A Logger System based on Web services

by B. Horn
H. Balakrishnan
B. T. Maniampadavathu
J. Warnes
D. A. Elko

The Logger System described in this article is a general purpose set of component interfaces for logging data in a distributed, heterogeneous computing environment. Fundamentally, it serves as an intermediary between log artifact producers and log artifact consumers. It is designed to virtualize existing logging systems including the z/OS® System Logger, Microsoft Windows® event logging, and the UNIX® syslog facility. The Logger System is outlined using the conventions defined in the recently released Web Services Resource Framework (WSRF) and provides an example of how this framework might be used to define a meta-OS (operating system) for grid computing.

The Logger System described in this paper serves as a manageable repository of log records. The term *log record* refers to the atomic unit read or written through the Logger System's interfaces. Components that write log records to the Logger System are referred to as *log artifact producers* or simply *producers*. Components that read from the Logger System are referred to as *log artifact consumers* or simply *consumers.* The Logger System serves as an intermediary, decoupling log artifact producers and log artifact consumers. Log artifacts written by log producers may or may not be read at a later time by log consumers. A component that manages a Logger System is referred to as a *log manager.* Log managers maintain the Logger System, performing a variety of administrative functions such as creating new logs, purging obsolete records, and managing retention policies.

The paper has the following structure. First, we summarize the logging requirements generated by grid-based usage scenarios. We then use conventions defined in WSRF (Web Services Resource Framework)[1,2] to describe the proposed Logger System and interfaces. Finally we present our conclusions and identify future work.

Requirements

The GGF (Global Grid Forum) OGSA-WG (Open Grid Services Architecture Working Group) has documented several usage scenarios that require logging services.[3,4] These scenarios include those based upon problem determination, metering resource consumption, failure recovery, transaction processing, and security. In these usage scenarios, log artifact consumers and producers place a number of requirements on the Logger System. In the following subsections we describe these requirements and provide a brief discussion describing how our proposed Logger System addresses each requirement. The requirements documented in these subsections represent the results of the analysis performed by the OGSA-WG focus session on logging.[5] While the requirements in this section are based on grid usage scenarios, they are also applicable to distributed, heterogeneous computing in general.

©Copyright 2004 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

Legacy logger systems. To ensure the general acceptance of the basic log semantics and to enable the exploitation of existing implementations, the Logger System should support key features found in existing logging implementations.[6–9] We have verified that the interfaces presented later in this paper represent a semantic superset of these key logging implementations.

Persistency. One of the basic goals of any logging system is to provide a mechanism to create persistent log records. The retention period for a log record should be determined by consumer requirements. For example, in a real-time monitoring application, data may become irrelevant in a very short time. In contrast, data for an auditing program may be needed for months or even years after it was generated. Some applications are best served by circular logs. Other applications depend on persistent log records to recover their state in case of failure. As discussed later in the section on specialized logs, the Logger System should support a variety of persistency qualities of service (QoS), including, for example, circular logs or logs with policy-specified record retention periods. We address the specification of persistency QoS later in the log:Log interface section.

Support for sequential writes. The Logger System should support sequential writes at the granularity of an individual record, with each record representing one log artifact. Writes always append a record to the end of a log stream. Byte-level access is not required. The semantics for log writes presented in the log:LogConnection interface section meet this requirement.

Stateful read cursor. The Logger System should permit consumers to sequentially access the records in a log stream using a stateful (having the capability to maintain state) cursor that is not invalidated by producer writes or reads from other consumers. The system should provide an efficient mechanism to position the cursor by using the record time stamp or a unique record identifier. The semantics for log reads presented in the log:LogConnection interface section also meet this requirement.

Ordering of log records. Logs are frequently used to record a time-ordered sequence of events. At a minimum, the Logger System should, for a given write connection, permit consumers to efficiently read (time-stamped) records in the same order that they were originally written. Other implementations, for example, globally ordered logs, may choose to provide different ordering requirements. This requirement represents a QoS underlying the log write semantics presented later in the section describing the log:LogConnection interface.

Standard schema for log records. A standard, structured log artifact schema facilitates effective intercommunication among disparate applications. In some cases, for example, when performance is paramount and interoperability is not a concern, less structured artifacts are more appropriate. The recently proposed Common Base Event[10,11] is designed to accommodate varying degrees of structure and is an ideal candidate for representing a structured log artifact. The details of this requirement are not addressed in this paper.

Decoupling of producers from consumers. The ultimate usage of a log artifact (for example, audit, system management, problem determination) should be determined, potentially at runtime, by the log artifact consumer, not by the log artifact producer. The Logger System should enable the decoupling of the log artifact producer from the log artifact consumer. This requirement is closely related to the following requirement.

Broker. The Logger System should support mechanisms that permit: (1) multiple producers to write records into a log stream, and (2) multiple consumers to read records from a log stream. Furthermore, the system should be able to coordinate concurrent reads and writes. This is a fundamental behavior supported by all production OS (operating system) logger systems. This requirement and the requirement that producers be decoupled from consumers are both met by interfaces presented in this paper.

Filtering. Frequently the quantity of logging data generated is much greater than the quantity of data actually consumed. To reduce resource consumption, the Logger System should provide filter mechanisms for both read and write operations. A filter on the write operation avoids the cost of storing filtered records. The LogConnection interface and the LogBrowseSession interface satisfy this requirement.

Merged log. Some situations (for example, a high-performance, merged log for a clustered system) require the logger to exploit underlying, hardware-assisted merge mechanisms. These merge mechanisms (for example, the zSeries* Coupling Facility) permit multiple producers operating in separate OS im-

ages to concurrently and efficiently write to a common (merged) log stream. Currently, hardware-assisted merged log capabilities are used in mainframe clusters composed of homogenous systems. In theory, however, such a capability could also be built for clusters composed of heterogeneous systems. The Logger System should support the exploitation of hardware-assisted merge mechanisms. In our proposed set of interfaces, a hardware-assisted merged log would be a QoS underlying the log:Log interface described later in this paper.

Synchronous and asynchronous write semantics. We have identified two required types of write behavior: ack and noack. For the ack type of behavior, consumers require either that every record written by a producer must be stored in the log or that the producer must be explicitly informed and corrective action taken. Such requirements are characteristic of applications involving, for example, transactional logs or metering for billing. Logger Systems supporting this requirement must acknowledge every write. Some producers prefer to wait for an acknowledgment from the Logger System (i.e., block) before executing any additional logic, whereas others may prefer to continue execution, obtain the acknowledgement using a callback mechanism, and then deal with failure using separate recovery/notification logic. The Logger System should support both of these acknowledgement mechanisms. For the noack type of behavior, characteristic of "fire and forget" behavior for noncritical, information-only monitoring, consumers do not require any acknowledgement. The Logger System should support all of the preceding write behavior patterns. We address write semantics in the section describing the log:LogConnection interface.

Deletion of log records. Log maintenance (for example, cleanup) requires a mechanism for performing time-stamp-based deletions of existing log records in a log stream. This requirement is met by the semantics described in the log:LogConnection interface.

Coexistence with the messaging fabric. Patterns for the consumption of events vary according to the needs of the consuming applications. In a distributed environment, reads on a log stream based on RPC (remote procedure call) may not be the best mechanism for consuming applications to access events. For example, a real-time monitoring application may need to be asynchronously notified whenever a high-severity artifact is logged (push mode). A different example would be a consumer that polls for metering data every 24 hours (pull mode). Experience has shown that these scenarios are better served by systems such as Web Services Notification (WS-Notification),[12] which are specifically designed to move messages in a distributed environment, rather than by using the Logger System. However, both of the above examples might require that a log be kept of all events that flow through a messaging broker. The Logger System should be designed such that a WS-Notification broker can act as a logging broker by implementing logger interfaces. The operations presented later in our discussion of the Logger System interfaces do not conflict with any of the WS-Notification operations. However, the behavior of a system implementing both log and messaging broker interfaces is not addressed in this paper.

Support for multiple query expression types. As described previously, basic log function requires only a stateful read cursor and support for efficient access using the record time stamp and record identifier. However, if a particular implementation supports other query semantics, then the Logger System should expose them through an extensible query mechanism. This requirement is addressed in the discussion of the log:LogBrowseSession interface.

Standard set of specialized logs. Different applications have different requirements for the features and QoS associated with a log. To accommodate these demands, the Logger System should support a finite set of specialized logs that somehow span the space of required features and QoS. The specification of QoS is associated with the log:Log interface.

Specializations should include the following types:

Secure logs—Some applications (for example, metering and authentication/authorization/accounting) require that logs be secure (for example, encrypted). OGSA Security mechanisms are currently being discussed in the OGSA-SEC Working Group.[13]
Globally ordered logs—Some parallel, transaction-oriented applications require merged logs that maintain a global order based on a common clock.
Circular logs—For some applications it is preferable to specify retention by using a space constraint instead of a time constraint.
Duplexed logs—A log may offer a high reliability QoS by duplexing all writes.
Compressed logs—Compressed logs trade performance for space.

Logger System interfaces

This section outlines our Logger System design using Web services interfaces. The design is expressed using the conventions and specifications established by WSRF. WSRF defines the Web Services Resource (WS-Resource) approach to modeling and managing state in a Web services context. WSRF consists of:

1. A set of six new Web services specifications
2. A set of conventions for existing technologies

We use interfaces defined in three of the new Web services specifications: Web Services Resource Properties (WS-ResourceProperties),[14] WS-Notification,[12] and Web Services Resource Lifetime (WS-ResourceLifetime),[15] to describe some of the semantics of the Logger System.

As part of the conventions for existing technologies, WSRF introduces the so-called *implied resource pattern* to describe how Web Services Addressing (WS-Addressing)[16] is used to associate a stateful resource with a Web services interface (using the Web Services Description Language (WSDL) 1.1[17] portType element). Let *wsa* represent the WS-Addressing namespace (a set of names that is defined according to some naming convention). The implied resource pattern requires that the wsa:EndpointReference element must include a wsa:ReferenceProperties child element to identify the resource associated with the address specified by the wsa:Address child element of this EndpointReference (EPR). Essentially, the EPR is a pointer containing, among other elements, an address (wsa:Address) of the service and an opaque expression of resource identity (wsa:ReferenceProperty).

Instead of providing explicit WSDL, we have chosen to outline the proposed design in terms of UML** (the Unified Modeling Language).[18] Specifically, we use UML to express:

1. The definition of WSDL interfaces
2. The inheritance of WSDL interfaces, as defined by WS-ResourceProperties
3. The XML (Extensible Markup Language) schema for WS-ResourceProperties resource property documents
4. The XML expression of a WS-Addressing EndpointReference (EPR)

The UML conventions that we use are illustrated in Figure 1. In this figure we show the myns:MyInterface interface as extending the urns:UrInterface interface. These interfaces are stereotyped as <<WSDL portType / Interface>>, indicating that the UML class represents the XML defining the WSDL portType. Also, both of the interfaces are associated with the WS-ResourceProperties resource property document. The relationship is stereotyped as <<wsrp:ResourceProperties>>. WS-ResourceProperties requires the cardinality on the resource property document side of the association to be [0,1]; that is, there is at most one resource property document per WSDL portType. In this figure we also show how UML is used to convey the XML schema for resource property documents. The schema for the resource property document associated with myns:MyInterface illustrates the existence of three types of child elements: myns:OneChildElementType, myns:AnotherChildElementType, and wsa:EPR (wsa:EndpointReferenceType). For associations between a parent elementType and a child elementType, if no cardinality is shown on the child side, then it is implied as [1] (minOccurs=1,maxOccurs=1). Element attributes are shown as UML class attributes. For example, myns:MyInterfaceRPType has one attribute, myns:anAttribute. The namespace of an attribute is not specified if it is the same as the parent element. In the interest of clarity, we omit content that is not directly relevant to our description. For example, the wsa:EPR type has additional child elements types (for example, wsa:ServiceName) that are not depicted. The namespaces used in the interface descriptions are shown in Table 1.

The proposed Logger System is composed of four interfaces. They are:

1. The log:LogManager interface
2. The log:Log interface
3. The log:LogConnection interface
4. The log:LogBrowseSession interface

The portTypes supporting these interfaces and their relationships are shown using UML in Figure 2 and are described in the following subsections. The figure follows the UML conventions established above. The new elements are shown on the bottom, and existing elements are shown on the top. This figure should be referred to when reading the interface descriptions.

The log:LogManager interface. The log:LogManager interface is responsible for managing an instance of the Logger System. The log:LogManager portType inherits from the wsrp:ResourceProperties,wsnt:Notifi-

Figure 1 UML conventions used for expressing the WSRF-based design

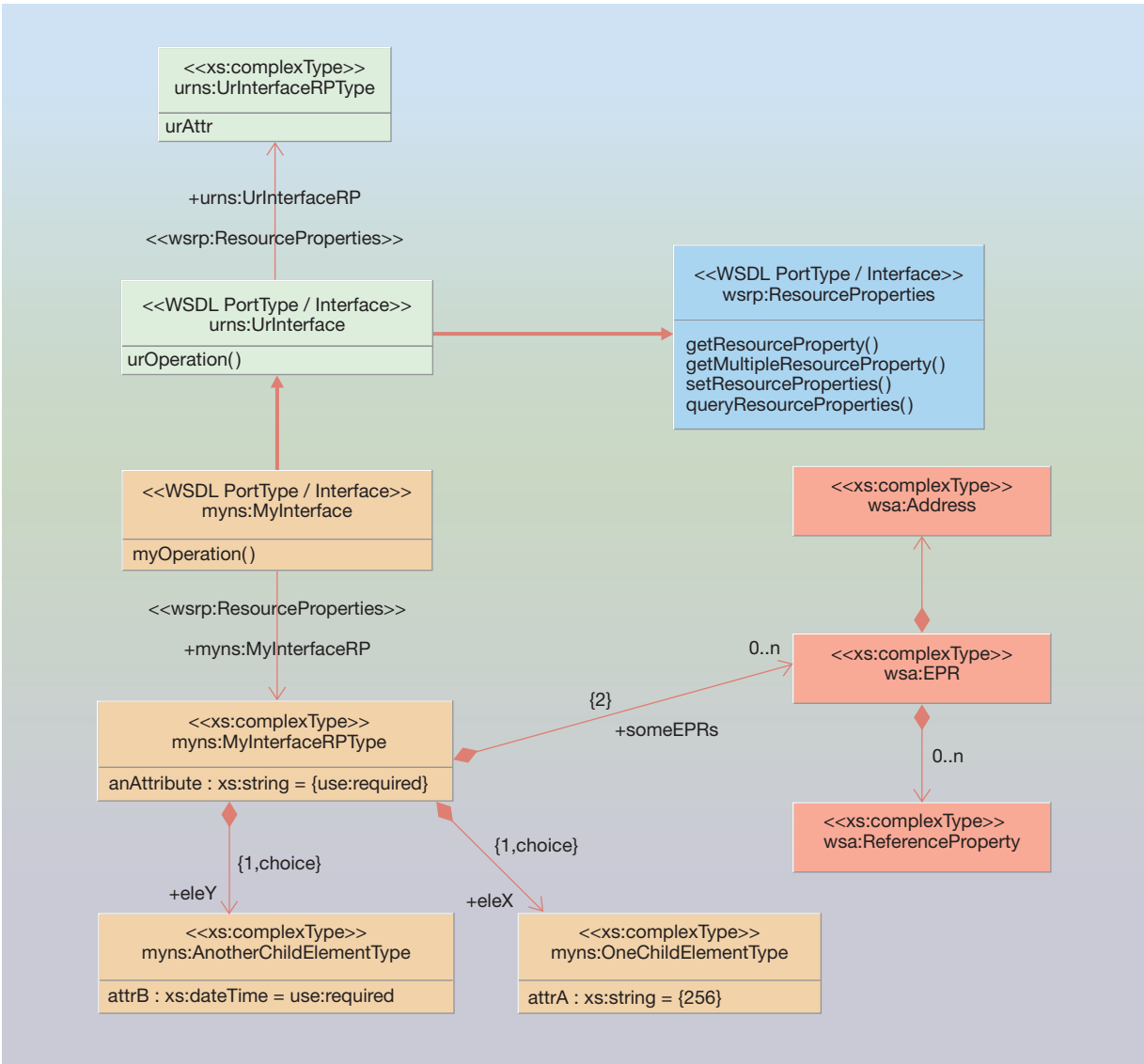


Table 1 Namespaces used in the interface descriptions

Prefix	Namespace
xsd	http://www.w3.org/2001/XMLSchema
log	http://TBD/log
wsa	http://schemas.xmlsoap.org/ws/2003/02/addressing
wsnt	http://www.ibm.com/xmlns/stdwip/web-services/WS-Notification
wsrl	http://www.ibm.com/xmlns/stdwip/web-services/WS-ResourceLifetime
wsrp	http://www.ibm.com/xmlns/stdwip/web-services/WS-ResourceProperties

Figure 2 Overview of Logger System interfaces

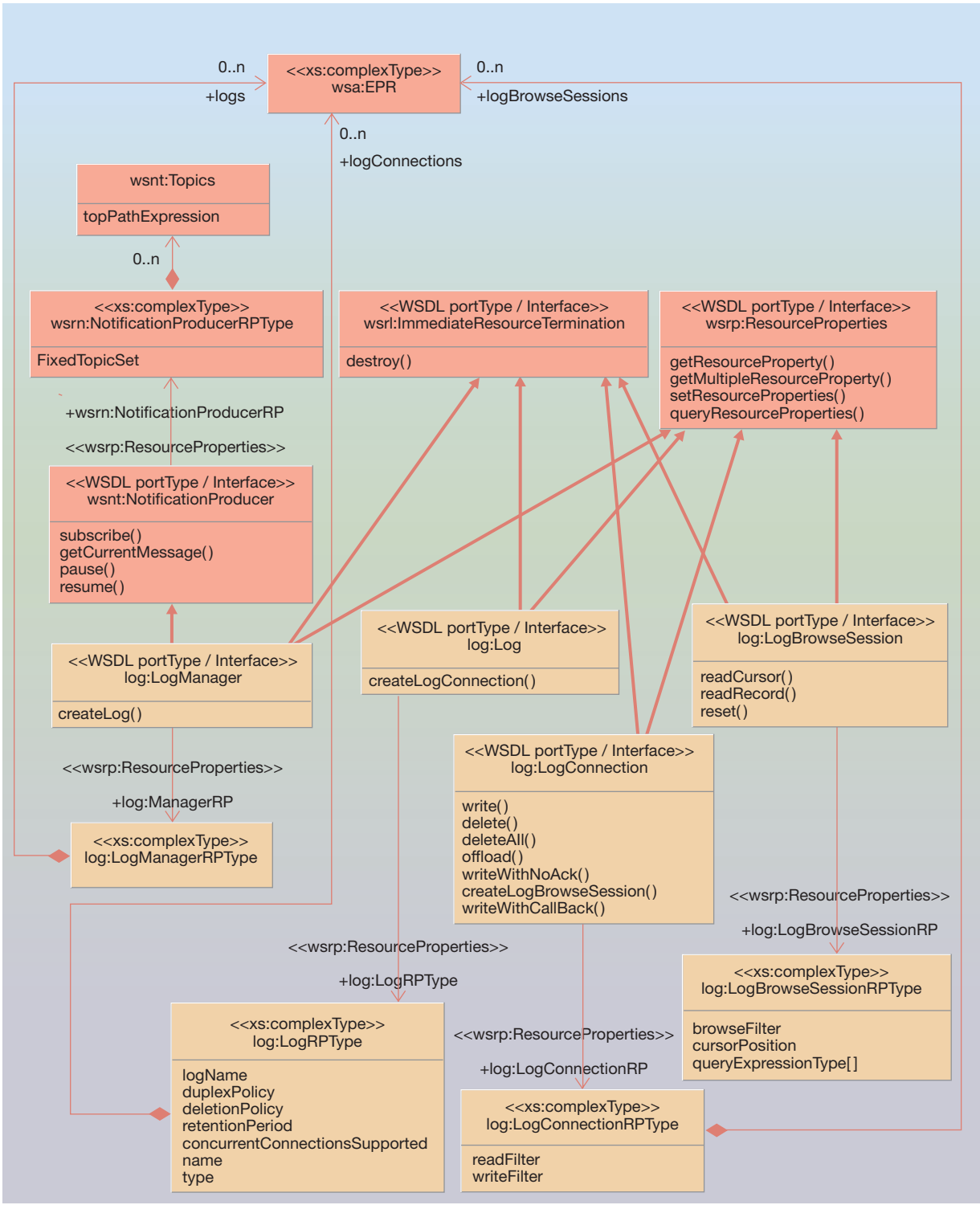
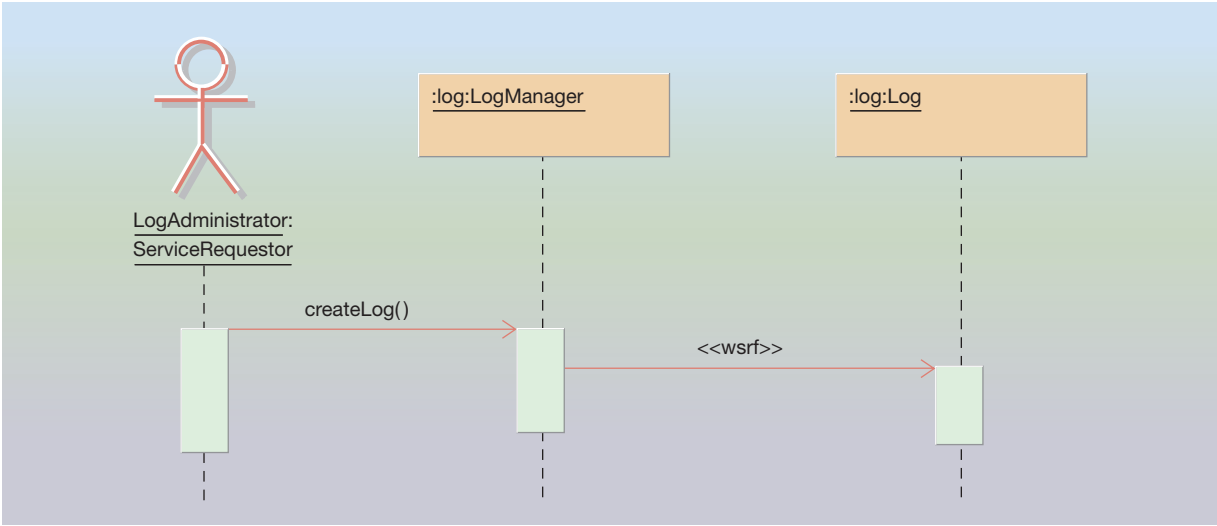


Figure 3 Setting up a log

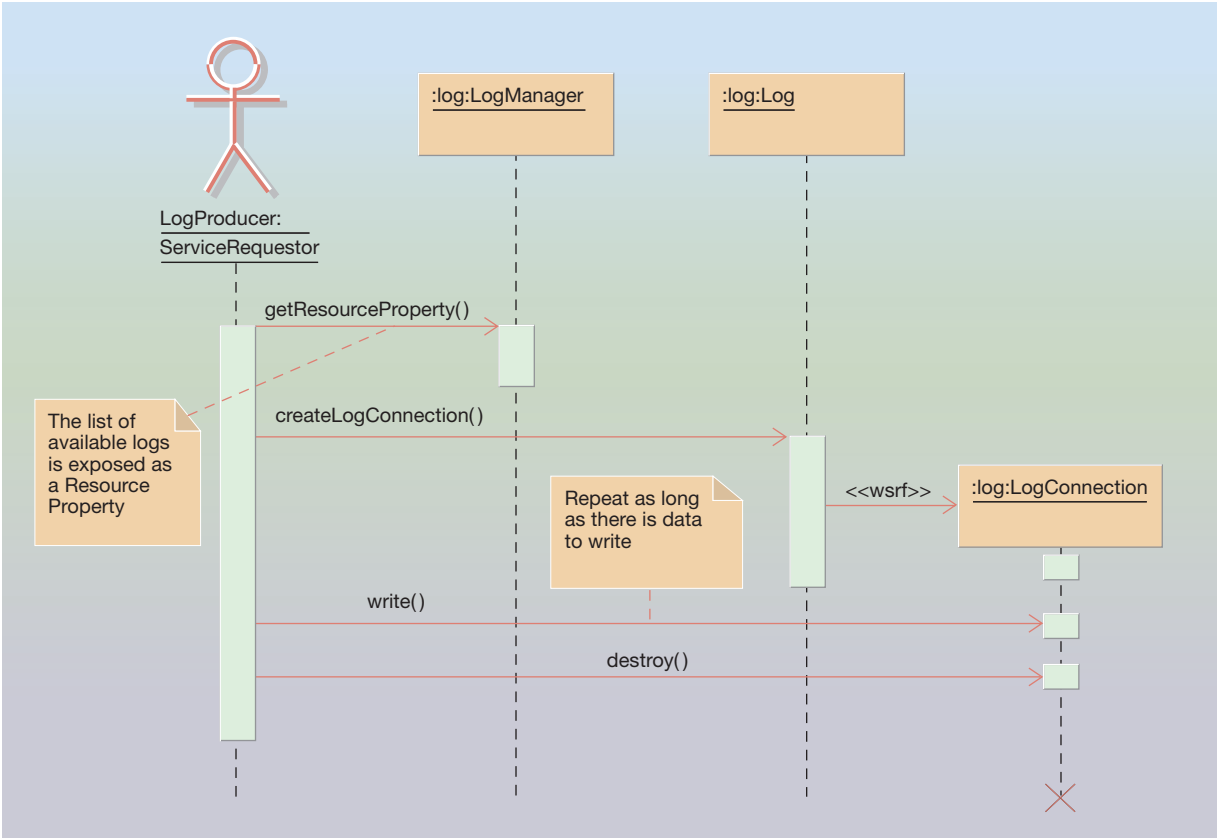


cationProducer, and wsrl:ImmediateResourceTermination portTypes. The createLog operation of the log:LogManager portType acts as a factory for log:Log resources. This operation may be able to instantiate logs with different QoS (i.e., those outlined previously in the description of specialized logs). For example, if the underlying system supports a hardware-assisted merge facility and the log:LogManager implementation exposes this capability, then a log with merge capability may be instantiated. The home collection of all log:Log instances created by an instance of the log:LogManager interface is stored as wsa:EPR child elements in its log:LogManager resource property document. If there are no active connections to any logs managed by a log:LogManager resource, then that resource may be explicitly destroyed with the wsrl:ImmediateResourceTermination portType destroy operation. An explicit destroy of a log:LogManager resource implicitly destroys all the underlying log:Log resources. A sequence diagram illustrating the creation of a log:Log is shown in Figure 3.

The Logger System is an important system and must be managed accordingly. It is necessary not only to monitor its performance but also to deal with storage space thresholds, low-space or insufficient-space conditions, periodic purging, access control, and many other management facets. Management events related to an instance of log:LogManager may be monitored using the inherited wsnt:NotificationProducer portType. All management-related events related to an instance of the Logger System are surfaced through this mechanism.

The log:Log interface. The log:Log portType inherits from the wsrp:ResourceProperties portType and the wsrl:ImmediateResourceTermination portType. A relatively long list of parameters defines the behavior of a particular log service instance. For example, the log resource state could be characterized by the following: duplex policy, deletion policy, retention period, and support for concurrent connections. Although not specified in this paper, it is envisioned that, when appropriate, the setResourceProperties operation of the wsrp:ResourceProperties portType would be used to change the policy of an existing log. Also, technologies related to the specification and deployment of policy could assist in the management of logs in a distributed environment composed of a large number of systems.[19] The createLogConnection operation of the log:Log portType acts as a factory for log:LogConnection resources. The home collection of all log:LogConnection instances created by an instance of log:Log is stored as wsa:EPR child elements in its resource property document. User-specified read and write filters may be set for log:LogConnection instances. If there are no active connections to a log:Log resource, then the log may be explicitly destroyed with the wsrl:Immedia-

Figure 4 Writing to a log

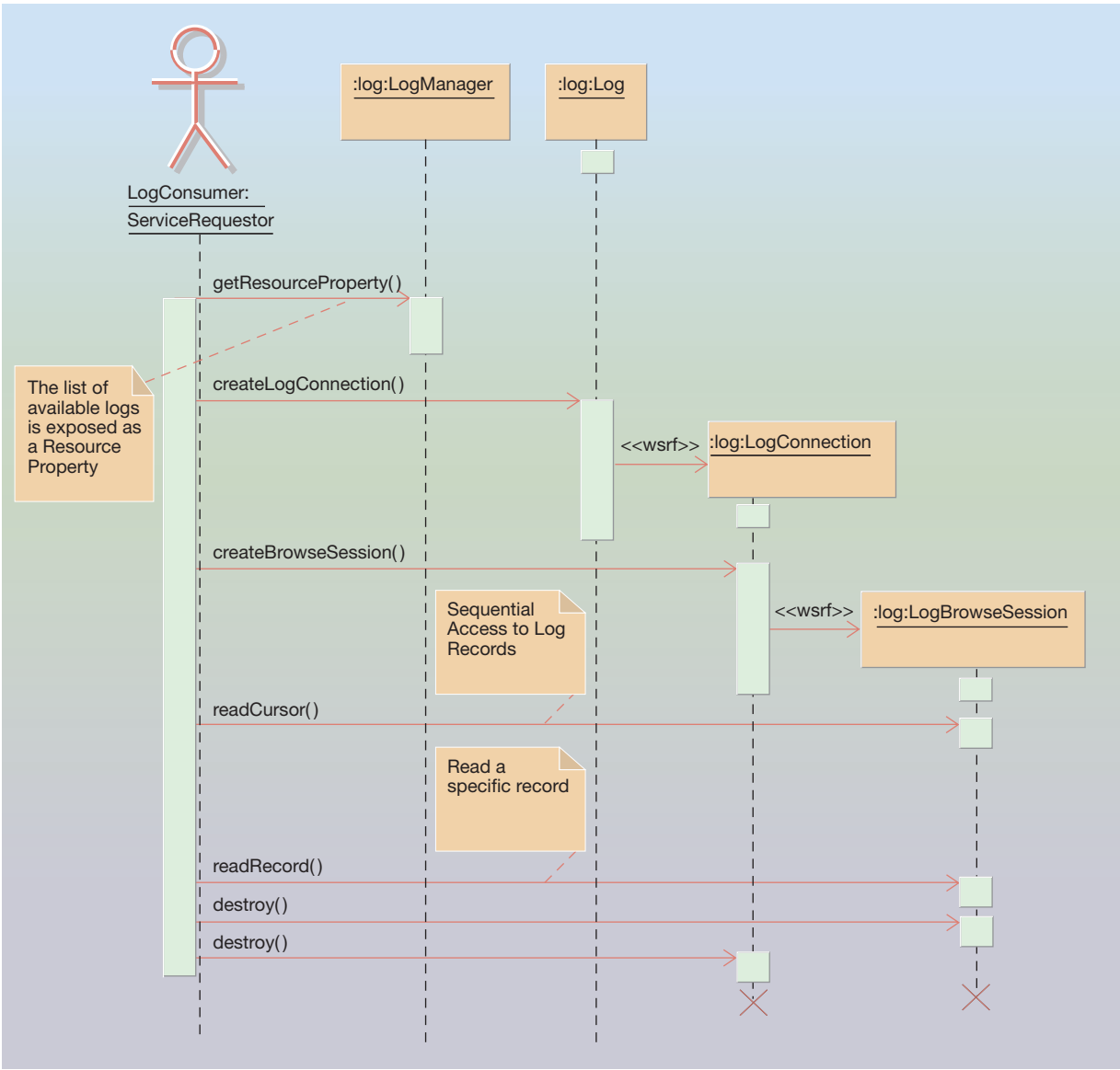


teResourceTermination portType destroy operation. If a log:Log resource is destroyed, then all log records associated with the log are lost.

The log:LogConnection interface. All accesses (read or write) to the records in a log are made through the log:LogConnection interface. The log:LogConnection portType inherits from the wsrp:ResourceProperties portType and the wsrl:ImmediateResourceTermination portType. The log:LogConnection portType contains the write operation, which is the basic mechanism for writing a log record. The writeWithCallback and the writeWithNoAck operations support the requirements outlined in our previous discussion of synchronous and asynchronous write semantics requirements. The delete operation deletes a range of log records from a log. All records older than the specified record are deleted. The deleteAll operation is used to delete all the log records from a log. The offload operation forces any cached log records to be written to a permanent store. The createLogBrowseSession operation of the log:LogConnection portType acts as a factory for log:LogBrowseSession resources. A user-specified read filter may be set for a LogBrowseSession instance. The home collection of all log:LogBrowseSession instances created by an instance of log:LogConnection is stored as wsa:EPR child elements in its resource property document. A log:LogConnection resource may be explicitly destroyed with the wsrl:ImmediateResourceTermination portType destroy operation; any log:LogBrowseSession resources in the home collection will also be destroyed. A sequence diagram illustrating the life cycle of a log:LogConnection is shown in Figure 4.

The log:LogBrowseSession interface. The log:LogBrowseSession interface is used to read log records from a log. The log:LogBrowseSession portType inherits from the wsrp:ResourceProperties portType and the wsrl:ImmediateResourceTermination portType. Each

Figure 5 Reading from a log



LogBrowseSession instance maintains a cursor in its resource properties for navigating its corresponding log. On creation, a custom read filter may be set for an instance of a LogBrowseSession interface. Reads are filtered by both the LogBrowseSession filter and the LogConnection read filter. The LogBrowseSession portType readRecord operation accesses records either by record identifier or by record time stamp. The readCursor operation reads a sequence of one or more records from the current cursor position. The reset operation resets the cursor position to either the oldest or youngest record in the log. As discussed previously, a browse session may support additional query mechanisms. A log:LogBrowseSession resource may be explicitly destroyed with the wsrl:ImmediateResourceTermination portType destroy operation. A sequence diagram illustrating the life cycle of a log:LogBrowseSession is shown in Figure 5.

Conclusion and future work

In this article we have outlined how the interfaces for a meta-OS Logger System that satisfies the requirements established in the OGSA-WG might be expressed by using the conventions and standards dictated by WSRF. It is our hope that the current work will help drive discussions surrounding the standardization of a WSRF-based Logger System. While we have addressed many of the requirements exposed by the OGSA-WG and documented in this paper, some require additional investigation. In particular, secure logging is an important topic that still needs to be studied. We also note that efficient local bindings for WSDL, specifically bindings much more efficient than SOAP over HTTP (Simple Object Access Protocol over HyperText Transfer Protocol), are required for practical implementations, particularly for log writes. We expect that such optimized bindings will become common in the not-too-distant future.

Acknowledgments

We would like to acknowledge the suggestions of the members of the OGSA-WG who reviewed and rereviewed the requirements described in this paper. Comments by Dave Berry, Abdeslem Djaoui, Andrew Grimshaw, Hiro Kishimoto, Takashi Kojo, Fred Maciel, Takuya Mori, Jeff Nick, Andreas Savva, Frank Siebenlist, David Snelling, Ravi Subramanian, Junichi Toyouchi, Jem Treadwell, and Jeffrin Von-Reich were particularly helpful. Jem Treadwell also helped us establish some of the specific requirements for the Logger System. Hany Salem, Mike Williams, and Bob Abrams have helped with requirements motivated by problem determination. In addition to helping with semantics, Jeff Frey provided much guidance on the use of WSRF. Benny Rochwerger helped author the initial logging submissions to the OGSA-WG logging focus session. Sylvain Reynaud and Jean-Pierre Prost also provided helpful reviews of these submissions. Finally, we would like to thank the anonymous reviewers for their suggestions and corrections.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Object Management Group, Inc.

Cited references

1. I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana, *Modeling Stateful Resources with Web Services, Version1.1,* http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf.
2. OASIS Web Services Resource Framework TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.
3. Open Grid Services Architecture Working Group, http://forge.ggf.org/projects/ogsa-wg.
4. OGSA Working Group Usecase Documents, https://forge.gridforum.org/docman2/ViewCategory.php?group_id=42&category_id=351.
5. OGSA Working Group Logging Focus Session, https://forge.gridforum.org/docman2/ViewCategory.php?group_id=42&category_id=468.
6. C. Lonvick, *The BSD syslog Protocol, RFC 3164,* http://www.ietf.org/rfc/rfc3164.txt.
7. *Microsoft Windows Event Logging*, Microsoft Corporation, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/debug/base/about_event_logging.asp.
8. *Systems Programmer's Guide to: z/OS System Logger,* SG24-6898-00, IBM Corporation (January 2004), http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246898.html.
9. J. Treadwell, *EVM: The Tru64 UNIX Event Management System*, Compaq White Paper, Compaq Corporation (July 2000).
10. *XML Schema for IBM Common Base Event, V 2.0* (October, 2003), http://xml.coverpages.org/CBE-SchemaV20.html.
11. D. Ogle, H. Kreger, A. Salahshour, J. Cornpropst, E. Labadie, M. Chessell, B. Horn, and J. Greken, *Canonical Situation Data Format: The Common Base Event ACAB.BO0301.2.0* (October 2003), http://xml.coverpages.org/CommonBaseEventSituationDataV210.pdf.
12. S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, S. Tuecke, W. Vambenepe, and B. Weihl, *Web Services Notification (WS-Notification), Version 1.0* (January 2004), http://www-106.ibm.com/developerworks/library/ws-resource/ws-notification.pdf.
13. Open Grid Services Architecture Security Working Group, http://forge.gridforum.org/projects/ogsa-sec-wg.
14. S. Graham, K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, F. Leymann, T. Maguire, N. Nagaratnam, M. Nally, T. Storey, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, and S. Weerawarana, *Web Services Resource Properties, Version 1.1* (March 2003), http://www-106.ibm.com/developerworks/library/ws-resource/ws-resourceproperties.pdf.
15. J. Frey, S. Graham, K. Czajkowski, D. F. Ferguson, I. Foster, F. Leymann, T. Maguire, N. Nagaratnam, M. Nally, T. Storey, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, and S. Weerawarana, *Web Services Resource Lifetime (WS-ResourceLifetime), Version 1.1* (March 2004), http://www-106.ibm.com/developerworks/library/ws-resource/ws-resourcelifetime.pdf.
16. A. Bosworth, D. Box, E. Christensen, F. Curbera, D. Ferguson, J. Frey, C. Kaler, D. Langworthy, F. Leymann, B. Lovering, S. Lucco, S. Millet, N. Mukhi, M. Nottingham, D. Orchard, J. Shewchuk, T. Storey, and S. Weerawarana, *Web Services Addressing (WS-Addressing)* (March 2004), ftp://www6.software.ibm.com/software/developer/library/ws-add200403.pdf.
17. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, *Web Services Description Language (WSDL)*, Version 1.1 (March 2001), http://www.w3.org/TR/wsdl.
18. *Unified Modeling Language,* http://www.omg.org/uml.
19. D. Box, F. Curbera, M. Hondo, C. Kaler, D. Langworthy, A. Nadalin, N. Nagaratnam, M. Nottingham, C. von Riegen, and J. Shewchuk, *Web Services Policy Framework (WSPolicy), Ver-*

sion 1.01 (June 2003), ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf.

Accepted for publication June 7, 2004.

Bill Horn *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (hornwp@us.ibm.com).* Dr. Horn is a member of the Scalable Systems Group at the IBM Thomas J. Watson Research Center. He received a Bachelor of Nuclear Engineering degree from the Georgia Institute of Technology, an M.E. degree in mechanical engineering from Cornell University, and a Ph.D. degree in engineering from Cornell University. Dr. Horn's current research interests include architectural issues related to distributed computing and, more specifically, architectural issues related to component transaction monitors. He also has an interest in the design and realization of commercial IT deployments that exploit distributed systems. His many years of engineering experience cover a wide variety of disciplines including projects in nuclear engineering, computational fluid dynamics, and product life-cycle management. More recent interests include visual and geometric computations.

Hariharan Balakrishnan *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (bharihar@in.ibm.com).* Mr. Balakrishnan is a software engineer at the IBM India Software Laboratories in Bangalore. He is currently on an international assignment at the Watson Research Center working on emerging technologies such as Web and grid services and on demand infrastructure development.

Biju T. Maniampadavathu *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (mbiju@in.ibm.com).* Mr. Maniampadavathu is a software engineer at the IBM India Software Laboratories in Bangalore. He is currently working as an international assignee in the Scalable Systems Group at the Watson Research Center, focusing on the logging and metering aspects of on demand infrastructure development. He received his B.S. degree in physics from Kerala University and a Master of Computer Applications degree from the National Institute of Technology in Calicut. His current work is on developing a distributed logging component for the IBM Virtualization Engine™.

James Warnes *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, NY 12601 (warnes@us.ibm.com).* Mr. Warnes is a member of the Advanced System Infrastructure Development Group at IBM Poughkeepsie. He received a Bachelor of Mathematics degree from Canisius College. His current interests lie in the area of on demand computing with a specific focus on logging. He has spent a significant amount of his career in z/OS® architecture, design, and development focusing on clustering technologies and shared message queues.

David A. Elko *IBM Systems and Technology Group, Austin Programming Lab, 904-3F18, Austin, Texas, 78758 (elko@us.ibm.com).* Dr. Elko is a member of the IBM Systems and Technology Group working in server technology and system architecture. He received his Bachelor of Mathematics degree from Indiana University of Pennsylvania (1976), and M.S. (1978) and Ph.D. (1984) degrees in mathematics from the University of Notre Dame, where his research interest was in algebraic topology. Dr. Elko joined IBM in 1980 working in the MVS™ operating system development organization. He later moved to the System/390® architecture department where he was a principle architect of the Parallel Sysplex® Coupling Facility. He received two corporate awards for his work on Parallel Sysplex, in 1994 and again in 2004. He has also worked on POWER™ architecture and memory compression techniques. He is a co-inventor on over 30 patents associated with high-end server architectures.