The editor would like to acknowledge the invaluable contributions made by the following individuals:

- Bill Wilbert, OSS through Java™
- Fergus O’Reilly, Highdeal S.A.
- Trevor Pirt, Motorola
- Rick Porter, Telcordia Technologies
- Kari Ronka, Nokia Networks
- Mikael Rutanen, Nokia Networks
- Michael Truss, Motorola
- Stefan Vaillant, Nokia Networks
1 Background

For the past several years, there has been a great deal of discussion on the relationship between 3GPP (and 3GPP2) and the OSS through Java™ Initiative (OSS/J). This document is intended to show how the two organizations are actually complementary. The document will begin by discussing the organizational differences between the two organizations, while subsequent sections will cover the relationship between the 3GPP and OSS through Java™ specifications.

One significant difference is the focus of the organizations. The 3G Partnership Program(s) focus solely on standards for the 3G Wireless Community, whereas OSS through Java™, while concerned with the needs of the 3G Wireless Community, is not focused on 3G. Instead, OSS/J is developing APIs that can be used in any information communications network, regardless of access technology.

Before discussing the technical aspects of the OSS/J $\leftrightarrow$ 3GPP/2\(^1\) relationship, it is important to understand the organizational aspects of the relationship.

The OSS through Java™ Initiative is a group of companies and individuals working within the auspices of the Java Community Process (JCP). OSS/J considers itself to be an implementation body. The concept of an implementation body differs from many of the past and existing efforts at Operation Support System (OSS) Application Programming Interface (API) specification or standardization in that the primary focus of OSS/J is to not only to create the specifics but to also provide near product ready implementations of those specifications. As such, while OSS/J does focus on the modeling and design of an API, OSS/J strives to re-use as much of the work of other standards and specifications as possible. OSS/J Reference Implementations are freely downloadable from the OSS/J web site (http://www.ossj.org/downloads/api.shtml).

This approach enables the members of the OSS/J Initiative to spend time developing robust, near production ready reference implementations of the APIs, rather than developing of those aspects of an API that can be drawn from the work of other bodies, such as 3GPP. OSS/ Reference Implementations are freely downloadable from the OSS/J web site (http://www.ossj.org/downloads). What's more, they are freely reusable; anyone or any organization may make use of the reference implementations in any way desired. This forward thinking approach enables the rapid development of an OSS/J compliant application. While the reference implementations are created by member companies, the hard work and dedication of the individual, academic, and corporate members of the Java Community are key to helping refine the API specification.

The 3G Partnership Program (3GPP) and its cousin, 3GPP2, are more traditional standard definition organizations (SDO). 3GPP/2 work to develop the specifications related to a particular functional area, such as performance management, defining a series of three documents. The first of the series covers requirements; the second covers an abstract, technology neutral analysis of the problem providing information and behavioral models; while the third document provides a distribution technology specific implementation of the requirements and information service documents (first and second documents, respectively). In practice, there are multiple versions of the

\(^{1} \text{3GPP/2 is commonly accepted short hand indicating both of the 3G Partnership programs, i.e. 3GPP and 3GPP2.}\)
third document (called a solution set) covering various distribution technologies, e.g. CORBA, CMIP and XML².

In summary, OSS/J is an implementation body that reuses the work of more traditional standards and specification definition organizations, such as 3GPP/2, ITU-T and the TeleManagement Forum. 3GPP/2, on the other hand, are traditional standards definition organizations from which OSS/J draws material to bootstrap the specification and implementation of an OSS API.

² At the time this document was written the effort to change the 3GPP XML file format definition documents to official solution sets was underway.
2 The OSS/J $\leftrightarrow$ 3GPP/2 Relationship Explained

The OSS through Java™ Initiative is an implementation body — that is, OSS/J does not develop standards per se, rather, OSS/J develops the functional, real world, out-of-the-box reusable API specifications and implementations of those specifications.

In general the relationship between OSS/J and other more traditional standards definition organizations and specification fora can be seen (depicted graphically) in the following figure:

![Diagram showing the relationship between OSS/J and 3GPP](image)

Figure 1

As shown in Figure 1 (above) the OSS through Java™ Initiative operates on a pull-push model. OSS/J starts with work from more traditional standards definition organizations and specification fora, creates an API specification and implementation according to the Java™ Community Process, and then makes that specification available to everyone.

The question “What is the relationship between the OSS through Java™ Initiative and 3GPP?” is made clear by Figure 1. However, this answer is at best incomplete. Furthermore, given that 3GPP defines solution sets for each of its integration reference points (IRPs) a question needs to be asked to complete the answer: “What is the relationship between the OSS through Java™ APIs and the 3GPP Solution Sets?”
3 OSS through Java™ APIs and 3GPP Solution Sets

Both OSS/J and 3GPP are doing work in multiple functional areas. The following tables shows, where APIs are targeting the same functionality:

<table>
<thead>
<tr>
<th>OSS/J API</th>
<th>3GPP/2 IRP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trouble Ticketing (TT)</td>
<td>--</td>
</tr>
<tr>
<td>Service Activation (SA)</td>
<td>--</td>
</tr>
<tr>
<td>Quality of Service (QoS)</td>
<td>Notification IRP, Alarm IRP</td>
</tr>
<tr>
<td></td>
<td>Performance Management IRP</td>
</tr>
<tr>
<td>Inventory</td>
<td>Inventory Management IRP</td>
</tr>
<tr>
<td>Service Quality Management (SQM)</td>
<td>--</td>
</tr>
<tr>
<td>Billing Mediation</td>
<td>Charging and Billing³</td>
</tr>
<tr>
<td>Discovery</td>
<td>--</td>
</tr>
<tr>
<td>Pricing</td>
<td>--</td>
</tr>
</tbody>
</table>

As can be seen, the OSS/J APIs TT, SA, SQM, Discovery, and Pricing do not have functionally equivalent counterparts in 3GPP. For those APIs, the questions of OSS/J and 3GPP is a non-issue: OSS/ simply addresses areas that are not covered by 3GPP.

Currently, there are five areas where the OSS through Java™ APIs and the 3GPP Solution Sets do overlap. They are:

1. Fault Management
2. Inventory Management
3. Performance Management
4. Charging and Billing
5. Service Activation

In the case of Fault Management, the OSS through Java™ QoS API contains an alarm management component that is a Java™ implementation of the 3GPP Release 4 Alarm Management IRP (TS 32-111.2).

In the case of inventory management, 3GPP does not currently have a solution set available. The development of a 3GPP Inventory Management IRP Solution Set⁴ may or may not be completed as part of the 3GPP Release 6². Work on the Inventory Management IRP will continue in 3GPP Release 7. Additionally the OSS through Java™ Inventory Management API has not yet been finalized. Alignment will be accomplished via a mapping of the 3GPP Inventory Information Model to the OSS/J Core Business Entities. This mapping will be accomplished using the TeleManagement Forum Shared Information/Data Model (SID).

The third area of overlap is Performance Management. From the OSS through Java™ Initiative, JSR-90, the OSS through Java™ Quality of Service API provides a service-oriented interface for measurements and threshold monitors (along with the aforementioned fault management component). From 3GPP there is the TS32.64x

---

³ The 3GPP Charging and Billing work is in development, and there are a number of approved documents. However, no interface specifications have been approved at the time of the writing of this document.

⁴ As part of 3GPP Release 5, a requirements and information service specification for the Inventory Management IRP were released. The 3GPP Specifications are TS32.691 and TS32.692 respectively.

⁵ 3GPP Release 6 is scheduled for completion by the end of 2004.
series of documents that are part of the 3GPP Release 6. At this time, both the 3GPP and OSS through Java™ PM interfaces support the same basic set of concepts:

- Measurement Results reported using the 3GPP XML Schema for performance measurement reporting (TS32.401)
- Creation and subsequent management of measurement jobs
- Creation and subsequent management of thresholding jobs
- Notification of threshold crossing via alarm management mechanism

Appendix A of this document provides a detailed coverage of the similarities and differences between the OSS through Java™ and 3GPP interfaces for Performance Management.

The fourth area of overlap is in the Charging and Billing domain. 3GPP separates online and offline charging.

**Offline Charging:** The OSS/J Billing Mediation API addresses the link between network or mediation layer elements and a billing system. It allows for the network/mediation layer to make events available for consumption by the billing system and thus accomplish a standards-based way to transfer Charging Data Records (CDRs) between these layers. Within the 3GPP Release 6 context, the corresponding logical function is the interface between the Charging Gateway Function and the Billing Domain, this 3GPP interface is called the Bx reference point and is further described in the draft technical specification document 3GPP TS 32.297. Although the 3GPP specs are still emerging, there is already good alignment at a high level between the OSS/J Billing Mediation API design and the Bx interface: both include the idea of filtering or selective subscription by the billing domain to events from the network/mediation domain: both support both push and pull methods for the exchange of UDRs. 3GPP will probably be specific and rigorous regarding the actual content and format of the UDR record files transferred across the Bx interfaces, and OSS/J is agnostic on the actual UDRs transferred and thus should be able to serve as an implementation of a transport for the 3GPP Bx interface.

**Online Charging:** Online charging standards are currently emerging within the 3GPP and targeted for 3GPP Release 6 — the current draft technical specifications are further described in 3GPP TS 32.296. Key interfaces in this domain are real-time connections between the event and session-based charging function, the external rating function, the prepaid account balance holder and, through the charging gateway function, to a billing system. OSS/J currently has no API in the online charging domain so there is no overlap here.

Finally, the fifth area of overlap is in Service Activation. The OSS through Java™ Service Activation API provides a highly extensible interface that can be effectively used for that activation of service at any level of the OSS stack. Within 3GPP, the work on subscription management is defined by TS32.140. Within 3GPP, subscription management (or SuM as is called within the 3GPP organizations) "...is primarily concerned with the ability to define subscription profiles and associate the profile with subscribers, users and services that are authorized by agreements. The subscription profile may be used in the process of configuring various network resources (access and core) to make the service a reality for the user. Thus there is an inherent relationship between subscription management and service activation."
4 Summary

For the following reasons, OSS/J and 3GPP/2 are not conflicting [THIS IS A NEGATIVE WAY TO SAY: For the following reasons, OSS/J and 3GPP/2 are complementary]:

- For five of the eight OSS/J APIs there are currently no equivalent 3GPP functional interfaces.
- OSS/J is a implementation body, whereas 3GPP is a traditional standards definition organization.
- OSS/J is access network technology agnostic, whereas 3GPP is 3G specific.

However, three of the eight OSS/J APIs have equivalent 3GPP interfaces. This was known when the OSS/J work began. Still, these three interfaces, like all OSS/J APIs, were intentionally designed by reusing existing specifications, and in the case of QoS (Fault Management and Performance Management), the 3GPP specifications were directly reused.

Therefore, for the Fault Management and Performance Management areas, the overall solution to the industry from both OSS/J and 3GPP can be thought of in the following manner:

- The requirements and information model by 3GPP
- CORBA and CMIP SS from 3GPP.
- Java/J2EE and XML SS from OSS/J.

**Conclusion:** OSS/J and 3GPP are complementary: OSS/J offers APIs that are not covered by 3GPP. For functionally similar APIs, OSS/J uses the same information model and simply complements the 3GPP offering with two additional “solution sets”, J2EE Java Value Type and XML/JMS.
Appendix A: SS/J QoS API and 3GPP PM IRP

OSS/J QoS API
The OSS/J QoS API consists of a number of interfaces. The interfaces have been designated as being either mandatory or optional.

The Mandatory Interfaces are:

1. JVTPerformanceMonitorHome
2. JVTPerformanceMonitorSession
3. JVTPerformanceMonitorSessionOptionalOpt
4. PerformanceAttributeDescriptor
5. PerformanceDataAvailableEvent
6. PerformanceDataAvailableDescriptor
7. PerformanceMonitorByObjectsValue
8. PerformanceMonitorKey
9. PerformanceMonitorState
10. PerformanceMonitorValue
11. PerformanceMonitorValueIterator
12. QueryByDNValue
13. QueryPerformanceMonitorValue
14. ReportFormat
15. ReportInfo
16. ReportInfoIterator
17. ReportMode

Striking interfaces that are “artifacts” of the J2EE environment removes JVTPerformanceMonitorHome from the list. Removing interfaces that are “artifacts” of the OSS/J Design Guidelines and have no impact on the overall functionality of the QoS API reduces the list to the following.

1. JVTPerformanceMonitorSession
2. JVTPerformanceMonitorSessionOptionalOpt
3. PerformanceDataAvailableEvent
4. PerformanceMonitorByObjectsValue
5. PerformanceMonitorKey
6. PerformanceMonitorState
7. PerformanceMonitorValue
8. QueryByDNValue
9. QueryPerformanceMonitorValue
10. ReportInfo

Striking interfaces that represent “results” (that is, information model elements), the list is further reduced to

1. JVTPerformanceMonitorSession
2. JVTPerformanceMonitorSessionOptionalOpt
3. PerformanceDataAvailableEvent
4. PerformanceMonitorByObjectsValue
5. PerformanceMonitorKey
6. PerformanceMonitorState
7. PerformanceMonitorValue
8. QueryByDNValue
9. QueryPerformanceMonitorValue
10. ReportInfo
1. JVTPerformanceMonitorSession
2. JVTPerformanceMonitorSessionOptionalOpt

For purposes of this comparison, the important result is to verify that all mandatory operations specified by 3GPP are also mandatory within OSS/J.

Towards this end the JVTPerformanceMonitorSessionOptionalOpt interface can be discarded from consideration at this point.

JVTPerformanceMonitorSession is the “primary” functional interface for the QoS API measurement functionality. It consists of a number of mandatory and optional operations. The mandatory operations are

1. getEventDescriptor()
2. getEventTypes()
3. getManagedEntityTypes()
4.getQueryTypes()
5. getSupportedOptionalOperations()
6. makeManagedEntityValue()
7. makeQueryValue()
8. queryManagedEntities()
9. getVersion()
10. makePerformanceMonitorValue()
11. getReportFormats()
12. getCurrentReportFormat()
13. getReportModes()
14. getObservableObjectClasses()
15. getObservableObjects()
16. getSupportedObservableObjects()
17. getObservableAttributes()
18. getSupportedGranularities()
19. getPerformanceReportInfo()
20. getPerformanceMonitorByKey()
21. getPerformanceMonitorsByKeys()
22. queryPerformanceMonitors()
23. createPerformanceMonitorByValue()
24. removePerformanceMonitorByKey()

Removing operations from the list of mandatory operations that are “artifacts” of the OSS/J design guidelines produces the following list of operations

1. makeManagedEntityValue()
2. makeQueryValue()
3. queryManagedEntities()
4. makePerformanceMonitorValue()
5. getReportModes()
6. getObservableObjectClasses()
7. getObservableObjects()
8. getSupportedObservableObjects()
9. getObservableAttributes()
10. getSupportedGranularities()
11. getPerformanceReportInfo()
12. getPerformanceMonitorByKey()
13. getPerformanceMonitorsByKey()
14. queryPerformanceMonitors()
15. createPerformanceMonitorByKeyValue()
16. removePerformanceMonitorByKey()

The optional operations of JVTPerformanceMonitorSession are

1. tryCreatePerformanceMonitorsByValues()
2. tryRemovePerformanceMonitorsByKeys()
3. suspendPerformanceMonitorByKey()
4. trySuspendPerformanceMonitorsByKeys()
5. resumePerformanceMonitorByKey()
6. tryResumePerformanceMonitorsByKeys()
7. getCurrentResultReport()

The OSS/J operations that begin with the prefix “try” are “artifacts” of the OSS/J design guidelines and represent “best effort interfaces,” removing these from the list of optional operations produces the following list

1. suspendPerformanceMonitorByKey()
2. resumePerformanceMonitorByKey()
3. getCurrentResultReport()

For threshold monitoring, repeating the above “exercise” yields the following results.
OSS/J QoS API “functional” interfaces for thresholding:

1. JVTThresholdMonitorSession
2. JVTThresholdMonitorSessionOptionalOpt

The following operations defined within the JVTThresholdMonitorSession are of interest

1. makeManagedEntityValue()
2. makeQueryValue()
3. queryManagedEntities()
4. getVersion() Mandatory
5. makeThresholdMonitorValue()
6. getObservableAttributes()
7. getObservableObjectClasses()
8. getObservableObjects()
9. getSupportedGranularities()}
10. getSupportedObservableObjects()
11. getThresholdMonitorByKey()
12. getThresholdMonitorsByKey()
13. createThresholdMonitorByValue()
14. queryThresholdMonitors()
15. removeThresholdMonitorByKey()

The following optional operations within JVTThresholdMonitorSession are of interest

1. resumeThresholdMonitorByKey()
2. suspendThresholdMonitorByKey()
The PM IRP

The most recent documents for the functional aspects of the PMIRP define four interfaces; of which two are mandatory. Within the optional interfaces, if the interface is supported, there are operations that must be supported and operations that are optional. The interfaces (with specification of the support qualifier) are as follows:

1. PMFileOperations
2. PMIRPOperations
3. «optional» PMIRPThresholdOperations
4. «optional» PMIRPThresholdModificationOperations

The operations specified by the PMFileOperations interface are:

1. listPMFiles()

The operations specified by the PMIRPOperations interface are

1. createMeasurementJob()
2. deleteMeasurementJob()
3. «optional» suspendMeasurementJob()
4. «optional» resumeMeasurementJob()
5. «optional» listMeasurementJobs()

The operations specified by the PMIRPThresholdOperations interface are

1. createThresholdMonitor()
2. deleteThresholdMonitor()
3. «optional» listThresholdMonitors()

The operations specified by the PMIRPThresholdModificationOperations interface are

1. modifyThresholdMonitor()
2. «optional» suspendThresholdMonitor()
3. «optional» resumeThresholdMonitor()
Comparison of Support

<table>
<thead>
<tr>
<th>PM IRP Operation</th>
<th>M</th>
<th>O</th>
<th>OSS/J QoS API Operation</th>
<th>M</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>listPMFiles</td>
<td>M</td>
<td>O</td>
<td>getPerformanceReportInfo</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>createMeasurementJob</td>
<td>M</td>
<td>O</td>
<td>createPerformanceMonitorByValue</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>deleteMeasurementJob</td>
<td>M</td>
<td>O</td>
<td>removePerformanceMonitorByKey</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>suspendMeasurementJob</td>
<td>O</td>
<td></td>
<td>suspendPerformanceMonitorByKey</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>resumeMeasurementJob</td>
<td>O</td>
<td></td>
<td>resumePerformanceMonitorByKey</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>listMeasurementJobs</td>
<td>M</td>
<td>O</td>
<td>queryPerformanceMonitors</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>createThresholdMonitor</td>
<td>M</td>
<td>O</td>
<td>createThresholdMonitorByValue</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>deleteThresholdMonitor</td>
<td>M</td>
<td>O</td>
<td>removeThresholdMonitorByKey</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>listThresholdMonitors</td>
<td>M</td>
<td>O</td>
<td>queryThresholdMonitors</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>modifyThresholdMonitor</td>
<td>M</td>
<td>O</td>
<td>Not directly supported</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>suspendThresholdMonitor</td>
<td>O</td>
<td></td>
<td>suspendThresholdMonitorByKey</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>resumeThresholdMonitor</td>
<td>O</td>
<td></td>
<td>resumeThresholdMonitorByKey</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>

As can be seen by from the above table, the only significant difference between the OSS through Java™ QoS API and the 3GPP PM IRP is the support for the ability to modify an existing threshold monitor job. To accomplish this functionality with the QoS API it is necessary to use the current threshold job to create a new threshold job (via a templated create mechanism) after having made the desired changes to the value object describing the threshold monitor, and then delete the old threshold monitor.
Appendix B: References

NOTE: FOR REFERENCES THAT DO NOT SPECIFY A VERSION OR DATE FOR THE DOCUMENT/TEXT, ASSUME
THE LATEST AVAILABLE VERSION