CAN 버스를 위한 내장형 CORBA에서 접속 지향과 가입 기반의 통합 통신 기법

(Integrating Subscription-Based and Connection-Oriented Communications into the Embedded CORBA for the CAN Bus)

김 기 문 † 김 태 형 ††
(Kimoon Kim) (Taehyung Kim)

요 약 본 논문에서는 CAN 기반 분산 제어 시스템을 위한 환경 명세 CORBA인 CAN-CORBA를 설계한다. CAN-CORBA의 ORB core는 다음과 같은 특성을 지닌다. (1) 기존의 접속 지향 점대점 통신 뿐만 아니라 가입 기반(subscription-based)의 그룹 통신을 지원한다. 이를 통해, 결과적으로 전형적인 제어 시스템의 데이터 멀티캐스팅(multicasting) 요구를 한층 충족시킨다. (2) CORBA 메소드(method) 호출에 필요한 메시지 전송량을 크게 감소시킨다. CAN 같은 저속 브로드케스트(broadcast) 버스에서도 CORBA 메소드 호출의 오버헤드를 감당 할 수 있게 한다. 자연스럽게 두 종류의 통신 방법을 본 논문에서 제안한 매우 작은 크기의 ORB에 통합하기 위하여, 최대 4가지의 상위 통신규약을 지원할 수 있는 CAN의 전송층(Transport layer) 통신규약을 설계한다. 또한 서울대학교에서 이미 개발된 mArx라는 실시간 운영체제 환경에서 CAN-CORBA를 구현하였다. 성능 평가를 통하여 매우 제한적인 자원을 지니는 분산 내장형 제어 시스템 개발에 CORBA를 사용하는 것이 매우 적합하다는 결과를 보인다.

Abstract *In this paper, we present the design of CAN-CORBA, an environment-specific CORBA for CAN based distributed control systems. The ORB core of the CAN-CORBA has the following properties. (1) It supports subscription-based group communication as well as the classical connection-oriented point-to-point communication of CORBA; as a result, it better services the data multicasting requirement of typical control systems. (2) It significantly lowers the amount of message traffic required for each CORBA method invocation so that even the slow broadcast bus of the CAN can tolerate the overhead of CORBA method invocations. To seamlessly integrate two communication schemes into our extremely light-weight ORB, we design a transport layer protocol on the CAN that can support up to four different upper level protocols. We have implemented the CAN-CORBA on the mArx real-time operating system we have developed at Seoul National University. Our experiments clearly demonstrate that it is feasible to use CORBA in developing distributed embedded control systems possessing severe resource limitations.*

1. 서 론

컴퓨터와 통신 기술의 빠른 발전은 컴퓨팅 환경과 프로그래밍 형태의 많은 변화를 가져왔다. 이러한 기술적 흐름은 매우 복잡한 분산 시스템으로 구성되는 기존의 실시간 컴퓨터 제어 분야에도 예외는 아니라 할 수 있다. 기존의 분산 제어 시스템들은 다수의 고가 마이크로콘트롤러와 마이크로프로세서를 실시간 네트워크로 연결하여 처리하는 경우가 일반화 되어왔다. 한편, 인터넷 관련 기술의 괄목할 만한 발전은 제어 시스템 또한 인터넷과 연결되게 만들었는데, 대표적인 예로 지능형 운송 시스템의 승객용 차량을 들 수 있다. 승객용 차량은 다수의 마이크로프로세서와 CAN(controller area network) 버스(bus)[2]를 장착하고 무선 네트워크를 통해 인터넷과 연결되어 운전자가 도로의 교통량 정보를 수신하고 차량의 속도를 송신할 수 있도록 하여 중앙 교통 통제소가 도로 교통 조건을 실시간에 감시할 수 있

† 학생회원 : 서울대학교 전기공학부
kmkim@redwood.snu.ac.kr
†† 정 회 원 : 한양대학교 컴퓨터공학과 교수
tkim@cse.hanyang.ac.kr
논문접수 : 2000년 7월 13일
심사완료 : 2000년 12월 1일

도록 한다. 그러나 네트워크로 연결된 분산 제어 시스템의 개발은 그 개발 소프트웨어의 복잡성 문제에 직면하고 있는데, 이를 내장형 소프트웨어의 위기라고도 한다. 내장형 소프트웨어 시스템은 여러 가지 이유로 매우 복잡한 형태를 지니는데, 매우 열악한 동작 환경, 이질적인 입출력 장치의 처리, 엄격한 시간 제한과 결함 허용에 관한 요구 조건 등이 그 원인이다.

최근 들어 CORBA[4], DCOM[18], Java RMI[19] 등이 이러한 소프트웨어 위기를 해결할 수 있는 객체 기반의 컴포넌트 기술로 각광받고 있지만, CORBA같은 일반 용도의 미들웨어 시스템은 몇 가지 이유로 인해 내장형 실시간 제어 시스템에 바로 응용할 수 없다는 단점을 가지고 있다. 첫째, 다른 미들웨어뿐만 아니라 CORBA 또한 많은 자원을 필요로 한다. 특정 ORB 구현이 경우에 따라 수 메가바이트 이상의 메모리를 요구하는 것을 쉽게 볼 수 있다. 둘째, CORBA는 구현된 시스템의 수행 동작을 예측하기 어려운 경우가 많은데, 이는 CORBA가 그 동작을 예측할 수 없는 상용 소프트웨어 시스템 위에 구축되기 때문이다. 세 번째, 접속 지향 통신 모델인 CORBA는 제어 시스템 응용에는 적합하지 않다. 일반적으로 많은 제어 시스템들에서는 요청이 없이도 데이타 수집 장치가 다중 콘트롤러에 측정된 데이타를 보내야함으로 멀티케스트(multicast) 통신 서비스가 필수적이다.

위에서 제시된 표준 CORBA의 문제점을 해결하기 위하여, [10]에서는 CAN 기반의 분산 제어 시스템을 위한 새로운 내장형 CORBA를 설계하였다. CAN은 전 세계적으로 자동차 산업 전반에서 널리 사용되고 있는 산업용 실시간 네트워크의 표준이다[2, 7]. CAN은 고작 최대 1Mbps의 전송 대역폭과 8 bytes 길이의 데이다 단위를 사용하기 때문에 CAN 기반의 분산 환경에서 CORBA 응용 프로그램을 실행하는 것 자체가 상당히 까다로운 일이 아닐 수 없다. 본 연구팀에서는 CAN의 효율적이며 신뢰성 있는 브로드케스트 방법을 효과적으로 활용하여 CORBA에 가입 기반(subscription-based) 그룹 통신 기법을 구현하였으며, 또한 주제기반 주소기법(subject-based addressing)을 효과적으로 실현한 전송층 통신규약을 정의한 새로운 내장형 CORBA인 CAN-CORBA를 설계하였다[10].

본 논문에서는 다음과 같은 설계 목표를 정하여 본 연구팀이 제안한 CAN-CORBA[10]를 확장한다.

- 다른 표준 ORB와의 상호 연동을 향상하기 위하여 CAN-CORBA에 접속 지향 점대점 통신과 가입 기반의 그룹 통신 방법을 자연스럽게 통합한다.
- 기존 CAN-CORBA의 부가되는 코드 크기와 자원 요구량을 최소화한다.

이러한 목표를 달성하기 위하여 본 논문에서는 가입 기반 통신용 호출 채널(invocation channel)과 접속 지향 통신용의 접속(connection)이라는 두 가지 개념을 제안하고, 세 개의 통신규약을 설계한다. 하나는 호출 채널 연결과 접속 생성을 위한 네트워크 관리 통신규약이며 나머지 두 개는 앞서 언급한 두 가지의 통신 방법 규약이다. 이 통신규약은 CAN을 위해 본 연구에서 설계한 전송층 통신규약의 지원을 받는다. 마지막으로 CORBA의 메소트 호출 오버헤드를 최소화하기 위하여 IDL 데이타 타입과 GIOP 메시지 헤더를 재설계한다.

[10]에서는 CAN-CORBA의 기본적인 설계만을 다루었다. 최근에 본 연구팀은 Red Hat의 ORBit 소스 코드[6]를 변형하여 제안한 확장 설계의 구현을 완료하였다. 확장된 시스템은 서울대학교에서 기 개발된 실시간 운영체제인 mArx[16]를 기반으로, 인텔의 내장형 프로세서인 Intel 386 EX가 장착된 CAN기반 분산 환경에서 수행시켜 실험 결과를 얻었다.

1.1 관련 연구

현재까지 CAN을 위한 많은 상위 레벨의 통신규약들이 개발되었다[1, 5, 8, 9]. Allen Bradley의 DeviceNet은 널리 사용되고 있는 통신규약 중에 하나이다[1]. DeviceNet은 종합적으로 컴포넌트 기반 분산 프로그래밍을 편리하게 하는 분산 객체 모델과 장치 프로필을 제공하며, 그룹 통신과 점대점 통신을 지원한다. DeviceNe의 통신 서비스는 점대점 접속 같은 기본 요소 위에 구축되었기 때문에, 그룹 통신은 두 통신 대상간의 양방향 접속을 필요로 한다. 따라서 발표자/가입자(publisher/subscriber) 모델 같은 익명의(anonymous) 그룹 통신을 제공하는 것이 매우 어렵다. 그러나 CAN-CORBA의 경우, 그룹 및 점대점 통신이 CAN-CORBA의 전송층 통신규약에 구축되어 각각이 독립적으로 동작하도록 설계되어 있으므로, 효율적이고 유연한 그룹 통신이 가능케 한다.

분산 실시간 시스템의 발표자/가입자 모델은 Rajkumar *et al.*[14]에의해 제안되었는데, 이 연구에서는 하부 네트워크가 점대점 전송층 위에 구축되는 것을 전제로 했다. 따라서 일대다 그룹 통신을 여러개의 점대점 접속을 통하여 가입자에게 메시지 복사본을 전송하여 실현한다. 그러나 이러한 방법은 CAN과 같은 브로드케스트 네트워크에는 효율적이지 못하다.

Kaiser *et al.*[8, 9]는 발표자/가입자 모델을 CAN 2.0B bus에 적용하였다. CAN 2.0B와 2.0A의 차이점은

식별자의 비트수인데, 2.0A가 11-bit만을 사용하는데 비해, 2.0B는 29-bit의 식별자를 사용한다는 점이 다르다. [9]에서는 이벤트(event) 채널이라는 기법을 제안하였는데, 이 기법은 발표자(publishers)와 가입자(subscribers)사이에 가상 연결을 생성한다. 각각의 이벤트 채널은 29-bit의 CAN 식별자 중에 14-bit로 구성된 전역 이벤트 태그로 구분되며, 나머지 15-bit는 메시지 우선순위와 노드 식별자로 쓰인다. 이 방법의 단점은 CAN 2.0A 버스에 효과적으로 적용할 수 없다는 점인데, 이 경우, 5-bit 정도의 메시지 우선 순위와 노드 식별자만을 사용하더라도 고작 64개의 이벤트 채널밖에는 제공할 수 없으며, 이점이 실제 구현상의 심각한 문제를 발생시킬 수도 있다. 확장된 2.0B의 식별자가 버스의 중재 오버헤드를 증가시킨다는 중요한 이유 때문에 2.0A를 선호하는 경향이 있으므로 본 연구에서는 CAN 2.0A의 상위층 통신규약에게 호출 채널을 사용하여 최고 512개의 포트를 지원하도록 하였다.

2. 대상 하드웨어 시스템 모델

본 연구에서 제안한 CAN-CORBA는 CAN 버스 상에 구축된 분산 내장형 제어 시스템에서 동작하도록 설계되었다. 본 장에서는 하드웨어 플랫폼의 제약과 특징을 쉽게 이해할 수 있도록 대상 시스템 하드웨어 모델을 소개한다.

대상 하드웨어는 내장형 제어 네트워크로 연결된 여러 개의 기능 제어 유닛(FCU)으로 구성된다. <그림 1>은 승객용 차량의 전자 제어 시스템의 예를 보인 것이다. 한 개 이상의 마이크로콘트롤러와 마이크로프로세서로 구성된 각 FCU는 감지기와 작동장치간의 인터페이스 기능과 미리 정의되어있는 제어 알고리즘 실행을 통해 할당된 제어 기능을 수행하는데, 시스템 설정에 따라 데이타 생산자나 소비자 또는 생산·소비자의 역할을 수행하게된다.

<그림 1>에서 내장형 제어 네트워크(ECN)는 저렴한

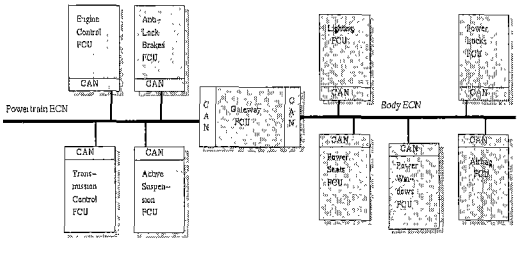


그림 1 내장형 분산 제어 시스템 예: 승용차 제어 시스템

버스 어댑터를 통해 FCU들을 연결한다. 이러한 ECN들은 실시간 메시지 전송 기능과 매우 엄격한 연산·기능 제약 준수 등을 필요로 하므로 본 연구에서는 이를 만족하며 국제적인 산업 표준으로 받아들여지고 있는 CAN[7]을 내장형 제어 네트워크로 선택하였다.

CAN 표준안은 OSI 참조 모델의 물리층와 데이타 링크층 통신규약을 명세하고 있는데, 이러한 CAN은 예측가능하고 우선 순위 기반인 버스 중개를 통해 메시지 전송 지연 시간을 제한 할 수 있음으로써 실시간 통신에 적합한 특성을 지닌다. CAN의 메시지는 식별자와 데이타, 오류, 승인 그리고 CRC 항목으로 구성된다, 식별자 항목은 CAN 2.0A의 11-bit 또는 CAN 2.0B의 29-bit로 구성되며 데이타 항목을 최대 8 바이트까지 확장 가능하다. CAN의 네트워크 어댑터는 메시지 전송 시, 먼저 식별자 항목을 전송하고 그 다음 데이타 항목을 보내게 되는데, 메시지의 식별자는 우선 순위의 역할을 하게되며, 이때, 높은 우선 순위의 메시지가 낮은 우선 순위의 메시지 보다 우선권을 지니게 된다.

CAN은 주제 기반 주소 기법[13]에서와 같은 특이한 주소 방법을 제공한다. CAN 상에서 네트워크에 실린 메시지는 도착주소를 가지고 있지 않다. 대신 그 메시지는 주제 태그(subject tag)를 지니는데, 주제 태그는 메시지 식별자에 미리 정의된 비트 패턴으로, 해당 데이타의 내용 정보를 암시한다. 수신 노드는 CAN 버스 어댑터가 특정 식별자 패턴을 지닌 메시지의 일부만을 수용하도록 프로그램 할 수 있다. 이러한 필터링 메커니즘은 CAN 인터페이스 칩에 내장된 마스크 레지스터와 비교 레지스터들로 구현된다. 주제 기반 주소 기법은 본 논문의 CAN-CORBA에서 제공하는 두 가지 통신 모델의 핵심 메커니즘이다.

본 연구에서는 CAN 2.0A 표준만을 고려하였다. CAN 콘트롤러들 중에 2.0A와 2.0B 모두를 지원하는 것이 있음에도 불구하고 DeviceNet과 같은 상용 상위 레벨 통신규약 제품들 대부분이 29-bit 식별자를 거의 지원하고 있지 않다. 이는 CAN 2.0B 네트워크가 기존의 구축된 2.0A 네트워크와의 호환성 문제뿐만 아니라, 더 나아가 2.0B 메시지의 추가된 18-bit가 버스 중개 오버헤드를 증가시키고 메시지 전송 중에 발생하는 잠재적 지터(jitter) 증가로 인한 예측가능성의 감소를 초래하기 때문이다.

3. CAN-CORBA의 전송 통신규약

CORBA는 TCP같은 표준 통신규약이 제공하는 점대점 전송 서비스만을 따르는 반면, 분산 컴퓨터 제어 시

스템은 점대점과 그룹 통신 기능을 모두 제공하여야 한다. 본 장에서는 접속 지향 통신을 위한 접속 생성 규약을 설계하고 이를 CAN-CORBA의 가입 기반 채널 바인딩 규약과 통합하는 기법을 서술한다.

3.1 통신규약 헤더 정의

이미 [10]에서 설명하였듯이, 본 논문에서는 전송층 통신규약 헤더를 정의하기 위해 CAN의 식별자 구조를 사용한다.

CAN의 식별자 구조를 사용하는 것은 CAN 자체가 다음과 같은 본질적인 제약들을 수반하기 때문에 결코 쉬운 작업이라 할 수 없다.

1. CAN의 식별자는 11-bit으로 구성되어 그 자체만으로도 사용하기에 부족한 특성을 지닌다.

2. 각각의 CAN 노드는 동일한 식별자를 가지고 서로 다른 메시지를 동시에 전송할 수 없다.

3. CAN의 식별자는 버스가 데이타를 중개하는 기간 동안에 메시지의 우선 순위로서 동작한다.

위의 제약점 이외에도 전송층 통신 규약이 다중의 상위 계층의 통신규약들을 효율적으로 지원해야 한다는 추가적인 부담도 가지고 있다. 따라서, 본 연구에서는 식별자 비트의 효율적인 사용과 통신규약 설계의 단순화에 집중하며, 적은 실행 오버헤드와 통신규약 스택의 코드를 가지고도 상위 CORBA 계층을 위한 해당 서비스를 제공 가능하도록 한다.

<그림 2>는 제안한 통신 규약 헤더의 형태를 보인 것이다. 그림에서 볼 수 있듯이 CAN의 식별자 구조를 세 개로 구분하였는데, 그 각각은 통신규약 ID(Proto), 전송노드주소(TxNode)와 포트번호 (TxPort)로 이루어진다. 세 개의 부분들은 각각 2bit, 5bit, 4bit 크기로 전체 11bit를 이루게된다. Proto 항목은 상위 계층의 통신 규약 식별자 정보를 표현하며, CAN 메시지 상에서의 식별자에 이은 데이타 항목은 Proto가 가지고 있는 상위 계층의 통신 규약 식별자 따라 다른 형식을 취한다..

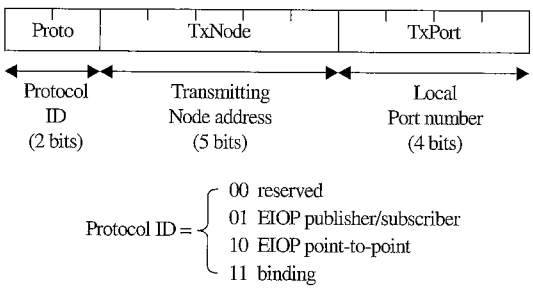


그림 2 CAN 식별자 구조를 이용한 통신 규약 헤더 포맷

현재까지 설계된 통신규약 헤더를 보면 Proto 항목이 가질 수 있는 4가지 값 중에 01_2 와 10_2 은 각각 EIOP의 발표자/가입자 통신규약과 클라이언트/서버 통신규약을 나타냄을 알 수 있다. 11_2 는 호출 채널 바인딩과 접속 생성을 명세하는 네트워크 관리 규약을 위해 사용되어진다. CAN 상에서는 작은 값을 갖는 메시지 식별자가 버스의 데이타 중개 시, 더 높은 우선 순위를 지니게 되는데, 결과적으로 본 연구에서 제안한 CAN-CORBA는 발표자/가입자 메시지가 클라이언트/서버 메시지와 네트워크 관리 메시지 보다 우선하게 된다. 이러한 방법으로 Proto를 할당하는 이유는 발표자/가입자 메시지가 항상 시간에 영향을 많이 받는 감지 데이타를 가지고 있고, 새로운 세션을 시작하는 것이 이미 허가된 실시간 메시지 트래픽을 간섭해서는 안되기 때문이다. 마지막으로, 최우선순위 통신 규약을 나타내는 Proto 항목의 값 00_2 은 우선 순위가 높은 잠재적 사용자 정의 긴급 통신 규약을 위해 남겨둔다. 이러한 통신규약의 메시지는 CORBA의 객체 추상화 계층과 EIOP 통신 규약 스택 모두를 바로 통과 가능하므로 매우 적은 메시지 전송 지연을 얻을 수 있다.

TxNode 는 송신 노드의 주소이다. 본 연구에서는 주어진 상위 계층 통신규약 하에서 하나의 노드가 동시에 32개까지 다른 노드와 CAN 버스를 통해 접속 가능하다. TxPort는 지역적인 특정 전송 노드의 포트 번호를 표현한다. TxNode는 전체 네트워크 상에서 전역적으로 식별 가능한 도메인 이름으로 동작하기 때문에, TxNode와 TxPort는 두 가지 정보를 합쳐 전역 포트 식별자로 사용한다. 이러한 방법은 서로 다른 노드의 포트들이 동일한 포트 번호를 가질 수 있게 하며, 소프트웨어 설계와 유지 시의 모듈화를 증가시킨다. TxPort는 각 노드의 최대 16개의 지역 포트를 지원할 수 있으므로, 특정 상위 계층 통신 규약하의 네트워크에서 512개의 전역 포트를 사용할 수 있게 된다.

헤더가 목표 노드의 주소에 대한 어떠한 정보도 가지고 있지 않고, CAN 버스 어댑터의 메시지 필터링 매커니즘을 사용하여 수신 CAN 노드가 특정 포트로부터 전송된 메시지를 선택적으로 수용할 수 있게된다. 이러한 방법에서는 주제 기반 주소가 매우 효과적으로 지원된다.

3.2 가입 기반 통신용 채널 바인딩 규약

본 연구에서 제안한 CAN-CORBA의 구독저 기반 통신의 주요 구성 요소는 "호출 채널(invocation channel)"과 "결합자(conjoiner)"이다. 호출 채널은 발표자들로부터 가입자들 그룹과의 가상 브로드케스트 채널

이며, 발표자는 발표자의 한 포트를 호출 채널에 접속한 후 가입자에게 호출 채널을 선언한다. 앞서 언급했듯이, 포트는 TxNode와 TxPort 쌍으로 이루어진 유일 주소 형태의 전송층의 개체이다. 호출 채널은 하나 이상의 접속된 포트를 가질 수 있다. 발표자는 접속된 포트를 사용, 호출 채널을 통해 메시지를 전송할 수 있으며, 가입자는 호출 채널을 통해 메세지를 수신 할 수 있다.

본 연구의 채널 바인딩 통신 규약은 "결합자"라 이름 붙혀진 중개 객체가 매우 중요한 역할을 수행한다. 결합자는 CAN 노드상에 존재하면서 그 노드의 식별자는 시스템에 존재하는 모든 발표자와 가입자에게 잘 알려져 있다. 따라서 결합자는 네트워크 초기화 시에 실행되어 시스템 서비스 기간 내내 동작하여야 한다.

전역 바인딩 데이타베이스(Gloabl binding database) : 결합자는 각 호출 채널에 대응하는 발표자에 의해 등록되거나 선언된 엔트리를 전역 바인딩 데이타베이스에 유지한다 <그림 3>은 결합자 기반의 발표자/가입자 구조와 전역 바인딩 데이타베이스를 나타낸 것이다. 그림에서와 같이 전역 바인딩 데이타베이스의 엔트리는 채널 태그, OMG IDL 식별자, TxNode, TxPort의 네 개 분야로 구성되어 있다. 채널 태그는 각 호출 채널에 연관된 유일한 상징적 이름으로 응용 프로그램을 작성할 때 프로그래머에 의해 정적으로 정의되어지고, 추후에 발표자와 가입자에 의해 전역 바인딩 데이타베이스의 탐색키로 사용되어진다. OMG IDL 인터베이스 식별자는 시스템 상의 각 IDL 인터페이스와 연관된 유일한 식별자로 OMG IDL 컴파일러가 IDL 인터페이스 식별자를 생성한다. CORBA 실행시 이러한 식별자는 각 메소드 호출 때의 형태 파악에 사용하며, 이는 CORBA 표준이 필요로 하는 엄격한 형태 안전을 보장한다. 채널 태그와 인터페이스 ID는 두 정보를 합쳐 각 호출 채널의 유일한 이름을 표현하며, 시스템 전체적으로 호출 채널에 유일한 이름을 정의하는 것은 프로그래머의 책임이다.

채널 선언과 가입(Channel announcement and subscription) : 결합자는 발표자와 채널 생성 메시지를 교환하고 가입자와는 채널 가입 메시지를 주고받는다. 만일 발표자가 호출 채널과 연결을 원하게 되면 발표자는 결합자에게 등록 메시지를 전송한다. 그러면 결합자는 전역 바인딩 데이타베이스를 갱신한 후 발표자에게 인식 메시지를 전달한다. 가입자는 결합자에게 호출 채널에 가입 요청 메시지를 송신하게되며, 만일 결합자가 전역 바인딩 데이타베이스에서 요청한 호출 채널의 해당 엔트리가 존재하면, 이에 대응되는 바인딩 정보를 가입자에게 전달한다. 따라서 가입자는 가입된 호출 채널에 발표자가 연결되었는지 연결이 단절되었는지 등의 변경 사항을 비동기적으로 참조할 수 있다. 지역 바인딩 에이전트(local binding agent)는 <그림 3>의 타원으로 표현된 subscriber S 안에 것으로, 이러한 갱신 작업을 수행한다.

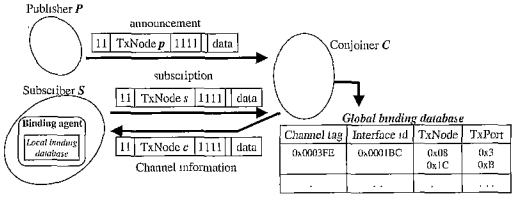


그림 3 결합자 기반 채널 바이딩 규약

결합자는 시스템 상의 어떠한 CAN 노드로부터도 메시지를 수신할 수 있도록 하기 위해 네트워쿠 관리규약하에서 특정한 지역 포트 번호(TxPort)인 1111_2을 사용하도록 한다. <그림 3>에서처럼 결합자에게 전송되는 모든 메시지는 이 지역 포트를 사용한다. 결과적으로 결합자는 Proto가 11_2 란 값을 지닐 때, 이 포트 번호를 갖는 메시지를 무조건 받아들이게 된다. 반면, 다른 CAN 노드들은 결합자의 TxNode와 TxPort를 이미 알고 있기 때문에 결합자로부터 메시지를 받을 수 있다.

바인딩 메시지의 데이타 항목은 전체 바인딩 정보나 실제 질의를 전달한다. 특히, 발표자의 등록 메시지는 채널 태그, OMG IDL 인터페이스 ID, 전역 포트 번호 등과 같은 데이타베이스 항목 구성에 필요한 모든 정보를 담고 있다. 이러한 정보를 이용하여, 결합자는 엔트리를 생성하거나 변경하게 된다. 가입자의 요청 메시지는 호출 채널을 위한 채널 테그와 IDL 인퍼페이스 식별자를 포함하고 있는데, 바인딩 데이타베이스로부터 하나 이상의 항목이 정상적으로 검색되어지면, 결합자는 검색된 항목들의 정보를 담은 응답 메시지를 전송하고 이를 가입자의 지역 바인딩 데이타베이스에 저장한다.

3.3 점대점 통신을 위한 접속 생성 규약

앞서 언급한대로 본 연구의 양방향 접속은 단방향 파이프의 쌍으로 이루어진다.

접속은 두 개의 지역 포트를 사용하여 생성되는데, 각 포트는 해당 파이프의 근원 노드에 속한다. 수신 노드가 송신노드로부터 보내지는 메시지를 수신하려면 송신노드의 TxNode와 TxPort 쌍을 알고 있어야 한다. 이러한 방법은 접속 생성이 쌍방 통신 끝점에서 이루어져야 한다. 따라서 접속 생성은 각 끝점의 포트 주소를 교환

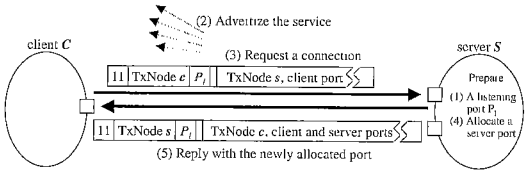


그림 4 접속 형성 과정

하여 서로 방향이 반대인 파이프 쌍을 조합하는 과정이 포함된다. 이때는 널리 알려진 listening 포트 개념을 활용하는 것이 적합한데, listening 포트는 서버에 위치하는 지역 포트이며, 서버에 접근하고자 하는 클라이언트에게 광고되어 진다. 이 개념은 채널 바인딩 통신 규약의 결합자처럼 서버는 어떠한 CAN 노드로부터도 접속 요청 메시지를 받을 수 있어야 한다. CAN 식별자의 listening 포트 번호를 포함하는 접속 요청 메시지를 수신하기 위하여 서버 노드는 노드의 CAN 필터레지스터들을 프로그램 한다. 이때 접속 요청 메시지는 메시지의 Proto 값으로 11_2 를 갖는다.

그림 4 는 접속 생성을 위한 다섯 단계의 과정을 보여 주고 있고, 그 과정은 다음과 같다.

(1) 서버는 듣기(listening) 포트를 설정한다.

(2) 서버는 노드 식별자를 포함하는 듣기 포트 번호를 시스템 상의 각 클라이언트에게 알린다.

(3) 클라이언트는 자신의 노드 식별자인 TxNode와 듣기 포트 번호로 구성된 CAN 식별자를 포함하는 접속 요청 메시지를 서버에게 전송한다. 이 메시지의 데이타는 접속하고자 하는 서버의 노드 식별자와 클라이언트의 사용 가능한 지역 포트 번호를 포함하고 있다.

(4) 접속 요청 메시지를 수신하면, 서버는 지역 비사용 포트 풀에서 하나의 포트를 선택하여 클라이언트에게 할당한다.

(5) 그리고 나서 새로 할당된 포트 번호를 클라이언트에게 전송하며, 이 포트와 클라이언트 포트는 클라이언트와 서버간의 일반적인 통신에 사용되어진다. 듣기 포트가 통신규약 식별자 11_2를 사용하는 것과는 달리, 이 때 할당된 포트는 통신 규약 식별자 10_2를 사용한다.

4. 두 가지 통신 기법의 프로그래밍 모델

본 연구에서 제안한 전송층 통신 규약은 CORBA 응용 프로그램의 프로그래밍 형태에 영향을 미친다. 본 장에서는 제안된 두 가지 통신 기법의 프로그래밍 모델에 대하여 서술한다. 실제 CAN-CORBA를 사용한 코드의 예는 참고문헌[11]에서 찾아 볼 수 있다.

4.1 익명 발표자/가입자(Publisher/Subscriber) 통신

<그림 5>는 호출 채널과 채널에 연결된 발표자들의 포트를 도식적으로 표현한 것이다. 호출 채널을 통해 전송되는 메시지는 메시지의 식별자로 01_2라는 통신규약 식별자를 갖는다. 또한 메시지의 송신노드와 지역 포트 번호도 포함하고 있다. 발표자/가입자 모델은 다음과 같은 특성을 가지고 있다.

- 호출 채널은 발표자의 수행 코드를 작성한 프로그래머에 의해 OMG IDL 의 인터페이스로 정의된다. 인터페이스는 가입자가 반드시 제공하여야 하는 전달 메소드의 서명(signature)을 명세한다. 클라이언트/서버 모델과는 달리, 이러한 인터페이스는 가입자에게 전달되며, 후에 발표자는 메시지를 가입자에게 전송하기 위해서 전달 메소드를 호출하게 된다.

- 호출 채널은 발표자에서 가입자로 익명의 통신을 제공한다. 가입자는 접속에 관여한 호출 채널의 IDL 인터페이스 식별자와 채널 태그만을 필요로 하며 발표자에 대한 정확한 정보는 요구하지 않는다.

- 표준 CORBA의 사건(event) 서비스와는 달리, 호출 채널은 발표자로부터 가입자로 메시지를 전달하여주는 중간 객체를 필요로 하지 않는다. CAN-CORBA에서는 가입자가 자신의 지역 바인딩 데이타베이스에 접속에 관여한 호출 채널의 모든 바인딩 정보를 유지하므로 효율적인 메시지 전송이 가능하게된다.

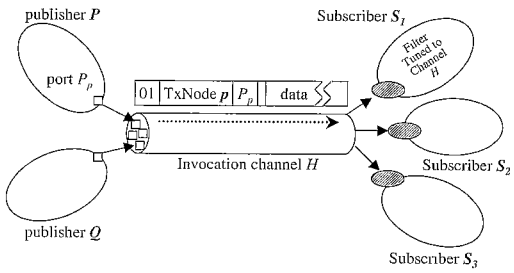


그림 5 호출 채널을 이용한 가입 기반 통신

4.2 클라이언트.서버 통신

subscription 기반 통신이 응용 프로그램을 제어하는데 자연스럽고 적절하기는 하지만, CAN-CORBA가 다른 표준 CORBA 구현들과의 연동을 향상시키기 위해서는 접속 지향 통신의 지원이 요구된다. <그림 6>은 두 개의 단방향 파이프를 사용하는 접속 지향 통신을 도식화 한 것이다. 접속을 통해 전송되어진 메시지는 접속 지향 통신을 나타내기 위해서 Proto에 10_2 값을 갖는다

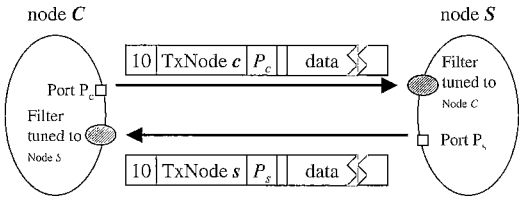


그림 6 단방향 파이프를 이용한 점대점 통신

5. 확장된 내장형 Inter-ORB 통신규약

CORBA의 원격 메소드 호출은 일반적인 ORB간 통신 규약(GIOP)을 통해 이루어진다. CORBA 2.2 GIOP는 CDR이라는 전송 구문 형식과 모든 요청/응답 경우을 수렴할 수 있는 8개의 메시지 형태를 정의한다. 그러나 GIOP는 메소드를 호출할 때마다 너무 많은 CAN 메시지 전송을 유발시키므로 본 연구에서 제안한 CAN- CORBA에는 부적절하다. 참고문헌[10]에서 우리는 CCDR이라고 하는 새로운 전송 구문 형식을 제안한 바 있으며, 우리의 inter-ORB 통신규약이 선택적으로 GIOP의 일부 메시지 형태를 지원하도록 하였다. 우리는 새롭게 정의한 inter-ORB 통신 규약을 내장형 inter- ORB 통신규약(EIOP)라 정의하였다. EIOP는 메시지 전송량을 효과적으로 줄일 수 있었지만, EIOP의 상호 연동성에도 많은 제약을 가져 왔다. 본 장에서는 CAN-CORBA의 CCDR에 대해 간단히 소개하고 가입 기반과 접속 지향 통신 모두를 지원할 수 있는 EIOP 메시지 설계와 상호 연동 향상 방안에 대하여 서술한다.

5.1 Compact Common Data Representation

CDR은 네트워크를 통해 GIOP가 IDL 데이타 형태를 송신하기 위해서, OMG IDL에서 정의한 데이타 형태를 네트워크 메시지 표현으로 사상하는 전송 구문 형식이다. 이와는 달리 CCDR은 압축 데이타 엔코딩과 가변길이 정수 엔코딩을 주요 최적화 기법으로 사용하였다. 따라서 CCDR의 압축 데이타 엔코딩은 32-bit 정수 경계에 자릿수를 맞춘(aligned) 정수 인스턴스(instance)를 필요로 하지 않으므로 공간을 채우기 위해 필요하였던 바이트(padding bytes)를 획기적으로 줄여준다. CCDR의 가변길이 정수 엔코딩의 경우, 하나의 정수가 표현하고자 하는 실제 값에 따라 1에서 5바이트 정도만을 차지하게 되는데 <표 1>은 이를 정리한 것이다. CDR에서는 정수가 4바이트에 저장되는 반면, IDL 프로그램의 대부분의 정수 인스턴스는 2^{32}-1 보다 작다. 예를 들어, CDR에서 정수는 매우 작은 IDL의 순차 데이타 형태와 문자열의 크기를 표현하는데 자주 사용된다.

표 1 Variable length integer encoding

two MSBs	SIZE (bytes)	Max. Value (unsigned)
00	1	2^6 - 1
01	2	2^{14} - 1
10	3	2^{22} - 1
11	5	2^{32} - 1

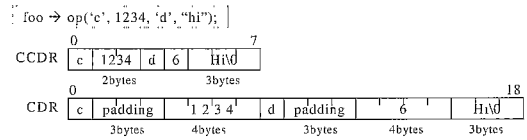


그림 7 CCDR encoding 예

<그림 7>은 메소드 호출 foo->op('c',1234,'d','' Hi'')가 CCDR로 엔코딩 예를 나타낸 것이다. <그림 7>의 예에서, CDR의 경우에 정수 1234와 3의 자릿수 정렬을 위해 필요한 6 byte가 CCDR에서는 불필요하므로 이를 절약할 수 있다. (정수 3은 내부적으로 문자열의 길이를 명세하는 데 사용됨) 또한 추가로 5바이트를 가변 길이 엔코딩 기법을 사용하여 줄일 수 잇다. 두 가지 방법의 결과를 통해, 매우 간단한 메소드 호출에서도 전체적으로 11 바이트를 절약 가능하고, 메소드 호출이 단일 CAN 메시지에 포함될 수 있음을 알 수 있다.

압축 엔코딩 방법은 메시지 엔코딩과 디코딩 및 노드의 추가적인 버퍼 공간 등의 오버헤드를 증가시킬 수 있다. 그러나 이러한 단점은 엔코딩된 메시지가 단일 CAN 메시지에 포함 될 수 있을 때 상쇄될 수 있으며, 이러한 경우는 내장형 제어 시스템 등에서 많이 볼 수 있다.

5.2 EIOP 메시지

CORBA에서 메소드 호출들은 네트워크를 통해 전송되어지는 메시지로 변환된다. <표 2>는 CORBA 2.2 GIOP가 지원하는 8가지 메시지 형태와 EIOP만이 지원하는 "Publish" 메시지 형태를 나타낸 것이다. "Publish" 메시지는 발표자/가입자 통신에서 발표자에 의해 비동기적으로 전송된 데이타를 담고 있다. 원래의 CAN-CORBA에서는, 요청 메시지가 데이타를 발표하는데 사용되어 졌다[10]. 따라서 <표 2>를 보면, 원래 CAN-CORBA는 요청 메시지 형태만을 지원하는 발표자/가입자 모델을 제공하는데 국한된다는 사실을 알 수 있다. 반면에, 새로운 CAN-CORBA는 5가지의 메시지 형태를 지원한다. "LocateRequest"와 "LocateReply"같이 지원되지 않는 GIOP 메시지 형태는 CORBA 객체

표 2 지원되는 EIOP 메시지 타입

Message type	EIOP P/S protocol	EIOP P-to-P protocol	GIOP	Originator
Publish	yes	no	no	Client
Request	no	yes	yes	Client
Reply	no	yes	yes	Server
CancelRequest	no	yes	yes	Client
CloseConnection	no	yes	yes	Server
MessageError	no	yes	yes	both
LocateRequest	no	no	yes	Client
LocateReply	no	no	yes	Server
Fragment	no	no	yes	both

의 이동을 위해서만 중요한 의미를 지니고, 실제로 내장형 제어 시스템의 경우 동적인 객체 이동이 거의 발생하지 않기 때문에 본 연구에서는 지원 대상에서 제외하였다.

CAN 메시지 전송량의 추가적인 감소를 위해서는, 공통적인 메시지 헤더와 형태 명세 메시지 헤더의 길이를 줄이는 것이 필수적이다. 따라서 본 연구에서는 발표 및 요청 메시지 헤더의 길이를 줄일 수 있는EIOP 메시지의 공통 헤더와 형태 명세 헤더의 자세한 내부 구조를 제안하였다.

6. 성능평가 및 결과

본 연구에서 제안한 CAN-CORBA는 GNU ORBit 버전 0.4.3을 사용하여 구현하였다[6]. 구현 시 사용된 하드웨어는 40Mhz의 내장형 프로세서인 i386 EX 가 장착된 3대의 PC와 인텔의 82527 CAN 콘트롤러가 장착된 KVASER의 PCcan 인터페이스 보드를 사용하였다. 본 시스템의 CAN 버스는 1Mbps의 데이타 전송율을 지니며, CAN 인터페이스 구동기와 mArx 실시간 운영체제와의 연결 부분은 직접 프로그래밍하여 실험에 적용하였다[15]. ORBit의 GIOP와 CDR은 EIOP와 CCDR 라이브러리로 대치했고 OMG의 최소 CORBA 명세[4]에 맞추기 위해 ORBit 상당부분 크기를 줄인였다. <표 3>은 하드웨어 소프트웨어 개발 환경을 요약한 것이다.

표 3 CAN-CORBA 구현에 사용한 하드웨어 및 소프트웨어 사용환경

Hardware
40MHz Intel 386 EX embedded processor (no cache) KVASER's PCcan CAN bus adaptor 2.0 [12] (Intel 82527 CAN controller [3])
Software
mArx real-time operating system [15, 16] CAN-CORBA (based on GNU ORBit 0.4.3) KVASER's PCcan device driver (ported onto mArx)

6.1 성능 평가 기준

CAN-CORBA는 두 가지의 중요한 설계 목표를 가진다. (1) 각 CORBA의 메소드 호출에 필요한 메시지 전송량 감소와 (2) ORB의 기억장치 요구량 최소화가 그것이다. EIOP의 간략화된 메시지 헤더가 메소드 호출 지연을 감소시키는 효과를 얻는 반면, CCDR이 압축 해제와 정수의 자릿수 재정렬 처리 오버헤드를 발생시킨다. 이러한 결과는 i386 EX 내장형 프로세서같은 저속 마이크로콘트롤러로 구축된 내장형 제어 시스템에서는 치명적인 문제가 된다. 따라서 본 논문에서는 구현된 CAN-CORBA 분석을 위해 다음과 같은 성능평가 기준을 사용한다.

- 통신규약 처리 지연시간 : CAN-CORBA에서 메시지 전송량의 절약은 일부 EIOP 메시지의 정렬(marshaling)과 비정렬(unmarshaling)을 포함하는 통신 규약 처리 오버헤드의 증가로 나타날 수 있다. 송신 측의 통신규약 처리 지연은 호출 stub, CAN 장치 구동기, 82527 CAN 콘트롤러의 실행 시간으로 정의된다. 수신 측의 통신 규약 처리 지연은 첫 번째 CORBA 메소드 호출의 CAN 메시지가 수신된 시간부터 골격 코드(skeleton code)가 불려질 때까지의 시간 간격으로 정의된다.

- 정적 메모리 사용 기록 : 본 논문에서는 CAN-CORBA의 정적 메모리 요구량을 측정하는데, 핵심 ORB와 이와 연관된 라이브러리의 코드와 데이타 섹션 용량을 모두 합한 것을 측정하게 된다. GNU glib V1.2.1은 CAN-CORBA와 ORBit을 위한 라이브러리이며 ACE는 TAO ORB 용 라이브러리이다.

6.2 EIOP 통신규약 처리 지연

본 논문에서는 발표자 측과 가입자측의 통신규약 처리 지연을 모두 측정하였다. 측정 결과를 종합하여 보면 <그림 8>과 같다. 측정 시 <그림 10>과 같은 온도 측정 코드를 사용하였는데, 코드에서 update_ temperature() 메소드는 char와 long 타입의 두 개의 매개변수를 가지고 있다. 이 실험에서 메소드 호출의 통신규약 처리 지연은 매개변수의 수가 증가하면 함께 증가한다. 따라서 메소드의 매개변수 수를 변화시키면서 서로 다른 통신 규약 처리 지연을 측정하였다. 가령 고정된 수의 매개변수를 사용하여도, 처리 지연은 CCDR이 가변길이 정수 엔코딩 기법을 사용하는 관계로 매개변수 값

```
// Publisher code in C++

// Define a channel tag for temperature monitoring.
#define TEMP_MONITOR_TAG 0x01

// Initialize the object request broker (ORB).
CORBA::ORB_ptr orb = CORBA::ORB_init(argc,argv);

// Get a reference to the conjoiner.
Conjoiner_ptr conjoiner = Conjoiner::_narrow(
    orb->resolve_initial_reference("Conjoiner"));

// Obtain a reference to the temperature monitor group
// TEMP_MONITOR_IFACE is an interface identifier generated
// by the IDL compiler.
TemperatureMonitor_ptr monitor =
    conjoiner->announce(TEMP_MONITOR_TAG, TEMP_MONITOR_IFACE);

while(1) {
    ...
    // Invoke a method of subscribers.
    monitor->update_temperature('A', value);
}
```

```
// Subscriber code in C++

// Define a channel tag for temperature monitoring.
#define TEMP_MONITOR_TAG 0x01

// Initialize the object request broker (ORB).
CORBA::ORB_ptr orb = CORBA::ORB_init(argc,argv);

// Get a reference to the conjoiner.
Conjoiner_ptr conjoiner = Conjoiner::_narrow(
    orb->resolve_initial_reference("Conjoiner"));

// Create a servant implementing a temperature monitor object.
TemperatureMonitor_impl monitor_servant;

// Assign a local CORBA object name to the monitor object.
PortableServer::ObjectId_ptr oid =
    PortableServer::string_to_ObjectId("Monitor1");

// Register the object name and servant to a portable object adaptor (POA).
poa->activate_object_with_id(oid, &monitor_servant);

// Bind the monitor object to the TEMP_MONITOR_TAG.
conjoiner->subscribe(TEMP_MONITOR_TAG, &monitor_servant);

// Enter the main loop where the monitor can receive temperature values
orb->run();
```

그림 8 Publisher/subscriber 예: 온도감시기

에 상당히 의존적이 된다. 측정을 하는 동안 최악의 처리 지연 값을 얻기 위해 매개변수가 가질 수 있는 가장 큰 값을 사용하였다. <그림 8>에서는 최악의 통신규약 처리 지연이 매개변수의 수가 매우 적을 때, 1ms 이하인 것을 보여준다(특히 매개변수가 6개일 경우). 이것은 대부분의 실제 응용에서 일반적인 특성이다[17]. 더욱 중요한 것은 순수한 EIOP 처리 지연이 전체 송신 측 통신 규약 처리 지연의 34.5%만을 차지하는 반면에, CAN 장치 구동기와

버스 어댑터가 각각 24.6%와 40.9%를 차지한다는 점이다. 실험을 통하여 평균적으로 EIOP는 GIOP 메시지 전송량의 37.5%를 감소시킴을 알 수 있다.

6.3 정적 메모리 요구량

본 연구에서는 CAN-CORBA와 GNU ORBit V0.4.3,

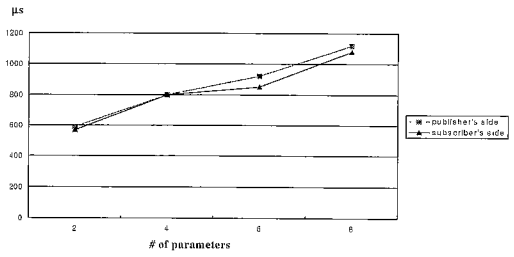


그림 9 프로토콜 처리 지연

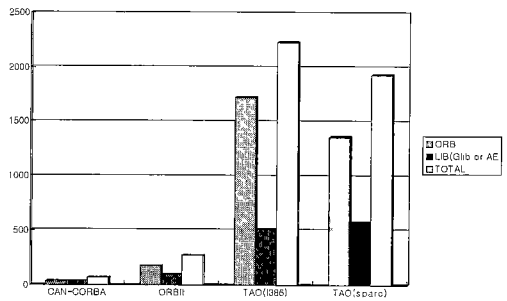


그림 10 3개의 ORB에 대한 정적 메모리 요구량 비교

그리고 GNU Size Utility를 실행시키기 위한 최소 TAO V1.0의 메모리 요구량을 측정하였다. 그 결과는 <그림 9>와 같다. 실험 결과는 GNU C 컴파일러 V2.8.1 Intel386 프로세서용으로 3개의 ORB를 구현하여 얻은 것이다. -03 -m386 -frepo 컴파일러 옵션으로 사용하였고, ORBit과 i386을 위한 최소 TAO는 Redhat Linux 5.1상에서 구현하였으며 CAN-CORBA는 mArx에서 구현하였다. <그림 9>의 TAO(sparc)값은 Sun Sparc 워크스테이션에서 동작하는 TAO의 메모리 크기를 나타낸 것이다. <표 4>에서 정리되어 나타난 바와 같이 본 논문에서 구현된 CAN-CORBA는 매우 작은 메모리만을 요구함으로 CAN bus와 같은 내장형 시스템에 사용되기에 적합하다.

7. 결 론

본 논문에서는 CAN 기반의 분산 제어 시스템을 위한 환경 명세 CORBA인 CAN-CORBA의 설계와 구현을 제시하였다.

CAN-CORBA의 핵심 ORB는 다음과 같은 특성을 지닌다. 첫째, CAN-CORBA의 ORB는 표준 CORBA의 전형적인 접속지향 점대점 통신뿐만 아니라 subscription기반의 그룹 통신을 지원한다. 결과적으로 분산

표 4 Memory requirements of CAN-CORBA and ORBit modules

Modules	Footprint(bytes)		Supported by minimum CORBA
	CAN-CORBA	ORBit	
IOP	7,493	19,305	yes
DII/skeleton	0	78,026	no
Dynamic any	0	32,932	no
POA	8,260	10,618	partially
Others	13,770	33,776	yes
Total	29,523	174,657	

된 시간-임계 감지 데이타를 위한 멀티캐스트 기능이 많이 요구되는 분산 제어 시스템에 더욱 적합하다. 두 번째, 각 CORBA 메소드 호출을 위해 요구되는 메시지 전송량을 상당히 줄일 수 있다. 이러한 특성은 CAN의 버스가 비록 저속일지라도 CORBA 메소드 호출의 오버헤드를 완화시킬 수 있게된다.

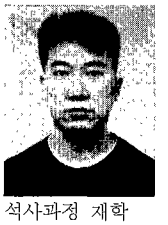
그러나, 메시지 전송량의 감소는 정수 압축 해제와 CCDR의 자릿수 재정렬 때문에 통신규약 처리 오버헤드의 증가를 초래 할 수도 있고, 이로 인해 효율적인 제어 없이는 i386 EX 내장형 프로세서 같은 저속 마이크로콘트롤러로 구축된 내장형 제어 시스템에서는 이 것이 치명적인 성능 문제로 대두될 수 있다. 본 연구에서 제안한 CAN-CORBA를 성능 관점에서 검증하기 위해, CAN 기반 분산 환경에서 이를 구현하고 몇 가지의 성능 평가를 수행하였다. 성능평가 결과, 불과 64.3Kbyte 정도의 메모리 요구량을 갖는 전송 측에서 평균적으로 700μs의 작은 메소드 호출 지연이 발생함을 보였다. 실험을 통하여 자원의 사용이 상당히 제한적인 분산 내장형 제어 시스템 구축에 CORBA를 사용하는 것이 가능하다는 결과를 얻었다.

현재 본 연구팀은 실시간 메시지를 위한 적정 시간 보장 및 집중된 결합자 객체를 위한 결함 허용 등을 지원하기 위해 CAN-CORBA를 확장하고 있다.

참 고 문 헌

[1] Allen-Bradley. DeviceNet specifications, release 2.0, Vol. I: Communication model and protocol, Vol. II: Device profiles and object library, 1997.
[2] Bosch. CAN specification, version 2.0, 1991.
[3] Intel Corporation. 82527 serial communications controller architecture overview, January 1996.
[4] Object Management Group. Minimum CORBA joint revised submission , OMG document orbos/98-08-01 edition, August 1998.
[5] CAN in Automation (CiA). Draft standard 301

version 3.0, CANopen, communication profile for industrial systems based on CAL.
[6] Red Hat Inc. ORBit, http://www.labs.redhat.com/orbit, 1999.
[7] ISO-IS 11898. Road vehicles interchange of digital information controller area network (CAN) for high speed communication, 1993.
[8] J. Kaiser and M. A. Livani. Invocation of real-ime objects in a CAN bus-system. In IEEE International Symposium on Object-oriented Real-time distributed Computing, May 1998.
[9] J. Kaiser and M. Mock. Implementing the real-time publisher/subscriber model on the controller area network (CAN). In IEEE International Symposium on Object oriented Real-time distributed Computing, May 1999.
[10] K. Kim, G. Jeon, S. Hong, S. Kim, and T.-H. Kim. Resource-conscious customization of CORBA for CAN based distributed embedded systems. In IEEE International Symposium on Object-oriented Real-time Computing, pages 34--41, March 2000.
[11] K. Kim, G. Jeon, S. Hong, T.-H. Kim, and S. Kim. Integrating subscription-based and connection-oriented communications into the embedded corba for the can bus. Technical Report SNU-EE-TR2000-??, School of Electrical Engineering, Seoul National University, March 2000.
[12] Sweden KVASER AB, Kinnahult. PCcan 2.0, September 1998.
[13] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The information bus -- an architecture for extensible distributed systems. In ACM Symposium on Operating System Principles, 1993.
[14] R. Rajkumar, M. Gagliardi, and L. Sha. The real-time publisher/subscriber inter-process communication model for distributed real-time systems: Design and implementation. In IEEE Real-time Technology and Application Symposium, June 1995.
[15] Seoul National University RTOS Lab. mArx: micro mArx, http://arx.snu.ac.kr, 1998.
[16] Y. Seo, J. Park, and S. Hong. Efficient user-level I/O in the ARX real-time operating system. In ACM Workshop on Languages, Compilers, and Tools for Embedded Systems, June 1998.
[17] Gonzalo Ulloa. Fieldbus application layer and real-time distributed systems. In IEEE International Conference on Industrial Electronics, Control and Instrumentation, IECON, October 1991.
[18] M. Horstmann and M. Kirtland, DCOM Architecture, http://msdn.microsoft.com/library/ default.asp?URL=/library/backgrnd/html/msdn_dcomarch.htm
[19] http://developer.java.sun.com/developer/technicalArticles/RMI/index.html



김 기 문
1993년 서울대학교 컴퓨터 공학과 졸업 (B.S.). 1998년 ~ 1999년 (주)나눔 기술 서버팀 연구원 1998년 ~ 1999년 서울대학교 전기공학부 실시간 운영체제 연구실 연구원. 1999년 ~ 현재 서울대학교 전기공학부 실시간 운영체제 연구실 석사과정 재학

김 태 형
1986년 서울대학교 컴퓨터공학과 졸업 (B.S.) 1988년 서울대학교 컴퓨터공학과 졸업 (M.S.) 1988년 ~ 1990년 현대전자산업(주) 응용S/W 개발부. 1996년 University of Maryland, Department of Computer Science (Ph.D.). 1996년 ~ 1999년 현대정보기술(주) 정보기술연구소 책임연구원. 1999년 ~ 2000년 한양대학교 컴퓨터공학과 전임강사