Multi-Purpose Models for QoS Monitoring

Stefan Wallin, Viktor Leijon.
Luleå University of Technology LTU Skellefteå, SE-931 87 Skellefteå Sweden
stefan.wallin@dataductus.se, leijon@csee.ltu.se

Abstract

Telecom operators face an increasing need for service quality management to cope with competition and complex service portfolios in the mobile sector. Improvements in this area can lead to significant market benefits for operators in highly competitive markets. We propose an architecture for a service monitoring tool, including a time aware formal language for model specification. Using these models allows for increased predictability and flexibility in a constantly changing environment.

1. Introduction

Service operators face many new challenges in network management [1]. Among the most important trends is the increasing move towards service centric management. It is necessary for an operator to deliver predictable quality of service.

There are currently few useful solutions for dealing with QoS for a large number of services types, service instances and users.

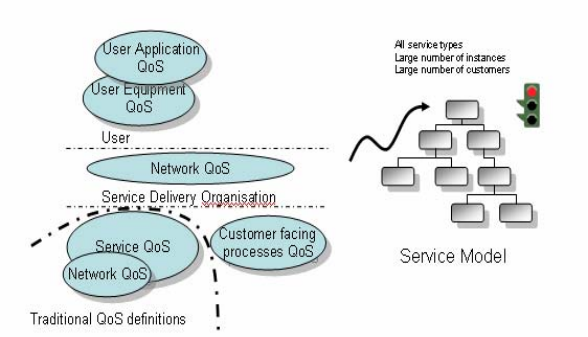

Figure 1: Overall goal for QoS monitoring

The picture above illustrates the overall QoS problem. To get a good general measure of QoS, multiple parameters need to be taken into account. It is fairly easy to have a state for a single service at a given time for one customer. But a large telecom operator needs to have an overall picture with support for different views. Different service views can include geographical, customer-based, SLA service etc. With such service views an operator can connect technical service quality with business goals.

We are not trying to define individual QoS parameters for specific network technologies or services. Neither are we trying to define "good service quality". These areas are already well covered by individual standards, research and products. Instead we are trying to provide general capability for building large scale, multi purpose service models

This paper defines the following components for a service management system:

- We give a simple definition of Quality of Service in section 2.1.
- Using this definition in Section 2.2 we examine what kinds of scenarios a service monitoring tool must be able to handle.
- The outline for the architecture of a service monitoring system is given in section 3.1.
- Finally, in Section 3.2, we present a novel service modeling language, with built in capabilities for dealing with time and relationships between objects

2. Overview

2.1. Quality Of Service

There are several different interpretations of what Quality of Service means, each with different scopes.

In the IETF the term Quality Of Service typically refers to technologies to achieve "good quality". QoS in this sense is a means to prioritize network traffic, manage bandwidth and congestion, and to help ensure that the highest priority data gets through the network as quickly as possible. The IETF uses the following definition of QoS [2]: "A set of service requirements to be met by the network while transporting a flow". Important solutions from this domain are IntServ [3], DiffServ [4], MPLS [5] and Policy solutions [6]. These are all important QoS solutions to achieve good service quality with Internet protocols. Similar solutions exist for other domains like fixed networks, 2G/3G networks [7] and specific services such as VoIP, customer care processes, etc. These QoS techniques are aimed at controlling and monitoring QoS in an *objective* way.

The ITU uses a more end-user focused definition of the term [8]: "the collective effect of service performance which determines the degree of satisfaction of a user of the service". Using this definition takes one step further in that it includes the *subjective/perceived* measurements by an end-user. Casas [9] presents a method to measure perceived Quality Of Service and the relationship between subjective and objective measurements.

The relationship between service quality and customer loyalty in general is discussed by Bloemer [10]. He also enhances the quality of service definition into a third level: *"service quality is a function of the difference scores or gaps between expectations and perceptions (P - E)"*.

Why is an overall picture of quality of service needed?

- **SLAs**: service providers want to sell SLAs where they promise a certain service quality. These promises should be expressed in a formal way that can be measured. SLAs often includes several kinds of QoS parameters ranging from technical parameters like jitter and delay to more indirect parameters like customer care or process metrics
- **Service quality**: service providers are often overwhelmed with individual QoS parameters, but these are not integrated into an overall service view. It is hard to get information about current, past and future service quality.
- **Dynamic networks**: services are carried over different networks like xDSL, WLAN, 3G. In order to support seamless roaming and still keep the perceived service quality we need calculations which span several different domains.
- **Quality Assurance**: organisations are currently introducing QoS mechanisms in the traditional sense, but not always monitor the quality of the services.
- **QoS definition framework**: new applications and services are constantly being defined. Each of these new services will have its own specific QoS parameters. There is a need for a common understanding about which these parameters are, and what their meaning is.
- **Governmental authorities** have a growing need to monitor the quality of operators from a functional and service perspective.

For a more in-depth coverage of the motivations behind an overall solution for monitoring QoS see for instance Espvik [11] or Räisänen [12].

We conclude that the definition of QoS must be neutral and generic. It must encompass statements such as: *- What is the quality of the Swedish GSM network? - What was the quality of the Swedish GSM network last Christmas Eve? - What is the quality of all the services provided to customer B? -How did a failure on a specific base station affect the GSM service in Stockholm at 10.00 AM yesterday?*

Considering these requirements have lead us to adopt the following simple definition:

> *Quality of Service is defined as a function of service parameters*.

These functions and the interpretation of the parameters are part of a specific model, and formulated by experts on the particular service. In some sense the function should represent the *"degree of conformance of the service delivered to a user by a provider in accordance with an agreement between them" [8]*.

2.2 Use Cases

In order to make more concrete the kind of functionality a system for service modeling must have we present a few important use cases.

Many of these use cases are represented by complicated models, and require the combination of several different areas of expertise. A service modeling language should be able to express them in the way the domain experts defines them.

2.2.1 An end-customer view. This is the traditional view in QoS, examining the quality of a single flow, for a single customer at a single point in time. This is the most basic requirement, and has been the focus of much previous research [13].

The requirement here is to be able to determine what QoS the customer is getting, and to compare that to the QoS the customer has purchased. How to compute the relevant parameters is service dependent.

2.2.2 A customer care view. If we instead view a customer care organization the picture changes slightly. The customer care representative is not as interested in the separate quality measures, since she may have responsibility for a hundred different types of service delivered to a million different customers.

For customer care, an aggregate picture is required. Something that gives a picture of the total quality delivered. There is also a need for drill down capability, to explore the root cause for customer complaints.

2.2.3 A marketing view. A marketing organization has a very different focus, it needs to be able to predict customer loyalty, based on service price and service quality.

This requires a model of customer behavior, and of how quality relates to loyalty [10]. This means that the same performance indicators could be used for marketing as for the end user view. But how they are viewed could be very different, and the model must be able to accommodate this.

2.2.4 A technical view. In contrast, a field engineer has a view that is relatively simple; all he wants to know is what he should fix next. He needs to be able to extract the highest-priority fault within his area of responsibility.

The challenge here is to calculate a priority from the model. This requires that QoS problems be mapped to a single piece of hardware.

2.2.5 A what-if scenario. Another interesting possibility is to examine what-if scenarios.

You should be able to experimentally change parameters and see their effect. If we cancel the lease on a backup link, for example, what would the effect on customer satisfaction be? What would the financial effect be like?

3. Reference Architecture for a QoS Monitoring Solution

3.1 Overall

In order to be able to fulfill these use-cases, we are suggesting a three pronged solution consisting of:

1. *A modeling language*, with sufficient power to express the complex models needed for the use-cases. This language must be able both to express the way that parameters are computed from each other and to express the structure of the service model.
2. *An analytic engine*, which can execute the modeling language and compute the values of all the parameters. This engine needs to have the appropriate interfaces, and be parallelizable so that it can be implemented in a scalable and fault-tolerant fashion.
3. *Information visualization systems*, interfaces to extract and present the relevant data from the analytic engine. This can be accomplished by a combination of integration with report generators and a general, data driven interface.

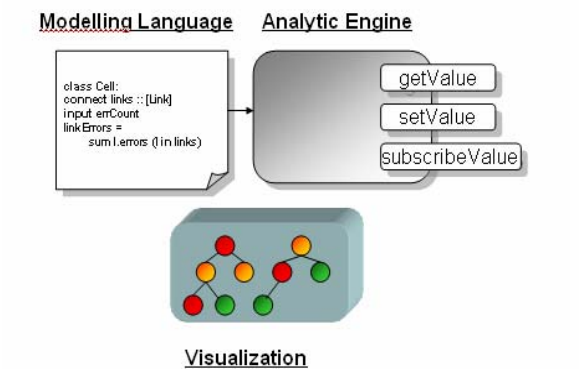

Figure 2: QoS Monitoring Framework

It is imperative that the system has enough power to express the models, and that it is simple enough that integrating it into the support systems of an operator is feasible. This forces us to make a design trade-off between power and flexibility.

There are already languages and system to create models such as Modelica [14] which is targeted at modeling physical systems. We have a different scope, but we note that just as in models of the physical world, the concept of time will be very important in our system.

3.2 A Language for Service Modeling

We use a tailor-made programming language for defining services and service level agreements. This enables us to create services using well understood methods of program construction. The language has two main purposes: first it will define the structure of the model, and second, it will define the relationship between parameters and determine how they will be computed.

We propose a simple, pure, functional language for defining calculation rules, with the following key features:

1. The language is object oriented to facilitate the object oriented structure of a service model.
2. We use type-inference to make the service modeling less error-prone and to facilitate composition of components.
3. Due to the nature of service modeling, the programming language must be able to treat time as an integral part of the syntax: all variables are seen as arrays, indexed by a time stamp.
4. It is possible to use the time-index syntax to retrospectively change the value of variables.
5. List comprehension and an extensive set of built-in functions provide the power needed to express complex models.

To make the language more concrete we present a simplified example taken from a model of a cellular network. The first class, `cell`, defines what properties we associate with a cell; and that it has a single measurement input (`errCount`). The definitions state how the parameters `errRate` and `linkErrors` should be computed from other parameters. Note the `@` sign, which is used to indicate access to a time indexed value.

The second definition, `CellSL`, defines a service level - a promise on the behavior of the underlying component - which encompasses the cell. It uses the parameters from the cell to form a view of the component. In effect saying "If we apply a Service Level on this cell, this is what the status would be like?".

Note that the service level is parametric with regards to the number of errors allowed. We provide the specialization `BronzeCellSL`, which gives specific values.

Finally we define the relationships between Cell and Links, Cell Service Levels and Cells. When relationships are defined it establishes implicit attributes: Class1 <=>* Class2 gives the implicit class2 attribute in Class1 (list value) and the class1 attribute in class2

Definition:

```
class Cell:
   input errCount
   errRate =
       (errCount@NOW - errCount@(NOW-10m))/10
   linkErrors =
       sum l.errors (l in link)
class CellSL:
   properties maxOwnErrors, maxLinkErrors
   status = worst errStatus  linkStatus
   errStatus  =
       linearThreshold cell.errRate 0
                   maxOwnErrors
   linkStatus =
        simpleThreshold
             cell.linkErrors maxLinkErrors

def BronzeCellSL =
   CellSL( maxOwnErrors=>10,
            maxLinkErrors => 15)

Cell <=>* Link
CellSL => Cell
```

Instantiation of the classes are separated from the definition. The example below shows the naïve case when objects are created one-by-one.

Instantiation:

```
cell1 = new Cell
link1 = new Link
cellsl1 = new CellSL
connect cell1 link1
```

3.3 The Engine

The language requires a special runtime environment, which is responsible for marshaling inputs and outputs from the language and evaluating the expressions.

All the services that the engine provides are related to variables, the main services are:

- **getValue**(variable,time)
- **setValue**(variable,time)
- **subscribeVariable**(variable,startTime,stopTime)

Since we define a purely functional language, there can be no side-effects of the computations except for the updating of variables. This means that the engine can utilize laziness to avoid computing unnecessary values, as well as caching to avoid re-computing values. The laziness and the possibility to implement parallelism in the engine will enable the engine to scale up to the size needed. The engine is designed to be able to handle systems of up to 2 million objects, with up to 5 million parameter calculations per minute.

Although we focus on monitoring and visualizing the status of the services, we consider automatic actions to be an important part of a service management solution. These automatic actions include controlling the network or notifying operations. This will be possible by subscribing to the appropriate parameters.

3.4 The Visualization Tools

The open design of the engine makes it possible, and desirable, to build many kinds of interfaces towards the system to extract relevant information. The initial effort focuses on two major parts, a data driven data visualization tool, where you can examine objects on a simple dashboard to display their associated variables, and, drill-downs in the object hierarchy. The second part is the integration with a report generation tool, to allow periodic reports on the status of a system.

4. Related Work

Previous efforts on QoS monitoring have been focused primarily on frameworks, which we explore in

section 4.1. There have also been some work on modeling, which we present in section 4.2

4.1 Quality of Service Frameworks

There are several different approaches to managing and measuring service quality. We have divided them into three categories.

- *Control:* frameworks that actually try to manage resources in order to achieve a QoS Level
- *Single – flow measurement:* focused on measuring a specific traffic flow.
- *End-to-end QoS Monitoring:* trying to measure and monitor the overall quality of service.

4.1.1 Control Frameworks. The primary QoS techniques developed by the *IETF* are *IntServ*, *DiffServ*, and QoS techniques for *MPLS* [3-5]. Another class of QoS control frameworks rooted in the IETF are different flavors of *Policy-based management* (PBM) [6]. PBM provides a way to allocate network resources, primarily network bandwidth, QoS, and security, according to business policies. The European Commission has funded two programs focusing on QoS; *AQUILA* [15], [16] and *TEQUILA* [17]. Both attempt to define service definition and traffic engineering tools for the Internet to obtain quantitative end-to-end Quality. They are more focused on IP services and actually managing QoS where we are defining a framework for any service and limited to monitoring. Tequila tried to establish an IETF working group on formal service level specifications [18].

4.1.2 Single-flow measurements frameworks. *Probes* simulate and measure end-users behavior using the network as a black-box in order to estimate the actual delivered network service quality. Probes exists for mobile services as well as IP protocols, all have the limitation of measuring a single user at a defined location.

4.1.2 End-to-end Monitoring Frameworks. An obvious way to estimate the end-to-end QoS is of course to involve the user. *MOS* [19] (Mean Opinion Score) is probably the most common method for voice QoS in this area.

Eurescom funded a end-to-end monitoring effort; *eQOS* [11][20]. eQOS gives an excellent background description of what we are trying to achieve. *TAPAS* [21] is a similar effort applying SLAs and QoS to application server solutions. It uses SLAng, see below, to define the SLAs. Services have a complex life-cycle, where monitoring QoS of a deployed service is only one phase. The EU FP6 project *MobiLife* [22] is studying packet-based services from an end-user viewpoint and addresses the whole life-cycle for services.

Service providers are looking for technical solutions for monitoring their operational service quality as well as to be able to sell and monitor customer specific SLAs. This has led new products in this area, for example HP OpenView SQM [23], Digital Fuel Service Flow [24], Managed Objects SLM [25]. These products are quite successful in collecting events and measurements and monitoring SLAs. Most of the tools have weaknesses in the service modeling area; they deploy various UML flavors or simple object modeling techniques which allow for very little static analysis, for instance to be able to determine dependency graphs to facilitate lazy computation of parameters. Our work aims to improve service modeling and computational aspects such as time based calculations, and maintaining the state of a massive number of services and users.

4.2 Service Modeling

Probably the most extensive standards effort within service modeling is *CIM* [26]. CIM uses UML as the modeling language, and defines an XML mapping to exchange the models. The key strengths in CIM are the modeling guidelines and patterns that are used by the standard models and by enterprise extensions. The most important implementation is Microsofts implementation of CIM in their solution for management of Windows, *"WMI"* [27]. As pointed out by Microsoft in their System Definition Model [28] CIM can *"become unwieldy if used to describe the abstracted virtual constructs of a distributed system"*. CIM is aimed more at instrumentation rather then end-to-end service modeling. IETF is reusing the CIM model in the *IETF Policy Framework WG*, [29]. Many of the industry players behind CIM, are now joining up behind Service Modeling Language [30]. Based on the experience from CIM it does not start with large models. An important part of any service model covering a defined QoS is of course the SLAs. SLAng [31] is a language focused on defining formal SLAs in the context of server products like J2EE and typical ASP environments with web services, server applications. While CIM, SML and SLAng are rooted in the IT segment, TeleManagement Forum is trying to define a telecom related information model; the System Information Model, *SID* [32]. It is comparatively high level and models entities in telecom operators' processes. However, SID is being refined and moving

closer to resources by incorporating CIM.

5. Conclusion and Future Work

We have shown a feasible architecture for a service monitoring framework, and described a new approach to service modeling using a formal language with suitable characteristics, such as an inherent notion of time. Our current focus is on developing and studying the formal properties of the service modeling language, and examining what kind of static analysis we can perform. We will also implement and evaluate a prototype of the framework. Another exciting angle is to study how service models are created and investigate what kind of generalizations are possible in the form of design patterns for models. We are also interested in finding characteristics for bad and good models respectively.

6. References

1. Wallin, S. and V. Leijon, *Rethinking Network Management Solutions*, in *IT Professional*. 2006. p. 19-23.
2. Crawley, E., *RFC2386, A Framework for QoS-based Routing in the Internet*, B.R. R. Nair, H. Sandick Editor. 1998, IETF.
3. Braden, R., *RFC 1633, Integrated Services in the Internet Architecture: an Overview*, S.S. D. Clark, Editor. 1994, IETF.
4. Blake, S., *RFC 2475, An Architecture for Differentiated Service*, M.C. D. Black, E. Davies, Z. Wang, W. Weiss, Editor. 1998, IETF.
5. Rosen, E., A. Viswanathan, and R. Callon, *RFC3031, Multiprotocol Label Switching Architecture*. 2001, IETF.
6. Verma, D.C., *Simplifying network administration using policy-based management.* Network, IEEE, 2002. 16(Mar/Apr 2002): p. 20-26.
7. 3GPP, *3GPP TS 23.107, Quality Of Service (QoS) Concept and Architecture, version 5.13*. 2004.
8. ITU-T, *ITU-T Recommendation E.860, Framework of a service level agreement.* 2002, ITU-T.
9. Casas, P., *User Perceived Quality of Service in Multimedia Networks: a Software Implementation*, in *MVD Telcom*, I.I. Diego Guerra, Editor. 2006: Montevideo Uruguay.
10. Bloemer, J., *Linking perceived service quality and service loyalty: a multi-dimensional perspective.* European Journal of Marketing, 1998. 33.
11. Espvik, O., *A Common Framework for QoS/Network Performance in a multi-Provider Environment.* 1999, Eurescom.
12. Räisänen, V., *Service Management Evolution*, in *IST Mobile and Wireless Communications Summit*. 2005: Dresden, Germany.
13. Galetzka, M., *User-Perceived Quality of Service in Hybrid Broadcast and Telecommunication Networks*, in *5th Workshop on Digital Broadcasting*. 2004.
14. association, M. *Modelica Homepage*. 2006 [cited 2006 November 24]; Available from: http://www.modelica.org/.
15. Hermann Granzer, e.a., *Aquila: Adaptive Resource Control for QoS Using an IP-based Layered Architecture*. 2003.
16. Koch, B.F. and H. Hussman. *Overview of the project AQUILA (IST-1999-10077)*. in *Architectures for Quality of Service in the Internet*. 2003. Warsaw, Poland: Springer.
17. Asgari, A., et al., *A Scalable Real-time Monitoring System for Supporting Traffic Engineering*, in *IEEE Workshop on IP Operations and Management (IPOM 2002),*. 2002: Dallas, USA.
18. Goderis, D., et al. *Service Level Specification Semantics and Parameters*. 2000 [cited; Available from: http://www.ist-tequila.org/standards/draft-tequila-sls-00.txt.
19. ITU-T, *Recommendation P.800.1 (07/06) Mean Opinion Score (MOS)*. 2006, ITU.
20. Jensen, T., *Managing Quality of Service in Multi-Provider Environment*, in *Telecom 99*. 1999: Geneva, Switzerland.
21. Lodi, G., et al. *Experimental Evaluation of a QoS-aware Application Server*. 2005.
22. Mrohs, B., C. Rack, and S. Steglich. *Basic building blocks for mobile service provisioning*. 2005.
23. HP. *Service Quality Manager*. 2006 [cited; Available from: http://openview.hp.com/products/sqm/index.html.
24. DigitalFuel. *Service Flow*. 2006 [cited; Available from: http://www.digitalfuel.com/products/sla-management.aspx.
25. ManagedObjects. *Business Service Level Manager*. 2006 [cited; Available from: http://www.managedobjects.com/solutions/bslm.jsp.
26. DMTF, *Common Information Model.* 2006.
27. Microsoft. *WMI - Windows Managment Instrumentation*. 2006 [cited; Available from: http://www.microsoft.com/whdc/system/pnppwr/wmi/default.mspx.
28. Microsoft. *System Definition Model*. 2006 [cited; Available from: http://www.microsoft.com/windowsserversystem/dsi/sdm.mspx.
29. Moore, B., et al., *RFC3060 Policy Core Information Model -- Version 1 Specification*. 2001, IETF.
30. Boucher, J., *Service Modeling Language Specification*. 2006, Microsoft.
31. D. D. Lamanna, J.S., and W. Emmerich. *SLAng: A language for service level agreements*. in *9th IEEE Workshop on Future Trends in Distributed Computing Systems*. 2003: IEEE Press.
32. TeleManagementForum, *Shared Information&Data SID Model*, in *Business View Concepts, Principles and Domains*. 2005, TeleManagement Forum.