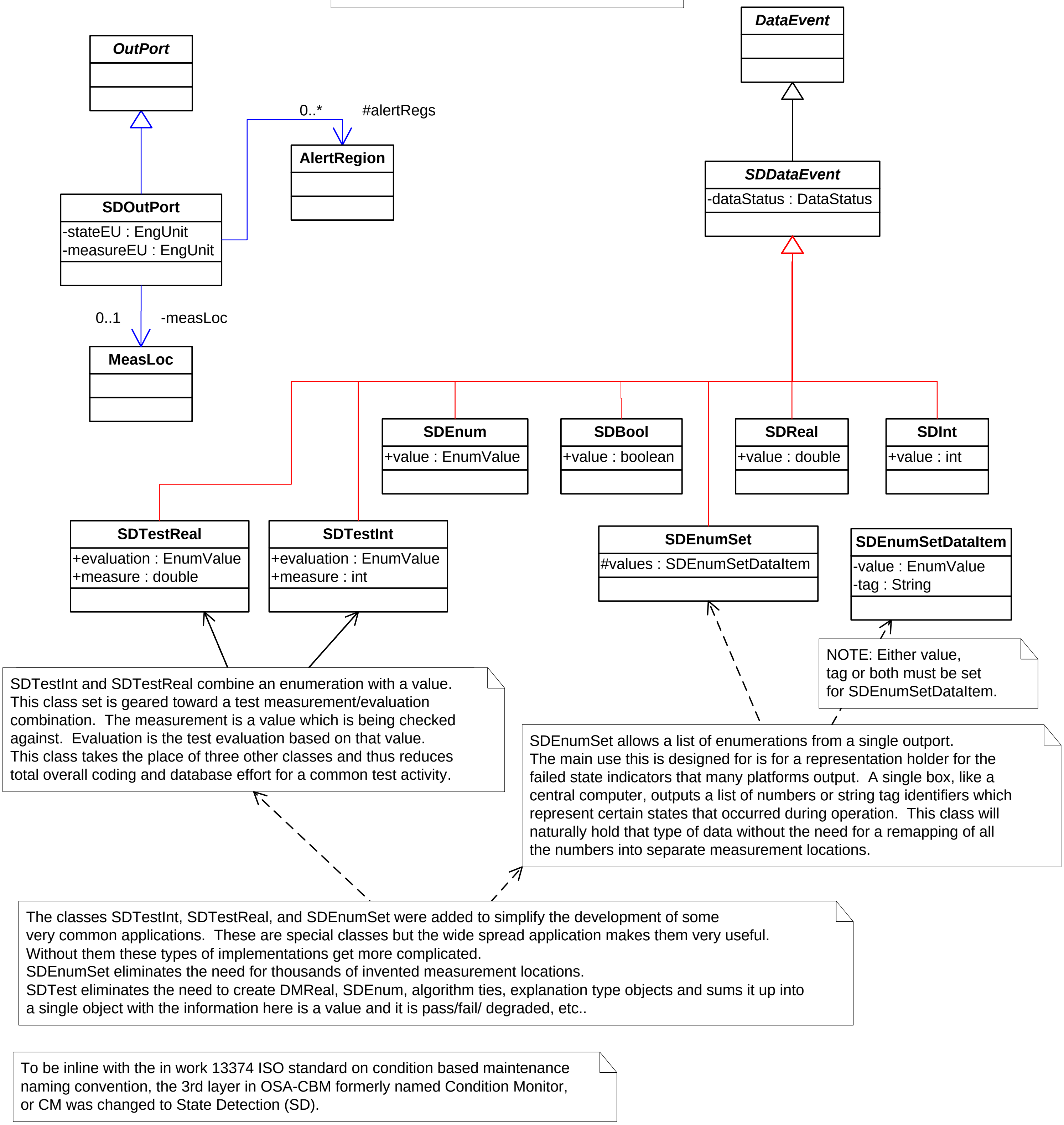
Composite Data is an optional attribute of Data class to allow for a general construction of a data item.

DA: Data Acquisition




DM: Data Manipulation
OutPort
DMOutPort
-valueEU : EngUnit
-xAxisEU : EngUnit
-phaseEU : EngUnit
0..*
#alertRegs
AlertRegion
1
0..1
-types
0..1
-measLoc
MeasLoc
MimTypeDescriptors
-postScalType : MIMExtType
-windowType : WindowType
-srcDetectType : MIMExtType
-xAxisMax : double
-xAxisMin : double
xAxisMin and xAxisMax apply
to Ampl as found in MIMOSA CRIS.
DataEvent
DMDataEvent
-dataStatus : DataStatus
Ampl
-phase : double
+value : double
DMVector
+xValue : double
+value : double
DMReal
+value : double
DMInt
+value : int
UserDef
1
1
+value
Data
CPB
+cpbBndType : BndType
+cntrBnd1Hz : double
+bndWidth : double
#values : double
DMBool
+value : boolean
DMDataSeq
-xAxisStart : double
#xAxisDeltas : double
#values : double
DMBLOBData
-mEventBlobType : MIMExtType
+value : BLOB
Note, UserDef
type may not be
well supported by
most applications.
RealFrqSpect
+xAxisMin : double
+xAxisDelta : double
#realValues : double
CmplxFrqSpect
+xAxisMin : double
+xAxisDelta : double
#realValues : double
#imagValues : double
RealWaveform
-xAxisStart : double
+xAxisDelta : double
#realValues : double
CmplxWaveform
-xAxisStart : double
+xAxisDelta : double
#realValues : double
#imagValues : double
Utility classes for DM Layer
WindowType
+id : MIMExtType
+pf_multiplier : double
«enumeration»
BndType
-percent = 0
-octave = 1

SD: State Detection



NOTE: Either value,
tag or both must be set
for SDEnumSetDataItem.

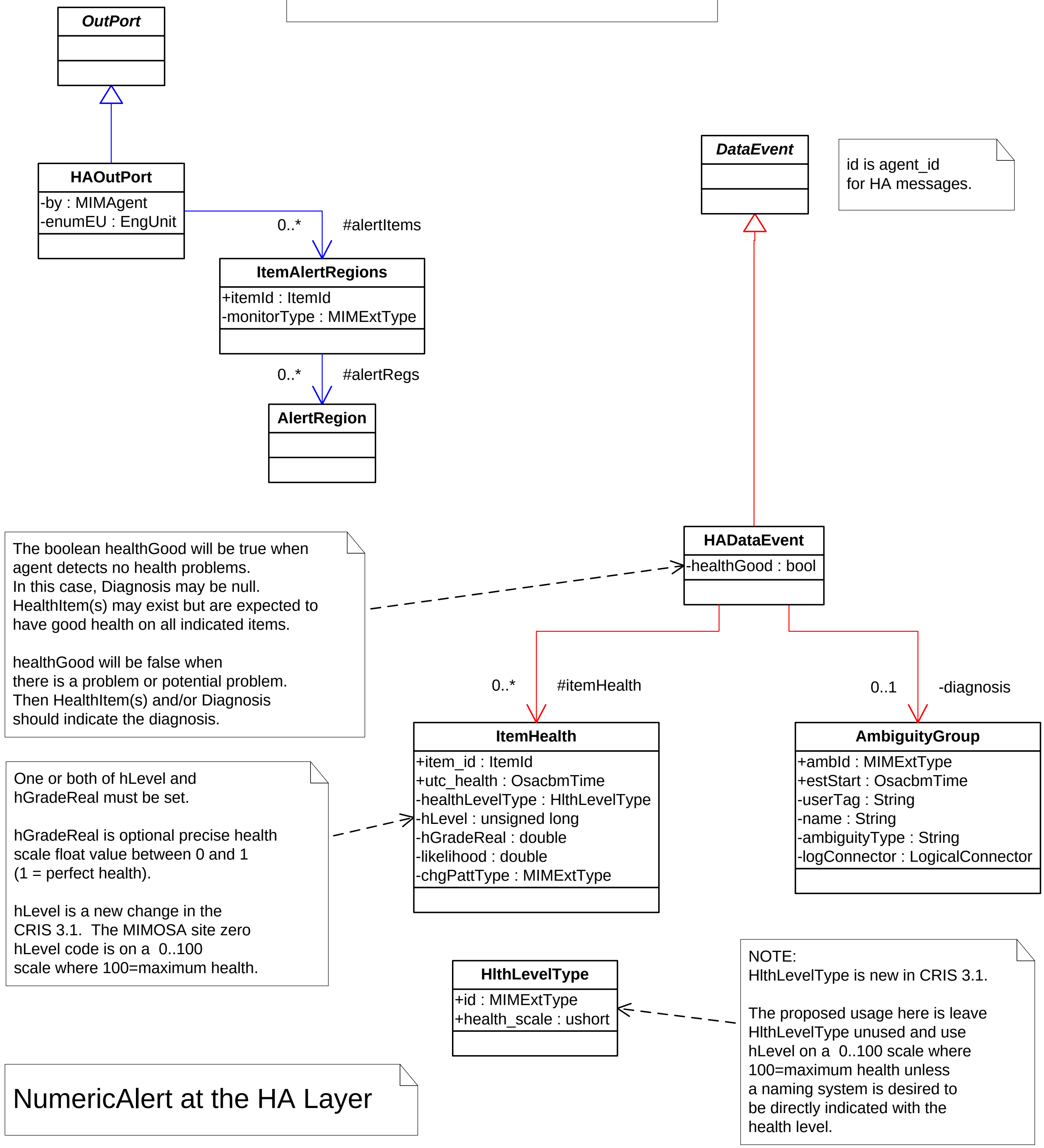
SDTestInt and SDTestReal combine an enumeration with a value. This class set is geared toward a test measurement/evaluation combination. The measurement is a value which is being checked against. Evaluation is the test evaluation based on that value. This class takes the place of three other classes and thus reduces total overall coding and database effort for a common test activity.

SDEnumSet allows a list of enumerations from a single outport. The main use this is designed for is for a representation holder for the failed state indicators that many platforms output. A single box, like a central computer, outputs a list of numbers or string tag identifiers which represent certain states that occurred during operation. This class will naturally hold that type of data without the need for a remapping of all the numbers into separate measurement locations.

The classes SDTestInt, SDTestReal, and SDEnumSet were added to simplify the development of some very common applications. These are special classes but the wide spread application makes them very useful. Without them these types of implementations get more complicated.
SDEnumSet eliminates the need for thousands of invented measurement locations.
SDTest eliminates the need to create DMReal, SDEnum, algorithm ties, explanation type objects and sums it up into a single object with the information here is a value and it is pass/fail/ degraded, etc..

To be inline with the in work 13374 ISO standard on condition based maintenance naming convention, the 3rd layer in OSA-CBM formerly named Condition Monitor, or CM was changed to State Detection (SD).

HA: Health Assessment



id is agent_id
for HA messages.

The boolean healthGood will be true when
agent detects no health problems.
In this case, Diagnosis may be null.
HealthItem(s) may exist but are expected to
have good health on all indicated items.

healthGood will be false when
there is a problem or potential problem.
Then HealthItem(s) and/or Diagnosis
should indicate the diagnosis.

One or both of hLevel and
hGradeReal must be set.

hGradeReal is optional precise health
scale float value between 0 and 1
(1 = perfect health).

hLevel is a new change in the
CRIS 3.1. The MIMOSA site zero
hLevel code is on a 0..100
scale where 100=maximum health.

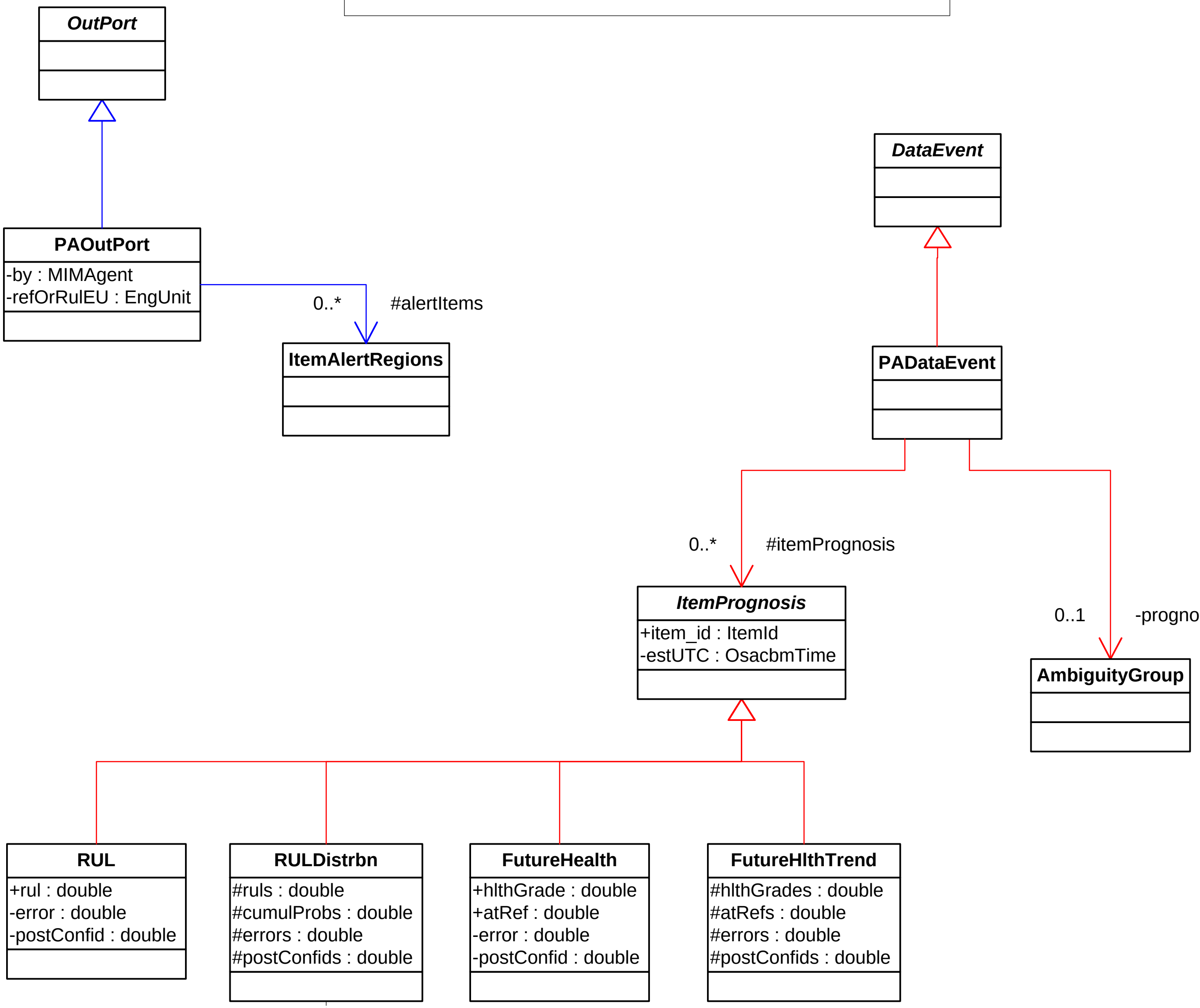
NOTE:
HlthLevelType is new in CRIS 3.1.

The proposed usage here is leave
HlthLevelType unused and use
hLevel on a 0..100 scale where
100=maximum health unless
a naming system is desired to
be directly indicated with the
health level.

NumericAlert at the HA Layer

What would NumericAlert be used for in the HA Layer?
The main use would be to have an active monitor on the health grade of a component.
This means the NumericAlert at the HA level requires the item monitored, i.e. ItemAlertRegions.

PA: Prognostics Assessment



RULDistrbn requires ruls and cumulProbs to be specified.
The arrays errors and postconfids are optional.

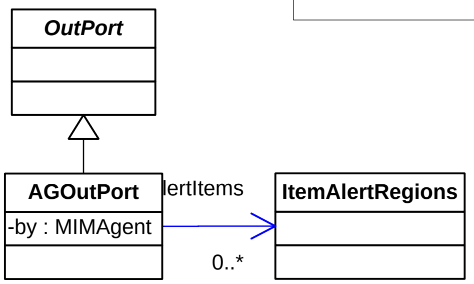
FutureHlthTrend requires hlthGrades and atRefs to be specified.
The arrays errors and postconfids are optional.

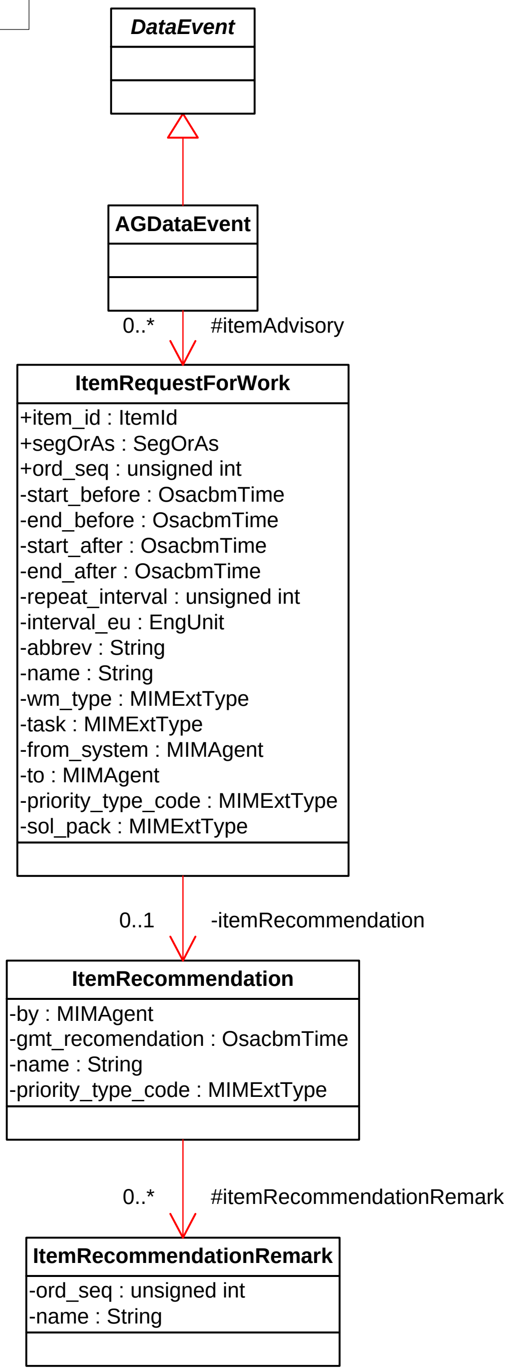
The length of all used arrays should be the same.

NumericAlert at the PA Layer

What would NumericAlert be used for in the PA Layer?
The main use would be to have an active monitor on the RUL or FutureHealth hlthGrate of a component. This means the NumericAlert at the PA level requires the item monitored.

AG: Advisory Generation





AG: Advisory Generation

The Advisory Generation is new in OSA-CBM. It allows for the direct request of maintenance for Assets and Segments. It associates to the following MIMOSA tables:

#134 - sg_req_for_work, #135 - as_req_for_work
#184 - sg_recommendation, #185 - as_recommendation
#130 - sg_rec_remark, #131 - as_rec_remark

These tables use the following utility tables:
#128 - work_manage_type, #108 - work_task_type,
#106 - priority_type, #181 - solution_package

NumericAlert in the AG Layer

What would NumericAlert be used for in the AG Layer?
The main use would be to have an active monitor on the remaining time duration to maintenance for actively updated time to maintenance lists.

Note of possible extra needs:
Simply giving a start_before / start_after may be insufficient.
It may be good to have additional task_begin time:
- Before next flight
- Before next mission of type X
- Immediately upon arrival
- Next Periodic Maintenance event

Request for Work for Segment
NOTE: start_before_gmt - Request for action to begin before this time
NOTE: end_before_gmt - Request for action to end before this time
NOTE: start_after_gmt - Request for action to start after this time
NOTE: end_after_gmt - Request for action to end after this time
NOTE: from_sy_agent_site - System the request generated from
NOTE: repeat_interval - Time interval to automatically have work re-submitted for time-based actions
NOTE: int_eu_db_site, int_eu_db_id, int_eu_type_code - Time interval eng unit reference (hours, days, months, etc.)
NOTE: to_agent_site, to_agent_id - Agent to recieve the request work
NOTE: sol_pack_db_site, sol_pack_db_id, sol_pack_id - Associated solution package
NOTE: rec_segment_site, rec_segment_id, rec_gmt_recomm, rec_by_agent_site, ec_by_agent_id
- Associated segment recommendation
NOTE: work_req_db_site, work_req_db_id, work_req_id - Associated Work Request in local or remote database
NOTE: abbrev - User-generated short work description
NOTE: name - User-generated full work description

Start of Note Pages

This pages ends the UML specification

and starts the notes pages for OSA-CBM.

Notes

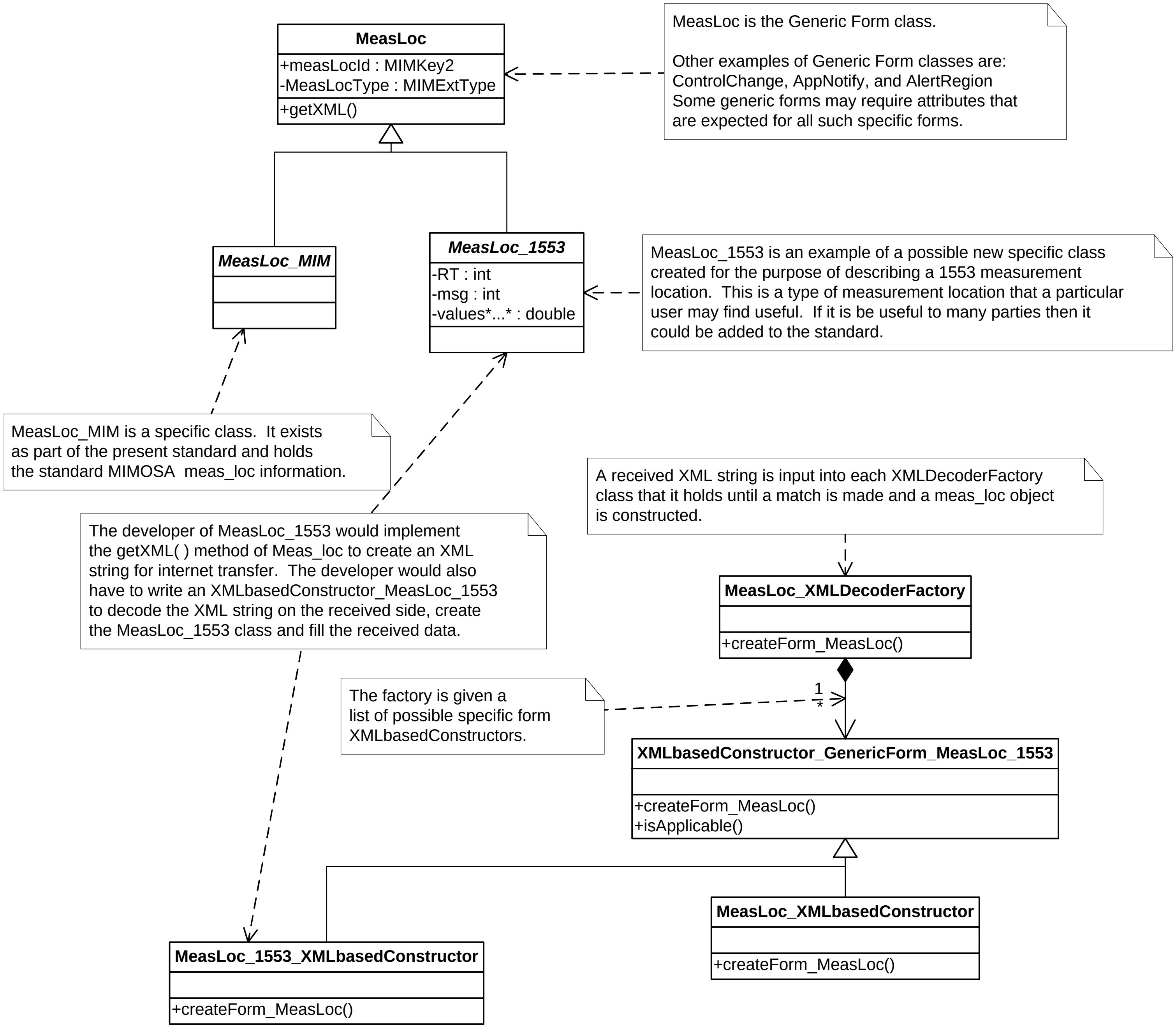
XML Extensibilty Concept

Extensible type classes are for application specific purposes. IVHM applications may require these specific categories of information for setup and control. These classes allow for a standard way to input and output application specific XML. The getXML is a suggested method for a parent wrapper class.

The main concept is to have a class that is closed to modification but extensible to users The XML string is used as the conveyor of data.

Implementations should have a getXML(...) method to retrieve a transmittable XML form.

Specific Implementations can use the specific form class structure.



MeasLoc is the Generic Form class.

Other examples of Generic Form classes are: ControlChange, AppNotify, and AlertRegion Some generic forms may require attributes that are expected for all such specific forms.

MeasLoc_1553 is an example of a possible new specific class created for the purpose of describing a 1553 measurement location. This is a type of measurement location that a particular user may find useful. If it is be useful to many parties then it could be added to the standard.

MeasLoc_MIM is a specific class. It exists as part of the present standard and holds the standard MIMOSA meas_loc information.

A received XML string is input into each XMLDecoderFactory class that it holds until a match is made and a meas_loc object is constructed.

The developer of MeasLoc_1553 would implement the getXML( ) method of Meas_loc to create an XML string for internet transfer. The developer would also have to write an XMLbasedConstructor_MeasLoc_1553 to decode the XML string on the received side, create the MeasLoc_1553 class and fill the received data.

The factory is given a list of possible specific form XMLbasedConstructors.

Mapping Methodologies

Mapping Methodologies

The goal is to have OSA-CBM map totally seamlessly into OSA-EAI. The grand picture is the following.

1. Simple 1-to-1 information component mapping.
2. OSA-CBM extensions to CRIS that have extra information that OSA-EAI does not need.
3. A mapping document where difficult mappings or mappings that have many potential solutions are specified to be done in only one way.

The following provides a quick overview.

Perhaps 90% or more of OSA-CBM will mapping directly into OSA-EAI with ease.
There will be some changes and additions in OSA-EAI to facilitate the mapping also.
However there are certain differences and some OSA-CBM needs which
are to be handled by OSA-CBM extensions in the MIMOSA specification.
The main requirement for the extensions deals with the ability to get data out of database
storage in the original OSA-CBM format with all class structure intact and easily retrievable
by a generic mechanism rather than having to hard code an expected form for a
particular known configuration.

Main areas for the extensions includes
1) The OSA-CBM class-type definition specifics.
The ability to know which OSA-CBM class was used to transmit the data.

2) OSACBM time stamp based message identification scheme.
In OSA-CBM all messages such as measurement_events and health_assessements
which includes proposed_events are identified by agent or meas_location, time stamp,
and item id in the HA and PA layers. OSACBM small signature vehicles are not expected
to generate new integer primary key signatures. The ability to do that would require
non-volatile memory storage of some form to remember last number used.
Instead, OSA-CBM offers a slightly different primary key basis (agent, time, item).
All the same important information components as those found in OSA-EAI are there.
When such a message reaches a the OSA-EAI database location the OSA-EAI proposed
event primary key signature may be generated.

3) Ambiguity Groups
OSA-EAI (V3.2?) will be enhanced to accommodate ambiguity groups.

4) Explanation
Explanation is the ability to state connection between data used as input and resultant data.
The OSA-CBM explanation uses references within the OSA-CBM context. OSA-CBM
extensions Explantion table will be used by those desiring this information to be retrievable.
OSA-EAI has many data tie tables for those desiring this information tie in the OSA-EAI context.