퍼지 및 지능시스템학회 논문지 2004, Vol. 14, No. 2, pp. 207-215

IEC1131-3 표준언어 처리를 위한 지능적 소프트웨어 PLC 개발

Development of an Intelligent Software Programmable Logic Controller for IEC1131-3 International Standard Languages

조영임

Young Im Cho

평택대학교 컴퓨터과학과

Dept. of Computer Science, Pyongtaek University

요 약

IEC1131-3의 PLC(Programmable Logic Controller) 프로그래밍 언어는 프로그래밍이 복잡하여 디버깅이 어려우며 범용성이 없을 뿐 아니라 국내는 PC기반 소프트웨어 PLC 연구개발이 매우 미약하다. 따라서 본 논문은 국제 PLC 표준언어로 제정된 5가지 언어 중 국내에서 90%이상을 사용하고 있는 PLC 언어인 LD언어에 대한 표준규격을 연구하고, LD를 기존 상용화된 편집기(Visual C++)에서 활용 가능한 C코드로 변환하여 LD에 익숙한 사용자나 고급언어에 익숙한 사용자 모두 사용할 수 있는 지능적 에이전트 기반의 통합 시스템 ISPLC(Intelligent Agent System based Software Programmable Logic Controller)를 개발하였다. ISPLC에서는 LD에서보다 C에서 논리오류 검출기능이 훨씬 효율적이며, GUI기반 인터페이스를 제공하며 에이전트에 의한 프로그래밍 코드를 제공한다. ISPLC는 초보자는 물론 PLC에 익숙한 사용자들에게도 효율적인 프로그래밍 플랫폼을 제공한다. 이러한 LD에서 IL로, IL에서 C로의 코드변환체제에 관한 연구는 국내외적으로 처음 시도되는 연구이다. ISPLC를 실제 실시간 교통량 제어 시스템에 적용하여 시뮬레이션한 결과 ISPLC가 오류검색 뿐 아니라 프로그래밍 시간을 기존 소프트에어 PLC에 비해 매우 단축시켜줌을 알 수 있었다.

Abstract

The PLC programming by IEC1131-3 is hard to handle to ordinary users as well as professionals. Also it has not a generality, so that it couldn't be debugging some logic errors easily. In order to be adapted for such environment, In this paper, I have developed the ISPLC(Intelligent Agent System based Software Programmable Logic Controller). In ISPLC system, LD programmed by a user is converted to the C code which can be used in a commercial editor such as Visual C++. The detection of logical errors in C code is more effective than PLC programming itself. ISPLC provides the GUI-based interface in web environment and an easy programming platform to such beginners as well as professionals. The study of code conversion of LD to IL as well as IL to C is firstly tried in the world as well as KOREA. To show the effectiveness of the developed system, I applied it to a practical case, a real time traffic control system. ISPLC is minimized the error debugging and programming time owing to be supported by windows application programs.

Key words : 소프트웨어 PLC, IEC1131-3, LD, PC기반 제어

1. 서 론

PLC(Programmable Logic Controller)란 PC라고도 불리우는 산업용 제어장비의 일종으로 PLC 프로그래밍에 의해 제어되고 있다[1,2,3,4]. 그러나 PLC 프로그래밍 언어는 특수목적 언어이므로 국가나 기종마다 서로 상이하고 표준화가 되어있지 않기 때문에 개방화와 분산화 시대에 맞지 않다는 문제점이 있다. 따라서 1993년 유럽을 중심으로 PLC 프로그램 언어의 표준활동이 추진되었는데 이 규격이 IEC의 TC65/SC65B/WG7/TF3에서 10여년 동안 검토 후 문서로 발행한 유일한 국제 산업용 표준인 IEC1131-3이다[5]

IEC1131-3에서는 다음과 같은 5개의 언어를 규정하고 있다. 텍스트계의 언어로 IL(Instruction List), ST(Structured Text)가 있고, 그래픽계의 언어로는 LD(Ladder Diagram)과 FBD(Function Block Diagram)가 있으며, 중요한 공통요소로 SFC(Sequential Function Chart)가 있다[6]. IL은 독일 비롯한 유럽에서 많이 쓰이며, FBD는 프로세스 제어의 신호 플로를 수반하는 애플리케이션용 회로도와 비슷한 형태의 언어이다. ST는 리얼타임 애플리케이션으로 개발된 프로그래밍과 유사하여 복잡한 펑션 블록의 정의에 효과적으로 사용된다. LD는 미국의 사다리형에서 기원한 언어인데, 일본이나 캐나다 등지에서 사용되고 있으며 우리나라에서도 90% 이상

접수일자 : 2003년 11월 5일
완료일자 : 2004년 1월 15일

의 기업들에서 LD를 사용하고 있다.

PLC는 PC개발과 함께 PC기반 제어(SoftLogic 또는 Software PLC)가 등장하게 되었다[1,2,3,4]. 1990년대 초 미국 GM사가 처음으로 PC기반으로 자동차 라인에 적용한 사례를 시점으로 현재는 미국을 비롯한 유럽 각지까지 PLC에서 PC기반 제어로의 전이가 일어나고 있다. PC 기반제어는 하드웨어적 PLC의 여러 단점들을 보완하고 뛰어난 성능과 유연성으로 PLC를 대체할 차세대의 솔루션으로 인정받고 있다.

그러나 국내에서는 이 기술에 대한 인지도 및 적용률이 미진하여 세계적인 변화에 편승하지 못하고 있는 실정이다. 또한 우리나라는 세계에서 장비출하 댓수가 1위임에도 불구하고 장비에 들어가는 제어기의 90% 이상을 일본에서 수입한 PLC 제어에 의존하고 있는 형편이다. 그러나 앞으로는 PC 중심의 반도체 장비 중심의 제어가 주류를 이루게 될 것이므로 PC 중심의 자동 제어방법에 관한 연구는 매우 중요한 의미를 갖는다. 반도체 장비에 하나의 패키지 형태로 제공하게 될 것이다.

IEC1131-3이라는 PLC 표준언어 규정에도 불구하고 여전히 PLC 프로그래밍 언어는 일반인이 사용하기에는 매우 어려운 언어이며 웹 환경에서 범용성을 갖기 어려운 언어이다. 또한 전문가라 해도 프로그램시 논리오류에 대한 디버깅이 매우 어렵고 복잡하다.

따라서 본 논문은 국제 PLC 표준언어로 제정된 5가지 언어 중 국내에서 90%이상을 사용하고 있는 PLC 언어인 LD 언어에 대한 표준규격을 연구하고, 이것을 중간코드인 IL언어로 변환하고 기존 상용화된 편집기(Visual C++)에서 활용가능한 표준 C코드로 변환함으로써 LD 전문가는 물론 고급언어에 익숙한 사용자들도 쉽게 사용할 수 있는 지능적 에이전트 기반의 소프트웨어 PLC 통합 시스템을 개발하고자 한다. IL이라는 중간코드 형태로 변환해야 할 필요성은 표준언어들간의 호환성을 주기 위함인데, IL은 어셈블리 언어와 유사한 형태로써 쉽게 표준언어들간의 변환이 가능하며 코드 재활용성이나 이식성 등 산업용으로 활용도가 매우 높기 때문이다. ISPLC에서는 LD에서보다 C에서 논리오류 검출기능이 훨씬 효율적이며, GUI기반 인터페이스를 제공하며 에이전트에 의한 프로그래밍 코드를 제공하므로 매우 효율적이다. LD와 IL언어간 호환에 관한 연구는 있어왔지만 IL을 고급언어로의 코드변환체제 및 LD에서 IL로, IL에서 C로의 코드변환체제에 관한 연구는 국내외적으로 처음 시도되는 연구이다. 본 논문에서 개발한 시스템을 실시간 교통량 제어 시스템(Real Time-Traffic Control System: RT-TC)에 적용하여 효율적 제어가 되도록 GUI 기반 웹 환경도 제공하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 국내외 PC-based control의 현황을 분석하고 3장에서는 개발시스템을 제안하고자 하며, 4장에서 RT-TC 시스템의 적용사례에 관한 시뮬레이션 및 결과를 토론하고 마지막으로 5장에서 결론 및 향후과제에 관해 설명하고자 한다.

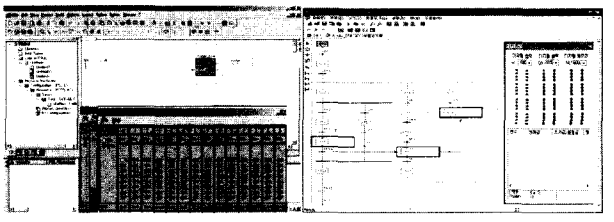
2. 국내외 소프트웨어 PLC 개발 현황 분석

기존에 개발된 PC-based control 제품들 중 국외 동향을 소개하면, Paradym-31[7]은 PC-PLC-I/O의 구성을 PC-I/O의 구성으로 단순화 시켜 가격과 신뢰도에 있어서 매우 경쟁력을 갖춘 것으로 각 PLC 마다 독특한 방식으로 프로그램하는 방식에서 표준화된 IEC 1131-3의 규격으로 프로그램을 함으로써 다양한 기종간의 프로그램에 있어서도 쉽게 구성할 수 있다. 독일의 GmbH에 의해 개발된 KW System[8]은 시스템의 환경, 규모, 복잡도가 증가함에 따라 사용자들이 툴을 사용하려는데 많은 노력과 시간을 소모하고 있는 점을 해결하고자 하여 사용자의 툴에 대한 친숙도를 고려하여 비쥬얼한 인터페이스를 제공해주고 있다. 사용자가 사용하기 원하는 툴의 구성을 윈도우 기반으로 5개 표준언어를 지원해 주고 있다. 이 밖에 관련 시스템인 Embedded Super PLCs[9]는 LD 프로그래밍을 기반으로 베이직언어와 통합하여 단일 비트와 디지털 I/O를 제어하는 데이터를 프로세싱하며 운영체제 환경에 따라 달리 프로그래밍 하도록 한 점이 특징이다. 이외에도 다양한 사례가 있으나 PLC언어를 고급언어로 변환한 사례는 아직 없다[10,11].

반면, 국내에는 소프트웨어 PLC가 산업용으로 개발되어 상용화된 사례가 아직은 없다. 그 이유는 국내 시장이 외국에 비해 비교적 작고 중소기업에서 투자하여 연구하기에는 인력 및 기술이 부족하고 수지타산이 맞지 않기 때문인 것으로 분석된다.

(주)리얼게인에서 교육용 PLC 언어 학습 패키지(RealPLC)를 개발하여 보급하고 있는데, PLC소프트웨어만으로 다양한 언어를 학습할 수 있고, 다양한 모니터링 및 애니메이션을 할 수 있어 적은 비용으로 최상의 PLC 학습을 할 수 있는 특징을 갖는다[12]. 그러나 이 제품은 교육용만으로 국한되어 있어 산업용으로는 활용하기 어렵다. 국내에서는 LG 산전과 삼성에서 개발 사용되고 있는 PLC 시스템들이 있다. 삼성에서 윈도우용 WinGPC[1]를 개발하였으나 자체 PLC용 장비에 대한 인터페이스만을 제공하고 있으므로 범용성이 부족하다.

그림 1에서 국외의 대표적인 소프트웨어 PLC인 KW System과 국내의 리얼게인 시스템의 인터페이스 및 장단점을 비교 분석하였다. 두 시스템 모두 IEC1131-3에서 정한 표준언어들의 인터페이스는 제공하나 표준 언어들간의 상호호환성이 부족하고 고급 언어로의 코드 변환은 제공하고 있지 않고 있지 않으므로 프로그래밍시 논리오류 검출기능이 없다.



(a) KW 시스템 (b) 리얼게인 시스템
(a) KW system (b) RealGain System
그림 1. KW System과 리얼게인 시스템 비교
Figure 1. The comparison of interface between KW system and Real Gain system

지금까지 살펴본 국내외 개발된 기존 소프트 PLC 시스템은 몇 가지 제한점이 있다. 즉, PLC 기반 언어로 프로그래밍을 하는 동안의 오류(특히 논리오류) 분석, 해석, 처리기능이 부족하고, 오픈 소스가 아닌 블랙 보드로서의 결과만을 확인 가능하다. 또한 IEC1131-3에서 제공하는 표준 언어들 간의 호환성이 부족하고 제한적인 시뮬레이션 기능을 가지고 있으므로 일반 사용자들의 기대를 만족시키기에는 부족한 점이

많으며 사용하기 어렵다. 또한 범용성이 부족하고 자체개발한 PLC용 언어만을 지원한다는 것이다.

따라서 본 논문에서는 에이전트[13,14,15]를 이용하여 PLC 프로그래밍이 가능한 툴 내에서 사용자에게 적합한 컴포넌트를 제공해 주고 그 결과를 사용자가 원하는 형태로 필터링하여 최적의 코드를 생성해 주며 프로그램시 발생하는 논리오류를 검출해 줌으로써 요구되는 사항들에 관한 자동적이고 자율적이며 전문적인 통합형 PLC 소프트웨어 에이전트 시스템을 개발하고자 한다.

3. IEC1131-3 표준언어 처리를 위한 소프트웨어 PLC 설계

3.1 개발 시스템의 개요

기존의 PLC를 대신하여 PC를 이용한 오픈 제어 시스템의 급속한 확산으로 PC상에서 운영되는 소프트웨어 PLC 시장이 급성장하고 하고 있으나 경쟁력 있는 국내 제품이 없는 실정이다. 따라서 본 논문에서는 해외 수출환경 및 국내 환경에 적합하고 90%이상 점유율을 나타내고 있는 IEC1131-3 표준 언어인 LD를 중간코드인 IL로 변환하고 고급언어(C)를 최종 코드로 함으로써 산업적 효율성이 높은 지능형 소프트웨어 PLC 시스템인 ISPLC(Intelligent Agent System based Software Programmable Logic Controller)을 개발하고자 한다.

본 논문에서 개발한 시스템은 하드웨어적이 아닌 소프트웨어적으로 PLC 시스템을 활용할 수 있게 함으로써 PLC는 물론 C언어와 같은 고급언어에 익숙한 사용자들에게 모두 활용될 수 있으며 프로그래밍상의 문법오류는 물론 논리오류를 체크함으로써 초보자나 숙련된 사용자 모두에게 활용가치가 높다. LD를 고급언어인 C 코드로 변환함으로써 LD에 익숙한 프로그래머나 고급언어인 C언어에 익숙한 프로그래머 모두 프로그래밍이 가능한 장점을 갖는다. 따라서 표준 C 컴파일러에서 제공하는 프로그래밍 기능을 모두 활용할 수 있다는 잇점을 갖는다. 이러한 연구는 국내는 물론 국외에서도 최초로 시도되는 연구이다.

3.2 개발 시스템의 구성

본 논문에서 개발한 ISPLC 시스템은 교통 신호 시스템, 엘리베이터 시스템, 공장 자동화 시스템 등과 같은 실세계에서 적용되는 산업용 기술을 바탕으로 제어 및 프로그램이 가능 할 수 있도록 하는데 목적이 있다. ISPLC의 전체적인 시스템 구조는 다음 그림 2와 같다.

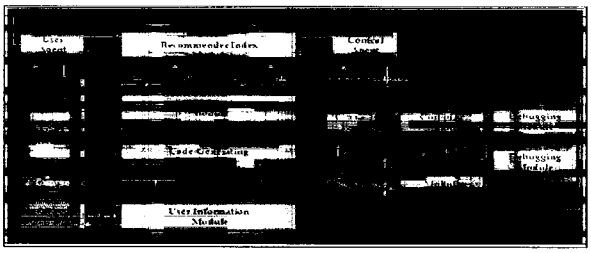


그림 2. ISPLC 전체 시스템 구조
Figure 2. The overview of ISPLC

ISPLC에서 사용된 7개의 주요 컴포넌트 모듈의 역할 및 특징은 다음과 같다. 사용자 추천 인덱스 모듈(User Recommendation Index Module)은 사용자의 프로그램 설계 과정을 패턴화하여 개인화 코드로 컴포넌트 또는 클래스 파일 형태로 저장한다. 사용자가 사용한 데이터 또는 사용할 데이터를 사용자의 프로그램 성향에 따라 데이터 히스토리를 인덱싱하는 역할을 담당한다. 이 모듈은 사용자 인터페이스 상에서 프로젝트나 시스템 개발시 이전의 프로그램 설계에 대한 컴포넌트 및 클래스 패턴 히스토리를 가지고 사용자에게 제안해주는 모듈이다. 코드 매핑모듈(Code Mapping Module)은 LD, IL, C 세 언어중 한 언어를 선택하여 프로그램을 하더라도 자동적으로 각각의 세 타입의 언어에 대한 상호 호환성을 위해 코드를 생성하며 매핑시키는 모듈이다. 코드 생성 모듈(Code Generating Module)은 LD언어로 프로그래밍시 사용자의 인터페이스내에 각각의 LD 모듈에 대한 프로그램이 가능할 수 있도록 C 코드로 변환하여 생성해준다. 또한 사용자가 LD 언어를 사용하지 못하더라도, C 언어로 변환되므로 구현 및 프로그램이 가능하다. 사용자 인포메이션 모듈(User Information Module)은 사용자의 히스토리 데이터, 즉 프로그램 가능한 컴포넌트나 클래스 데이터를 최적화하여 재사용성을 제공하기 위한 모듈이다. 컴파일러 모듈(Compiler Module)은 시스템의 툴 내에서 제공하는 인터프리터로서 파싱 처리가 가능하다. 필터링 모듈(Filtering Module)은 시스템내에서의 불필요한 코드를 최적화 시켜주는 기능을 제공한다. 디버깅 모듈(Debugging Module)은 시스템내에서의 문법이나 논리오류 발생을 방지하기 위한 역할을 한다.

이 시스템은 사용자가 인터페이스를 통해 사용자 에이전트에게 LD 질의를 요청하면 사용자 에이전트와 컨트롤 에이전트는 질의를 갖고 통신을 한다. 이때, 컨트롤 에이전트는 요청받은 질의를 컴포넌트 데이터 스토리지에서 검색하여 질의에 맞는 컴포넌트를 사용자 에이전트에게 전달한다. 질의에 따라 생성된 컴포넌트를 갖고 사용자 에이전트는 인터페이스를 통해 사용자에게 프로그래밍이 가능할 수 있도록 생성해 주며, 컴포넌트 히스토리를 사용자 추천 인덱스 라이브러리에 저장함으로써 재사용이 가능할 수 있도록 한다.

다음 그림 3은 ISPLC 전체 구조에 포함된 주요 에이전트들의 관계를 나타낸 그림이다.

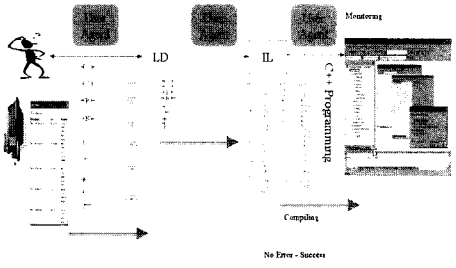


그림 3. ISPLC에서 에이전트들의 관계
Figure 3. The interrelationship among agents in ISPLC

ISPLC에서 주요 에이전트간 플로우 다이어그램을 설명하면, 먼저 사용자는 LD로 에디터 상에서 프로그래밍 한다. LD로 프로그래밍 된 시스템은 자동적으로 IL로 변환되며, 그 변환 모듈은 일대일 매핑과정을 통해 이루어진다. IL로 변환된 모듈은 Visual C++ 형태로 조건 제어의 스텝 컴포넌트 별로 구성된다. 이로써 사용자 에이전트는 사용자가 프로그래밍한 LD 모듈을 바로 Visual C++ 코드로 생성함으로써,

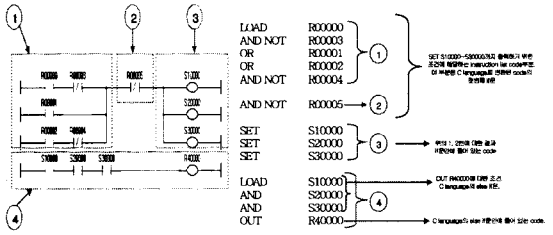
신속한 에러 검출과 각각의 LD 모듈을 Visual C++ 코드의 스텝별 컴포넌트로 구성함으로써 보다 쉽고 편리하게 프로그래밍이 가능하다. LD를 IL로 매핑하는 부분은 Visual C++ 코드상에서 바로 변환될 수 있도록 API(application programmable interface)를 생성한다. ISPLC에서는 사용자 에이전트와 컨트롤 에이전트는 LD->IL->C 또는 Visual C++ 코드로의 변환과정을 자동적으로 수행한다. 에이전트에 의한 단계별 수행 알고리즘 패턴은 다음과 같다.

[1단계] LD 작성
- Step 1: 임의의 LD로 작성
- Step 2: 파라메터 값을 갖는 초기값으로 표준화 수행
- Step 3: LD 루프수행
- Step 4: 새로운 프로토타입과 라인을 이용하여 프로그래밍
- Step 5: 시뮬레이션

[2단계] IL 변환
- Step 1: LD를 기본으로 파라메터값을 갖고 초기 함수 수행
- Step 2: 코드 매핑수행
- Step 3: 함수에 메소드 추가
- Step 4: 컴파일링 수행

[3단계] C 또는 Visual C++ 변환
- Step 1: 단계 1과 단계 2를 이용한 초기화
- Step 2: 초기 코드로 표준화 수행
- Step 3: 메소드와 파라메터 값을 매핑
- Step 4: 해당 클래스와 객체 생성
- Step 5: 컴포넌트의 필터링 및 재생성
- Step 6: 컴파일링 수행(최종단계)

본 논문에서 제안한 LD에서 IL로, IL에서 C로의 코드변환 알고리즘을 단계별로 설명하면 다음과 같다.



(a) 초기상태 (b) IL로 변환된 코드

그림 4. LD에서 IL로의 코드변환 관계
Figure 4. The code conversion of LD to IL

코드 변환을 설명하기 위해 각각의 접점 및 코일 램프들을 위 그림의 번호에 따라 ① R00000, R00001, R00002, R00003, R00004: external switch, ② R00005 : external switch. ③ S10000, S20000, S30000 : internal coil. ④ R40000 : external lamp. 라고 하였다. 이때 R00000 ~ R00002 중 한 가지만 ON이 되면(1이면), 내부 코일S10000 ~ S30000는 모두 ON이 되어, R40000 lamp에 불이 들어오게 한다(1이 된다). R00005가 ON 된다면, 모든 회로는 동작하지 않는다. R00003이 ON되어 있을 때는 R00000을 눌러도 회로는 동작하지 않는다. 마찬가지로, R00004가 ON 되어 있으면 R00002를 아무리 눌러도 회로는 작동하지 않는다.

본 논문에서는 LD에서 IL로 변환할 때 노드와 입출력 관계만을 고려하여 변환하는 알고리즘을 사용한다. 위의 예에서의 IL을 보면, LOAD R00000, AND NOT R00003가 끝난 다음 바로 R00005을 AND NOT하지 않고, 노드를 고려하여 R00001을 먼저 OR해 준다. OR를 모두 고려해 준 뒤 R00005을 AND NOT 한다. 멀티출력형식으로 구성되어 있는 경우는 거의 대부분 내부 코일을 이용하여 타이머나 카운터를 동작시키기 위해 사용한다. 이와 같이 IL 프로그램에서의 적절한 규칙을 찾아내서 규칙베이스에 저장하고 새로 변환되거나 입력된 IL 코드를 이러한 규칙기반 시스템에서 찾아내어서 표준 C 코드로 변환하여 사용자 에이전트가 사용자 추천 모듈에서 추천하게 된다.

LD를 IL로 변환 할 때 대응하는 부분을 그림 4(b) 빨간색 동그라미의 숫자로 구분해 보면, 1번과 2번은 SET S10000~SET S30000을 출력하기 위한 조건이다. 3번은 SET출력이다. 4번은 2번째 AND까지가 조건문이고, OUT은 출력이다. 즉, "IF A(조건부) THEN B(결론부)" 형태의 규칙을 찾아내기 위해"LOAD ~ AND 또는 AND NOT" 를 규칙의 조건부로 표현하고 "SET" 이하나 "OUT"이하를 규칙의 결론부 형태로 IL 패턴 규칙베이스를 구성한다.

본 논문에서 구현한 LD에서 IL로의 코드변환을 위한 세부 알고리즘 스텝은 다음 그림 5와 같다.

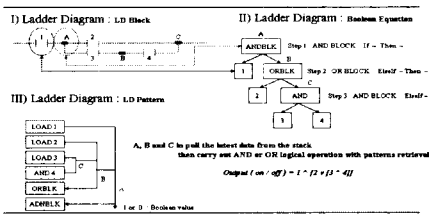
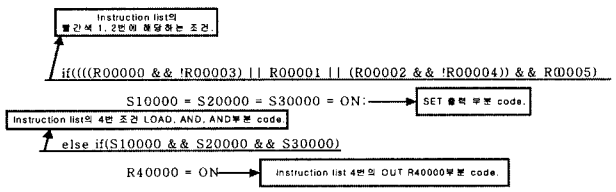


그림 5. LD에서 IL로의 세부 코드변환 스텝
Figure 5. The code conversion steps of LD to IL

위 알고리즘 스텝 I)에서 LD를 컴포넌트 별로 분류하여 II)에서와 같이 부울함수에 의해 분류한다. 이때 AND, OR 연산에 의해 IF ~ ElSE IF 문 등과 같은 형태로 패턴을 분류하고, III)에서와 같이 LD 패턴을 찾아서 하나의 패턴으로 규칙베이스에 저장한다. 위의 IL은 다음과 같이 규칙기반 시스템으로부터 추론된 IF~THEN 형태의 C 코드로 변환된다.



IL에서 C 코드로 변환할 때는 IL에서 어떠한 조건에 의해 결과가 출력되는지 출력에 대한 명령어가 나타나는 순서로 IL이 구성되어 있으므로 이러한 관계를 패턴분류하여 규칙베이스를 구성한다.

ISPLC의 중요한 두개의 에이전트는 다음과 같다.

(1) 사용자 에이전트(User Agent : UA)

UA는 사용자와 컨트롤 에이전트와의 상호 작용을 위한 인터페이스 역할을 하는 에이전트이다. 구체적으로는 ISPLC의 동작을 위하여 사용자로부터 요구를 획득하고 LC(LD component) 및 ALCL(automatic LD library)제공을 담당한다. 그리고 인터페이스를 통해 컴포넌트 데이터 스토리지의 데이터베이스를 관리할 수 있도록 한다. 사용자는 인터페이

스를 통해 사용하고자 하는 LC를 사용자 에이전트에게 요청하면 사용자 에이전트는 사용자가 원하는 정보를 받아 에러를 검출하여 사용자 에이전트는 사용자를 대신하여 최적의 LC를 제공하며, 다시 사용자에게 피드백 한다. 또한 사용자 에이전트는 사용자 추천 인덱스 메카니즘에 LC 정보를 자동으로 피이드백 한다.

사용자 에이전트는 사용자가 초기 LD를 시작으로 한 모듈이 생성되면 IL로 생성된 하나의 모듈을 Visual C++ 스텝 1로 구성한다. 각 LD 모듈에서의 다중 제어는 Switch_Case 구문으로 조건 제어에 적합하도록 모듈을 자동 변환할 수 있도록 사용자에게 알려 준다. 사용자는 사용자 에이전트의 메시지에 따라 LD 모듈을 재구성하며, pseudo code를 이용한 컴파일링을 통해 에러를 검출한다. 에러가 발생하지 않을 경우 다음 스텝으로 LD 프로그래밍을 할 수 있도록 지시한다. 만약 LD 모듈로 초기화만 생성할 경우 스텝별 조건 제어에 맞게 구성할 수 있도록 사용자에게 사용자 인덱스 라이브러리에 저장된 모듈 히스토리를 제공해준다. 단, 사용자 인덱스 라이브러리 히스토리는 사용자가 초기 생성 시 사용하기 어려운 단점이 있지만, 한번 프로그래밍 수행이후 자동 저장되며, 각각의 모듈을 유니트 또는 컴포넌트와 클래스 단위로 저장하여 히스토리 데이터를 만든다.

사용자 에이전트의 알고리즘과 주요 코드는 다음과 같다. 즉, 사용자가 인터페이스 LD 컴포넌트 모듈을 그리면, LD의 이름과 LD의 위치정보, 시간 등의 정보를 이용해 VC++ 코드를 생성한다. LD_N, LD_P는 각각 LD의 이름과 파라메터 값을 의미하는 변수이다.

```
U A (LD [Name], LD [Parameters value], Time)
{
   Start(0 0 0);
   While (Still not 0 and 0 0)
            Initializing LD Component Start
              If LD Code_P(value) = 01
              {
                  Seq_P = Para_val
                  // Initialized
              }
            Then UA(LD_N, LD_P)Next Loop;
}
Return Seq_P();
```

1) Code Generating을 위한 Dialogue Data Message생성 // LD->C로 변환 부분

```
class UAAboutDlg : public UAAboutDlg {
public :  UAAboutDlg();
Dialog Data
 {{AFX_DATA(CAAboutDlg)
   enum { IDD = IDD_ABOUTBOX };
  }}AFX_DATA
   ClassWizard generated virtual function overrides
  {{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);
   }}AFX_VIRTUAL
Implementation
}
```

2) void UAAboutDlg::DoDataExchange(CDataExchange* pDX) // C코드로 변환 수행

```
{
 CDialog::DoDataExchange(pDX);
   {{AFX_DATA_MAP(UAAboutDlg)
    }}AFX_DATA_MAP  }
 BEGIN_MESSAGE_MAP(UAAboutDlg, CDialog)
   {{AFX_MSG_MAP(UAAboutDlg)// message handlers
   }}AFX_MSG_MAP
 END_MESSAGE_MAP()
```

3) Error Detect // LD에 대한 논리 오류 검출기능

```
 private void exec_UA_SCode(object detect, System EventUA ua)
 private void exec_menu_Click(object sender, System.EventArgs e)
{errBox.Clear() ;
   if Module_Block[9,0] == 5 || Module_Block[0,1]                -- 6
|| Module_Block[0,2] == 9 )
{errBox.AppendText("Detecting Error!\r\n") ;
 errBox.AppendText("Generating Agent Code....\r\n") ;
  debugCodeWrite() ;}
else
{ errBox.AppendText("No Error and Warnning") ;          }}
```

(2) 컨트롤 에이전트(Control Agent : CA)

사용자가 이용할 수 있는 응용 부분에서 UA와의 통신을 위한 에이전트이다. 사용자의 요구로부터 사용자가 원하는 LD 및 ALCL(Automatic LC Library)에서 사용자가 원하는 기능을 제공하고 이를 위하여 UA와 컴포넌트 데이터 스토리지와의 통신을 통해 개발 환경에 알맞는 컴포넌트 및 컴파일러를 제공해주는 역할을 한다.

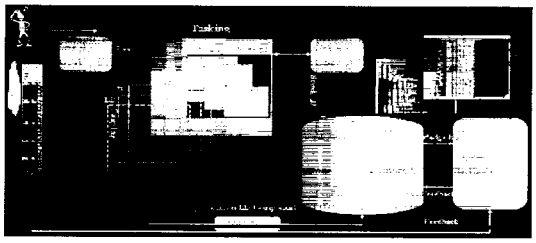


그림 6. 컨트롤 에이전트의 데이터 플로우 다이어그램
Figure 6. The dataflow diagram of control agent

컴포넌트 데이터 스토리지(Component Data Storage: CDS)는 ISPLC에서 모든 데이터를 관리하고 저장하는 역할을 담당하는 데이터베이스이다. 이는 UA와 CA에게 사용자가 원하는 데이터 즉, 컴포넌트들을 제공하며, 사용자 인포메이션에 지식 기반 히스토리를 제공한다. 또한 UA에 필요한 컴포넌트를 URI를 통해 사용되어 질 수 있도록 한다. CA에게서 필터링 및 최적화된 컴포넌트를 UA의 요구에 맞게 구성하는 역할을 한다. 사용자 인포메이션(User Information)은 UA , CA 그리고 CDS의 모든 필터링 및 최적화된 컴포넌트를 사용자에게 필요로 하는 정보 히스토리를 담당한다. 이는 사용자 인터페이스를 제공하도록 하는 기능이 있으며, CDS에게 데이터를 피드백한다.

ISPLC의 전체적인 지능적 에디팅 과정은 다음 그림 7과 같다. ISPLC는 IEC1131-3의 표준 언어가 중간코드형태인 IL로 변환되어 C언어에서 컴파일되어 윈도우 환경에서 실행 가능하도록 구성한다. 에디터는 GUI기반으로 되어 있으며 에이전트들에 의해 지능적으로 오류 수정 및 코드 변환이 수행되는 점이 특징이다.

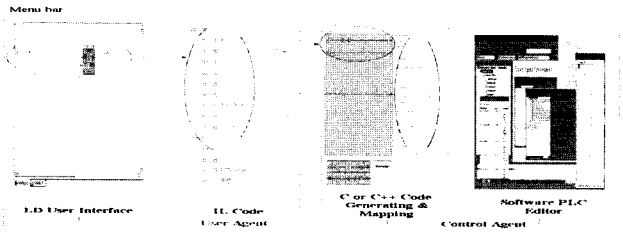


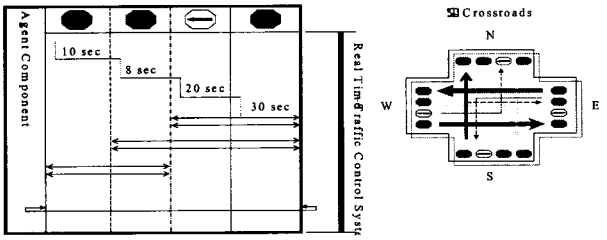
그림 7. ISPLC의 지능적 에디팅 과정 흐름도
Figure 7. The intelligent editing flow of ISPLC

4. 실시간 교차로 신호등 제어 시스템

4.1 시스템 모델링

본 논문에서는 ISPLC를, 가장 많이 사용하는 PLC 적용 시스템인 실시간 교차로 신호등 제어 시스템(Real Time-Traffic Control System: RT-TC System)에 적용하여 시뮬레이션 하고자 한다. 이는 실세계의 도로상태를 소프트웨어 기반의 환경으로 옮겨 차량 운행에 대한 교통 신호를 제어하기 위한 것이다. 본 논문에서는 RT-TC의 하드웨어와 소프트웨어 환경을 모두 컨트롤하기 위해 ISPLC를 적용하여 에이전트의 기능을 추가함으로써 지능형 시스템을 생성하였다. 사용자는 로드맵 상에서의 신호등 제어에 관한 LD로 프로그래밍시 사용자 에이전트는 LD상의 하나의 모듈을 IL로 매핑시켜 각각의 모듈을 Visual C++로 자동 변환시킨다.

RT-TC 시스템의 모델링 구성도는 그림 8과 같다. 그림 8(a)의 제일 왼쪽이 빨간색(적), 다음이 주황색(황), 다음이 좌회전 화살표(좌), 다음이 초록색(청) 등을 나타내고 있다. 각 타이머의 설정시간은 직진신호30초, 좌회전 신호 20초, 경고신호 8초로 하였다. 그림 8(b)에서 각 방향별 출력번호는 W(West)의 청, 좌, 황, 적 순으로 R3.0, R3.1, R3.2, R3.3이며, N(North)의 청, 좌, 황 적 순으로 R3.4, R3.5, R3.6, R3.7이며, S(South)의 청, 좌, 황, 적 순으로 R3.8, R3.9, R3.10, R3.11이며, E(East)의 청, 좌, 황, 적 순으로 R3.12, R3.13, R3.14, R3.15를 나타낸다. 그림 8(b)는 RT-TC 신호등의 PLC 출력 접점을 표시한 것이다.



(a) RT-TC의 타이머 설정시간 (b) RT-TC 신호등 체계
(a) RT-TC timer setting (b) RT-TC traffic states
그림 8. RT-TC 시스템 모델링
Figure 8. The system modeling of RT-TC system

RT-TC 시스템의 신호등 체계의 동작은 다음 8가지로 나뉠 수 있다. ① 동쪽에서 남쪽으로, 서쪽에서 북쪽으로 각각 좌회전하도록 좌회전 신호등을 점등한다. ② 동쪽과 서쪽 신호등은 모두 황색등을 점등한다. ③ 동쪽<->서쪽 직진 신호를 점등한다. ④ 남쪽에서 서쪽으로, 북쪽에서 동쪽으로 각각 좌회전하도록 좌회전 신호등을 점등한다. ⑤ 남쪽과 북쪽 신호등은 모두 황색등을 점등한다. ⑥ 남쪽<->북쪽 직진 신호를 점등한다. ⑦ 남쪽과 북쪽 신호등은 모두 황색등을 점등한다. ⑧ 수신호를 위한 황색등 점멸 회로를 작동시킨다.

PLC와 관련하여 에이전트 기반의 컴포넌트를 생성하여 크로스로드 맵에 따른 최종의 지능형 시스템의 모델을 구현한다. 아래의 표 1은 위의 8가지 동작을 각각의 교차로 로드맵에 따른 시그널 루틴을 나타낸 것이다. 입력값은 R0.0은 신호등 시작, R0.1은 신호등 정지, R0.3은 타이머 정지, R0.7은 수신호용 황색등 시작을 각각 나타낸다. TC다음의 숫자는 교통제어(Traffic control) 타입을 말한다. 예로 TC1은 좌회전이며, 좌회전시 20초 타이머가 세팅(ON)되어있고 이때 TC2는 타이머가 오프되어 있어야 함을 나타낸다.

표 1. 교차로 로드 맵 시그날 서브루틴 기능
Table 1. The signal functions of cross road map

서브루틴 번호	출발지 기준 수행내용 (Processing)	출력 번호 (output number)	타이머 On (Timer On)	타이머 Off (Timer Off)
1	E ----- W 직진	R3.0, R3.12	TC3	TC4
2	E ----- W 좌회전	R3.1, R3.13	TC1	TC2
3	E ----- W 경고 (황색등)	R3.2, R3.14	TC2, TC4	TC3, TC5
4	E ----- W 정지 (적색등)	R3.3, R3.15	TC5, TC1	TC8, TC2
5	S ----- N 직진	R3.4, R3.8	TC7	TC8
6	S ----- N 좌회전	R3.5, R3.9	TC5	TC6
7	S ----- N 경고 (황색등)	R3.6, R3.10	TC6, TC8	TC7, TC1
8	S ----- N 정지 (적색등)	R3.7, R3.11	TC1, TC5	TC5, TC6

(E: East, W: West, S: South, N: North)

4.2 시스템의 수행 알고리즘

본 논문에서 구현한 RT-TC의 수행 알고리즘의 그래픽적인 순서와 신호등 알고리즘은 다음 그림 9와 같다. Signal Light(신호등)는 로드 상에서의 신호등 타임과 자동차의 움직임 제어를 위한 클래스 함수로 구성한다. 이러한 함수들은 각각의 스케쥴러(Scheduler), 자동차(Car), 타임 밸런스(Time), 보행자 신호등(Pedestrian Light), 보행자 버튼(Button) 그리고 로드 상태(Road)에 따라 함수의 메소드들이 수행하며, 각 스텝에 따라 반복된다. 먼저 클럭은 로드상에서의 신호 제어를 위한 타임을 수행한다. 그 다음으로 스케쥴러에 따라 자동차의 이동 제어를 하며, 보행자 신호와 교차로 신호에 따라 순차적으로 수행한다. 논문에서 제안한 사용자 에이전트는 신호등 알고리즘을 조절한다.

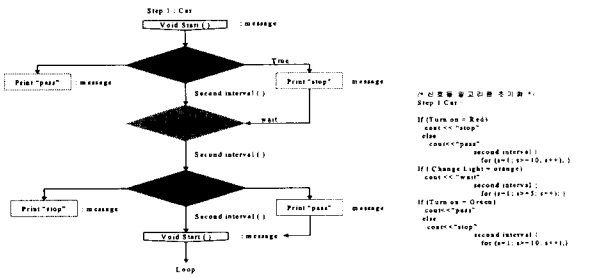


그림 9. RT-TC 시스템의 수행 알고리즘
Figure 9. The execution algorithm of RT-TC system

아래 그림 10은 RT-TC 시스템의 시뮬레이션을 위해 시뮬레이션 루프를 모델링 한 시퀀셜 다이어그램이다.

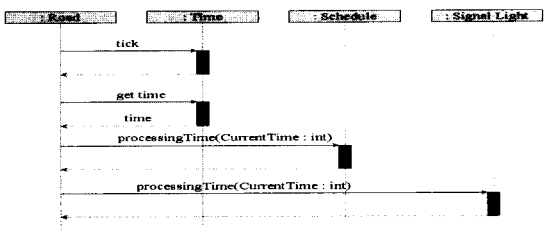


그림 10. 시뮬레이션 루프 모델링 시퀀셜 다이어그램
Figure 10. The sequential diagram of simulation loop

시퀀셜 다이어그램에서 두 객체들 사이의 메시지는 실선 형태를 이용해서 한 객체에서 받는 객체로의 방향을 갖는다. 메시지는 받는 객체에서 해당 연산을 수행하며, 이때 매개변수들을 포함한다. 이들의 결과는 Time으로 반환하며, 한 방향의 시퀀셜 다이어그램의 수행이 완료되면 다른 객체의 모델링 시뮬레이션이 생성되어 루프를 돌게 된다. 여기서 Road는 로드상태를, Time은 타이머 설정시간을, Scheduler는 신호등에 따른 자동차 이동 제어를, Signal Light는 4가지 형태의 신호등을 나타내는 기호이다.

자동차 신호와 보행자 신호는 초기 타임을 가지고 보행자 신호의 버튼이 누름과 동시에 신호의 입력을 받아 신호가 바뀌며, 로드상에서의 자동차는 이동한다. 이때 보행자 신호는 로드의 상태에 따라 일정한 타임이 지나면 보행자의 신호가 변하며, 그 신호에 따라 로드 상의 신호 또한 대기함으로써 하나의 로드상의 교통 제어 플로우가 형성된다. 이를 신호등의 타임 밸런스에 따라 시그널 상태 변이도는 다음 그림 11과 같다. 시그널 상태 변이에 따라 보행자와 자동차 신호등에 따른 어드레스 번지를 지정하여 처리하였다.

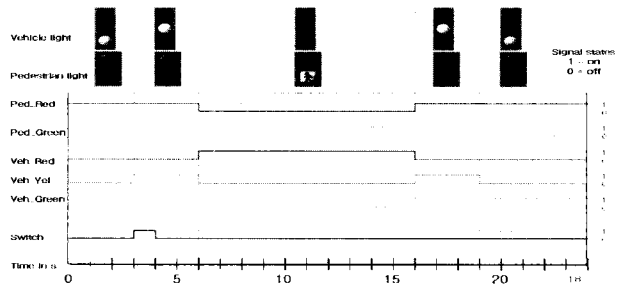


그림 11. 신호등의 타임 밸런스에 따른 각각의 시그널 상태 변이도

Figure 11. The signal state diagram according to time balance of signal lights

각 스텝별로 지정된 주소에 따라 신호등 LD 알고리즘 요소를 10개의 네트워크로 표현할 수 있으며 이것은 하나의 로드 사이클을 형성한다.

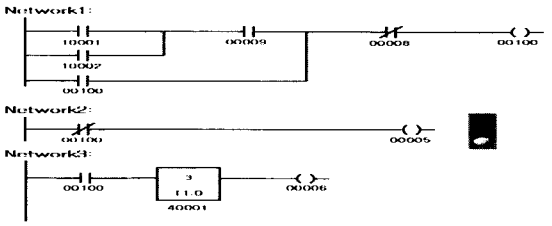


그림 12. RT-TC 시스템의 네트워크 구조

Figure 12. The network structures of RT-TC system

네트워크 1은 보행자 신호등의 빨간색 또는 초록색 신호에 따라 보행자 신호등이 선택적으로 수행하게 되며 초록색 신호가 수행될 시 시그널 스위치는 메모리 비트에 저장되며, 자동차 신호는 대기하게 된다. 네트워크 2는 보행자 신호의 타임이 종료되면, 자동차 신호로 바뀌며 신호등의 색이 로드상에 있는 자동차가 움직일 수 있도록 수행된다. 네트워크 3은 다시 메모리 비트에 저장되며, 펑션 블록에 따른 자동차 신호가 바뀌게 된다. 또한 본 논문에서 구현한 나머지 네트워크를 설명하면, 네트워크 4는 자동차 신호가 로드상의 자동차를 대기 상태의 신호로 변화하고, 네트워크 5는 자동차의 신호가 로드상의 자동차가 정지할 수 있도록 신호 변화를 수행하며, 네트워크 6은 펑션 블럭에 의해 보행자 신호의 체계가 변화된다. 네트워크 7은 보행자 신호가 변화되는 동안 자동차는 정지 상태를 유지하며, 네트워크 8은 메모리 비트에 저장되며, 보행자의 신호와 자동차 신호가 변화 가능 하도록 펑션 블록에서 신호의 체계를 제어하고, 네트워크 9는 메모리 비트의 여부와 자동차와 보행자 신호의 여부에 따라 다시 초기의 보행자 신호 상태로 전이하도록 수행한다. 초기의 상태로 자동차 신호는 다시 로드상의 차들을 이동 수행하며, 펑션 블럭에 의해 보행자 신호가 다음에 수행될 수 있도록 신호 상태를 전이한다.

4.3 구현 및 시뮬레이션

(1) ISPLC를 적용한 RT-TC 구현 및 시뮬레이션

에이전트 기반의 ISPLC 시스템을 실제 RT-TC 시스템에 적용하기 위해 구현된 예를 보면 다음 그림 13과 같다. 이 시스템은 LD에서 IL로, IL에서 C로 자동 변환시킴으로써 IEC1131-3의 표준언어인 LD에 익숙하거나 IL 또는 고급언어인 C 언어에 익숙한 사용자들에게 프로그래밍시 좋은 프로그래밍 전략을 안내한다. 이렇게 함으로써 프로그래밍시 발생할 수 있는 문법 오류는 물론 논리 오류를 최소화 시킬 수 있다. ISPLC는 LD를 가지고 Visual C++ 언어와 비쥬얼한 환경에서의 에디팅이 가능하도록 하였다. 각각의 요소들은 LD 모듈에 맞게 그래픽 라인 모듈을 생성했으며, 이 모듈을 파라미터 값으로 변환하여, 비트맵 이미지로 생성시킨 후 사용자가 에디팅을 수행할 수 있도록 하였다. 또한 LD 에디터에서도 프로그램에 대한 에러 체크를 위해 에이전트를 시스템 모듈내에서 자동적으로 컨트롤 가능하도록 하였다.

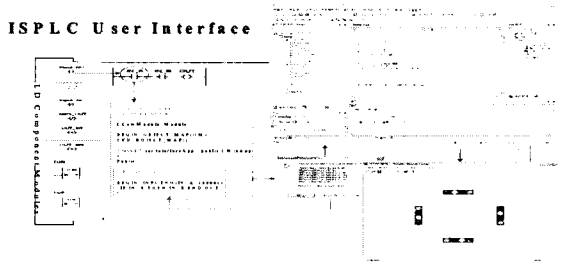


그림 13. ISPLC의 RT-TC를 위한 사용자 인터페이스

Figure 13. The user interface for RT-TC implementation of ISPLC

그림 13의 ISPLC 사용자 인터페이스에서 왼쪽 모듈은 LD 사용자 인터페이스이며, 안쪽 아래 모듈은 LD Component이다. LD 명령어 삭제기능도 있으며 메시지 박스 모듈은 LD Compiling, 사용자 에이전트 코드 뷰에 의한 오류 검출기능이 있다. 오른쪽 모듈은 LD 코드를 Visual C++ 코드로 변환시키는 모듈이며 오른쪽 아래 모듈은 Visual C++ 코드 형태로 에이전트가 추천해주는 코드를 나타내는 모듈을 나타낸다.

일반적으로 시스템 구현을 위해 사용자의 요구분석, 시스템 설계, 시스템 구현 등의 과정에서 사용되는 모델링 언어가 UML(unified modeling language) 이다[16]. ISPLC의 RT-TC 시스템을 구현하기 위해 UML다이어그램으로 나타내어 처리하였는데 다음 그림 14와 같다.

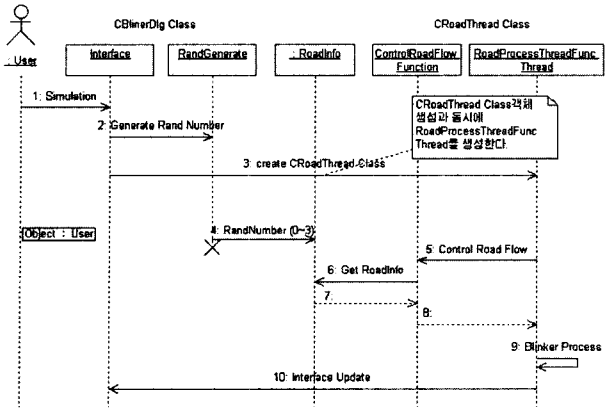


그림 14. RT-TC 시스템을 위한 UML 다이어그램
Figure 14. The UML diagram for RT-TC system

(2) 시뮬레이션 결과

ISPLC를 이용한 에이전트 기반의 RT-TC 시스템의 시뮬레이션 화면을 나타내면 다음 그림 15와 같다. 이 시스템은 각각의 사용자 에이전트와 컨트롤 에이전트가 실제 PLC로 구현한 코드를 자동 변환하여 비쥬얼한 환경내에 수행가능할 수 있도록 구현된 점이 특징이다.

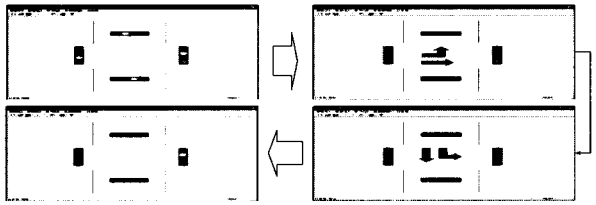


그림 15. 에이전트 기반 RT-TC 시뮬레이션 결과
Figure 15. The simulation results of RT-TC system based on agents

본 논문에서 구현한 시스템의 LC에서 IL, IL에서 C 코드로의 변환과정을 보여주는 인터페이스는 다음 그림 16과 같다. 그림에서 왼쪽이 LD의 사용자 인터페이스(UI)이며 아래쪽 버튼들은 선택할 수 있는 LD 컴포넌트들을 나타내며, LD 컴파일과정에서 오류발생시 검출해주는 기능이 같이 나타나 있다. 또한 지금까지 그린 LD를 삭제하는 기능도 있으며, 화면의 오른쪽은 LD 코드를 최종적으로 실시간으로 C로 변환하여 보여주는 화면을 나타내고 있다. 화면의 오른쪽 아래는

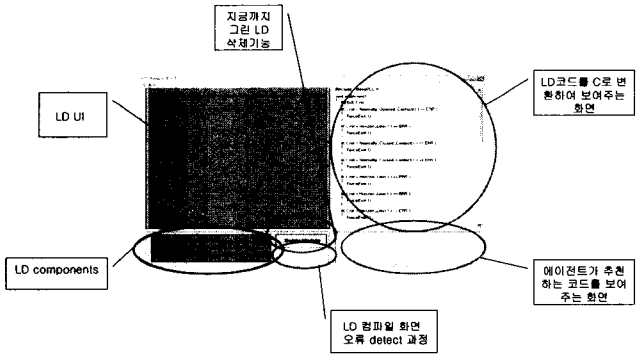


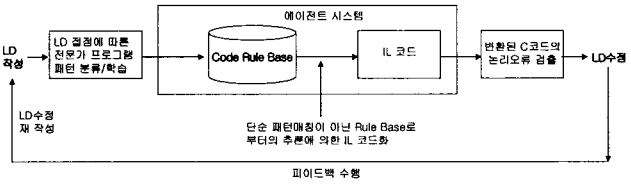
그림 16. LD, IL, C 코드 변환 인터페이스
Figure 16. The interface for LD, IL and C code conversion

에이전트가 라이브러리로부터 추천하는 코드를 보여주는 화면으로 사용자는 이 코드를 이용하여 LD 또는 C 코드를 마음대로 수정할 수 있다.

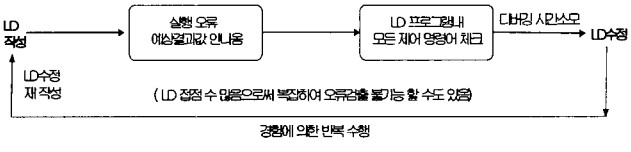
LD 컴포넌트를 이용해 프로그래밍 하면 실시간으로 오류검색을 하여 오류 발견시 에이전트에 의해 자동적으로 알맞은 C 코드를 추천한다. 이렇게 추천된 코드로부터 사용자는 Visual C++ 컴파일러 상에서 수정이 가능하며 이것은 실시간으로 LD로 변환 가능하므로 프로그램의 논리오류 수정이 LD상에서 하는 것 보다 훨씬 효율적이고 프로그래밍 시간이 단축된다.

(3) 결과 비교 분석

ISPLC와 기존 소프트웨어 PLC의 오류처리 알고리즘을 설명하면 ISPLC에서는 LD 접점에 따른 전문가 작성 프로그램의 패턴을 분류하여 사용자 에이전트가 규칙베이스를 구축하여 IL 코드로 변환한다. 사용자 에이전트는 사용자 데이터를 관리하고 사용자의 사용패턴을 학습한다. 컴포넌트 라이브러리 등을 제공하고 사용자에게 올바른 C코드를 추천해 준다. 이것을 위해 IL에서C 로의 규칙적인 학습패턴을 유지하여 규칙베이스에 저장함으로써 가능하게 된다. 따라서 ISPLC는 사용자가 많이 사용하면 할수록 C코드로의 변환이 더 효율적으로 된다. 컨트롤 에이전트는 실제로 변환된 C 코드로부터 컴파일을 하고 오류나 패턴을 필터링하여 사용자에게 피이드백 함으로써 LD를 재작성 할 수 있도록 상호작용한다.



(a) ISPLC에서의 논리오류 디버깅 과정
(a) The debugging process of logic errors in ISPLC



(b) 기존 소프트웨어 PLC에서의 논리오류 디버깅 과정
(b) The debugging process of logic errors in conventional software PLC

그림 17. ISPLC와 기존 소프트웨어 PLC의 디버깅 과정
Figure 17. The comparison of debugging process between ISPLC and conventional software PLC

ISPLC와 기존 소프트웨어 PLC의 논리오류 디버깅에 소요되는 시간을 측정하기 위해 RT-TC에서 LD 명령어 패턴수를 증가시키면서 실험을 해보았는데 결과, ISPLC는 논리오류 만큼의 프로그램내의 특정 부분의 제어관련 명령어들을 확인하고 메시지 흐름의 순서에 따라 원인을 순서대로 규명해야 되므로 프로그램이 아무리 복잡해지고 패턴수가 증가하더라도 평균적으로 $\sqrt{n}$(n: LD명령어패턴의갯수) 정도의 디버깅 스텝수를 갖는 경향이 있음을 알 수 있었다. 그러나 기존 소프트웨어 PLC는 LD 프로그램 작성 후 실행이 안되었을 경우 디버깅하기 위해서는 프로그램내의 모든 제어관련 컴포넌트들을 확인하고 제어흐름의 순서에 따라 모든 명령어들의

상관관계를 분석하면서 오류 원인을 차례로 규명해야 하므로 $nPn = n!$의 디버깅 스텝수를 가짐을 알 수 있었다. 따라서 LD 접점수가 많으면 많을수록 오류검출 가능성은 줄어들게 된다. 이것을 그래프로 나타내면 지수함수 비율로 디버깅 스텝수가 증가함을 알 수 있었다(그림 18). 따라서 ISPLC를 사용할 경우 기존 소프트웨어 PLC보다 논리오류를 현저하게 줄여줌을 알 수 있었다.

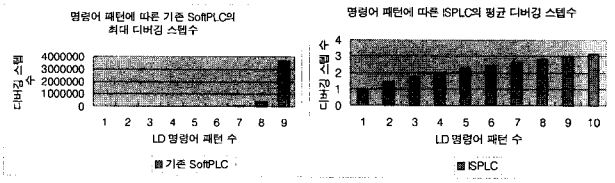


그림 18. ISPLC와 기존 소프트웨어 PLC의 디버깅 스텝수 비교

Figure 18. The comparison of debugging steps between ISPLC and conventional software PLC

5. 결 론

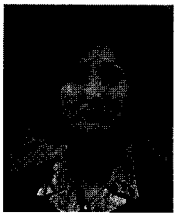
본 논문에서는 실시간 지능형 소프트웨어 PLC인 ISPLC (Intelligent Agent System based Software Programmable Logic Controller)를 구현하였다. ISPLC는 5개의 표준 언어들 중 90%이상을 점유하고 있는 LD를, 중간코드이며 5개 표준 언어중의 하나인 IL로 변환하고 이를 다시 고급 언어인 표준 C 또는 Visual C++ 코드로 실시간으로 변환시키는 통합된 인터페이스 환경을 제공함으로써 표준 언어 상호간 호환성은 물론 LD 에 익숙한 사용자나 고급언어인 C언어에 익숙한 사용자들에게 편리함을 도모하고자 하였다. 따라서 표준 C 컴파일러에서 제공하는 기능상의 장점을 활용할 수 있을 뿐 아니라 웹 환경에 적용할 수 있다. 이러한 연구는 국내외적으로 개발된 소프트웨어가 없었으므로 처음으로 시도된 연구이다. 본 논문에서는 ISPLC를 지능형 시스템에서의 LD, IL, C 또는 Visual C++ 매핑과 더불어 에이전트를 기반으로 한 실시간 교통 신호 제어 시스템(Real Time Traffic Control System)에 적용하여 구현하였다. RT-TC 시스템은 실시간으로 사용자가 제어하기 편리한 인터페이스 환경을 제공한다. 이것을 위해 클래스 라이브러리와 에이전트 변환 모듈을 구현하였다. ISPLC에서는 LD를 C코드로 변환시킴으로써 표준 C컴파일러 상에서 지능적 에이전트에 의한 논리오류를 체크할 수 있는 장점이 있다. 이것은 LD상에서 논리오류를 찾아내는 것 보다는 C 컴파일러상에서 에이전트에 의한 논리오류의 수정기능을 갖는 것이 훨씬 효율적이기 때문이다.

앞으로 ISPLC를 보다 지능적인 시스템이 되게 하기 위해서는 숙련된 개발 툴과 알고리즘을 적용하여 보다 효율적이고 구조적인 환경과 사용자의 수행 능력, 전문가 시스템을 위한 노력이 많이 필요하다. 또한 실제 산업용에서는 IL을 중간코드로 한 표준 언어들과 IL, IL과 표준 언어들간의 양방향 변환 모듈의 개발이 필요하다고 한다. 따라서 향후 IEC1131-3 표준 언어들 중 LD 뿐 아니라 다른 표준 언어들에 대해서도 IL을 통한 C 언어로 변환할 수 있는 통합 시스템 개발이 필요하다.

참고문헌

[1] 원태현외, PLC 제어기술, 제2판, 복두 출판사, 2001.
[2] 박양수외, FA를 위한 PLC 실습, 복두 출판사, 1998.
[3] 김종부외, PLC 이론 및 실습, 복두 출판사, 2002.
[4] PLC 이론과 실습, 삼성전자 사내교육 자료.
[5] Norme Internationale International Standard, CEI IEC 1131-3, Premiere edition, First edition, 1993.
[6] http://www.adaptfav.bo.it/prod01.htm
[7] http://www.intellution.co.kr
[8] http://www.kwsoftware.com
[9] http://www.deltaww.com
[10] http://www.angelfire.com/in/bsommer
[11] IsaGRAF user's guide, Version 2.1, CJ International, 1994
[12] http://www.realgain.co.kr
[13] Russell and Norvig, Artificial Intelligence a Modern Approach 2/E Chap 2, Prentice Hall International Co., 1994.
[14] FIPA Agent Management Specifications http://www.fipa.org/repository/managementspecs.html
[15] John Graham, Real-Time Scheduling for Distributed Agents, AAAI-Spring Symposium on Real-Time Autonomous Systems, Palo Alto, CA, March, 2000
[16] 조남규, Applying UML and Patterns(인터넷 발표 자료), 2003.10

저 자 소 개



조영임(曺永任)
1987 : 고려대학교 생물학과 학사
1988 : 고려대학교 컴퓨터학과 학사
1990 : 고려대학교 컴퓨터학과 석사(인공지능전공)
1994 : 고려대학교 컴퓨터학과 박사(인공지능전공)
1995-1996 : 삼성전자 멀티미디어연구소 선임연구원
1999-2000 : Univ. of Massachusetts, at Amherst, Dept of Computer Science, Post-doc.
2003-현재 : 한국퍼지및지능시스템학회 총무, 편집위원
2004-현재 : 한국여성정보인협회 운영위원
1996-현재 : 한국정보과학회 종신회원