□ 기술해설 □

실시간 자바(Real-Time Java)

경희대학교 김형일·이승룡*·배수강

1. 서 론

자바는 분산 환경에 적합하고, 코드 재사용이 용이하며, 이식성이 높고, 프로그래밍이 단순하다는 장점을 가지고 있다. 이러한 이유로 자바는 인터넷 브라우저 환경뿐 아니라 데이타베이스 연동, 내장 시스템, 전자 화폐에 이르기까지 다양한 분야에서 적용되고 있다.

Sun사에서는 이미 EmbeddedJava와 PersonalJava[2]를 발표하고, 실시간 운영체제 개발 회사들에게 라이센스를 제공하여 자바 가상 머신을 포팅하는 작업을 진행하고 있다. 내장 자바가 실시간 운영체제 상에서 구현되는 이유는 대부분의 내장 시스템이 시간적 제약을 가지는 실시간 시스템이기 때문이다. 실시간 시스템은 계산의 논리적 정확성뿐 아니라 계산 결과를 마감시간 내에 도출하는 엄격한 시간적 제약을 가진 시스템이다. 이러한 특성을 지닌 실시간 시스템을 개발하기 위해서는 최악의 실행시간에 대한 분석이 가능해야 하며, 이를 바탕으로 주어진 마감시간 안에 처리할 수 있는 기능을 가져야 한다.

그러나, 현재의 자바 가상 머신은 하드웨어 중립적인 바이트 코드를 기반으로 하여 이기종 컴퓨터간의 호환성은 증진되었으나 네이티브 코드에 비하여 속도가 느리며, 쓰레드의 동기화 성능이 떨어지고, 가베지 콜렉터와 JIT (Just-In-Time) 컴파일러에 의한 예측할 수 없는 지연이 발생하는 등 시간적 요구 사항을 만족하기 어려운 문제들을 가지고 있다[6]. 따라서, 자바가 실시간 시스템에 적용되려면 실시간 자바 가상 머신을 새로 설계하여 구현해야 하며 언어적 측면에서도 추가적인 고려사항이 필요하다. 이러한 관점에서 본 논문은 실시간 시스템을 위한 실시간 자바의 연구동향을 소개한다.

본 논문의 구성은 다음과 같다. 2장에서는 실시간 프로그래밍 언어의 연구동향과 실시간 시스템의 입장에서 자바의 장단점에 대하여 살펴보고, 3장에서는 Newmonics사의 Nilsen [10]과 일본 게이오대학의 Tokuda팀[7]의 실시간 자바 확장 방법에 대한 연구를 소개하며, 4장에서 결론을 맺는다.

2. 실시간 프로그래밍 언어와 자바

본 장에서는 실시간 프로그래밍 언어의 연구동향을 소개하고 실시간 시스템 입장에서 자바의 장단점에 대하여 알아본다.

2.1 실시간 프로그래밍 언어 연구동향

실시간 시스템은 일반적으로 외부적인 이벤트와 주기적인 타이머의 동작을 통해서 계산 작업이 수행된다. 따라서, 하드웨어적으로 입출력 이벤트 및 타이머 처리가 가능해야 하며, 이를 프로그래밍 언어로 나타낼 수 있어야 한다. 시간적 제약을 만족시키기 위해서는 계산 작업에 소요되는 시간과 필요 자원을 정확하게 알아야 한다. 그러나, 시간에 대한 문제는 CPU의 계산 속도뿐 아니라 메모리 접근 속도, 입출력 속도와 지연 등 다양한 문제가 한꺼번에 영향을 미치기 때문에 간단하지 않다. 예를 들면, CPU의 캐쉬는 분기점에서의 예측 결과

* 종신회원

에 따라 계산 속도가 달라지며, 입출력 작업은 외부적인 작용에 의해서 지연이 결정되므로 정확한 계산 시간을 측정하기란 거의 불가능하다. 그러므로, 실시간 시스템은 런타임 시의 정확한 계산 시간을 바탕으로 구현되는 것이 아니라, 최악의 실행 시간을 바탕으로 시간 제약을 만족하도록 구현된다. 이를 위하여 실시간 프로그래밍 언어는 시간 제약을 명시적으로 표현할 수 있는 구조를 가져야 하며, 아울러 실시간 프로그램의 예측성과 스케쥴링 가능성을 증진시킬 수 있는 컴파일러와 강력한 도구 개발이 수반되어야 한다.

현재 실시간 프로그래밍 언어에 대한 연구로는 Toronto 대학에서 개발한 Real-Time Euclid[13], [14]가 있다. 이는 실시간 구조와 컴파일 시간에 시간적 제약을 가진 소프트웨어를 검증할 수 있는 스케쥴 가능성 분석 기능을 가지고 있다. Real-Time Euclid는 예측성을 확보하기 위하여 루프는 일정한 수치의 카운트를 가지고, 재귀호출과 동적 변수가 허용되지 않으며, 예외 처리기를 비롯한 대기와 장치 조건 변수들은 모두 제한된 시간 범위 내에 있다.

RTC^{++}[3]은 Carnegie Mellon 대학에서 개발한 언어로써 C^{++}에 실시간 객체 모델을 첨가해 확장한 것이다. RTC^{++}는 활동중인 객체를 최악의 실행시간(Bound), 마감시간(Within)과 같은 시간 제약들을 가지도록 정의할 수 있다. RTC^{++}은 스케쥴 가능성 분석을 위해 스케쥴링 모델로써 rate monotonic(RM) scheduling을 채택하였으며, 동시에 발생하는 쓰레드가 자원을 공유할 때의 우선 순위 역전문제를 제어하기 위해 우선 순위 계승프로토콜을 채택하였다.

Flex[4],[5]는 Illinois대학에서 명시적인 시간제약과, imprecise computation, 단일 계산의 다중구현을 지원할 수 있도록 개발된 실시간 언어이며, C^{++}에서 유도되었다. Flex 컴파일러는 C^{++}컴파일러로 컴파일될 수 있는 C^{++} 전처리기로 구성되어 있다.

2.2 실시간 시스템에서 본 자바의 장단점

실시간 시스템의 요구사항에 비춰볼 때 자바는 여러 가지 장단점을 가지고 있으며 이는 표 1에 나타나 있다.

표 1 실시간 시스템 환경에서 본 자바의 장단점

장 점	단 점
• 프로그래밍 언어 차원에서 쓰레드 지원으로 사용이 간편함 • 바이너리 수준의 호환성 보장 • 프로그래밍 시의 오류 최소화 • 네트워크 기본 지원 • 하드웨어 대량 생산을 통한 개발비용 절감	• 쓰레드의 우선순위 역전 현상 발생 • 쓰레드 간의 간섭으로 인하여 정확한 동작 불가 • 자바 가상 머신으로 인한 속도 저하 • 가베지 콜렉터와 JIT 컴파일러에 의한 예측할 수 없는 지연 발생 • 메모리를 직접 제어할 수 있는 명시적인 자원 관리 방법이 제공되지 않음

3. 실시간 자바의 확장 방법

자바는 가상 머신을 기반으로 동작하기 때문에 다양한 플랫폼 하에서 동작할 수 있다. 마찬가지로 실시간 자바는 그림 1과 같이 실시간 하드웨어 플랫폼, 일반 하드웨어 플랫폼, 실시간 운영체제 환경, 일반 운영체제 환경 등 다양한 환경에서 동작할 수 있다. 그림 1-1은 실시간 운영체제 위에 실시간 자바 가상 머신을 구현한 형태이며, 그림 1-2는 실시간 운영체제 위에 Sun사의 내장 자바를 지원하는 자바 가상 머신을 구현한 형태이다. 또한, 그림 1-3은 실시간 자바를 지원하는 하드웨어 위에 실시간 자바 가상 머신을 바로 구현한 형태이며, 그림 1-4는 일반 운영체제와 실시간 자바 가상머신 위에서 실시간 자바 애플리케이션을 수행하는 형태이다.

실시간 자바는 주로 세 가지 다른 관점에서 연구가 진행되고 있다. 첫번째는 새로운 실시간 자바 플랫폼을 설계하는 방법으로 실시간

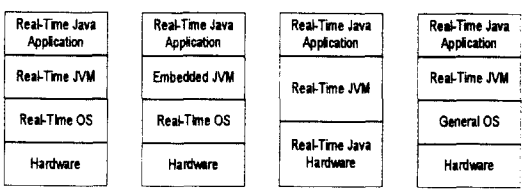

그림1-1 그림1-2 그림1-3 그림1-4

그림 1 실시간 자바의 확장 구조

가베지 콜렉션을 지원하는 자바 가상 머신인 PERC[11]를 개발한 Newmonics사의 Nilsen 팀이다(그림 1-4 방법). 두번째 실시간 프로그래밍 언어적인 측면에서 Real-Time Mach을 기반으로 하여 실시간 자바를 개발하는 일본 게이오대학의 Tokuda 연구팀이며(그림 1-1 방법), 세번째는 Sun사의 내장 자바를 기반으로 한 기존 실시간 운영체제 개발사들을 중심으로 자바 가상 머신을 자신들의 실시간 운영체제에 동작하도록 포팅 하는 것이다(그림 1-2 방법). 본 장에서는 이러한 접근 방법 중 Nilsen과 Tokuda팀의 연구를 소개한다.

3.1 Nilsen의 접근 방법 [10], [11]

Nilsen은 실시간 가베지 콜렉터를 이용하여 실시간 자바 가상 머신을 만들고, 자바 언어에 timed와 atomic 키워드를 추가하여 시간적인 제약 조건을 명시하는 새로운 스펙을 제안하였다. 또한, 기존의 자바와 호환성을 고려하여 추가된 키워드를 표준 자바 언어로 바꾸어주는 방법도 제안하였다. Nilsen은 실시간 자바 API를 새로 설계하였는데, 여기에는 시간 제약을 나타내고 분석하는 일련의 도구들이 포함되어있다. 특히, 자바 가상 머신의 경우 실시간 제약 조건을 충분히 만족할 수 있도록 실시간 가베지 콜렉터와 정밀한 시간 제어가 가능한 하드웨어를 바탕으로 구현하는 것이 바람직하다고 하였으며, 실시간 자바는 일반 운영체제에서 소프트웨어적인 방법으로 해결할 수 있는 문제는 아니라고 하였다. 또한, 바이트 코드 생성시 최악의 실행시간을 계산할 수 있는 장치가 필요하고 표준 자바 환경에서 동작하는 바이트 코드는 바이트 코드 분석기에 의해서 요구되는 시간 안에 동작할 수 있는 지를 분석할 수 있어야 한다고 하였다. 이 절에서는 Nilsen이 제안한 실시간 자바 구조와 이를 구성하고 있는 구성 요소들을 살펴본다.

• 실시간 자바 구조

Nilsen이 제안한 실시간 자바 프로그램은 쓰레드로 동작하는 여러 개의 실시간 액티비티(real-time activity)로 구성되어 있다. 실시간 액티비티는 실시간 태스크들로 구성되어 있으며, 실시간 태스크는 실시간 executive로도 불리운다. 실시간 액티비티는 구성 관리자(configuration manager), 총괄 관리자(administrator)와 함께 동작한다. Nilsen이 제안한 실시간 자바의 구조와 동작 순서는 그림 2와 같다.

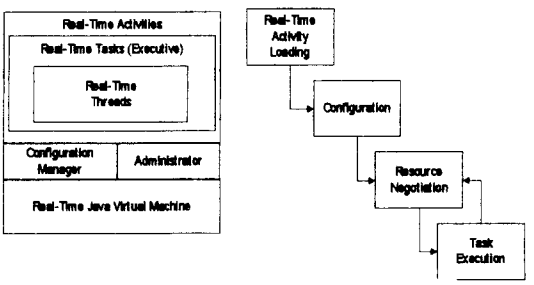

그림 2 실시간 자바의 구조

• 실시간 액티비티(Real-Time Activity)

같은 목적으로 동작하는 태스크의 집합을 실시간 액티비티라고 하며, 이는 실시간 태스크, 구성관리자, 총괄 관리자로 구성되어 있다. 실시간 쓰레드는 공유 변수를 가지고 있는 일종의 태스크라고 할 수 있다. 실시간 태스크에는 성격에 따라 cyclic, spontaneous, ongoing으로 나뉜다. 스케쥴러는 기본적으로 고정 우선순위인 RM 스케쥴링을 이용하며, 다른 스케쥴러를 사용자가 추가하여 사용할 수 있다. 실시간 태스크 환경에서는 실시간 태스크의 우선순위는 모든 비실시간 프로세스에 비하여 높게 책정된다.

• 구성관리자(Configuration Manager)

구성 관리자는 JIT 컴파일러를 통하여 자바 바이트 코드를 런타임 시 네이티브 코드로 바꾸고, 태스크의 실행시간을 계산하고, 동작에 필요한 메모리 요구사항을 파악한다. 구성 관리는 필요 시 태스크 사이의 블로킹을 분석하여 이를 처리한다. 실시간 자바 소프트웨어 구조에 따라 관련이 있는 태스크들 사이에서만 블로킹이 발생하게 된다.

• 총괄 관리자(Administrator)

총괄 관리자는 각각의 실시간 액티비티에 필요한 자원 요구량에 따라 실시간 executive의 CPU 시간과 메모리 사용량을 계산하고, 이를 바탕으로 실행 시간과 동적 메모리에 하여 실시간 executive와 협상을 진행한다. 또한, 초

기의 자원을 할당한 후에도 개별 태스크들은 자신의 적당한 자원을 사용하도록 조정할 수 있다. 자바는 동적으로 수행되므로 자원의 크기는 계속적으로 파악되어야 한다. 이 경우에는 총괄 관리자는 다시 개별 태스크와 자원 사이의 협상을 수행한다.

• 실시간 태스크 수행

모든 실시간 태스크는 초기화 코드와 마감 코드, 수행 컴포넌트로 구성된다. 초기화 코드와 마감 코드는 반드시 실행시간 분석이 가능해야 하고, 이들 세그먼트를 실행시키기 위한 최소한의 요구 시간을 알아야 한다. 수행 컴포넌트의 실행 시간은 측정에 의해 알 수 있으며, 할당된 실행 시간은 보장할 수 있는 실행시간과 보장할 수 없는 실행시간의 합으로 주어진다.

• 블로킹 (Blocking)

Nilsen은 실시간 시스템의 시간적 제약을 명시하기 위하여 timed와 atomic 키워드를 자바에 추가하였다. timed는 시간 안에 코드 블럭이 수행되지 않으면 마감시간을 위반하는 태스크로 판단하고 이에 대한 처리를 할 수 있도록 지원한다. atomic은 코드 블럭이 마치 하나의 명령어처럼 동작하게 한다. 이렇게 함으로써 코드 블럭이 수행 중에 중지되었을 경우 치명적인 문제가 발생하는 것을 막을 수 있다.

timed와 atomic은 표준 자바에서 지원하는 것이 아니기 때문에 실제 컴파일 시 전처리기인 RTJPP(real-time java pre-processor)에 의해서 표준 자바 코드로 바꾸어 처리되거나, 실시간 자바 컴파일러로 컴파일되어 실시간 자바 가상 머신인 RT-JVM 위에서 동작할 수 있다.

• 실시간 가베지 콜렉터

가베지 콜렉터는 메모리 관리를 자동화하여 프로그래머에게 메모리 관리의 부담을 덜어주며, 메모리 관련 버그를 줄일 수 있다. 그림 3은 기본적인 가베지 콜렉터의 동작 순서를 나타낸다.

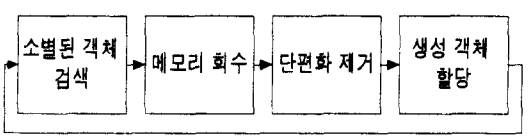

그림 3 가베지 콜렉터의 동작 순서

이상적인 가베지 콜렉터는 객체가 소멸할 때 바로 메모리 회수와 동시에 단편화 제거(defragment)가 이루어지는 것이지만 실제는 가베지 콜렉터가 먼저 소멸된 객체를 인식하는 작업이 진행된 후에 단편화 제거가 수행될 수 있다. 논리적으로는 할당 가능한 메모리가 존재함에도 불구하고 메모리가 회수되지 않거나 전체 할당 가능한 메모리의 합이 할당할 메모리 량보다 크지만 단편화가 제거되지 않았기 때문에 할당할 수 없는 경우가 발생하게 된다. 따라서, 문제는 어떻게 메모리 회수와 단편화 제거가 메모리 할당이 요구되는 객체 생성 이전에 완료되도록 할 수 있는가에 초점이 맞추어져 있다.

한편, 실시간 가베지 콜렉션의 목표는 어플리케이션이 메모리 할당을 요구한 시점에서 실시간 제약성을 가진 다른 태스크의 실행에 방해를 주지 않으면서 메모리 할당이 가능하도록 하는 데 있다[8], [9].

실시간 가베지 콜렉터를 구현하는 경우에는 소프트웨어적인 방법과 하드웨어적인 방법이 있다. 대부분의 경우는 소프트웨어적인 방법을 사용하지만, 하드웨어적인 방법의 경우에 얻을 수 있는 이득도 많다. 물론 하드웨어적으로 지원이 되면 그 만큼의 추가적인 비용이 들지만, 구현속도가 빠르고 가베지 콜렉션 작업 자체에 소요되는 메모리가 필요하지 않게 되어 이로 인한 이득이 더 큰 경우도 있다. 참고로 Nilsen에 의하면 하드웨어로 구현한 경우 전체 시스템에 대하여 약 30~50%의 성능향상이 있었다[10].

2.2 Tokuda의 접근 방법[6]

Tokuda는 마이크로 커널 기반의 실시간 Mach를 이용하여 상위 레벨에 자바 가상 머신인 자바 서버 개발 연구를 진행하고 있다. 이 연구는 크게 1단계인 roast, 2단계인 grind, 3단계인 brew로 나뉜다. 1단계에서는 실시간 자바 쓰레드와 자바 가상 머신에서 실시간 동기화를 구현하고, 마감시간을 넘긴 경우 이를 처리할 수 있는 메카니즘을 제안했다. 2단계는 사용자 수준의 실시간 쓰레드와 동기화 메카니

즘을 RTC-쓰레드 패키지를 이용하여 구현하고, 우선 순위를 지원하는 네트워크와 사용자 수준의 소켓을 지원하도록 하며, ARTIFACT [12]를 이용한 실시간 윈도우 시스템을 제공하는 것을 목표로 하고 있다. 3단계는 실시간 가베지 콜렉터를 설계하고 실행 시 최악의 실행시간을 분석할 수 있는 도구와 승인 제어 메카니즘을 구현하는 것이다. 이 연구는 현재 1단계를 완료하고 2단계의 연구를 진행중이다. 이 절에서는 실시간 쓰레드, 동기화 문제, 그리고 API에 대하여 알아본다.

• 실시간 쓰레드

Tokuda 연구팀이 제안한 실시간 쓰레드는 RT-Mach[15] 위에서 커널 수준의 서버로 구현되었다. RT-Mach은 Carnegie Mellon 대학에서 개발한 Mach 3.0 마이크로 커널을 실시간 운영체제로 확장한 것으로 실시간 쓰레드, 실시간 스케쥴러, 실시간 동기화, 실시간 IPC 그리고, 프로세서 자원 예약 기능들을 제공한다.

자바 서버는 RTS(Real-Time Server)[7]를 기반으로 만들어진 사용자 수준의 서버이다. RTS는 단순한 객체 기반의 서버로 태스크 관리, 파일 관리, 네임 서비스, 예외 처리 등의 기능을 가지고 있다. 자바 서버는 RTS를 확장한 것으로 자바 가상 머신 뿐 아니라 네이티브 코드도 실행이 가능하다. 파일 시스템은 하나의 in-memory 파일 시스템과 다양한 파일 시스템을 부착할 수 있으나 실제로는 자바 메소드가 실행될 때에 디스크 블로킹을 막기 위해 Unix 파일 시스템으로부터 필요한 파일을 in-memory 파일 시스템에 복사하여 초기화한 뒤 동작한다. 실시간 쓰레드란 빠른 속도의 쓰레드를 의미하는 것이 아니라, 주어진 최악의 실행시간 안에 예측성 있게 동작할 수 있는 쓰레드를 의미한다. 그것은 일반 쓰레드가 예기치 않은 블로킹으로 인하여 마감시간 안에 처리될 수 없는 경우가 발생하기 때문이다. RT-Mach에는 커널 레벨의 RT-Threads와 사용자 레벨의 RTC-Thread가 존재하는데 실시간 쓰레드는 RT-Thread를 이용하였으며, 쓰레드의 우선순위는 일반 우선순위, house-keeping, 실시간 우선순위로 나뉜다. 실시간 쓰레드의 동작은 그림 4의 모델과 같이 메인 쓰레드가 마감

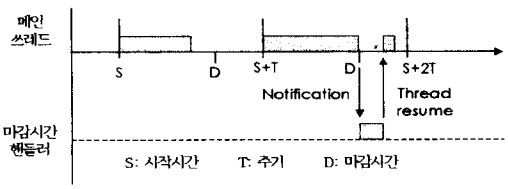

그림 4 실시간 쓰레드 모델

시간을 넘기게 되면 타이머가 이를 감지하고 마감시간 핸들러에 의해서 처리를 한 뒤에 메인 쓰레드가 나머지 작업을 수행하도록 되어 있다.

• 동기화

실시간 동기화를 제공하기 위해서, RT-Mach가 제공하는 rt-mutex-lock과 rt-mutex-unlock을 이용하였다. 따라서, 우선순위 역전 현상을 해결할 수 있는 우선순위 계승 프로토콜을 이용할 수 있었다. 자바가 기본적으로 제공하는 synchronized 키워드는 객체의 내포되어 있는 로킹을 허용하지만, RT-Mach의 rt-mutex-lock과 동일한 의미와 기능을 제공하진 않는다. 이러한 연유로 실시간 쓰레드의 동기화는 마감시간 핸들러에 의해서 교착상태가 발생하게 된다. 그러나, 이같은 교착상태는 동기화 블럭에서 마감시간을 넘기더라도 마감시간 핸들러를 수행하지 못하게 하여 쉽게 방지할 수 있다.

• API

실시간 쓰레드를 위한 API는 Thread를 계승하여 실시간 지원을 위한 RtThread 클래스와 이를 지원하는 RtHandler와 Time 클래스로 구성되어 있다. 실시간 자바 쓰레드는 우선순위 P, 시작시간 S, 주기 T, 그리고, 마감시간 D를 초기값을 지정하여 사용하고, 다음과 같은 메소드를 가지고 있다.

```
RtThread rtth = new RtThread(P, S, T, D);
rtth.setHandler(RtHandler handler, String meth);

〈RtThread의 메소드〉
void setAttr(int priority, Time start, Time period,
     Time deadline)
Time getAttrStart( )
Time getAttrPeriod( )
Time getAttrDeadline( )
void RtThread.setHandler(RtHandler handler, String
     meth)
```

RtThread를 지원하기 위해서 추가된 RtHandler와 Time 클래스는 각각 마감 시간을 넘긴 태스크가 발생할 경우에 예외처리를 위하여 존재하고, Time 클래스는 시간을 초 단위와 *ns* 단위로 표현하는 기능을 갖고 있다. 또한, 하나의 RtHandler는 여러 개의 실시간 쓰레드에 대한 예외 처리 핸들러로 사용할 수 있다.

4. 결 론

본 논문에서는 실시간 자바의 연구동향에 대하여 살펴보았다. Nilsen의 방법은 내장 시스템을 위한 자바 가상 머신을 만들고, timed와 atomic을 추가하여 실시간 제약조건을 쉽게 표현할 수 있도록 한 것으로 실용적인 측면을 강조하였다. Tokuda는 RT-Mach에서 예측성 있고 안정적으로 동작하는 실시간 자바 쓰레드를 만들었다. 이와같이 실시간 자바에 대한 연구는 자바의 가장 큰 장점인 이식성과 단순성을 유지하면서 시간적 제약조건을 만족시키기 위한 기능을 확장하는 방향으로 나아가고 있다.

실시간 자바가 기존의 실시간 시스템의 개발 도구를 대치할 수 있기 위해서는 실시간 쓰레드와 가베지 콜렉터 등이 하드웨어적으로 구현되는 것 뿐 아니라 기존의 실시간 운영체제와 일반 운영체제에서도 구현될 수 있어야 할 것이다. 또한, 실시간 자바로 구축된 시스템이 예측성과 QoS(Quality of Service)를[1] 효과적으로 지원하기 위해서는 CPU나 메모리와 같은 자원에 대한 관리 방법에 대한 연구가 더 진행되어야 할 것이다.

현재로는 실시간 자바가 예상보다 좋은 성능을 나타내지 못하여 개발자들의 폭넓은 관심을 유도하는데 성공적이지 못하였다는 시각도 있다. 하지만, 그것은 다른 개발 도구나 플랫폼이 초창기부터 만족스러운 평가를 얻지 못한다는 일반적인 경향에 해당되는 것으로써 크게 우려할 바는 아니다.

참고문헌

[1] A. Campbell, G. Coulson, D. Hutchison, "A Quality of Service Architectures", ACM SIGCOMM Computer Communication Review, April 1994.

[2] R. Cook, "Java Embeds Itself in the Control Market", http : //www.javaworld.com/javaworld/ jw-01-1998/jw-01-embedded.html, JavaWorld, January 1998.

[3] Y. Ishikawa, H. Tokuda, and C.M. Mercer, "An Object-Oriented Real-time Programming Language", IEEE Computer, 25(10) : pp. 66-73, October 1992.

[4] Kelvin B. Kenny and K. J. Lin, "Building Flexible Real-Time Systems Using the Flex Language", IEEE Computer, pp. 70-78, May 1991.

[5] K. J. Lin and S. Natarajan, "Expressing and Maintaining Timing Constraints in FLEX", In Proceedings of 9th Real-Time Systems Symposium", pp. 96-105, December 1988.

[6] A. Miyoshi and T. Kitayama and H. Tokuda, "Implementation and Evaluation of Real-Time Java Threads", In Proceedings of the 18th IEEE Real-time Systems Symposium", pp. 166-175, December 1997.

[7] T. Nakajima, T. Kitayama, and H. Tokuda, "Experiments with Real-Time Servers in Real-Time Mach", In Proceedings of USENIX 3rd Mach Symposium, 1993.

[8] K. Nilsen and W. J. Schmidt, "Preferred Embodiment of a Hardware-Assisted Garbage-Collection System", Technical Report ISU TR-92-15, November 1992.

[9] K. Nilsen and W. J. Schmidt, "Cost-Effective Object-Space management for Hardware-Assisted Real-Time Garbage Collection", Letters on Programming Language and Systems, 1(4) pp. 338-354, December 1992.

[10] K. Nilsen, "Issue in the Design and Implementation of Real-Time Java", Iowa State University, Center for Advanced Technology Development : Ames, 1995.

[11] K. Nilsen and S. Lee, "PERC Real-Time API (Draft 1.2)", NewMonics Inc., July 1997.

[12] J. Sasinowski and J. Strosnider. "ARTIFACT : A Platform for Evaluating Real-Time Window System Designs", In Proceedings of the 16th IEEE Real-time Systems Symposium, December 1995.

[13] A. D. Stoyenko, 'A Schedulability Analyzer for Real-Time Euclid", In Proc. of 8th Real-Time Systems Symposium, pp. 218-227, December 1987.

[14] A. D. Stoyenko and W. A. Halang, "High-integrity pearl : A Language for Industrial Real-Time Applications", IEEE Software, 10(4) : pp. 65-74, July 1993.

[15] H. Tokuda, T. Nakajima, and P. Rao, "Real-Time Mach : Towards a Predictable Real-Time System", In Proceedings of USENIX Mach Workshop, October 1990.

김 형 일

1994 경희대학교 물리학과 이학사
1996 경희대학교 전자계산공학과 석사
1996~현재 경희대학교 전자계산공학과 박사과정 재학중
관심분야 : 실시간 시스템, 멀티미디어 시스템, 자바

이 승 룡

1978 고려대학교 재료공학과 공학사
1986 Illinois Institute of Technology 전산학과 석사
1991 Illinois Institute of Technology 전산학과 박사
1992~1993 Governors State University, Illinois 조교수
1993~현재 경희대학교 전자계산공학과 부교수
관심분야 : 실시간 시스템, 실시간 고장허용시스템, 멀티미디어 시스템

배 수 강

1997 경희대학교 전자공학과 공학사
1997~현재 경희대학교 전자계산공학과 석사과정 재학중
관심분야 : 실시간 시스템, 실시간 가베지 콜렉션, 자바