Publish-Subscribe Notification for Web services

Version 1.0

03/05/2004

Authors

Steve Graham, IBM (editor)
Peter Niblett, IBM (editor)
Dave Chappell, Sonic Software
Amy Lewis, TIBCO Software
Nataraj Nagaratnam, IBM
Jay Parikh, Akamai Technologies
Sanjay Patil, SAP AG
Shivajee Samdarshi, TIBCO Software
Igor Sedukhin, Computer Associates International
David Snelling, Fujitsu Laboratories of Europe
Steve Tuecke, Globus / Argonne National Laboratory
William Vambenepe, Hewlett-Packard
Bill Weihl, Akamai Technologies

Abstract

The Event-driven, or Notification-based, interaction pattern is a commonly used pattern for inter-object communications. Examples exist in many domains, for example in publish/subscribe systems provided by Message Oriented Middleware vendors, or in system and device management domains. This notification pattern is increasingly being used in a Web services context.

WS-Notification is a family of related white papers and specifications that define a standard Web services approach to notification using a topic-based publish/subscribe pattern. It includes: standard message exchanges to be implemented by service providers that wish to participate in point to point notifications, standard message exchanges for a notification broker service provider (allowing publication of messages from entities that are not themselves service providers), operational requirements expected of service providers and requestors that participate in notifications, and an XML model that describes topics of subscription. The WS-Notification family of documents includes: a white paper: *Publish-Subscribe Notification for Web services* as well as three normative specifications: WS-BaseNotification, WS-BrokeredNotification, and WS-Topics.

This document introduces the notification pattern, sets the goals and requirements for the WS-Notification family of specifications and describes each of the specifications that make up this family. It also defines a set of terms and concepts used in the specifications, provides some examples, and includes a discussion of security considerations.

Status

This whitepaper is an initial draft release and is provided for review and evaluation only. The Companies hope to solicit your contributions and suggestions in the near future. The Companies make no warranties or representations regarding the specification in any manner whatsoever.

Table of Contents

1 INTRODUCTION 4

1.1 GOALS AND REQUIREMENTS 5
1.1.1 Requirements 5
1.1.2 Non-Goals 6

2 OVERVIEW OF THE WS-NOTIFICATION SPECIFICATIONS 6

3 TERMINOLOGY AND CONCEPTS 8

4 EXAMPLE 12

5 SECURITY CONSIDERATIONS 15

5.1 SECURING THE MESSAGE EXCHANGES 15
5.2 SECURING SUBSCRIPTIONS AND NOTIFICATIONS 17

6 ACKNOWLEDGEMENTS 18

7 REFERENCES 18

1 Introduction

The Event-driven, or Notification-based, interaction pattern is a commonly used pattern for inter-object communications. Examples exist in many domains, for example in publish/subscribe systems provided by Message Oriented Middleware vendors, or in system and device management domains. This notification pattern is increasingly being used in a Web services context.

In the *notification pattern* a Web service, or other entity, disseminates information to a set of other Web services, without having to have prior knowledge of these other Web services. Characteristics of this pattern include:

- The Web services that wish to consume information (which we call NotificationConsumers) are registered dynamically with the Web service that is capable of distributing information. As part of this registration process the NotificationConsumers may provide some indication of the nature of the information that they wish to receive.
- The distributing Web service disseminates information by sending one-way messages to the NotificationConsumers that are registered to receive the information. It is possible that more than one NotificationConsumer is registered to consume the same information. In such cases, each NotificationConsumer that is registered receives a separate copy of the information.
- The distributing Web service may send any number of messages to each registered NotificationConsumer; it is not limited to sending just a single message. Note also that a given NotificationConsumer may receive zero or more NotificationMessages throughout the time during which it is registered.

WS-Notification standardizes the roles, concepts, message exchanges, WSDL 1.1 and XML Schema renderings required to express the pattern. The benefits of standardization include:

- **Interoperation between NotificationProducers and NotificationConsumers.** WS-Notification specifies a standard set of message exchanges that define the roles of NotificationProducer and NotificationConsumer. Any services that implement these message exchanges will be able to exchange notifications, subject to there being a common transport binding.
- **Interoperation between middleware providers.** The WS-Notification interfaces have been defined in a way which allows the implementation to be delegated to a middleware provider. This specification allows different middleware providers to interoperate.
- **Standardized mechanism to develop Topic taxonomies.** Interoperation is facilitated by having a standard way to name and describe Topics that is not tied to a particular implementation.
- **Standardized concepts and terminology.** A common set of concepts and terms simplifies the job of the application developer as well as aiding interoperation.

1.1 Goals and Requirements

The goals of WS-Notification are to standardize the roles, terminology, concepts, message exchanges and the WSDL needed to express the notification pattern, and to provide a language to describe Topics.

1.1.1 Requirements

In meeting these goals, the WS-Notification specifications must explicitly address the following requirements:

- **Must support resource-constrained devices.** The specifications must be factored in a way that allows resource-constrained devices to participate in the Notification pattern. Such devices will be able to send information to, and receive information from Web services, without having to implement all the features of the specifications.
- **Must support both direct and brokered Notification:** The specifications must define a NotificationBroker role. A NotificationBroker is a Web service that acts as a intermediary between the producer of the information and the NotificationConsumers that receive it. The specifications must allow a given NotificationProducer or NotificationConsumer to participate in both brokered and non-brokered configurations.
- **Must permit transformation and aggregation of Topics:** It must be possible to construct configurations (using intermediary brokers) where the Topic subscribed to by the NotificationConsumer differs from the Topic published to by the NotificationProducer, yet NotificationMessages from the NotificationProducer are routed to the NotificationConsumer by a broker that is acting according to administratively-defined rules.
- **Must provide runtime metadata:** There must be a mechanism that lets a potential Subscriber discover what elements available for subscription are provided by a NotificationProducer, and in what formats the subscription for notification can be made.

In addition, the WS-Notification specifications must allow for the following requirements to be met

- **WS-Notification must be independent of binding-level details:** Transport protocol details must be orthogonal to the subscription and the delivery of the notifications, so that the specification can be used over a variety of different transports.
- **Must allow for Message Oriented Middleware implementations.** The design of the WS-Notification specifications must allow a service that is acting as a NotificationProducer to delegate its implementation of WS-Notification semantics to a Message Oriented Middleware provider.
- **Must allow for federation of brokers.** It must be possible to build configurations with multiple intermediary broker services in a dynamic fashion. The specifications must allow for a variety of broker topology usage patterns. Among other things, these allow for greater scalability and permit sharing of administrative workload.
- **Relationship to other WS-* specifications:** WS-Notification must be composable with other Web services specifications, in particular WS-Security, WS-Policy, WS-Federation, WS-Addressing, WS-Coordination, WS-ResourceProperties, WS-ResourceLifetime, WS-ReliableMessaging [WS-ReliableMessaging] and the WS-Resource framework [State Paper].

1.1.2 Non-Goals

The following topics are outside the scope of these specifications:

- **Defining the format of notification payloads:** The data carried in NotificationMessage payloads is application-domain specific, and WS-Notification does not prescribe any particular format for this data**.**
- **Defining any Events or NotificationMessages.** The specifications do not define any "standard" or "built-in" notification situations, events or messages.
- **Defining the mapping between Situations and NotificationMessages.** The specifications do not define the circumstances under which a potential producer of information should decide if and when it should actually notify the registered NotificionConsumers. However they do define how it performs the notification once it has decided to do so.
- **Defining the means by which NotificationProducers and NotificationBrokers are discovered by subscribers** It is beyond the scope of this specification to define the mechanisms for runtime discovery of NotificationProducers and NotificationBrokers.
- **Defining the specific policy language to be used to govern specifics of the notification message exchange between the NotificationProducer and the NotificationConsumer.** The current family of specifications does not define this policy language. It is expected that this will be added in the future as a further specification.

2 Overview of the WS-Notification Specifications

WS-Notification is packaged as a family of related specification documents. This follows the standard Web services practice of having small composable specifications, and makes the functional layering of the total specification more apparent to readers. This approach also provides a flexible framework so that, over

time, additional functionality can be added without requiring a revision of existing specifications.

The term **WS-Notification** is used to refer to this family of specifications as a whole. This family consists of the following documents:

Publish-Subscribe Notification for Web Services (this document)

This document introduces the Notification pattern, sets the goals and requirements for the WS-Notification family of specifications and describes each of the specifications that make up this family. It also defines a set of terms and concepts used in the specifications, provides some examples, and includes a discussion of security considerations. This document should be read before reading the other WS-Notification specifications.

Web Services Base Notification

The WS-Base Notification specification defines the Web services interfaces for NotificationProducers and NotificationConsumers. It includes standard message exchanges to be implemented by service providers that wish to act in these roles, along with operational requirements expected of them. This is the base document on which the other WS-Notification specification documents depend. An implementer interested just in direct, point to point, notification need only read the *WS-Base Notification* specification, together with the *Publish-Subscribe Notification for Web Services* white paper.

Web Services Topics

The WS-Topics specification defines a mechanism to organize and categorize items of interest for subscription known as "topics". These are used in conjunction with the notification mechanisms defined in *WS-Base Notification*. WS-Topics defines three topic expression dialects that can be used as subscription expressions in subscribe request messages and other parts of the WS-Notification system. It further specifies an XML model for describing metadata associated with topics. The WS-Topics specification should be read in conjunction with the *WS-Base Notification* specification and the *Publish-Subscribe Notification for Web Services* white paper.

Web Services Brokered Notification

The WS-Brokered Notification specification defines the Web services interface for the NotificationBroker. A NotificationBroker is an intermediary which, among other things, allows publication of messages from entities that are not themselves service providers. It includes standard message exchanges to be implemented by NotificationBroker service providers along with operational requirements expected of service providers and requestors that participate in brokered notifications. This work relies upon *WS-Base Notification* and *WS-Topics*, as well as the *Publish-Subscribe Notification for Web Services* document.

Wherever possible, WS-Notification composes with other WS-* specifications, in order to avoid duplication of function. In particular it leverages the work defined in the WS-Resource [State Paper] family of specifications. A WS-Resource is a term used to describe the relationship between a Web service and a stateful resource by defining a so-called *implied resource pattern*. This pattern standardizes the way in which a Web service message contains an identifier of the stateful resource to be

used in the execution the message. Several aspects of WS-Notification, particularly Subscriptions, utilize this concept.

WS-Notification also uses WS-ResourceProperties [WS-ResourceProperties]. WS-ResourceProperties defines a mechanism by which a WSDL portType can be associated with an XML element that describes the data associated with a resource. This specification includes standard message exchanges associated with reading and writing a resource's data.

WS-Notification uses WS-ResourceLifetime [WS-ResourceLifetime] to provide direct, immediate destruction of resources as well as scheduled destruction of resources, based on a leasing or time-based model.

3 Terminology and Concepts

The following definitions outline the terminology and usage in the WS-Notification family of specifications.

Situation:

- A Situation is some occurrence within a Web service or its environment of interest to third parties.
- A Situation could be a change of the internal state of a Web service or an associated resource or could be environmental, such as a timer event. It could also be an external event, such as a piece of news that has been supplied by a news-feed service.
- WS-Notification does not specify what a Situation is or is not, nor does it define the relationship between a Situation and the NotificationMessage(s) that are used to describe it.

NotificationMessage:

- A NotificationMessage is an artifact of a Situation containing information about that Situation that some entity wishes to communicate to other entities.
- A NotificationMessage is represented as an XML element with a Namespace qualified QName and a type defined using XML Schema.
- A typical usage pattern is to define a single NotificationMessage type (to be precise, its defining XML element) for each kind of Situation, containing information pertinent to that kind of Situation; in this case one can think of a NotificationMessage instance as in some sense *being* (or at least representing) the Situation.
- A designer could choose to associate several different NotificationMessage types with a Situation, for example, describing different aspects of the Situation, destined for different target recipients, etc. Conversely it is possible that several essentially different Situations give rise to NotificationMessages of the same type.

Notification:

- A Notification is the act of transmitting a NotificationMessage to an interested party.

NotificationProducer:

- A NotificationProducer is a Web service that implements the message exchanges associated with the NotificationProducer interface and supports one or more Topics.
- A NotificationProducer is capable of distributing NotificationMessages. It maintains a list of Subscription resources and when it has a NotificationMessage to distribute; it matches the NotificationMessage (and its associated Topic) against the interest registered in each Subscription in its list. If it identifies a match it issues a Notification to the NotificationConsumer associated with that Subscription.
- A Web Service that implements the message exchanges associated with NotificationProducer can be a Publisher (i.e. it creates the NotificationMessages itself) or it can be a NotificationBroker, distributing NotificationMessages that were produced by a separate Publisher entity.
- It is the factory for Subscription resources.

NotificationConsumer:

- A NotificationConsumer is a Web Service that receives NotificationMessages from a NotificationProducer.
- A NotificationConsumer may implement the generic Notify message exchange, or it may be able to process one or more domain-specific NotificationMessage types.

Subscription:

- A Subscription is a WS-Resource, following the implied resource pattern defined in [State Paper]. A Subscription represents the relationship between a NotificationConsumer, NotificationProducer, Topic and various other optional filter expressions, policies and context information.
- A Subscription resource is created when a Subscriber sends the Subscribe request message to a NotificationProducer.
- Subscription resources are manipulated by messages sent to the SubscriptionManager Web service associated with the Subscription resource, using the implied resource pattern.

SubscriptionManager

- A SubscriptionManager is a Web service that implements message exchanges associated with the SubscriptionManager interface.
- A SubscriptionManager provides services that allow a service requestor to query and manipulate Subscription resources that it manages. A SubscriptionManager is a Web service that participates in the implied resource pattern.
- A SubscriptionManager is subordinate to the NotificationProducer, and MAY be implemented by the NotificationProducer service provider. However WS-Notification permits it to be implemented by a separate service provider, should an implementer so desire.

Subscriber:

- A Subscriber is an entity (often a Web service) that acts as a service requestor, sending the subscribe request message to a NotificationProducer.

- Note that a Subscriber may be a different entity than the NotificationConsumer that actually receives the NotificationMessages.

Topic:

- A Topic is the concept used to categorize Notifications and their related NotificationMessage schemas.
- Topics are used as part of the matching process that determines which (if any) subscribing NotificationConsumers should receive a NotificationMessage.
- Every NotificationMessage instance generated by a Publisher is associated with a Topic. The relation between Situation and Topic is not specified by WS-Notification but MAY be specified by the designer of the TopicSpace.
- A synonym in some other publish/subscribe models is *subject*.

Topic Space:

- A forest of Topic Trees grouped together into the same namespace for administrative purposes.

Topic Tree:

- A hierarchical grouping of Topics.

NotificationBroker:

- A NotificationBroker is an intermediary Web service that decouples NotificationConsumers from Publishers.
- It implements both the NotificationProducer and NotificationConsumer interfaces.
- Because a NotificationBroker is an intermediary, it provides additional capabilities to the basic NotificationProducer interface:
    - It can relieve a Publisher from having to implement message exchanges associated with NotificationProducer; the NotificationBroker takes on the duties of a SubscriptionManager (managing subscriptions) and NotificationProducer (distributing NotificationMessages) on behalf of the Publisher.
    - It can reduce the number of inter-service connections and references, if there are many Publishers and many NotificationConsumers
    - It can act as a *finder service*. Potential Publishers and Subscribers can in effect find each other by utilizing a common NotificationBroker.
    - It can provide anonymous Notification, so that the Publishers and NotificationConsumers need not be aware of each others identity.
- An implementation of a NotificationBroker may provide additional added-value function that is beyond the scope of this specification, for example logging NotificationMessages, or transforming Topics and/or NotificationMessage content. Additional function provided by a NotificationBroker can apply to all Publishers that utilize it.

Publisher:

- A Publisher is an entity that creates NotificationMessages, based upon Situation(s) that it is capable of detecting and translating into NotificationMessage artifacts. It does not need to be a Web service.

- A Publisher MAY be a Web service that implements the message exchanges associated with the NotificationProducer interface, in which case it also distributes the NotificationMessages to the relevant NotificationConsumers.
- If a Publisher does not implement the message exchanges associated with NotificationProducer, then it is not required to support the Subscribe request message and does not have to maintain knowledge of the NotificationConsumers that are subscribed to it; a NotificationBroker takes care of this on its behalf.

PublisherRegistration:

- A PublisherRegistration is a resource, following the implied resource pattern identified in WS-Resource. A PublisherRegistration represents the relationship between a Publisher and a NotificationBroker, in particular which topic(s) the publisher is permitted to publish to.
- A PublisherRegistration resource is created when a Publisher sends the RegisterPublisher request message to a NotificationBroker.
- PublisherRegistration resources are manipulated by messages sent to a PublisherRegistrationManager Web service.

PublisherRegistrationManager:

- A PublisherRegistrationManager is a Web service that implements message exchanges associated with the PublisherRegistrationManager interface.
- A PublisherRegistrationManager provides services that allow a service requestor to query and manipulate PublisherRegistration resources that it manages. A PublisherRegistrationManager is a Web service that participates in the implied resource pattern.
- A PublisherRegistrationManager is subordinate to the NotificationBroker, and MAY be implemented by the NotificationBroker service provider. However WS-Notification permits it to be implemented by a separate service provider, should an implementer so desire.

Demand-Based Publisher:

- Some Publishers may be interested in knowing whether they have any Subscribers or not, since producing a NotificationMessage may be a costly process. Such Publishers can register with the NotificationBroker as a *Demand-Based Publisher*.
- Demand-Based Publishers implement message exchanges associated with the NotificationProducer interface.
- The NotificationBroker subscribes to the Demand-Based Publisher. When the NotificationBroker knows that there are no Subscribers for the NotificationMessages from a Demand-Based Publisher it pauses its Subscription with that Publisher; when it knows that there are some Subscribers, it resumes the Subscription.
- This way the Demand-Based Publisher doesn’t need to produce messages when there are no Subscribers, however a Demand-Based Publisher is only required to support a single Subscriber on any given Topic, and so can delegate the management of multiple Subscribers, delivery to multiple NotificationConsumers and other related issues (for example security) to the NotificationBroker.

The following terms, defined in the WS-Resource Framework family of specifications [State Paper], are used in WS-Notification. Their definitions are included here for ease of reference.

WS-Resource:

- A Web service having an association with a stateful resource, where the stateful resource is defined by a resource properties document type and the association is expressed by annotating a WSDL portType with the type definition of the resource properties document.

Implied Resource Pattern:

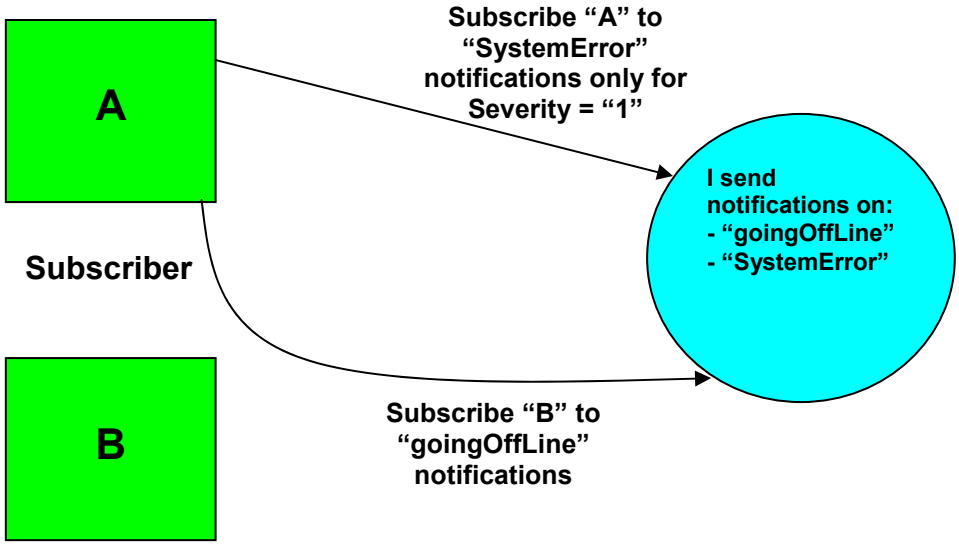
- The way WS-Addressing MUST be used to designate the stateful resource component of the WS-Resource to be used in the execution of message exchanges.
- An EndpointReference that follows the implied resource pattern may include a ReferenceProperties child element that identifies the stateful resource component of the WS-Resource to be used in the execution of all message exchanges performed using this EndpointReference.
- A message that follows the implied resource pattern MUST be sent to a Web service referred to by an EndpointReference that follows the implied resource pattern, and MUST conform to the WS-Addressing requirements on that message including adding the ReferenceProperties information, if present, from that EndpointReference to the message.
- A Web service that follows the implied resource pattern MAY use the ReferenceProperties information from a message that follows the implied resource pattern in order to identify the stateful resource to be used in the execution requested by that message.

WS-Resource Qualified Endpoint Reference:

- An Endpoint Reference used to refer to a WS-Resource composed of a Web service and a stateful resource.
- A stateful resource identifier MAY be contained within the ReferenceProperties element of the Endpoint Reference.
- The address of the Web service associated with the WS-Resource MUST be contained in the Address element of the Endpoint Reference.

4 Example

Consider the following depiction:

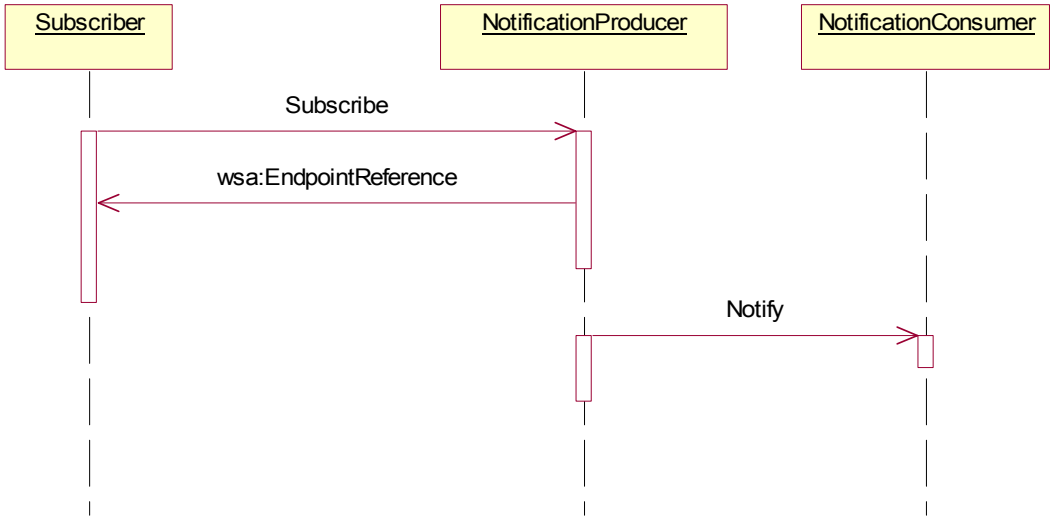


The figure above shows a Web service (a simple Web application server manager, for example) that is a NotificationProducer producing two kinds of notification: "goingOffLine" and "systemErrorDetected".

The service requestor (A), playing the role of subscriber, registers interest in the receiving notifications of a given kind, by sending the Subscribe request message to the NotificationProducer Web service, as depicted by arrows on the above figure.

When certain Situations happen within the operating environment of the NotificationProducer service, one or more NotificationMessages are generated on one or more of its Topics. NotificationMessages are sent to the interested parties (NotificationConsumers) to notify them of the Situation.

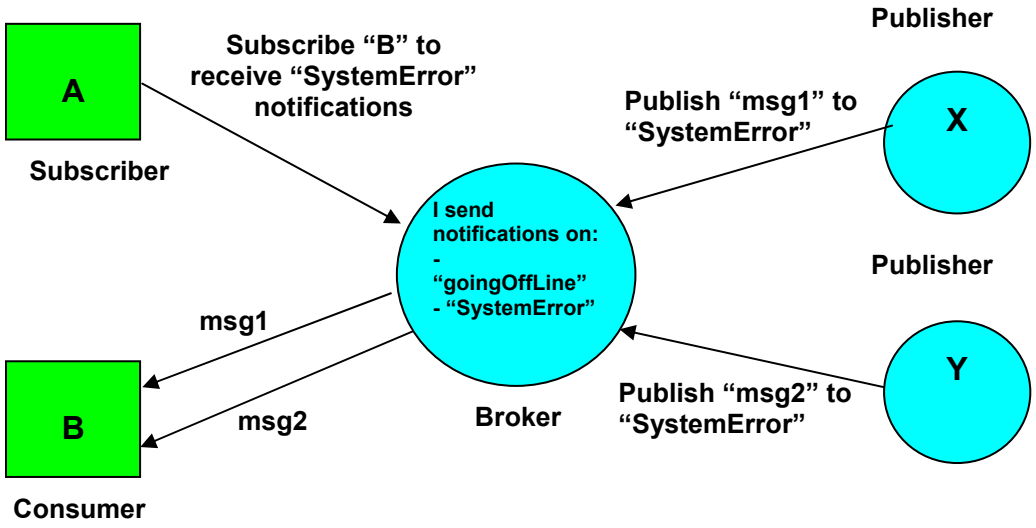
The following diagram illustrates a possible sequence of operations:



The Subscriber sends a Subscribe request message to the NotificationProducer, indicating the address of the NotificationConsumer, the kinds of notification for the Subscription, and other related Subscription information. In response to this message, the NotificationProducer creates a Subscription resource and returns an EndpointReference [WS-Addressing] to this Subscription WS-Resource.

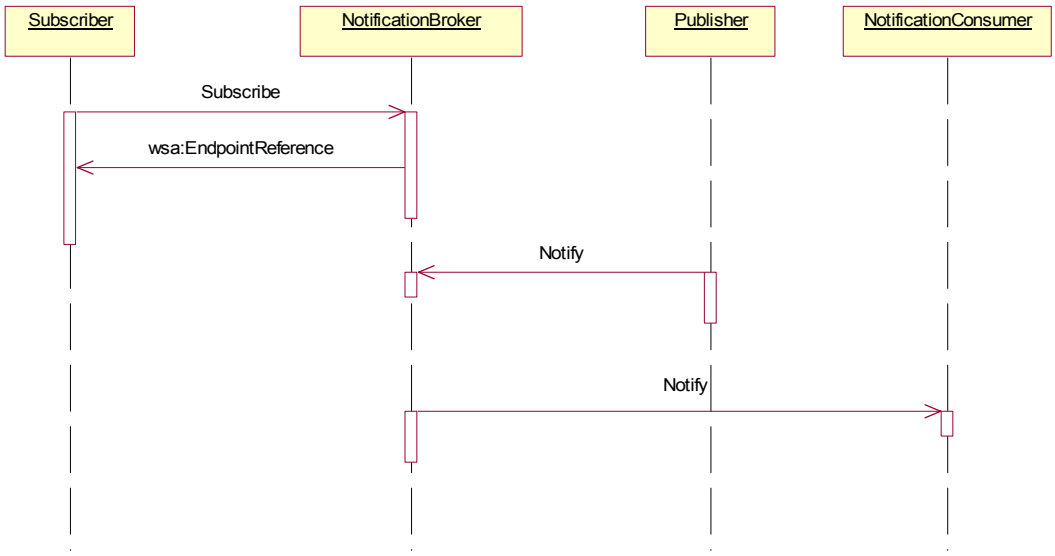
At some later time, the NotificationProducer issues a Notification that matches the Subscription. The NotificationProducer uses the Notify message to deliver this to the NotificationConsumer, or, if indicated on the Subscription, sends the NotificationMessage as an application-defined message, without using the generic Notify message exchange.

Some notification patterns involve an intermediary, or broker. This arrangement is depicted in the following figure:



In the above figure, Subscriber "A" subscribes the NotificationConsumer named "B" to receive "SystemError" notifications provided by the NotificationBroker. Other entities (for example, publisher X) are allowed to publish SystemError notifications (in this example publisher X has published msg1). Some time later, another entity (publisher Y) publishes msg2 to the NotificationBroker as a "SystemError" notification. The NotificationBroker delivers both msg1 and msg2 to "B" as they both match the subscription that "A" established for "B".

The interactions between the Subscriber, NotificationBroker and one of the Publishers are shown in the following interaction diagram:



In the brokered case, the sequence of message exchanges between Subscriber and the NotificationBroker is the same as the sequence of message exchanges between Subscriber and NotificationProducer in the non-brokered case.

Instead of interacting directly with the ultimate NotificationConsumers, Publishers interact with the NotificationBroker using a sequence of message exchanges supported by the NotificationBroker. The Publisher publishes a NotificationMessage to the NotificationBroker, using the Notify message. At some point subsequent to this publication, the NotificationBroker delivers the NotificationMessages to any NotificationConsumer identified by Subscriptions which match the publication.

Note: the Notify message exchange can be used for two purposes: to publish a NotificationMessage to the NotificationBroker and to deliver a NotificationMessage to a NotificationConsumer. This allows chaining of NotificationBrokers and anonymous intermediation of NoificationBrokers between NotificationProducers (publishers) and NotificationConsumers.

5 Security Considerations

This section deals with the security aspects of WS-Notification. It deals with (a) securing the standard message exchanges defined in this specification, and (b) authorization and denial of service considerations.

5.1 Securing the Message Exchanges

As defined in WS-Notification, Subscription messages are exchanged between Subscribers and NotificationBrokers; NotificationMessages are exchanged between Publishers and NotificationBrokers, and NotificationBrokers and NotificationConsumers. It is important to ensure NotificationMessages cannot be tampered with during delivery to NotificationConsumers and to ensure confidentiality

of sensitive NotificationMessages. Therefore, when these messages are exchanged between these services, it is strongly RECOMMENDED that the communication between services be secured using the mechanisms described in WS-Security [WS-Security]. In order to properly secure messages, the body and all relevant headers need to be included in the digital signature so as to prove the integrity of the message. In addition the reference properties within an EndpointReference, passed in a Subscribe or in other request messages, MAY be encrypted to ensure their privacy. In the event that a requestor communicates frequently with a NotificationConsumer Web service, for example, sending the Notify message, it is RECOMMENDED that a security context be established using the mechanisms described in WS-Trust [WS-Trust] and WS-SecureConversation [WS-SecureConversation]allowing for potentially more efficient means of authentication.

It is common for communication between requestors and Web service associated with a resource through the implied resource pattern to exchange multiple messages. As a result, the usage profile is such that it is susceptible to key attacks. For this reason it is strongly RECOMMENDED that the keys used to secure the channel be changed frequently. This "re-keying" can be effected a number of ways. The following list outlines four common techniques:

- Attaching a nonce to each message and using it in a derived key function with the shared secret
- Using a derived key sequence and switch "generations"
- Closing and re-establishing a security context
- Exchanging new secrets between the parties

It should be noted that the mechanisms listed above are independent of the security context token (SCT) and secret returned when subscribed the first time. That is, the keys used to secure the channel during notifications may be independent of the key used to prove the right to subscribe with a NotificationProducer.

The security context MAY be re-established using the mechanisms described in WS-Trust and WS-SecureConversation. Similarly, secrets can be exchanged using the mechanisms described in WS-Trust. Note, however, that the current shared secret SHOULD NOT be used to encrypt the new shared secret. Derived keys, the preferred solution from this list, can be specified using the mechanisms described in WS-SecureConversation.

The following list summarizes common classes of attacks that apply to this protocol and identifies the mechanism to prevent/mitigate the attacks:

- **Message alteration** – Alteration is prevented by including signatures of the message information using WS-Security.
- **Message disclosure** – Confidentiality is preserved by encrypting sensitive data using WS-Security.
- **Key integrity** – Key integrity is maintained by using the strongest algorithms possible (by comparing secured policies – see WS-Policy [WS-Policy] and WS-SecurityPolicy).
- **Authentication** – Authentication is established using the mechanisms described in WS-Security and WS-Trust. Each message is authenticated using the mechanisms described in WS-Security.

- **Accountability** – Accountability is a function of the type of and string of the key and algorithms being used. In many cases, a strong symmetric key provides sufficient accountability. However, in some environments, strong PKI signatures are required.
- **Availability** – Many services are subject to a variety of availability attacks. Replay is a common attack and it is RECOMMENDED that this be addressed as described in the next bullet. Other attacks, such as network-level denial of service attacks are harder to avoid and are outside the scope of this specification. That said, care should be taken to ensure that minimal processing be performed prior to any authenticating sequences.
- **Replay** – Messages may be replayed for a variety of reasons. To detect and eliminate this attack, mechanisms should be used to identify replayed messages such as the timestamp/nonce outlined in WS-Security and the sequences outlined in WS-ReliableMessaging.

5.2 Securing Subscriptions and Notifications

Given WS-Notification provides mechanisms for publishing, and subscribing to topics, security policies should be established such that

1. only authorized principals can subscribe for Notifications
2. only authorized principals can modify or delete Subscriptions
3. only authorized principals can modify or delete PublisherRegistrations (see [WS-BrokeredNotification]).
4. only authorized principals can publish NotificationMessages
5. only authorized principals can create new child Topics (where the TopicSpace definition permits this) (see [WS-Topics]).

It is recommended that the authorization policies be specified at the granularity of the Topic. It should be noted that even though Subscriptions may be done by authorized principals, the NotificationMessages may be delivered to NotificationConsumers whose identity may be different from the Subscriber. Message protection policies as outlined in the previous section can be used to ensure that sensitive NotificationMessages are not delivered to malicious endpoints. For example, a key may need to be specified or generated during the process of Subscription, so that the NotificationMessages can be encrypted using the key to ensure confidentiality of the messages. The mechanism by which the key is specified is governed by the Subscription policy.

Given that WS-Notification uses WS-ResourceProperties, the security considerations outlined in WS-ResourceProperties need to be taken into account. Authorization policies for those Resource Properties should be put in place so that the implications of providing the state information (through GetResourceProperty request messages) or through notification of state change and modification of the resource properties (through SetResourceProperty request messages), are taken into account.

This specification provides a mechanism by which Subscribers can specify a subscription policy. Such a policy may contain security policy about protecting the message exchanges resulting from the Subscription. Security policy for Subscription message exchanges needs to take this into consideration so that the Subscription policies are protected. Also, given this policy may be contained in the resource

properties of the subscription maintained by the SubscriptionManager, the resource properties must be appropriately secured.

WS-Notification uses WS-ResourceLifetime to manage the lifetime of subscriptions and PublisherRegistrations. Authorization policies should be defined so that the implications, of destroying a resource either through explicit requests or by setting TerminationTime are considered. Therefore, the security considerations outlined in WS-ResourceLifetime need to be taken into account in order to secure Subscriptions and PublisherRegistrations appropriately.

6 Acknowledgements

This specification has been developed as a result of joint work with many individuals and teams. The authors wish to acknowledge the contributions from many people, including:

Tim Banks (IBM), Nick Butler (IBM), Glen Daniels (Sonic Software), Doug Davis (IBM), John Dinger (IBM), Don Ferguson (IBM), Jeff Frey (IBM), David Hull (Tibco), Andreas Koeppel (SAP), Heather Kreger (IBM), Kevin Liu (SAP), Tom Maguire (IBM), Susan Malaika (IBM), David Martin (IBM), Bryan Murray (HP), Martin Nally (IBM), Jeff Nick (IBM), Claus von Riegen (SAP), Rick Rineholt (IBM), John Rofrano (IBM), Eugène Sindambiwe (SAP), Jay Unger (IBM), Mark Weitzel (IBM), Dan Wolfson (IBM).

7 References

[WS-Addressing]

http://www.ibm.com/developerworks/webservices/library/ws-add/

[State Paper]

http://www-106.ibm.com/developerworks/webservices/library/ws-resource/ws-modelingresources.pdf

[WS-BaseNotification]

ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-BaseN.pdf

[WS-BrokeredNotification]

ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-BrokeredN.pdf

[WS-Policy]

http://www-106.ibm.com/developerworks/library/ws-polfram/

[WS-ReliableMessaging]

http://www-106.ibm.com/developerworks/webservices/library/ws-rm/

[WS-ResourceLifetime]

http://www-106.ibm.com/developerworks/webservices/library/ws-resource/ws-resourcelifetime.pdf

[WS-ResourceProperties]

http://www-106.ibm.com/developerworks/webservices/library/ws-resource/ws-resourceproperties.pdf

[WS-SecureConversation]

http://www-106.ibm.com/developerworks/library/ws-secon/

[WS-Security]

http://www.oasis-open.org/committees/download.php/5531/oasis-200401-wss-soap-message-security-1.0.pdf

[WS-Topics]

ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-Topics.pdf

[WS-Trust]

http://www-106.ibm.com/developerworks/library/ws-trust/