An Event-Based Approach to Reducing Coupling in Large-Scale Applications

Bartosz Kowalewski[1], Marian Bubak[1,2], and Bartosz Baliś[1]

[1] Institute of Computer Science, AGH, Krakow, Poland
[2] Academic Computer Centre CYFRONET AGH, Krakow, Poland
kowalewski.bartosz@gmail.com, {bubak,balis}@agh.edu.pl

Abstract. Large-scale distributed applications tend to become more and more complex and hard to develop, and execute. Approaches used when building such systems have to be revised, in order to decrease coupling within the code and increase productivity during the development process. Especially event-based programming applied to Web services should gain much attention. In this paper, we present an experimental publish/subscribe infrastructure, which introduces robust event-based mechanisms to be used with Web services-enabled applications. The concept of this solution is built upon the extensibility and configurability principles. We show that performance gap between traditional distributed event-based technologies and the Web also services-based approach is not necessarily as significant as most people tend to think.

Keywords: Large-scale computing, distributed computing, decoupling, Web services, event infrastructure, publish/subscribe, WS-Notification.

1 Introduction

The main characteristic of an event-based design is its striving to decrease coupling in a system [5]. Coupling, usually contrasted with cohesion, is the degree of association between modules, components, subsystems, etc. [6] Over the past three decades a lot of attention has been paid to this concept, both in scientific and commercial computing. Besides many other measures, the level of coupling started to be treated as quality metrics for software design. It has been observed that tight coupling can lead to problems in all phases of application's life cycle. It can make introducing changes in particular parts much complicated and force developers working on distinct modules to frequently synchronize their activities. Tight coupling can also make it impossible to test components separately. On the other hand, loose coupling is usually achieved at the cost of performance: "Loose coupling intentionally sacrifices interface optimization to achieve flexible interoperability among systems that are disparate in technology, location, performance and availability" [11].

Cain and McCrindle state that unmanaged coupling is an indicator of potential productivity bottlenecks [1]. They accentuate the impact of a flawed

M. Bubak et al. (Eds.): ICCS 2008, Part III, LNCS 5103, pp. 358–367, 2008.

architecture on the number of developers that can work in parallel. They conclude with the statement that improperly coupled software causes people to be improperly coupled.

The event-based programming paradigm is built upon the concept of an event object, an entity that represents a situation, an occurrence of interest to third parties. This approach reduces the overall complexity of a system, providing much flexibility when coding, testing and maintaining the application. Unfortunately, it can lead to an increase in complexity of the parts' internals, making it extremely difficult to understand their operation without analyzing the rest of the system.

Event-based approach is becoming more important in contemporary software systems, as the volume of real-time data that enterprises must process and manage continues to increase. The most advanced event-based approach – Complex Event Processing (CEP), which is predicted to become mainstream in the nearest future, emphasizes the act of processing multiple events, aiming at identifying the meaningful ones and discovering complex events. CEP introduces new areas where events could be applicable and defines additional challenges to be faced [14]. Complex Event Processing is even predicted to be heavily used in Enterprise Service Buses.

Many real world examples could also be observed in the Grid environment. The Flood Forecasting Simulation Cascade (FFSC) [9], developed as a part of the K-Wf Grid project, is a loosely coupled large-scale application addressing the complex problem of flood prediction. FFSC takes advantage of the Service-Oriented Architecture (SOA) to handle real-time forecasting. In such application scenarios Web services-based event-driven mechanisms seem a natural approach, providing a flexible and efficient means to handle high data rates.

The publish/subscribe pattern, decoupling subscribers from publishers, has become a critical part of many system architectures [3] and is increasingly being used in a Web services context. The Web services, which are currently the preferred standards-based way to realize Service-Oriented Architecture (SOA), are emerging as the next generation platform for large-scale distributed applications. While the standard enables applications to communicate over Internet protocols, it is still lacking built-in mechanisms supporting the publish/subscribe model. As a response to the need of standardization two partially overlapping specifications, offering a foundation for event-driven architectures built using Web services, were proposed: Web Services Notification (WSN) [13] and Web Services Eventing [12]. Both of these families of specifications and related white papers define standard interoperable protocols through which Web services can exchange event objects. The single greatest drawback of the event-based approach applied to Web services is the delegation of validation of message payload against some contract to the application logic. The WSDL documents based on WS-Notification or WS-Eventing standards will not provide definition for the format of message payload.

Running event-based applications using the Web services technology still remains a challenging problem. There is no single widely accepted standard,

providing interoperability between any Web services-based publish/subscribe applications. Another important issue is that the Web Services Notification and Web Services Eventing specifications do not address all of the problems which arise when using publish/subscribe communication with Web services and are considered to be rough drafts, not solid, stable specifications.

Currently none of the publish/subscribe infrastructures based on the above-mentioned specifications provide features that would make them successfully compete with the traditional approaches, like Java Message Service (JMS) or CORBA Notification Service. The performance of these frameworks is far from being satisfactory. What is more, these solutions fail to provide straightforward ways to be employed into an application without the need to understand various complicated mechanisms used. The main assumption is that the designer and the developer should be unaware of the complexity of the infrastructure.

In this paper, we describe our approach to development of an efficient publish/subscribe infrastructure compliant with the Web Services Notification specification (WSN-PSI). The main advantages of this solution are its extensibility and configurability which make it possible to adjust the solution to meet various requirements. While providing these features, WSN-PSI still puts emphasis on performance, showing that the gap between traditional distributed event-based technologies and Web services-based solutions does not have to be enormous.

2 Background

A number of projects is being developed, that aim in providing event-based infrastructures build upon Web services. Those solutions implement WS-Eventing, WS-Notification, or both of these specifications. Hopefully, a new composite standard, WS-EventNotification, was announced and is planned to be available in 2008 [4]. It is intended to solve the interoperability problems between the two specifications currently used.

The WS-Messanger [10] is an implementation of both the Web Services Notification and Web Services Eventing specifications. This solution attempts to support mediation between these two incompatible standards.

Apache Muse [7] provides the functionality defined in Web Services Notification. It also implements other WS-* specifications – Web Services Resource Framework and Web Services Distributed Management.

ServiceMix [8] is an Open Source Enterprise Service Bus (ESB) with support for WS-Notification. This functionality is provided as a binding component and cannot run separately.

A new emerging framework for notification in Grid systems is presented in [2]. This content-based distributed notification service is based on WS-Notification. Unfortunately, the project is still in its initial phase.

Despite the number of Web services-based publish/subscribe infrastructure providers, none of them offers a high degree of extensibility and configurability. What is more, those solutions do not meet the performance requirements of contemporary distributed systems and cannot compete with the traditional

event-based solutions. Furthermore, most of the WS-Notification implementations mentioned above are inherent parts of larger, more complex solutions, and cannot run separately. One is unable to only use the publish/subscribe modules.

3 Concept of WSN-PSI

3.1 Requirements

The basic and obvious assumption that the solution is to be founded on the concept of Service-Oriented Architecture and Web services implicates numerous requirements, like service contract or reusability. These obvious requirements will not be enlisted within the objectives below. The following paragraphs present features that would define a noteworthy Web services-based publish/subscribe solution.

Among the most significant functional requirements we can mention:

- well-defined message exchanges – all of the communication scenarios associated with actions other than notification exchange should be precisely defined, enabling full compatibility between different implementations or versions of the infrastructure,
- dynamic reconfiguration of the environment – endpoints taking part in communication need have no knowledge of other endpoints prior to registration or subscription, nodes can be created and destroyed without having negative impact on infrastructure operation,
- messaging brokers – it should be possible to decouple producers and consumers by placing intermediary services (brokers) between them,
- topics – a topic (subject) should be attached to every message, enabling clients to separate independent message exchanges,
- filtering – it should be possible to limit number of messages sent to a consumer using filter constraints.

Non-functional requirements concerning our publish/subscribe solution are as follows:

- standards-based approach – will provide optimal solution adoption and reduce efforts involved in building and maintaining applications founded on this infrastructure,
- performance – reduce complexity to get as fast and efficient infrastructure as it is possible with contemporary Web services technologies,
- usability – the solution has to be easy to employ in order to produce a specialized messaging infrastructure, minimizing the effort required to switch from using the standard request/response pattern to the event-based approach,
- extensibility – should provide for change while minimizing impact to existing infrastructure fragments,
- configurability – the constituent parts of the solution should be configurable and exchangeable,
- scalability – the infrastructure should be capable of dynamic messaging endpoint allocation and environment reconfiguration.

3.2 Design

We decided to build the infrastructure on the basis of the already mentioned WS-Notification (WSN) specification to ensure better adoption of the solution. WSN is a natural successor of the Open Grid Services Infrastructure (OGSI) Notification standard, which makes it the approved event-based approach for the Grid environment. Moreover, WS-Notification provides many advanced features, mechanisms and application scenarios. Last but not least, using WS-Notification lets us observe which areas are sill covered inadequately by the available standards and need to be improved.

The overall architecture of the system is founded on entities defined in WS-Notification. WSN-PSI implements a large part of WS-BaseNotification and WS-BrokeredNotification, and a basic subset of concepts defined in WS-Topics. Worth mentioning is the fact that WSN-PSI does not restrict selection of cooperating nodes to some predefined architectures and does not require endpoints taking part in communication to have knowledge of other endpoints prior to subscription or registration. Because of this approach, WSN-PSI provides really flexible and dynamically reconfigurable environment. Also by providing support for intermediary services WSN-PSI empowers the process of decoupling.

The whole infrastructure is built with the assumption that the only way to make the system usable is to provide a high level of extensibility and configurability. WSN-PSI provides a collection of configurable building blocks, enabling one to construct a specialized publish/subscribe solution. They were designed with care for loose coupling, high cohesion, and the fundamental object-oriented design principle – the single responsibility principle. Fig. 1 presents the approach used to implement WSN services.

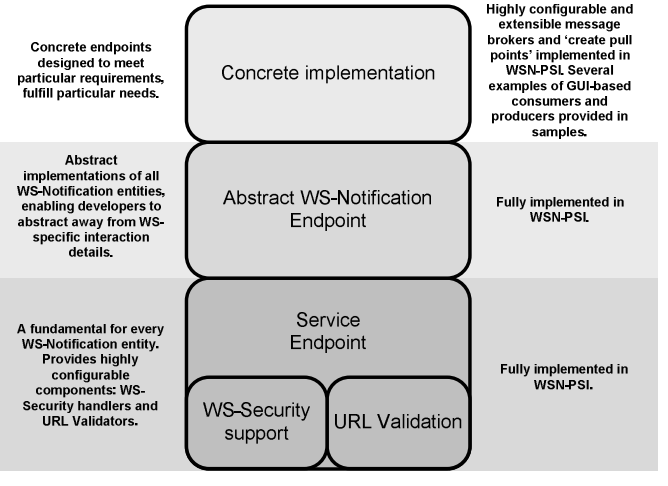

Fig. 1. Design of a WS-Notification-based entity

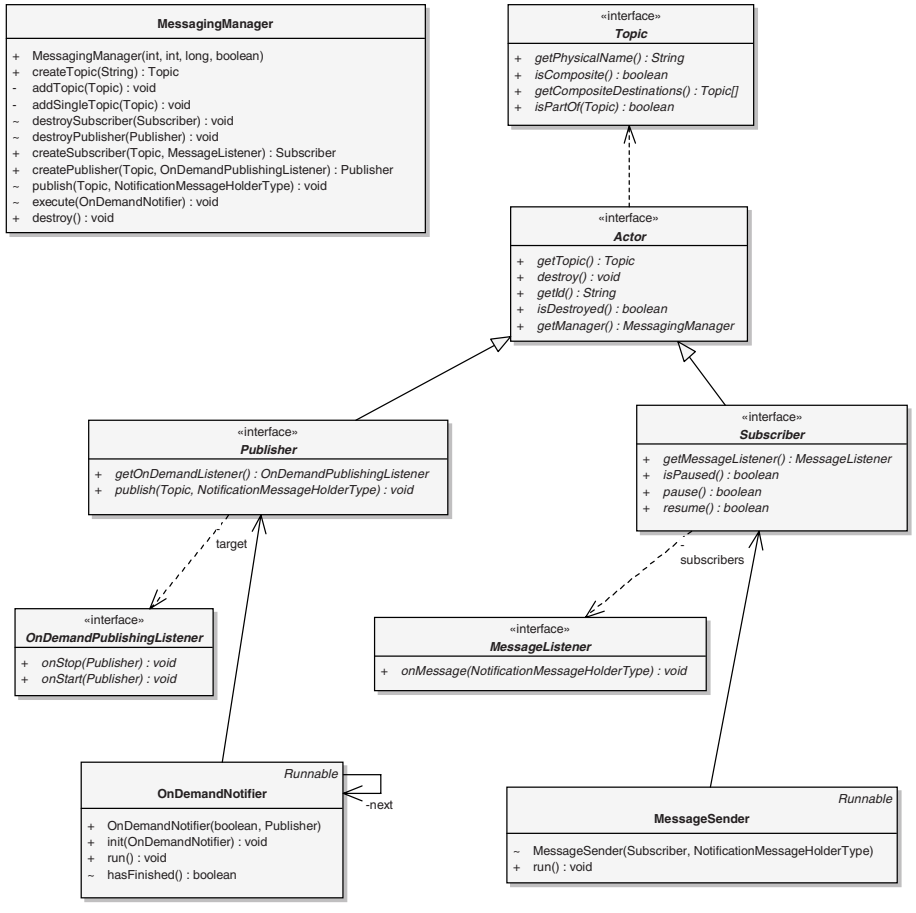

Fig. 2. WSN-PSI: a simplified class diagram of the internal brokering system

The full-featured and highly configurable notification broker is a good illustration of the application of this model. Mechanisms dedicated to broker customization include, among others, the ability to select the method of event notification dispatching inside the broker. One can currently choose between JMS-based dissemination and an efficient internal dispatching technique. Fig. 2 provides a simplified class diagram, presenting the design of this internal brokering system incorporated into WSN-PSI. This module defines only two types of actors: *Publishers* and *Subscribers*. They can be created and are managed by a centralized *Messaging Manager*, build upon a thread pool dedicated to executing long tasks. Every *Subscriber* is provided a *Message Listener*. Each time a notification message is routed to the *Subscriber*, a task wrapped into a *Message Sender* object is sent to the *Messaging Manager*, which plays the role of a task scheduler. Similarly, every demand-based *Publisher* is provided an *On-Demand*

Publishing Listener. *Publishers* utilize *On-Demand Notifiers* to wrap tasks associated with pausing and resuming WSN on-demand publishers. The advantage of the described approach is that the messages are dispatched locally, no external messaging provider is used. This reduces additional overhead associated with transferring the messages. What is more, this broker directly uses build-in Java thread pooling mechanisms which also causes an increase in performance. One significant drawback of this approach is that, unlike in JMS-based broker, it is impossible to plug into the local messaging system using non-WSN clients.

The design is not the only element that makes the solution highly usable and efficient. Also the selection of technologies has a significant impact on the characteristics of the infrastructure. WSN-PSI is founded on many widely accepted standards and libraries. As it was stated when specifying requirements, one of the main principles employed when designing the infrastructure was to use standardized approaches whenever possible. Adoption of well-known standards makes the infrastructure easier to understand and use, reducing efforts involved in building and maintaining applications founded on WSN-PSI. Standards-based approach also simplifies the process of evolution of the infrastructure and increases portability. To ensure required performance level, technologies had to be picked carefully. For example XFire, which proved to be really efficient, was chosen as the SOAP framework.

4 Performance Analysis

The approach described here attempts to show that performance of Web services-based solutions does not necessarily have to be much worse than performance of the traditional distributed event-based technologies. To prove this hypothesis a series of tests was prepared and executed. All the tests were run on a single PC machine and were designed to present the same single scenario implemented using different technologies. This scenario defined two endpoints, a master and a slave, communicating through a broker. Messages were sent from the master to the broker, further to the slave node and back to the master through the intermediary service. Each time round trip time (RTT) was evaluated and the time required to send a message from one node through the broker to the other endpoint was calculated. This scenario was repeated 1000 times using messages with 1024 characters long payloads.

The tests cover two configurations of WSN-PSI infrastructure and the most significant WS-Notification implementations currently available. The first WSN-PSI notification broker version was based on internal message dispatching mechanisms, while the second one used dispatching mechanisms implemented on the basis of JMS. The ServiceMix ESB was run with the Servicemix-wsn2005 binding component deployed. The WS-Notification implementation provided by the Apache Software Foundation, Apache Muse, had to be modified to be suitable for the test scenario. Muse does not implement the notification broker functionality and an internal bridge between notification consumer and producer had to be added. To have the possibility to compare Web services-based solutions and the

distributed event-based technologies a sample Java Message Service (JMS) specification implementation provided by JBoss Application Server was also taken into account. Tab. 1 presents results of the experiments.

Table 1. Average delivery time

Technology/solution	Average delivery time (ms)
Java Message Service (JMS)	11.8
WSN-PSI (internal)	16.1
WSN-PSI (JMS)	26.4
ServiceMix	28.5
Apache Muse	42.2
WS-Messenger	106.9

The experimental data shows that it is possible to create a Web services-based solution that will successfully compete with JMS. The results also suggest that most of the infrastructures currently available tend to underperform and could easily be improved. On the other hand, one should take into account that using the same payload sizes for the messages being disseminated in different tests does not make the overall size of the messages equal. Various strategies to building notification messages, possibly incorporating WS-Addressing support, may have considerable impact on the empirical results.

Additionally, we conclude that the performance gap between traditional event-based approaches and the Web services-based solutions grows considerably with the number of advanced filtering and Quality of Service (QoS) mechanisms employed.

5 Possible Applications

WSN-PSI is designed to cover a wide spectrum of scenarios. It provides a high level of scalability, being 100% compliant with the WS-Notification specification. This causes the topology not to be restricted by any means, enabling developers or users to have full freedom to build their own specialized configuration. WSN-PSI also uses a layered architecture with many interchangeable modules, easily configurable using XML files.

The intermediary service is fully implemented and could be used with minimum changes in the configuration files. The project also provides many stubs and samples written in Java programming language. These could straightforwardly be used to incorporate Web Services Notification functionality into an existing large-scale application. All of these services could be deployed using the embedded Jetty WebServer. This significantly simplifies the process of packaging and running the application, and could be used when the standard, Web ARchive-based, approach to deployment is undesirable.

WSN-PSI could also be used in scenarios requiring support for security. The infrastructure uses WS-Security, which provides basic security mechanisms for Web Services Notification message exchanges. Unfortunately, the mechanisms employed when handling WS-Security data cause a serious decrease in performance.

Worth mentioning is the fact that the Java programming language was chosen for developing the experimental solution, to provide sufficient level of portability and to be able to take advantage of existing implementations of Web services-based technologies. Java is the only programming language used in WSN-PSI. Development of sample infrastructures for other languages is considered to be out of scope of this work and may be added in the future phases of the project. Also no cross-language clients are provided. Fortunately, the Web Services Notification compliant WSDL files shipped with WSN-PSI could by used to generate clients in any programming language.

6 Conclusions and Future Work

In this paper we presented WSN-PSI: an efficient publish/subscribe infrastructure that could be successfully used to decrease coupling in large-scale Web services-based applications. The solution was designed to meet tough performance requirements and make Web services-based technologies an option when choosing communication means for a highly decoupled system.

One considerable drawback of using the event-based approach with Web services, which was identified during early development, is the incompleteness of Quality of Service (QoS) mechanisms available. QoS comprises various aspects of messaging, including reliable delivery, order, security, duplicate elimination and many more. The family of second-generation Web services documents (WS-*) provides WS-Security, WS-ReliableMessaging and WS-Reliability specifications. Unfortunately, these standards are immature and do not cover all of the desirable mechanisms. Moreover, both WS-Notification and WS-Eventing leave many areas unstandardized, increasing the risk of incompatibility between various infrastructures. Absence of some advanced mechanisms, like support for event object routing, may also cause the more complex configurations to malfunction.

Future plans for the WSN-PSI project include, amongst all the other goals, experimenting with various technologies that could be applicable to the infrastructure. It also seems reasonable to experiment with some specifications from the huge family of second-generation Web services documents, other than WS-Notification and WS-Security already used in WSN-PSI. Furthermore, the project lacks more complex tests in a real grid environment.

Acknowledgments. This work was supported by EU projects ViroLab IST-027446 and CoreGRID IST-004265 with the related Polish grants SPUB-M.

References

1. Cain, J., McCrindle, R.: An Investigation into the Effects of Code Coupling on Team Dynamics and Productivity. In: IEEE 26th Annual International Computer Software and Applications Conference (COMPSAC) (2002), http://www.blunder1.demon.co.uk/research/COMPSAC2002.pdf
2. Quiroz, A., Parashar, M.: A Framework for Distributed Content-based Web Services Notification in Grid Systems. Future Generation Computer Systems 24, 452–459 (2008)
3. Carzaniga, A., Di Nitto, E., Rosenblum, D., Wolf, A.: Issues in Supporting Event-based Architectural Styles. In: ICSE 1999 Workshop on Engineering Distributed Objects (EDO 1999) (1999), http://www-serl.cs.colorado.edu/~carzanig/papers/isaw3.pdf
4. Hewlett Packard Corporation, IBM Corporation, Intel Corporation, and Microsoft Corporation. Toward converging Web service standards for resources, events, and management, http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/Harmonization_Roadmap.pdf
5. Eugster, P., Felber, P., Guerraoui, R., Kermarrec, A.-M.: The Many Faces of Publish/Subscribe. ACM Computing Surveys, http://www.irisa.fr/paris/Biblio/Papers/Kermarrec/EugFelGueKer03ACMSurvey.pdf
6. Faison, T.: Event-Based Programming: Taking Events to the Limit, ch. 1. Apress (2006)
7. The Apache Software Foundation. Apache Muse, http://ws.apache.org/muse/
8. The Apache Software Foundation. ServiceMix – WSN 2005 (2005), http://incubator.apache.org/servicemix/servicemix-wsn2005.html
9. Habala, O., Mališka, M., Hluchý, L.: Service-based Flood Forecasting Simulation Cascade in K-Wf Grid. In: K-WfGrid - The Knowledge-based Workflow System for Grid Applications, Proceedings of CGW 2006, vol. II (2006)
10. Huang, Y., Slominski, A., Herath, C., Gannon, D.: WS-Messenger: A Web Services-based Messaging System for Service-Oriented Grid Computing, http://www.cs.indiana.edu/~yihuan/research/HuangY-WSMessenger.pdf
11. Kaye, D.: Loosely Coupled: The Missing Pieces of Web Services, ch. 10. RDS Associates (2003)
12. Microsoft, IBM, TIBCO, Bea Systems, and Computer Associates. Web Services Eventing, http://www.w3.org/Submission/WS-Eventing/
13. OASIS. Web Services Notification, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn
14. Wu, E., Diao, Y., Rizvi, S.: High-Performance Complex Event Processing over Streams. In: 2006 ACM SIGMOD International Conference on Management of Data (2006), http://avid.cs.umass.edu/sase/uploads/pubs/sase-sigmod2006.pdf