반도체 제조 장비를 위한 개방형 제어기 플랫폼 설계

A Design of an Open Architectural Controller Platform for Semiconductor Manufacturing Equipment

。장 성 진*, 김 홍 록*, 서 일 홍*

*한양대학교 전자공학과(Tel : +82-31-408-5802; Fax : +82-31-408-5803;
E-mail : ihsuh@email.hanyang.ac.kr)

Abstract : This paper presents some ideas about an open architectural controller platform for semiconductor manufacturing equipment. First, we proposed modular-typed software architecture. Each module is composed of commands and status sets. Second, common bus protocol is suggested in order to communicate with other modules. It is designed with visual c++ programming. Finally, job program is consisted of simple commands and status. Consequently, Controllers are easily developed with some required modular assembling.

Keywords : open architecture, controller, platform, module, protocol

1. 서론

현재의 시스템은 과거에 비해 더욱 복잡하고 다양화 되어 가고 있으며, 이들의 제어기 또한 지속적으로 빠르게 발전되어오고 있다. 특히, 전자부품 및 반도체 조립을 위한 장비는 산업기술의 발전이 가속화됨에 따라 점점 더 고정밀도의 높은 기술력을 요구하게 되었다. 이에, 전용 제어방식으로 시스템을 구성하면 높은 성능의 원하는 결과를 얻을 수는 있지만 변화되는 장비들마다 새로운 제어 시스템으로 구축하는 과정은 많은 시간과 비용을 요구하는 어려움이 있다. 따라서 동일한 연구투자로 활용도를 극대화 할 수 있는 방안인 PC기반의 개방형 제어기 플랫폼을 제안하고자 한다.

앞에서 언급한 PC기반의 개방형 제어기의 플랫폼은 모듈화 방식으로 설계하는 것이다. 모듈의 의미는 재사용성, 확장성 및 이식성을 고려한 의미의 단독구동 가능한 형태의 하드웨어 구동프로그램 및 전체구성 프로그램을 이루고 있는 부분 프로그램 또는 이들의 조합을 말한다. 이러한 모듈화 방식의 프로그램 구성을 통해, 전체 프로그램은 레고 블록처럼 표준화된 단위의 모듈로 조립하는 방식을 갖게 된다. 각 모듈은 표준적인 외부 인터페이스를 가지며, 모듈기능을 보조하는 자기등록 기능의 스크립트 파일을 가지고 있어 PNP(Plug & Play)기능을 가질 수 있게 구성이 된다. 이러한 모듈방식의 제어기 개발은 추후에 새로운 제어기 구성을 위한 라이브러리로 등록되어 새로운 제어기 구성시의 데이터 베이스로 활용되어질 수 있는 것이다.

결과적으로 모듈화는 전체 제어기 구성의 유연성을 증가시켜, 기능의 확장 및 추가 시에 필요 모듈의 추가 혹은 변경을 통해 원하는 제어기를 개발할 수 있는 것이다. 이에, 실제 반도체 제조 장비인 Chip Mounter의 모듈화 방식의 제어기 개발 적용사례를 통해 개발의 이점을 살펴보기로 한다.

2. 개방형 제어기 연구의 바탕

우선 전용 제어기와 개방형 제어기를 간단히 비교해 보면, 표 1에서 보는 바와 같이 개방형 제어기가 성능을 제외한 부분에서 전용 제어기보다 우수함을 알 수 있다. 하지만 제어기에서 성능요소는 다른 어느 부분보다 중요시되는 항목이며, 완성된 제어기가 필요로 하는 성능을 갖지 못하면 다른 평가요소가 아무리 우수하여도 사용할 수 없다.

표 1. 전용 제어기와 개방형 제어기의 비교

평가요소	전용 제어기	개방형 제어기
성능	높음	낮음
비용	높음	보통
제작기간	길다	짧다
재사용성	낮음	높음
확장성	낮음	높음

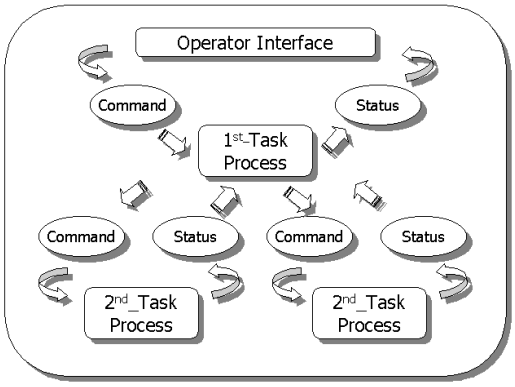
따라서 개방형 제어기 플랫폼 구성에 있어 중요시하고자 하는 것은 전용 제어기의 장점인 높은 성능부분을 개방형 제어기 쪽으로 가져오는 것이다. 하지만 기존의 전용 제어기는 필요로 하는 높은 성능을 유지하기 위해, 대부분 전용 구동프로그램 및 전용 하드웨어 인터페이스 방식을 가지고 있다. 이러한 이유로 기존 제어기를 새로운 제어기로 적용 및 확장하고자 할 때 구성의 많은 부분을 바꾸어야 한다. 하지만 이러한 프로그램 혹은 하드웨어 부분들이 공통의 표준적인 외부 인터페이스로 연결되어있다면, 전체 제어기는 부분적인 구성의 추가 및 변경을 통한 확장 및 변환 시 구성부분의 연결 및 데이터 전달이 용이한 구조를 가지게 된다. 즉, 개별적인 구성부분은 내부적으로 전용 제어기 방식으로 구성이 되지만 외부 연결 및 데이터 전송을 위한 통신채널부분은 공통된 표준 규약으로 구성되는 것이다. 이러한 방식의 모듈은 내부적으로는 전용 제어기의 부분으로써 높은 성능을 유지할 수 있으며, 외부적으로는 서로 다른 이종간의 연결 및 데이터 전송을 편리하게 하여 모듈화 구성을 통한 전체 제어기 구성을 쉽게 이룰 수 있게 한다.

따라서 앞의 내용을 바탕으로 전용 제어기의 성능평가 요소를 유지하면서 개방형 제어기로의 플랫폼을 구성하는 것을 우선원칙으로 정한다.

일반적으로 상위에서의 제어명령은 그림 1에서 보는 바와 같이, 하위로 전달되면서 몇 개의 조합적인 반응을 통해서 작업을 수행하게 된다. 또한 하위의 상태 값들은 상위로 전달되어 상위 작업에 대한 참조자료 혹은 사용자 관찰자료가 된다. 장비에 따라 구성 단

계의 차이는 있지만, 이는 하나의 작업을 세분화하여 구성할 수 있다는 것을 의미하며 제어기 설계시의 모듈 구분의 단위가 된다. 즉, 모듈 구분의 단위를 작게는 Task단위로 볼 수도 있고, 작업 구성의 성능 및 효율성을 위해서는 몇 개의 Task의 조합을 하나의 모듈 단위로 구성할 수 있는 것이다.

그림 1. 일반적인 제어 명령과 상태 값의 전달방식

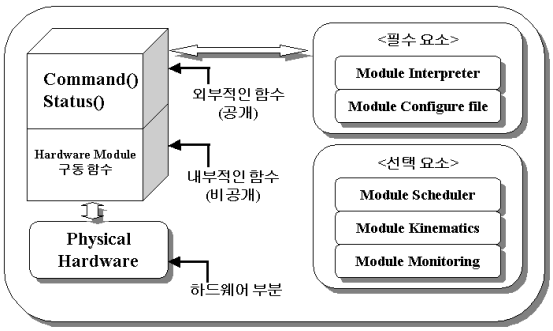


따라서 각각의 모듈은 기능의 단위에 따라 세분화 할 수 있으며, 또한 새로운 구성 부분의 조합으로 융통성 있게 구분되어질 수 있다. 이러한 작업 기능의 세분화에 따른 모듈화로의 구성을 개방형 제어기 플랫폼을 구성의 두 번째 원칙으로 정한다.

3. 모듈의 구성

지금까지 언급한 모듈의 구성 요소를 살펴보면 그림 2에서 보는 바와 같이 하나의 모듈에는 여러 부가적인 요소를 같이 가지게 된다.

그림 2. 모듈(Module) 구성

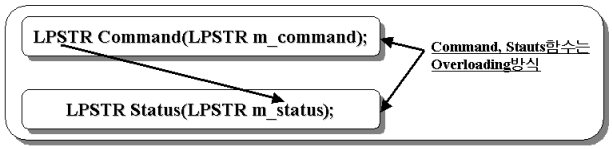


즉, 외부적인 공개함수는 Command()와 Status()를 가지며, 내부적인 비공개 함수는 모듈 구동 함수로 구성되는 기본 모듈의 필수적인 요소인 Module Interpreter와 Module Configure file를 동반하고, 모듈이 적용되는 형태의 성격에 따라 Module Scheduler, Module Kinematics, Module Monitoring 등을 선택적인 요소로 갖는다. 구체적으로 Module Interpreter는 외부 공개함수를 통해서 들어온 명령을 내부 비공개함수로 변환해주는 부분이며, Module Configure file는 모듈이 갖는 성격을 알려주는 파일로서 외부구동 명령어 및 상태 값의 의미를 메인 프로그램에 등록하는데 사용하는 파일이다. 이는 또한 모듈의 plug and play기능을 S/W적으로 구현하기 위한 부분이기도 하다. 추가적으로 Module Scheduler는 Module Interpreter에서 변환해준 내부명령을 task time table에 맞게 체계적으로 작업 관리하는 부분이며, Module Kinematics는 모듈이 구동부를 가질 때 기구학을 계산해 주는 부분이다. 마지막으로, Module Monitoring은 모듈의 현재 상태 및 명령의 전달확인을 상위 모듈로 알려주는 부분으로서 Module Scheduler의 데이터자료가 되며, 상위 GUI 방식의 MMI(Man Machine Interface)에서는 제어기의 동작 상태를 모니터링할 수 있는 참조자료가 된다.

따라서 모듈 개발자는 재사용성과 확장성을 고려한 부분단위로 Module의 사이즈를 결정한다. 그리고 모듈의 내부기능을 전용 제어기 방식의 독자적인 기술로 구현을 한 후 내부 함수에 접근할 수 있는 외부 함수 통로를 표준적인 인터페이스로 만든다. 부가적으로 모듈의 기능을 참조할 수 있는 Module Interpreter, Module Configure file을 작성한다. 만일 모듈이 구동부를 가진다면 Module Kinematics, Module Scheduler등을 추가적으로 필요에 따라 구성한다. 모듈 사용자는 개발된 모듈을 메인 프로그램에 Module Configure file을 통해 내부 구동함수 접근을 위한 외부 명령어 및 상태 값의 의미를 등록한 후 외부 개방함수를 통하여 내부 기능을 전용제어기의 장점을 살리면서 사용한다. 결국, 사용자는 모듈 내부의 함수작용을 단지 외부 명령어의 기능만으로 충분히 제어할 수 있으며, 개발자 입장에서의 모듈은 독자적인 기술의 전용 제어기의 높은 성능을 가질 수 있는 것이다.

여기서 외부 개방의 Command()함수와 Status()함수의 구성을 살펴보면 그림 3과 같은 형태로 이루어진다.

그림 3. 명령함수와 상태함수



유일한 외부 연결 통로인 2개 함수의 입력인자와 반환인자의 형태 모두 char *인 LPSTR로 구성되어져 있다. 따라서, 이들의 명령과 상태 값은 char string형태의 연속된 문자열로 표현이 된다. 이러한 방식의 장점은 m_command와 m_status의 길이를 사용자가 정의할 수 있으며, 전문적인 구동 함수의 사용방식을 모르는 상황에서도 표준적인 Command string을 통해 내부 함수를 사용할 수 있다는 것이다.

그림 4는 m_command와 m_status의 표준통신 프로토콜로서 char string을 구성하는 기준형태를 나타낸 것이다.

그림 4. 표준통신 프로토콜

COMMAND[STATUS] START	송신 ID	수신 ID	COMMAND[STATUS]	CHECK SUM	STOP

LPSTR형태(COMMAND 혹은 STATUS는 Module의 정의에 따름)

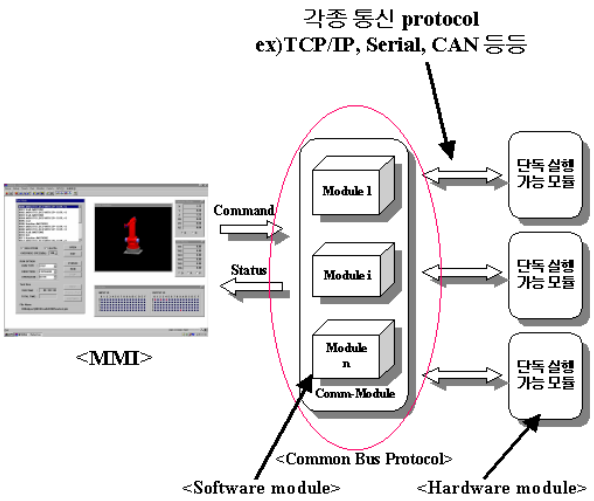
이처럼, 단일한 형태의 프로토콜 구성은 이종간의 연결 시에도 프로토콜의 변환 과정 없이 간단하게 연결할 수 있으며, 단지 프로토콜의 내용을 해석할 수 있는 Module Interpreter를 통해 이들의 명령 및 상태 값을 각각의 내부 프로토콜 명령 및 상태 값으로 변환하여 모듈간에 데이터 전송 및 구동을 쉽게 한다. 이에 표준적인 프로토콜을 사용하여 외부함수를 사용하는 간단한 예를 들어 살펴보면 다음과 같다.

```
Command("send ID, receive ID, command, parameters…");
Ex) Command("001,003,MOVJ, tp1,70,…");
Status(Command("…"));
Ex) Status(Command("001,003,MOVJ,tp1,70,…"));
```

char string사이의 ","은 명령 및 parameter의 구분자로서

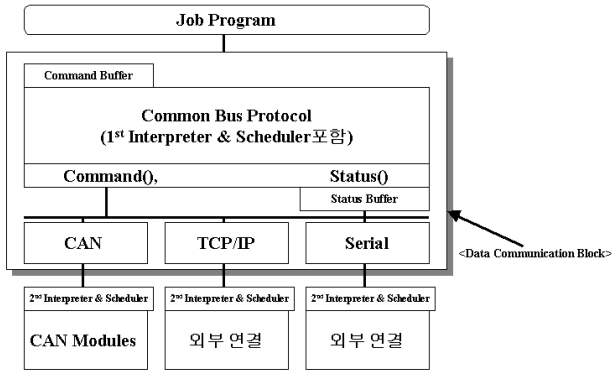
Module Interpreter내부의 string분석의 인식 문자로 쓰인다. 이에 모듈간의 조합과 MMI와의 연결을 살펴보면 그림 5와 같이 구성할 수 있으며, 이종간의 모듈은 내부적으로는 다른 프로토콜로 구성이 되지만, 외부적으로는 앞에서 언급한 표준적인 프로토콜인

그림 5. 모듈간의 통신



Common Bus Protocol블록을 통해서 서로 데이터를 주고 받게 된다. 즉, 모듈이 가지는 각각의 통신 데이터들은 Common Bus Protocol을 통해서 공통된 통신 protocol로 전환되어 처리되는 것이다. 그림 6은 Chip Mounter적용 시의 Common Bus Protocol의 모습을 참고로 나타낸 것이다.

그림 6. Chip Mounter적용 시의 Common Bus Protocol

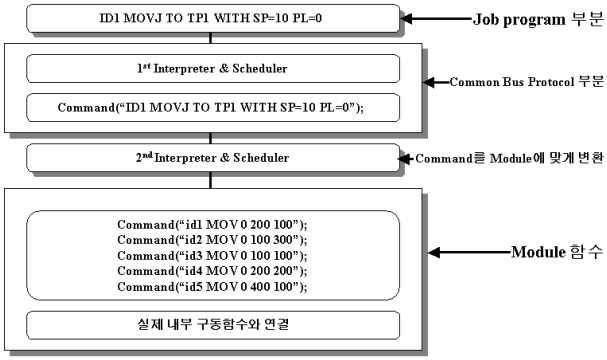


4. 작업파일 구성

모듈화로 전체프로그램을 구성한 후에 필요한 것이 구동 프로그램인 작업파일을 구성하는 것이다. 그림 7에서 보는 바와 같이 전용 제어기에서 사용하는 작업파일은 제어기마다의 특징에 따른 의존성이 강하며, 단순 텍스트방식이 많아서 작성의 어려움 및 오류 발생의 가능성이 높다. 그래서 사용자가 쉽게 사용할 수 있는 GUI 방식의 작업프로그램 에디터를 사용하여 상위단계의 작업프로그램을 만들고 구동 시에는, 파일을 다시 1차적으로 상위의 외부 공개 함수로 변환하고 하위의 모듈에서 다시 이를 내부구동 함수로 변환하는 2단계의 Interpreter방식을 사용한다.

이러한 방식의 2중 Interpreter방식에 따른 작업프로그램의 변환은 내부적으로 일어나는 과정이며, 실제 사용자는 상위단계의 GUI방식의 Job프로그램만 구성하면 되는 것이다.

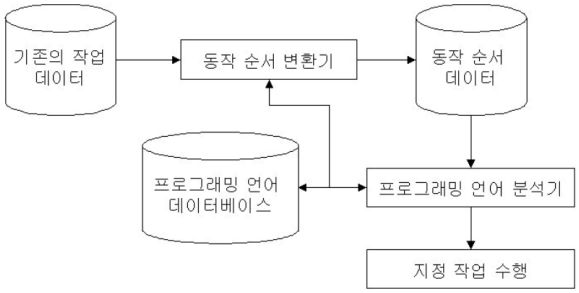
그림 7. 2중 Interpreter에 의한 Job프로그램의 변환 예



결국, 사용자는 기본 GUI프로그램만을 익히고 새로운 제어기 구성에 따른 Module Interpreter의 추가로서 내부적인 2단계의 Interpreter과정을 편리하게 사용할 수 있는 것이다.

이에, 실제적으로 Interpreter과정에 사용되는 언어분석기의 적용방법을 살펴 보면 그림 8과 같다.

그림 8. 동작 순서 변환기와 프로그래밍 언어 분석기의 적용

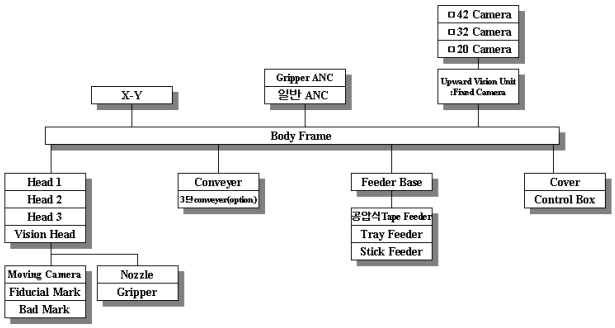


먼저 기존의 작업순서를 규정하고 있는 데이터 파일에 대해 동작 순서 변환기를 이용하여 미리 규정된 프로그래밍 언어들로 이루어진 동작 순서 파일로 변환시킨다. 그리고, 이렇게 생성된 프로그래밍 언어들은 언어 분석기에 의해 실제의 동작 순서로 변환되어 최종적으로 사용자 또는 개발자가 정의한 작업을 수행할 수 있게 되는 것이다.

5. Chip Mounter에 적용사례

실제 반도체 제조 장비인 Chip Mounter에 Module화 방식의 플랫폼 적용사례를 살펴보기로 하자. 그림 9는 일반적인 Chip Mounter의 기구부를 나타낸 것이다.

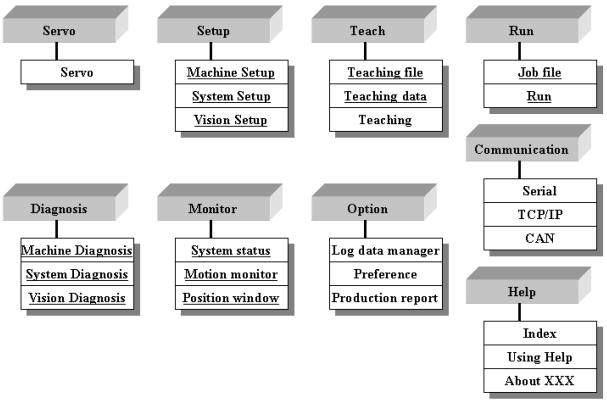
그림 9. 장비의 기구부 구성도



이 장비에 대해 모듈화 방식의 MMI구성을 위한 기본 Menu의 틀을 구성하면 그림 10과 같이 구성할 수 있다. 여기에서

크게 9가지의 메뉴로 구성한 것은 일반적인 제어 기기의 Menu구성을 따른 것이며, 개발자가 개발한 모듈들은 이 메뉴들의 서브 메뉴로 구성된다.

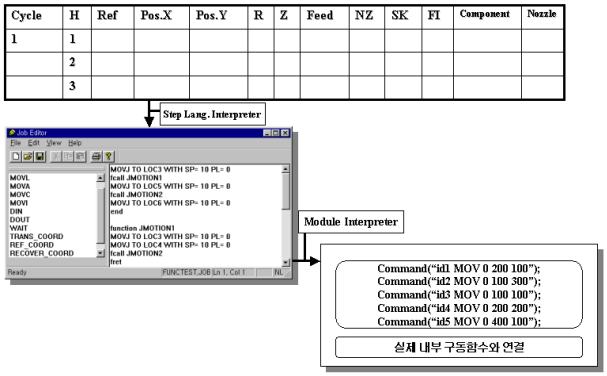
그림 10. 기본 제어 Menu구성



MMI에 사용되는 각각의 Menu는 하위 모듈구동을 위해서 앞에서 언급한 Common Bus Protocol부분을 거치게 되며, 하위 구동모듈의 상태 값도 상위로 전달될 때에 반드시 이 부분을 통하게 된다.

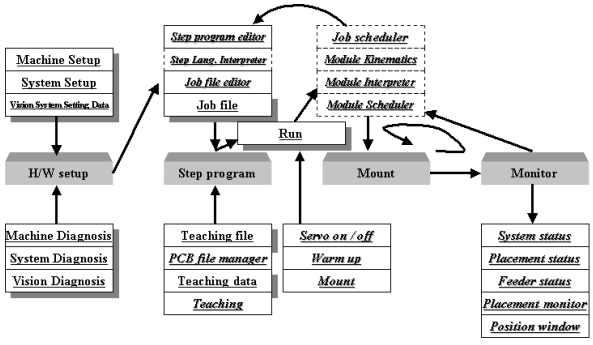
여기서 사용하는 GUI방식의 Job Program을 살펴보면 다음 그림 11과 같은 구조로 되어있다.

그림 11. 작업 프로그램의 구동모습



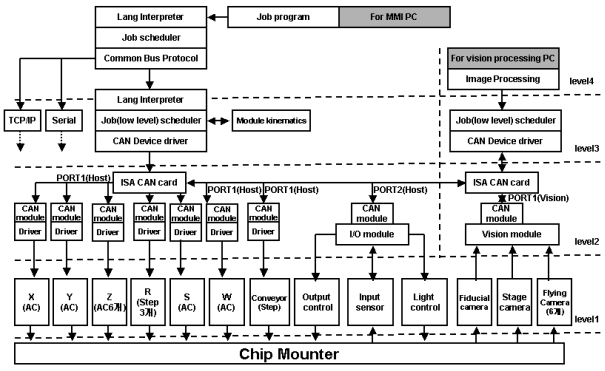
이에, GUI방식의 Job프로그램을 통한 작업 프로그램의 실행 시의 각각의 Sub Menu가 구동하는 모습을 살펴보면 그림 12와 같은 모습으로 동작된다.

그림 12. Sub Menu 프로그램의 구동모습



결과적으로, Module화 방식의 Chip Mounter로의 적용 시의 Main block 하드웨어 및 소프트웨어의 전체적인 연결은 그림 13과 같은 구조를 이루게 된다.

그림 13. Chip Mounter Main block의 모듈화 적용



6. 결론

높은 제어성능을 요구하는 시스템을 전용 제어기를 사용하여 제어하면 원하는 결과를 얻을 수는 있지만, 추후 제어기의 재사용성과 확장성을 고려하면 PC바탕의 개방형 제어기가 우수하다. 이에 개방형 제어기이면서 전용제어기의 우수한 성능을 가지기 위해서는 우리가 제안한 모듈화 방식으로 제어기를 구성하면 높은 성능을 가지는 제어기를 구성할 수 있다. 실제, 반도체 제조 장비인 Chip Mounter에 모듈화 방식의 제어기 구성 사례를 통해 모듈화 방식의 제어플랫폼 구성의 우수함을 검증하고 있으며, 추후에 여기서 개발된 제어 모듈들을 Wire Bonder로의 적용을 통해 다시 한번 플랫폼 검증의 기회를 가질 것이다. 결론적으로, 앞으로 모듈화 방식의 제어기 설계는 더욱 발전적인 연구를 통해 PC바탕의 개방형 제어기에 적용될 수 있는 기술적인 토대를 이룰 것으로 기대되고 있다.

참고문헌

[1] F. T. Cheng, M. T. Lin, R. S. Lee, "Developing a Web-enabled Equipment Driver for Semiconductor Equipment Communications", *Proc. 2000 IEEE*, pp. 2203-2210, 2000

[2] M. L. Moore, V. Gazi, K. M. Passino, "Complex Control System Design and Implementation Using the NIST-RCS Software Library", *1999 IEEE Control Systems*, pp. 12-28, 1999

[3] I. H. Suh, H. J. Yeo, J. S. Ryoo, S. R. Oh, C. W. Lee and B. H. Lee, "Design of a Supervisory Control System for Multiple Robotic Systems", *Proc. IROS96*, pp. 332-339, 1996

[4] 이낙형, 서동수, 엄광식, 서일홍, 여인택, 김성락, "A Design Experience on PC-Based Open Architectured Controller for Robot Systems", *Proc. 13th KACC*, pp. 430-433, 1998

[5] 추성호, 박홍성, 강원준, "A Study of Object Agent for Open Architecture Control system", *Proc. 14th KACC*, pp. A21-A24, 1999

[6] http://www.osaca.org

[7] http://www.arcweb.com/omac

[8] http://icat.snu.ac.kr:3333/open/index.html

[9] http://www.isd.mel.nist.gov/projects/rcs_lib

[10] http://www.sei.cmu.edu/str/descriptions/com_body.html