지능적 에이전트에 의한 실시간 소프트웨어 PLC 편집기 및 실행엔진 개발

(Development of an Editor and Running Engine for Realtime Software Programmable Logic Controller based on Intelligent Agents)

조 영 임 †

(Young Im Cho)

요 약 PC-based control은 현재 제어분야에서 비약적 발전을 하고 있으나 일반 사용자들이 PC에서 PLC 프로그래밍하기에는 어렵다는 단점이 있다. 따라서 본 논문은 국제 PLC 표준언어로 제정된 5가지 언어 중 90%이상 사용하는 LD언어에 대한 표준규격을 연구하고, 이것을 중간코드인 IL(Instruction List) 언어로 변환하고 기존 상용화된 편집기(Visual C++)에서 활용 가능한 표준 C코드로 변환함으로써 LD에 익숙한 사용자나 고급언어에 익숙한 사용자 모두 사용할 수 있는 편집기 및 실행엔진 기능을 갖춘 지능적 에이전트 기반의 통합 시스템 ISPLC(Intelligent Agent System based Software Programmable Logic Controller)를 개발하였다. ISPLC에서는 LD에서보다 C에서 논리오류 검출기능이 훨씬 용이하며, GUI기반 인터페이스를 제공하며 에이전트에 의한 프로그래밍 코드를 제공하므로 가독성이 높다. 이러한 LD->IL->C로의 코드변환체제에 관한 연구는 국내외적으로 처음 시도되는 연구이다. ISPLC를 실제 실시간 교통량 제어 시스템(Real Time Traffic Control System)에 적용하여 현장 적응성이 우수한 실행엔진을 개발하여 시뮬레이션 하였으며, ISPLC는 오류검색 뿐 아니라 프로그래밍 시간을 매우 단축시켜줌을 알 수 있었다.

키워드 : PC 기반 통합제어, 소프트웨어 PLC, HMI(Human-Machine-Interface), IEC1131-3

Abstract Recently, PC-based control is incredibly developed in the industrial control field, but it is difficult for PLC programming in PC.

Therefore, I need to develop the softeware PLC, which support the international PLC programming standard(IEC1131-3) and can be applied to diverse control system by using C language. In this paper, I have developed the ISPLC(Intelligent Agent System based Software Programmable Logic Controller). In ISPLC system, LD programmed by a user which is used over 90% among the 5 PLC languages, is converted to IL, which is one of intermediate codes, and IL is converted to the standard C code which can be used in a commercial editor such as Visual C++. In ISPLC, the detection of logical error in high level programming(C) is more eaier than PLC programming itself. The study of code conversion of LD->IL->C is firstly tried in the world as well as KOREA.

I developed an execution engine with a good practical application. To show the effectiveness of the developed system, I applied it to a practical case, a real time traffic control(RT-TC) system. ISPLC is minimized the error debugging and programming time owing to be supported by windows application program.

Key words : PC based Control, Software PLC, HMI(Human-Machine-Interface), IEC1131-3

1. 서 론

과거 유무접점 릴레이를 이용한 각종 산업용 제어장치에 대하여 구비해야 할 조건을 명기함으로써 시퀀스 컨트롤러가 출현하게 되었다. 자동화를 위하여 종전에는 제어시스템의 회로도에 따라 릴레이, 접점, 타이머, 카운

† 종신회원 : 수원대학교 IT대학 컴퓨터학과 교수
ycho@suwon.ac.kr
논문접수 : 2003년 11월 22일
심사완료 : 2005년 6월 2일

터 등을 직접 접속하여 사용하였으나, 이는 다품종 소량 생산 체제에 따른 제어 시스템의 변경에 많은 시간과 비용이 요구되었다. 이 문제를 해결하기 위하여 컴퓨터의 비약적인 발전과 더불어 1970년대 미국에서 프로그램이 가능한 제어시스템이 개발되었으며 이것을 1978년 미국의 전기협회 규격 NEMA(National Electrical Manufacturing Association)에서 PC라고 명명하였는데, 개인용 컴퓨터와의 구분을 위해 PLC로 부르게 되었다[1-4].

PLC 프로그래밍 언어는 나라마다 컴퓨터 기종마다 서로 상이하고 표준화가 되어있지 않기 때문에 오픈화와 분산화 시대에 맞지 않다는 문제점이 있다. 따라서 1993년 유럽을 중심으로 PLC 프로그램 언어의 표준활동이 추진되었는데 이 표준규격이 IEC1131-3이다. 이 표준규격은 산업용 컨트롤 프로그램의 표준화를 목적으로 개발된 유일한 국제 표준의 산업용 컨트롤 프로그래밍 언어이다. 1992년에 승인되고 1993년에 문서로서 발행된 이 표준규격은 IEC의 TC65/SC65B/WG7/TF3에서 약 10년에 걸쳐 검토한 결과이다[5].

IEC1131-3에서는 다음과 같은 5개의 언어를 규정하여 처리하고 있다. 즉, 텍스트계의 언어로 IL(Instruction List), ST(Structured Text)가 있고, 그래픽계의 언어로는 LD(Ladder Diagram)과 FBD(Function Block Diagram)이 정의되고 있으며, 중요한 공통요소로 SFC(Sequential Function Chart)가 있다[5]. 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 입출력을 조합하여 프로그래밍을 한다. 우리나라에서도 90% 이상의 기업들에서 LD를 사용하고 있다. 이렇게 정의된 5개의 언어들은 어느 나라나 지역에 편중되지 않고 전통적인 PLC용 언어를 지원하므로 프로그래머의 능력에 따라 자유롭게 실제의 애플리케이션으로 개발할 수 있다.

그러나 국내에서는 이 기술에 대한 인지도 및 적용률이 미진하여 세계적인 변화에 편승하지 못하고 있는 실정이다. 우리나라는 세계에서 장비출하 댓수가 1위인 나라임에도 불구하고 장비에 들어가는 제어기의 90% 이상을 일본에서 수입한 PLC 제어에 의존하고 있는 형편이다. 그러나 앞으로는 PC 중심의 반도체 장비 중심의 제어가 주류를 이루고 있으므로 향후 PC 중심의 자동제어방법이 중요하게 대두될 것이다. 이렇게 되면 반도체 장비에 하나의 패키지 형태도 제공함으로써 PC기반에서도 제어가 가능하게 될 것이다.

개방화, 오픈화 플랫폼 경향에 따라 기존 PLC 하드웨어 공급자에게 의존하는 방식은 점차 PC-based control을 이용하여 사용자 자신의 프로세스에 맞는 자원을 선택할 수 있도록 윈도우즈 환경 하에서 보다 쉽고 빠른 개발시간과 안정된 관리 및 유연한 확장성을 필요로 하게 되었다. 그동안 산업현장에서의 PC는 데이타 관리나 단지 HMI(human Machine Interface)를 위한 용도로 인지되어 왔었으나 이제 산업현장에서도 PC의 빠른 발전속도와 뛰어난 성능을 접목시켜 직접 제어하는 시대가 도래하고 있으므로 이 연구는 이러한 시점에서 매우 중요한 의미를 갖는다. 즉, PC-based control은 하드웨어적 PLC의 여러 단점들을 보완하고 뛰어난 성능과 유연성으로 PLC를 대체할 차세대의 솔루션으로 인정받고 있다.

그러나 IEC1131-3이라는 표준언어 제정에도 불구하고 여전히 PLC 프로그래밍 언어는 일반인이 사용하기에는 매우 어려운 언어이며 웹 환경에서 범용성을 갖기 어려운 언어이다. 또한 전문가라 해도 프로그램시 논리오류를 찾기는 매우 어렵다는 문제점을 갖는다.

따라서 본 논문은 국제 PLC 표준언어로 제정된 5가지 언어 중 대미 수출은 물론 국내에서 90% 이상을 사용하고 있는 PLC 언어인 LD언어에 대한 표준규격을 연구하고, 이것을 중간코드인 IL(Instruction List)언어로 변환하고 기존 상용화된 편집기(Visual C++)에서 활용 가능한 표준 C코드로 변환함으로써 LD 전문가는 물론 고급언어에 익숙한 사용자들도 쉽게 사용할 수 있는 편집기 및 실행엔진 기능을 갖춘 지능적 에이전트 기반의 통합 시스템 ISPLC(Intelligent Agent System based Software Programmable Logic Controller)를 개발하고자 한다. ISPLC에서는 LD에서보다 C에서 논리오류 검출기능이 훨씬 용이하며, GUI기반 인터페이스를 제공하며 에이전트에 의한 프로그래밍 코드를 제공하므로 매우 가독성이 높다. 이러한 LD->IL->C로의 코드변환체제에 관한 연구는 국내외적으로 처음 시도되는 연구이다.

ISPLC를 실제 실시간 교통량 제어 시스템(Real Time-Traffic Control System: RT-TC)에 적용하여 현장 적응성이 우수한 실행엔진을 개발하여 시뮬레이션 하였다.

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

2. 국내외 소프트웨어 PLC 기술현황 분석

2.1 국내외 기술현황분석

국외 SoftLogic 개발사례 중 대표적인 독일에서 실험적 상업적 시스템 툴로써 이미 개발된 KW System[6]은 시스템의 환경에 따라 그 규모와 복잡도가 증가함에 따라 대부분의 사용자들은 툴을 사용하려는데 많은 노력과 시간을 소모하고 있는 점을 해결하고자 비쥬얼한 인터페이스를 제공해주는 시스템으로 가장 많이 사용되고 있다. 그러나 IEC1131-3 표준언어와 고급언어와의 호환성을 구현한 소프트웨어 PLC 사례는 없다.

국내에는 SoftLogic이 산업용으로 아직 개발된 사례가 없는데, (주)리얼게인에서 교육용 PLC 언어 학습 패키지(RealPLC)를 개발하여 보급하고 있다. 이 시스템은 PLC소프트웨어만으로 다양한 언어를 학습할 수 있고, 다양한 모니터링 및 애니메이션을 할 수 있어 적은 비용으로 최상의 PLC 학습을 할 수 있는 특징을 갖는다. 그러나 이 제품은 교육용만으로 국한되어 있어 산업용으로는 활용하기 어렵다[7]. 리얼게인 시스템은 KW System과 매우 유사하나, 주된 차이점은 KW System과 달리 표준언어들 중 LD을 기반으로 프로그래밍을 하며, 저 가격으로 사용할 수 있다는 점이다. 이는 애니메이션과 사용자가 쉽게 실험할 수 있는 테스트 베드를 가지고 있기 때문이다. 국내에서는 LG 산전과 삼성에서 개발 사용되고 있는 소프트웨어 PLC 시스템들이 있다. 삼성에서 윈도우용 WinGPC[1]를 개발하였으나 자체 PLC용 장비에 대한 인터페이스만을 제공하고 있으므로 범용성이 부족하다.

2.2 문제점

지금까지 살펴본 국내외 개발 시스템은 몇 가지 제한점이 있다. 첫째, PLC 기반 언어로 프로그래밍을 하는 동안의 오류(특히 논리오류) 분석, 해석, 처리기능이 부족하다. 둘째, 오픈 소스가 아닌 블랙 보드로서의 결과만을 확인 가능하다. 셋째, IEC1131-3에서 제공하는 표준 언어들 간의 호환성이 부족하다. 넷째, 제한적인 시뮬레이션 기능을 가지고 있으므로 일반 사용자들의 기대를 만족시키기에는 부족한 점이 많으며 사용하기 어렵다. 다섯째, 범용성이 부족하고 자체개발한 PLC용 언어만을 지원한다는 것이다.

따라서 본 논문에서는 에이전트[8-11]를 이용하여 PLC 프로그래밍이 가능한 툴 내에서 사용자를 대신하여 지적 대리인으로써 사용자가 원하고 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 지능형 PLC 소프트웨어 에이전트 시스템을 개발하고자 한다. 이 시스템에서는 IEC1131-3에서 제공하는 표준 4개의 언어를 다른 한 표준언어이며 중간형태인 IL로 변환하여 표준 언어간 호환성을 갖게 하여 표준 C 컴파일러에서 컴파일링함으로써 전문가는 물론 고급언어에 익숙한 사용자가 프로그래밍시 발생하는 논리오류를 최소화시킴으로써 최적화된 환경에서 PC기반으로 제어하는 소프트웨어 PLC 시스템을 개발하는 것으로 국내외적으로 최초로 시도되는 연구이다.

3. 지능적 소프트웨어 PLC(ISPLC) 설계 및 구현

3.1 연구동기

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 대미 수출환경 및 국내 환경에 적합하고 90% 이상 점유율을 나타내고 있는 IEC1131-3 표준 언어인 LD를 IL로 변환하고 코드크기를 줄이고 향후 상하위 확장성 및 이식성을 고려한 고급언어인 C-언어를 최종 코드로 함으로써 산업적 효율성이 높은 편집기 및 실행엔진 기능을 갖춘 지능형 소프트웨어 PLC 에이전트 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 PLC 언어는 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 장점을 모두 활용할 수 있고 문법적 오류 수정이나 디버깅 기능 등을 기본적으로 사용할 수 있다는 장점을 갖는다. 이러한 연구는 국내는 물론 국외에서도 최초로 시도되는 연구이다.

IL로 변환해야 할 필요성은 IEC1131-3 표준언어들을 C로 각각 변화시키는 것보다 IL이라는 중간코드를 활용하는 것이 언어간 상호호환성은 물론 직접 C로의 변환보다 훨씬 용이하기 때문이며 산업적 활용성이 높기 때문이다.

3.2 ISPLC 개요

본 논문에서 연구개발한 실시간 지능형 통합 소프트웨어 PLC시스템인 ISPLC 시스템은 사용자 에이전트

(User Agent : UA), 컨트롤 에이전트(Control Agent : CA), 컴포넌트 데이타 스토리지(Component Data Storage : CDS), 사용자 인포메이션(User Information : UI), 사용자 추천 인덱스 모듈(User Recommendation Index Module: URIM) 등으로 구성되어 있다. 각 모듈의 설계와 필요한 메카니즘, 에이전트간 협상 기술과 같은 관련요소 기술을 개발하고 최종적으로는 모든 에이전트를 통합하여 사용자를 대신하여 최적의 조건의 자동화되는 소프트웨어 PLC 에이전트 시스템을 구축하여 하드웨어 및 소프트웨어 PLC를 위한 지능형 시스템의 기본 모형을 제안한다.

ISPLC의 개발 환경은 Window XP에서의 Visual C++로 구현하였으며, 일반 시스템과 에이전트 기반의 시스템의 논리검출에 따른 디버깅 스텝을 비교 평가하였으며, 실제 비주얼한 환경에서 보여주기 위해 LD를 IL로 변환하고 다시 C로 변환할 수 있는 에디터도 개발하였다.

3.3 ISPLC 세부모듈

ISPLC의 전체적인 구조도는 다음 그림 1과 같다.

본 논문에서 개발한 ISPLC 시스템은 교통 신호 시스템, 엘리베이터 시스템, 공장 자동화 시스템 등과 같은 실세계에서 적용되는 산업용 기술의 제어 및 프로그램이 가능 할 수 있도록 하는데 목적이 있다.

ISPLC를 위한 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. 또한 이 모듈은 사용자 인터페이스 상에서 프로젝트나 시스템 개발시 이전의 프로그램 설계에 대한 컴포넌트 및 클래스 패턴 히스토리를 가지고서 사용자에게 제안해주는 모듈이다. 코드 매핑모듈(Code Mapping Module)은 LD, IL, C or Visual C++, 세 언어 중 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 프로그래밍 시 사용자의 인터페이스내에 각각의 LD 모듈에 대한 프로그램이 가능할 수 있도록 소프트웨어 코드 즉, C 또는 Visual C++ 변환하여 생성해준다. 또한 사용자가 LD 언어를 사용하지 못하더라도, C 언어로 변환되므로 구현 및 프로그램이 가능하다. 사용자 인포메이션 모듈(User Information Module)은 사용자의 히스토리 데이타, 즉 프로그램 가능한 컴포넌트나 클래스 데이타를 최적화하여 재사용성을 제공하기 위한 모듈이다. 컴파일러 모듈(Compiler Module)은 시스템의 툴 내에서 제공하는 인터프리터로서 파싱처리가 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 데이타 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 저장함으

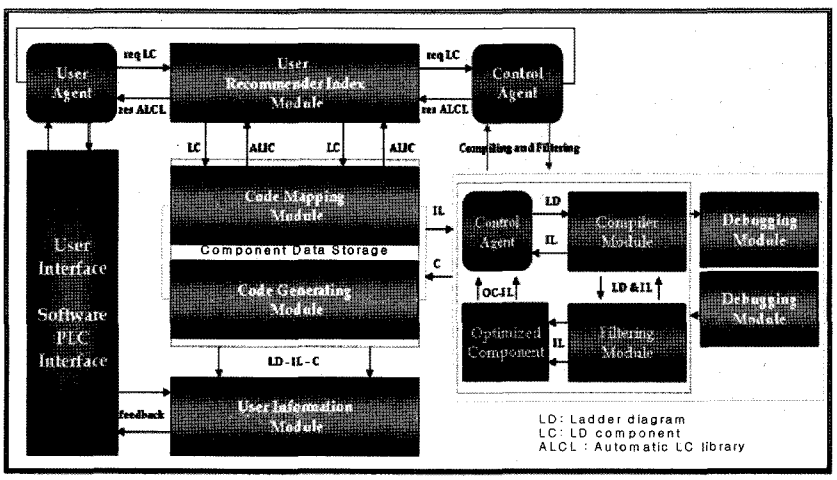


그림 1 ISPLC 전체 시스템 구조

로써 재사용이 가능할 수 있도록 한다.

ISPLC에서 주요 에이전트간 플로우 다이어그램을 설명하면, 먼저 사용자는 LD로 에디터 상에서 프로그래밍한다. LD로 프로그래밍 된 시스템은 자동적으로 IL로 변환되며, 그 변환 모듈은 일대일 매핑과정을 통해 이루어진다. IL로 변환된 모듈은 Visual C++ 형태로 조건제어의 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.

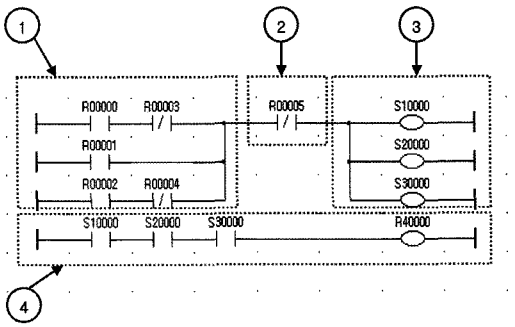
[1단계] LD 작성
- Step 1: 임의의 LD로 작성
- Step 2: 파라메터 값을 갖는 초기값으로 표준화 수행
- Step 3: LD 루프수행
- Step 4: 새로운 프로토타입과 라인을 이용하여 프로그래밍
- Step 5: 시뮬레이션

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

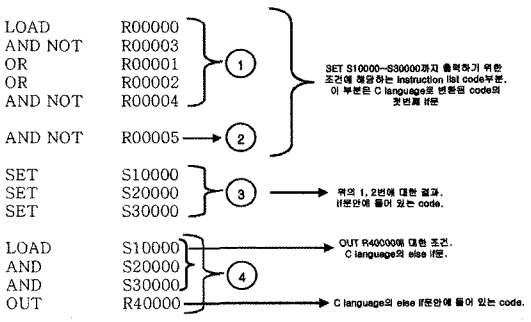
[3단계] C 또는 Visual C++ 변환
- Step 1: 단계 1과 단계 2를 이용한 초기화
- Step 2: 초기 코드로 표준화 수행
- Step 3: 메소드와 파라메터 값을 매핑
- Step 4: 해당 클래스와 객체 생성
- Step 5: 컴포넌트의 필터링 및 재생성
- Step 6: 컴파일링 수행(최종단계)

본 논문에서 제안한 LD에서 IL로, IL에서 C로의 코드변환 알고리즘을 단계별로 설명하면 다음 그림 2와 같다.

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.



(a) 초기상태



(b) IL로 변환된 코드

그림 2 LD에서 IL로의 코드변환 관계

이때 R00000~R00002 중 한 가지만 ON이 되면(1이면), 내부 코일 S10000~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.

본 논문에서 구현한 LD에서 IL로의 코드변환을 위한 세부 알고리즘 스텝은 다음 그림 3과 같다.

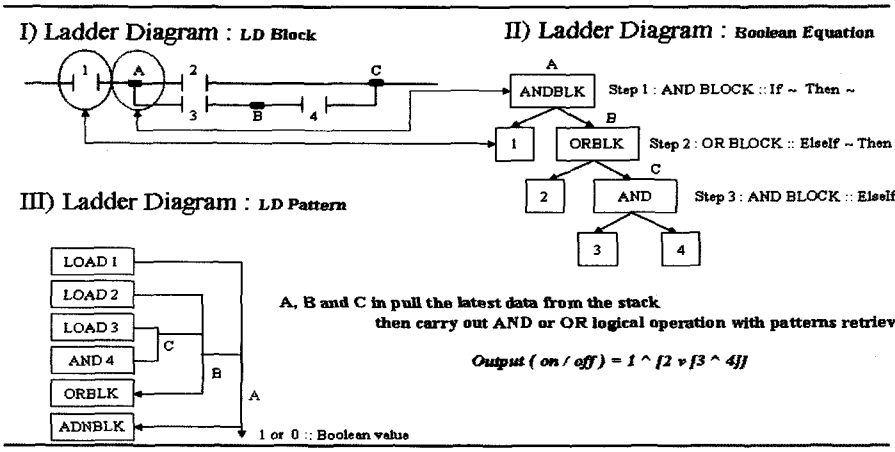
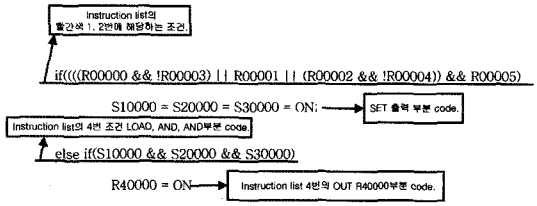


그림 3 LD에서 IL로의 세부 코드변환 스텝

위 알고리즘 스텝 I)에서 LD를 컴포넌트 별로 분류하여 II)에서와 같이 부울함수에 의해 분류한다. 이때 AND, OR 연산에 의해 IF~EISE 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 패턴 분류하여 규칙베이스를 구성한다.

ISPLC의 각각의 에이전트 및 개발 모듈에 관한 설명은 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 데이타 스토리지의 데이타베이스를 관리할 수 있도록 한다. 이 밖에 사용자의 지적 대리인으로서의 동작을 위해 UA는 각 사용자의 개발 환경과 언어 그리고 요구에 따라 자동적인 컴포넌트를 생성 및 제공해주는 메카니즘을 갖추고 있다. 사용자는 인터페이스를 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 컴포넌트 정보를 자동 피드백 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 컴파일을 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 수행 이후 자동 저장되며, 각각의 모듈을 컴포넌트와 클래스 단위로 저장하여 히스토리 데이타를 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 파라메터값을 의미하는 변수이다.

```
UA (LD [Name], LD [Parameters value], Time)
{
   Start(0.0.0);
   While (S1 is not 0 and 0.0)
```

```
Initializing LD Component Start
  If (LDCode_P(value) = 0)
        {
          Seq_P += Par_val
          // Initialized
        }
    Then UA(LD_N, LD_P) Next Loop;
}
Return Seq_P( );
```

(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 데이타 스토리지와의 통신을 통해 개발 환경에 알맞는 컴포넌트 및 컴파일러를 제공해주는 역할을 한다. 사용자 에이전트는 컨트롤 에이전트에게 사용자로부터 요청받은 질의 LD를 요청하면 컨트롤 에이전트는 컴포넌트 데이타 스토리지와 사용자 인포메이션을 통해 사용자가 원하는 컴포넌트 정보를 인터페이스의 작업 환경에 맞게 제공한다. 각각의 컴포넌트는 모듈 단위로써 최적화 및 지식 기반 데이타를 제공한다.

(3) 사용자 추천 인덱스 모듈(User Recommendation Index Module: URIM)

사용자가 사용한 데이타 또는 사용할 데이타를 사용자의 프로그램 성향에 따라 데이타 히스토리를 인덱싱하는 역할을 담당한다. 이 모듈은 최적화된 컴포넌트로 구성되어 있으며, 컴포넌트 데이타 스토리지와 사용자 인포메이션과 유사한 구조를 가지고 있다. 다음 그림 4는 ISPLC에서 사용자 성향을 학습하기 위해 구현한 사용자 에이전트에서의 사용자 표준언어(LD) 컴포넌트 기반의 지식기반 규칙 베이스에 관한 라이브러리를 구성한 경우를 나타낸 것이다. 만약 그림에서 왼쪽의 첫 번째 모양의 컴포넌트이면 시작을 나타내므로 이후에 나타낼 수 있는 컴포넌트를 규칙으로 갖고 있으며 sequence, If/else, switch문과 같은 형태로 규칙베이스를 구성한다.

(4) 컴포넌트 데이타 스토리지(Component Data Storage : CDS)

ISPLC에서 모든 데이타를 관리하고 저장하는 역할을 담당하는 데이타베이스이다. 이는 UA와 CA에게 사용자가 원하는 데이타 즉, 컴포넌트들을 제공하며, 사용자 인포메이션에 지식 기반 히스토리를 제공해 준다. 또한 UA에 필요한 컴포넌트를 URI를 통해 사용되어 질 수 있도록 한다. CA에게서 필터링 및 최적화된 컴포넌트를 UA의 요구에 맞게 구성하는 역할을 한다.

(5) 사용자 인포메이션(User Information : UI)

UA, CA 그리고 CDS의 모든 필터링 및 최적화된 컴포넌트를 사용자에게 필요로 하는 정보 히스토리를 담당하는 메카니즘이다. 이는 사용자 인터페이스를 제공하도록 하는 기능이 있으며, CDS에게 데이타를 피드백한다.

4. RT-TC 시스템의 시뮬레이션

4.1 실시간 교차로 신호등 제어 문제

본 논문에서는 ISPLC의 적용 시스템으로써, 가장 많이 사용하는 PLC 적용 시스템인 실시간 교차로 신호등 제어 시스템(Real Time-Traffic Control System: RT-TC System)에 적용하여 시뮬레이션 하고자 한다. 이 시스템은 실세계의 도로상태를 소프트웨어 기반의 환경으로 옮겨 차량 운행에 대한 교통 신호를 제어하기 위

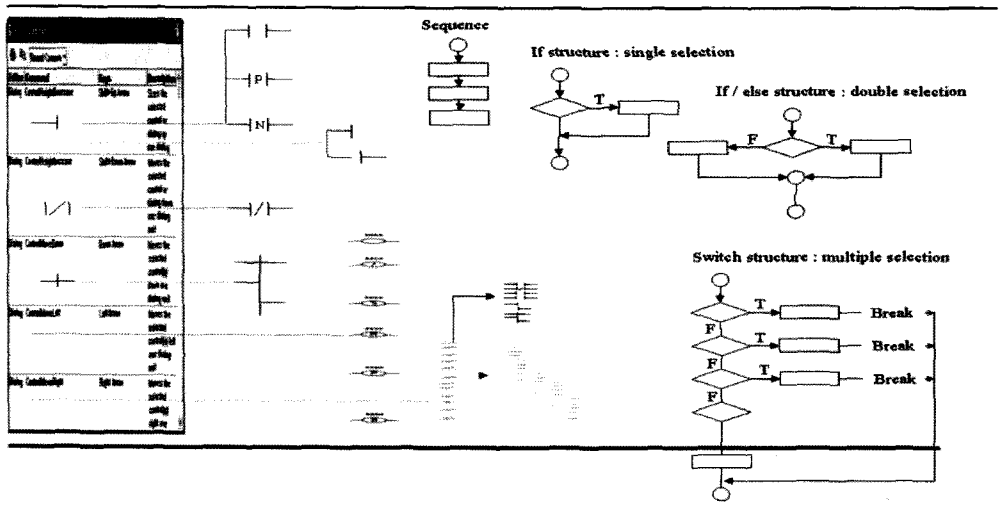


그림 4 사용자 추천 인덱스 라이브러리에 의한 규칙기반 시스템

한 시뮬레이션이다. 이 시스템의 목적은 차량 통제에서의 오버플로우 현상을 방지하기 위함이다. 하지만 단지 오버플로우에 대한 실시간 타임을 적용시켜 제어함으로써 차량 소통에는 문제가 없지만, 사용자 입장에서의 프로그램시 제어에 대한 시스템적 성능에 문제점이 발생된다. ISPLC 시스템을 좀 더 효율적으로 구현하고자 에이전트 모듈을 생성했으며, 하드웨어 컨트롤을 위한 LD 언어를 소프트웨어 환경에서의 LD 에디터를 이용하였다. 모든 오퍼레이팅 시스템은 LD 에디터 상에서 제어 가능 할 수 있도록 하였으며, 사용자 입장에서의 프로그램이 가능할 수 있도록 Visual C++로 또한 구현 설계하였다. 이는 하드웨어 및 소프트웨어 두 환경을 모두 컨트롤 하기 위함이며, 에이전트의 기능을 추가함으로써 지능형 시스템을 생성하였다.

RT-TC 시스템의 모델링 구성도는 그림 5와 같다. 그림 5(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 5(b)에서 각 방향별 출력 번호는 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 5(b)는 RT-TC 신호등의 PLC 출력 접점을 표시한 것이다.

4.2 RT-TC 시스템 알고리즘

RT-TC의 수행 알고리즘은 다음 그림 6과 같다. 알고리즘에서 사용된 PLC 명령어를 설명하면 다음과 같다. JMP(Jump) 명령어는 프로그램 처리 중에 회로의 일부를 수행하지 않고 강제로 점프할고자 할 때 사용하는 명령이고 LBL은 라벨번호로써 점프할 위치를 나타

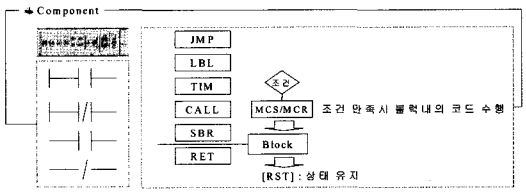
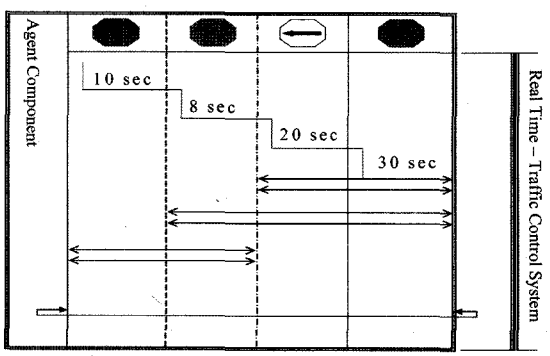


그림 6 RT-TC 알고리즘

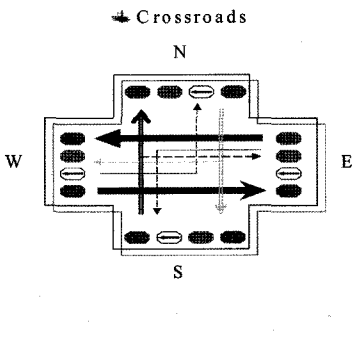
내므로, JMP와 LBL 명령은 짝을 이루어야 한다. TIM (On Delay Timer)은 타이머를 설정하는 명령으로 0.01초~6553.5초 까지 설정할 수 있으며 설정시간동안 한시적으로 정지한다. CALL-SBR-RET 명령은 서브루틴을 작성하고 이를 호출하기 위해 사용하는 명령이다. CALL 명령은 서브루틴을 호출하는 명령이고, SBR은 서브루틴의 시작을 나타내고, RET는 서브루틴의 끝을 나타낸다. CALL 명령에서 지정한 Sb 번호를 가지는 SBR 명령까지 분기한다. 해당 SBR-RET 사이의 명령들을 수행한 후 CALL 명령 다음으로 복귀한다. MCS (*Master Control Set*)/MCR(*Master Control Reset*) 명령은 지정된 조건을 사용하여 블록 단위의 회로를 실행시키는 명령으로, 조건에 의한 회로 블록 실행 시작과 종료 시점에 각각 사용되며 항상 짝을 이루어 실행된다. SET, RST(Reset)명령은 연산조건이 만족되면 지정한 접점을 각각 ON/OFF 상태를 유지시키는 명령으로 SET은 RST 명령으로 리셋시키기 전까지는 계속 ON을 유지한다.

본 논문에 사용된 RT-TC 시스템의 알고리즘을 PLC 명령어에 따라 LBL로 점프하여 설정된 타이머 동안 정지하고 서브루틴을 호출하여 조건 만족시 코드 수행하고 다시 와서 다음 명령을 수행하는 방식으로 RT-TC 시스템이 동작한다.

Signal Light(신호등)는 로드 상에서의 신호등 타임과



(a) RT-TC 시스템의 각 타이머 설정 시간



(b) RT- TC 신호등 체계

그림 5 RT-TC 시스템 모델링

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 스케쥴러에따라 자동차의 이동 제어를 하며, 보행자 신호와 교차로 신호에 따라 순차적으로 수행한다.

자동차 신호와 보행자 신호는 초기 타임을 가지고서 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 로드상의 신호 또한 대기함으로써 하나의 로드상의 교통 제어 플로우가 형성된다.

신호등 LD 알고리즘 엘리먼트를 10개의 네트워크로 표현할 수 있으며, 이것은 하나의 로드 사이클을 형성한다. 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. 네트워크 4는 자동차 신호가 로드상의 자동차를 대기 상태의 신호로 변화한다. 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 유지하도록 한다. 네트워크 8은 메모리 비트에 저장되며, 보행자의 신호와 자동차 신호가 변화 가능 하도록 펑션 블록에서 신호의 체계를 제어한다. 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 RT-TC 시스템 구현 및 시뮬레이션

(1) ISPLC를 적용한 RT-TC 시스템 구현 개요

에이전트 기반의 ISPLC 시스템을 실제 RT-TC 시스템에 적용하기 위해 구현된 예를 보면 다음 그림 7과 같다. 이 시스템은 LD->IL->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

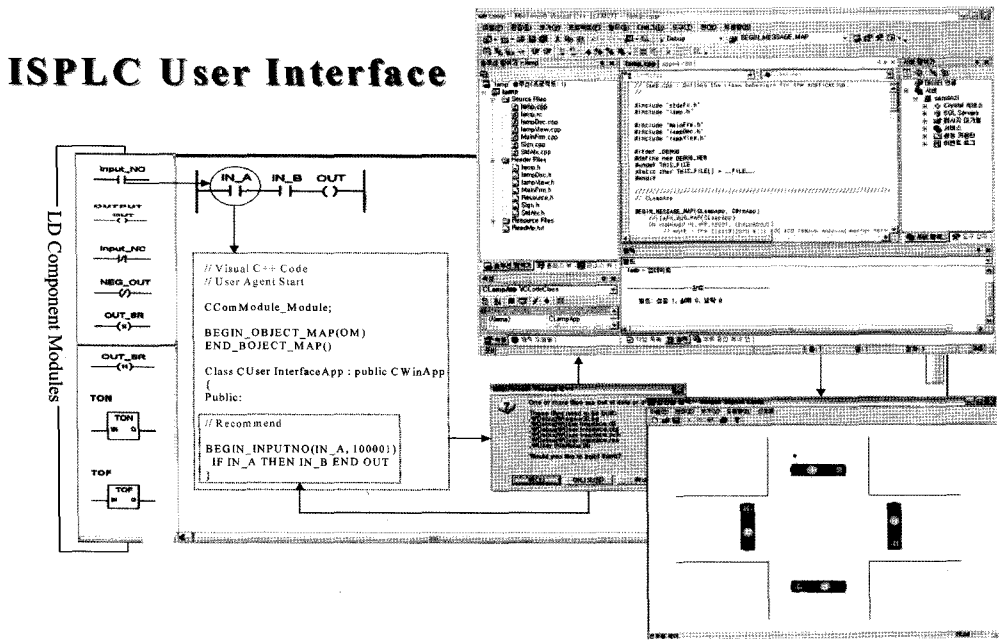


그림 7 에이전트 기반 ISPLC의 RT-TC 구현을 위한 LD->IL->C 변환 인터페이스

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.

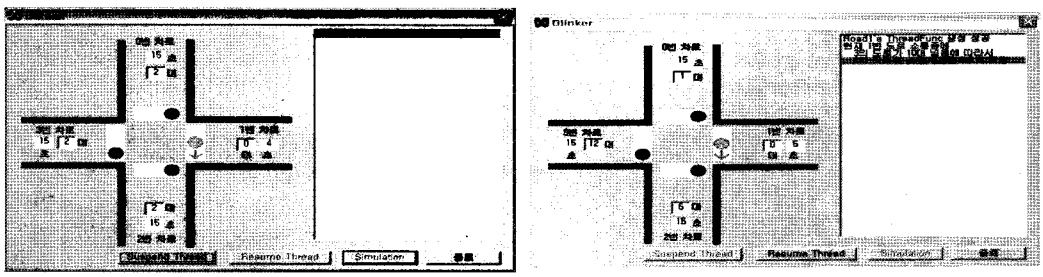
위 그림에서 각 모듈을 설명하면 ① 왼쪽 모듈 : LD 사용자 인터페이스 ② 왼쪽 아래 모듈 : LD Component ③ 삭제 모듈: LD 명령어 삭제기능 ④ 메시지 박스 모듈 : LD Compiling, 사용자 에이전트 코드 뷰에 의한 오류 검출기능 ⑤ 오른쪽 모듈 : LD 코드를 Visual C++ 코드로 변환시키는 모듈 ⑥ 오른쪽 아래 모듈 : Visual C++ 코드 형태로 에이전트가 추천해주는 코드를 나타내는 모듈이다.

(2) 4개 차로를 갖는 실제 RT-TC 구현

그림 8은 ISPLC 시스템을 4개의 차로를 갖는 교차로에서 각 차로에서의 차량의 흐름에 따라 신호등 시간을 동적으로 제어할 수 있도록 구현하여 적용한 예이다. 그림 8(a)는 RT-TC의 초기 화면으로, 0번~3번의 총 4개의 차로의 자동차 댓수와 평균 소통시간을 나타낸 화면으로 15초를 기본 소통시간으로 세팅한다. Suspend Thread 버튼을 누르면 1번 차로의 ThreadFunc이 생성된다. 그림 8(b)는 현재 1번 차로 소통 중에 3번 차로의 차량이 10대가 넘으므로 2번 차로의 소통시간을 5초로 조정해야 함을 나타내는 화면이다.

(3) LD->IL-> C 코드로의 변환과정을 보여주는 인터페이스

LC->IL->C 코드로의 변환과정을 보여주는 인터페이스는 다음 그림 9와 같다. 그림에서 왼쪽이 LD 창이며 아래쪽 버튼들은 선택할 수 있는 LD 컴포넌트들을 나타내며, LD 컴파일과정에서 오류발생시 검출해주는



(a) RT-TC 초기 화면 (b) 1번 차로의 쓰레드 함수 수행 후 화면

그림 8 RT-TC 시스템의 인터페이스

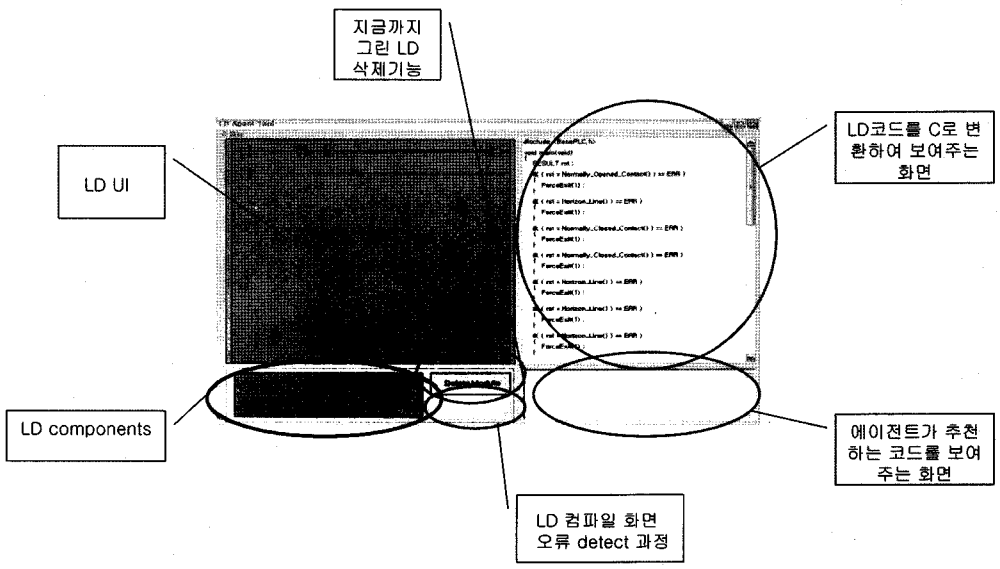


그림 9 LD->IL->C 코드로의 변환 인터페이스

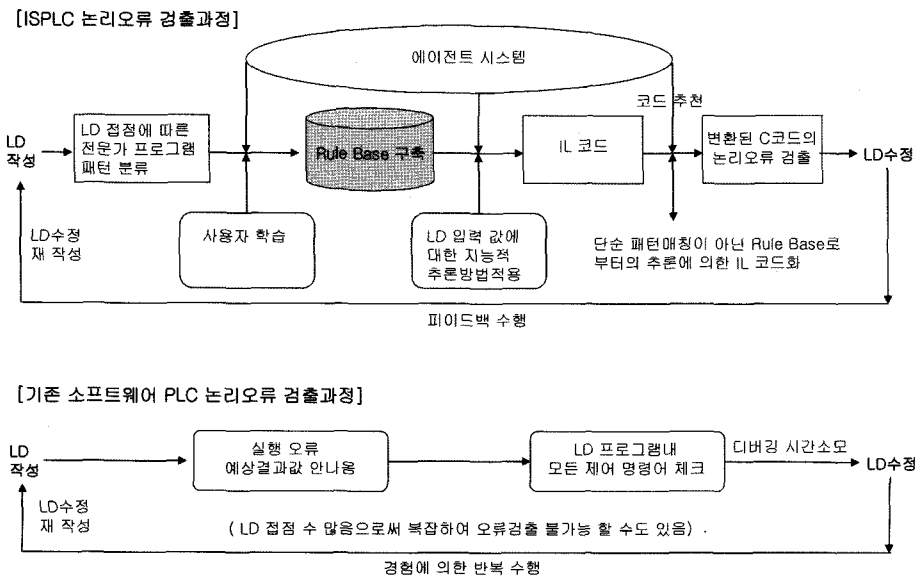


그림 10 ISPLC와 기존 소프트웨어 PLC의 디버깅 과정 비교

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

LD 컴포넌트를 이용해 프로그래밍 하면 실시간으로 오류 검색을 하여 오류 발견시 에이전트에 이해 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 효율적이 된다.

4.4 연구결과 비교분석

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 데이타를 관리하고 사용자의 사용패턴을 학습한다. 컴포넌트 라이브러리 등을 제공하고 사용자에게 올바른 C코드를 추천해 준다. 이것을 위해 IL->C로의 규칙적인 학습패턴을 유지하여 규칙베이스에 저장함으로써 가능하게 된다. 따라서 ISPLC는 사용자가 많이 사용하면 할수록 C코드로의 변환이 더 효율적이며 논리오류를 최소화시킬 수 있게 된다. 컨트롤 에이전트는 실제로 변환된 C 코드로부터 컴파일을 하고 오류나 패턴을 필터링하여 사용자에게 피이드백함으로써 LD를 재작성할 수 있도록 상호작용한다.

ISPLC와 기존 소프트웨어 PLC의 논리오류 디버깅에 소요되는 시간을 측정하기 위해 RT-TC에서 LD 명령어 패턴수를 증가시키면서 시뮬레이션한 결과, ISPLC는 에이전트를 도입함으로써 에이전트가 검출한 논리오류만큼의 프로그램내의 특정 부분의 제어관련 명령어들을 확인하고 메시지 흐름의 순서에 따라 원인을 순서대로 규명해야 되므로 프로그램이 복잡해지더라도 평균적으로 $\sqrt{n}(n: LD$명령어패턴의갯수) 정도의 디버깅 스텝수를 갖는 경향이 있음을 시뮬레이션결과 알 수 있었다. 그러나 기존 소프트웨어 PLC는 LD 프로그램 작성 후 실행이 안되었을 경우 디버깅하기 위해서는 에이전트에 의해 논리오류를 체크하는 기능이 없으므로 프로그램내의 모든 제어관련 컴포넌트들을 확인하고 제어흐름의 순서에 따라 모든 명령어들의 상관관계를 분석하면서 오류 원인을 차례로 규명해야 하므로 $nPn = n!$의 디버깅 스텝수를 가짐을 시뮬레이션결과 알 수 있었다. 따라서 LD 접점수가 많으면 많을수록 오류검출 가능성은 줄어들게 된다. 이것을 그래프로 나타내면 지수함수 비율로 디버깅 스텝수가 증가함을 알 수 있었다(그림 11).

따라서 ISPLC를 사용할 경우 기존 소프트웨어 PLC보다 논리오류를 현저하게 줄여줌을 알 수 있었다.

5. 결 론

본 논문에서는 실시간 지능형 통합 소프트웨어 PLC

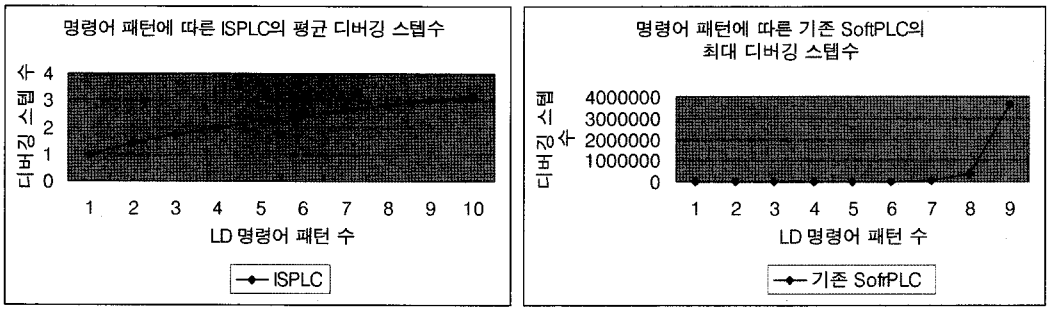


그림 11 ISPLC와 기존 소프트웨어 PLC의 디버깅 스텝수 비교

인 ISPLC(Intelligent Agent System based Software Programmable Logic Controller)를 구현하였다. ISPLC는 5개의 표준 언어들 중 90% 이상을 점유하고 있는 LD(Ladder Diagram)를, 중간코드이며 5개 표준 언어 중의 하나인 IL로 변환하고 이를 다시 고급 언어인 표준 C 또는 Visual C++ 코드로 실시간으로 변환시키는 통합된 인터페이스 환경을 제공함으로써 표준 언어 상호간 호환성은 물론 LD에 익숙한 사용자나 고급언어인 C언어에 익숙한 사용자들에게 편리함을 도모하고자 하였다. 이러한 연구는 국내외적으로 개발된 소프트웨어가 없었으므로 처음으로 시도된 연구이다.

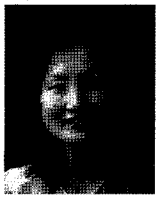
본 논문에서는 ISPLC를 지능형 시스템에서의 LD, IL, C 또는 Visual C++ 매핑과 더불어 에이전트를 기반으로 한 실시간 교통 신호 제어 시스템(Real Time Traffic Control System)에 적용하여 구현하였다. RT-TC 시스템은 실시간으로 사용자가 제어하기 편리한 인터페이스 환경을 제공한다. 이것을 위해 클래스 라이브러리와 에이전트 변환 모듈을 구현하였다. ISPLC에서는 LD를 C코드로 변환시킴으로써 표준 C컴파일러 상에서 지능적 에이전트에 의한 논리오류를 체크할 수 있는 장점이 있다. 이것은 LD상에서 논리오류를 찾아내는 것보다는 C 컴파일러상에서 에이전트에 의한 논리오류의 수정기능을 갖는 것이 훨씬 효율적이기 때문이다.

그러나 ISPLC는 LD->IL->C로의 코드 변환을 통합된 환경에서 제공하나 역관계로 코드변환이 이루어지지 않으므로, 만약 논리 오류 발견시 실제 사용자는 변환된 코드로부터 다시 LD 프로그램을 수정해야 하므로 개발툴의 활용능력이 우수해야 한다는 단점을 갖는다. 즉, ISPLC는 완전 자동화가 아닌 반 자동화 형식으로 사용자에게 코드변환 관계를 설명한다. 또한 ISPLC에서 사용되는 에이전트의 기능이 다소 제약적이므로 보다 효율적인 에이전트 알고리즘이 개발되어야 한다.

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.

참 고 문 헌

[1] 원태현외 6인, PLC 제어기술, 제 2판, 복두 출판사, 2001.
[2] 박양수외 2인, FA를 위한 PLC 실습, 복두 출판사, 1998.
[3] 김종부외 3인, 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] KW 시스템 개발사이트: http://www.kw-software. com
[7] 리얼게인 개발사이트: http://www.realgain.co.kr
[8] Russell and Norvig, Artificial Intelligence a Modern Approach 2/E Chap 2, Prentice Hall International Co., 1994.
[9] FIPA Agent Management Specifications http://www.fipa.org/repository/managementspecs.html
[10] John Graham, Real-Time Scheduling in Distributed Multi Agent Systems, Ph.D. Dissertation, University of Delaware, January, 2001.
[11] John Graham, "Real-Time Scheduling for Distributed Agents," AAAI-Spring Symposium on Real-Time Autonomous Systems, Palo Alto, CA, March, 2000.



조 영 임

1988년 고려대학교 전산과학과 졸업 1990년 고려대학교 일반대학원 전산과학과 석사. 1994년 고려대학교 일반대학원 전산과학과 박사. 2005년 3월~현재 수원대학교 IT대학 컴퓨터학과 교수. 1996년~2005년 2월 평택대학교 컴퓨터과학과 교수. 1995년~1996년 삼성전자 멀티미디어 연구소 선임연구원. 1999년~2000년 University of Massachusetts, Post-doc. 1996년~현재 한국정보과학회 편집위원. 2003년~현재 한국퍼지 및 지능시스템학회 이사(편집위원). 2003년~현재 제어자동화시스템공학회 이사. 2004년~현재 한국전자상거래학회 이사