효율적인 이벤트 데이터 전송을 위한 이벤트 필터링 채널의 설계

채영진[0] 노희영
강원대학교 컴퓨터과학
yjchae@mirae.kangwon.ac.kr, young@cc.kangwon.ac.kr

A Design of Event Filtering Channel for Efficient Event Data Delivery

Young-Jin Chae[0] Hee-Young Roh
Dept. of Computer Science, Kangwon National University

요 약

현재 클라이언트/서버 네트워크 환경은 시스템이 점점 대형화되면서 유지보수 및 네트워크 트래픽에 대한 문제로 인하여 분산 객체 개념을 이용한 분산 환경의 패러다임이 필요하게 되었다. 이에 OMG의 CORBA는 분산 환경을 위한 표준을 제공하며, 분산 객체간의 비동기 호출을 위한 이벤트 서비스를 명시하고 있다. 분산 객체간의 데이터 전송은 전체 시스템의 성능을 좌우하는 중요한 요인으로 평가받고 있으나 이벤트 서비스는 이벤트 데이터를 위한 필터링 기능을 제공하고 있지 않아 비효율적인 데이터 전송을 하고 있다. 본 논문에서는 이벤트 채널의 필터링을 위한 모듈을 정의하여 기존의 서비스에서 제공하는 이벤트 채널보다 효율적인 데이터 전송이 가능한 이벤트 필터링 채널을 설계한다.

1. 서 론

분산 컴퓨팅 환경은 독립적인 많은 컴퓨터를 상호 연결하여 요구되는 정보에 대한 처리를 최대한 만족시켜줄 수 있으며, 기본 구조는 중간 계층인 미들웨어를 두어, 클라이언트와 서버 사이에서 네트워크와 데이터베이스의 일관성 있는 접근 등 다양한 서비스와 투명성을 제공한다.

분산 환경을 위한 OMG의 CORBA는 객체 관리 기술을 기반으로 한 이기종의 분산 환경을 위한 표준이며, 미들웨어인 ORB에 의해서 클라이언트는 원격 객체의 위치와는 관계없이 서버의 구현객체에 접근하여, 메소드를 호출할 수 있도록 한다. 또한 원격 객체의 관리를 위한 계층 구조의 네이밍 서비스를 제공하고, 비동기 호출을 위한 이벤트 서비스를 제공한다.

이벤트 서비스는 서버와 클라이언트의 더욱 분리된 객체간의 통신 모델과 이벤트 채널을 제공하여 효율적인 전송을 가능하게 한다.

이벤트 채널은 실제 이벤트 데이터를 생성하는 공급자(supplier)와 이벤트 데이터를 받아서 처리하는 소비자(consumer)와의 중계자 역할을 하며, 시스템의 전체의 성능은 이벤트 채널의 성능에 상당히 의존하고 있다.

그러나, 이벤트 서비스는 단일 또는 다수의 공급자가 이벤트 채널에 연결되어 이벤트 데이터를 전송할 때 이벤트 채널에 연결된 모든 공급자의 이벤트 데이터가 모든 소비자에게 전달 되게 된다. 이것은 이벤트 데이터와 관련이 없는 소비자에게도 전송하게 되어 네트워크의 트래픽을 증가시키고 이벤트 채널에 대한 리소스 낭비를 가져오는 문제점을 가지고있다.

따라서, 공급자의 이벤트 데이터가 관련이 있는 소비자에게만 데이터를 전송할 수 있는 필터링 기능이 필요하게되었다.

본 논문에서는 이벤트 채널이 필터링 기능을 수행할 수 있는 모듈을 설계하여 효율적인 이벤트 데이터 전송을 위한 이벤트 필터링 채널을 설계한다.

본 논문의 구성은 다음과 같다. 2장에서는 비동기 호출 방법인 콜백과 이벤트 서비스 그리고 이벤트 채널에 대해서 설명한다. 3장에서는 이벤트 필터링을 위한 방법과 이벤트 필터링 채널을 설계하고, 마지막 4장에서는 결론 및 향후 연구과제에 대하여 설명한다.

2. 관련 연구

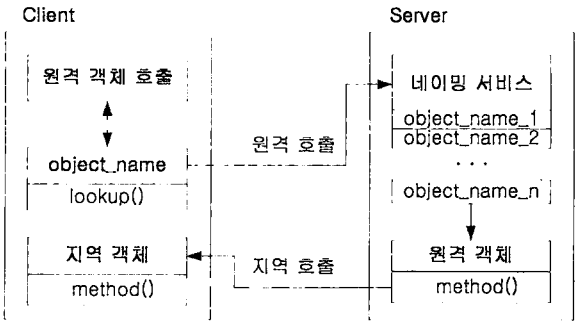
2.1 비동기 호출

분산된 객체간의 통신을 위한 일반적인 방법으로 동기 호출을 사용한다. 동기 호출은 클라이언트가 서버에게 요구를 하고, 서버는 그 요구에 대한 결과를 응답한다. 클라이언트는 서버에서 응답을 받을 동안 블록킹(blocking) 상태가 된다. 이러한 경우 시스템에서 심각한 장애를 발생하게 될 수 있기 때문에 클라이언트와 서버간의 비동기 호출이 필요하게 되었다. 이번 장에서는 비동기 호출을 가능하게 하는 콜백과 이벤트 서비스에 대해서 설명한다.

2.1.1 콜백(callback)

콜백은 비동기 호출을 제공하는 가장 일반적인 방법으로, 클라이언트가 보낸 요구에 대한 결과로서 서버에서 클라이언트에게 다시 요구 메시지를 보내는 것이다.

아래의 [그림 2.1]은 콜백의 구조를 나타내고 있다.



【그림 2.1】 callback 호출 구조

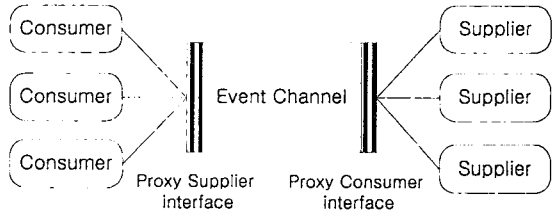
콜백은 많은 요구 메시지가 들어오면 메시지를 폴링하게 되며, 폴링은 각각의 요구 메시지에 대한 프로세스와 메모리가 소모하게 된다. 또한 폴링에 대한 핸들러가 요구되기 때문에 어플리케이션을 확장하는데 큰 어려움이 있어 CORBA 시스템이 제공하는 이벤트 서비스의 통신 모델을 사용함으로써 이벤트 기반의 복잡한 메시지 전송을 효율적으로 할 수 있다.

2.1.2 이벤트 전송 모델 및 이벤트 채널

이벤트 서비스는 어플리케이션이 엄격히 연결된 클라이언트/서버 호출보다 더욱 분리된 통신 모델을 제공함으로써 이벤트 메시지의 전송을 효율적으로 전송하도록 한다. 이벤트 서비스는 이벤트 데이터를 생성하는 공급자(supplier)와 이벤트를 받아서 처리하는 소비자(consumer)를 기반으로 하는 모델이며, 공급자로부터 소비자에게 이벤트 데이터를 전달하는 중추적인 역할을 하는 것이 이벤트 채널이다.

이벤트 데이터의 전송을 위한 push 모델과 pull 모델을 제공하며, push 모델은 공급자가 이벤트 채널에게 이벤트를 전송하고, 이벤트 채널은 소비자에게 이벤트를 전송한다. pull 모델은 push 모델과는 반대로 소비자가 이벤트 채널로부터 이벤트 데이터를 얻게 되고, 이벤트 채널은 공급자로부터 이벤트를 얻는다.

이벤트 채널은 실제 공급자와 소비자와의 상호 작용과 논리적인 연결을 위해서 이벤트 채널 자체에 프록시라는 인터페이스를 가지고 있다.



【그림 2.2】 이벤트 채널 인터페이스

분산 객체간의 데이터 전송은 실제 이벤트 채널에 의하여 이루어지고, 이벤트 채널이 응용 프로그램의 성능에 심각한 영향을 준다.

그러나, 이벤트 서비스는 단일 공급자 또는 다수의 공급자가 이벤트 채널에 연결된 경우 공급자에서 발생은 모든 이벤트 데이터가 채널에 연결된 모든 소비자에게 이벤트 데이터가 전송되는 문제가 발생한다. 이것은 공급자에서 발생한 이벤트 데이터가 이와는 관련이 없는 소비자에게 전송되어 네트워크 트래픽을 증가시키고, 이벤트 채널의 리소스 낭비를 가져오게 한다.

소비자에서 발생한 모든 이벤트 데이터를 관련된 소비자에게만 전송할 수 있는 이벤트 필터링 기능이 요구된다.

통지 서비스는 이벤트 서비스에서 제공하지 않는 이벤트 필터링과 다양한 QoS를 정의하고 있으며, 이벤트 서비스의 호환성을 위해서 이벤트 서비스에 정의된 모듈을 사용하고 있다. 그러나 통지 서비스는 이벤트 필터링과 다양한 QoS를 위한 이벤트 타입과 이벤트의 우선순위 등 많은 기능을 포함하고, 제한(constraint)의 복잡하며, 실제 서비스에서 사용하지 않는 기능까지 포함하고 있어 효율적인 이벤트 채널의 구현을 어렵게 하고 있다.

따라서, 복잡하고 사용하기 어려운 통지 서비스의 이벤트 필터링 기능을 이벤트 서비스에서 사용할 수 있는 이벤트 필터링을 위한 이벤트 채널을 설계한다.

3. 시스템 설계

CORBA에서 분산 되어있는 객체간 데이터의 전송은 원격 객체로 등록 된 인터페이스에 의하여 전송이 일어난다. 클라이언트는 서버와 통신하기 위한 서버의 인터페이스를 가지고 원격 객체의 레퍼런스를 얻으며, 서버는 클라이언트에게 데이터를 전송하기 위해서 클라이언트의 인터페이스를 가지고 있어야 한다.

3.1 이벤트필터 모듈 설계

공급자는 발생한 이벤트를 이벤트 채널과 연결하고 전송하기 위한 이벤트 채널의 인터페이스를 가지고 있으며, 채널은 공급자와 소비자 모두의 인터페이스를 가지고 있다.

기존의 이벤트 서비스에서 제공하는 모듈을 사용하고 새로운 이벤트 필터링을 위한 EventFilter 모듈을 정의한다.

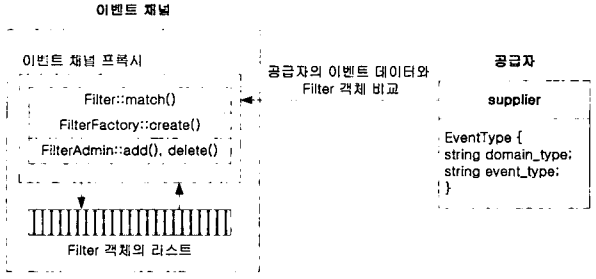
```
module EventFilter {
        struct FilterType {
                string domain_type;
                string event_type;
        };
        typedef sequence<FilterType> FilterTypeSeq;
        interface Filter {
                boolean match();
        }; // 이벤트 객체와 비교하기 위한 인터페이스
        sequence <Filter>
        interface FilterFactory {
                Filter create();
        }; // Filter 객체를 생성하기 위한 인터페이스
        interface FilterAdmin {
                long add();
                void remove();
                Filter get();
        }; // 채널의 Filter 리스트를 위한 인터페이스
};
```

이벤트 채널은 Filter 인터페이스를 정의하고 있으며, 공급자로부터 전송된 이벤트 데이터를 필터링하기 위한 match 메소드를 가진다.

또한 Filter 객체를 생성하기 위한 FilterFactory 인터페이스와 공급자의 이벤트 데이터와 비교하여 필터링 하는 Filter 리스트를 관리하기 위한 FilterAdmin 인터페이스를 정의한다.

Filter 인터페이스는 필터링 매칭을 위한 domain_type과 event_type 인스턴스를 가지고 있으며, FilterAdmin 인터페이스는 Filter 리스트를 검색하고, 리스트에 Filter 객체를 추가, 삭제하는 오퍼레이션을 정의한다.

공급자는 발생한 이벤트를 소비자에게 전달하기 위해 이벤트 채널과 연결하고, 이벤트 채널은 전송된 이벤트를 필터링 하여 이벤트 데이터와 관련 있는 소비자로 이벤트 데이터를 전송한다.



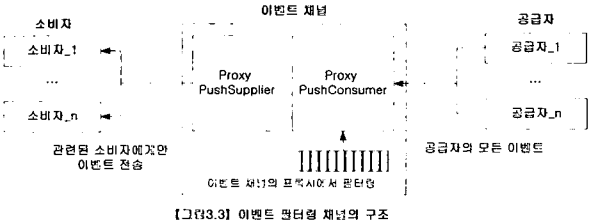
【그림3.2】 필터링을 위한 이벤트 프록시

3.2 이벤트 필터링 채널의 설계

이벤트 필터링 채널은 실제 이벤트 데이터를 프록시를 통하여 관련된 이벤트를 전송하며, 프록시 인터페이스는 이벤트를 필터링하기 위한 필터 객체 리스트를 가지고 있다. 필터 인터페이스의 인스턴스는 프록시 인터페이스가 생성될 때 만들어지며, 필터 객체가 생성될 때 그 객체가 이벤트 필터링 할 수 있는 정보를 가지고 있다.

필터링을 위한 객체 인스턴스는 이벤트 데이터가 전송되어야 하는 소비자를 나타내는 domain_type 인스턴스와 이벤트의 종류를 나타내는 event_type 인스턴스 두 개를 가진다.

이벤트 채널의 필터 객체는 이벤트를 비교할 수 있는 match 연산을 정의하며, domain_type과 event_type의 두 인스턴스를 전송된 이벤트와 비교하여 다른 곳으로 포워딩 하는 이벤트를 결정하여 필터링 한다.



【그림3.3】 이벤트 필터링 채널의 구조

필터링 가능한 이벤트 채널은 위의 【그림3.3】에 나타내고 있으며, 이것은 이벤트 채널이 하나의 공급자나 또는 여러 공급자에게 연결이 되어있더라도 공급자에서 발생한 이벤트는 관련된 소비자에게로만 전송할 수 있도록 설계한 것이다.

4. 결론 및 향후 연구과제

본 논문에서는 분산 객체간의 데이터 전송을 효율적으로 하기 위해서 필터링이 가능한 이벤트 채널을 설계하였다. 필터링 기능을 제공하지 않는 이벤트 채널은 서버에서 생긴 모든 이벤트들이 관련이 없는 클라이언트까지 이벤트 데이터가 전송되는 단점을 가지고 있어 이벤트 전송에 대하여 비효율적이었다.

이벤트 채널은 이벤트 기반의 시스템에서 시스템 전체의 성능을 좌우할 수 있는 중요한 역할을 담당하며, 이러한 이벤트 채널을 필터링이 가능한 이벤트 채널로 설계함으로서 이벤트 채널의 리소스 낭비와 네트워크의 트래픽을 감소시키는 효율적인 전송을 가능하게 한다.

또한, 여러 공급자와 소비자를 갖는 다수의 이벤트 채널로의 확장이 가능하여, 대규모 시스템에서 분산 객체간의 데이터 전송 효율을 최대한 높일 수 있다.

본 논문에서 설계한 이벤트 필터링 채널을 사용함으로서 네트워크와 시스템 자원에 대하여 더욱 효율적인 이벤트 시스템을 가능하게 한다.

향후 연구과제로는 공급자에서 발생되는 이벤트 데이터에 대한 상세한 모듈과 이벤트 전송에 따른 효율적인 QoS를 위한 연구가 필요하다.

참고 문헌

[1] Kurt Wallnau 외, "Distributed Object Technology with CORBA and Java". Technical Report. 1997

[2] Halldor Fossa, Morris Sloman, "Interactive Configuration Management for Distributed Object Systems", IEEE Proc. 1997

[3] Stephen Crane, Naranker Dulay, "A Configurable Protocol Architecture for CORBA Environments", 1997

[4] Douglas C 외, "A High-performance Endsystem Architecture for Real-time CORBA", Distributed Object Computing in IEEE Communications, 1997

[5] OMG. 2001. the Common Object Request Broker Architecture. Revision 2.4.2

http://www.omg.org

[6] OMG. 2001. Event Service Specification. Version 1.1, http://www.omg.org

[7] OMG. 2000. Notification Service Specification. Version 1.0, http://www.omg.org

[8] Document formal/01-09-34 (CORBA 2.5 Specification - supersedes formal/01-02-33