hp OpenView
Service Quality Manager

Information Modeling Reference Guide

Edition: 1.2.02 (SP1)

for the HP-UX and Microsoft Windows Operating Systems

August 2005

Legal Notices

Warranty

Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office.

Restricted Rights Legend

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company

United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Copyright Notices

Trademark Notices

Adobe® and Acrobat® are trademarks of Adobe Systems Incorporated.

HP-UX Release 10.20 and later and HP-UX Release 11.00 and later (in both 32 and 64-bit configurations) on all HP 9000 computers are Open Group UNIX 95 branded products.

Java™ is a U.S. trademark of Sun Microsystems, Inc.

Microsoft® is a U.S. registered trademark of Microsoft Corporation.

Netscape is a U.S. trademark of Netscape Communications Corporation.

NMOS™ is a trademark of RiverSoft Technologies Limited.

Oracle® is a registered U.S. trademark of Oracle Corporation, Redwood City, California.

Oracle7™ and Oracle7 Server™ are trademarks of Oracle Corporation, Redwood City, California.

PostScript® is a trademark of Adobe Systems Incorporated.

Riversoft™ is a trademark of RiverSoft Technologies Limited.

UNIX® is a registered trademark of The Open Group.

Windows® and Windows NT® are U.S. registered trademarks of Microsoft Corporation.

X/Open® is a registered trademark, and the X device is a trademark of X/Open Company Ltd. in the UK and other countries.

All other product names are the property of their respective trademark or service mark holders and are hereby acknowledged.

Contents

Preface

This document provides reference information to help you create and implement a service with the HP OpenView SQM object model.

Intended Audience

This document addresses:

- Solution architects
- Service designers

Prerequisite Reading

This document assumes that you have read the *OpenView SQM Overview*.

Associated Documents

The HP OpenView SQM documentation set includes the following documents:

- *OpenView SQM SLA Monitoring UI User's Guide*
- *OpenView SQM Service Designer UI User's Guide*
- *OpenView SQM SLA Administration UI User's Guide*
- *OpenView SQM Overview*
- *OpenView SQM Getting Started Guide*
- *OpenView SQM Information Modeling and Reference Guide*
- *OpenView SQM Installation Guide*
- *OpenView SQM Administration Guide*
- *OpenView SQM Reference Guide for Oracle Use*
- *OpenView SQM Datamart User's Guide*
- *OpenView SQM Reporting Customization and User's Guide*

Refer to the following document for useful reference information:

- *TeleManagement Forum Service Level Agreement Management Handbook, v 1.5*

Supported Software

The term UNIX is used as a generic reference to the operating system, unless otherwise specified.

The supported software referred to in this document is as follows:

Product Version	Operating System
OpenView Service Quality Manager 1.2	HP-UX 11.11 Windows XP

Typographical Conventions

`Courier` Font:

- Source code and examples of file contents.
- Commands that you enter on the screen.
- Path names
- Keyboard key names

Italic Text:

- Filenames, programs and parameters.
- The names of other documents referenced in this manual.

Bold Text:

- To introduce new terms and to emphasize important words.

Support

Please visit our HP OpenView web site at: HP OpenView

There you will find contact information as well as details about OpenView products, services and support.

The OpenView support area of the OpenView web site includes:

- Downloadable documentation
- Troubleshooting information
- Patches and updates
- Problem reporting
- Training information
- Support program information

Chapter 1

Introduction to the OpenView SQM Object Model

This chapter introduces the object model implemented by OpenView Service Quality Manager (SQM). It briefly revisits the Service Level Agreement (SLA) lifecycle and describes the OpenView SQM service object model.

1.1 Overview of the SLA Lifecycle

This chapter introduces the SLA lifecycle. The SLA lifecycle describes the main functional components needed to manage service levels and SLAs. It includes the following phases:

- **Service design phase**

  In the service design phase, you identify all the technical resources needed to provide an end-to-end service. You model each of these resources as a service component and describe the service components with a set of parameters. The service model allows you to use these parameters to evaluate the service quality you deliver.

  The result of this phase is a unique service class, broken down into service components that provide the relevant service quality parameters.

  The service design phase is a critical, stand-alone process that aims to create a common definition of services, service parameters, and service level objectives. These common definitions can then be shared between the service provider, users, suppliers, and partners.

- **Class of service definition phase**

  Once you have described a service and selected the service parameters used to evaluate the quality of service, you must define a standard class of service, or **service level**, for that service.

  A service level consists of a set of objectives defined for the service parameters. Service parameters are evaluated directly from measurements made in the service infrastructure, or computed from the evaluated parameters. Different service levels correspond to different objectives set for the same service parameters.

  Service levels do not need to be defined for a specific customer, but instead can be a part of a general service definition.

- **Service instantiation phase**

  Before the service instantiation phase, a negotiation and sales phase occurs that leads to an SLA contract signature for a particular service. Negotiation and sales is outside the scope of the SLA lifecycle.

In the service instantiation phase, a new service needs to have its quality monitored. For example, a service model needs to be instantiated when a customer buys a new service. Service quality management begins as soon as you launch the service.

This phase results in a service instance and triggers the collection of service quality parameters.

- **SLA creation phase**

  In this phase, the service provider creates an SLA for a specific service instance or a group of service instances, depending on the contract. The service provider also identifies the threshold values of the service level for each service quality parameter managed for the SLA.

  This phase results in an SLA that:

  Associates the service instance(s) with the service level and a customer (for customer SLAs).

  Associates the service instance(s) with the service level and an internal department (for operational SLAs).

- **Service monitoring phase**

  In this phase, the service provider monitors the service and the service level agreements in real-time.

  The goal of the service monitoring phase is to avoid SLA violations by rapidly responding to service degradations.

- **Service reporting phase**

  In the service level reporting phase, the service provider generates reports about service quality.

  This phase assesses service quality on a long-term basis (monthly or quarterly) to analyze service trends or to provide customer service quality reports.

After the generation of reports, the service provider can periodically assess the service quality being provided to a particular service and the overall service quality for all customers. Depending upon new goals and changing customer needs, the service provider can make changes to the service design, completing the SLA lifecycle.

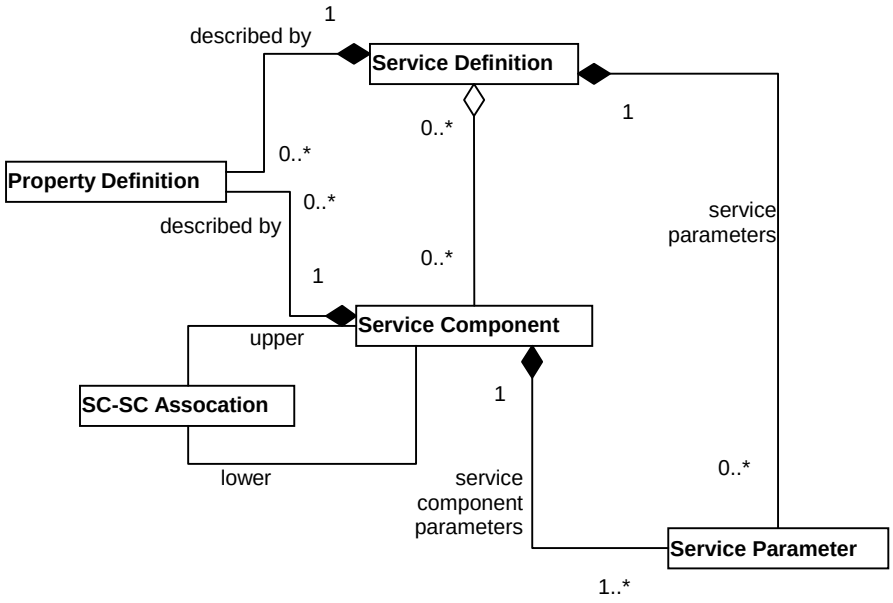
OpenView SQM provides the tools you need to easily manage each phase of the SLA lifecycle.

1.2 Introduction to the OpenView SQM Service Object Model

The OpenView SQM service object model represents a service as a collection of service components. A *service component* corresponds to hardware, software elements, or the underlying communications medium used by the service. Services and service components contain *service parameters*, values that are periodically updated and that help determine the quality of service (from either a customer or a network operator perspective).

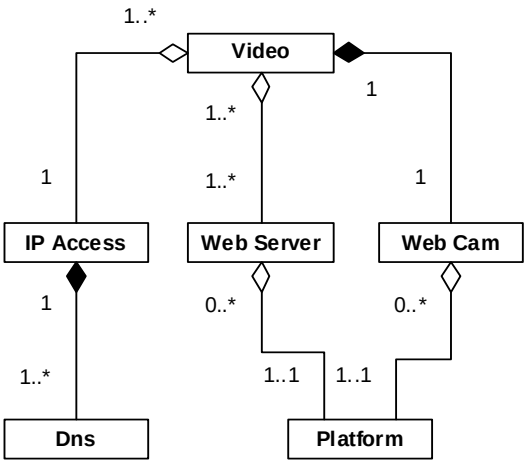
Figure 1 illustrates in UML the service object model supported by OpenView SQM.

Figure 1 OpenView SQM Object Model



You can use this generic service object model to model any service. For example, an ISP wants to create a new video service that allows subscribers in the United States and Europe to download movies over a wireless LAN and watch them on an iPaq terminal. This video service is composed of a Video Server and several Web servers (for load balancing). The ISP models this service as described in Figure 2.

Figure 2 Example Object Model for a Video Service



You can model any type of relationship between service components. To describe these relationships, OpenView SQM uses cardinality, or a number that indicates how many instances of each component object can be present.

The OpenView SQM repository manager stores and manages the service object model. For more information about the database and the architecture of OpenView SQM, refer to the *OpenView SQM Overview*.

Chapter 2

Object Model Concepts

This chapter describes the elements of the service object model, including the following:

- Section 2.1 Service Parameters
- Section 2.2 Service Properties
- Section 2.3 Service Components
- Section 2.4 Service Instances
- Section 2.5 Object Identifiers
- Section 2.6 Data Feeders and Data Acquisition
- Section 2.7 Expressions
- Section 2.8 Service Level
- Section 2.9 Service Level Agreements
- Section 2.10 Service Collection Management

2.1 Service Parameters

The HP OpenView SQM service object model defines service parameters that it measures globally or for specific customers.

Parameters can depend on customers as follows:

- Global parameters, which measure the quality of service without any consideration of particular customers. For example, a CPU load parameter measures the performance of the host component for all customers.
- Customer parameters, which measure the quality of service for a given customer (organization or corporation) or for a specific member of this organization (a user or a subscriber). The value of a customer parameter is unique for each customer. For example, a video player parameter that gives the number of retransmitted packets contains a value that applies to a particular subscriber.

You can provide the following additional information about the parameter using the “Category” attribute:

- Gauge, such as a minimum and maximum value that fluctuates.
- Cumulative counter. We recommend that you avoid this type of measure and instead convert counters into rates when possible.
- Percentage.
- Rate, which provides value for a unit of time.

You have the option to organize service parameters in the following logical partitions:

- QoS metrics (performance, errors, availability)
- State
- Usage
- Characteristics
- Compliance

2.1.1 Primitive Data Types of Parameters

To simplify the object model and service adapters interface, OpenView SQM uses only the primitive data types listed in Table 1.

Table 1 Primitive Data Types

Data type	Size	Description	Example
Display string	4000	Printable string (can be multiple lines)	“Mr. Smith HP Computer Corp”
Integer	Signed 64b	Signed integer	123
Enumeration	Signed 64b	Open enumeration definition	Off or “4”
Real	Signed 64b	Signed Float	-1E4 1267.43233E12 12.78e-2
Relative Time	Signed 64b	Number of milliseconds	4623767 (stands for 1h 17min 03sec 767mil)
Absolute Time	ISO 8601	*CCYY-MM-DD*T*hh:mm:ss.s* Only UTC notation is supported.	1999-05-31T13:20:00.234

2.1.1.1 Enumeration

An enumeration is defined as follows:

```
<sc:EnumDatatypeDef name="OperState">
              <sc:EnumElt name="Enable" value="1"/>
              <sc:EnumElt name="Idle" value="2"/>
              <sc:EnumElt name="Disable" value="3"/>
</sc:EnumDatatypeDef>
```

Any integer value is legal, even if there is not a literal definition defined for this value. You can also use undefined values, such as `"6"`. OpenView SQM displays these values in the user interfaces.

We recommend that you provide a name for each enumeration definition, even if version 1 of OpenView SQM does not support type sharing.

2.1.2 Parameter Binding Types

The OpenView SQM object model defines two kinds of parameters:

- Service quality measurements, which OpenView SQM gathers directly from service adapters.
- Service quality parameters. These parameters can be **collected** directly from a measure or **computed** by OpenView SQM from other service quality parameters.

OpenView SQM includes predefined expressions for computing parameters, such as minimum, maximum, and average. You can create your own expressions using the following:

- Java for the calculation expressions used to compute service quality measurements.
- PL/SQL for the calculation expressions used to compute service quality parameters.

Service parameters are interdependent. For example, an upper service parameter, such as a service key quality indicator (KQI), can be computed from lower service parameters, such as service key performance indicators (KPI). This computation can also occur within the same service component.

Refer to section 2.7 for more information about calculation expressions.

2.1.3 Parameter Visibility

To perform calculations using expressions, the HP OpenView Service Designer UI adds some parameters that act as auxiliary calculation variables. Refer to section 2.7 for more information about calculations.

When you declare these parameters as “not visible”, by default the HP OpenView Service Designer UI hides them because they are considered private parameters. Therefore, these parameters are not displayed by the Service Administration, Service Monitor and Service Reporting UIs.

The calculation engine never exposes the associated parameter values.

2.2 Service Properties

The Service Designer UI can define some attributes called **properties** to describe instances of the following objects:

- Service components
- Services and service groups
- Data feeders

A property receives a value when you first create an instance of the object.

Following are some examples of properties:

- Location
- Inventory Identifier

2.3 Service Components

A **service component** corresponds to a physical element (such as hardware) or a logical element (such as the underlying communications medium or a business process) used to implement a service.

You can create two types of service component definitions:

- Shared definitions. These shared definitions act as a template that other services can use. OpenView SQM ensures that changes made to the definition are applied to all the service definitions that share it.
- Private definitions that you associate (directly or indirectly) with a service definition.

The remainder of this section describes sharing service component definitions, sharing service component instances, and controlling the visibility of service components.

2.3.1 Sharing Service Component Definitions

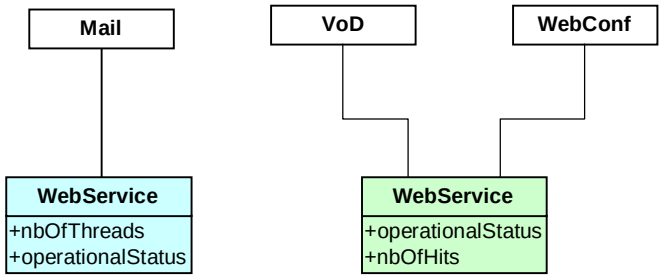
When several service definitions share a service component definition, each service contains a copy of the service component definition. OpenView SQM considers the first service component definition as the master copy. When you modify the definition of shared service components, OpenView SQM changes the other involved service components automatically and transparently.

Figure 3 illustrates an example of two services, Video On Demand (VoD) and WebConf, which share a component definition for a WebService component. When OpenView SQM loads the WebService component of the VoD service, it checks that the definition corresponds to the master definition managed by the WebConf service.

Note

A “private” definition of the WebService component with different component parameters can be used by another service, such as the Mail service.

Figure 3 Example of a Shared Component Definition



2.3.2 Sharing Service Component Instances

A service component can have several parent service component instances (upper cardinality greater than 1), belong to several service instances, or both.

OpenView SQM can share instances in the following two ways:

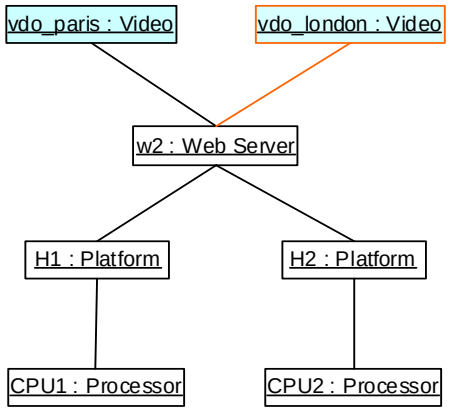
- Locally, between instances of the same service definition.
- Globally, between service instances of different service definitions.

The following sections provide examples of each type of instance sharing.

2.3.2.1 Local Sharing of Service Component Instances

Figure 4 illustrates an example where the Lyon and Paris instances of the Video service locally share the Web server instance.

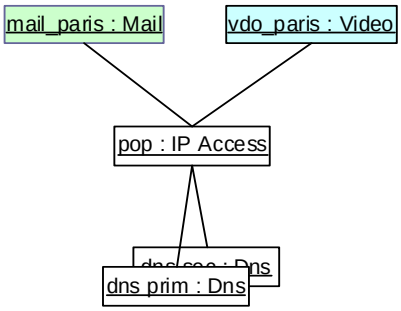
Figure 4 Example of a Locally Shared Component Instance



2.3.2.2 Global Sharing of Service Component Instances

You can share service component instances with service instances that have a different definition. This type of sharing is called “global,” and only occurs between service component definitions that share service definitions.

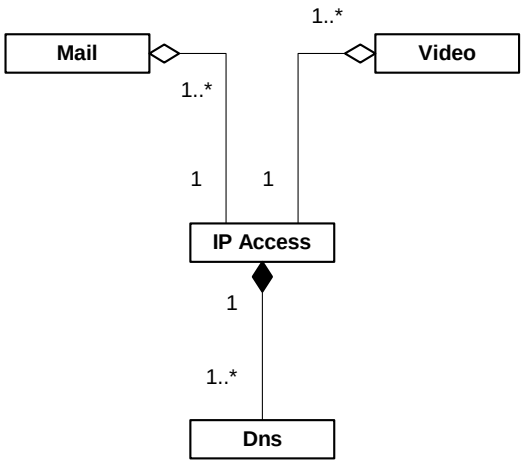
Figure 5 Example of Global Service Component Sharing



OpenView SQM automatically shares all of the service component instances of the shared service component.

Figure 6 illustrates how the DNS component instance is automatically shared between the Video and Mail services.

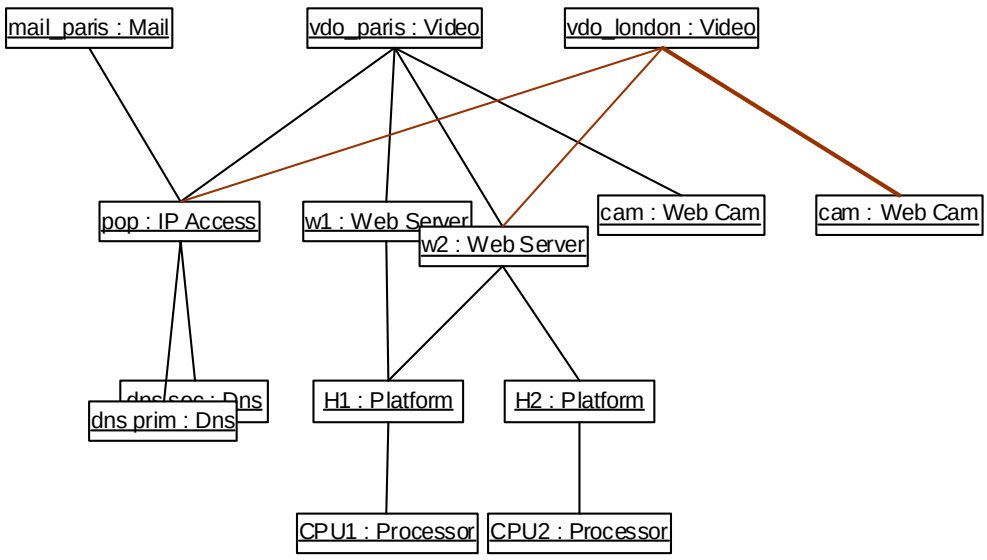
Figure 6 Sharing of Service Component Instances



2.3.2.3 Example of Component Sharing

Figure 7 illustrates two potential instances of the Video service defined in Section 1.2.

Figure 7 Instances of the Video Service



The instances in Figure 7 are shared as followed:

- The WebCam instances are not shared.
- The IP Access component is shared globally by both instances of the Video service and the Mail service.
- The Platform instances are shared locally by the two instances of the same Web server definition.

2.3.3 Service Component Visibility

Service components can be visible to the service customer or hidden. You might hide service components because they are a part of the infrastructure (such as network elements and resources) or because they are used as part of a calculation and are too complex or irrelevant to the service operator.

You can set component visibility using an attribute of the service component definition.

2.4 Service Instances

OpenView SQM can monitor two types of service:

- Customer services: a service provided to several customers. The measures performed are specific to each customer, such as measuring ADSL access.
- Network services: a service managed globally. All of the parameters in the service definition of a network service are related to global parameters, such as a short message service (SMS).

2.5 Object Identifiers

A definition or instance of an object is identified by its name. The "Label" attribute is used to define and display a more user-friendly name.

The naming of definition objects is structured. For example, a component is unique in the scope of the service definition.

Unlike definitions, all instances are globally identified in a flat naming space.

Table 2 defines how OpenView SQM uniquely identifies object definitions.

Table 2 OpenView SQM Object Identifiers

Object	Identifier
Service Definition	<Name>
Service Component	Service Def=<Name> + Component=<Name>
Parameter	Service Def=<Name> + Component=<Name> + Parameter=<Name>
Service Level	<Name>
Component Service Level	Service Level=<Name> + Component Service Level=<Name>
SLO	Service Level=<Name> + Component Service Level=<Name> + SLO=<Name>
Threshold	Service Level=<Name> + Component Service Level=<Name> + SLO=<Name> + Threshold=<Name>
Service Instance	<Name>
Service Group	<Name>
Service Component Instance	<Name>
Customer/Operation	<Name>
SLA	<Name>
Data Feeder Definition	Data Feeder Def=<Name> + DFD=<Version>

Object	Identifier
Data Feeder Instance	Data Feeder Def=<Name> + DFD=<Version>+MRP=<Name> or DFI=<Short ID>

2.5.1 Naming Rules

For all the Object Identifiers listed in Table 2 above, the <Name> must respect the following rules:

- The first character is one of a-z or A-Z.
- The other characters can be any of a-z, or A-Z, or [0-9] or '-', or '_', or ':'.
- The maximum number of characters for the Parameter <Name> is 12.
- The maximum number of characters for all the other <Name> is 16.

2.6 Data Feeders and Data Acquisition

HP OpenView SQM service adapters gather performance data and fault statistic data from various services by using periodic polling mechanisms or by receiving spontaneous events (attribute values or state changes).

A service adapter exposes one or more data feeders that correspond to potential logical measurement points existing in the underlying service infrastructure and the type of information collected. Each data feeder models service resources by defining one or more data feeder parameters.

The remainder of this section describes what a data feeder definition contains, how data feeder instances collect data, data feeder identifiers, and how data feeders bind.

2.6.1 Data Feeder Definition

A data feeder definition contains a logical description of parameters that a given data feeder exposes. The definition is characterized by the following:

- A version (default v1_0).
- A set of data feeder definition parameters. These parameters are used to build service component measurement parameters. Like service component parameters, data feeder parameters are either global or customer specific.
- A measurement reference point (MRP) naming scheme. The MRP naming scheme describes how OpenView SQM names the measurement point (for example, by concatenating data feed property values with fixed strings).

2.6.2 Data Feeder Instance

A data feeder instance collects parameters (such as usage information, errors, and performance) for a specific MRP. When a data feeder defines customer parameters, it measures the QoS perceived by all users and subscribers QoS for a particular MRP.

For information about how the data feeder parameters are associated with service component parameters, refer to Section 2.6.2.3.

Data feeders are created as follows:

- They are pre-registered by service operators. For example, they are created when service adapters or service elements are not yet deployed.
- They are discovered automatically by the service adapter, during a process called declaration.

The following sections illustrate how to identify an MRP and define an MRP naming scheme.

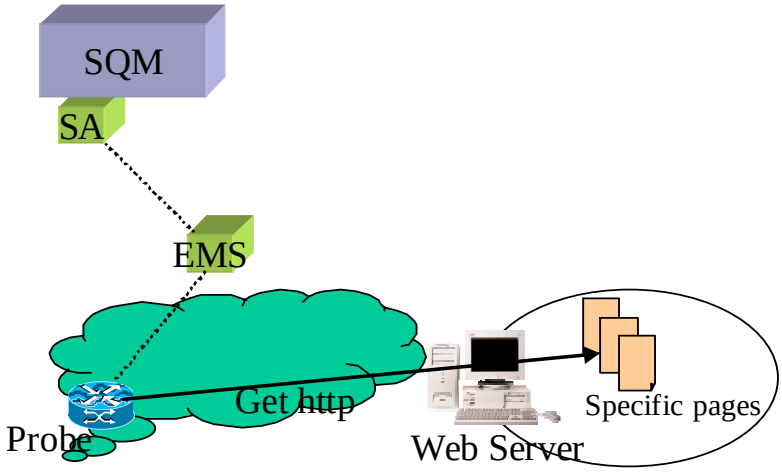
2.6.2.1 Measurement Reference Point

The MRP is a static, logical point in the service infrastructure. It is usually an interface between two elements of the service value chain (such as a radio domain and an XGSN domain).

The MRP does not specify which service adapter is used to perform a measure. It also does not specify the type of information collected, such as performance or fault statistic parameters.

Figure 8 illustrates a typical case, where a service adapter (SA) interfaces with an element manager (EMS) to collect performance data for a given customer service access point to a Web service. The MRP is the "perceived customer QoS from this access point to a specific Web service".

Figure 8 Example of a MRP



The MRP is identified by the following:

- IP address of the probe
- Tested Web server
- Specific URL
- Several data feeders can be modeled and defined for this specific MRP:
- IP performance (transit delay, jitter, and so on)
- HTTP performance (bandwidth, and so on)
- The MRP naming scheme is composed of data feeder properties and fixed strings.

For example, a naming scheme is defined as follows

```
<URL> + _Of_ + <server_hostname> + _From_  +
<probe_location>
```

The resulting MRP name follows:

```
"online-shopping_Pf_www.hp.com_from_Paris"
```

2.6.2.2 Data Feeder Instance Identification

A data feeder instance is identified by one of the following:

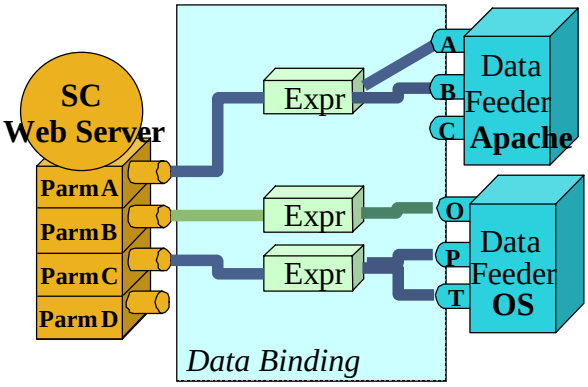
- Tuples: the data feeder name, the data feeder version, and the MRP name.
- A short name called the data feeder identifier such as `char(16)`. This identifier is generated automatically from the tuples by the service adapter's configuration tools.

2.6.2.3 Data Feeder Binding

The connection of one service component parameter to one or more data feeder parameters is called data feeder data binding. Any service component in the service component of a service can be given a value by one or several data feeders.

Figure 9 illustrates the association of two data feeder definitions with a service component definition.

Figure 9 Associating Service Components and Data Feeder Definitions



All data feeder parameters are not necessarily used to provide a value for a service component parameter.

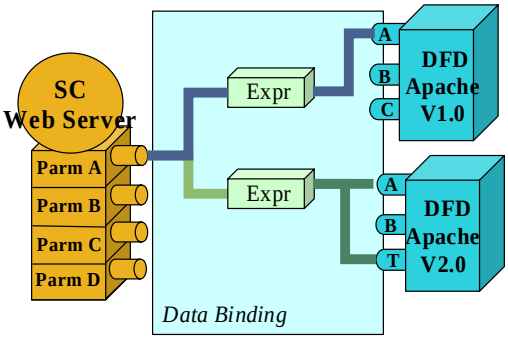
For example, in Figure 9 Parameter C of the Apache data feeder is not used.

In the context of a service definition, only one binding can exist between a service component parameter and a data feeder definition.

Note that it is possible to define different bindings for several versions of a data feeder definition.

For example, in Figure 10 two bindings are defined for two different versions of the Apache data feeder definition.

Figure 10 Data Binding for Duplicate Data Feeder Definitions

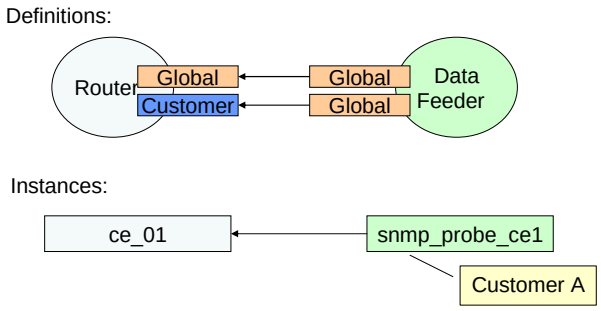


The measurement type (global or customer related) of a service component parameter must be the same for all the data feeder parameters used to compute it.

NOTE: There is an exception to this rule.

Because a data feeder instance can be dedicated to one customer, one or more global data feeder parameters can be used to compute a customer-qualified parameter.

For example, a data feeder definition is defined to model a router and provides global indicators for each port. When a port is dedicated to a customer, you can use the data feeder definition to bind it to a customer related service component parameter.

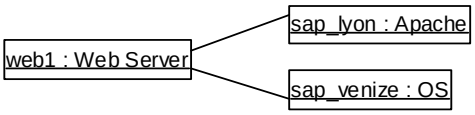


2.6.2.4 Instance Binding

A service component instance can be associated to only one instance of the same data feeder definition.

Figure 11 illustrates how an instance of the Web server service component is associated with two data feeders (from two different data feeder definitions).

Figure 11 Associating a Service Component with Data Feeder Instances



Service adapters can automatically declare data feeder instances. How the data feeder is defined depends on the configuration and environment of the service adapter.

When you plug a new service adapter into OpenView SQM, it automatically does the following:

- Declares the data feeder definitions, including parameters, version, and characteristics.
- Declares data feeder instances (DFI discovery).

By default, a newly declared data feeder instance does not collect any information (the administrative state is locked). OpenView SQM determines whether to enable or disable data collection using the SLA monitoring state and the service instance administrative state.

2.7 Expressions

You can use expressions in OpenView SQM for three purposes:

- Data feeder binding. Expressions are used to map one or more parameters from a service adapter to a parameter of a service component (service quality measurements).
- Service component binding. Expressions compute service quality parameters from one or several parameters.
- Buffering operations. Expressions are used to define how the calculation engine will elect a value from a list of values a parameter received during a calculation interval.

Expressions describe how to calculate the service parameter values from the service quality parameter data collected. You can use two types of expressions:

- Simple expressions, which are used for parameters coming from one instance. Simple expressions can be used for data feeder binding operations and service quality parameter binding.
- Aggregation expressions, which are used for parameters coming from one or more instances. Aggregation expressions can be used for service quality parameter binding and buffering.

OpenView SQM provides a set of pre-defined expressions for basic calculations. However, if you need an expression not provided by OpenView SQM, you can implement custom expressions in PL/SQL or Java (restricted to data feeder binding).

These custom expressions can be reused elsewhere in the same service or in another service. OpenView SQM accesses these expressions by reference, meaning that you do not need to duplicate code.

The remainder of this section describes each type of expression in detail as well as describing calculation rules, data type casting, store and forward mode, and error handling.

2.7.1 Simple Expressions

Simple expressions make calculations using parameters from one service component instance. This section describes the following types of simple expressions:

- Component binding expression
- Data feeder binding expression

2.7.1.1 Component Binding Expressions

You can use simple component binding expressions to:

- Calculate a service parameter of an upper service component from one or more parameters of a directly related lower service component instance.
- Calculate a service parameter from one or more parameters within the same service component instance.

Component binding expressions are written in PL/SQL and have the following format:

f(p1, …, pn) returns an output parameter

Following is an example of a component binding expression:

```
CREATE OR REPLACE FUNCTION sum_2Int (a NUMBER, b NUMBER) RETURN NUMBER IS
BEGIN
```

```
res := a + b;
RETURN res;
END;
/
```

2.7.1.2 Data Feeder Binding Expressions

You can use simple data feeder binding expressions to calculate measurement parameters from the data collected by data feeders. These expressions are written in Java.

Data feeder binding expressions have the following format:

f(x1, …,xn, version) returns one value (i.e. output parameter)

Each argument in the expression represents an individual argument of the function with the proper data type. An additional argument provides the version of the data feeder.

Table 3 Mapping between DFD parameter type and argument type

Data type	Argument type
String	string
Integer	long
Real	double
Relative Time	long
Absolute Time	java.util.Date

Following is an example of a data feeder binding expression:

```
package com.compaq.temip.servicecenter.expressions;

public class ASimpleCalculationExpression {

    public static double aSimpleCalculationExpression
       (int a,
       double b,
       String dfdVersion) {
         double diff = a-b;
         return diff;
    }
}
```

2.7.2 Aggregation Expressions

OpenView SQM supports instance aggregation expressions that calculate one service parameter for a service component instance using one or more parameters of several service component instances.

Aggregation expressions are written in PL/SQL and have the following format:

f(coll p1, …, coll pn) returns a unique value (i.e. output parameter)

Following is an example of an aggregation expression:

```
CREATE OR REPLACE FUNCTION avg_aggr_Float (a float_coll) RETURN NUMBER IS
    res NUMBER :=0;
    BEGIN
       FOR I IN a.FIRST..a.LAST
       LOOP
```

```
            res := res + a(i);
        END LOOP;

        res := res/a.COUNT

        RETURN res;
    END;
    /
```

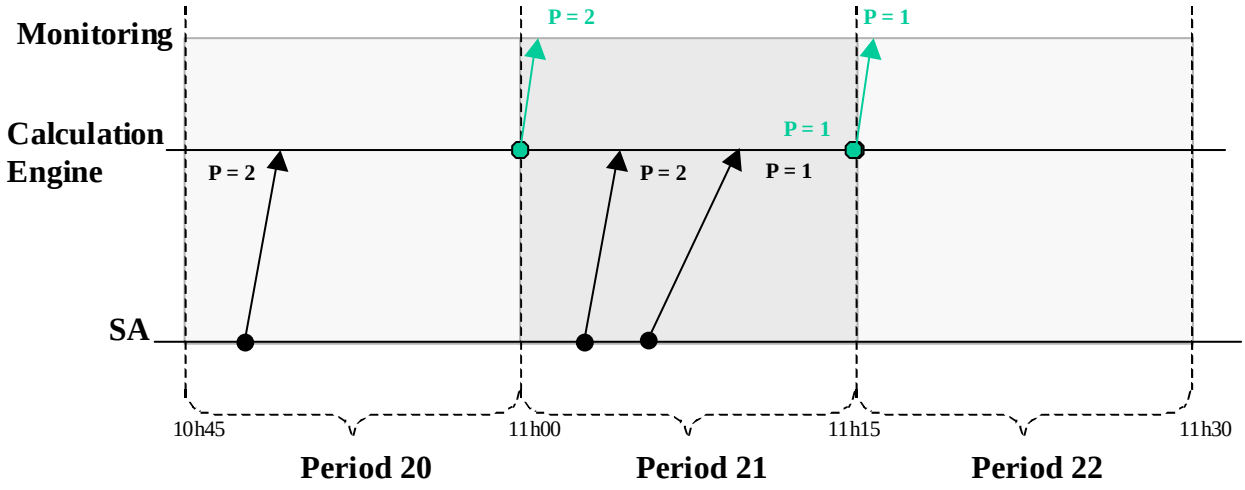
2.7.2.1 Buffering Expressions

When the calculation engine receives several collected values between two calculation runs (for example, during Period 21 illustrated in Figure 12), it elects the value it will use for its computation with a buffering expression. The buffering expression is an instance aggregation expression (such as average) defined for each parameter in the object model.

By default, the buffering expression is “Last”.

Figure 12 illustrates how buffering expressions elect values.

Figure 12 Using Buffering Expressions



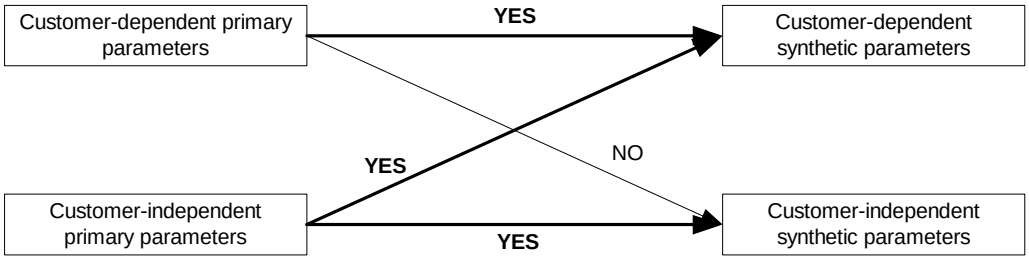
The value of parameter `z` is computed from the values of `x` and `y`. When several values of `x` are received during an interval of `t1` to `t2`, the engine needs to elect one value to use in computing `z`. The buffering aggregation expression defined for `x` is used to elect a value (for example, the worst value) at `t3`.

2.7.3 Calculation Rules

OpenView SQM enforces a set of calculation rules to ease the evolution of a service definition and provide access to previously stored data and generated reports.

Depending upon whether or not a parameter is provided by a customer, some expressions are not supported. Figure 13 illustrates whether or not expressions are supported for customer independent and customer dependent parameters.

Figure 13 Summary of Supported Expressions



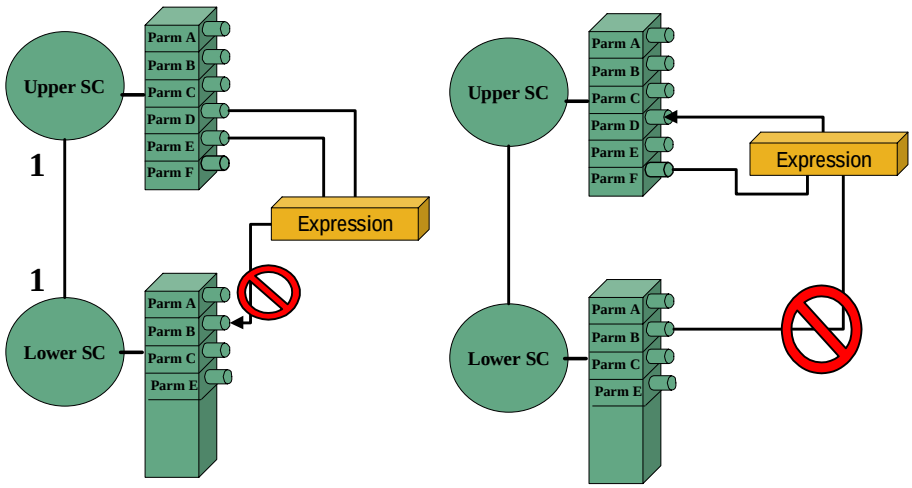
Expressions are not supported when a customer-independent parameter has to be computed from at least one customer-dependent parameter.

Expression must always follow one association between two service components and its direction. In other words:

- Expressions cannot have input parameters belonging to several service components.
- You cannot specify expressions between service components that have no relationship.
- Expressions must be defined between parameters of the same service component or two consecutive service components. Expressions cannot “jump” services.
- The output parameter is always at the upper service component level (that is, the service component closest to the service level).
- An expression must be specified on service components that are related.
- An expression cannot have output computed from “higher” inputs.

Figure 14 illustrates incorrectly specified component binding.

Figure 14 Inappropriate Component Binding



The common practice to cope with these constraints consists of adding some parameters dedicated to the calculation (visibility flag to False).

2.7.4 Data Type Casting

To ensure that binding expressions are easily reusable, OpenView SQM uses the type casting conventions summarized in Table 1.

For example, an expression that works with an input float parameter can be invoked with an Integer parameter.

Table 4 Data Type Casting

	integer	enum	float	absTime	relTime	string
Integer						
Enum						
Float						
AbsTime						
RelTime						
String						

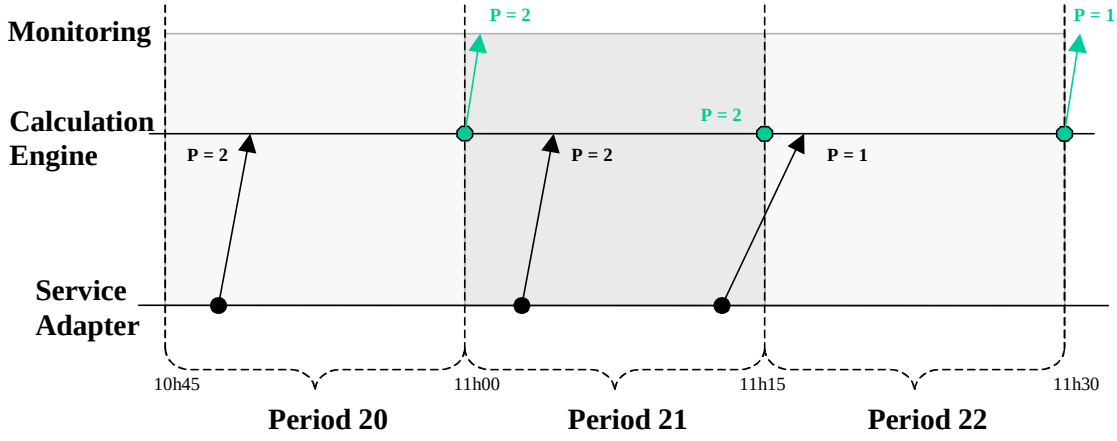
Mapping OK
Mapping not allowed
Warning

2.7.5 Auto Propagation Mode

Figure 15 illustrates the processing performed by the calculation engine for collected service parameters.

When a measure is received, it is first stored and then, at the end of a period (in this example, Period 20), the calculation engine publishes changes.

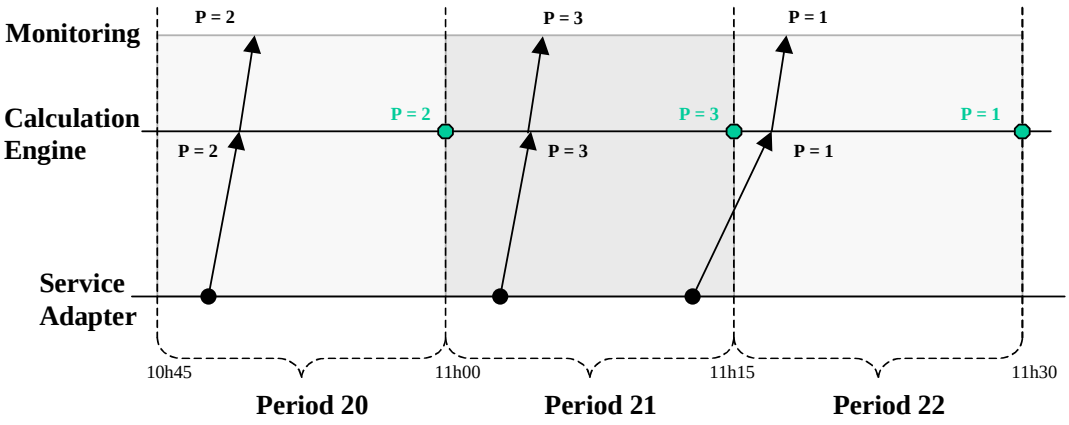
Figure 15 Auto Propagation After a Period Has Passed



To validate a critical parameter as soon as it is collected, you can publish the parameter immediately after it is stored by the calculation. You can use the "Automatic propagation flag" property of the parameter to define this immediate forwarding behavior.

As illustrated in Figure 16, the value of the parameter P is stored and published as soon as the calculation engine processes it.

Figure 16 Auto Propagation Immediately After Storage



Whenever a service quality parameter is computed from one immediate service quality measurement, the parameter will be recomputed the next time the expression is executed.

2.7.6 Collection and Computation Error

When a service adapter encounters a collection error a special value, `noValue,` is assigned to faulty parameters.

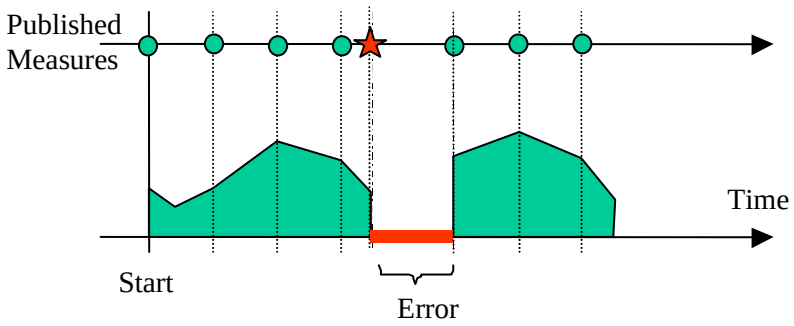
Errors also occur when the calculation of a collected parameter cannot be performed (one input parameter is missing) or when a parameter cannot be computed by the calculation engine (internal error encountered by the expression).

The administrator of OpenView SQM can control the propagation of such errors in the object model using the “NoValue Propagation” flag. These flags give you control over propagation between the following:

- From data feeder instance parameters to service quality measurements.
- From service quality measurements to the resulting service quality parameters.

Figure 17 illustrates how OpenView SQM collects errors over time.

Figure 17 Example of Error Collection



2.8 Service Level

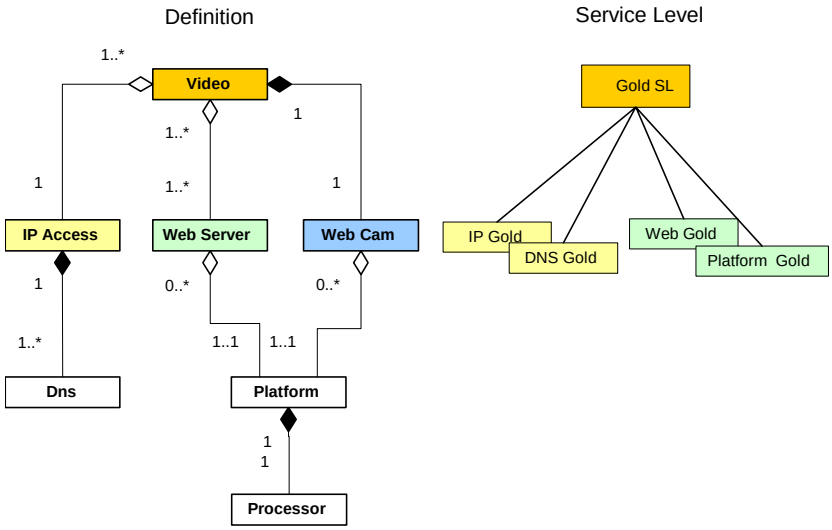
A service level is always defined for a given service definition. The following sections describe component service levels, service level objectives (SLO), objective thresholds, and example service level validations.

2.8.1 Component Service Levels

A service level is composed of one or more component service levels. The component service level defines the service level for each component.

Each component service level is defined under the service level (a flat model) as described in Figure 18. The service level hierarchy does not reflect the service definition hierarchy.

Figure 18 The Relationship between Definitions and Service Levels



2.8.2 Service Level Objectives

Once you have specified service parameters, you can add service level objectives. For example, a service designer specifies the “Transit Delay” parameter. The service administrator later adds a service level objective of “ < 60ms” to this parameter.

Each service level objective defines:

- A crossing type (up, down, equal, not equal, is valued). An objective is not met when the parameter value crosses the critical threshold.
- An action executor, action name, action on (action when quality of service decreases), action off (action when quality of service increases) and some additional action information (optional).
- Weight (optional). By default, the weight is 100%.

A service level objective is always associated with a service parameter. However, you are not required to specify an SLA for a service parameter. For example, if a service parameter belongs to a service resource (a non-visible service component), it should not have an objective.

2.8.3 Objective Thresholds

A service level objective is defined using the following thresholds:

- One violation threshold (mandatory).
- One violation clearance (optional).
- Several degradation thresholds (optional).
- One degradation clearance (optional).

You can associate each intermediary threshold with an action and additional information.

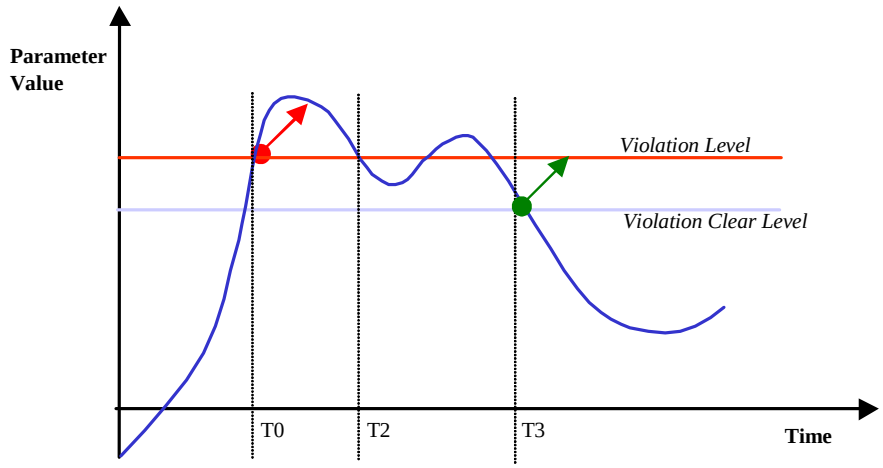
Each threshold is defined by the following:

- A threshold level (that is, a reference value).
- A threshold type (degradation or violation).
- An action executor, action name, action on (action when quality of service decreases), action off (action when quality of service increases) and some additional action information (optional).
- A service degradation factor (SDF) that varies from zero (no degradation) to 100% (service failure). Intermediate values characterize a degraded objective.

When a threshold is crossed, the triggered action is deduced from information defined at threshold level (when it is provided) and then from information defined at the SLO level. In others words, the action configuration defined at SLO level acts as a default value.

Figure 19 illustrates the benefits of defining a clearance threshold level. The service violation ends only when the parameter crosses the clear level, T3, and not T2.

Figure 19 Violation Clearance Level



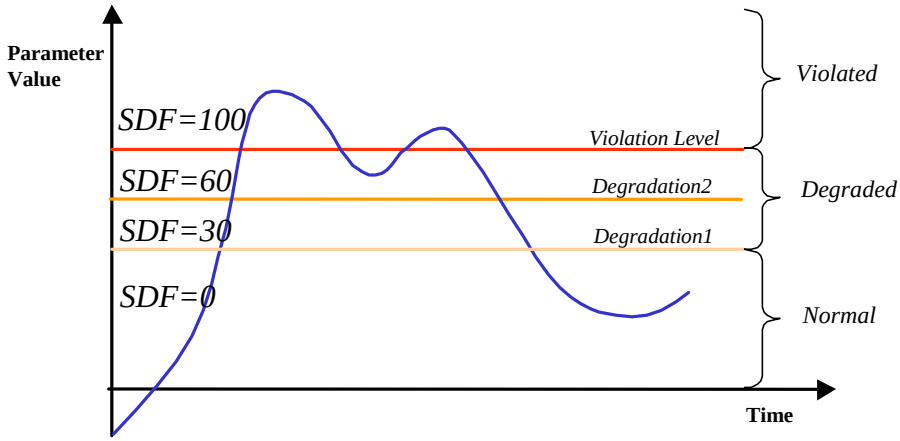
2.8.3.1 Service Degradation

The result when a parameter value is validated against a service level objective depends on the service degradation factor. This result is called the service objective status (SOS). The SOS provides a range varying from 0% (failure) to 100% (operational). Intermediate values indicate in what proportion the service has altered.

The user interfaces and datamart map the SOS to an elementary state of `OK`, `Degraded`, or `Violated`. Usually, a clear threshold equals 0%. A violation threshold is equal to 100%.

Figure 20 illustrates a service level objective with several degradation thresholds.

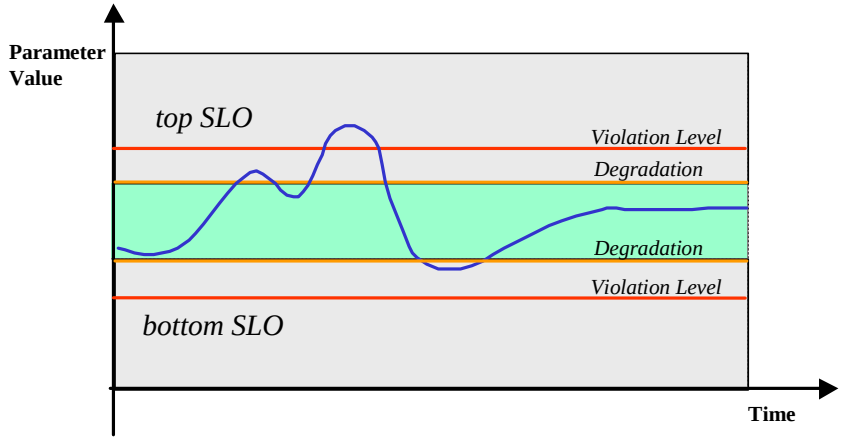
Figure 20 Service Degradation



2.8.3.2 Range SLO

To validate that a parameter value is compliant to up and down boundary conditions, the service designer can define two service level objectives as shown in Figure 21.

Figure 21 Setting a Service Level Objective Range



2.8.4 Sharing Service Levels

You can share service levels across multiple SLAs.

There is no concept of sharing component service levels in version 1.x of OpenView SQM. However, we recommend that you give the same name to duplicated component service levels.

2.8.5 Example Service Level Validation

This section describes an example of a service level objective and its validation. Table 5 contains the definition of the service level.

Table 5 Example Service Level

Name	TransitDelay	
Label	Transit Delay SLO	
Description	“Transit delay is still acceptable”	
Crossing Type	Up	
Threshold Levels		
	Name	Degradation_1
	Value	70
	Degradation Factor	0.5 (50%)
	Action Executor	**OVOServiceAlarm**
	Action Name	SendMessage
	Action On	Major
	Action Off	“”
	Action Information	“Early congestion detection”
	Name	Degradation_2
	Value	80
	Degradation Factor	0.7 (70%)
	Action Executor	**TrafficShaper**
	Action Name	DiscardLowPriority
	Action On	“”
	Action Off	“”
	Action Information	“ftp”
	Name	Degradation Clearance
	Value	40
	Degradation Factor	0
	Action Executor	**OVOServiceAlarm**
	Action Name	SendMessage
	Action On	“”
	Action Off	Normal
	Action Information	“Not More Problem”
	Clearance	True
	Name	Violation
	Value	90
	Degradation Factor	1 (100%)
	Action Executor	**OVOServiceAlarm**
	Action Name	SendMessage
	Action On	Critical

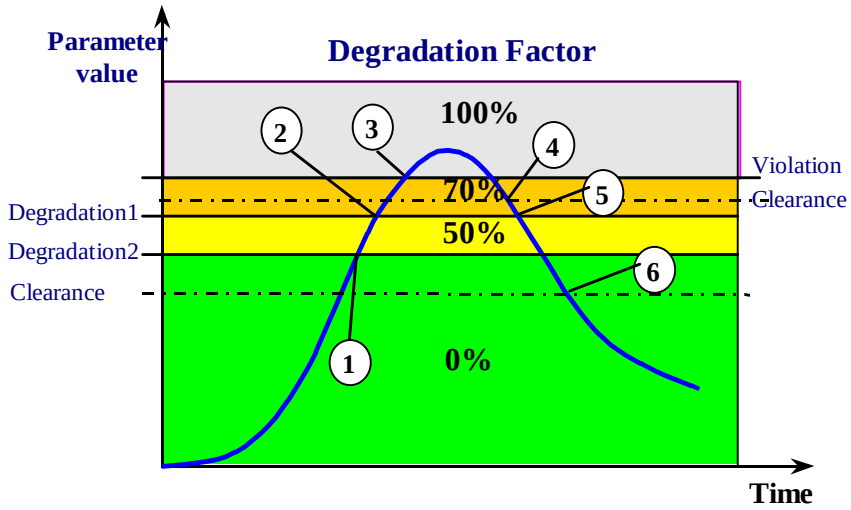
	Action Off	“”
	Action Information	“not acceptable response time”
	Name	Violation Clear
	Value	60
	Action Executor	**OVOServiceAlarm**
	Action Name	SendMessage
	Action On	“”
	Action Off	Major
	Action Information	“”
	Clearance	False

The Service Level Monitor sends the results of its validation in a single message. This message contains three parts:

- The list of triggered actions.
- The crossed thresholds, associated information (such as triggered actions), and information about the affected customers and SLAs.
- For each SLA affected, the SOS value that results.

Figure 22 illustrates the variation of a parameter value over time.

Figure 22 Example SLO Degradation



According to Figure 22, when the thresholds are crossed the Service Level Monitoring function publishes the following Service Status:

- Crossed threshold = Degradation_2
  QoS: Decreasing
  Degradation: Start
  Service Level Objective Status = 50%
  Triggered Action= (OVOServiceAlarm, Major , …)

- Crossed threshold = Degradation_1
  QoS: Decreasing
  Service Level Status = **30%**
  Triggered Action= (TrafficShaper, “DiscardLowPriority”, …)
- Crossed threshold = Violation
  QoS: Decreasing
  Violation: **Start**
  Service Level Status = **0%**
  Triggered Action= (OVOServiceAlarm, Critical , …)
- Crossed threshold = Violation Clearance
  QoS: Increasing
  Violation: **End**
  Service Level Status = **30%**
  Triggered Action= (OVOServiceAlarm, Major , …)
- Crossed threshold = Degradation_1
  QoS: Increasing,
  Service Level Status = **50%**
  Triggered Action= (TrafficShaper, “DiscardLowPriority”, …)
- Crossed threshold = Degradation_Clearance
  QoS: Increasing
  Degradation: **End**
  Service Level Status = **100%**
  Triggered Action= (OVOServiceAlarm, Clear, “No More Pb!”)

2.8.5.1 Making Service Objective Status Calculations

From the elementary parameter SOS, the monitoring function computes the resulting status of the following objects:

- Service components
- Service instances
- SLAs

The upper SOS percentage obtained for the parameters can be aggregated to get the global SOS for each object.

Each objective specified (whether service level objective or component service level) is given weight in the overall objective hierarchy. The weight of the objective determines the SOS at a parent objective level.

You can calculate the following service status objectives:

- Worst
- Weighted worst
- Weighted average
- Weighted sum

Figure 23 Example Weighted Average Formula

$$\text{SCI SOS} = \frac{\sum_{i=SLO\ 1}^{i=SLO\ n} (\text{SLO weight} * \text{Service Level Objective Status})}{\sum_{i=SLO\ 1}^{j=SLO\ n} (\text{SLO weight})}$$

$$\text{Service SOS} = \frac{\sum_{I=SLO\ 1}^{I=SLO\ n} (\text{SI Objective weight} * \text{SI SLO Status}) + \sum_{I=SCI\ 1}^{I=SCI\ n} (\text{CSL weight} * \text{SCI SOS})}{\sum_{I=SLO\ 1}^{I=SLO\ n} (\text{SI SLO weight}) + \sum_{I=SCI\ 1}^{I=SCI\ n} (\text{CSL weight})}$$

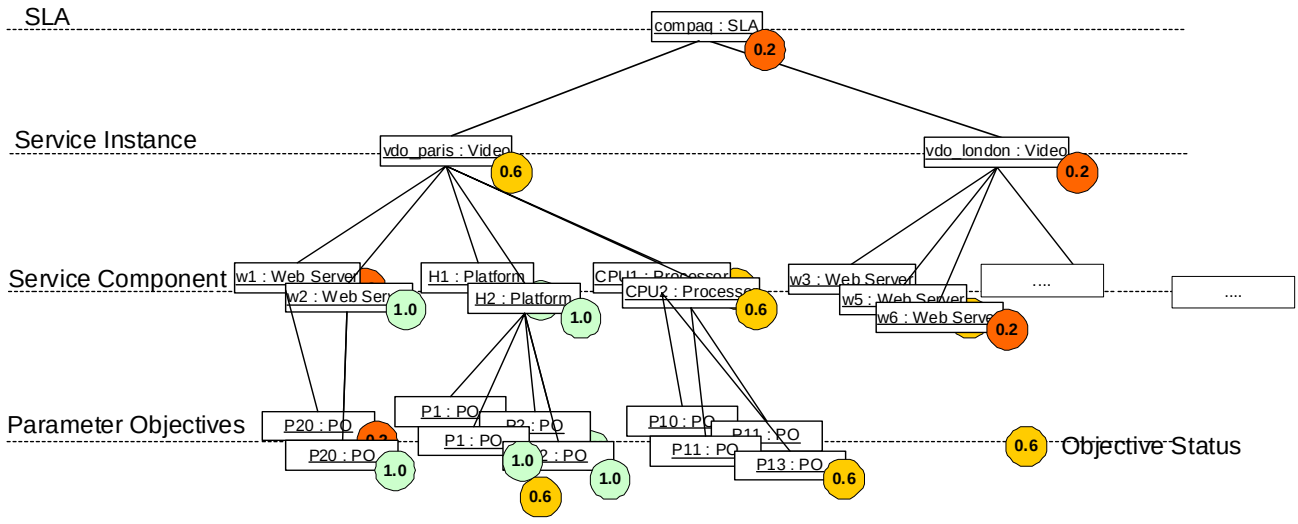
$$\text{SLA SOS} = \frac{\sum_{I=SI\ 1}^{I=SI\ n} (\text{weight} * \text{SI Objective Status})}{\sum_{i=SI\ 1}^{i=SI\ n} (\text{weight})}$$

The first schema in the example introduces the propagation of objective status from the parameter up to the service.

The second schema illustrates the propagation with instantiation information, using the Video Service example.

Figure 24 illustrates the objective hierarchy.

Figure 24 Objective Hierarchy



2.8.5.2 Examples of Worst Objective Calculation

Following are several examples of worst objective calculations.

Case 1: No Weight, a Service Component Instance is Down

The following table describes the service component weights and their status.

Service Component	Weight	Status
SC1	100%	0%
SC2	100%	100%
SC3	100%	70%

The service instance status is the status of the service component having the worst weighted degradation:

```
  SC1 = 1*(1-0)   = 1
  SC2 = 1*(1-1)   = 0
  SC3 = 1*(1-0.7) = 0.3
Selected service component is SC1:
      SI Status = 0% => Violated
```

Case 2: Major Service Component Instance is Down

The following table describes the service component weights and their status.

Service Component	Weight	Status
SC1	50%	0%
SC2	40%	100%
SC3	10%	100%

The service instance status is the status of the service component having the worst weighted degradation:

```
  SC1 = 0.5*(1-0) = 0.5
  SC2 = 0.4*(1-1) = 0
  SC3 = 0.1*(1-1) = 0
Selected service component is SC1:
    SI Status = 0% => Violated
```

Case 3: Minor Service Component Instance is Down

The following table describes the service component weights and their status.

Service Component	Weight	Status
SC1	50%	100%
SC2	40%	70%
SC3	10%	0%

The service instance status is the status of the service component having the worst weighted degradation:

```
  SC1 = 0.5*(1-1)   = 0
  SC2 = 0.4*(1-0.7) = 0.12
  SC3 = 0.1*(1-0)   = 0.1
Selected service component is SC2:
    SI Status = 70% => Degraded
```

Case 4: All Service Components have Weight = 0%

The following table describes the service component weights and their status.

Service Component	Weight	Status
SC1	0%	0%
SC2	0%	50%
SC3	0%	100%

In this special case, the statuses of the service components do not impact the service instance status (all weight are 0%).

```
SI Status = 100% => OK
```

2.8.5.3 Examples of Weighted Worst Objective Calculation

Following are several examples of weighted worst objective calculations.

Case 1: No Weight, a Service Component Instance is Down

The following table describes the service component weights and their status.

Service Component	Weight	Status
SC1	100%	0%
SC2	100%	100%
SC3	100%	70%

The service instance status is the weighted status the service component having the worst weighted degradation. It is calculated as follows:

```
SI Status = 1 – Max(1*(1-0), 1*(1-1), 1*(1-0.7))
          = 1 – Max(1, 0, 0.3)
          = 1 – 1 = 0% => Violated
```

Case 2: Major Service Component Instance is Down

The following table describes the service component weights and their status.

Service Component	Weight	Status
SC1	60%	0%
SC2	40%	100%
SC3	10%	100%

The service instance status is the weighted status the service component having the worst weighted degradation. It is calculated as follows:

```
SI Status = 1 – Max(0.6*(1-0), 0.4*(1-1), 0.1*(1-1))
          = 1 – Max(0.6, 0, 0)
          = 1 – 0.6 = 40% => Degraded
```

Case 3: Minor Service Component Instance is Down

The following table describes the service component weights and their status.

Service Component	Weight	Status
SC1	50%	100%
SC2	40%	70%
SC3	10%	0%

The service instance status is the weighted status of the service component having the worst weighted degradation. It is calculated as follows:

```
SI Status = 1 – Max(0.5*(1-1), 0.4*(1-0.7), 0.1*(1-0))
```

```
= 1 – Max(0, 0.12, 0.1)
= 1 – 0.12 = 88% => OK
```

2.8.5.4 Examples of Weighted Average Calculation

Following are several examples of weighted average calculations.

Case 1: No Weight, a Service Component Instance is Down

The following table describes the service component weights and their status.

Service Component	Weight	Status
SC1	100%	0%
SC2	100%	100%
SC3	100%	70%

The service instance status is calculated as follows:

```
SI Status = (1*0+1*1+1*0.7)/3
          = 56% => Degraded
```

Case 2: Major Service Component Instance is Down

The following table describes the service component weights and their status.

Service Component	Weight	Status
SC1	50%	0%
SC2	40%	100%
SC3	10%	100%

The service instance status is calculated as follows:

```
SI Status = (0.5*0+0.4*1+0.1*1)/1
          = 50% => Degraded
```

Case 3: Minor Service Component Instance is Down

The following table describes the service component weights and their status.

Service Component	Weight	Status
SC1	50%	100%
SC2	40%	70%
SC3	10%	0%

The service instance status is calculated as follows:

```
SI Status = (0.5*1+0.4*0.7+0.1*0)/1 =
          = 78% => OK
```

2.8.5.5 Examples of Weighted Sum Objective Calculation

Following are several examples of weighted sum objective calculations.

Case 1: No Weight, a Service Component Instance is Down

The following table describes the service component weights and their status.

Service Component	Weight	Status
SC1	100%	0%

SC2	100%	100%
SC3	100%	70%

The service instance status is the sum of the weighted status the service component in the range [0%, 100%]. It is calculated as follows:

```
SI Status = 1 – Sum(1*(1-0), 1*(1-1), 1*(1-0.7))
          = 1 – Sum(1, 0, 0.3)
          = 1 – 1.3 = 0% => Violated
```

Case 2: Major Service Component Instance is Down

The following table describes the service component weights and their status.

Service Component	Weight	Status
SC1	60%	0%
SC2	40%	80%
SC3	10%	50%

The service instance status is the weighted status the service component in the range [0%, 100%]. It is calculated as follows:

```
SI Status = 1 – Sum(0.6*(1-0), 0.4*(1-0.8), 0.1*(1-0.5))
          = 1 – Sum(0.6, 0.08, 0.05)
          = 1 – 0.73 = 27% => Degraded
```

Case 3: Minor Service Component Instance is Down

The following table describes the service component weights and their status.

Service Component	Weight	Status
SC1	60%	100%
SC2	40%	80%
SC3	10%	0%

The service instance status is the weighted status of the service component in the range [0%, 100%]. It is calculated as follows:

```
SI Status = 1 – Sum(0.6*(1-1), 0.4*(1-0.4), 0.1*(1-0))
          = 1 – Sum(0, 0.08, 0.1)
          = 1 – 0.18 = 82% => OK
```

2.9 Service Level Agreements

A service level agreement (SLA) specifies the service quality to be achieved using the service levels defined for the targeted service.

A customer uses the service through a set of instances, such as the instances located in a geographical region. You can gather these individual instances into a group that is then associated with an SLA. A group of service instances is called a service group.

There are two types of service level agreement:

- Customer SLAs, which are a contract between a service provider and a customer that specifies in measurable terms what the service provider provides to its customers.

- Operational SLAs, which the service provider uses internally to define requirements for everything from help desk services to network performance and availability, application performance and availability, and internal processes.

An SLA can monitor only one service definition.

Table 6 illustrates the types of service level agreements and how you can use them.

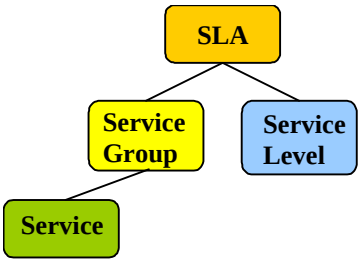
Table 6 Types of Service Level Agreements

SLA Type	Service Type	Monitoring type
Customer	Customer Service	Instance
Operational	Customer Service	Aggregation view. Only customer independent parameters constitute the aggregation View
	Network Service	Instance

2.9.1 Customer SLA

The customer uses the service through the instances of a service group. Therefore, the customer SLA consists of a service group (to know the set of instances involved) and a service level (to know the objectives) as described in Figure 25.

Figure 25 Components of a Customer SLA



Note

You can create an SLA without associating any service level. For example, you can specify the service level later in an SLA after you have identified the right objective thresholds.

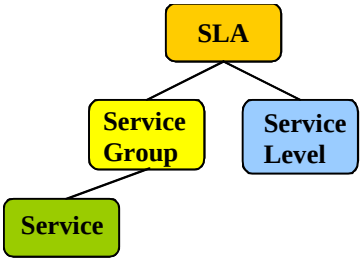
2.9.2 Operational SLA

Operational SLAs (also called OLAs) can be defined to monitor services independent from the customer. This means that service providers can set their own service levels on a service instance so that degradations are quickly detected before the customer encounters a problem.

You can create two types of operational SLAs:

- SLAs that monitor network services with no customer measures (monitored at the service instance level).

Figure 26 Operational SLA That Monitors the Service Instance



- SLAs that monitor only global parameters of a Customer service.

Figure 27 Operational SLA That Monitors Global Parameters

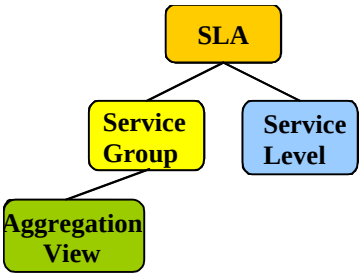
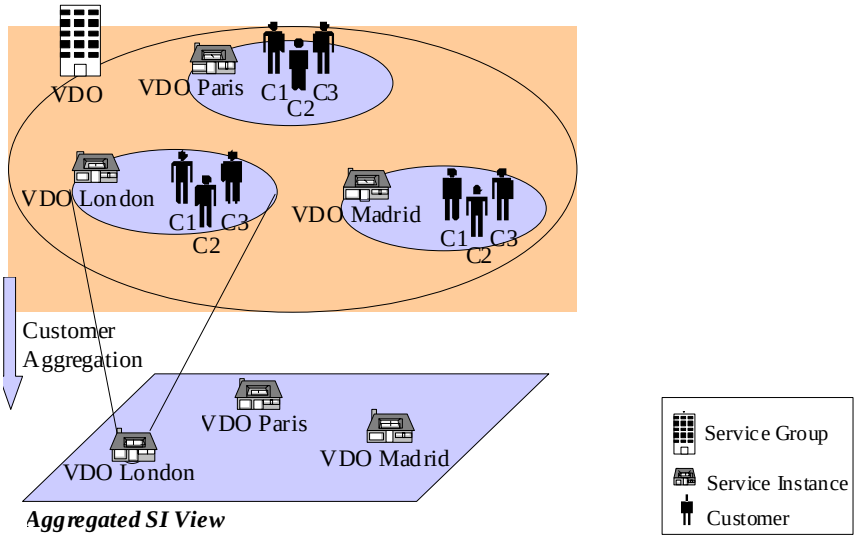


Figure 28 Aggregation View



2.9.3 Default SLA and Service Group

To ease the configuration for each service definition:

- A default service group is automatically provided with all existing service instances.
- A default SLA is associated to this service group. The SLA contains no specific service level.

2.9.4 SLA Compliance

OpenView SQM supports both SLA status and SLA compliance. For example, if a service has an availability objective defined as “99.5% daily availablity,” then the availability status refers to an instant availability value (99.0%), whereas the compliance refers to the daily average calculation (99.5% daily average). Therefore, over the defined period, the SLA compliance can meet the 99.5% daily average value although some of the SLA status measurements during that day might have been below 99.5%.

SLA compliance serves the goal of measuring the service availability. The SLA status is also of high interest to service operators because it alerts service operations staff in real time to service quality issues. Together with the ability to set multiple thresholds on service levels, the visualization of the SLA status helps accelerate problem resolution times, since issues can be prioritized and addressed well before SLA compliance is at risk.

For OpenView SQM version 1.x the real time compliance violation levels and monitoring are only supported when there is only one SLA for a given service instance used by a given customer. For example, Vdo lyon can be used by several customers, but for MyCorp only one SLA is applicable for this delivered instance.

The compliance is not calculated for Aggregation Views.

2.9.4.1 Compliance Violation Level

The compliance of service quality parameters is calculated based on the comparison of a parameter against the violation threshold.

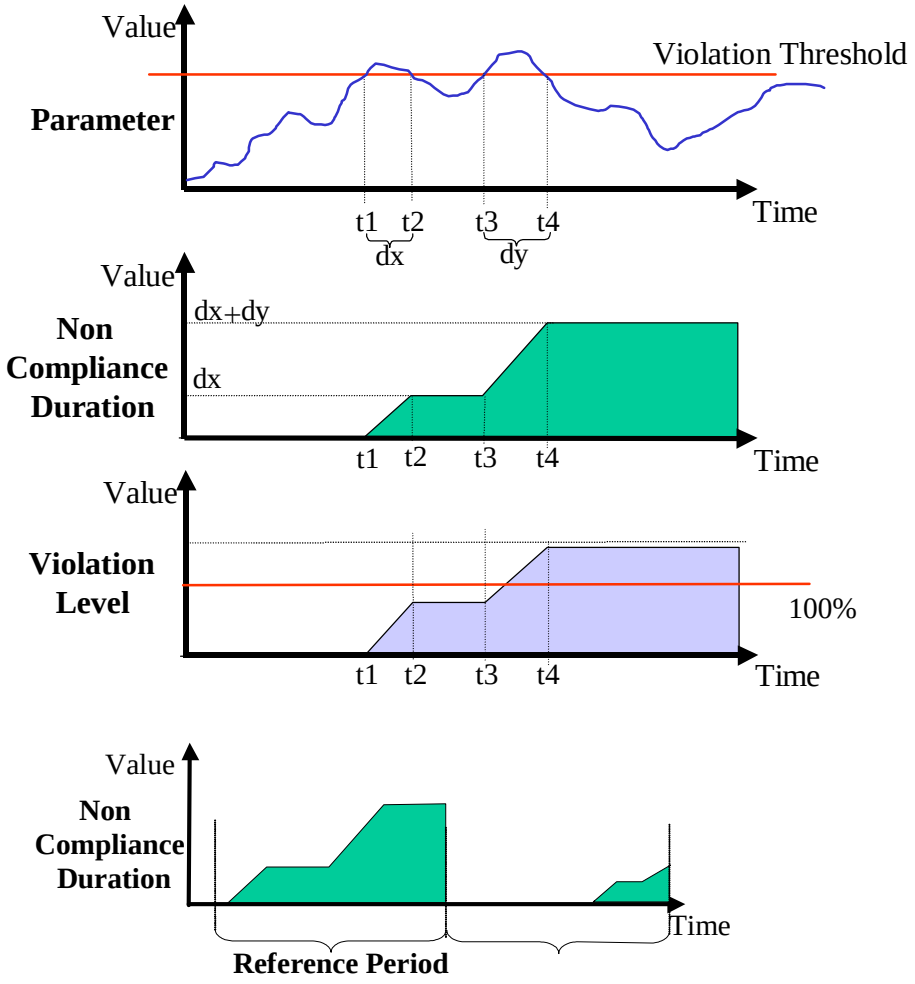
When the parameter crosses the violation threshold, then the SLO becomes non-compliant. Non-compliance time is accumulated until the parameter is under the violation threshold.

The compliance violation level (CVL) is expressed as the ratio between the non-compliance accumulated time and the total allowed violation duration for this service level objective.

The non-compliance time is accumulated over the contractual reporting period, called the reference period. For example, a reference period could be one month. At the end of the period, it is reset to zero. Figure 29 illustrates a measurement of compliance violation levels over time.

The contractual reporting period is defined at the SLA level.

Figure 29 Measuring Compliance Violation Levels



2.9.4.2 SLA Compliance Example

For a reference period of one month, the service is offered 8-8 M-F, meaning from 8:00 a.m. until 8:00 p.m. Monday to Friday.

So the overall in service time in this reference period is 22 open days * 12 hours * 60 minutes = 15840 minutes.

If the targeted Service Availability (SA%) defined in the contract is 99%, then the maximum allowed non-compliance duration is 1% of 15840, or 158 minutes.

If after 11 days and 2 hours of operation, the non-compliance accumulated time is 100 minutes, then the current compliance violation level at that time will be 100/158 = 63%.

2.9.4.3 "In Service" Hours

Service hours are defined at the SLA level rather that at the instance level.

The "in service" hours are expressed as a compound schedule:

- Recurrence rules: weekly, time per day, days per week. For example, Mo-Fr 8 a.m. – 6 p.m.
- Exclusion rules (optional): exclude exact absolute date. For example, exclude Jul 14 2002.

For example, a service provider wants to have the following in service hours:

Service Hours: 7 days, 24 hours
Except Saturdays from 3:00 p.m. - 6:00 p.m.

This schedule is achieved by defining the following three recurring schedules:

Days selected: Mo, Tu, We, Th, Fr, Su
Start Time: 12:00 a.m.
Finish Time: 12:00 p.m.

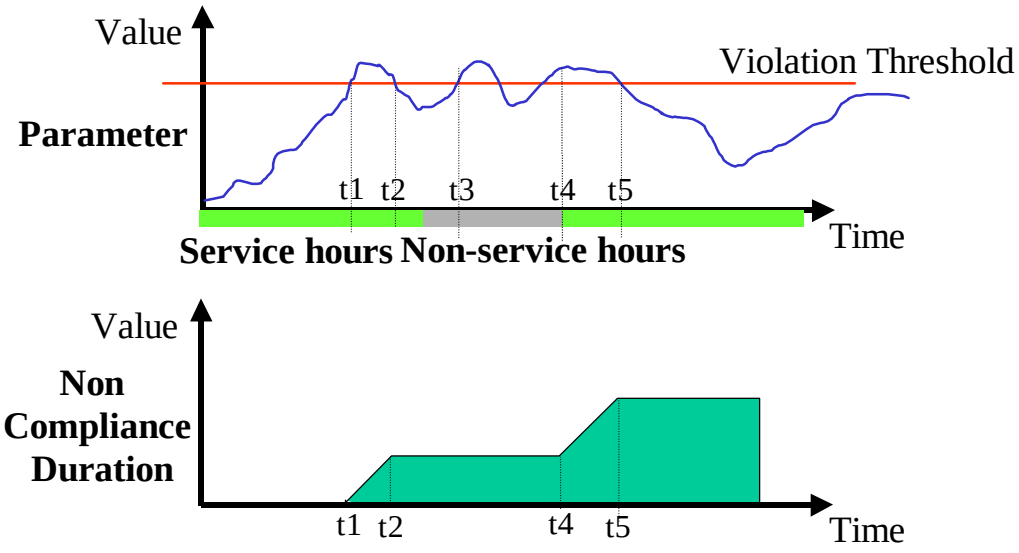
Day selected: Sat
Start Time: 12:00 a.m.
Finish Time: 03:00 p.m.

Day selected: Sat
Start Time: 06:00 p.m.
Finish Time: 12:00 p.m

When a service level objective exceeds a violation threshold during “off service” hours, it does not affect the compliance violation level although it still affects the service (for example, t3 in Figure 30).

If a service level objective exceeds a violation threshold and accumulates non-compliance time, it stops accumulating non-compliance time as soon as the service hours end, even if it is still over the violation threshold. When the “off-service” hours end, the non-compliance time accumulation will be resumed only if the parameter is still over the threshold (for example, t4 in Figure 30).

Figure 30 Monitoring In Service Hours



2.9.4.4 Instance Compliance Level and SLA Compliance Level Calculations

Each time a parameter value is validated against a given service level objective (such as threshold) a service objective status is computed. Note that this status is a percentage (1 – Service Degradation Factor).

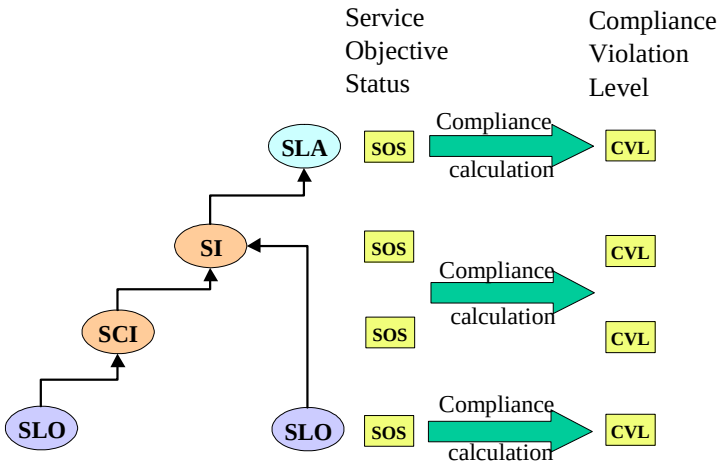
From this elementary SLO service objective status, the Service Level Object engine SLOM computes the service objective status of:

- Service component instances
- Service instances
- SLAs

The compliance violation level of component instances, service instances, and SLAs are calculated by applying the same formula as was applied for the SLO compliance. The formula consists of accumulating the non-compliance time according to the service hours, that is, the time when the service objective status is violated.

Figure 31 illustrates instance compliance level and SLA compliance level calculations.

Figure 31 Compliance Calculations from the Service Objective Status



2.9.4.5 Operational Compliance Monitoring

The service compliance violation level is monitored as any other parameter: in real time at the service level objective, service component, service, and SLA levels.

This means that the Service Designer UI can associate specific actions (such as sending an alarm) when a SLA is not compliant or will soon no longer be compliant (for example, the jeopardy threshold has been reached).

2.9.4.6 Service Health Indicators

Service objective status, compliance violation levels, and associated status can be aggregated over time, for example on a monthly basis. The aggregations provide the following service health indicators:

- Service availability percentage
- Mean time between failure (MTBF)
- Mean time to repair (MTTR)

2.10 Service Collection Management

The OpenView SQM administrator can control how the SLA and service data is collected.

When an SLA is unlocked, OpenView SQM automatically:

- Enables collection on all service instances.
- Enables all of the requested data feeder instances.

Each level of the data collection chain (data feeder instances, services, service groups) implements an “Operational” and “Availability” collection status.

The data feeder instances collect the data and propagate it to the SLA. Therefore, the availability collection state of SLA information depends on the overall instance hierarchy. When binding fails between service component instances and data feeder instances, the collection status of upper levels is affected.

You can manage the administrative state of SLAs, service instances, and data feeder instances by setting the administrative state to “unlocked” to activate data collection or to “locked” to inactivate data collection.

When the administrative state is set to “unlocked” for a data feeder instance, the responsible service adapter tries to start data collection. If the data feeder instance has already been deployed, data collection begins automatically.

The administrative state of an SLA indicates whether it is monitored or not. Moreover, all instances, whatever their level in the hierarchy, have two state parameters:

- Operational status (disabled and enabled).
- Availability status (in test, failed, power off, off line, off duty, dependency, degraded, not installed, and log full).

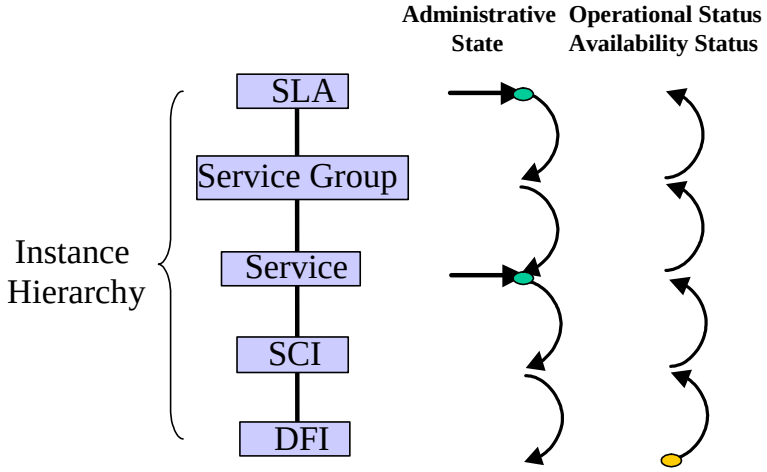
Unlike OSI standard definitions, the availability is not a collection of values. In addition, an enumeration has been added to represent the valid status (normal).

The state parameters indicate if all data, some data, or no data is collected for a particular instance. The value of the parameter is calculated from the underlying instances. Consequently, each time the collection status changes on a data feeder instance, the data collection status of all upper instances needs to be recomputed.

Finally, descriptive information is associated with the collection status in case further information about the degradation is available.

Figure 32 describes how a service is monitored.

Figure 32 Monitoring a Service

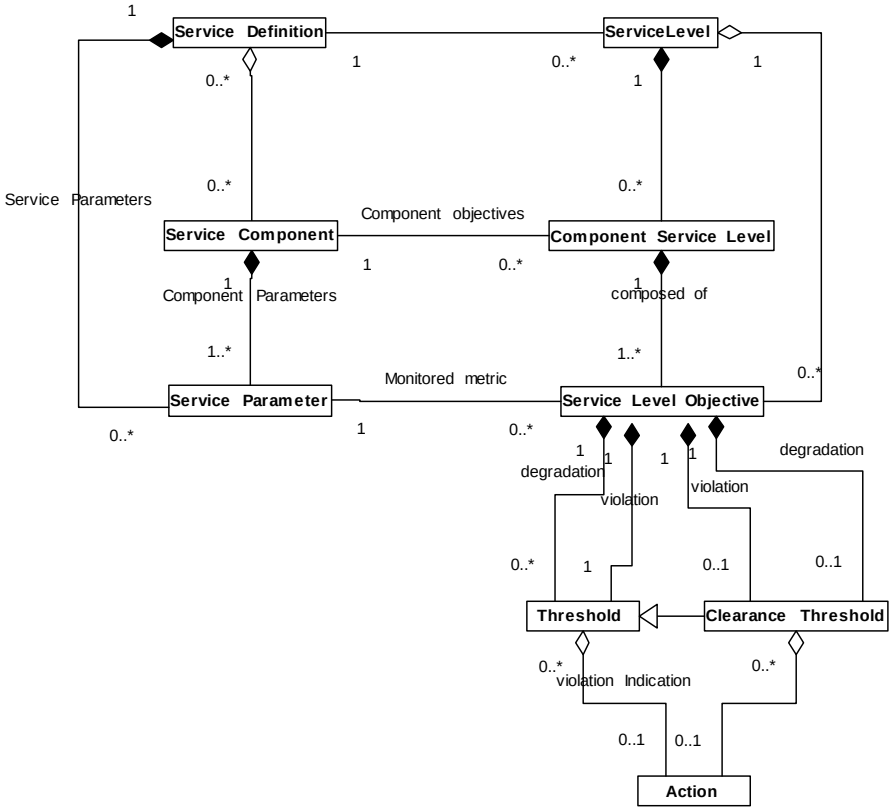


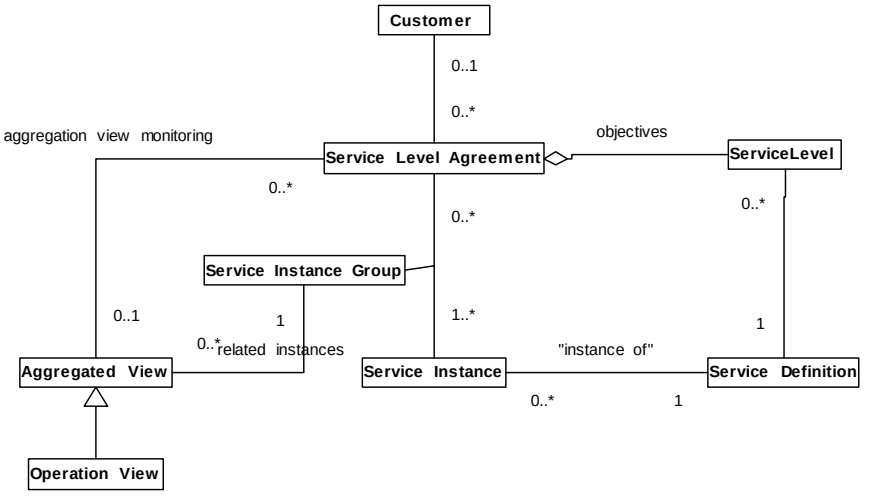
The different administrative states have the following dependencies:

- When an SLA is monitored, data is automatically collected on the involved service instances and data feeder instances.
- When data collection is enabled on a service instance, data is collected on all data feeder instances below it.
- When SLA monitoring is stopped, data collection is stopped on all of the associated service component instances and data feeder instances not used by another SLA.
- When data collection is disabled on a service instance, collections stops on all of the data feeder instances not used by another service instance.

When a data feeder instance is not defined for a given service component instance, it has an availability status of “Failed” and an explanation that indicates it is undefined. When a data feeder instance of a given service component instance is pre-registered, it has an availability status of “Not Installed”.

2.10.1 Summary of the UML Model





Chapter 3

Structuring the Service Object Model

This chapter describes how to structure your service object model using the OpenView Service Designer UI. The structure of your service model allows you to benefit from the OpenView Service Designer UI capabilities and facilitates the reuse of various definitions.

This chapter contains the following sections:

- Section 3.1 Using UML Packages
- Section 3.2 Controlling Unit Packages
- Section 3.3 Proposed Structure

3.1 Using UML Packages

In UML, a package allows you to group any model elements together. Centralizing the definitions you specify in a package helps you find and reuse them.

We recommend that you:

- Group all of your shared service components in a “Service Components” package even if they belong to different services.
- Group all of your data feeders in a “Data Feeders” package.
- Group the predefined and custom expressions in an “Expressions” package.
- Group your custom data types in the “Data Types” package.
- Define specific packages for your service models, such as “Video Service”.

3.2 Controlling Unit Packages

A Rose CAT file (`.cat`) corresponds to control units, packages that are shared. You can load these files in the OpenView Service Designer UI, meaning that other service models can reuse definitions included in the package.

Generally, you should share the package units of the following packages:

- The shared service components
- The data feeders
- The expressions

Of course, you can define other packages to suit your needs. However, we do not recommend using subfolders within packages.

Note

When you share a unit package containing service component definitions, the UI automatically configures the definitions as “Shared Service Definitions”.

3.3 Proposed Structure

The structure of your service model should consist of the following:

- The “Main Class Diagram” in the logical view of the OpenView Service Designer UI, which contains all of the packages.
- The “Service Components” package, which contains:

  All the service component definitions found in the main class diagram.

  One expression sequence diagram for each service component.

  For each service parameter of a service component, a sequence diagram that describes how to calculate its values from the data retrieved (either from data feeders or from the service parameters of other service components).
- The “Data Feeders” package, which contains all of the data feeder definitions from the main class diagram.
- The “Expressions” package, which contains all of the operations you can use to build expressions (predefined and custom) in a class diagram. Develop custom expressions in Java and PL/SQL, the language you use depends on the type of parameter you are calculating.
- The “Data Types” package, which contains all of the custom data types needed and their possible values in a class diagram.
- The package for your service (such as “Video Service”), which contains the following:

  A service model for your service in a class diagram that shows the relationships between the service and service components, as well as the associations between the service, service components, and data feeders.

  An expressions sequence diagram of your service. For each service parameter, the sequence diagram describes how to calculate the parameter’s values from the data retrieved (either from data feeders or from the service parameters of other service components).

Chapter 4

Using Expressions

This chapter describes how to use the expressions that calculate service parameters. It contains the following sections:

- Section 4.1 Predefined Expressions
- Section 4.2 Implementing Custom Expressions

4.1 Predefined Expressions

OpenView SQM comes with a list of existing expressions to simplify the creation of your own service model. These expressions cover most of your needs.

When you install the OpenView SQM Service Designer UI for the first time, the initial framework comes with all of the predefined expressions so that you can use them immediately to create your service.

The predefined expressions are also preloaded in OpenView SQM when you install it for the first time.

Predefined expressions are provided in two formats:

- Java expressions for data feeder binding.
- PL/SQL expression for component binding.

Simple and aggregation expressions are provided in both formats.

4.1.1 Java Expressions

All of the Java expressions are simple expressions. Table 7 lists the Java expressions provided with OpenView SQM.

Table 7 Simple Expressions Provided by OpenView SQM

Expression	Purpose	Parameters Type	
		Input	Output
assign_Float	Returns a float input parameter.	float	float
assign_Int	Returns an integer input parameter.	integer	integer
assign_RelTime	Returns a relative time.	reltime	reltime
assign_String	Returns a string.	string	string
assign_AbsTime	Returns an absolute time.	abstime	abstime

Expression	Purpose	Parameters Type	
		Input	Output
assign_Enum	Returns an enumeration.	enum	enum
avg_2Float	Computes the average of the two float input parameters.	float, float	float
avg_2RelTime	Computes the relative time from the two input parameters.	reltime, reltime	reltime
divide_2Float	Divides two float input parameters (first parameter/second parameter).	float, float	float
max_2Float	Computes the maximum of two float parameters.	float, float	float
max_2Int	Computes the maximum of two integer parameters.	integer, integer	integer
max_2RelTime	Computes the maximum of two relative time input parameters.	reltime, reltime	reltime
min_2Float	Computes the minimum of two float input parameters.	float, float	float
min_2Int	Computes the minimum of two integer input parameters.	integer, integer	integer
min_2RelTime	Computes the minimum of two relative times.	reltime, reltime	reltime
min_2AbsTime	Computes the minimum of two absolute times.	abstime, abstime	abstime
minus_2Float	Subtracts two float parameters (first parameter – second parameter).	float, float	float
minus_2Int	Subtracts two integer parameters (first parameter – second parameter).	integer, integer	integer
minus_2RelTime	Subtracts two relative time parameters (first parameter – second parameter).	reltime, reltime	reltime

Expression	Purpose	Parameters Type	
		Input	Output
minus_2AbsTime	Subtracts two absolute time parameters (first parameter – second parameter).	abstime, abstime	abstime
multiply_2Float	Multiply two float parameters (first parameter * second parameter).	float, float	float
multiply_2Int	Multiply two integer parameters (first parameter * second parameter).	integer, integer	integer
sum_2Float	Sums two float parameters (first parameter + second parameter).	float, float	float
sum_2Int	Sums two integer parameters (first parameter + second parameter).	integer, integer	integer
sum_2RelTime	Sums two relative time parameters (first parameter + second parameter).	reltime, reltime	reltime
sum_2String	Sums two string parameters (first parameter + second parameter).	string, string	string

4.1.2 PL/SQL Expressions

OpenView SQM provides the same simple expressions in PL/SQL as it does in Java. Refer to Table 7 for a complete list of simple expressions.

OpenView SQM also provides the aggregation expressions in PL/SQL. Table 8 lists the aggregation expressions available for your use.

Table 8 Aggregation Expressions Provided by OpenView SQM

Expression	Purpose	Parameters Type	
		Input	Output
avg_agg_Float	Computes the average from a list of float input parameters.	list of floats	float
avg_agg_RelTime	Computes the average from a list of relative time input parameters.	list of reltimes	reltime
max_agg_Float	Computes the maximum from a list of float input parameters.	list of floats	float
max_agg_Int	Computes the maximum from a list of integer input parameters.	list of integers	integer
max_agg_RelTime	Computes the maximum from a list of relative time input parameters.	list of reltimes	reltime
max_agg_AbsTime	Computes the maximum from a list of absolute time input parameters.	list of abstimes	abstime
min_agg_Float	Computes the minimum from a list of float input parameters.	list of floats	float
min_agg_Int	Computes the minimum from a list of integer input parameters.	list of integers	integer
min_agg_RelTime	Computes the minimum from a list of relative time input parameters.	list of reltimes	reltime
min_agg_AbsTime	Computes the minimum from a list of absolute times input parameters.	list of abstimes	abstime
sum_agg_Float	Computes the sum of a list of float input parameters.	list of floats	float
sum_agg_Int	Computes the sum of a list of integer input parameters.	list of integers	integer
sum_agg_RelTime	Computes the sum of a list of relative time input parameters.	list of reltimes	reltime

Expression	Purpose	Parameters Type	
		Input	Output
sum_agg_String	Computes the sum of a list of strings input parameters.	list of strings	string
union_agg_String	Computes a union of a list of string input parameters.	list of strings	string

4.1.3 PL/SQL Expressions and management of "NoValue"

OpenView SQM provides a way to manage the case when there's no value available as input of a. PL/SQL expression (i.e. input parameter value is "NoValue").

Note: parameters with a "NoValue" are marked as "Not Available" in the OV SQM Real Time Monitoring User Interface.

The management of the "NoValue" case is based on the SPDM configuration property "*PropagateUnavailableFlag*".

4.1.3.1 Ignore "NoValue" as input of the PL/SQL Expressions

If the "*PropagateUnavailableFlag*" configuration property is set to "false", then the SPDM discards the "NoValue" and it uses instead the latest available parameter value as input of the PL/SQL expressions.

Example:

param3 = sum_2Int(param1, param2)

At step 1:

param1 = 10 and param2 = 2

→ param3 = sum_2Int(10,2) = 10 + 2 = 12

At step 2:

param1 = "NoValue" and param2 = 5

→ param3 = sum_2Int(10,5) = 10 + 5 = 15

(To compute the value of param3 with the sum_2Int() expression, the SPDM has replaced the "NoValue" of param1 by its latest available value).

4.1.3.2 Use “NoValue” as input of the PL/SQL Expressions

If the “*PropagateUnavailableFlag*” configuration property is set to “true”, then the SPDM uses “NoValue” as input of the PL/SQL expressions.

This allows full control of the PL/SQL expressions and implementation of the desired behavior in case of “NoValue” as input of the PL/SQL expressions.

Case 1: Expected behavior of the PL/SQL expression is to return a “NoValue” in case of input “NoValue”

This behavior is implemented by the simple and aggregated PL/SQL expressions provided along with OV SQM Service Designer and described in Table 7 and Table 8.

Case 2: Expected behavior of the PL/SQL expression is to ignore “NoValue” provided as input

This behavior is implemented by the simple and aggregated PL/SQL expressions provided along with OV SQM Service Designer in a specific “IgnoreNoValue” package.

This package contains a related “IgnoreNoValue” PL/SQL expression for each simple and aggregated expression described in Table 7 and Table 8.

param3 = sumIgnoreNoValue_2Int(param1, param2)

At step 1:

param1 = 10 and param2 = 2

→ param3 = sumIgnoreNoValue_2Int(10,2) = 10 + 2 = 12

At step 2:

param1 = “NoValue” and param2 = 5

→ param3 = sumIgnoreNoValue_2Int(“NoValue”,5) = 5

(OV SQM has ignored the “NoValue” of param1).

Case 3: Other expected behavior of the PL/SQL expression

If the expected behavior of the PL/SQL expression does not match case 1 or case 2 (for instance, if a “NoValue” should be replaced by a default value), then it’s possible to implement a custom PL/SQL expression that manages the input “NoValue”.

Refer to chapter 4.2.4 Managing “NoValue” in Custom PL/SQL Expressions for an example of custom PL/SQL expression that manages “NoValue” provided as input value.

4.2 Implementing Custom Expressions

You can create custom expressions to address the particular needs of your service. This section describes when you need to create custom expressions, how to implement them in PL/SQL and Java, and how to load them.

4.2.1 When Do You Need to Develop Your Own Expressions?

The predefined expressions should cover most of your needs, but sometimes calculating a parameter requires operations that are more complex.

You can create custom expressions to address the particular needs of your service. You can develop any type of expression: simple, aggregation, and buffering.

Note

The primary Custom expressions (Java ones) must not start with the keyword *assign*.

For example, the Video Service designer wants to calculate the worst operational state of a set of three operational states (the enum values of `Enable`, `Disable`, and `Idle`) collected by a service adapter from a Web server component. OpenView SQM comes with a predefined expression called Max_AggrInt (int a, int b, int c …) that computes the maximum from a list of Integers given in the parameter. The Video Service designer can use this expression if the integer value of the disabled enumeration is the biggest compared to Enable and Idle. Otherwise, the designer needs to develop a custom expression.

The Video Service designer uses the OpenView Service Designer UI to define custom expressions without implementing them. The implementation and the loading can be done in another step.

4.2.2 Implementing Custom Java Expressions

You implement expressions with Java when the output parameter is a service quality measurement.

For example, the video service provider wants to implement the following expression with Java:

f(x1,…, xn, version)

The expression contains input parameters and a version argument. The version argument identifies the version of the data feeder definition. As data feeder definitions evolve, the version argument ensures that different versions of a data feeder can co-exist and apply their corresponding expressions.

The Java code used to implement the example expression follows:

```
package com.compaq.temip.servicecenter.expressions;

public class ASimpleCalculationExpression {

    public static double aSimpleCalculationExpression
(int a,
       double b,
```

```
        String dfdVersion) {
         double diff = a-b;
         return diff;
    }
}
```

4.2.3 Implementing Custom PL/SQL Expressions

Expressions need to be implemented in PL/SQL when:

- The output parameter is a service quality parameter.
- The expression defines an election policy.

For example, the video service provider wants to implement the following simple expression for a service quality parameter:

f(p1, …, pn)

The PL/SQL code used to implement the expression follows:

```
CREATE OR REPLACE FUNCTION sum_2Int (a NUMBER, b NUMBER) RETURN
NUMBER IS
BEGIN
   res := a + b;
   RETURN res;
END;
/
SHOW ERR;
```

The video service provider also wants to implement the following aggregation expression for a service quality parameter:

f(coll p1, …, coll pn)

The PL/SQL code used to implement this expression follows:

```
CREATE OR REPLACE FUNCTION avg_aggr_Float (a float_coll) RETURN
NUMBER IS
    res NUMBER :=0;
BEGIN
    FOR I IN a.FIRST..a.LAST
       LOOP
     res := res + a(i);
    END LOOP;

    res := res/a.COUNT

    RETURN res;
END;
/
SHOW ERR;
```

4.2.4 Managing “NoValue” in Custom PL/SQL Expressions

Expressions implemented in PL/SQL can take into account the case where a “NoValue” is provided as input value of an expression.

Reminder: this occurs only when the SPDM “*PropagateUnavailableFlag*” configuration property is set to “true”.

A “NoValue” is represented in the PL/SQL code of the expression by a NULL value.

For example, the video service provider wants to implement the following simple expression for a service quality parameter:

myFunct(p1, p2)

This function does a sum of p1 and p2.

If p1 or p2 is not available (i.e. “NoValue”), then the video service provider needs to use 10 as default value for p1 and 20 as default value for p2.

The PL/SQL code used to implement the expression follows:

```
CREATE OR REPLACE FUNCTION myFunct (p1 NUMBER,p2 NUMBER) RETURN
NUMBER IS
BEGIN

   -- If input for p1 is “NoValue”
   IF ( p1 IS NULL )
   THEN
      -- Use 10 as default value
      p1 := 10;
   END  IF;

   -- If input for p2 is “NoValue”
   IF ( p2 IS NULL )
   THEN
      -- Use 20 as default value
      p2 := 20;
   END  IF;


   res := p1 + p2;
   RETURN res;
END;
/
SHOW ERR;
```

4.2.5 Loading Custom Expressions

Once you have created your custom PL/SQL or Java expressions, you need to load them into OpenView SQM.

First, you need to generate the XML file that will contain the expression source code required by OpenView SQM. The OpenView Service Designer UI allows you to generate this XML file. For more information, refer to the *OpenView Service Designer UI User’s Guide.*

Next, you need to use the CLUI on your UNIX machine to send the XML file to OpenView SQM as follows:

```
temip_sc_load_definition –e Custom_Expression.xml
```

Chapter 5

Mapping Subscriber IDs

This chapter describes how to map subscriber IDs to customers. It contains the following sections:

- Section 5.1 About Mapping Subscriber IDs
- Section 5.2 Creating the Mappings
- Section 5.3 Using the customer premise flag

5.1 About Mapping Subscriber IDs

An ISP can offer services, such as mail and Web services, to other businesses. These businesses, in turn, can give employees and subscribers access to the services. For example, Software House A buys the Video service from the ISP. Software house A then gives its employees access to this service. For the ISP to manage service quality, it needs to be able to map service interactions made by Software House A employees with the correct customer SLA.

To make this mapping between subscribers and customers, OpenView SQM uses subscriber IDs. The subscriber ID identifies the user of the service in a service quality measure. The subscriber ID can be an IP address, a user login name, or an identifier directly linked with a user, such as a DSL port number.

Sometimes, the user happens to be directly the Customer. So, the subscriber ID maps directly to the customer ID. In such case, that does not require a mapping the domain is by default the generic “ServiceCenter” domain.

5.2 Creating the Mappings

This section describes how to map subscriber IDs. It tells you what to do before you begin mapping, describes the structure of a mapping, how to write a mapping in XML, and how to provide the mapping file to OpenView SQM.

5.2.1 Before You Begin

Customers are created through the OpenView SLA Administration UI and stored in the OpenView SQM repository. Subscribers are created through an independent tool.

Therefore, before you begin mapping subscriber IDs, you must have previously created customers using the OpenView SLA Administration UI.

5.2.2 About the Mapping Structure

A subscriber ID mapping has a simple structure that consists of a customer name and a naming plane. A naming plane defines the customer identifiers for each sub-domain. These identifiers are:

- Explicit (for example, a specific IP address).
- Range of valid identifiers (such as an address that includes regular expressions).

An example of a subscriber ID mapping follows:

```
Customer Name: MyNewCustomer
Naming Planes:
{
Domain Identifier: IP
ipaddress: {“16.18.*.*”, “16.23.*.*”}
}
{
Domain Identifier: IMSI
ipaddress: {“12202???”, “2002???”}
}
{
Domain Identifier: Mail
ipaddress: {“*@hp.com”}
}
```

Domain identifiers are free text. However, an IP is a keyword and can be verified as a valid IP address.

5.2.3 Writing the Mapping in XML

You must provide your subscriber ID mappings to OpenView SQM in an XML file. The following example illustrates how to set the customer name to the `MyNewCustomer` subscriber ID defined previously.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE sc:NewSubscribersReq SYSTEM "DTD/tsc_SubscriberNamingServiceSrv.dtd">
<sc:NewSubscribersReq msg.id = "3" xmlns:sc =
”<http://www.compaq.com/TeMIP/ServiceCenter>””
customer.name = "MyNewCustomer" customer.label = "My new Customer label">
      <sc:NamingPlanes>
            <sc:NamingPlane subscriber.domain = "IP" domainType = "ip" > <sc:Descr>My
new customer IP domain</sc:Descr>
            <sc:SubscriberNameRanges>
                  <sc:SubscriberNameRange subscriber.nameRange="16.18.*.*"/>
                  <sc:SubscriberNameRange subscriber.nameRange="16.23.*.*"/>
            </sc:SubscriberNameRanges>
            </sc:NamingPlane>
            <sc:NamingPlane subscriber.domain = "IMSI" domainType = "imsi" >
<sc:Descr>My new customer IMSI domain</sc:Descr>
            <sc:SubscriberNameRanges>
                  <sc:SubscriberNameRange subscriber.nameRange="12202???"/>
                  <sc:SubscriberNameRange subscriber.nameRange="2002???"/>
            </sc:SubscriberNameRanges>
            </sc:NamingPlane>
            <sc:NamingPlane subscriber.domain = "MAIL" domainType = "mail" >
<sc:Descr>My new customer MAIL domain</sc:Descr>
            <sc:SubscriberNameRanges>
                  <sc:SubscriberNameRange subscriber.nameRange="*@hp.com”/>
            </sc:SubscriberNameRanges>
            </sc:NamingPlane>
      </sc:NamingPlanes>
</sc:NewSubscribersReq>
```

5.2.4 Providing the Mapping File to OpenView SQM

To provide a new naming plan to OpenView SQM, use the following command:

```
temip_sc_ns_admin_tool.sh –platform <Platform> -director
<Director> -application <Naming Service Application>
filename.xml
```

For a standard SQM installation (typical), the command to enter is:

```
temip_sc_ns_admin_tool.sh –platform slmv10 -director
slmonitoring -application NS filename.xml
```

5.2.5 Customer dedicated Data Feeder Instance

A measurement point can be dedicated to a given customer when for instance a resource is allocated for a given customer (example a port on a Premise Equipment Router).

In such case the customer ID is defined when either the Data Feeder Instance is pre-registered or after it’s discovered.

Chapter 6

Updating the Object Model

This chapter describes how to update the service object model and the implications of your updates. It contains the following sections:

- Updating Service Definitios and Expressions
- Updating Data Feeders
- Updating Service Level Agreements

6.1 Updating Service Definitions and Expressions

After you have created, implemented, and instantiated a service definition, or an expression you can update it. The following sections describe the types of updates you can make to service definitions that are already in product and the implications of your changes.

6.1.1 Authorized and unauthorized Service Definition updates

The following table lists authorized/ forbidden update operations. Note that even if the update is theoretically allowed, this does not necessarily mean that the input XML is semantically correct (for example a parameter is deleted but the corresponding bindings are kept: the transition is authorized but an inconsistency will be detected by the semantic validation because bindings cannot refer to an inexistent parameter).

Add a Service Component Definition as a leaf	YES
Add a Service Component Definition hierarchy as a leaf	YES
Add a Service Component Definition or Service Component Definition hierarchy between two existing Service Component Definitions	YES See Chapter: “Safe” Updates – Object Deletion (because Service Component Definition Associations are deleted)
Remove a leaf Service Component Definition	YES See Chapter: “Safe” Updates – Object Deletion.
Remove a leaf Service Component Definition hierarchy	YES See Chapter: “Safe” Updates – Object Deletion.

Remove a Service Component Definition or Service Component Definition hierarchy between two existing Service Component Definitions	YES See Chapter: “Safe” Updates – Object Deletion (because Service Component Definition Associations are deleted)
Update the SD attributes	YES
Update the Service Component Definition attributes	YES except the Definition Sharing flag and the Instance Sharing flag
Add a relationship between two Service Component Definitions	YES provided the child Service Component Definition is added at the same time
Remove a relationship between two Service Component Definitions	YES provided the child Service Component Definition is removed at the same time
Update the cardinalities of a relationship between two Service Component Definitions	YES provided the instantiation is compatible with the new cardinalities. Indeed the cardinality update may change the Instantiation Sharing flag of the Service Component Definition: NONE->LOCAL: OK LOCAL->NONE: This works if no Service Component Instance (of the Service Component Definition) is associated to several Service Instances.
Add a Parameter (SD/Service Component Definition levels)	YES
Add/Remove/Update a Buffering expression to a Parameter	YES
Remove a Parameter (SD/Service Component Definition levels)	YES
Update the Parameter attributes	YES except the datatype
Add a Primary Binding	YES
Add a Secondary Binding	YES
Remove a Primary Binding	YES
Remove a Secondary Binding	YES
Change the list of Input Parameters of a Binding (Prim/Sec)	YES
Change the expression used within a Binding (Prim/Sec)	YES
Change the expression source	YES

code	

You can update a service definition with any combination of these modifications at the same time.

Any other modification will not be considered as an update of a service definition. For example, you cannot insert a service component between two other service components. Instead, you must create a new service definition.

6.1.2 Implications of Your Changes

Your updates to service definitions have the implications described in the following sections.

6.1.2.1 Processing Time Delay

Updating a service definition in OpenView SQM takes processing time and can delay the validation of service quality data, as well as the generation of SLA degradation and violation information in a deployed environment. However, no service quality data will be lost, as all measures are kept in a queue during the processing time.

Note also that it is not necessary to restart SQM Core components in order to take into account modifications.

6.1.2.2 Binding Service Components to Data Feeders

When you update a service definition, the old service instances entered with the old service definition are updated to stay consistent with the service model. As a result, some modifications that imply a new data feeder or a new service component require that you do the binding between new service component instances and the associated data feeder instances for all old service instances.

6.1.2.3 "Safe" Updates – Object Deletion

Most model updates (except the deletion of "global" objects such as Service Definition, Service Instances… that can be "forced") are performed in "safe" mode, which means that it is not authorized to delete an object that has some "external" dependencies.

For example it is not possible to delete a Service Component Definition which is still instantiated (Service Component Instances).

As a consequence, the user needs first to remove the object dependencies. Then he will be able to remove safely the object itself.

Here are the dependencies to take into account when deleting/updating Service Definitions/ Data Feeder Definitions/ Calculation Expressions.

Safe Operation/Object	**Dependencies**
Delete Service Definition	• Parameter: o Any existing binding for which this parameter is used as input for calculation of another parameter in another Service Definition • Service Instances • Service Levels • Service Instance Groups • Service Level Agreements
Update Service Definition	• If Parameter to be deleted: see related operation below

	• If Property to be deleted: see related operation below • Update/Delete Service Component Definition (see related operation below) • If Primary Binding to be deleted: see related operation below • If Service Component Definition Associations to be deleted: see related operation below
Delete Data Feeder Definition	• Parameter: see related operation below • Data Feeder Instances
Update Data Feeder Definition	• If Parameter to be deleted: see related operation below • If Property to be deleted: see related operation below
Update Service Component Definition	• If Parameter to be deleted: see related operation below • If Property to be deleted: see related operation below
Delete Service Component Definition	• Parameter: o Any existing binding for which this parameter is used as input for calculation of another parameter • Service Component Instances • Component Service Levels
Delete Parameter (Service Definition/ Service Component Definition)	• Service Level Objectives • Any existing binding for which this parameter is used as input for calculation of another parameter
Delete Parameter (Data Feeder Definition)	• Any primary binding which use this Data Feeder parameter
Delete Property (Service Definition/ Service Component Definition/ Data Feeder Definition)	• Property Values referenced in model instances (Service Instance Groups, Service Instances, …)
Delete Primary Binding – Parameter computed from Data Feeder Definition parameter(s)	• Primary Associations whose: o Parent Service Component Instance references the Service Component Definition of the binding o Child DFI references the Data Feeder Definition of the binding
Delete Service Component Definition Association	• Secondary Associations whose: o Parent Service Component Instance references the Parent Service Component Definition of the Association o Child Service Component Instance references the Child Service Component Definition of the Association
Update Calculation Expression	• None
Delete Calculation Expression	• Any primary and secondary binding which use this expression

Examples:

1. When the user aims at removing a Service Component Definition through a Service Definition update (Service Definition being responsible for the Service Component Definition), he first needs to:
   - Remove corresponding Component Service Levels from Service Levels,
   - Remove corresponding Service Component Instances from Service Instances (only "responsible"- see chapter 6.1.2.4 - Service Instances need to be re-loaded).
   - Remove the bindings (in other Service Definitions) that reference as input parameters the deleted Service Component Definition parameters.
2. When the user aims at removing a Parameter of a Service/Service Component Definition through a Service Definition update (Service Definition being responsible for the Parameter), he first needs to:
   - Remove corresponding Service Level Objectives from Service Levels.
   - Remove the bindings (in other Service Definitions) that reference as input parameters the deleted parameter.

6.1.2.4 Deletion and Sharing

Service Component Definition deletion:

- The Service Component Definition is not definition shared: whether the Service Component Definition is deleted from an Update or a Delete Service Definition request, the Service Component Definition is fully removed from the model.
- The Service Component Definition is definition shared:
  - Update Service Definition: if the Service Definition is responsible for this Service Component Definition, the Service Component Definition and its sub-hierarchy is automatically deleted from all the Service Definitions that were referencing the Service Component Definition.
  - Update Service Definition: if the Service Definition is not responsible for this Service Component Definition, the Service Component Definition is not removed from the SRM model: the Service Component Definition (and its sub-hierarchy) is just no more referenced in the Service Definition.
  - Delete Service Definition:
    - When the Service Definition is responsible for the shared Service Component Definition, the Service Component Definition (and its sub-hierarchy) is removed from the SRM model.
    - When the Service Definition is not responsible for the shared Service Component Definition, the Service Component Definition (and its sub-hierarchy) is not removed from the SRM model.

Service Component Instance deletion:

- The Service Component Instance is not shared: Whether the Service Component Instance is deleted from an Update or a Delete Service Instance request, the Service Component Instance and its sub-hierarchy is fully removed from the model.
- The Service Component Instance is shared:
  - Update Service Instance: if the Service Instance is responsible for this Service Component Instance, the Service Component Instance and its sub-hierarchy is automatically deleted from all the Service Instances that were referencing the Service Component Instance.

- Update Service Instance: if the Service Instance is not responsible for this Service Component Instance, the Service Component Instance is not removed from the SRM model: the Service Component Instance (and its sub-hierarchy) is just no more referenced in the Service Instance.
- Delete Service Instance: whether the Service Instance is responsible or not for the shared Service Component Instance, the Service Component Instance (and its sub-hierarchy) is not removed from the SRM model.

Service Component Instance update:

The Service Component Instance updates (whether the Service Component Instance is shared or not) can only be performed from its "responsible" Service Instance. Let's notice that other Service Instances referencing the Service Component Instance don't need to be reloaded: the master Service Instance propagates automatically the Service Component Instance hierarchy structure update to the "secondary" Service Instances.

Service Component Definition update:

The Service Component Definition updates (whether the Service Component Definition is shared or not) can only be performed from its "responsible" Service Def.

Let's notice, for definition shared Service Component Definitions, that other Service Definitions referencing the Service Component Definition don't need to be reloaded:

- The Service Component Definition is globally shared: the master Service Definition propagates automatically the Service Component Definition (including its hierarchy) structure update to the "secondary" Service Definitions.
- The Service Component Definition is not globally shared: the master Service Definition propagates automatically the Service Component Definition (not its hierarchy) structure update to the "secondary" Service Definitions.

6.2 Updating Data Feeders

Data feeders have a version, meaning that different versions of the same data feeder can be used simultaneously. The following sections describe the types of updates you can make to data feeders and the implications of your changes.

6.2.1 Updates You Can Make

A data feeder definition can be updated as follows:

- Add a new parameter
- Remove an old parameter

6.2.2 Implications of Your Updates

Updates to data feeder definitions affect the definition of the source of service quality data. Therefore, you may need to update the service adapters that retrieve the service quality data.

6.2.2.1 Service Adapter Development Requirements

Usually, the generic service adapters provided with OpenView SQM do not need further development to implement collection of a new parameter. However, if you are using a custom service adapter, ensure that it can retrieve this new information.

6.2.2.2 Configuring the Service Adapter

You need to update the mapping between OpenView SQM and the new parameter on the data source. Update the mapping in the TiBCO repository. After the update, you must reload the service adapter.

6.2.2.3 Reloading the Service Adapter

The generic service adapters provided with OpenView SQM service adapters automatically reload updates to the data feeder definitions. However, if you are using a custom SA, ensure that the updates have been reloaded.

6.3 Updating Service Level Agreements

Each Service Level Agreement applies to a Customer or an Operation. Updating the Customer/Operation information for a given SLA is not authorized.

Chapter 7

Implementing the Service Model in XML

This chapter describes how to implement the service model in XML. It includes the following sections:

- Developing Your Own XML Code
- Example service model in XML

7.1 Developing Your Own XML Code

You can use the command line to interface with OpenView SQM and perform all of the operations available through the UIs, including:

- Load service definitions.
- Synchronize SQM with an inventory system (service instances).
- Define SLAs, customers, and so on.

Contact your HP representative for the document type definition (DTD) files you need. The `"*"Def.dtd` files they will provide you describe the object definitions.

Using these DTD files, write the following XML:

- Data feeder XML files (one for each data feeder).
- Expression XML files (one for each custom expression).
- Service XML files (one for each service).

7.2 Example of Service Definition

Example of XML files are provided under the following folder:
/opt/OV/SQMV120/Definitions/Services/Video/

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sc:ServiceDef SYSTEM "DTD/tsc_ServiceDef.dtd">

<sc:ServiceDef sd.name="TestServiceDef" version="v1_0" sd.label="Test Service
Definition" xmlns:sc="http://www.compaq.com/TeMIP/ServiceCenter">
    <sc:Descr>Test Service Definition.</sc:Descr>
    <sc:PropertyDefs>
    </sc:PropertyDefs>
    <sc:ParameterDefs>
        <sc:ParameterDef parameter.label="Total number of connected User"
parameter.name="TotNbConUser" datatype="Int" category="Rate" partition="QoS"
bindingType="Secondary" visibility.flag="True" autoForwarding.flag="False"
```

```
customerDepend.flag="False">
            <sc:Descr>Total number of connected User:
                        Sum(Connected User of Server Component)</sc:Descr>
        </sc:ParameterDef>
    </sc:ParameterDefs>
    <sc:SCDefs>
        <sc:SCDef scd.name="ServerGroup" scd.label="Sample Server Group"
stereotype="ServiceResource" instanceSharing.type="None" definitionSharing.flag="False"
visibility.flag="True">
            <sc:Descr> Aggregation of Component Sample Aggregation Component
</sc:Descr>
            <sc:ParameterDefs>
                <sc:ParameterDef parameter.label="Total Nb Connect User"
parameter.name="NbConUser" datatype="Int" category="Rate" partition="QoS"
visibility.flag="True" autoForwarding.flag="False" customerDepend.flag="False">
                </sc:ParameterDef>
            </sc:ParameterDefs>
            <sc:PropertyDefs>
            </sc:PropertyDefs>
        </sc:SCDef>
    </sc:SCDefs>
    <sc:SCDAssociations>
        <sc:SCDAssociation name="TestServiceDef_ServerGroup">
            <sc:AssoParentEnd scd.name="TestServiceDef" card.min="1" card.max="1"/>
            <sc:AssoChildEnd scd.name="ServerGroup" card.min="1" card.max="1"/>
        </sc:SCDAssociation>
    </sc:SCDAssociations>
    <sc:PrimaryBindings>
        <sc:PrimaryBinding expr.name="assign_Int" expr.language="Java"
sd.name="TestServiceDef" scd.name="ServerGroup" parameter.name="NbConnUser">
            <sc:DFDInputParams>
                <sc:DFDInputParam dfd.name="ServerDFD" dfd.version="v1_0"
parameter.name="NbConnUser"/>
            </sc:DFDInputParams>
        </sc:PrimaryBinding>
    </sc:PrimaryBindings>
    <sc:SecondaryBindings>
        <sc:SecondaryBinding expr.name="assign_Int" expr.language="PL_SQL"
sd.name="TestServiceDef" scd.name="TestServiceDef" parameter.name="TotNbConUser"
type="Simple">
            <sc:SCDInputParams>
                <sc:SCDInputParam scd.name="ServerGroup" parameter.name="NbConUser"/>
            </sc:SCDInputParams>
        </sc:SecondaryBinding>
    </sc:SecondaryBindings>
</sc:ServiceDef>
```

Appendix A

Service Parameter and Property Attributes

This appendix describes the attributes of the service parameters and service properties.

A.1 Service Parameter Attributes

Name	Data Type	Mandatory	Description
Name	char (16)	Yes	Parameter identifier. For more information about parameters, refer to Chapter 1.
Label	char (512)	No	Presentation label.
Description	char (1024)	No	Description of the parameter.
Datatype	enumeration	Yes	Primitive data type of the parameter. For more information about parameters, refer to Chapter 1.
Category	enumeration	No	For more information about parameters, refer to refer to Chapter 1.
Partition	enumeration	No	For more information about parameters, refer to refer to Chapter 1.
Units	string	No	Free text that indicates the units of this parameter (for example, ms).
Visibility	Boolean	No	By default, visibility is set to True.
Customer Specific	Boolean	No	By default, this parameter is set to False.
AutoForward	Boolean	No	By default, this parameter is set to True.

Name	Data Type	Mandatory	Description
Buffering Expression	String	No	Indicates that the expression is used to elect a parameter. By default, this parameter is set to Last.

A.2 Service Property Attributes

Name	Data Type	Mandatory	Description
Name	char (16)	Yes	Parameter identifier. For more information about parameters, refer to refer to Chapter 1.
Label	char (512)	No	Presentation label.
Description	char (1024)	No	Description of the parameter.

Appendix B

Acronyms

The following table describes the acronyms commonly used in this document:

Term	Description
ATM	Asynchronous Transfer Mode
BI	Business Intelligence
BO	Business Object
BSS	Billing Support System
CNM	Customer Network Management
CRM	Customer Relationship Management
DC	Data Collector
DFD	Data Feeder Definition
DFI	Data Feeder Instance
DMZ	Demilitarized Zone
EAI	External Application Interface
GUI	Graphical User Interface
IP	Internet Protocol
ISP	Internet Service Provider
IT	Information Technology
MRP	Measurement Reference Point
MSL	Management Specification Language
MTBF	Mean Time Between Faults
MTTR	Mean Time To Repair
MVNO	Mobile Virtual Network Operator
NOC	Network Operation Center
OS	Operating System
OSI	Open Systems Interconnection
OSA	Open Service Architecture
OSS	Operation Support System
QoS	Quality of Service

Term	Description
SA	Service Adapter
SAI	Service Adapter Instance
SAP	Service Access Point
SC	Service Component
SD	Service Definition
SI	Service Instance
SIA	Service Impact Analysis
SIM	Service Integration Map
SLA	Service Level Agreement
SLM	Service Level Management
SNMP	Simple Network Management Protocol
SOC	Service Operation Center
SPD	Service Parameter Definition
SPDM	Service Performance Data Manager
SR	Service Resource
SRM	Service Repository Manager
SQL	Standard Query Language
TeMIP	Telecom Management Information Platform
TOM	Telecommunications Operation Map
UI	User Interface
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
XML	eXtensible Mark-up Language
WAP	Wireless Access Protocol

Glossary

This glossary defines terminology commonly used in HP OpenView Service Quality Manager.

Auto instantiate (SLA Administration)

This action automatically creates an Instance of the Object selected. When the instance is created, the initial values of its instance variables are assigned.

BI

See business intelligence.

Business intelligence (BI)

A broad category of applications and technologies for gathering, storing, analyzing, and providing access to data that helps users make better business decisions.

Collected binding

Describes how *collected parameters* are filled from *measurement parameters*: either directly assigned or through a more complex expression.

Collected parameters

Known as KPI in the TMF, they represent the parameters collected from the Service Adapters (*measurement parameters*) and mapped into SQM *service component* parameters.

Computed binding

Describes how *computed parameters* are filled from *collected parameters*: either directly assigned or through a more complex expression.

Computed parameters

Known as KQI in the TMF, they represent the parameters calculated from *collected parameters*.

CNM

See customer network management

Customer

Companies or organizations that make use of the *services* offered by a *service provider*, based on a contractual relationship.

Customer network management

Customer network management is enabled by means of tools that provide business customers with access to management information originating from the service provider.

Data collection interval

The interval of time over which performance parameters are retrieved from the monitored service resources. This interval does **not** have to be the same as the *measurement interval* because *service adapters* or service resources may buffer statistics.

Data feeder

OpenView Service Quality Manager's source of data. A data feeder models service resources by defining one or more service parameters.

Data feeder definition

The static definition of a data feeder that models service resources by defining one or more service parameters.

Degraded service

The presence of anomalies or defects that cause degradation of the *quality of service*, but do not result in the total failure of the *service*.

Instantiate (SLA Administration)

Instantiate differs from Auto Instantiate in that items are instantiated individually.

Measurement interval

The interval of time over which each service parameter is measured. For example, a parameter may be the number of discarded packets, measured over a 15-minute measurement interval.

Measurement parameters

They represent the parameters directly collected by the Service Adapters. These parameters are defined in the *Data Feeders*.

Measurement Reference Point (MRP) naming scheme

This is the formal description of how the measurement point name is built, that is, by concatenating the values of Data Feeder properties and fixed strings.

Mobile virtual network operator

A mobile operator that does not own its own spectrum and usually does not have its own network infrastructure. Instead, MVNOs have business arrangements with traditional mobile operators to buy **minutes of use** for sale to their own customers.

MRP

See Measurement Reference Point.

MVNO

See mobile virtual network operator.

Parameter

A value or set of values that are periodically updated and that help determine the quality of service.

Parameter objective

A set of objectives for the parameters belonging to a service.

Property

Special static parameters that are given a value only when an instance of an OpenView Service Quality Manager Object is created. For example, a Service Component can have a property called “location”.

QoS

See quality of service.

Quality of service (QoS)

The ITU-T has defined quality of service as "the collective effect of service performances that determine the degree of satisfaction of a user of the service".

Service

A Service is a set of independent functions (Service Components) that consist of hardware and software elements and an underlying communications medium. A Service can include anything from a single leased-line service, to a complex application, such as vision conferencing.

Service availability

A measurement made in the context of a service level agreement that is expressed as a percentage. This percentage indicates the time during which the service is operational at the respective service access points.

ServiceCenter Repository

The ServiceCenter Repository is the storage center for all Service Quality Manager data. It receives data from the various Service Quality Manager interfaces and each interface can request information from the Repository.

Service component

An independent function that is part of a service, such as a hardware or software element, or the underlying communications medium.

Service component instance

The instance of a Service Component Definition that is active in the network, such as an instance of the IPAccess Service Component definition called “pop”.

Service level (SL)

Defines Service Parameters and operational data enforced by the Service Level Agreement (for example, Max Jitter < 10 ms).

Service level agreement (SLA)

There are two type of Service Level Agreement, the Customer Agreement: a contract between a service provider and a customer, which specifies in measurable terms what the service provider supplies to its customers, and the Operational Service Level Agreement, which specifies in measurable terms the operational levels of the Service. A service level agreement is composed of individual objectives.

Service level objective (SLO)

The set of objectives for the parameters belonging to a Service or Service Component.

Service parameter

See *parameter*.

Service provider

A company or organization that provides *services* as a business. Service providers may operate networks or may integrate the services of other providers.

Service instance (SI)

The instantiated service definition that is active in the network, such as an instance of the video service definition called “Paris”.

Service instance group (SIG)

A **group** of *service instances* against which the *service availability* must be reported. Each *service instance* belongs to one or more Service Instance Groups and each SIG contains at least one Service Instance. The relationship between the SIG and the Service Instances is defined in their *service level agreement*.

Service quality parameters

They represent *computed* and *collected* parameters

SI

See Service Instance.

SIG

See Service Instance Group.

SL

See Service Level

SLA

See Service Level Agreement.

SLO

See service level objective.

Subscriber

The entity responsible for the payment of charges incurred by one or more users.

User

An entity designated by a customer to use the services of a telecommunication network, such as a person using a UMTS mobile station as a portable telephone.