제어 및 모니터링 소프트웨어 자동 생성을 위한 XML 기반 프레임웍

(A XML Based Framework for Automatically Generating Control and Monitoring Software)

유 대 승 † 김 종 환 † 이 명 재 ††
(Yoo Dae-Seung) (Kim Jong-Hwan) (Yi Myeong-Jae)

요 약 본 논문에서는 여러 자동화 장비들의 제어 및 모니터링 소프트웨어에 대한 쉬운 개발, 유지보수, 확장성을 향상시킬 수 있는 프레임웍을 제안하고자 한다. 본 논문에서 제안하는 프레임웍은 세 가지(IID, MAP, CMIML)의 XML 문서와 두개(VI Wizard, Generator)의 툴로 구성된다. IID는 장비에 대한 인터페이스를 기술하고, MAP은 IID에서 기술된 인터페이스와 실제 장비 드라이버 API간의 연결정보를 기술하며, CMIML은 제어 및 모니터링 소프트웨어를 기술한다. 제안하는 프레임웍의 범용성과 플랫폼 독립성을 지원하기 위하여 IID, MAP, CMIML은 XML 문서 형식으로 기술되었다. VI Wizard는 IID와 기작성된 CMIML을 입력으로 소프트웨어를 기술하는 CMIML(플랫폼 독립적인 중간 문서)을 생성하고, Generator는 VI Wizard에서 생성된 CMIML과 MAP을 이용하여 제어 및 모니터링 소프트웨어(플랫폼 종속적인 코드)를 자동 생성한다. 제안하는 프레임웍은 GUI 기반으로 제어 및 모니터링 소프트웨어를 자동 생성함으로써 쉬운 개발과 유지보수성을 제공하고, XML 기반의 기술문서 사용으로 플랫폼 독립성을 제공하면서 범용적으로 사용할 수 있도록 한다. 또한 플랫폼 종속적인 코드 재사용이 아닌 플랫폼 독립적인 소프트웨어 기술 문서를 재사용함으로써 재사용성을 증가시킬 수 있다.

키워드 : 제어 및 모니터링 소프트웨어, 자동생성, XML, IID, MAP, CMIML, VI Wizard, Generator

Abstract In this paper, we present a framework which is used to develop, modify, maintain and extend a control and monitoring software easily for any kind of automatic instruments. The proposed framework is composed of three XML documents (IID, MAP, CMIML) and two tools (Virtual Instrument Wizard, Generator). Interface information of behaviors and states of instrument is written on IID. Mapping information between the interface information in IID and API of a real instrument driver is written on MAP. Final information of the control and monitoring software is written on CMIML. IID, MAP and CMIML are written by XML format to provide a common usage and platform independence of the proposed framework. VI Wizard generates CMIML intermediate platform independent document using IID and existing CMIML, and Generator generates the source code of a control and monitoring software platform dependent code automatically using CMIML and MAP. The suggested framework provides an easy development and maintenance because it automatically generates a control and monitoring software in GUI environment and it also provides common usage and platform independence in virtue of using description document of XML format. Also, reusability can be increased by reusing platform independent software description document and not reusing platform dependent software code.

Key words : Control and Monitoring Software, Automatically Generating, XML, IID, MAP, CMIML, VI Wizard, Generator

· 이 논문은 2004년 울산대학교 연구비 및 산업자원부와 울산광역시 지원 울산대학교 네트워크 기반 자동화연구센터의 지원에 의한 것입니다.

† 정 회 원 : 울산대학교 컴퓨터정보통신공학부
ooseyds@mail.ulsan.ac.kr
bearknight@nate.com
†† 종신회원 : 울산대학교 컴퓨터정보통신공학부 교수
ymj@mail.ulsan.ac.kr

논문접수 : 2005년 1월 21일
심사완료 : 2005년 11월 25일

1. 서 론

산업의 발전과 그에 따른 작업 환경의 변화는 점점 생산 현장에서 사용되는 장비들을 자동화시켜갔고 이런 자동화 장비들은 필드버스, 이더넷과 같은 산업용 네트워크를 통해서 서로 연결이 되어 크고 복잡한 설비를 이루게 된다. 다양하고 복잡해져 가는 설비에 대한 문제를 해결하기 위해서 장비들은 규격화되고 일반화되어 가고 있지만 장비 종속적인 제어기술과 제어방법에 대한 표준은 아직 미비한 상황이다. 복잡한 설비들을 제어하기 위하여 장비들을 직접 제어하기보다는 PC 기반으로 제어하게 됨에 따라 이런 자동화 장비들의 제어 및 모니터링을 위한 소프트웨어의 중요성이 점차 증대되고 있다.

다양한 네트워크를 통해서 연결된 많은 자동화 장비들의 제어와 모니터링을 위한 소프트웨어 개발에는 몇 가지 문제가 존재한다. 자동화 장비들은 제조사 별로 제공하는 드라이버 API가 다르기 때문에 복잡한 설비를 이루고 있는 복합 장비들을 통합적으로 제어하기 위하여 장비별로 별도의 소프트웨어를 개발해야 한다. 또한 장비의 드라이버 API가 변경되거나 새로운 요구가 발생되는 경우 재개발해야 한다는 어려움이 있다.

그리고 장비들을 연결하는 네트워크와 장비가 지원하는 플랫폼이 다양하기 때문에 소프트웨어가 다양한 네트워크와 플랫폼을 지원하도록 개발되어야 하는 문제가 있다. 이런 문제들로 인해서 자동화 장비들의 제어와 모니터링을 위한 소프트웨어의 개발과 유지보수에 많은 비용이 소요된다.

본 논문에서 설계하고 구현한 프레임웍은 장비뿐 아니라 제어 및 모니터링 소프트웨어도 XML 기반의 문서로 기술함으로써 장비와 소프트웨어간의 종속성을 최소화한다. 또한 소프트웨어를 기술하는 문서(플랫폼 독립적)와 실제 코드(플랫폼 종속적)를 분리함으로써 코드 재사용이 아닌 소프트웨어를 기술하는 문서 재사용을 통해서 재사용성을 증가시키고 개발 및 유지보수에 대한 비용을 감소시킬 수 있다. 그리고 GUI 기반의 툴을 제공함으로써 쉽고 빠른 개발이 가능하고, 또한 유지보수성을 향상시킬 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 국내외 연구 동향에 대하여 기술한다. 3장에서는 본 논문에서 제안하는 제어 및 모니터링 소프트웨어 자동생성 프레임웍에 대하여 설명한다. 4장에서는 시스템 구현과 사용예를 보이고, 과거 연구와 비교를 통한 평가를 기술한다. 마지막으로 5장에서는 결론 및 향후 연구과제에 대하여 기술한다.

2. 관련연구

2.1 국내외 연구 동향

기존의 산업 현장에서 사용되던 장비나 설비는 내부에 제어를 위한 프로세서를 탑재하고 있었고, 사용자의 직접적인 조작을 통해서 동작이 되어졌다. 이런 장비들은 대부분 상당히 고가였고, 유지보수에도 상당한 비용이 소요되었다. 따라서 산업체들은 비용의 절감을 위해 장비나 설비의 자동화에 관심을 갖게 되었고, 자동화 장비의 구현이나 장비들을 연결하는 프로토콜의 구현에 대한 연구가 진행되어 왔다.

90년대 초 PC 기반의 자동화 시스템에 대한 연구가 이루어지기 시작했다. 당시에는 외산DCS가 많이 사용되었으나, 중소규모의 공장에서 사용하기에는 너무 고가였고 유지보수 비용 또한 많이 소요되었다. 이런 문제에 대한 해결책으로 상업화되어 있는 기존 소프트웨어를 분석해서 소프트웨어의 구조를 개선하고, 기존의 대규모 공정에 맞도록 설계된 알고리즘을 이용해서 중소 규모의 공정에 알맞은 제어시스템을 개발하는 연구[1]가 있었지만 당시에는 소프트웨어의 수준이나 성능에 미흡한 점이 많이 존재하고 있었다.

90년대 후반, 2000년에 이르러 본격적으로 PC 기반의 제어 시스템의 개발에 대한 연구가 이루어지기 시작했다. 윈도우 운영체제의 성능이 많이 개선되었고, PC의 가격은 저렴해지고 성능은 오히려 더 좋아지면서 점점 PC 기반의 제어 시스템에 대한 연구가 활발히 진행되는 시점이었고, 다양한 분야에서 활용되어 지고 있었다[2]. 이와 더불어 생산시스템이 빠르게 변화하는 생산기술과 컴퓨터 관련 기술에 빠르게 적응하기 위해 유연성, 통합성 및 동시성을 지원하는 개방구조로의 전환이 요구되어 지고 있었고, 이를 위해 PC 기반의 개방형 시스템에 대한 연구가 진행되었다[3]. 이런 PC 기반의 생산 시스템은 운용소프트웨어의 개발과 유지보수에 많은 비용이 들어가기 때문에 이를 객체지향방법을 이용한 소프트웨어 개발 방법에 대한 연구가 있었다[4]. 그러나 아직 이런 PC 기반의 생산 시스템에 대한 연구의 주된 주제는 장비들의 자동화에 대한 것이고, 소프트웨어의 개발에 대한 연구는 앞서 언급했던 몇몇 연구를 제외하고는 거의 전무한 실정이다.

국외에서는 이미 오래전부터 많은 연구가 진행되어 오고 있고, 이미 상용화 되어 있는 관련 소프트웨어들이 출시되어 있으며, 그 성능이 상당한 수준에 이르러있다. 최근에는 산업용 네트워크 프로토콜인 필드버스에 대한 연구가 활발히 진행되고 있다. 또한 범용적인 인터넷 프로토콜인 TCP/IP 기반의 제어 시스템에 대한 연구가 진행 되고 있으며 실시간성 제공에 대한 제약이 해결된다면 비용절감 및 다른 시스템과의 원활한 통합 등의 장점을 가질 수 있다.

제어 시스템이 장비와 장비에 대한 API에 종속적인 문제점들을 해결하기 위한 방법으로 장비에 대해 기술하는 문서를 작성하고 그 문서를 통해서 장비의 제어와 모니터링을 수행하도록 하는 방법에 대한 연구가 있었다.

대표적으로 필드버스 표준 중 하나인 Profibus의 EDDL(Electronic Device Description Language)[5]과 GSD(Generic Station Description)[6], 그리고 다른 표준인 CANOpen의 EDS(Electronic Data Sheet)[7]가 있다. EDDL은 Profibus에서 사용되어지는 디바이스들의 내부구조와 사용자 API, 그리고 도움말에 대해 기술하는 파일이고, GSD는 Profibus에 연결된 장비들의 통신 설정이나, 다른 파일을 얻어오기 위한 통신 파라미터들을 기술하는 파일이다. 그리고 EDS는 CANOpen에서 사용되어지는 디바이스에 대해 기술하는 파일이다. 이 파일들을 HMI(Human Machine Interface)와 같은 제어시스템에 활용하여 장비의 API가 제어 및 모니터링 소프트웨어에 끼치는 영향을 어느 정도 줄일 수 있다. 그러나 앞서 언급한 파일들은 내용이 방대하고 복잡해서 전문적인 지식이 없는 사람들은 사용하기 어렵고, 각각의 필드버스의 명세를 따르고 있어 범용적으로 사용되는 데는 한계가 있다. 그래서 좀더 쉽게 사용할 수 있고 범용적으로 사용되어질 수 있도록 하기 위해서 XML을 이용해서 디바이스를 기술하는 방법에 대한 연구가 시작되게 된다. 대표적인 것으로 CANOpen의 시스템을 기술하는 CoML(CANOpen Markup Language)[8]과 NASA의 IML(Instrument Markup Language)[9]이 있다.

2.2 EDS(Electornic Data Sheet)와 CoML(CANOpen Markup Language)

2.2.1 EDS(Electornic Data Sheet)

EDS는 CANOpen 필드버스에 사용되어지는 디바이스들을 제어하고 모니터링하기 위한 소프트웨어 도구에서 사용하는 표준화된 디바이스 기술 파일 포맷이다. EDS를 구성하고 있는 요소에는 File Information, General Device Information, 그리고 Object Dictionary가 있다. 먼저 File Information은 EDS파일 자체에 대한 정보를 기술하고 있는 문서로써 EDS파일의 버전 정보 관리에 유용하게 사용되어 진다. General Device Information은 각 디바이스의 특징적인 정보들을 기술하는 부분으로, 이 부분에는 디바이스의 ID, 데이타 전송률 등이 기술되어 진다. Object Dictionary는 디바이스를 객체로 표현했을 때의 정보들을 기술하는 부분으로, 객체의 ID, 파라미터, 데이타 형식 등이 기술되어 진다. 소프트웨어 도구에서는 이렇게 작성되어진 EDS 파일을 이용해서 다양한 장비들을 쉽게 제어할 수 있게 된다. 그러나 EDS파일은 그 내용이 방대하고 복잡해서 문서를 작성하기 위해서는 디바이스와 EDS에 대해서 전문적인 지식이 필요하고, 디바이스와 소프트웨어 도구가 필드버스 프로토콜과 플랫폼에 종속적이어서 사용하기에도 어려움이 많았다. 그래서 필드버스 시스템과 프로세스 데이타를 일반적이고 플랫폼에 독립적으로 접근하고 제어할 수 있도록 하기 위한 연구가 시도되었고, 그 결과 CoML이 등장하게 되었다.

2.2.2 CoML(CANOpen Markup Language)

CoML은 CAN 필드버스 시스템과 필드버스 내의 프로세스 데이타를 표현하기 위한 일반적이면서 플랫폼에 독립적으로 표현하기 위해 개발되어진 XML 어플리케이션이다. CoML은 CANOpen 시스템 정보와, 프로세스 데이타, 그리고 모니터링 속성과 접근권한을 XML로 표현하기 위한 마크업 언어이다. CoML은 크게 Module, CanSetup, StateInformation의 세 가지 요소로 구성된다. Module요소는 EDS 파일을 XML 형태로 기술하는 부분으로 하위 요소로 EdsFileInfo, DeviceInfo, Object-Dictionary를 가지며, 이 세 가지 요소는 EDS의 세 가지 구성요소를 표현한다. CanSetup 요소는 전체 자동화 시스템을 표현하는 부분으로 CANOpen 시스템이 어떤 디바이스로 어떤 구조로 이루어져있는지에 대한 정보를 기술한다. StateInformation 요소는 Module요소에서 포함하지 못하는 파라미터 값들을 표현한다. CoML을 사용하면 플랫폼이나 프로그램 언어로부터 독립적으로 시스템을 구성하는 것이 가능해지고, 시스템의 유연성이 높아지게 된다. 그러나 결국 CoML도 CANOpen이라는 필드버스 프로토콜 내에서 사용되어 지는 것이므로 다른 디바이스에는 적용될 수 없다는 한계가 존재한다.

2.3 IML(Instrument Markup Language)

IML은 장비에 대한 정보를 기술하는 XML 기반의 마크업 언어이다. IML에 기술되어지는 내용에는 장비의 커멘드 집합, 커멘드 파라미터, 파라미터의 데이타 타입과 유효한 값과 범위, 데이타 필드의 데이타 형식과 포맷, 통신 메커니즘 등이 있다. 제어 시스템에서는 IML을 이용해서 장비와의 통신을 설정하고 장비를 제어하고 모니터링 하게 된다. 그러므로 API가 바뀌더라도 제어시스템에서는 API를 교체하는 것이 아니라 IML문서를 교체함으로써 변경된 사항을 제어 시스템에 적용시킬 수가 있게 된다. 이를 통해 일반적인 장비들의 제어와 모니터링에 있어서 API가 제어 시스템에 끼치는 영향이 최소화 되어지게 된다. IML은 천문학이나 생태학, 또는 제조업 등의 다양한 분야에 적용될 수 도 있으며 IML을 천문학 분야에 적용을 시킨 것이 AIML(Astronomical Instrument Markup Language)이다. AIML은 NASA에서 천체 관측에 사용되어 지는 장비들(천체 망

원경, 카메라 등등)을 원격에서 제어하고 측정 결과를 모니터링 하기 위하여 IML을 확장하여 정의되어졌다. AIML은 IML의 기본적인 요소들에 천체 관측에 사용되는 장비들을 기술하기 위한 요소들을 포함하여 정의되었다. AIML은 확장이 용이한 XML의 특성을 반영한 결과라고 할 수 있다.

본 프레임웍에서 제안하는 IID와 CMIML은 IML을 기반으로 재정의 되었다. IML은 장비에 대한 인터페이스와 소프트웨어에 대한 기술을 하나의 통합된 문서로 기술한다. 본 논문에서는 장비에 대한 인터페이스는 IID에서 기술하고, 소프트웨어는 CMIML에서 기술하도록 분리해서 장비 인터페이스와 소프트웨어간의 독립성을 제공한다. 그리고 CMIML에 프로그램 언어가 가지는 기본적인 요소인 제어문(조건문, 반복문)에 대한 기술과 장비의 연속동작(본 논문에서는 스케줄이라 함)에 대하여 기술할 수 있는 요소를 추가하여 소프트웨어 기술 방법을 확장하였다. 또한 장비 인터페이스와 실제 API 간의 연결정보를 기술하는 MAP 문서를 정의함으로써 기술 문서와 실제 구현과의 종속성을 최소화하도록 하였다.

3. 제어 및 모니터링 소프트웨어 자동생성 프레임웍

본 장에서는 먼저 제안하는 프레임웍의 전체적인 구성에 대해서 설명하고, 프레임웍의 관련자별 역할과 제어 및 모니터링 소프트웨어 생성 과정에 대해서 설명하도록 하겠다. 다음으로 프레임웍을 구성하는 세 가지의 표준문서와 구성 시스템에 대해서 설명하도록 하겠다.

3.1 프레임웍 구성

본 논문에서 제안하는 프레임웍은 그림 1과 같이 세 가지 표준문서형식(IID, MAP, CMIML), VI Wizard, Generator로 구성된다.

VI Wizard는 표준문서형식(IID, CMIML)을 내장하고, 입력받은 IID(Instrument Interface Description)와 기 작성된 재사용 모듈인 CMIML을 이용해서 그림 1과 같이 GUI 기반으로 VI(Virtual Instrument)와 UI(User Interface)를 구성하고, 새롭게 구성된 VI와 UI를 이용하여 CMIML(Control & Monitoring Instrument Mark-up Language)을 생성한다. Generator는 표준문서형식(CMIML, MAP)을 내장하고, VI Wizard에서 생성한 CMIML 문서와 MAP 문서를 입력으로 장비에 대한 제어 및 모니터링 소프트웨어를 자동 생성하게 된다.

그림 1에서 보는바와 같이 VI Wizard에서 생성된 CMIML은 IID와 함께 재사용되어 질 수 있다. 본 프레임웍에서의 재사용은 Generator에 의해 생성되는 플랫폼 종속적인 코드에 대한 재사용이 아닌 플랫폼에 독립

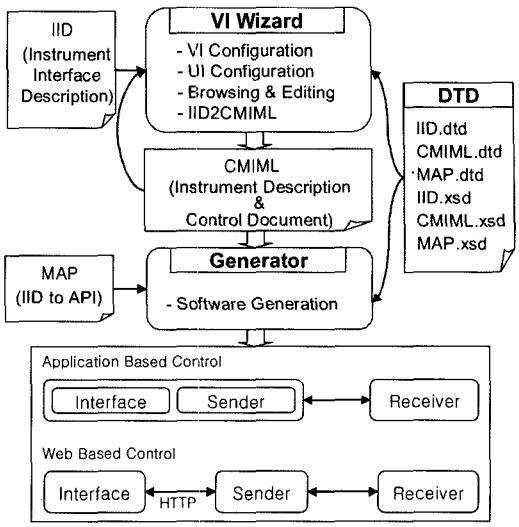

그림 1 CMIML Framework Architecture

적인 제어 및 모니터링 소프트웨어를 기술하는 문서 재사용을 의미한다. 플랫폼에 종속적인 코드 재사용에 비해서 플랫폼에 독립적인 기술 문서 재사용은 재사용성을 향상시킬 수 있다. 또한 VI Wizard를 이용해서 생성된 CMIML은 다양한 Generator에 의해서 다양한 플랫폼에 맞는 실제 코드로 생성되어 질 수 있다. 이것은 XML이 가지는 장점으로 하나의 XML 문서가 다양한 뷰어에 의해서 다양한 형태로 보여 질수 있다는 것과 흡사하다고 할 수 있다.

그림 1의 MAP 문서는 장비에 대한 인터페이스와 실제 API간의 연결 정보를 기술하는 것이다. 따라서 API가 변경되어도 소프트웨어를 수정 또는 재개발 할 필요 없이 MAP 문서에서 변경된 API에 대한 정보만 수정하면 된다. 이는 유지보수에 대한 비용을 감소할 수 있을 것이다.

3.2 프레임웍 관련자별 역할 및 제어 및 모니터링 소프트웨어 생성 과정

본 논문에서 제안하는 프레임웍의 관련자에는 프레임웍 구축을 위한 관련자와 프레임웍을 사용하는 관련자가 있다. 그림 2의 (a)는 프레임웍의 구축에 관여하는 관련자를 나타내고 그림 2의 (b)는 프레임웍 사용자에 대한 역할을 나타내고 있다. 그림 2의 (c)는 본 논문에서 제안하는 프레임웍을 사용하여 제어 및 모니터링 소프트웨어를 생성하는 과정을 보이고 있다.

먼저 장비 개발자가 장비에 대한 IID 문서를 작성하고 장비 드라이버 개발자가 장비의 인터페이스를 분석해서 MAP 문서를 작성한다. 장비 소프트웨어 개발자는 VI Wizard를 이용하여 IID 문서와 재사용 가능 모듈(기 작성된 CMIML)을 입력으로 VI와 UI를 구성하고, 새롭게

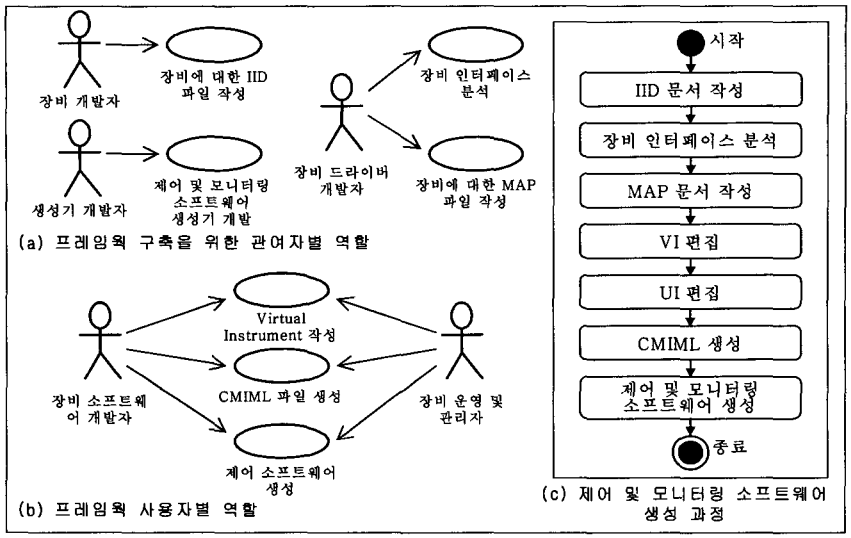

그림 2 관련자별 역할 및 제어 및 모니터링 소프트웨어 생성 과정

구성된 VI에 대한 CMIML 문서를 생성한다. 소프트웨어 생성기(Generator)는 VI Wizard에 의해 생성된 CMIML 문서와 MAP 문서를 입력으로 제어 및 모니터링 소프트웨어 코드를 생성해 낸다. 본 프레임웍의 VI Wizard와 Generator는 GUI 환경을 제공하면서 제어 소프트웨어 개발에 대한 전문적인 지식 없이도 쉽게 제어 및 모니터링 소프트웨어를 개발할 수 있도록 한다. 따라서 전문적인 장비 소프트웨어 개발자가 아닌 장비 운영 및 관리자도 쉽게 제어 소프트웨어를 개발할 수 있다.

3.3 프레임웍을 구성하는 기술 문서

본 논문에서 제안하는 프레임워크에서는 제어 및 모니터링 소프트웨어에 대한 자동생성을 위하여 세가지(IID, MAP, CMIML)의 XML 문서를 작성한다. IID는 장비에 대한 행위정보와 상태정보를 나타내는 인터페이스 정보를 기술하고, MAP은 IID에 기술된 인터페이스와 실제 장비 드라이버 API 또는 Address Map의 연결 정보를 기술한다. 마지막으로 CMIML은 작성된 IID 파일을 이용해서 제어정보, 사용자 인터페이스, 모니터링 정보, 통신방법 등에 대한 정보를 기술한다. 초기에는 세가지 XML 문서들에 대한 형식을 XML DTD로 기술하였으나 현재는 XML Schema도 정의하고 있다. XML Schema로 정의함으로써 다양한 데이타형의 표현이 가능하고, 유연성, 확장성 등의 장점을 가진다. XML Schema에 비해 XML DTD가 이해하기 쉬우므로 본 논문에서는 각 문서들에 대한 XML DTD를 중심으로 주요 요소들과 기능들을 세부적으로 설명하도록 하겠다.

3.3.1 장비 인터페이스 기술문서 - IID(Instrument Interface Description)

장비 개발자는 장비의 동작정보와 상태 정보를 정의한 후 IID 문서를 작성한다. IID는 유효한 XML문서이어야 하고 장비를 제어하기 위한 인터페이스정보(동작정보, 상태정보)를 기술해야 한다. 그림 3은 IID에 대한 DTD를 Wattle Software[10]의 XMLWriter 2.0을 이용하여 전체 구조를 보인 것이다.

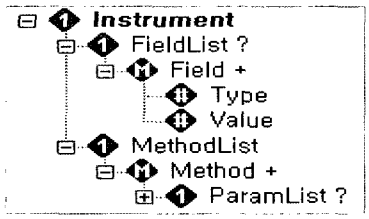

그림 3 The IID DTD Overview

IID의 최상위 요소는 Instrument이고 하위에 장비의 상태정보를 표현하기 위한 FieldList 요소와 장비의 동작정보를 표현하기 위한 MethodList 요소를 가진다.

다음은 IID문서 DTD의 MethodList에 대한 DTD 정의다.

```
<!ELEMENT MethodList (Method)+>
<!ELEMENT Method (ParamList)?>
<!ATTLIST Method Name CDATA #REQUIRED
        Desc CDATA #IMPLIED
        RetType CDATA #REQUIRED>
<!ELEMENT ParamList (Param)+>
<!ELEMENT Param (Name, Type)+>
<!ATTLIST Param Desc CDATA #IMPLIED>
```

IID의 실제 예는 다음에 보여진다.

```
<!DOCTYPE Instrument SYSTEM "IID.dtd">
<Instrument Name="Robot" Desc="Simple Robot made by RCX">
  .....
  <MethodList>
    <Method Name="forward" RetType="void" Desc="Causes to rotate forwards">
      <ParamList>
        <Param Desc="A period value">
          <Name>aPeriod</Name>
          <Type>int</Type>
        </Param>
      </ParamList>
    </Method>
    .....
  </MethodList>
</Instrument>
```

이 예제(rcxrobot.iid)는 LEGO Mindstorms로 제작한 RCXRobot의 인터페이스를 기술하고 있다. RCX-Robot이 수행하는 동작을 기술하기 위해 <Method>태그를 사용하였고, 동작이 수행될 때 필요한 파라미터를 기술하기 위해 <Param>태그를 사용하였다.

3.3.2 장비와 실제 API의 연결 정보를 기술 - MAP(IID to API)

장비 드라이버 개발자는 장비의 API를 개발한 후 장비의 인터페이스(상태정보 및 동작정보)와 API를 연결시키기 위해서 유효한 XML 문서인 MAP파일을 작성한다. 그림 4는 MAP 문서의 전체 구조를 보여준다.

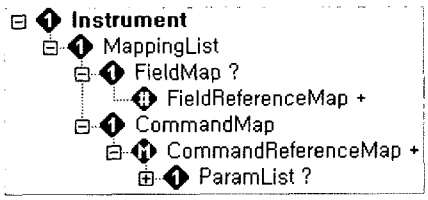

그림 4 The MAP DTD Overview

MAP문서는 상태정보 연결을 위한 FieldMap 요소와 동작정보 연결을 위한 CommandMap 요소를 가진다.

FieldMap은 하나 이상의 FieldReferenceMap을 가질 수 있다. FieldReferenceMap은 IID에 기술되어 있는 상태정보를 나타내는 FieldReference 속성과 실제로 연결될 API의 변수명을 나타내는 MapTo 속성을 가진다.

CommandMap은 하나 이상의 CommandReferenceMap을 가질 수 있다. CommandReferenceMap은 IID에 기술되어 있는 동작정보를 나타내는 CommandReference와 실제로 연결될 API의 함수명을 나타내는 MapTo 속성을 가지고, 함수가 가지는 파라미터 정보를 나타내기 위해서 ParamList 요소를 하위요소로 가질 수 있다.

다음은 MAP문서 DTD의 CommandMap에 대한 DTD 정의다.

```
<!ELEMENT CommandMap (CommandReferenceMap)+>
<!ELEMENT CommandReferenceMap (ParamList)?>
<!ATTLIST CommandReferenceMap
      CommandReference CDATA #REQUIRED
      MapTo CDATA #REQUIRED>
<!ELEMENT ParamList (Parameter)+>
<!ELEMENT Parameter ANY>
<!ATTLIST Parameter Type CDATA #REQUIRED
          Value CDATA #REQUIRED>
```

MAP의 실제 예는 다음에 보여진다.

```
<!DOCTYPE Instrument SYSTEM "MAP.dtd">
<Instrument Name="Robot" Desc="Motor A or B or C">
  <MappingList>
    <CommandMap>
      <CommandReferenceMap CommandReference="forward" MapTo="RCXRobot.forward"/>
      .....
    </CommandMap>
  </MappingList>
</Instrument>
```

이 예제(rcxrobot.map)는 4.1에서 정의한 rcxrobot.iid의 인터페이스와 leJOS의 API 간의 연결에 대해서 기술하고 있다. <CommandReferenceMap>태그의 Command-Reference속성은 IID에서 기술된 인터페이스 정보를 나타내고, MapTo속성은 leJOS의 API를 나타낸다. 이 예제에서는 rcxrobot.iid의 forward 인터페이스가 leJOS의 RCXRobot.forward()와 연결이 된다는 것을 나타내고 있다.

3.3.3 소프트웨어를 기술 - CMIML(Control & Monitoring Instrument Markup Language)

VI Wizard를 통해서 생성되는 CMIML은 장비에 대한 인터페이스 정보를 기술하고 있는 IID의 확장이라고 볼 수 있다. IID가 단순히 장비의 인터페이스정보만을 기술하고 있는 반면 CMIML은 IID에 기술되어 있는 인터페이스정보에 제어 및 모니터링 소프트웨어의 사용자 인터페이스 정보, 장비의 모니터링 정보, 장비와 PC사이

의 통신 방법, 장비의 스케줄 정보까지 기술하고 있다.

그림 5는 CMIML문서의 전체 구조를 보여준다.

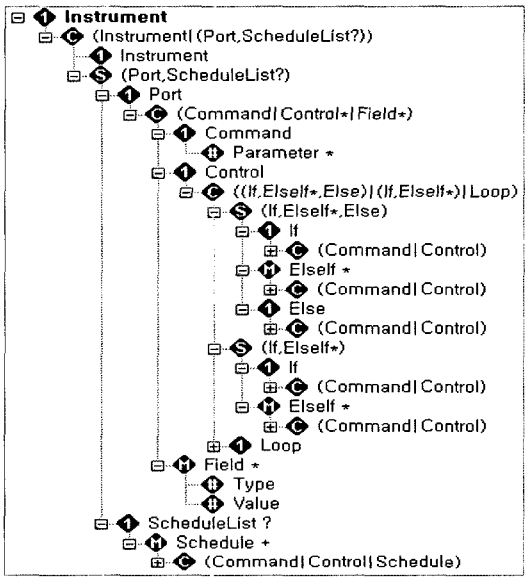

그림 5 The CMIML DTD Overview

CMIML의 최상위 요소는 Instrument이고, 설비는 여러 장비가 모여 구성될 수 있으므로 Instrument 요소는 다시 Instrument를 하위요소로 가질 수 있다. 그리고 하나의 장비는 통신 방법을 정의하는 Port 요소와 장비에 대한 일련의 동작을 기술하는 ScheduleList 요소를 하위 요소로 가진다.

다음은 CMIML문서 DTD의 ScheduleList에 대한 DTD 정의다.

```
<!ELEMENT ScheduleList (Schedule)*>
<!ELEMENT Schedule (Command|Control|Schedule)+>
<!ATTLIST Schedule Name #REQUIRED
    UI CDATA #REQUIRED
    UI_IDS CDATA #IMPLIED
    UI_IDE CDATA #IMPLIED>
```

CMIML의 실제 예는 다음에 보여진다.

```
<Instrument Name="Robot" Desc="RCXRobot">
....
  <ScheduleList>
    <Schedule Name="Sample1" UI="False" UI_IDS="0"
    UI_IDE="0">
      <Command Name="Robot.forward" RetType="void"
      Val="" Desc="Causes robot to rotate forward"
      UI="False" UI_ID="0">
        <Parameter Name="aPeriod" Desc="A period value"
        Type="int" Value="2" UI="False" UI_ID="0">
        </Parameter>
      </Command>
      .....
    </Schedule>
  </ScheduleList>
</Instrument>
```

이 예제(sample1.cmiml)는 VI Wizard를 이용해 구성한 VI로부터 생성한 CMIML을 보여준다. <Port>태그는 현재 RCXRobot이 적외선 통신을 하고 있기 때문에 기술하지 않는다. RCXRobot이 연속적인 동작을 수행하게 하기 위해 <Schedule>태그를 사용하고 그 하위에 순차적으로 수행되는 동작들에 대한 정보를 기술한다. 각 동작들을 기술하기 위해 <Command>태그를 사용하고, 동작들이 수행될 때 필요한 파라미터 정보를 기술하기 위해 <Parameter>태그를 사용한다.

3.4 프레임웍을 구성하는 시스템

본 논문에서 제안하는 프레임워크는 CMIML을 생성하기 위한 VI Wizard와 소프트웨어 코드를 생성하기 위한 Generator로 구성된다.

3.4.1 VI Wizard

VI Wizard는 장비에 대한 인터페이스 정보를 기술한 IID파일을 이용해서 GUI 기반의 VI 편집 환경을 제공하고, 새롭게 구성된 VI를 CMIML로 변환하는 도구이다. 생성된 CMIML은 Code Generator에 의해 제어 소프트웨어로 생성되어 운용되거나 다른 VI 구성을 위해 재사용될 수 있다.

그림 6은 VI Wizard의 세부적인 시스템 구조이다. 그림 6과 같이 VI Wizard는 4개의 모듈, 3개의 매니저와 2개의 데이타저장소로 구성된다.

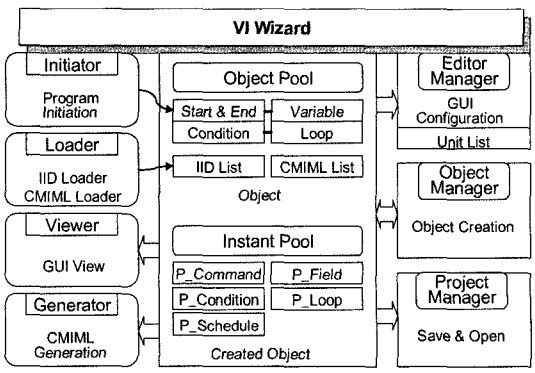

그림 6 VI Wizard 시스템 구성도

3.4.2 Generator

Generator는 제안한 프레임웍의 VI Wizard에 의해서 생성된 CMIML 문서를 이용해서 GUI 기반의 사용자 인터페이스 편집 환경을 제공하고, CMIML을 소프트웨어 코드로 자동 생성하는 도구이다. Generator를 통해서 장비에 대한 전문적인 지식이 없이도 제어 소프트웨어를 생성할 수 있게 된다.

그림 7은 Generator의 세부적인 시스템 구조도를 보이고 있다. 그림 7과 같이Generator는 3개의 모듈과 1개의 데이타 저장소로 구성된다.

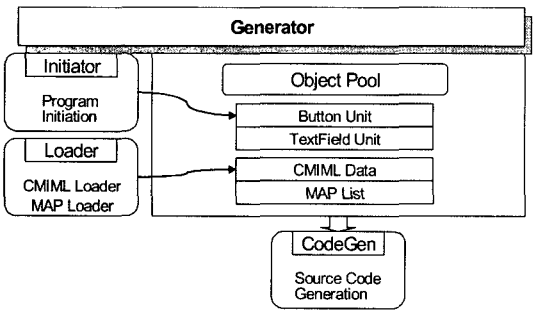

그림 7 Generator 시스템 구성도

4. 시스템 구현 및 평가

본 장에서는 제안하는 프레임웍을 구성하는 핵심적인 도구인 VI Wizard와 Generator의 구현과 프레임웍의 사용 예를 보이고, 과거 연구와의 비교를 통한 평가를 기술하겠다.

4.1 시스템 구현

VI Wizard와 Generator는 모두 Visual Basic 6.0 환경에서 XML 파서로 MSXML 3.0을 사용하여 구현하였다.

그림 8과 9는 LEGO Mindstorms[11]의 RCXRobot을 제어하는 제어 코드를 생성하는 화면이다. RCX-Robot의 주요 구성요소는 RCXTM Microcomputer, 2개의 모터, 2개의 터치센스 등이며, PC와 적외선 통신을 통해서 구동된다. 우리는 RCXTM Microcomputer에 자바 기반의 운영체제인 leJOS[12]를 탑재하고 leJOS에서 제공하는 API를 사용하였다.

그림 8은 VI Wizard의 실행 화면이다. 그림 8의 (a)는 Initiator와 Loader에 의해 Object Pool에 등록된 객체들을 그룹별로 보여준다. (b)는 선택된 객체들의 속성 정보를 보여준다. (c), (d), (e)는 Editor Manger에 의해 GUI 기반으로 편집된 VI를 나타낸다. (d)는 조건문, (e)는 반복문이 수행하는 내부적인 동작을 나타낸다. (f)는 CMIML Generator에 의해 새롭게 구성된 VI를 CMIML로 변환된 문서다.

현재 Generator는 LEGO Mindstorms의 RCXRobot을 제어하기 위한 JAVA 기반의 소스코드를 생성하는 것과 AT89C4051 마이크로 프로세스를 탑재한 4관절 ArmRobot을 제어하기 위한 C 기반의 소스코드를 생성하는 것이 개발된 상태이다.

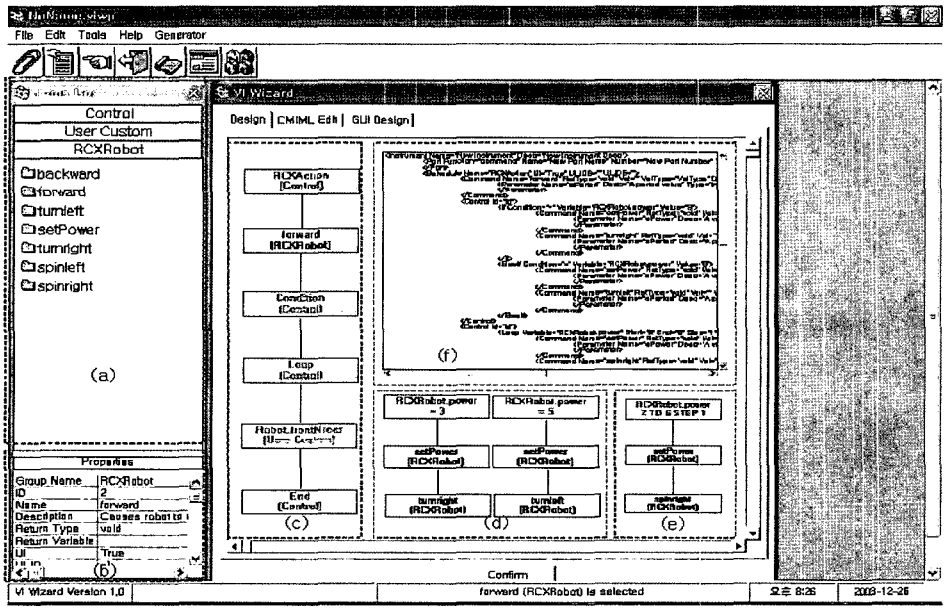

그림 8 VI Wizard 실행화면

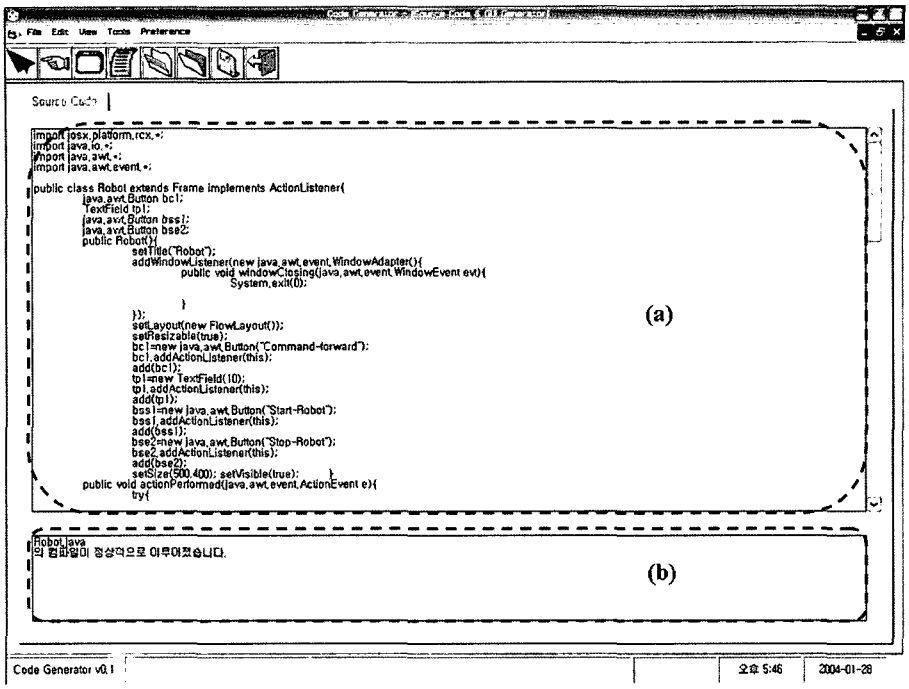

그림 9 Generator의 실행화면

그림 9는 RCXRobot 제어에 대한 Java 소스코드 생성하는 Generator의 실행 화면이다. 그림 9의 (a)는 Loader Module(CMIML Loader, MAP Loader)에 의해서 읽혀진 CMIML 문서와 MAP 문서의 데이타를 기반으로 생성해낸 제어 소프트웨어의 소스코드를 보여준다.

그림 9의 (b)는 생성된 소스코드를 외부 컴파일러를 통해 컴파일 했을 때의 결과를 보여주는 화면이다.

4.2 사용 예

우리는 LEGO Mindstorms로 제작한 RCXRobot과 AT89C4051 마이크로 프로세스를 탑재한 4관절 Arm-Robot에 우리가 제안하는 프레임웍을 적용하여 보았다. RCXRobot과 ArmRobot에 대한 IID 문서와 MAP 문서를 정의한 후 VI Wizard를 이용하여 VI를 구성하고 CMIML을 생성하였다. 우리는 RCXRobot 제어를 위한 Java 코드를 생성하는 Generator와 ArmRobot 제어를 위한 C 코드를 생성하는 Generator를 개발한 후 VI Wizard를 이용해서 생성한 CMIML을 제어 소프트웨어 코드로 변환하였다.

4.2.1 LEGO Mindstorms의 RCXRobot에 적용

LEGO Mindstorms로 제작한 RCXRobot를 제어하기 위한 몇 가지 테스트를 수행하였다. 이 테스트에는 본 논문의 3.3.1과 3.3.2에서 예를 든 rcxrobot.iid와 rcx-robot.map을 사용하였다. 그림 10은 RCXRobot의 제어를 위해 VI Wizard로 구성한 VI를 보여준다.

그림 10의 (a)는 RCXRobot의 연속적인 동작을 위해 구성한 간단한 VI이다. (a)의 VI는 CMIML로 변환된 후 다른 VI에 의해 재사용되는 것이 가능하다. (b)는 (a)의 VI를 재사용하고 있는 모습을 보여준다. (b)의 RCXRobot.Sample1은 (a)의 VI를 CMIML로 저장한 후, (b)의 VI에서 읽어 들여 추출한 스케줄 정보를 나타낸다. (c)는 (a)의 VI를 읽어 들여서 재구성한 VI를 보여준다. 이렇게 구성되어진 VI는 CMIML로 변환되어진 후 Generator에 의해서 RCXRobot을 제어하기 위한 소스코드로 변환되어진다.

다음의 표 1은 그림 10의 (a), (b), (c)의 VI를 가지고 생성한 CMIML을 간략히 보여준다.

표 1의 (1)은 그림 10의 VI (a)로부터 생성된 CMIML을 보여준다. (2)와 (3)은 각각 그림 10의 VI (b)와 (c)로부터 생성된 CMIML을 보여준다. (1)은 (2)에서 재사용되어졌고 (3)에서 재구성되어졌다.

다음의 표 2는 표 3의 각 CMIML이 Generator에 의해서 생성된 제어 소프트웨어 코드를 보여준다.

표 2의 (A)는 표 1의 CMIML (1)을 이용해서 생성한 제어 소프트웨어 코드이다. CMIML의 <Schedule> 태그는 하나의 독립된 메소드로 표현이 되고, <Command>태그는 MAP파일(rcxrobot.map)에 기술된 해당 API로 기술된다. 파라미터를 가지는 API라면 <Para-

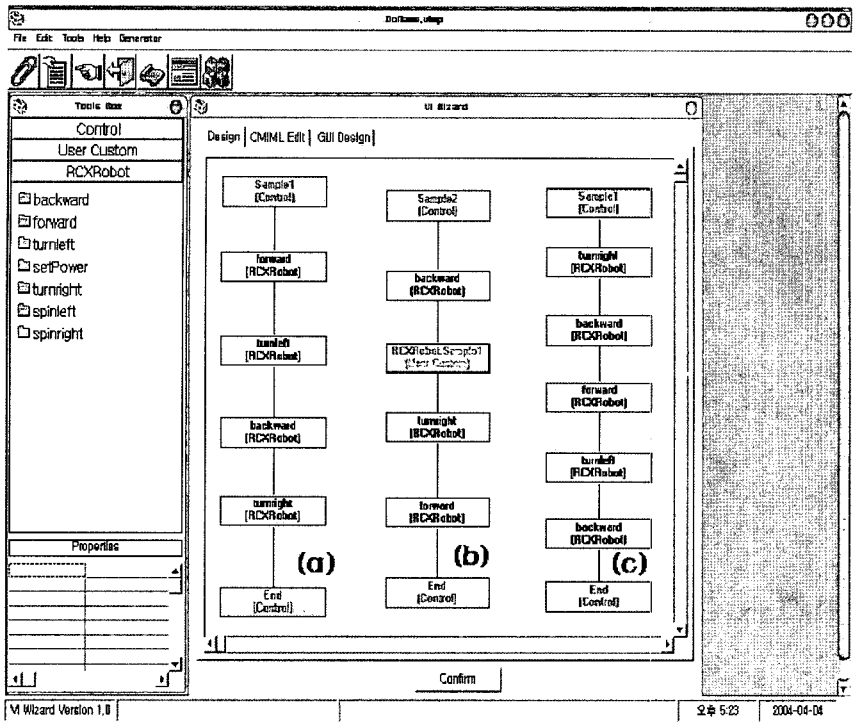

그림 10 RCXRobot의 제어를 위한 VI 편집 화면

표 1 VI로부터 생성된 CMIML

(1)

```
<Instrument Name="Robot"
Desc="RCXRobot">
  .....
  <ScheduleList>
    <Schedule Name="Sample1"
    UI="False" UI_IDS="0"
    UI_IDE="0">
      <Command Name=
      "Robot.forward" ...>
        <Parameter Name="aPeriod" ...>
        </Parameter>
      </Command>
      .....
    </Schedule>
  </ScheduleList>
</Instrument>
```

(2)

```
<Instrument Name="Robot"
Desc="RCXRobot">
  .....
  <ScheduleList>
    <Schedule Name="Sample2"
    UI="False" UI_IDS="0"
    UI_IDE="0">
      <Command Name=
      "Robot.backward" ...>
        .....
      </Command>
      <Schedule Name="Sample1"
      UI="False" UI_IDS="0"
      UI_IDE="0">
        .....
      </Schedule>
      .....
    </Schedule>
  </ScheduleList>
</Instrument>
```

(3)

```
<Instrument Name="Robot"
Desc="RCXRobot">
  .....
  <ScheduleList>
    <Schedule Name="Sample1"
    UI="False" UI_IDS="0"
    UI_IDE="0">
      <Command Name=
      "Robot.turnright" ...>
        <Parameter Name="aPeriod" ...>
        </Parameter>
      </Command>
      .....
    </Schedule>
  </ScheduleList>
</Instrument>
```

meter>태그의 Value속성이 순서대로 기술된다. (B)와 (C)는 표 1의 CMIML (2)와 (3)을 이용해서 생성한 코드이다.

4.2.2 AT89C4051을 탑재한 4관절 ArmRobot에 적용

ArmRobot에 제안하는 프레임웍을 적용하기 위하여 ArmRobot의 인터페이스를 기술하는 armrobot.iid와 실제 API와 인터페이스에 대한 연결정보를 가지는 armrobot.map을 정의하였다. 그림 11은 ArmRobot의 제어를 위해 VI Wizard로 구성한 VI를 보여준다.

표 2 생성된 제어 소프트웨어 소스 코드 (RCXRobot)

(A)

```
import josx.platform.rcx.*;
import java.io.*;
public class Robot{
  public static void main(String args[])
  throws Exception{
    Sample1();
  }
  public static void Sample1() throws
  Exception{
    RCXRobot.forward(2);
    .....
  }
}
```

(B)

```
import josx.platform.rcx.*;
import java.io.*;
public class Robot{
  public static void main(String args[])
  throws Exception{
    Sample2();
  }
  public static void Sample2() throws
  Exception{
    RCXRobot.backward(5);
    Sample1();
    .....
  }
  public static void Sample1() throws
  Exception{.....}
}
```

(C)

```
import josx.platform.rcx.*;
import java.io.*;
public class Robot{
  public static void main(String args[])
  throws Exception{
    Sample1();
  }
  public static void Sample1() throws
  Exception{
    RCXRobot.turnright(5);
    .....
  }
}
```

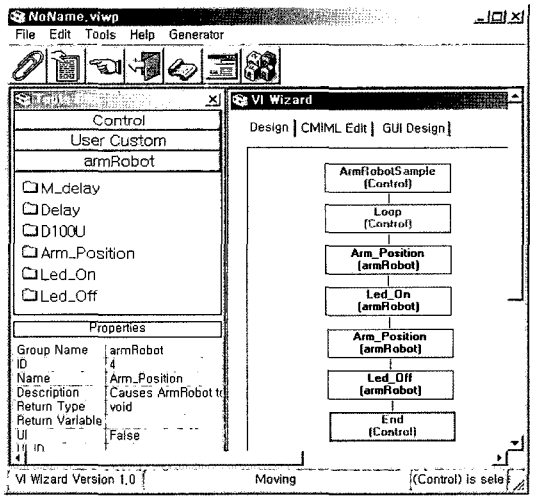

그림 11 ArmRobot의 제어를 위한 VI 편집 화면

표 3 생성된 제어 소프트웨어 소스 코드(ArmRobot)

```
#include "ArmRobot.h"
void main(void)
{
        BYTE i;
        for(int i=1;i>2;i++){
                led_on();
                mdelay(200);
                led_off();
                mdelay(200);
        }
        led_on();
        armposition(20, 10, 20, 10, 100);
        armposition(30, 20, 30, 20, 100);
        led_off();
}
```

앞의 표 3은 그림 11의 VI를 CMIML로 변환한 후 ArmRobot용 C 코드 Generator를 이용해서 생성한 제어 소프트웨어 코드이다.

4.3 평가

다음의 표 4는 앞에서 이미 설명한 관련연구들과 본 논문에서 제안하는 프레임웍의 장단점들에 대하여 요약 정리한 것이다.

다음의 표 5는 관련연구들과 본 논문에서 제안하는 프레임웍을 5가지의 기준에 따라 비교한 것이다.

본 논문에서 제안하는 프레임웍은 GUI 기반의 VI 편집 환경을 제공함으로써 저수준 프로그래밍 방식에 비해서 쉽고 빠른 개발과 유지보수를 지원하는 장점을 가지면서, 이런 도구들의 단점으로 볼 수 있는 제어 및 모니터링 시스템이 장비 제조업체에서 제공하는 장비와 장비에 대한 API에 종속적인 문제를 해결하기 위하여 EDDL, GSD, EDS와 같이 장비에 대해 기술하는 문서를 작성하고 그 문서를 통해서 장비의 제어와 모니터링을 수행하도록 하는 방법을 채택 했다. 장비를 기술하는 문서 형식에 있어 EDDL, GSD, EDS 등이 내용이 방대하고 복잡하여 전문적인 지식이 필요하고 Profibus와 CANOpen과 같은 특정 필드버스의 명세를 따르고 있어 범용적인 사용에 한계가 있다는 점에 착안하여 CoML과 IML과 같이 XML 기반의 기술 문서를 사용하도록 하였다. 하지만 CoML은 여전히 CANOpen이라는 필드버스 프로토콜 내에서 사용되어 범용적인 사용에 한계가 있다. IML은 특정 플랫폼에 종속적이지 않아 범용적으로 사용되어 질 수 있지만 장비에 대한 인터페이스와 소프트웨어에 대한 기술을 하나의 통합된 문서로 기술되어 장비 인터페이스와 소프트웨어간의 종속성을 가진다. 본 논문에서는 장비에 대한 인터페이스를 기술하는 문서와 소프트웨어를 기술하는 문서를 IID와 CMIML로

표 4 관련연구들의 장단점 비교

구분	장점	단점
EDDL GSD EDS	1)장비에 대한 API에 종속적인 면을 어느 정도 해소	1)내용이 방대하고 복잡 2)전문적인 지식이 필요 3)각각의 필드버스의 명세를 따르고 있어 범용적인 사용에 한계
CoML	1)XML 기반의 문서 형식 사용 2)EDS에 비해 쉬운 사용	1)CANOpen 필드버스의 명세를 따르고 있어 범용적인 사용에 한계
IML	1)XML 기반의 문서 형식 사용 2)플랫폼 독립성(범용성)을 제공 3)구현과 기술의 분리	1)장비 인터페이스와 소프트웨어에 대하여 단일 문서로 기술되어 유연성이 부족 2)제어문(조건문, 반복문)에 대한 표현 방법이 없음 3)연속동작(스케줄)에 대한 기술이 안됨 4)실제 API에 대한 기술 문서 없음
본 논문에서 제안하는 프레임웍	1)XML 기반의 문서 형식 사용 2)플랫폼 독립성(범용성)을 제공 3)GUI 기반의 쉬운 개발,유지보수,확장성 4)IML에서 장비기술과 소프트웨어 기술을 분리 5)제어문과 연속동작에 대한 기술 6)장비와 실제 API의 매핑정보 기술 7)문서 재사용으로 재사용성 향상 8)구현과 기술의 분리	1)소규모의 시스템에 적용하기에 적당하나 대규모의 분산 제어 시스템에 적용 곤란 2)소프트웨어 자동생성 시스템의 한계 3)자동생성 소프트웨어는 Generator에 의존적

표 5 관련연구와의 비교

구분	EDDL GSD EDS	CoML	IML	본 논문에서 제안하는 프레임웍
기술문서형식	자체언어	XML	XML	XML
플랫폼 독립성	X	X	O	O
GUI 개발환경	X	X	X	O
제어문 지원	X	X	X	O
통합 개발 툴	X	X	X	O

분리하여 기술할 수 있도록 정의하였다. 그리고 프로그램 언어가 가지는 기본적인 요소인 제어문(조건문, 반복문)에 대한 기술과 장비의 연속동작에 대하여 기술할 수 있는 요소를 추가하여 소프트웨어 기술 방법을 확장하였다. 또한 장비 인터페이스와 실제 API간의 연결정보를 기술하는 MAP 문서를 정의함으로써 기술 문서와 실제 구현과의 종속성을 최소화하도록 하였다.

결과적으로 본 논문에서 제안하는 프레임웍은 GUI 기반으로 제어 및 모니터링 소프트웨어를 자동 생성함으로써 쉬운 개발과 유지보수성을 제공하고, XML 기반의 기술문서 사용으로 플랫폼 독립성을 제공하면서 범용적으로 사용할 수 있도록 한다. 또한 플랫폼 종속적인 코드 재사용이 아닌 플랫폼 독립적인 소프트웨어 기술 문서를 재사용함으로써 재사용성을 증가시킬 수 있다.

5. 결론 및 향후 연구과제

본 논문에서는 생산현장의 다양한 장비와 설비들의 제어 및 모니터링 소프트웨어에 대한 쉬운 개발과 유지보수성을 향상시킬 수 있는 프레임웍을 제안하였다. 본 논문에서 제안한 프레임웍은 세 가지의 XML문서(IID, MAP, CMIML), VI Wizard, Generator로 구성되었다.

IID는 장비에 대한 인터페이스정보를 기술하며 MAP은 IID의 인터페이스와 실제 API의 연결 정보를 기술하고 CMIML은 장비의 제어정보 및 사용자 인터페이스 정보, 모니터링 정보, 장비와 PC사이의 통신방법, 장비의 스케줄링 정보를 저장한다.

VI Wizard는 표준문서형식(IID, CMIML)을 내장하고, 장비의 인터페이스 정보를 기술하고 있는 IID 문서와 재사용 모듈로 사용되는 CMIML을 이용해서 GUI 기반으로 VI(Virtual Instrument)를 구성하고, 새롭게 구성된 VI를 이용하여 장비의 제어정보, 사용자 인터페이스 정보, 모니터링 정보, 통신 정보, 스케줄 정보 등을 기술하는 CMIML 문서를 생성한다.

Generator는 표준문서형식(MAP, CMIML)을 내장하고, VI Wizard에서 생성한 CMIML과 입력받은 MAP 문서를 이용해서 장비에 대한 모니터링 소프트웨어를 생성한다.

VI Wizard와 Code Generator를 이용하여 전문적인 지식(장비제어, 제어 소프트웨어, XML)없이도 자동으로 제어 소프트웨어를 생성함으로써 원격으로 제어되는 생

산 현장의 장비나 설비들에 적용될 수 있는 제어 소프트웨어를 적은 비용으로 쉽고 빠르게 개발하고 유지보수 할 수 있을 것이다. 또한 제안하는 프레임웍은 XML 기반의 문서 형식을 사용함으로 XML이 가지는 유연성과 확장성과 같은 다양한 장점을 충분히 활용하여 다양한 응용에 활용할 수 있다.

향후에는 본 논문에서 정의한 세 가지의 XML 문서들을 지속적인 연구를 통하여 다양한 제어 및 모니터링 소프트웨어에 대한 요구사항들을 반영할 수 있도록 확장하겠다. 그리고 다양한 플랫폼에 따른 표준적인 생성기 모듈에 대한 연구를 수행하여 자동 생성기 개발의 표준적인 방법과 다양한 사용자 인터페이스를 기술할 수 있는 방법에 대하여 연구하도록 하겠다. 또한 소프트웨어 코드의 변경을 최소화하면서 무정지 시스템을 가능하게 하는 실시간 참조 제약사항에 대한 정의와 자동 생성된 소프트웨어의 안정성을 검증할 수 있는 가상 시뮬레이션에 대한 연구가 필요하다. 그리고 현재 제안하는 프레임웍이 OPC(OLE for Process Control)[13]를 지원할 수 있도록 하겠다.

참 고 문 헌

[1] 구영재, 이준서, 이인범, 장근수, 'PC를 이용한 자동제어 시스템 개발', 한국자동제어학술회의논문집(KACC), pp. 322-326, 1991.

[2] 변승현, 마복렬, '대용량 플랜트 제어를 위한 PC 기반 I/O 인터페이스 시스템 구축에 관한 연구', 한국자동제어학술회의논문집(KACC), pp. b438-b441, 1999.

[3] 김정구, 최경현, 홍금식, 'PC에 기반을 둔 개방형 로봇 제어시스템:PC-ORC A PC-Based Open Robot Control System:PCORC', 제어.자동화.시스템공학 논문지, 제 6권, 제 5호, pp.415-425, 2000

[4] 박남준, 김홍석, 박종구, 'PC기반의 생산시스템을 위한 운용소프트웨어 구조', 제어.자동화.시스템공학 논문지, 제 7권, 제 1호, pp.50-58, 2001.

[5] Vol. 2: EDDL Specification; Specification for PROFIBUS Device Description and Device Integration "www.profibus.com"

[6] Vol. 1: GSD Specification; Specification for PROFIBUS Device Description and Device Integration "www.profibus.com"

[7] CiA DSP 306 V 1.1: CANopen electronic data sheet (EDS) specification for CANopen "www.canopen.org/canopen"

[8] Buhler Dieter, "The CANOpen Markup Language Representing Fieldbus Data with XML," Industrial Electronics Society, 2000. IECON 2000. 26th Annual Confjerence of the IEEE, Volume: 4, 22-28 Oct. 2000 Pages:2449-2454.

[9] Troy Ames, Lisa Koons, Ken Sall, Craig Warsaw, "Using XML and Java for Astronomical Instrumentation Control", Proc. of 6th Int'l Conf. on Space Operations(SpaceOps 2000), June 2000, France

[10] Wattle Software "http://xmlwriter.net/"

[11] LEGO Mindstorms home "http://mindstorms.lego.com/eng/default.asp"

[12] leJOS home "http://lejos.sourceforge.net/"

[13] OPC Foundation, "http://www.opcfoundation.org"

유 대 승

1998년 울산대학교 전자계산학 졸업 2001년 동 대학원 석사. 2003년 동 대학원 박사수료. 2005년~현재 울산대학교 컴퓨터정보통신공학부 객원교수. 관심분야는 소프트웨어 공학, 자동화 시스템, 웹 서비스

김 종 환

2003년 울산대학교 컴퓨터정보통신공학부 졸업. 2005년 동 대학원 석사. 2005년~현재 (주) 드림위즈. 관심분야는 컴포넌트 프로그래밍, 자동화 시스템

이 명 재

1987년 서울대학교 전산과학과 졸업 1989년 동 대학원 석사. 1995년 동 대학원 박사. 1996년~현재 울산대학교 컴퓨터정보통신공학부 부교수. 관심분야는 소프트웨어 공학, 공장 자동화