Push 기반 이벤트 알림 서비스에 대한 연구

한영태 민덕기
건국대학교 컴퓨터·정보통신공학과
{headline, dkmin}@konkuk.ac.kr

A Study of Push-Style Event Notification Service

Youngtae Han Dugki Min
Department of Computer Science and Engineering, Konkuk University

요 약

이벤트 알림 시스템에 대한 연구는 비동기적으로 발생되는 이벤트를 기반으로 실행되는 응용 프로그램을 지원하기 위하여 많은 연구가 이루어져 왔다. 본 논문은 확장성과 효율성을 보장하기 위한 Push 기반 이벤트 알림 서비스를 제시한다. 특히 이벤트 중재자(Broker)를 확장하기 쉽게 구현 하였으며, 병렬적 데이터 전송 등을 통하여 데이터 전송 효율성을 제공해 주고 있다. 또한 XML을 사용한 레코드 기반 이벤트 모델을 구현하여 이기종 호환성 보장과 구현 언어 독립적인 구조를 제공하고 있다. 구현된 이벤트 알림 서비스에 대해 구현 이슈를 살펴보고, 성능 측정하고 그 결과를 분석한다.

1. 서 론

이벤트 기반 시스템에 대한 연구는 분산 어플리케이션 미들웨어, 이벤트 기반 메시징 모델, 이벤트 구동 시스템, 그리고 메시지 기반 미들웨어(MOM: Message Oriented Middleware) 등의 영역에서 많은 연구들이 이루어지고 있다[1]. 특히 인터넷 기반에서 확장성 있는 분산 시스템 구축을 위해 좀더 유연성 있는 통신 구조와 시스템 구성을 위한 연구가 이루어졌다.

본 논문에서는 Push 기반의 이벤트 알림 서비스를 제시한다. 이 서비스는 XML을 사용한 레코드 기반 이벤트 모델 구현함으로써 이기종 호환성 보장과 구현 언어 독립적인 특성을 가지고 있어 시스템의 유연성과 확장성을 보장하기 위해서나 Web Service와 같이 XML을 기반으로 하는 시스템에 적합하다. 또한 성능 측정을 통하여 제시된 이벤트 서비스의 구현 상 중요한 포인트인 레코드 기반 이벤트 모델 구현 방법과 통신 방식에 대해 분석한다.

본 논문의 구성은 다음과 같다. 2 장에서는 기존에 개발되어진 이벤트 기반 시스템을 소개한다. 3 장에서는 Push 기반의 이벤트 시스템을 제시한다. 4 장에서는 XML을 사용한 레코드 기반 이벤트 모델과 통신 방식에 따른 구현 이슈에 대하여 알아본다. 5 장에서는 4 장에서 제시된 구현 방법에 따라 구현하여 성능 측정한 결과를 살펴본다. 그리고 마지막 6 장에서 결론 및 향후 연구 방향을 제시한다.

2. 이벤트 기반 시스템 분류

본 장에서는 분산 환경에서 이벤트 서비스를 제공하기 위한 기존 연구들을 분류하고 그 특징을 알아본다.

2.1 이벤트 모델

이벤트 기반 시스템은 이벤트 모델의 구성 형태에 따라 다음의 3 가지 모델로 나눌 수 있다. 튜플 기반, 레코드 기반, 그리고 객체 기반 모델로 나눌 수 있다[2]. 튜플 기반 모델에서는 이벤트 정보를 문자열의 셋으로 정의하고 있으며, 예를 들어 (EventDomain, 5, datainfo, ...) 과 같이 표현되는 것을 말하는 것이다. 레코드 기반 모델에서는 이벤트 정보를 이름과 값을 가지는 정형화된 필드의 집합으로 정의하고 있다. 대부분의 이벤트 서비스가 이 모델을 사용하고 있다. 마지막으로 객체 기반 모델은 레코드 필드와 더불어 메소드의 집합으로 이벤트를 정의하고 있다.

2.2 이벤트 시스템 아키텍쳐

이벤트 시스템은 기본 소프트웨어 구성 요소의 아키텍쳐 관점에서 클라이언트-서버 모델과 Peer-to-peer 분산 모델로 나눌 수 있다. 이벤트 시스템의 기본 소프트웨어 구성요소로는 이벤트 서버와 이벤트 클라이언트 또는 두 기능을 다하는 컴포넌트가 있다.

클라이언트-서버 모델에서 이벤트 서버는 이벤트를 수신하고 저장하고 분배하는 역할을 수행하며 확장성을 보장하기 위하여 분산 서버 환경으로 구현되기도 한다. 이벤트 클라이언트는 이벤트를 생성하여 이벤트 서버에 전송하거나 이벤트를 소비거나, 또는 두 가지 역할을 다하기도 한다[2,3,4,5,6].

Peer-to-peer 분산 모델에서는 모든 노드가 동일한 기능을 하게 구현되어 있다. 이벤트를 분배하기 위한 Root 노드를 중심으로 응용 프로그램 레벨의 멀티캐스트 라우팅으로 이벤트를 전송하는 방식이다[7,8,9,10]. 따라서 모든 노드는 이벤트 생성, 이벤트 분배, 그리고 이벤트 소비하는 등의 모든 역할을 다하는 방식이다.

3. 이벤트 시스템 아키텍쳐

본 논문에서는 Push 기반의 이벤트 알림 서비스를 제시한다. Push 기반 이벤트 알림 서비스는 기본적으로 이벤트 생성자(Producer)가 발생한 이벤트를 이벤트 소비자(Consumer)에게 통지하는 방식이다. 이들을 연결하기 위하여 이벤트 중재자(Broker)들을 두고 있다.

그림 1은 Push 기반의 이벤트 알림 서비스 개념도를 나타낸 것이다. 구성 요소로는 이벤트 중재자, 이벤트 생성자, 그리고 이벤트 소비자가 있다. 이벤트 중재자는 이벤트 소비자와 이벤트 생성자를 연결시켜 주는 역할을 하며, 부가적인 기능으로 필터링, 큐잉, 이벤트 감시, 모니터링, 관리 GUI, 그리고 형식

변환 등을 해준다. 이벤트 소비자는 관심있는 이벤트의 주제나 이벤트의 패턴등을 등록하거나(Subscribe) 등록 해지를 하며(Unsubscribe), 발생된 이벤트들 중에서 관심 있는 내용을 비동기적으로 통지 받는다(Notify). 이벤트 생성자는 관련 이벤트를 이벤트 중재자에게 광고하거나(Advertise) 광고 해지를 하며(UnAdvertise) 생성된 이벤트를 통지한다(Publish).

본 논문에서 제안하는 이벤트 알림 서비스는 그림 2 와 같이 구성되어 있다. 이벤트 생성자와 이벤트 소비자는 이벤트 중재자들과 연결되어 있으며, 이벤트 중재자들은 상호 연결을 통하여 이벤트 생성자와 이벤트 소비자들을 연결하는 기능을 수행한다.

이벤트 알림 서비스 개발에 중요한 요소로 확장성, 신뢰성, 보안성, 그리고 효율성을 들 수 있다. 예를 들어 네트웍 관리 응용 프로그램이나 시스템 관리 프로그램의 경우 관리 대상 장치나 시스템에서 발생된 이벤트는 관리 서버로 수집되고 분석되어 진다. 이러한 응용 프로그램의 경우 확장성과 효율성은 더욱 중요한 요소이다. 확장성은 이벤트 중재자의 확장이 용이한 구조, 이기종 호환성, 그리고 구현 언어 독립적인 시스템 구현을 의미한다. 특히 이기종 호환성과 구현 언어 독립적인 시스템 구현을 위하여 이벤트 모델을 XML 기반의 이벤트 데이터를 제시한다. 또한 효율성을 보장하기 위하여 미리 정의한 룰에 따라 이벤트 소비자와 이벤트 생성자를 연결할 수 있게 지원하고 있다. 이는 이벤트 생성자나 이벤트 소비자의 이벤트 주제나 패턴을 등록하는 로직과 이벤트 생성자의 광고 로직의 반복 작업을 줄였다. 그리고 병렬적인 이벤트 분배 로직을 구현과 네트웍 오류에 따른 시스템 오류 영향 최소화된 구현을 하였다.

4. 이벤트 시스템 구현 이슈

본 장에서는 이벤트 시스템 구현에 있어 XML 기반의 이벤트 모델을 정의할 때의 구현 이슈와 통신 방식에 따른 구현 이슈를 정의하고 장단점을 살펴본다.

4.1 XML을 사용한 레코드 기반 이벤트 모델

XML을 사용한 레코드 기반 이벤트 데이터 표현은 시스템의 유연성을 보장해 주기 위해서나 Web Service 와 같이 XML을 기반으로 운영되는 시스템에 적용하기 위한 이벤트 서비스에 유용하다. 이 모델은 플랫폼과 언어 독립적인 구현이 가능하다는 장점이 있다. 또한 정의한 레코드를 XML 문서로 엔코딩과 XML 문서를 미리 정의한 레코드 형식으로 인식하기 위한 디코딩 로직을 일반화 할 수 있다는 장점이 있다. 우리는 객체를 XML 로 표현하거나 XML을 객체로 변환하는 방법을 연구하였다. 그림 3 은 XML 로 정의된 이벤트 데이터의 예이다.

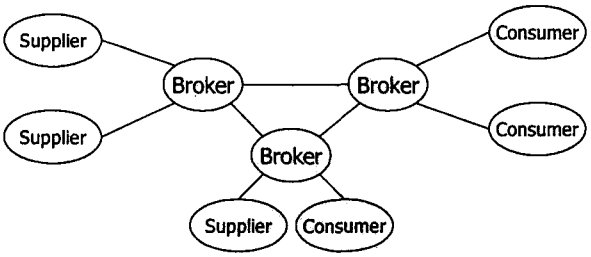

그림 1 Push 기반 이벤트 알림 서비스 구조도

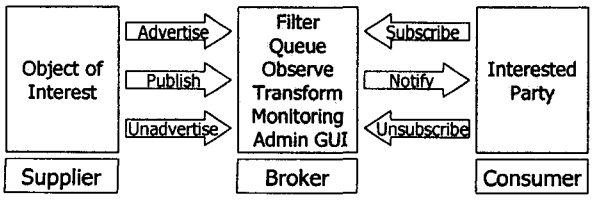

그림 2 Push 기반 이벤트 알림 서비스 개념도

XML을 사용할 때의 문제점으로는 XML 문서에 대한 엔코딩과 디코딩의 오버헤드가 커서 이벤트 시스템 처리 시간의 많은 영향을 주는 단점이 있다. 이는 5 장에서 성능 측정 결과를 통하여 더 자세히 알아보겠다. 좀더 나은 처리 능력을 보장해 주기 위해서는 XML 문서에 대한 오버헤드를 최소화할 수 있는 구현이 필요하다. 그리고 이벤트 중재자에서 내용 기반 필터링을 사용하는 경우 기본적으로 모든 이벤트에 대하여 엔코딩과 디코딩을 해야하는 단점이 있다. 이를 해결하기 위해서는 XML 문서를 객체로 전환하지 않고 필요한 레코드만을 검출하는 방식 등에 대한 연구가 필요하다.

한편 레코드 기반 이벤트 모델 정의 방식으로 많이 사용되는 것으로는 객체 직렬화 기법을 있다. 객체 직렬화 기법은 Java 나 C++ 같은 객체 지향 언어에서 기본적으로 제공하고 있으며 이를 사용하는 것이 일반적인 구현 방식이다. 그러나 JECho[3] 등의 일부 이벤트 시스템에서 자체 객체 직렬화 구조를 개발하여 사용한 예가 있다. 객체 직렬화 기법을 사용하는 경우 장점은 구현이 쉽다는 것이다. 그러나 언어나 시스템에 의존된다는 단점이 있어 유연성이 좋지 않다.

4.2 통신 방식에 따른 구현 이슈

본 논문에서 제시한 이벤트 알림 서비스는 시스템 독립적인 구현을 위하여 인터넷 표준인 UDP와 TCP를 기반으로 구현되었다. UDP 프로토콜의 경우 에러가 네트웍 에러가 발생할 수 있다는 단점이 있으며, TCP 프로토콜의 경우 에러 상황 시 운영 체제에 따라 시스템 멈춤 현상이 생길 수 있다는 단점이 있다. 5 장의 성능 측정 결과 작은 사이즈의 이벤트 데이터를 전송하는 상황에서는 두 프로토콜이 큰 차이가 없는 것으로 보여진다.

5. 성능 측정

5.1 성능 측정 환경

본 논문에서 제시한 Push 기반의 이벤트 알림 서비스에 대한 테스트 환경은 Intel PIII 800 MHz CPU, 256 MB 메모리를 사용하는 시스템이며 운영 체제는 RedHat Linux 9.0 이다. 그리고 Java 언어로 개발된 이벤트 생성자, 이벤트 소비자, 그리고 이

```
<?xml version="1.0" encoding="euc-kr"?>
<configuration>
  <START_CLASS type="class" classname="dmics.ems.StructuredEvent">
    <domainName type="string">SystemManagement</domainName>
    <typeName type="string">ApplicationCheck</typeName>
    <eventName type="string">HTTPD_Error</eventName>
    <timestamp type="long">1062434821250</timestamp>
    <priority type="int">5</priority>
    <headerFields type="hashtable">
      <HASHTABLE>
        <KEY type="string">EventTime</KEY>
        <VALUE type="string">2003.08.01 00:00:00</VALUE>
      </HASHTABLE>
    </headerFields>
    <dataFields type="hashtable">
      <HASHTABLE>
        <KEY type="string">ErrorRoot</KEY>
        <VALUE type="string">MemoryOverflow</VALUE>
      </HASHTABLE>
    </dataFields>
    <optionalField type="string">NULL</optionalField>
  </START_CLASS>
</configuration>
```

그림 3 XML을 사용한 레코드 기반 이벤트 표현 예

벤트 중재자 프로그램은 JDK 1.3 기반에서 실행하였다.

모든 테스트는 초당 20 개의 이벤트를 발생하는 이벤트 생성자를 늘려가는 방식을 사용하고 있으며 발생되는 모든 이벤트는 포아송 분포를 따르고 있다. 이벤트 소비자는 하나만 두고 있다.

5.2 성능 측정 결과

앞에서도 언급하였듯이 XML 파서에 따른 이벤트 처리량은 시스템 성능에 매우 중요한 요소이다. 그림 4 는 XML 파서를 DOM 방식을 사용했는지 SAXP를 사용했는 지에 따라 산출한 처리량을 보여주고 있다. DOM 방식을 파서를 사용하는 경우 최대 처리량이 118 Event/sec 정도이며 SAXP 방식의 파서는 182 Event/sec 로 더 높은 처리량을 보여주고 있다.

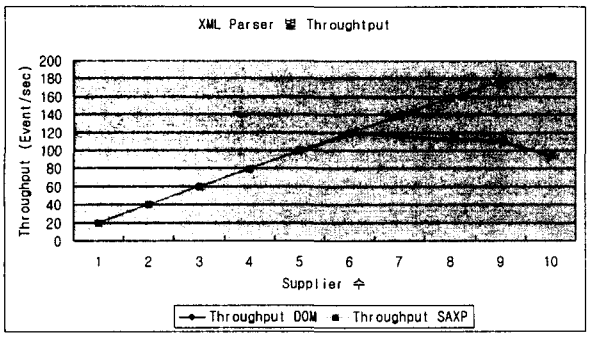

그림 4 XML 파서에 따른 처리량 비교

이벤트 서비스의 성능에 중요한 요소가 되는 것이 XML 파서라는 것을 측정을 통하여 알수 있었다. 표 1에서 보는 바와 같이 DOM 방식의 최대 처리량을 보이는 생성자 6 개 이상에서 Encoding 시간과 Decoding 시간이 급격히 커진 것을 확인할 수 있다.

Supplier 수	Encoding		Decoding	
	DOM	SAXP	DOM	SAXP
1	940.079365	429.662699	3173.01587	1723.4127
2	744.047619	413.888889	4130.35714	2073.01587
3	743.253968	493.452381	6519.64286	2718.65079
4	779.563492	569.940476	4263.19444	3849.60317
5	708.829365	676.190477	7298.4127	4535.01984
6	735.31746	745.039683	47606.6468	5501.78571
7	579.662698	657.638889	188010.417	7212.5

표 1 Parser 에 따른 Encoding/Decoding 시간(usec)

그림 5는 네트웍 통신 프로토콜에 따른 전송 시간을 실제 성능 측정을 통하여 산출해 보았다. 그림 5에 나타나 있듯 네트웍은 시스템에 큰 영향을 미치지 않는 것으로 나타났으며, 이는 어떠한 프로토콜을 사용하는 지는 용도에 따라 선택하는 것이 적당하리라 생각된다.

6. 결론

본 논문에서는 Push 기반의 이벤트 알림 서비스를 제시한다. 이 서비스는 XML을 사용한 레코드 기반 이벤트 모델 구현함으로써 이기종 호환성 보장과 구현 언어 독립적인 특성을 가지고 있다. 또한 효율성을 보장하기 위하여 미리 정의한 룰에 따라 이벤트 소비자와 이벤트 생성자를 연결할 수 있게 지원하고 있으며, 병렬적인 이벤트 분배 로직을 구현, 그리고 네트웍 오류에 따른 시스템 오류 영향 최소화된 구현을 하였다.

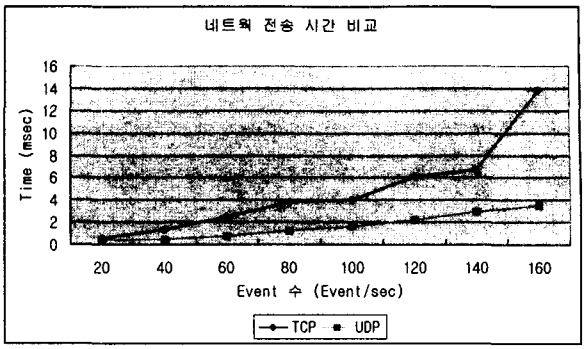

그림 5 네트웍 전송 시간 비교

향후 연구 과제로는 더욱 효율적인 이벤트 전송을 위하여 이벤트 데이터 필터링 구현과 Correlation 엔진에 대한 연구가 필요하다. 또한 신뢰성을 보장하기 위한 전송 메커니즘에 대한 연구와 보안성을 지원하기 위한 메커니즘 연구도 필요하다.

7. 참고 문헌

[1] M. D. Spiteri, "An Architecture for the Notification, Storage and Retrieval of Events", PhD Thesis, University of Cambridge, January 2000

[2] Cugola Gianpaolo, Di Nitto Elisabetta and Fuggetta Alfonso, "The JEDI Event-based Infrastructure and its Application to the Development of the OPSS WFMS", IEEE Transactions of Software Engineering, 2001

[3] Dong Zhou, Karsten Schwan, Greg Eisenhauer and Yuan Chen, "JECho--Interactive High Performance Computing with Java Event Channels", 3rd International Workshop on Java for Parallel and Distribute Computing, 2001.

[4] A. Carzaniga, "Architectures for an Event Notification Service Scalable to Wide-area Networks", PhD Thesis, Politecnico di Milano, December 1998

[5] Bill Segall and David Arnold, "Elvin has left the building: A publish/subscribe notification service with quenching", Proceedings AUUG97, September 1997

[6] B. Krishnamurthy and D. S. Rosenblum, "Yeast: A General Purpose Event-Action System", IEEE Transactions on Software Engineering, October 1995

[7] A. Rowstron, A-M. Kermarrec, M. Castro and P. Druschel, "SCRIBE: The design of a large-scale event notification infrastructure", NGC2001, UCL, London, November, 2001

[8] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz and J. Kubiatowicz, "Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination", NOSSDAV, 2001

[9] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward, "Gryphon: An Information Flow Based Approach to Message Brokering", International Symposium on Software Reliability Engineering, 1998

[10] Luis Felipe Cabrera, Michael B. Jones, and Marvin Theimer, "Herald: Achieving a Global Event Notification Service.", Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII), May 2001

[11] Object Management Group, "Notification Service", August 2002, http://www.omg.org/docs/formal/02-08-04.pdf