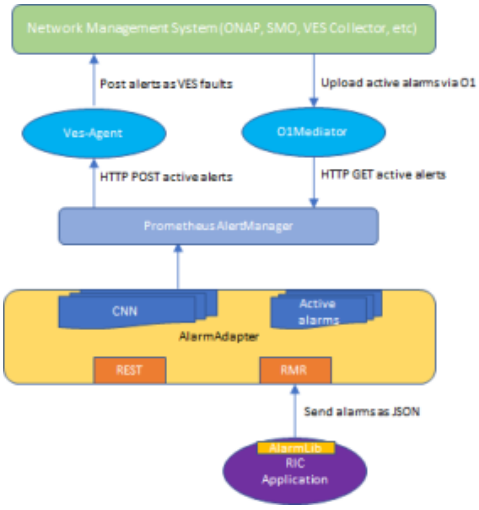
RIC Alarm System

High level architecture



Overview

RIC alarm system implements two components: Alarm Adapter and application library interface

The *AlarmAdapter* is responsible for managing alarm situations in RIC cluster and interfacing with Northbound applications such as *Prometheus AlertManager* to post the alarms as alerts. *AlertManager* takes care of deduplicating, silencing and inhibition (suppressing) of alerts, and routing them to the *VESAgent*, which, in turn, takes care of converting alerts to faults and sending them to ONAP as VES events.

The *Alarm Library* provides a simple interface for RIC applications (both platform application and xApps) to raise, clear and re-raise. The *Alarm Library* inter acts with the *AlarmAdapter* via RMR interface.

Alarm Adapter

TODO

Alarm Library

Initialization

A new alarm instance is created with InitAlarm()-function. ManagedObject (mo) and Application (ap) identities are given as a parameter

Alarm Context/Object

The Alarm object contains following parameters:

- * *SpecificProblem*: problem that is the cause of the alarm
- *PerceivedSeverity*: The severity of the alarm, see above for possible values

- * *ManagedObjectId*: The name of the managed object that is the cause of the fault
- * *ApplicationId*: The name of the process raised the alarm
- *AdditionalInfo*: Additional information given by the application
- * *IdentifyingInfo*: Identifying additional information, which is part of alarm identity

Ietms marked with ⭐*, i.e., ManagedObjectId* (mo), *SpecificProblem* (sp), *ApplicationId* (ap) and *IdentifyingInfo* (IdentifyingInfo) make up the identity of the alarm. All parameters must be according to the alarm definition, i.e. all mandatory parameters should be present, and parameters should have correct value type or be from some predefined range. Addressing the same alarm instance in a *clear*() or *reraise*() call is done by making sure that all four values are the same is in the original *raise()* / *reraise()* call.

Alarm APIs

- *Raise*: Raises the alarm instance given as a parameter
- *Clear*: Clears the alarm instance given as a parameter, if it the alarm active
- *Reraise*: Attempts to re-raise the alarm instance given as a parameter
- *ClearAll*: Clears all alarms matching **moId** and **appId** given as parameters

Example on how to use the API

```go
package main

import (
    alarm "gerrit.o-ran-sc.org/r/ric-plt/alarm-go/alarm"
)

func main() {
    // Initialize the alarm component
    alarmer, err := alarm.InitAlarm("my-pod", "my-app")

    // Create a new Alarm object (SP=8004, etc)
    alarm := alarmer.NewAlarm(8004, alarm.SeverityMajor, "NetworkDown", "eth0")

    // Raise an alarm (SP=8004, etc)
    err := alarmer.Raise(alarm)

    // Clear an alarm (SP=8004)
    err := alarmer.Clear(alarm)

    // Re-raise an alarm (SP=8004)
    err := alarmer.Reraise(alarm)

    // Clear all alarms raised by the application - discussion ongoing if to be deprecated.
    err := alarmer.ClearAll()
}
```

Example VES event

```
INFO[2020-06-08T07:50:10Z]
{
  "event": {
    "commonEventHeader": {
      "domain": "fault",
      "eventId": "fault0000000001",
      "eventName": "Fault_ricp_E2 CONNECTIVITY LOST TO G-NODEB",
      "lastEpochMicrosec": 1591602610944553,
      "nfNamingCode": "ricp",
      "priority": "Medium",
      "reportingEntityId": "035EEB88-7BA2-4C23-A349-3B6696F0E2C4",
      "reportingEntityName": "Vespa",
      "sequence": 1,
      "sourceName": "RIC",
      "startEpochMicrosec": 1591602610944553,
      "version": 3
    },

    "faultFields": {
      "alarmCondition": "E2 CONNECTIVITY LOST TO G-NODEB",
      "eventSeverity": "MAJOR",
      "eventSourceType": "virtualMachine",
      "faultFieldsVersion": 2,
      "specificProblem": "eth12",
      "vfStatus": "Active"
    }
  }
}
INFO[2020-06-08T07:50:10Z] Schema validation succeeded
```

Additional information

The README.md file in the ric-plt/alarm-go repository contains additional information about the alarm system.