主題

RSCA: 분산 로봇 플랫폼에서 임베디드 소프트웨어의 동적 재구성을 지원하는 통합 미들웨어

서울대학교 전기.컴퓨터공학부 교수 홍성수

차 례

요 약

정보통신부에서는 기술 혁신을 통해 로봇 시스템 보급과 실용화를 앞당기기 위하여 URC 로봇 프로젝트를 진행하고 있다. 본고에서는 URC 로봇의 응용 소프트웨어를 위한 표준 시스템 소프트웨어 구조로 개발 중인 RSCA를 소개한다. RSCA는 로봇 응용 소프트웨어에게 표준화된 운영 환경을 제공하고, 이들의 개발을 용이하게 하는 프레임워크도 제공한다. 구체적으로 RSCA의 운영 환경은 실시간 운영체제, 분산 미들웨어, 배치 미들웨어의 3 계층으로 정의된 구조를 갖는다. 여기서 실시간 운영체제는 다양한 하드웨어 디바이스 위에서 로봇 응용을 신뢰성 있고 안정적으로 처리하는 동시에 탄력적이고 유연성 있게 구동하기 위하여 필요한 기본적인 추상화 계층을 제공한다. 분산 미들웨어는 URC 로봇의 분산 노드들의 다양한 이기종성을 숨기고 분산 응용의 부분들이 유연하게 상호 작용할 수 있도록 분산성을 감추는 추상화 계층을 제공한다. 마지막으로 배치 미들웨어는 로봇 응용의 재구성성을 지원하며 분산 컴포넌트 기반 응용 프로그램의 배치를 지원한다. 이는 응용의 다운로드와 설치 및 제거, 응용의 생성과 소멸, 시작과 정지를 포괄하는 응용 컴포넌트들의 재구성 과정을 지원하는 계층이다.

현재 RSCA 표준과 구조를 만족시키는 프로토타입이 구현되었으며, URC 로봇에 적용 중이다.

Ⅰ. 서 론

지난 세기 후반부터 시작되었던 컴퓨터, 가전, 통신 기술의 융합은 IT 산업기술에 엄청난 영향을 끼쳤다. 이에 따라 정보가전이라는 새로운 산업 분야가 탄생하였으며, 현재 우리나라의 많은 기업들이 이 분야에서 기술을 선도하고 있다. 한편 최근 들어서는 이런 IT 기술의 융합 정도가 더욱 심화되고 있다. 이런 현상의 대표적인 예로, 전자 정보 소프트웨어 기술이 자동제어나 메카트로닉스 기술과 급격하게 결합하고 있는 자동차 산업 분야를 들 수 있다. 이런 추세에 맞추어 최근 각광을 받고 있는 분야 중에 하나가 바로 로봇 분야이다.

로봇은 인간의 노동을 대신할 수 있다는 유용성 때문에 전통적으로 인간의 동경의 대상이 되어 왔다. 그러나 핵심 요소 기술이 성숙되지 않아서 그 실제적 확산이 매우 더뎠다. 최근 고성능 CPU의 가격이 하락하고 통신 기술이 빠르게 발전하게 되면서 지능형 서비스 로봇을 개발할 수 있다는 기술적 낙관론이 급격히 힘을 얻고 있다.

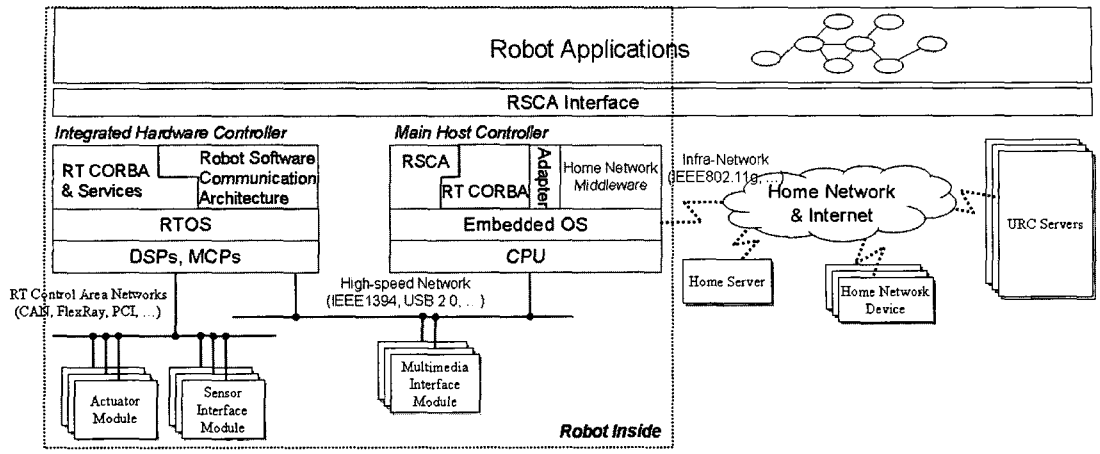
이런 추세의 선두에 있는 것이 작년부터 기획되어 개발되고 있는 URC 로봇이다. URC란 ubiquitous robotics companion의 약자로서 "언제 어디서나 우리 주변에서 친구처럼 서비스를 지원하는 로봇"을 의미한다. 이 URC 로봇의 목표는 로봇 기술의 혁신을 통해 로봇 시스템 보급과 실용화를 앞당기는 것이다. 이를 위해 로봇의 기능을 지능화시키면서도 개인이 소유할 수 있을 정도로 가격을 낮추어야 한다. 그러나 현재의 기술 수준으로 볼 때, 이 두 가지 목표는 다분히 상호 모순적이다. URC 로봇에서는 광대역 통신망(BcN) 기술과 고성능 서버 기술을 통해 이런 상호모순성을 극복하려 하고 있다. 즉, IT 분야의 씬-클라이언트(thin client) 단말기처럼, 원격지의 고성능 서버를 통해 로봇의 주요 컴퓨팅 요구를 만족시켜 저렴한 하드웨어로 로봇을 구현하려고 하는 것이다. 예를 들면, 고성능 MPU나 DSP를 요구하는 로봇의 비젼 시스템이나 운항 시스템을 원격지 서버로 구현하여 로봇 자체를 경량화시키는 것이다.

이와 같은 URC 로봇의 원대한 목표가 달성되려면 로봇을 구성하는 소프트웨어가 고도화되어야 한다. 이에 따라 로봇 소프트웨어 시스템은 프로그램들이 분산된 환경에서 수행시킬 수 있는 (1) 분산성, 응용 프로그램을 동적으로 적재하거나 재구성할 수 있는 (2) 동적 재배치 기능, 엄밀한 시간제약에 맞도록 (3) 실시간성, 로봇의 비젼과 음성 처리를 위한 (4) QoS 기능, 불완전한 상황에서도 신뢰성 있게 동작하는 (5) 결함 허용성을 지원하여야 한다. 뿐만 아니라 로봇의 특성상 고도의 자원제약성과 하드웨어의 다양성을 극복할 수 있어야 한다.

이와 같이 다양하고 어려운 기술적 요건을 만족시키는 로봇을 위한 소프트웨어 구조를 개발하는 것은 매우 어려운 일이다. 이에 따라 세계적으로도 많은 연구가 진행되어 왔으나 아직 국제적 동의를 이루는 소프트웨어 표준이 등장하지 못하고 있는 실정이다. 본고에서는 URC 로봇 연구의 일환으로 개발되고 로봇 소프트웨어 통신 구조(RSCA: robot software communications architecture)에 대해서 기술하고, 이 RSCA가 URC 로봇의 어려운 요구사항들을 어떻게 만족시킬 수 있는지 설명하도록 한다.

Ⅱ. URC 로봇과 시스템 소프트웨어 개관

URC 로봇의 가장 핵심적인 특성은 (1) URC 서비스 사업자들이 제공하는 고용량 서버를 로봇의 연산 수행을 위해서 활용하고, (2) 홈네트워크에 연결된 다양한 정보가전기기 및 센서 네트워



〈그림 1〉 URC 로봇의 하드웨어와 소프트웨어 구성

크 등을 로봇이 활용한다는 점이다. 즉 URC 로봇은 오직 로봇만으로 이루어진 시스템이 아니라, URC 서버와 다양한 홈네트워크 기기들이 포함된 복잡한 분산 시스템이다.

로봇의 응용 소프트웨어는 이처럼 여러 하드웨어 노드들로 구성된 분산 시스템에 맞게 설계되고 구현되어야만 한다. 따라서 재구성성과 유연성이 필요하며, 이를 충족시켜줄 수 있는 컴포넌트 지향 소프트웨어 모델에 따라서 응용 소프트웨어가 구성되어야 한다. 한편으로는 이와 같은 분산 컴포넌트 소프트웨어 모델을 지원하는 프레임워크로서의 시스템 소프트웨어 구조가 필요한데 바로 RSCA가 URC 로봇을 위한 표준 시스템 소프트웨어 구조이다.

이 장에서는 먼저 URC 로봇의 하드웨어 구성에 대해서 자세히 살펴본다. 그런 다음 URC 로봇의 시스템 소프트웨어가 이런 구조를 잘 지원하기 위해 가져야 할 특징들에 대해서 살펴본다.

1. URC 로봇의 내부 하드웨어 구성

그림 1은 URC 로봇의 하드웨어와 소프트웨어의 구성을 나타낸다. 우선 하드웨어 구성을 살펴보면 다음과 같다.

URC 로봇의 내부에는 하나의 MHC(Main Host Controller)와 여러 개의 IHC(Integrated Hardware Controller)가 있다. MHC는 로봇-사용자 간 인터페이스를 담당하고 홈네트워크를 통하여 로봇을 외부 인터넷에 연결하는 게이트웨이로서의 역할을 한다. IHC는 DSP, FPGA, ASIC 등을 인터페이스하며, 이들은 센서의 입력을 받고, 모터 등의 액추에이터를 제어하는 역할을 한다.

IHC와 MHC는 비디오 데이터와 같은 큰 양의 데이터를 주고받기 때문에 Giga-bit Ethernet, USB 2.0, 또는 IEEE1394 등과 같은 초고속 통신망으로 연결된다. MHC와 홈네트워크는 IEEE 802.11b/g로 연결된다. IHC와 DSP 등의 센서와 액추에이터 인터페이스들은 CAN, FlexRay 등의 실시간 제어 통신망으로 연결된다.

2. URC 로봇의 시스템 소프트웨어 개관

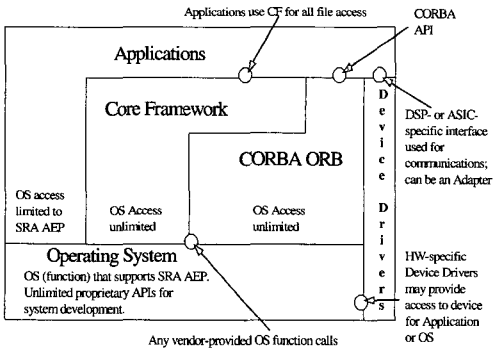
URC 로봇의 시스템 소프트웨어는 다음과 같은 URC 로봇의 응용 특성들을 지원할 수 있는 구조이어야만 한다.

- 이기종 분산처리(heterogeneous distributed computing)

- 동적 시스템 재구성(dynamic system reconfiguration)
- 서비스 품질과 실시간성 보장(QoS and real-time guarantee)
- 이종 자원 관리(heterogeneous resource management)와 자원 제약(resource constraints)

그림 2는 로봇의 응용 소프트웨어와 운영 환경(operating environment)과의 상관관계를 나타낸 것이다. 로봇의 시스템 소프트웨어인 운영 환경은 다시 코어 프레임워크(Core Framework)로 불리는 로봇 소프트웨어 배치 미들웨어와, 분산 미들웨어인 RT-CORBA[1], 그리고 실시간 운영체제(RTOS)로 구성된다.

이러한 운영 환경을 가정하고 있는 로봇의 응용 소프트웨어는 결국 이 운영 환경이 제공하는 서비스들만을 이용하여 프로그래밍된다. 즉 응용 소프트웨어는 코어 프레임워크가 지원하는 파일 시스템 서비스를 이용하여 파일에 접근하고, 다음 장에서 자세히 설명할 RT-CORBA와 RTOS가 제공하는 표준 인터페이스와 서비스만을 사용한다. 즉, 표준화된 운영 환경은 로봇 소프트웨어에 공통된 소프트웨어 플랫폼을 제공함으로써 로봇 응용 소프트웨어의 재사용성을 극대화시키는 것이다.



〔그림 2〕 RSCA 운영 환경의 개관

Ⅲ. URC 로봇 RTOS 설계

URC 로봇의 핵심적인 시스템 소프트웨어들 중에 가장 밑바탕이 되는 것은 실시간 운영체제(RTOS, Real-Time Operating System)이다. 본 절에서는 일반적인 RTOS의 개념과 역할, URC 로봇을 위한 RTOS의 설계, 그리고 RSCA에서 지정한 표준안으로써 IEEE POSIX.13[4]에 대해 알아보도록 한다.

1. RTOS의 개념과 역할

RTOS는 실시간 시스템을 위한 OS(Operating Systems, 운영체제)로 실시간 응용들이 제한된 시간 내에 올바른 결과를 생성할 수 있도록 지원하는 역할을 한다. RTOS가 시스템에서 하는 주된 역할은 실시간 태스크의 지원과 하드웨어 특성의 추상화, 효율적인 자원 관리 등이다. 아울러 RTOS는 파일 시스템, 메모리 할당, 네트워크 프로토콜 제공 등의 다양한 역할을 수행한다.

2. URC 로봇을 위한 RTOS 설계

RTOS를 설계할 때 가장 먼저 해야 할 일은 태스크 모델을 정립하고 이에 적합한 스케줄링 기법을 지원하는 것이다. URC 로봇 소프트웨어는 상호 작용하는 여러 개의 분산된 태스크들의 집합으로, 태스크들은 주기적으로 센서 값을 읽거나 비주기적으로 발생하는 중요 이벤트들을 실시간으로 처리하는 작업을 담당한다. 따라서 URC 로봇 시스템에는 주기 및 비주기 태스크가 혼재하며 각 태스크들은 경성 종료 시한을 갖는다.

이러한 URC 로봇의 태스크 모델을 위해서는 기존의 RTOS들과는 다른 스케줄링 알고리즘이

필요하다. 일반적으로 실시간 시스템에서는 고정 우선순위를 기반으로 동작하는 스케줄링 알고리즘이나 동적 우선순위 기반의 알고리즘들이 주로 사용된다. 대표적인 스케줄링 알고리즘으로 RM(Rate-Monotonic)[4]과 EDF(Earliest Deadline First)[5]가 있다. 이들 기법들은 주로 주기 태스크만을 고려하며, 비주기 태스크들을 고려하지 않거나, 단지 연성 실시간 스케줄링만 허용하는 등의 문제가 있다. 따라서 URC 로봇의 경성 실시간 주기와 비주기 태스크의 스케줄링 요구 조건을 모두 만족시키기 위해서는 보다 개선된 스케줄링 기법을 사용해야 한다. 논문 [7]의 저자는 기존의 EDF 알고리즘을 발전시켜 주기 뿐만 아니라 경성 실시간성 보장을 요구하는 비주기 태스크들을 포함한 통합된 스케줄링 기법을 제안하고 있다.

아울러 RTOS에는 자원 공유를 위한 태스크들 간의 프로토콜이 필요하다. 그런데 이를 허용하게 되면 우선순위 역전 현상이 발생할 수 있으므로, 이 문제를 해결할 수 있는 알고리즘이 필요하다. 논문 [8]의 저자들은 이에 대한 해결 방안으로 PCP(Priority Ceiling Protocol)를 제안하였다. 하지만 PCP는 실제 구현 시 런 타임 오버헤드가 지나치게 크다는 문제가 있어, 이를 보완한 IIP(Immediate Inheritance Protocol)가 고안되었다. IIP는 적은 오버헤드로 자원 공유 문제를 효율적으로 해결하였으며, 문맥 교환 회수를 절감하는 효과도 있다.

URC 로봇 시스템의 다양한 하드웨어를 제어하기 위해 RTOS는 하드웨어 추상화 계층을 제공하여야 한다. 또한 이에 덧붙여 하드웨어 자원을 효율적으로 관리하는 기능이 필요하다. URC 로봇을 구성하는 하드웨어는 안테나, RF 모뎀, DSP, 각종 센서, 모터 제어용 ECU 등 기능에 따라 다양한 프로세서들로 이루어진다. 만약 RTOS 없이 응용 프로그램들이 이러한 장치들을 직접 조작하려 한다면, 이는 불가능하지는 않다 하더라도 소프트웨어의 유연성, 탄력성, 유지 보수, 그리고 안정성의 모든 측면에서 매우 불합리한 일이 될 것이다. 또한 URC 로봇과 같이 하드웨어 자원의 제약이 강한 내장형 시스템에서는 자원들을 효율적으로 관리하는 기능이 필수적이다.

끝으로 URC 로봇을 위한 RTOS는 각종 네트워크 프로토콜을 지원해야 한다. 로봇이 기본적으로 외부 인터넷 서버와 연결되어 각종 서비스를 제공하기 위해서는, HTTP나 TCP/IP 같은 통신 프로토콜들을 지원해야 한다. 또한 스트리밍 데이터(비전, 음성 데이터 등), 혹은 센서 데이터들을 적은 오버헤드로 처리하기 위해 UDP 프로토콜도 지원해야 한다. URC 내부의 MHC(Main Host Controller)와 IHC(Integrated Hardware Controller), DSP들은 IEEE1394나 CAN, TTP, FlexRay와 같은 제어 네트워크로 연결되므로, RTOS는 이들 프로토콜들도 지원할 수 있어야 한다. 이러한 노드들 간의 통신은 실시간성이 매우 중요하므로 RTOS는 이를 만족시킬 수 있어야 한다.

3. URC 로봇을 위한 RTOS 표준

URC 로봇 응용 프로그램의 이식성과 상호 운용성을 보장하기 위해서는 표준화된 RTOS 인터페이스가 필요하다. RTOS의 공통 인터페이스로 IEEE에서 정의한 POSIX 표준이 가장 주목할 만하다. POSIX는 'Portable Operating System Interface (into a UNIX like kernel)'의 약어이다. 이는 IEEE에서 제정하여 산업계 표준으로 받아들여지고 있는 UNIX 계열의 운영체제 API의 표준이다. 실시간 내장형 시스템의 운영체제는 POSIX의 API들을 전부 지원할 필요가 없기 때문에 POSIX의 부분집합만을 지원하게 된다. 따라서 이러한 POSIX API의 부분집합에 대하여

표준을 마련할 필요가 있게 된다. 이러한 필요성에 의하여 생겨난 표준이 POSIX.13 실시간 시스템 프로파일[4]이다. POSIX.13은 실시간 시스템이 시스템의 특성에 따라 지원할 필요가 있는 POSIX API의 일부분들만 그룹지어 정의한 표준이다. 이는 PSE51부터 54까지 총 4개의 프로파일로 구성되는데, URC 로봇 시스템에서는 뒤에서 설명할 분산 미들웨어들을 충분히 지원하기 위해서 멀티쓰레드와 간단한 파일 시스템 인터페이스를 제공하는 PSE 52 이상의 RTOS를 필요로 한다.

Ⅳ. URC 로봇 미들웨어 설계

URC 로봇의 시스템 소프트웨어 중 미들웨어는 그림 2에서 보는 것처럼 분산 미들웨어와 배치 미들웨어로 구성된다.

1. 분산 미들웨어 (CORBA)

본 절에서는 URC 로봇의 시스템 소프트웨어로서 분산 미들웨어의 개념과 역할, 그리고 RSCA에서 지정한 표준안으로서 CORBA에 대해 알아본다.

1) 분산 미들웨어의 개념과 역할

(1) 분산 시스템에서 이기종에 대한 의존성 제거

URC 로봇과 같은 분산 시스템에서 수행될 프로그램을 작성하는 것은 매우 어려운 일이다. 그 중 가장 큰 문제점은 분산 시스템을 구성하는 노드들의 이기종성이다. 이기종성은 하드웨어, 운영체제, 네트워크 프로토콜, 프로그래밍 언어 등의 다양성을 의미한다.

이런 문제점을 효과적으로 해결하는 것이 바로 분산 미들웨어이다. 즉, 미들웨어를 사용하면 한 번만 프로그램을 작성해도 그 프로그램이 어떤 종류의 프로세싱 노드에서 수행되던지 같은 방식으로 동작하도록 만들 수 있다. 이에 따라 미들웨어를 사용하면 분산 시스템에서 동작할 프로그램을 만들 때 다음의 항목들의 이기종성에 대해 고려하지 않아도 된다.

- 하드웨어: 예, big-endian/little-endian의 차이
- 운영체제: 예, 윈도우즈의 Win32API와 리눅스의 POSIX API의 차이
- 네트워크 버스 프로토콜: 예, IEEE 802.3과 CAN bus의 차이
- 네트워크 소프트웨어 프로토콜: 예, TCP/IP와 UDP/IP의 차이
- 프로그래밍 언어: 예, C++와 Java의 차이

(2) 분산 환경에서 필요한 다양한 서비스 제공

프로그램이 분산 환경에서 동작하기 위해서는 프로그램 그 자체만으로도 고려해 주어야 할 사항이 많다. 그 중 대표적인 것이, 통신 하고자 하는 개체가 어떤 노드에 있는지 찾을 수 있어야 한다는 것이다. 이는 분산 시스템에서는 어떤 개체(프로그램 혹은 객체)가 특정 노드에서 수행되는 것이 정적으로 결정되는 것이 아니라, 시스템의 상황에 따라 동적으로 변화하기 때문이다. 또한, 상대 노드가 작동 불능일 경우에 대비하여야 한다. 분산 시스템에서는 네트워크 문제나, 노드의 하드웨어 문제 등과 같은 다양한 이유들로 인하여, 상대 노드와의 통신이 불가능할 경우가 종종 발생한다. 이런 경우 분산 미들웨어는 우아하게 대처할 수 있어야 한다.

이러한 문제들은 네트워크의 이기종성 때문에 생기는 문제이기 보다는 분산 프로그래밍 자체에서 유발된 문제들이다. 미들웨어는 이러한 문제들을 해결해 주기 위하여 다음과 같은 다양한 서비스를 제공해 준다.

- Naming Service: 분산 노드에 퍼져 있는 수

많은 객체들에게 이름을 부여하고, 그 이름만으로 객체에 대한 reference를 얻을 수 있게 해 주는 서비스

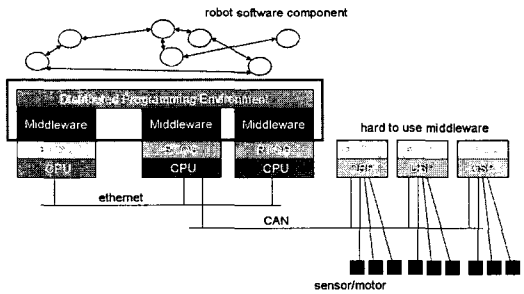
- Event Service: 원격 노드에서 특정 이벤트가 발생하면 이를 알려주는 서비스
- Time Service: 분산 시스템에서의 각 노드의 시간을 동기화 하는 서비스
- 이 외에도 Security Service, Trading Service, Relationship Service, Property Service 등 다양한 서비스가 존재한다.

(3) URC 로봇에서 분산 미들웨어의 위치

URC 로봇에서 분산 미들웨어는 MHC와 IHC의 두 종류의 노드에서만 수행된다. URC 로봇의 또 다른 종류의 노드인 DSP에서는 분산 미들웨어를 수행시킬 수 없다. DSP는 미들웨어와 같은 복잡한 소프트웨어를 원활히 수행시키기에는 프로세싱 능력이 부족하기 때문이다. 또한 미들웨어는 전통적으로 범용 프로세서에서 개발되었기 때문에 DSP와 같은 특수 목적의 프로세서에서 사용되도록 수정(포팅)하기가 매우 어렵기 때문이다. 이런 이유로 DSP에는 미들웨어가 사용되지 않는다. 그림 3에 이런 상황이 묘사되어 있다.

2) URC 분산 미들웨어

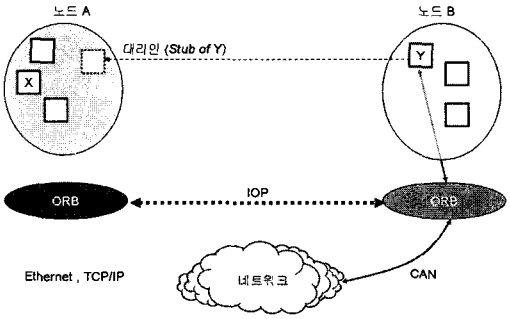
URC를 위한 분산 미들웨어로는 CORBA가 사용된다. CORBA(Common Object Request Broker Architecture)[9]는 OMG(Object Management Group)에 의해서 정의된 분산 미들웨어의 표준이다. 현재 버전 3까지 정의되어 있으며 이는 분산 실시간 환경을 지원하기 위한 RT-CORBA 기능이 포함되어 있다. 여기에서는 CORBA의 핵심 개념인 ORB, IOP, stub 객체에 대해 설명한다.



〔그림 3〕 URC 로봇에서 미들웨어의 위치

(1) ORB와 IOP

ORB는 프로토콜 변환기이다. ORB는 각 노드마다 하나씩 존재하는데, 다른 노드에 있는 객체와 통신을 하려면 ORB에게 메시지를 보내야 한다. 그런데 각 노드마다 ORB를 사용하는 방식과 ORB에게 보내지는 메시지의 모양은 다르다. 각 노드마다 endian, 프로그래밍 언어, 통신 프로토콜, 운영체제 등이 다르기 때문이다. 그러나 ORB들은 자신이 받은 메시지를 IOP라는 표준의 방식으로 변환하여 네트워크로 보낸다. 메시지를 받은 ORB는 다시 그것을 자신의 노드에서 사용하는 방식에 맞게 변환하고, 적절한 객체에게 변환된 메시지를 전달한다. 그림 4은 ORB와 IOP의 역할을 나타낸 것이다.



〔그림 4〕 CORBA의 개념

(2) Stub 객체

CORBA에는 Stub 객체라는 개념이 있다. 그림 4에서 볼 수 있듯이 객체 X가 객체 Y와 통신을 하기 위해서는 객체 X가 있는 노드에 객체

Y의 Stub 객체가 필요하다. Stub 객체는 한 객체의 대리인 역할을 한다. 이에 따라 통신을 위해 객체 X는 Y의 Stub이 Y라고 생각하고 메시지를 보낸다. 그러면 Y의 Stub 객체는 ORB를 사용하여 그것을 진짜 Y에게 전달한다.

2. 분산 소프트웨어 배치 미들웨어 (RSCA Core Framework)

본 절에서는 URC의 시스템 소프트웨어로서 배치 미들웨어인 RSCA 코어 프레임워크의 개념, 역할, 구조 등에 대해서 살펴본다.

1) 배치 미들웨어의 개념과 역할

URC 로봇 응용 소프트웨어는 그림 ?에서 볼 수 있듯이 서로 연결되어 협동하는 응용 소프트웨어 컴포넌트들로 구성된다. 배치 미들웨어는 여러 개의 응용 소프트웨어 컴포넌트들로 구성된 URC 로봇 소프트웨어를 URC 로봇의 각 프로세싱 노드에 배치하는 일을 하는 소프트웨어이다.

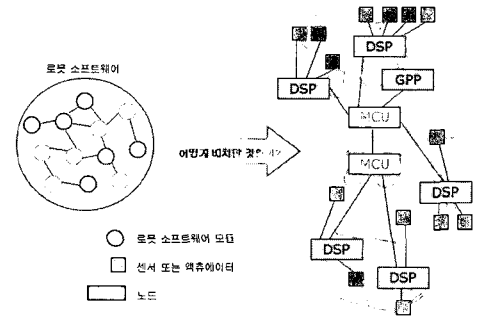
구체적으로 배치란, 어떤 로봇 소프트웨어 모듈을 어떤 프로세싱 노드에서 수행시킬 것인지를 결정하고, 각 로봇 소프트웨어 모듈들이 서로 통신할 수 있도록 연결을 해 주고, 전체 URC 로봇 소프트웨어를 시작하고 중지하는 등의 제어를 위한 일련의 작업들을 의미한다.

본고에서 설명하고 있는 배치 미들웨어는 다음과 같은 배치 관련 기능들을 제공하고 있다.

- 로봇 소프트웨어의 설치와 제거
- 로봇 소프트웨어의 생성과 소멸
- 로봇 소프트웨어의 시작과 중지
- 로봇 소프트웨어와 소프트웨어 모듈의 설정 변경
- 공통 서비스(파일 시스템, 로깅 등)

2) URC 배치 미들웨어

배치 미들웨어는 서로 협력하면서 작동하는



〔그림 5〕 배치 미들웨어의 역할

여러 개의 객체로 이루어져 있다. 로봇 소프트웨어를 설치하고 제거하는 일은 DomainManager 객체가, 로봇 소프트웨어를 생성하는 일은 ApplicationFactory 객체가, 로봇 소프트웨어의 시작, 정지, 소멸은 Application 객체가 구현하고 있다. 그리고 파일 시스템 인터페이스, 로그 등의 기본 서비스들도 각각 FileSystem, LogService의 객체들이 구현하고 있다.

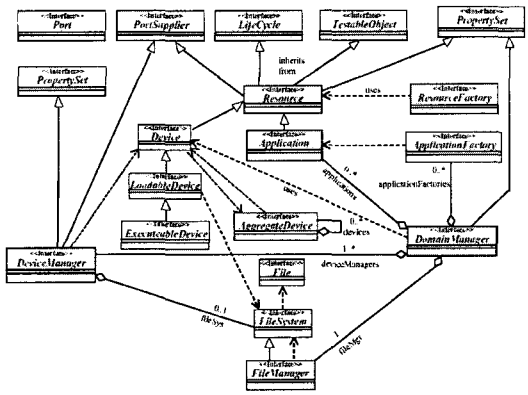
그림 6은 URC 배치 미들웨어에서 사용하는 각 객체 인터페이스의 관계를 나타낸다. 이 그림의 사각형들 중 빗금이 없는 것은 배치 미들웨어가 외부에 제공해 주는 인터페이스로서 코어 프레임워크 인터페이스라 불린다. 배치 미들웨어를 사용하는 사람(혹은 프로그램)은 이 인터페이스를 통하여 배치 미들웨어를 조작하여 로봇 소프트웨어를 배치, 생성, 제거, 시작의 동작을 시킬 수 있다.

사각형들 중 빗금이 있는 사각형들은 URC 로봇 소프트웨어가 배치 미들웨어에게 제공해 주어야 하는 인터페이스를 뜻한다. URC 로봇 소프트웨어는 이 인터페이스를 구현하게 된다. 여기에 뒤에서 설명될 도메인 프로파일이 적절히 추가로 제공되면, 로봇 소프트웨어를 이루는 많은 소프트웨어 모듈들이 자동으로 배치되고 연결되어 수행된다.

본 절에서는 코어 프레임워크 인터페이스와

도메인 프로파일에 관해서 설명한다.

(1) 코어 프레임워크 인터페이스



[그림 6] 배치 미들웨어의 구조

DomainManager(도메인 관리자)는 배치 미들웨어의 핵심 객체이다. DomainManager는 Application(응용)들, ApplicationFactory(응용 생성자)들, 하드웨어 장치들(Device 인터페이스) 및 DeviceManager(장치 관리자)들을 관리한다. 따라서 DomainManager는 소프트웨어를 설치, 제거하고 장치와 장치 관리자를 등록, 해제하기 위한 인터페이스를 제공한다.

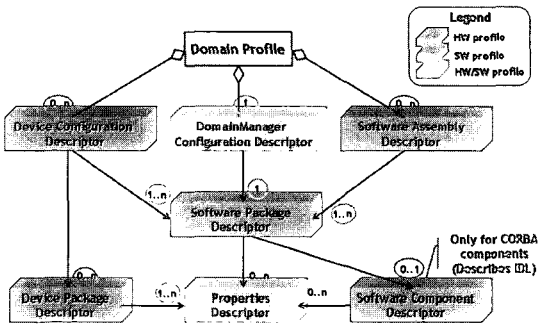
Application은 Resource(구성 컴포넌트) 타입으로서 통상 한 개 또는 그 이상의 Resource로 구성된다. Application은 사용자가 새로운 소프트웨어(응용)를 설치했을 때 배치 미들웨어 내에 자동으로 생성되는 객체로서 사용자가 설치한 응용을 시작하고 정지할 수 있는 인터페이스를 제공한다. 이는 다시 응용을 구성하는 각 컴포넌트(Resource)의 시작과 정지 동작에 대한 구현을 필요로 한다. 따라서 이런 시작과 정지 인터페이스는 LifeCycle 인터페이스에서 제공하고 Resource와 Application이 각각 상속하는 방식이다. 이것이 로봇의 분산 환경에서 가지는 의미는 여러 노드에 분산되어진 각 응용 컴포넌트를 일괄적으로 시작, 정지할 수 있다는 점이다.

Device는 응용이 사용할 수 있는 장치를 대표하는 데, 하드웨어 장치에 대한 단순한 디바이스 드라이버일 수도 있고 FPGA, DSP, GPP 등의 프로세싱 노드에 대한 논리적 장치일 수도 있다. Device 타입의 하나인 LoadableDevice는 장치에 코드 등을 다운로드할 수 있는 인터페이스를 제공하고, ExecutableDevice는 장치에 다운로드된 코드를 실행(execute)할 수 있는 인터페이스를 제공한다.

DeviceManager는 Device들을 통합적으로 관리(예. 다른 응용 컴포넌트들이 해당 장치를 사용할 수 있도록 DomainManager에 등록하는 일 등)하기 위해서 사용된다. 통상 하나의 프로세싱 노드에 하나의 DeviceManager가 있어서 해당 노드에 연결된 장치들을 통합적으로 관리한다. 그러나 DSP나 FPGA 등과 같이 자원 제약이 심하여 CORBA가 동작할 수 없는 노드에는 별도로 DeviceManager가 존재하는 대신에 다른 DeviceManager가 대리하여 관리할 수 있다. 이러한 경우 대리해서 관리해 주는 노드에는 프록시 디바이스가 동작하여 DSP나 FPGA 내부에서 동작하는 서버와의 통신을 통해 서비스를 제공한다.

FileManager는 배치 미들웨어가 제공하는 파일 시스템 서비스를 관리해주는 역할을 하며 분산 파일 시스템과 같이 도메인 내에 분산되어 있는 파일들을 동일한 주소 공간에 있는 것처럼 만들어 주는 역할을 한다. 통상 DeviceManager가 새로운 파일 시스템을 생성하여 FileManager에 등록하고 FileManager는 이를 해당 DeviceManager의 이름으로 된 디렉토리에 마운트한다.

(2) 도메인 프로파일



〔그림 7〕 도메인 프로파일 구성

도메인 프로파일은 URC 로봇 시스템의 하드웨어와 소프트웨어의 구성 정보를 기술하기 위한 XML 디스크립터 파일들로 구성된다(그림 7 참조). 이는 설치된 하드웨어와 소프트웨어 컴포넌트의 정보를 알아내는데 사용될 뿐만 아니라, 새로운 소프트웨어를 설치할 때 컴포넌트들의 조합에 대한 설치 정보를 기술하는 부가 정보로서 사용될 수 있다.

먼저 장치 구성 기술자(Device Configuration Descriptor)는 하드웨어의 구성을 기술하며, 소프트웨어 조립 기술자(Software Assembly Descriptor)는 소프트웨어의 구성 정보를 기술한다. 이들은 한 개 이상의 소프트웨어 패키지 기술자로 구성되는데, 각 소프트웨어 패키지 기술자는 소프트웨어 컴포넌트 또는 개별 하드웨어 장치를 기술한다. 이외에 DomainManager 구성 기술자(DomainManager Configuration Descriptor)는 DomainManager 컴포넌트를 기술하는 소프트웨어 패키지 기술자를 포함하고, DomainManager가 사용하는 서비스들에 대해서 기술한다. 속성 기술자(Properties Descriptor)는 개별 컴포넌트가 제공하는 재설정 가능한 속성들, 초기값, 수행 파라미터들 등에 대해서 기술한다.

(3) URC 배치 미들웨어의 특성과 기능

앞에서 URC 배치 미들웨어는 (1) 이기종 분산성, (2) 동적 시스템 재구성성, (3) 서비스 품질과 실시간성의 보장, (4) 이기종 자원의 관리 등을 지원할 수 있다고 하였다. 본 절에서는 이들이 각각 어떤 식으로 URC 배치 미들웨어에 의해서 지원되는지 살펴보도록 하겠다.

ㅇ이기종 분산성의 지원

먼저 CORBA의 이기종성 지원과 함께 URC 배치 미들웨어는 이러한 특성을 도메인 프로파일에 기술하도록 함으로써 배치 시에 이들을 활용할 수 있도록 한다. 또한 임의의 컴포넌트는 배치될 시스템의 특성에 따라 여러 개의 구현물을 가질 수 있고 이러한 정보는 소프트웨어 패키지 기술자에 기술되어 배치 시에 활용된다. 이를테면 로봇의 비젼 컴포넌트는 JVM 위에서 수행되는 구현물, Windows XP에서 수행되는 구현물, Linux가 수행되는 SPARC 머신용 구현물 등을 포함할 수 있고, 배치 미들웨어는 가용한 프로세싱 노드 중 적절한 곳에 이들 중 하나를 선택하여 배치할 수 있다.

한편 분산성 역시 CORBA에 의해서 일부 지원된다. 여기에 더하여 URC 배치 미들웨어는 분산되어 있는 노드들을 하나의 가상 시스템(도메인)처럼 보이도록 함으로써 응용에 분산성을 숨기는 기능을 제공한다. 로봇 응용들은 배치될 도메인이 몇 개의 프로세싱 노드로 구성되는지, 어떻게 연결되어야 하는지 등에 관해서 관여하지 않아도 된다.

ㅇ동적 시스템 재구성성

시스템의 재구성성은 크게 개별 컴포넌트의 재구성, 응용의 재구성, 배치 시의 재구성으로 나누어 볼 수 있다. 먼저 개별 컴포넌트의 재구성은 개별 컴포넌트가 가지는 재설정 가능한 파라미터들을 정형화된 형태로 제공하는 것, 그리고

전술한 것처럼 원하는 조건들(하드웨어 특성, 메모리 및 성능 요구량, QoS 및 실시간성 요구량 등)을 만족하는 다양한 컴포넌트 구현물을 포함할 수 있도록 하는 것 등을 통해서 URC 배치미들웨어에 의해 지원된다. 이 중 재설정 가능한 파라미터들을 정형화된 형태로 제공하는 것은 URC 배치 미들웨어 인터페이스 중 PropertySet을 통해서 제공된다.

응용의 재구성은 소프트웨어 조립 기술자가 여러 가능한 응용의 구성을 기술할 수 있도록 함으로써 지원된다. 응용 개발자는 컴포넌트의 구성과 설정값들을 달리함으로써 다양한 조건과 시스템 특성을 만족하는 소프트웨어 조립을 만들어 낼 수 있고, URC 배치 미들웨어는 이들 중 현재의 자원 상황에 맞는 조립을 선택하여 사용할 수 있다.

배치 시의 재구성은 URC 배치 미들웨어 중 ApplicationFactory에 의해서 제공된다. ApplicationFactory는 임의의 구성 컴포넌트를 배치하기 위하여 구성 컴포넌트의 시스템 의존성과 자원 요구량을 만족하는 최적의 프로세싱 유닛(Device들)을 찾는다.

• 서비스 품질(QoS)과 실시간성

배치 미들웨어는 분산 미들웨어, RTOS와 함께 서비스 품질과 실시간성을 지원한다. 분산 미들웨어와 RTOS가 개별 컴포넌트 자신의 자원 요구량과 실시간성을 만족할 수 있도록 하는 기능을 제공하는 한편, 배치 미들웨어는 개별 컴포넌트의 조합인 전체 응용의 서비스 품질과 실시간성을 만족시킬 수 있도록 한다.

특히 ApplicationFactory는 도메인 프로파일 중 소프트웨어 조립 기술자에 기술된 서비스 품질과 실시간성을 만족시키기 위하여, 소프트웨어 컴포넌트 기술자에 기술된 자원 요구량, 시스템 의존성, 그리고 현재 가용한 자원의 상황에 기반한 최적의 배치를 찾고 필요한 자원을 각 장치에 예약하는 기능을 제공한다.

• 이기종 자원의 관리

URC 배치 미들웨어는 Device 인터페이스를 통하여 이기종 자원의 관리를 지원한다. 먼저 Device 인터페이스는 Capacity를 할당하고 반환할 수 있는 인터페이스를 제공하는데, 여기서의 Capacity는 메모리 요구량, CPU 요구량, 대역폭 요구량 등 임의의 용량일 수 있다. 또한 자원의 이용 상태, 관리 상태 등을 지원하여 자원 사용을 동기화할 수 있는 기능을 지원한다. 물론 어떤 식으로 자원을 할당하고 동기화할 것인가는 사용자가 선택하여 구현하여야 하고, 이에 따라서 자원 사용의 효율성이 결정된다.

V. RSCA를 이용한 로봇 소프트웨어 작성 방법

1. 역할 분담

RSCA를 이용하여 로봇 소프트웨어를 작성하는 과정에는 하드웨어 장치 개발자, 응용 컴포넌트 개발자, 그리고 응용 개발자라는 세 부류의 개발자가 관여하는데, 이들은 각각 로봇 소프트웨어의 서로 다른 부분을 제작하게 된다.

우선 하드웨어 장치 개발자가 먼저 로봇의 각 하드웨어 장치들과 함께 그 장치들을 제어하는 장치 드라이버를 제공한다. 또한 하드웨어 장치 개발자는 RSCA의 Device 인터페이스를 제공하는 논리적인 장치 소프트웨어도 함께 제공한다. 그리고 이러한 논리적인 장치 소프트웨어는 도메인 프로파일을 통해 별도로 기술된다.

응용 컴포넌트 개발자는 응용을 이루는 각 컴포넌트를 개발한다. 이 컴포넌트의 예는 시각 처

리 컴포넌트, 주행 컴포넌트, 미로 찾기 컴포넌트 등으로 다양하다. 이러한 컴포넌트는 RSCA의 Resource 인터페이스를 제공해야 한다. 응용 컴포넌트 역시 별도로 기술된 도메인 프로파일을 필요로 한다. 이 도메인 프로파일에는 그 응용 컴포넌트가 수행될 수 있는 조건들(CPU, 메모리 요구량, 하드웨어, 운영체제 버전 등)이 함께 기술된다.

응용 개발자는 최종적으로 로봇 응용을 제작하는 사람이다. 특정 로봇 응용을 구현하기에 알맞은 응용 컴포넌트들을 구입하여 그 컴포넌트들을 적절히 연결하고 설정한다. 응용 개발자는 구입한 컴포넌트들과 함께 이 컴포넌트들의 연결과 설정 사항을 도메인 프로파일에 기술한다.

2. 하드웨어 장치 개발자

하드웨어 장치 개발자는 로봇에 부착될 다양한 하드웨어 장치를 개발하는 사람이다. 이런 하드웨어 장치의 예로는, 모터, 시각 센서, 적외선 센서, 스피커 등이 있다. 로봇을 만들고자 할 때에는 하드웨어 장치 개발자들(혹은 회사)에게서 각 종류의 하드웨어 장치를 구입하여 그것들을 조립하여 사용하게 된다.

하드웨어 장치 개발자는 단순히 물리적인 로봇 하드웨어 장치만을 제공하는데 그치지 않고 하드웨어 장치가 응용 개발자에게 논리적인 장치로 보이도록 부가적인 소프트웨어를 함께 제공한다. 이런 부가 소프트웨어를 개발하기 위해 1차적으로 그 로봇 하드웨어 장치를 제어할 소프트웨어인 장치 드라이버가 필요하다. 보통 이런 장치 드라이버는 운영체제에 종속적인 운영체제의 모듈로 제작된다.

그 다음에는 이러한 장치 드라이버가 로봇 응용 개발자에게 논리적인 장치로 보이도록 하는 wrapper가 필요하다. 이런 wrapper가 있음으로 해서 RSCA는 서로 다른 종류의 장치들을 동일한 방식으로 제어할 수 있다. RSCA에서는 이 wrapper가 제공해야 하는 인터페이스를 자세하게 정의하고 있다. 이렇게 정의된 인터페이스가 구현되기 위해서는 기본적으로 Device 인터페이스가 제공되어야 한다. 만약 대상 장치가 CPU나 DSP처럼 코드를 로드하거나 수행시킬 수 있는 장치라면 LoadableDevice나 ExecutableDevice의 인터페이스가 제공되어야 한다.

아울러 각 Device 인터페이스의 특징들이 도메인 프로파일로 기술되어야 한다. 여기에 사용되는 도메인 프로파일은 SPD(Software Package Descriptor)이다. 이 SPD에는 Device 인터페이스가 수행될 수 있는 환경(운영체제, CPU 아키텍쳐, CPU 요구량, 메모리 요구량, 프로그래밍 언어 등)이 서술되어 있으며, 부가적으로 그 Device에서 설정 가능한 옵션들(센서의 센싱 주기 등)들이 명시된다.

3. 응용 컴포넌트 개발자

응용 컴포넌트 개발자는 말 그대로 로봇 응용을 이루는 각 컴포넌트를 개발하는 사람(혹은 회사)을 뜻한다. 로봇 응용은 서로 협동하는 여러 개의 컴포넌트로 이루어져 있다.

이들 컴포넌트의 외부 구조는 RSCA에 명시되어 있다. 이들 컴포넌트는 모두 Resource 인터페이스를 제공해야만 한다. Resource 인터페이스는 RSCA가 각 응용 컴포넌트들을 동일한 방식으로 조작하기 위해 필요한 인터페이스이다.

이들 응용 컴포넌트도 도메인 프로파일을 통해서 별도로 기술이 된다. 여기에 사용되는 도메인 프로파일 역시 SPD이다. 이 SPD에는 그 응용 컴포넌트가 수행될 수 있는 환경(운영체제, CPU 아키텍쳐, CPU 요구량)이 서술되어 있으며, 부가적으로 그 컴포넌트에서 제공하는 설정 가능한 옵션이 서술된다.

4. 응용 개발자

응용 개발자란 앞서 설명한 하드웨어 장치 wrapper들과 소프트웨어 컴포넌트들을 잘 조합하여 하나의 독립된 일을 수행하는 응용을 개발하는 사람(혹은 회사)을 의미한다. 여기서 소프트웨어 컴포넌트는 기존에 사용하던 것을 재활용할 수도 있고, 다른 개인이나 회사로부터 구입할 수도 있으며, 자신이 직접 개발할 수도 있다.

각 컴포넌트 간의 연결은 도메인 프로파일의 SAD(Software Assembly Descriptor)에 기술된다. SAD에는 응용이 사용하는 컴포넌트의 종류와 인스턴스들, 이들의 초기 설정값, 그리고 가장 중요한 정보로서 각 컴포넌트 인스턴스들 간의 연결 관계가 기술된다. 컴포넌트의 종류를 기술할 때는 해당 컴포넌트의 SPD 파일 경로를 함께 적어 주도록 한다. 또한, 연결 관계를 기술할 때는 해당 인스턴스의 이름과 이를 얻을 수 있는 방법들을 기술한다. 예를 들면, CORBA naming service를 이용하였을 경우 등록할 때 사용한 이름을 함께 기술해야 한다.

VI. 결 론

본고에서는 URC의 분산 로봇 플랫폼에서 임베디드 소프트웨어의 동적 재구성을 지원하는 통합 미들웨어로서 RSCA(Robot Software Communications Architecture)의 구조에 대해서 설명하였다. 구체적으로 URC의 하드웨어와 응용 소프트웨어 구조를 소개하고, RSCA의 표준 운영 환경의 필요성과 역할에 대해서 살펴보았다. 이런 RSCA의 표준 운영 환경은 RTOS, RT-CORBA 분산 미들웨어, 배치 미들웨어(RSCA 코어 프레임워크)로 구성된다.

RTOS는 다양한 하드웨어 디바이스 위에서 로봇 응용을 신뢰성 있고 안정적으로 처리하는 동시에 탄력적이고 유연성 있게 구동하기 위하여 필요한 기본적인 추상화 계층을 제공한다.

분산 미들웨어는 URC 로봇의 분산 노드들의 다양한 이기종성을 숨기고 분산 응용의 부분들이 유연하게 상호 작용할 수 있도록 분산성을 숨기는 추상화 계층을 제공한다. 또한 RT-CORBA 분산 미들웨어는 로봇 응용의 분산 컴포넌트 지향 컴퓨팅을 위한 소프트웨어 컴포넌트 모델을 지원한다.

RSCA의 배치 미들웨어인 코어 프레임워크는 로봇 응용의 재구성성을 지원하며 분산 컴포넌트 기반의 응용 프로그램의 배치를 지원한다. 이는 응용의 다운로드와 설치 및 제거, 응용의 생성과 소멸, 시작과 정지를 비롯한 응용 컴포넌트들의 재구성 과정을 쉽게 할 수 있도록 지원한다.

마지막으로 본고에서는 RSCA를 이용하여 응용을 개발하는 방법과 과정을 설명하였다. 특히 응용의 개발 과정에 참여하는 하드웨어 개발자, 컴포넌트 개발자, 응용 개발자의 역할과 RSCA의 활용 방법을 설명하였다.

본 연구진은 본고에서 제시한 RSCA를 실제 로봇에 적용할 수 있도록 그 유용성을 검증하기 위하여 프로토타입을 구현하였다. RTOS와 RT-CORBA를 위해서는 공개 소프트웨어인 Linux와 TAO를 각각 사용하였고, RSCA 배치 미들웨어는 C++을 이용하여 구현하였다. 현재는 검증에 사용할 로봇 응용을 구현 중이며, RSCA를 사용함으로써 생길 수 있는 성능상의 문제점을 파악하기 위한 연구를 진행하고 있다.

본 연구진의 이러한 경험에 비추어 보았을 때 RSCA는 앞서 제시한 URC 로봇의 요구사항을 잘 만족시키는 핵심 소프트웨어라고 평가할 수 있다. 계속해서 이를 실제적으로 적용하기 위해서 성능 개선, 고장 감내, 보안, QoS 등에 대한 연구를 진행하고 있다.

참 고 문 헌

[1] Object Management Group. "Real-Time CORBA Specification Revision 1.1." OMG document formal/02-08-02 (August 2002).

[2] Erann Gat. On Three-Layer Architectures. Artificial Intelligence and Mobile Robots, 1997. MIT/AAAI press.

[3] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. "An Architecture for Autonomy," the International Journal of Robotics Research Special Issue on "Integrated Architectures for Robot Control and Programming (1998).

[4] ISO/IEC ISP 15287-2, IEEE Std 1003.13, Information Technology - Standardized Application Environment Profile - POSIX Realtime Application Support (AEP), February 2000.

[5] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," In Proceedings of IEEE Real-Time Systems Symposium, pp. 166-171, Dec. 1989.

[6] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," J. ACM, vol. 20, no. 1, pp. 46-61, Jan. 1973.

[7] J. Park, M. Ryu, and S. Hong, "Deterministic and Statistcal Admission Control for QoS-Aware Embedded Systems," Journal of Embedded Computing. 2004.

[8] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," IEEE Transactions on Computers, 39(3), pp. 1175-1185, Sep. 1990.

[9] Object Management Group. "The Common Object Request Broker Architecture: Core Specification Revision 3.0," December 2002.

용 성 수

1986년 2월 : 서울대학교 컴퓨터공학과 졸업

1988년 2월 : 동대학원 석사

1994년 12월 : University of Maryland at College Park, Dept. of Computer Science 박사

1988년 2월 ~ 1989년 7월 : 한국전자통신연구원 연구원

1994년 12월 ~ 1995년 3월 : Faculty Research Associate (University of Maryland at College Park)

1995년 4월 ~ 1995년 8월 : Silicon Graphics Inc. 연구원,

1995년 9월 ~ 1997년 9월 : 서울대학교 전기컴퓨터공학부 전임강사

1997년 10월 ~ 2001년 10월 : 서울대학교 전기컴퓨터공학부 조교수

2001년 10월 ~ 현재 : 동학부 부교수.

<관심분야> 실시간 시스템, 실시간 운영체제, 정보가전, 실시간 시스템 설계 방법론, 임베디드 미들웨어, 소프트웨어 공학.