지식 기반 지능형 로봇의 행위 지정을 위한 구조적 반응 언어

(A Structured Reactive Robot Programming Language for Knowledge-Based Intelligent Robots)

이 재 호 † 곽 별 샘 ††
(Jaeho Lee) (Byul-Saim Kwak)

요 약 지능 로봇은 복잡하고 동적인 환경 내에서 상황의 변화에 적절히 적응하여 사용자에게 다양한 서비스를 제공하는 지능 시스템이다. 따라서 로봇은 행위를 수행하는 동안 지속적으로 상황의 변화를 감지하여 변화에 적절히 반응해야 하며 주어진 상황에 대해 최선의 행위를 결정하여 수행할 수 있어야 한다. 또한 때때로 임의의 행동을 결정하여 보다 지능적인 행위를 수행할 수 있어야 한다. 본 논문에서는 이러한 지능 로봇의 복잡한 행위를 효과적으로 정의하고 구현하기 위해 Structured Circuit Semantics(SCS)에 기초를 둔 프로그래밍 언어(Structured Programming for Reactive Intelligent Tasks, SPRIT)와 SPRIT으로 작성된 프로그램을 다양한 로봇 환경에서 실행하고 검증하기 위한 작업 실행기(Task Executor)를 제시한다.

키워드 : 로봇, 반응형 시스템, 로봇 프로그래밍 언어, 지능형 로봇

Abstract An Intelligent service robot performs various complex tasks in dynamic environment, providing useful intelligent services for human users. The robot needs to continuously monitor dynamically changing environment and reactively choose the best behavior for the changing context. The selected behaviors may include nondeterministic or parallel actions. In this paper, we present a structured reactive robot programming language, SPRIT that is based on Structured Circuit Semantics (SCS). SPRIT is fully implemented as a task executor and tested for reactive robot tasks in dynamic environment to show that it can be used to explicitly represent and effectively implement the complex reactive behaviors of intelligent robot systems.

Key words : Robot, Reactive System, Robot Programming Language, Intelligent Robot

1. 서 론

지능 로봇은 제한된 환경 내에서 정해진 작업을 반복적으로 수행하는 산업용 로봇과는 대조적으로 복잡하고 동적인 환경 내에서 상황의 변화에 적절히 적응하여 사용자에게 다양한 서비스를 제공하는 지능 시스템이다 [1-3]. 따라서 지능 로봇은 다양한 인식 기능을 이용하여 사용자 의도를 파악하고 그것에 따라 적절한 작업 목표를 설정하며, 주변 환경 및 상황의 변화를 지속적으로 감시하여 상황에 따라 적절한 행위를 결정하고 수행할 수 있어야 한다.

로봇의 환경이 매우 복잡하고 동적으로 변화함에 따라 사용자의 의도를 만족시키기 위한 로봇의 행위는 산업용 로봇처럼 유일하게 결정될 수 없으며, 따라서 로봇은 같은 목표를 달성하더라도 상황에 따라 다양한 행위를 수행할 수 있어야 한다.

또한 이러한 다양한 행위에 대해 로봇은 상황에 따라 최선의 행위를 결정하여 수행할 수 있어야 한다. 그러나

· 이 논문은 2007년도 서울시립대학교 학술연구조성비에 의하여 연구되었음
· 이 논문은 2009추계 인공지능연구회 워크샵에서 '실시간 반응형 지능 로봇 시스템의 행위 지정을 위한 프로그래밍언어'의 제목으로 발표된 논문을 확장한 것임

† 종신회원 : 서울시립대학교 전자전기컴퓨터공학부 교수
jaeho@uos.ac.kr
†† 학생회원 : 서울시립대학교 전자전기컴퓨터공학부
semix2@naver.com

논문접수 : 2010년 2월 8일
심사완료 : 2010년 3월 3일

특정 시점에서 결정된 최선의 로봇 행위는 상황이 변화함에 따라 항상 최선임을 보장할 수 없으므로 로봇은 지속적으로 상황을 감시하여 상황 변화로 인해 행위를 지속할 수 없거나 더 나은 행위가 존재하는 경우 수행을 대체할 수 있어야 한다.

한편 로봇은 크기나 비용 등의 제약으로 인해 제한적인 인식 기능을 수행하며, 이로 인한 인식 결과의 불확실성으로 인해 상황을 올바르게 인식하지 못하거나 다양한 상황을 동일한 상황으로 인식할 수 있다. 이러한 경우 로봇이 항상 동일한 상황에 대해 동일한 행위를 결정하면 그 행위는 매번 실패하게 된다. 따라서 이러한 경우 로봇은 특정 상황에 대해 임의의 행위를 결정하고 수행할 수 있어야 한다. 이러한 비결정적 로봇 행위는 특정 상황에서의 지속적인 실패를 극복할 수 있을 뿐만 아니라 단조로움을 피해 보다 지능적인 행위를 수행할 수 있게 한다.

이러한 특징은 지능 로봇의 설계 및 구현을 보다 효과적으로 하기 위한 설계 및 구현 방법과 이에 대한 실행 시스템을 요구한다. 전통적인 프로그래밍 언어와 개발 방법론은 대부분 절차 지향식이기 때문에 상황에 따른 로봇의 복잡하고 다양한 행위를 효과적으로 정의하고 구현하기 어렵기 때문이다.

이러한 문제를 해결하기 위해 본 논문에서는 Structured Circuit Semantics(SCS)에 기초를 둔 구조적 프로그래밍 언어인 Structured Programming for Reactive Intelligent Tasks(SPRIT)와 SPRIT으로 작성된 로봇 프로그램을 다양한 로봇 환경에서 실행하고 검증하기 위한 작업 실행기(Task Executor)를 제시한다. SPRIT은 최선 우선 행위와 비결정적 행위 등 상황에 기반한 복잡하고 다양한 로봇의 행위를 매우 효과적으로 정의할 수 있으며, 기존의 다양한 로봇 소프트웨어 컴포넌트와 쉽게 연계가 가능하다. 한편 작업 실행기는 자바 언어로 작성되어 윈도우, 리눅스 등 자바 언어를 실행할 수 있는 다양한 로봇 실행 환경에서 SPRIT으로 작성된 프로그램을 수행하고 검증한다.

본 논문에서는 먼저 제2장에서 로봇의 행위를 효과적으로 정의하고 구현하기 위한 관련 연구를 살펴보고, 제3장에서는 SCS에 기반한 프로그래밍 언어인 SPRIT과 작업 실행기를 제시한다. 제4장에서는 시뮬레이션 로봇 환경 기반의 실험을 통해 SPRIT과 작업 실행기를 검증하며, 마지막으로 제5장에서 결론 및 향후 연구 계획을 기술한다.

2. 관련 연구

지능 로봇은 복잡하고 동적인 환경으로부터 지속적으로 정보를 수집하고 목적과 상황에 따라 적절한 행위를 결정하는 매우 복잡한 시스템이기 때문에 이를 빠르고 효과적으로 구현하기 위한 다양한 연구들이 있다.

Microsoft Robotics Developer Studio(MSRDS)는 물리 엔진 기반의 로봇 환경 시뮬레이션 도구와 다양한 종류의 하드웨어들과 정보를 송수신할 수 있는 서비스 지향 런타임 환경, 그리고 드래그-엔-드롭 방식으로 쉽게 사용할 수 있는 비주얼 프로그래밍 언어 등으로 구성된 통합 로봇 개발 환경이다[4]. MSRDS의 비주얼 개발 환경은 매우 직관적이고 쉽게 비교적 간단한 로봇 프로그래밍을 가능하게 하지만 복잡하고 역동적으로 변화하는 환경에서 상황을 추론하고 변화에 적응하는 로봇을 구현하기에는 어려움이 따른다. 대신 C# 언어를 통해 보다 섬세한 조작이 가능하지만 C# 언어는 로봇의 지능적인 제어보다는 범용 어플리케이션 개발을 목적으로 하기 때문에 상황에 따라 적절히 반응하며 항상 최선의 행위를 선택하여 수행하는 로봇의 지능적 작업 계획을 구현하는데 어려움이 따른다.

한편 로봇의 복잡하고 다양한 행위를 보다 효과적으로 정의하고 구현하기 위해 로봇에 특화된 프로그래밍 언어를 정의한 사례로 URBI와 RADAR가 있다.

먼저 URBI는 로봇의 다양한 행위를 지정하기 위한 프로그래밍 언어와 실행 환경을 제공한다[5]. 이것은 C+, Java, Matlab 등 다양한 언어로 구현된 기존 시스템과 쉽게 연계할 수 있도록 지원하며, 서버-클라이언트 구조를 통해 다양한 플랫폼에서 독립적으로 수행이 가능하다. URBI는 이벤트 기반 동작 방식을 취하여 이벤트 발생 시 로봇이 수행해야 하는 응답을 효과적으로 지정할 수 있게 할 뿐만 아니라 로봇에서의 동시성 처리를 매우 효과적으로 관리한다.

RADAR는 로봇이 다루어야 할 다양한 상황을 이벤트로 정의하고 이와 함께 이벤트에 대한 응답을 명시적으로 정의하여 이벤트와 응답간의 연결을 독립적으로 지정할 수 있는 로봇 프로그래밍 언어이다[6]. 이는 로봇이 다루어야 할 다양한 상황을 추상화하고 실제 환경에서 로봇이 점진적으로 보다 많은 영역의 상황을 다루는 것에 대해 유연하게 대처할 수 있게 한다. 또한 이 언어는 특정 로봇 아키텍처에 종속되지 않고 개발자의 요구사항에 적합한 실행 의미를 제공한다.

그러나 URBI와 RADAR는 동적인 상황 변화에 유연하게 대처가 가능하지만 로봇의 비결정적 행위나 최선 우선 행위와 같이 로봇의 복잡한 행동 결정 방식을 서술하기 위한 방법을 제공하지 않는다.

JAM은 복잡하고 동적인 환경에서 상황의 변화에 능동적으로 대처하여 주어진 목표를 달성하는 BDI(Belief-Desire-Intention) 기반의 지능형 에이전트 시스템이다[7]. JAM은 목적을 달성하기 위한 다양한 실행 방법을

계획으로 미리 정의하는데 사전 조건, 지속 조건, 유틸리티 등을 명시적으로 서술하여 상황 변화에 능동적으로 대처한다. 목표가 지정되면 JAM은 정의된 계획들 가운데 유틸리티를 기반으로 최선의 계획을 선택하여 수행하며, 계획 내에 비결정적 행위를 지정할 수 있는 것이 특징이다. 그러나 JAM은 해석기의 동작 방식이 명시적으로 정의되어 있지 않고 시스템 내부에 감추어져 있어서 주어진 목적을 달성하기 위한 계획을 선택하고 수행하는지 명시적으로 나타나지 않으며, 이러한 해석기의 방식을 변경할 수 없다.

한편 Structured Circuit Semantics(SCS)는 다양한 반응형 계획 실행 시스템을 일관된 방법으로 정의하기 위한 프레임워크를 정의하여, 그것들을 효과적으로 비교, 분석하기 위해 제안되었다[8]. 이것은 행위를 수행하기 위해 필요한 사전 조건과 행위를 지속하기 위해 필요한 지속 조건을 명시적으로 정의하고 비결정적 행위나 최선 우선 행위 등을 서술할 수 있게 하여 지능 로봇과 같이 복잡하고 동적인 환경에서 적절히 반응하여 주어진 목적을 달성하는 반응형 시스템을 효과적으로 정의할 수 있게 한다. 그러나 SCS는 반응형 시스템의 동작을 구현 수준까지 구체적으로 정의하지 못하며 그것을 실행하기 위한 해석기가 제공되지 않기 때문에 SCS로 정의된 시스템을 검증할 방법이 없고 실질적인 운용이 불가능하다.

3. Structured Programming for Reactive Intelligent Tasks

SCS는 지능 로봇과 같은 실시간 반응형 시스템의 복잡한 행위를 매우 효과적으로 정의할 수 있게 하지만 구현을 위한 실질적인 언어와 이에 대한 실행 및 검증 시스템이 존재하지 않는다. 따라서 SCS로 정의된 시스템은 검증이 쉽지 않으며, 이렇게 정의된 시스템은 SCS에 대응하는 프로그래밍 언어가 없기 때문에 C++이나 자바와 같은 언어로 구현되어 설계와 구현에 차이가 발생하므로 또 다른 문제를 야기할 수 있다.

본 논문에서는 SCS에 기반한 프로그래밍 언어인 SPRIT(Structured Programming for Reactive Intelligent Tasks)와 이에 대한 실행 및 검증을 위한 작업 실행기(Task Executor)를 제시하여 이러한 문제를 해결하고자 한다.

3.1 설계 목표

SPRIT은 로봇의 복잡하고 다양한 행위를 보다 효과적으로 정의하고 구현할 수 있어야 하며, 이렇게 구현된 프로그램은 실제 로봇에서 수행될 수 있어야 한다. 또한 이미 구축된 다양한 로봇 소프트웨어 컴포넌트들을 효과적으로 활용할 수 있어야 한다. 설계 목표를 정리하면 다음과 같다.

- **다양한 실행 환경 지원:** 지능 로봇의 실행 환경은 그것의 요구사항과 제약 조건에 따라 매우 다양하다. SPRIT으로 정의된 프로그램이 다양한 로봇 환경에서 동작하기 위해서는 이것의 해석기가 다양한 로봇 실행 환경에서 동작할 수 있어야 한다.
- **기존 시스템(legacy system)과의 효과적 연계:** 로봇은 인식, 행위, 지능을 구현하는 다양한 로봇 기술들로 구성되며 이러한 로봇 기술들은 C/C++, 자바 언어등과 같은 다양한 개발 언어로 구현되고 있다. SPRIT은 이러한 로봇 기술들을 쉽게 통합하고 활용하여 지능 로봇을 효과적으로 구현할 수 있어야 한다.
- **개발 편의성:** 새로운 언어가 개발자에게 많은 학습을 요구하면 실질적인 활용도가 떨어진다. 따라서 SPRIT은 기존의 개발 방식을 최대한 수렴하여 개발자에게 익숙한 방법을 제공하고 학습 비용을 최소화해야 한다.

3.2 언어 정의

SPRIT은 SCS로 정의된 시스템을 실질적으로 구현하고 검증하는 목적을 갖고 있기 때문에 정의와 구현간의 차이가 크면 클수록 시스템 검증의 신뢰성을 잃게 된다. 따라서 차이를 최소화하기 위해 SPRIT은 실행 구조와 관련된 키워드를 SCS와 동일하게 정의한다. SPRIT의 문법은 표 1과 같다.

표 1 Structured Programming for Reactive Intelligent Tasks의 문법

```
action      ::= ( var_decl | java_statement )
var_decl    ::= "var" variable [ "=" java_expression ]
                ( "," variable [ "=" java_expression ] )*
step        ::= ("do" | "do*" | "repeat" | "repeat*" )
                [ "any" | "all" | "first" | "best" ]
                [ ( "when" | "unless") "(" java_expression ")" ]
                [ ( "while" | "until" ) "(" java_expression ")" ]
                [ "utility:" java_expression ]
                "{" ( step | action )+ "}"
```

시스템 구현을 위한 언어는 정보를 조작하거나 행위(action)을 지정하기 위한 방법을 제공해야 한다. SPRIT은 정보의 조작과 행위 지정을 위해 객체 지향 언어에서의 객체를 사용하도록 하였다. 이로 인해 기존 개발 언어의 장점을 최대한 활용할 수 있고 개발자로 하여금 새로운 언어에 대한 학습 비용을 최소화할 수 있다.

특히 다양한 로봇 환경에서 SPRIT으로 작성된 프로그램을 수행시키기 위해 작업 실행기를 자바 언어를 이용하여 구현하기로 결정하였으며, 따라서 SPRIT 내에서는 자바 객체를 사용하고 객체를 조작하기 위한 문법 또한 자바의 문법을 그대로 따른다. 그러나 SPRIT은 객체를 직접 정의하는 방법을 제공하지는 않는다. 다만

모듈화에 대한 지원 여부는 현재 검토 중에 있다.

데이터를 저장하거나 객체를 참조하기 위한 변수는 var 키워드를 이용하여 정의하며, 선언 시 타입을 결정하지 않는다. 대신 변수의 타입은 실행 시간에 결정된다. 이러한 동작 방식은 자바 언어와 차이를 보이는데 지능 로봇과 같이 매우 동적인 환경에서 동작하는 시스템에서는 변수의 타입을 초기에 결정하는 것보다 실행 시간에 상황에 따라 결정되는 것이 더 효과적일 것이라고 판단했기 때문이다.

따라서 객체에 정의된 함수의 인자로 변수가 주어질 때 실행 시간에 타입을 검사하여 적절한 함수가 호출된다. 예를 들어 표 2와 같이 자바 클래스가 정의되고 표 3과 같이 SPRIT 프로그램을 작성한 경우 첫 번째 TestClass.test() 호출은 변수 $value가 int 타입으로 결정되어 A가 호출되고, 두 번째 호출은 $value가 String 타입으로 결정되어 B가 호출된다.

표 2 변수의 동적 타입 검사를 테스트하기 위한 자바 클래스

```
public class TestClass {
    public static void test(int value) { // A
        System.out.println("Integer: " + value);
    }
    public static void test(String s) { // B
        System.out.println("String: " + value);
    }
}
```

표 3 변수의 동적 타입 검사를 테스트하기 위한 SPRIT 프로그램

```
do {
    var $value;
    $value = 10;
    TestClass.test($value); // Integer: 10
    $value = "Hello";
    TestClass.test($value); // String: Hello
}
```

또한 SPRIT은 대부분의 개발 언어와 마찬가지로 배열을 지원한다. 그러나 대부분의 개발 언어가 배열을 참조하는 변수를 선언할 때 대괄호를 사용하는 것과는 달리 SPRIT은 배열 조차도 하나의 타입으로 간주하므로 앞서 살펴본 바와 같이 var 키워드로 정의된 변수로 배열을 참조한다. 이 때 배열은 중괄호를 사용하여 정의하는데 배열의 원소는 특정 타입에 종속될 필요가 없으며, 따라서 배열은 다양한 타입의 원소를 동시에 가질 수 있다. 배열의 원소를 참조하기 위해서는 변수 바로 뒤에 대괄호를 쓰고 그 사이에 인덱스를 지정한다.

실행 구조는 SCS의 스텝(step)으로 정의된다. 스텝은 한 번만 수행하는 do와 반복적으로 수행하는 repeat 이렇게 두 종류가 있으며 각각은 any, all, first, best로 그것의 동작 방식을 지정할 수 있다. 동작 방식이 별도로 지정되지 않은 스텝은 그것의 내부에 정의된 행위를 순차적으로 수행한다. 스텝은 그것의 동작 방식에 따라 성공과 실패가 결정되는데 do*, repeat*과 같이 스텝 뒤에 '*'를 수식하면 스텝이 지속 조건을 만족하는 동안 성공을 반환할 때까지 계속 반복한다. 따라서 '*'로 수식된 스텝은 항상 성공을 반환하게 된다.

동작 방식이 any로 지정된 스텝은 그것의 내부에 정의된 일련의 행위들 중에서 사전 조건과 지속 조건을 만족하는 행위들 가운데 임의의 행위를 선택하여 수행한다. 만약 선택한 행위가 실패하면 시도하지 않은 다른 행위들 중에서 다시 조건을 만족하는 임의의 행위를 선택하여 수행한다. 내부에 정의된 행위들 가운데 하나라도 성공하면 스텝은 성공하지만 모든 행위가 실패하면 스텝은 실패한다. 따라서 any 키워드는 비결정적 행위를 지정할 수 있게 한다.

SCS에서 스텝의 동작 방식이 all로 지정되면 그것의 내부에 정의된 일련의 행위는 동시에 수행하거나 임의의 순서로 수행하도록 정의하고 있다. 그러나 행위의 동시 처리는 공유 데이터의 동기화와 조건 처리의 동시성 문제로 인해 작업 실행기의 구현에 어려움이 따른다. 따라서 현재의 작업 실행기는 all로 지정된 스텝에 대해 내부의 행위를 임의의 순서로 수행한다. 내부의 스텝 중 하나라도 실패하면 all로 지정된 스텝은 실패하고 모두 성공하면 스텝은 성공한다. 행위의 동시 수행은 차후 구현될 예정이다.

동작 방식이 first 또는 best로 지정된 스텝은 최선우선 행위를 구현한다. 스텝 내부에 정의된 일련의 행위 중 사전 조건과 지속 조건을 만족하는 행위에 대해 first 동작 방식은 첫 번째 만족되는 행위를 수행하고, best 동작 방식은 조건을 만족하는 행위 중 유틸리티(utility)가 가장 높은 행위를 수행한다. 이 때 유틸리티는 정수 또는 실수를 반환하는 자바 표현식으로 정의된다. first 동작 방식의 경우 내부의 행위를 선택하여 수행하던 도중 그것보다 먼저 정의된 행위의 조건이 만족되면 그것으로 수행을 대체하여 항상 정의된 행위들 가운데 가장 먼저 조건을 만족하는 행위를 수행하게 된다. 반면 best 동작 방식의 경우 내부의 행위가 정의된 순서와 관계 없이 유틸리티가 가장 높은 행위를 수행하게 된다. 유틸리티는 지속적으로 평가되므로 수행 중 값이 변경될 수 있으며, 따라서 선택되어 수행 중인 행위보다 더 높은 유틸리티를 갖는 행위가 발생하면 수행을 대체한다.

SCS에 기반한 SPRIT이 다른 개발 언어와 다른 가장 큰 특징은 어떤 행위에 대해 사전 조건과 지속 조건을 지정할 수 있다는 것이다. 사전 조건이 지정된 행위는 그 조건을 만족해야만 행위를 시작할 수 있으며 따라서 사전 조건은 행위 수행 직전에 한번만 검사된다. 반면 지속 조건이 지정된 행위는 그 조건을 만족해야만 행위를 지속할 수 있으며 행위 수행 중 지속 조건을 만족하지 못하면 즉시 그 행위의 수행을 중지한다. 그러나 이것은 행위의 실패를 의미하지는 않는다. SPRIT에서 사전 조건과 지속 조건은 true/false를 반환하는 자바 표현식으로 정의된다. 다시 말해 조건은 <, <=, ==, =!, >, >= 등의 비교 연산자와 &&, ||, ! 등의 논리 연산자, 그리고 true/false를 반환하는 객체의 함수로 지정할 수 있다.

지속 조건이 지정된 행위는 조건이 실패하면 그 즉시 행위를 중지하는데, 경우에 따라서는 실패를 지연할 필요가 있다. 예를 들어 데이터베이스에 연결 중이거나 파일을 여는 등 시스템의 중요한 자원을 할당하여 수행하는 프로그램은 자원을 안전하게 반환할 수 있도록 해당 프로그램 영역을 보호할 수 있어야 한다. 이러한 영역은 atomic 키워드를 이용하여 해당 영역이 조건의 실패에 영향을 받지 않도록 보호할 수 있다. 따라서 atomic 키워드로 보호된 영역이 수행 중일 때 지속 조건을 실패하면 그 즉시 행위를 중지하는 대신 atomic 영역을 모두 수행하고 행위를 중지한다. 표 4는 atomic 영역을 활용한 예이다.

표 4 atomic 영역을 테스트하기 위한 SPRIT 프로그램

```
import java.util.LinkedList;

do {
    var $list = new LinkedList();
    var $array = { 1, 2, "Hello World" };
    var $arrayIndex = 0;
    repeat while ($arrayIndex < 3) {
        $list.add($array[$arrayIndex]);
        $arrayIndex = $arrayIndex + 1;
        // A - 아래 코드는 실행되지 않는다.
        do when ($arrayIndex == 3) {
            System.out.println("end of array");
        }
        // A
    }
    var $listIndex = 0;
    repeat while ($listIndex < $list.size()) {
        System.out.println($list.get($listIndex));
        atomic {
            $listIndex = $listIndex + 1;
            // B - 아래 코드는 A와 달리 수행된다.
            do when ($listIndex == $list.size()) {
                System.out.println("end of list");
            }
            // B
        }
    }
}
```

표 4의 SPRIT 프로그램은 $array에 저장된 배열의 값을 $list에 차례로 추가하고, $list에 저장된 요소들을 차례로 출력한다. 이 때 주석으로 표시된 A 영역은 실행되지 않는다. 왜냐하면 그것을 포함하는 repeat 스텝에 while 키워드로 정의된 지속 조건에 의해, 인덱스를 증가하여 마지막 인덱스를 가리키는 순간 지속 조건을 만족시키지 못하므로 repeat 스텝의 수행이 중지되기 때문이다. 그러나 B 영역은 같은 지속 조건임에도 불구하고 수행되는데, 이는 atomic 영역으로 선언하여 영역 내의 행위가 실패하지 않는 한 원자적으로 수행되어 지속 조건의 실패에 영향을 받지 않기 때문이다.

3.3 작업 실행기(Task Executor)

SPRIT은 다양한 로봇 기술과 쉽게 연계될 수 있어야 하고, 다양한 로봇 플랫폼 및 개발 환경에서 동작할 수 있어야 한다. 이러한 설계 목표를 달성하기 위해 우리는 SPRIT의 작업 실행기를 자바 언어를 이용하여 구현하였다. 따라서 해석기는 자바 가상 머신이 동작하는 환경이라면 어디서든 동작이 가능하므로 다양한 플랫폼에서 SPRIT 프로그램을 수행시킬 수 있다.

그러나 많은 경우 로봇의 인식 및 기능을 구현한 소프트웨어들은 C/C++ 언어로 구현되며 이렇게 구현된 로봇 소프트웨어는 자바 언어에서 직접 사용할 수 없기 때문에 SPRIT 역시 직접 사용할 수 없다. 이러한 경우 Java Native Interface(JNI) 기술을 이용하면 C/C++로 작성된 함수나 객체를 자바 객체화할 수 있으며[9], 이를 통해 C/C++로 작성된 로봇 소프트웨어를 SPRIT에서 사용할 수 있다.

SPRIT과 자바 객체를 연동하기 위해 해석기는 Reflection 기술[10]을 사용한다. 이 기술을 사용하면 동적으로 자바 객체의 멤버 및 함수를 접근할 수 있다. 그러나 SPRIT에서 변수는 자바 언어에서와 달리 타입이 비결정적이므로 실행 시간에 타입을 결정해야만 한다. 변수의 타입을 결정하는 시점과 결정 방법은 표 5와 같다.

4. 실 험

SPRIT이 지능 로봇의 복잡한 행위를 효과적으로 정의할 수 있으며, 기존의 다양한 로봇 기술과 매우 쉽게 연계될 수 있음을 확인하기 위해 표 6과 같은 시나리오를 정의하고 SPRIT을 이용하여 로봇 시스템을 구현하였다.

시나리오에서의 제약 조건을 분석하면, 로봇의 작업은 중요도에 있어 일련의 순서를 갖는다. 빨간 공을 주시하는 작업의 우선 순위가 가장 높으며 공간 탐색 작업의

표 5 변수의 타입 결정 시점과 방법

결정 시점	결정 방법
사칙 연산	+, -, *, /, %와 같은 연산을 수행할 때 변수가 사용된 경우 변수의 평가 값에 대해 타입을 Integer, Long, Float, Double, String 타입 중 하나로 결정한다.
논리 연산	&&, ||, !과 같은 연산을 수행할 때 변수가 사용된 경우 변수의 평가 값에 대해 타입을 Boolean 타입으로 결정한다.
변수로부터 객체의 멤버/함수 접근	변수가 객체를 참조하고 있고, 이 변수를 이용하여 객체에 정의된 멤버 또는 함수를 접근하는 경우 변수의 평가 값을 해당 객체의 타입으로 결정한다.
변수가 객체에 정의된 함수의 인자로 주어지는 경우	객체에 정의된 함수를 호출할 때 변수가 함수의 인자로 주어지는 경우, 우선 인자로 주어지는 각각의 변수를 평가한다. 그리고 나서 자바 reflection 기술을 이용하여 객체에 정의된 함수 목록을 취득하고 각각 정의된 함수에 대해 함수가 정의한 인자의 타입으로 변수의 평가 값을 캐스팅할 수 있는지 판단하여 결정한다. 만약 정의된 함수의 인자와 변수의 평가 값이 배열인 경우 원소들의 타입이 단일화 될 수 있는지 판단하여 결정한다.

표 6 실험 시나리오

로봇은 다음 중 하나의 행위를 수행한다.
- 아무 것도 하지 않는다.
- 주변 공간 탐색한다.
- 사용자의 조작에 따라 이동한다.
- 빨간 공을 발견하면 그것을 주시한다.

그리고 다음과 같은 제약 조건을 갖는다.
- 공간 탐색보다 사용자의 조작이 우선한다.
- 빨간 공을 주시하는 행위는 다른 행위들보다 우선한다.

우선 순위가 가장 낮다. 이러한 로봇의 동작 방식은 first 키워드를 이용하여 표 7과 같이 정의할 수 있다.

그러나 로봇의 작업을 우선 순위에 따라 순서대로 정의하면 새로운 작업을 로봇에 추가할 때 기존 작업들의 우선 순위를 모두 고려해야 하므로 확장성이 떨어진다. 그렇기 때문에 이러한 경우 로봇의 행위를 우선 순위에

표 7 동작 방식을 first로 지정한 로봇 시스템

```
repeat first {
    repeat while [Camera.hasRedBall] {
        lookAt;
    }
    repeat while [Joystic.isButtonPressed] {
        ratioMove;
    }
    do any {
        repeat { wander; }
        repeat { doNothing; }
    }
}
```

표 8 동작 방식을 best로 지정한 로봇 시스템

```
repeat best {
    repeat while [Camera.hasRedBall] utility: 100 {
        lookAt;
    }
    repeat while [Joystic.isButtonPressed] utility: 50 {
        ratioMove;
    }
    repeat utility: 0 { wander; }
    repeat utility: 0 { doNothing; }
}
```

기반한 first 스텝으로 정의하는 것보다 유틸리티에 기반하여 최선 우선 행위를 수행하는 best 스텝으로 정의하는 것이 보다 효과적이다. best 스텝을 이용하여 다시 정의된 로봇 시스템은 표 8과 같다. 유틸리티는 계산식이나 함수 호출 등으로도 표현 가능하지만 여기서는 간단히 정수로 정의하였다. 아무 일도 하지 않거나 탐색 작업을 수행하는 것은 any 스텝으로도 정의 가능하지만, 같은 유틸리티를 갖는 개별 작업으로도 정의가 가능하다. 평가 시점에서 같은 유틸리티를 갖는 행위는 해석기가 임의의 하나를 선택하므로 any 스텝과 동일한 방식으로 동작한다.

실험을 위한 로봇 프로그램은 SPRIT을 이용하여 표 8의 SCS를 기반으로 작성하였고 MobileRobots 사의 로봇 시뮬레이터인 MobileSim[11]을 이용하여 로봇의 동작을 검증하였다. MobileSim은 Pioneer 시리즈의 로봇과 동일한 인터페이스를 갖는 로봇 시뮬레이션 도구이다. 또한 사용자 조작을 위해 닌텐도 Wii 리모컨을 사용하였으며, 빨간 공의 추적을 위해 색상 추적 시스템인 ACTS[12]를 사용하였다(그림 1).

사용된 로봇의 기능 중 일부는 C++ 언어로 구현되어 있었으나 Java Native Interface(JNI) 기술을 이용하여 쉽게 자바 객체와 연계할 수 있었으며 이렇게 생성된 자바 객체는 SPRIT 내에서 문제 없이 바로 사용할 수 있었다.

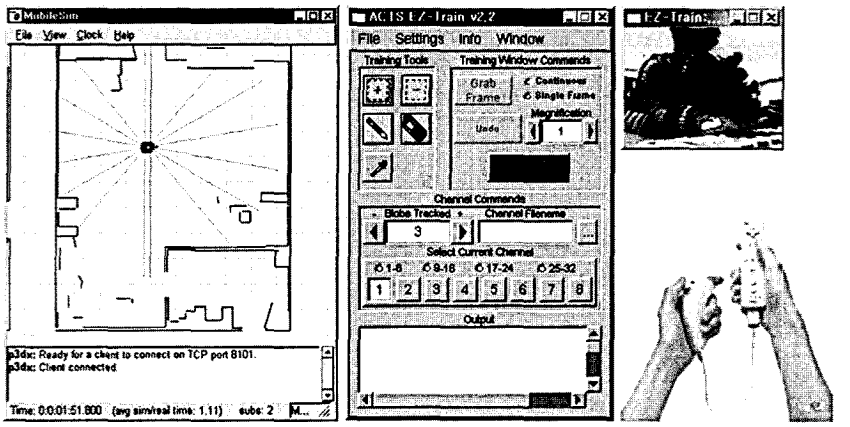

그림 1 실험 환경

이렇게 SPRIT으로 구현된 로봇은 일정 시간 동안 아무 일도 하지 않거나 주변 공간을 탐색하였으며 임의의 시점에 조이스틱을 조작하면 그 즉시 하던 일을 중지하고 조이스틱의 입력에 따라 이동하였다. 또한 탐색을 하거나 조이스틱의 입력에 따라 이동하던 중이라도 빨간 공을 발견하면 그 즉시 하던 일을 멈추고 빨간 공을 주시하였다.

5. 결론 및 향후 연구 계획

본 논문에서는 복잡하고 다양한 환경 내에서 상황 변화에 적절히 적응하여 행위를 수행하는 지능 로봇을 보다 효과적으로 구현하기 위한 프로그래밍 언어인 SPRIT과 SPRIT으로 구현된 프로그램을 다양한 로봇 환경에서 실행하고 검증하기 위한 작업 실행기를 제시하였다.

SPRIT은 SCS에 기초를 두어 상황에 따른 최선의 행위 결정, 비결정적 행위 지정 등을 매우 효과적으로 지정할 수 있으며, 각각의 행위는 로봇 환경 및 상황에 따라 수행이 결정되도록 사전 조건과 지속 조건을 지정할 수 있다. 또한 자바 객체를 직접 사용할 수 있으며, 프로그래밍 문법이 자바 언어와 매우 유사함에 따라 개발자가 새로운 언어를 학습하는 어려움을 최소화할 수 있다.

SPRIT에서 자바 객체를 직접 사용함에 따라 기존 로봇 소프트웨어 컴포넌트들을 매우 쉽게 연계할 수 있다. 다양한 로봇 소프트웨어 컴포넌트들이 자바나 C, C++ 언어 등으로 구현되는데 자바 언어로 구현된 소프트웨어 컴포넌트는 SPRIT 내에서 직접 사용이 가능하며, C/C++ 언어로 구현된 소프트웨어 컴포넌트는 JNI 기술을 통해 자바 객체를 생성하여 사용할 수 있다.

작업 실행기는 자바 언어로 구현되어 자바 가상 머신이 동작하는 모든 실행 환경에서 동작할 수 있으므로 다양한 로봇 플랫폼 및 실행 환경에서 SPRIT 프로그램을 실행할 수 있다.

그러나 현재의 SPRIT은 프로그램의 모듈화를 지원하기 않기 때문에 지능 로봇의 모든 행위를 단일 프로그램으로 구현할 수밖에 없고 구현된 SPRIT 프로그램을 다른 로봇 환경에서 재사용할 수 없다. 따라서 향후 SPRIT에 모듈화를 지원할 계획이며, 모듈화 지원을 통해 기존에 구현된 다양한 단위 로봇 행위들을 조합하여 새로운 로봇을 보다 빠르고 효율적으로 구현할 수 있을 것으로 기대된다.

참 고 문 헌

[1] Jaeho Lee and Byulsaim Kwak, "A Task Management Architecture for Control of Intelligent Robots, PRIMA 2006, pp.59-70.

[2] Robert T. Pack, "IMA: The Intelligent Machine Architecture," PhD thesis, Vanderbilt University, Nashville, Tennessee, 2003.

[3] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[4] Microsoft, Microsoft Robotics Developer Studio, http://msdn.microsoft.com/robotics

[5] Jean-Christophe Baillie, "URBI: toward a universal robotic low-level programming language," *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Aug, 2005.

[6] Geoffrey Biggs and Bruce A. MacDonald, "Evaluating reactive semantics for robotics," *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept, 2008.

[7] Marcus Huber, "JAM, A BDI-theoretic Mobile Agent," *Proceedings of the Third International Conference on Autonomous Agents (Agents-99)*, May, 1999.

[8] Jaeho Lee and Edmund H. Durfee, "Structured circuit semantics for reactive plan execution systems," *Proceedings of the twelfth national conference on Artificial intelligence* (vol. 2), pp.1232-1237, October 1994.
[9] Java Native Interface, http://java.sun.com/javase/6/docs/technotes/guides/jni/
[10] Java Reflection, http://java.sun.com/j2se/1.5.0/docs/guide/reflection/index.html
[11] MobileSim, http://robots.mobilerobots.com/wiki/MobileSim
[12] ACTS, http://robots.mobilerobots.com/wiki/ACTS

이 재 호

1985년 서울대학교 계산통계학과 학사
1987년 서울대학교 계산통계학과 석사
1997년 University of Michigan, Computer Science & Engineering, 박사
1996년~1998년 Principal Engineer, ORINCON Corporation, San Diego, USA
1998년~현재 서울시립대학교 전자전기컴퓨터공학부 교수
관심분야는 인공지능, 에이전트 시스템, 지능 로봇, 전자문서

곽 별 샘

2006년 서울시립대학교 전자전기컴퓨터공학부 학사. 2008년 서울시립대학교 전자전기컴퓨터공학부 석사. 2010년~현재 서울시립대학교 전자전기컴퓨터공학부 박사과정. 관심분야는 지능형 에이전트 시스템, 지능 로봇 시스템