Service Delivery Broker

Service Management Interface

Version 0.4 – 2014-05-09

Table Of Contents

1. Legal Disclaimer

THE CONTENTS OF THIS DOCUMENT ARE PROVIDED “AS IS”. THIS INFORMATION COULD CONTAIN TECHNICAL INACCURACIES, TYPOGRAPHICAL ERRORS AND OUT-OF-DATE INFORMATION. THIS DOCUMENT MAY BE UPDATED OR CHANGED WITHOUT NOTICE AT ANY TIME. USE OF THE INFORMATION IS THEREFORE AT YOUR OWN RISK.

IN NO EVENT SHALL PORTUGAL TELECOM BE LIABLE FOR SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM OR RELATED TO THE USE OF THIS DOCUMENT.

2. Abstract

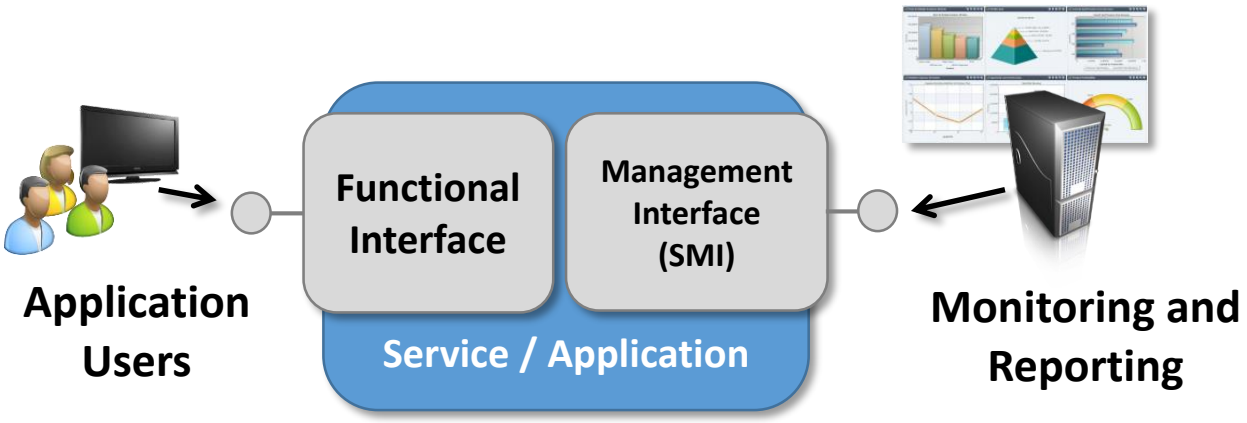
This document presents the Service Management Interface (SMI), an interface to manage a cloud/digital service or application. This document also presents some guidelines to implement SMI event-driven.

3. Introduction

SMI – Service Management Interface

The Service Management Interface (SMI) is an interface to manage a cloud / digital service or application. The main idea is to have a different interface from the functional interface to monitoring and configure a service/application.

The following image illustrates the management interface pattern:



The SMI is divided in two components:

- **Set of operations**
  SMI provides a set of simple operations that allows activating or deactivating a service and check the health or state of a service.
- **Event-driven**
  Using a standard Management Report, it is possible to send events of service failures as they occur.

With SMI we have a standard interface to manage services, querying the service state and health, update service configurations, etc.

4. SMI Operations

SMI provides a set of operations for service management.

The set of operations are:

- **GetExecutionState**
  Returns an ExecutionState describing the current state of a service instance.

- **SetExecutionState**
  Allows activating or suspending service execution. It has one output parameter (a Boolean) that will return the value 'True' if the change of service execution state requested by the consumer was made successfully. It has one input parameter that specifies if it is to activate or suspend the service instance.

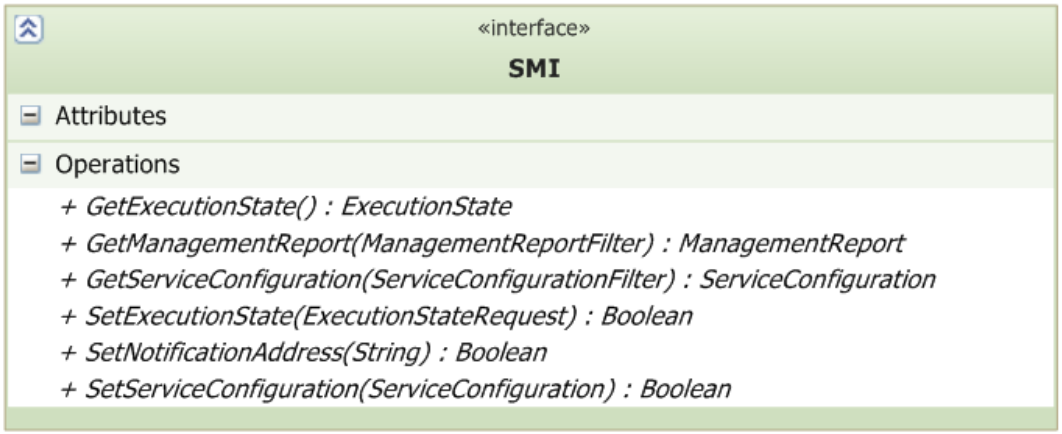
- **GetServiceConfiguration**
  Returns a ServiceConfiguration containing a list of pairs Key/Value that describe the current set configuration values used by the service instance. Optionally, it accepts a ServiceConfigurationFilter input parameter that allows the service consumer to select which information will be returns and also to control the amount of configuration values returned, using a Pagination element in the ServiceConfigurationFilter. The operation will return a complete ServiceConfiguration when the parameter ServiceConfigurationFilter is not received.

- **SetServiceConfiguration**
  Configures the current set configuration values used by the service instance. It requires a ServiceConfiguration input paramenter containing a list of pairs Key/Value that describe the current set configuration values to be used by the service instance.

- **GetManagementReport**
  Returns a ManagementReport containing information about the service instance health, execution state, eventual failures and metrics. Optionally, it accepts a ManagementReport input parameter that allows the service consumer to select which information will be returned and also to control the amount of Metrics returned, using a Pagination element in the ManagementReportFilter. The operation will return a complete ManagementReport when the paramenter ManagementReportFilter is not received.

- **SetNotificationAddress**
  It has one input parameter that configures the communication endpoint address that the service instance must use to deliver a report containing information about their health, execution state, eventual failures and metrics. It has one output parameter (a Boolean) that will return the value 'True' if the change of the notification address requested by the service consumer was made successfully.

The following image illustrates the SMI interface:



5. Management Report

In the previous section, it was mentioned that the GetManagementReport operation returns a ManagementReport object.

A ManagementReport contains the information about the service instance health, execution state, eventual failures and metrics.

In this section it will be described in detail the composition and meaning of each field of the ManagementReport objects.

Object Definition

ManagementReport

Name	Type	Description
ID	string	Identifies the managent report
SourceID	string	Identifies the source of the report
DateTime	DateTime	The date of the report
State	State	The State of the service
Metrics	Metric[]	Array of metrics

State

Name	Type	Description
Health	HealthState	The health state
Execution	ExecutionState	The execution state
Failures	Failure[]	Array of failures

Metric

Name	Type	Description
MetricID	string	Identifies the metric
SourceID	string	Identifies the source
CategoryID	string	Identifies the category of the metric
Reference	string	
DateTime	DateTime	Date
Value	string	Value of the metric
Details	object	The details of the metric

ExecutionState – enumeration

Name	Description
Activating	The service instance is activating
Active	The service instance is active
Suspending	The service instance is suspending
Suspended	The service instance is suspended

HealthState – enumeration

Name	Description
Unkown	The health is unkown
Operational	The service is operational without failures
OperationalWithFailures	The service is operational but it has failures
Inoperational	The service is inoperational

Failure

Name	Type	Description
FailureID	string	Identifies the failure type
SourceID	string	Identifies the source of the failure (not of the report!)
Detail	string	The details of the failure

ManagementReport Samples

As mentioned before, a ManagementReport can contain a list of failures of the service or application. It will be shown two examples of a ManagementReport, one without failures and one with failures.

An example of a ManagementReport without failures is:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
   <GetManagementReportResponse xmlns="http://schemas.tmforum.org/2011">
      <GetManagementReportResult xmlns:i="http://www.w3.org/2001 ">
        <ID>91c334aa-1552-4700-9fd8-198bc73e4094</ID>
        <SourceID>48FCCA1D-8F84-47A0-BFDA-80B8819F87D0</SourceID>
        <DateTime>2013-07-03T17:42:04.3696339+01:00</DateTime>
        <State>
          <Execution>Active</Execution>
          <Health>Operational</Health>
          <Failures/>
        </State>
      </GetManagementReportResult>
    </GetManagementReportResponse>
  </s:Body>
</s:Envelope>
```

This example shows a report of an active service instance with a health state of “Operational” and with no failures.

Another example of a ManagementReport is:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <GetManagementReportResponse xmlns="http://schemas.tmforum.org/2011">
      <GetManagementReportResult xmlns:i="http://www.w3.org/2001">
        <ID>f35ea993-0b54-490e-9dd2-a4d014750bea</ID>
        <SourceID>48FCCA1D-8F84-47A0-BFDA-80B8819F87D0</SourceID>
        <DateTime>2013-07-03T18:01:17.8649857+01:00</DateTime>
        <State>
          <Execution>Active</Execution>
          <Health>OperationalWithFailures</Health>
          <Failures>
            <Failure>
              <FailureID>DB.0001</FailureID>
              <SourceID>A3E19321-CC49-4BA5-95A6-5FF37AA50FD5</SourceID>
              <Detail>
                Unable to connect to database.
                Exception Message: Failed to connect to '10.9.8.7'.
                Exception StackTrace: in C:\NpgsqlClosedState.cs:line 204
              </Detail>
            </Failure>
          </Failures>
        </State>
      </GetManagementReportResult>
    </GetManagementReportResponse>
  </s:Body>
</s:Envelope>
```

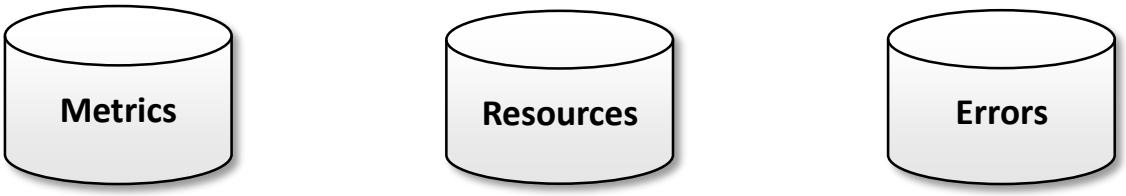
This example reports a failure “DB.0001” which corresponds to “Unable to connect to database”. The failure contains a SourceID field that identifies the source of the failure. In this case it identifies the database that failed. The failure details contain the information relevant for the notification receiver. The heath state of this service instance is “OperationalWithFailures”.

6. Data Sources

All of the information should be used in a standard way, using metric, failure and resources catalogues. With standard information one can say, for example, that metric “123” corresponds to “Requests per second” and use the code to refer the metric. The same is done to failures types (e.g. failure “789” is “Database connection error”) and source (e.g. source “DB1” is “Database 1”).

Every service or application has its own dependencies that could be other services or applications, databases, etc. With a graph of dependencies defined when a service S1 detects a failure connecting to database DB1, the service can send a management report with the failure and both service S1 and database DB1 owners can be notified.

To achieve this standardization there are three catalogues: Metric Catalogue, Resource Catalogue and Failure Catalogue. These catalogues are common to every SMI implementation, therefore if the ID “DB1” is used to identify “Database 1” for Service A then for Service B “DB1” also represents “Database 1”. There cannot exist two resources, failures or metrics with the same ID.



Metric Catalogue

The metric catalogue contains all the possible metrics and correspondent identifiers. These identifiers present in this catalogue are used as ‘MetricID’ field of the Metric object described in Section 5.

Resource Catalogue

The resource catalogue contains the identifiers of the resources. These identifiers are used in the ‘SourceID’ field of the Failure object and the ‘SourceID’ field of the ManagementReport object described in Section 5. With these catalogue it is possible to create dependency graphs once each resource has a unique identifier. The dependency graph allows to notify all the owners of resources that share a dependency that failed, for example.

Failure Catalogue

The failures catalogue contains the identifiers of the failures. These identifiers are used in the ‘FailureID’ field of the Failure object described in Section 5. This catalogue will allow to have

certain metrics that otherwise would be difficult to have. For example, it is possible to determine what is the most common error reported, or trigger an event when it is not possible to connect to a database ten time in a minute, independently of the service that could not connect.

These catalogues and identifiers are provided by the SDB. To get an identifier of a failure, metric or resource you must contact the SDB in order to provide you the identifiers. It is possible that not all the resources, metrics or failure are present in the catalogues. If you need an identifier that is not registered you should request to add it to SDB.

7. Event-driven notifications

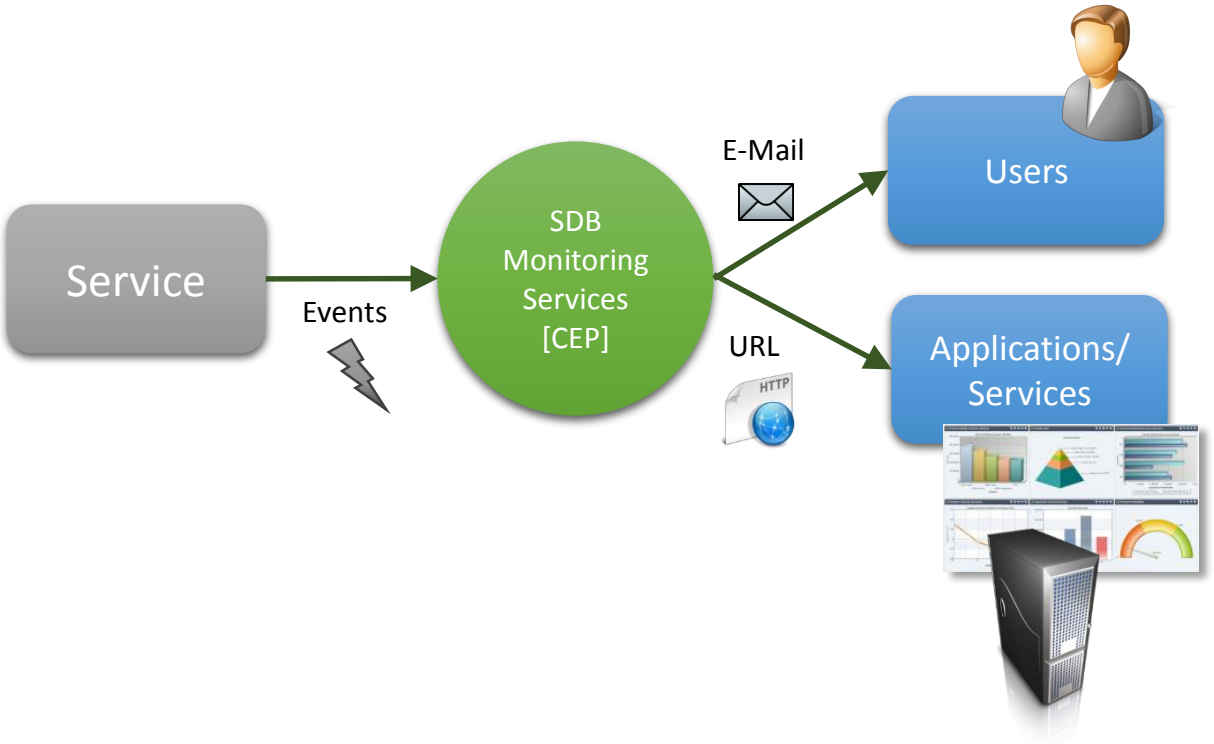
Using the SMI ManagementReport it is possible to be notified when a failure occurs in a service in a standard way.

The event-driven component of the SMI allows a service to send a ManagementReport with the information about the failure that occurred, the health state or the metrics defined. The ManagementReport is sent to the notification address that can be defined with the SetNotificationAddress operation.

The management report contains information about the failures, source of the failures and details that are useful to determine the cause of the failure. Each failure can have a description so that the context of the failure and the condition in which the failure occurred can be analyzed by the management report receiver.

It is possible to integrate the event-driven component of SMI with the Service Delivery Broker (SDB). It is possible to send the management report to SDB Monitoring Services (CEP) that process the failures reported and redirects or notify the correct person. These notifications can be sent to an endpoint defined by the SDB and it is different for each application.

The following image illustrates the flow of an event-driven notification:



Guidelines

To implement event-driven SMI it is good to follow these guidelines:

- **Identify the possible failures**
  First it is necessary to verify which failures we want to report. Use the respective code of the failure to report if it exists.

- **Identify the code parts that can originate a failure**
  It is necessary to identify the code that can cause or detect a failure. E.g. an access to a database, consuming other service, etc.

- **Create a ManagementReport object**
  When starting a request process initialize a management report with the date and the ID (SourceID) of that service instance.

- **Capture the failures**
  When a failure occurs, add it to the list of failures of the management report. The failure must contain the respective FailureID and SourceID. The SourceID field is the source of the failure (e.g. the ID of the database 1). The detail field should contain all the relevant information to determine the cause of the failure or possible information to solve the problem.

- **Send the report**
  At the end of processing the request, send the report to the notification address defined.

Publishing Management Reports

The event-driven Management Reports can be published to an endpoint provisioned by SDB.

The reports can be sent in **XML** format or in **JSON**. The endpoint will be defined for each application by the SDB. Example of an HTTP POST request for a dummy application in XML format:

```
POST http://services.sapo.pt/foo/smi/notification/
Content-Type: application/xml:

<ManagementReport>
  <ID>fb924412-240f-4b3b-80dc-c9ebbcaeb2c4</ID>
  <SourceID>3</SourceID>
  <DateTime>2014-03-24T13:09:19.4196243Z</DateTime>
  <State>
    <Execution>Active</Execution>
    <Health>OperationalWithFailures</Health>
    <Failures>
      <Failure>
        <FailureID>1</FailureID>
        <SourceID>2</SourceID>
        <Detail>detail</Detail>
      </Failure>
```

```
      <Failure>
        <FailureID>2</FailureID>
        <SourceID>2</SourceID>
        <Detail>detail2</Detail>
      </Failure>
    </Failures>
  </State>
  <Metrics>
    <Metric>
      <MetricID>1</MetricID>
      <SourceID>2</SourceID>
      <CategoryID>0</CategoryID>
      <DateTime>2014-03-24T13:09:19.4216245Z</DateTime>
      <Value>3</Value>
    </Metric>
  </Metrics>
</ManagementReport>
```

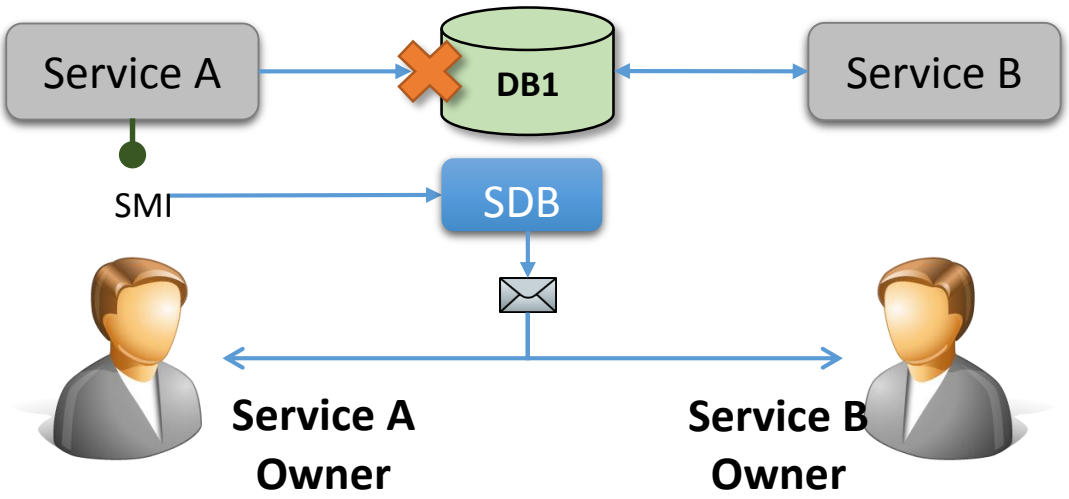
Example of the same ManagementReport in JSON format:

```
Payload=
{
   "ID":"fb924412-240f-4b3b-80dc-c9ebbcaeb2c4",
   "SourceID":"3",
   "DateTime":"2014-03-24T13:09:19.4196243Z",
   "State":
   {
       "Health":"OperationalWithFailures",
       "Execution":"Active",
       "Failures":
       {
           "Failure":[
               {
                   "FailureID":"1",
                   "SourceID":"2",
                   "Detail":"detail"
               },
                {
                    "FailureID":"2",
                    "SourceID":"2",
                    "Detail":"detail2"
               }
           ]
       }
   },
   "Metrics":
   {
       "Metric":[
           {
               "MetricID":"1",
               "SourceID":"2",
               "CategoryID":"0",
               "DateTime":"2014-03-24T13:09:19.4216245Z",
               "Value":"3"
           }
       ]
   }
}
```

8. Use Cases

Failure in a common resource

Service A reports that its database dependency DB1 is not available. As service B also uses the same database, both service A and B owners receive a service dependency failure notification.



Opening a ticket by processing ManagementReports

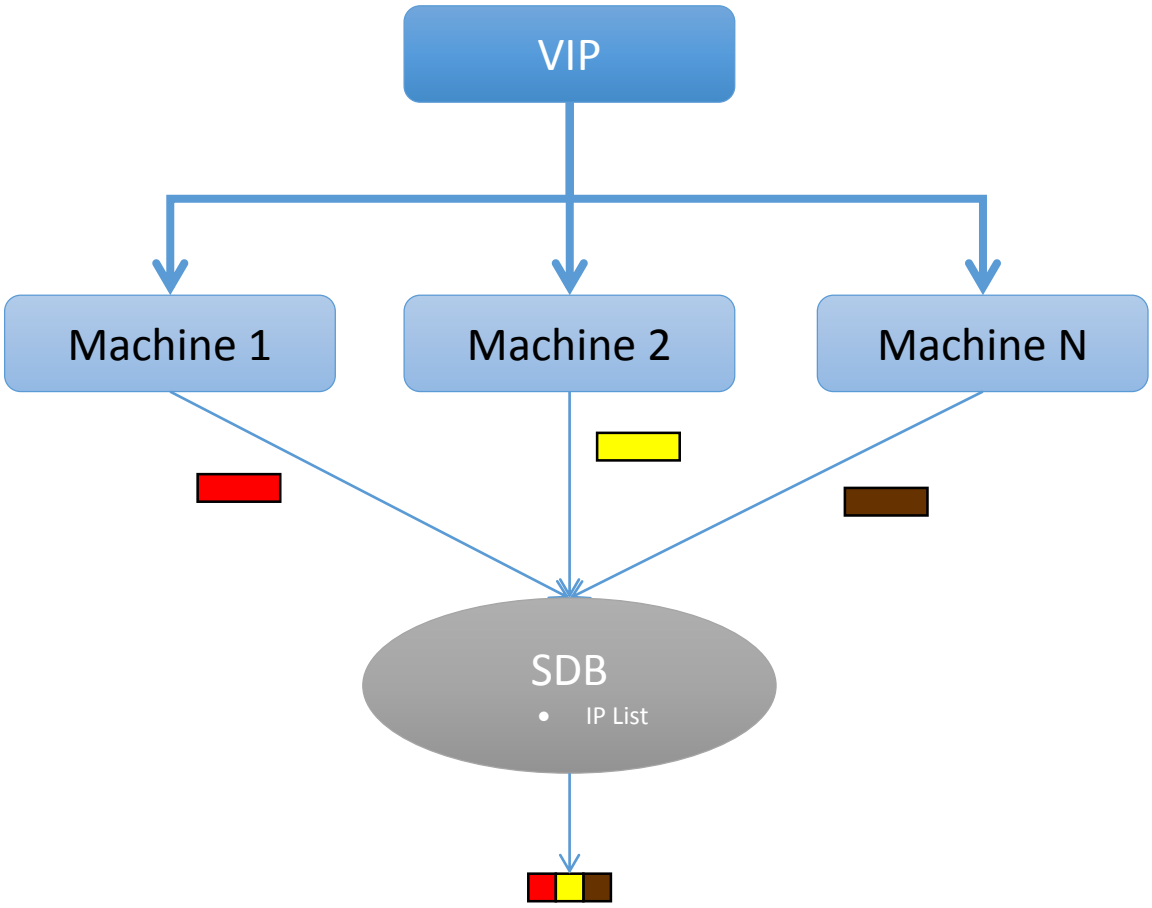
Aplication A publishes a ManagementReport that contains a User Interface error due to Service A inconsistent behavior. The ManagementReport is processed and a ticket is automatically opened in Problem Management and assigned to Service A owner.

Metric usage

The number of occurrences of an Error Code in a set of Premium Services has raised above X% for a given time period. A notification is sent to the services owners.

Simple Management Interface Design Example

It is common to want to implement the Simple Management Interface on a service that is in a farm, with different instances of the service.



In this example, Machine 1, 2 and 3 send a ManagementReport and the SDB has the ability to merge the three reports in just one report. To ease the merge of the reports it is possible to add a field 'AgentID' in the ManagementReport object and the Failure object. When Machine 1 sends a report, it add a field 'AgentID' in the ManagementReport object with value 'M1' for example (these identifiers should also be used according to the resource catalogue). The Machine 2 uses the value 'M2' for the 'AgentID' field. When the reports are merged, it is added the 'AgentID' field in each Failure indicating which service instance reported that failure.

9. Document Versions

Version	Date	Change Log
0.4	2014-05-09	Added Data Sources Section.
0.3	2014-03-24	Changed document structure.
0.2	2014-03-24	Added event-driven publishing details.
0.1	2013-12-02	Initial version.