제18회 한국정보처리학회 추계학술발표대회 논문집 제9권 제2호 (2002.11)

분산 시스템을 위한 이벤트 통지 서비스의 설계 및 구현

염귀덕*, 민덕기*
*건국대학교 컴퓨터 · 정보통신공학과
e-mail:{kdyeom, dkmin}@konkuk.ac.kr

Design and Implementation of Event Notification Service for Distributed Systems

Kiduk Yeom*, Dugki Min*
*Dept of Computer Science and Engineering, Konkuk University

요 약

분산 어플리케이션 개발시 내부의 컴포넌트들은 다양한 종류의 미들웨어서비스를 필요로 한다. 특히, 컴포넌트들은 다양한 이벤트를 발생시키고 통지하는 상호작용을 하기 때문에 이벤트 통지 서비스는 컴포넌트들 간의 비동기적인 통신을 위해 필수적인 서비스이다. 본 논문에서는 자바기반의 다양한 분산 시스템에 적용될 수 있는 일반적인 이벤트 통지 서비스 프레임워크을 제시한다. 특히, 이벤트의 타입을 인터페이스기반으로 정의하고 이벤트객체로 작동되도록 함으로서 확장가능한 구조를 가지고 있다. 응용 사례로 분산 서버들의 동적 형상 관리 서비스에 적용해 보았다.

1. 서론

분산 어플리케이션의 내부 컴포넌트들은 다양한 형태의 상호작용을 필요로 한다[4]. 일반적으로, 통신 구조는 요청과 응답을 동기적으로 처리하는 동기적인(Synchronous) 통신과 요청과 응답 사이에 중간층을 두어 서로 비의존적인 관계로 클라이언트와 서버가 비동기적으로 통신할 수 있게 하는 비동기적인(Asynchronous) 통신으로 나눌 수 있다. 동기적인 통신은 미리 정해진 시나리오에 따라서 정해진 상대에게 메시지 전달로 통신을 하는 즉, 정적인 상호작용을 하는 통신방식이며 통신하는 객체들끼리 결합력이 높다. 반면에, 비동기적인 통신은 주로 이벤트를 발생해서 통지하는 방식인데, 어떤 컴포넌트에서 언제 이벤트가 발생할지 모르기 때문에 동적으로 상호작용을 하며 컴포넌트간 결합력이 낮다.

기존에 이미 다양한 이벤트 통지 서비스들이 제안되어 사용되어 왔다. 본 연구에서 고려하는 서비스도 기존 이벤트 통지 서비스들과 비슷한 일반적인 구조를 지닌다.

본 논문에서는 다음과 같은 두가지 요건을 만족하는 자바기반의 이벤트 통지 서비스의 프레임워크에 대한 설계 및 구현를 제시한다. 첫째로, 일반성이다. 즉, 분산환경의 자바기반 어떤 시스템에서도 적용될 수 있다. 둘째로, 확장성이다. 특히, 이벤트의 타입을 기본적인 것만 인터페이스로 정의해 두고 이벤트객체로 작동되기 때문에 확장가능한 구조로 이루어져 있다. 또한, 분산 시스템에 존재하는 다양한 서비스들을 정적 또는 동적으로 관리하는 동적 서비스 관리 프레임워크[5]에 우리의 이벤트 통지 서비스를 적용한 적용사례를 간단히 설명한다.

논문의 구성은 일단, 기존의 관련 연구들 중 가장 널리 알려져 있는 OMG의 Event Service, Notification Service와 J2EE에 포함되어 있는 JMS에 대하여 살펴본다. 그 다음 단락에서는 본 논문에

서 제시하는 이벤트 통지 서비스 프레임워크에 관한 아키텍쳐와 설계 및 구현에 대해서 설명하고 그 다음으로는 동적 서비스 관리 프레임워크에 우리가 개발한 이벤트 통지 서비스의 적용사례를 보여준다. 마지막 단락은 결론과 향후 추가되어야 할 내용으로 마무리 한다.

2. 관련 연구

기존의 이벤트 통지 서비스에 대하여는 많은 연구들이 이루어져 왔다[1,2,3]. 본 논문에서는 그 중에서 가장 보편적으로 알려지고 많이 사용하는 OMG(Object Management Group)에서 제시하고 개발한 CORBA(Corba Object Request Broker Architecture)의 Event Service, Notification Service와 Sun Microsystems의 JMS(Java Message System) 에 대하여 설명한다.

2.1 CORBA의 Event Notification Service

CORBA의 이벤트 서비스는 이벤트를 생성하는 역할을 하는 공급자(Supplier), 공급자로부터 생성된 이벤트를 받아 처리하는 소비자(Consumer), 그리고 공급자와 소비자의 통신 매개체 역할을 하는 이벤트 채널(Event Channel)로 구성된다. 이벤트 소비자는 관심있는 이벤트에 대한 통지를 채널에다 등록시킬 수 있으며 자신이 어떤 통신 모드로 동작할지를 지정할 수 있다. 통신 모드는 Push 방식과 Pull 방식으로 나눌 수 있다. Push모델에서는 공급자가 이벤트를 생성하면 생성된 이벤트를 소비자에게 전달한다. 소비자는 이벤트가 공급자로부터 전달되기를 수동적으로 기다린다. Pull모델에서는 소비자가 이벤트를 요구함으로써 이벤트 통신이 시작되며 Push모델과 정반대이다. 공급자와 소비자는 서로에 대한 정보 없이 이벤트를 전달할 수 있기 때문에 새로운 공급자나 소비자를 쉽게 시스템에 추가할 수 있다. 주로 전자통신 영역에서 사용하도록 개발되었으며 네트워크나 시스템 관리에는 적당하지 않기 때문에 CORBA의 Notification Service는 이벤트 서비스를 관리목적에 맞게 확장시킨 것이다. CORBA의 Notification Service는 Notification의 필터링(Filtering)과 서비스의 질(Quality of Service)를 보장함으로써 Event Service와 구분된다.

2.2 Sun의 JMS (Java Message System)

자바 메세지 서비스(JMS)는 Sun Microsystems에서 만든 기업환경(enterprise)의 메시징을 위한 API이다. JMS는 두가지 형태의 통신모델 발행/구독(publish-subscribe)과 지점간 연결(point-to-point)이 있다. 발행/구독에서 메시지를 생성하는 생산자는 토픽(topic)이라는 가상 채널을 통해 많은 소비자들에게 메시지를 자동적으로 브로드캐스트 해주는 큰 규모의 푸시 기반 모델(push-based model)이다. 생산자와 소비자간 결합력이 없다는 장점을 가진 구조이다. 지점간 연결 메시징 모델은 생산자가 큐(queue)라는 가상채널을 통해 동기적 또는 비동기적으로 메시지를 주고 받을 수 있도록 해주는 풀 또는 폴링 기반 모델(pull 또는 polling-base model)이다. JMS는 기업환경의 분산 어플리케이션의 소프트웨어 컴포넌트들 사이의 신뢰성있고, 비동기적이고, 확장성 있는 메시지 전달이 가능하도록 개발되었지만 JMS가 제공하는 기능을 이용하기 위해서는 JMS Provider를 구현한 메시징 제품을 이용해야 한다는 단점이 있다.

3. 이벤트 통지 서비스 프레임워크

서론 부분에서 설명한 일반성과 이벤트 타입의 확장성에 중점을 둔 자바기반의 이벤트 통지 서비스 프레임워크의 아키텍쳐와 설계 및 구현에 대해서 제시한다.

3.1 아키텍쳐

본 논문에서 제시한 이벤트 통지 서비스는 서비스 초기화 단계에서, Notification Target은 관심있는 이벤트에 대한 통지를 서비스(Notification Channel)에 등록할 수 있으며 통지 방식(Push 혹은 Pull 모드)을 지정할 수 있다. Push 모드는 이벤트 공급자가 이벤트를 생성하면 생성된 이벤트를 소비자에게 전달한다. 여기서는 소비자가 수동적으로 이벤트를 통지 받는 반면 Pull 모드에서는 소비자가 능동적으로 공급자가 발생하는 이벤트를 요구한다. 즉, 공급자가 이벤트를 발생시키면 바로 소비자에게 통지한다. 그림 1은 본 논문에서 제시한 이벤트 통지 서비스 프레임워크의 구조도이다. 아키텍쳐는 크게 다음과 같은 세 개의 컴포넌트로 나누어진다. 첫째로, Event Service는 이벤트를 발생시키려는 Event Source가 이벤트 발생 요청을 하면 해당하는 Event를 생성한 다음 필터링 한 후에 Event Dispatcher를 통해 이벤트의 정보를 Notification Service로 전달하는 역할을 수행한다. 둘째로, Notification Service

는 Event Service로부터 넘겨받은 이벤트를 필터링 한 후 Notification 수신인의 목록을 가진 Dispatcher가 채널을 통해서 Notification 수신인에게 통지하도록 하는 일을 수행한다. 마지막으로, Notification Channel 서비스는 이벤트 소스와 Notification Target 사이의 중재자 역할을 하며 이벤트를 통지할 때 몇 개의 채널을 열지, 통지 방식을 Push 혹은 Pull로 할지에 관한 역할을 수행하는 Channel Manager가 있다.

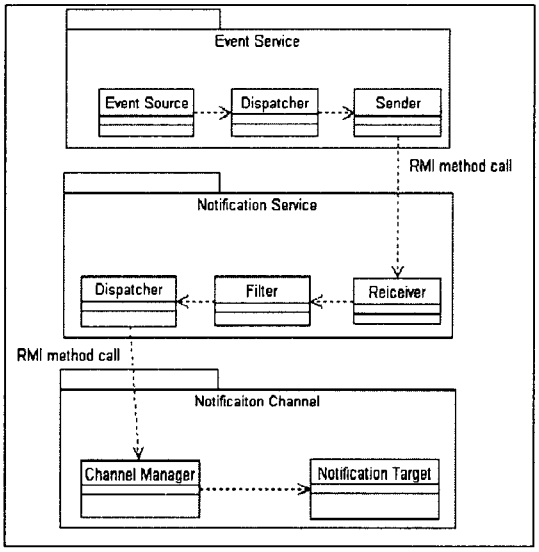

그림 1 Architecture of Event Notification Service Framework

3.2 설계 및 구현

여기서는, Event Service 컴포넌트의 설계에 대하여 좀 더 자세히 설명하고 앞으로 구현할 Notification Service의 구조를 제시한다. Event Service는 이벤트의 발생 요청을 받으면 그에 해당하는 이벤트를 발생시켜서 필터링의 과정을 거친 후 해당하는 이벤트에 관심있는 Event Target에게 이벤트를 전해주는 역할을 수행한다. Event Service는 그림 2와 같이 구성된다. 이벤트를 발생시키려는 Event Source는 MEventGenerator를 통해서 발생시키게 되는데 MEventGenerator는 자바의 Singleton 패턴을 이용해서 객체가 한번만 생성되도록 구현하였다. 생성된 이벤트는 이벤트 객체로 작동하게 되며 원하는 이벤트인지 아닌지를 여과하는 필터링 과정을 거치게 되는데 Filter를 인터페이스기반으로 정의하여 확장성을 강조하였다. 본 논문에서는 발생한 이벤트는 다 통과시키는 구조를 가지고 있는데 향후에는 이벤트 타입별로 필터링하는 기능[6]으로 향상되어야 할 것이다. 필터링 된 이벤트는 인터페이스로 정의되고 RMI 서버로 구현된 Dispatcher로 전달된다.

Dispatcher는 내부 벡터(vector)에 리스너(Listener)가 등록되어 있어서 이벤트가 들어오면 벡터에 등록된 해당 리스너 즉, 로컬 상의 Event Target에게 이벤트를 전달하거나 바로 MEvent Sender를 통해 리모트에 있는 Notification Service의 MEvent receiver에게 전달하게 된다. 앞으로 구현할 Notification Service의 구조는 그림 3과 같으며 각 컴포넌트의 역할은 다음과 같다.

- MEvent receiver: MEvent Sender로부터 이벤트 정보를 받는 역할을 수행한다.
- Notification Filter: MEvent receiver로부터 받은 이벤트를 이벤트 타입별로 필터링을 한다.
- Notification Encryptor: 암호화 알고리즘을 이용해서 이벤트를 암호화한다.
- Notification Remote Dispatcher: 통지 수신인의 목록을 가지고 있어서 해당하는 수신인에게 분배하는 일을 수행하며 통지는 로컬에 로그로써 저장될 수 있다.
- Notification Channel: 이벤트 통지시 중간매개체 역할을 하며 push, pull 통신방식 둘 다를 동시에 지원한다.
- Channel Manager: 채널의 관리자로서 채널을 몇 개 사용할 지에 관한 결정과 통지방식인 Push, Pull 모드의 관리에 관한 역할을 수행한다.
- Notification Listeners: 이벤트 통지의 수신인으로서 관심있는 이벤트를 통지 받을 수 있도록 Notificiation Channel에 등록할 수 있다.

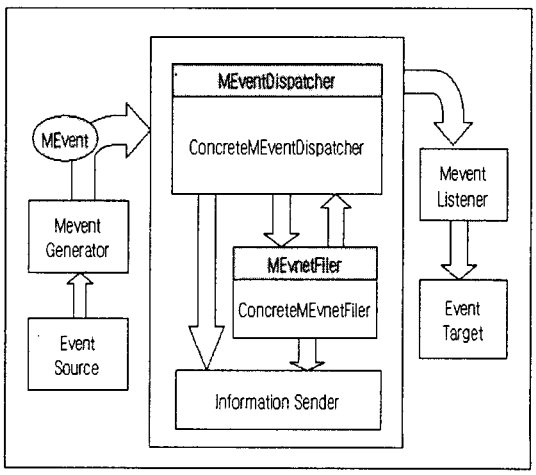

그림 2 Event Service Structure

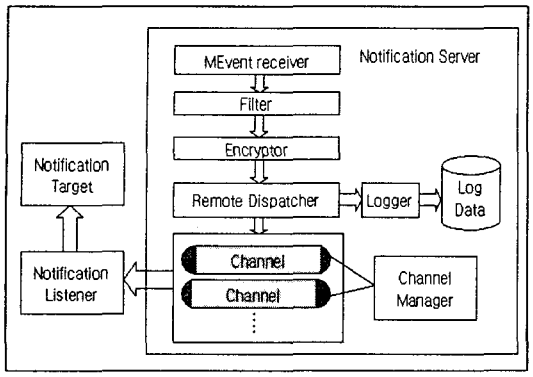

그림 3 Notification Service Structure

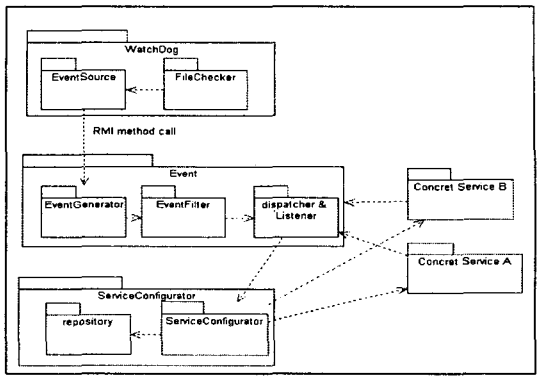

그림 4 Architecture of Dynamic Service Configuration Framework

4. 적용사례

여기서는 본 논문의 서론부분에서 설명한 동적 서비스 관리 프레임워크에 우리의 이벤트 서비스를 실제로 적용한 적용사례를 설명한 부분이다. 동적 서비스 관리 프레임워크는 분산환경에서 분산 어플리케이션을 개발 시 환경설정에 대한 변경을 실시간으로 탐지해서 동적으로 환경 설정을 해주는 역할을 하는데 각 서비스들 간에 이벤트 기반으로 통신을 할 수 있는 메카니즘이 필요하다. 그림 4은 동적 서비스 관리 프레임워크의 구조도이다. 아키텍쳐는 크게 다음과 같은 세 개의 컴포넌트로 구성된다. 첫째로, WatchDog 컴포넌트는 자신이 관리해야 할 환경 설정 파일에 대한 정보를 가지고 실시간으로 이 파일에 대해서 변경 여부를 주기적으로 검사해서 실제로 변경이 생겼을 때 이벤트의 종류 및 환경설정 파일과 같은 정보를 파라메터로 해서 이벤트 서비스의 EventGenerator를 통해서 이벤트를 발생시킨다. 둘째로, Event Service는 WatchDog과 같이 이벤트를 발생시키려는 Event Source에 의해서 EventGenerator를 통해서 해당 이벤트를 생성해서 정책에 따라서 필터링을 한 후에 Event Dispatcher를 통해서 특정 종류의 이벤트에 관심있는 Event Listener에게 이러한 이벤트를 넘겨주는 역할을 수행한다. 실제로 이는 하나의 어플리케이션 안에서 서비스들 사이의 통신 메커니즘을 제공하는 Event Coordinator로서의 역할이다. 마지막으로 Service Configurator는 서비스의 환경설정에 관련된 다양한 이벤트에 대한 처리를 수행하는 컴포넌트이며 Repository에 실제 구동되고 있는 서비스에 대한 모든 리스트를 가지고 있다. Concrete Service는 실제로 시스템이나 어플리케이션 안에서 사용되고 있는 다양한 종류의 서비스를 말한다.

5. 결론 및 향후 연구

본 논문은 자바기반의 다양한 분산 시스템에 적용될 수 있는 일반적인 이벤트 통지 서비스 프레임워크를 제시하였고 응용 사례로 분산 서버들의 동적 형상 관리 서비스에 적용해 보았다. 향후에는 다양한 이벤트 처리 모듈을 지원하는 Customizable할 수 있는 프레임워크로 바꿀 예정이다.

6. 참고 문헌

[1] Object Management Group, "The Common Object Request Broker: Architecture and Specification, Revision 2.2

[2] Sun Microsystems, "Java Message Service" http://www.javasoft.com/products/jms/docs.html

[3] Sun Microsystems, "JMX(Java Management extension)/JDMK(Java Dynamic Management Toolki)" http://java.sun.com/products/jmx/docs.html

[4] M. TOMONO, "Event Notification: An Event Notification Framework based on Java and CORBA," IEEE, 2000

[5] J. Oueichek and X. Rousset de Pina, Dynamic Configuration Management in the Guide Object-oriented Distributed System , Proc. of 3rd IEEE Intl. Conf. On Configurable Distributed Systems, Annapolis, pp. 28-35,May 1996

[6] M. Malowidzki, "Event Filtering: Advanced Event Filtering Approach for CORBA-Based Management System," IEEE, 2000

[7] I. W. Marshall, H. Gharib, J. HardWicke, C. RoadKnight, "A novel architecture for active service management," IEEE, 2001

[8] Luis Felipse Cabrera, Michael B. Jones, Marvin THeimer, "Herald: Achieving a Global Event Notification Service," IEEE, 2001

[9] G. Cugola, E. Di Nitto, A. Fuggetta, "Exploiting an event-based infrastructure to, develop complex distributed systems

[10] Peter R., Pietzuch, Jean M. Bacon, "Hermes: A Distributed Event-Based Middleware Architecture,"