CAN 네트워크를 위한 내장형 ORB 프로토콜의 설계

김기문 *○ 김선일 ** 김태형 + 전광일 ++ 홍성수 *
* 서울대 전기공학부 ** 홍익대 컴퓨터공학과 + 한양대 전자계산학과 ++ 서울대 컴퓨터공학과

Design of Embedded Inter-ORB Protocol for CAN Network

Kimoon Kim *○ Sunil Kim ** Taehyung Kim + Gwangil Jeon ++ Seongsoo Hong *
* School of Electrical Engineering, Seoul National Univ. ** Dept. of Computer Engineering, Hongik Univ.
+ Dept. of Computer Science & Engineering, Hanyang Univ. ++ Dept. of Computer Engineering, Seoul National Univ.

요 약

많은 내장형 시스템이 분산 제어 방식을 채용하고 있는 현재, 소프트웨어 개발자들은 이기종으로 구성된 시스템의 설계 및 관리에 많은 어려움을 겪고 있다. 이에 따라 범용 통신 미들웨어인 CORBA, DCOM 등을 내장형 시스템에 사용하고자 하는 요구가 높아지고 있으나 이들은 분산 내장형 시스템의 고유한 제약 조건을 만족하지 못해 내장형 시스템에 사용된 전례가 드물다. 본 논문은 표준 CORBA의 ORB 프로토콜인 GIOP가 CAN 네트워크와 같은 내장형 콘트롤러 네트워크에 적용되었을 때 드러나는 문제점을 살펴보고, 이러한 고찰의 결과에 근거해 내장형 시스템 환경에 최적화된 새로운 ORB 프로토콜, EIOP (Embedded Inter-ORB Protocol)을 제안한다.

1. 개요

내장형 시스템의 응용 분야가 급격히 확대되면서 많은 내장형 시스템이 분산 제어 구조를 채택하고 있다. 일례로 자동차 내부 제어에는 엔진 제어 유닛, 기어 박스, 가속 제어 장치등의 제어 시스템이 병렬적으로 수행된다. 이러한 시스템에는 작은 전용 프로세서를 여러 개 사용하는 분산 구조가 중앙 집중 제어 방식보다 비용면에서 훨씬 효율적이다. 분산형 구조는 또한 모듈형 설계를 가능하게 하여 일반적으로 개발 기간을 단축하고 기존 시스템의 재구성 또한 쉽게 해준다.

그러나, 소프트웨어 면에서는 이러한 분산 제어 방식의 복잡도가 중앙 집중 제어 방식보다 훨씬 높다. 무엇보다 제어 응용 프로그램을 이종의 프로세서와 RTOS등 소프트웨어 플랫폼, 다양한 네트워크 하드웨어에 독립적으로 개발하는 것이 쉽지 않은 문제로 대두된다.

분산 응용 소프트웨어를 플랫폼 독립적으로 개발하는 문제는 범용 시스템에서는 통신 미들웨어인 CORBA [OMG 98], DCOM, Java RMI 등에 의해 쉽게 해결할 수 있는 것으로 알려져 있다. 이들 미들웨어는 객체 지향성에 의해 모듈형 설계를 강제하고 콤포넌트의 재사용성을 고취하는 등 많은 부가적인 장점도 아울러 가지고 있다. 따라서 분산 내장형 시스템에서도 이러한 미들웨어를 사용할 수 있다면 많은 문제를 해결할 수 있겠다.

그러나, 범용 미들웨어는 내장형 시스템에서 사용하기에 (1) 많은 메모리 자원을 요구하고 [Gokhale 99], (2) 네트워크 대역폭을 낭비하는 요소가 많으며, (3) 내장형 시스템에서 사용되는 다자간 통신 모델 등을 지원하지 못하는 등의 문제점을 가지고 있다.

이러한 문제를 해결하기 위해 본 논문은 CAN 네트워크 [Bosch 91]를 배경으로 내장형 콘트롤러 네트워크 환경에 걸맞는 ORB(Object Request Borker)간 프로토콜인 *EIOP (Embedded Inter-ORB Protocol)*를 제안한다. EIOP는 CORBA 표준 GIOP을 적은 대역폭을 사용하고 다자간 통신 모델을 효율적으로 지원하도록 개선한 프로토콜이다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 시스템 모델과 이와 연관된 소프트웨어 이슈를 설명한다. 이어 3장은 EIOP의 특징을 개략적으로 살펴본다. 4장은 EIOP의 구성 요소인 CAN 트랜스포트, 네트워크 데이터 포맷, EIOP 메시지 포맷을 자세히 설명한다. 마지막으로 5장은 앞으로의 연구 방향에 대해 소개한다.

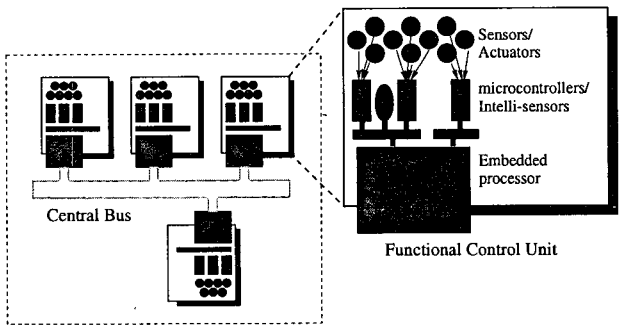

그림 1: 내장형 콘트롤러 네트워크

2. 시스템 모델과 소프트웨어 이슈

2.1 내장형 콘트롤러 네트워크

그림 1에서 보는 것처럼, 내장형 콘트롤러 네트워크는 여러개의 **기능형 제어 유닛(functional control unit)**이 하나의 버스를 공유하면서 네트워크로 연결되어 있는 제어 시스템을 지칭한다. 기능형 제어 유닛은 특정 전기-동력 장치를 제어하는 부분적인 기능을 수행할 수도 있고 다른 유닛들을 감시하고 조정하는 핵심 기능을 맡기도 한다. 자동차 내부를 예로 든다면 엔진 제어 유닛 (ECU), 가속 조정 장치 (ASC) 등이 이러한 기능형 제어 유닛에 해당하겠다.

하나의 기능형 제어 유닛 내부에는 많은 전자 장치가 있다. 이러한 전자 장치들은 그 처리 능력과 자원의 양에 따라 센서/구동기 (sensor/actuator), 마이크로 콘트롤러, 내장형 프로세서의 세 가지 종류로 나뉜다. 센서와 구동기는 제어 신호를 샘플링하거나 촉발시키는 역할을 한다. 마이크로 콘트롤러는 이러한 센서/구동기를 주어진 조건에 따라 지능적으로 관리하는 역할을 한다.

내장형 프로세서는 하나의 기능형 제어 유닛 안에 있는 모든 전자 장치들을 총체적으로 관리하는 핵심 기능을 수행한다. 즉, 내장형 프로세서는 마이크로콘트롤러에 의해 미리 가공된 제어

프로토콜	최대 대역폭 (Mbit/s)	프레임 데이터 (바이트)	최대 프레임수 (개/s)	멀티캐스트 지원
CAN	1	8	8,000	지원
LonTalk	1.25	229	569	지원

표 1: 내장형 네트워크의 대역폭과 데이터 수송량

변수들을 수집하고 특정 제어 단계를 계획하고 가동하는 기능을 한다.

한편, 이러한 내장형 프로세서들은 다른 기능형 제어 유닛의 프로세서들과 중앙 버스를 통해 수시로 정보를 교환한다. 이러한 기능형 제어 유닛간 통신은 수많은 제어 정보를 공유하고 전체 시스템 동작을 조정하기 위해 행해진다. 중앙 버스로는 상대적으로 신뢰성이 높고 고 대역 폭을 제공하며 여러 제조업자의 폭 넓은 지원을 받을 수 있는 CAN 네트워크나 VME-BUS, 이더넷 등이 사용되곤 한다.

2.2 소프트웨어 이슈

이러한 내장형 콘트롤러 네트워크에 공통의 통신 소프트웨어 플랫폼을 구축하는 것은 매우 어려운 일이다. 특히 CORBA, DCOM 등의 범용 미들웨어는 아래의 문제점을 해결하지 않고는 내장형 콘트롤러 네트워크에 채용될 수 없다.

- 표 1에서 보듯이 내장형 네트워크는 많은 양의 데이터 전송을 수용하기에는 대역 폭이 부족하고 하나의 데이터 프레임의 크기 또한 매우 작다. 반면 대부분의 범용 미들웨어는 수 Mbps 이상의 대역 폭을 제공하는 고속 네트워크에 익숙해 있으며 데이터 전송에 있어 미들웨어 메시지 포맷 자체가 부가하는 오버헤드가 매우 크다.
- 내장형 네트워크에서는 전통적인 일 대 일 통신보다는 여러 노드가 개입하는 N:N 통신이 더 일반적이다. 예를 들어 차량 내부에는 여러 부위에 부착된 다수의 열 감지 센서가 다수의 열 관리 모듈에 정보를 배포하게 된다. CORBA의 GIOP같은 전통적인 IOP는 연결 회선(connection)에 근거한 일대 일 통신 모델만을 고려하고 있다.

3. EIOP의 구성과 특징

3.1 프로토콜 콤포넌트

본 장에서는 앞서 설명한 문제점을 해결하고자 내장형 콘트롤러 네트워크에 맞춰 설계된 EIOP (Embedded Inter-ORB Protocol)의 특징을 간단히 살펴본다. EIOP는 아래의 여러 면에서 표준 GIOP (General Inter-ORB Protocol)과는 확연히 다른 내장형 시스템에 최적화된 ORB 프로토콜이다.

- **트랜스포트:** GIOP와는 다르게 EIOP는 1:1 연결 구조로 네트워크 트랜스포트를 제약하지 않는다. 또한 EIOP는 다자간 통신 모델을 최적으로 지원하기 위해 멀티캐스트 메시지와 멀티-타겟 IOR을 정의한다.
- **네트워크 데이터 포맷:** 네트워크상에 전송되는 데이터 포맷으로 CORBA는 표준 CDR (Common Data Representation)을 사용한다. EIOP는 CDR을 개선해 네트워크 대역 폭을 훨씬 적게 사용하는 네트워크 데이터 포맷을 설계하였다.
- **메시지 포맷:** EIOP의 메시지는 GIOP의 메시지보다 훨씬 크기가 작다. EIOP는 GIOP의 메시지 헤더로부터 많은 불필요한 필드를 제거하였으며 객체 키와 함수 명등에 쓰이던 문자열 기반의 식별자들을 제거하였다.

이러한 EIOP의 특징은 EIOP의 구현 대상으로 염두에 두고 있는 CAN 네트워크의 특성을 반영한 것이다. CAN은 자동차, 공작 기계, 의료용 기기에 널리 사용되는 실시간 직렬 버스 시스템으로 내장형 콘트롤러 네트워크에 전형적인 다음의 특징을 보여주고 있다. 먼저, CAN은 단지 1 Mbps의 전송 대역 폭을 제공하며 한 프레임에 전송될 수 있는 데이터가 8 바이트에 불과하다. 두번째로, CAN 표준 프로토콜은 OSI 7 계층 구조상에서 네트워크 층 아래쪽만 규정하고 있을 뿐 트랜스포트 계층에 대해 아무것도 규정하고 있지 않다. 마지막으로 CAN 노드간의 통신은 멀티캐스트 전송에 근거하고 있다.

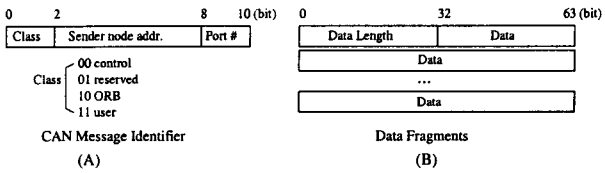

그림 2: CAN 트랜스포트 프로토콜

4. 설계

본 장에서는 EIOP의 구성 요소별로 적용된 설계 원칙과 제기되었던 여러 문제점에 대한 해결책을 설명한다.

4.1 CAN 트랜스포트 프로토콜

CAN 데이터 프레임은 11 비트 메시지 식별자와 8 바이트 데이터 필드로 나누어진다. CAN은 이러한 필드의 구성에 대해 아무것도 지정하지 않고 있으며 개발자가 임의로 설계할 수 있도록 허용하고 있다. 그림 2는 본 논문에서 설계한 CAN 필드의 구조다.

본 논문에서는 먼저 CAN 메시지 식별자를 메시지 클래스 타입(2 비트), 발송 노드 주소 (6 비트), 포트 번호 (3 비트)의 세 서브 필드로 나누었다. 클래스 타입에는 제어, 예약, ORB, 사용자의 네 가지 종류가 있다. 이 중 제어 클래스는 실시간성이 필요한 제어 메시지의 전송에 쓰이고 ORB와 사용자 클래스는 ORB 프로토콜과 사용자가 정의한 임의의 프로토콜에 각각 쓰인다.

발송자 노드 주소와 포트 번호는 각각 64, 8 개의 식별자를 제공한다. CAN에서는 메시지가 브로드캐스트 형식으로 전송되기 때문에 메시지 식별자에 수신자 측의 정보가 들어갈 필요가 없다. CAN은 이러한 메시지 식별자를 메시지가 충돌할 때의 우선순위 결정에도 사용한다. 따라서 본 논문에서는 메시지 클래스에서 포트 번호로 이어지는 계층적인 우선순위 구조를 정의한 셈이 된다.

CAN 데이터 프레임의 포맷은 메시지 클래스에 따라 결정된다. 그림 2 (B)는 ORB와 사용자 클래스에서 사용하는 형식을 그린 것이다. 한 데이터 프레임이 8 바이트의 데이터만을 전송할 수 있는 CAN에서는 어쩔 수 없이 응용 프로그램의 메시지가 여러 프래그먼트(fragment)로 분해되어야 한다. 본 논문에서는 첫번째 프래그먼트의 처음 4 바이트를 전체 메시지의 크기를 기록하는 데 사용했다. 나머지 프래그먼트들은 단순히 데이터만을 담게 된다.

4.2 네트워크 데이터 포맷

표준 GIOP는 IDL 데이터 타입을 네트워크로 전송하기 위해 CDR 포맷을 사용한다. CDR은 호스트 프로세서가 인코딩/디코딩 (encoding/decoding) 작업을 효율적으로 할 수 있도록 설계되었다. 특히 CDR은 litte endian과 big endian을 모두 지원하고 32 bit 프로세서에 적합한 데이터 정렬(align)을 사용함으로써 비슷한 아키텍처의 두 프로세서간에 포맷 변환을 거의 하지 않고 데이터가 전송될 수 있도록 한다.

그러나, 이러한 CDR의 성능은 경우에 따라 네트워크 대역 폭을 낭비하면서 얻어진다. 이에 따라 본 논문에서는 아래의 두 가지 점에서 CDR을 변경하였다.

첫째, CDR은 가변 길이 데이터를 인코딩할 때 언제나 그 크기로 4 바이트를 사용한다. 본 논문에서는 이를 데이터의 길이 자체의 값에 따라 1, 2, 4 바이트를 사용할 수 있도록 변경하였다. IDL에서 가변 길이 데이터는 `string, sequence` 등에 해당하는데 이들 데이터 타입의 인스턴스은 발생 빈도가 높고 대부분 그 길이 자체는 작으므로 이러한 최적화의 의미는 간과할 수 없다.

둘째, 제안된 네트워크 데이터 포맷에서는 정렬(align)을 하지

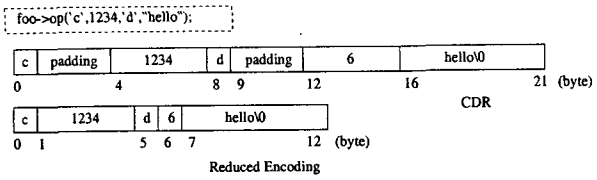

그림 3: 인코딩의 예

않는 축소 데이터 인코딩(packed data encoding)을 지원함으로써 불필요한 패딩(padding) 바이트 들을 제거하였다. 물론 이것은 프로세서의 인코딩/디코딩 작업을 더 복잡하게 함으로써 성능을 저하시킬 여지가 있다. 하지만 CAN의 8 바이트에 불과한 데이터 필드를 고려하면 이러한 최적화는 경우에 따라 매우 중요하리라 여겨진다.

그림 3은 간단한 함수 호출에 대해 두 방식을 비교한 것이다. 그림에서 보듯이 CDR이 22 바이트를 요구하는 데 제안된 인코딩 방식은 그 절반에 가까운 13 바이트만을 사용한다.

4.3 EIOP 메시지 포맷

표준 CORBA에서 모든 ORB 메시지는 GIOP 헤더를 앞에 붙여 네트워크에 전송된다. 한 GIOP 헤더는 12 바이트의 공통 헤더와 메시지 타입에 따른 추가 헤더로 이루어진다. Request 메시지 타입의 경우 추가 헤더의 크기는 보통 수십 바이트에 이른다.

이러한 메시지 헤더는 모든 메시지에 포함되는 것이므로 그 크기를 줄이는 것은 매우 중요하다. 특히 내장형 콘트롤러 네트워크에서는 응용 프로그램에서 보내는 메시지 본체의 크기가 매우 작으므로 GIOP 헤더와 같은 오버헤드가 큰 헤더는 사용할 수 없게 된다. 본 논문에서는 아래의 두 가지 면에서 GIOP 헤더를 근본적으로 바꾸었다.

첫째, GIOP 헤더는 많은 단순한 응용 프로그램에서는 거의 사용되지 않을 옵션 필드들을 많이 정의하고 있다. 이러한 예로는 보안 서비스와 트랜색션 서비스를 위해 정의된 `service_context`와 `requesting_principal`등을 들 수 있겠다. EIOP는 이러한 필드를 모두 제거하였다. 차후 이런 서비스를 사용할 필요가 생길 경우에는 EIOP 헤더의 플래그들 중 예약된 것을 이용해 확장할 수 있으리라 생각된다.

둘째, GIOP 헤더의 많은 필드들이 인터페이스 이름, 메쏘드 명, 객체 어댑터(object adaptor) 이름과 객체 식별자(object id)와 같은 길이가 긴 문자열 식별자들이다. 이러한 필드들은 IOR에도 포함되는 항목인데, IOR은 객체 호출을 위해 클라이언트에게 반드시 넘겨져야 하는 정보이므로 이러한 문자열 기반 식별자를 제한하는 것은 여러 면에서 매우 중요하다.

EIOP는 이러한 필드들이 4 바이트 정수에 근거한 식별자를 쓰도록 변경하였다. 인터페이스 이름과 메쏘드 이름의 경우에는 IDL 컴파일러를 변경하고 개발자들이 몇 가지 규칙을 지킴으로써 쉽게 정수 식별자를 사용할 수 있으리라 보여진다. 개발자들은 먼저 중앙 집중적인 인터페이스 적재소 (interface repository)에 자신들의 인터페이스를 등록하고 고유한 정수 아이디를 발급받는다. 이렇게 발급받은 식별자는 IDL 파일에 추가되고 IDL 컴파일러에 의해 검사된다. 메쏘드 식별자는 한 인터페이스 내에서만 유일하면 되므로, IDL 컴파일러가 일련번호를 발급함으로써 정수로 대치될 수 있다.

객체 어댑터는 개발자가 API를 통해 지정하는 것으로 정수를 받아들이는 API를 만드는 것으로 쉽게 수정이 가능하다. 객체 식별자는 POA 정책(Portable Object Adaptor policy)에 따라 시스템이 만들어낼 수도 있고 사용자가 지정할 수도 있는데 양쪽 모두 별다른 어려움없이 정수에 기반한 식별자를 만들어 낼 수 있다. 그림 4는 개선된 EIOP 헤더를 정의한 것이다.

마지막으로 EIOP는 다자간 그룹 통신을 지원한다. 그림 4에서 보듯이 EIOP의 메시지 요청 (Request) 헤더와 IOR은 모두 `struct receiver`의 `sequence`로 수신자를 지정한다. 물론 다자간 통신은 CAN의 경우처럼 하드웨어적인 메커니즘을 통해 실

```
module EIOP {
  typedef struct receiver {
        unsigned short          endpoint;
                                // node id and port
        sequence<octet>         object_key;
  } receiver;
  struct MessageHeader_1_0 {
        octet                   version;
        octet                   flags;
  };
  struct RequestHeader_1_0 {
        unsigned short          interface_id;
        unsigned short          operation_id;
        sequence <receiver>     receivers;
  };
  struct IOR_1_0 {
        unsigned long           interface_id;
        sequence <receiver>     receivers;
  };
}
```

그림 4: EIOP 메시지와 IOR의 포맷

제로 이루어진다. EIOP 헤더와 IOR에 있는 정보는 이러한 하드웨어 메커니즘을 보조해 수신자 노드가 동일한 발송자 노드에게서 온 메시지 중에 불필요한 메시지가 왔을 때 이를 걸러 내기 위해 사용된다.

5. 결론과 향후 작업

본 논문에서는 내장형 콘트롤러 네트워크에 CORBA GIOP와 같은 통신 미들웨어를 적용시킬 때 제기되는 문제점을 고찰하였다. 본 연구에서는 이에 GIOP를 개선한 내장형 ORB 프로토콜 EIOP를 설계하였다. EIOP는 대역폭 효율성을 고려해 네트워크 데이터 포맷과 메시지 포맷을 설계하였고 CAN 등의 콘트롤러 네트워크의 멀티캐스팅 방식을 효과적으로 사용할 수 있는 다자간 그룹 통신 모델을 지원한다.

한편 이러한 EIOP를 효과적으로 사용할 수 있기 위해서는 1) 경량의 ORB 엔진을 구현, 2) CORBA를 이용한 제어 응용 프로그램을 case study 하는 것이 아울러 요구된다.

본 연구에서는 현재 수백 KB 미만의 메모리만을 요구하는 경량의 ORB 엔진을 OMG의 Minimum CORBA 규약에 맞춰 개발하고 있다. 또한 CAN 네트워크 상에 EIOP를 구현하는 작업을 함께 진행 중이다. 응용 프로그램 모델로는 로봇 제어등에서 광범위하게 사용되는 모델인 producer/consumer 관계의 정보 공유 모델을 CORBA를 이용해 설계하였을 때의 문제점을 살피고 있다.

이러한 일련의 작업은 내장형 콘트롤러 네트워크의 응용 프로그램 설계에 있어서 확장성, 조립성, 개발의 효율성, 신뢰성, 재사용성 등을 획기적으로 개선하는 성과를 낳을 수 있으리라 여겨진다.

참고 문헌

[Bosch 91] CAN Specification, ver. 2.0, Robert Bosch GmbH, Stuttgart, 1991

[Gokhale 99] A. Gokhale and D. C. Schmidt, *Technique for Optimizing CORBA Middleware for Distributed Embedded Systems.*, appeared in the proceedings of IEEE INFOCOM '99, New York, March 21-25th, 1999

[OMG 98] Object Management Group, *The Common Object Request Broker: Architecture and Specification.* Revision 2.3, 1998

[OMG 98-2] Object Management Group, *Minimum CORBA - Joint Revised Submission*, OMG Document orbos/98-08-04 edition, August 1998