An Efficient Design of Publication and Subscription Model Based on WSN

Xianli Li

Abstract The purpose of the Web Services Notification (WSN) is to define a set of specifications that standardize the way Web services interact using "Notifications" or "Events". They form the foundation for Event Driven Architectures built using Web services. These specifications provide a standardized way for a Web service, or other entity, to disseminate information to a set of other Web services, without having to have prior knowledge of these other Web Services. They can be thought of as defining "Publish/Subscribe for Web services". 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 helps 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 behalf to manage their distribution requirements.

Keywords 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 from 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 for status changes or subscribe for status changes. In polling case, the client is configured to actively ask the resource

X. Li (✉)
Computer Network Information Center, Yangtze Normal University, Chongqing, China
e-mail: lisansanchqi@163.com

B. Xu (ed.), *2012 International Conference on Information Technology and Management Science (ICITMS 2012) Proceedings*, DOI 10.1007/978-3-642-34910-2_52,

for its latest state. The more often the client polls for state, the more likely the client has an accurate resource representation. However, frequent polling requires bandwidth and resources on both sides 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 it.

WS-Notification has been standardized by OASIS and is a standard that solves this business problem of event distribution in heterogeneous complex event processing systems. It specifies an interface for a consumer to subscribe, filter notifications, and manage subscriptions and an interface for publishers to send notifications. Further, it describes a notification broker to allow for scaling of the system (Chumbley and Eisinger 2009).

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 (pub/sub) 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 to notification using a topic-based publish/subscribe notification pattern.

WS-Notification family is made up of the following three components (Sams 2006): 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 content 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 a consumer) of a proxy service,

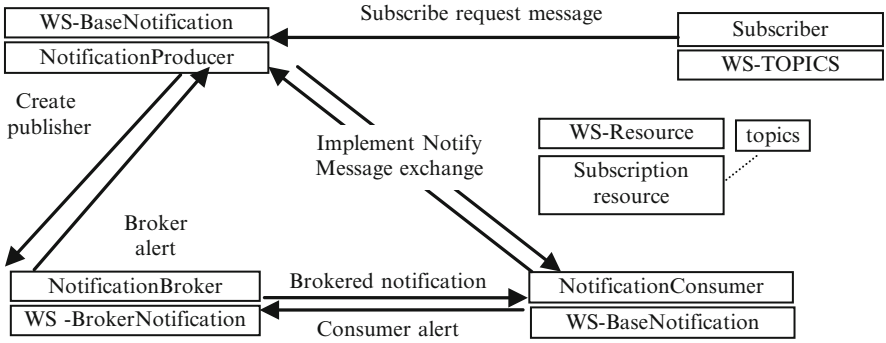

Fig. 1 Relationship between WS-Topics, WS-BaseNotification and WS-BrokeredNotification

or agent, that includes the engines to enforce and manage these constraints at runtime. 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 (Czajkowski et al. 2004).

2.1 WS-Topics

The WS-Topics specification (Vambenepe et al. 2006a) 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 to, 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 having to manually subscribe 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 that 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.

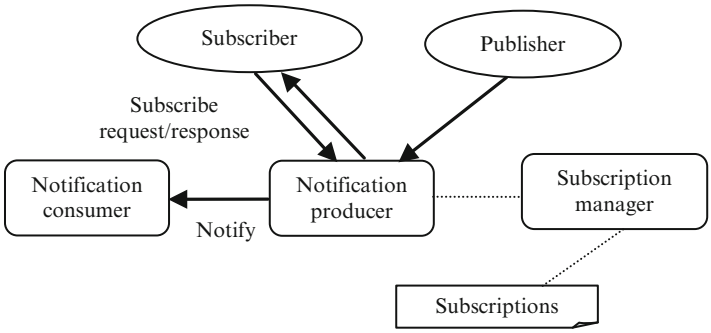

Fig. 2 A typical WS-notification interaction

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 that wish to act in these roles, along with operational requirements expected of them. Notification producers have to expose a subscribe 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 (Sotomayor 2007; Vambenepe et al. 2006b). Figure 2, "A typical WS-Notification interaction" shows how the five primary entities 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 subscription that has been registered, and, if so, to send the notification message to the consumer.

2.3 WS-BrokeredNotification

In even the most simple publish/subscribe environments, the amount of connections and boot strapping information can grow very quickly. If there are only two

publishers and two consumers, and each consumer wants to be notified from each publisher, four connections need to be set up. Add one more consumer and you now have six 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 is m × n connections. To simplify this topology, WS-Notification standardized a notification broker that acts as an intermediary between publishers and consumers. Here, publishers and consumers are decoupled from each other and instead only require boot strap information to the broker. So, 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 a NotificationBroker, which is an intermediary service to which 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 some of the 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 having to worry about the challenging 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 is already subscribed to the broker with x.
- Provides anonymous notification, which means that publishers and consumers need not 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 providing 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 (Vambenepe et al. 2006c). As shown in Fig. 3, "A typical brokered WS-Notification interaction", the producer must register 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.

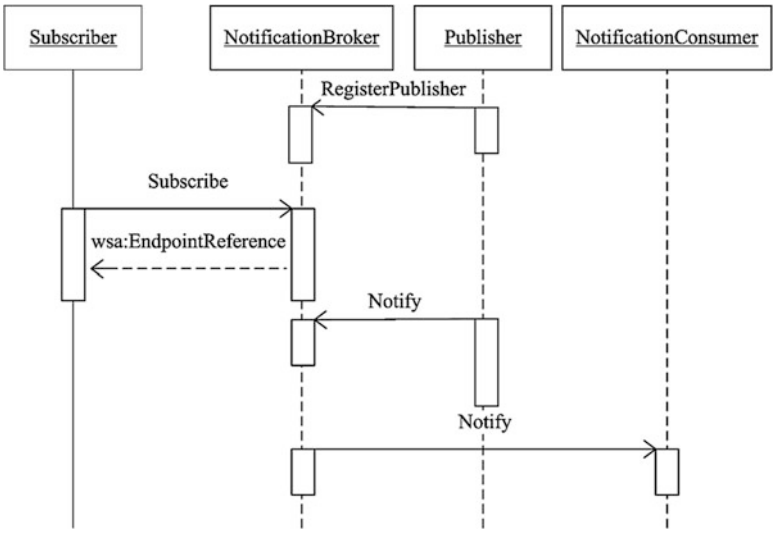

Fig. 3 A typical brokered WS-notification interaction

3 Publish/Subscribe Model Based on WS-Notification

The modified publish/subscribe architectural model is as shown in Fig. 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 the publish and subscribe subscription (Bou-Ghannam and Roberts Matt 2007).

Additionally, this architecture enables non-pub/sub-enabled systems (that is, information consumers that 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 Fig. 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 to a 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

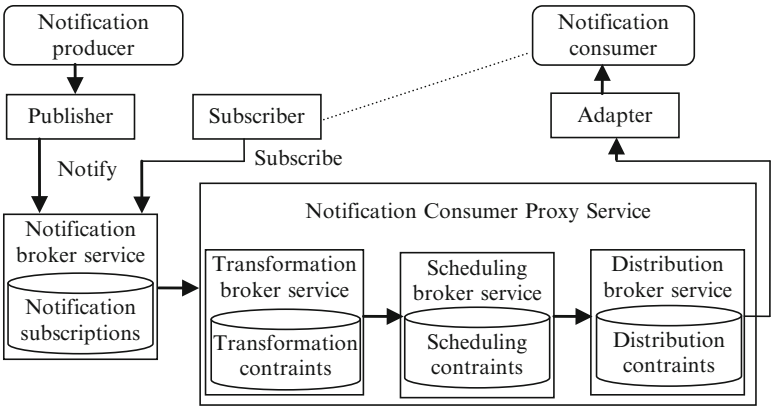

Fig. 4 The publish/subscribe architectural model

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 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.

4 Conclusions

This paper discussed how the WS-Notification bundle of standards, WS-BaseNotification, WS-Topics, and WS-BrokeredNotification, can be used as a general purpose publish/subscribe interface for a Service Oriented Architecture.

We described an adaptive, policy-driven notification architectural model to support a generalized publish/subscribe interaction pattern. This model is based on the WS-Notification standards, a set of reusable integration services. We introduced the teacher-student interactive scenario to help demonstrate the WS-Notification features and explained how the publish/subscribe model is the standard of choice for event distribution and processing.

References

Bou-Ghannam A, Roberts M (2007) GPASS: a generalized publish and subscribe solution using WS-Notification standards. 2007-8. Online available at http://www.ibm.com/developerworks/websphere/library/techarticles/0708_boughannam/0708_boughannam.html. Accessed 27 Aug 2009

Chumbley RB, Eisinger JD (2009) Leveraging key WS-notification features in your business applications, 2009-04. On line available at http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-wsnotificationWAS7/ws-wsnotificationWAS7-pdf.pdf. Accessed 25 Aug 2009

Czajkowski K, Ferguson D, Foster I et al (2004) WS-resource framework. 2004-06-9. Online available at http://www.globus.org/wsrf/specs/ws-wsrf.pdf. Accessed 9 May 2009

Sams Publishing (2006) WS Notification and WS Topics in the WS resources framework, 2006-07. Online available at http://www.devarticles.com/c/a/Web-Services/WS-Notification-and-WS-Topics-in-the-WS-Resources-Framework/. Accessed 9 May 2009

Sotomayor B (2007) The globus toolkit 4 programmer's tutorial, 2007-08-19. Online available at http://gdp.globus.org/gt4-tutorial/multiplehtml/ch08s02.html. Accessed 9 June 2009

Vambenepe W, Graham S, Niblett P (2006a) wsn-ws_topics-1.3-spec-c, 2006-10-09. Online available at http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-cs. Accessed 10 May 2009

Vambenepe W, Graham S, Niblett (2006b) P wsn-ws_base_notification-1.3-spec-cs.pdf, 2006-08-09. Online available at http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-cs-01.pdf. Accessed 9 June 2009

Vambenepe W, Graham S, Niblett P (2006c) wsn-ws_brokered_notification-1.3-spec-cs, 2006-08-09. Online available at http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-cs-01.pdf. Accessed 9 June 2009