Proceedings of the 12th
KACC, October 1997

실시간 분산처리 시뮬레이터 및 제어기 구조 설계

Designing of Real-time Distributed Simulator and Controller Architecture

°양광웅, 박재현

인하대학교 자동화공학과(Tel: (032)860-7713; Fax:(032)873-4386; E-mail: {ygkgwg,jhyun}@rcsl.inha.ac.kr)

Abstract High performance digital computer technology enables the digital computer-based controllers to replace traditional analog controllers used for factory automations. This replacement, however, brings up the side effects caused by discrete quantization and non-real-time execution of control softwares. This paper describes the structure of real-time simulator and controller that can be used for design and verification of real-time digital controllers. The virtual machine concept adopted by real-time simulator make the proposed simulator be independent from the specific hardware platforms. The proposed system can also be used in the loosely coupled distributed environments connected through local area network using real-time message passing algorithm and virtual data table based on the shared memory mechanism.

Keywords Real-time simulator, real-time control, open architecture, virtual machine, object oriented programming.

1. 서론

디지털 컴퓨터 기술의 비약적인 발전으로 대부분의 아날로그 제어기들은 디지털 제어기로 대체되고 있으며, 전기적인 회로로 구성되었던 제어 알고리즘은 제어기상에서 동작하는 소프트웨어로 바뀌었다. 제어해야 할 시스템의 범위가 커지고 복잡하게 발전함으로 인해 수학적인 접근이 용이하지 않으며 시뮬레이션을 통해서 시스템의 특성을 해석하고 개선해 나가게 된다. 시스템 해석을 위하여 다양한 패키지가 개발되어 있으나, 현재 주로 사용되고 있는 시뮬레이션 패키지들 (Ctrl-C, EASY5, Simnon, SIMULINK, MATRIXx)은 제어알고리즘의 최적의 해를 구하거나 제어모델의 응답특성을 해석하는데 주 목적이 있기 때문에 실시간 시뮬레이션에는 큰 비중을 두지 않았다. 하지만 실시간으로 동작하는 플랜트와 연계하여 제어기의 구조를 모델링 하거나, 제어기의 실시간 동작 특성을 확인하기 위해서는 시간 제약조건을 만족하는 분산처리 시뮬레이터 및 제어기의 사용을 필요로 한다.

실시간 시뮬레이션 영역에는, 정확한 모델을 구하는 분야와 주어진 모델을 효과적으로 시뮬레이션하는 방법에 대한 연구, 크게 두 분야로 연구 영역이 나뉜다. 이 중, 시뮬레이션 방법에 대한 연구는 주로 수백에서 수천 개의 비선형 미분 방정식으로 이루어진 대규모 시스템에 관한 것도 포함되어 있다. 그래서 발전소와 같은 대규모 시스템 시뮬레이션의 계산 시간을 단축하기 위해서 멀티프로세서를 이용하거나 DSP 모듈을 장착하는 것은 시뮬레이터 및 제어기 성능개선의 한계를 보이게 된다. 하지만, 모델을 모듈별로 나누고 각각의 노드로 부하를 분산시킨다면 실시간 제어 및 시뮬레이션이 가능해진다. 제어기의 개발과정에서는 제어 알고리즘의 검증이 필요하며, 플렌트에 직접 연결하여 검증하는 것은 문제가 있기 때문에 시뮬레이터를 사용하는 것이 필요하다[1]. 분산처리 시뮬레이터의 제어 알고리즘은 두 시스템 간의 공통된 인터페이스를 제공하는 message queue와 공유메모리를 통해서 제어대상과 연결되므로, 제어대상을 실제의 플랜트가 아닌 가상의 플랜트 모델을 설정하여 제어 알고리즘의 성능 분석을 하는 것이 가능하다[2]. 따라서, 제어기의 모델을 개발하고자 할 때 실제의 플랜트가 필요 없게 되며, 개발된 제어기 모델은 수정 없이 실제의 제어기에 도입된다.

제어 알고리즘의 개발에 객체지향 프로그래밍 방법을 사용하여 모델을 모듈별로 나누어 프로그램을 작성하여 모듈간의 결합도를 낮추고 모듈의 재사용성을 높일 수 있다[3][4]. 객체지향 언어를 지원하는 시스템 디자인 패키지 구조가 [5]에서 제시되었으며, 새로 개발되는 제어기 구조에는 여기에 가상머신을 도입하여 다양한 컴퓨터로 구성된 분산환경에서의 이식성을 제공한다. Java 가상머신은 Java에게 높은 이식성과 객체지향 개념을 제공한다. 하지만 제어모델 해석에 주로 사용되는 메트릭스 연산을 하기 위해서는 상당히 많은 Java bytecode가 필요하며, 특정한 머신에서 emulation되어 수행되는 Java bytecode는 수행속도가 현저히 느려지게 된다. 따라서, Java 가상머신을 시뮬레이터 및 제어기에 수정 없이 도입한다면 속도가 느리기 때문에, 사용이 빈번한 기능이나 제어 알고리즘은 가상머신 내부에 sub-function 형태로 제공된다.

Carnegie Mellon University의 Stewart 등은 port-based object를 이용한 실시간 처리 소프트웨어의 구조를 제안하였다[6]. 하지만 각각의 태스크간에 데이터를 전달하기 위해 global state variable table을 사용하기 때문에 프로세서간에 비동기적으로 발생하는 메시지의 전달에는 어려운 점이 있고, 하드웨어와 인터페이스 하기위해서 port-based object를 이용하는 것은 다양한 종류의 하드웨어 인터페이스 상에서 동작하는 port-based object를 필요로 하여 소프트웨어의 확장성을 떨어뜨린다. Hoger 등은 분산 시뮬레이션(distributed simulation)과 공동 시뮬레이션(concurrent simulation)을 통합한 시뮬레이터 모델을 제안하여 시뮬레이션 모델의 확장성을 높이고, 노드에 부하가 고르게 분포하지 않을 때의 시뮬레이터 성능저하를 보여주고 있다[7].

본 논문에서는 현재 개발중인 실시간 분산처리 시뮬레이터 및 제어기*(Real-time Distributed Simulator and Controller; RDSC)*의 내부구조와 실시간 분산처리 환경을 제시하고 있다. 제2장에서는 분산환경에서 각각의 노드 간에 비주기적으로 발생하는 메시지와 주기적으로 발생하는 데이터의 교환 구조를 제시하고, 제3장에서는 RDSC의 시스템을 구성하는 *MDLI(Model Description Language Interpreter)*, 컴파일러, 그리고 가상머신의 구조를 제시하고자 한다.

2. 실시간 분산처리 환경

분산처리 시뮬레이션에는 여러 개의 프로세서가 하나의 공유메모리와 message queue를 공유하는 *concurrent simulation*과 각각

본 연구는 97년도 인하대학교 연구비의 지원에 의하여 수행되었습니다.

의 프로세서마다 지역메모리와 message queue를 따로 가지는 *distributed simulation*으로 나뉜다. Concurrent simulation에서는 모든 프로세서의 이벤트들을 공유되는 message queue에서 스케쥴링하여 시뮬레이션 하게 되며, 공유메모리는 하나의 프로세서에 의해서만 읽고 쓸 수 있으므로 각 프로세서 간의 mutual exclusion을 요구한다. 각각의 프로세서마다 작업량을 일정하게 분배할 수 있고 병목현상을 줄일 수 있지만, 많은 프로세서가 공유자원을 엑세스 하기 위해서 자원의 lock/unlock 과정이 추가되어야 하기때문에 공유자원의 엑세스 시간이 길어진다. 따라서 시뮬레이션 모델을 확장하여 프로세서 수를 늘이는 데에는 한계가 있다. Distributed simulation에서는 통신링크를 통해 전달되는 타임 스탬프를 가지는 메시지의 교환에 의해 비동기적으로 시뮬레이션이 진행된다. 시뮬레이션 모델을 확장할 때는 프로세서의 개수를 늘임으로 해결할 수 있지만, 하나의 프로세서에 작업량이 집중될 때에는 작업이 끝날 때 까지 다른 프로세서가 기다려야 하므로 시뮬레이션 도중에 병목현상을 보이게 된다[8].

RDSC에 도입된 분산처리 시뮬레이션에서의 데이터와 메시지교환 구조는 concurrent simulation에서 프로세서간에 공유되어 사용되는 state variable table과, distributed simulation에서의 message queue 구조이다. 노드에서 발생하는 연속적인 데이터의 전달에는 state variable table이 사용되고, 비주기적으로 발생하는 메시지의 처리에는 우선순위를 가지는 message queue가 사용되었다. 위와 같은 구조는 시뮬레이션 모델의 확장이 용이하고, 각각의 노드에서 수행되는 작업을 다이나믹하게 스케쥴링 가능하고, 한 노드에 작업량이 집중되더라도 다른 노드는 영향을 적게 받기 때문에 병목현상을 줄일 수 있다. 그림 1은 슬레이브 노드들이 지역변수테이블(Local Variable Table; LVT)을 경유하여 마스터 노드의 공유변수테이블(shared variable table; SVT)과 데이터의 동기를 취하는 구조를 보여준다.

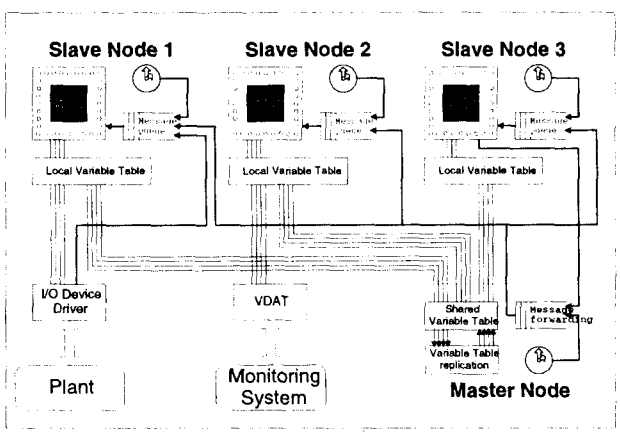

그림 1: 마스터/슬레이브 노드간의 연결 구조.
Fig. 1: Connection structure of Master/slave nodes.

2.1 비주기적 메시지 전달

Error, warning과 같이 비주기적으로 발생하는 이벤트는 타임 스탬프를 가지는 메시지 형식으로 노드에 전달되어 message queue에 삽입된다. 그리고 각각의 슬레이브 노드를 동기 시키기 위해서 null 이벤트를 사용한다. 위와 같은 비주기적 메시지들 중에는 중요도가 높은 것이 많으므로 대부분 가장 높은 우선순위를 가지고 message

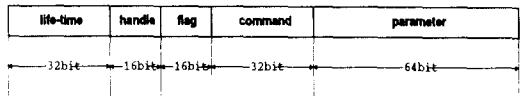

그림 2: 메시지 패킷의 구조.
Fig. 2: The structure of message packet.

queue에서 스케쥴링 된다.

그림 2에서 메시지 패킷의 구조를 보여주며 *lifetime, handle, flag, command, parameter*로 구성되어 있다. Handle은 메시지를 받을 노드의 주소를 담고 있다. Handle이 0을 가지면 마스트 노드에 대한 handle을 의미하며, 0 이외의 값은 마스트 노드로 전달 된 다음 다시 목적 노드로 forwarding 된다. 목적 노드에 도달한 Message queue는 새로운 메시지를 받아서 스케쥴링 알고리즘에 따라 동적으로 메시지의 우선순위를 재배열 시킨다. RDSC에서 사용하는 스케쥴링 알고리즘은 메시지의 잔여수명 (residual life-time)이 빠를수록 message queue의 앞쪽에 위치하게 된다. Flag는 긴급, 응답요구 등의 메시지 상태를 기록하는데 사용된다. Message queue에서 꺼내진 메시지는 command와 parameter에 따라 태스크에서 수행되는 작업의 종류가 결정된다. 수행이 끝나고 나면, flag의 상태에 따라 응답 메시지를 송신측으로 되돌려 주기도 하며, 마스터/슬레이브 노드의 메시지 전달 구조에서 한 주기의 요구/응답 사이클을 완료한다.

2.2 주기적 데이터 교환

발전소와 같은 대규모 시스템은 여러 가지 모듈들이 합쳐진 형태이며, 이러한 시스템은 공간적인 위치나 기능에 따라 나누어져 다중 프로세서에 의해서 처리된다. 이 경우 각 구성요소의 특성에 따라 처리되는 프로세서의 주기가 달라진다. RDSC는 이러한 요구를 만족시키기 위해 각 노드의 수행 주기에 따라 메시지를 발생시키는 클럭을 가지고 있다. 이 구조는 각 노드에 일정한 기저부하를 걸어주므로 시뮬레이션이 무리 없이 수행되기 위해서는 일정한 주기로 수행되는 태스크의 평균 수행시간이 메시지의 발생주기보다 짧아야 한다.

주기적인 메시지는 주로 입출력 디바이스로부터 데이터 수집, 모니터링 시스템으로 데이터 출력, 제어 알고리즘의 반복 수행 등의 작업을 하여 처리되는 연속성을 가지는 데이터 교환을 목적으로 LVT와 SVT를 사용한다. LVT와 SVT는 공유메모리 (shared memory)와 같은 역할을 하도록 Ethernet, fieldbus, 비동기 통신과 같은 네트웍 레이어를 추상화 시킨 상위 레이어이다. LVT의 변수들은 *export*와 *import*의 두 가지 속성을 가진다. export로 선언된 변수에 데이터를 기록하게 되면 슬레이브 노드의 LVT에서 마스터 노드의 SVT로 전달되고, 마스터 노드에서 다시 각각의 슬레이브 노드가 가진 LVT로 변수 테이블이 복사된다.

3. RDSC의 시스템 구조

RDSC는 MDLI와 컴파일러, 가상머신으로 구성되어있다. 시각적으로 모델링 된 시스템은 MDLI에 의해 객체지향 소스코드로 변환된다. 컴파일러는 소스코드를 컴파일 하여 가상머신에서 수행되는 기계어 코드를 만든다. 컴파일 된 기계어코드는 이것을 수행시킬 노드에 다운로드 되고, 이것은 노드에 설치 된 가상머신 상에서 마스터 노드와 슬레이브 노드들은 메시지와 SVT를 통해 데이터를 주고받으며 수행된다. 이러한 RDSC 시스템 구조는 시뮬레이션할 시스템의 구조를 시각적으로 표현하고 분리하는 것이 가능하고

분산처리 환경에서 태스크의 수행과 각 노드간의 통신구조 설정을 쉽게 만든다[9].

3.1 *MDLI*

MDL(Model Description Language)은 시스템을 모델링 할 때 시각화 된 모델의 각 모듈의 선언과 연결구조를 표현하는 언어이다. 사용자는 GUI(Graphic User Interface)환경에서 아이콘으로 표현된 각종 객체들을 끌어놓은 후 객체들을 선으로 연결하여 모델을 편집하게 된다. 이러한 식의 시각화 된 모델은 사용자가 쉽게 이해하는 것이 가능하며 객체별로 모듈화 된 소스코드는 모델의 수정, 추가, 유지보수가 용이하다. 시각화 된 모델은 MDL에 의해 표현이 된다. MDLI는 MDL로 표현된 각 객체와 연결구조에 해당하는 클래스와 클래스간의 메시지 전달구조를 가지는 소스코드를 만들어 내는 전처리 과정이다.

시각화 된 모델을 구성하는 port-based object는 알고리즘 구성의 최소 단위가 되며, 객체의 내부는 외부로부터 숨겨지고 다른 객체와 통신하기 위해 다수의 입출력 포트를 사용한다. port-based object의 입출력 포트는 태스크간의 통신을 위해서 SVT를 경유하거나 메시지를 주고받는다. 한 개의 출력 포트가 여러 개의 입력 포트에 연결될 수 있으며, 두 개 이상의 출력포트를 서로 연결 시킬 때는 joint connector를 이용한다. 다수개의 port-based object가 모여서 하나의 객체를 형성하는 것이 가능하며, 다음과 같은 이점을 제공한다. 1)객체들은 객체지향 언어의 캡슐화와 상속성을 이용하여 쉽게 객체지향 소스코드로 바뀐다. 그리고 2)복잡한 알고리즘을 표현하는 객체들을 작고 단순한 하나의 객체로 묶는 것이 가능하다.

3.2 컴파일러

컴파일러는 전처리*(pre processing)*, 구문 분석*(lexical analysis)*, 파싱*(parsing)*, 그리고 링킹*(linking)*의 네 가지의 과정으로 나누어진다 [10]. 구문 분 석기는 소스코드를 읽어 들여 파서가 인식할 수 있도록 문자열이나 기호 단위로 분해하여 토큰으로 잘라낸다. 파서는 토큰의 문법 체크를 하고 제어문장 구조에 따라 분기되도록 배치시키며, 수식을 표현하는 토큰을 연산 순위에 따라 syntax tree를 만들어내고 이러한 구조를 기계어로 변환하여 파일에 저장한다. 컴파일 된 기계어 코드는 필요로 하는 라이브러리나 다른 목적 코드들과 연결되어 가상머신에서 수행될 수 있는 실행코드가 만들어진다.

컴파일러가 인식하는 제어 문장은 if(조건)수행 else수행, for(시작조건;조건;수행)수행, while(조건)수행, break, continue 등의 C 컴파일러의 제어문장 구조와 흡사한 형태를 제공한다. C++에서 지원하는 클래스 구조와 같은 class, private, public, protected등의 추가 예약어를 지원하여 모델을 클래스 구조로 표현하는 것이 가능하다. 그리고 수식의 표현 식은 C에서 사용하는 구조와 비슷한 연산자와 함수호출형식, 숫자표현 형식을 제공하며, 추가로 벡터와 메트릭스에 관한 연산자와 표현형식들이 제공된다.

3.3 *가상머신(virtual machine)*

컴파일 된 기계어 코드는 수행될 노드로 Ethernet, fieldbus, 혹은 비동기 통신을 통해 다운로드 된다. C/C++과 같은 언어로 작성된 소스 코드는 수행되는 컴퓨터가 바뀌면 소스코드를 다시 컴파일 하여야 하지만, 가상머신은 컴파일 된 기계어 코드를 각 노드에서 코드의 수정 없이 수행시킬 수 있도록 설계된 추상화 된 컴퓨터이다. 그림 3에서 보여주는 RDSC의 가상머신은 스택(stack)을 중심으로 동작 하기 때문에 PC(program counter), SP(stack pointer), BP(base pointer), CP(class pointer)등의 적은 수의 register를 가지며, 표 1과 같이 instruction set의 구성을 단순화 시키고 이들의 구현을 쉽게 한다.

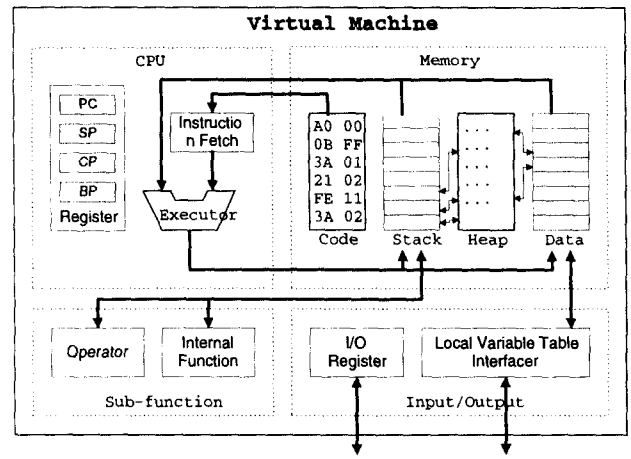

그림 3: 가상머신의 내부 구조.
Fig. 3: The internal structure of virtual machine.

표1: 가상머신의 인스트럭션.
Table1: Instructions of the virtual machine.

type	*range*	*instructions*
Stack	20 - 3F A0 - AF	PUSH, POP, ENSS, RELS, PEEK
Arithmetic	40 - 5F	ADD, SUB
Flow-control	10 - 1F 60 - 6F	JP, JR, CFJP, CFJR, CTJP, CTJR, RNJP, RNJR, CALL, RET, IFUNC
Move	70 - 9F	MOV, MOVF
Vector, Matrix	C0 - DF	EVCT, EMAT, MATX, CPLX, ELEM, VVCT, PVCT
Operator	E0 - FF	PLU, MIN, ADD, SUB, MUL, DIV, IDIV, POW, AND, OR, NEG, GT, LT, ASSIGN, VECT, MERG, IR, LE, GE, EQ, NE

인스트럭션은 1-byte의 op-code와 없거나 한 개 이상의 operand로 구성되어 있다. 가상머신에서 사용되는 메모리는 코드 영역, 데이터 영역, 스택 영역, 그리고 힙(heap) 영역으로 구성된다. 코드 영역에는 가상머신에서 수행될 기계어 코드가 저장되며, 데이터 영역에는 전역변수의 저장공간으로 사용되고, 함수의 호출이나 함수 내부의 지역변수 저장에는 스택 영역이 사용되고, 힙 영역에는 프로그램 수행 중 가변적인 크기를 가지는 변수가 사용하는 메모리 공간이 할당된다. Executor는 PC register가 가리키는 곳의 코드 영역에서 명령어를 가져와 해석하고 이를 수행한다. Operator는 논리연산과 사칙연산을 제공하며, internal function들은 태스크의 수행속도를 최대한 증가 시키기 위해서 가상머신과 같이 컴파일 되어있다.

RDSC의 가상머신은 가상머신이 특정한 컴퓨터에 이식되었다는 전제 하에 기계어 코드를 하드웨어 플랫폼이나 운영체제에 무관하

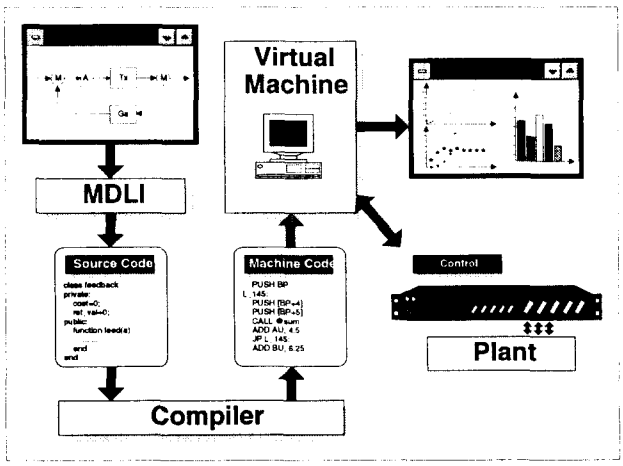

그림 4: Real-time Distributed Simulator and Controller의 구성.
Fig. 4: Parts of Real-time Distributed Simulator and Controller.

게 수행 시킬 수 있으므로 RDSC가 이식성을 가지게 한다.

4. RDSC의 구현

RDSC의 전체적인 구성과 데이터 흐름은 그림 4에서 보여주고 있다. RDSC는 현재 C++ 언어를 이용하여 개발되고 있으며, 60,000라인 정도의 소스코드 분량이 이미 개발되었다. 소스코드는 Windows 95/NT 플랫폼에서 동작하도록 컴파일 되며, 크게 *MDLI*, 컴파일러&링커, 가상머신, 그리고 가상머신 인터페이스로 나누어져 개발되고 있다. MDLI와 컴파일러는 전처리, 구문 분석, 파싱, 그리고 링킹의 네 가지의 과정을 거치 는 일반적인 컴파일러를 모델로 하여 개발되고 있으며, 가상머신은 라이브러리나 DLL 형태로 제공되어 가상머신 인터페이스에서 사용된다. 그림 5는 가상머신 인터페이스를 통해 가상머신을 사용자와 연결해 주는 윈도우 화면이며, 시뮬레이션이 진행되는 동안 타스크 로드의 변화를 기록하고 보여준다.

사용자가 제어 알고리즘을 개발 할 때 기본적으로 필요로 하는 기능을 제공하는 함수들은 태스크의 속도증가를 위해서 operator와 internal function 형태로 가상머신에서 제공된다. 실수와 복소수의 기본적인 사칙 연산자와 수학 함수를 제공하며, 행렬이나 polynomial에 관한 연산과 조작을 위한 다양한 연산자와 함수들을 제공한다. 가상머신에서 사용되는 자료 구조 중에서 기본 구조는 실수(real), 복소수(complex), 그리고 문자열(string) 이다. 실수는 8byte의 크기의 IEEE 형식을 따르고, 복소수(complex)는 각각 8byte의 실수부(real number)와 허수부(image number)로 구성되어 있다. 문자열에서 한 문자는 1byte의 크기를 가진다. 그리고 기본 구조를 메트릭스 형태로도 표현이 가능하다.

5. 결론

위에서 제시하는 실시간 분산처리 시뮬레이터 및 제어기 구조는 다기종으로 구성된 지역 네트웍 환경을 수정 없이 이용가능하고 제어 알고리즘의 개발에는 객체지향 방법을 사용하므로, 시스템 개발기간을 단축할 수 있으며 개발된 시스템의 유지보수 비용도 줄일 수 있으므로 실질적인 생산성향상에 기여 하게 된다. 시뮬레이터와 제어기가 다기종으로 구성된 컴퓨팅 환경에서 동작하기 위해서는 가상머신이 다양 한 하드웨어 플랫폼에 따라 개발 되어야 하며, 분산환경에서 각 노드에서 수행되는 태스크의 부하를 적절이 분배

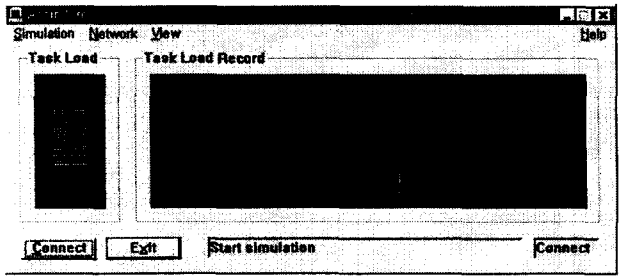

그림 5: 시뮬레이션 수행 윈도우.
Fig. 5: Simulation running window.

하기 위해서는 태스크의 수행시간을 예측하는 알고리즘의 개발이 필요하다.

참고문헌

[1] S. Bennett, *Real-time Computer Control.* Maylands Avenue Hemel Hempstead: Prentice Hall, 1994.

[2] J. Park, Z. J. Pasek, Y. Shan, Y. Koren, K. G. Shin, and G. Ulsoy, "An open architecture real-time controller for machining processes," in *Proceedings of CIRP 27*, (Michigan, U.S.A), May 1995.

[3] H. El-Rewini and S. Hamilton, "Object technology," *IEEE Computer*, vol. 28, pp. 58–72, Oct. 1995.

[4] E. H. Khan, M. Al-A'ali, and M. R. Girgis, "Object-oriented programming for structured procedural programmers," *IEEE Computer*, vol. 28, pp. 48–57, Oct. 1995.

[5] G. Yang and J. Park, "Development of object oriented computer aided control system design (oo-cacsd) package," in *Proceedings of the 11th Korea Automatic Control Conference*, (Korea), pp. 441–444, Oct. 1996.

[6] D. B. Stewart, R. A. Volpe, and P. K. Khosla, "Design of dynamically reconfigurable real-time software using port-based objects," tech. rep., Carnegie Mellon University, Pittsburgh, PA 15213, July 1993.

[7] H. R. Hoeger and D. W. Jones, "Integrating concurrent and conservative distributed discrete- event simulators," *Simulation*, vol. 66, pp. 303–314, Nov 1996.

[8] C. G.Cassandras, *Discrete Event Systems:modeling and performance analysis.* Richard D.Irwin,Inc., and Aksen Associates,Inc., 1993.

[9] T. Kamigaki and N. Nakamura, "An object-oriented visual model-building and simulation system for fms control," *Simulation*, vol. 67, pp. 375–385, Dec. 1996.

[10] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers.* Massachusetts: Addison-Wesley Publishing Company, 1986.