계획 지식 모델링 도구의 설계 및 구현

최재혁° 김인철
경기대학교 전자계산학과
{01choi°, kic} @kyonggi.ac.kr

Design and Implementation of a Plan Knowledge Modeler

Jae-hyuk Choi°, In-Cheol Kim
Department of Computer Science, Kyonggi University

요 약

전통적인 인공지능 계획방식은 완전한 월드 상태모델과 시스템 동작모델에 기초하여 처음부터 자동으로 작업계획을 생성하려는 접근방식이다. 그러나 지능로봇제어와 같이 불확실성과 가변성이 높은 실 세계 응용분야에서 이와 같은 전통적인 인공지능 계획방식은 효과를 얻기 어렵다. 반면에 많은 실 세계 응용분야에서는 그 분야에서 이미 잘 알려져 있는 작업 영역지식이나 제어지식들이 존재하며, 이들을 효과적으로 이용하는 것이 매우 중요하다. 이러한 방법 중의 하나로서 복잡도가 높은 작업계획을 전문가가 직접 편집해서 입력하는 방식이 널리 쓰인다. 기본 동작모델과는 달리, 일반적으로 작업계획 표현언어는 복잡한 제어구조를 포함하는 하나의 작업 프로세스로 계획을 표현한다. 따라서 이러한 복잡한 절차적 지식인 작업계획을 편집하고 검증하기 위해서는 편리한 모델링 도구의 개발이 필요하다. 본 연구에서는 PRS 계열의 작업계획을 비주얼 환경에서 편집할 수 있고, 가상 시뮬레이션 기능과 작업 계획기와의 연동 기능을 갖춘 PKM시스템의 설계와 구현에 대해 설명한다.

1. 서론

지능로봇제어와 같은 실 세계 응용분야에서 전통적인 인공지능 계획방식을 적용하기에는 몇 가지 어려움이 있다. 첫째는 전통적인 인공지능 계획방식에서는 작업계획 과정과 실행 과정이 서로 분리되어 있으므로, 실 세계의 불확실성과 가변성을 계획생성 단계에서 충분히 고려하기 어렵다는 점이다. 둘째는 많은 실 세계의 계획문제는 복잡도가 높아서 이미 잘 알려져 있는 영역지식과 제어지식을 효과적으로 이용하지 못하면 자동 계획생성이 매우 어려울 수 있다는 점이다. 셋째는 대부분의 전통적인 인공지능 계획방식은 동작들의 시퀀스(sequence)로 표현된 작업계획을 생성할 뿐이나 지능로봇제어와 같은 실 세계 응용분야에서는 조건부 분기(conditional branch)나 반복(loop) 등의 복잡한 제어구조를 가진 작업계획을 요구한다는 점이다. 따라서 최근 들어서는 순수한 자동 계획생성보다 작업계획을 전문가에 의해 수동 혹은 반자동의 방법으로 생성하고 관리하는 지식공학적 접근방식이 많이 시도되고 있다.

작업계획 관련 지식공학 도구들로는 크게 계획 영역지식 편집기와 작업계획 편집기들이 있다. 본 논문에서는 이들의 특징과 대표적인 예들을 살펴본 뒤, PRS 계열의 작업계획을 편집할 수 있는 비주얼 모델링 도구인 PKM 시스템의 설계와 구현에 대해 자세히 설명한다.

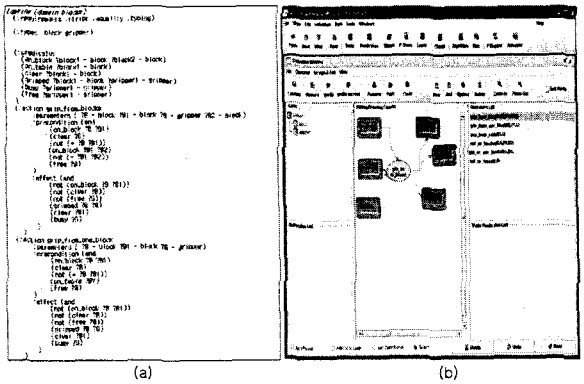
2. 관련연구

2.1 영역지식 편집기

전통적인 인공지능 계획기(planner)의 입력으로 주어지는 동작모델(action model)과 계획문제(planning problem)를 주로 계획영역지식(planning domain knowledge)이라 부른다. 이러한 계획영역지식을 위한 표현언어는 STRIPS와 ADL, UWL 등을 거쳐 PDDL(planning domain definition language)이라는 표준언어로 발전하였다. 최근 들어서는 PDDL 언어를 지원하는 몇 가지 비주얼 편집 도구들이 개발되어 인공지능 계획기의 활용을 더욱 촉진하고 있다.

GIPO [1]는 [그림1]의 (b)와 같은 비주얼 인터페이스를 이용하여 영역지식을 표현하는 편집기이다. GIPO는 영역지식을 정의하기 위해 사물들의 종류(sort)와 이들의 속성(attribute)을 나타내는 서술자(predicate)들을 먼저 정의한다. 그리고 서술자의 집합으로 가능한 상태들을 모두 열거한 다음, 이전 상태와 이후 상태의 쌍으로 각 동작을 정의한다. 이와 같이 정의된 영역지식은 GIPO의 기본언어인 OCL외에 표준언어인 PDDL로 변환이 가능하다. [그림1]의 (a)는 PDDL로 변환된 영역지식을 보여주고 있다. 또, GIPO는 이 밖에도 자체 작업계획기를 내장하고 있어 편집된 영역지식을 바탕으로 직접 계획생성을 해볼 수 있고, 편집된 지식에 대한 검증 기능과 시뮬레이션 기능 등도 제공한다. GIPO 이외의 대표적인

영역지식 편집기로는 itSIMPLE과 ModPlan, PlanWorks 등이 있다. 이들 중 특히 itSIMPLE[2]은 영역지식을 UML형태로 표현할 수 있는 비주얼 편집 기능을 제공하며, PDDL, XML등의 다른 언어로 변환할 수 있는 기능을 제공한다. [표 1]은 계획 영역지식 편집기들의 특징을 비교하여 나타내고 있다.



(a) (b)

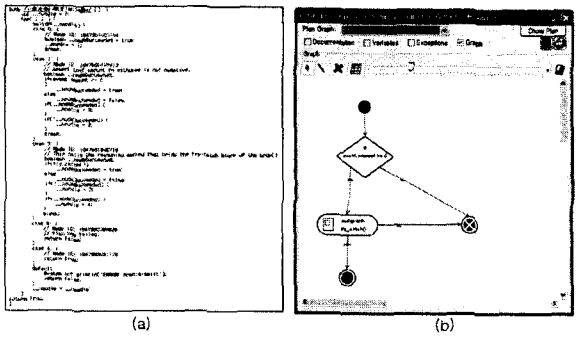
그림 1. 영역지식 편집기 GIPO

표 1. 영역지식 편집기의 비교

비교	GIPO	itSIMPLE	ModPlan	PlanWorks
표현언어	OCL, PDDL	PDDL, XML	STRIPS, PDDL	NDDL, PDDL
내장 계획기	HyPlan	N/A	Durative-FF	EUROPA2
실행환경	Unix, Windows	Windows	Windows (Cygwin)	Unix, Mac
개발언어	Java	C/C++	Java, C/C++, Python	Java
특징 및 부가기능	Verification, Animation, Stepper	UML model *XML files* Classical PDDL model	*Validation*, Visualization	*Debug*, Visualization

2.2 작업계획 편집기

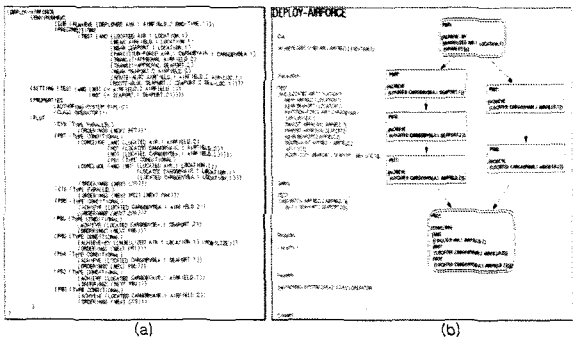
계획 영역지식 편집기가 작업 계획기의 입력이 되는 동작모델과 계획문제를 편집하는 것과는 달리, 작업계획 편집기는 작업계획 자체를 표현하고 수정하는 지식공학 도구이다. 계획 영역지식은 PDDL과 같은 표준언어가 존재하는데 반해, 작업계획 표현언어는 아직 표준이 마련되지 않은 상태이다. 하지만 지능로봇제어 분야와 소프트웨어 에이전트 분야 등 다양한 응용분야에서 PRS(Procedural Reasoning System)에 기초한 작업계획 표현언어들이 가장 보편적으로 쓰이고 있다.



(a) (b)

그림 2. JACK Development Environment

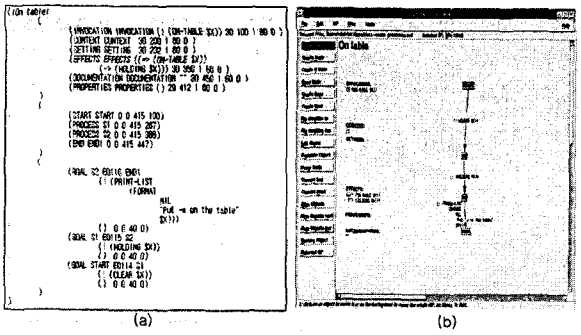
대표적인 작업계획 편집도구의 하나로 JACK Development Environment[3]가 존재한다. JACK은 Java기반의 지능형 에이전트 개발 환경으로서, 실행엔진 이외에 다양한 지원도구들을 제공한다. 이들 중 특히 JACK Development Environment는 에이전트 행위의 기초가 되는 계획을 시각적으로 정의할 수 있는 도구이다. JACK DE에서는 작업계획의 Annotation 정보를 입력할 수 있는 인터페이스와 더불어 [그림 3]의 (b)와 같이 작업계획의 Body부분을 구성하는 프로시저를 편집할 수 있는 비주얼 인터페이스를 제공한다. JACK DE를 이용해 작성된 에이전트의 작업계획은 [그림 3]의 (a)처럼 텍스트 형태로 변환되고 실행된다.



(a) (b)

그림 3. ACT Editor

또 다른 작업계획 편집기인 ACT Editor[4]는 작업 계획기인 SIPE와 실행기인 PRS의 공통 계획 표현언어인 ACT를 지원한다. ACT에서 하나의 작업계획은 CUE, Precondition 등을 포함하는 Annotation부분과 Body 프로시저로 구성된다. ACT Editor는 [그림 3]의 (b)와 같이 시각적으로 정의된 작업계획을 [그림 3]의 (a)와 같이 ACT 언어로 표현된 작업계획으로 변환해준다. ACT Editor에서 작업계획의 Body를 구성하는 단위 동작은 하나의 그래프 노드(node)로 표현되고, 동작들간의 실행제어 흐름은 유향 아크(directional arc)로 표현된다.



(a) (b)

그림 4. OP Editor

이 밖에도 Open PRS시스템에서 제공하는 PRS 작업계획을 정의하기 위한 도구인 OP Editor[5]가 있다. OP Editor는 작업계획 편집을 위한 [그림 4]의 (b)와 같은 사용자 인터페이스를 제공하며, 이것을 이용해 [그림 4]의 (a)와 같은 하나의 PRS 작업계획을 생성할 수 있다. [표 2]는 작업계획 편집기들의 특징을 비교하여 나타내고 있다.

표 2. 작업계획 편집기의 비교

비교	JACK DE	ACT Editor	OP Editor
표현언어	JAL	ACT	PRS
제어구조	Java control constructs	PARALLEL, CONDITIONAL, ACHIEVE	IF/THEN/ELSE, GOAL,
실행환경	Unix, MS Windows	Solaris X-Win	Unix X-Win
개발언어	Java	C	C
특징 및 부가기능	Debugger, Compiler, Runtime Environment	Dictionary, Verification, Birds-Eye View	TeX-doc export, Visualization

작업 영역지식 편집기들과는 달리, 기존의 작업계획 편집기들은 대부분 계획 시뮬레이션 기능이나 실행 모니터링 기능을 제공하지 못하며, 작업 계획기 및 실행기와의 연동 기능도 충분히 제공하지 못하고 있다. 또 대부분의 편집기 실행환경이 특정 플랫폼에 종속적인 경우가 많고 상용제품인 경우도 있어, 이용에 어려운 점이 많다.

3. 작업계획 언어

본 논문에서 JAM 기반의 작업계획을 편집하고 실행할 수 있는 지식공학 도구인 PKM을 설계한다. JAM[6]은 BDI(Belief-Desire-Intention) 구조에 기초한 작업계획 실행기이며, 가장 보편적인 작업계획 표현언어인 PRS를 확장한 JAM 스크립트언어로 작업계획을 표현한다. JAM 작업계획 실행기는 Interpreter, Plan Library, World Model, Observer, Intention Structure, Sensor와 Effector 등으로 구성된다. Plan Library는 작업계획이 저장되어 있는 곳이며, World Model은 외부 상태 정보를 저장한다. Interpreter는 상황에 적합한 작업계획을 선택하여 실행하는 JAM의 핵심부이다. Sensor와 Observer를 통해 받아들여진 외부 상태 정보로 World Model을 갱신하고, Interpreter는 이렇게 갱신된 World Model의 상태 정보를 바탕으로 Plan Library에 저장된 작업계획들 중 하나를 선택한다. 선택된 작업계획은 *Intention Structure*에 저장되었다가 Effector를 통해 실행된다.

```
Plan :
{
        GOAL : [goal specification]
               OR
        CONCLUDE : [world model relation]
        NAME : [string]
        BODY : [procedure]
        <DOCUMENTATION : [string]>
        <PRECONDITION : [expression]>
        <CONTEXT : [expression]>
        <UTILITY : [numeric expression]>
        <FAILURE : [non-subgoaling procedure]>
        <EFFECTS : [non-subgoaling procedure]>
        <ATTRIBUTES : [string]>
}
```

그림 5. 작업계획 표현

하나의 JAM 작업계획은 다른 PRS 계열의 작업계획 표현언어와 마찬가지로 Annotation과 Body프로시저로 나눌 수 있다. [그림5]는 JAM 스크립트 언어로 정의되는 작업계획의 구조를 나타낸다. JAM 작업계획은 NAME, GOAL, PRECONDITION, CONTEXT, UTILITY, FAILURE, EFFECTS, ATTRIBUTTS 등 다양한 Annotation 정보를 포함할 수 있다. Body 프로시저는 크게 Action과 Constructor 등 두 가지 형태의 구성요소들로 표현되며, Action은 단순한 단위동작을, Constructor는 반복이나 분기와 같은 복잡한 제어구조를 나타낸다. Action들로는 ACHIEVE, PERFORM과 같이 새로운 목표를 설립하는 동작들과 EXCUTE, TEST 와 같이 외부 함수 호출이나 테스트 동작, SUCCEED, FAIL과 같은 기타 동작들이 있다. 이에 반해 Constructor들로는 WHILE과 같은 반복, AND나 OR와 같은 논리연산, IF 와 같은 분기 제어구조들이 있으며, 이 제어구조들은 다수의 *Action*들을 포함할 수 있다.

4. 시스템의 설계

본 연구에서는 PRS 계열의 작업계획 표현언어인 JAM 스크립트언어를 지원하는 비주얼 작업계획 편집기 PKM을 개발한다. 이 작업계획 편집기는 실행기인 JAM과 연동되어 편집 작업과 시뮬레이션 작업을 수행한다. JAM 작업계획 편집기인 PKM을 개발할 때, 이미 존재하는 JAM의 내부 자료구조를 이용하고 확장하는 방법과, 별도의 독립적인 자료구조로 구현하는 방법이 존재한다. JAM의 자료구조를 확장하는 방법은 작업계획 실행기인 JAM과의 연동에서는 높은 효율성을 갖출 수

있으나, JAM에 대한 의존성이 높아 작업계획 편집기의 범용성에 제한이 있다. 반면에, JAM과는 독립적인 자료구조들로 작업계획 편집기를 구현할 경우, JAM과의 연동을 위해 부가적인 인터페이스 구현이 요구되지만 향후에 있을 표준 작업계획 언어에 대한 대처가 용이하고 범용성과 확장성 면에서 유리하다.

본 연구에서 설계한 PKM(Plan Knowledge Modeler)은 JAM 스크립트언어에 기초한 작업계획 편집 기능을 제공할 뿐 아니라, 작업 실행기인 JAM과 연동함으로써 작업계획에 대한 시뮬레이션 기능을 제공할 수 있고, 외부 작업 계획기와의 연동을 통해 편집 중에 작업계획의 Body 프로시저를 자동 생성하거나 변경할 수 있는 기능을 제공한다

[그림6]은 PKM 시스템의 전체 구성을 나타내고 있다. PKM에는 사용자가 작업계획을 편집할 수 있는 사용자 인터페이스가 있고, 사용자 인터페이스에서는 작업계획의 Annotation 정보를 편집할 수 있는 툴과 작업계획의 Body 프로시저를 정의할 수 있는 모델러를 제공한다. Annotation 편집은 사용자가 정보를 입력기를 통해 입력한다. Body 프로시저는 모델러를 이용해 그래픽 아이콘으로 표현하고, 이 아이콘을 링크로 연결하여 작업계획으로 표현한다. 표현된 작업계획은 그래픽 표현을 위한 내부 자료구조로 저장된다. 이때 설계된 내부자료는 PRS 계열 작업계획 언어들의 공통요소를 중심으로 설계함으로써 향후 표준으로의 확장을 용이하게 설계하였다.

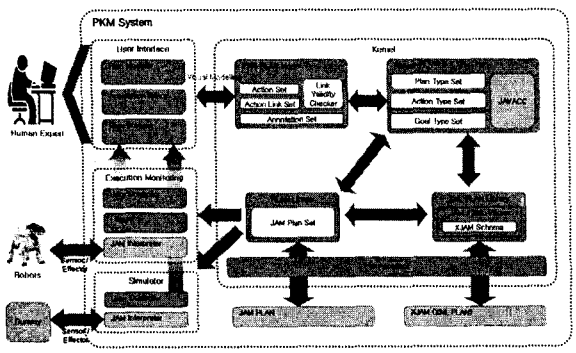


그림 6. 시스템 구성도

사용자가 비주얼 편집기를 이용하여 작업계획을 정의하거나 편집하면, 변환기를 통해 이것을 텍스트 기반의 JAM 작업계획으로 변환할 수 있다. 변환된 작업계획은 [그림 6]과 같이 JAM의 Plan Library에 저장된다. 이와 같은 과정으로 생성된 작업계획들은 외부의 가상함수와 연결한 뒤 JAM 실행기를 수행함으로써 기능을 시뮬레이션 해 볼 수 있다. 또한 이들 작업계획을 실제 로봇의 Sensor와 Effector에 대한 호출함수와 연결함으로써 작업계획의 실행에 따른 로봇의 상태를 추적하고 모니터링 할 수 있다. 모니터링과 시뮬레이션 과정을 시각화하는 그래픽 사용자 인터페이스를 갖추고 있어, 사용자가 쉽게 작업계획의 실행상태를 파악할 수 있도록 한다. 실행 상태에 따라서는 사용자가 PKM을 이용해 온라인 상태에서 기존의 작업계획을 변경하거나 새롭게 정의해줌으로써 실행 중 발생 가능한 예외 상황에 대처할 있다. 또한 작업계획의 공유와 재활용성을 높이기 위해 작업계획을 XML 형태로 변환해주는 기능도 제공한다.

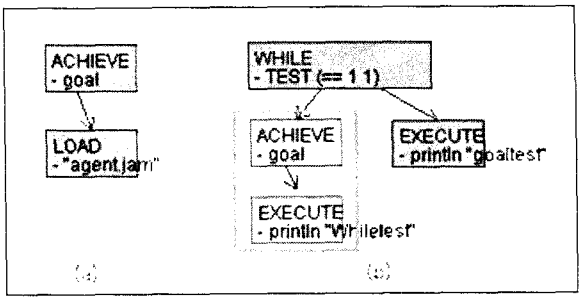


그림 7. 작업계획 요소의 시각화

JAM 작업계획의 Body 프로시저 편집 기능은 PKM 시스템의 가장 핵심 기능이다. 임의의 한 Body 프로시저는 몇 개의 Action과 Construct들로 구성된다. Body 프로시저 편집기에서 Action과 Construct는 [그림 7]과 같은 시각요소(visual object)들로 표현한다. [그림 7]의 (a)은 Action으로만 구성된 한 작업계획을 표현한 것이고, [그림 7]의 (b)는 하나의 WHILE Constructor가 포함된 작업계획을 표현한 것이다. 모든 단위 동작들, 즉 Action들간의 실행순서는 화살표로 표현하였다. 따라서 작업계획이 일련의 Action들로만 구성되는 경우, [그림 7]의 (a)와 같이 Action 노드들과 그들의 연결하는 유향 링크로만 작업계획이 표현된다. 반면에 작업계획에 Constructor를 포함하는 경우, [그림 7]의 (b)와 같이 조건 테스트를 나타내는 하나의 Constructor 노드와 좌우 2개의 가지(branch)로 표현된다. 그리고 각 가지는 하나의 서브 그래프를 가질 수 있다. 좌측 가지에는 Constructor 노드의 조건 테스트가 만족할 때 실행할 Action 노드들의 집합이 포함되고, 우측 가지에는 그 Construct 다음에 실행할 Action들의 집합이 포함된다. Constructor 노드의 좌측 가지에 포함되는 Action들은 하나의 박스로 묶어 그룹화하였고 박스 내부의 각 Action 노드들도 기본적인 Action 노드와 다른 색상으로 표시하였다. 우측 가지에 등장하는 Action 노드들은 기본적인 Action 노드와 동일한 모양과 색상을 사용하여 표현하였다.

5. 구현 결과

본 논문에서는 [그림 8]에 표현된 작업계획을 이용하여 구현된 결과물을 표현하였다. 이 작업계획에는 Annotation 부분에 작업계획 이름과 수행해야 할 Goal이 명시되어 있고, Body 프로시저에 적절하게 Action과 Constructor가 섞여 표현되어 있어 구현 결과물을 표현하는데 효과적이다.

```
PLAN {
        NAME:
                "GoToNextNode"
        GOAL:
                ACHIEVE GoToNextNode $nodeNumber $distance;
        BODY:
                RETRIEVE currentTarget $privNode;
                EXECUTE SetCurrentNode $privNode;
                EXECUTE SetTargetNode $nodeNumber;
                EXECUTE Stand;

                ACHIEVE SearchTarget;
                ACHIEVE Target;
                RETRIEVE currentTarget $currentTarget;

                WHILE : TEST(!= $currentTarget $nodeNumber)
                {
                        ACHIEVE SearchTarget;
                        ACHIEVE Target;
                        RETRIEVE currentTarget $currentTarget;
                };

                ACHIEVE TurnBody;

                RETRIEVE testFact $testFact;
                WHEN: TEST (== $testFact "FALSE")
                {
                        ACHIEVE SearchTarget;
                        ACHIEVE TurnBody;
                        UPDATE (testFact) (testFact "TRUE");
                };

                ACHIEVE WalkForward $distance;
}
```

그림 8. 작업 계획의 예

구현된 PKM은 [그림 9]와 같이 그림의 왼편에 작업계획의 목록이 나열되고, 상단에 각 기능별 아이콘이 존재하여, 각 아이콘을 클릭하여 기능들을 사용할 수 있다. 그림은 Annotation 정보를 편집할 수 있는 편집기의 그림으로, 이 편집기에서는 Annotation 부분에 포함되는 작업계획 이름과 설명, Goal, Context, Utility, Failure, Effects를 편집할 수 있다. 이름과 설명은 직접 기술하고, 표현하고자 하는 Annotation 타입을 선택하여, Annotation의 매개변수로 Action과 Action의 매개변수를 입력한다. 각 Action별로 사용법을 알 수 있도록 JAM의 작업계획 표현방법에 대한 가이드도 제공된다.

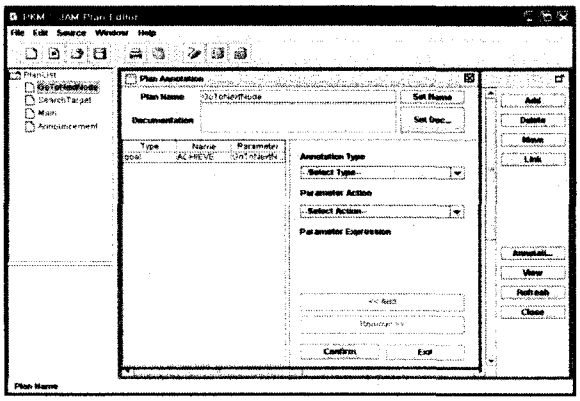


그림 9. Annotation 편집기

[그림 8]의 작업 계획 Body 프로시저는 [그림 10]과 같이 편집된다. 각각의 Action들은 다음순서의 Action들과 연결 되어 있고, Constructor는 내부 Action들의 집합과 다음 순서의 Action의 연결로 표현되어 있다. 사용자가 Action 선택과 매개변수의 입력을 하면 아이콘으로 표현 되며, 아이콘간의 연결 관계를 선택하면 그래프로 표현된다. Action과 Constructor는 크기로 구별되고, Action과 Constructor에 포함된 Action은 색과 Action을 포함하는 박스로 구별된다.

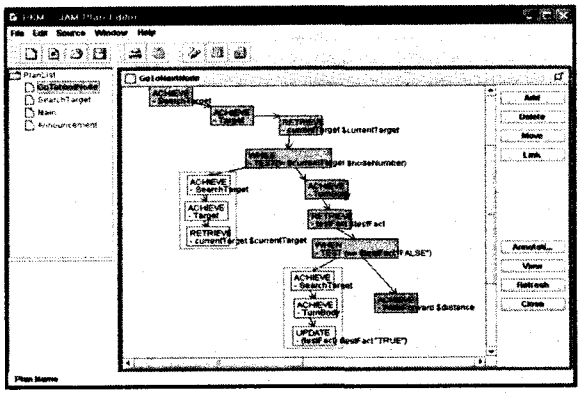


그림 10. Body 프로시저 편집기

아이콘으로 작성된 작업계획은 텍스트 파일로 된 작업계획의 형태로 변환하여 출력하거나 확인할 수 있는 기능으로 [그림 11]과 같은 Export 기능이 있다. 작업계획의 Annotation과 Body 프로시저의 내부 자료구조를 텍스트 형태의 작업계획으로 표현하여 나타낸다. Export에서는 각각의 작업계획만을 출력하여 주기도 하고, 여러 개의 작업계획이 모인 전체의 작업계획을 변환하여 시뮬레이션 할 수 있게 지원된다.

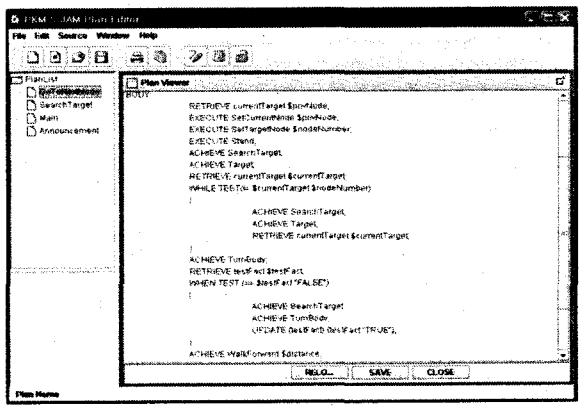


그림 11. 작업계획 Exporter

6. 결론

본 논문에서는 계획 영역지식 편집기와 작업계획 편집기들의 특징들을 살펴본 뒤, PRS 계열의 작업계획을 편집할 수 있는 비주얼 모델링 도구인 PKM 시스템의 설계와 구현에

대해 설명하였다. 이러한 PKM 시스템은 복잡도가 높은 실세계의 응용분야에서 신뢰도가 높은 양질의 작업계획을 편집하고 관리할 수 있는 지식공학 도구로 활용될 수 있을 것이다.

참고문헌

[1] R. M. Simpson, T. L. McCluskey, W. Zhao, R. S. Aylett and C. Doniat 2001. " An Integrated Graphical Tool to support Knowledge Engineering in AI Planning" Proc. of the 2001 European Conference on Planning, Toledo, Spain, 2001.

[2] TONIDANDEL, Flavio; VAQUERO, Tiago; SILVA, José Reinaldo. " The itSIMPLE tool for Modelling Planning Domains." Proc. of the International Competition on Knowledge Engineering for Planning (ICKEP) at International Conference on Automated Planning and Scheduling (ICAPS), 2005.

[3] Agent Oriented Software Pty. Ltd. " JACK Intelligent Agents." http://agent-software.com.au/jack.html

[4] Karen L. Myers. " The ACT Editor User' s Guide." Artificial Intelligence Center, SRI International, Menlo Park, CA, 1993.

[5] LAAS/CNRS. " LAAS Open Software for Autonomous Systems." http://softs.laas.fr/openrobots

[6] Marcus J. Huber, " JAM: A BDI-theoretic Mobile Agent Architecture", Proc. of the Third International Conference on Autonomous Agents (Agents'99), pp236-243, Seattle, WA, May 1999.