CAN-CORBA 시스템에서 실시간/비실시간 객체간의 원활한 통합

한양대학교 김태형*
서울대학교 홍성수**

1. 서 론

컴포넌트 기반 미들웨어 기술은 최근의 내장형 소프트웨어 시스템 개발상의 위기를 해결하기 위한 기본방향중 유력한 대안중 하나로 부각되고 있다. 하지만 많은 내장형 시스템은 실시간 요구사항을 가지고 있고, 고도의 신뢰성이 요구되는 실시간 환경에서의 작동을 목표로 하고 있으므로 이를 위한 미들웨어 시스템을 개발하여 실제의 응용에 활용한다는 것은 대단히 도전적인 작업이 아닐 수 없다. CORBA와 DCOM과 같은 컴포넌트 기반 미들웨어 기술을 내장형 시스템 개발에 적용하는 데 있어서 고려해야 할 중요한 문제중의 하나는 대부분의 내장형 시스템은 사용할 수 있는 자원이 매우 제한적이라는 것이다. 내장형 시스템에서 일반적인 데스크탑 환경에서와 같은 대용량의 메모리나 고속의 프로세서 처리를 기대할 수 없으며 심지어 작동시 소모전력에 대한 제한등 사용할 수 있는 자원에 뚜렷한 차이가 있다는 것에 주의하여야 한다.

최근 들어 CORBA, DCOM, Java RMI 등이 이러한 소프트웨어 위기를 해결할 수 있는 객체기반의 컴포넌트 기술로 각광받고 있지만, CORBA같은 일반 용도의 미들웨어 시스템은 몇가지 이유로 인해 내장형 실시간 제어 시스템에 바로 응용할 수 없는 이유를 다음의 세 가지로 나누어 볼 수 있겠다.

첫째, 다른 미들웨어뿐만 아니라 CORBA 또한 많은 자원을 필요로 한다. 특정 ORB(Object Request Broker) 구현이 경우에 따라 수 메가바이트 이상의 메모리를 요구하는 것을 쉽게 볼 수 있다. 이는 제한적인 자원을 가지고 있는 내장형 시스템에서 작동되기에 매우 어려운 문제를 야기하게 될 것이다. 둘째, CORBA는 구현된 시스템의 수행 동작을 예측하기 어려운 경우가 많은데, 이는 CORBA가 그 동작을 예측할 수 없는 상용 소프트웨어 시스템 위에 구축될 뿐 아니라 고도의 소프트웨어 계층을 가지고 있어서 특정 메시지에 대하여 전송시간등에 대한 확증을 주거나 예측하는 것이 매우 어렵게 되고 이는 실시간 제어 시스템 개발에 심각한 문제가 아닐 수 없다. 셋째, 접속 지향 통신 모델인 CORBA는 제어 시스템 응용에는 적합하지 않다. 일반적으로 많은 제어 시스템들에서는 요청이 없이도 데이터 수집 장치가 다중 콘트롤러에 측정된 데이터를 보내야함으로 멀티케스트(multicast) 통신 서비스가 필수적이다. 점대점(point-to-point) 접속지향 통신을 기본 통신 모델로 하고 있는 CORBA와 멀티캐스트 통신 모델을 기본으로 제공하고 있는 제어 시스템 하드웨어의 특징은 적절한 방법으로 통합하여야 한다.

위에서 제시된 표준 CORBA의 문제점을 해결하기 위하여, 우리는 CAN(Controller Area Network) 기반의 분산 제어 시스템을 예제 시스템으로 하여 새로운 내장형 CORBA를 설계한 바 있다. CAN은 전세계적으로 자동차 산업 전반에서 널리 사용되고 있는 산업용 실시간 네트워크의 표준이다. 이것은 고작 최대 1Mbps의 전송 대역폭과 8 bytes 길이의 데이다 단위를 사용하기 때문에 CAN 기반의 분산 환경에서

* 정 회 원
* 종신회원

CORBA 응용 프로그램을 실행하는 것 자체가 상당히 까다로운 일이 아닐 수 없다. 최근에 본 저자들은 한정된 자원을 가지고 있는 (예, 매우 적은 메모리 footprint, 좁은 대역폭을 갖는 네트워크 환경 등) CAN 버스 환경을 목표 시스템으로 하여 컴포넌트 기반의 응용 프로그램 개발을 가능하게 하기 위한 CAN-CORBA 시스템을 개발하였다. CAN 버스에서 작동할 수 있는 CORBA ORB를 개발하기 위해 주제기반 통신 방법(subject-based communication scheme), EIOP(Embedded Inter-ORB Protocol), 및 CCDR(Compact Common Data Representation)의 다양한 새로운 기법을 사용하였으며 이에 대하여 최근 학술회의에 보고된 바 있다 [3,5,6].

본 논문에서는, 내장형 소프트웨어에서 시간적인 제약점(timing constraints)이 있는 객체와 없는 객체가 동시에 존재하는 상황이 빈번한 현실에 주목하여, 두 종류의 객체들을 CAN-CORBA 시스템상에서 원활하게 통합시켜 하나의 응용 프로그램으로 수행되는 것을 가능하게 하는 방법을 살펴보도록 한다. 시간적 제한을 가지고 있는 객체가 CAN-CORBA ORB가 요구하는 객체간 통신 방법을 따라 메시지를 전송할 경우 시간적 제한을 보장하기 어려우므로 실시간 객체일 경우 "fast-track message"를 가능하도록 하고 또한 이것이 별도의 개발과정을 갖게 함으로써 개발상의 복잡성을 발생시키지 않게 하기 위해 "확장된 IDL(extended IDL)" 명세를 통하여 일반적인 CORBA 응용 프로그램 개발 과정과 일치시키는 것이 가능함을 제시하려고 한다.

본 논문은 먼저 목표 하드웨어 시스템의 구성을 설명한 후, CAN 버스라고 하는 특수한 상황에서 CORBA의 기능을 작동시키기 위해 새로 설계한 트랜스포트 프로토콜을 간략히 설명하고 일반적인 IDL(Interface Description Language)에서 시간적 제약점을 기술하기 위해 확장한 xIDL을 이용하여 이들 제약이 표현되는 방법을 설명하면서 이렇게 표현된 실시간 객체가 전체 CORBA 환경에서 다른 비실시간 객체들의 개발 과정과 원활하게 통합되어 설계되는 것이 가능하다는 논의하고 결론을 맺으려 한다.

2. 시스템 모델

본 연구에서 제안한 CAN-CORBA는 CAN 버스[1] 상에 구축된 분산 내장형 제어 시스템에서 동작하도록 설계되었다. 본 장에서는 하드웨어 플랫폼의 제약과 특징을 쉽게 이해할 수 있도록 대상 시스템 하드웨어 모델을 소개한다.

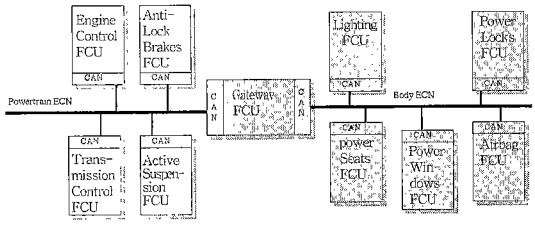

그림 1 내장형 분산 제어 시스템(승용차 제어 시스템의 예)

2.1 목표 시스템 하드웨어 모델

대상 하드웨어는 내장형 제어 네트워크로 연결된 여러 개의 기능 제어 유닛(FCU)으로 구성된다. 그림 1은 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 데이터 생산자나 소비자 또는 생산·소비자의 역할을 수행하게된다.

그림 1에서 내장형 제어 네트워크(ECN)는 저렴한 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[1]을 내장형 제어 네트워크로 선택하였다. CAN 표준안은 OSI 참조 모델의 물리층와 데이터 링크층 통신 규약을 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 데이터, 오류, 승인 그리고 CRC 항목으로 구성된다, 식별자 항목은 CAN

2.0A의 11-bit 또는 CAN 2.0B의 29-bit로 구성되며 데이터 항목을 최대 8 바이트까지 확장 가능하다. CAN의 네트워크 어댑터는 메시지 전송시, 먼저 식별자 항목을 전송하고 그 다음 데이터 항목을 보내게 되는데, 메시지의 식별자는 우선 순위의 역할을 하게 되며, 이때, 높은 우선 순위의 메시지가 낮은 우선 순위의 메시지보다 우선권을 지니게 된다[1,2].

CAN은 주제기반주소기법[7,9] 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 데이터의 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 주소기법은 본 논문의 CAN-CORBA에서 기본이 되는 통신 방법이다.

2.2 CAN-CORBA에서의 통신 방법

CAN-CORBA가 제공하는 주제기반주소기법[5]은 "blindcast" 혹은 publisher/subscriber 모델이라고 불리는 방법으로서 Rajikumar 등에 의해 실시간 분산 제어 시스템용으로 제안되었고[9], Kaiser와 Mock에 의해 CAN 버스상에서 최초로 적용된 바 있다[4]. CAN-CORBA에서는 "호출 채널(invocation channel)"과 "결합자(conjoiner)"를 이용하여 기본적인 통신이 이루어지도록 하였는데, 이는 점대점 통신을 기본으로 하는 일반 CORBA와 통신 하드웨어상 브로드캐스트를 기본 모델로 하는 CAN 버스의 상이점을 해결하는 중요한 방법이다.

호출 채널은 발표자(publisher)들로부터 가입자(subscriber) 그룹과의 가상 멀티캐스트 채널이며, 발표자는 한 포트를 호출 채널에 접속한 후 가입자를 위한 호출 채널을 선언한다. 포트는 그림 2와 같이 TxNode와 TxPort 쌍으로 이루어진 유일한 주소 형태의 전송층(transport layer) 개체이다. 호출 채널은 하나 이상의 접속된 포트를 가질 수 있다. 발표자는 접속된 포트를 사용하여 미리 선언한 호출 채널을 통해 메시지를 전송할 수 있으며, 가입자는 호출 채널을 통해 메시지를 수신할 수 있다. 또한 결합자는 발표자와 가입자가 서로의 존재를 알 수 없으므로 이를 중개해 주는 역할을 한다. 이것은 시스템에 존재하는 모든 발표자와 가입자에게 잘 알려진 식별자를 가진 특별한 노드에 존재한다. 이렇게 하지 않을 경우 결합자 노드와의 통신을 위해서는 별도의 중개가 필요없도록 하기 위한 것이다.

CAN-CORBA는 또한 다른 표준 CORBA에서 기본으로 되어 있는 점대점 방식의 통신 모델도 지원한다. CAN 버스가 기본적으로 브로드캐스트 네트워크이므로 점대점 방식의 통신을 구현하여 실행하면 매우 비효율적이다. 그럼에도 불구하고 이 방식을 제공하는 이유는 데이터의 상호운용성(interoperability) 때문이다. CAN 버스 내에서만 작동한다면 위에서 설명한 주제기반의 발표자/가입자 통신 모델이 더 자연스럽고 효율적이지만, 표준 ORB에서 작동하는 객체와의 통신을 위해서는 점대점 방식도 지원되어야 하기 때문이다. 일반 표준 ORB들은 주제기반 통신 모델을 지원하지 않으므로 점대점 통신을 CAN-CORBA가 지원하지 않는다면 상호운용성상의 문제가 발생할 것이다.

2.3 CAN-CORBA의 전송 계층 프로토콜

CAN 표준은 OSI 참조 모델상에서 물리적 계층 프로토콜과 데이터 링크 계층 프로토콜만을 지원한다. CORBA ORB 구현을 위해서는 전송계층 프로토콜(transport layer protocol)이 반드시 필요하다. 따라서 본 연구에서는 그림 2와같은 프로토콜 헤더 지정을 이용하여 전송계층

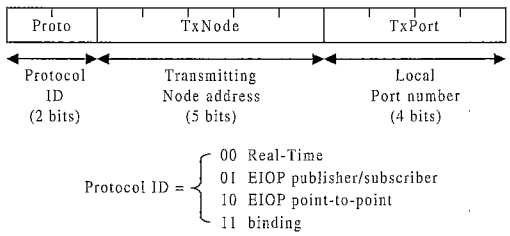

그림 2 CAN 식별자 구조를 이용한 통신 규약 헤더 포맷

프로토콜을 지원하도록 하였다.

그림에서와 같이 프로토콜 헤더는 프로토콜 식별자 (Proto), 전송 노드 주소 (TxNode), 그리고 포트 번호(TxPort)의 세부분으로 구성되며 각각 2 비트, 5 비트 및 4 비트를 차지한다. Proto 항목은 상위 계층의 통신규약 식별자 정보를 표현하며, CAN 메시지 상에서의 식별자에 이은 데이터 항목은 Proto가 가지고 있는 상위 계층의 통신 규약 식별자 따라 다른 형식을 취한다. Proto 항목은 2비트로 구성되므로 모두 4가지 값 11, 10, 01, 00을 가질 수 있다. "binding"은 버스를 통하여 전송되는 메시지가 결합자(conjoiner)를 이용한 통신 채널 바인딩을 위해 사용되는 경우이며, "01"과 "10"은 각각 EIOP(Embedded Inter-ORB Protocol)의 발표자/가입자 방식과 점대점 클라이언트/서버 통신 방식으로 메시지가 전송되고 있을 경우의 메시지 헤더중 Proto 항목이 갖는 값이다. 특별히 실시간 메시지를 위해서는 "00" 값을 사용하도록 하였다. CAN 버스는 식별자가 작은 값을 가질 경우 더 높은 전송우선순위를 갖게 되므로 위의 네 가지 값은 사실상 우선순위를 고려하여 결정한 값이다. 즉, 실시간 메시지가 최우선으로 전송되며, 점대점 클라이언트/서버 통신은 앞에서 설명한 바와 같이 단지 상호운용성을 위해 제공되는 것이고 보통의 경우에는 발표자/가입자 방식의 통신을 사용하게 될 것이므로 더 낮은 우선순위로 결정한 것이다. 객체간 전송할 메시지가 많이 있을 경우 새로운 통신 채널 형성을 위한 바인딩 메시지는 우선순위가 더 낮아져야 하므로 최하위 우선순위를 갖도록 설계하였다.

특별히 Proto 항목이 00인 경우, 즉, 실시간으로 전송되어야 하는 긴급한 메시지인 경우에는 TxNode와 TxPort에 해당하는 9비트는 각각 전송노드주소와 포트 번호를 나타내지 않고 CAN 버스용 식별자 고유의 역할(즉, 전송될 메시지의 우선순위를 결정하는 것)만을 담당함에 주의할 필요가 있다. 이러한 긴급한 메시지를 "fast-track" 메시지라고 하기로 하자.

2.4 Fast-track 메시지와 CAN-CORBA 메시지의 비교

Fast-track 메시지와 CAN-CORBA 메시지는 모두 궁극적으로는 같은 CAN 버스용 메시지 포맷이지만 전체 소프트웨어 버스 계층 안에서 전혀 다른 전송 경로를 갖게 된다. CAN-CORBA 메시지는 CAN-CORBA ORB와 채널 바인딩을 통한 호출 채널 등 많은 소프트웨어 추상화 계층간의 단계를 거치게 되므로 전송시간에 대한 어떠한 보장이 불가능하며 예측가능하지도 않다. 이와같은 상황에서 예측가능성과 시간에 대한 어떠한 보장을 요구하는 실시간 객체간의 메시지 전송에 CAN-CORBA 전송 프로토콜을 따르도록 하는 것은 실시간 응용 프로그램 구성상의 심각한 문제점이 될 수밖에 없다. 따라서 우리는 "fast-track" 메시지라고 하는 긴급한 형태의 메시지를 고안하였다. 이것은 모든 소프트웨어 버스 구성상의 추상화 단계를 거치지 않고 CAN 버스 하드웨어로 직접 연결된다. 따라서 "fast-track" 메시지의 헤더는 CAN 버스용 식별자가 바로 되도록 하는 것이다. 요약하면, fast-track 메시지는 일반 CAN 프로그래머가 개발 도구 없이 직접 사용하는 CAN 메시지이며, CAN-CORBA 메시지는 미들웨어 구현상의 모든 의미를 다 포함하고 있는 CORBA 메시지이다. 모든 CORBA 메시지는 결국 여러 단계를 거쳐 CAN 메시지로 변환된다.

3. 시간적 제약을 표현하기 위한 확장된 IDL

객체의 상호연결 작업은 개별 객체를 구현하는 것과는 본질적으로 다르며 구별되어야 하는 지적인 활동으로 이해되고 있다. C++나 Java와 같은 모듈 구현 언어와는 달리 IDL은 클라이언트 객체들간의 호출이나 객체가 제공하는 구현 메쏘드의 인터페이스를 표현하기 위해 사용되는 언어이다[8]. 그러므로 IDL로 쓰여진 인터페이스 정의는 모듈 구현에 관한 세부사항에 관련된 것이 아닌 객체 상호간 통신만을 완전하게 정의하고 있다.

분산 제어 시스템에서 실시간 컴포넌트는 태스크 단계의 end-to-end 시간적 제약점들로 열거될 수 있기 때문에 그러한 시간적 제약점들을 IDL에서 객체 상호간 행동의 세부사항의 하나로서 표현하는 것이 정당화될 수 있다. 기존의 IDL은 이러한 요소들을 표현할 이유가 없었으므로

실시간 요소들을 표현하기 위해 IDL을 확장하여 사용할 수 있다.

이것은 C++, Java 등과 같은 일반 모듈 구현 언어에 시간적 제약점을 특별히 표현하기 위한 실시간 프로그래밍 언어를 사용하도록 하는 것보다 훨씬 나은 접근방법이다. 왜냐하면 IDL수준의 수정이 새로운 컴파일러를 작성하는 것보다 훨씬 수월할 뿐 아니라, 모듈의 기능과 모듈간의 시간적 제약점을 포함한 작동 행태가 제어 프로그램의 작성과 수정을 쉽게 하기 때문이다. 모듈 기능상의 변화 없이 단지 시간적 제약점만이 변화되었을 경우 프로그래머는 원시 코드를 변경하여 다시 컴파일할 필요가 없으며 단지 IDL상의 인터페이스 정의를 다시 작성하면 된다.

아래에서 우리는 소위 "확장된 IDL(xIDL: eXtended IDL)"에서 어떻게 이와같은 시간적 제약을 기술할 수 있는지, 그리고 CAN-CORBA에서 xIDL을 이용하여 실시간 객체와 비실시간 객체들로 구성된 내장형 분산 제어용 응용 프로그램을 개발할 수 있는지를 살펴보도록 한다.

3.1 실시간 응용 모델

본 논문에서 다룰 실시간 분산제어 응용 프로그램의 구성 모델을 간략히 언급하고자 한다. 실시간 응용 프로그램은 실시간 메시지(전형적인 제어용 메시지)의 정적인 집합으로 구성되어 있고, 각각의 메시지는 정적으로 우선 순위가 주어졌다고 가정한다. 각각의 실시간 태스크용 객체 또한 특정한 CAN 노드에 정적으로 지정되어 존재한다. 메시지 수신 태스크는 관련이 없는 메시지를 필터링하는 CAN 버스 메카니즘을 통과한 후 CAN 버스 디바이스 콘트롤러에 의해 발생하는 인터럽트 처리 루틴에 의해 호출된다. 송신 태스크는 주기적으로 비실시간 메시지보다 우선 순위가 높은 control 메시지를 fast-track 메시지의 형태로 broadcast한다.

Tindell과 Burns에 의하면, 만약 각각의 메시지가 제한된 크기와 제한된 rate를 가지고 있다면, hard 실시간 메시지의 최악조건 응답시간을 제한하는 것이 가능하다는 것이 밝혀졌다[10]. 따라서 위와 같은 가정하에서 실시간 메시지를 fast-track의 형태로 전송한다면 최악조건 응답시간을 미리 결정할 수 있으므로 실시간 응용 프로그램 구성이 가능하다는 것을 알 수 있다.

이와 같은 모델에서, 프로그래머가 schedulability 확인 도구나 혹은 어떠한 종류의 고정 우선 순위 스케쥴링 방법의 도움을 받아 실행하기 전에 미리 주기적인 메시지 집합에 대한 실행 가능한 schedule을 가지고 있고 또한 모든 필요한 실시간 요소들이 미리 결정되어 있다면 xIDL로 이러한 실시간 설계 결과를 표현할 수 있다는 것이다. 때때로 발생하고 불규칙하게 발생하는 메시지들을 가진 동적 스케쥴링 방법은 여기에서는 논외로 하기로 한다. 정적인 방법만 제공한다고 해도 실시간 응용 시스템 개발자는 많은 도움을 받을 수 있고, 동적인 환경을 제공하는 것은 별도의 확장이 필요하기 때문이다. 현재까지 CAN 프로그래머들은 미들웨어 기능이 지원되지 않은 환경에서 직접 CAN 버스의 디바이스 컨트롤러의 기능을 이용하여 프로그램 해 왔다는 것을 기억하여야 할 것이다.

```
interface TemperatureMonitor {
    oneway void update_temperature(in char
locationID, in long temperature)
    with (period = 200, deadline = 150,
release_time = 10, priority = 15) raises
(DeadlineMissed);
}
```

그림 3 실시간 메쏘드를 위한 xIDL 정의

3.2 확장 IDL 세부사항

xIDL은 그림 3에서와 같이, 주기(period), 수행 완료 제한 시간(deadline), 메시지 릴리스 제한 시간(release time), 우선순위(priority)의 네 가지 시간적 제약으로 구성되는 하나의 with 절(clause)을 각 메쏘드마다 포함하도록 확장하였다. 하나의 인터페이스는 메쏘드들과 속성의 집합으로 구성된다. 한 인터페이스는 비실시간(즉, 보통의 CORBA) 메쏘드과 실시간 메쏘드을 갖는 것에 아무런 문제가 없다. 단지 실시간 방법만 시간전 제약점을 가지고 있을 뿐이다. 즉, with-절을 가지고 있는 메쏘드가 실시간 객체를 구동시키는 메쏘드이다. 그림 3에서 period, deadline, release_time, priority를 가지고 있는

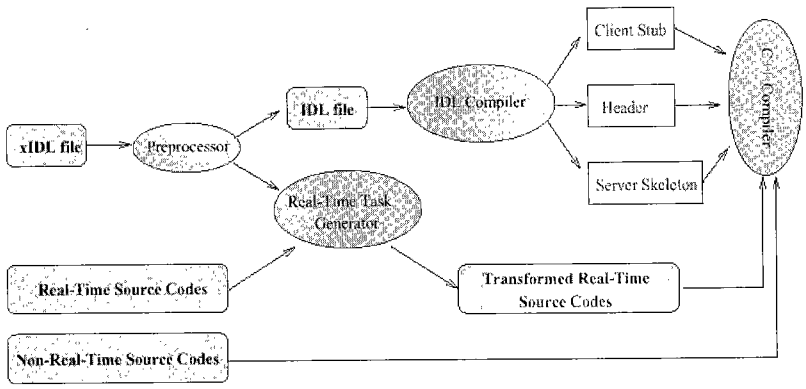

그림 5 실시간 CAN-CORBA에서 xIDL로부터 실시간 응용 프로그램 개발 과정

update_temperature 메쏘드는 실시간 메쏘드임을 보여준다. 이 메쏘드는 매 200 millisecond마다 150 millisecond보다 늦지 않고 10 millisecond보다 빠르지 않은, 우선 순위 15를 가진 실시간 제어 메시지를 방출한다. CAN-CORBA는 전체에서 우선 순위를 512(즉, 29)까지 표현할 수 있다. 따라서 우선 순위는 [0..511]의 범위를 가진다. CAN 2.0A identifier field는 11 bit로 구성되어있고 최대 유효 비트 두 개는 프로토콜 식별자(Proto)를 표시하기 위해 예약되어 있음을 상기하라. 만약 하나의 제약이라도 어기게 되면, 예외를 발생시킨다.

3.3 실시간 응용 프로그램 개발 과정

그림 4는 실시간 CAN-CORBA에서 xIDL 세부사항 수준에서 시작해서 CAN 응용프로그램 개발의 전 과정을 보여주고 있다.

만약 xIDL 정의가 보통의 IDL 표현과 같이 with-절로 된 시간적 제약점을 포함하고 있지 않다면 실행 프로그램 생성경로는 보통의 CORBA[8] 혹은 CAN-CORBA[3,5,6]에서 이루어 지는 것과 정확하게 일치한다. 그림의 "Preprocessor"는 입력된 xIDL과 동일한 IDL을 생성해 내게 된다. 그러나 만일 xIDL에 with-절로 된 시간적 제약점이 표현되어 있으면 전처리기는 with-절을 제외한 일반 IDL 파일을 생성해내고, 시간적 제약요소들을 실시간 태스크 생성기(Real-Time Task Generator)로 보내어 해당 제약조건을 포함하는 실시간 프로그램을 합성하게 된다. 즉, IDL 컴파일러를 위해 xIDL로부터 일반 IDL을 만들어 내고, with-절 정보는 고정된 우선순위 스케줄링 기법에 의한 "fast-track" 메시지를 생성할 수 있는 프로그램은 합성하기 위해 사용되는 과정을 거친다. 실시간 태스크 생성기는 실제적으로 소스코드 단계의 변환을 수행한다.

예를 들어, 센서에 의해 감지된 온도 메시지를 전송하는 한 객체가 있다고 가정하자. 또한 주어진 비율로 주기적인 메시지를 전달하도록 xIDL로 표현된 인터페이스 정의가 있다고 가정하자. 그러면 with-절을 가진 메쏘드는 실시간 태스크를 갖는 객체가 되어야 하고, 그 메시지들은 다른 비실시간 CAN-CORBA 메시지들보다 높은 CAN 버스 충돌 중재 하드웨어가 우선적으로 처리할 수 있도록 높은 우선순위를 가져야 한다. 또한 여러 실시간 task들간의 메시지들은 주어진 고정된 우선 순위 schedule에 대한 우선 순위들에 따라서 정렬되어야 한다. 이러한 우선순위는 실시간 프로그램 개발 도구에 의해 결정되거나 숙련된 실시간 프로그램 개발자에 의해 주어지는 것이므로 실시간 CAN-CORBA 시스템 외부에서 결정되어 오는 정보이다. 단, 이를 위해 실시간 객체를 생성하기 위한 소스 코드 xIDL에 표현된 시간적 제약점들을 만족시키는 프로그램이 변환생성되기 위한 과정을 거친다. 이런 방법으로, 실시간 통신은 응용 프로그래머의 xIDL 수준에서의 최소한의 변경으로 CAN-CORBA의 일반 통신 방법들과 원활하게 통합 개발되어 질

수 있다. 약간의 실시간 객체를 지원하기 위해 전혀 다른 별도의 프로그램 개발 과정을 밟을 필요는 없으며 분산제어 프로그램 개발자들도 CORBA와 같은 고급 프로그램 개발 환경의 도움을 받을 수 있다는 것을 보여주는 것이다.

4. 실시간 태스크 생성기

그림 4의 실시간 태스크 생성기(Real-Time Task Generator)의 역할을 좀 더 자세히 살펴보도록 한다. 우리는 일반적인 형태의 with-절, with (priority = p, period = t, deadline = d, release_time = r)의 명세를 갖는 메쏘드에 대한 실시간 태스크를 어떻게 생성할 수 있는지를 보임으로써 실시간 통신과 일반 CORBA 통신과의 원활한 통합이 가능하다는 것을 구체적으로 보이고자 한다.

4.1 Fast-Track 메시지를 위한 메시지 식별자 준비

메시지 식별자는 CAN 버스에서 메시지의 우선 순위를 결정한다. 고정된 우선 순위 스케줄링 알고리즘을 통해 각 메시지의 우선순위 p는 이미 결정되어 있음을 앞에서 설명하였으며 이것을 CAN 통신규약상의 메시지 헤더로 표현해 주면 된다.

모든 비실시간 메시지보다 높은 우선순위를 갖게 하기위해 fast-track 메시지를 위한 헤더 메시지의 최고위 2개 비트는 002 값을 갖도록 설계되었다. 따라서 with-절의 priority 속성값으로 주어진 정수값을 0과 511사이의 고정된 범위를 갖는 값으로 변환하여 주면 된다. 이보다 더 많은 우선 순위는 표현할 수 없겠지만 512개 이상의 서로 다른 우선 순위를 갖는 실시간 제어 프로그램은 실제 상황에서는 거의 발생하지 않으므로 문제가 되지 않을 것이다.

```
int k = 1;
while (1) {
    start = gettime();
    // calculate the message contents
    msg = ReadTemperature();
    // keep the release time spec.
    while (gettime() < start + r);
    if (gettime() > t * k + d)
        DeadlineMissed()l
    while (gettime() <= k * t);
    k++;
}
```

그림 5 변환생성된 실시간 태스크 소스 코드

4.2 수행 코드의 생성

그림 5에서 프로그래머에 의해 작성된 프로그램은 시간에 관련된 부분이 전혀 없이 순수하게 모듈의 기능만을 수행하는 부분으로만 구성한다. 즉, 모듈의 기능과 모듈간의 상호작용으로 생기는 문제를 구별하여 프로그래밍하도록 하는 것이 보다 효율적이라는 소프트웨어 공학상의 기본 전제를 바탕으로 한다. 그림에서 모듈의 기능에 해당하는 부분은 "msg = ReadTemperature()" 라는 함수로 표현되어 있다. 나머지 부분들은 모두 실시간 제약을 실제로 구현하기 위해 xIDL에서 제공된 정보를 이용하여 실시간 태스크 생성기에 의해 변환 생성된 결과이다.

그림 5의 코드는, 온도 센서가 온도를 읽어 온 후부터 (ReadTemperature()) 함수에 의한 결과, release time r이 지난 후 메시지를 실제로 전송하는 것을 보여준다. 여기서 send() 함수는 실시간 메시지를 fast-track 형태로 적절한 우선 순위를 갖는 CAN message 헤더를 구성하여 버스에 데이터를 upload하는 함수이다. 전송이 완료되면 deadline을 만족시키지 못하였을 경우 DeadlineMissed()라는 예외상황 처리 함수를 호출한다. 제대로 수행되었으면 주어진 주기 t가 경과한 후 다시 온도 센서로부터 읽은 온도를 전송하는 과정을 반복한다. 대부분의 실시간 제어 프로그램들은 이와 같은 주기적으로 반복되는 메시지를 발생시키는 프로그램으로 구성되어 있다.

여기서 실시간 프로그램을 개발할 때 p, t, d, r과 같은 속성 값을 간단하게 결정해 주는 개발 과정은 현실적으로 존재하지 않는다. 몇몇 스케줄링 알고리즘을 이용하고 개발자가 가지고 있는 오래된 경험을 이용하여 결정하게 되는 것이다. 다시 말하면 여기서 주어진 속성값은 실제로 수행시켜 보았을 때 시스템 요구사항을 만족시키지

못하는 경우가 얼마든지 생길 수 있다. 본격적으로 시스템을 출하하기 이전에 시행착오를 거쳐 값이 최종 결정될 수도 있을 것이고 또 제어 프로그램의 유지, 보수 과정에서 새로운 값을 가질 수도 있는 것이다. 새로운 속성값을 가질 때마다 해당하는 코드를 일일이 찾아 변경하는 것은 현실적으로 바람직하지 않으며 그렇지 않아도 복잡한 프로그램 개발 과정을 더욱 복잡하게 할 뿐이다.

본 논문에서 제안한 방법은 프로그래머가 미들웨어 환경을 이용할 수 있을 뿐 아니라, 실시간 요구사항 변경시 IDL 단계에서 손쉽게 변경이 가능하므로 개발 과정을 크게 단순화하여 내장형 소프트웨어 개발 위기를 극복하는 데 많은 도움을 줄 것으로 생각된다.

5. 결 론

내장형 분산 제어 소프트웨어 개발은 복잡하고, 시간이 많이 필요하며, 에러가 발생하기 쉬운 작업이다. 유용한 소프트웨어 공학적 지원이 소프트웨어 위기에 대처하기 위해 필수불가결한 요소이지만 여러 제한을 갖는 내장형 시스템의 속성상 단순한 적용이 어려움을 지적하였다. 결과적으로 분산제어 프로그램 개발자들은 소프트웨어 개발 도구의 지원 없이 복잡한 작업을 수행하여 왔다. CAN-CORBA 시스템은 내장형 시스템 개발자들도 컴포넌트기반 미들웨어 시스템의 풍부하고 유용한 기능들을 사용하는 것을 가능하게 하였다.

CAN-CORBA 시스템을 사용할 경우 개발자들은 고급의 프로그램 개발 환경의 혜택을 누릴 수 있게 되었으나 결과적으로 실시간 메시지를 예측가능한 형태로 작동시키는 것에 관한 보장을 할 수가 없게 된다. 미들웨어 시스템은 소프트웨어 구조적인 본질상 매우 복잡한 계층을 갖게 되고 이 모든 계층을 통과하면서 최악조건 응답시간을 보장하는 시스템을 설계하는 것은 현실적으로 불가능하기 때문이다. 따라서 본 논문에서는 실시간 메시지 전송을 위해서는 응답시간을 보장하기 위해 여타 비실시간 메시지보다 높은 우선순위를 갖는 메시지를 발생시켜서 CAN-CORBA ORB를 관통하여 직접 CAN 버스를 이용할 수 있도록 하였다. 이렇게 하였을 경우 별도의 프로그램 개발 과정을 만들게 되어 새로운 복잡성을 발생시키지 않기 위해 원활하게 실시간/비실시간 통신이 통합되는 과정을 확장된 IDL의 개념을 이용하여 설명하였다.

이러한 노력을 계속한다면 내장형 소프트웨어 개발자들도 low-level programming 단계를 벗어나, 최신 소프트웨어 개발 기술 경향인 컴포넌트 기반 방법론을 사용하는 것이 가능할 뿐 아니라, 그렇게 하는 것이 소위 말하는 "embedded software crisis"에 대처할 수 있는 합리적인 대안임을 다시 한번 강조하고자 한다.

참고문헌

[1] Bosch (1991). CAN specification, version 2.0.

[2] ISO-IS 11898 (1993). Road vehicles-interchange of digital information-controller area network (CAN) for high speed communication.

[3] Jeon, G., T. Kim, S. Hong and S. Kim (2000). A fault tolerance extension to the embedded corba the can bus systems. In: ACM SIGPLAN 2000 Workshop on Languages, Compilers, and Tools for Embedded Systems.

[4] Kaiser, J. and M. Mock (1999). Implementing the real-time publisher/subscriber model on the controller area network (CAN). In: IEEE International Symposium on Object-oriented Real-time distributed Computing.

[5] Kim, K., G. Jeon, S. Hong, S. Kim and T. Kim(2000a). Resource-conscious customization of CORBA for CAN-based distributed embedded systems. In: IEEE International Symposium on Object-Oriented Real-Time Computing.

[6] Kim, K., G. Jeon, S. Hong, T. Kim and S. Kim(2000b). Integrating subscription-based and connection-oriented communication into the embedded corba for the can bus. In: IEEE Real-time Technology and Application Symposium.

[7] Oki, B., M. Pfluegl, A. Seigel and D.

Skeen (1993). The information bus - an architecture for extensible distributed systems. In: ACM 14th Symposium on Operating System Principles.

[8] OMG (1998). The Common Object Request Broker: Architecture and specification revision 2.2.

[9] Rajkumar, R., M. Gagliardi and L. Sha(1995). The real-time publisher/subscriber interprocess communication model for distributed real-time systems: Design and implementation. In: IEEE Real-Time Technology and Application Symposium.

[10] Tindell, K. and A. Burns (1994). Guaranteeing message latencies on controller area network(can). In: Proceedings of the International CAN Conference.

김 태 형

1986 서울대학교 컴퓨터공학과 졸업(B.S.)
1988 서울대학교 컴퓨터공학과 졸업(M.S.)
1988~1990 현대전자산업(주) 응용S/W 개발부
1996 University of Maryland, Department of Computer Science(Ph.D.)
1996~1999 현대정보기술(주) 정보기술연구소 책임연구원
1999~2000 한양대학교 컴퓨터공학과 전임강사
E-mail:tkim@hanyang.ac.kr

홍 성 수

1986 서울대학교 컴퓨터공학과 졸업(B.S.)
1988 서울대학교 컴퓨터공학과 졸업(M.S.)
1988~1989 한국전자통신연구소 (연구원)
1994 University of Maryland, Department of Computer Science(Ph.D.)
1994~1995 University of Maryland, Department of Computer Science(Faculty Research Associate)
1995 Silicon Graphics Inc.(Member of Technical Staff)
1995~1997 서울대학교 전기공학부 전임강사
1997~현재 서울대학교 전기공학부 조교수
E-mail:sshong@redwood.snu.ac.kr