전자통신동향분석 제20권 제2호 2005년 4월

로봇 소프트웨어 아키텍처의 연구동향과 현황

Research Trends and Status of Robot Software Architecture

지능형 로봇 특집

이승익 (S.I. Lee)	로봇소프트웨어아키텍처연구팀 선임연구원
장철수 (C.S. Jang)	로봇소프트웨어아키텍처연구팀 선임연구원
정승욱 (S.W. Jung)	로봇소프트웨어아키텍처연구팀 선임연구원
김중배 (J.B. Kim)	로봇소프트웨어아키텍처연구팀 책임연구원, 팀장

목 차

비구조화되고 예측 불가능한 환경에서 동작하는 지능형 로봇의 제어를 위한 프로그램의 개발은 범용 컴퓨터에서 수행되는 일반 응용프로그램과는 달리 로봇이 존재하는 세계와의 복잡한 상호작용을 전제로 하고 있다. 이러한 전제는 로봇 프로그램에게 순차성과 더불어 병행성, 예외처리, 외부세계와의 인터페이스 등을 요구하며 더불어 로봇 소프트웨어 제어구조가 특정 하드웨어나 플랫폼에 의존적이지 않고 여러 하드웨어 플랫폼에 두루 적용될 수 있는 구조가 요구된다. 로봇 소프트웨어 아키텍처는 이러한 요구에 기반하여 프로그래밍의 복잡성과 반복성을 줄이고 로봇을 보다 효율적으로 제어할 수 있는 구조를 제공하는 것을 목적으로 한다. 본 논문에서는 로봇 소프트웨어 아키텍처에 대한 최근의 연구동향에 대하여 살펴보고 신성장동력의 하나인 IT 기반 지능형 서비스 로봇을 위한 로봇 소프트웨어 아키텍처의 최근의 연구현황에 대하여 소개한다.

Ⅰ. 서론

1983년 스탠리 큐브릭의 단편소설 “Super-toys Last All Summer Long”을 원작으로 한 영화 A.I.는 영화의 도입부에서 이 영화의 주제를 제시하는 주제어를 던지고 있다. 인간의 감정을 지닌, 아니 꼭 인간만의 감정은 아니겠지만, 사랑을 할 줄 아는 로봇을 만들겠다는 과학자들에게 마치 경고라도 하듯이 “그러나 제 질문에 제대로 답변하지 않으셨군요. 사랑 받는 사람은, 그렇다면 그 로봇에 대해서 어떤 책임을 지니는지요?” 이처럼 인간의 감정을 지닌 로봇과 인간의 관계에 대해 고민하는 A.I.를 포함하여 로봇을 주제로 하는 많은 다른 영화에서도 미래에는 지능적일 뿐만 아니라 기뻐하고 슬퍼하는 등의 감정과 영화 바이센테니얼맨과 같이 늙고 죽기까지 하는 생물학적 특성까지도 표현되고 있다.

물론, 현재의 로봇 연구의 수준은 기존 산업용 로봇에서 지능형 서비스 로봇으로 통칭되는 비산업용 로봇에 대한 산업화 및 대중화를 모색하는 단계에 와 있다. 영화에서 보여지는 로봇의 수준과는 상당한 괴리가 있는 것이 사실이지만, 앞으로의 성장 잠재력은 무궁무진하다고 보여지며 앞으로 로봇이 국내의 성장을 주도하고, 세계시장에서의 로봇 관련 제품 시장을 선점하기 위하여 정보통신부에서는 ‘IT 기반 지능형 서비스 로봇’으로서 URC라는 개념을 독창적으로 창안하여 사업을 추진하고 있다.

URC는 “언제 어디서나, 나와 함께 하며, 나에게 필요한 서비스를 제공하는 로봇”으로 간단하게 정의할 수 있다. 일견 단순해 보이는 이 정의는 많은 내용을 함축하고 있지만 가장 중요한 것은 URC 로봇이 유선 혹은 무선 통신이 가능한 각종 외부 디지털 디바이스와 상호 연동할 수 있어야 한다는 것이다. 이를 통하여 환경 인식이나 음성 인식 등과 같이 로봇이 수행해야 할 기능을 외부 디바이스에 분담시킴으로써 로봇의 하드웨어 구성을 단순화시켜 제작 원가를 절감하고, 날씨나 교통 정보와 같이 로봇 단독으로는 획득할 수 없는 정보 등을 이용하여 보다 다양한 서비스를 할 수 있다는 점이다. 즉, URC 로봇을 기존 서비스 로봇보다 향상된 서비스를 지원하면서도 더 싸게 만들 수 있게 된다.

산업용 로봇이 일반인의 접근이 통제된 고정된 환경에서 미리 순서가 정해진 일을 반복 수행하는 것과는 달리 서비스 로봇은 보고 듣고 말할 수 있으며 주행 혹은 보행 기능을 가지고 일반인과 상호작용할 수 있는 최소한의 지능을 보유해야 하고, 값싸게 제작하여 널리 보급될 수 있어야 하며, 이를 위해서는 로봇 관련 하드웨어 및 소프트웨어에 대한 집중적인 연구가 요구되고 있다. 특히, 소프트웨어 분야는 음성이나 비전과 같이 기능모듈과 함께 로봇을 보다 효율적으로 제어하고 프로그래밍할 수 있는 플랫폼 역할을 수행하는 로봇 소프트웨어 아키텍처에 대한 관심과 연구가 활발히 진행되고 있다.

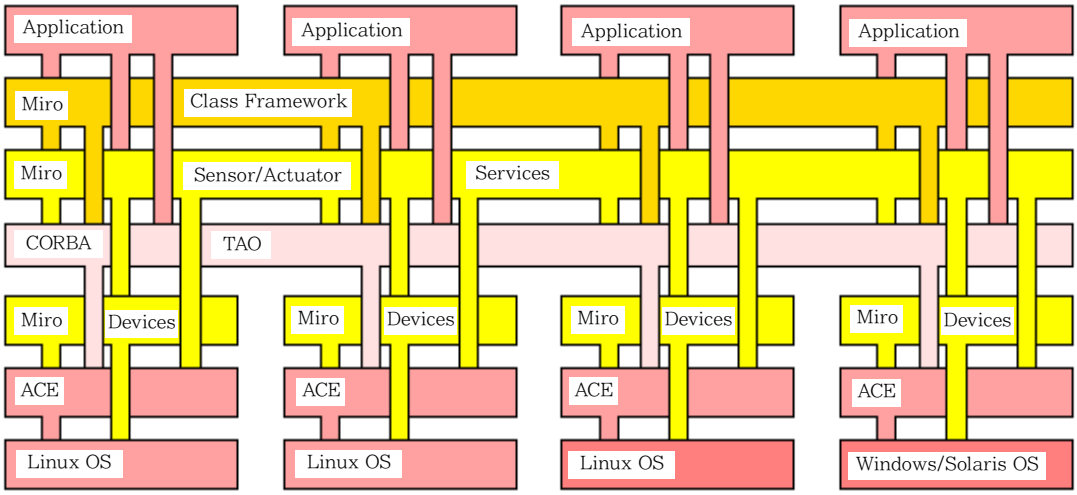
본 기고에서는 로봇 소프트웨어 아키텍처에 대한 외국의 연구개발 동향에 대해서 소개하고 URC 로봇에 대한 소프트웨어 아키텍처를 기술하고자 한다.

Ⅱ. 연구개발 동향

국내에서는 아직 로봇 소프트웨어 아키텍처에 관한 연구가 활발하지 않지만 외국의 사례를 볼 때 다양한 형태의 로봇 소프트웨어 아키텍처에 관한 연구가 진행되고 있다[1]-[5]. 본 절에서는 URC 환경에 적합한 소프트웨어 아키텍처를 수립하기 위해서 기존에 수행되었던 외국의 프로젝트 사례 중 MIRO[2], RT-Middleware[4], ERSP[3]에 대해 간략하게 소개하고 URC 소프트웨어 아키텍처 수립에 활용하고자 한다.

1. MIRO

MIRO[2]는 독일 Ulm 대학의 전산학과 로보틱스연구실에서 상이한 운영체제, 상이한 프로그래밍 언어, 상이한 개발 환경을 가진 여러 종류의 모바일 로봇을 개발하면서 느꼈던 로봇 미들웨어에 대한 필요성 때문에 시작되었다. MIRO는 이질적



(그림 1) MIRO 구조

(heterogeneous) 분산 구조를 가진 모바일 로봇을 대상으로 CORBA[6] 기술을 기반으로 한 로봇용 분산 객체지향 프레임워크를 제공하는 것을 목적으로 한다. (그림 1)은 MIRO의 전체구조를 보인다.

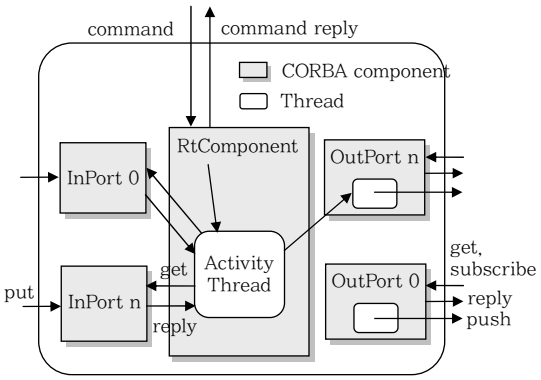
MIRO 디바이스 계층은 CAN 버스로 연결된 저 수준의 디바이스 및 제어보드와의 통신 링크에 대해 마치 일반 메소드 호출처럼 사용할 수 있게 해주는 클래스들을 구현하여 상위 계층에게 제공한다.

MIRO 서비스 계층은 CORBA IDL을 이용하여 각종 디바이스에 대해 표준화된 서비스 인터페이스를 정의하고 있으며, 이들 서비스를 CORBA 객체로 제공한다. 로봇 응용 프로그래머는 표준화된 CORBA 프로토콜을 이용하여 원하는 기능을 처리할 수 있다. 이 계층에서의 IDL을 이용한 클래스 계층화는 센서나 액추에이터에 대해서 추상화(abstraction)와 일반화(generalization)를 제공하여 로봇 소프트웨어의 설계 및 개발 생산성을 향상시킬 수 있다. 또한, CORBA 통지(notification) 서비스를 이용한 이벤트 기반 통신을 지원하여, 폴링 기법과 같은 전통적인 동기화 방식이 적당하지 않은 디바이스들을 위한 비동기 통신 방법을 제공한다.

MIRO 클래스 프레임워크는 모바일 로봇에서 통상적으로 자주 사용되는 기술을 구현한 클래스들을 제공하는 계층이다. MIRO 프로젝트에서는 현재, 우선 순위 기반 중재자(arbiter) 방식의 행위 제어 패러다임을 지원하는 behavior engine과 미디어 필터 방식의 video image processing 및 sample-based pose estimation 기능 등을 구현해 놓았다. 향후, generic mapping 기술과 path planning 기술을 클래스 프레임워크에 추가할 예정이다.

2. RT Middleware

RT middleware[4]는 일본 산업기술총합연구소(AIST) 산하 지능시스템 연구부문에서 2002년부터 2004년까지 3년간 추진한 "로봇의 개발 기반이 되는 소프트웨어 기반정비" 프로젝트의 연구 결과물이다. 본 프로젝트는 로봇을 구성하는 요소기술들을 통신 네트워크를 통해 조립함으로써 다양한 로봇을 빠르고 손쉽게 구축할 수 있도록 로봇의 기반 미들웨어 기술을 확립하고 표준화하는 것을 목표로 하고 있다. RT middleware는 로봇 시스템을 구성하는 각 소프트웨어 요소들을 CORBA와 같은 분산객체 기술을 이용하여 컴포넌트화하고 이들을 조립하여 로봇 시스템을 구성할 수 있도록 소프트웨어 기반을 제공하는 로봇용 미들웨어로 정의된다.



(그림 2) RT Component 구조

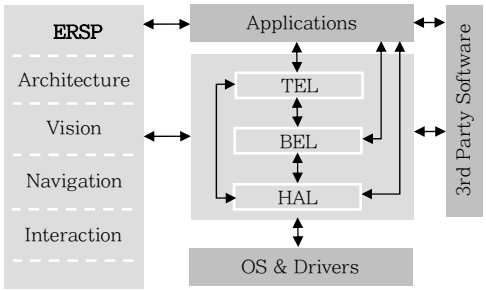
RT middleware에서 컴포넌트에 대한 규격은 RT component로 정의되며, RT component의 동작 방식과 구성은 (그림 2)와 같다.

RT component는 수동적인 서버 형태의 CORBA 객체와는 달리, (그림 2)의 InPort와 OutPort처럼 컴포넌트 내부에서 스스로 필요한 데이터를 수집하거나 이벤트를 발생시키는 기능을 수행하며, 수신된 이벤트를 처리하는 고유의 기능(activity thread)과 다른 RT component에 클라이언트로 동작할 수 있는 기능을 포함하고 있다. 향후 RT middleware와 RT component 규격은 OMG에 상정하여 국제 표준으로 확대할 예정이다.

3. ERSP

ERSP[3]는 Evolution Robotics(http://www.evolution.com)사가 2002년 개발한 로봇 소프트웨어 플랫폼이며 현재 버전 3.0이 나와 있다. ERSP는 기본적으로 모바일 로봇의 제반 소프트웨어를 개발하기 위한 플랫폼으로서, 행동기반 로보틱스(behavior-based robotics)의 개념을 지원하며, 계층구조를 통한 모듈화, 서로 다른 운영체제 및 하드웨어를 지원하기 위한 이식성, 커스터마이징, 하드웨어 독립성, 그리고 새로운 하드웨어의 추가가 용이한 확장성을 제공한다.

태스크 실행 계층(TEL), 행위 실행 계층(BEL), 그리고 하드웨어 추상화 계층(HAL)의 세 계층으로



(그림 3) ERSP의 계층적 모듈 구조

구성된다((그림 3) 참조). HAL 계층은 하드웨어 장치와 운영체제 의존성에 대한 추상화, 응용 프로그램과 아키텍처의 이식성, 장치 드라이버와 상호작용, 그리고 자원, 장치, 버스(bus)에 대한 기술과 명세를 XML 형식으로 제공한다. BEL 계층은 행위들의 활동을 중재하고, 상충을 해결하고, 자원 스케줄링을 제공한다. 각 행위들은 XML 파일로 정의되는 행위 네트워크 안에서 포트들간의 연결을 통하여 데이터를 교환한다. TEL 계층은 목적 지향적 태스크의 개발과 태스크들의 실행에 대한 중재기능을 제공한다. 태스크는 순차적/병렬적 실행 가능하며 사용자 정의 이벤트에 의해 실행이 시작된다.

Ⅲ. URC 소프트웨어 아키텍처

URC 소프트웨어 아키텍처(이하 URC-SA)는 URC 로봇의 응용 프로그램 개발을 지원하기 위한 각종 라이브러리 및 응용 프로그램의 동작을 위한 소프트웨어 플랫폼을 제공한다. 1절에서는 URC 로봇과 동작환경의 특성에 대하여 분석하고 이를 바탕으로 URC-SA의 요구사항을 도출한다. 2절에서는 도출된 요구사항을 충족시킬 수 있는 URC-SA의 구조에 대하여 설명한다.

1. 요구사항 분석

URC 프로젝트의 목적은 "언제 어디서나, 나와 함께 하며, 나에게 필요한 서비스를 제공하는 로봇"으로 요약된다. 로봇의 동작환경 측면에서 보면, 로

봇 전문가들이 로봇을 위해서 가장 최적의 구조화된 환경을 구축할 수 있는 연구 실험실과는 달리 일반 비전문가들이 상주하는 공간이므로 비구조화되고(unstructured), 동적으로 변화하며(dynamic), 부분적으로 관찰 가능한(partially observable) 환경이다. 이러한 환경은 근래의 문헌[7]에서도 나타나듯이 로봇 연구에 있어서 가장 도전할 만한 환경으로 생각된다. URC 로봇과 동작 환경의 특성을 바탕으로 URC-SA를 위한 요구사항을 정리하면 다음과 같다.

가. 분산성

기존의 연구는 하나의 보드를 가진 모바일 로봇을 주요 대상으로 하였다. 하드웨어 가격이 급격이 하락하고 SoC화가 진행됨에 따라, 앞으로는 기능별로 분화되고 모듈화된 형태의 하드웨어가 될 것으로 예상되고 있다. 또한, 독립된 로봇에 대한 연구에서 네트워크 인프라를 충분히 활용하는 네트워크화된 로봇에 대한 연구로 전환될 것으로 보인다.

URC 로봇은 내부적으로는 기능적으로 분산된 하드웨어 모듈로 구성되며, 네트워크상의 URC 서버 및 인터넷 자원을 최대한 활용하기 위한 구조를 지원한다. 분산환경에서는 하드웨어 및 소프트웨어 기능 모듈 간의 효율적인 통신, 재사용성 향상, URC 서버 및 인터넷 자원의 효율적 이용을 지원하기 위한 URC-SA에 대한 구조가 필요하다.

나. 반응성(reactiveness)

앞 절에서 살펴보았듯이 URC 로봇이 동작해야 할 환경은 비구조화되고, 동적이며 부분관찰 가능한 환경이기 때문에, 환경변화를 미리 예측하고 계획된 행동을 하기가 곤란하다(예를 들면, 로봇이 이동하는 도중에 갑자기 장애물이 앞에 떨어진다든지 하는 상황은 사전에 예측이 불가능한 환경변화이다). 이런 특성이 심화되면 될수록 미리 계획된 행동보다는 그때 그때의 상황에 따라서 즉각적으로 반응을 보일 수 있는 행동특성이 중요성을 갖게 된다.

다. 하드웨어 독립성/운영체제 독립성

URC 로봇은 그 기능에 따라 하드웨어적으로 다양한 구성을 지닐 것으로 예상되며, 개발자들이 사용하게 될 운영체제도 다양할 것으로 예측된다. 이렇듯 다양한 스펙트럼을 갖는 하드웨어 및 운영체제에서 동작해야 할 URC-SA는 하드웨어 및 운영체제의 구성이 달라지더라도 이식(porting)이 가능한 구조를 지녀야 할 것으로 보인다.

라. 강건성(robustness)

URC 로봇은 실제 가정에서 서비스를 제공하기 때문에 실행시간에 발생할 수 있는 각종 오류에 대하여 적절히 대처하는 강건성이 필요하다. 일부 센서의 고장과 같은 하드웨어적 오류, 환경이나 로봇의 내부 상태에 대한 판단 오류, 다른 로봇이나 디지털 장비 혹은 사람과 같은 다른 에이전트의 간섭에 의한 오류 등이 실행시간에 발생 가능한 오류의 예이다. URC-SA에서는 이러한 오류 상황을 정확히 진단하고 적절히 대응할 수 있는 구조를 지원해야 할 것으로 보인다.

마. 이벤트/시간 기반 태스크

항상 나와 함께 하며 내가 원하는 서비스를 제공하기 위해서 URC-SA는 이벤트 및 시간에 의해서 태스크를 시작할 수 있는 메커니즘을 제공해야 할 것이다. 기본적으로 로봇은 외부 환경과의 상호 작용 하에 동작하게 되며, 외부 세계의 변화나 내부 상태 변화에 따른 명령을 받아들여 다음 번 취해야 할 행동을 결정한다. 따라서 이러한 환경의 변화에 기반한 태스크의 수행을 지원해야 할 것으로 보인다.

또한, 로봇은 정해진 시간 또는 주기로 태스크를 수행해야 할 것으로 보인다. 예를 들어, 나는 매일 아침 7시에 출근해야 하므로, 로봇이 아침에 나를 깨워주고 출근을 준비해 주길 원한다. 로봇은 7시에 "출근 준비 태스크"를 실행할 수 있어야 하므로, 정해진 시간(7시)에 주기적으로(월~토) 주어진 임무

를 수행할 수 있는 메커니즘이 지원되어야 할 것으로 보인다.

바. 순차성/병렬성/반복성

URC 로봇의 행위는 본질적으로 순차성, 병렬성, 반복성을 지닐 수 있다. 순차성은 일련의 행위를 시간적인 순서를 갖고 실행하는 것이고, 병렬성은 두 개 이상의 행위를 동시에 실행하는 것을 의미하며, 반복성은 같은 행위를 반복적으로 수행하는 것을 의미한다.

- 순차성: 예) move_to_place(room1) → pick_up("magazine A" book)
- 병렬성: 예) play_music, go_to_goal
- 반복성: 예) deliver_coffee_for_10_people

사. 부가적 요구사항

그 밖에 부가적으로 필요한 요구사항은 태스크의 계층적 구성 능력이나, 태스크 실행 문맥의 유지관리 등이 있다.

2. URC-SA 구조

1980년대 중반 MIT의 Rodney Brooks가 포섭구조(subsumption architecture)[8]를 발표하여 행위기반 로보틱스의 새로운 장을 열었다. 행위기반 로보틱스는 종래의 SPA[7] 구조의 단점이라고 할 수 있는 느린 응답성을 극복하고자 하였다.

행위 기반 로보틱스의 주요 특징은 행동지향 센싱(action-oriented sensing), 행위의 병렬 수행, 그리고 표현(representation)의 제한적 사용이라고 볼 수 있다. 기존의 접근방식이 범용적 인지를 목적으로 한 것이라면 행동지향 센싱은 각 행동을 수행하기 위해 필요한 정도의 인지만을 수행하는 것으로써, 각 행동별로 최적화된 인지 알고리듬이 결부된다. 이 방식의 장점은 인지과정이 빠르고 상대적으로 간결하다는 장점이 있다. 표현에 있어서 이전의 접근방식이 전역적인 표현을 사용하였지만, 행위기반 로보틱스에서는 표현의 사용을 극도로 자제하거나 최소한의 지역적인 표현만을 사용하는 것을 특징으로 한다[8].

행위기반 로보틱스적 접근방식은 곤충수준의 지능을 표현할 수 있다는 점을 증명하였으나, 그 이상의 복잡한 태스크를 수행할 수 있다는 점을 증명하지는 못했다. 따라서 이후의 연구결과들은 전통적인 SPA의 계층적 구조의 장점과 행위기반 로보틱스의 장점을 적절히 조합한 하이브리드형 제어구조가 주류를 이루게 되었다.

이 연구에서는 구조의 모듈화를 꾀하고 빠른 응답성 및 상위 수준의 지능을 부여하기 위하여 하이브리드 구조를 채택한 URC-SA를 제안한다((그림 4) 참조). URC-SA가 기존의 하이브리드 구조와 다른 점은 기존의 하이브리드 구조[9],[10]가 주로 단품로봇 위주의 제어구조이고 비분산화된 환경에서의 동작을 가정하고 만들었다면, 제안하는 URC-SA의 구조는 로봇 내/외부의 분산환경과 네트워크 환경을 고려하고 이식성 및 재사용성을 높이는 데 목적이 있다.

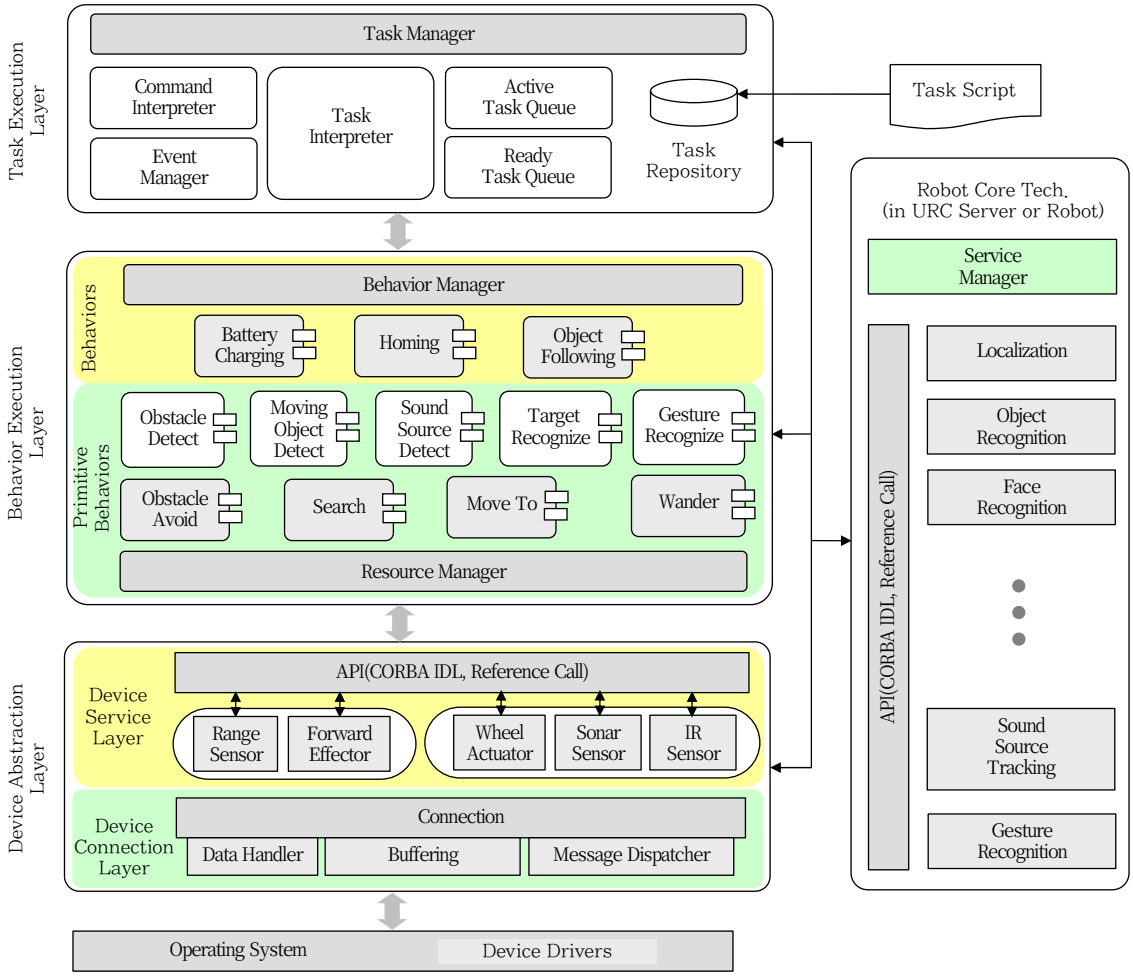
URC-SA의 특징은 다음과 같다. 첫째, 계층적 구조로 되어 있어 로봇의 반응성을 보장(behavior execution layer)함과 동시에 심의적(deliberative) 행위를 보장(task execution layer)한다. 둘째, 상이한 하드웨어 구성을 지닌 로봇의 특성에 의존적이지 않도록 하기 위해서 장치 추상화 계층(device abstraction layer)에서 하드웨어 의존적인 부분을 숨김으로써 상위 계층에서는 하드웨어에 의존적이지 않아도 된다는 장점이 있다. 셋째, 통신 미들웨어를 이용하여 계산시간이 많이 필요한 부분은 URC-서버에 두고 이를 로봇에서 이용할 수 있도록 함으로써 로봇의 실행속도를 향상시킴과 동시에 로봇 하드웨어의 저가화를 이루기 위한 기반을 제공한다는 점이다. 넷째, 각 행위나 태스크는 컴포넌트 모델을 사용하여 별도의 추가 삭제가 가능하여 점진적 개발이 가능하다. 마지막으로, 스크립트 형태의 태스크 기술 언어를 이용하기 때문에, 범용 언어를 사용할 때의 컴파일 과정이나 임베디드

시스템 프로그래밍 과정에서 필요한 크로스 컴파일 등의 복잡성이 줄어든다는 특징을 지니고 있다.

가. 태스크 실행 계층

태스크 실행 계층은 (그림 4)에서 보이듯이 상위의 태스크 계획 계층과 하위의 행동 실행 계층 사이에 존재하며, 이들 사이의 연결을 담당하는 역할을 수행한다. 태스크 계획 계층이 로봇이 수행해야 할 목적과 최상위의 계획을 제공하고 행동 계층이 로봇의 행동을 제어하는 역할을 수행하므로, 태스크 실행 계층은 로봇에게 부여된 계획을 완수하기 위하여 아래 계층인 행동 실행 계층의 행동들의 실행을 적절히 제어해 주는 역할을 수행한다.

태스크 관리자(task manager)는 로봇 외부에서 들어오는 새로운 태스크를 태스크 저장소(task repository)에 저장하거나 삭제한다. 또한, 태스크 관리자는 실행조건을 만족시키는 태스크를 실행하거나 실행을 중지시키는 역할을 수행한다. 태스크 해석기(task interpreter)는 스크립트 형태의 태스크를 해석하여 태스크 매니저가 이를 실행할 수 있는 형태로 변환한다. 명령 해석기(command interpreter)는 주로 사용자의 상호작용을 통하여 태스크 실행에 대한 명령을 처리하는 모듈이다. 사용자의 명령은 음성이나 텍스트 형태 등 멀티모달적



(그림 4) URC-SA 구조

인 성격을 띠게 된다. 이벤트 매니저(event manager)는 로봇 내/외부로부터 발생하는 환경의 변화를 감지하고 이를 태스크 매니저에게 전달함으로써 태스크 매니저가 태스크의 실행이나 중지 또는 선택을 할 때 도움을 주거나 태스크에게 필요한 정보를 제공한다.

하나의 태스크를 구성하는 주요 항목은 〈표 1〉과 같다. 대략적으로 살펴보면, 첫째로 태스크는 주어진 임무를 완수하기 위한 행동이나 서브 태스크들을 기술한다. 행동이나 서브 태스크는 순차적/병렬적/반복적인 성격을 지닐 수 있으므로, 이를 통해 개념적으로는 일련의 행동으로 구성되는 행동 네트워크를 구축하게 된다.

둘째로, 행동 선택 시 적절한 파라미터를 제공한다. 행동은 그 자체적으로 완결성(파라미터가 필요 없는)을 갖고 있는 경우도 있으나, 많은 경우 실행 시에 적절한 파라미터를 필요로 한다. 예를 들어, 트래킹이라고 하는 행동이 객체를 따라가는 행동을 수행한다고 할 때, 어떤 객체를 따라갈지를 결정해 주어야 하는데, 이러한 점을 실행시간에 제공할 수 있도록 태스크 실행 계층에서 해당 파라미터를 제공하는 역할을 수행한다.

마지막으로 불확실성(uncertainty)에 대한 적절한 대처를 수행한다. 여기서 불확실성은 로봇이 주어진 행동을 수행할 때, 그 결과를 미리 예측하기가 어렵다는 예측불가능성(unpredictability) 또는 불결정성(undeterministic)에서 기인한다.

예를 들면, grasp_object(aBook)이라는 행동이 지시된 책을 잡는 행동이라고 가정했을 때, 이 행동을 태스크 실행 계층에서 실행되도록 선택되었다고 하자. 하지만, 실제 실행이 이루어질 때, 이 행동의 원래의 목적인 책을 잡는 행동이 성공적으로 완료된다고 보장할 수는 없다.

그 이유로는, 실행 도중에 다른 로봇이나 또는 사람이 그 책을 다른 곳에 치웠거나, 행동이 실행되기 이전의 위치에서 다른 곳으로 자연적으로 옮겨지거나, 로봇은 지시 받은 책이라고 인식하였지만 센서 능력의 한계로 인해 실제로는 다른 물체이거나, 주어진 책을 집는 도중에 하드웨어적인 오류 또는 외부환경의 변화에 의하여 책을 떨어뜨릴 수도 있다. 이렇듯이, 로봇 자체 또는 외부환경의 불확실성으로 인하여 어떤 행동이 선택되어 실행되더라도 그 결과가 예측한대로 이루어진다는 것을 보장할 수가 없으므로, 태스크 실행 계층에서는 이러한 상황변화에 맞도록 적절한 대처를 수행해야 한다.

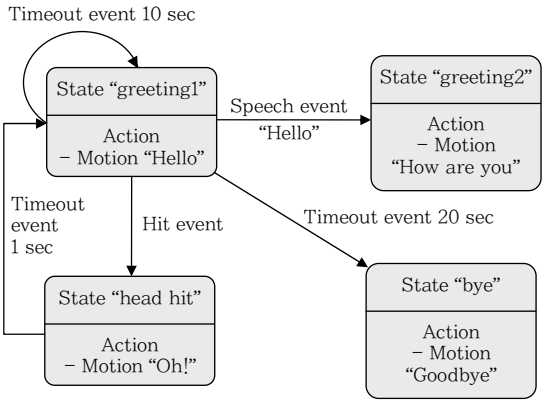
구현 언어의 관점에서 태스크를 기술하기 위한

〈표 1〉 태스크 구성 항목

기술 항목	설명
태스크 실행 조건	- 이벤트 기반: 외부 환경이나 로봇 내부의 상태를 기술한다. 이 상태가 만족되면 실행 가능하다. - 시간 기반: 시작 시간 기술 태스크가 수행이 시작되어야 할 시간을 기술하고 해당 시간이 만족되면 태스크를 수행한다. 마감시간은 태스크의 실행이 완료되어야 하는 시간을 지정한다.
태스크 종료 조건	- 이벤트 기반: 외부 환경의 변화, 로봇 내부의 상태 변화, 다른 태스크의 종료/성공 여부 등의 종료 조건 기술 - 시간 기반: 태스크가 반드시 완료되어야 할 마감 시간을 지정
태스크 성공 조건	- 태스크에 부여된 임무의 완수를 검사할 수 있는 조건 기술
실행시간 오류 대처 기술	- 로봇의 외부환경의 변화 및 내부의 센싱 능력이나 하드웨어적인 고장 등으로 인한 실행시간 오류에 어떻게 대처할지를 기술
실행 조건 보호	- 이 태스크가 수행되는 동안에 지속적으로 만족해야만 하는 조건과 복구과정 기술. 예를 들면, 책 배달하는 태스크라면 수행중에는 책을 항상 들고 있어야 한다는 조건을 만족해야 한다.
기회 활용	- 태스크 수행 도중에 보다 효율적으로 태스크의 목적을 완료할 수 있는 기회가 생길 수 있는데, 이를 활용할 수 있는 방법을 기술
계층적 태스크 기술	- 하나의 태스크가 다른 여러 개의 서브 태스크로 구성

접근방식은 크게 두 가지로 요약될 수 있다. 하나는 범용 프로그래밍 언어를 사용하는 방법이다. 이는 실행이 상대적으로 빠르고 개발자에게 익숙한 언어라는 장점이 있으나, 한번 컴파일되어 로봇에 탑재되면 다시 컴파일하기 전에는 변경이 불가능하다는 단점이 있다. 반면, 스크립트 형태의 언어를 사용하는 접근방식은 태스크가 로봇에 탑재된 후라도 수정이 가능하다는 장점이 있으나, 실행이 느리며 개발자가 새로운 언어를 배워야 한다는 단점이 있고, 스크립트 형태로 개발자가 원하는 기능을 충분히 지원할 수 있는가에 대한 의문이 남는다.

이 연구에서는 숙련된 개발자로 하여금 보다 세련된 로봇 제어를 수행할 수 있도록 함과 동시에 XML 형태의 태스크 기술을 함께 지원하고 이를 별도의 개발도구를 통하여 자동화 함으로써 비숙련자도 손쉽게 로봇 응용 프로그램을 제작할 수 있도록 한다. 예를 들어, (그림 5)와 같은 제어 흐름을 갖는



(그림 5) 로봇 응용 태스크 예

```
〈task name='EXAMPLE'〉
  〈state name='greeting1' action='Hello'
   classname='App.Example.HelloImpl'〉
    〈transition when='event==timeout value=1sec'
     nextstate= 'greeting1'/〉
    〈transition when='event==speech; value=hello'
     nextstate= 'greeting2'/〉
  〈/state〉
  ...
〈/task〉
```

(그림 6) 태스크 기술 XML 파일

로봇 응용 태스크를 가정해 보자.

각각의 상태에서 실행될 로봇행동, 상태간의 전이(transition) 조건을 포함하는 태스크 의미는 (그림 6)과 같이 XML 구성파일에 기술할 수 있다.

나. 행위 실행 계층

행위 실행 계층은 상위 태스크 계층의 요구에 의해 일련의 행위들을 실행시키고 이에 따른 결과를 태스크 계층에게 반환한다. 이를 위해 행위 실행 계층은 실행 가능한 행위 및 행위들에 대한 정보를 관리해야 하며, 태스크 계층의 요구 시 해당 행위들을 실행시켜야 한다. 또한, 행위 실행 시 여러 원인으로 인해 해당 행위들에서 예외(exception)가 발생하였을 때 이를 태스크 계층에게 전달해야 한다. 또한, 여러 행위들이 액추에이터와 같은 동일한 리소스를 사용하려고 할 때 이를 해결해야 한다(〈표 2〉 참조).

행위 실행 계층은 하나 또는 그 이상의 행위들과 이를 관리하기 위한 행위 관리자(behavior manager)로 구성된다. 행위(behavior)는 태스크의 수행을 위한 기본적인 빌딩 블록(building block)으로서 각자 별도의 실행 스레드를 지닌다.

〈표 2〉 행위 계층의 요구사항

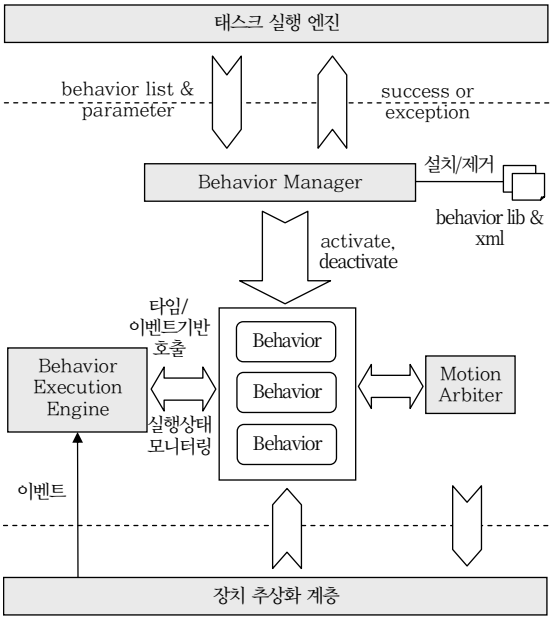
요구 사항	내용
행위 모델 제시	재사용이 용이한 행위 모델 제시 행위 개발을 위한 템플릿 제공
행위 관리	행위 등록/제거, 행위의 동적 재배치
행위 실행	행위 실행/중지 Time-driven & Event-driven 방식 모두 제공 행위실행 상태 모니터링, 예외처리 행위내부 스레드 생성 및 관리
행위간 통신	비동기 이벤트를 이용한 행위간 통신 지원, 이벤트 QoS 보장
공유 자원 관리 및 자원 충돌 조정 기능	Behavior 간 자원 충돌 조정 기능 공유 자원(센서, 액추에이터) 관리
행위 개발 도구	행위 개발 행위 네트워크 개발 행위 배포

행위는 시간적으로는 짧은 실행시간을, 제어적으로는 센서 및 액추에이터가 밀접히 연관된 일종의 피드백 제어 루프를 구축한다. 행위 관리자는 각 행동에 대한 시작/중지 및 행위 간의 충돌을 중재하는 역할을 수행한다.

행위가 수행해야 할 기능은 다음과 같다. 첫째, 행위는 센서와 액추에이터의 긴밀한 연결을 통한 로봇의 제어를 수행한다. 이는 전통적인 행위 기반 로보틱스의 접근 방법에 기반하고 있다. 둘째, 행위는 반응성을 보장해야 한다. 이를 위해서 행위는 별도의 스레드를 지니며, 액션에 필요한 센싱만을 수행하는 액션지향 인지(action-oriented perception)를 지향하고, 전역적인 표현(global representation) 대신 필요한 경우 지역적 표현(local representation)만을 이용한다. 셋째, 행위의 수행이 본래 목적을 완수하지 못할 경우 상위 계층에서 이를 적절히 처리할 수 있도록 오류 보고 기능을 수행한다. 이를 통하여 상위 계층에서는 오류를 판단하고 이를 극복할 수 있는 새로운 행동을 실행시키거나 다른 태스크가 수행될 수 있는 기회를 줄 수 있다. (그림 7)은 행위 실행 계층의 구조를 나타낸 것이다.

행위 실행 계층은 행위 관리자, 행위 실행 엔진(behavior execution engine), 모션 조정자(motion arbiter)로 구성된다. 행위 관리자는 개발된 행위들을 저장하고 제거하는 기능을 수행한다. 또한, 행위 관리자는 태스크 실행 엔진의 요구에 따라 실행해야 할 행위들을 활성화 시키거나 비활성화 시키는 역할을 수행한다. 행위 실행 엔진은 상위 계층의 태스크 실행 엔진의 요구에 따라 행위들을 실행하는 기능을 수행한다. 행위는 일정한 시간마다 실행되는 타임 기반 방식과 특정 이벤트가 발생하면 실행되는 이벤트 기반 방식이 존재하며 행위 실행 엔진에서는 두 방식을 모두 지원한다. 또한, 행위 실행 엔진은 행위의 실행 상태를 모니터링하고 실행 시 발생하는 예외를 처리하며, 행위 실행을 위한 스레드를 생성하고 관리한다.

행위간의 통신은 이벤트를 통해 이루어진다. 행



(그림 7) 행위 실행 계층 전체 구조

위는 언제든지 이벤트를 발생할 수 있으며, 행위 실행 엔진은 그 이벤트를 처리할 행위에게 해당 이벤트를 전달한다. 우선 순위가 높은 행위에게는 우선 순위가 낮은 행위보다 먼저 이벤트를 전달해야 한다. 이를 위해 행위 실행 엔진은 이벤트의 큐잉 및 이벤트 우선 순위 보장 등의 이벤트 QoS 기능을 제공한다.

여러 행위가 동시에 실행되면 같은 자원에 대해 동시에 접근하는 경우가 발생할 수 있다. 모션 조정자는 행위 간 자원 충돌 문제를 해결하는 기능을 수행한다. 행위의 수행 결과는 보통 바퀴의 속도라든지 로봇의 방향을 수정하는 명령일 것이고, 여러 행위가 동시에 속도나 방향을 수정하려고 하면 이를 우선순위 방식에 따라 조정한다.

행위의 수행 결과가 단순한 행위가 아닌 여러 행위가 포함된 복합 행동일 수 있다. 이 경우 로봇의 위치는 고정된 상태에서 복합 행동 편집기에 의해 미리 정의된 복합 행동들을 수행한다. 복합 행동은 로봇의 위치가 고정된 상태에서 다양한 형태의 행동들을 시간의 흐름에 따라 표현하는 것이다. 예를 들어, 손님이 왔을 때 “안녕하세요”라는 말과 동시에

팬/틸트를 동시에 움직여 좀 더 친숙한 행동을 손님에게 보여줄 수 있다. 행위 계층은 이러한 복합 해동을 실행할 수 있는 복합 행동 실행 엔진을 제공한다.

다. 디바이스 추상화 계층

특정한 로봇 플랫폼을 타깃으로 개발된 응용 프로그램은 다른 플랫폼에서는 전혀 동작되지 않으며, 대개의 경우 하드웨어 의존적인 부분이 프로그램 곳곳에 산재되어 이식작업 조차 매우 곤란한 것이 현실이다. 이는 결과적으로 로봇기능 및 응용 프로그램들을 중복 개발하게 하여 로봇기술의 발전을 저해하는 중요한 요인이 되고 있다. DAL의 목적은 URC-로봇에 탑재될 각종 센서 및 액추에이터들에 대한 표준화된 하드웨어 추상화를 제시하고 추상화된 디바이스를 접근할 수 있는 플랫폼을 제공하는 것이다.

로봇의 각종 디바이스들은 여러 제어보드 등에 분산 배치되기도 하며, 특히 컴퓨터 비전을 이용하는 경우 컴퓨팅 파워가 부족하여 다중 컴퓨터를 이용하기도 하는 등 태생적으로 분산성을 특징으로 하고 있다. 또한, 행위 기반 로봇에서는 센서 및 액추에이터들은 즉각적인 반응을 할 수 있도록 동시에 계속 동작하고 있어야 하는 동시성의 특징을 갖고 있다.

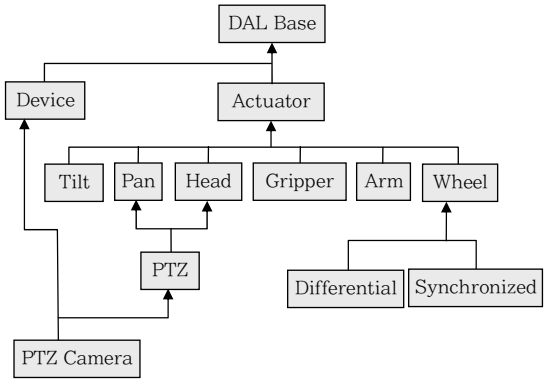
로봇의 분산성과 동시성을 지원하기 위해 DAL에서는 멀티스레딩 기술과 CORBA 기술을 사용하여 표준화된 방법으로 로봇의 각종 디바이스를 접속할 수 있는 방법을 제공한다. 프로그래머는 DAL을 이용하여 코드를 작성함으로써 표준을 준수하기가 용이하며, 작성된 코드는 재사용하기가 쉬워진다. 더구나, 다양한 하드웨어와 운영체제 위에서 동작되는 CORBA 기술을 사용함으로써 시스템을 확장하거나 다른 플랫폼에 이식하기가 용이해진다.

DAL은 (그림 4)에서와 같이 센서 및 액추에이터들을 추상화시켜 표준화된 API를 정의하고 있는 device service layer, 그리고 물리적인 하드웨어와 직접 연결되어 통신링크를 관리하고 통신링크에서 사용하는 프로토콜에 따라 메시지를 생성하거나 추출하는 device connection layer 두 계층으로 나눌 수 있다.

상위 계층인 device service layer는 (그림 8)과 같이 클래스 계층화를 통해 각종 디바이스들을 추상화하며 CORBA IDL을 이용하여 디바이스에 대한 인터페이스를 정의하고 객체로서 모델링 한다. 객체지향 패러다임을 이용한 추상화 및 인터페이스의 제공은 수많은 종류의 장치를 통합할 수 있는 핵심 요소라 할 수 있으며 객체지향 개념이 갖고 있는 다양한 장점을 통해서 로봇 소프트웨어의 설계 및 개발생산성을 향상시킬 수 있다.

정의된 인터페이스를 구현한 CORBA 객체는 실제 물리적인 디바이스의 대리인 역할을 수행하며 사용자의 명령을 대신 받아 처리한다. 대리인 역할을 수행하는 CORBA 객체의 메소드를 호출하는 방법으로는 일반적인 동기화 기반 호출 방법뿐만 아니라 이벤트 기반 비동기 통신방법을 지원하기 때문에 디바이스 종류별로 적합한 통신방법을 이용할 수 있게 되어 자원 사용의 효율성이 높아진다.

예를 들어, 범퍼센서와 같은 이벤트가 발생하자마자 로봇은 이벤트에 즉각 반응하여 멈추어야 한다. 단지 하나의 이벤트를 감지하기 위해 수천 번의 폴링(polling)을 계속 한다는 것은 비효율적이다. 이런 이유로 device service layer에서는 CORBA notification service를 바탕으로 비동기 통신 방법을 지원한다. 클라이언트는 임의의 이벤트에 대해 notification 채널에 가입하고 생산자



(그림 8) Device Service Layer의 디바이스 계층도

(producer)쪽에서는 이벤트가 발생하면 즉시 클라이언트들에게 이벤트를 넘겨줄 수 있도록 하고 있다.

CORBA 객체는 네이밍 기술을 통해 네이밍 서비스에 등록된다. 응용 프로그래머는 필요한 경우 네이밍 서비스로부터 객체를 검색하여 해당 객체를 사용하므로 프로그래머는 사용하려는 디바이스가 로컬에 있든지 원격지에 있는지 신경쓰지 않아도 된다.

DAL의 하위 계층인 device connection layer는 하드웨어 장치와 대화하기 위한 계층으로서 상위 계층에서 받은 추상화된 사용자 명령을 실제 물리적인 장치에게 원시 데이터 형태로 명령을 내려 수행토록 하는데, 일반적으로 운영체제의 장치 드라이버를 이용한다. Device connection layer는 다수의 클라이언트들이 동시에 사용 가능하도록 지원해야 하며, 또한 장치와 비동기적인 통신이 가능하도록 지원해야 한다.

이러한 요구사항들을 만족하기 위해 device connection layer는 일반적으로 데이터 핸들러, 버퍼링, 메시지 디스패처 등으로 구성되어 있다. 데이터 핸들러는 장치로부터 주기적 혹은 불규칙적으로 들어오는 원시 데이터를 읽는 기능을 수행하며, 원시 데이터를 장치와의 통신 규약에 따라 해석하여 완전한 형태의 메시지로 만드는 역할을 수행한다. 버퍼링에서는 데이터 핸들러로부터 전달받은 메시지를 저장하고 상위 계층에서 센서 등의 정보를 요청할 때 장치와 다시 연결하여 데이터를 읽는 대신 이미 저장해 놓은 정보를 제공해주어 불필요한 질의를 막으며 다수 클라이언트에 대한 동시 지원을 제공하는 기능을 수행한다. 메시지 디스패처는 버퍼링에서 넘겨 받은 메시지를 하드웨어의 이벤트나 데이터를 기다리고 있는 device service layer의 CORBA 객체에게 분배해주는 기능을 수행한다.

Ⅳ. 결론

지능형 서비스 로봇의 산업화 및 대중화를 목표로 출범한 URC 사업 중에서 소프트웨어 아키텍처 측면에서 요구사항을 도출하고 이를 충족시킬 수 있는 구조에 대하여 살펴보았다. 제시한 아키텍처를 설계할 때 추구한 주요 목표 중의 하나는 로봇이 외부의 명령을 받아서 태스크들을 순차적으로 처리하는 것이 아니라 상황에 맞게 스스로 선택적으로 수행할 수 있는 최소한의 자율성(autonomous)을 갖도록 한 점이다. 이를 위하여 최상위 계층에 태스크를 계획 또는 실행할 수 있는 기능을 두었으며 디바이스 추상화 계층의 센싱 결과에 따라 즉각적인 반응을 필요로 하는 행동에 대해서는 행위 실행 계층에서 즉시 처리할 수 있도록 이원화 하였다. 또한 URC-로봇은 하드웨어 구성 수준에 따라 통신 모듈, 센서, 그리고 액추에이터로만 구성된 초 저가형부터 보행이 가능한 휴머노이드까지 다양한 로봇을 대상으로 하고 있다. 이들 로봇의 하드웨어 구성 및 서비스 가능한 태스크 유형에 따라 제시한 아키텍처 계층의 일부 혹은 전부를 선택적으로 사용할 수 있는 유연한 구조를 갖도록 하였다. 이때 일부 계층만 사용할 경우 나머지 계층은 외부 디바이스(URC-서버)에 장착되어 상호 연동할 수 있는 구조를 지향하고 있다.

이러한 URC 소프트웨어 아키텍처를 성공적으로 개발하여 국내 업계 및 학계의 로봇 개발자에게 널리 보급하고, 이를 기반으로 다양한 형태의 URC-로봇들이 신속히 개발되어 일상 생활에서 활용됨으로써 지능형 서비스 로봇이 국가 신성장 동력의 하나로써 역할을 하는 데 일조할 수 있을 것으로 기대한다.

약어 정리

API	Application Programming Interface
BEL	Behavior Execution Layer
CORBA	Common Object Request Broker Architecture
DAL	Device Abstraction Layer
ERSP	Evolution Robotics Software Platform
HAL	Hardware Abstraction Layer
IDL	Interface Definition Language

MIRO	Middleware for RObot
OMG	Object Management Group
RT	Robot Technology
SPA	Sense Plan Act
TEL	Task Execution Layer
URC	Ubiquitous Robotic Companion

참 고 문 헌

[1] Orocos(Open Robot COntron Software) 프로젝트, http://www.orocos.org

[2] H. Utz, S. Sablatnog, S. Enderle, and G. Kraet-zschmar, "Miro-Middleware for Mobile Robot Application," *IEEE Transactions on Robotics and Automation*, June 2002.

[3] Evolution Robotics, ERSP 3.0 User's manual, http://www.evolution.com

[4] 일본 로봇공업회, "RT 오픈 아키텍처와 보급 시스템의 조사연구 성과 보고서," 2004. 3.

[5] B. Gerkey, R.T. Vaughan, and A. Howard, "The Player/Stage Project: Tools for Multi-robot and Distributed Sensor Systems," *Proc. of the 11th Int'l Conf. on Advanced Robotics*, Coimbra, Portugal, June 2003, pp.317-323.

[6] Real Time CORBA with TAO, http://ace.ece.uci.edu/TAO

[7] G.S. Sukhatme and M.J. Matarik, "Robots: Intelligence, Versatility, Adaptivity," *Communications of the ACM*, Vol.45, Issue 3, 2002, pp.30-32.

[8] R.A. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal on Robotics and Automation*, Vol.2, No.1, 1986.

[9] J. Connell, "SSS: A Hybrid Architecture Applied to Robot Navigation," *Proc. of the IEEE Conf. on Robotics and Automation* (*ICRA*), 1992, pp.2719-2724.

[10] E. Gat, "Reliable Goal-directed Reactive Control for Real-world Autonomous Mobile Robots," Ph.D. Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1991.