이동 내장형 미들웨어와 개발 툴셋†

경희대학교 이승룡* · 허서경 · 서영준** · 송영재*

1. 서 론

포스트 PC, 정보가전, 이동 단말시스템과 같은 내장형 모바일 컴퓨팅 시스템이 차세대 정보통신 산업의 주력으로 부상하게 됨에 따라 이동 내장형 시스템의 기술 확보가 중요한 이슈로 대두되고 있다. 내장형 시스템은 대규모 시스템의 일부로서 내장된 컴퓨터로, 일반적으로 센서나 액츄에이터(actuators)를 통해서 실세계와 직접 상호작용을 한다. 내장형 시스템을 개발하기 위해서는 도메인 지식, 실시간 운영체제 및 미들웨어, 최적화, 계산이론, 병행성, 컴파일러, 잘 정의된 네트워크 프로토콜, 소프트웨어 설계기법 등을 동시에 고려해야만 하는데, 지금까지 내장형 시스템 개발은 주로 도메인 지식을 가진 전문가가 내장형 소프트웨어를 개발하는 형태로 진행되어 왔다[1].

내장형 시스템은 도처에 편재된(ubiquitous) 형태로 구성되어 있으며 자원의 제약이 심한 환경하에서 작동되고, 저 가격(minimal product cost), 빠른 시장 출시(quick time-to-market)등의 시장요구사항을 만족해야 된다. 그리고, 신뢰성, 안전성, 고장허용, 보안성, 저전력, 이동성, 실시간성, 융통성, 상호 운용성(interoperability), 이식성 등의 기능을 지원해야 하며 멀티미디어 서비스 제공과 함께 사용의 편의성을 요구받는다. 따라서, 분산되고 편재되어있는 환경에서 다양한 내장형 시스템들간의 상호 운용성, 플랫폼 독립성과 이식성을 지원할 수 있는 환경을 제공하는 내장형 미들웨어의 중요성이 부각되고 있다.

미들웨어 기술은 응용 소프트웨어 설계자에게 상위 수준의 추상화를 제공하며 분산환경의 이질적인 특성에서 비롯된 복잡성을 은폐하기 위하여 만들어졌으며, 보통 운영체제와 응용 프로그램 사이에 존재하는 소프트웨어이다. 이러한 미들웨어 기술들은 유선 네트워크상의 정적인 분산 시스템 환경에서는 성공적으로 사용되어 왔지만 이동 내장형 시스템에 이를 바로 적용하기에는 많은 난제들을 가지고 있다. 그 이유로는 첫째, 분산 트랜잭션, 오브젝트 요청이나, RPC(Remote Procedure Call) 등은 컴포넌트들간에 항상 가용한 높은 대역폭의 네트워크 연결을 가정하지만, 이동 내장형 시스템들은 단절(unreachability)과 낮은 네트워크 대역폭을 표준으로 고려한다. 둘째, CORBA와 같은 객체 지향 미들웨어 시스템은 대부분 동기화된 P2P(Peer-to-Peer) 통신을 지원하지만, 내장형 시스템은 저전력 제약으로 인하여 서비스 지역 이탈과 같은 부득이한 단절 등으로 인하여 클라이언트와 서버간의 연결이 동일한 시간에 이루워지지 않는다. 셋째, 전통적인 분산 시스템들은 안정적이며, 높은 대역폭, 고정된 위치, 잘 알려진 서비스와 같은 정적인 실행환경에서 작동되는 것을 가정하지만, 이동 내장형 시스템들은 계속되는 디바이스 위치 변경, 이동 중 새로운 서비스 발견과 같은 동적인 환경에서 실행된다. 따라서, 전통적인 상용 미들웨어인 CORBA나 DCOM등을 이동 내장형 시스템에 적용하기에는 위에서 언급한 바와 같은 문제점들과 함께 너무 무겁다(heavy)는 제약을 가지고 있다.

한편, 내장형 시스템의 특성상 개발자들은 충분히 필드에서 검증된 환경만을 사용하려는 보수적인 성향을 가지고 있다. 또한, 내장형 시스템 시장은 인텔사의 프로세서나 마이크로소프트사의 윈도우 운영체제처럼 어느 하나의 승자에 의해 전체 시장이 주도되는 것이 아니라 다양한 목적과 시장의 요구에 따라 각기 특화된 형태의 기술과 제품이 각각 해당 표준화

† 본 연구는 한국과학재단 목적기초연구(과제번호 : R01-2001-000-00357-0)에 의해 일부 지원받았음.

* 종신회원
* 학생회원

단체를 통해 경쟁적으로 개발되고 공존될 것으로 예상된다. 그리고, 내장형 시스템의 개발은 각 목적에 적합한 형태로 개발되어야 하는데 다양한 디바이스에 맞는 미들웨어를 각기 따로 개발하는 것은 지극히 어려운 일이다. 이를 위해서는 미들웨어의 핵심부품들은 빠르게 구축되어야 하고 제품의 품질 개선이 용이해야하며 부품의 재사용이 가능하여 개발비용을 절감해야 되고, 소프트웨어의 증가하는 복잡도를 감소시킬 수 있는 소프트웨어 개발 기술이 요구되고 있다. 더욱이, 이동 내장형 시스템의 서비스가 멀티미디어로 확대됨에 따라, 이를 효과적으로 지원할 수 있는 하드웨어도 재구성(reconfigurable)이 가능하고 재 프로그램 가능한(reprogrammable) 프로세서 아키텍처로 전이되고 있다. 이러한 변화와 아울러 내장형 소프트웨어도 운영체제, 미들웨어, 응용 수준에 이르는 모든 영역에서 재구성, 재사용, 적응성을 효과적으로 지원할 수 있는 컴포넌트기반 소프트웨어 개발 방법론과 함께 이에 기반한 개발 툴셋에 대한 연구[2],[3]가 주목받고 있다.

본 고에서는 전통적인 고정 분산 미들웨어와 이동 미들웨어의 종류를 알아보고, 이동 내장형 미들웨어로서 자바가상머신 JVM(Java Virtual Machine)을 소개한다. 그리고 컴포넌트 기반 실시간 내장형 시스템 개발 툴셋에 대한 연구들을 소개함으로서 향후 내장형 시스템 미들웨어 및 개발 툴셋의 개발 방향과 특징에 대하여 살펴보기로 한다.

본 고의 구성은 다음과 같다. 2장에서는 전통적인 분산시스템의 미들웨어를 소개한 뒤, 3장에서 이동내장형 미들웨어의 연구 동향에 대하여 소개한다. 4장에서는 내장형 미들웨어의 주요한 고려대상인 자바가상머신의 연구를 실시간 지원기능, 경량화, 성능향상 관점에서 연구동향을 살펴보고, 5장에서는 내장형 시스템을 위한 컴포넌트기반 내장형 시스템개발 툴셋에 대하여 알아본 다음, 6장에서 결론을 내린다.

2. 고정 분산 시스템 미들웨어

전통적인 분산 시스템들은 높은 대역폭과 안정된 통신의 연결을 보장받는다. 그리고, 정적인 환경에서 실행되며 네트워크에 영속적(permanently)으로 연결된 고정된 호스트들의 집합이다. 따라서 물리적인 하부구조(infrastructure)위에서 동작하는 분산 어플리케이션 개발자들은 확장성, 개방성, 이기종성, 고장허용, 자원 공유 등과 같은 non-functional 요구사항들을 보장해야 한다. 고정 분산시스템을 위한 미들웨어 기술은 크게 객체 지향(Object-oriented) 미들웨어, 메시지 지향(Message-oriented) 미들웨어, 트랜잭션 지향(Transaction-oriented) 미들웨어로 구분할 수 있다. 각각의 특징을 살펴보면 다음과 같다.

● 객체지향 미들웨어

객체 지향 미들웨어는 분산 오브젝트들간의 통신을 지원한다. 즉, 클라이언트 객체가 다른 호스트에 존재할 수도 있는 서버 객체들에게 오퍼레이션의 실행을 요청한다. 객체지향 미들웨어 시스템은 RPC 기법[4]에서 발전하였으며, 동기적인 상호작용(synchronous interaction)에 바탕을 두고 있어 클라이언트 객체의 요청은 서버 객체가 응답할 때까지 대기하게 된다. 이 범주의 제품에는 OMG CORBA[5], IONA사의 Orbix[6], Borland사의 VisiBroker[7], CORBA 컴포넌트 모델(CCM)[8], Microsoft사의 COM[9], Java/RMI[10] 및 엔터프라이즈 Java Beans[11] 등이 있다. 고정된 분산시스템에서의 괄목할만한 성공에도 불구하고, 객체지향 미들웨어는 많은 계산 부하(heavy computational load), 오브젝트 요청 시 동기 통신(synchronous communication), 인지(awareness)를 가로막는 투명성 원리의 견지(adhering to the principle of transparency) 등은 이동 내장형 시스템으로의 적용을 어렵게 하는 요인이다.

● Message-oriented 미들웨어

메시지 지향 미들웨어는 메시지 패싱을 통해 분산 컴포넌트들 간의 통신을 지원한다. 즉, 클라이언트 컴포넌트들이 네트워크를 통하여 서비스 실행 요청 및 해당 파라미터들이 포함된 메시지를 서버 컴포넌트에 보내면 서버 측에서는 서비스 실행의 결과가 포함된 응답 메시지를 되돌려준다. 메시지 지향 미들웨어는 이동형 시스템에서 요구되는 것처럼 클라이언트와 서버를 분리(de-coupling)하는 방법으로 아주 자연스럽게 비동기 통신을 지원한다. 즉, 미들웨어가 메시지를 받자마자 클라이언트는 처리(processing)를 계속 수행할 수 있기 때문에, 궁극적으로 서버가 응답 메시지를 보내면 클라이언트는 편리한 시간에 그들을 수집할 수 있다. 그러나, 이러한 미들웨어 시스템들은 자원이 풍부한 디바이스들을 요구한다. 특히, 아직 처리되지 않는 수신된 메시지들의 큐를 지속적으로 저장할 수 있을 만큼의 충분한 메모리가 필요하다. 이러한 이유로, Sun사의 Java Message

Queue[12], IBM사의 MQSeries[13] 등은 휴대용 디바이스에 적용되기 어려우며, 더욱이 미들웨어가 어플리케이션으로부터 아무런 정보도 제공받지 않은 채 메시지를 투명하게 관리해야 하기 때문에 어떠한 형태의 인지도 지원할 수 없다.

- Transaction-oriented 미들웨어

트랜잭션 지향 미들웨어 시스템은 컴포넌트들이 데이터베이스 응용인 아키텍처에서 주로 사용된다. 이들은 분산 호스트들에서 작동하는 컴포넌트들의 트랜잭션을 지원한다. 즉, 한 클라이언트 컴포넌트가 단일 트랜잭션 내에 여러 오퍼레이션을 클러스터(cluster)하면, 미들웨어는 네트워크를 통하여 서버 컴포넌트들에게 클라이언트, 서버 모두에게 투명하게 전송한다. 이러한 미들웨어는 이기종 호스트들을 통하여 동기 및 비동기 통신 모두를 지원하고 높은 신뢰성을 지원한다. 즉, 네트워크에 가담한 서버들이 two-phase-commit 프로토콜을 구현한다면 트랜잭션의 원자성(atomicity property)은 보장된다. 그러나 자주 트랙잭션을 사용할 필요가 없는 이동 시스템의 경우에는 이것이 과도한 오버헤드를 발생시키는 요인이 된다. 다시 말하면, 계산 부하와 투명성 지원으로 인하여 트랜잭션 지향미들웨어인 IBM사의 CICS[14]와 BEA사의 Tuxedo[15] 등은 이동 내장형 시스템으로의 적용이 쉽지 않다.

3. 이동 내장형 미들웨어

언제(anytime) 어디서나(anywhere) 개인정보 또는 공공 자원에 손쉽게 접근할 수 있는 이동 내장형 시스템이 급격히 확산 되어가고 있다. 그러나, 전통적인 분산 시스템을 위해 개발된 미들웨어는 앞서 언급한바와 같이 많은 계산 부하, 동기 통신 패러다임, 투명성을 견지하는 설계 등으로 인해 이동 내장형 시스템에 바로 적용하기에는 어려움이 있다. 이동 내장형 시스템을 위한 미들웨어는 적은 계산 부하, 비동기 통신 패러다임, 문맥인지(context awareness)등의 세 가지 요구사항을 만족해야 한다. 이들은 경량이며, 재구성 가능한 reflective 미들웨어, 네트워크 단절 문제에서 기인되는 비동기 통신을 지원하기 위한 tuple-space 미들웨어, 인지가 가능한 문맥인지 미들웨어로 구분할 수 있다(표 1).

- Reflective Object-Oriented 미들웨어

Reflection의 개념은 '프로그램 자신의 해석에 접근, 추론 및 변경이 가능하다는 원리'로 Smith[16]에 의해 소개되었다. Reflection의 개념은 초기에는 프로그램 언어 설계 분야에 그 응용이 국한되었으나[17] 이후에는 운영체제 분야로[18] 또한, 최근에는 분산 시스템까지 영역이 확대되었다[19]. Reflection의 개념을 가진 미들웨어로는 OpenORB [20], Open Corba[21], dynamicTAO[22], Blair et al. work[23], MULTE-ORB[24], Flexinet [25], Globe[26, 27] 등이 있다.

분산 시스템에서 reflection의 역할은 미들웨어 플랫폼에 개방성과 융통성을 더욱 효율적으로 제공하는 것이다. 전통적인 미들웨어는 분산환경에서 기인된 복잡성을 추상화 개념을 통하여 해결한다. 이는 구현과정의 세부사항들은 사용자와 어플리케이션 설계자 모두에게 은폐시키며 미들웨어 자체는 캡슐화 시킴을 말한다. 비록 이러한 접근방법이 전통적인 분산 시스템에서는 성공을 거두었지만 이를 이동 내장형 시스템에 적용시킬 경우에는 여러가지 문제점이 생길 수 있다.

구현의 세부사항을 은폐한다는 것은 모든 복잡성을 미들웨어 계층에서 내부적으로 처리해야 한다는 것을 의미한다. 즉, 미들웨어는 프로그램의 수행에 영향을 끼치지 않으면서 어플리케이션을 대신하여 결정을 내려야할 책임을 갖게된다. 이러한 점이 미들웨어 시스템을 무겁게 만드는 요인이 된다. 다시 말하면, 어떠한 종류의 문제를 투명하게 처리하거나 또

표 1 미들웨어 시스템의 특징

구 분	Fixed distributed system	Ad-hoc distributed system and Nomadic distributed system
Computational Load	Heavy-weight (Resource-rich Fixed device)	Light-weight (Resource-scare Device)
Communication Paradigm	Synchronous (Permanent Connection)	Asynchronous (Intermittent Connection)
Context Representation	Transparency (Static Context)	Awareness (Dynamic Context)

는 어플리케이션에 최상의 서비스 품질을 보장하기 위해서는 많은 양의 코드와 데이터가 필요하다. 그러나, 이와같이 무거운 시스템은 이동 디바이스에서는 효과적으로 작동될 수 없다. 이동 시스템에서 모든 상세한 구현 사항을 사용자로부터 은익 한다는 것은 항상 가능하지도, 또한 바람직하지도 않다. 이처럼 근본적인 문제는 구현 세부사항을 은폐함으로서 어플리케이션이 보다 효율적이고 적법한 결정을 내릴 수 있는 정보가 있음에도 불구하고 미들웨어가 이를 대신하여 결정을 내려야 한다는 것이다.

Reflective 시스템은 검사(inspection)와 적응(adaptation)의 수단을 가지고 자신을 스스로 변경할 수 있다. 검사 수단으로 시스템 내부 행위(behavior)가 노출됨에 따라 미들웨어 구현 시 부가적인 행위를 모니터에게 곧바로 추가할 수 있다. 그리고, 적응 수단으로 새로운 것들을 추가하고 기존의 특징들을 수정함으로써 시스템 내부 행위는 동적으로 변경될 수 있다. 즉, 최소한의 기능을 제공하는 미들웨어 핵심(core)을 이동 내장형 디바이스에 설치한 후에, 어플리케이션이 자신의 요구에 따라 미들웨어의 동작을 모니터링하고 시스템을 새롭게 구성할 수 있다. 이러한 방법을 통해 문맥인지를 지원하는 경량 미들웨어를 구현할 수 있다.

Reflection이 가지고 있는 많은 가능성에도 불구하고 이는 대부분 CORBA ORB의 reflective extensions을 제공하는데 주로 응용되기 때문에, 컴포넌트들간의 동기 통신만을 지원한다. 더욱이 이러한 시스템들은 여전히 CORBA의 기본적인 오버헤드 때문에 작은 디바이스 상에서 작동되기에는 너무 무겁다.

- TupleSpace-based 미들웨어

협소하고 변동이 심한 대역폭, 빈번한 단절 현상으로 대변되는 무선 통신 미디어 환경을 지원하기 위해서는 네트워크 단절이 발생하더라도 작업을 지속적으로 수행할 수 있어야 하며, 네트워크에 연결되었을 때 이를 시의 적절하게(opportunistic) 이용해야되므로 전통적인 분산 시스템들이 지원하는 동기 통신 패러다임은 새로운 개념의 비동기 통신 형태로 바뀌어야 한다.

이러한 통신의 문제점은 tuple space 기반 시스템들에 의해 이슈화되었다. 원래 이 개념은 병행 프로그램을 위한 협동 언어(coordination language)인 Linda[28]로부터 유래되었으며, 무선 통신에 대한 많은 유용한 편의성을 제공한다. Linda의 tuple space는 전역으로 공유(globally shared)되며, tuple이라고 하는 자료구조의 저장소로 동작한다. 이러한 tuple들이 tuple space 시스템의 기본 구성요소로 프로세스에 의해 생성되고, write 프리미티브를 이용하여 tuple space에 위치하며, read와 take 프리미티브를 사용하는 여러 프로세스들에 의해 동시에 액세스될 수 있다. Tuples들은 익명성을 갖기 때문에 tuple 내용에서 패턴 매칭을 통하여 그들의 선택이 이루어진다. 통신은 시간과 공간상에서 분리되기 때문에 송신자나 수신자는 같은 시간에 동작해야 할 필요는 없다. 왜냐하면 tuples는 자신의 생명 기간 (life span) 가지고 있고, 그들을 생성하는 프로세스와는 독립적이며, 시스템이나 플랫폼 범위에 상관없이 tuple space는 전역 공유데이터 영역과 같이 간주되어 데이터 교환을 위해 자신들의 위치에 대한 상호 지식이 필요 없기 때문이다.

이런 분리 형식은 이동이나 연결 패턴에 의해 통신환경이 동적으로 변화되는 이동형 시스템에서는 매우 중요하다. 그러나 전통적인 tuple space의 구현만으로는 이러한 통신 문제를 충분히 해결할 수는 없다. 예를 들면, 전역으로 공유되는 데이터 공간을 이동 내장형 호스트들에서 어떻게 표현할 것이며, 영속성은 어떻게 유지할 것인가? 와 같은 문제는 여전히 남아있기 때문이다. 이 범주에 속하는 미들웨어들은 TSpaces[29], JavaSpace[30], Lime[31] 등이 있다.

- 문맥인지 미들웨어

사용자의 환경 변화, 이기종 호스트와 네트워크에 적용될 수 있는 어플리케이션을 위해서 시스템은 자신이 사용되는 문맥을 인지할 수 있는 이동 어플리케이션을 제공해야 한다. 문맥은 개체와 개체를 둘러싸고 있는 환경사이에서 영향을 주고 변화해가는 정보들을 말하며 엔티티(entity)에 영향을 주는 속성(attribute)이라 할 수 있다. 사람과 컴퓨터간의 교류를 위하여 관련된 많은 엔터티 정보들은 모두 문맥요소들이 될 수 있다. 사용자 문맥에는 시스템의 위치 정보, 근접한 프린터나 데이터베이스 등과 같은 연관된 위치 정보, 프로세싱 파워나 입력 장치 등과 같은 디바이스의 특성 정보, 잡음수준과 대역폭과 같은 물리적 환경 정보, 사용자의 움직임 정보 등이 있다. 이러한 데이터들은 매우 유동적이므로 컴퓨팅 환경에서 이같은 정보들을 활용하기 위해서는 인지 메카니즘이 필요하다. 문맥인지는 '사람, 장소, 혹은 물리적 계산적 객체들이 있는 엔티티의 상황을 특징화하여

사용되어질 수 있는 정보'라고 할 수 있다. 표 2는 문맥의 모델과 타입을 보여주고 있다.

문맥인지 컴퓨팅은 이미 10여년 전에 제안되었으며[32], 많은 연구자들이 문맥 정보를 수집하여 변경에 적응하는 시스템을 연구 개발하여 왔다. 실제로 위치 정보를 이용한 것에는 Shopping Assistant[33], CyberGuide[34], Teleporting[35], People and Object Pager[36], Conference Assistant[37] 등이 있다. 대부분의 이러한 시스템들은 위치정보를 끌어내기 위하여 기반 네트워크 운영체제와 상호 작용하며, 그들을 처리하고, 사용자에게 편리한 형태의 포맷으로 보여준다.

표 2 문맥의 모델 및 타입

실제 모델	문맥 타입	장점
Auto Lights On/Off	Room	편리성
File System	Personal Identity	정보 검색
Calendar Reminders	Time	메모리
Smoke Alarm	Room	안전성
Barcode Scanners	Object Identity	Efficiency

위치 기반 서비스(location-based services)와 어플리케이션 개발을 용이하게 하고 개발 주기를 단축시키기 위해, 공통적인 인터페이스를 사용하여 상이한 위치 기술들을 통합한 미들웨어 시스템들이 개발되어왔다. 이런 유형의 시스템들로는 Oracle iASWE[38], Nexus[39], Alternis[40], SignalSoft[41], CellPoint[42] 등이 있다.

● 다른 종류의 미들웨어 솔루션

지금까지 언급한 세 가지 유형의 미들웨어 시스템 이외에도 다른 많은 시스템들이 이동성과 관련하여 개발되었다. 중요한 이슈중의 하나로는 단절된 오퍼레이션을 위한 지원(support for disconnected operations)이며, 이를 위해서는 사용자가 복제본에 접근할 수 있도록 하여 데이터의 이용을 최대화 해야하며, 이런 유형의 미들웨어들로는 Coda[43], Odyssey[44], Bayou[45], XMIDDLE[46] 등이 있다.

또 다른 중요한 문제는 서비스의 발견(discovery of services)이다. 동적 이동 환경에서는 호스트들의 연결 상태가 급속하게 변화한다. 처음 네트워크에 연결 당시에는 해당 서비스의 사용이 가능했다 하더라도 재연결 시에 그것이 불가능할 수도 있다. Jini[47]는 디바이스 그룹과 소프트웨어 컴포넌트들이 단일의 동적 분산 시스템으로 연합(federation)할 수 있도록 하는 아키텍처 이다. 표 3은 지금까지 언급한 내장형 미들웨어에 대한 유형, 특징과 예를 보여준다.

4. 자바 가상머신

동적인 어플리케이션 다운로드, 크로스 플랫폼 호환성, 빠른 응답성, 비연결성, 보안 기능 제공 등의 이유로 내장형 시스템 개발자들은 자바를 사용하고 있다. 그러나 실시간 또는 자원의 제약이 심한 내장형 시스템에 응용하기 위해서 자바를 수정하는 연구가 활발히 진행중이며, 또한 자바가 가지는 장점을 유지하면서 어떻게 성능을 향상시킬 것인가에 대한 연구도 진행 중에 있다.

표 3 개발되고 있는 이동 내장형 미들웨어 종류

유 형	특 징	예
Reflective Object-oriented Middleware	· 더욱 큰 개방성과 유연성 제공함 · 시스템 내부 동작이 동적으로 변경 가능함 · 미들웨어의 핵심 기능이 최대한 경량화됨	OpenCorba, Next Generational Middleware, Globe
TupleSpace-based Middleware	· 협소하고 일정하지 않은 네트워크 대역폭, 빈번한 네트워크 단절 등을 해결하기 위함 (Asynchronous communication)	Lime, TSpaces, JavaSpace
Context-Aware Middleware	· 호스트와 네트워크 이기종성, 사용자의 환경변화를 해결 · 위치, 상대위치, 디바이스 특성, 물리적 환경, 사용자의 활동성 등과 같은 정보를 응용에 포함	Oracle iASWE, Nexus, Alternis, SignalSoft, CellPoint
Other Solutions	· Disconnected operation 지원 · 서비스 발견(discovery of service) 문제 해결	Coda, Odyssey, Baou, XMIDDLE, Jini

● 자바 내장형 실시간 시스템

실시간 내장형 시스템 구축에 자바가 사용되는 이유는 동적 클래스 로딩, 쓰레기 수집기, 멀티쓰레딩 등의 기능을 제공하기 때문이다. 반면에, 느린 JIT (Just-In-Time) 컴파일러의 런타임 번역이나 바이트 코드 해석, 쓰레드의 행위가 JVM이나 운영체제에 의존해 있기 때문에 실시간 태스크를 다루는데 적절하지 못하고, 현재 자바가상머신 쓰레기 수집기가 실시간 시스템을 지원하기가 어렵다는 이유로 자바가 실시간 내장형 시스템에 바로 적용하기가 적합하지 못하다. 이러한 문제를 해결하기 위하여 미국 National Institute of Standards and Technology (NIST)의 관리를 받고 있는 자바 플랫폼의 실시간 확장에 관한 요구사항(Requirements for Real-time Extensions for the Java Platform) 구룹(http://www.nist.gov/rt-java)은 표준 실시간 자바 확장에 대한 요구사항(Real-Time Java : RTJ)을 정의하고 있으며, 다음과 같은 기초 요구사항 문서를 출간하였다.

첫째, 자바프로그래밍 환경에서 디바이스 드라이브를 구현하거나, 물리적 메모리에 직접 접근할 필요가 있거나 인터럽트를 처리하기를 원할 때 사용자가 하드웨어에 직접 접근할 수 있는 기능을 추가한다. 둘째, Rate Monotonic(RM) 스케줄링 또는 Earliest Deadline First (EDF) 알고리즘과 같은 선점형 우선순위 기반 스케줄링 정책을 지원해야 한다. 셋째, 우선순위 역전현상과 같은 동기화 문제를 해결하기 위하여 우선순위계승(priority inheritance) 또는 우선순위 상한(priority ceiling) 프로토콜을 지원해야하며, 큐잉 정책을 보장해줄 수 없는 자바의 모니터 기능을 개선해야 한다. 넷째, 비동기 이벤트 처리 기능을 지원해야하며 다섯째, 자원 어카운팅(accounting), 자원 회수(reclamation), 자원 할당, 자원 협상과 같은 자원관리 기능을 제공해야하고, 마지막으로, 실시간 시스템에 적합한 예를 들면 점진적(incremental) 쓰레기 수집기를 지원해야 한다.

이는 API 기반의 확장을 지원하고 있으며, NIST 문서를 기반으로 하고 있는 자바 내장형 실시간 시스템에 대한 접근방법은 크게 실시간 API v0.2(RTJ-02)[48], Real-Time Specification for Java, v0.8.1 (RTJ-08)[49], Real-Time Core Extension for the Java Platform(RT Core)[50], 실시간 JavaO 사양 (RT JavaO)[51]으로 구성된다. NIST에서 문서를 만들기 전에 소개되었던 해결책 중에 하나는 Real-Time Java Threads(RTJT)[52]이며, 다른 하나는 Portable Executive for Reliable Control(PERC) [53]으로 이것은 실시간 시스템에 대한 추상을 제공하는 실시간 패키지와 하드웨어를 접근하기 위해 하위 레벨 추상을 제공하는 내장 패키지로 구성되어 있다. 그리고 CSP algebra, Occam2 언어, Transputer 마이크로프로세서를 기반으로 하는 Communication Threads for Java(CTJ)[54]가 있다. 다른 유형의 접근방법은 자바가상머신을 운영체제에 통합하는 것으로 General Virtual Machine(GVM)[55]에 의해서 이루어졌다. 자바에 성능을 향상시키는 또 다른 방법으로 JVM을 마이크로프로세서에 통합하는 것으로 Rockwell Collins의 JEM-1 마이크로프로세서(pico Java-I [56])가 있다. 표 4는 각 접근방법의 특징들을 비교한 것이다. 여기에서 "A"는 해당 이슈를 상세하게 설명하였음을 의미하며,"M"은 부분적으로, "-"는 전혀 설명하지 않았음을 의미한다.

표 4 내장형 실시간 자바 시스템에 대한 접근방법 비교

Feature	HW Access	Task Scheduling		Synchronization Communication		Event Handling		Resource Management		Garbage Collection	
Solution		Priorities	Deadlines	Java Threads	Task Thread	Internal	External	Budgets	Negotiation	Preemptable	Bounded
RTJ-02	-	A	-	-	-	A	A	A	-	A	-
RTJ-08	A	A	A	-	A	A	A	A	-	A	A
RT Core	A	A	A	-	A	A	A	A	-	A	-
RT JavaO	M	A	-	M	M	M	M	-	-	A	-
PERC	A	A	M	-	-	A	M	A	A	A	M
CTJ	A	A	-	-	-	-	-	-	-	-	-
RTJT	-	A	A	-	-	-	-	M	-	M	-
GVM	M	A	-	-	-	-	-	-	-	-	-
pico-Java-I	A	-	-	-	-	-	-	-	-	A	-

● 실시간 자바 전문가 그룹(Real-Time Java Expert Group: RTJEG)

1998년 실시간 자바 플랫폼을 개발하고 있던 NewMonics 사와 NIST, IBM, Sun이 하나로 합쳐져 실시간 요구사항 워킹 그룹이 만들어졌다. 이 그룹이 "자바 플랫폼의 실시간 확장에 관한 요구사항"라는 문서를 발간하는 과정에서 Sun은 1998년 말에 자바의 새로운 요구사항들을 검토하여 스펙을 만드는 기구인 Java Community Process(JCP)를 만들었으며, 워킹 그룹의 연구를 토대로 IBM이 작성하여 제출한 Java Specification Request(JSR)를 채택하였다 (JSR-000001). 워킹 그룹의 리더인 Greg Bollella는 실시간 자바 스펙(Real-Time Specification for Java: RTSJ)을 정의하는 그룹과 컨설턴트 그룹으로 구성

된 RTJEG[57]을 결성하였으며, 이 그룹의 주요 구성원은 IBM, Cyberonics, Aonix/Ada Core Technologies, Microware Systems Corporation, QNX System Software Lab, Sun Microsystems, Rockwell-Collins/aJile, Nortel Networks 등이다. 이 구룹에서는 실시간성을 지원하기 위해 고려되어야 할 것들을 다음과 같이 정의하였다.

첫째 쓰레드 스케줄링으로, 실시간 스케줄링을 위하여 스케줄링 객체는 힙 영역에 생성되어 쓰레기 수집기의 영향을 받는 RealtimeThread와 힙 영역의 데이터를 접근할 수는 없지만 쓰레기 수집기의 영향을 받지 않는 NoHeapRealtimeThread를 사용한다. 스케줄링을 하기 위한 방법은 우선순위에 기반한 스케줄링 기법을 제공해야 하며, 이를 위해서는 기본적으로 최소한 28개의 유일한 우선순위를 갖는 고정된 우선순위 선점형 디스패쳐를 제공해야 한다.

둘째, 메모리 관리로써 실시간성을 지원하기 위하여 메모리는 쓰레기 수집기의 예측할 수 없는 지연 문제를 해결할 수 있는 메카니즘을 가져야 하며, 쓰레기 수집기의 영향을 받지 않기 위해 메모리 영역을 분리한다.

셋째, 동기화로써, 동기화를 위해서는 우선순위를 사용한다. 높은 우선순위를 갖는 쓰레드는 실행하기 위해 대기하고 있는 쓰레드 중에서 스케줄러가 선택한 쓰레드가 된다. 우선순위를 사용하기 위해서 대기 큐가 필요하며 이 큐는 FIFO방식으로 하여, 쓰레드의 우선순위가 같은 경우가 발생했을 경우 처리가 용이하도록 한다. 또한, 우선순위 계층 모니터 제어 정책을 구현하여야 한다.

넷째, 비동기 이벤트 핸들링으로, 비동기 이벤트 핸들링을 위해서는 AsyncEvent와 AsyncEvent Handler가 필요하다. AsyncEvent는 어떤 이벤트가 발생할 경우 핸들러들을 unblocking하고, 핸들러들의 셋을 이벤트와 결합시킨다. AsyncEventHandler는 쓰레드와 비슷하게 작동을 하며, 이벤트 발생 시 결합된 핸들러들을 스케줄 한다.

마지막으로, 물리적 메모리 접근으로 RTSJ는 프로그래머들이 직접 물리적 메모리에 접근할 수 있는 클래스를 정의하였다. RawMemoryAccess는 프로그래머가 물리적 주소의 범위와 byte, short, int, long, float, double로 물리적 메모리에 접근할 수 있는 객체를 생성할 수 있도록 정의하였다.

- Sun 사의 J2ME 플랫폼 및 KVM

현재 자원이 제약된 소형의 이동단말기 시스템을 위한 자바플랫폼은 Sun 사에서 Java 2 Micro Edition(J2ME) Connected, Limited Device Configurtion(CLDC)[58], [59]/MIDP와 CLDC/ PDAP가 있다. CLDC/MIDP는 휴대전화기를 위한 것이고, CLDC/PDAP는 PDA를 위한 자바 플랫폼이다. 퍼스널 자바와 내장형 자바는 J2ME 플랫폼이 등장하기 이전부터 내장형 시장을 목표로 한 자바 플랫폼이었다. 인터넷 셋탑박스와 같은 나름대로의 성과도 있었지만, pJava와 eJava는 시장에서 주목받을 만한 결과를 출시하지 못했다. 그것은 기본적으로 Sun 사가 하드웨어 설계 및 제조와 관련해서는 내세울 만한 강점이 없었기 때문이었다. 즉, 하드웨어 업체들의 적극적인 지지 없이는 자바의 내장형 시장 진출은 어려운 과제였다. 그리고, 수많은 소비자/내장형 디바이스들에 공통적으로 적용할 수 있는 플랫폼을 개발한다는 것도 내부적인 한계가 있었다. 다양한 하드웨어와 운영체제, 전혀 호환성 없는 유저 인터페이스까지 이 모든 차이점을 극복하면서 기존의 J2SE 기반의 자바와의 호환성까지 유지할 수 있는 솔루션은 불가능하다는 현상을 인식하기 시작한 것이다. 그래서 등장한 개념이 컨피규레이션과 프로파일로의 플랫폼 분할이다. 컨피규레이션이란 자바가상머신과 코어 API들에 대한 명세를 의미하고, 프로파일은 그 상위의 클래스 라이브러리, 즉 표준 API 집합에 대한 명세를 의미한다. 이러한 개념적인 분할이 필요한 이유는 메모리와 CPU 등의 크기와 성능이라는 측면에서의 요구사항이 동일한 디바이스들의 집합을 하나로 묶어 컨피규레이션을 정의하고, 이러한 컨피규레이션을 바탕으로 각 디바이스들의 기능 혹은 버티컬 시장의 요구사항에 맞춰 프로파일을 정의함으로써 플랫폼의 통일성과 다양성을 동시에 만족시킬 수 있기 때문이다.

경량 내장형 자바가상머신 분야에서는 Sun 사가 2000년 J2ME을 개발하면서 이의 핵심 컴포넌트인 KVM을 선보였다(http://java.Sun.com/ products/cldc)[60]. KVM은 40에서 80 킬로바이트의 메모리 영역에서 구현되며 이식성, 모듈성을 지원한다. KVM은 128 킬로바이트 미만의 메모리를 갖는 16/32 bit RISC/CISC 마이크로프로세서에 적합하며, 셀룰러 폰, 호출기, PDA, POS 단말기 등에 적용된다. KVM은 가상머신, 최소한의 자바 클래스 라이브러리, 자바 어플리케이션이 동작할 수 있는 여분의 힙(heap) 영역을 갖는다. 또한 경량화를 위하여, 플로팅

포인트를 지원하지 않고, 클래스 인스턴스의 finalization을 지원하지 않으며, JNI를 지원하지 않는다. 그리고, 사용자 정의 클래스 로더가 없으며, 리플랙션(reflection) 기능이 없고, 쓰레드 그룹이나 데몬 쓰레드를 위한 지원 기능이 없고, 약 참조(weak references)가 없으며, 에러 처리기능을 제한함으로써 경량화를 지원한다. 그림 1은 자바 2 플랫폼을 보여주며, 여기서 CDC는 Connected Device Configuration, J2EE는 Java 2 Enterprise Edition, J2SE는 Java 2 Standard Edition을 일컫는다.

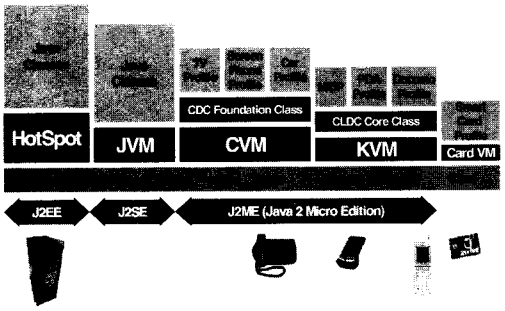


그림 1 내장형 시스템을 위한 자바 2 플랫폼

• Java 가상머신 성능

자바가상머신의 성능 향상을 위해서는 잘 설계된 소프트웨어, 최적화 컴파일러, 아키텍처의 지원, 그리고 효율적인 런타임 라이브러리 등이 잘 조화를 이루어야 한다. 자바가상머신의 성능 향상 연구에 대한 소개는[61],[62]에서 잘 보여주는데 이는 크게 컴파일 개선(compilation improvement) 런 타임(runtime improvement) 개선, 하드웨어적인 접근 방법으로 자바프로세서로 구분된다. 런 타임 개선으로서는 컴파일 개선에는 JIT 컴파일러, 직접 컴파일러(direct compiler), 바이트코드 대 소스 변환기(bytecode-to-source translator)의 성능을 개선해야 된다. 런 타임 개선으로는 바이트코드 최적화, 병행 및 분산 수행, HotSpot와 같은 동적 컴파일(dynamic compilation)이 있다. 그리고 자바가상머신의 성능을 개선시키는 기법으로는 쓰레드 동기화, RMI (Remote Method Invocation), 고성능 수치계산, 클래스로더, 객체 할당기, 그리고 쓰레기 처리기의 성능을 개선시켜야 한다. 표 5는 위에서 언급한 자바 가상머신의 연구를 실시간, 경량화, 성능개선 관점에서 본 기술의 특성을 보여준다.

5. 내장형 시스템 개발 툴셋

VDC[63]에 따르면 1998년 동안 내장형 소프트웨어 개발 툴과 관련된 서비스의 매출이 8억 1470만 달러에 이르러 1997년 매출의 19.4%가 증가하였다. 그 중 설계 자동화 툴(Design Automation Tool)은 1998년에도 여전히 전체 내장형 개발 솔루션 시장 매출액의 가장 작은 19.6%를 차지하였으나, 1997년과 비교해 본다면 23.8% 성장하여 가장 빠르게 점유율을 높이고 있다. Telelogic, ObjectTime 벤더등의 소프트웨어 모델링 툴의 매출은 7,090만 달러를 차지하며, 전자통신 장비 제조의 변화와 다른 영역(군사, 항공,

표 5 자바 기술의 특성

분야	기술 특성
자바 내장형 실시간시스템	Accessing Hardware, Scheduling (RMA, EDF), Synchronization, Asynchronous Event Handling, Resource Management, Dynamic Error Handling
실시간 자바 전문가그룹	No Garbage Collection, No Dynamic Initialization and Resolution, Stack Allocation, Partitioning Memory and CPU Time
경량자바 가상머신	-No Floating Point Support, No Support for Finalization of Class Instances, No Support for the JNI, No User-Defined Class Loader, No Reflection Features, No Support for the Thread Groups or Daemon Threads, No Weak References, Limitation on Error Handling
자바 가상머신 성능개선	-Compilation Improvement JIT(Just-In-Time), Direct Compiler, Bytecode-to-Source Translator, -Runtime Improvement Bytecode Optimization, Dynamic Compilation, Executing Java in Parallel Improve JVM Thread Synchronization, Remote Method Invocation, High Performance, Numeric Computation, Garbage Collection -Hardware Processor (Java Processor)

항공전자공학, 산업자동화)으로의 소프트웨어 모델링 도구의 적용과 인식 증가, 그리고 선호하는 비주얼 모델링 언어로서 UML의 절대적인 우세에 힘입어 성장하였다(VDC 1999.6). Gartner Dataquest[64]에 따르면 2000년 전세계 내장형 소프트웨어 시장규모가 8억 6,600만 달러에 이르러 1999년의 6억 2,300만 달러 대비 37%의 성장률을 기록한 것으로 나타났다. 회사별 동향을 살펴보면 Wind River Systems가 동 시장의 거의 1/3을 점유하고 있으며, 그 뒤를 Rational Software와 Palm Computing이 따르고 있는 것으로 보인다. 이들 3개 벤더는 세계 시장의 절반 이상을 점유, 동 시장을 선도하고 있는 것으로 나타났다(Dataquest 2001.11). 이처럼, 최근 빠르게 증가하고 있는 내장형 시스템 소프트웨어를 저렴한 가격으로, 신속하고, 신뢰성 있게 생산하기 위하여 재사용, 적응성을 효율적으로 지원할 수 있는 개발 방법론이 주목을 받고 있다. 컴포넌트기반 실시간 내장형 시스템 개발 툴셋으로는 VEST, MetaH, CIP, 그리고 Rational Rose RT(Real-Time)등이 있으며, 각각의 특징은 다음과 같다.

5.1 VEST(Virginia Embedded Systems Toolset)

VEST[65, 66, 67, 68]는 미국 버지니아 대학의 Stankovic 교수팀이 컴포넌트기반 내장형 실시간 시스템의 개발, 구현, 평가를 개선할 목적으로 개발하고 있으며 조립과 분석 아키텍쳐를 포함한다. VEST는 기본적인 라이브러리와 컴포넌트를 제공하며 다양한 예제의 의존성 검사를 수행하는 버전 1이 현재 구현 중이다.

VEST의 특징은 컴포넌트 기반, 실시간, 내장형 시스템을 개발, 분석하기 위한 통합 환경이며 수동형 컴포넌트를 생성 또는 선택한다. 그리고, 컴포넌트를 런타임 구조나 하드웨어에 매핑하며 의존성 검사(dependency check), 비기능적 분석(non-functional analysis)을 수행한다.

VEST는 소프트웨어, 하드웨어 컴포넌트를 포함하는 다양한 라이브러리와 구성 도구(configuration tool)와 분석 도구를 포함하는 대화식 개발 도구로 이루워져 있다. 그림 2는 VEST의 시스템 구조를 보여주고 있는데, 여기서 라이브러리는 특정 도메인 컴포넌트를 저장하는 기반 라이브러리(base library)와 사용자 결과물을 저장하는 제품 라이브러리(product library)로 구성되며 저장되는 소프트웨어, 하드웨어 컴포넌트는 계층적으로 생성 가능하다. VEST의 컴포넌트는 표 6과 같다. 구성 도구는 컴포넌트 조립이나 의존성 검사를 수행하거나 컴포넌트를 프로세서나 하드웨어에 매핑 한다. 분석 도구는 실시간, 신뢰성, 스케쥴링 분석을 위해 분석 도구를 호출한다.

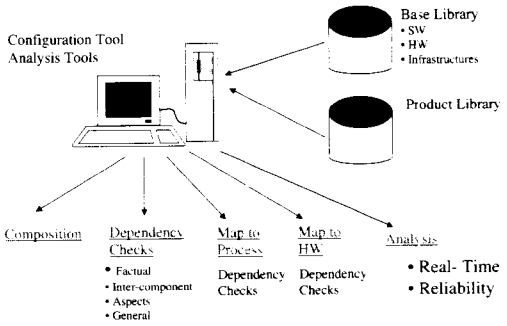


그림 2 VEST의 시스템 구조

그리고, VEST의 의존성 검사는 조립된 시스템의 신뢰도를 증가시키며 특히 다른 툴셋과의 큰 차이점이 된다. 의존성의 복잡성 때문에 의존성 검사는 표 7과 같이 4가지 형태로 나뉘며 각 형태마다 세부적인 검사 리스트를 포함한다. 첫째, 사실 검사(factual checking)는 가장 간단하며 리스트는 확장 가능하다. 검사되는 의존성 요구사항 수가 많을수록 간단한 실수를 피할 수 있다. 예를 들면, 메모리 크기 검사는 각 컴포넌트의 메모리 요구 사항과 하드웨어 컴포넌트에 의해 제공되는 메모리를 비교한다. 둘째, 컴포넌트간 검사(inter-component checking)는 컴포넌트 사이의 의존성을 검사한다. 예를 들어, 인터페이스 호환성(interface compatibility)은 반환형이 호출 컴

표 6 VEST의 컴포넌트

Software Source Objects	Hardware Objects	Software Higher-Level Objects	Hardware Higher-Level Objects
Fragment/Function Component	Device	Thread Process	Composed Device System

포넌트의 반환형과 호환성이 있는 지, 또는 입력 매개변수 리스트가 호출 컴포넌트의 입력 매개변수 리스트에 호환성이 있는지 여부를 검사한다. 셋째, Aspect는 일반적인 절차로는 명확히 캡슐화할 수 없거나 컴포넌트의 성능에 영향을 끼치는 문제를 다룬다. 예를 들면, 시스템에서 메시지 손실이 발생했는지 여부를 찾기 위해 버퍼크기 문제를 검사한다. VEST는 각 컴포넌트의 메시지 손실 문제에 관련된 필요한 상호연관 정보(reflective information: 메시지 소비, 생산률, 입출력 메시지 크기 등)를 제공한다.

표 7 의존성 검사 리스트

Type	Check List
Factual	worst case execution time, memory needs(memory size), data requirements, timing constraints(deadline, period, jitter), power requirements, initialization requirements
Inter-component	call graphs, interface compatibility, exclusion requirements
Aspect	end-to-end deadline/real-time, concurrency, persistence requirements, fault tolerance, preemption vs. non-preemption, optimizations
General	No deadlock, No livelock

5.2 MetaH

MetaH[69, 70, 71, 72]는 미국의 Honeywell사에서 개발한 신뢰성 있는 실시간 다중 프로세서 항공공학 시스템을 개발하기 위한 언어이자 툴셋 이다. MetaH의 특징은 전통적인 언어로 쓰여진 코드 모듈을 어플리케이션을 형성하기 위해 조립하는 방법을 규정한다. 그리고, 특정 하드웨어 시스템의 구조를 규정하며 소프트웨어가 하드웨어에 할당되는 방법도 규정한다. 또한, 특정한 종류의 객체를 제공하며 다양한 분석을 수행한다.

MetaH 툴셋은 그래픽, 텍스트 포맷으로 MetaH 명세를 변환하며 자동으로 실행 이미지를 생성한다. 또한, 실시간 스케쥴 가능성, 신뢰성, 안전, 보안 모델링 분석 도구 등을 사용하여 다양한 분석을 수행한다. MetaH 툴셋에서 사용하는 MetaH 언어는 코드 모듈의 인터페이스와 속성을 기술하며 코드 모듈들이 상위 레벨의 객체로 조립되는 방법도 기술한다. 객체는 두 개의 그룹(소프트웨어, 하드웨어 객체)으로 분류된다. 소프트웨어 객체는 두 개의 서브그룹으로 나뉘어 지는데 소스객체와 소스객체의 그룹화를 기술하는 상위 레벨 객체이다. 유사하게 하드웨어 객체는 하드웨어의 특정 부분을 기술한 하드웨어 컴포넌트 객체와 그룹화를 기술한 상위 레벨 객체로 나뉘어 진다. 표 8은 MetaH에서 사용하는 객체의 리스트이다[70].

그림 3은 MetaH 툴셋의 시스템 구조이다. 각 부분의 기능으로 첫째, 그래픽과 텍스츄얼 엔트리(graphical and textual entry) 기능으로 MetaH 언어는 그래픽, 텍스트 구문법(syntax) 둘 다를 가지며 그래픽, 텍스트 버전 각각은 그래픽, 텍스트 편집기를 사용해 편집 가능하고 두 표현사이에는 자동 변환을 지원한다. 둘째, 소프트웨어, 하드웨어 바인더 기능으로서 다중 프로세서 시스템의 각 프로세스를 특정 프로세서에 바인딩 시키며 프로세스 사이에 공유되는 패키지, 모니터는 특정 메모리에 바인딩 시킨다. 또한 객체를 특정한 하드웨어 객체에 바인딩 하는 역할을 수행한다. 셋째, 스케쥴 가능성 모델링과 분석으로서 MetaH 툴셋은 자동적으로 스케쥴 가능성 모델을 생성하고 분석을 수행한다. 이는 어플리케이션이 스케쥴링 되는지 여부와 모든 프로세스가 규정된 빈도와 마감시간안에 실행되는지 여부를 분석한다. 넷째, 신뢰성 모델링과 분석 기능으로 결함 이

표 8 MetaH 객체 리스트

Software Source Objects	Hardware Component Objects	Software Higher-Level Objects	Hardware Higher-Level Objects
event, port, type, type package, monitor, subprogram, package.	event, port, type, type package monitor, memory processor, device channel	application, error model, connection, process macro, mode, path	application, error model, connection, system

벤트와 에러 상태의 집합을 규정하기 위해 사용하며 자동으로 Markov 신뢰성 모델을 생성한다. 마지막으로 안전, 보안 모델링과 분석 기능으로서 각 프로세스를 위한 시 공간 보안을 제공한다. 예를 들어, 불안전한 객체의 부적절한 동작이 안전한 객체의 동작에 영향을 끼칠 수 없게 확인하며 객체가 권한을 갖고 있지 않은 데이터를 포함하거나 접근할 수 없도록 확인한다.

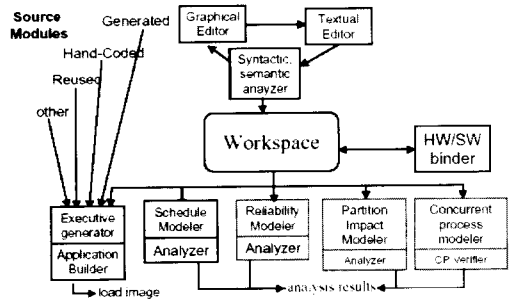


그림 3 MetaH의 시스템 구조

그림 4는 mode M 인터페이스(왼쪽 그림)와 M.EXAMPLE 구현(오른쪽 그림)을 위한 그래픽 표현을 보여준다. 인터페이스 명세는 공유 모니터와 패키지, 그리고 포트와 이벤트라 불리는 소스 엔티티들의 선언들을 포함할 수 있으며(예를 들면, mode M은 인터페이스로 하나의 포트를 갖는다) 구현은 컴포넌트 사이의 컴포넌트 엔터티와 연결을 위한 선언들을 포함할 수 있다(예를 들면, P1과 P2는 M. EXAMPLE내의 mode 구현내의 컴포넌트들이다).

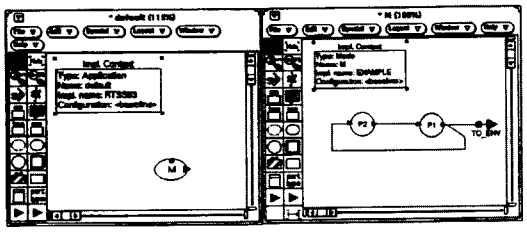


그림 4 그래픽 MetaH 툴셋 예제

5.3 CIP(Communicating Interacting Processes)

CIP[73, 74]는 스위스의 CIP System AG사에서 개발한 방법론(method)이자 툴셋 이다. CIP 방법론은 내장형 시스템을 위한 모델 기반 소프트웨어 개발 방법이며 CIP 툴셋에 의해 개발된다. CIP 툴셋은 자동 코드 생성기와 검증 함수를 갖는 그래픽 모델링 프레임워크이며 CIP 모델을 실행할 수 있는 컴포넌트로 변환한다. CIP 모델의 완전성은 자동화된 검증 함수에 의해 체크된다. CIP의 특징은 첫째, 기능과 연결 문제를 분리하였다. 외부 프로세스의 이벤트에 대한 기능적인 올바른 반응을 나타내는 기능 문제와 실세계와 컴포넌트 사이의 정보 전송을 나타내는 연결 문제의 분리는 개발 프로세스에 이득이 된다. 둘째, 화이트 박스 조립에 의한 컴포넌트기반 조립 방법으로 CIP 모델은 모델 컴포넌트를 생성, 조립, 연결함으로서 구성된다. 셋째, 블랙 박스 조립에 의한 컴포넌트 기반 구현으로 CIP 모델을 실행할 수 있는 소프트웨어 컴포넌트로 자동 변환한다. 내장형 컴포넌트의 재사용은 컴포넌트가 특정 실세계의 특정 행위를 규정하므로 불가능한 경우도 있으나 컴포넌트 조립은 여전히 중요하다.

CIP 방법론을 가지고 내장형 시스템은 확장된 유한 상태 머신으로서 설계된 프로세스들의 구조와 행위 모델들에 의하여 규정된다[74]. CIP 모델은 채널, 프로세스, 메시지 또는 오퍼레이션과 같은 모델링 요소들로 구성되며 이러한 모델링 요소들은 CIP 객체라 불리운다. CIP 툴셋은 유연한 방법으로 이들 요소들을 생성, 조립, 연결하도록 허용한다. 툴의 편집기 프레임워크는 CIP 모델의 조립적인 구조를 반영하며 시스템, 클러스터, 프로세스, 모드는 CIP 브라우저에서 생성된다. 한 편집기에서 이루어진 변화는 즉시 모든 의존적인 모델의 부분과 뷰에 갱신되므로 일관성 있는 CIP 모델을 구축할 수 있다.

CIP 모델은 각각 동기적으로 협력하는 프로세스들로 구성되는 기능 블록인 클러스터들의 집합으로 구성된다. 프로세스는 프로세스 포트에 부착된 비동기적인 채널에 의해 서로 간에 그리고 실세계와 통신할 수 있다. CIP 모델이 구현 단위로 분할 될 때 입출력 채널은 생성된 소프트웨어 컴포넌트를 위한 인터페이스 모델을 정의한다.

그림 5는 CIP 툴셋의 메인 화면이며 크게 COMMUNICATION NET 편집기, INTERACTION NET 편집기, MODE 편집기로 나누어진다. COMMUNICATION NET 편집기는 그림 5의 좌측 상단(1)의 그림이며 채널에 의해 다양한 클러스터들(Bonding Control, Dispensing, PickAndPlace)의 프로세스들을

상호연결하기 위해 사용된다. 클러스터 그룹뿐만 아니라 단일 클러스터는 동시발생으로 실행할 수 있는 컴포넌트로 자동으로 변형될 수 있다. 그림 5의 우측 상단([2]) 화면은 CIP 툴셋의 INTERACTION NET 편집기이며 한 클러스터(BondingControl)의 프로세스들(operator, manager, StepCtrl, loader, indexer, store, bonding) 사이의 멀티캐스트 연결을 생성한다. CASCADES는 루트 프로세스의 개별적인 외부 입력에 의해 유발되는 모든 상호작용 순서를 표현한다. 그림 5의 하단 중앙([3]) 화면은 상태 전이 다이어그램에 의해 한 프로세스 행위의 규정을 허용하는 MODE 편집기이다. 프로세스들은 상태 전이에서 수행되는 정적 변수, 입출력 데이터 타입, 오퍼레이션에 의해 확장된 상태 머신이다. 한 프로세스의 행위는 하나 이상의 양자택일 모드들에 의해 규정된다.

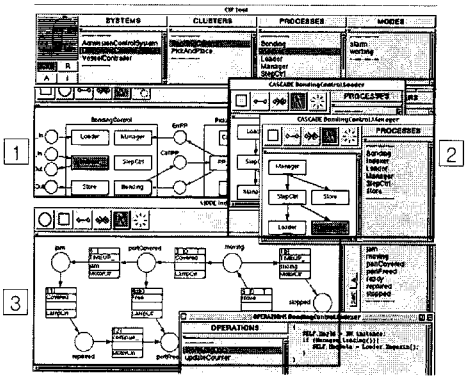


그림 5 CIP 툴셋의 메인 화면

CIP 모델은 프로세스들의 집합인 클러스터들로 조립되며 프로세스는 확장 유한 상태 머신으로 구성된다. 서로 다른 클러스터의 프로세스들끼리 또는 주변 환경과는 채널에 의해 통신한다. CIP 모델의 구현을 위해 클러스터들의 집합은 CIP 단위(unit)로 분할되며 자동으로 소프트웨어 컴포넌트로 변환된다. 여기서 CIP 단위는 CIP 쉘과 머신으로 구성되는데 CIP 쉘은 CIP 단위의 인터페이스를 표현하며 입출력 채널로 구성되는 반면에 CIP 머신은 CIP 단위의 반응 행위를 구현하는 수동 객체이다.

5.4 Rational Rose RT(Real-Time)

1999년 Rational Software 사가 객체지향 분석, 설계 도구인 Rational Rose와 객체지향 기반 실시간 내장형 시스템 개발 도구인 ObjectTime 사의 Object Time Developer를 통합하여 UML기반 실시간 내장형 시스템 개발 도구인 Rational Rose RealTime [75, 76, 77]을 공동 개발하였다. 그리고, ObjectTime이 고수해 온 방법론인 ROOM(RealTime Object-Oriented Methodology)를 포기하는 대신 UML을 확장한 UML-RT를 채택하였다.

Rational Rose RT의 특징은 개발 초기부터 지속적으로 이루어지는 통합 및 검증 작업, 코드 생성 자동화, 반복적인 개발 방법으로 생산성을 향상시킨다. 그리고, 실시간 내장형 시스템 개발에 UML을 사용함으로써 팀원들 간의 의사소통이 개선되며 코드 생성을 자동화한다. 이렇게 함으로서 설계를 코드로 구현하는 수작업 과정에서 생기는 오류를 제거할 수 있고 시간과 비용 절약을 통해 생산성이 높아질 수 있다. 또한, 컴포넌트 구조와 행위의 비주얼 모델링을 허용하며 동시성, 분산, 그리고 적시성(timeliness)과 같은 이슈들의 명시적 모델링을 지원한다.

모델들은 포트라고 불리우는 신호기반 객체를 통해 서로 상호 작용하는 능동의 객체(캡슐)를 사용해 조립되며 각 캡슐은 연관된 상태 기계를 갖는다[76]. 캡슐은 또한 서브 캡슐들을 포함할 수도 있으며 완전한 C 또는 C++ 기반 코드가 다양한 하드웨어, 실시간 운영체제를 대상으로 UML 모델로부터 직접 생성될 수 있다. [그림 6]은 어떻게 시스템이 Rose-RT로 규정될 수 있는지를 보여준다. 1에서 시스템 명세는 유스케이스 다이어그램으로서 기술된다. 2에서 클래스 다이어그램은 클래스들과 그들의 관계를 기술하기 위해 사용된다. 3에서 계층적인 상태 머신은 캡슐 클래스들의 행위를 획득하기 위해 사용된다. 4에서 캡슐 구조는 구조 다이어그램을 사용하여 획득된다.

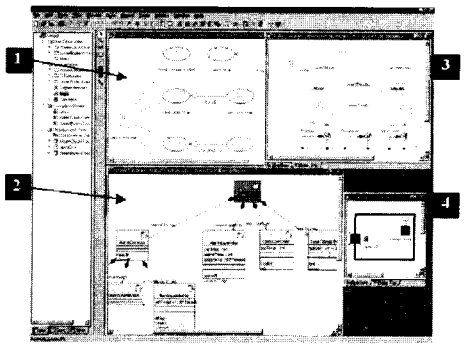


그림 6 Rational Rose RT의 주 화면

UML-RT를 형성하기 위해 UML 모델링 구조에 추가되는 구성물은 캡슐, 포트, 커넥터이며 프로토콜은 행위 모델링에 추가된다. 캡슐은 시스템에서 논리적으로 캡슐화된 쓰레드를 표현하는 컴포넌트이다. UML-RT에서 캡슐은 클래스에 의해 표현되며 특히 스테레오타입 <<Capsule>>에 의해 구체화된다. 캡슐은 조립 객체로 예를 들면, 캡슐은 수동성 객체 또는 다른 캡슐들에 의해 조립되어질 수 있는 내부 구조를 갖는다. 그림 7은 서브캡슐인 "CapsuleClassB"를 갖는 캡슐 "CapsuleClassA"를 보여준다. 캡슐의 내부 구조는 구조 다이어그램으로 규정되며 "CapsuleClassA"에 일치하는 구조 다이어그램은 그림 8에서 보여준다.

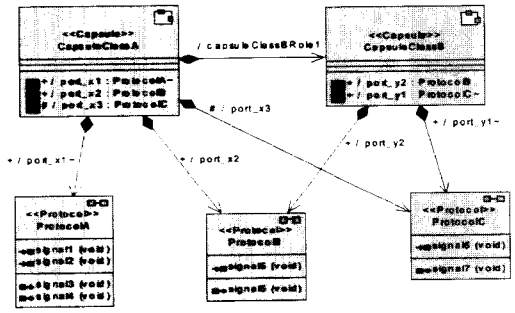


그림 7 캡슐을 위한 UML 표기

포트는 캡슐의 상호작용을 중재한다. 포트는 캡슐 구현의 물리적 부분이며 다중의 서로 다른 인터페이스를 위한 메카니즘을 제공한다. 캡슐의 외부로부터 볼 때 포트는 ID와 타입을 제외하고는 구분할 수 없으며 내부로부터 볼 때 포트는 두 종류(relay 포트, end 포트)가 될 수 있다. Relay 포트는 서브캡슐에 연결되는 반면에(예, 그림 8의 port_x2) end 포트는 직접 캡슐의 상태 기계에 연결된다(예, 그림 8의 port_x1과 port_x3). Relay 포트는 캡슐의 경계상에서만 나타날 수 있으며 결과적으로 늘 공용 가시성(public visibility)을 갖는 반면에 end 포트는 캡슐의 경계 또는 완전히 캡슐의 안쪽에서 나타날 수 있다. 커넥터는 포트를 연결하기 위해 사용되며 신호 기반 통신 채널의 추상 뷰이다.

프로토콜은 캡슐들의 연결된 포트들 사이의 정보(신호)의 유효한 흐름을 정의하며 포트로부터 보내고 받을 수 있는 입출력 신호들의 집합 명세이다. 프로토콜은 포트 타입을 정의하며 그것은 포트가 프로토콜에 의해 규정된 행위를 구현한다는 것을 의미한다.

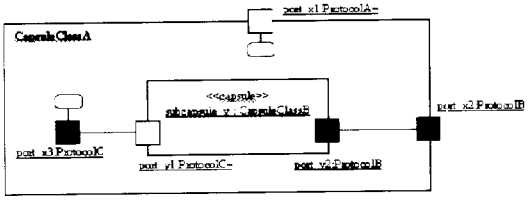


그림 8 포트의 UML 표기를 보여주는 Capsule ClassA의 구조 다이어그램

5.5 각 도구들간의 비교

본고에서 언급한 툴셋들을 비교해 보면 표 9와 같으며 각각의 특성은 다음과 같다. 첫째, 사용한 컴포넌트의 타입으로 실시간 컴포넌트는 수행해야하는 요구사항을 규정한 능동형(active) 컴포넌트와 다른 컴포넌트를 위한 서비스로서만 오직 수행되는 수동형(passive) 컴포넌트로 나누어진다[78]. 네가지 툴셋 모두 두 종류의 컴포넌트 또는 객체를 지원하고

표 9 내장형 시스템 개발 툴셋의 특성 비교

Property / Toolset	Component Type	Code Generation capabilities	Dependency Check	Analysis capabilities	Component Model
VEST	active passive	Not supported	Supported	Schedulability Reliability Real-Time	-
MetaH	active passive	Ada, C	Supported(limited)	Schedulability Reliability Safety Security	-
CIP	active passive	Java	Not supported	Interaction	CIP Model
Rational Rose RT	active passive	C, C++, Java	Supported(limited)	Schedulability	UML-RT

있으며 VEST의 경우 수동형 컴포넌트로는 메모리가 능동형 컴포넌트로는 프로세스가 이에 해당한다. 둘째, 코드 생성 능력으로 VEST는 현 버전에서는 지원하고 있지 않으며 나머지 툴셋들은 기본적인 C 언어 코드를 포함하여 C++, Java등의 언어로 생성되는 코드 생성 능력을 가지고 있다. 셋째, 의존성 검사 수행으로 오직 VEST만이 네가지 형태의 완전한 의존성 검사를 수행하며 MetaH는 제한된 의존성 검사를 수행한다. 그 외 CIP 툴셋은 의존성 검사는 아니지만 모델의 완전성을 자동 검사하는 기능을 포함하고 있다. 넷째, 분석 능력으로 VEST와 MetaH, Rose RT는 기본적으로 스케쥴 가능성 검사를 포함하는 분석 능력을 갖추고 있으며 CIP는 구성 시에 모든 잠재적인 상호작용 순서를 표현하는 것을 허용하는 자동화된 상호작용 분석만을 지원한다. 다섯째, 컴포넌트 모델은 컴포넌트 표준의 두 레벨인 조립과 상호작용 표준을 정의한다. VEST, MetaH 그리고 CIP는 툴셋에 필요한 모델을 정의하여 사용하였으며, Rose RT는 실시간 내장형 시스템 개발 방법론인 UML-RT에서 정의한 모델을 채택하였다.

6. 결 론

본 고에서는 분산 이동 내장형 시스템 지원을 위한 다양한 형태의 미들웨어 및 자바가상머신의 연구 개발 현황을 살펴보았고 이들을 신속하고, 효율적으로 개발하고 검증할 수 있는 개발 툴셋에 대한 연구 동향을 소개하였다. 내장형 시스템 연구들은 제한적인 하드웨어 및 소프트웨어 자원을 효율적으로 관리하거나(경량), 전력 소모를 줄이려는 연구(저전력), 사용자가 요구하는 수준의 서비스를 제공(QoS) 하려는 연구 그리고, 분산환경에서 자원을 공유해야하는 특징에 따라 취약한 보안의 허점을 개선하려는 연구(보안), 안전성 및 신뢰성을 확보하거나 고장허용을 지원하려는 연구가 각기 진행중이다. 그러나, 최근에는 내장형 실시간 시스템 환경에서 각기 다른 단일목표 지향적으로 진행되고 있는 현재의 연구 방법과는 달리 위에서 언급한 특징들을 통합된 단일 프레임(unified framework)내에서 사용자의 요구에 따른 이해득실을 고려하여, 최적화된 컴포넌트기반 미들웨어를 효과적으로 도출해 낼 수 있는 방법과 그를 위한 지원도구인 개발 툴셋에 대한 연구가 새로운 주제로 대두되고 있다.

참고문헌

[1] Edward A. Lee, "What's Ahead for Embedded Software?", IEEE Computer, September 2000, pp. 18-26

[2] Wolfgang Fleisch, Applying Use Cases for the Requirements Validation of Component-Based Real-Time Software. In Proceedings of 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'99), IEEE Computer Society Press, 1999

[3] Clemens Szyperski, Component Software-Beyond Object-Oriented Programming, Addison-Wesley, 1997.

[4] Stevens, W. R., "UNIX Network Programming. Prentice Hall," 1997.

[5] Pope, A., "The Corba Reference Guide: Understanding the Common Object Request Broker Architecture," Addison-Wesley, 1998.

[6] Baker, S., "Corba Distributed Objects : Using Orbix," Addison-Wesley, 1997.

[7] Natarajan, V., Reich, S., and Vasudevan, B., "Programming With Visibroker : A Developer's Guide to Visibroker for Java," John Wiley & Sons, 2000.

[8] OMG., "CORBA Component Model," http://www.omg.org/cgi-bin/doc?orbos/97-06-12, 1997.

[9] Rogerson, D., "Inside COM," Microsoft Press, 1997.

[10] Pitt, E. and McNiff, K., "Java.rmi : The Remote Method Invocation Guide," Addison Wesley, 2001.

[11] Monson-Haefel, R., "Enterprise Javabeans," O'Reilly & Associates, 2000.

[12] Monson-Haefel, R., Chappell, D. A., and Loukides, M.,"Java Message Service," O'Reilly&Associates, 2000.

[13] Redbooks, I., "MQSeries Version 5.1 Administration and Programming Examples," IBM Corporation, 1999.

[14] Hudders, E., "CICS: A Guide to Internal

Structure," Wiley, 1994.

[15] Hall, C., "Building Client/Server Applications Using TUXEDO," Wiley, 1996.

[16] Smith, B., "Reflection and Semantics in a Procedural Programming Language. Phd thesis(Jan), MIT, 1982.

[17] Kiczales, G., Des Riveres, J., And Borrow, D., "The Art of the Metaobject Protocol," The MIT Press, 1991.

[18] Yokoter, Y., "The Apertos reflective operating system: The concept and its implementation," In Proceedings of OOPSLA '92, pp. 414-434. ACM Press, 1992.

[19] McAffer, J., "Meta-level architecture support for distributed objects," In Proceedings of Reflection '96, pp. 39-62, 1996.

[20] ExoLab, "OpenORB," http://openorb.exolab.org/openorb.html, 2001.

[21] Ledoux, T., "OpenCorba: a Reflective Open Broker," In Reflection '99, Vol. 1616 of LNCS (Saint-Malo, France, 1999), pp. 197-224. Springer, 1999.

[22] Kon, F., Roman, M., Liu, P., Mao, J., Yamane, T., aes, L. M., and Cambpell, R., "Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB," In International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000), pp. 121-143 ACM/IFIP, 2000.

[23] Blair, G., Coulson, G., Robin, P., and Papathomas, M., "An Architecture for Next Generation Middleware," In Proceedings of Middleware '98(Sept. 1998), pp. 191-206., Springer Verlag, 1998.

[24] Plagemann, T., Eliassen, F., Goebel, V., Kristensen, T., and Rafaelsen, H., "Adaptive QoS Aware Binding of Persistent Objects," In Proceedings of International Symposium on Distributed Objects and Applications(DOA '99), IEEE Computer Society Press, 1999.

[25] Hanssen, ø., and Eliassen, F., "A Framework for Policy Bindings," In Proceedings of International Symposium on Distributed Objects and Applications(DOA '99), IEEE Computer Society Press, 1999.

[26] van Steen, M., Hauck, F., Homburg, P., and Tanenbaum, A., "Globe: a Wide-Area Distributed System," IEEE Concurrency 7, 1(March), 70-78, 1999.

[27] Bakker, A., van steen, M., and Tanenbaum, A., "From Remote Objects to Physically Distributed Objects," In Proc. 7th IEEE Workshop on Future Trends of Distributed Computing Systems(Cape Town, South Africa, Dec. 1999), pp. 47-52, 1999.

[28] Gelernter, D., "Generative Communication in Linda," ACM Transactions on Programming Languages and Systems 7, 1, 80-112, 1985.

[29] Wyckoff, P., McLaughry, S. W., Lehman, T. J., and Ford, D. A., "T Spaces," IBM Systems Journal 37, 3, 454-474, 1998.

[30] Waldo, J., "Javaspaces specication 1.0," Technical report (March), Sun Microsystems, 1998.

[31] Picco, G., Murphy, A., and Roman, G.-C., "Lime: Linda meets Mobility," In Proc. 21st Int. Conf. on Software Engineering (ICSE-99) (May 1999), pp. 368-377. ACM Press, 1999.

[32] Schilit, B., Adams, N., and Want, R., "Context-Aware Computing Applications," In Proc. of the Workshop on Mobile Computing Systems and Applications(Santa Cruz, CA, Dec. 1994), pp. 85-90, 1994.

[33] Asthana, A. and Krzyzanowski, M. C. P., "An indoor wireless system for personalized shopping assistence," In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications (Santa Cruz, California, Dec. 1994), pp. 69-74. IEEE Computer Society Press, 1994.

[34] Long, S., Kooper, R., Abowd, G., and Atkenson, C., "Rapid prototyping of mobile context-aware applications: the Cyberguide case study," In Proceedings of the Second Annual International Conference on Mobile Computing and Networking (White Plains,

NY, Nov. 1996), pp. 97-107. ACM Press, 1996.

[35] Bennett, F., Richardson, T., and Harter, A., "Teleporting-making applications mobile," In Proc. of the IEEE Workshop on Mobile Computing Systems and Applications (Santa Cruz, California, Dec. 1994), pp. 82-84. IEEE Computer Society Press, 1994.

[36] Brown, P., "Triggering information by context," Personal Technologies 2, 1 (March), 1-9, 1998.

[37] Dey, A., Futakawa, M., Salber, D., and Abowd, G., "The Conference Assistant: Combining Context-Awareness withWearable Computing," In Proc. of the 3rd International Symposium on wearable Computers (ISWC '99) (San Fransco, California, Oct. 1999), pp.21-28. IEEE Computer Society Press, 1999.

[38] Oracle Technology Network, "Oracle9i Application Server Wireless," http://technet.oracle.com/products/iaswe/content.html., 2000.

[39] Fritsch, D., Klinec, D., and Volz, S., "Positioning and Data Management Concepts for Location Aware Applications," In Proceedings of the 2nd International Symposium on Telegeoprocessing (Nice-Sophia-Antipolis, France, 2000), pp. 171-184, 2000.

[40] Alternis S.A., "Solutions for Location Data Mediation," http://www.alternis.fr/, 2000.

[41] SignalSoft. "Wireless Location services," http://www.signalsoftcorp.com/, 2000.

[42] CellPoint, Inc., "The CellPoint System," http://www.cellpt.com/thetechnology2.htm, 2000.

[43] Satyanarayanan, M., Kistler, J., Kumar, P., Okasaki, M., Siegel, E., and Steere, D., "Coda: A Highly Available File System for a Distributed Workstation Environment," IEEE Transactions on Computers 39, 4 (April), 447-459, 1990.

[44] Satyanarayanan, M., "Mobile Information Access," IEEE Personal Communications 3, 1 (Feb.), 26-33, 1996.

[45] Demers, A., Petersen, K., Spreitzer, M., Terry, D., Theimer, M., and welch, B., "The Bayou Architecture: Support for Data Sharing among Mobile Users," In Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications(Santa Cruz, California, Dec. 1994), pp. 2-7, 1994.

[46] Mascolo, C., Capra, L., and Emmerich, W., "An XML-based Middleware for Peerto-Peer Computing," In Proc. of the International Conference on Peer-to-Peer Computing (P2P 2001) (Linkopings, Sweden, Aug. 2001), 2001.

[47] Arnold, K., O'Sullivan, B., Scheifler, R. W., Waldo, J., and Wollrath, A., "The Jini[tm] Specication," Addison-Wesley, 1999.

[48] Sun's Java Community Process Real-Time Expert Group, "Proposed Java Real-Time API, v0.2," Technical report, Java Software Division of Sun Microsystems, December 1998. http://www. sdct.itl.nist.gov/carnahan/real-time/Sun/api

[49] The Real-Time for Java Expert Group, "Real-Time Specification for Java, v0.8.1," Technical report, RTJEG, September 1999. http:// www.rtj.org.

[50] J Consortium, "Draft International J Consortium Specification," Technical Report, J Consortium, September 1999. http://www.j-consortium.org/.

[51] K. H. Krause and W. Hartmann, "RT Java Proposal," Technical Report, A&D GT 1, 1999. http://www.j-consortium.org/rtjw

[52] A. Miyoshi, H. Tokuda, and T. Kitayama, "Implementation and Evaluation of Real-Time Java Threads," In Real-Time Systems Symposium. IEEE Computer Society, December 1997. page 166-174

[53] K. Nilsen, "Adding Real-Time Capabilities to Java," Communications of the ACM, 41(6), June 1998. page 49-56

[54] G. Hilderink, "A new Java thread model for concurrent programming of real-time systems," Real-Time Magazine, January 1998, pp. 30-35

[55] G. Back, P. Tullmann, L. Stoller, W. Hsieh,

and J. Lepreau. Java Operating Systems: Design an implementation. Technical report, Department of Computer Science, University of Utah, August 1998, www.cs.utah.edu/projects/flux.

[56] H. McGhan and M. O'Connor, picoJava: a direct execution engine for Java bytecode, IEEEComputer, 31(2), October, 1998

[57] G. Bollella, B. Brosgol, P. Dibble, et al., "The Real-Time Specification for Java", The Real-Time for Java Expert Group, December 1999.

[58] Java Community Process, "J2ME Connected, Limited Device Configuration," JSR-000030, May, 2000

[59] Java Community Process, "Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices," A White Paper, May, 2000.

[60] Java Community Process, "The K Virtual Machine," A White Paper, June 1999.

[61] I. H. Kazi, H.H. Chen, B. Stanley, and D. Lilja, "Techniques for Obtaining High Performance in Java Programs," ACM Computing Surveys Vol. 32, No.3, September 2000, pp. 213-240

[62] R. Radhakrishnan, et. al, "Java Runtime Systems: Characterization and Architectural Implications," IEEE Transactions on Computers, Vol. 50, No. 2, Feb. 2001, pp. 131-146

[63] Nozomi Nishimura, Paul Czerny, Maurice Klapfish, "1999 Embedded Operating Systems And Software Development Tools Strategic Market Intelligence Program: Real-time Operating Systems And Software Development Tools", A White Paper, Venture Development Corporation (http://www.vdc-corp.com), June 1999.

[64] Nancy Wu, "2001 Embedded Software Tools Worldwide Forecast", Gartner Dataquest (http://www.gartner.com), Nov 2001.

[65] J. Stankovic, "VEST: A Toolset For Constructing and Analyzing Component-Based Operating Systems for Embedded and Real-Time Systems," University of Virginia TR CS-2000-19, July 2000.

[66] J. Stankovic, et. al. "VEST: Virginia Embedded Systems Toolkit", IEEE/IEE Real-Time Embedded Systems Workshop, Dec 2001.

[67] J. Stankovic, et. al. "VEST: User's Manual", UVA, technical report, June 2001.

[68] J. Stankovic, "VEST: A Toolset for Constructing and Analyzing Component Based Embedded Systems", to appear, Springer-Verlag lecture notes.

[69] S. Vestal, "MetaH: Support for Real-Time Multi-Processor Avonics," Real-Time Systems Symposium, 1997

[70] Honeywell Technology Center, "MetaH User's Manual", technical report, 1998.

[71] Pam Binns, Steve Vestal, "Formalizing Software Architectures for Embedded Systems", International Workshop on Embedded Software, Oct 2001.

[72] Steve Vestal, Pam Binns, "Scheduling and Communication in MetaH", IEEE Proceedings, pp. 194-200, Dec 1993.

[73] Hugo Fierz, "The CIP Method: Component- and Model-Based Construction of Embedded Systems", European Software Engineering Conference 1999 - ESEC 99, LNCS Vol. 1687, pp. 374-391, 1999.

[74] CIP Tool-User Manual(1995-1999). CIP System AG, Solothurn, Swizerland. Internet: http://www.ciptool.ch

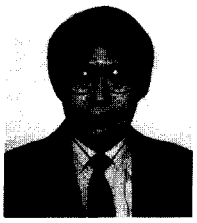
[75] http://www.rational.com

[76] Ross Albert McKegney, "Application of Patterns to Real-time Object-oriented Software Design", OOPSLA, 2000.

[77] Honourable mention, Magnus Antonsson, Pernilla Hansson, "Modeling of Real-Time Systems in UML with Rational Rose and Rose Real-Time based on RUP", RTiS'01, 2001.

[78] David C. Sharp, "Reducing Avionics Software Cost Through Component Based Product Line Development", In Software Technology Conference, Apr 1998.

이 승 룡



1978 고려대학교 재료공학 (학사)
1987 Illinois Institute of Technology (IIT), Chicago, Illinois, USA, 전산학 석사
1991 IIT, Chicago, Illinois, USA, 전산학 박사
1991~1993 Governors State University, Illinois, Department of Computer Science 조교수
1993~현재 경희대학교 전자정보학부 교수
관심분야:내장형 시스템, 실시간 시스템, 미들웨어 시스템, 멀티미디어 시스템
E-mail:sylee@oslab.kyunghee.ac.kr

허 서 경

2001 경희대학교 전자계산공학(학사)
2001~현재 경희대학교 전자계산공학과 석사과정
관심분야:가비지 컬렉션, 실시간 스케줄링, 자바 미들웨어
E-mail:skheo@oslab.kyunghee.ac.kr

서 영 준

1999 경희대학교 전자계산공학과(공학사)
2001 경희대학교 전자계산공학과(공학석사)
2001~ 경희대학교 전자계산공학과 박사과정
관심분야:CBSE, 디자인패턴, CASE 도구, S/W 재사용
E-mail:cionblue@orgio.net

송 영 재

1969 인하대학교 전기공학과(공학사)
1976 일본 Keio Univ. 전산학과(공학석사)
1979 명지대학교 대학원 졸업(공학박사)
1971~1973 일본 Toyo Seiko 연구원
1982~1983 미국 Maryland Univ. 전산학과 연구교수
1989~1990 일본 Keio Univ. 전산학과 객원교수
1984~1989 경희대학교 전자계산소장
1993~1995 경희대학교 교무처장
1996~1998 경희대학교 공과대학장
1998~2000 경희대학교 기획조정실장
1976~현재 경희대학교 전자정보학부 교수
2001~현재 경희대학교 산업정보대학원장
관심분야:소프트웨어공학, CBSE, CASE 도구, S/W 재사용
E-mail:yjsong@khu.ac.kr