

Copyright 2008, Machinery Information Management Open Systems Alliance

OSA-CBM stands for Open System Architecture for Condition Based Maintenance.

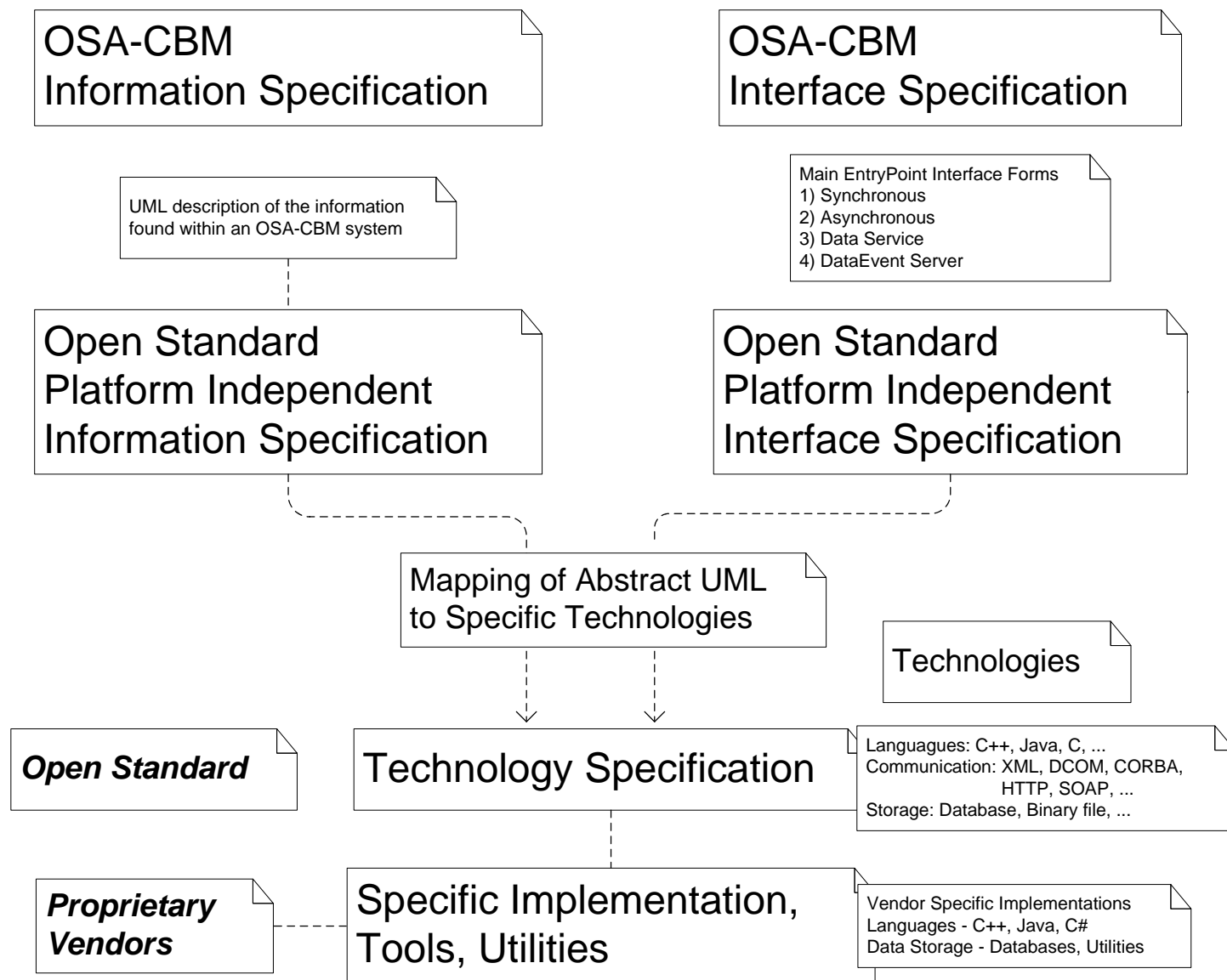
This specification is offered by the MIMOSA organization. Information on this organization can be found at <http://www.mimosa.org>.

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.

OSA-CBM version 3.2 is compatible with the OSA-EAI version 3.2.

OSA-CBM conforms to ISO 13374, a Condition Based Maintenance standard.

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.



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 provides the abstract UML description of the specification.

This architecture is divided into the information specification, which defines the information that can be moved around in a CBM system, and the interface specification, which is 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 interfaces and merge them with the information specification into a complete package. 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. It also defines several interfaces that may be used to transport information.

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 will then be used to validate a system that is required to output OSA-CBM XML.

Notes on Compliance

The initial implementations for the OSA-CBM 3.2 technology mapping are an XML schema for the information content and a WSDL document for the interface implementation. Programs can be verified against the XML schema and WSDL document.

Other implementations will be added as they are developed.
New technology mappings will determine their means of compliance at the time of acceptance.

Interfaces

Interface Types

- 1) Synchronous
- 2) Asynchronous
- 3) Data Service
- 4) DataEvent Server

There are several interface types to accomodate different purposes and technological capabilities. A given implementation will likely not implement every interface. Rather, the technology of choice will specify possible interfaces, and an implementation will select appropriate interface(s) from that subset.

Example: A Web server returning XML over HTTP would likely use the synchronous interface.

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 - Asynchronous data return path for requested information.

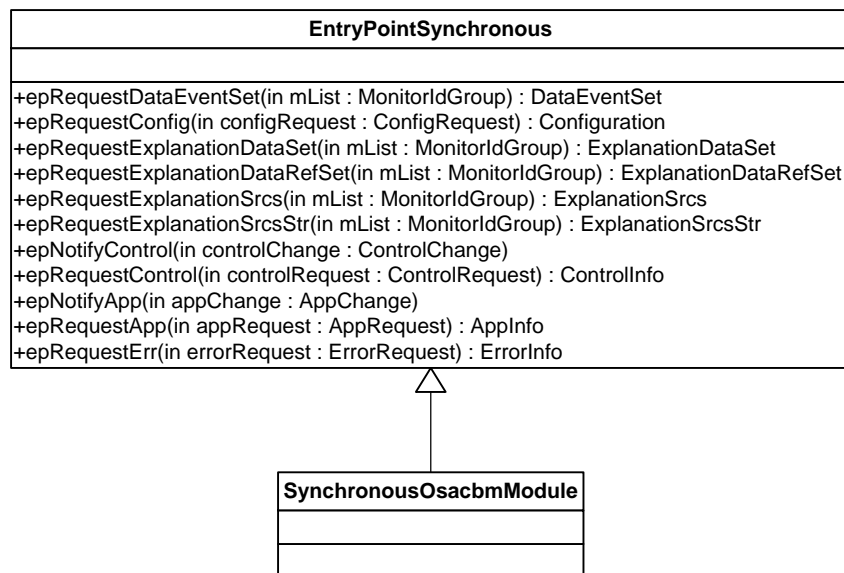
Synchronous Interface - Information is returned by the Request method.

Asynchronous Interface - Information is returned as available via an EntryPointSink object.

Synchronous Interface

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

Example (C++):
newData = moduleEPptr->requestDataEventSet(...);



Asynchronous Connected Module Interface

The Asynchronous Interface allows any number of higher modules to establish and maintain a two-way connection for duration of need.

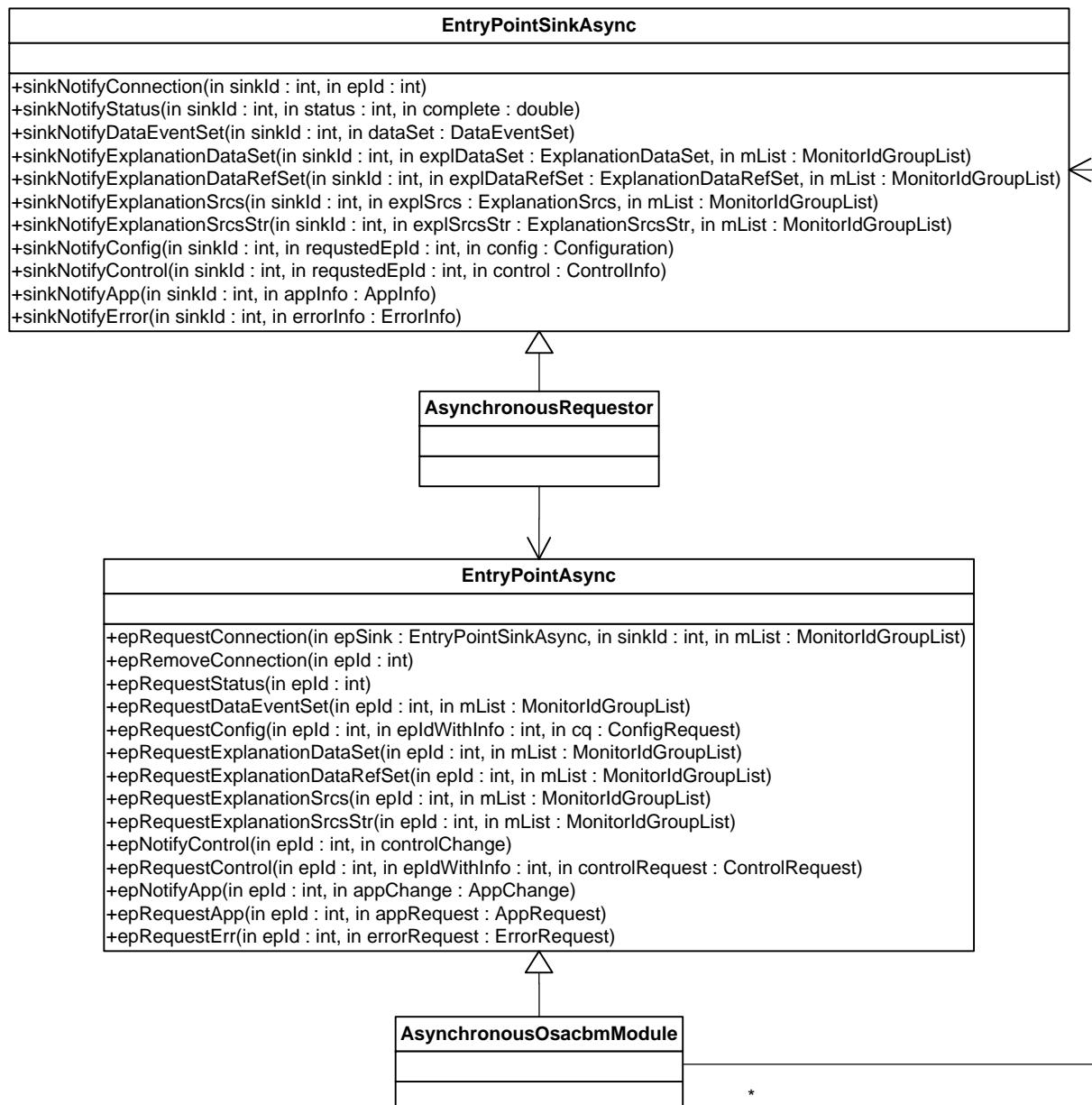
The sinkId and epId are used for specific sink and entry point identification.

The two-way connection has two main advantages over synchronous calls:

- 1) It is typically faster in usage since the overhead of connection occurs only once.
- 2) It allows for three different modes of communication.
 - Return on request, which was the main OSA-CBM 1.0 style of communication.
 - Return on alert, such as when a threshold is exceeded. The connection can be set up for notification only on alert.
 - Push All, which pushes data to the higher connected module every time it collects data without the need for a request beforehand.

Example method call interplay with the connection oriented Asynchronous Interface (C++):

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



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

EntryPointService
+serviceNotifyDataSet(in dataSet) : EPSSStatus +serviceNotifyConfig(in config) : EPSSStatus +serviceNotifyExplanationDataRefSet(in explDataRefSet) : EPSSStatus

EPSSStatus
+status : string

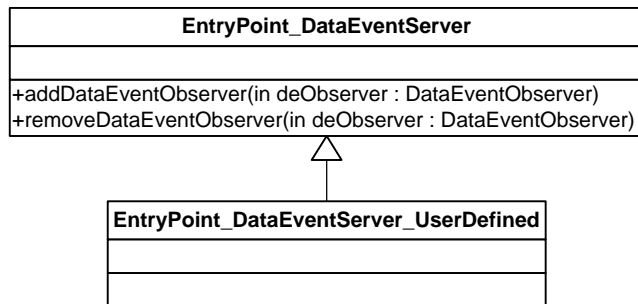
EPSSStatus is a return indicator of how an input message was received.

DataEvent Server

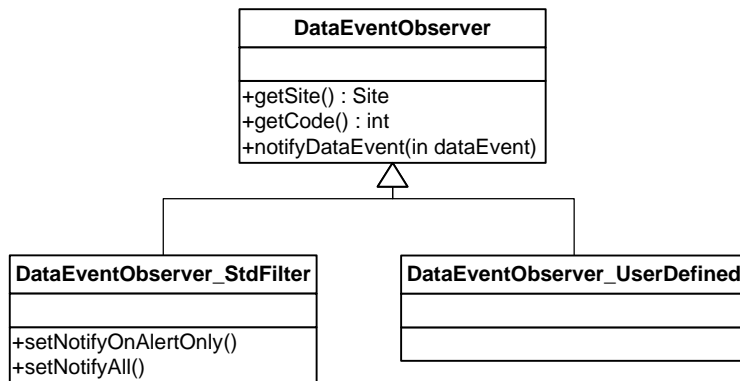
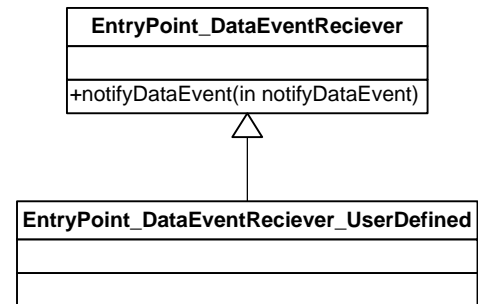
In many systems, signals are moved individually.
This page describes an interface mechanism for handling an individual DataEvent in a simplified interface.

All classes marked with "_UserDefined" are abstract representations of some class that is written with any desired class name with specifically desired functionality for special handling of signals.

Serving module interface



Receiving module interface



A module that provides DataEvents will inherit from the EntryPoint_DataEventServer.

A module that is to receive 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.

It is possible to put filtering of events into the DataEventObserver class via a child class. DataEventObserver_StdFilter is designed 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)

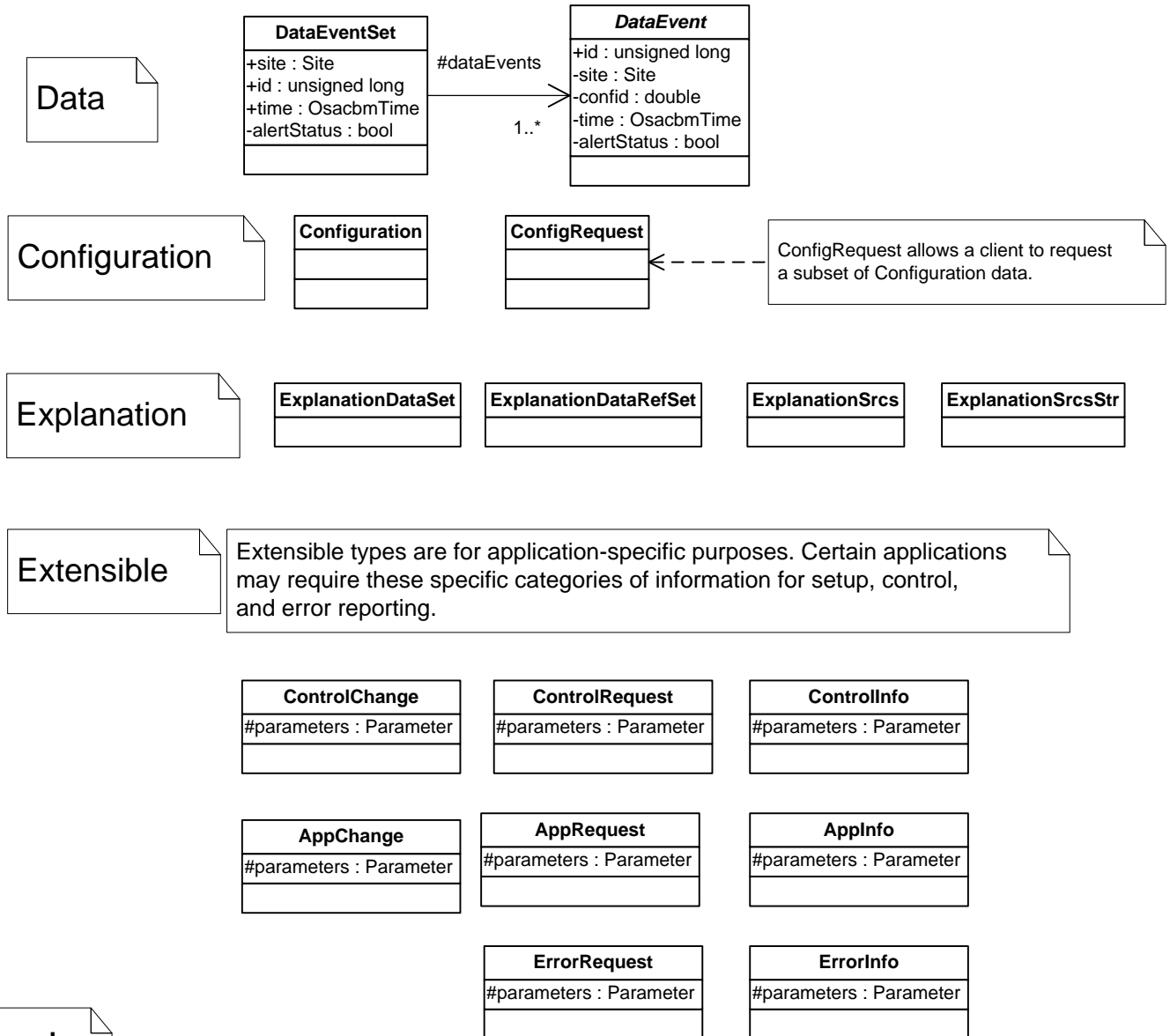
Each of these is individually addressable in the interfaces.

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 known 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 which contain the configuration information may exist at servicing locations.

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.



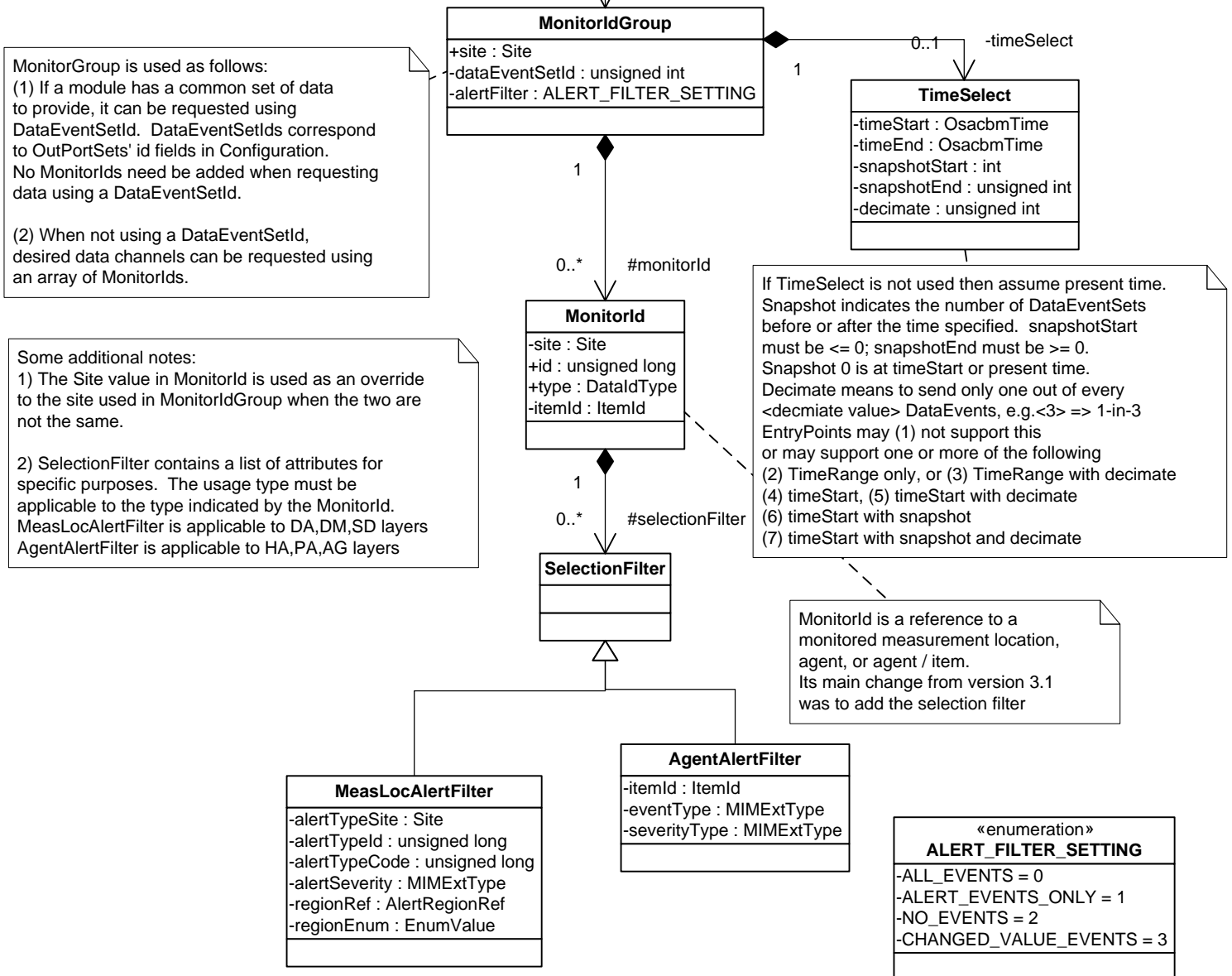
Legend

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

The legend explains some of the OSA-CBM specific nomenclature. Note especially the '-', '+', and '#' used to indicate parameter optionality and count. These symbols have different meanings from those in the standard UML specification.

MonitorIdGroupList

```
MonitorIdGroupList
MonitorIdGroup
MonitorId
```

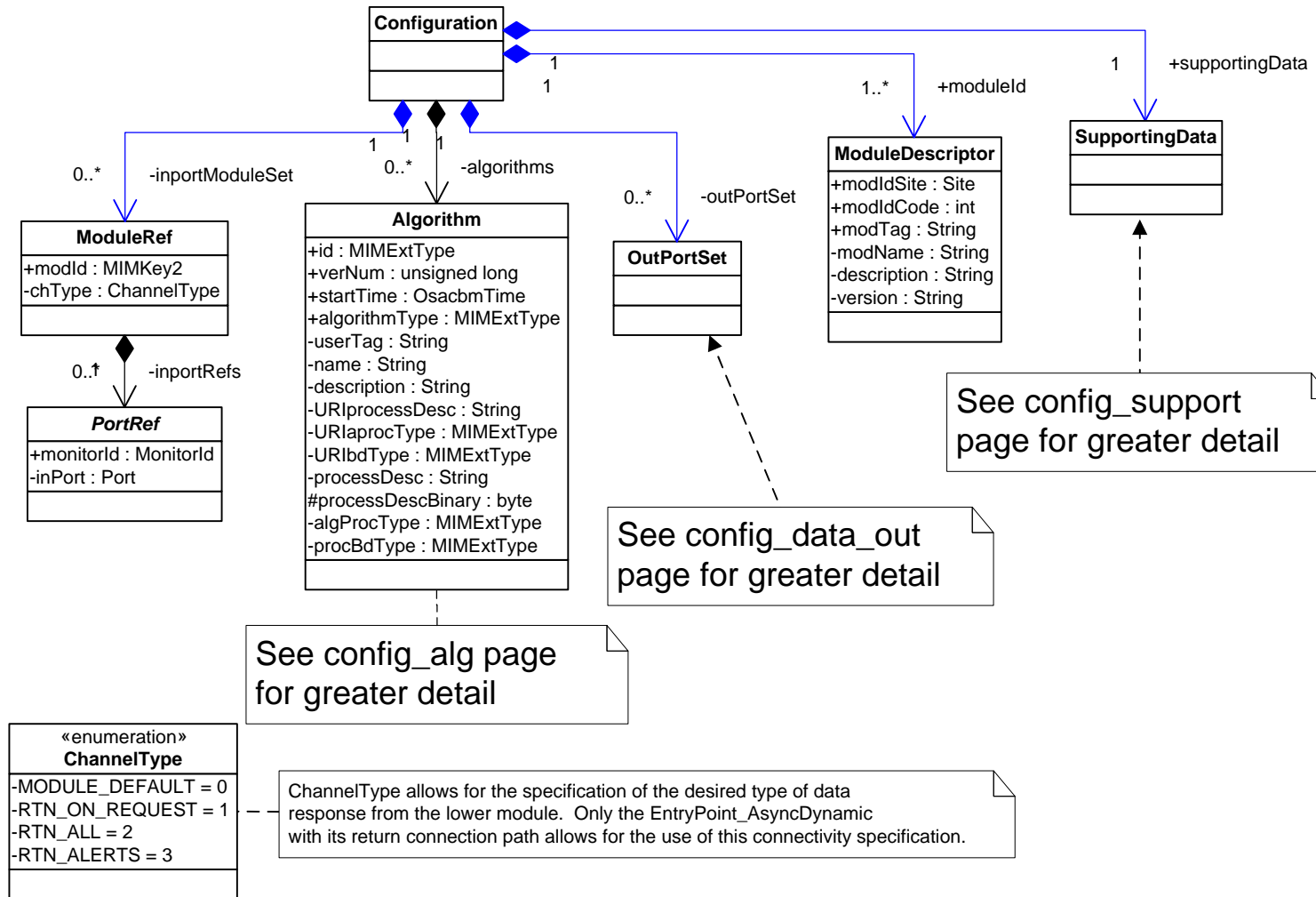


The Synchronous Interface must return all data in one call. Therefore `epRequestDataSet` will return all applicable `dataEvents` in the one `DataEventSet`. Additionally, the Synchronous Interface will mainly be for historical data.

Copyright 2008, Machinery Information Management Open Systems Alliance

Configuration

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



ModuleRef gives information about where a module gets data from.

Algorithm describes the process used to generate a DataEvent.

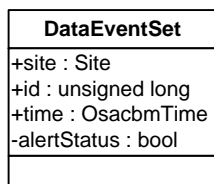
OutPortSet lists each Port provided by the module. A Port is a 'data channel' and the Port class gives specific configuration data for that data channel.

SupportingData gives additional information about MIMOSA primary key references which may be used elsewhere in this architecture.

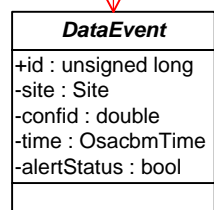
ModuleDescriptor gives a top level description of the module/entryPoint. modIdCodes should correspond to OutPortSet ids and DataEventSet ids

Copyright 2008, Machinery Information Management Open Systems Alliance

DataEventSet



1..* #dataEvents



Port contains configuration/metadata 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.

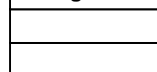
OutPortSet is optional, but any application that wants or requires strong meta information, such as engineering units, should use it.

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

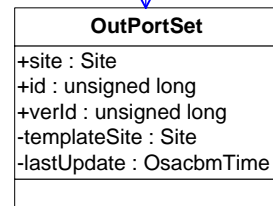
The DataEvent optional parameters on time and site override the values present in DataEventSet.

OutPortSet

Configuration

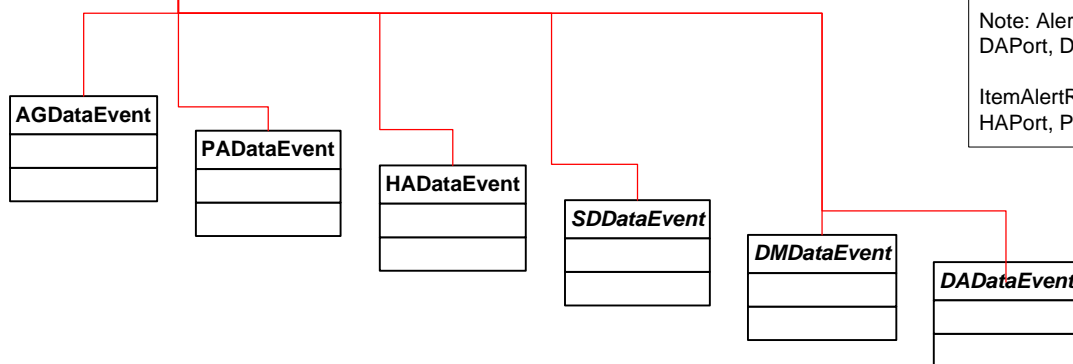
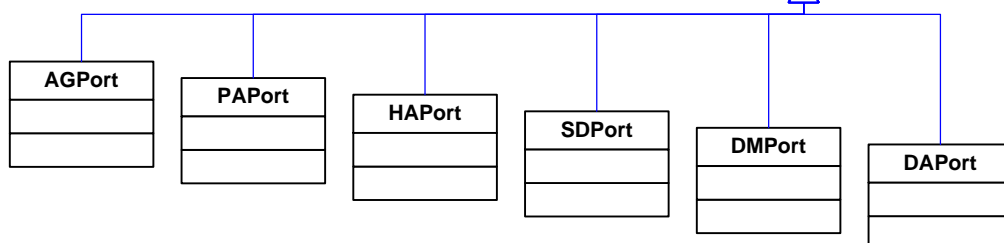
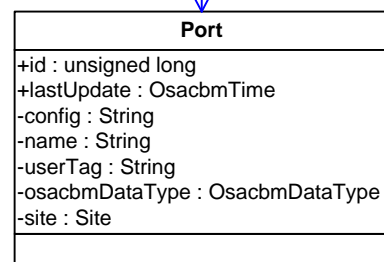


0..* -outPortSet



#outPorts

1..*



Note: AlertRegion is used by DAPort, DMPort, and SDPort.

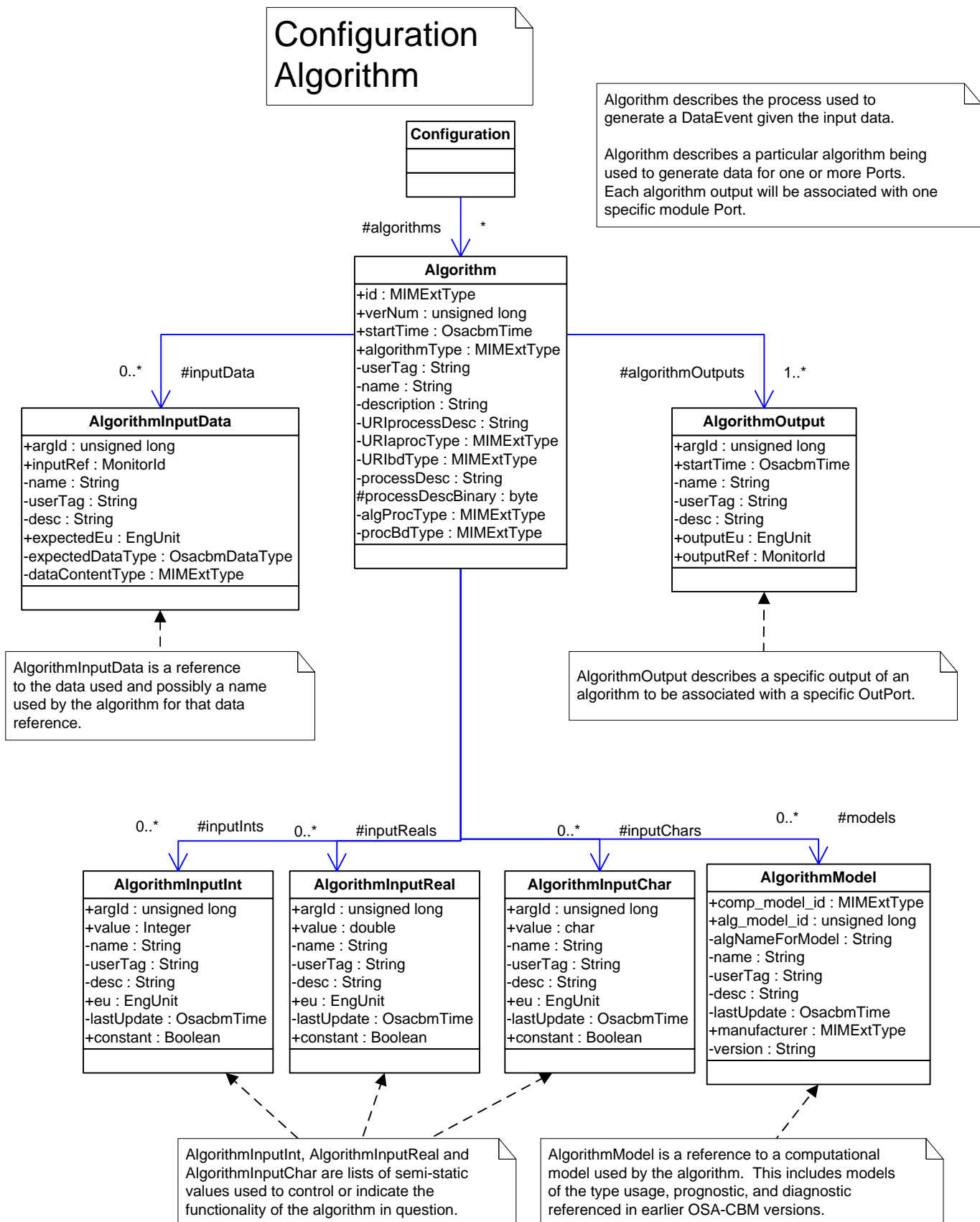
ItemAlertRegion is used by HAPort, PAPort, and AGPort.

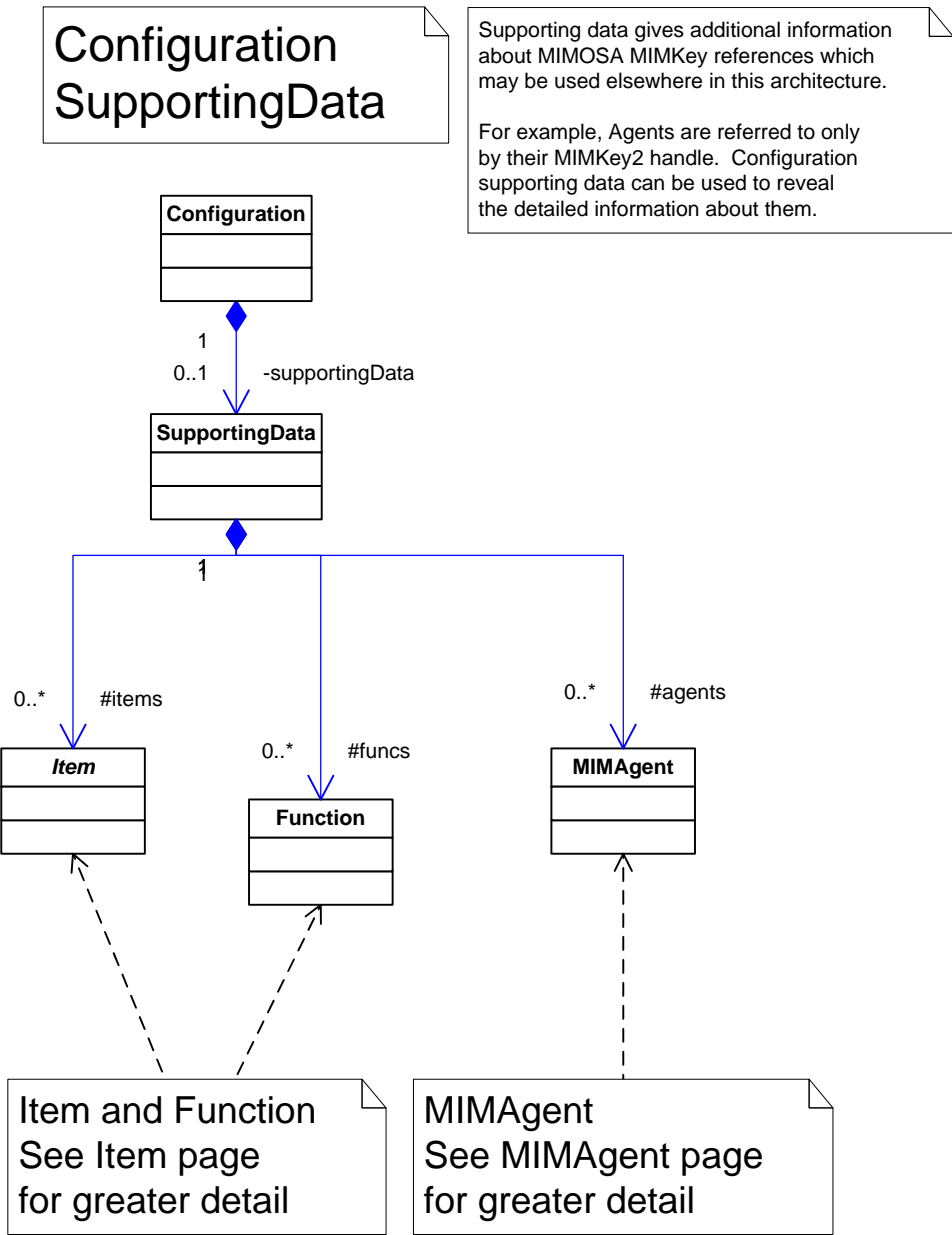
DataEvent contains the data generated by one event of one Port.

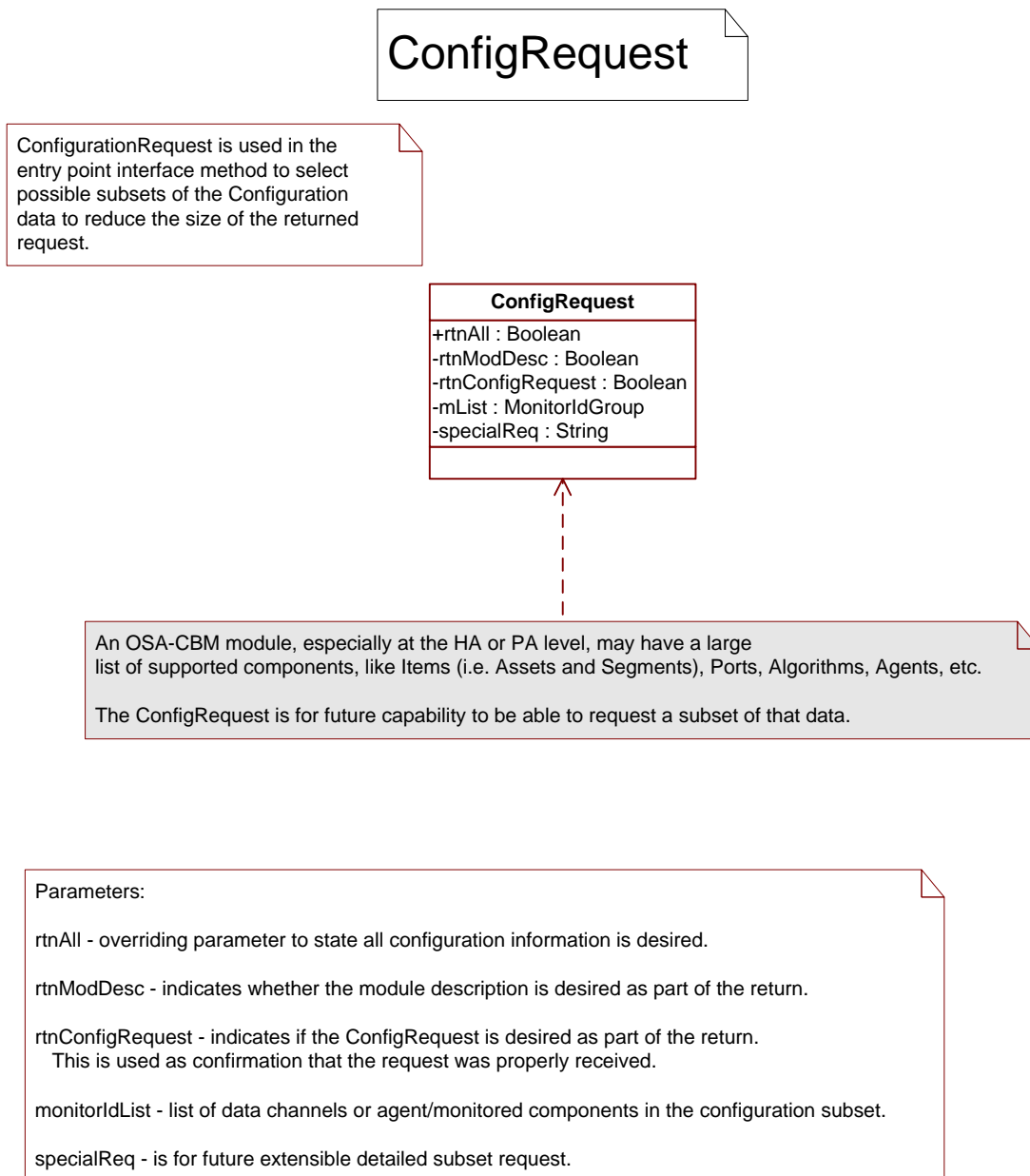
The DataEvent child hierarchy below it is associated with particular layers in the OSA-CBM architecture. Those classes have child classes below them describing particular data types.

Port contains the configuration information specific to one output channel of a module. The Port 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 when fetching single DataEvents from a DataEvent Server.







Explanation

Explanation is the data or a reference to the data used by a module to produce an output.

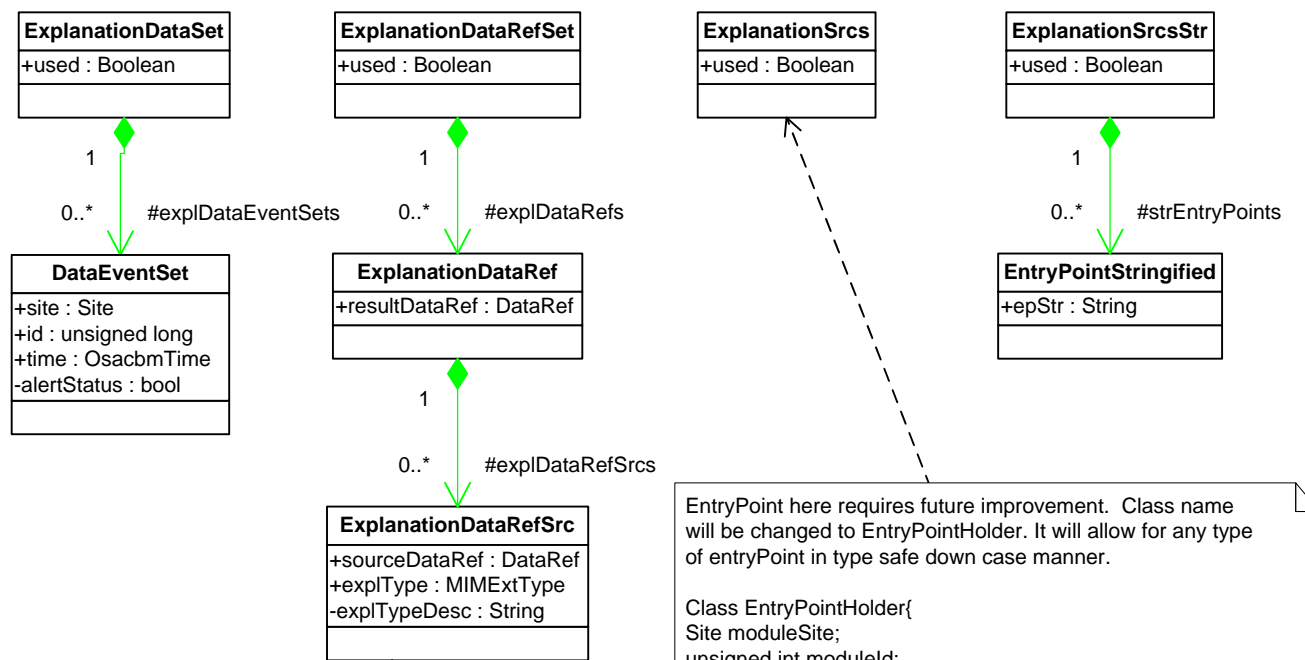
Explanation consists of four possible forms depending upon the application.

The ExplanationDataSet is simply the data used for a calculation.

ExplanationDataRefSet is 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.

ExplanationSrcs and ExplanationSrcsStr are two different ways of giving direct access to the modules supplying the data. The former is a set of direct pointers to modules. The other latter 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, set the boolean 'used' to false and return an empty set.

Extensible Components

These are called the extensible components because they are application specific and meant to hold any kind of data. The parameter class can be extended as necessary to provide ways to read and write parameters on a module.

GeneralParameter is recommended to provide application-specific parameters in a standard, human-readable format.

Because these classes are application-specific, they are optional. It is acceptable 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.

epNotify

ControlChange

#parameters : Parameter

epNotify

AppChange

#parameters : Parameter

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

epRequest

ControlRequest

#parameters : Parameter

epRequest

AppRequest

#parameters : Parameter

epRequest

ErrorRequest

#parameters : Parameter

Notify
Rtn

ControllInfo

#parameters : Parameter

Notify
Rtn

Notify
Rtn

AppInfo

#parameters : Parameter

Notify
Rtn

ErrorInfo

#parameters : Parameter

GeneralParameter is an all-purpose parameter that can be implemented in nearly any language. More specific parameter types can be added, inheriting from Parameter to implement strong typing, better indexing, or other functionality.

Parameter

-description : String

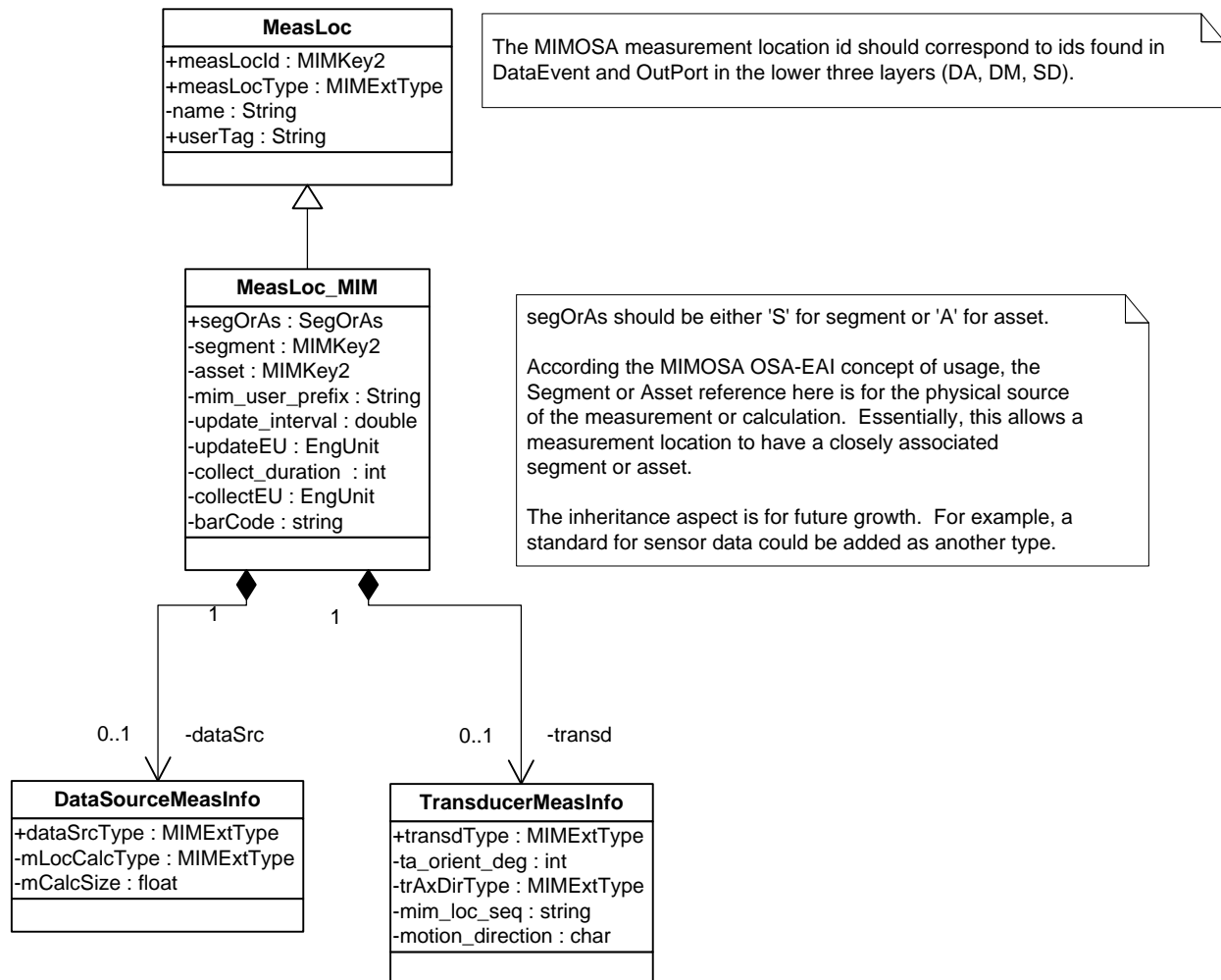
GeneralParameter

+dataType : XmlDataType
+name : string
+value : string

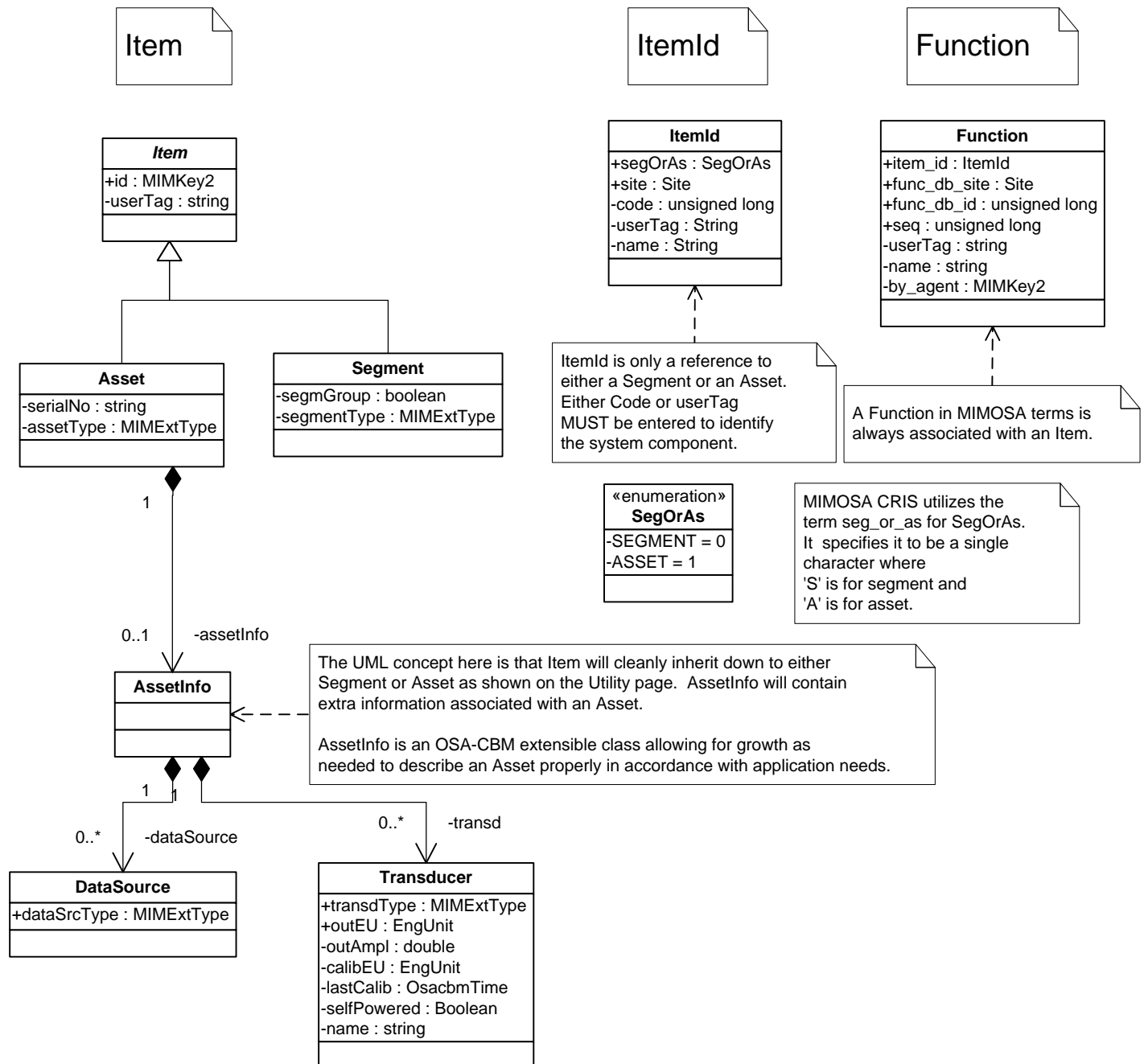
«enumeration»
XmlDataType

+boolean = 0
+dateTime = 1
+decimal = 2
+double = 3
+float = 4
+integer = 5
+string = 6

Measurement Location (DA, DM, SD)

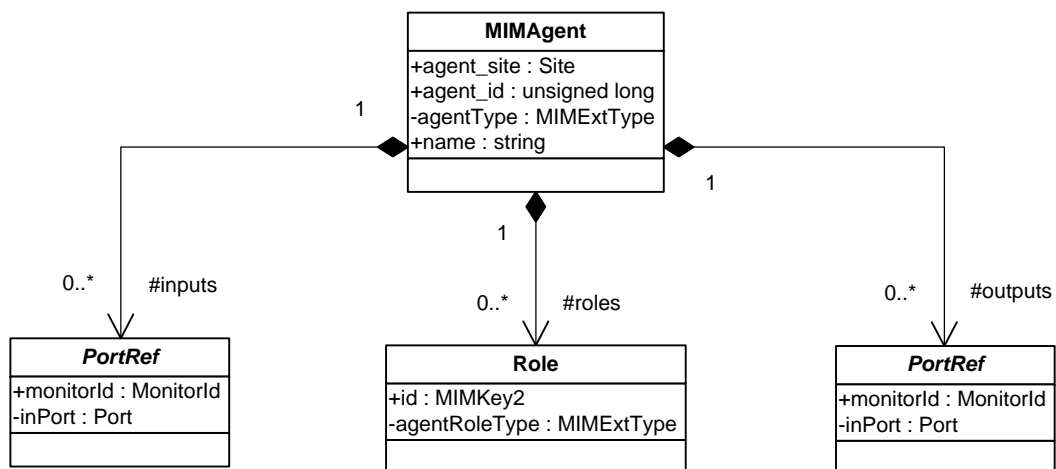


The Item page details the Item and Function classes that may be used or referenced elsewhere in this document.



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

MIMAgent and Roles



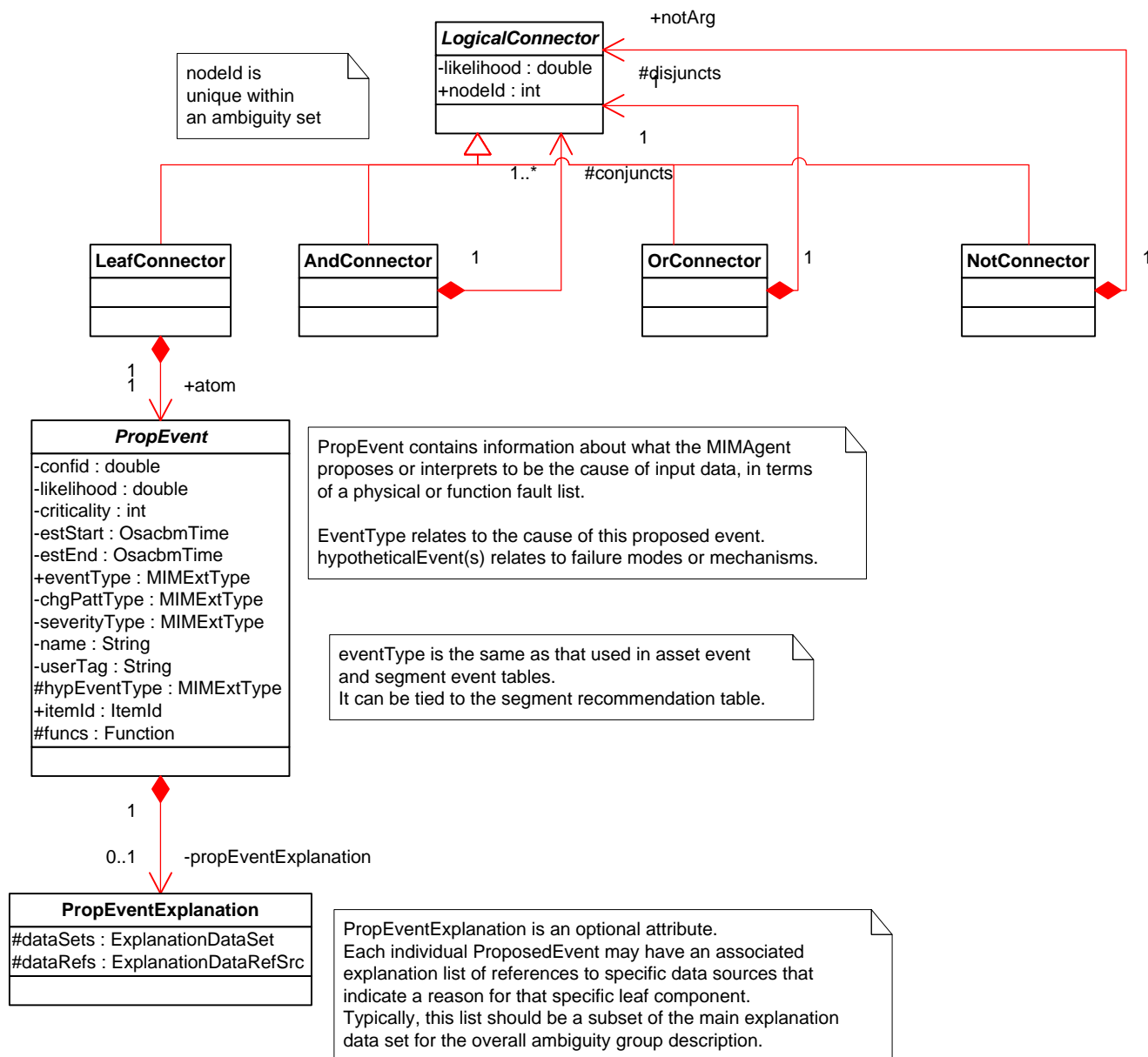
Proposed Event for Failure Descriptions (HA,PA)

Copyright 2008, Machinery Information Management Open Systems Alliance

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.



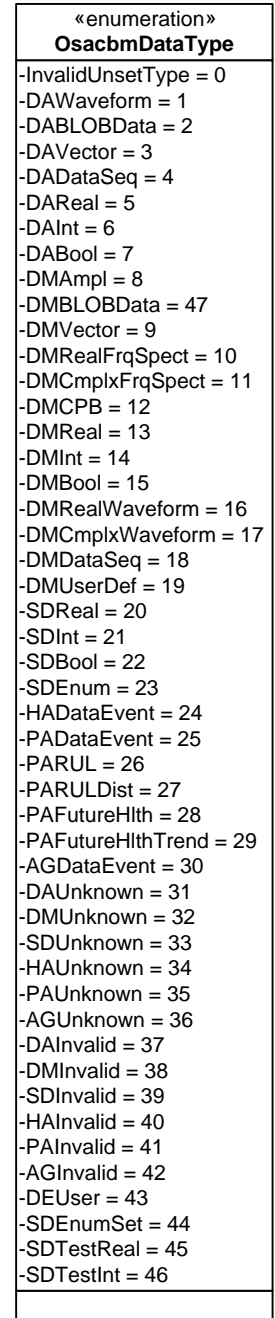
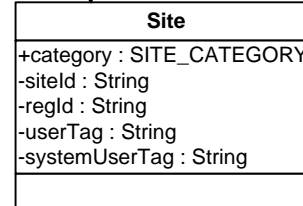
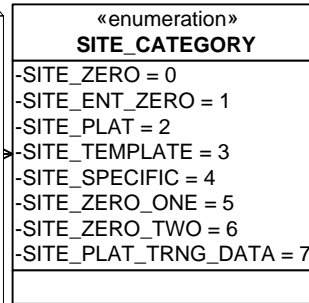
Site

Site is globally uniquely identified by one of two methods. Either the MIMOSA assigned 16 hex character siteld or the (regld, userTag) string combination where regld 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.

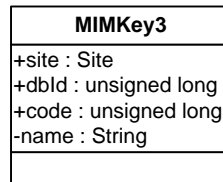
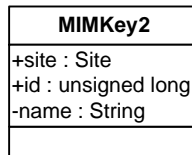
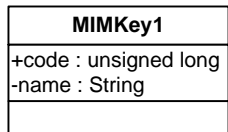
More specifics on these strings is described in the MIMOSA CRIS documentation.

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).
 SITE_PLAT_TRNG_DATA is for simulated platforms.



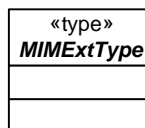
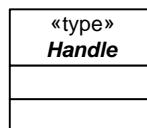
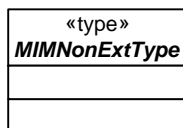
MIMKeys: MIMOSA Table Keys



MIMKey Type Defs

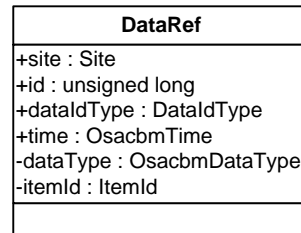
```

typedef MIMKey1 MIMNonExtType
typedef MIMKey2 Handle
typedef MIMKey3 MIMExtType
  
```



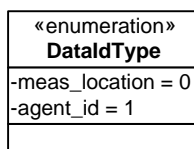
MIMNonExtType is a MIMOSA non-extensible type. It is therefore a single integer.
 Handle is used to indicate a specific MIMOSA measurement location (DA, DM,SD) or agent id (HA,PA,AG).
 MIMExtType is a MIMOSA extensible type. It has three keys: site, dbld, and code.
 (Handle, code) may be put into a MIMExtType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

DataReference



DataRef 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



DataIdType describes what kind of id a DataEvent id references. 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 ids from these different data types therefore correspond to these two types of sources: agents and measurement locations.

Time

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

OSA-CBM uses the name `OsacbmTime` to eliminate clashing of class names. Either 'time' (as a string) or `time_binary` should be used.

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 for a specific embedded system.

'time' as string should be set to the MIMOSA format which is a string conforming to ISO 8601. See description to the right.

'time_binary' is defined for a binary format indicator of time. There are two main types: POSIX and tick.

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.ffffff
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)
ffffff = represents a decimal fraction of a second to the billionth of a second

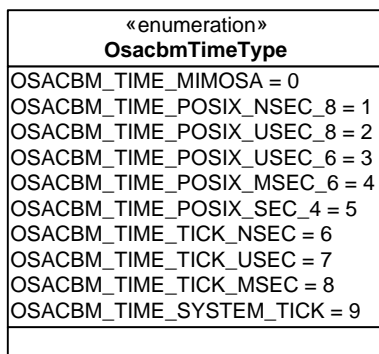
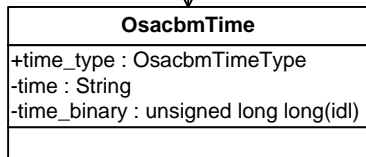
Year, month, and day must be specified. Additional timestamp content should be provided, if known. Zeros will be assumed for the omitted values. Negative DATETIME is not supported. All suffixes after the 29th character provided in the ISO 8601 specification, such as "Z" (representing Coordinated Universal 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.

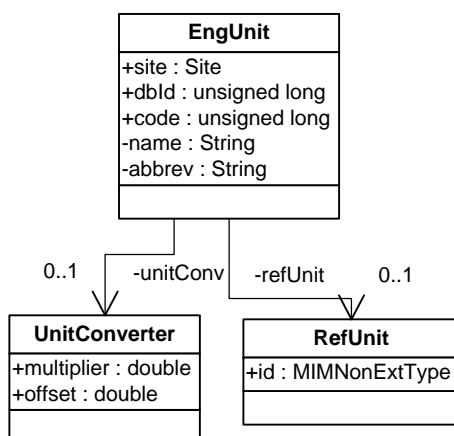
POSIX time is a signed integer indicating a time since the Unix epoch time 00:00:00 UTC on January 1, 1970. LSB value is a multiplier to give time distance from the epoch time. Bytes used can vary and determines how wide the time range around the epoch time can be. The `OsacbmTimeType` for POSIX indicates the LSB value and number of bytes used. The following indicates some of the possible time ranges

# Bytes	LSB	Delta Time to 1970
4 Signed	1 sec	+/- 68 years 1901 to 2038
6 Signed	1 mSec	+/- 4459 years
8 Signed	1 nSec	+/- 292years
8 Signed	1 uSec	+/- 74 million years

Time Tick is for an embedded system that uses a counter as a timer, typically since power up. Three specified versions are for nano, micro, and milliseconds. Other variations from this should use `_SYSTEM_TICK`. It is a systemic problem to convert tick time into a UTC time. This should be done for any stored or transmitted data.



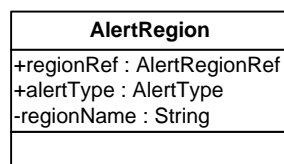
EngUnit and Enum Type



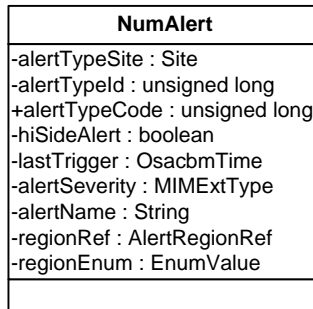
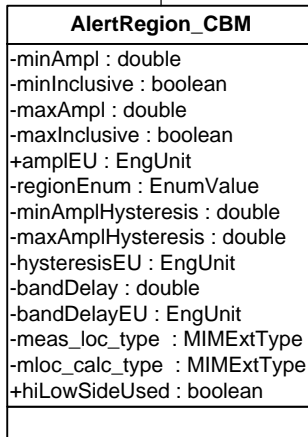
Enum value is uniquely identified by EngUnit plus value. Name may be transmitted optionally.

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

Alerts and Regions



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

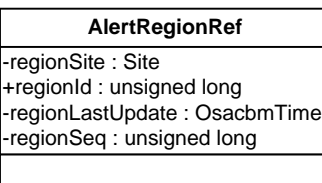


NumAlert carries with it information about:

- (1) AlertType directly from AlertRegion.
- (2) Optional regionId for direct tie to the region.
- (3) Optional enum value associated with region.

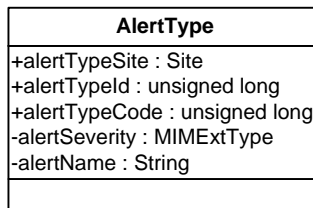
Simplest usage:

- 1) Set alertTypeCode 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 an 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) .



Usage Concept

A set of AlertRegions will be associated with a specific OutPort. The value associated with the AlertRegion set is contained in the DataEvent with the same id as the OutPort.

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 occurrence 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

Hysteresis 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 time.

Hi-Low

hiLowSideUsed default is false. MIMOSA OSA-EAI does not presently support this functionality.

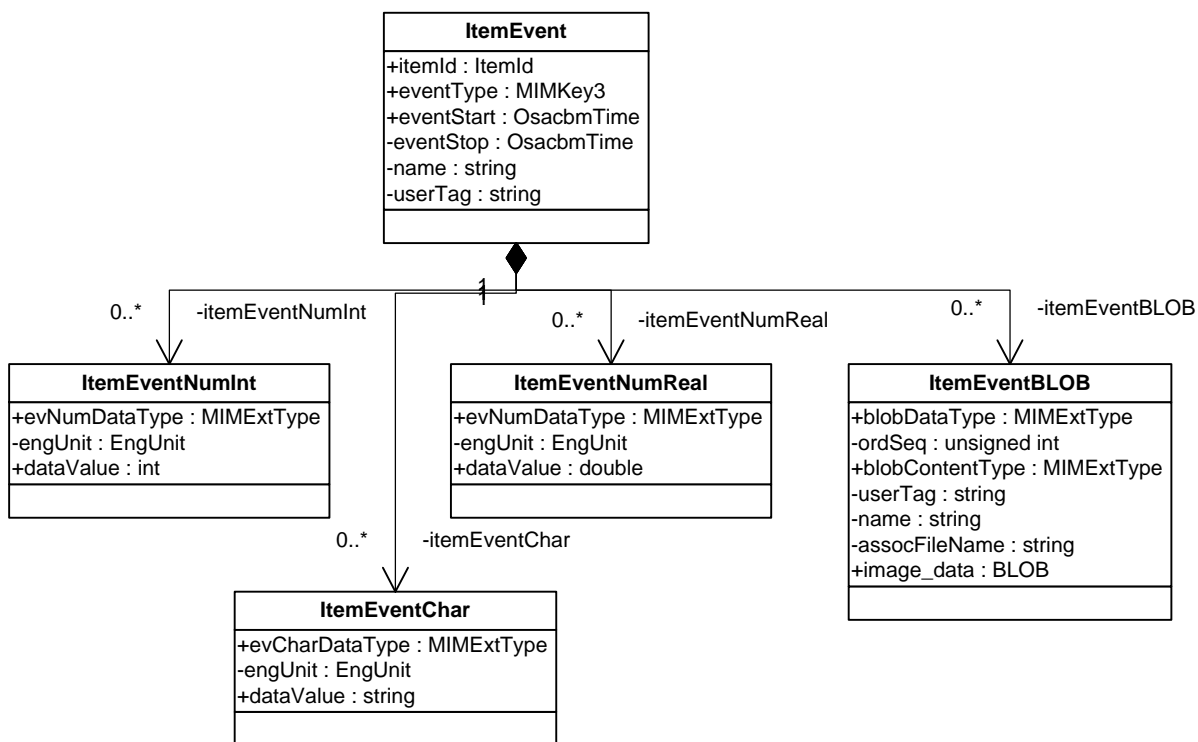
The simplest method of use is to have a single set of alertRegions for an output and a direct set of alert types for those regions. The system should be set up so that AlertTypes and AlertRegions are all unique within the system. Then only a single int is needed to identify the code, the RegionId, and 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 hysteresis and hiLowSideAlert 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.

ItemEvent

ItemEvent supports events in the State Detection layer that might correspond to the MIMOSA OSA-EAI tables segment_event and asset_event



ItemEvent can be an OSA EAI asset_event or segment_event. For those ItemEvents which do not have a specialized measurement location, a single measurement location should be assigned to a top level machine segment or asset. Usage is mainly aimed at legacy OSA-EAI applications which used segment_events and asset_events without measurement locations.

The OSA-CBM best practice is to use enumerated types rather than ItemEvents wherever possible because enumerations tie together events of a particular type.

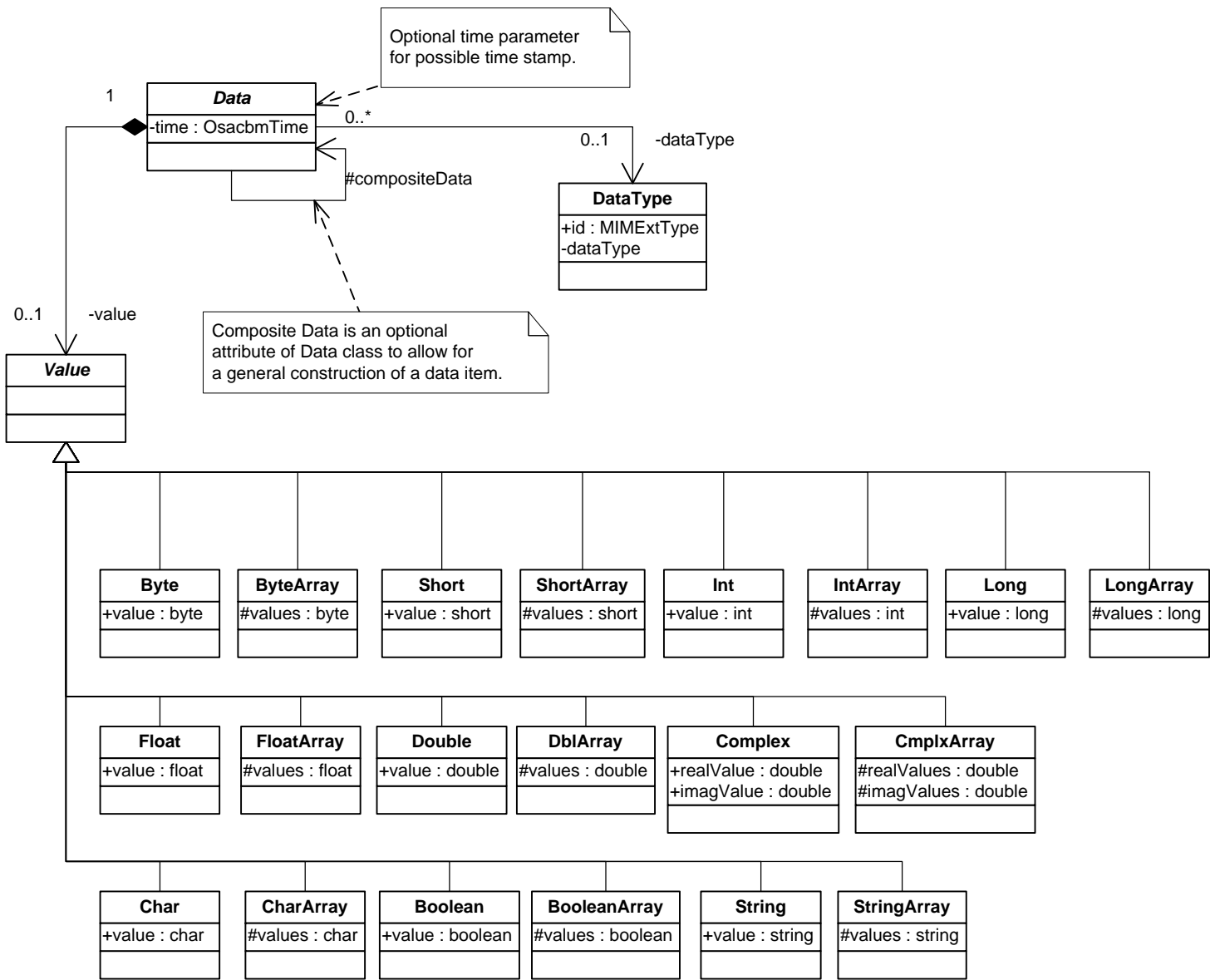
Data class for-user definable types. Read note below on usage.

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

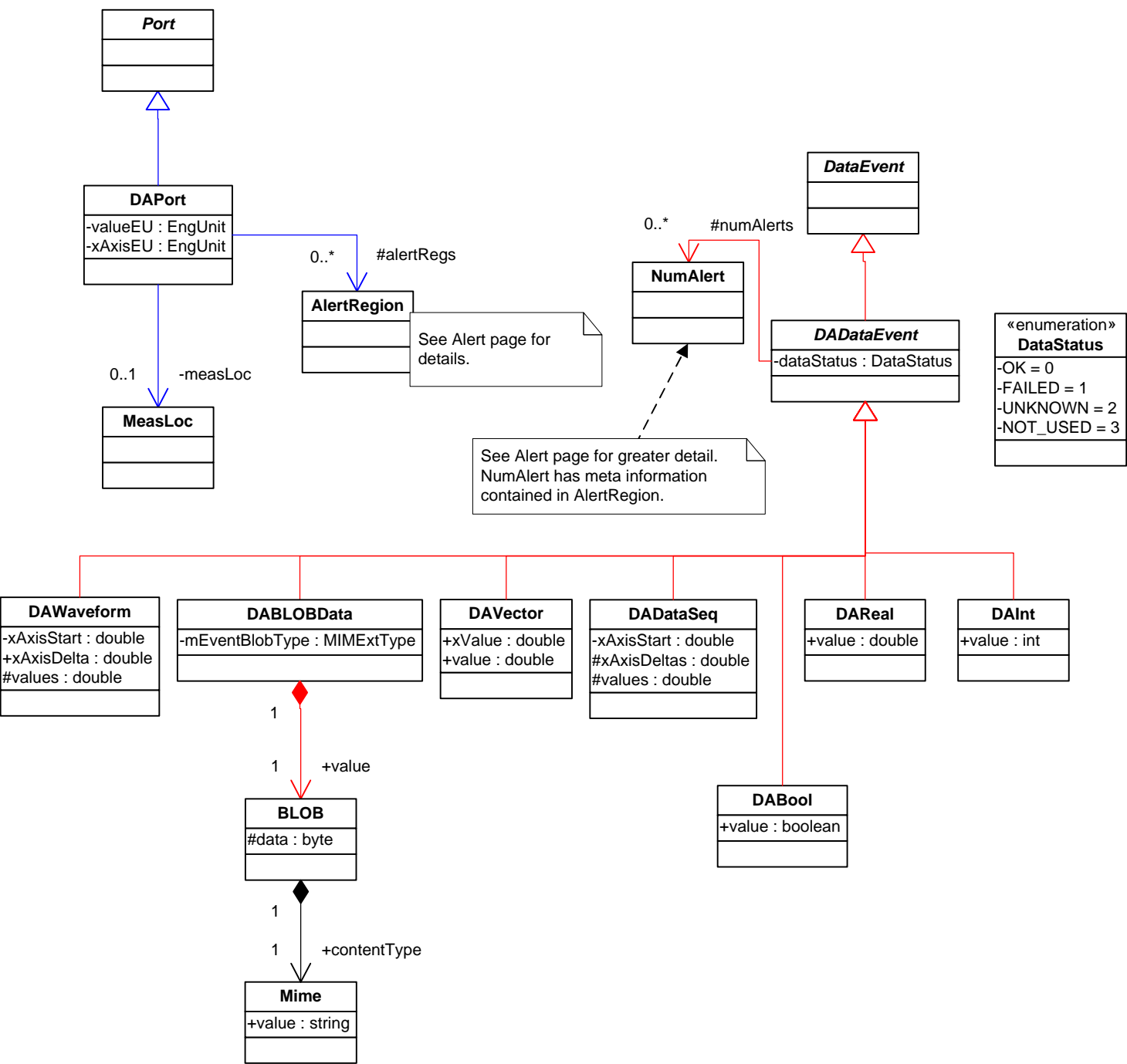
It is preferred that if there exists an information class that can contain your data in OSA-CBM then that should be used.

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

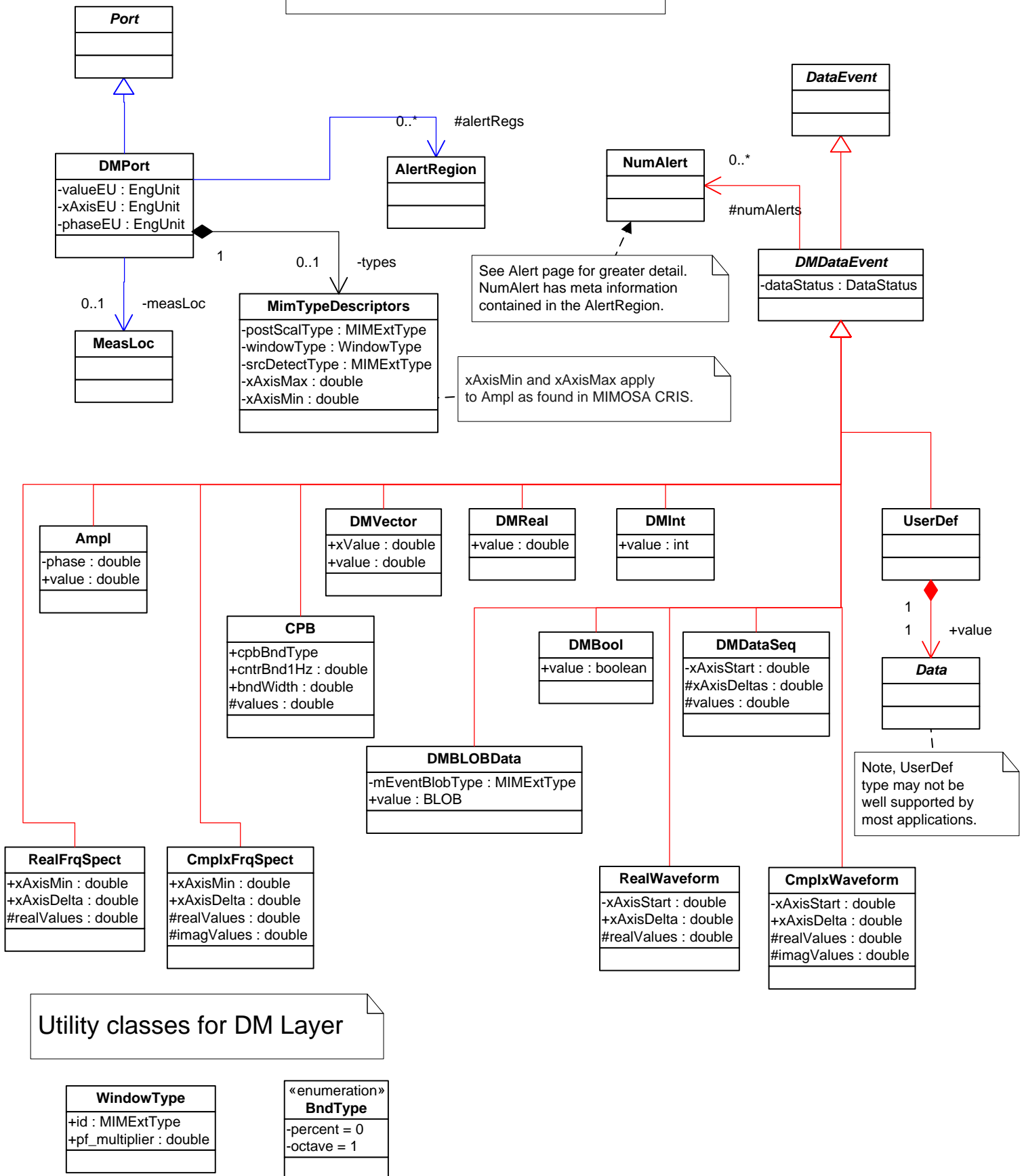
Additionally new types, such as multidimensional arrays, can be added as a new class to the OSA-CBM specification fairly quickly.



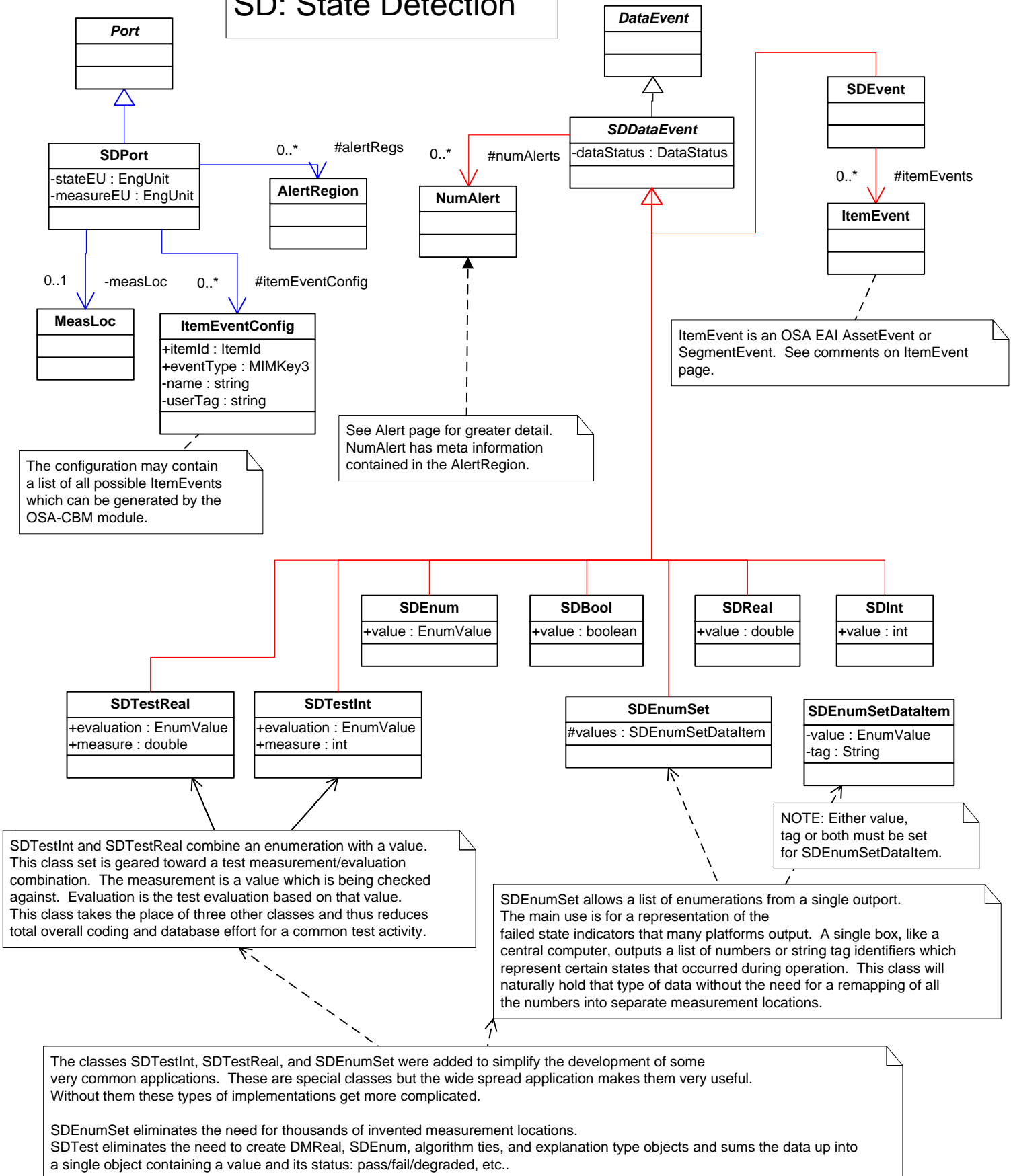
DA: Data Acquisition



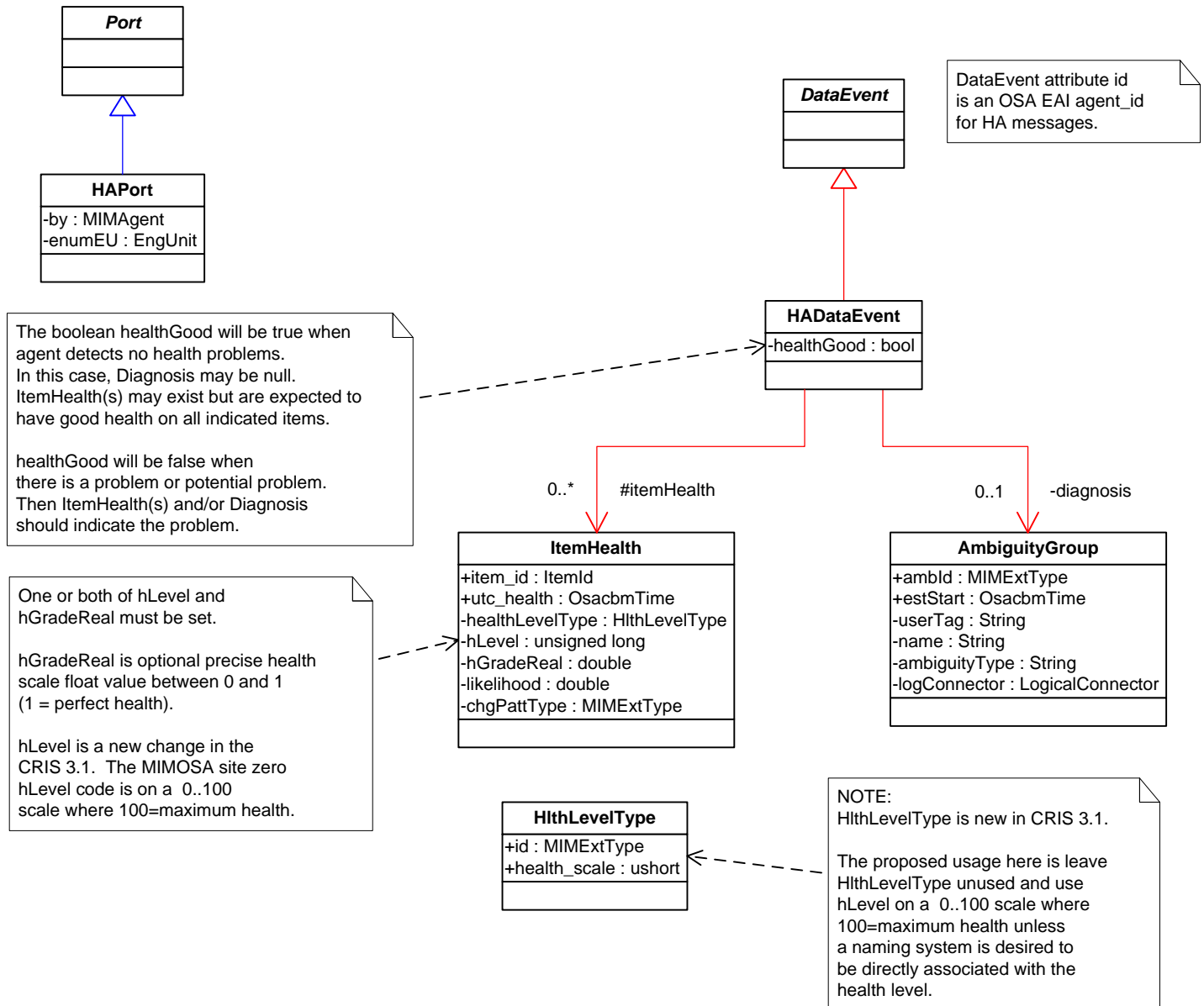
DM: Data Manipulation



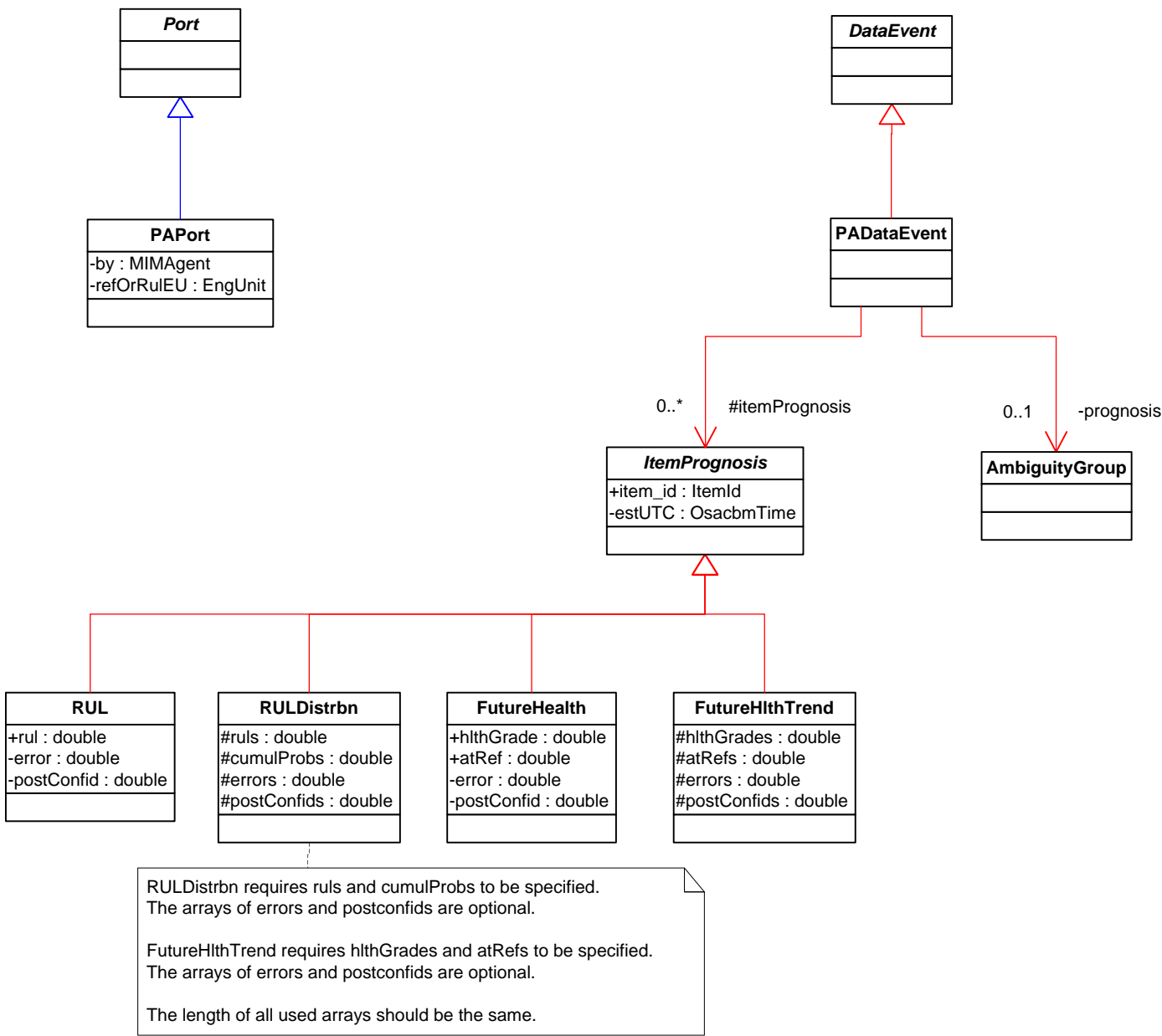
SD: State Detection



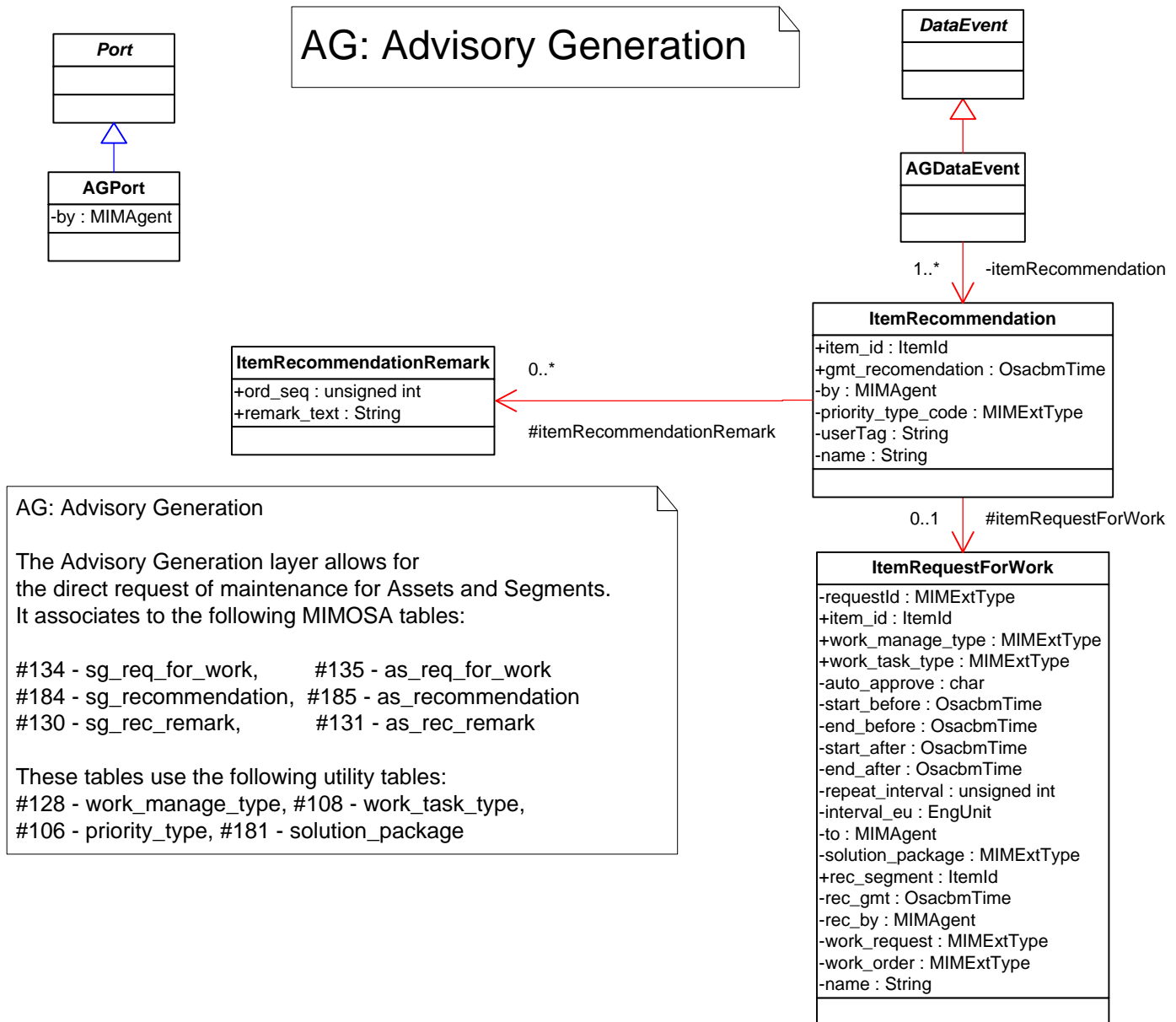
HA: Health Assessment



PA: Prognostics Assessment



Copyright 2008, Machinery Information Management Open Systems Alliance



Notes "byAgent", if not used, will be the (Site, DataEvent.id) agent.

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 receive 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, rec_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

Copyright 2008, Machinery Information Management Open Systems Alliance

Start of Note Pages

This page ends the UML specification
and starts the notes pages for OSA-CBM.

Notes

XML Extensibility 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.

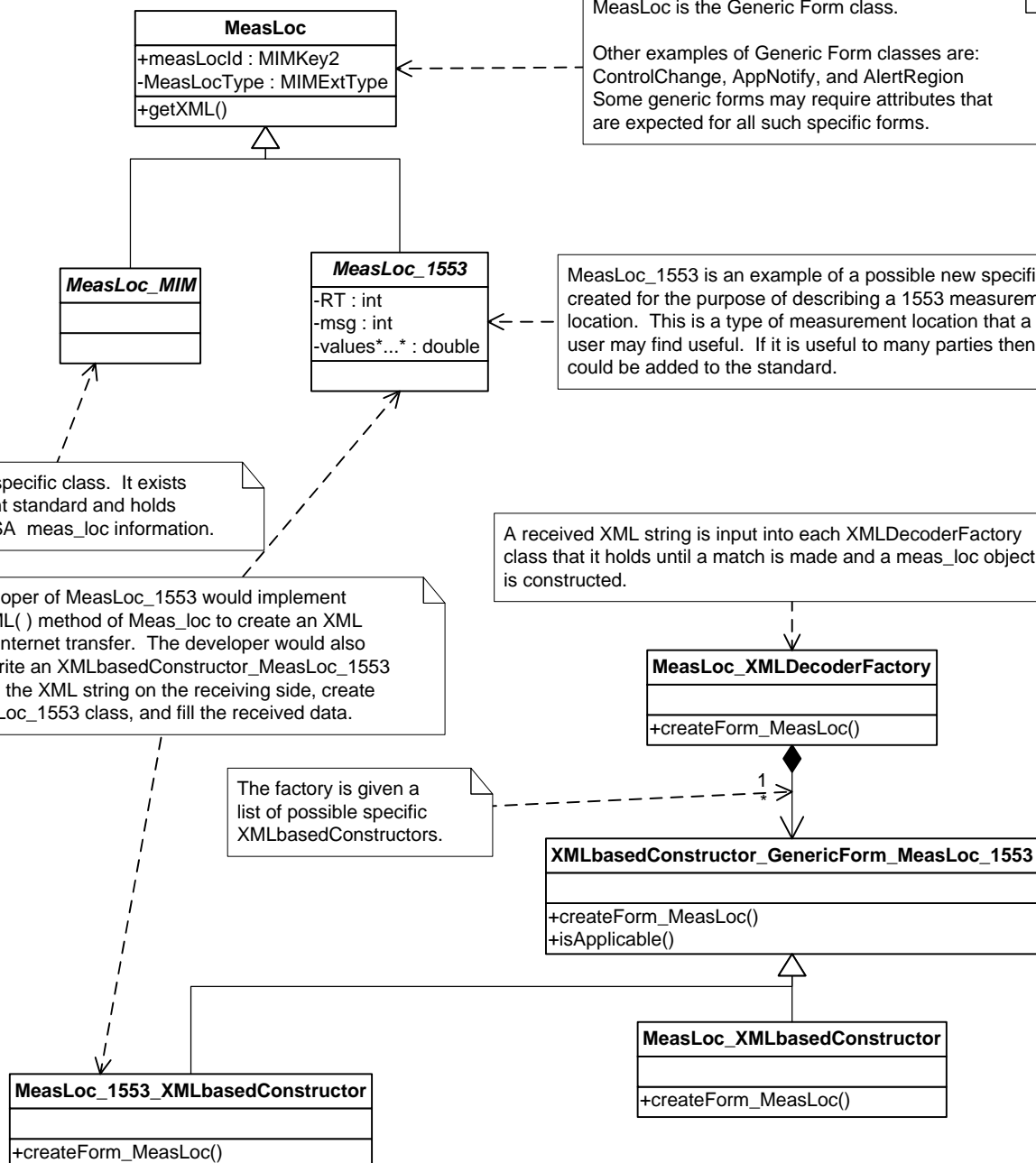
MeasLoc_1553 is an example of a possible new specific class created for the purpose of describing a 1553 measurement location. This is a type of measurement location that a particular user may find useful. If it is useful to many parties then it could be added to the standard.

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

The developer of MeasLoc_1553 would implement the getXML() method of MeasLoc 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 receiving side, create the MeasLoc_1553 class, and fill the received data.

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 factory is given a list of possible specific XMLbasedConstructors.



Mapping Methodologies

Mapping Methodologies

The goal is to have OSA-CBM map seamlessly into OSA-EAI.

The OSA-CBM to OSA-EAI mapping will have the following components:

1. A 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 map directly into OSA-EAI with ease.

There also will be some changes and additions in OSA-EAI to facilitate the mapping.

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 include:

- 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, health assessments, and proposed events are identified by agent or meas_location, time stamp,

and, in the HA and PA layers, item id. OSA-CBM 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 has been enhanced to accommodate ambiguity groups.

4) Explanation

Explanation is the ability to show a connection between data used as input and resultant data.

The OSA-CBM explanation classes use references within the OSA-CBM context. An OSA-CBM extensions Explanation table will be used by those desiring this information to be retrievable.

OSA-EAI has many data tie tables for those desiring this information in the OSA-EAI context.