Roland Volk

Integrated Service Management with OSS/J

Integrated service management is a mandatory prerequisite for communication service providers (CSPs) to successfully deploy and operate a competitive portfolio of customer services. Unfortunately, missing open and standardised integration approaches led to proprietary solutions which were expensive in terms of capital and operational expenditures. With the advent of TMF's NGOSS and OSS/J Initiative, CSPs have now the opportunity to migrate smoothly towards a standardised integrated service management solution. This paper discusses the technological aspects of the combined NGOSS-OSS/J approach in more detail.

Introduction

Communication service providers (CSPs) are rapidly extending their portfolio of services to address the growing need for custom-made, innovative and compelling services while maintaining committed to service quality levels for increased customer retention. On the other hand, the overall economic situation in the telecommunication market calls for reduction of capital and operational expenditures and, thus, forces CSPs to achieve a higher level of automation of their key operational processes spanning a multitude of existing business and operation support systems (B/OSS).

In particular, an integrated service management (ISM) domain is key to the provision and operation of competitive services; for example, voice over IP (VoIP). This domain is characterised, besides the growing number of users and services, by a plethora of complex network technologies and standards and a large number of specialised network management/service management (NM/SM) applications as well as the constant challenge to maintain good service quality levels and high network availability, reliability, and security.

This paper describes an approach for building an ISM solution which follows the B/OSS industry's general move towards component-based, service-oriented B/OSS architectures as proposed by the TeleManagement Forum's (TMF) NGOSS specifications (New Generation Operations Systems and Software). The proposed approach is based on the OSS through Java™ (OSS/J) Initiative's range of open and standardised B/OSS application programming interfaces (APIs), which is the first technology-specific implementation of NGOSS.

A typical ISM solution consists of three distinct domains with respective tasks:

- service fulfilment (manage order entry, fast modification of services, rapid service activation and delivery, track and manage inventory);
- service assurance (ensure service availability, monitor performance, minimise time to repair, manage service level agreements); and
- service billing (usage-based and content-based billing, capture usage data for billing, capacity planning, and marketing, analyse subscriber behaviour).

To highlight the main achievements of the combined NGOSS-OSS/J approach, this paper focuses on the service assurance domain. It pinpoints the benefits of the presented approach through industry experiences with OSS/J technology gathered in a recent NM/SM integration project at Vodafone D2, Germany.

Integrated Service Management—A Syntactic and Semantic Challenge

Integration of existing B/OSS systems means always resolving two challenges in order to automate system-spanning business processes:

- B/OSS systems usually exchange data (business objects) using different proprietary interfaces (syntactic challenge), and
- B/OSS systems need to 'understand' data received from other IT systems (semantic challenge).

While efforts to resolve the syntactic challenge have led already to several technological levels of evolution in enterprise application integration (EAI)—for example, database level integration, point-to-point interface integration, message-oriented middleware (MOM)—open and standardised integration approaches emerged only recently through activities from the TMF and particularly the OSS/J Initiative. The section 'OSS through Java Initiative' discusses this in more detail.

Unfortunately, the semantic challenge is even more complex than the syntactic challenge. The reason for this is the fact that almost every B/OSS has an own proprietary

Author
Roland Volk
Managing Director
IP VALUE GmbH,
Stockholmer Allee 24,
44269 Dortmund,
Germany
Tel: +49 2 31 4 76 42 2 17
Email: roland.volk@ip-value.de

database with its own proprietary data model and business objects. As a result, CSPs' IT landscape consists of a multitude of system-specific (sub-) data models and business objects; that is, they operate a multitude of databases in a distributed IT environment. However, unified data semantics between involved B/OSS is the crucial factor for a successful integration project with the goal of automating system-spanning business processes.

Simple questions in the service management domain such as:

- What type of service is affected by a network outage or performance issue?
- What customers or, at least, types of customer are affected?
- Are there redundancy options available?
- Are claims for compensation to be expected due to breaches of service level agreement (SLA)?

are not answerable in CSPs' daily operation unless common data semantics between involved NM/SM-specific IT systems are achieved.

For this reason, TMF defined the *Shared Information/Data Model* (SID) for the telecommunications industry as part of the overall NGOSS specification work, which has now reached maturity. Furthermore, the OSS/J Initiative's specification of the data semantics for their functional APIs (see later) emerged directly from the SID definitions (see References 2 and 4 for detailed specifications).

However, the pure existence of a common (meta-) data model does not answer the question how to implement such a model into a CSP's IT environment while supporting existing system-specific data models and databases.

TeleManagement Forum's Approach for a Service-Oriented Architecture (SOA)

The current industry trend towards a component-based, service-oriented architecture is backed by recent advancements in the standardisation area: TMF's NGOSS programme[1] is a business-oriented, technology-neutral framework specifying a methodology for building OSS components. Additionally, NGOSS' associated Enhanced Telecom Operations Map (eTOM)[3] defines the basic OSS concepts and process framework which is already widely deployed in the industry.

An overview of the structure of the NGOSS architecture is provided in Reference 5. Generally, the NGOSS work covers four distinct areas:

- systems analysis and design (Shared Information/Data Model (SID)),
- business analysis and design (Enhanced Telecom Operations Map (eTOM)),
- solution analysis and design (contract interface and technology-neutral architecture), and
- solution conforming testing (compliance tests).

The resulting characteristics of an NGOSS system are among other things:

- the usage of a common information model for enabling integration and interoperability;
- the separation of the hard-coded behaviour of components from software that automates business processes across the components—that is, an NGOSS system should be composed of defined services that can be orchestrated using scripting/process management technologies;
- in particular, a business process model may invoke lower-level business process models;
- functionality of business applications is accessible through NGOSS contractual interfaces; and
- existence of distribution and transparency services for supporting interaction patterns between components (naming, repositories, registration services, service location services).

While focusing on business aspects, the challenge of the NGOSS programme is its general nature and complexity leading to implementation and interoperability issues. Therefore, the OSS/J Initiative took another but complementary approach and focused on implementation aspects with the ultimate goal of promoting the delivery of reusable B/OSS solutions to CSPs.

OSS/J is an important step in B/OSS standardisation as it extends TMF's NGOSS programme to cover the B/OSS system implementation aspect by defining functional APIs with implementations based on mainstream IT technology. OSS/J leverages Java™ and J2EE technology (Java 2 Enterprise Edition) as the basic middleware to support tightly coupled systems integration used within an enterprise as well as loosely coupled systems integration (used within an enterprise and for business-to-business applications) with extensible markup language (XML) and web services. In fact, OSS/J is the first technology-specific implementation of NGOSS.

Particularly, with its inherent encapsulating approach for legacy B/OSS systems OSS/J offers a smooth migration strategy towards an NGOSS compliant architecture.

A detailed study of how the OSS/J programme complements TMF's NGOSS technology neutral architecture in terms of architectural principles, contract schemata, and information-modelling techniques can be found in Reference 6.

To summarise, recent trends in B/OSS standardisation lead to more pragmatic implementation of process-integrated B/OSS. NGOSS' goal is to define the basic concepts for the B/OSS industry while OSS/J complements NGOSS by establishing the specific IT technology approach for the actual implementations.

OSS through Java™ Initiative (OSS/J)

Introduction and Overview

Building on the success of Java 2 Platform, Enterprise Edition (J2EE™) technology in enterprise applications and e-commerce, the OSS/J Initiative is chartered to develop functional APIs for interchangeable, interoperable components that can be rapidly and cost effectively assembled into end-to-end telecommunications solutions that are easy to maintain and adapt to support new functionality. The Initiative's APIs are standardised under the latest Java Community Process℠ (JCP) programme. JCP deliverables for each application area consist of a specification, a reference implementation (RI), and a technology compatibility kit (TCK).

Beside the functional APIs, further major deliverables of the OSS/J Initiative are the Common API[8] and Design Guidelines (DG)[7] which ensure quick and uniform integration of all applications developed according to OSS/J specifications. By using the OSS/J DG and Common API implementation, it is possible to efficiently implement further future-proof software components. Such software components will be compliant with existing OSS/J standards and easily adaptable to evolving OSS/J standards. OSS/J also supports emerging web services approaches to CSP's integration and automation challenges (see later).

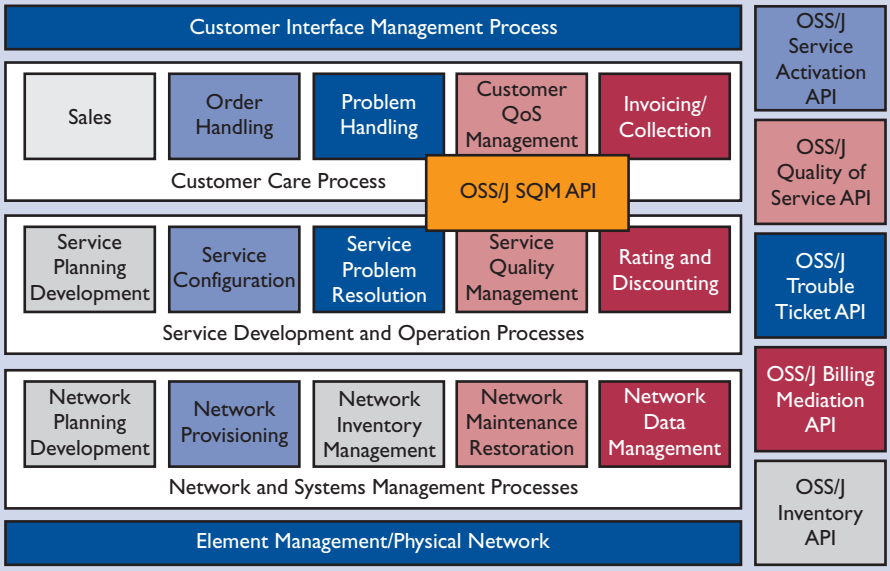
The benefits of applying OSS/J APIs are increased vendor interoperability, cost reduction when integrating existing systems and applications, faster time to market for new services, better investment protection, a more rapid deployment of new B/OSS solutions, as well as reusable software components for 'point-to-multi-point' interfaces. Detailed information can be found at http://www.ossj.org.

OSS/J's functional APIs for B/OSS integration

OSS/J Initiative's work today comprises the following functional APIs which are directly derived from the TMF eTOM processes[3]:

- Service Activation API[9]: deals with any type of orders and services;

Figure 1 Mapping of eTOM processes and OSS/J APIs



- Quality of Service API[10]: performance, threshold, and fault monitoring of any data source;
- Trouble Ticket API[11]: deals with any type of trouble ticket and defines respective events;
- Billing Mediation API[12]: deals with any type of billing data and defines respective events; and
- Inventory API[13]: common access to any type of inventory data (product, service, resource).

Relevant APIs for integrated service management including the preliminary Service Quality Management API[14] are discussed in more detail later in the section on service modelling.

Figure 1 identifies how the OSS/J core APIs map to the TMF eTOM processes (operations part only).

The main difference in scope between the OSS/J roadmap of APIs and the eTOM is that the roadmap defines the APIs which are essential to automatic, flow-through service management. In essence, the APIs cover a subset of the eTOM which identifies general processes regardless of whether they are automatic or must be accomplished by humans.

Principles of OSS/J-based integration

OSS/J-based integration follows the principle of functional decomposition (that is, separation of concern) between B/OSS systems to increase flexibility and adaptability and decrease maintenance cost and vendor dependencies. The functional decomposition is achieved with separated OSS/J API and adaptor components which support encapsulation of existing and new (non-OSS/J-compliant) systems.

The respective OSS/J API component provides common, functional attributes and methods agreed by global telecommunications-specific focus groups, while the adaptor component is responsible for mapping of proprietary attributes to the OSS/J attributes. This approach results in a completely reusable API component allowing the implementation of 'point-to-multi-point' interfaces and minimised adaptation effort in the respective adaptor. Since each adaptor requires knowledge of only one of the B/OSS systems (total system independence), integrating another system comes down to adapter replacement. Figure 2 illustrates the OSS/J integration principles with local attribute mapping and complete functional decomposition between two B/OSS systems.

Every OSS/J API supports meta-data with respective query mechanisms and is extensible. The term *meta-data* refers to the OSS/J core information model of shareable data transfer objects (core business entities (CBE)[4]) directly aligned with TMF SID. In fact, the extension capability of each OSS/J API is the key to solve the semantic challenge described earlier in the section on integrated service management; that is, it enables the integration of existing B/OSS systems with their respective proprietary data models in a step-by-step approach.

While OSS/J APIs on the one hand describe and implement CSP-specific semantics—that is, specify the implementation of CSP-specific managed entities (trouble ticket, product, order, SLA, fault, etc.) and methods (create, read, update, delete, query)—they have to provide on the other hand appropriate communications mechanisms for different business scenarios (see Figure 3).

Knowing that business-to-business (B2B) transactions will be done through XML-related technologies, such as electronic business XML (ebXML), the OSS/J API

Figure 3 OSS/J integration concepts

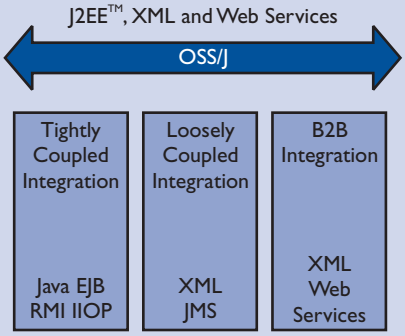
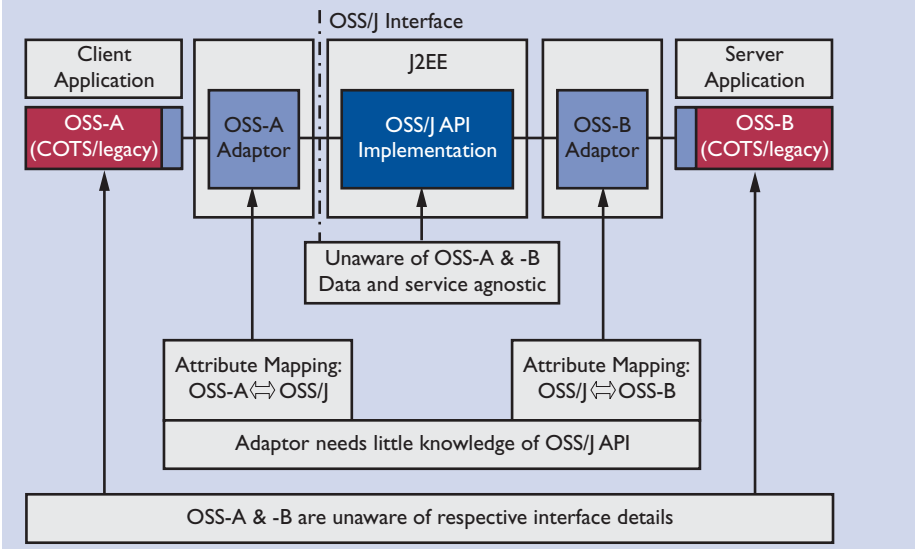


Figure 2 General OSS/J component architecture



design guidelines have unified the EJB interfaces with the XML interfaces. There is therefore an uniformity across the JVT session interfaces and XML messaging interfaces for each OSS/J API specification. The XML message payloads are specified in such a way that they are independent of message packaging and transport, potentially allowing the payloads to be bridged between SOAP, JMS, and other messaging services. Within a J2EE container, JMS is recommended for packaging and transporting the XML message payloads. SOAP is the message packaging and transport selected by ebXML for B2B transactions.

Service Modelling as a Basic Building Block of Integrated Service Management

A general requirement and a basic building block of integrated service management is the modelling of services and the associated inventory. CSP's services offered to customers—that is, product offerings to customers—are usually built from basic technical services delivered by infrastructure elements. Examples are mobile network services such as GSM, GPRS, UMTS and server-based services such as VoIP or WAP, SMS, MMS in the mobile communications landscape. Those basic services are configured, amended with quality metrics, and bundled to meet requirements of customer segments (private, business, etc.) or individual customers (usually corporate customers) with their specific service-level expectations (platinum, gold, silver, etc.).

The information about customers, the products they have subscribed to, the basic services those products are built from, and the resources those services use is essential for many different business and operation support domains. Product management, customer care (CRM), service provisioning, billing and service management are among the most important areas. Those domains have the need to create, read, update, delete and query (CRUD&Q) respective inventory information often distributed over a variety of heterogeneous B/OSS systems.

Figure 4 Typical product, service, resource inventory actors

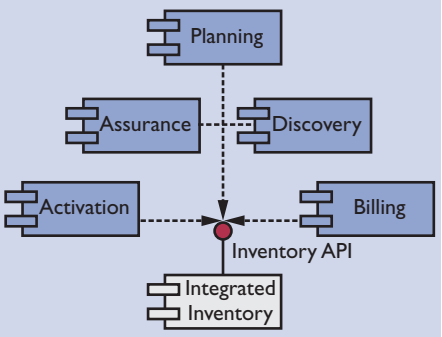
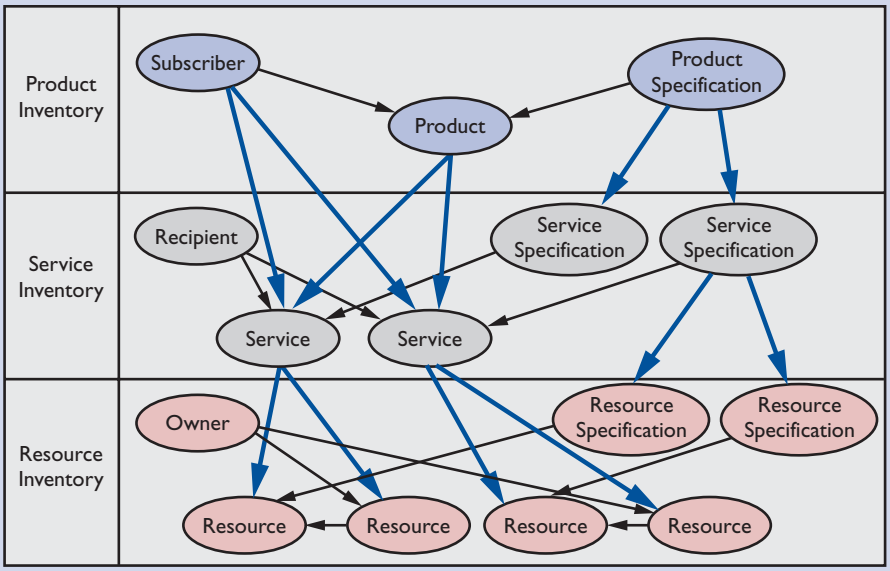


Figure 5 OSS/J Inventory API object model



An integrated service management system approach needs to support navigation through the inventory for top-down (subscriber – product – service – resource) and bottom-up analysis of product/service or resource issues. The OSS/J Inventory API provides the key methods to perform such operations southbound towards network-facing systems as well as northbound towards customer-facing systems. Figure 4 illustrates typical actors accessing respective inventory data.

OSS/J Inventory API

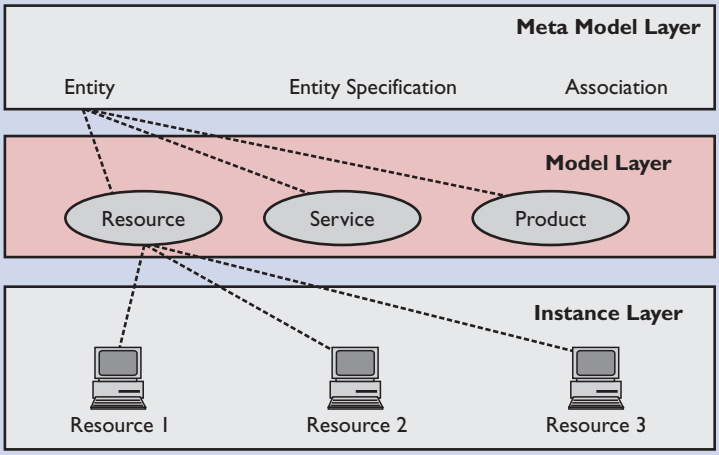
The Inventory API provides interfaces for the following operations:

- create, remove, update and query inventory entities, entity specifications and associations;
- invoke named queries and update procedures;
- perform meta-data queries;
- receive notifications of inventory events;
- monitor resource utilisation; and
- import and export inventory data.

These base operations allow client interactions for querying, allocating, reserving, planning, monitoring and updating inventory resources, services and products. The instances of inventory entities, specifications and associations as defined in the core models of each of the inventory functions are presented in Figure 5.

The framework for the modelling of inventory data is based on a three-layer architecture as shown in Figure 6. The preferred way to extend the model is through sub-classing of core model entities, specifications or associations. As a result, characteristics of any complex entity—for example, product bundles containing multiple products or product hierarchy—can be defined by entities

Figure 6 Meta modelling concepts



participating in associations with other entities.

The inventory meta model extends the Unified Modelling Language (UML) vocabulary with the following concepts:

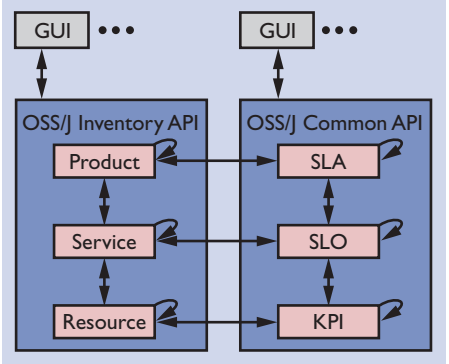
- Entity with stereotype 'Entity' (<<Entity>>). Entities are value type objects representing inventory concepts such as 'Product', 'Service' and 'Resource'.
- Entity Specification with stereotype 'Specification' (<<Specification>>). Entity Specifications are value type objects representing specifications of inventory entities.
- Association with stereotype 'Association' (<<Association>>). Associations are value type objects representing inventory associations; for example, 'ResourceSupportsServiceAssocValue'.

Application of OSS/J Inventory API for service quality management

The extensible object model of the OSS/J Inventory API can be implemented to suffice the typical object model requirements of service quality management solutions. In such a scenario, it is advised to use separate data model and interface implementations for the service quality management aspects and the inventory management aspects. An implementation based on the OSS/J Common API specification would be used to realise both the service quality management model and service quality management interface for the time the OSS/J SQM API specification is not yet finalised.

Figure 7 provides a high-level view of the proposed approach. The graphical user interfaces (GUIs) have to be implemented according to the different roles and functional requirements. The TMF SID should form the basis for the SQM object model and its associations to the inventory model. Using the OSS/J Common API specification ensures efficient migration to the planned OSS/J SQM API.

Figure 7 Application of Inventory API for SQM



Other OSS/J APIs for Integrated Service Management

OSS/J Trouble Ticketing API (TT)

The OSS/J Trouble Ticketing API provides interfaces, as specified by the OSS/J Design Guidelines, for creating, tracking, and deleting trouble tickets. The TT API enables creation and tracking of trouble tickets. It receives trouble ticket information either from the customer, or from service and network management applications such as service impact analysis, and root-cause alarm analysis/fault monitoring. It enables tracking of the problem until resolution and notification of clients when the problem has been resolved and the trouble ticket cleared.

The OSS/J TT API provides interfaces that allow clients:

- to create, remove and cancel trouble tickets;
- to change the values of trouble tickets; and
- to be informed of trouble ticket changes via notifications.

The OSS/J TT API specification is based on the following standards:

- TMF eTOM: Problem Handling,
- NMF 501: Customer to Service Provider Trouble Administration Business Agreement,
- NMF 601: Customer to Service Provider Trouble Administration Information Agreement,
- ITU-T X.790: Trouble Management Functions for ITU-T applications, and
- ITU-T M.3400: Trouble Administration function set group.

OSS/J Quality of Service API (QOS)

OSS/J QoS Fault Monitoring API (QOS-FM)

The OSS/J Fault Monitoring API, part of the QOS API, provides interfaces, as specified by the OSS/J Design Guidelines, which allow clients to collect and acknowledge alarms. The API enables reception of alarms, state changes, and threshold-crossing alerts from the network and maintains a list of active alarms. It also ensures availability of the most current view of the alarm state of the network.

The OSS/J FM API provides interfaces that allow clients:

- to acknowledge and un-acknowledge alarms in an alarm list;
- to query an alarm list;
- to query the alarm count from an alarm list; and
- to receive notifications for new, changed, and cleared alarms, alarm ACK state changes, and alarm list rebuild events.

The OSS/J FM API specification is based on the following standards:

- TMF eTOM: Resource Trouble Management, and
- ITU-T M.3400: Alarm Reporting function set; Alarm Summary function set; Log Control function set.

OSS/J QoS Performance Monitoring API (QOS-PM)

The OSS/J Performance Monitoring API, part of the OSS/J QOS API, provides interfaces, as specified by the OSS/J Design Guidelines, for creating and deleting metric and threshold objects. The API supports the collection of performance data from the network, setting thresholds, and generating/forwarding threshold crossing events. Data collected may be directly from devices (for example, a management information base (MIB) value) or may be a combination of metric values used in a calculation to create a more meaningful performance data value. The same principle applies to thresholds. Performance monitoring may be configured to generate aggregated thresholds out of single thresholds.

The OSS/J PM Measurement API provides interfaces that allow clients:

- to create, remove, suspend and resume measurement jobs;
- to get measurement jobs;
- to get supported report formats;
- to get supported observable objects;
- to retrieve measurement reports; and
- to use bulk/mass operations.

The OSS/J PM Threshold API provides interfaces that allow clients:

- to create, remove, suspend and resume threshold monitor jobs;
- to query threshold monitor jobs;
- to discover system performance measurements via a discovery mechanism that exposes a system's observable objects;
- to determine the supported granularities of a system based on a threshold monitor job configuration; and
- to use bulk/mass operations.

The OSS/J PM API specification is based on the following standards:

- TMF eTOM: Resource Performance Management;
- ITU-T M.3400: Network performance monitoring event correlation and filtering function set; Data aggregation and trending function set; traffic performance monitoring function set; NE threshold crossing alert processing function set; NE trend analysis function set; Performance monitoring and accumulation function set, etc.; and
- IPDR: Network Data Management Usage.

OSS/J Usage Monitoring API (QOS-UM)

The OSS/J Usage Monitoring API, part of the OSS/J QOS API, provides interfaces, as specified by the OSS/J Design Guidelines, for creating and deleting metric objects. The API supports the collection of usage data

from the network. Data collected may be directly from devices (for example, a MIB value) or may be a combination of metric values used in a calculation to create a more meaningful usage data value.

The OSS/J UM Measurement API provides interfaces that allow clients:

- to create, remove, suspend and resume measurement jobs;
- to get measurement jobs;
- to get supported report formats;
- to get supported observable objects;
- to retrieve measurement reports; and
- to use bulk/mass operations.

The OSS/J UM API specification is based on the following standards:

- TMF eTOM: Resource Data Collection & Processing; and
- ITU M.3400: Traffic Performance Monitoring Function Set; various in Usage Measurement Function Set Group.

OSS/J Service Quality Management API (SQM)

The OSS/J Service Quality Management API (still in the building phase through JCP) will provide interfaces, as specified by the OSS/J Design Guidelines, for querying, creating, updating and deleting service level specifications objects, service quality objective objects, and service quality report objects, as well as subscribing for notifications on object violation events and availability of new service quality reports.

The OSS/J SQM API enables definition and calculation of service-related quality objectives against pre-provisioned metrics. It enables correlation of service instance information, service alarms from service impact analysis, and network performance metrics from performance monitoring to assess availability and quality metrics, and create service-oriented threshold crossing events. Service quality measurements and traffic conditioning agreements (TCAs) available through the API are consumed by customer service level agreement (SLA) management to assess SLA violations.

The OSS/J SQM API supports and extends numerous interfaces from the OSS/J QOS API and it relies on the CBE service model as the core information model. In particular, the OSS/J SQM API provides interfaces that allow clients:

- to create, update, delete and query service level specification objects, corresponding to line-items on SLAs;
- to create, update, delete and query service quality objective objects (service objectives) against key quality indicator (KQI) parameters;
- to create, update, delete and query service quality report objects (historical reports on service quality objectives and/or service level specification evolution);
- to subscribe for notifications on object violation events (service level specifications or service quality objectives being compromised); and
- to subscribe for notifications on availability of new service quality reports.

The OSS/J SQM API specification will be based on the following standards:

- TMF eTOM: Service Quality Management;
- TMF 506: Service Quality Management Business Agreement;
- TMF GB923: Wireless Service Measurement Handbook; and
- TMF GB917: Service Level Agreement Management Handbook.

Indeed, whereas OSS/J QOS API provides performance thresholding, performance measurement and fault reporting capabilities, the OSS/J SQM API will focus on different capabilities such as:

- identifying service effecting parameters,
- defining service class quality indicators,
- reporting violations against service quality objectives on service instances, and
- combining raw network quality parameters into higher level (service level) performance parameters.

In return, the flow of information generated by the OSS/J SQM API can be used to report violations of SLAs northbound towards customer-facing systems.

Benefits of Applying OSS/J Technology

The major benefits of applying OSS/J technology as a component-based approach are:

- investment protection—greater use can be made from existing B/OSS applications, and there is no need for system replacement;
- step-by-step realisation—projects can be justified individually, nevertheless, contributing towards an overall strategic integration approach;
- integration effort can be reduced up to 60–70% due to reusability of components and reduced complexity (according to Reference 15);
- no need for an overall proprietary 'common data model' for integration of different B/OSS systems in different countries—instead, a standardised data model in combination with functional decomposition helps eliminating huge project start up costs; and
- reduced time to market—new third-party applications (once B/OSS are OSS/J compatible) can be integrated in 'days and weeks' instead of 'months and years', reducing time to market by up to 30% (according to Reference 15).

Industry Experience with OSS/J for Integrated Service Management

Vodafone, as one of the world's largest mobile telecommunications network companies, deploys advanced distributed, heterogeneous information and communication technologies as well as multi-vendor OSS applications in its complex customer and service-centric NM/SM domain. To cope with challenges in this domain arising from growing markets, ever-changing services and OSS technologies, Vodafone has initiated a project to redraw the Vodafone NM/SM IT architecture map to achieve a cost-effective approach to a clear business-oriented, integrated NM/SM solution.

Key objectives of this initiative are the reduction of the high integration cost for existing and new applications, and to obtain reusable NM/SM applications, building blocks and standardised interfaces, ultimately resulting in rapidly deployable NM/SM services.

Before starting implementation work, Vodafone D2 compared a traditional EAI approach with the new OSS/J-based approach in a feasibility study. The results were promising so that Vodafone decided to verify the inherent benefits of an OSS/J-based architecture by implementing a proof of concept (PoC) scenario for their trouble ticket management and service monitoring systems as part of their *Service Management Integration Architecture* (SeMIA).

Vodafone D2's *Service Monitoring System* (SMoS) is based on NETeXPERT modules by Agilent Technologies. SMoS collects data from the various element management and probing systems and initiates TT creation using SMoS alarm management as a TT client. The *Trouble Ticket and Work Order System* (TTWOS) is implemented on top of Remedy's ARS™.

The two systems were connected through an implementation of the OSS/J TT API with respective adaptors. The PoC covered the various interaction scenarios between the existing OSS systems, such as handling of loss of connectivity while sending TT, adding vendor specific attributes to TT, sending TT across OSS systems and others.

The PoC has successfully demonstrated the use of open standard interfaces between different OSS applications from different vendors as well as the capability of OSS/J architecture to provide 'point-to-multi-point' integration solutions. Meanwhile, the PoC was put into operations and has run successfully since spring 2004.

Generally, the technical capabilities and commercial advantages of OSS/J

systems integration shown by the PoC were so convincing that Vodafone D2 has decided to follow this approach for building their integrated service management architecture and to deploy it also internationally in other Vodafone companies.

Outlook

As a result of these successful proofs, OSS/J compliance is becoming a key vendor-evaluation factor for leading CSPs as they develop next-generation B/OSS systems procurements. Therefore, the OSS/J Initiative has recently formed a service provider advisory council that will share business and operational experiences, shape the strategic direction of the Initiative and help align standards approaches in ways that accelerate industry adoption of OSS/J technology. The service provider advisory council is an open body that allows service providers to participate in OSS/J governance and strategies, as well as the definition and deployment of component-based end-to-end solutions.

References

1 TeleManagement Forum. The NGOSS™ Technology-Neutral Architecture Specification. TMF053, Version 4.0 (Member Evaluation), Jan. 2004.

2 TeleManagement Forum. Shared Information/Data (SID) Model. GB922, Version 3.1 (Member Evaluation), July 2003.

3 TeleManagement Forum. Enhanced Telecom Operations Map™ (eTOM). GB921, Version 4.0, Mar. 2004.

4 OSS through Java™ Initiative. Core Business Entities Model Interfaces. Version 2.1 final, May 2004.

5 Fleck, J. J. Overview of the Structure of the NGOSS Architecture. Hewlett-Packard Company, May 2003.

6 OSS through Java™ Initiative. OSS through Java as an Implementation of NGOSS—A White Paper. Version 1.0, Apr. 2004.

7 OSS through Java™ Initiative. OSS through Java J2EE Design Guidelines. Version 1.1, Oct. 2001.

8 Java Specification Request 000144. OSS Common API. Final Release, Apr. 2002.

9 Java Specification Request 000089. OSS Service Activation API. Final Release, Apr. 2002.

10 Java Specification Request 000090. OSS Quality of Service API. Final Release, Nov. 2002.

11 Java Specification Request 000091. OSS Trouble Ticket API. Final Release, Feb. 2002.

12 Java Specification Request 000130. OSS Billing Mediation API. Final Release 2, Feb. 2004.

13 Java Specification Request 000142. OSS Inventory API. Proposed Final Draft, Apr. 2004.

14 Java Specification Request 000210. OSS Service Quality Management API. Expert Group Formation, Apr. 2003.

15 Benni, Enrico; Hjartar, Klemens; and Laartz, Jürgen. The IT factor in mobile services. *The McKinsey Quarterly*, 2003, Number 3.

Biography

Roland Volk
IP VALUE GmbH

Roland Volk is co-founder and Managing Director of IP VALUE Technologies. He had a successful career in the German telecommunications industry. As a member of the board of the TIMe Group, he supervised the build and start-up phase of several competitive local exchange carriers (CLECs) and a national backbone carrier. From 1995–1997, he led a profit centre for network/systems management solutions in a German systems integration company. Previously, he had several leadership roles for over five years at the American-Israeli network equipment provider Fibronics. Roland Volk holds a Computer Engineering Degree from the Darmstadt University for Applied Science, Germany.

Abbreviations/Acronyms

API Application programming interface
ARS™ Action Request System™
B2B Business-to-business
BSS Business support system
CBE Core business entities
CRM Customer relationship management
CRUD&Q Create, read, update, delete & query
CSP Communication service provider
DG OSS/J Design Guidelines
EAI Enterprise application integration
ebXML electronic business XML
EJB Enterprise Java Bean
FM Fault monitoring
eTOM Enhanced Telecom Operations Map™
GPRS General packet radio services
GSM Global system for mobile communications
GUI Graphical user interface
IIOP Internet inter-orb protocol
IP Internet protocol
IPDR Internet protocol detail record
ISM Integrated service management
IT Information technology
ITU International Telecommunication Union
J2EE™ Java™ 2 Platform, Enterprise Edition
JCP Java Community Process℠
JMS Java messaging service
JSR Java specification request
KQI Key quality indicators
MOM Message-oriented middleware
MMS Multimedia messaging service
NE Network element
NGOSS™ New Generation Operation System and Software
NM/SM Network management/service management
NMF Network Management Forum
OSS Operation support system
OSS/J OSS through Java™ Initiative
PM Performance monitoring
PoC Proof of concept
QoS Quality of service
RI Reference implementation
RMI Java remote method invocation
SeMIA Service Management Integration Architecture
SID Shared Information/Data Model
SLA Service level agreement
SMoS Service monitoring system
SMS Short messaging service
SOA Service oriented architecture
SQM Service quality management
TCA Traffic conditioning agreement
TCK Technology compatibility kit
TMF TeleManagement Forum
TT Trouble ticket
TTWOS Trouble Ticket and Work Order System
UM Usage monitoring
UML Unified modelling language
UMTS Universal mobile telecommunications system
VoIP Voice over Internet protocol
WAP Wireless application protocol
XML Extensible markup language