자바-네이티브 조합모델을 이용한 실시간 임베디드 미들웨어 시스템에 관한 연구

論 文
54D-3-2

Real-time Embedded Middleware System using Java-Native Combination Model

鄭 峻 永† · 金 光 手* · 鄭 民 手**
(Jun-Young Jung · Kwang-Soo Kim · Min-Soo Jung)

Abstract - In the field of electrical industry, embedded computing environment including hardware and software is getting more important as the industry shifts to the knowledge-based one. Java could play a great role as bridging technology in such a transition because it provides a lot of benefits like dynamic application download, compatibility of cross platform, and its own security solution. However, the Java technology has a limitation of real-time problem when it is applied to the embedded computing system of the electrical industry. To solve the problem, a novel java-native combination model has been proposed and designed to a firmware level. This scheme has been employed in four kinds of control boards. The result shows that the proposed model has great potential to implement the real-time processing in control of the devices.

Key Words : Java-Native, Combination Model, Real-time, Middleware, Embedded System

1. 서 론

유비쿼터스는 정보통신 및 산업분야의 차세대 패러다임으로서 각광받고 있다. 이러한 유비쿼터스 패러다임을 실현하기 위한 P2P, P2M, M2M, RFID와 같은 각종 네트워킹 기술과 더불어 하드웨어와 소프트웨어 기술들이 연구되고 있다. 하지만 이들 기술들의 최대 문제는 실생활에 사용되고 있는 각종 디지털화 가능한 기기들이 하드웨어나 소프트웨어적으로 너무나 다양하다는데 기인한다. 이러한 근본적인 문제는 산업분야에서도 예외가 아니다. 이와 더불어 중장기 설비를 이용하거나 생산 및 제조시설을 갖추고 있는 산업분야에서는 전통적인 컴퓨팅 환경과 통신방법 및 매체의 업그레이드가 쉽지 않다는 것이다. 물론 컴퓨팅 환경을 포함한 시설 및 설비들을 생산하는 업체들의 다양한 하드웨어 기술을 표준화 및 통일화시키기는 많은 노력과 비용이 소비될 것이다. 그리고 이미 설치되어 운영되고 있는 각종 시설 및 설비에 대한 유지보수도 다양한 하드웨어 특성으로 인해 용이하지는 않다. 이러한 제품 다양성으로 인한 하드웨어적인 컴퓨팅 환경의 이질성을 소프트웨어적으로 완화시켜 줄 수 있는 기술이 특히 산업분야에서는 필요하며 이를 가능하게 해주는 기술이 자바이다.

초창기 자바의 태생은 다양한 각종 가전기기에 탑재되어 운영되어질 표준화된 소프트웨어 언어이자 플랫폼으로서, 하드웨어적인 다양성을 소프트웨어적으로 동일한 런타임 환경으로 전환할 수 있을 뿐만 아니라, 이기종간의 네트워킹 환경을 고려한 실행환경을 동질화시킬 수 있다[1,2,3,10,11]. 그리고 자바언어 자체적으로도 보안성을 가지고 있으며, 90년대 후반부터 지금까지 인터넷이라는 패러다임과 만나면서 네트워크 언어로서 그 사용이 폭발적으로 증가하였다. 네트워킹 환경을 통합할 수 있는 기술로서 인지되고, 차세대 유비쿼터스 환경을 앞당길 수 있는 기술로서 더욱 부각되기 시작한 기술이 자바이다. 따라서 자바기술은 정보통신분야와 산업분야의 현격한 기술 격차를 충분히 소프트웨어적으로 극복 가능할 것으로 충분히 예상된다. 이러한 자바기술의 장점은 오히려 실시간적인 태스크 처리를 실현하는데 있어서 자바 런타임 환경의 특성으로 인해 그 한계가 있으며[3,4,5], 이를 극복하는 것이 산업분야에서의 자바기술을 확산시키는 가장 큰 쟁점이 될 것이다.

본 논문에서는 많은 노력과 비용을 절감하면서도 산업용 유비쿼터스 환경으로의 전환을 보다 용이하게 하기 위하여 산업분야에 적합한 실시간 연산처리를 고려한 자바 네이티브 조합 모델을 제안한다. 그리고 제안된 모델을 기반으로 모든 산업분야의 다양한 대상 시스템을 유동적인 컴퓨팅 환경으로 통합하기 위한 실시간 임베디드 미들웨어 시스템을 설계 및 구현한다.

본 논문의 구성은 다음과 같다. 2장에서는 임베디드 시스템에서 적용 가능한 자바기술에 대한 시스템 환경 특성에 대해 살펴보고, 산업분야에서 적용 가능한 자바기술의 연구사례 및 모델에 대해 기술한다. 그리고 본 논문에서 활용할 자바 네이티브 조합모델을 제안하고, 제안된 모델을 이용한 실시간 임베디드 미들웨어 시스템을 설계 및 구현과 동시에 실험한 결과를 기술한다. 마지막으로 3장에서는 본 논문에서 제안한 시스템이 가지는 의미를 설명하는 결론으로써 끝맺는다.

† 교신저자, 正 會 員 : 慶南大 컴퓨터工學科 博士課程
E-mail : mrj@hawk.kyungnam.ac.kr
* 正 會 員 : 韓國電氣硏究院 責任硏究員 · 工博
** 正 會 員 : 慶南大 情報通信工學部 敎授 · 工博
接受日字 : 2004年 10月 29日
最終完了 : 2005年 2月 1日

2. 본 론

2.1 자바 플랫폼 기반 컴퓨팅 시스템 환경

2.1.1 Real-time Java

자바 프로그래밍 언어와 자바가상기계를 포함하는 자바기술은 1995년에 소개된 이래 개발자 커뮤니티에서 광범위하게 채택되어졌다. 자바의 단순화된 객체 모델, 강력한 안정성과 보안성, 필수적인 멀티쓰레딩 지원, 그리고 WORA(Write Once, Run Anywhere) 는 임베디드 시스템 환경에서의 많은 혜택을 제공하였다. 하지만 대부분의 자바 구현시 큰 사이즈의 플랫폼, 비결정적 동작(nondeterministic behavior) 그리고 탁월하지 못한 수행성능은 실시간과 임베디드 환경의 개발에 있어서 자바의 선택에 악영향을 끼쳤다[5,6].

RTSJ(Real-Time Specification for Java)에서는 이러한 문제에 초점을 맞춰 실시간 자바환경의 구축에 대한 기본적인 가이드 라인을 다음과 같이 제시하고 있다. 먼저, 특정한 자바 환경에 적용가능하게 작성하되, 기본적인 자바 사용의 제한은 있어서는 안된다는 것이다. 둘째, 비실시간 자바 프로그램 또한 실시간 자바 프로그램과 호환성 있게 작성하라는 것이다. 셋째, 어떠한 플랫폼에서도 한번 작성된 프로그램은 원활히 수행이 가능해야하며, 실시간 프로그램의 구현도 적절하게 사용하라는 의미도 포함되어 있다. 넷째, 실시간 프로그램을 위한 우선순위 실행은 때때로 수행성능의 트레이드오프(trade-off)를 발생할 수 있으므로 주의하며, 마지막으로 실시간 자바 프로그램을 위한 어떠한 구문 수행이나 수정은 WORA에 위배되므로 안된다는 것이다[7].

이러한 가이드 라인을 적용한 실시간 자바를 위해 패키지 형태의 자바 API(Application Program Interface)로서 RTSJ는 제공하고 있으나, 자바환경이 내포하고 있는 실시간 문제를 근본적으로 해결하지는 못한다. 자바 런타임 환경이 갖추어진 모든 플랫폼에서의 동일한 수행결과를 기대하는 자바기술은 임의적인 자바의 구문 및 스펙 변경은 용인되지 못하기 때문이다.

2.1.2 JavaOS

자바 플랫폼은 기존 플랫폼의 최상위에 위치하며 바이트코드를 실행하는데, 이 코드들은 특정 하드웨어나 기기에 종속되지 않는 가상기기(Virtual Machine)를 위한 명령어이다[1,2,3,10]. 그림 1과 같이, JavaOS는 자바 애플릿이나 애플리케이션을 실행할 수 있는 자바 플랫폼을 구현한다. 즉 호스트 운영체제의 지원없이도 윈도우즈, 네트워킹 및 파일 시스템의 기본 기능과 자바가상기계(Java Virtual Machine)을 구현한다. JavaOS는 명령어셋과 하드웨어에 의존적인 네이티브 코드와 특정 플랫폼에 독립적인 Java 코드를 결합한 것으로, CPU, 물리적인 메모리 및 모든 부속 디바이스, 버스 슬롯과 같은 플랫폼들로 규정한다. 운영체제에서 플랫폼 독립적인 컴포넌트를 JavaOS 런타임이라 하며, 플랫폼에 의존적인 부분을 JavaOS 커널이라고 한다. JavaOS 커널은 기본적으로 호스트 운영체제가 수행하는 부팅, 예외, 쓰레드, 메모리관리, 모니터링, 파일시스템, 타이밍, 네이티브 코드 라이브러리 관리, 인터럽트, DMA, 디버깅 그리고 다양한 플랫폼등을 제어할 수 있어야 한다[8].

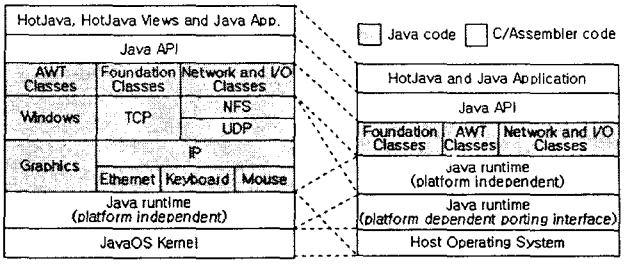

그림 1 호스트 운영체제의 유무에 따른 자바 소프트웨어 아키텍처 상관관계

Fig. 1 Relation of Java software architecture depend on host OS

2.1.3 자바기술을 적용한 OpenPLANET

일본의 시코쿠 전기전력 회사에 의해 개발된 OpenPLANET(Open Platform for Appliances NETworking)은 이미 구축되어 있는 전기배선 인프라를 활용하여, 옥내의 네트워크는 전기배선으로, 옥외의 네트워크는 전화선을 사용하여, 모든 기기의 이질적인 네트워크 환경의 경유로 전세계 어디에서나 컨트롤 할 수 있는 획기적인 '원격 감시 제어 시스템'이다. 이 기술은 에너지 서비스, 설비 자동화 및 모니터링, 보안서비스, 환경 모니터링, 경고 시스템, 교통량 제어, 그리고 원거리 제품 서비스분야등에서 폭넓게 적용할 수 있다. 그림 2는 OpenPLANET 시스템 아키텍처로서, 리얼머신과 버추얼머신은 자바기술을 채택한 OpenPLANET 시스템의 중요한 컴포넌트이다[12].

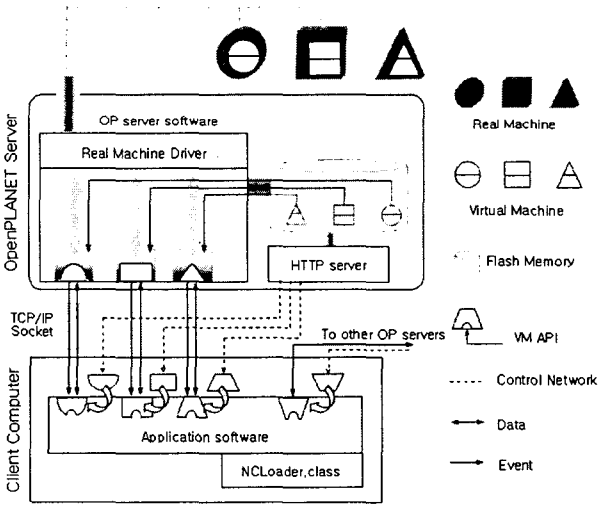

그림 2 OpenPLANET 시스템 아키텍처

Fig. 2 OpenPLANET system architecture

리얼머신은 감시 제어의 대상이 되는 여러 가지 전기기기 제어계 네트워크상에서의 통신기능을 가진 원칩 마이크로컴퓨터를 내장함으로써 정보의 송수신 능력을 갖고 있다. 버추얼머신은 데이터 통신에 의해 실기계와 동일하게 동작하는 가상 기기. 자바 언어로 만들어진 소프트웨어 오브젝트로서, 인터넷 등의 정보계 네트워크상을 자유자재로 이동할 수 있다. 버추얼머신 서버는 OpenPLANET의 소프트웨어를 내장

하고 가상기계 관리나 배송, 액세스 제어, 통신 시큐리티 관리 등을 하는 관리용 퍼스널컴퓨터, 일반적인 시판 퍼스널컴퓨터를 그대로 사용할 수 있는 것 외에 보다 콤팩트화, 저가격화를 지향한 전용 서버를 현재 개발중이다[12].

2.1.4 자바기술의 성능향상을 위한 방법

자바기술의 실시간 문제는 자바명령어의 실행주체가 하드웨어 기계가 아닌 소프트웨어 기계라는 것이다. ARM 사의 Jazelle, aJile 시스템즈사의 aJile 등과 같은 자바 하드웨어 프로세서는 자바 명령어를 직접 수행하여 높은 수행 성능과 실시간적인 연산처리를 지원한다[13,14]. 하지만 산업분야의 현존하는 대상 시스템을 교체한다는 것은 엄청난 노력과 비용을 초래할 뿐만 아니라 개발자들이나 사용자에 의해 많은 검증과 보완을 필요로 한다.

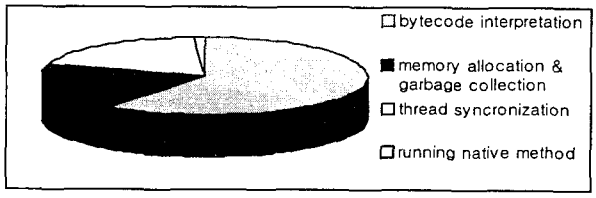

그림 3 자바가상기계의 수행성능을 감소시키는 요인
Fig. 3 Decreasing factors in the execution performance of java virtual machine

데스크탑 시스템에 비해 임베디드 시스템의 성능은 매우 제약적이며, 이러한 임베디드 시스템 환경을 기반으로 구성되어질 자바 런타임 환경은 충분한 성능을 기대하기 힘들다. 그림 3은 자바 소프트웨어 기계 즉 자바가상기계를 적용한 자바 런타임 환경에서의 수행성능을 감소시키는 요인을 보여준다[1,2,3,10,11]. 그림 3에서 보듯이 자바가상기계의 성능을 감소시키는 요인으로 바이트코드 해석, 메모리 할당과 가비지 컬렉션, 쓰레드 동기화, 그리고 네이티브 메소드 실행이다. 자바가상기계의 성능을 떨어뜨리는 가장 큰 요인은 바이트 코드 해석과 메모리 할당 및 가비지 컬렉션이다. 이 세가지 요인의 성능 개선 여부에 따라 자바가상기계의 수행능력 향상을 기대할 수 있다[1,2,3].

먼저 바이트 코드 해석은 자바 프로그램의 코드를 네이티브 코드로 해석하여 실행하는 부분으로 시스템 성능에 상당한 부담을 준다. 가비지 컬렉션의 실행주체인 가비지 컬렉터는 생성된 객체에 대한 메모리를 스캔하고 객체의 사용유무의 정도에 따라 활성화 객체와 비활성화 객체로 분류한다. 비활성화 객체는 낮은 우선 순위의 쓰레드로서 동작되는 가비지 컬렉터의 실행에 의해 메모리로부터 해제된다[1,3].

자바에서의 메모리 관리는 가비지 컬렉션과 관련되어지는데, 가비지 컬렉터의 실행주기의 영향을 받는다. 임베디드 시스템에서 동작하는 애플리케이션의 프로그램 기능은 특정한 작업을 처리하는데 그 목적을 두고 있다. 이런점을 고려한다면, 임베디드 시스템에서의 프로그램 생명주기와 활성화 객체 생명주기는 동일한 경향을 띤다. 비활성화 객체는 다른 컴퓨팅 시스템에서의 자바 런타임 환경에서처럼 존재하지는 않는다. 따라서 대상 시스템에 적용할 임베디드 시스템의 자바 활성화 객체 개수와 가비지 컬렉션 실행주기를 조절한다면 기대 이상의 성능향상을 기대할 수 있다.

2.2 실시간 임베디드 미들웨어 시스템의 설계 및 구현

2.2.1 자바 네이티브 조합모델

본 논문에서 실시간 임베디드 미들웨어 시스템 환경을 구성하기 위해 그림 4에서와 같이 자바와 네이티브의 적절한 조합모델 사용을 제안한다. JNI (Java Native Code) 기술은 동일한 런타임 실행환경을 위한 시스템 통합을 위해서는 바람직하지는 않지만, 특히 산업분야에서의 실시간 수행성능의 중요성을 고려했을 때 활용할만한 모델로서 가치를 지닌다[9]. 자바기술은 전체적인 임베디드 미들웨어 시스템 컴포넌트를 구성하고, 네이티브 모듈은 실시간적인 수행성능에 많은 영향을 미치는 연산처리에 사용한다. 이렇게 함으로써, 다양한 산업분야에서 운영되고 있는 컴퓨팅시스템의 실행환경에 대한 유연성과 동일성을 부여함과 동시에 실행성능을 극대화하고자 한다. 이는 정보통신분야에서의 자바기술에 대한 장점을 최대한 활용하고 산업분야에서의 자바기술의 단점을 상당부분 보완할 수 있기 때문이다. 그림 4는 자바와 네이티브의 시스템 런타임 환경 구성을 위한 조합모델을 제시하고 있다[9].

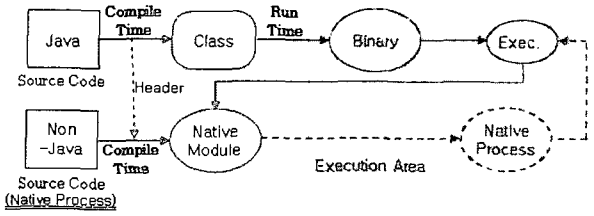

그림 4 자바와 네이티브 조합모델을 위한 실행
Fig. 4 Execution for combination model with java and native

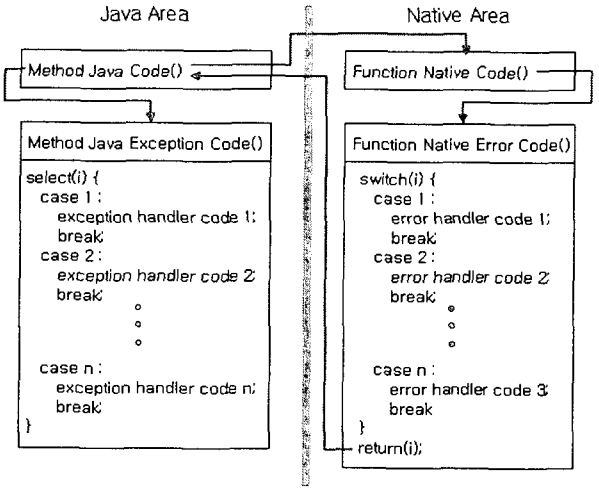

그림 5 자바와 네이티브 실행영역에서의 예외와 에러처리
Fig. 5 The process of exception and error in a java and native execution area

자바와 네이티브의 적절한 조합 모델을 의미하는 JNI 기술은 기본적으로 두가지 의미를 포함하고 있다. 먼저 자바기술

은 시스템의 런타임 환경을 동일화시키는데 주안점을 두는 반면에, 네이티브는 빠른 연산처리를 필요로 하는 구문이나 모듈을 적용하여 시스템의 성능을 상당히 높여주는데 초점을 맞춘다. 이러한 자바와 네이티브 조합모델을 이용한 JNI 기술을 시스템에 적용시 고려해야할 사항으로 예외와 오류에 대한 처리방법이다. 이것은 자바코드와 네이티브 코드의 실행영역이 다르기 때문에 그 관계를 명확히 정의해야한다. 자바 실행영역에서는 비정상적인 실행에 따른 올바르지 못한 수행을 예외로서 처리하고, 네이티브 실행영역에서는 자바 실행영역의 예외처리를 명시적으로 오류코드로서 해결한다. 이러한 자바 실행영역과 네이티브 실행영역 사이의 예외와 오류에 대한 처리를 사전에 정의하지 않을 경우 시스템의 신뢰성을 중요하게 여기는 산업분야에서 대상 시스템에 치명적인 사고의 요인이 될 수 있기 때문이다. 이러한 자바 실행영역과 네이티브 실행영역의 예외와 오류 처리에 대한 실행흐름은 그림 5와 같이 정의한다.

2.2.2 자바 네이티브 조합모델을 이용한 실시간 임베디드 미들웨어 시스템의 설계

실시간 임베디드 미들웨어의 시스템은 하드웨어 시스템과 소프트웨어 시스템으로 구분한다. 그림 6에서 보는바와 같이, 하드웨어 시스템 구성은 세가지 구성요소로서 이루어지며, 시스템 관리 보드(System Management Board : 이하 SMB), 모듈 다운로드 보드(Module Download Board : 이하 MDB) 그리고 실시간 제어 관리보드(Control Management Board : 이하 CMB)이다.

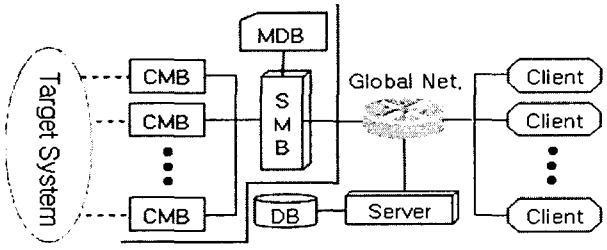

그림 6 실시간 임베디드 미들웨어 시스템의 하드웨어 구조
Fig. 6 Hardware structure for real-time embedded middleware system

먼저 SMB는 설비나 시설의 시스템의 전반적인 컴퓨팅 환경을 관리 및 유지하는 중심적인 구성요소로서, MDB와 CMB 사이를 유기적으로 모듈 연동 및 관리를 위한 임베디드 시스템이다. CMB는 설비 대상 시스템으로부터 제공되어지는 각종 저수준의 데이터를 분석 및 가공하여 고수준의 정보로 변환하여, 시스템의 상태를 제어한다. MDB는 CMB에서 필요로 하는 각종 연산 및 제어 처리 모듈을 제공하는 시스템 모듈 저장소로서 실시간적으로 CMB가 필요로하는 연산처리 및 제어 모듈을 제공한다. 즉 CMB에서 필요한 연산 처리 및 제어 모듈을 SMB에 요청하게 되면 SMB는 MDB의 저장된 연산처리 및 제어 모듈 정보를 다시 CMB에 전송하게 된다. 연산처리 및 제어 모듈정보를 전송받은 CMB는 MDB와 네트워크를 이용하여 실시간적으로 필요한 연산처리 및 제어 모듈을 제공받게 되는 것이다.

실시간 임베디드 미들웨어 시스템의 소프트웨어 구성은 그림 7과 같이 SMB, CMB, MDB에 따라 나누어진다. SMB의 모듈정보요청(Module Information Request)과 CMB 관리(Management) 모듈은 다수의 CMB에서 필요로 하는 연산처리 및 제어 모듈의 정보를 MDB로부터 수신하여 CMB에게 제공할 뿐만 아니라 CMB가 수행하는데 필요한 CMB간의 관계를 관리하는 모듈들이다. CMB의 데이터 분석(Data Analysis), 시스템 진단 및 제어(System Diagnosis & Control), 모듈 다운로드 및 재시작(Module Download & Restart) 그리고 데이터 송•수신(Data Transmit & Receive) 모듈은 대상 시스템의 데이터를 수신하여 시스템의 상태를 분석 및 진단하고, 추가적으로 필요한 모듈을 MDB로부터 다운로드하여 모듈 교체를 통해 보다 효율적인 진단 및 제어 환경을 구축하는 것이다. MDB의 모듈 제공자(Module Provider), 모듈 검색(Module Search) 그리고 모듈 업데이트(Module Update)는 CMB에서 요청되어진 연산처리 및 진단 모듈을 검색하여 MDB에 존재할 경우 CMB에 모듈을 제공하고, 필요한 모듈이 없을 경우에는 시스템 관리자로부터 연산처리 및 제어 모듈을 MDB에 업데이트하여 능동적인 임베디드 미들웨어 시스템을 구현하기 위한 모듈이다.

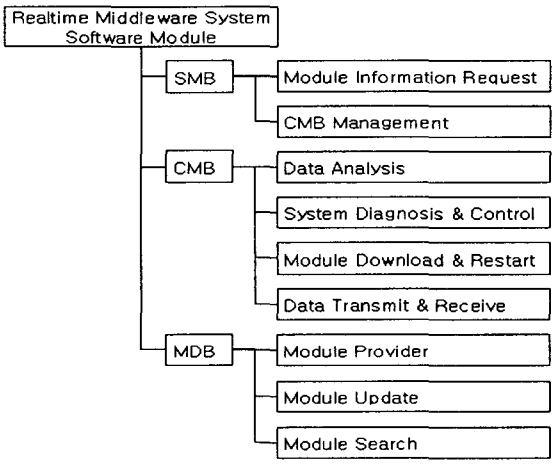

그림 7 실시간 임베디드 미들웨어 소프트웨어 구조
Fig. 7 Software structure for real-time embedded middleware system

그림 8은 실시간 임베디드 미들웨어 시스템의 하드웨어와 소프트웨어 구성요소들을 기반으로 실행되는 시스템 흐름도를 나타낸다. 먼저 관리자나 클라이언트는 SMB에 네트워크 연결을 통해 대상 시스템의 진단 및 제어를 위한 필요한 각종 사용자 인터페이스 모듈을 다운로드하며, 실시간 임베디드 미들웨어 시스템의 사용에 적합한 환경을 자동적으로 설정하게 된다. 관리자나 클라이언트 컴퓨팅 환경의 설정이 완료되면 내부 네트워크 환경에 접속하게 된다. 여기서 외부 네트워크와 내부 네트워크 환경의 차이는 관리자나 클라이언트 컴퓨터 환경의 설정의 유무에 따라 감시 및 진단하고자 하는 대상 시스템의 CMB에 접속할 수 있느냐의 차이다. 관리자나 클라이언트가 이용하고자 하는 CMB는 내부 네트워크에 속해 있기 때문이다. 이렇게 관리자나 클라이언트가 감시 및 진단하고자 하는 CMB의 내부 네트워크 환경에 접속하게 되

면 각각의 대상시스템을 감시 및 진단이 가능하다. 그러면 CMB는 접속된 관리자나 클라이언트에게 대상 시스템의 원시 데이터(raw data)와 함께 각종 진단 모듈에 의해 처리된 정보들을 전송하게 된다. 관리자나 클라이언트는 수신된 정보들을 기반으로 비주얼하게 시스템의 증상 및 이상 유무를 판별 가능하게 된다. 만약 보다 정밀한 진단 알고리즘 모듈을 이용하거나 업데이트된 진단 알고리즘 모듈을 이용할 경우 대상시스템을 감시 및 진단하고 있는 CMB에 요청하게 되면 CMB에서는 SMB에게 진단 및 제어 모듈의 업데이트 정보를 요청하게 된다. 요청된 진단 및 제어 모듈 정보를 수신한 SMB에서는 MDB에게 진단 및 제어 모듈 정보를 전송하게 되며, MDB에서는 전송된 진단 및 제어 모듈 정보를 이용하여 직접 CMB와 내부 네트워크를 통해 CMB에게 필요한 진단 및 제어 모듈을 다운로드 시킨다. 업데이트된 진단 및 제어 모듈의 다운로드가 완료되면 CMB에서는 보드의 하드웨어적인 시스템 재부팅 없이 새로운 모듈의 실행을 시작한다.

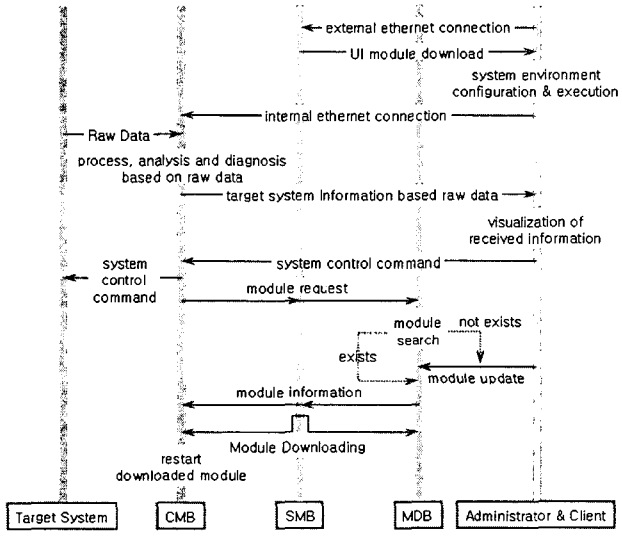

그림 8 실시간 임베디드 미들웨어 시스템 흐름도

Fig. 8 The flow of Real-time embedded middleware system

2.2.3 시스템의 구현 및 성능 실험

본 논문에서 제안하고 있는 실시간 임베디드 미들웨어 시스템의 구현은 네개의 CMB와 하나의 SMB 그리고 MDB로 구성하였으며, 4개의 CMB는 4개의 서로 상이한 대상 시스템을 진단 및 제어하는 모듈이 탑재되어 있다. 각각의 대상 시스템을 감시하고 진단하며 이를 기반으로 제어할 수 있는 CMB, 상이한 대상 시스템을 동질의 런타임 실행환경으로 구성할 수 있는 SMB, 그리고 각 대상 시스템에게 실시간적으로 필요한 각종 진단 및 제어 모듈들을 제공할 수 있는 MDB이다. 이들 여섯 개의 임베디드 시스템 보드들은 서로 다른 하드웨어적인 플랫폼을 사용하고 있으며, 실시간 연산처리 성능을 향상시키기 위해 자바 네이티브 조합모델을 적용한다. 그리고 실시간 임베디드 미들웨어 시스템의 소프트웨어 환경은 자바 네이티브 조합모델을 지원하기 위한 자바 런타임 실행환경을 갖추고 있으며, 네트워크 환경의 연결을 용이하도록 시리얼 통신과 이더넷 통신이 가능한 임베디드 시스템 보드이다.

그림 9는 서로 다른 대상 시스템을 실시간 임베디드 미들웨어 시스템 기반으로 동질의 런타임 실행환경을 구성하여 실험한 모델을 나타낸다.

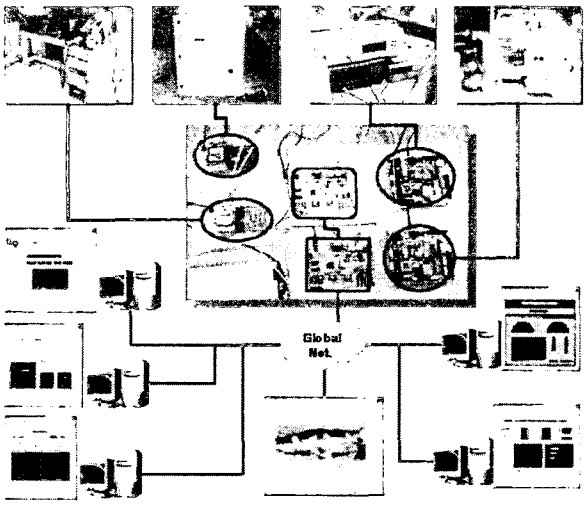

그림 9 실시간 임베디드 미들웨어 시스템 기반 통합 시스템 구현

Fig. 9 The Integration system based on real-time embedded middleware system

실시간 임베디드 미들웨어 시스템의 자바 네이티브 조합모델의 성능실험은 복잡한 진단 알고리즘 연산처리 모듈을 순수 자바코드만을 사용한 경우와 순수 네이티브 코드만을 사용한 경우를 비교 대상으로 한다.

표 1 연산처리 모듈의 구현방법에 따른 실행성능 결과

Table 1 Performance by the implementation method of operation module

Method / The Number of Execution	Native	Java + Native	Java
1st	0	12	322
2nd	0	12	322
3rd	0	10	317
4th	0	10	325
5th	0	9	319
6th	0	9	319
7th	10	9	320
8th	0	9	321
9th	0	10	319
10th	10	9	319
Average(ms)	**2**	**10**	**320.3**

표 1과 그림 10은 실시간 임베디드 미들웨어 시스템에 적

용한 자바 네이티브 조합 모델의 성능을 10회 실험한 결과로서, 자바의 연산처리 성능이 네이티브 연산처리 성능에 비해 상당히 떨어지는 것으로 나타내고 있으며, 자바 네이티브 조합 모델은 자바와 네이티브의 중간정도의 실행성능을 나타내고 있다. 표 1의 Native 구현방법에 의한 실행성능 결과의 '0'이라는 값은 연산처리 모듈을 처리하는데 소비된 시간이 밀리세컨드(millisecond) 이하로 이루어졌기 때문에 나타나는 수치이다.

그림 10은 표 1의 실행성능결과를 바탕으로 도출해낸 자바의 평균 실행성능지수를 1로 보았을때, 자바 대비 네이티브의 실행성능지수는 평균 160배, 그리고 자바 대비 자바-네이티브 조합은 평균 32배의 실행성능지수 결과를 나타내고 있다.

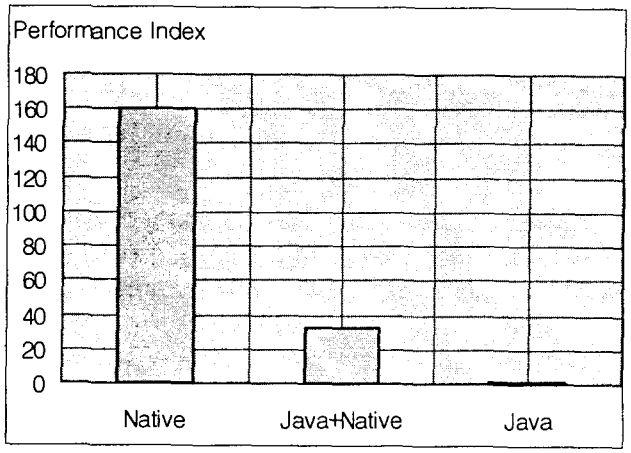

그림 10 모듈구현방법에 따른 실행성능지수

Fig. 10 Performance index by the implementation method of operation module

위 실험의 결과는 네이티브 모듈 방법이 우수한 실행 성능을 보여준다. 하지만 네이티브 모듈은 하드웨어와 소프트웨어에 종속적인 코드들로 구성되어 하드웨어와 운영체제를 포함한 컴퓨팅 소프트웨어에 따른 필요모듈을 개발해야 하는 단점이 있으며, 이는 다양한 산업용 디지털 기기의 컴퓨팅환경 제약조건으로 작용하고 있다. 이러한 문제점을 해결해보고자 특정 하드웨어에 종속적인 컴퓨팅환경을 소프트웨어적으로 가상화 시킨 기술이 자바이다. 자바는 컴퓨팅환경의 가상화기술로 인한 오버헤드로 초래하여 그다지 뛰어난 실행성능을 보여주지는 못한다는 단점을 가지고 있다. 따라서 본 논문에서 제안하고 있는 자바-네이티브 조합 모델을 적용한 실시간 임베디드 미들웨어 시스템은 다양한 대상 시스템으로 인한 다양한 컴퓨팅 환경의 실행환경을 소프트웨어적인 동일한 시스템 환경으로 재구성함과 동시에 실시간적인 실행성능의 가능성을 제시해준다. 이것은 정보통신분야뿐만 아니라 모든 산업분야에서 요원하고 있는 다양한 각종 기기 및 설비에서 운영되고 있는 컴퓨팅 실행환경의 동일화 및 유연한 시스템 통합 그리고 네트워크화를 고려해본다면, 소프트웨어적인 실행환경의 통합과 실시간적인 연산처리 가능성을 보여줄 수 있는 자바-네이티브 조합모델을 적용하는데 보다 큰 의미를 둔다.

3. 결 론

본 논문에서 제안하고 있는 자바 네이티브 조합모델을 이용한 시스템 환경은 자바기술의 네트워크 기능과 플랫폼 독립성, 플랫폼 자체가 가지는 보안성과 안정성을 최대한 활용하면서, 네이티브의 우수한 연산성능을 기반으로 하는 실시간 임베디드 미들웨어이다. 자바 네이티브 조합모델을 바탕으로 다양하고 이질적인 산업용 컴퓨팅 환경과 네트워크 환경을 유연성있게 통합하기 위한 임베디드 시스템을 구축하고, 임베디드 시장에서 각광받고 있는 자바기술을 활용하여 실시간 네트워크 미들웨어 실행환경을 재구성 가능하도록 하는 것이다. 이는 임베디드 시스템 기반 자바기술의 활용으로, 산업기기의 네트워크화를 실현하고, 기기간의 커뮤니케이션과 새로운 기능에 능동적으로 변화 및 대처할 수 있도록 한다. 또한 앞으로 모든 산업분야에 확산될 유비쿼터스 환경으로의 유연한 전환이 가능하게 하는 실시간 임베디드 미들웨어의 필요성은 점차 대두될 것이다.

참 고 문 헌

[1] B. Venners, Inside Java Virtual Machine, McGraw-Hill, 1997.

[2] Tim Lindholm and Frank Yellin, The Java Virtual Machine Specification, ADDISON-WESLEY, 1997.

[3] A. Taivalsaari, Implementation a Java Virtual Machine in the Java programming Language, SUN Lab, 1997.

[4] Greg Bollella, The Real-Time Specification for Java, Addison-Wesley, 2000.

[5] Peter C. Dibble, Real-Time Java Platform Programming, Prentice Hall PTR, March 11, 2002.

[6] Lisa Carnahan and Marcus Ruark, Requirements For Real-time Extensions For the Java Platform, NIST, September, 1999.

[7] Dobb's, The Real-Time Specification for Java, RTSJ, February, 2000.

[8] Peter W. Madany, JavaOS:A Standalone Java Environment, JavaSoft, 1996.

[9] Dkramer, Java Native Interface Specification, SUN Microsystems, 1997.

[10] http://Java.sun.com/, Sun Microsystems, Java Home Page.

[11] http://www.Javaworld.com/, Java World COMPANY.

[12] http://www.openplanet.co.jp/, OpenPlanet Project, Shikoku Electric Power COMPANY.

[13] http://www.arm.com/products/solutions/Jazelle.html, ARM Jazelle Technlogy, ARM COMPANY.

[14] http://www.ajile.com/aj100.htm, aj-100 embedded low power java microprocessor, ajile SYSTEM.

저 자 소 개

정 준 영(鄭 峻 永)

1999년 경남대 전산통계학과(학사).
2001년 동 대학원 컴퓨터공학과(석사)
2001년~현재 동 대학원 컴퓨터공학과 박사과정
E-mail : mrj@hawk.kyungnam.ac.kr

김 광 수(金 光 手)

1983년 서울대 전기공학과(학사)
1985년 동 대학원 전기공학과(석사)
1998년 Texas A&M Univ., Electrical Eng. Ph.D.
1986년~현재 한국전기연구원 책임연구원
E-mail : kskim@keri.re.kr

정 민 수(鄭 民 手)

1986년 서울대 컴퓨터공학과(학사)
1988년 한국과학기술원 전산학과(석사)
1994년 한국과학기술원 전산학과(박사)
1990년~현재 경남대 정보통신공학부 교수
E-mail : msjung@eros.kyungnam.ac.kr