J. Software Engineering & Applications, **2009, 2: 295-300**
doi:10.4236/jsea.2009.24038 Published Online November 2009 (http://www.SciRP.org/journal/jsea)

Research of Publish and Subscribe Model Based on WS-Notification

Huilian FAN[1], Guangpu ZEN[1], Xianli LI[2]

[1]School of Mathematics and Computer Science, Yangtze Normal University, Fuling, China; [2]Chongqing college of Electronic Engineering, Chongqing, China.
Email: fhlmx@163.com

Received August 14th, 2009; revised September 14th, 2009; accepted September 27th 2009.

ABSTRACT

WS-Notification bundle of standards, WS-BaseNotification, WS-Topics, and WS-BrokeredNotification, can be used as a general purpose publish/subscribe interface for Service Oriented Architectures. We provide an overview of the WS-Notification specification and describe a modified publish and subscribe model based on WS-Notification. The model is an adaptive policy-driven notification framework that can help enterprises to meet the flexibility and responsiveness requirements of the enterprise. With the modified publish/subscribe model, information consumers can dynamically and declaratively create and configure entities on their behalves to manage their distribution requirements.

Keywords: *WS-Notification, Publish/Subscribe, Notification Broker Service, Notification Consumer Proxy Service*

1. Introduction

In a Service Oriented Architecture (SOA), there is often a need to monitor situations. This occurs to a computer management perspective or a much more broad scope of keeping up to date on real world events such as weather, financial transactions, etc. To monitor these events, a client can poll status changes or subscribe for status changes. In polling case, the client is configured to actively ask the resource for its latest state. The more frequently the client polls state, the more likely the client has an accurate resource representation. However, frequent polling requires both of bandwidth and resources to handle the connection. Thus polling is useful when monitoring timeliness is not an issue or network and hardware resources are abundant. But in the SOAP world, polling is less common as a client typically receives requests from the producer of events. With this peer to peer style, a publish/subscribe system can be created. In this system, the client requests that notifications be sent when they occur. This reduces the latency between the event occurring and the client processing.

WS-Notification has been standardized by OASIS and it is a standard that can solve this business problem of event distribution in heterogeneous complex event processing systems. It specifies an interface for consumer to subscribe, filter notifications, and manage subscriptions. It is also for publishers to send notifications. Further, it describes a notification broker to allow for scaling of the system [1].

2. WS-Notification

Associated with the WS-Resource Framework, IBM, Sonic, and other companies introduced a family of related specifications called WS-Notification. The basic idea behind WS-Notification is to standardize the way that a Web service can notify interested parties (other Web services) that something of interest has happened. It is not meant to replace all messaging infrastructure such as low latency buses, industry standards, or existing infrastructure like JMS. However, WS-Notification systems should be able to integrate with these systems through simple adapters.

Obviously, the key value to WS-Notification is its ability as a standard to allow for greater interoperability with a greater number of vendors and, thus, a lower cost for implementation. The key features of WS- Notifications allow for it to be used as well in general purposes publish/subscribe situations. It defines a unified message format to achieve interoperability between kinds of systems, procedures and components in different platforms and systems, and it defines a set of a standard Web services approach using a topic-based publish/subscribe notification pattern .

WS-Notification family is made up of the following three components [2]: WS-Topics, WS-BaseNotification and WS-BrokeredNotification. Figure 1 shows the relationship between them.

Based on the WS-Notification, the publish/subscribe model is needed to handle the real-world information integration scenario by allowing a subscriber to specify on behalf of an information consumer filtering rules and policy constraints, not only to select what types of messages or contents the subscriber wants the consumer to receive, but also to specify transformation, scheduling, distribution, and other constraints to be applied to selected messages before they reach the consumer. The architecture must enable the generation (on behalf of consumer) of a proxy service, or agent, which includes the engines to enforce and manage these constraints at run-time. The proxy service can receive the messages on behalf of the information consumer and apply the specified constraints to the message before delivering it to its consumer [3].

2.1 WS-Topics

The WS-Topics specification [4] defines a mechanism to organize and categorize items of interest for subscription known as "topics." This is achieved by associating each notification with a topic, and means that subscribers can define the specific category of event that they are interested in hearing about. A web service can publish a set of topics used to organize and categorize a set of notification messages that clients can subscribe, and receive a notification whenever the topic changes.

WS-Topics are very versatile, as they even allow us to create topic trees, where a topic can have a set of child topics. By subscribing to a topic, a client automatically receives notifications from all the descendant topics (without manual subscribing to each of them). As part of the publication of a notification-message, the publisher associates it with one or more topics.

WS-Topics also provide a coarse-grained filtering mechanism which allows large sets of uninteresting notifications to be excluded quickly. For example, in a sport results scenario a subscriber can indicate that he or she is only interested in receiving notifications about football, which excludes any events about baseball or hockey. More fine-grained control of filtering can be achieved using other filtering mechanisms, such as the message content filter defined in WS-BaseNotification. For example, by selecting only those football games in which the home team beat the away team. In many situations, the topic does not actually appear in the body of the notification message itself since the classification of the notification is made at a higher level than the generation of the notification content.

In order to avoid naming collisions, and to facilitate interoperation between independently developed Notification Producers and Subscribers, every WS-Notification Topic is assigned to an XML Namespace. The set of Topics associated with a given XML Namespace is termed a Topic Namespace.

2.2 WS-BaseNotification

The WS-BaseNotification specification defines the standard Web services interfaces for Notification Producers and Notification Consumers. It includes standard message exchanges to be implemented by service providers who wish to act in these roles, along with operational requirements expected by them. Notification producers have to expose a subscribing operation that notification consumers can use to request a subscription. Consumers, in turn, have to expose a notify operation that producers can use to deliver the notification [5,6]. Figure 2, "A typical WS-Notification interaction" shows that the five primary entities how to work together to pass data through the WS-BaseNotification. Initially, the subscriber is responsible for setting up a subscription between the NotificationProducer Web service and a NotificationConsumer Web service. This subscription is managed by the SubscriptionManager Web service working on behalf of the producer. Subsequently, when a Situation is observed by the Publisher, the Publisher creates a notification message and passes it to the NotificationProducer. It is the responsibility of the producer to establish whether the notification message matches the registered subscription or not, and, if so, send the notification message to the consumer.

2.3 WS-BrokeredNotification

Even in the simplest publish/subscribe environment, the amount of connections and boot strapping information can grow very quickly. If there are only *2* publishers and *2* consumers, and each consumer wants to be notified from each publisher, *4* connections need to be set up.

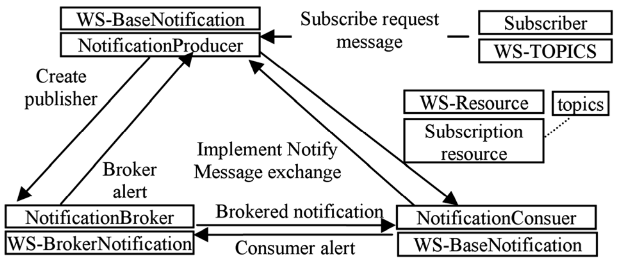

Figure 1. Relationship between WS-Topics, WS- aseNotification and WS-BrokeredNotification

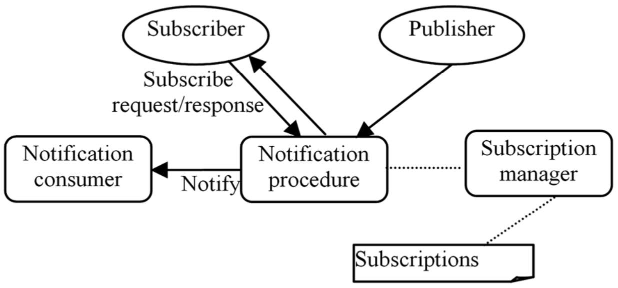

Figure 2. A typical WS-Notification interaction

Add one more consumer and you now have *6* connections. The number of connections starts to grow very quickly as more distributors and consumers are added; the required number of connections for *m* publishers and *n* consumers are $m \times n$ connections. To simplify this topology, WS-Notification standardizes a notification broker which acts as an intermediary between publishers and consumers. Here, publishers and consumers are decoupled from each other and substitute only required boot strap information to the broker. Therefore, in the scenario of *m* publishers and *n* consumers, the required number of connections is $m + n$.

The WS-BrokeredNotification specification extends the definitions made in the WS-BaseNotification specification to define the concept of NotificationBroker, which is an intermediary service to those producers and consumers can connect in order to pass notifications. Critically, the NotificationBroker is capable of accepting subscription requests from consumers, as well as receiving notification messages from producers. The broker is then responsible for matching the notifications with the subscriptions and sending them to the consumer. In this way, the broker takes on more painstaking functions of the producer, freeing developers of producer applications to concentrate on the business task of observing situations and generating the appropriate notifications without worrying about the challenge but mechanical task of managing subscriptions and matching them to notifications. Advantages of the brokered notification pattern are as follows:

- Relieves the publisher of having to implement message exchanges associated with the notification producer. For example, managing subscriptions (SubscriptionManager) and distributing notifications (NotificationProducer).
- Avoids the need for synchronous communications between the producer and the consumer.
- Can reduce the number of inter-service connections and references.
- Acts as a finder service. For example, if a new publisher is added that publishes notification *x*, a consumer does not have to issue a new subscription if it has already subscribed to the broker with *x*.
- Provides anonymous notification, which means that publishers and consumers do not need to be aware of each other's identity.

In many scenarios, the NotificationBroker service is implemented by a middleware provider, ensuring that the brokering facilities are written to enterprise quality expectations and often provide additional value-add services over and above the basic definition of the service, for example, logging, transformation, or quality of service enhancements above those required by the specification [7]. As shown in Figure 3, "A typical brokered WS-Notification interaction", the producer must register

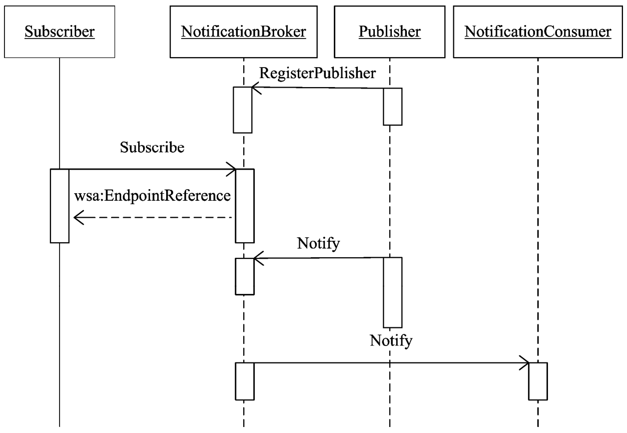

Figure 3. A typical brokered WS-Notification interaction

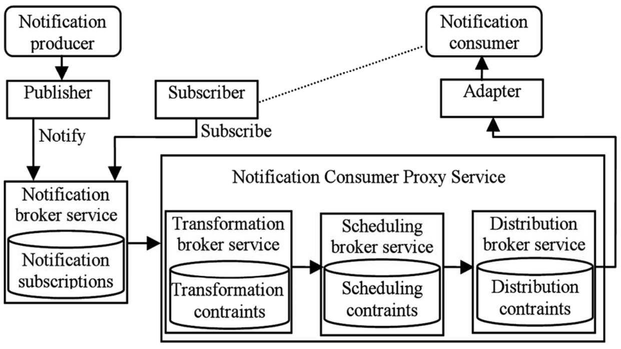
Notification producer
Notification consumer
Publisher
Subscriber
Adapter
Notify
Subscribe
Notification Consumer Proxy Service
Notification broker service
Notification subscriptions
Transformation broker service
Transformation contraints
Scheduling broker service
Scheduling contraints
Distribution broker service
Distribution contraints

Figure 4. The Publish/Subscribe architectural model

with the broker and publish its topics there. The subscriber must also subscribe through the broker, not directly with the producer. Finally, when a notification is produced, it is delivered to the consumer through the broker.

3. Publish/Subscribe Model Based on WS-Notification

The modified publish/subscribe model is as shown in Figure 4. It extends the basic publish and subscribe pattern by extending the subscription capabilities to include the specification of transformation, distribution, and scheduling constraints as part of publish and subscribe subscription [8].

Additionally, this architecture enables non-pub/sub-enabled systems (that is, information consumers which are not able to consume notification messages of the pub/sub system) to participate in the pub/sub pattern by allowing the model to dynamically create a proxy service to receive pub/sub notifications on behalf of the consumer. This is the Notification Consumer Proxy Service (NCPS) shown in Figure 4, which also manages the distribution of notifications to the consumer based on the transformation, distribution, and scheduling constraints specified by the consumer upon subscribing.

As shown, the model highlights the following components:

Notification producer: Contains information of interest for consumer. Good examples of information producers are systems that manage business information for an enterprise and include master data stores for customer, product, order information, and so on, in addition to enterprise operational data stores.

Notification consumer: Depends on and must consume information from an information producer. For example, many enterprise business applications like order fulfillment systems depend on data from the business information sources.

Subscriber: Requests creation of a subscription. It sends a subscribe request message to a notification broker (pub/sub broker). The subscribe request message identifies a notification consumer. A subscription is an entity that represents the relationship between an information consumer and an information producer. It records the fact that the consumer is interested in some or all of the notifications that the producer can provide. It can contain filter expressions, and may be long-running or have a limited lifetime.

Publisher: Creates notification message instances. A publisher receives information from entities in the information producer that monitor and detect a situation. A situation is an occurrence that is noted by one party and is of interest to other parties. A notification is a one-way message that conveys information about a situation to other services.

Notification broker service: Performs a notification broker function between notification consumers and notification producers, and it is responsible for sending notifications to the appropriate consumers. It also acts as a subscription manager and manages requests to query, delete, or renew subscriptions.

Notification Consumer Proxy Service (NCPS): Receives notifications from the notification broker on behalf of the information consumer. Typically, the consumer is not able to receive notification messages, hence the need for this service to act on its behalf, collect the notifications, perform some business logic (if the scenario calls for it), enforce the transformation, scheduling, and distribution constraints for the consumer, and then send the results to the consumer.

Adapter: An entity that enables the interaction with an information consumer.

To demonstrate the publish/subscribe model, event distribution was applied to solve a situation of teacher-student interaction. The situation occurs when students have questions to ask teacher. WS-Notification was used in the scenario for the notifications from the student to the broker and from the broker to the teacher. This was done through a number of different steps:

1) The teachers registered their interest with the broker.

2) The students were either configured to send notifications to the broker or configured to register themselves as a publisher to the broker.

3) If students registered themselves as a publisher to the broker, the broker would ask for notifications from the students.

4) The students would send question notifications to the broker.

5) The broker would forward those question notifications to interested teacher.

In the situation, teacher is not only subscriber but also information consumer, and student is information producer. From user perspective, there are two interfaces: the student and the teacher. The student interface is responsible for allowing students to ask questions. The teacher interface allows teachers to monitor notifications as they arrive. The interfaces of the publish/subscribe model among the teacher, notificationbroker, NCPS and one of the students are following:

1) Subscribe: A subscriber, on behalf of the notification consumer, sends an XML subscription request message to notification broker service. This subscription message specifies transformation, distribution, and scheduling constraints for the information consumer, as well as the basic subscription constraints on information content from the producer that the consumer is interested in.

For a Web service to act as a NotificationProducer it must support the Subscribe message exchange – that is to say the WSDL for the Web service must be included in its portType definition an operation that contains the subscribe request and response messages defined by the WS-Notification specification. By implementing the Subscribe exchange, the producer service is required to send a notification to each notification consumer with a subscription registered whenever the producer has a message be sent and the filter conditions expressed in the subscription are satisfying. In the example, the teacher subscription request is a message sent from the teacher to the broker to request notifications be sent. It contains the type id for a question that the teacher wishes to answer it. In this subscription request, the *Teacher* requests for notifications for question type with id *C001*, as shown in the Listing 1.

Two of the elements of the above request message are of particular interesting in the Publish/Subscribe environment:

● The ConsumerReference is a WS-Addressing endpoint reference that identifies the location of the Notification Proxy Service Consumer service to which matching notification messages will be sent by the producer.

● The Filter element, and in particular the TopicExpression filter, is used to determine the specific subset of all available notification messages that should match this new subscription. This is important because the majority of notifications are of interest only for a small number of

Listing 1. SOAP body contents of a subscribe request for notification delivery

```
<wsnt:Subscribe xmlns:wsa="http://www.w3.org/2005/08/addressing"
   xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
         <wsnt:ConsumerReference>
               <wsa:Address>

http://notify.yznu.cn:9080/Teacher/NCProxyServiceEndpoint
               </wsa:Address>
         </wsnt:ConsumerReference>
         <wsnt:Filter>
       <wsnt:TopicExpression                    Dialect="http://docs.oasis-
open.org/wsn/t-1/TopicExpression/Simple"

xmlns:typeID="http://www.ibm.com/xmlns/wsn/question/typeID">
                         typeID:C001
               </wsnt:TopicExpression>
       </wsnt:Filter>
<wsnt:InitialTerminationTime>P0Y0M0DT1H0M0S</wsnt:InitialTer
minationTime>
</wsnt:Subscribe>
```

consumers, so we improve the efficiency of the system by avoiding sending unwanted notifications.

2) Notify: The publisher entity creates a notification message when it receives information from entities in the notification producer that monitor and detect a situation, such as put new questions that are of interest for consumers. The publisher sends the notification message to notification broker so it can be distributed to the appropriate consumers that have subscribed to that message.

3) Notification brokering: The notify message sent by the publisher is routed by the notification broker service to the appropriate notification consumer proxy service. The notification broker matches notification messages to the consumers that are subscribed to these notifications. The information consumer proxy service receives the notification message and performs whatever business logic it needs to do in order to create and aggregate the appropriate response to the notification consumer. For example, the consumer proxy service might need to access additional information from the information producers to get all the needed data associated with the change. As a result of applying its specific business logic, the notification consumer proxy service assembles a message or a set of messages to be sent to the consumer.

A NotificationBroker may be a WS-Resource, and if it is, it must support the required message exchanges defined by the WS-ResourceProperties specification and MUST also support message exchanges and may support Resource Property elements defined by the following interfaces:

● NotificationProducer
● NotificationConsumer
● RegisterPublisher

The NotificationBroker portType aggregates the three portTypes and it is not the only way to implement a broker. A distributed broker implementation can be achieved by hosting NotificationProducer, NotificationConsumer, or RegisterPublisher portTypes at one or more physical endpoints.

4) Apply constraints:

Transformation constraints: The notification message is applied against the transformation constraints to determine what transformation module or modules to apply to the message.

Scheduling constraints: The scheduling service applies the scheduling policy constraints if any were specified as part of the subscription. These constraints relate to the specified delivery schedule for the information consumer. Notifications that cannot be transmitted due to the conditions specified in the policy are queued by the scheduling service for delivery during the next available window.

Distribution constraints: The distribution service applies the distribution policy constraints if any were specified as part of the subscription. One distribution policy specifies the size limit of a notification message transmitted to the consumer. If a notification message exceeds this size, it will be broken into a sequence of pieces which are smaller than the size limit and transmitted to the consumer individually by the distribution service.

5) Deliver: Once the model satisfies the scheduling and distribution constraints mentioned in the previous step, it sends the message to the information consumer through an adapter service.

4. Conclusions

This paper discusses how to implement a general purpose publish/subscribe interface for a Service Oriented Architecture through the WS-Notification bundle of standards, WS-BaseNotification, WS-Topics, and WS-BrokeredNotification. We describe an adaptive, policy-driven notification architectural model that can support a generalized publish/subscribe interaction pattern. This model is based on the WS-Notification standards, a set of reusable integration services. We introduce the teacher-student interactive scenario to help to demonstrate the WS-Notification features and explain that the publish/subscribe model is the standard of choice for event distribution and processing.

5. Acknowledgements

This work is supported by the Natural Science Project of Chongqing Municipal Education Commission (Project No.KJ091305)

REFERENCES

[1] R. B. Chumbley and J. D. Eisinger, “Leveraging key WS-notification features in your business applications,” April 2009, http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-wsnotificationWAS7/ws-wsnotification

WAS7-pdf.pdf.

[2] Sams Publishing, WS Notification and WS Topics in the WS Resources Framework, July 2006, http://www.devarticles.com/c/a/Web-Services/WS-Notification-and-WS-Topics-in-the-WS-Resources-Framework/.

[3] K. Czajkowski, D. Ferguson, I. Foster etc., WS-Resource Framework, 9th June 2004, http://www.globus.org/wsrf/specs/ws-wsrf.pdf.

[4] W. Vambenepe, S. Graham, and P. Niblett, 9th October 2006, http://wsn-ws_topics-1.3-spec-c, http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-cs.

[5] B. Sotomayor, “The globus toolkit 4 programmer’s tutorial,” August 19th 2007, http://gdp.globus.org/gt4-tutorial/multiplehtml/ch08s02.html.

[6] W. Vambenepe, S. Graham, and P. Niblett, August 9th 2006, http://wsn-ws_base_notification-1.3-spec-cs.pdf, http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-cs-01.pdf.

[7] W. Vambenepe, S. Graham, and P. Niblett, August 9th 2006, http://wsn-ws_brokered_notification-1.3-spec-cs, http://docs.oasis-open.org/wsn-/wsn-ws_brokered_notification-1.3-spec-cs-01.pdf.

[8] A. Bou-Ghannam and M. Roberts, “GPASS: A generalized publish and subscribe solution using WS-Notification standards,” August 2007, http://www.ibm.com/developerworks/websphere/library/techarticles/0708_boughannam/0708_boughan nam.html.