ART : 임베디드 시스템에 적용 가능한 능동객체 실행시간 지원 시스템의 구현

박 윤 용[†]·임 동 선[††]·정 부 금[†††]·이 경 오[††††]·박 정 호[†]

요 약

능동객체는 스레드와 같은 독립적인 실행단위로서 객체가 생성될 때 CPU의 스케줄을 받아 실행되어 지는 개체이다. 본 논문에서는 이러한 능동객체를 정의하고, 능동객체의 생성 및 실행을 제어하는 능동객체실행시간 지원시스템(Active object RunTime support systems ; ART)을 제시하였다. 또한 ART는 분산 환경에서 사용자들에게 위치 투명성이 보장되고, 편리한 방식으로 분산 프로그래밍을 할 수 있도록 능동객체 사이에 메소드 호출을 지원할 수 있고, 이를 위한 통신 기법을 설계하고 구현하였다. 그리고 본 논문에서 제안하는 ART를 임베디드 시스템과 같이 시스템 자원이 제한되어 있고, 분산 환경으로 구성되어 있는 시스템에 적용하여 구현하였다.

ART : An Implementation on the Active_object RunTime Systems Applicable for the Embedded Systems

Yoon-Young Park[†] · Dong Sun Lim[††] · Bu-Geum Jung[†††]
Kyung-Oh Lee[††††] · Jung-Ho Park[†]

ABSTRACT

Active object is an independent runnable unit which is scheduled by CPU in creation time. In this paper, we define the active object and suggest ART(Active object RunTime support systems) which controls creation and execution of the active object. ART can provide users locational transparency and support easy method call mechanism. We also designed a communication model among active objects and implemented a communication method to make the distributed programing possible. The implementation target platform of ART is an embedded system which has only limited resources and runs in the distributed computing environment.

키워드 : 능동객체(Active Object), 스레드(Thread), 실행시간 지원시스템(Run Time Support System), 임베디드 시스템(Embedded System), 분산 시스템(Distributed System)

1. 서 론

단일 프로세서에서 사용자 프로그램들을 병행적(concurrently)으로 실행시키기 위한 시도가 지속적으로 발전하여 왔다. 초기의 운영체제에는 멀티프로세싱(multiprocessing)을 이용한 시분할 시스템에서 단일 실행 모듈 또는 프로세스 단위의 병행 수행을 오랫동안 사용하였고, 근래에는 프로세스 내의 스레드들을 병행적으로 실행시키면서 병행 수행의 단위를 세분화하였다. 일반적으로 스레드들을 병행 수행함으로써 CPU의 이용도를 더욱 향상시킬 수 있고, 병행 수행 단위를 스레드까지 세분화 함으로써 프로세스의 실행에 높은 효율을 기대할 수 있다.

※ 본 논문은 정보통신부 기초기술연구지원사업(C1-2002-068-0-4)으로 수행되었음.
† 종신회원 : 선문대학교 컴퓨터정보학부 교수
†† 정 회 원 : 한국전자통신연구원 네트워크연구소 책임연구원
††† 정 회 원 : 한국전자통신연구원 네트워크플랫폼팀 책임연구원
†††† 정 회 원 : 선문대학교 컴퓨터정보학부 교수
논문접수 : 2002년 11월 18일, 심사완료 : 2003년 3월 13일

또한, 근래에 객체지향 프로그램 언어들이 많이 사용되면서 객체들을 병행적으로 실행시키고자 하는 시도가 이루어지고 있다. 이러한 시도는 사용자들이 객체지향 언어의 근본적인 장점들을 이용하여 프로그램들을 개발할 수 있게 하고, 객체 수준에서 병행성을 보장함으로써 병행 수행의 단위를 더욱 세분화하여 프로그램 실행에 병행성을 높일 수 있다.

기존의 객체지향언어에 병행성을 추가하고자 하는 시도가 있었다. Actors 언어는 각각의 메소드를 스레드에 일치시키는 방법을 제안하였고[3], Ada언어에서는 객체를 프로세스와 일대일로 매핑시키는 방법을 제시하였다. Eiffel에서는 비동기적 채널과 예외를 사용하여 객체를 병행적으로 실행하고자 시도하였다[4]. 그러나 이러한 방식들은 대부분 객체를 병행적으로 실행하고 객체에 병행성을 표현하는 방식보다는 단순히 프로그램 실행 속도를 증가시키는 것에 중점을 두었다[2].

본 논문에서는 기존의 방식으로 실행되는 객체를 수동개

체(passive object)라 하고, 스레드와 같은 형태로 병행 수행시킬 수 있는 객체를 능동객체(active object)라고 정의하였다. 또한, 기존의 C++ 언어와 같은 객체지향언어에서 객체들을 병행 수행할 수 있는 능동객체 실행시간 지원 시스템(ART : Active_object Run Time support systems)을 제시하였다. ART는 기존의 C++ 언어에서 #DCL, #CALL과 같은 구문을 첨가하여 능동객체를 사용할 수 있게 하고, 실행시간에 능동객체의 생성 및 소멸 등의 생명주기를 관리하고, 능동객체 사이에 메소드 호출이 가능하도록 지원한다.

본 논문에서 제시하는 ART는 기존의 방식에서는 고려하지 않았던 분산환경에서의 능동객체들의 통신과 메소드 호출을 효율적으로 지원하고, 사용자들에게 위치투명성(location transparence)을 제공하기 때문에 쉽고, 편리하게 분산 프로그램이 가능하도록 하였다. 또한, CORBA 또는 RPC (Remote Procedure Call) 형태의 분산 프로그램과 유사한 방식을 적용하여 사용자들이 소규모의 분산 시스템에서 효율적으로 분산 프로그램을 작성할 수 있게 하였다. 따라서, ART는 임베디드 시스템과 같이 시스템 자원이 제한되어 있고, 분산환경으로 구성되어 있는 시스템에 적용할 수 있다.

임베디드 시스템은 시간적 제한성과 논리적 정확성을 동시에 요구하는 시스템으로서 산업체의 공정제어 및 자동화, 비행기의 관제 및 제어, 원자력 발전소의 통제 같은 분야에서 널리 사용되어져 왔다. 근래에 반도체 및 통신망 같은 컴퓨터 관련 기술의 급격한 발전과 정보화 사회에 대한 시대적 요구에 힘입어 임베디드 시스템은 그 응용 분야가 다양하게 확대되고 있다. 특히, 멀티미디어 초고속 통신, 인공지능 로봇, 첨단 군용 무기와 라우터 또는 교환기와 같은 통신 장비들에 이르기까지 다양한 응용분야에 임베디드 시스템이 적용되고 있다. 따라서 본 논문에서는 제안하는 ART를 임베디드 시스템에 사용할 수 있게 하기 위해 임베디드 시스템의 한 분야인 전전자 교환기 시스템에 적용하여 구현하였다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 기존의 객체지향언어에서 객체들을 병행 수행하는 방법들에 관하여 조사하였다. 3장에서는 능동객체를 정의하고 능동객체의 구성요소등을 설명하였다. 4장에서는 능동객체실행시간 지원시스템에 관하여 기술하였고, 능동객체와 스레드 통신 모델에 관하여 설명하였다. 또한, 능동객체실행지원시스템을 임베디드 시스템의 한 분야인 전전자 교환시스템에 구현하여 실행한 내용을 기술하였다.

2. 사례 연구

현재까지 객체지향언어에 병행실행기능을 추가하기 위한 시도가 있었지만, 객체를 병행적으로 실행시키기 위한 모델 및 병행적으로 실행되는 객체들을 관리하기 위한 방법들은 아직 체계적으로 정립되지 못한 실정이다. 근래에 프로그램 언어에 병행성을 추가하고자 하는 요구가 다양한 응용분야에서 확대됨으로써 병행객체지향 프로그래밍(Concurrent Object-Oriented Programming) 기법에 관한 많은 연구가 진행되어 왔고, Actor[8], CC++[9, 10]와 같은 언어들이 개발되어 졌다. 최근에는 C++ 언어에 병행성을 추가하고자 하는 시도가 증가하면서, ABC++, CC++, CHAOS++, COOL, C++ //, ICC++, MPC++, UC++ 등이 개발되었다[1, 8, 9, 10].

이러한 병행 객체지향 프로그램 언어의 모델은 크게 3가지 분야로 구별된다[1]. 첫 번째, 기존의 C++ 언어에 병행성을 지원하는 언어의 특성을 추가하여 C++ 언어를 확장하는 방법과 두 번째로는 병행성을 지원하는 라이브러리 클래스와 함수들을 사용하여 병행성을 제공하는 방법이 있다. 마지막으로 특정 언어에 새로운 구문(construct)을 추가하고 보조수단으로 라이브러리 또는 기타 도구(tools)들을 지원하여 새로운 언어로 확장해 가는 방법이 있다. 일반적으로 라이브러리에 기반한 접근방식은 구현하고 사용하기 쉽지만, 이러한 접근 방식을 사용하여 작성된 프로그램들은 사용자들이 읽기가 어렵고, 프로그램 구조를 이해하기 어렵게 된다. 이에 비해 새로운 구문을 추가하여 기존의 언어를 확장하여 병행성을 제공하는 방법은 프로그램을 이해하기 쉽게 하고 프로그램 판독성을 높일 수 있다. 따라서 본 논문에는 기존의 C++ 언어에 새로운 구성체들을 추가하고 다양한 런 타임 라이브러리와 도구들를 제공하는 방법을 적용하여 C++ 언어에서 병행성을 사용할 수 있게 하였다.

ConC++(또는 Concurrent C++)은 기존의 C++에 3가지의 키워드를 추가하고 구문을 약간 변경하여 병행 수행을 지원한다[1]. 특히, ConC++은 프로그램을 효율적으로 병행 수행할 수 있게 하고, 사용하기 쉽고 프로그램 판독성을 높이는 것을 설계의 목표로 하였다. ConC++은 병행클래스(concurrent class)로부터 생성된 능동객체(active object)와 보호된 클래스(protected class)로부터 생성된 수동객체(passive object)등 두 가지 종류의 객체를 제공한다. 객체를 병행수행의 단위로 하였고, 능동객체는 생성자 함수가 실행된 후에 능동함수(active function)가 실행되도록 하였다. 능동함수는 객체를 스레드와 같이 독자적으로 실행할 수 있게 하는 함수로써 함수 앞에 # 기호를 사용하여 일반 함수들과 구별하게 하였다.

sC++에서는 능동(active)객체의 구문을 수동(passive)객체에서 사용하는 구문과 동일하게 사용할 수 있게 하고, 능동객체를 스레드와 같은 형태로 병행적으로 실행할 수 있게 하였다[2]. sC++에서는 다른 능동객체에 의해 메소드가 호출되어 질 때가지 메소드의 실행을 지연시킬 수 있다. 또한, 여러 능동객체들이 동시에 같은 메소드를 호출하면 호출 정보를 대기 리스트(waiting list)에 저장하였다가 순서적으로 메소드를 호출할 수 있게 한다. 그리고 sC++에는

본체(body)라고 불리는 특별한 형태의 메소드가 능동객체의 내부 실행을 결정하고, 이 특별한 메소드는 클래스이름과 같은 이름을 사용하며 메소드 이름 앞에 @ 기호를 첨가하여 구분한다. 이러한 메소드는 ART에서는 리셉터 함수와 유사한 역할을 한다.

Balinda C++[5]은 Linda[6]에서 유래되었고, ANSI C++에 여섯 개의 구문(construct)을 추가하여 만들어진 병행 객체 지형언어이다. 즉, Balinda C++은 기본적으로는 ANSI C++의 객체 지향 언어의 특성을 유지하면서, 기존의 C++ 언어에 튜플(tuple)관련 연산 문장인 IN, OUT, RD와 병행수행을 지원하는 EXEC, ENDOFTASK 그리고 동기화 관련된 문장인 SYNCHRONIZE와 같은 구성체를 추가하여 병행 객체지향언어를 구성하였다. Balinda C++에는 위와 같은 구성체를 번역할 수 있는 전처리기와 병행 수행 기능을 지원하는 런타임시스템으로 구성되었다. 또한, Balinda C++은 운영체제와 같은 시스템 소프트웨어와의 인터페이스가 없도록 구성하여 이식성을 높였고, 사용자들이 프로그래밍을 쉽게 할 수 있도록 하였다. 또한, 적은 비용으로 병행 수행을 지원할 수 있다[7].

본 논문에서 제안하는 방법은 기존의 C++ 언어에 새로운 구문을 추가하여 객체 지향 언어에 병행성을 지원하는 방법은 동일하다. 그러나 제안하는 병행객체실행시간 지원시스템은 미들웨어 구조를 가지기 때문에 커널이나 사용하는 프로그램 언어의 종류와 독립적으로 실시간에 객체들의 병행 수행을 지원할 수 있다. 즉, ART는 C++뿐만 아니고 다른 객체 지향 언어에서의 객체들의 병행 수행을 지원한다. 본 논문에서는 ATM 교환기인 HANbit ACE 시스템에서 C++뿐만 아니고 OO-CHILL언어에서의 병행 수행을 지원하도록 구현하였다. 또한, 기존의 UNIX 스레드는 분산환경에서의 스레드 사이의 통신을 지원하지 못하지만, 본 논문에서는 분산 환경에서 스레드 또는 능동객체 사이의 메소드 호출을 메시지 전달 형태로 완벽하게 지원한다. 그리고 ART는 전처리기 등의 도구를 이용하여 사용자들에게 위치 투명성(location transparency)이 쉽게 보장되도록 하며, 효율적으로 분산 프로그램을 작성할 수 있게 한다.

3. 능동객체 실행 모델

본 절에서는 능동객체를 정의하고, 능동객체의 구성요소에 관하여 기술하였다. 또한, 능동객체를 생성하기 위해 (그림 3)과 같은 형태로 능동클래스를 정의하고, ACTCLASS와 DATA 그리고 FUCTION등의 키워드를 사용하여 멤버변수와 메소드들을 정의할 수 있고, 같은 능동클래스로부터 생성된 능동객체들은 메소드들을 공유하고 멤버변수들은 능동객체 단위로 독립적으로 사용할 수 있다. 따라서 능동객체의 멤버변수들에는 객체 지향의 개념에서 제공되는 기본적인 정보은폐(information hiding)와 캡슐화(encapsulation)의 개념들이 적용된다. 본 논문에서는 일반적으로 객체지향언어에서 사용되는 객체를 수동객체 또는 일반객체로 정의하고, 수동객체와 구별되는 능동객체를 다음과 같이 정의한다.

[정의 1] 능동 객체의 정의

능동객체(active object)는 능동클래스(active class)로부터 생성되는 객체로서 스레드와 같이 독립적으로 스케쥴되어 CPU에 의해 실행될 수 있는 실행단위이다.

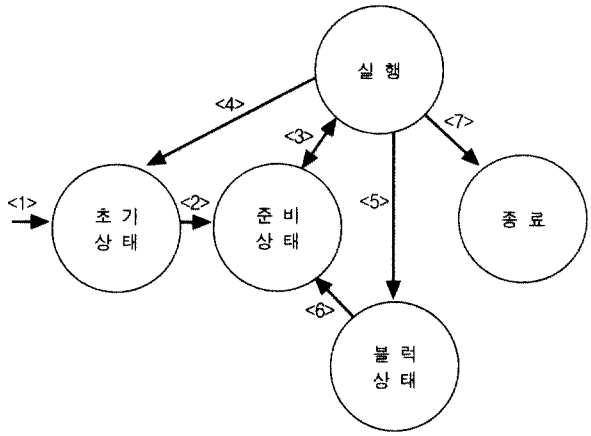
능동객체는 일반객체와 같이 멤버변수와 메소드들로 구성되어 있고, 일반 또는 수동 객체에서 생성자(constructor) 함수와 유사한 기능을 하는 리셉터 함수(receptor function)로 구성되어 있다. 또한, 능동객체는 스레드와 같이 독립적으로 실행할 수 있는 단위이기 때문에 능동객체의 메소드 호출은 단순한 서부루틴 호출로는 구현할 수 없다. 따라서 능동객체의 메소드 호출을 구현하기 위해 객체 사이의 메소드 호출을 메시지 전달을 통한 통신 기능으로 변환해 주고 수신한 메시지를 분석하여 연관된 메소드를 호출해 주는 기능이 요구된다. 이러한 기능을 지원하기 위해 ART에서는 리셉터 함수를 구현하였다.

[정의 2] 리셉터 함수(receptor function)

능동객체가 생성되어 처음으로 시작하는 루틴(startup routine)으로 수신된 메시지를 분석하여 메시지에 포함된 인수(arguments)들을 가지고 해당 메소드를 호출해 주는 기능을 가진 함수를 능동객체의 리셉터 함수라고 정의한다.

즉, 리셉터 함수의 역할은 기본적으로 생성된 능동객체를 메시지 수신을 위한 블록상태로 존재하게 하고, 능동객체가 다른 객체로부터 메시지를 수신하면 메시지 내용을 분석하여 관련된 메소드를 호출하여 처리하는 과정을 반복적으로 수행하게 한다.

능동객체는 기본적으로 일반 프로세스 또는 스레드와 같이 준비 및 실행(ready and run) 그리고 블록(block) 상태를 가진다. 그러나 능동객체는 능동클래스로부터 생성되어 초기상태(initial state)를 유지한다. 초기상태는 (그림 1)의 <1>과 같이 능동객체가 생성되어 처음 시작될 때의 상태로서, [정의 2]에 정의된 디폴트 시작 루틴인 리셉터 함수(receptor function)가 실행되는 상태이다. 능동객체는 초기상태에서 다른 스레드 또는 능동객체로부터 메소드 호출 요구를 수신하여 (그림 1)의 <2> → <3>과 같이 관련된 메소드를 실행하게 된다. 메소드 실행이 종료되면 다른 객체들의 메소드 호출 요구를 수신하기 위해 초기상태로 전환된다((그림 1)의 <4>). 또한, 능동객체는 실행 중에 입출력 요구등에 의해 (그림 1)의 <5>와 같이 블록상태에 있을 수 있다.



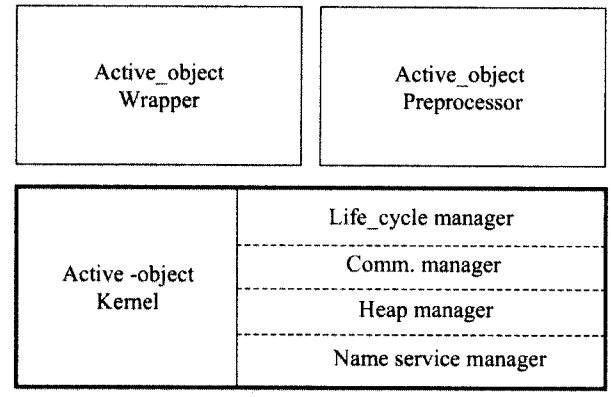
(그림 1) 능동객체의 상태 전이도

능동객체는 다른 객체의 메소드 호출을 처리하기 위해 한 번 생성되면 죽지 않고 계속적으로 살아 있게 된다. 따라서 시간이 지나면 살아 있는 능동객체의 수가 증가하여 시스템 자원들이 고갈되는 문제가 발생하기 때문에 ART에서는 일정한 스레쉬홀드(threshold) 값을 정하여 능동객체의 수를 유지하게 한다. 이 때 생성된 능동객체를 죽이기 위해서는 해당 능동객체에 kill 시스널이 있는 메소드를 호출하여 (그림 1)의 <2>→<3>→<7> 단계를 거쳐 능동객체를 종료시킨다.

일반적으로 종료된 능동객체에게 메소드 호출을 하게되면, 메시지를 수신하여 처리할 수 없게 된다. 이와 같이 처리되지 못하고 시스템 내의 메시지 큐에 남아 있는 메시지를 고아 메시지(orphan message)라고 한다. 고아 메시지는 처리되지 못한 메시지가 시스템 내의 버퍼에 쌓이게 하고, 송신자(sender)를 무한정 블록상태로 만들 수 있기 때문에 시스템 성능을 나쁘게 한다. 따라서 본 논문에서는 생성된 능동객체를 정적(static)하게 유지하여 종료되지 않게 구성함으로써 고아메시지가 발생하지 않도록 하였다. 이것은 대규모의 분산 시스템에서는 어렵지만, 본 논문에서 고려하는 작은 규모의 임베디드 시스템으로 구성된 분산환경에서는 생성된 능동객체들을 정적으로 유지하는 것이 시스템의 성능을 최적으로 유지할 수 있게 한다.

4. ART의 설계와 구현

ART는 기존의 C++과 같은 객체지향 언어에서 능동객체를 사용할 수 있도록 하는 실행시간 지원 시스템이다. ART는 (그림 2)와 같이 능동객체 래퍼(Active-object Wrapper), 능동객체 전처리기(Active-object Preprocessor) 그리고 능동객체 커널(Active-object Kernel)로 구성되어 있다. 능동객체 커널은 능동객체를 사용하기 위한 프리미티브들로 구성되어 있고, 능동객체 래퍼와 전처리기는 사용자들이 능동객체 커널에 관한 지식이 없이도 능동객체를 쉽고, 편리하게 사용할 수 있도록 하는 도구들이다.



(그림 2) ART의 구성요소

4.1 능동객체 커널(Active-object Kernel)

능동객체 커널은 (그림 2)와 같이 C++로 작성한 사용자 프로그램이 분산환경에서 실행하는 동안 능동객체의 생성, 소멸 등을 관리하는 생명주기 관리기(life_cycle manager)와 능동객체의 메소드 호출을 관리하는 통신 관리기(communication manager), 능동객체의 멤버데이터와 메소드 호출에 사용되는 메시지를 저장하는 히프메모리 관리기(heap manager), 그리고 분산 환경에서 능동객체 사이의 메소드 호출을 지원하기 위한 네이밍서비스 관리기(name service manager)등으로 구성되어 있다. 이러한 기능들을 프리미티브 형태로 사용자 프로그램에 제공한다.

4.1.1 능동객체 생명주기 관리기

실행시간에 능동객체의 생성에서 종료까지의 생명주기를 관리한다. 능동 객체를 생성하여 실행시키기 위해

```
actobj_start( char *cname, char *oname, object_id *oid,
int dsize, char *attr, void (*func)(), char *arg) ;     (1)
```

프리미티브를 사용한다.

ART는 능동객체(또는 스레드)가 생성되는 시점에 ART 내의 히프메모리 공간에 acb(active_object control block) 또는 tcb(thread control block)를 만들어서 생성된 능동객체들을 유지, 관리한다. 이러한 acb 또는 tcb 내에는 능동객체(또는 스레드)의 식별명, 메시지 버퍼(m_buf : message buffer) 그리고 능동클래스에 정의된 멤버변수들이 저장된 주소 및 기타 정보 등을 저장한다.

위의 식 (1)의 함수에서 인수 cname은 능동객체를 생성하는 능동클래스 이름이고, oname은 생성하고자 하는 능동객체의 이름으로 사용자가 작성한 프로그램에서는 능동객체의 이름으로 능동객체들을 접근하고 메소드 호출을 할 수 있게 한다. oid는 능동객체를 성공적으로 생성한 경우에 ART에서 복귀해 주는 능동객체 식별명이다. dsize는 능동

클래스에 정의된 멤버변수들의 크기이다. 이것은 능동객체를 생성할 때 능동객체에 필요한 멤버변수들의 저장 공간을 ART 내의 히프메모리 공간에 확보하기 위해 사용된다. 따라서 멤버변수들은 각각의 능동객체에 대해서 별도의 멤버변수 저장공간이 확보된다. func()은 [정의 2]에서 정의한 리셉터 함수로서 능동객체가 생성되어 실행되는 시작 함수의 이름이다.

이외에도 실행 중인 능동객체를 종료하기 위해 actobj_exit(), 특정 능동객체의 실행을 중지하기 위해 actobj_kill() 그리고 특정시간 동안 능동객체를 블록상태로 하기 위해 actobj_sleep()과 같은 프리미티브들을 제공한다.

4.1.2 능동객체와 스레드 통신 관리기

능동객체는 스레드와 같이 독립적으로 실행할 수 있는 단위이므로 능동객체의 메소드 호출은 일반객체의 메소드 호출과는 다른 방식으로 구현된다. 일반적인 UNIX에서도 스레드 사이의 통신은 동일한 실행파일 내에서만 제한적으로 지원되고 있다. 본 논문에서 제안하는 ART는 같은 프로세스 뿐만 아니고 다른 프로세스에 존재하는 능동객체(또는 스레드)사이의 통신을 효율적으로 지원한다.

ART는 분산 환경에서 능동객체의 메소드 호출을 지원하기 위한 메시지박스(m_box : message box)와 전역 네이밍 서비스 기능을 지원한다. 메시지버퍼, m_box는 호출한 능동객체와 수신한 능동객체 사이에서 메소드 호출이 이루어지도록 메시지 내용을 저장한다. ART는 능동객체의 메소드 호출을 지원하기 위해 actobj_send()와 actobj_rcv() 함수를 제공한다. 능동객체의 메소드를 호출하기 위해 호출하는 능동객체는

```
actobj_send( char *hostname, int *cname, int *oname,
int mid, char *arg, char *ret_addr, int ret_type) ;    (2)
```

프리미티브를 사용하고, 수신하는 능동객체는

```
actobj_rcv( char *rcvmsg ) ;                           (3)
```

프리미티브를 사용한다.

분산 환경에 적용하기 위해 위의 식 (2)에서와 같이 분산 시스템 내의 호스트 이름(hostname)과 클래스 이름(cname), 객체 이름(oname)으로 멤버함수를 호출할 수 있도록 설계하였다. 인수 mid는 호출되는 메소드의 함수 번호로 ART에서 자동으로 부여하여 관리하는 함수번호이다. 사용자는 일반 객체의 멤버함수 호출과 같은 방식으로 호출하면 ART에서는 해당 멤버함수의 고유한 번호로 변환하여 멤버함수 호출을 한다.

또한, ART에서는 비동기 및 동기적 방식의 메소드 호출을 지원한다. 식 (2)에 인수 ret_type에 따라서 복귀값이 필요없는 경우에는 비동기 메소드 호출 방식으로 실행되게 되고, 메소드를 호출하는 능동객체는 전달하고자 하는 메시지 만을 ART에 m_box에 저장하고 자신은 계속 진행하게 된다. m_box에 저장되어 있는 메시지는 수신하는 능동객체가 식 (3)에 있는 actobj_rcv()함수를 통해 전달받게 된다. 복귀값이 필요한 경우에는 동기적 멤버함수 호출 방식이 적용되고, 메소드의 실행 결과가 전달되어 질 때까지 호출한 능동객체는 블록상태로 유지하게 한다.

4.1.3 능동객체의 네이밍 서비스 관리기

네이밍 서비스 기능은 능동 클래스와 능동 객체 이름을 분산 환경 내에서 해당하는 고유한 식별명으로 상호 변환하여 주는 기능을 제공한다. 능동 클래스와 능동객체의 식별명은 능동객체가 생성되는 시점에 ART에 의해 부여되는 고유한 번호로서 ART 내에서 능동객체를 관리하기 위해 사용한다. 따라서 프로그래머들은 능동객체의 식별명과 관계없이 자신이 정의한 능동객체의 이름으로 프로그램할 수 있다.

ART에서는 능동클래스 및 능동객체의 식별명을 입력으로 하여 능동클래스 및 능동객체의 이름을 제공해 주는 actobj_classname(), actobj_objectname()의 프리미티브를 구현하였다. 또한, 능동클래스 이름과 능동 객체의 이름을 입력으로하여 능동객체의 식별명을 제공해 주는 actobj_name_resolve() 프리미티브를 제공한다. 이외에도 현재 실행 중인 능동객체의 식별명을 제공하는 actobj_this() 및 actobj_getpid() 등의 프리미티브를 구현하였다.

4.1.4 능동객체의 히프메모리 관리기

히프메모리 관리기는 ART내에서 정의된 능동클래스 및 생성된 능동객체에게 요구되는 메모리 공간을 제공하는 기능을 한다. 또한, 능동객체간의 메소드 호출을 지원하기 위해 메소드 호출의 내용을 메시지 형태로 저장하고 관리하기 위해 필요한 메모리 공간을 제공한다. 그리고 전역 네이밍 서비스를 제공하기 위해 정의된 능동클래스 및 능동객체의 이름과 식별명을 저장하기 위한 공간을 관리한다. 이를 위해 ART내의 효율적으로 메모리를 할당하고 반납하기 위한 ART_malloc() 및 ART_free() 함수를 구현하였다.

4.2 능동객체 래퍼(Active-object Wrapper)

능동객체 래퍼는 기존의 C++과 같은 객체 지향 언어에서 능동객체를 쉽게 정의하고 생성할 수 있게 한다. 특히, 능동객체의 초기 실행 코드인 리셉터 함수의 코드를 자동으로 생성해 준다.

능동객체 래퍼는, UNIX의 rpcgen 명령어와 유사하게, 능동클래스 정의 파일로부터 능동객체 실행 코드를 생성해 준다. 능동클래스 정의 파일은 UNIX의 원격프로시져 프로그램 방식과 유사하게 작성하고, 능동객체 래퍼는 사용자가 작성한 능동클래스 정의 파일을 해석해서 생성자 함수에

해당하는 리셉터 함수의 코드를 자동으로 생성하여 준다.

(그림 3)은 능동클래스 grid를 정의하기 위한 능동클래스 정의 파일로서 ACTCLASS, DATA, FUCTION등의 키워드(keywords)를 사용한다. 키워드 ACTCLASS 뒤에는 능동클래스의 이름을 정의하고, 키워드 DATA 뒤에는 멤버변수들을 정의한다. 키워드 FUCTION 다음에는 능동클래스가 사용하는 멤버함수의 원형을 정의한다. 따라서 (그림 3)은 능동클래스 이름이 grid이고 멤버변수로 정수 A와 문자배열 B를 가지고, sum()이라는 메소드로 구성된 능동클래스를 정의하게 된다.

[정의 3] 능동객체 래퍼(Active-object Wrapper)

능동객체 래퍼는 능동클래스정의 파일로부터 능동객체의 실행코드인 리셉터 함수와 관련된 헤더파일을 자동으로 생성해 주는 도구이다.

```
ACTCLASS grid {
DATA :
  int A ;
 char B[10] :
FUCTION :
  sum( int x, int y) ;
} 0×1 ;
```

(그림 3) grid.a

```
struct grid_data {
    int a ;
    char   b[ 10 ] ;
    int     *self ;
} ;
#define A   lvar -> a
#define B   lvar -> b
```

(그림 4) grid.h

능동객체 래퍼는 (그림 3)의 능동객체 정의파일을 분석하여 (그림 4)와 같은 능동객체와 관련된 헤더파일 grid.h와 (그림 5)와 같은 능동객체 실행코드 파일 grid.pc를 자동으로 생성한다. 파일 grid.pc에는 grid형의 능동객체들이 사용하는 리셉터 함수 grid_receptor()가 자동 생성되고, 이 함수

```
# include   "grid.h "
void grid_receptor() {
{
    /* 중략 1 */
        while(){
        actobj _ rcv (& rcvmsg rcvmsg) ;
        switch(rcvmsg.fnum) {
        {
            case  0 ;
              actobj_exit (EXIT_ACTOBJ) ;
              breaK ;
            case  GRID_SUM :
              /* 중략 2 */
              grid_sum (lvar, _int_arg [0], _int _ arg [1] ;
              break ;
               /* 중략 3 */
        }
    }
}

grid_sum( struct grid_data * lvar, int x, int y)
{
/* 중략 4 */  put user code hrer */
}
```

(그림 5) grid.pc

는 grid 형의 능동객체들이 생성될 때마다 시작하여 actobj_rcv() 함수에 의해 다른 능동객체들로부터의 메소드 호출 요구를 처리하기 위해 대기한다.

사용자들은 (그림 5)에서 '중략 4' 부분에 자신이 구현하고자 하는 함수의 기능을 코딩하면 된다. '중략 1' 부분은 능동객체가 생성되어 필요한 기본적인 초기화 과정의 코드가 자동 삽입되고, '중략 2' 부분에서는 수신한 메시지를 분석하여 메소드가 사용하는 인수를 찾아내는 코드가 자동 생성되어 해당 메소드를 호출하게 된다. '중략 3' 부분에는 능동클래스정의 파일에 추가적으로 정의한 메소드들을 연결하기 위한 코드가 자동 생성된다.

4.3 능동객체 전처리기(Active-object Preprocessor)

능동객체 전처리기는 C++과 같은 객체 지향 프로그램을 사용하는 일반 사용자들이 ART의 능동객체 커널(active object kernel)에서 제공하는 프리미티브에 관한 지식이 없이도 능동객체의 선언 및 생성과 메소드 호출을 사용할 수 있도록 해 주는 도구이다. 즉, (그림 6)과 같이 능동객체의 선언 및 생성을 위해 #DCL 구문을 이용하게 하고, 능동객체의 메소드 호출을 위해 #CALL 이라는 구분을 사용하게 함으로써 일반 객체의 생성 및 메소드 호출과 유사하게 사용할 수 있게 하였다. (그림 6)에서는 능동클래스 grid형의 능동객체 tiger를 선언 및 생성하고, 호스트이름 rainbow에 있는 능동객체 tiger의 메소드 sum()을 호출하는 예를 보여주는 것이다.

```
# include   "grid.h"
main ()
{
# DCL   grid    tiger ;

   /* 중략 1*/
# CALL  ret = rainbow.grid.tiger.sum (10,  20) ;

   /* 중략 2 */
}
```

(그림 6) sample.pc

사용자들은 일반 객체의 생성과 유사한 구문으로 능동객체를 사용하고 전 처리기는 식 (1)의 프리미티브를 사용해서 능동객체를 생성하는 코드를 만들어 준다. 메소드 호출의 경우에도 사용자들은 일반적인 닷 표기법을 이용하여 호출하면, 전처리기에 의해 식 (2)의 프리미티브를 생성할 수 있게 하였다. 따라서 능동객체 전처리기는 사용자들이 작성한 확장자가 pc로 만들어진 파일을 분석해서 능동객체 생성 및 메소드 호출 관련 코드를 생성해 주는 기능을 한다.

4.4 ART에서 응용 프로그램 개발

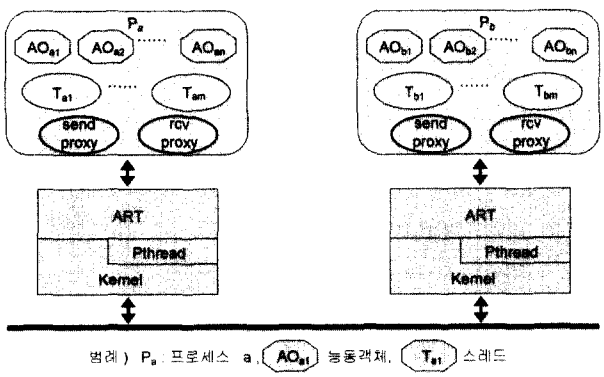
ART에서 응용 프로그램을 개발하는 절차를 요약하면 아래와 같다. 기본적으로 사용자들은 자신의 사용하는 분산환

경에 관하여 호스트 이름, IP 주소, 그리고 능동객체들의 정적(static)인 구성(configuration)등에 관하여 설계한다. 첫 번째 단계는 능동클래스 정의 파일 작성하는 것이다. 파일의 확장자는 '*.a'를 사용하고, (그림 3)과 같은 형식을 사용한다. 이것은 UNIX에서 RPC 프로그램을 위해 작성하는 정의파일과 유사하다.

두 번째 단계에서는 '능동객체 래퍼'를 이용하여 능동클래스 정의 파일로부터 능동 객체 코드를 자동 생성한다. 이 경우에 필요한 헤더파일과 리셉터 함수 그리고 메소드 함수의 원형(prototype)이 자동 생성되고, 파일 이름의 확장자는 '*.pc'가 사용된다. 다음 단계에는 사용자의 응용 프로그램들을 작성하게 된다. 사용자들은 (그림 6)에서와 같이 '*.pc'의 확장자 명으로 된 응용 프로그램들을 작성한다. 네 번째 단계에서는 '*.pc'로 작성된 사용자 프로그램을 능동객체 전처리기를 이용하여 확장자가 '*.c'인 파일을 생성한다. 이후에 컴파일러를 이용하여 실행코드를 생성하게 된다.

4.5 능동객체와 스레드 통신 모델

능동객체는 스레드와 같이 독립적으로 스케쥴 되어 실행되는 단위이기 때문에 능동객체의 메소드 호출을 지원하기 위한 별도의 통신 모델이 요구된다. 현재 UNIX에서 스레드 사이의 통신은 동일한 실행파일(또는 프로세스) 내에서만 제한적으로 지원되고, 다른 프로세스에 존재하는 스레드 사이의 통신은 지원되지 않는다. 본 논문에서 제안한 ART는 같은 프로세스 뿐만 아니고 다른 프로세스에 존재하는 능동객체와 능동객체, 스레드와 스레드 그리고 능동객체와 스레드 사의의 통신을 효율적으로 지원할 수 있다.

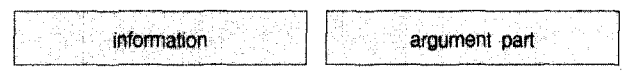


(그림 7) ART의 능동객체 통신 모델

(그림 7)은 ART를 이용한 능동객체 또는 스레드 사이의 통신 모델을 도시한 것이다. 프로세스 P_a와 P_b는 각각 능동객체(AO_{a1}~AO_{an}, AO_{b1}~AO_{bn})들과 스레드(T_{a1}~T_{am}, T_b~T_{bm})들로 구성되어 있다. 본 논문에서는 같은 프로세스 내에 있는 능동객체(또는 스레드) 사이의 통신을 지역 메소드 호출(local method call), 다른 프로세스 내에 있는 능동객체(또는 스레드) 사이의 통신을 원격 메소드 호출(remote method call) 이라고 정의한다.

ART는 원격 메소드 호출을 지원하기 위하여 스레드 형태를 가지고 있는 send_proxy와 rcv_proxy를 (그림 7)과 같이 각 프로세스가 생성될 때 자동으로 만들어 준다. send_proxy는 송신 요구를 받아 원격지의 rcv_proxy에게 전달한다. rcv_proxy는 원격의 send_proxy로부터 수신한 메시지를 분석하여 해당 능동객체(또는 스레드)에게 메시지를 전달한다.

또한, ART는 메소드 호출을 단순한 함수 호출이 아니고, 능동객체 사이의 통신 형태로 지원하기 위하여 메소드 호출에 사용된 인수들을 메시지 형식으로 전환한다((그림 9)의 라인 1). 이와 같은 작업을 마셜링(marshalling)이라 하고, 사용자들이 (그림 6)의 sample.pc 파일과 같은 형식으로 메소드를 호출하면, (그림 8)과 같은 메시지 형태로 만들어 준다. (그림 8)의 '인수저장부(argument part)'는 마셜된 인수들을 저장하기 위해 사용되고, '정보저장부(information part)'는 호스트이름, 능동클래스 이름, 능동객체 이름 그리고 메소드 함수의 식별명 등이 마셜되어 저장된다. 지역메소드 호출인 경우에는 '인수저장부'만이 메시지형태로 사용되고, 원격 메소드 호출인 경우에는 '정보저장부'까지 마셜링되어 메시지형태로 만들어 진다. 그리고 (그림 5)의 리셉터 함수에서는 수신된 메시지를 정상적인 함수 호출 형식으로 만들어 주는 언마셜링(unmarshalling) 기능을 제공한다.



(그림 8) ART의 메시지 형식

ART는 지역 및 원격 메소드 호출을 지원하기 위하여 능동객체(또는 스레드)가 생성되는 시점에 능동객체(또는 스레드)에 관한 정보를 저장하고 유지, 관리한다. ART 내에서 이와 같은 정보를 유지하는 장소를 각각 acb(active_object control block) 또는 tcb(thread control block)라 하고, 능동객체(또는 스레드)의 식별명, 메시지 버퍼(m_buf : message buffer) 및 기타 정보 등을 저장한다.

ART는 메소드 호출을 지원하기 위해 메시지를 저장하는 공간으로 m_buf(메시지 버퍼)와 m_box(message box : 메시지 박스)를 제공한다. m_buf는 능동객체(또는 스레드)의 acb 또는 tcb 내에 존재하는 저장공간으로 m_buf는 메소드를 호출한 능동객체가 전송한 메시지를 저장하기 위한 공간으로 사용된다((그림 9)의 라인 3, 4, 5). 수신할 능동객체가 메시지를 수신할 수 없는 경우에는 전송된 메시지를 ART내의 m_box에 저장하였다가 수신자에게 전달된다((그림 9)의 라인7).

ART는 원격 메소드 호출을 지원하기 위하여 스레드 형태를 가지고 있는 send_proxy와 rcv_proxy를 (그림 7)과 같이

각 프로세스가 생성될 때 자동으로 만들어 준다. (그림 9)의 라인 10, 11, 12에서 송신자가 원격으로 메소드 호출을 하는 경우에 메시지를 TCP/IP 형식의 메시지 형태로 만들어 send_proxy에게 전달한다. send_proxy는 송신 요구를 받아 원격지의 rcv_proxy에게 전달한다. rcv_proxy는 원격의 send_proxy로부터 수신한 메시지를 분석하여 (그림 10)과 같은 알고리즘을 이용하여 해당 능동객체(또는 스레드)에게 메시지를 전달한다. (그림 10)은 메시지를 수신하는 알고리즘으로 ART에서 제공하는 actobj_rcv() 함수에 의해 사용된다. ART는 자신이 관리하는 m_box내에 수신자에게 도착한 메시지가 존재하면, 메시지를 수신자의 m_buf에 복사한다(그림 10의 라인 1, 2, 3). 도착된 메시지가 없으면, 수신자를 블록상태로 만든다(그림 10의 라인 5).

```
1 :  Marshall the method call's arguments to
                                the arguments part in a message
2 :  if( local methods call )
3 :      if( exists the receiver )
4 :              copy the message to the receiver's m_ buf ;
5 :              wakeup the receiver ;
6 :      else
7 :              store the message to the          ART's    m_box;
8 :      endif
9 :  endif
10 : if( remote method call )
11 :     marshall    the information and arguments part to a message
                                                  on the TCP/IP message format ;
12 :     send the message to send_proxy ;
13 :  endif
14 : if( exist the return value )
15 :     block the sender until the return value receives ;
16 :     transfer the return value to the sender's local address space ;
17 : endif
```

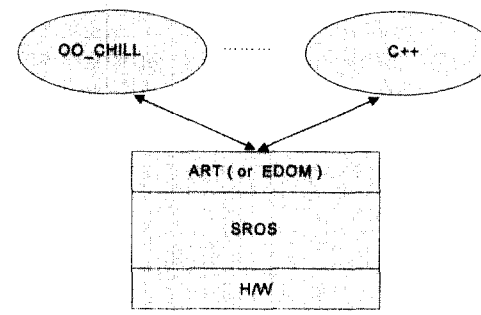
(그림 9) 송신자 알고리즘

```
1 :  if( exists the message for the receiver in m_box ) then
2 :      copy the message from m_box to the receiver's m_buf ;
3 :    return ;
4 :  else
5 :    block until the message will be arrived ;
6 :  endif
```

(그림 10) 수신자 알고리즘

4.6 ART의 구현

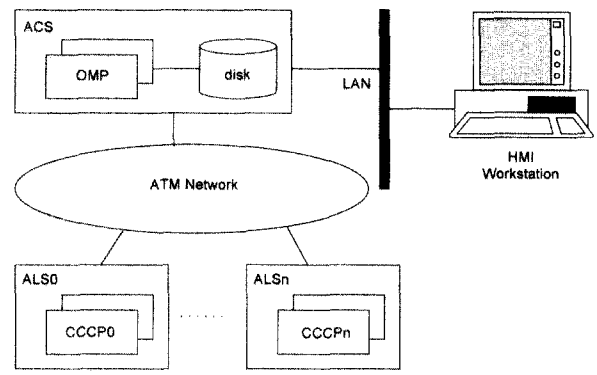
최근 정보통신 선진국을 중심으로 급격히 확산되고 있는 PDA, HDTV, 스마트폰 및 원격검침 장비와 같은 제한적인 컴퓨터 자원을 가지면서도 매우 섬세하고 고품질의 서비스를 요구하는 임베디드 시스템에 관한 요구가 급증하고 있다. 이와 같은 필요성에 의해 본 논문에서는 B-ISDN의 핵심이 될 ATM 교환기인 HANbit ACE 시스템의 운영체계로 개발된 SROS (Scalable Real-time Operating System)에 ART를 적용하여 (그림 11)과 같이 OO-CHILL과 C++로 작성된 사용자 프로그램들이 능동객체를 사용하여 실행하도록 구현하였다. (그림 11)의 EDOM(Etri Distributed Object Manager)은 ART를 SROS에서 사용하기 위하여 작명된 별칭이다.



(그림 11) HANbit ACE의 S/W 구조

ATM 교환 시스템은 분산 실시간 시스템으로 시간의 제약을 갖는 실시간 처리 및 고신뢰의 다양한 실시간 서비스를 요구한다. 이를 위하여 시스템 하드웨어의 제어 및 소프트웨어의 운용 환경을 제공하는 SROS는 마이크로 커널 기반의 운영체계로 시스템의 기본 제어 기능을 하는 커널과, 시스템 내 응용에 따라 다양한 서비스를 제공하는 시스템 서버들로 구성되어 있다. 이러한 시스템 서버들은 ATM 네트워크를 통한 통신 기능, 이더넷을 통한 통신 기능, 입출력 기능, 보조 기억장치 제어, 고장 감내를 위한 이중화 기능 등을 제공한다. 마이크로 커널 상에 이러한 다양한 기능들을 스케일러블하게 재구성할 수 있게 함으로써 시스템의 변화 및 새로운 요구 조건에 빠르게 적응할 수 있는 구조를 가진다. 또한 교환기용 소프트웨어 개발 언어로 사용되고 있는 OO-CHILL 언어를 지원한다.

ATM 교환 시스템은 분산 실시간 시스템으로 다수 개의 서브 시스템들이 분산 망에 연결되어 서로 통신을 하면서 정보를 전달하여 교환기의 기능을 수행하도록 구성되어 있다. 각 서브 시스템에는 SROS가 적용되어 실행되는데 구조는 다음 (그림 12)와 같다.



(그림 12) HANbit ACE 교환 시스템의 구조

ACS(ATM Central Switching Subsystem)에는 교환기 시스템 전체의 운용과 유지 보수를 위한 기능을 수행하기 위하여 OMP(Operation and Maintenance Processor)와 디스크가 실장되어 있다. ALS(ATM Local Switching Subsystem)에는 호처리 기능을 수행하기 위하여 CCCP(Call

and Connection Control Processor)가 실장되어 있다. 그리고 각 프로세서는 이중화되어 있어 한 프로세서가 다운되더라도 대기중인 다른 프로세서로 실행 제어를 옮겨서 서비스의 연속성이 이루어지도록 구성되어 있다. 또한 OMP는 운용자 터미널(HMI Workstation)에 LAN으로 연결되어 고속의 통신으로 운용자의 명령어 처리 및 교환 시스템 내의 정보를 웍스테이션의 터미널로 출력해주거나 과금 등의 정보를 저장할 수 있는 경로를 제공한다.

ART(또는 EDOM)을 HANbit ACE 시스템에 적용하여 실행한 결과 다음과 같은 장점들이 발견되었다. 첫 번째, 능동객체를 사용함으로써 병행성을 높일 수 있었다. 기존의 응용 프로그램에서는 스레드 또는 태스크 단위의 병행성 만을 구현하였으나, 객체 수준에서의 병행성을 표현함으로써 더욱 세부적인 단위의 병행성을 구현할 수 있었다. 두 번째, 사용자들은 복잡한 분산 프로그래밍을 쉽게 작성하고 실행할 수 있다. 전자교환기와 같은 분산 시스템에서 사용자들은 시스템 구조와 TCP/IP 등의 네트워크 프로그램에 관한 사전 지식이 없이 호스트이름과 능동객체의 이름만으로 분산 프로그램을 할 수 있었다. 세 번째, 사용자들은 ART(또는 EDOM)내의 구성요소들을 전혀 몰라도 능동객체를 사용할 수 있다. ART에서 제공하는 능동객체 래퍼와 능동객체 전처리기를 이용하여 ART내의 프리미티브들의 인터페이스에 무관하게 쉽게 능동객체를 사용할 수 있었다. 또한, ART에서 제공하는 도구를 이용하여 기존의 일반객체를 사용하는 것과 동일한 방법으로 능동객체를 사용할 수 있었다. 네 번째, C와 C++과 같은 기존 언어에서 다른 실행 파일에 존재하는 스레드간의 통신에 적용할 수 있었다.

7. 결 론

본 논문에서는 기존의 객체지향언어에 새로운 구문을 추가하여 능동객체를 정의하고 사용할 수 있는 능동객체실행 시스템(ART)을 설계하고 구현하였다. 현재까지 제시되었던 방식과는 다르게 제안하는 ART는 분산환경에서 사용자에게 위치 투명성이 보장되고, 편리한 분산 프로그램이 가능하도록 설계되었다. 또한, 분산환경에서 다른 프로세스에 존재하는 스레드 사이의 완벽한 통신이 제공되고, 메소드 호출도 복귀 값의 존재에 따라서 동기적 또는 비동기적인 방식의 메소드 호출을 지원한다. 그리고 ART는 동적(dynamic) 라이브러리 형식으로 구성되어 실행파일의 크기가 작기 때문에 시스템 자원이 제한되어 있고, 소규모 분산 환경으로 구성되어 있는 임베디드 시스템에 효율적으로 적용될 수 있다.

또한, 기존의 응용 프로그램에서 스레드 또는 태스크 단위의 병행성 만을 구현하였으나 ART를 이용하여 능동객체를 사용함으로써 병행 수행 능력을 높일 수 있었다. 그리고 사용자 프로그램에서 편리하고, 사용하기 쉬운 방법으로 병행성을 표현하기 위하여 병행객체래퍼와 병행객체 전처리기 등의 도구를 제공함으로써 사용자에게 친숙한 인터페이스를 제공하였다. 또한, 본 논문에서 제안하는 능동객체와 스레드의 통신 모델은 능동객체의 메소드 호출뿐만 아니고, 스레드 사이의 통신에도 적용될 수 있다. 기존의 UNIX 스레드 사이의 통신은 같은 실행파일 내에서만 제한적으로 제공되었지만, 본 논문에서는 분산 환경에서 다른 실행 파일에 존재하는 스레드 사이의 통신도 완벽하게 지원한다. 그리고 본 논문에서는 ART를 구현하여 임베디드 시스템의 한 종류인 HANbit ACE 전자 교환 시스템에 적용하여 실행하였다.

현재 ART를 최적화하여 좀 더 작은 규모의 임베디드 시스템에 적용시키기 위한 연구가 진행되고 있다. 또한, ART를 임베디드 시스템의 성능 평가 및 제어 등의 임베디드 시스템을 위한 모니터링 시스템 구현에 응용하기 위한 연구를 진행하고 있다. 그리고 기존의 객체지향언어와 관련된 컴파일러를 연구하여 본 논문에서 제안하는 능동객체를 표현하고 사용하기 위해 기능을 직접 컴파일러에 추가하는 연구 과제는 향후 연구 분야가 될 수 있다.

참 고 문 헌

[1] Bangquing Li and Baowen Xu etc., "ConC++ : A Concurrent C++," Computer Software and Applications Conference, 2000, COMPSAC 2000, The 24th Annual International, pp.223-227, 2000.

[2] Petitpierre, C., "Synchronous C++ : A Language for Interactive Applications," IEEE Computer, Vol.31, Issue 9, pp.65-72, Sept., 1998.

[3] G. Agha, P. Wegner and A. Yonezawa, "Research Directions in Concurrnt Programming," MIT Press, Cambridge, Mass., 1993.

[4] D. Caromel, "Toward a Method of OO Coucurrent Programming," Comm. ACM, pp.90-116, Sept., 1993.

[5] H. C. Wang, C. K. Yuen and M. D. Feng, "Balinda C++ : Run-Time Support for Concurrent Object-Oriented Language," Parallel Architectures, Algorithms, and Networks, 1999, (I-SPAN '99) Proceedings. Fourth International Symposium on, pp.36-41, 1999.

[6] N. Carriero and D. Gelernter, "Linda in context." Communication of the ACM, 32(4), pp.444-458, 1989.

[7] M. D. Feng, Y. Q. Gao and C.K. Yuen, "Implementation Linda Tuplespace on Distributed System," International Journal of High Speed Computing, 7(1), pp.125-144, 1995.

[8] Yonezawa, A., et al., "Object Oriented Concurrent Programming," The MIT press, 1987.

[9] Chandy, M., et al., "CC++ : A Declarative Concurrent Object_Oriented Programming Notation," The MIT Press, Agha G., et al. edit, pp.281-313, 1993.

[10] Wilson, G. and Lu, P., "Parallel Programming Using C++," The MIT Press, 1996.

박 윤 용

e-mail : yypark@sunmoon.ac.kr

1983년 숭전대학교 계산통계학과(학사)

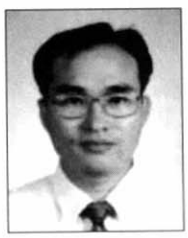
1985년 서울대학교 대학원 계산통계학과 (이학석사)

1994년 서울대학교 대학원 계산통계학과 (이학박사)

1985년~1992년 한국전자통신연구소 연구원

1992년~현재 선문대학교 컴퓨터정보학부 부교수

관심분야 : 분산처리운영체제, 분산객체지향처리시스템, 분산객체표준화



임 동 선

e-mail : dslim@etri.re.kr

1986년 숭실대학교 전자계산학과(공학사)

1996년 KAIST 정보및통신공학과(공학석사)

1986년~현재 한국전자통신연구원 네트워크 연구소 책임연구원

관심분야 : 실시간 시스템, 통신망(분배망), 멀티미디어 서비스

정 부 금

e-mail : bgjung@etri.re.kr

1986년 부산대학교 계산통계학과(학사),

1991년 숙명여대 전산학과(석사),

1986년~현재 한국전자통신연구원 네트워크 플랫폼팀 책임연구원

관심분야 : 실시간 운영체제, 라우터 운영체제, 리눅스, 이중화 기술

이 경 오

e-mail : leeko@rainbow.sunmoon.ac.kr

1989년 서울대학교 계산통계학과 학사

1994년 서울대학교 전산과학과 석사

1999년 서울대학교 전산과학과 박사

1999년~현재 선문대학교 컴퓨터정보학부 조교수

관심분야 : 데이터베이스, 멀티미디어, 모바일 컴퓨팅

박 정 호

e-mail : jhpark@omega.sunmon.ac.kr

1980년 성균관대학교 사범대학(문학사)

1980년~1982년 성균관대학교 경영대학원 정보처리학과(경영학석사)

1985년~1987년 日本 오사까대학교 대학원 정보공학전공(공학석사)

1987년~1990년 日本 오사까대학교 대학원 정보공학전공(공학박사)

1996년~현재 한국정보처리학회 사업이사

1991년~현재 선문대학교 컴퓨터정보학부 교수

관심분야 : 분산알고리즘, 원격교육, XML, 소프트웨어공학