uniORB 상에서의 Event Service 구현 및 Notification Service 설계

윤교철[0*], 박성우*, 황선태*, 김영만*,
이동길**, 백의현**, 장종현**
국민대학교 컴퓨터학부*, 한국전자통신연구원 교환전송기술연구소 개방형 플랫폼 팀**
charlie@cs.kookmin.ac.kr[0*]

Implementation of Event Service and Design of Notification Service Based on the uniORB

Kyo Chul Yoon[0*], Sung Woo Park*, Suntae Hwang*, Young Man Kim*,
Dong Gil Lee**, Eui Hyun Paik**, Jong Hyun Jang**
School of Computer Science, Kookmin University*,
ETRI**

요 약

본 논문에서는 실시간 시스템인 전화 교환기에서 운영되는 C 언어 기반 분산 처리 시스템인 uniORB 상에서 이벤트 전송을 담당하는 서비스인 Event Service를 구현하였으며 다양한 이벤트의 타입과 필터링 기능, QoS 를 제공하는 Notification Service에 대한 실시간 버전의 설계를 하였다.

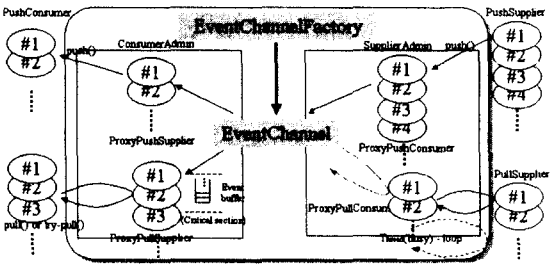
1. 서 론

기존의 전화 교환기에 있어서는 CHILL (CCITT High Level Language)[1]을 사용하여 사용자의 요구를 처리할 수 있도록 하였다. 그러나 다양한 사용자의 요구 처리와 확장성 등 요구 사항이 늘어남으로 인해 분산 처리 시스템이 요구 되었으며 대표적인 표준 안으로서 CORBA[2]가 사용되기 시작하고 있다. 일반적으로 CORBA 플랫폼상의 응용프로그램은 JAVA, C++ 언어 등을 통해 작성되고 있지만 속도를 중요시하는 전화 교환기에서는 C 언어를 사용한 CORBA 플랫폼(예를 들면, ORBit)을 사용하는 것이 필요하며 SDL(Specification & Description Language)도 C 언어 기반으로 운영되고 있기 때문에 ORBit에 SDL을 탑재한 분산 처리 시스템인 uniORB를 구현한 바 있으며 본 논문에서는 이벤트의 전송을 담당하는 CORBA 표준 서비스인 Event Service[3]를 구현하고 또한 다양한 이벤트의 타입과 필터링 기능 및 QoS를 제공하는 Notification Service[4]를 설계한다.

2. Event Service 구현

Event Service 는 분산 처리 시스템인 CORBA 규격내의 표준 서비스로서 이벤트의 공급을 담당하는 Supplier 와 이벤트를 소비하는 Consumer 가 있으며 이벤트의 제공 및 소비하는 방식에 따라 Push, try_Pull(non-block), Pull(block) 의 형태가 있다.

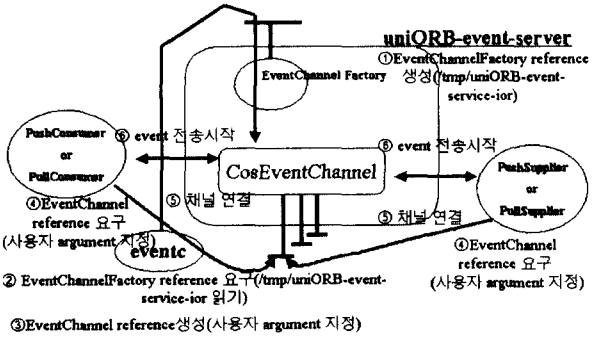
[그림 1]은 uniORB 상에서 구현한 uniORB 이벤트 서비스의 구조를 보여주고 있다.



[그림 1. uniORB 이벤트 서비스의 구조]

uniORB 이벤트 서비스의 구조[그림1]에서는 표준 이벤트 서비스의 구조를 만족하도록 구현하였다. 양쪽에는 어플리케이션 프로그램으로 오른쪽에는 이벤트의 공급을 담당하는 PushSupplier 와 PullSupplier를 구현했으며 왼쪽에는 이벤트의 소비를 담당하는 PushConsumer 와 PullConsumer를 구현했다. 중앙에는 이벤트의 전송을 담당하는 이벤트 서비스 서버가 존재하며 이벤트 채널에는 각 Supplier 와 Consumer의 연결을 담당하는 Proxy 객체가 있고 각 Proxy 의 생성 및 제거를 담당하는 Admin 객체가 있다. 이벤트 서버는 이벤트 채널 팩토리를 생성하고 생성된 오브젝트를 참조하여 이벤트 채널을 생성, Supplier 와 Consumer의 연결 및 제거, 이벤트 전송을 담당하도록 구현하였다.

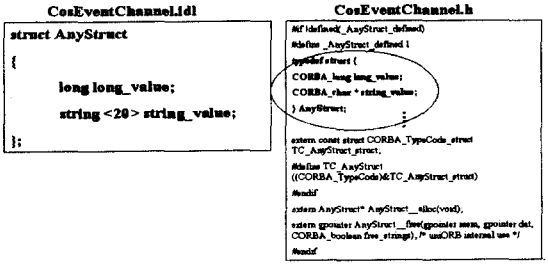
[그림 2]는 uniORB 이벤트 서비스를 구현한 프로그램 관계도이다.



[그림 2. uniORB 이벤트 서비스 프로그램 관계도]

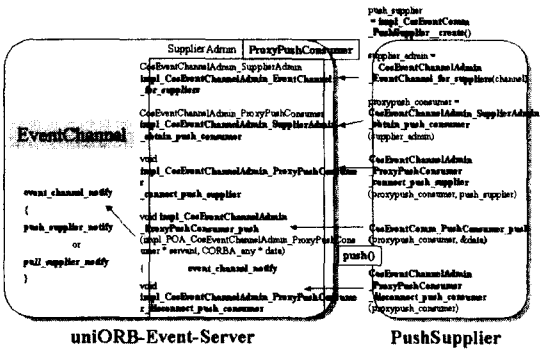
uniORB 이벤트 서비스 프로그램의 실행 순서를 보면, ① uniORB-event-server 는 이벤트 채널 팩토리 레퍼런스를 생성 및 저장을 하고 대기 상태로 실행된다. ② 다양한 이벤트 채널을 생성하기 위해서 eventc 프로그램은 이벤트 채널 팩토리 레퍼런스를 읽어 들인다. ③ 읽어 들인 채널 팩토리 레퍼런스를 통해 사용자로부터 입력 받은 값으로 이벤트 채널 레퍼런스를 생성하여 저장한 후 종료된다.④ 이벤트의 공급 및 소비를 하는 Supplier 와 Consumer 는 eventc 프로그램이 저장한 이벤트 채널 레퍼런스를 읽어들인다. ⑤읽어들인 이벤트 채널 레퍼런스를 통해 uniORB-event-server에게 채널 연결을 한다. ⑥연결이 완료되면 이벤트를 공급하거나 소비한다.

[그림 3] 은 이벤트 서비스에서 사용되는 Any Type 의 이벤트를 IDL 내에 정의하고 IDL 컴파일을 통해 생성된 코드를 보여주고 있다.



[그림 3. 이벤트 서비스에서 사용되는 Any Type]

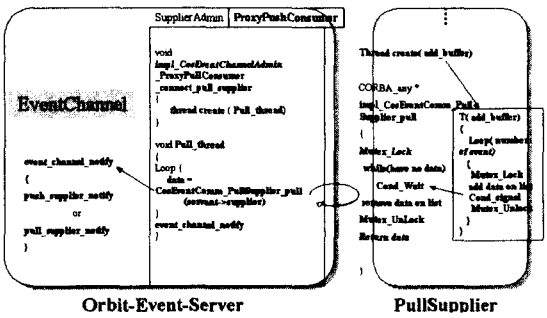
[그림 4] 는 PushSupplier 와 uniORB-Event-Server 의 구현 코드로서 PushSupplier 가 uniORB-Event-Server 에게 연결요청 , 이벤트 공급(Push), 연결 해제에 관한 함수 맵핑과 흐름을 나타낸다. PushSupplier 는 사용자가 입력한 이벤트의 수에 따라 이벤트를 생성하여 uniORB-Event-Server 에게 Push 함수를 통해 Any 타입의 이벤트를 공급한다. 공급된 이벤트는 uniORB-Event-Server 와 연결된 Consumer 의 형태에 따라 다른 형태의 이벤트 전달이 이루어지는데 PushConsumer 가 연결 된 경우에는 내부함수인 event-channel-notify 내의 push_supplier_notify 함수를 호출한 후 PushConsumer 의 Push 함수를 호출하여 이벤트를 전달하고 PullConsumer 가 연결되어 있을 경우에는 pull_supplier_notify 함수를 호출하여 각 Proxy 내의 이벤트 버퍼에 이벤트를 저장하면 PullConsumer 의 try_Pull(nonblock) 이나 Pull(block) 함수의 요청에 따라 ProxyPullSupplier는 이벤트 버퍼에서 이벤트를 전달하고 버퍼 내의 이벤트를 제거한다.



[그림 4. uniORB-Event-Server 와 PushSupplier 구현코드]

uniORB-Event-Server 와 Supplier, Consumer 의 연결 과정은 표준 안에 제시된 함수를 적용하여 모두 동일한 형태의 함수를 호출하도록 구현했으며 단지 이벤트의 공급 및 소비에 관한 함수만을 다른 형태로 구현하였다.

[그림 5]는 이벤트를 Pull 형태로 제공하는 PullSupplier 와 uniORB-Event-Server를 구현한 코드이다.

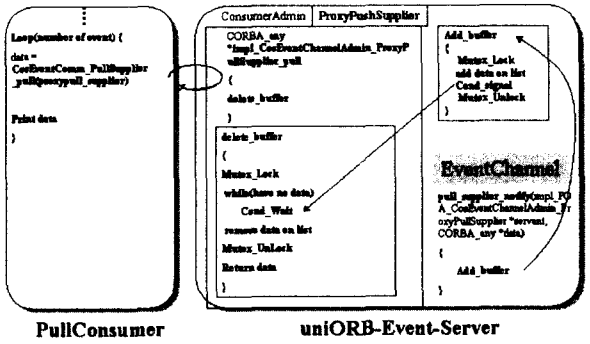


[그림 5. uniORB-Event-Server 와 PullSupplier 구현코드]

Push 형태의 이벤트 공급과는 달리 Pull 형태의 이벤트 공급은 uniORB-Event-Server가 PullSupplier에게 이벤트를 요청한다. 그러므로 PullSupplier가 uniORB-Event-Server 에게 연결 요청 시 uniORB-Event-Server 는 PullSupplier로부터 이벤트를 가져 올 수 있도록 Thread (Pull_thread)를 생성하여 구현하였으며 PullSupplier 는 이벤트를 공급하도록 Thread(add_buffer)를 생성하여 사용자의 제공 이벤트 수만큼 공급을 한다. 이후 이벤트의 전달은 PushSupplier 와 동일하게 각 Consumer에게 전달된다. 이벤트 공급이 끝나면 마지막 이벤트 다음으로 이벤트 공급의 끝을 알리는 특정이벤트(DISCONNECT)를 버퍼에 넣는다. uniORB-Event-Server는 Thread(Pull_thread)를 통해 PullSupplier로부터 Pull(block) 함수를 통해 이벤트를 가지고 오고 특정이벤트(DISCONNECT)를 받으면 PullSupplier 에게 disconnect 요청을 하고 자신의 Proxy 객체를 제거한 후 Pull_thread 는 종료된다. Nonblock 함수인 try_Pull 또한 Pull 과 동일한 방식으로 구현하였다.

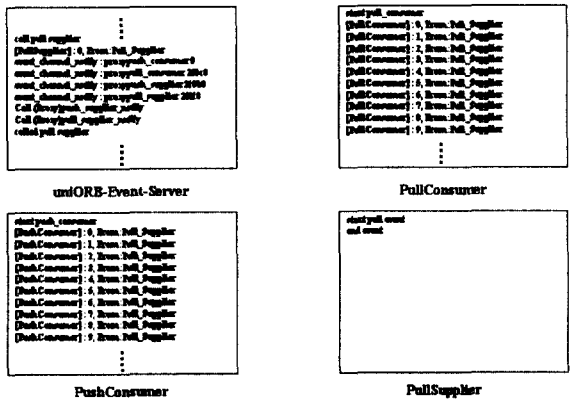
[그림 6]은 PullConsumer 와 uniORB-Event-Server 의 구현 코드로서 [그림 1]에서 보는 바와 같이 PullConsumer 와 연결된 ProxyPullSupplier 는 Supplier 측으로부터 전달된 이벤트를 저장하는 버퍼가 필요하며 PullConsumer 의 요청에 따라 이벤트

버퍼에서 이벤트를 가져가고 버퍼 내의 이벤트를 제거한다. 따라서 ProxyPullSupplier 의 이벤트 버퍼는 Supplier 의 이벤트 공급과 PullConsumer 의 이벤트 소비가 이루어지므로 동기화가 필요하게 된다. 따라서 내부 함수인 Add_buffer 와 delete_buffer 를 통해 이벤트의 추가 및 삭제에 대한 동기화를 위하여 모니터를 사용하여 구현하였다



[그림 6. PullConsumer 와 uniORB-Event-Server 구현코드]

[그림 7] 은 uniORB Event Service 의 구현 결과로서 uniORB-Event-Server, PullConsumer, PushConsumer, PullSupplier 가 실행되고 있는 일부분을 나타낸 것이다.



[그림 7. uniORB Event Service 구현 결과]

uniORB-Event-Server에서 나타내는 결과는 PullSupplier에게 이벤트를 요청하도록 함수를 호출하고 받은 이벤트의 결과를 나타낸 후 현재 Supplier 와 Consumer에 연결된 각각의 Proxy를 표시하고 연결된 Consumer 들에게 이벤트를 전달하는 과정을 나타낸다.

PullConsumer 와 PushConsumer 측은 현재 자신이 PullConsumer, PushConsumer 임을 나타내고 이벤트의 순서와 PullSupplier로부터 이벤트가 공급되었음을 나타낸다

PullSupplier 측은 이벤트의 순서와 "From:Pull_Supplier"를 Any Type 의 스트링에 담아 이벤트를 uniORB-Event-Server 에게 공급하며 이벤트의 공급의 시작과 끝을 나타낸다.

3. Notification Service 설계

Notification Service도 또한 분산 처리 시스템인 CORBA 규격내의 표준 서비스로서 Event Service 의 모든 기능을 포함하며 다양한 이벤트의 타입과 필터링 기능 및 QoS 를 제공하는 Service 이다.

본 논문에서는 Notification Service 을 설계하는데 있어서 모든 표준을 만족하는 것이 아니라 속도가 가장 중요한 전화 교환기에서 사용될 수 있는 최소한의 서비스만을 제공하도록 설계한다.

(1) Type 의 결정

uniORB-Event-Service에서 제공되는 Any 타입의 이벤트를 만족하도록 하고 또한 Consumer 측에서 흥미 있는 이벤트만을 받을 수 있도록 하거나 원하는 이벤트 만을 전달할 수 있도록 하는 필터링 지원을 위한 Structed 타입, 한 번의 호출을 통해서 이벤트의 리스트를 전송할 수 있는 Sequence 타입이 있으며 전화교환기에 사용되어질 이벤트 타입을 결정하도록 한다.

(2) Filtering 기능

Notification Service 의 가장 큰 특징 중 하나인 필터링 기능은 Consumer 측에서는 Supplier 의 모든 이벤트를 전달 받지 않고 흥미 있는 이벤트만을 전달 받을 수 있도록 하거나 서버 측에서도 Supplier 에게 요구되는 이벤트만을 공급할 수 있도록 하는 등의 필터링을 통해서 이벤트 전송 효율을 높일 수 있도록 한다.

(3) QoS(Quality of Service)

Notification Service 내에는 많은 QoS가 정의 되어 있으나 각 이벤트의 가중치(Prioroty)를 주어 이벤트의 전달 순서만을 결정할 수 있도록 제한한다.

(4) 어플리케이션 레벨에서의 thread 생성

uniORB Event-Service 의 uniORB-Event-Server 와 PullSupplier 의 관계와 같이 Notification Service 또한 어플리케이션 레벨에서 여러 개의 필수 thread 생성하게 된다. 그러나 시스템 오버헤드를 고려하여 필요한 thread 수를 최소화하도록 설계한다.

4. 결론

속도가 중요시되는 전화 교환기에 적용되는 분산 처리 시스템인 uniORB 는 C 코드로 작성이 되어있으며 여기에 이벤트 전달 서비스인 Event Service를 구현하였다. 차후 위에 제시된 Notification Service 설계에 따라 속도를 고려하여 Notification Server를 구현할 것이다.

5. 참고문헌

[1] ISO, *ITU-T Recommendation Z.200* (1999.11)
[2] OMG, *CORBA Specification V2.4.2* (2001.02.01)
[3] OMG, *Event Service Specification V1.0* (2000.06.15)
[4] OMG, *Notification Service Specification V1.0* (2000.06.20)
[5] OMG,*Common Object Services Specification V1.0* (1994. 03. 01)
[6] Douglas C.Schmidt, Steve Vinoski, *Object Interconnections, The OMG Events Service (Column 9)* (1997. 02. 01)
[7] Object-Oriented Concepts,Inc. *ORBacus Notify V1.0.1* (2001. 01. 04)