

# A Sensor Abstraction Layer for FMA

version 0.1

Cynthia McGuire

cindi@sun.com

## 1 Introduction

This paper proposes extensions to the fault management architecture (FMA) [1] to support a sensor abstraction layer for the collection and analysis of sensor based telemetry that can be used in fault and resource management. The proposal suggests a phased approach to delivery in order to close the gap on our ability to effectively use sensor telemetry in fault diagnosis and administrative alerts.

### 1.1 The Problem

How do we manage raw telemetry data kept, maintained and exported by disparate sources for the purposes of fault management and resource management and budgeting? Today, there are a number of sensor collection mechanisms exported by hardware and software. For the most part, the exported information is haphazardly presented and accessed according to ad-hoc operating system interfaces (e.g. kstat(1M)), per-platform methods or per-subsystem industry standards (e.g. SMBus, SMART and IPMI). Using this data for fault or resource management is clumsy and typically requires low-level system knowledge baked into higher-level management applications.

### 1.2 Key Objectives

As part of an overall sensor abstraction layer based on our current fault management architecture, we can solve the problem described in section 1.1 and provide a better understanding of the overall health and usage of a system through more sophisticated diagnosis technologies and fine-grained observability of sensor data via common access methods. A sensor abstraction layer must possess:

1. the ability to alert the administrator to conditions observed by platform sensors that may impact the operational state of the platform.
2. the ability to alert the administrator to conditions that resolve themselves as observed by platform sensors.
3. the ability to watch one or more sensors and correlate the data for predictive fault analysis or resource management.
4. the ability to continuously record sensor data and retrieve it from systems for offline analysis, future system design or development of more advanced diagnosis algorithms.
5. the ability for administrators and service personnel to manually inspect sensor values without having to understand the exact implementation (e.g. IPMI or SMBus).
6. the ability to connect sensor data to higher-level diagnosis (e.g. SMART disk data to SCSI and ZFS diagnosis engines)
7. the ability to understand and observe performance and power budgets based on raw sensor data.

A sensor abstraction layer to address the objectives detailed above are split into three distinct sub-layers: a sensor provider layer, a sensor collection layer and a analyzer layer. At the lowest level, the sensor provider layer exports interfaces to read sensor values without having to understand the

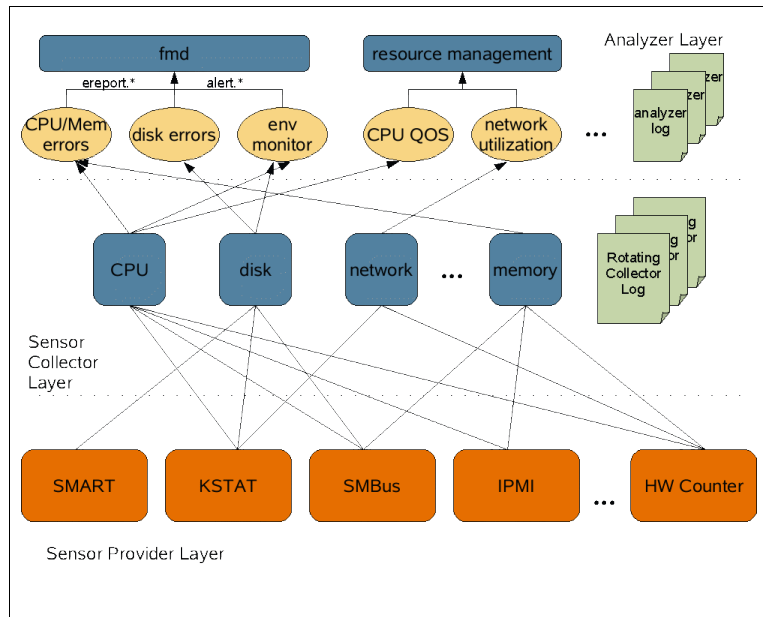
implementation details of a sensor.

Sensor values are read and logged according to collection criteria established for a sensor or set of sensors at the collector layer. For example, we may want to collect telemetry for a CPU from a thermal sensor acquired via IPMI and power status exported by a kstat. Collection parameters may be configured per sensor and the telemetry data collected is continuously logged as a *black box* for offline analysis, future system design or development of more advanced diagnosis and management.

Sensor telemetry is passed from collectors to the analyzer layer for the purpose of online analysis. Analysis software may be simple: an IPMI-based environmental temperature monitor. Analysis software may also be complex: power budget analyzer .

While it is desirable to meet all of our objectives with a single project, there are a number of pressing problems that need to be addressed by a phased delivery of extensions to the current FMA to meet service requirements for platforms shipping today and in the not too distant future.

The remainder of this document describes phase I of a larger project to deliver sensor-based diagnosis and an alerts facility to FMA. The full sensor abstraction project will be delivered as part of a separate or multiple follow-on projects.



## 2 Phase I

### 2.1 The Problem

How do we manage sensor data to offer fault diagnosis and recovery for internal platform faults and alert the administrator and service personnel to possible external contributing factors?

For example, an over-temperature condition detected by a platform sensor may be explained by a fan that has stopped spinning, or a bad sensor. These problems are diagnosed by the platform fault manager and acted upon by platform-specific agents to record the faulty ASRU and log a fault message to the console that something is in need of repair. However, in this situation and others like it, there may be external contributors to the over-temperature condition: the HVAC system is broken, there is a blockage in the air ducts or the room is on fire. The list of possibilities is quite long and impossible to accurately diagnose inside the platform fault domain.

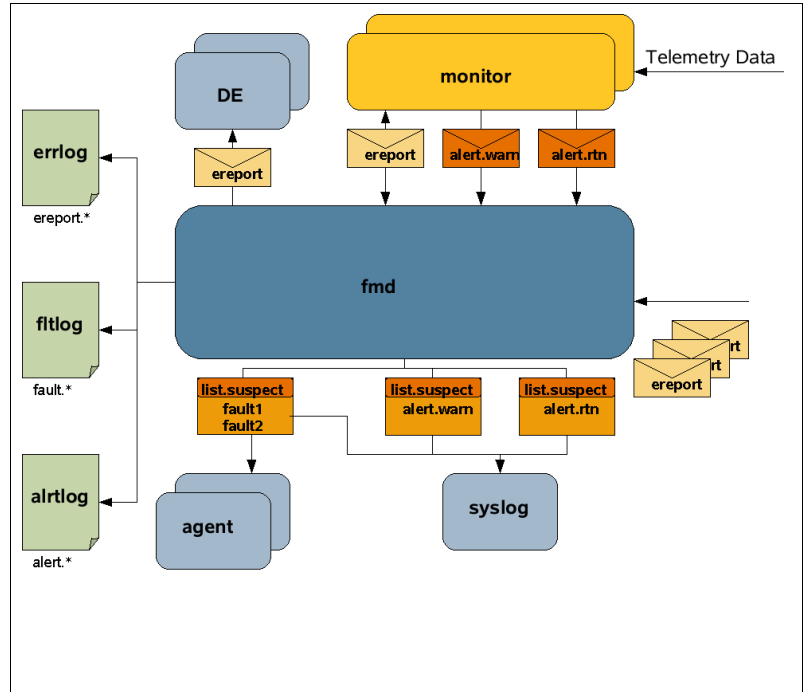
The fault management architecture is designed to diagnose error telemetries for which it a) has an understanding of how fault propagations flow in its fault domain and b) act on ASRUs that are contained in its fault domain. Exceptions to this rule exist when the platform fault manager DE is implemented to understand external fault propagations and a higher-level topological view of the resources, ASRUs and FRUs for which it will diagnose as faulty. Unfortunately, we do not have the ability to do either of those things for some classes of environmental problems. In phase I, we provide a small number of changes to the architecture and suggestions for implementing platform solutions. We note that the phase I solution does not meet all of the requirements of the overall Sensor Abstraction project but does meet some pressing requirements and provides a migration path to a more

comprehensive and feature-rich solution.

## 2.2 Proposed Solution

In phase I of the Sensor Abstraction project, we propose to extend the fault management architecture by supplying:

- a fault managed resource identifier FMRI specification for sensors and indicators
- an FMA event protocol for alert warning and return to normal (RTN) events triggered by external conditions
- extensions to the `fmd(1M)` module API to publish `ereport`, alert warning return to normal (RTN) events
- an observable log of alert warning and return to normal events
- a message standard for alert and return to normal events with connections to <http://sun.com/msg> knowledge articles



These extensions address objectives #1, #2 and possibly #6. The proposed implementation and prototype calls for one or more sensor monitor plugins to `fmd(1M)`. The plugins use the existing fault manager module APIs and may consume sensor telemetry in one of two ways: a) via `ereport` events and raw telemetry data published externally by another software component or b) by polling for changes in sensor data. It is up to the sensor monitor to implement its own access methods for sensor collection if a poll model is employed. If the sensor telemetry is consumed as `ereport` events, the events must be throttled by the event generator such that the `fmd` and the error log is not burdened by an over-abundance of events. In other words, we do not expect to consume raw sensor telemetry such as polled temperature data as an `ereport` event.

The sensor monitor must be written in C and implement its own algorithms for analysis. The sensor monitor may publish `ereport` events for diagnosis of faults within the containing fault domain or alert warning and return-to-normal events for conditions outside of the fault domain. The sensor monitor may be written as a stand-alone software component that publishes `ereport` and alert events via the `fmd` ETM. If this implementation is chosen, it may not consume `ereport` events.

## 2.3 FMRI Specification for Sensors and Indicators

Sensors and indicators are hardware or software components representable by a FMRI. For example, a sensor that detects thermal conditions for a CPU may be physically connected to and accessed via a I2C bus. Using the `hc` scheme, the sensor FMRI is constructed by following its physical connection properties

hc:///chassis=0/i2c-bus=0/temperature-sensor=0

The problem is that there is no link between the service provided by a sensor or indicator to the resource using that service. Using the example above, we see that there is no topological association between

hc:///chassis=0/i2c-bus=0/temperature-sensor=0

and the CPU for which the sensor is observing temperature changes

hc:///chassis=0/motherboard=0/chip=1/cpu=2

This makes it difficult to programmatically make a predictive diagnosis for CPU 2 when its thermal threshold has reached critical levels. To solve this problem, we propose an extension to the current FMRI specification to describe sensor and indicator bindings (associations) for fault managed resources. The FMRI specification is extended to include a *facility* member to represent the binding of a FMRI and a sensor or indicator acting on its behalf. A *facility* may be represented by a resource that is physically connected to another part of the system but is not required to be.

#### FMRI Specification

Name	Data Type	Description
scheme	string	scheme used for FMRI
version	uint32	version of scheme specification
authority	name-value pair list	optional authority (identity) of FMRI
<i>scheme-specific-payload</i>		resource path
facility	Facility name-value pair list (see below)	facility elements for FMRI

#### Facility FMRI Specification

Name	Data Type	Description
facility-type	string	type of <i>facility</i> : <b>sensor</b> or <b>indicator</b>
facility-name	string	name value for <i>facility</i>

The FMRI string syntax is changed as follows to include a facility leaf element delimited by a question mark, “?”.

<scheme>://[authority]/<resource-path>[?<fac\_type>=<fac-name>]

Continuing with our CPU temperature example, we can see how to link the physical location for the CPU to its temperature sensor using our new syntax:

```
hc:///chassis=0/chip=1/cpu=0?sensor=hc:///chassis=0/i2c-bus=0/temperature-sensor=0
```

The *facility* name may not necessarily be a in FMRI format if it does not have an underlying FMRI representation. For example, the physical path to a SES disk LED may be difficult to determine and the *facility* takes on a name relative only to the disk slot exporting the indicator:

```
hc:///ses-chassis=0/disk-slot=1?indicator=ok2rm
```

Similarly, the we may want to use platform labeling conventions to assign the *facility* name:

```
hc:///chassis=0/chip=1/cpu=0?sensor=CPU-TEMP
```

## 2.4 Alert Warning and Return to Normal Event Specification

The sensor abstraction project defines a new FMA protocol [1] event class, `alert`. `alert` events are organized into two sub-categories, `warn` and `RTN` (return to normal) such that all `alert` event class names begin with `alert.warn` or `alert.RTN`. Like `fault` events, `alert` events are published in a higher order `list.suspect` event. `list.suspect` events for `alert` conditions, however, may contain one and only one `alert` event. In other words, `fault` events may not be published in a `list.suspect` event that contains an `alert` event nor can there be multiple `alert` events in a `list.suspect` event. All `alert` events are defined to contain a set of common payload members.

**alert Event Specification**

Payload Member Name	Data Type	Description
<code>version</code>	<code>uint8</code>	common event payload member
<code>class</code>	<code>string</code>	common event payload member
<code>detector</code>	FMRI	detector of this <i>alert</i>
FRU	FMRI	optionally available FRU
<code>auto-resolve</code>	<code>boolean</code>	indicates whether or not the <i>alert</i> cache should automatically to the resolved state when the fault manager restart or the system reboots.

`alert.RTN` events are defined to include one additional payload member to permit tracking of the original `alert.warn` associated with the *return to normal*. `warn-uuid` is the UUID of the

list.suspect event containing the original alert.warn event.

### Additional Return to Normal Payload

Payload Member Name	Data Type	Description
warn-uuid	string	UUID of original alert.warn

## 2.5 Alert Warning and Return to Normal Message Specification

Trivial modifications to the standard FMA message format are required in order to accommodate *alert* messages. Two new message types are added and an expanded definition of the Minor severity level is provided with this project. There will be two additional message types added: ALERT and ALERT-RTN are used to respectively describe an *alert warning* or *return to normal* message. The basic elements and format of the messages remain the same as we see in this example of a temperature warning and its corresponding return to normal.

MSG-ID: ENV-8000-AV, TYPE: ALERT, VER: 1, SEVERITY: MINOR

EVENT-TIME: Tue Apr 06 17:19:56 PDT 2007

PLATFORM: SPARC-Enterprise, CSN: nwk-dc2-1, HOSTNAME: nwk-dc2-1-sc0

SOURCE: sensormod, REV: 1.0

EVENT-ID: 18a7990d-214e-4e10-c7b6-ec4323369596

DESC: Ambient air temperature has exceeded the warning level. Refer to <http://www.sun.com/msg/ENV-8000-AV> for more information.

AUTO-RESPONSE: None at this time.

IMPACT: The system will power down if the ambient temperature continues to rise above operational limits.

REC-ACTION: Check facility temperature controls and for obstructions to air flow near this system. The identity of the specific ambient air temperature sensor can be determined using `fmdump -w -v -u <EVENT_ID>`. Please consult <http://www.sun.com/msg/ENV-8000-AV> for more information.

MSG-ID: ENV-8000-7U, TYPE: ALERT-RTN, VER: 1, SEVERITY: MINOR

EVENT-TIME: Tue Apr 06 17:22:34 PDT 2007

PLATFORM: SPARC-Enterprise, CSN: nwk-dc2-1, HOSTNAME: nwk-dc2-1-sc0

SOURCE: sensormod, REV: 1.0

EVENT-ID: e154a864-dd64-e671-8b1f-aaf994476832

DESC: Air temperature that was previously reported as exceeded a warning limit has returned to normal operating limits. Refer to <http://www.sun.com/msg/ENV-8000-7U> for more information.

AUTO-RESPONSE: None at this time.

IMPACT: None. This an informational message indicating a Return To Normal (RTN) operation.

REC-ACTION: The previously reported air temperature alert associated with this RTN message can be determined using `fmdump -w -v -u <EVENT_ID>`. Please consult <http://www.sun.com/msg/ENV-8000-7U> for more information.

There is no need for changes in the FMA Event Registry [2] infrastructure or the existing content that drives syslog(3C) messaging and the <http://sun.com/msg> articles.

## 2.6 fmd Module API Extensions

This section describes the application programming interface (API) and infrastructure changes required to support alert events and messaging in the Solaris *fault manager*.

### 2.6.1 Case Management

The case management states are extended as follows to allow cases associated with alert events to be instantiated by a sensor monitor plug-in, managed by the fault manager and acted upon by an agent.

Unsolved	No Change
Solved	The case has been <i>solved</i> as a result of the module adding one alert event to it and calling <code>fmd_case_solve()</code> . The case may contain one <code>alert.warn</code> event or one <code>alert.rtn</code> . Any other event is considered a programming error. The fault manager, <code>fmd</code> , will add an associated entry for the alert event to the alert cache if the case contains an <code>alert.warn</code> event.
Close_Wait	No Change
Closed	The case transitions to the <i>Closed</i> state because it was <i>solved</i> and an <i>agent</i> has acted upon it. The fault manager dispatches a <code>list.isolated</code> event indicating that the agent software has taken the appropriate steps (i.e. lighting LEDs or shutting down or bring back system services) to alert the administrator of potential system damage or return to normal system operation.
Repaired	The alert condition associated with this case is no longer in effect. The case will transition to the <i>Repaired</i> state when <code>fmadm repair &lt;uuid&gt;</code> is used to manually indicate an alert is no longer in effect or by a call to <code>fmd_case_repair()</code> from a sensor monitor or agent. If <code>fmd_case_repair()</code> is called for a case containing an <code>alert.rtn</code> event, <code>fmd</code> will transition the case associated with the corresponding <code>alert.warn</code> event to the <i>Repaired</i> state and update the alert cache. When the case transitions to the <i>Repaired</i> state, the fault manager dispatches a <code>list.repaired</code> event.

### 2.6.2 Alert Cache

The fault manager maintains a cache of outstanding alert conditions that have been referenced by a diagnosis. The alert cache is composed of a collection of log files, each associated with an `alert.warn` event. The log files persist across system reboots and `fmd` restarts to permit outstanding `alert.warn` events to be replayed as required. Each entry in the alert cache can be in one of three states:

Outstanding	A warning ( <code>alert.warn</code> ) is in effect. This state applies only to <code>alert.warn</code> events.
OK	This was return to normal ( <code>alert.RTN</code> ) event or a warning ( <code>alert.warn</code> ) event no longer in effect.

When the fault manager initializes, the set of persistent alert logs is examined to populate the alert cache. If the entry is in the *OK* state and its age exceeds a configurable threshold, the alert log is permanently deleted. If the entry is in the *Outstanding* state, the the fault manager checks the `auto-resolve` member of the `alert.warn` event assigned to this entry. If `auto-resolve` is set, the entry transitions to *OK*. Otherwise, the `alert.warn` event is replayed to any subscribing modules, permitting agents to light appropriate indicators or shutdown system resources.

The alert cache is maintained as a set of persistent alert logs, which are kept in the directory `/var/fm/fmd/alrt` and are considered Project Private. To aid field personnel in the event of a problem, the `fmadm(1M)` command provides facilities for manually flushing alert cache entries. Each alert log is maintained using the same Extended Accounting log format used for the `fmd(1M)` ASRU cache logs except that events logged to these files are of class `alert.fm.*`, where each event represents a change in state and contains the UUID of the case associated with this change, and a copy of the relevant alert event for replay. `fmdump(1M)` can also be used to analyze the history of a particular alert log. Each alert log is named by the same UUID associated with the case that instantiated the log. A new `fmadm(1M)` sub-command, `alerts`, may be used to observe the state of the alert cache along with the sensor FMRI that detected it. Only entries in the *Faulty* state are displayed and permits us to answer the request to *show me all outstanding alerts on my system*.

```
# fmadm alerts
```

```
ALERT          TIME          UUID
-----
overtemp       Apr 06 17:22:34  18a7990d-214e-4e10-c7b6-ec4323369596
under-voltage  Apr 06 17:24:06  42af921a-d2e6-ccce-f18c-d9d7fe6cd650
```

## 2.7 fmd Alert Log and Observability

A history of all alert warning and return to normal events will be captured in an alert log and persisted in `/var/fm/fmd/alrtlog`. The data is format is project private and visible via `fmdump(1M)` with the `-w` option is used to view the complete history of alert warning and return to normal events. For example,

```
# fmdump -w
```

```
TIME          UUID          SUNW-MSG-ID
Apr 06 17:19:56.6432 18a7990d-214e-4e10-c7b6-ec4323369596 ENV-8000-AV
Apr 06 17:22:34.5102 e154a864-dd64-e671-8b1f-aaf994476832 ENV-8000-7U
Apr 06 17:24:06.4973 42af921a-d2e6-ccce-f18c-d9d7fe6cd650 ENV-8000-8L
Apr 06 17:26:12.7746 0272b736-aa1b-cb8d-fb4f-ec98a9af5b04 ENV-8000-5M
```

```
# fmdump -w -v -u 18a7990d-214e-4e10-c7b6-ec4323369596
```

```
TIME          UUID          SUNW-MSG-ID
Apr 06 17:19:56.6432 18a7990d-214e-4e10-c7b6-ec4323369596 ENV-8000-AV
alert.warn.overtemp
```

```
Alert by: hc:///chassis=0/chip=1/cpu=0?sensor=over-temp
```

```
Affects: hc:///chassis=0/chip=1/cpu=0
```

```
# fmdump -w -v -u e154a864-dd64-e671-8b1f-aaf994476832
```

```
TIME                               UUID                               SUNW-MSG-ID
```

```
Apr 06 17:22:34.5102 e154a864-dd64-e671-8b1f-aaf994476832 ENV-8000-7U
```

```
  alert.rtn.overtemp
```

```
    Alert by: hc:///chassis=0/chip=1/cpu=0?sensor=over-temp
```

```
      Affects: hc:///chassis=0/chip=1/cpu=0
```

```
      warn-UUID: 18a7990d-214e-4e10-c7b6-ec4323369596
```

## 2.8 SNMP Connector Agent

To be filled in

## 3 Future Phases

TBD

## 4 Appendix

### 4.1 Use Cases

This use case assumes an air cooled system, with internal air movers and temperature sensors and a means for monitoring both. This case applies to systems with a service processor (SP).

Preconditions

Error Checking capabilities:

Ambient air temperature thresholds Fan rotation/speed thresholds Component level temperature sensors (heat sinks, core temps, etc.) There is AUX power or sufficient power hold up to latch evidence of this error in NVRAM (System Event Log or SEL). The SP is designed to handle this error condition Standard FMA software is installed including the required DE(s) and supporting Platform Sensor FM modules Diagnosis is performed by a chassis DE installed on the platform (either SP or host based). All diagnosis engines responsible for hardware faults subscribe to P&E errors and understand the propagation effects associated with the hardware components for which they provide diagnosis coverage. The platform provides the appropriate event transport to enable subscription by all involved diagnosis engines. Incoming air temperature thresholds are set to generate an error before internal temperature thresholds cause an emergency power off of the platform.

Trigger: A sensor value indicates the ambient temperature has exceeded a present error threshold.

Success Scenario:

1. The BMC polls for sensor transitions every 5 seconds and discovers that an ambient air overtemp transition has occurred.
2. The BMC generates a IPMI trap for the ambient air overtemp transition.
3. The IPMI sensor data record for this event is recorded in the system event log (SEL).

4. The IPMI trap is recognized by the platform sensor monitor module who opens a case and publishes `list.suspect` event with an `alert.warn.overtemp` event.
5. The supporting knowledge article for this message ID indicates that the product reported an ambient air overtemp condition and they should check for a facility temperature control problem or an air inlet obstruction for the identified platform. If none is found and the problem persists, contact their service provider....something along those lines.
6. No entries are made in the `fmd` ASRU cache.
7. The alert event is logged by `fmd` in the alert log file.
8. The syslog agent subscribes to `alert.warn` events and generates a standard FMA event message of the type `ALERT`.
9. The `fmd(1M)` SNMP agent subscribes to alert events and generates the specified trap (and we have the associated objects in the MIB)

Trigger: A sensor value indicates the ambient temperature has transitioned to below its error threshold.

1. Return to normal/repair: The BMC polls for sensor transitions every 5 seconds and discovers that the ambient air temperature has transitioned to an overtemp condition. The BMC generates a IPMI trap for the ambient air overtemp transition. The IPMI Sensor Data Record for this event is recorded in the System Event Log (SEL). The platform sensor monitor listens for and recognizes the IPMI trap. It opens a case and publishes a `list.suspect` event that contains a return to normal event, `alert.RTN.overtemp`. The monitor closes the case associated with the originating `list.suspect` or `alert` event. An alternate approach would be to have an agent subscribe to the `alert` events and be responsible for case closure.
2. The syslog agent subscribes to `alert.RTN` events and generates a standard FMA message of type `ALERT`.
3. The `fmd(1M)` SNMP agent subscribes to `alert.RTN` events and generates an associated trap (and we have the associated objects in the MIB).
4. `fmd` logs the `alert.RTN` event in its alert log.

## References

[1] Fault Management Architecture Overview

[http://www.sun.com/bigadmin/content/selfheal/selfheal\\_overview.pdf](http://www.sun.com/bigadmin/content/selfheal/selfheal_overview.pdf)

[2] FMA Events Registry

<http://opensolaris.org/os/project/events-registry>