OpenClovis Software Development Kit (SDK) Service Description and API Reference for Alarm Service

For OpenClovis SDK Release 2.3 V2.0

Document Revision Date: February 22, 2007

Contents

Chapter 1

Functional Overview

The ASP Alarm service enables applications to notify the north bound entity about erroneous conditions that can occur on managed resources. This service is compliant with the X.733 specification. It provides the functionality to model probable cause, severity, and category compliant to X.733 (ITU-T) specification. ASP Alarm service also provides filters that can be applied on the Alarm It provides a mechanism to specify soaking time, generation rule, and suppression rule. Alarm service also enables application to poll the managed resources and raise Alarms based on the state of the resource.

Chapter 2

Service Model

2.1 Usage Model

The usage model of the Alarm service is a producer-consumer model. Applications that raise the Alarm are the producers of the Alarms and the management applications are the consumers of these Alarms. Any application can raise an Alarm, on detecting an erroneous condition on a managed resource. The usage model is similar to a publisher-subscriber model because application components and the management applications are unaware of each other and each Management application receives the Alarms after subscribing to the Alarm's event channel.

2.2 Functional Description

2.2.1 Working with Alarm Service

2.2.1.1 Alarm Characteristics

The mandatory parameters of X.733 specification are incorporated as part of the attributes of an Alarm. The category, probable cause, and the perceived severity of the Alarm are the mandatory parameters of Alarm notification. There are five categories of Alarm and the probable cause parameter defines the probable cause of the Alarm. The severity parameter defines six severity levels: cleared, indeterminate, warning, minor, major, and critical. The following table illustrates the mapping between the category and the list of probable causes of an Alarm.

Alarm Category	Probable Cause
Communication	Loss of signal Loss of frame Framing error Local node transmission error Remote node transmission error Call establishment error Degraded signal Communications subsystem failure Communications protocol error LAN error DTE-DCE interface error
Quality of service	Response time excessive Queue size exceeded Bandwidth reduced Retransmission rate excessive Threshold crossed Performance degraded Congestion Resource at or nearing capacity
Processing Error	Storage capacity problem Version mismatch Corrupt data CPU cycles limit exceeded Software error Software program error Software program abnormally terminated File error Out of memory Underlying resource unavailable Application subsystem failure Configuration or customization error

Alarm Category	Probable Cause
Equipment	Power problem Timing problem Processor problem Dataset or modem error Multiplexer problem Receiver failure Transmitter failure Receive failure Transmit failure Output device error Input device error I/O device error Equipment malfunction Adapter error
Environmental	Temperature unacceptable Humidity unacceptable Heating/ventilation/cooling system problem Fire detected Flood detected Toxic leak detected Leak detected Pressure unacceptable Excessive vibration Material supply exhausted Pump failure Enclosure door open

2.2.1.2 Alarm MSO Class

In ASP, every managed resource that needs to raise an Alarm is modeled as an Alarm MSO class. The managed resource has to create MSO class. A maximum of 16 types of Alarms can be associated with a resource. MSO attributes are exported to management applications such as SNMP. The following table lists the Alarm MSO attributes that are visible to the management application, specifies the attribute type, and provides a brief description for each attribute.

Attribute Name	Attribute Type	Description
Probable Cause	Configuration attribute	This attribute identifies the Alarm which is modeled on a managed object. For example, Loss of Signal.
Category	Configuration attribute	This attribute identifies the category of the Alarm based on the probable cause. For example, Communication.
Severity	Configuration attribute	This attribute identifies the severity of the Alarm. For example, Critical.
Specific Problem	Runtime attribute	This parameter, when present, identifies further refinements to the Probable cause of the Alarm. Alarm service does not interpret this attribute but leaves it to the application.
Alarm Enable	Runtime attribute	This attribute identifies if the Alarm has been enabled for the managed object.
Alarm Active	Runtime attribute	This attribute indicates the current status of the Alarm. A value of 1 means that the Alarm is asserted and a value of 0 means that it is cleared.
Alarm Suspend	Runtime attribute	If the Alarm needs to be suspended, this attribute has to be set to 1.
Alarm Clear	Runtime attribute	This attribute indicates if the Alarm needs to be cleared from the north bound. By setting this attribute the Alarm gets cleared, if it is already raised.
Event Time	Runtime attribute	This attribute indicates the time when the Alarm was last raised. This attribute is reset to zero when the Alarm gets cleared.

2.2.1.3 Alarm State Definition

When the Alarm is raised till it is reported, it goes through the following stages:

- Alarm Raised State: The Alarm is in the raised state when the application, after detecting the erroneous condition, raises the Alarm.
- Alarm Soaked State: After the Alarm is soaked for a certain period of time, it enters the Alarm soaked state.
- Alarm Generated State: After the Alarm qualifies the generation rule it enters the Alarm generated state. If the Alarm does not qualify the generation rule because of some dependent Alarm, it remains in the soaked state till the dependent Alarms are generated.
- Alarm Masked State: If the Alarm is being masked because of the masking logic, it enters into the Alarm masked state.
- Alarm Reported State: If the Alarm is not masked, it enters the Alarm reported state.

2.2.1.4 Raising an Alarm on a managed resource

Managed resources are modeled as Managed Objects. The Alarms on the managed resources correspond to the probable cause attribute of the managed object. An application can raise an Alarm on a managed resource using the ASP Alarm service when it detects a deviation from the normal operation.

Alarm service also allows the application to pass additional context information when raising the Alarm. After the Alarm is successfully reported, Alarm Manager returns a unique Alarm handle for the raised Alarm.

For example:

An application managing a GigePort resource must follow these steps while raising an Alarm:

1. The application can use the Alarm service by creating an Alarm MSO class for the GigePort resource.
2. The likely Alarms that can occur on GigePort resource such as Loss of Signal (LOS) and Loss of Frame(LOF) are associated with the GigePort class while modeling.
3. The application managing the resource, on detecting an erroneous condition on the specific instance of the GigePort, raises an Alarm by specifying the MOID and the probable cause.

2.2.2 Alarm Filters

The filtering mechanism provided by the Alarm service prevents spurious Alarms from flooding the Alarm Manager. It also applies constraints that an Alarm must satisfy before it is reported as an Alarm. The filtering mechanism, can be classified as soaking, masking, generation, and suppression rule.

2.2.2.1 Soaking

Soaking is the time duration for which the erroneous condition must persist before it is reported as an Alarm. Soaking allows an Alarm to be monitored for a specific time period.

The following figures contain the notations "A" and "B" that implies the detection and clearing of the erroneous condition, respectively.

Figure 2.1 illustrates an Alarm raised for a specified assert soaking time. In case 1, the Alarm is raised and cleared within the assert soaking time. Hence, the Alarm is not processed. In case 2, the Alarm stays raised, qualifies the assert soaking time, and is processed by the Alarm Manager.

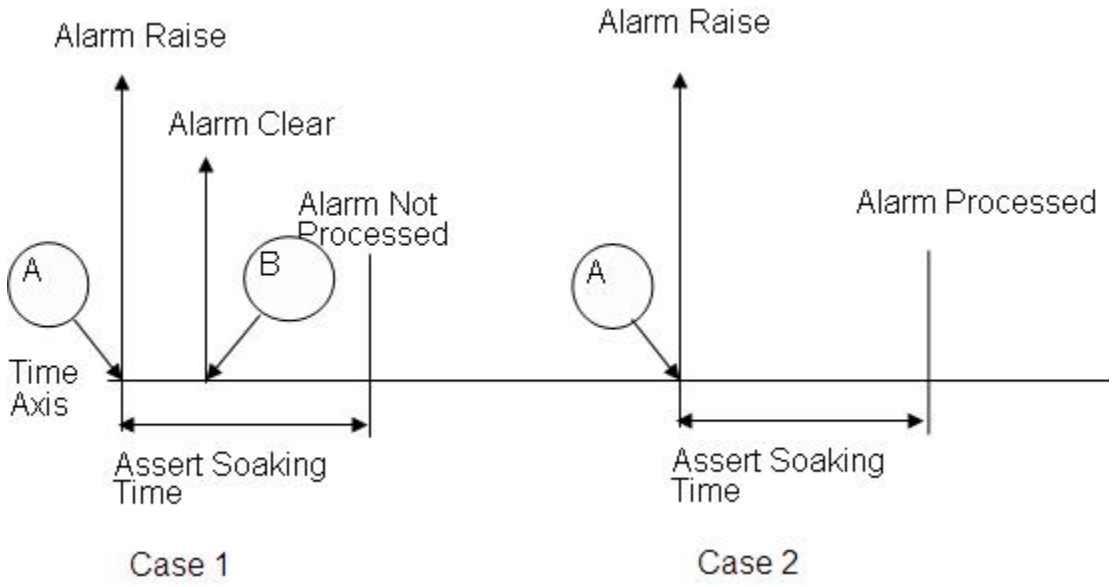


Figure 2.1: Assert Soaking Time

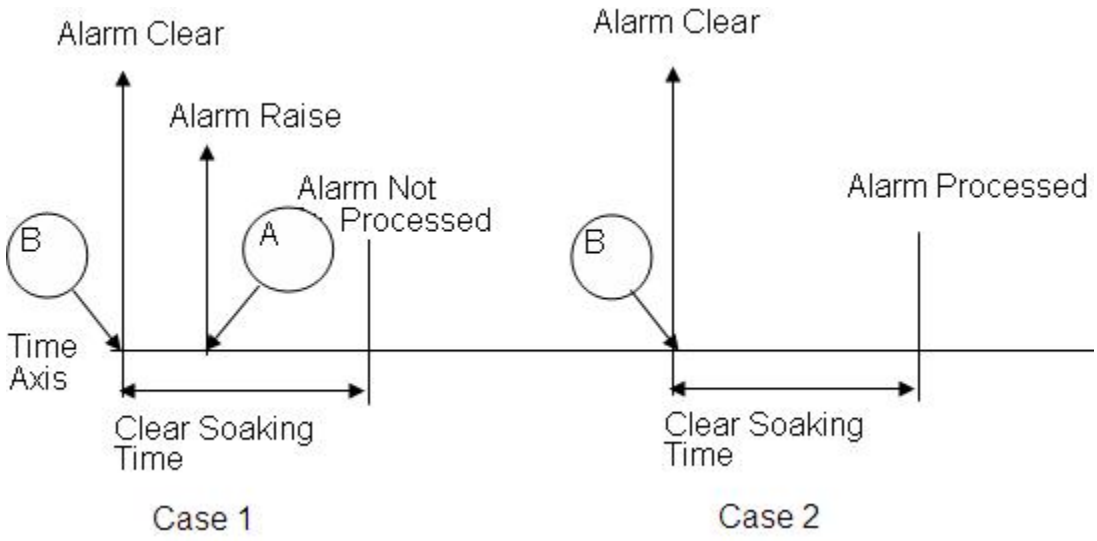


Figure 2.2: Clear Soaking Time

Figure 2.2 illustrates the clearing of a raised Alarm for a specified clear soaking time. In case1, the Alarm is cleared and raised within the clear soaking time and so, the Alarm is not processed. In case2, the Alarm stays cleared, qualifies the clear soaking time. Hence it is processed by the Alarm Manager.

Soaking time is configured as per the Alarm. Two different soak periods can be configured, one for assert and the other for clear.

2.2.2.2 Generation and Suppression Rules

An application can specify a generation rule to model dependencies between Alarms. A generation rule specifies the following:

- The Alarm that needs to be generated.
- Set of dependent Alarms.
- The condition that has to be satisfied for the Alarm to be generated. This condition is specified as a logical relationship between the dependent Alarms.

  For example:

  Generation rule of A1.

  A1 = A2 OR A3 OR A4

  implies that Alarm A1 is in generated state, when either A2, A3 or A4 Alarms are generated.

A suppression rule specifies the following:

- The Alarm that needs to be suppressed.
- Set of dependent Alarms.
- The condition that needs to be met for the Alarm to be generated.

  This condition is specified as a logical relationship between the dependent Alarms.

  For example:

  Suppression rule of A1.

  A1 = A5 AND A6

  implies that Alarm A1 is in the suppressed state, when both A5 and A6 are suppressed. The Alarm A1 is generated when it qualifies the generation rule and when A5 or A6 is in cleared state.

When referring to either generation rule or suppression rule, Alarm rule is used in the general context. Generation rule provides a mechanism to specify constraints on generation of Alarms. Suppression rule specifies constraints on the suppression of Alarms. There can be a maximum of four Alarms in an Alarm rule. An Alarm cannot be generated if it is not specified in the Alarm rule. An Alarm rule can either have logical OR or logical AND operators but not both.

2.2.2.3 Alarm Masking

The Alarm service views the containment relationship of the Alarm MSO to implement hierarchical masking. The Alarm masking algorithm masks all Alarms of the same category within a subtree in the hierarchy. The Alarm masking logic is explained in the following example:

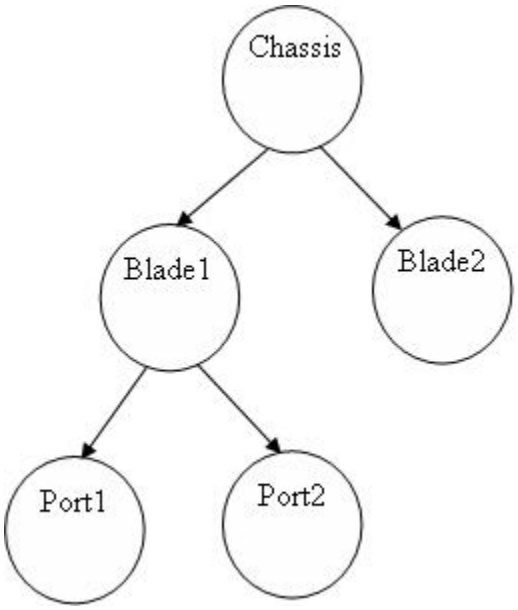


Figure 2.3: COR Class Tree

An Alarm of probable cause Loss of Signal that belongs to Communication category is reported on Blade1. Alarm service then masks all Alarms of Communication category on its children: Port1 and Port2 as Loss of Signal falls under the Communication category. After the Loss of Signal Alarm is cleared on Blade1, the masked Alarms that satisfy the hierarchical masking rule are published as Alarm notifications and the status is updated in COR.

2.2.3 Polling

Alarm service provides an infrastructure that allows to specify the polling function for each MSO class. This polling function is called periodically by the Alarm service. Using the polling function, applications can probe the state of the managed resource to detect erroneous condition. As each erroneous condition is modeled as a probable cause (in a MO class), the application needs to return the status of the probable cause in the polling function. The Alarm service raises the appropriate Alarm based on the value returned by the polling function.

The polling interval is configured for each MSO class. The Alarm client can poll the current status of probable causes that have been configured as polled.

2.2.4 Event Generation

An event is published for each reported Alarm. Interested services can obtain these events by subscribing to the Alarm event channel. The payload of the event consists of the Alarm information and an Alarm handle. The Alarm information comprises the probable cause, category, severity, specific problem, resource on which the Alarm was reported, Alarm State - raised or cleared, time stamp, and additional information of the Alarm. The Alarm handle is unique for every node and is also known as the Notification Identifier. The mapping of the Alarm handle to the reported Alarm is maintained in the Alarm Manager.

2.2.5 Alarm Life Cycle

2.2.5.1 State Machine

Figure 2.4 illustrates the different stages of an Alarm, when the Alarm is raised till it is reported.

1. The Alarm enters the *raised* state when an application detects an erroneous condition and raises the Alarm. A *raised* Alarm can have an assert or clear status.
2. After the Alarm is soaked it enters the *soaked* state.
3. When the Alarm qualifies the generation rule it enters the *generated* state. This leads to generation of other Alarms dependent on this Alarm. If the Alarm does not qualify the generation rule, the Alarm stays in the soaked state.
4. The Alarm is reported if the status of the Alarm is *clear* after it successfuly qualifies the generation rule,
5. If the status of the Alarm is *assert* after the *generated* state, the Alarm is checked against the hierarchical masking rule. If the parent MO has an Alarm of the same category, the Alarm enters the *masked* state. Otherwise, the Alarm enters the *reported* state.
6. Alarms in the masked state move into *reported* state after the Alarm on the parent MO of the same category is cleared.

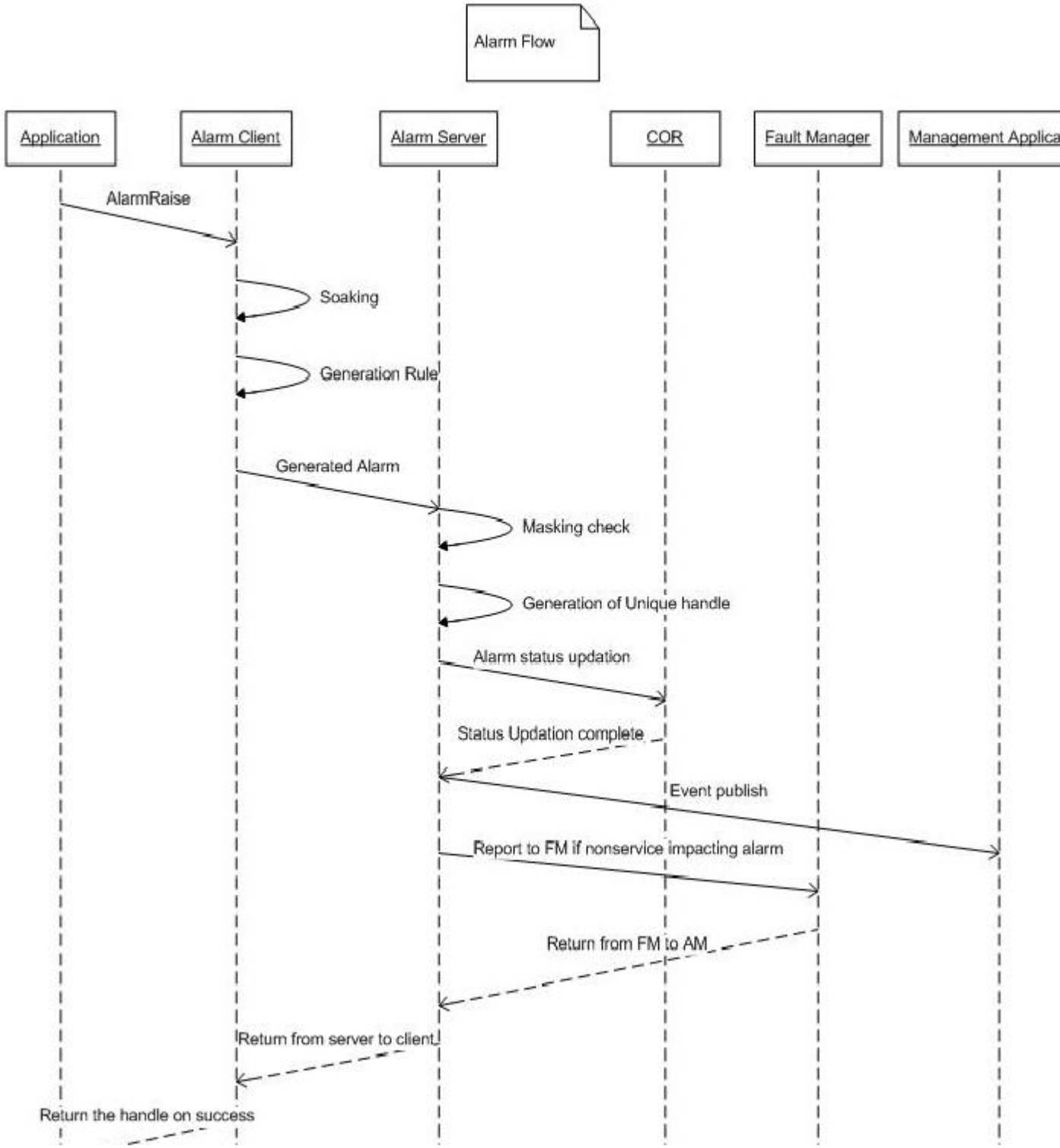


Figure 2.4: Alarm State Machine

2.2.5.2 Alarm Flow

Figure 2.5 illustrates the Alarm flow from the detection of the erroneous condition to reporting of the Alarm.

1. The application detects the erroneous condition and raises the Alarm.
2. Alarm client soaks the Alarm and on persistence of the erroneous condition it moves the Alarm from raised to soaked state.
3. When the Alarm qualifies the generation rule, the Alarm client moves the Alarm from the soaked state to the generated state.
4. After the generation of the Alarm, it is passed to the Alarm server. If the Alarm is not being masked or if the status of the Alarm is clear then the Alarm server performs the following:
   (a) Generates an unique Alarm handle.
   (b) Updates the Alarm status in COR.
   (c) Publishes an event as described in the *Event Generation* section.
   (d) Reports the Alarm to the Fault Manager if it is a non service impacting Alarm. Non service impacting Alarms are described in the next section.

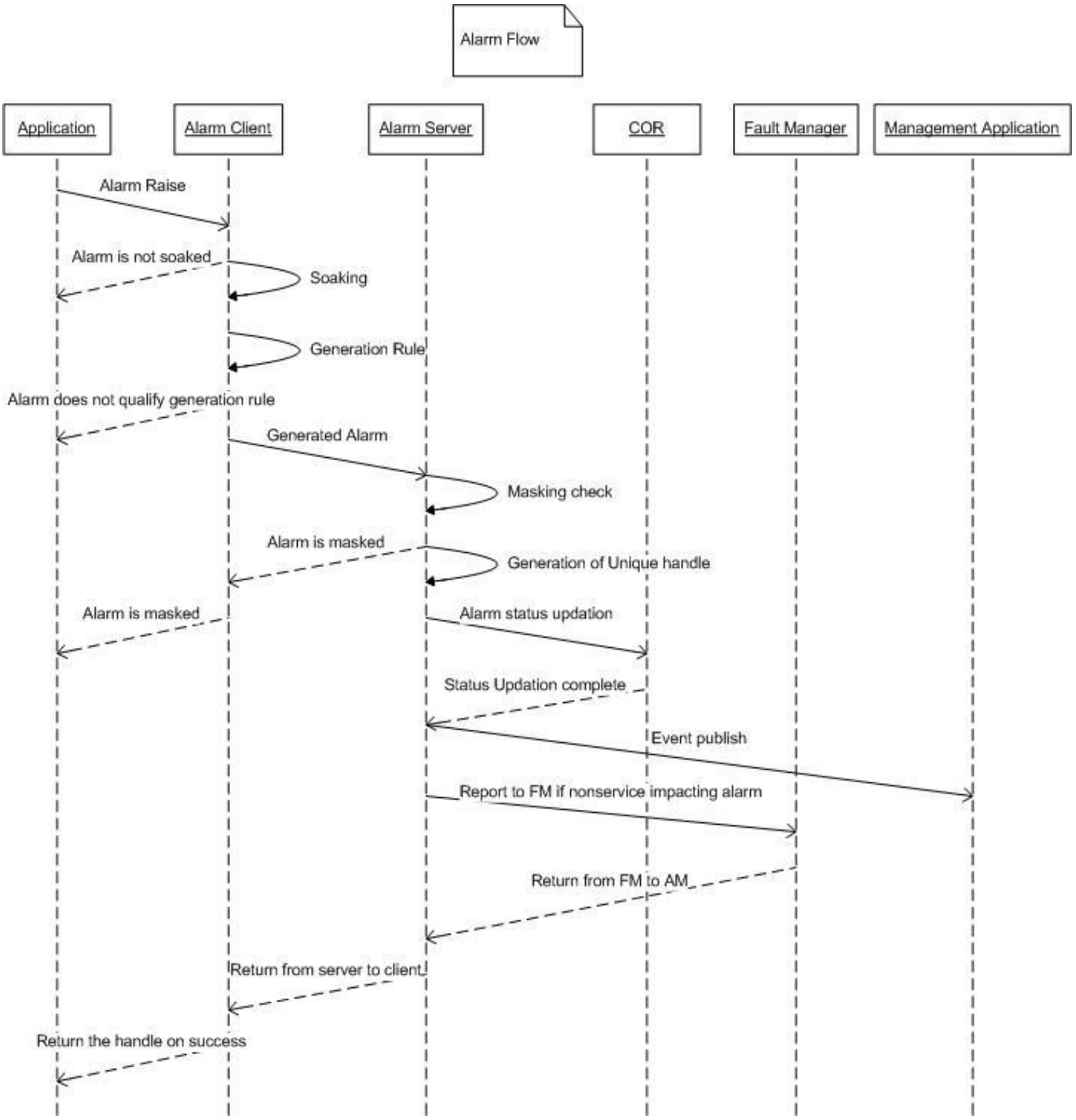


Figure 2.5: Alarm Flow Sequence

2.2.6 Modules in Alarm Detection and Reporting

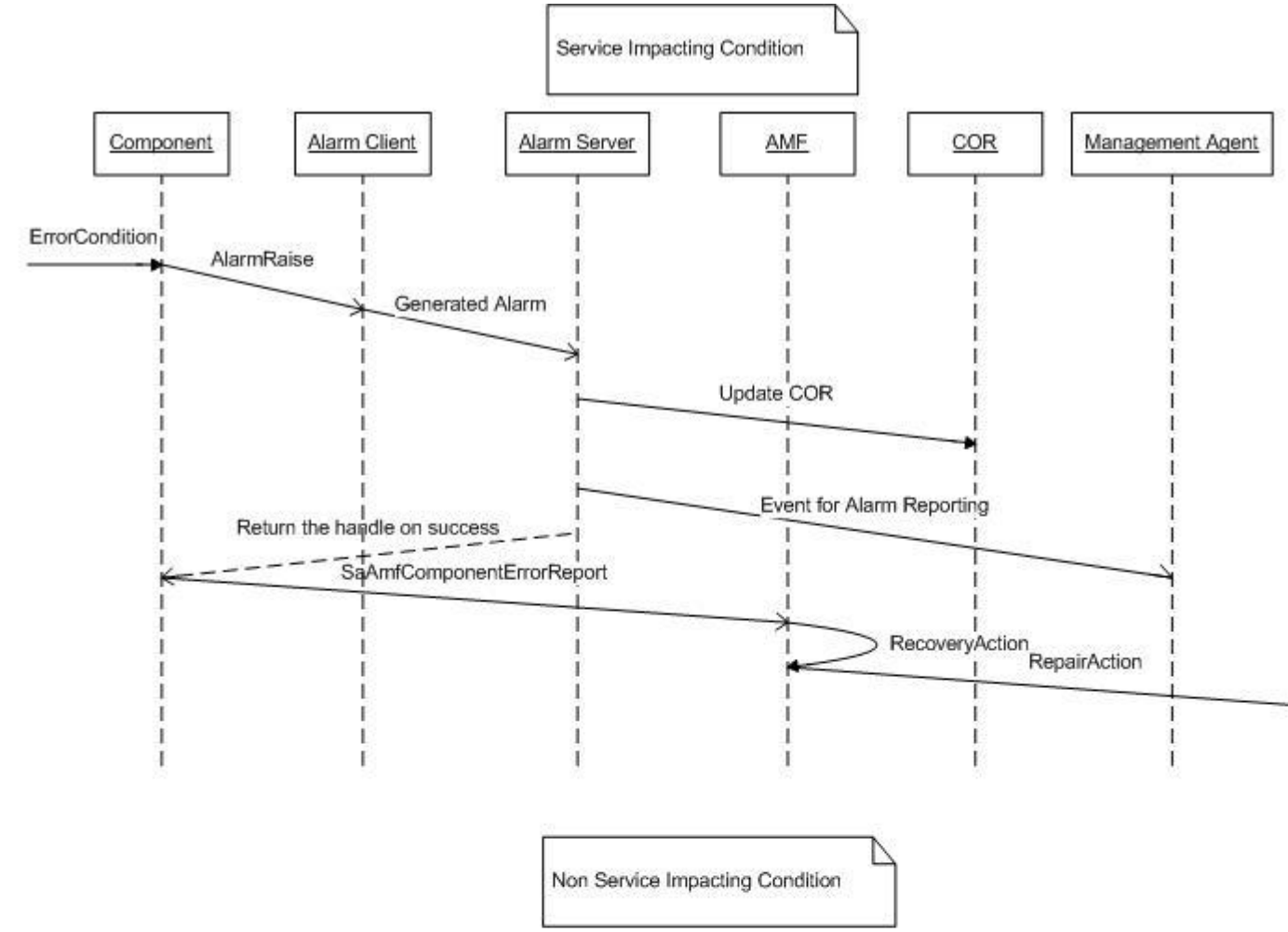
2.2.6.1 Component Level Interaction of Alarm Manager

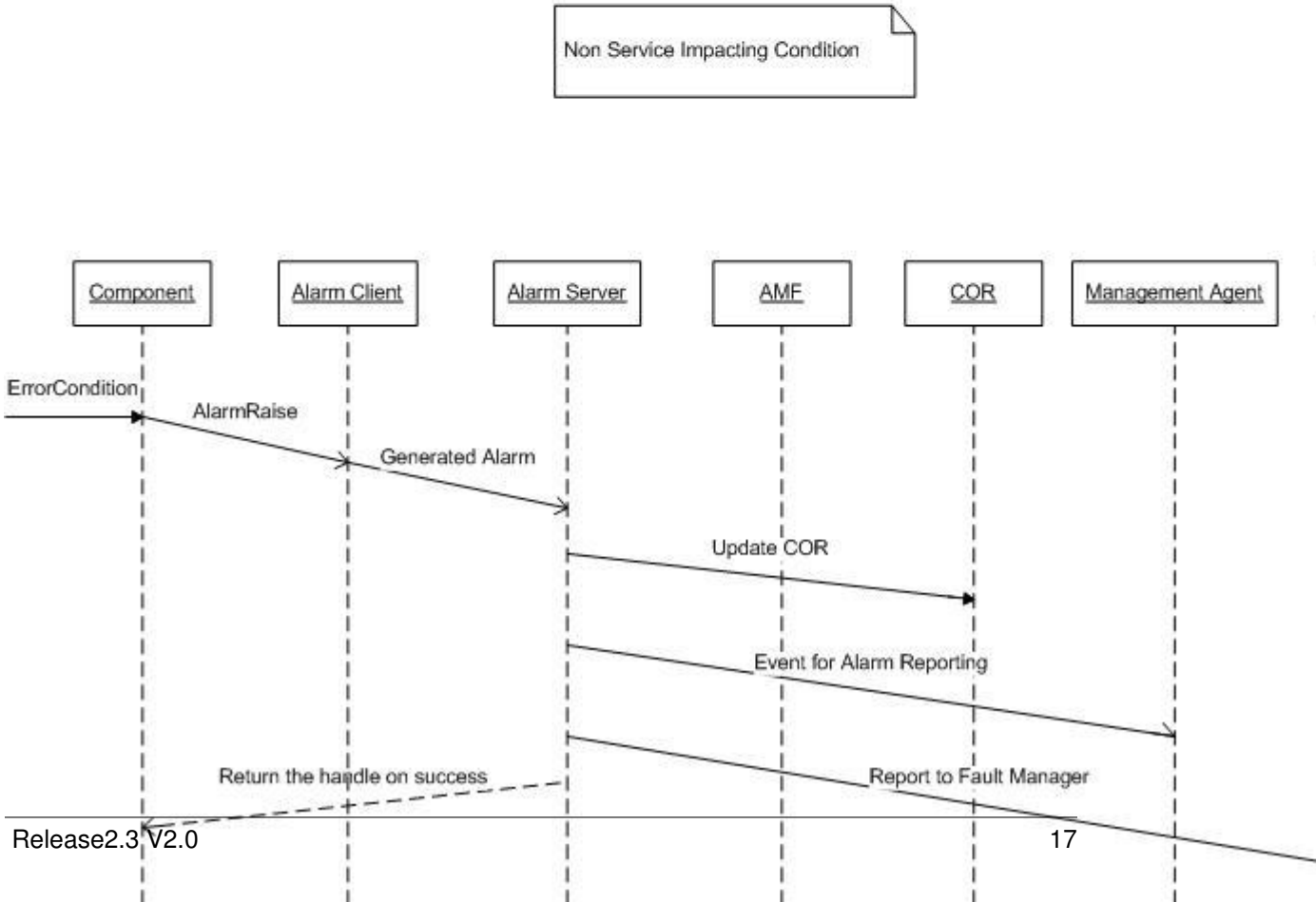
The application component detects the erroneous condition and raises the Alarm. The Alarm Manager updates the status in COR only after it qualifies/satisfies the generation rule. It publishes an event for this notification after the update is performed in COR. These notifications can result in a trap generated by SNMP. It reports the non service impacting Alarms to the Fault Manager.

Figure 2.6 shows the component level interaction of Alarm Manager:

1. Application component polls the hardware for any hardware Alarms. The application component can use the Alarm service by linking with the Alarm client library.
2. The application component raises an Alarm using the API provided by the Alarm client.
3. Alarm client soaks the Alarm and after it qualifies the generation rule, sends it to Alarm Server.
4. Alarm Server performs a check if the Alarm is masked. If it is not masked, it updates the current Alarm status and time stamp in COR. The Alarm status indicates if it is asserted or cleared.
5. Alarm Server also publishes a notification for this Alarm.
6. Alarm Server reports to the Fault Manager, if it is a non service impacting Alarm. The next section provides more information about non service impacting Alarms.

2.2 Functional Description





2.2.6.2 Service Impacting and Non Service Impacting Alarms

Figure 2.7 explains how service impacting and non service impacting Alarms are handled. It describes the interaction between the Alarm Manager(AM), Fault Manager(FM), Clovis Object Repository(COR), Availability Management Framework (AMF), application component that raises an Alarm, and the management application.

1. The application component uses the Alarm service by linking with the Alarm client.
2. When an erroneous condition is detected, it raises an Alarm by calling the Alarm raise API provided by the Alarm client.
3. After the Alarm satisfies the soaking time and the generation rule, it is sent to the Alarm server.
4. Alarm server checks if the Alarm qualifies the hierarchical masking rule, updates COR, and publishes an event.
5. Alarm Server returns a unique handle called the notification identifier to the application component that raised the Alarm. The mapping of this handle to the reported Alarm is maintained in the Alarm server side. The Alarm handle is unique for every node.
   (a) If the Alarm is a service impacting Alarm, Alarm server does not report the Alarm to the Fault Manager.
   (b) The application component calls the `saAmfComponentErrorReport` API for service recovery and the Alarm handle is passed to AMF. After recovering the service, AMF calls the Fault Manager to repair the managed resource.
6. If the Alarm is a non service impacting Alarm, Alarm server reports the Alarm to the Fault Manager.

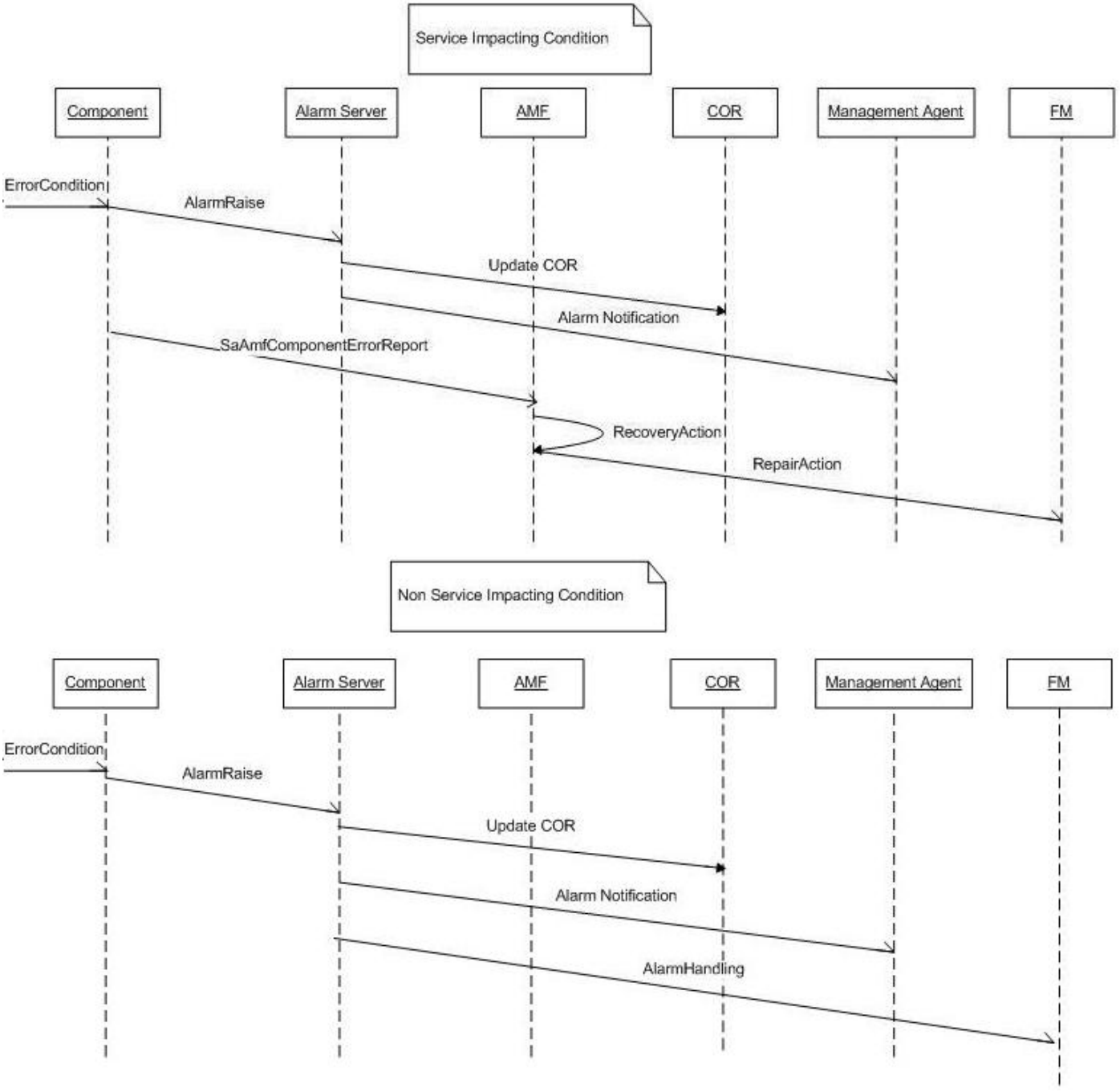


Figure 2.7: Service Impacting and Non Service Impacting Alarms

Chapter 3

Service APIs

3.1 Type Definitions

3.1.1 ClAlarmInfoT

typedef struct {
ClAlarmProbableCauseT probCause;
ClNameT compName;
ClCorMOIdT moId;
ClAlarmStateT AlarmState;
ClAlarmCategoryTypeT category;
ClAlarmSpecificProblemT specificProblem;
ClAlarmSeverityTypeT severity;
ClTimeT eventTime;
ClUint32T len;
ClUint8T buff [1];
} ClAlarmInfoT;

ClAlarmInfoT contains the list of Alarm attributes that include probable cause, MOID, category, severity, and event time along with additional information for the Alarm. It also indicates if the Alarm is asserted or cleared. The attributes of this structure have the following interpretation:

- *probCause* - This is the probable cause of the Alarm being raised.
- *compName* - This is the name of the component that is raising the Alarm.
- *moId* - This is the resource on which the Alarm is being raised.
- *AlarmState* - This attribute indicates if the Alarm is for assert or for clear. A value of `1` indicates assert and a value of `0` indicates clear.
- *category* - This is the category of the Alarm being raised and is closely associated with the probable cause. The probable cause dictates which category the Alarm belongs to. For example, if the probable cause is *Loss of Signal*, the category of the Alarm is *Communication*. This is an in/out parameter, indicating that if there is a mismatch in probable cause and category, the valid category matching the probable cause is returned.
- *specificProblem* - The type of an identifier to the specific problem of the Alarm. This information is not configured but is assigned a value at run-time. This value segregates Alarms having the same category and probable cause but different in their manifestation.

- *severity* - This indicates the severity of the Alarm that is being raised. This is an in/out parameter, which means if the value does not match the value provided while modeling, the severity configured during modeling is returned.
- *eventTime* - This is the time stamp of the Alarm. This value indicates the time when the Alarm was detected and not the time it was reported.
- *len* - This is the length of the additional information contained in `buff`.
- *buff* - Additional information can be sent using this parameter.

3.1.2 ClAlarmEventName

#define ClAlarmEventName "CL_Alarm_EVENT_CHANNEL"

The type of the handle used for identifying a raised Alarm.

3.1.3 ClAlarmHandleT

typedef ClUint32T ClAlarmHandleT;

Name of the Alarm event channel. This is the channel on which the subscriber waits for notifications.

3.1.4 CL_Alarm_EVENT

#define CL_Alarm_EVENT 1

The type of the event.

3.1.5 ClAlarmProbableCauseT

typedef enum {
CL_Alarm_PROB_CAUSE_LOSS_OF_SIGNAL,
CL_Alarm_PROB_CAUSE_LOSS_OF_FRAME,
CL_Alarm_PROB_CAUSE_FRAMING_ERROR,
CL_Alarm_PROB_CAUSE_LOCAL_NODE_TRANSMISSION_ERROR,
CL_Alarm_PROB_CAUSE_REMOTE_NODE_TRANSMISSION_ERROR,
CL_Alarm_PROB_CAUSE_CALL_ESTABLISHMENT_ERROR,
CL_Alarm_PROB_CAUSE_DEGRADED_SIGNAL,
CL_Alarm_PROB_CAUSE_COMMUNICATIONS_SUBSYSTEM_FAILURE,
CL_Alarm_PROB_CAUSE_COMMUNICATIONS_PROTOCOL_ERROR,
CL_Alarm_PROB_CAUSE_LAN_ERROR,
CL_Alarm_PROB_CAUSE_DTE,
CL_Alarm_PROB_CAUSE_RESPONSE_TIME_EXCESSIVE,
CL_Alarm_PROB_CAUSE_QUEUE_SIZE_EXCEEDED,
CL_Alarm_PROB_CAUSE_BANDWIDTH_REDUCED,
CL_Alarm_PROB_CAUSE_RETRANSMISSION_RATE_EXCESSIVE,
CL_Alarm_PROB_CAUSE_THRESHOLD_CROSSED,
CL_Alarm_PROB_CAUSE_PERFORMANCE_DEGRADED,

```
        CL_Alarm_PROB_CAUSE_CONGESTION,
        CL_Alarm_PROB_CAUSE_RESOURCE_AT_OR_NEARING_CAPACITY,
        CL_Alarm_PROB_CAUSE_STORAGE_CAPACITY_PROBLEM,
        CL_Alarm_PROB_CAUSE_VERSION_MISMATCH,
        CL_Alarm_PROB_CAUSE_CORRUPT_DATA,
        CL_Alarm_PROB_CAUSE_CPU_CYCLES_LIMIT_EXCEEDED,
        CL_Alarm_PROB_CAUSE_SOFWARE_ERROR,
        CL_Alarm_PROB_CAUSE_SOFTWARE_PROGRAM_ERROR,
        CL_Alarm_PROB_CAUSE_SOFWARE_PROGRAM_ABNORMALLY_TERMINATED,
        CL_Alarm_PROB_CAUSE_FILE_ERROR,
        CL_Alarm_PROB_CAUSE_OUT_OF_MEMORY,
        CL_Alarm_PROB_CAUSE_UNDERLYING_RESOURCE_UNAVAILABLE,
        CL_Alarm_PROB_CAUSE_APPLICATION_SUBSYSTEM_FAILURE,
        CL_Alarm_PROB_CAUSE_CONFIGURATION_OR_CUSTOMIZATION_ERROR,
        CL_Alarm_PROB_CAUSE_POWER_PROBLEM,
        CL_Alarm_PROB_CAUSE_TIMING_PROBLEM,
        CL_Alarm_PROB_CAUSE_PROCESSOR_PROBLEM,
        CL_Alarm_PROB_CAUSE_DATASET_OR_MODEM_ERROR,
        CL_Alarm_PROB_CAUSE_MULTIPLEXER_PROBLEM,
        CL_Alarm_PROB_CAUSE_RECEIVER_FAILURE,
        CL_Alarm_PROB_CAUSE_TRANSMITTER_FAILURE,
        CL_Alarm_PROB_CAUSE_RECEIVE_FAILURE,
        CL_Alarm_PROB_CAUSE_TRANSMIT_FAILURE,
        CL_Alarm_PROB_CAUSE_OUTPUT_DEVICE_ERROR,
        CL_Alarm_PROB_CAUSE_INPUT_DEVICE_ERROR,
        CL_Alarm_PROB_CAUSE_INPUT_OUTPUT_DEVICE_ERROR,
        CL_Alarm_PROB_CAUSE_EQUIPMENT_MALFUNCTION,
        CL_Alarm_PROB_CAUSE_ADAPTER_ERROR,
        CL_Alarm_PROB_CAUSE_TEMPERATURE_UNACCEPTABLE,
        CL_Alarm_PROB_CAUSE_HUMIDITY_UNACCEPTABLE,
        CL_Alarm_PROB_CAUSE_HEATING_OR_VENTILATION_OR_COOLING_SYSTEM_PROBLEM,
        CL_Alarm_PROB_CAUSE_FIRE_DETECTED,
        CL_Alarm_PROB_CAUSE_FLOOD_DETECTED,
        CL_Alarm_PROB_CAUSE_TOXIC_LEAK_DETECTED,
        CL_Alarm_PROB_CAUSE_LEAK_DETECTED,
        CL_Alarm_PROB_CAUSE_PRESSURE_UNACCEPTABLE,
        CL_Alarm_PROB_CAUSE_EXCESSIVE_VIBRATION,
        CL_Alarm_PROB_CAUSE_MATERIAL_SUPPLY_EXHAUSTED,
        CL_Alarm_PROB_CAUSE_PUMP_FAILURE,
        CL_Alarm_PROB_CAUSE_ENCLOSURE_DOOR_OPEN
} ClAlarmProbableCauseT;
```

The values of the *ClAlarmProbableCauseT* enumeration type indicate the possible causes of Alarms. They have the following interpretation:

Following are the probable causes for Communication related Alarms:

- *CL_Alarm_PROB_CAUSE_LOSS_OF_SIGNAL*
- *CL_Alarm_PROB_CAUSE_LOSS_OF_FRAME*
- *CL_Alarm_PROB_CAUSE_FRAMING_ERROR*
- *CL_Alarm_PROB_CAUSE_LOCAL_NODE_TRANSMISSION_ERROR*
- *CL_Alarm_PROB_CAUSE_REMOTE_NODE_TRANSMISSION_ERROR*

- *CL_Alarm_PROB_CAUSE_CALL_ESTABLISHMENT_ERROR*
- *CL_Alarm_PROB_CAUSE_DEGRADED_SIGNAL*
- *CL_Alarm_PROB_CAUSE_COMMUNICATIONS_SUBSYSTEM_FAILURE*
- *CL_Alarm_PROB_CAUSE_COMMUNICATIONS_PROTOCOL_ERROR*
- *CL_Alarm_PROB_CAUSE_LAN_ERROR*
- *CL_Alarm_PROB_CAUSE_DTE*

Following are the probable causes for Quality of Service related Alarms:

- *CL_Alarm_PROB_CAUSE_RESPONSE_TIME_EXCESSIVE*
- *CL_Alarm_PROB_CAUSE_QUEUE_SIZE_EXCEEDED*
- *CL_Alarm_PROB_CAUSE_BANDWIDTH_REDUCED*
- *CL_Alarm_PROB_CAUSE_RETRANSMISSION_RATE_EXCESSIVE*
- *CL_Alarm_PROB_CAUSE_THRESHOLD_CROSSED*
- *CL_Alarm_PROB_CAUSE_PERFORMANCE_DEGRADED*
- *CL_Alarm_PROB_CAUSE_CONGESTION*
- *CL_Alarm_PROB_CAUSE_RESOURCE_AT_OR_NEARING_CAPACITY*

Following are the probable causes for Equipment related Alarms:

- *CL_Alarm_PROB_CAUSE_POWER_PROBLEM*
- *CL_Alarm_PROB_CAUSE_TIMING_PROBLEM*
- *CL_Alarm_PROB_CAUSE_PROCESSOR_PROBLEM*
- *CL_Alarm_PROB_CAUSE_DATASET_OR_MODEM_ERROR*
- *CL_Alarm_PROB_CAUSE_MULTIPLEXER_PROBLEM*
- *CL_Alarm_PROB_CAUSE_RECEIVER_FAILURE*
- *CL_Alarm_PROB_CAUSE_TRANSMITTER_FAILURE*
- *CL_Alarm_PROB_CAUSE_RECEIVE_FAILURE*
- *CL_Alarm_PROB_CAUSE_TRANSMIT_FAILURE*
- *CL_Alarm_PROB_CAUSE_OUTPUT_DEVICE_ERROR*
- *CL_Alarm_PROB_CAUSE_INPUT_DEVICE_ERROR*
- *CL_Alarm_PROB_CAUSE_INPUT_OUTPUT_DEVICE_ERROR*
- *CL_Alarm_PROB_CAUSE_EQUIPMENT_MALFUNCTION*
- *CL_Alarm_PROB_CAUSE_ADAPTER_ERROR*

Following are the probable causes for Environmental related Alarms:

- *CL_Alarm_PROB_CAUSE_TEMPERATURE_UNACCEPTABLE*
- *CL_Alarm_PROB_CAUSE_HUMIDITY_UNACCEPTABLE*
- *CL_Alarm_PROB_CAUSE_HEATING_OR_VENTILATION_OR_COOLING_SYSTEM_-PROBLEM*
- *CL_Alarm_PROB_CAUSE_FIRE_DETECTED*
- *CL_Alarm_PROB_CAUSE_FLOOD_DETECTED*
- *CL_Alarm_PROB_CAUSE_TOXIC_LEAK_DETECTED*
- *CL_Alarm_PROB_CAUSE_LEAK_DETECTED*
- *CL_Alarm_PROB_CAUSE_PRESSURE_UNACCEPTABLE*
- *CL_Alarm_PROB_CAUSE_EXCESSIVE_VIBRATION*
- *CL_Alarm_PROB_CAUSE_MATERIAL_SUPPLY_EXHAUSTED*
- *CL_Alarm_PROB_CAUSE_PUMP_FAILURE*
- *CL_Alarm_PROB_CAUSE_ENCLOSURE_DOOR_OPEN*

3.1.6 ClAlarmStateT

```
typedef struct {
        CL_Alarm_STATE_CLEAR = 0,
        CL_Alarm_STATE_ASSERT = 1
} ClAlarmStateT;
```

The *ClAlarmStateT* contains the list of enumeration values for the state of the Alarm.

- *CL_Alarm_STATE_CLEAR* - indicates Alarm condition has cleared.
- *CL_Alarm_STATE_ASSERT* - indicates Alarm condition has occurred.

3.1.7 ClAlarmCategoryTypeT

```
typedef struct {
        CL_Alarm_CATEGORY_INVALID = 0,
        CL_Alarm_CATEGORY_COMMUNICATIONS = 1,
        CL_Alarm_CATEGORY_QUALITY_OF_SERVICE = 2,
        CL_Alarm_CATEGORY_PROCESSING_ERROR = 3,
        CL_Alarm_CATEGORY_EQUIPMENT = 4,
        CL_Alarm_CATEGORY_ENVIRONMENTAL = 5
} ClAlarmCategoryTypeT;
```

The *ClAlarmCategoryTypeT* enumeration type contains the various category for Alarms.

- *CL_Alarm_CATEGORY_COMMUNICATIONS* - Category for Alarms that are related to communication.
- *CL_Alarm_CATEGORY_QUALITY_OF_SERVICE* - Category for Alarms that are related to quality of service.

- *CL_Alarm_CATEGORY_PROCESSING_ERROR* - Category for Alarms that are related to processing error.
- *CL_Alarm_CATEGORY_EQUIPMENT* - Category for Alarms that are related to equipment.
- *CL_Alarm_CATEGORY_ENVIRONMENTAL* - Category for Alarms that are related to environmental conditions.

3.1.8 ClAlarmSpecificProblemT

typedef ClUint32T ClAlarmSpecificProblemT;

The type of an identifier to the specific problem of the Alarm.

3.1.9 ClAlarmSeverityTypeT

```
typedef struct {
        CL_Alarm_SEVERITY_INVALID=0,
        CL_Alarm_SEVERITY_CRITICAL=1,
        CL_Alarm_SEVERITY_MAJOR=2,
        CL_Alarm_SEVERITY_MINOR=3,
        CL_Alarm_SEVERITY_WARNING=4,
        CL_Alarm_SEVERITY_INDETERMINATE=5,
        CL_Alarm_SEVERITY_CLEAR=6
} ClAlarmSeverityTypeT;
```

The enumeration *ClAlarmSeverityTypeT* contains the various severity levels of an Alarm.

3.1.10 ClAlarmHandleInfoT

```
typedef struct {
        ClAlarmHandleT AlarmHandle;
        ClAlarmInfoT AlarmInfo;
}ClAlarmHandleInfoT;
```

The *ClAlarmHandleInfoT* structure contains the handle and the information of the Alarm. The published event contains this information.

3.1.11 ClAlarmPollInfoT

```
typedef struct ClAlarmToPoll
        {
        ClAlarmProbableCauseT probCause;
        ClAlarmStateT AlarmState;
}ClAlarmPollInfoT;
```

The *ClAlarmPollInfoT* data structure contains the probable cause and the Alarm state of the Alarm.

3.2 Functional APIs

3.2.1 clAlarmRaise

clAlarmRaise

Synopsis:
Raises an Alarm on a component.

Header File:
clAlarmApi.h

Syntax:

```
ClRcT clAlarmRaise(
            CL_IN ClAlarmInfoT* almInfo
                        CL_OUT ClAlarmHandleT *pAlarmHandle);
```

Parameters:

almInfo: (in/out) Pointer to the structure that contains the Alarm information. Only the category and severity fields of this structure are in/out parameters. If invalid `category` and `severity` values are given, the Alarm manager returns the values configured during modeling.

pAlarmHandle: (out) Pointer to the Alarm handle. The Alarm manager maintains the mapping of the raised Alarm to the Alarm handle.

Return values:

CL_OK: The function executed successfully.

CL_Alarm_ERR_INVALID_MOID: `MoId` is invalid.

CL_Alarm_ERR_INVALID_Alarm: The probable cause (identifier of an Alarm) is invalid.

CL_Alarm_ERR_RAISE_Alarm_FAILED: Request to raise the Alarm failed.

Description:
This function is used by a component to generate an Alarm on a Managed Object (MO) having an erroneous condition. A unique Alarm handle is generated and returned to the application component that raised the Alarm. The Alarm handle is then used by the Fault Manager to query for the Alarm information. Before invoking this function, the application raising the Alarm must supply valid attribute values in the `AlarmInfo` structure. For details on this structure, refer to 3.1.1 section in the Service APIs chapter.

Library File:
ClAlarmClient

Related Function(s):
None.

3.2.2 clAlarmEventDataGet

clAlarmEventDataGet

Synopsis:

Unmarshalls the Alarm information containing Alarm handle and data within `ClAlarmInfoT`.

Header File:

clAlarmApi.h

Syntax:

```
        ClRcT clAlarmEventDataGet(
CL_IN ClUint8T *pData,
CL_OUT ClAlarmHandleInfoT *pAlarmHandleInfo,
CL_IN ClSizeT size);
```

Parameters:

pData (in) The data received through the event on the Alarm event channel.

pAlarmHandleInfo (out) This parameter contains the un marshaled Alarm information.

size (in) The size of `pData`.

Return values:

CL_OK: The function executed successfully.

CL_ERR_NULL_PTR: `pData` or `pAlarmHandleInfo` is a NULL pointer.

Description:

This function is used to Unmarshalls data from the Alarm client. Alarm Manager publishes an event on the channel called `ClAlarmEventName`. Interested services can subscribe to this event and use the `clAlarmEventDataGet()` API to un marshal the payload of the event data.

Library File:

ClAlarmClient

Related Function(s):

None.

3.2.3 fpAlarmObjectPoll

fpAlarmObjectPoll

Synopsis:
The function pointer for the user-defined polling function.

Header File:
clAlarmApi.h

Syntax:

```
ClRcT    (*fpAlarmObjectPoll)(
                          CL_IN CL_OM_Alarm_CLASS*   objPtr,
                          CL_IN ClCorMOIdPtrT        hMoId,
                          CL_INOUT ClAlarmPollInfoT AlarmsToPoll[]);
```

Parameters:
objPtr: (in) This parameter points to an internal structure.

hMoId: (in) This parameter contains the pointer to the MOID.

AlarmsToPoll: (in/out) This parameter contains the list of the probable causes and their state.

Return values:
CL_OK: The function executed successfully.

CL_ERR_NULL_PTR: `hMoId` is a NULL pointer. is not supported.

Description:
This polling function is called periodically by the Alarm service. An application can probe the state of the managed resource to detect erroneous condition using this function. The application returns the status of the probable cause in the polling function. The Alarm service raises the appropriate Alarm based on the value returned by the polling function.

Library File:
ClAlarmClient

Related Function(s):
None.

Glossary

Glossary of COR Service Terms

Error A deviation of a system from the normal operation.

Fault The physical or algorithmic cause of malfunction. Faults manifests themselves as errors.

Alarm A notification of a specific event. An Alarm may or may not represent an error.

Erroneous Condition A condition which the application needs to notify the north bound entity.

Unmarshalled The Alarm information is stored in the XDR format. This is called Marshalling of data. The reverse process of retrieving data into the native format is called Unmarshalling of data.

XDR refers to eXternal Data Representation Standard (described in RFC 1832). It is a means of storing data in a machine independent format.

MSO Managed Service Object

MOID Managed Object Identifier

References

CCITT Recommendation X.733(1992)|ISO/IEC 10165-4 : 1992, Information technology - Open System Interconnection - Systems Management: Alarm Reporting Function.

Index