uniORB 상에서의 Notification Service 설계 및 구현

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

Design and Implementation 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 Service 의 기능을 보완하고 다양한 이벤트의 타입과 필터링 기능, QoS 를 제공하는 Notification Service에 대한 실시간 버전의 설계 및 구현을 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][8]를 바탕으로 하여 다양한 이벤트의 타입과 필터링 기능 및 QoS를 추가제공하는 Notification Service[4]의 설계 및 구현을 하고자 한다.

2. Notification Service 설계

Notification Service는 Event Service와 같이 분산 처리 시스템인 CORBA 규격내의 표준 서비스로서 기본적으로 Event Service 의 모든 서비스를 포함하며 고급서비스로서 다양한 이벤트의 타입과 필터링 기능 및 QoS 를 제공하는 Service가 있다.

본 논문에서는 Notification Service표준에 정의된 기능중 전화 교환기의 처리속도를 크게 저하시키지 아니하고 사용될 수 있는 서비스들을 제공하도록 설계 및 구현하였다.

2.1 Type 의 결정

개발된 Notification Service는 세가지 이벤트 타입을 지원하고 있다. 우선, 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 Structured 타입, 한번의 호출을 통해서 이벤트의 리스트를 전송할 수 있는 Sequence 타입이 있다.

2.2 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 Supplier 측에서도 Client에게 요구되는 이벤트만을 공급할 수 있도록 하는 등의 필터링을 통해서 이벤트 전송 효율을 높일 수 있도록 하였다.

2.3 어플리케이션 레벨에서의 thread 생성

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

3. Notification Service 구현

Notification Service는 Event Service와 같이 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-blocking), Pull(blocking) 의 형태가 있다.

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

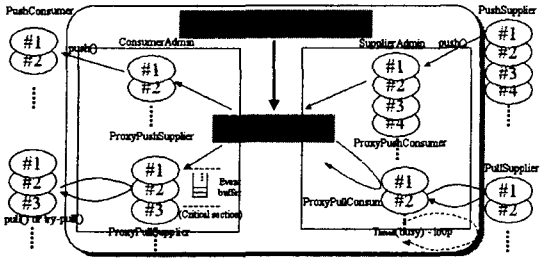

그림 1 uniORB Notification 서비스의 이벤트 전송 모델 구조

uniORB Notification 서비스의 이벤트 전송 모델 구조[그림1]에서는 표준 Notification 서비스의 이벤트 전송 모델 구조를 만족하도록 구현하였다. 양쪽에는 어플리케이션 프로그램으로 오른쪽에는 이벤트의 공급을 담당하는 PushSupplier 와 PullSupplier 를 구현했으며 왼쪽에는 이벤트의 소비를 담당하는 PushConsumer 와 PullConsumer 를 구현했다. 중앙에는 이벤트의 전송을 담당하는 Notification 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 채널 팩토리 객체를 참조하여 이벤트 채널을 생성하여 Supplier 와 Consumer의 연결 및 제거, 이벤트 전송을 담당하도록 구현하였다.

[그림 2]는 uniORB Notification 서비스에서 지원하는 Type 들을 나타내는 구조도이다.

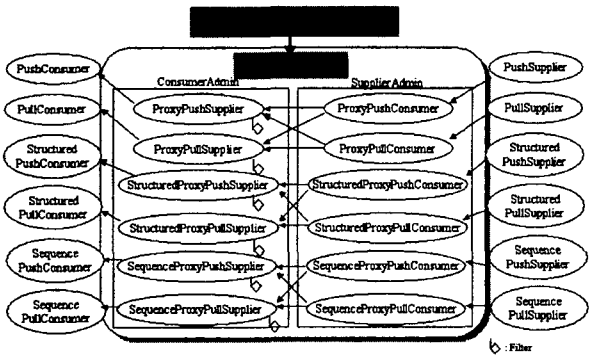

그림 2 Notification 서비스에서 지원하는 Type 구조도

[그림 2]에서 나타내는 바와 같이 Notification 서비스에서는 Event 서비스에서 정의된 Any 형 타입의 이벤트 외에도 2.1 에서 소개 한 바와 같이 Structured 타입, Sequence 타입을 지원하도록 구현하였다. 또한 2.2 에서 소개 한 바대로 Notification 서비스의 가장 큰 기능 중 하나인 필터링 기능을 추가하여 구현하였다. 필터링 기능은 Consumer 측에서는 Supplier가 생성한 모든 이벤트를 전달 받지 않고 흥미 있는 이벤트만을 전달 받을 수 있도록 하거나 Supplier측에서도 Client에게 요구되는 이벤트만을 공급할 수 있도록 하여 이벤트 전송 효율을 높일 수 있도록 하는 기능이다.

[그림 3]은 uniORB Notification 서비스의 프로그램 관계도를 나타내고 있다.

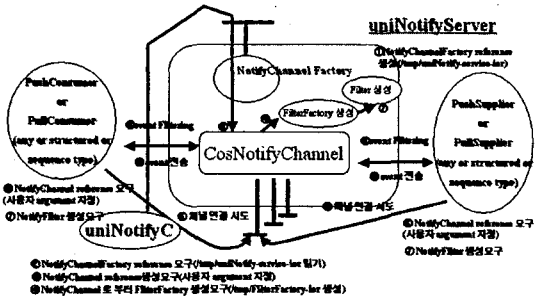

그림 3 uniORB Notification 서비스 프로그램 관계도

uniORB Notification 서비스 프로그램의 실행 순서를 보면, ①uniNotifyServer 는 Notify 채널 팩토리 레퍼런스를 생성 및 저장하고 대기 상태로 들어간다. ② 다양한 Notify 채널을 생성하기 위해서 uniNotifyC 프로그램은 이벤트 채널 팩토리 레퍼런스를 읽어 들인다. ③ 읽어 들인 채널 팩토리 레퍼런스를 통해 사용자로부터 입력 받은 값으로 Notify 채널 레퍼런스를 생성하여 저장한다. ④ NotifyChannel 로부터 필터링 기능을 위한 FilterFactory 생성요구를 한다. ⑤이벤트의 공급 및 소비를 하는 Supplier 와 Consumer 는 uniNotifyServer와 채널 연결 시도를 한다. ⑥uniNotifyC 프로그램이 저장한 Notify 채널 레퍼런스를 읽어들인 후 Notify 채널 레퍼런스를 통해 uniNotifyServer 에게 채널 연결을 한다. ⑦ NotifyFilterFactory 에게 NotifyFilter 생성 요구를 한다. ⑧ uniNotifyServer 는 해당 Consumer 나 Supplier 가 등록한 Constraint 에 대해 필터링객체를 생성하고 실행한다. ⑨ Supplier 는 이벤트를 공급하고 Consumer 는 이벤트를 소비한다.

[그림 4]는 Any 타입의 이벤트를 전송하는 PushSupplier 가 uniNotifyServer 에게 연결되는 구조도를 나타낸다.

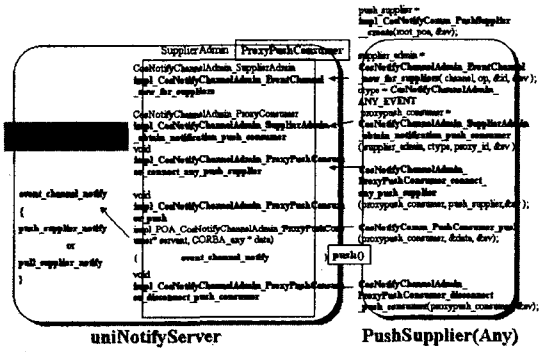

그림 4 Any 타입의 PushSupplier 와 uniNotifyServer 의 연결 구조도

PushSupplier 와 uniNotifyServer 의 연결 구조도내의 구현 코드는 PushSupplier 가 uniNotifyServer 에게 연결요청 , 이벤트 공급(Push), 연결 해제에 관한 함수 맵핑과 흐름을 나타낸다. PushSupplier 는 사용자가 입력한 이벤트의 수에 따라 이벤트를 생성하여 uniNotifyServer 에게 Push 함수를 통해 Any 타입의 이벤트를 공급한다. 공급된 이벤트는 uniNotifyServer 와 연결된 Consumer 의 형태에 따라 다른 형태의 이벤트 전달

이 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 일시저장하며 PullConsumer 가 Proxy객체함수인 try_Pull(nonblock) 이나 Pull(block)을 호출하면 ProxyPullSupplier는 이벤트 버퍼에서 대기중인 이벤트를 전달하고 버퍼 내의 해당 이벤트를 제거한다.

uniNotifyServer 와 *Supplier, Consumer* 의 연결은 표준 안에 제시된 함수를 적용하여 대부분 동일한 형태의 함수를 호출하도록 구현했으나 이벤트의 공급 및 소비에 관한 함수만은 다른 형태를 가지고 있다.

[그림 5] 는 PushConsumer 가 uniNotifyServer 에게 연결하고 Filter 를 등록하는 과정을 나타내는 그림이다.

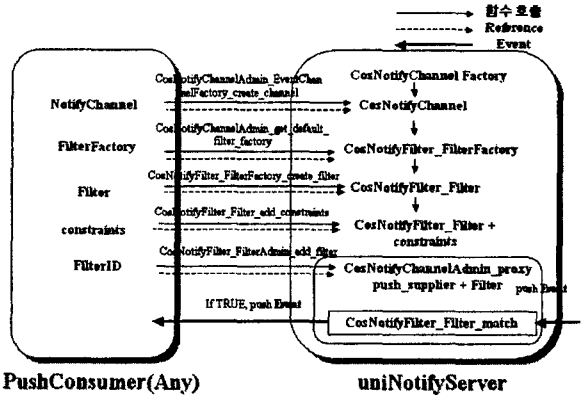

그림 5 CosNotifyFilter 생성과정

PushConsumer 는 uniNotifyServer 에게 NotifyChannel 를 통해서 CosNotifyFilter_FilterFactory 레퍼런스를 얻어와 FilterFactory에게 Filter생성을 요구한다. Filter객체 생성후 Consumer는 원하는 이벤트 유형(constraints)을 Filter객체내에 등록한다. 이벤트 유형이 등록된 Filter 객체를 FilterAdmin 에 추가함으로서 PushConsumer 를 위한 Filter 객체는 CosNotifyChannelAdmin_proxypush_supplier 에 내부적으로 연결되어지며 uniNotifyServer 에게 전달된 이벤트는 Filter 객체내의 match 함수를 통해 이벤트 유형에 일치하는 이벤트만이 최종적으로 PushConsumer 에게 전달된다.

[그림 6] 은 uniORB 상에서 사용되어지는 Notification Service 에서 any 타입의 이벤트 데이터 유형과 필터링을 위해 사용되어지는 이벤트 유형(Constraints)의 일례를 나타내는 그림이다.

[그림 7] 은 구현된 Notification 서비스를 실행하여 나온 결과이다. uniNotifyServer 에서는 Consumer 의 접속에 따라 Filter 의 match함수의 실행과정을 나타내며 PushConsumer 에서는 해당 이벤트 데이터 타입에 따라 필터링을 거쳐 전송되어진 것을 보여주고 있고 PullSupplier 에서는 structured 이벤트를 공급하는 장면을 나타낸다.

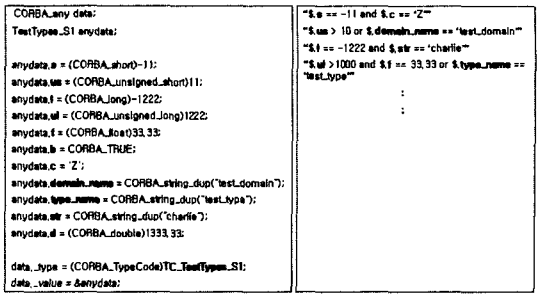

그림 6 Any 형 데이터 및 Constraints

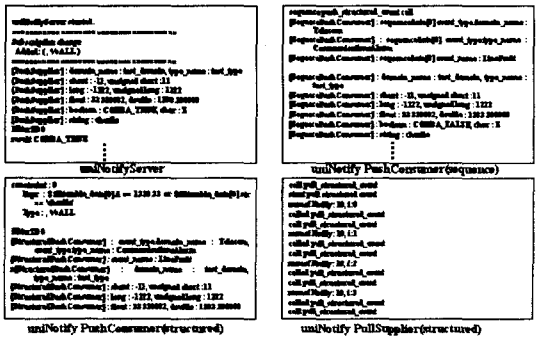

그림 7 uniORB Notification 서비스 결과

4. 결론

본 논문에서는 이벤트 전달 및 필터링 기능이 보완된 Notification Service 를 실시간 고성능 전화 교환기에서 성능을 그다지 줄이지 않는 서비스만을 제공하도록 설계 및 구현하였다. 차후 연구방향으로는 일반 콘솔창 화면에서 실행되는 상황들을 GUI 환경에서 Event 서비스 및 Notification 서비스를 실행할 수 있도록 하여 사용자나 관리자가 이벤트의 진행 상황을 알기 쉽도록 하는 것이 있다.

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)
[8] 한국정보과학회, 윤교철, 박성우, 황선태, 김영만, 이동길, 백의현, 장종현, uniORB 상에서의 Event Service 구현 및 Notification Service 설계 (2001, 10,20)