A Comparative Study of Web Services-based Event Notification Specifications

Yi Huang and Dennis Gannon
Department of Computer Science, Indiana University,
Bloomington, IN, 47405, USA
Email: {yihuan, gannon}@cs.indiana.edu

Abstract—**Web services-based event notification is an emerging technology that combines the asynchronous communication feature of event notification mechanisms and the interoperability feature of Web services technologies. Web services-based event notification systems are important components for service-oriented Grid computing. *WS-Eventing* and *WS-Notification* are two major competing specifications for these systems. This paper is a comparative study of these specifications. The focuses of this research are on identifying the similarities and differences between these two specifications and identifying their evolutionary path from previous specifications. We found that competing Web services specifications take ideas and concepts from each other during the development progress, which is good for the maturity of the Web services-based event notification technology. We also identified several major changes from previous event notification systems to Web services-based event notification systems. In the end, we will present our *WS-Messenger* project that supports both *WS-Eventing* and *WS-Notification* specifications and provides mediation between them.**

Index Terms— **event notification, Grid computing, mediation, publish/subscribe, Web service, WS-Eventing, WS-Notification**

I. Introduction

Event notification systems enable asynchronous communications among different entities in distributed systems. Web services (WS) technologies address the interoperability issues in heterogeneous distributed systems. Web services-based (WS-based) event notification systems combine features of both. They are essential components for Grid computing as it evolves a service-oriented architecture (SOA). Event notifications are disseminated for various purposes in Grid computing applications, such as logging, monitoring and auditing. Possible events include computation results, status updates, errors, exceptions, and so on.

Multiple competing specifications have been proposed to define the standard operations and message formats for WS-based systems. Two major competing specifications are Web Services Eventing (WS-Eventing) specification [1] and Web Services Notification (WS-Notification) specifications [2-4]. They are incompatible with each other. Prior to their emergence, several other specifications for event notification systems were proposed by various organizations and companies, including the CORBA notification service specification [5], the Java Message Service (JMS) specification [6] and the notification specification in Open Grid Services Infrastructure (OGSI) [7]. They also define interfaces for event notification systems to accommodate for the interoperability.

Both WS-Eventing and WS-Notification specifications can be used to build Open Grid Services Architecture (OGSA) [8] as described in [9]. What are the differences between these two WS-based event notification specifications? What problems are not addressed by previous event notification specifications? To answer these questions, we conducted a comprehensive comparative study between WS-Eventing and WS-Notification specifications based on our experiences with the *WS-Messenger* project [10]. We also compared different versions of these two specifications and compared them with previous major event notification specifications to find the evolutionary trends of event notification systems. In this paper, we will present our research results.

II. Related Works

Much literature is available on event notification systems, such as [11-13]. They discussed different projects and architectures of event notification systems. We focus our research on the evolution of major specifications because specifications are supported by multiple parties and are representatives of the evolution of event notification systems.

WS-Eventing and WS-Notification specifications received much attention since they are crucial for asynchronous Web services. Many news articles talked about both specifications. However, they are not comprehensive and in-depth. [14, 15] discuss each specification separately. A short paper [16] compares WS-Eventing and the old version of WS-Notification. However, in the comparison, it does not take into full account of WS-Resource Framework (WSRF) required in the old version of WS-Notification and reaches inappropriate conclusions that WS-Notification cannot *unsubscribe* and cannot send *SubscriptionEnd* notices. Our study differs from previous efforts in following aspects:

(1) It is based on the latest versions of both specifications. WS-Notification version 1.3 has had major updates. It no longer depends on WSRF and has included functions that previously only existed in WS-Eventing, such as the pull delivery mechanism.

(2) It is a comprehensive comparison between these two specifications. Our study focuses on the key functions of these

two specifications and provides side-by-side tables and graphs in our comparisons.

(3) Our study places these two specifications within a historical context and compares them with previous event notification specifications. This helps us understand the evolution of event notification systems. We also compared different versions of these two WS-based event notification specifications and found that these two specifications have been learning from each other and exhibit a trend of convergence.

(4) Our comparisons are based on our experiences in implementing both specifications so that we can detect detailed differences between these two, such as the difference in the *subscriptionID* enclosing element. Our WS-Messenger project implements both specifications and can mediate between them.

The rest of this paper is organized as follows: section III briefly introduces WS-based event notification systems and related specifications. Section IV discusses major changes in different versions of WS-Eventing specifications and WS-Notification specifications. Section V compares the latest version of these two specifications. In section VI, we will analyze evolutionary trends of event notification systems by comparing them with previous event notification specifications. We then present our WS-Messenger project in section VII and our conclusions in Section VIII.

III. Web service-based event Notifications

The Publish/Subscribe paradigm [11] is a practical pattern for event dissemination in distributed systems. In this paradigm, a *subscriber* subscribes to specific kinds of events to be sent to one or more *event consumers*. An *event producer* publishes events. Events are delivered to the event consumers based on the subscriptions. Notification brokers can be added to publish/subscribe systems to decouple event producers and event consumers. It provides flexibility and scalability.

Web services can integrate heterogeneous applications on the Internet. They are based on a set of open and widely adopted Internet standards. Features of Web services technologies include platform-independent, programming language-independent and transport-independent. Web service messages follow the XML standard [17] which defines a flexible and easy-to-extend data format and is supported on virtually every platform [18]. XML messages are encapsulated in SOAP [19] envelopes. Web Service Description Language (WSDL) [20] defines valid XML document structures for message exchanges to enable the interoperability feature of Web services. Web service messages can be transported using various transport mechanisms. SOAP messages are composable. Features like security, reliability and transaction can be added to the messages using respective specifications, such as WS-Security [21].

WS-based event notification systems utilize the Web service technology to deliver event notifications and manage subscriptions. A *subscriber* sends a SOAP-formatted subscription request message to an *event producer* Web service, requesting the delivery of certain kinds of notification messages to one or more *event consumer* Web services. When events are created in one service, other services can receive notification messages in the SOAP message format. The locations of the *event consumer* Web services are specified using the WS-Addressing specification [22]. Notification messages can be transported through intermediary and can use various transportation mechanisms.

In order to achieve interoperability among different vendors of WS-based notifications, vendors need to agree on a specification that defines the message formats and Web service interfaces for notification delivery and for subscriptions creation and management. Ideally, we wish there was only one specification agreed by all vendors. However, three similar specifications have been proposed in this area: WS-Events [23], WS-Eventing [1] and WS-Notification [2-4]. WS-Events was the earliest one. It was created by HP in mid-2003. HP joined others and proposed WS-Notification, which replaced WS-Events.

A. *WS-Eventing*

The Web Services Eventing (WS-Eventing) specification [1] has two released versions, the 1/2004 version and the 8/2004 version. The first version was released in January 7, 2004, led by Microsoft. The second version was released in August 2004. It received broader vendor supports. IBM, Sun and CA joined the supporters for this specification.

B. *WS-Notification*

The Web Services Notification (WS-Notification) specification was first released in January 20, 2004, led by IBM and Globus Alliance. It was refactored into a family of three individual specifications and a whitepaper in March 2004. These three specifications are: the Web Services Base Notification (WS-BaseNotification) specification [2], the Web Services Brokered Notification (WS-BrokeredNotification) specification [3] and the Web Services Topics (WS-Topics) specification [4]. WS-BaseNotification defines basic interactions between notification producers and notification consumers. WS-BrokeredNotification defines interfaces for notification brokers. WS-Topic defines a hierarchical topic space. The WS-Notification family was submitted to the standard organization, OASIS [24], in April 2004. Globus toolkit 4.0 [25] implements WS-BaseNotification.

WS-BaseNotification is very similar to WS-Eventing in architectures and functions. It has 3 major versions so far, 1.0, 1.2 and 1.3. Version 1.0 was released in March, 2004 by refactoring the first WS-Notification specification. Version 1.2 is the version submitted to OASIS, it is very similar to version 1.0. Version 1.3 has some major changes from previous versions. It has completed second public review at the time of this writing (2/2006).

The WS-Notification family was released together with the WS-Resource framework (WSRF). They are both developed by the Grid computing community. WSRF is a framework designed to manage Grid computing resources through Web services standards. It replaces the OGSI [7] specification. Before version 1.3, WS-Notification and WSRF are dependent on each other. In version 1.3 of the WS-Notification specification, WSRF becomes optional.

IV. Comparison of Different versions of WS-Eventing and WS-Notification Specifications

In this section, we will compare two existing versions of the WS-Eventing specification and Version 1.0 and 1.3 of the WS-BaseNotification specification. We do not include version 1.2 of WS-BaseNotification since it is very similar to version 1.0. An interesting observation we found in this comparison is that although these two specifications are competing with each other, they are converging with each other with each version update.

The first convergence happened when WS-Notification was refactored into three specifications. WS-BaseNotification is separated as an individual specification that is very similar to WS-Eventing.

The second convergence happened when WS-Eventing is updated in August 2004. (1) From the architecture perspective, it separates the *"subscription manager"* from the *"event source"* and the *"subscriber"* from the *"event sink"* following WS-Notification's architecture. (2) It also adopted WS-Notification's approach to treat subscriptions as resources in the subscription managers. Instead of using separate elements, the *subscriptionId* values are returned as *ReferenceParameters* in Web services addresses of subscription managers. (3) A new "*getStatus*" operation is added to query the statuses of subscriptions. This is similar to the *getResourceProperties* operation in WSRF. (4) The new version also adds the possibility to support a wrapped message delivery mode which has been defined in WS-BaseNotification. However, it does not specify message formats of the wrapped notification messages. (5) The pull delivery mode is added in this new version which is important for many scenarios, such as delivering messages to consumers behind firewalls. The first version of neither specification defined the pull delivery mode.

The third convergence is underway in the proposed version 1.3 of WS-BaseNotification. It adopts several functions already defined in WS-Eventing, including the pull delivery mode, the option to specify the subscription expiration using duration instead of absolute time, and the XPath-based subscription dialect [26]. One important change is that it makes WSRF optional by introducing the "renew" and "Unsubscribe" operations. Also the topic-based subscription is no longer required.

The highlighted cells in the upper part of Table 1 summarize the aforementioned evolutions. Although having competing specifications causes interoperability problems, the good point we can see from the comparisons is that these two specifications are taking ideas from each other to make up their own deficiencies. Architectural entities and functions are added, removed or modified with each update. This is a benefit of having competing specifications. It is good for the maturity of WS-based event notification technology in the long run.

	WSE 01/04	WSN 1.0	WSE 08/04	WSN 1.3
Version date	1/2004	3/2004	8/2004	2/2006
Separate Subscription Manager & Event Source	No	Yes	Yes	Yes
Separate subscriber & Event Sink	No	Yes	Yes	Yes
Getstatus operation	No	Yes	Yes	Yes
Return subscriptionId in WSA of Subscription Manager	No	Yes	Yes	Yes
Support Wrapped delivery mode	No	Yes	Yes	Yes
Support Pull delivery mode	No	No	Yes	Yes
Specify subscription expiration using duration	Yes	No	Yes	Yes
Specify XPath dialect	Yes	No	Yes	Yes
Filter element in Subscription message	Yes	No	Yes	Yes
Require WSRF	No	Yes	No	No
Require a topic in subscription	No	Yes	No	No
Require Pause/Resume subscriptions	No	Yes	No	No
GetCurrentMessage operation	No	Yes	No	Yes
Define Wrapped message format	No	Yes	No	Yes
Separate EventProducer & Publisher	No	Yes	No	Yes
Define PullPoint interface	No	No	No	Yes
Specify pull delivery mode in subscription	No	No	Yes	No
Require Getstatus	Yes	Yes	Yes	No
Require SubscriptionEnd	Yes	Yes	Yes	No
WS-Addressing version	2003/03	2003/03	2004/08	2005/08

Table 1. Comparisons among different versions of WS-Eventing (WSE) and WS-Notification (WSN) specifications.

We can also see from the highlighted cells in the lower part of table 1 that these two specifications still have some gaps in the architecture and the functions. We will study more in the next section.

V. Comparisons between WS-Eventing and WS-Notification

In this part, we will compare the latest versions of WS-Eventing (2004/08 version) and WS-Notification (version 1.3, Public Review Draft 2) from different perspectives, including architecture, functions, message delivery, message formats and broker supports.

In general, WS-Eventing is simpler than WS-Notification;

WS-Notification has more features than WS-Eventing and can be used in full-fledged notification systems. Since Web services specifications are composable, both WS-Eventing and WS-Notification define only key publish/subscribe related functions. Other functions, such as security, reliability and transaction management, depend on other WS-* specifications. For example, WS-Security [21] can be used to achieve secure delivery of messages.

1) *Architecture comparison*

WS-Eventing and WS-BaseNotification have almost identical WS-based architectures. They both follow the Publish/Subscribe paradigm. They both define the *subscriber* and s*ubscription manager* entities. The *event sink* defined in WS-Eventing is comparable to the *notification consumer* defined in WS-BaseNotification. In both specifications, subscribers are separated from notification consumers so that notification consumers only need to handle received messages. They do not need to know broker locations and create subscriptions. WS-Eventing does not separate the *publisher* from the *event source*. The Event source in WS-Eventing has the functions of both the *notification producer* and the *publisher* defined in WS-BaseNotification. Fig. 1 and Fig. 2 show the entities defined in WS-Eventing and WS-BaseNotification and their interactions. The bold lines indicate Web services interfaces.

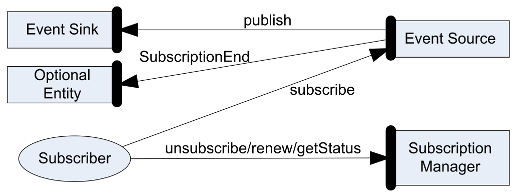

Fig. 1 WS-Eventing Architecture and Operations

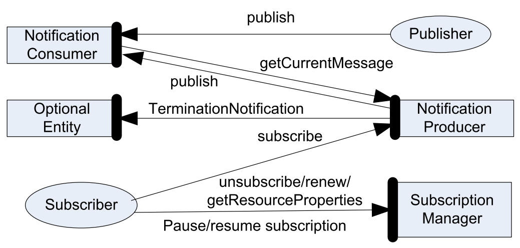

Fig. 2 WS-BaseNotification Architecture and Operations

2) *Function Comparison*

We can also find many similarities in functions of WS-BaseNotification and WS-Eventing. WS-Eventing defines five Web service operations: *Subscribe, Renew*, *GetStatus*, *Unsubscribe* and *SubscriptionEnd*. The *"Subscribe"* message is used to create a subscription for an event sink. The *"Renew"*, *"GetStatus"* and *"Unsubscribe"* messages are sent from subscribers to subscription managers to manage existing subscriptions. The *"SubscriptionEnd"* message is generated when an event source terminates a subscription unexpectedly. It is sent to an address specified in the subscription request. If this address is not presented in the subscription request, this *"SubscriptionEnd"* message is not generated.

WS-Eventing	WS-BaseNotification
Subscribe	Subscribe
Renew	Renew
Unsubscribe	Unsubscribe
GetStatus	Not defined, can use getResourceProperties in WSRF
SubscriptionEnd	Not defined, can use TerminationNotification in WSRF
Not available	Pause/resume Subscription
Not available	GetCurrentMessage

Table 2: Function Comparison

WS-BaseNotification has comparable operations for the above five operations. Although it does not define *GetStatus* and *SubscriptionEnd* operations, these operations can be achieved with the optional WS-ResourceFramework (WSRF) since WS-Notification can treat subscriptions as WS-Resources in WSRF. Table 2 shows how WS-BaseNotification achieves the 5 functions defined in WS-Eventing. Besides these five operations, WS-Notification defines three more operations than WS-Eventing. It defines how to *pause* and *resume* a subscription and how to get the current message (*getCurrentMessage*).

3) *Message Delivery Comparison*

In this section, we will compare in detail how to specify message delivery in subscription requests of WS-Eventing and WS-Notification.

Delivery mode: Both WS-Eventing and WS-Notification can use "push", "pull" and "wrapped" mode to deliver notification messages. The "wrapped" mode can pack several notification messages into one message for efficient delivery. WS-Eventing defines the "push" mode as the default delivery mode. It uses the "*Delivery*" extension point in a subscription message to support other delivery modes. Notification message formats are not defined in the specification. WS-Notification defines a "PullPoint" interface, but it can not specify using the "pull" delivery mode in a subscription message. A 'pullpoint" needs to be created before creating a subscription and is treated as a regular "push" event consumer from a publisher's prospective.

Filter: WS-Notification defines three types of message filters: TopicExpression, ProducerProperties and MessageContent. A subscriber can use any or all of these filters. WS-Eventing allows at most one filter in subscription requests. The default filter is a content-based filter using XPath expressions. Both specifications can use any expressions (xsd:any) in a specified dialect that evaluates to a Boolean value as a filtering criteria. WS-Eventing does not specify a way to filter messages using the ProducerProperties of publishers.

Message encapsulation: WS-Notification defines two ways to send notification messages. The first way is to wrap the notification message content into a "Notify" message element and add additional WS-Notification-defined information (such as a Topic) to it. The second way is just to send the "raw" notification message content in the body of a SOAP message. WS-Eventing just uses the raw message approach.

4) *Message Formats comparison*

Web services specifications define SOAP message formats to encapsulate request and response messages. Since WS-Eventing and WS-Notifications are two different specifications, their message formats are different. When comparing the request and response SOAP messages in corresponding operations, such as the *subscribe* operation, many differences exists. The differences can be summarized in the following categories:

(1) Element names or attribute names difference: The element names or attribute names for the same content are different. For example, WS-Eventing encloses the *subscriptionID* value in a subscription response message using the *ReferenceParameters* element defined in WS-Addressing, while WS-BaseNotification encloses it in the *ReferenceProperties* element defined in WS-Addressing.

(2) Namespaces difference: The namespaces of these two specifications and some namespaces used in the specifications are different, such as the WS-Addressing namespace.

(3) Versions difference of underlying specifications: The WS-Addressing versions used in these two specifications are different. WS-Notification version 1.3 uses the *2005/08* version, while WS-Eventing uses the *2004/08* version.

(4) Message contents difference: The specifications define different required values for certain XML elements in SOAP messages. For example, different values are required by different specifications for the "*action*" elements in the WS-Addressing part of SOAP headers.

(5) SOAP message structures difference: Different specifications define different XML message structures SOAP messages. For example, a wrapped WSN notification message encloses the message payload in a *NotificationMessage* element which is again enclosed in a *Notify* element. WSE notification messages do not need such structures.

(6) Content locations difference: The same semantic information may appear in different locations in SOAP messages. For example, a wrapped WS-Notification notification message requires a *topic* element in the *SOAP body*, while a WSE notification message needs to place it in the *SOAP header* if needed.

5) *Broker Support Comparison*

The WS-BrokeredNotification specification in the WS-Notification family defines broker supports between notification producers and notification consumers. It is the extension of the WS-BaseNotification specification. Notification brokers can handle publisher registrations and support demand-based publishers. A d*emand-based publisher* only publishes messages when there are consumers who are interested in these messages. A notification broker can keep track of the number of consumers to each kind of messages and can pause or resume subscriptions to publishers based on the demand. WS-Eventing does not define how to use a broker as the intermediary between *eventSink* and *evnetSource*. However, it is possible to create a broker that implements both the *eventSink* interface and the *eventSource* interface. Neither publisher registrations nor demand-based publishers are defined in WS-Eventing.

VI. Previous Event Notification Specifications

Prior to WS-based event notification specifications, several other specifications have attempted to define a standard way of sending event notifications in distributed systems. WS-Eventing and WS-Notification specifications have many similarities to previous specifications. However, they address some unique issues that are not covered by previous specifications, such as XML format and XPath filter. In this section, we will present a review of major event notification specifications prior to the announcements of these two WS-based event notification specifications and compare them with these two new specifications. The comparisons are based on the latest released versions of each specification.

A. CORBA Event service specification and Notification service specification

The Common Object Request Broker Architecture (CORBA) [27] specification was developed by the Object Management Group (OMG), which is a consortium of over 700 member companies. CORBA is designed to be programming language-, operating system-, and vendor-independent. It defines common interfaces for different programming languages and allows different programs to communicate through Object Request Broker (ORB). CORBA uses General Inter-ORB Protocol (GIOP) for intranet communications and Internet Inter-ORB Protocol (IIOP) for the Internet communications. IIOP maps requests and replies of GIOP to the Internet's TCP layer in each computer. The message payload is in a binary format known as Common Data Representation (CDR).

CORBA defines Event services and Notification services to support interactions among CORBA objects. The specifications of these services define both the interfaces and the underlying infrastructures for CORBA notification systems. They are based on the publish/subscribe paradigm. Event suppliers and event consumers communicate with each other through event channels.

The CORBA Event Service specification [28] was first introduced in March, 1995. It is intended to decouple clients and servers so that the servers do not have to keep a list of client callback registrations. According to this specification, the event supplier publishes events to a CORBA event service channel. The event consumers get events from the channel. Both "Push" and "Pull" modes are supported. CORBA event service achieves asynchronous communications between suppliers and consumers. They are location transparent. Although CORBA Event Service defines a simple mechanism for event propagation, it has noticeable drawbacks. It does not address event filtering and Quality of Service (QoS). A consumer receives all events on a channel.

The CORBA Notification service specification [5] is an enhancement to the CORBA event service specification. It adds supports for event filtering and Quality of Service (QoS). The CORBA notification service specification introduced "Structured Events" which provides a well-defined data structure to map a generic event to a well structured event. The structured event is useful for efficient filtering. The event filtering in the notification service is based on a filter object. The filter language is an expression whose syntax follows the extended Trader Constraint Language.

CORBA Notification specification defines 13 QoS properties that must be understood by all implementations even though they are not required to be implemented. Other QoS properties can be extended.

Although CORBA has implementations on different platforms and in different programming languages, the reality is that any solution built on CORBA will depend on a single-vendor's implementation. Vendors like to deploy their products on every nodes and using their own middleware to integrate these nodes. They do not have the incentive to achieve interoperability with others. Implementations from different vendors cannot interoperate well, especially when it comes to security, transaction management and performance optimization [18]. CORBA can only achieve interoperability on the intranet scale, where the distributed environment is well managed and has predictable latencies.

B. *JMS Specification*

Java Message service (JMS) [6] is a specification created by Sun Microsystems. It describes the APIs for Java programs to create, send, receive and read an enterprise messaging system's message. JMS is widely used in J2EE enterprise applications.

JMS defines two messaging styles: the point-to-point message queue style and the publish/subscribe style. It also defines five message types: textMessage, byteMessage, mapMessage, streamMessage and objectMessage.

JMS messages have well defined structure in the header field for efficient filtering. Subscribers can express their interests in JMS messages using queue names, topic names or message selectors. A message selector defines selecting criteria based on the header fields using an expression whose syntax is a subset of the SQL92 conditional expression. QoS criteria defined in JMS are priority, persistence, durability, transaction and message order.

The limitation of the JMS specification is that it only works on Java platforms.

C. *Notification in the OGSI Specification*

The Open Grid Services Infrastructure (OGSI) specification [7] is created by the Global Grid Forum. It is targeted for Grid computing which tries to coordinate computing resources across the Internet. OGSI defines mechanisms for creating, managing, and exchanging information among Grid services. By using OGSI, various Grid resources (e.g. CPUs, storage devices, databases) provide uniform interfaces to the upper level services defined in OGSA (Open Grid Services Architecture). Grid services use an extension of WSDL [20] to define services interfaces.

Notification is an important part of OGSI. The OGSI Notification specification is very simple. It is also based on the publish/subscribe paradigm. A "*NotificationSink"* sends subscription to a "*NotificationSource"* indicating the *service data* name (a string) it is interested in. The "notificationSource" pushes notification messages to the "notificationSink" when the specified service data is changed. The notification framework allows both direct service-to-service notification message delivery and integration of intermediary delivery services [6].

OGSI notification is an intermediary step towards WS-based event notification. It uses XML documents as the message payload and uses HTTP as the transport protocol. However, since Grid services are extensions of Web services, it is hard to use widely-available Web services tools to develop Grid services. With the introduction of the Web services resource framework (WSRF) and WS-Notification in January 2004, Grid community and Web services community are converged to Web services standards. WSRF replaces OGSI as the foundation of OGSA. OGSI Notification is replaced by WS-Notification.

D. *Comparison with Web services-based event notification specifications*

Table 3 compares the four specifications we discussed in this section with major WS-based event notification specifications. From this comparison, we can see how the event notification specifications have been evolving over time. Several interesting observations are found in this comparison:

(1) The event delivery scope is extended to the Internet scale. The message delivery mechanism is moving towards

	CORBA Event Service	CORBA Notification Service	JMS	OGSI-Notification	WS-Notification	WS-Eventing
First Release	3/1995	6/1997	1998	6/27/2003	1/20/2004	1/7/2004
Latest Release	10/2/2004	10/11/2004	4/12/2002	6/27/2003	2/2006	8/30/2004
Creator(s)	OMG	OMG	Sun Microsystems	Global Grid Forum	IBM, Sonic, TIBCO, Akamai, SAP, CA, HP Fujitsu, Globus,	IBM, BEA, CA, Sun, Microsoft, TIBCO
Message transport	RPC	RPC	RPC	HTTP RPC	Transport independent	Transport independent
Intermediary	EventChannel object	EventChannel object	Message Queue, Pub/Sub broker	directly or through intermediary	directly or through broker	directly or through broker
Delivery Mode	Push, pull & both	Push, pull & both	Pull, Push	Push	Push, Pull	Push by default, Can use Pull or other modes
Message Structure	Generic (Anys), Typed	Generic (Anys), Typed, Structured, sequences of structured	TextMessage, ByteMessage, MapMessage, StreamMessage, ObjectMessage	SOAP with Xml based Service data Elements	SOAP (with Raw XML data or wrapped messages)	SOAP (with Raw XML data only). Can use wrapped mode.
Filter	No	Channel, Filter Object.	Queue/topic name, message selector on header fields	ServiceDataName. Can add other filter services.	Hierarchy Topic tree; Content Selector. Producer properties.	A "Filter" element for any filter. At most 1 filter.
Filter language	No	Extended Trader Constraint Language	a subset of the SQL92 conditional expression syntax	ServicedDataName String or other expressions.	Any expression (xsd:any) that evaluates to a Boolean. e.g. XPath	Default XPath. Can use any expression (xsd:any) that evaluates to a Boolean.
QoS criteria	Not defined	Defined 13 QoS properties, can be extended to others	Priority; persistence; durable; transaction; message order	Not defined	Depends on composition with other WS* specification	Depends on composition with other WS* specification
Subscription Timeout	No	No	No	Absolute Time	Absolute Time or duration	Absolute time or duration
Demand-based	No	Defined	No	No	Defined	No
Management operations	connect_*, obtain _(typed)_push /pull_supplier/ consumer	connect_*, obtain_notification_p ull/push_supplier/con sumer, suspend/resume_con nection., get/set/validate_qos, add/remove/get/getAl l/removeAll_filter, obtain_subscription/o ffered_types	createSubscriber, createDurableSu bscriber, unsubscribe	Subscribe, requestTermination After, requestTermination Before, destroy	Subscribe,Renew, unsubscribe, Pause/resume subscription, get/getMultiple/set/query ResourceProperties, TerminationNotification, Destroy, SetTerminationTime	Subscribe, Renew, GetStatus, Unsubscribe, SubscriptionEnd

Table 3: Comparison among specifications on event notifications

transport-independent.

(2) XML-based SOAP messages are used as message payloads.

(3) The message filtering mechanism is moving from the simple subject-based topic filtering to the content-based XPath filtering.

(4) The criteria of Quality of Service (QoS), such as reliability, transaction, are no longer defined in the specifications. Instead, they depend on the composition with other WS-* specifications, such as WS-Reliability, WS-Transaction.

(5) The soft-state management (timeout) of subscription terminations is used. The connections to event consumers do not always keep alive.

(6) Interoperability concerns are shifted from the fine-grained API level to the more coarse-grained service interfaces and SOAP messages level. Event producers, event consumers and brokers can interoperate with each other using SOAP messages with standard formats. They do not need to use implementations from the same vendor.

VII. The WS-Messenger Project

The WS-Messenger project is an on-going project at Indiana University. It aims to create a scalable, reliable and efficient WS-based message broker that sends WS-based event notification messages among heterogeneous applications and platforms. It implements both WS-Eventing and WS-Notification specifications and can support both specifications at the same time through a mediation approach. To our best knowledge, WS-Messenger is the first open source project that support two competing Web services specifications and provides mediation between them.

The mediation techniques used in WS-Messenger reconcile the differences between WS-Eventing and WS-Notification specifications. WS-Messenger automatically detects which specification the incoming SOAP messages use and processes them accordingly. Response messages follow the same specifications as request messages. When delivering notification messages, WS-Messenger makes sure that notification messages follow the expected specifications of the target event consumers. The specification type of a target event consumer is determined by the subscription request message type for that notification consumer. In such way, an event producer can publish event notifications using either the WS-Eventing specification or the WS-Notification specification. It makes no difference to the event consumers since WS-Messenger performs mediations automatically.

Besides using the default message filtering, WS-Messenger provides a generic interface that can use existing publish/subscribe systems as the underlying message systems. In this way, WS-Messenger provides Web service interfaces to existing messaging systems. The architecture of WS-Messenger and its application in Grid computing projects are presented in [10].

VIII. CONCLUSIONS

WS-based event notification systems are key components in service-oriented Grid-computing. WS-Eventing and WS-Notification are two competing specifications for such systems. In this paper, we compared these two specifications side-by-side based on our experiences with the *WS-Messenger* project. By comparing them with previous specifications for event notification systems, we studied the evolution of event notification specifications and identified some key shifts from previous specifications to the WS-based event notification specifications.

WS-Eventing and WS-Notification specifications are not yet finalized and we expect both specifications to have future updated versions. By comparing different versions of these two specifications, we found that they are adopting ideas and concepts from each other and getting more mature with each update. We see a trend of convergence of both specifications. Recently, a white paper [29] from IBM, Microsoft, HP and Intel proposes creating a new standard, WS-EventNotification, that will integrate functions from WS-Notification with WS-Eventing. However, both WS-Eventing and WS-Notification will still be supported.

WS-Messenger is an on-going project that supports both WS-Eventing and WS-Notification specifications and provides mediation between them. It also has the capability to wrap other publish/subscribe messaging systems as WS-based event notification systems.

REFERENCES

[1] D. Box, L. F. Cabrera, et al., "Web Services Eventing", Available: http://ftpna2.bea.com/pub/downloads/WS-Eventing.pdf

[2] S. Graham and B. Murray, "Web Services Base Notification(v1.2)", Available: http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf

[3] D. Chappell and L. Liu, "Web Services Brokered Notification (v1.2)", Available: http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BrokeredNotification-1.2-draft-01.pdf

[4] W. Vambenepe, "Web Services Topics (v1.2)", Available: http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-01.pdf

[5] OMG, "CORBA Notification Service Specification", Available: http://www.omg.org/technology/documents/formal/notification_service.htm

[6] M. Hapner, R. Burridge, et al., "Java Message Service (Version 1.1)", Available: http://java.sun.com/products/jms/

[7] S. Tuecke, I. Foster, et al., "Open Grid Services Infrastructure ", Available: http://www-unix.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf

[8] I. Foster, C. Kesselman, et al., "Grid Services for Distributed System Integration," *Computer*, vol. 35, 2002.

[9] M. Humphrey, G. Wasson, et al., "Alternative Software Stacks for OGSA-based Grids," *Proceedings of Supercomputing 2005*, 2005.

[10] Y. Huang, A. Slominski, et al., "WS-Messenger: A Web Services based Messaging System for Service-Oriented Grid Computing," *6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid06)*.

[11] P. T. Eugster, P. Felber, et al., "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, vol. 35, 2003.

[12] C. Wang, A. Carzaniga, et al., "Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems," *Hawaii International Conference on System Sciences*, 2002.

[13] A. Carzaniga, D. S. Rosenblum, et al., "Achieving scalability and expressiveness in an internet-scale event notification service," *Proceeding of Nineteenth ACM Symposium on Principles of Distributed Computing (PODC 2000)*, 2000.

[14] S. Vinoski, "Web Services Notifications," *IEEE Internet Computing*, vol. 8, 2004.

[15] S. Vinoski, "More Web Services Notifications," *IEEE Internet Computing (May/June, 2004)*, vol. 8, pp. 90-93.

[16] S. Pallickara and G. Fox, "An Analysis of Notification Related Specifications for Web/Grid applications," *International Conference on Information Technology: Coding and Computing (ITCC'05)*, 2005.

[17] W3C, "Extensible Markup Language (XML) 1.0 (Second Edition)", Available: http://www.w3.org/TR/2000/REC-xml-20001006

[18] D. Gisolfi, "Is Web services the reincarnation of CORBA?" Available: http://www-106.ibm.com/developerworks/webservices/library/ws-arc3/#resources

[19] W3C, "SOAP Version 1.2", Available: http://www.w3.org/TR/soap12-part1/

[20] W3C, "Web Services Description Language (WSDL) 1.1", Available: http://www.w3.org/TR/wsdl

[21] A. Nadalin and et.al, "WS-Security 1.0", Available: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[22] D. Box, F. Curbera, et al., "Web Services Addressing (WS-Addressing)", Available: http://www.w3.org/Submission/ws-addressing/

[23] N. Catania and et.al, "Web Services Events (WS-Events) Version 2.0", Available: http://devresource.hp.com/drc/specifications/wsmf/WS-Events.pdf

[24] "OASIS organization", Available: http://www.oasis-open.org/

[25] B. Sundaram, "WS-Notification and the Globus Toolkit 4 WS-Java Core", Available: http://www-128.ibm.com/developerworks/grid/library/gr-wsngt4/

[26] J. Clark and S. DeRose, "XML Path Language (XPath) Version 1.0", Available: http://www.w3.org/TR/xpath

[27] OMG, "Common Object Request Broker Architecture: Core Specification", Available: http://www.omg.org/docs/formal/04-03-01.pdf

[28] OMG, "CORBA Event Service Specification", Available: http://www.omg.org/technology/documents/formal/event_service.htm

[29] K. Cline, J. Cohen, et al., "Toward Converging Web Service Standards for Resources, Events, and Management," 2006.