2011 International Conference on Economics and Business Information
IPEDR vol.9 (2011) © (2011) IACSIT Press, Bangkok, Thailand

Publish/Subscribe Architecture with Web Services

Thanisa Numnonda and Rattakorn Poonsuph
School of Applied Statistics,
National Institute of Development Administration, Bangkok, Thailand
Emails: thanisak@gmail.com, rattakorn@as.nida.ac.th

Abstract. A publish/subscribe model, one of the current enterprise application integration techniques, is suitable for a notification architecture. A web service is a promising technology to reach the interoperation of heterogeneous environments. By merging them together, a distributed push-based architecture is introduced in this paper. This architecture is aimed to reduce unnecessary network traffic and solve bottleneck problems in the traditional pull-based architecture. By using the same service contract in the publish/subscribe model, a SOAP-based message can be directly sent from a publisher (a service provider) to all relevant subscribers (service clients) via a broker. When subscribers have no need to keep checking for new or updated data from the publisher, service response time and workload on the publisher are reduced. To proof of the concepts, two experiments on comparison between pull-based and push-based architecture are provided. Our experimental results show that the average response time and the workload are significantly minimized in the push-based architecture. Thus, the architecture can serve the future trend of thin service providers and can be applied to several urgent business situations.

Keywords: Publish/Subscribe, Web Service, Push-based, Pull-based

1. Introduction

With enormous development in information technology, most enterprises are required to integrate applications among its heterogeneous systems increasingly. Heterogeneous systems have been developed since past decades using several platforms, computer languages, and different technologies. They existed from not only within the enterprise but also including its business partner systems. A traditional custom integration solution can cause inconsistent and production chaos afterward. Therefore, a modern systematic integration approach is needed to improve efficiency and provide less maintenance in the future. Several approaches have been introduced in order to integrate heterogeneous systems. Among these are Enterprise Application Integration (EAI), Message-Oriented Middleware (MOM), and Web Services (WS).

EAI emerged in the mid-1990s [1], as enterprises tried to integrate by using point-to-point connections between their applications. It was successful in that era since there were only limited applications to integrate. However, the complexity of linkages between applications and difficulty of maintenance integration portions tend to be problems when many more applications are needed to be integrated. Moreover, data transformation and code conversion always form constraints of implementation. Therefore, the middleware concept was introduced to allow applications to pass messages to others easier and more maintainable.

MOM provides asynchronous and loosely-coupled communications. It supports both queue and topic (publish/subscribe) model of messaging. A message queue is a one-to-one communication between sender and receiver. This topology has limitation for active communication among multiple applications in heterogeneous systems. A better model, publish/subscribe (or pub/sub) model [2], can easily solve this problem by extending MOM functionalities to support one-to-many, many-to-one and also many-to-many communications. A pub/sub model normally consists of three basic elements: publisher, subscriber and broker. An application can be either a publisher or a subscriber, or can be both a publisher and a subscriber at the same time. Additional the number of publishers and subscribers can grow and shrink over time.

Publishers can multicast a message of a topic to every interested subscriber who is listening on that topic [3]. Although publishers and subscribers are loosely coupled and do not need to know about each other, they still have to run on the MOM infrastructure to communicate. Therefore, MOM may be considered to be a barrier for interoperability systems.

WS is the most promising technology to reach the interoperation of heterogeneous environments by using standard specifications such as SOAP [4] and WSDL [5] which are based on XML. However, traditional centralized Web service is synchronous and normally used in the way that all service clients send requests to and get responses from a service provider. As a result it cannot well support some situations that service clients must have critical new or updated data immediately from the service provider. For this reason, the service clients are force to keep regularly checking for new or updated data which certainly wastes much network traffic and definitely increases service response time. In addition, workload on a service provider can be very high when it faces many requests from many service clients simultaneously.

This research introduces distributed push-based architecture to combine a Web service technology with a pub/sub model by using the concept of a canonical message. A broker is responsible for defining the canonical message comprised of a name and a service contract of a topic. By using the canonical message, the broker, the publisher, and all subscribers use the same service contract via WSDL. Therefore, when the publisher pushes a SOAP-based message to the broker, the broker can then propagate that to all relevant subscribers. The overview of distributed push-based architecture is shown in Fig. 1. This can make all subscribers to have new or updated data at the moment when the data are available. Subscribers do not need to regularly keep checking for new or updated data. Obviously, unnecessary network traffic and the service response time can be reduced. Furthermore, by using the broker to handle all services, bottleneck problems on the publisher can be eliminated. Therefore, the publishers can be very small and thin in ubiquitous computing such as sensor devices or mobile devices.

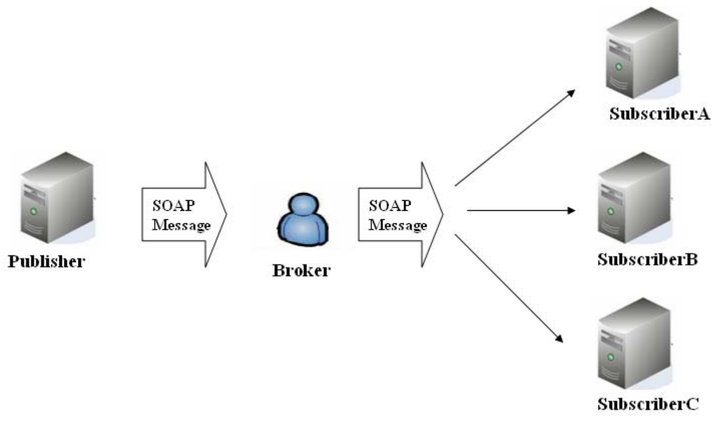

Fig. 1: An overview of distributed push-based architecture

Two competing specifications, Web Services Notification (WS-Notification) [6] and Web Services Eventing (WS-Eventing) [7], are crucial for asynchronous Web service-based event notifications. In [8], Huang and Gannon describe the comparison between them in details. Some works have been done to implement and extend WS-Notification [9-11]. In this paper, however we focus on different aspects from the previous works. Our implementations are designed to compare the response time and workload performance between pull-based and push-based architectures.

The rest of this paper is organized as follows. Section 2 describes a pub/sub model in the way of pull-based and push-based architectures and comparison between them. Section 3 then explains research methodology in details to compare performance of pull-based and push-based architectures. Section 4 provides the proof-of-concept by showing some experimental results. Finally, section 5 discusses conclusion and future works.

2. Publish/Subscribe Model

2.1. Pull-Based Architecture

The vast majority of pub/sub model usages are in the pull-based architecture. When there is a new or updated data message available at a publisher, the publisher sends a notification message to a broker. The broker will then propagate that notification message to all interested subscribers. After that, subscribers have to send request messages to the publisher in order to get a data message. Finally, if the broker is supposed to keep track of successful or failure transmissions, acknowledges should be sent from the subscribers to the broker (see Fig. 2).

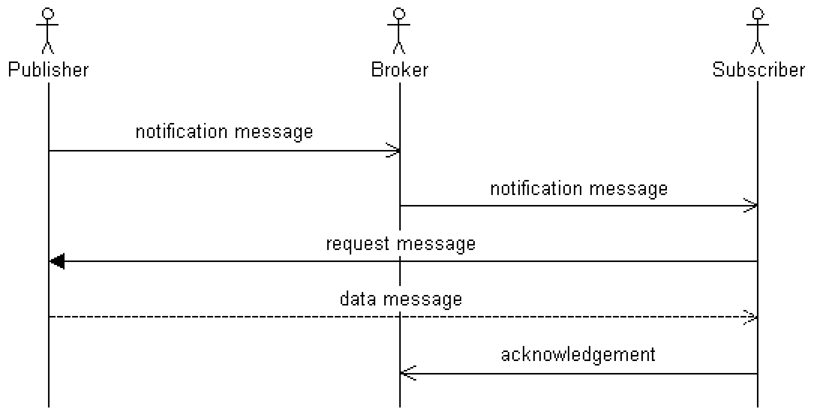

Fig. 2: Sequence Diagram for pull-based Web service architecture

This architecture has two limitations. First, the workload on publishers can be very high when they face many requests from subscribers simultaneously. Second, the response time is also likely to be very high since this architecture requires four one-way communications before being able to get a data message.

2.2. Push-Based Architecture

In the push-based architecture, the transfer of a data message is triggered by some pre-defined events at a publisher. The publisher then pushes the data message to a broker and the broker multicasts that message to all corresponding subscribers. Therefore, a subscriber can have a desired data message without having to request for it. Subscriber acknowledges should be sent to the broker in order to keep track of successful or failure transmissions (see Fig. 3).

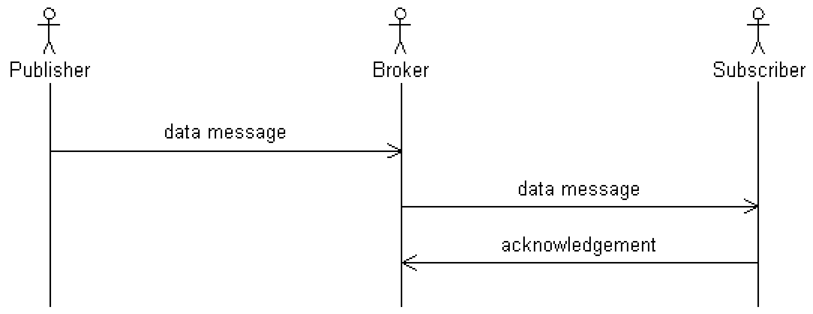

Fig. 3: Sequence Diagram for push-based architecture

This architecture is good for wide-area distributed systems since publishers do not have to process numerous requests from subscribers. For this reason, a publisher can be very thin and small. Moreover, the response time for a subscriber to get a data message is minimized into two one-way communications.

3. Research Methodology and Experimental Results

In this section, we present implementation processes and some experimental results to compare performance between pull-based and push-based architectures. Two systems are set up with the same running environment. There is a broker in each system. Any application interested to be a publisher or a subscriber has to register for a specific topic at the broker. The broker is responsible to keep all important details and invoke methods at the subscribers.

To provide the proof-of-concept, our experiments are set up on five machines with ten simulated subscribers on each machine. Business processes according to the above sequence diagrams are implemented in the Web Services Business Process Execution Language (WS-BPEL) processes (see Fig. 4). The Composite Application (CA) is to bring together our business processes and associated data sources. We generate CA Service Assembly (CASA) (see Fig. 5) and deploy to a GlassFish's BPEL Engine. The engine is to provide a runtime environment for execution.

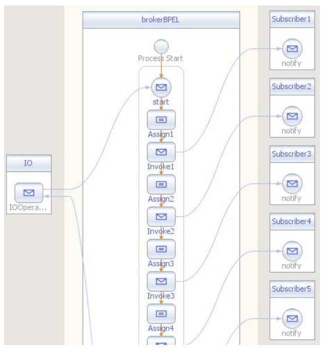

Fig. 4: Design of WS-BPEL

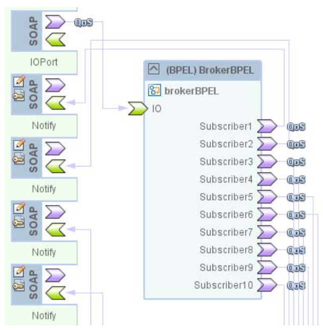

Fig. 5: Design of CASA

In the Fig. 6, the comparison of average response times between pull-based and push-based architectures is shown. The Fig. 6(a) reveals the average response times for up to fifty subscribers. As expected, with the increase of the number of subscribers, the average response time of pull-based grows much higher than that of push-based. In another experiment, when we fix the number of subscribers to fifty nodes, it reveals in Fig. 6(b) that the average response time of pull-based also grows almost linearly when the number of data messages is increased. In addition, the workload on the publisher of the pull-based is much higher than that of the push-based.

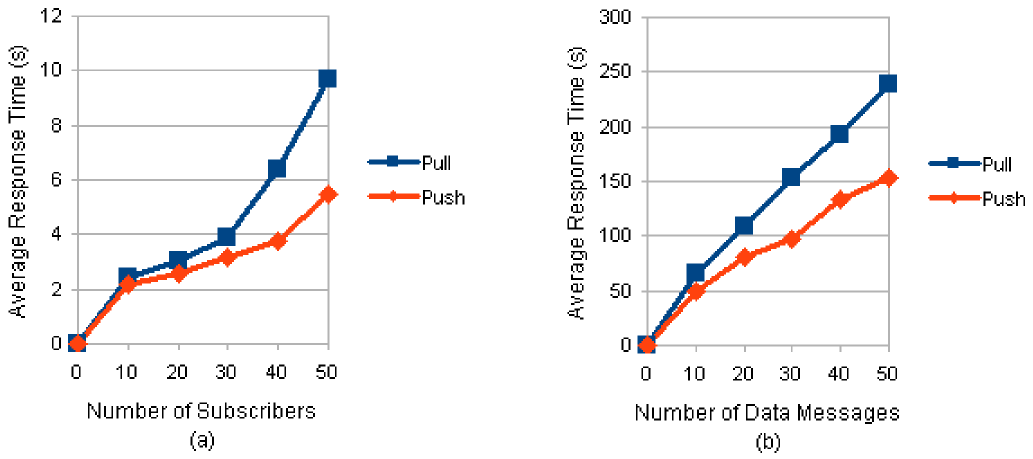

Fig. 6: Comparison of average response times between pull-based and push-based architecture

4. Conclusion and Future Works

Our architecture is aimed to merge the potential advantages of pub/sub model and Web service technology together and replace a traditional pull-based architecture with a new distributed push-based architecture. The broker is a core component of this architecture. It acts as a middleware for receiving and sending messages. It also could possibly perform several functions such as data transformation, code conversion and conditional routing. Therefore, it must be designed to be able to handle heavy workloads. To make our architecture interesting to invest, the proof-of-concept experimental results are provided. They showed that the average response time and the workload on publishers are significantly minimized in the push-based architecture. Therefore, this architecture could support the future trend of thin clients in ubiquitous computing such as sensor devices or mobile devices. This paper does not mention about the security, quality of service (QoS) and transaction. However, the concept of Web Services Atomic Transaction (WS-AtomicTransaction) [13] can be applied to enhance the reliability which is considered to be our future work.

Our architecture may applicable in some applications that need to distribute or multicast many data messages to client in urgent situation such as Tsunami alert system. It can also be applied to several business applications such as stock quotes, currency exchange, oil price, gold price, customer blacklist or news headline. To potentially enhance the performance of application integration, distributor can also use this architecture to quickly distribute updated data. The scenario of a retail stock distribution is explored as when the rate goes beyond a threshold price, all registered subscribers then can get the new price immediately. This shows that our architecture can efficiently support tracking data for many purposes.

5. References

[1] J. Lee, K. Siau, S. Hong. Enterprise integration with ERP and EAI. *Communications of the ACM*. 2003, **46** (2): 54–60.

[2] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*. 2003, **35** (2): 114-131.

[3] I. Gorton. *Essential Software Architecture*. Springer, 2006.

[4] W3C. SOAP Version 1.2 Part 1. Available: *http://www.w3.org/TR/soap12-part1/*

[5] W3C. Web Services Description Language (WSDL) Version 2.0 Part 1. Available: *http://www.w3.org/TR/wsdl20/*

[6] OASIS. Web Services Notification (v 1.3). Available: *http://docs.oasis-open.org/wsn/*

[7] W3C. Web Services Eventing. Available: *http://www.w3.org/Submission/WS-Eventing/*

[8] Y. Huang, and D. Gannon. A Comparative Study of Web Services-based Event Notification Specifications. *Proc. of the 2006 International Conference on Parallel Processing Workshops (ICPPW'06)*, 2006.

[9] A. Quiroz, and M. Parashar. Design and Implementation of a Distributed Content-based Notification Broker for WS-Notification. *Grid Computing Conference*, 2006.

[10] S. D. Labey, and E. Steegmans. Extending WS-Notification with an Expressive Event Notification Broker. *2008 IEEE International Conference on Web Services*, 2008.

[11] Y.Huang, A.Slominski, C. Herath, and D. Gannon. WS-Messenger: A Web Services-based Messaging System for Service-Oriented Grid Computing. In: Y.Huang, et al (eds.). *Proc. of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, 2006.

[12] OASIS. WS-AtomicTransaction (v 1.2). Available: *http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec.html*