論文 95-12-12-6

C++의 병행성을 위한 클래스 라이브러리 구현

正會員 李 俊*, 金 成 槿**, 劉 載 祐***, 宋 後 鳳***

Class Library Implementation for Concurrency of C++

Joon Lee*, Sung Keun Kim**, Chae Woo Yoo***, Hoo Bong Song*** Regular Members

要 約

가상적으로 어떠한 실세계의 객체들은 객체지향 프로그래밍 시스템 안에서 모델화되어질 수 있으며 실세계의 객체는 병행적으로 존재하며 또한 활동하고 있다. 그러나 대표적인 객체지향 언어 중의 하나인 C++ 언어는 순차적인 객체 실행 모델만을 제시하고 있다. 본 논문에서는 C++ 언어에 병행 객체를 제공하기 위해 병행 클래스 및 이의 제반 라이브러리를 제공하였으며, 이를 위해 능동 객체 및 원격 메소드 호출(remote method invocation) 방법 등을 선택하였다. 이는 C++ 언어가 가지는 기존의 순차 실행 모델을 변경하지 않고 병행 클래스 라이브러리를 제공함으로써, 병행 객체지향 프로그래밍 실행 환경을 구현할 수 있음을 제안하고자 한다. 이러한 실행 환경은 소프트웨어의 재사용 및 호환성에 기초하여 기존의 순차적 객체지향 언어인 C++의 라이브러리를 확장함으로써 경제적인 병행성 구현 방법이라고 할 수 있다.

ABSTRACT

Virtually any real-world object can be modeled in an OOP system. Real-world objects exist and are active concurrently. But C++, which is a popular object-oriented language, provides only sequential object execution model. This paper provide concurrent classes for concurrent objects in C++, and their associate libraries. And, we choose active objects and remote method invocation for that. So we can implement concurrent object-oriented programming environment without altering the C++'s sequential execution model by providing concurrent class libraries. This execution-environment is economic concurrency implementation method, because it extends C++ library based on reusability and portability.

*조선대학교 컴퓨터공학과
**숭실대학교 대학원 전자계산학과
***숭실대학교 컴퓨터학부
論文番號 : 95227-0629
接受日字 : 1995年 6月 29日

Ⅰ. 서 론

모든 분야에서 컴퓨터가 이용되어지면서 프로그램의 영역이 점점 다양하고 복잡하게 되었다. 이러한 복잡한 영역에서 사용되는 프로그램은 계산 효율의 증대 및 여러 사건을 빠르고 동시에 처리해야 할 필요성이 증대하였고 이러한 요구를 만족하기 위해 병행성을 가지는 언어의 요구가 증대하였다. 이와 같은 연구로 실세계를 더 정확하고 효과적으로 표현하기 위해 인간의 사고 체계와 유사한 객체지향 모델을 도입하려는 경향이 나타났으며, 이러한 요구를 만족하는 병행성을 가지는 객체지향 언어의 필요성이 증대되었다[1].

가상적으로 어떠한 실세계의 객체들은 객체지향 프로그래밍 시스템 안에서 모델화되어질 수 있으며 실세계의 객체는 병행적으로 존재하며 또한 활동하고 있다. 그러므로, 객체지향 프로그래밍은 역시 병행 시스템의 설계 및 구현하는데 유용하다고 할 수 있다.

이러한 객체지향 및 병행성을 지원하기 위한 프로그래밍 언어들은 여러 가지 다른 시각에서 접근되었는데 병행 객체지향 언어를 새로 설계하는 방법과 기존의 순차적 언어에 병행성을 도입하는 방식으로 크게 나누어질 수 있다.

표 1. 병행 객체지향 언어의 특성
Table 1. Characteristics of Concurrent Object-Oriented Language

	atomicity	active object	implementation strategy
ABCL/1	Required for serialized objects	Yes	New language
CEffel	Not required	No	Extensions
Effel	Yes	No	Extensions
Effel//	Yes	Yes	Extensions+library
Emerald	No	No	New language
Maude	Yes	No	New language - Integration with rewriting logic
POOL/T	Yes	Yes	New language
Sina	Not requried	No	New language - Integration with composition filters

객체지향 언어 가운데 많이 사용되는 언어인 C++는 순차 실행 모델을 기반으로 하는 언어이다. 본 논문은 C++언어에 병행성을 도입하는 모델을 제시한다.

이러한 방식은 새로운 병행성 지원 언어를 설계하는데 드는 비용이나 사용자가 적응하는데 필요한 노력 등을 감안한다면 가장 접근하기 쉬운 방법을 제공하게 된다.

Ⅱ. 병행 객체지향 언어 구현의 접근 방법

프로그래밍 언어에 병행성을 구현하는 방법은 일반적으로 세 가지 접근 방법을 사용한다. 이러한 접근 방법은 우선 새로운 병행성 언어를 설계하는 방법과 존재하는 프로그래밍 언어를 확장하는 방법 그리고 라이브러리를 설계하는 방법으로 구분할 수 있다. 초기에 병행성을 지원하는 시스템은 대부분 새로운 언어를 설계하는 방식을 선택하였으며 이러한 예로는 Hybrid[2], Pool[3], SR[4] 그리고 ABCL/1[5] 등을 들 수 있다. 이러한 언어들은 강력한 병행성 및 프로그래밍의 일반적인 기능을 제공하고 있다.

두 번째 접근 방법은 기존의 언어를 확장하는 것으로 다음의 세 가지 기술을 통해 병행성을 지원하도록 구현되었다.

(1) 언어에 문법이나 의미(semantic)를 수정하거나 확장하기 위해 특별한 키워드(keyword) 나 프리프로세스(preprocess) 기술을 사용한 것으로 μC++[6] 나 CEiffel[7] 등을 들 수 있다.

(2) 특별한 병행성 클래스를 인식할 수 있도록 컴파일러를 수정하는 방법으로 Eiffel//[8], PRESTO[9] 등이다.

(3) ACTOR 모델과 같은 병행성 파라다임을 제공하기 위해 문법이나 의미를 확장하는 방식으로 ACT++[10], Actalk[11] 등이 대표적이다.

위에서 언급된 접근 방법의 구분은 시대적인 흐름에 따라 진행되어 왔다고 볼 수 있으며, 가장 오래된 접근 방법은 병행 객체지향 언어를 설계하는 방법이다. 객체지향 패러다임이 점차 성숙되고 순차적 객체지향 언어가 보편화되면서 병행 객체지향 언어로 확장하는 두 번째 접근 방법이 점차 사용되었다.

그리고 현재는 대부분의 접근 방법이 객체지향 소프트웨어 공학의 기본적 요구 사항인 소프트웨어의 재사용 및 호환성을 효과적으로 지원할 수 있는 라이브러리를

이용한 접근 방법이 보편화되고 있다. 이것을 특성에 의하여 구분하면 아래와 같은 표 1과 같이 구분할 수 있다[12].

본 논문에서 병행성을 구현하기 위해 사용된 메커니즘은 기존의 객체지향 언어에 외부 라이브러리를 이용하여 병행성을 제공하는 세 번째 방식이다.

이러한 라이브러리에 기초한 해결 방법은 프로세스의 클래스 정의를 통하여 병행성을 도입하려는 접근 방식으로 Choice 객체지향 운영체제[14]가 대표적인데 기존 소프트웨어의 사용자를 확보하면서 병행성을 제공해 줄 수 있다는 장점이 있다.

소프트웨어의 재사용 및 호환성에 기초하여 기존의 순차적 객체지향 언어인 C++에 병행성 지원을 위해 라이브러리를 확장함으로써 경제적인 병행성 구현 방법을 제시하였다.

Ⅲ. 설계 및 구현

본 논문의 가장 큰 목적은 순차적 객체지향 언어의 본질을 파괴하지 않으면서 병행 객체를 제공하는데 있다. 이를 위해 아래와 같은 요구 사항을 정의하고 병행 객체의 특성 및 구현 사항을 제시하였다.

1. 설계 단계의 요구 사항

클래스 라이브러리를 이용하여 병행성을 구현하기 위해서는 다음과 같은 요구 사항을 만족하여야 한다.[6,14,15]

(1) 객체간의 통신에 대한 정적 타입 검사(static type checking)가 가능해야 한다. 이는 정적 타입 검사가 오류 검출 및 효과적인 코드 생성에 영향을 미치기 때문이다.

(2) 객체간의 상호작용이 가능하여야 한다. 객체지향 언어는 객체가 어떤 작업을 하기 위해서는 요구(request)를 보내야 하는데 이 요구로 인해 연산을 호출하게 되고 이를 통해 적절한 메소드를 실행하게 된다. 송신된 메시지는 주어진 객체에서 지정된 연산을 수행하고 결과를 되돌려 받을 수 있기 때문이다.

(3) 객체간의 통신에 있어 메시지 전달은 동기적 전송 혹은 비동기적 전송이 적용될 수 있는데, 비동기적 전송을 제공하기 위해서는 능동 객체의 버퍼링(buffering) 기능이 필요하다.

(4) 능동 객체도 다른 객체와 같이 영역 규칙(scope rule)과 생명 시간(life time)이 적용되어야 한다. 어떤 객체가 생명 시간이 끝났는데도 그 객체가 살아 있다면 이것은 언어가 가지는 생명 시간 특성에 위배되며 사용자 혹은 시스템에 오류를 유발할 수 있기 때문이다.

(5) 본 논문은 순차적 C++ 언어가 가지는 상속성의 속성을 기본으로 하여 클래스의 라이브러리를 설계하였으며 캡슐화의 속성을 가지는 클래스를 정의하였다.

(6) 새로운 병행 객체 모델은 기존 순차 클래스의 재사용이 가능하여야 한다. 이것은 순차 C++ 언어가 가지는 여러 클래스 라이브러리의 사용에 지장이 없도록 하기 위해서이며, 재사용이라는 소프트웨어 공학적 측면을 고려하였다.

위와 같은 조건을 만족하는 병행 객체지향 언어의 설계 및 구현을 위해 다음과 같은 능동 객체 및 원격 메소드 호출(remote method invocation), 스케쥴링 방법을 설계하고 구현하였다.

2. 능동 객체(active object)

객체는 능동 객체(active object)와 수동 객체(passive object)로 구분된다. 여기서 능동 객체는 독립된 객체를 말하며 다른 객체에 의해 조작되어지지 않는 객체를 의미한다. 또한 능동 객체는 이를 위해 하나의 프로세스(process)를 갖으며 이러한 개념은 ACTOR 모델 등에서 소개되고 있다. 수동 객체는 어떤 객체의 요청 메시지를 전달받았을 때만이 활성화되는 객체를 말하며, 순차적인 객체지향 언어는 수동 객체만을 가지고 있다.

능동 객체의 생성을 위하여 병행 클래스의 상속에 의해 이루어지도록 설계하였다. 병행성의 구현은 이러한 능동 객체들의 생성 및 능동 객체와 수동 객체들간의 상호작용에 의해 이루어진다. 즉, 다수의 능동 객체는 능동/수동 객체에서 보내 온 메시지를 가지고 다수의 수동 객체를 조작할 수 있으며 이 결과를 능동/수동 객체에게 메시지를 통하여 보내게 된다. 여기서 능동 객체는 요청 메시지를 받았을 때만이 활성화되는 것은 아니고 독립적으로 활동하고 있다. 하나의 응용 프로그램에서 여러 개의 능동 객체가 동작하는 병행 프로그램을 작성하기 위

해서는 병행성을 다루는 어떤 특별한 기능을 가지는 프로그래밍 모델이 필요하다.

본 논문에서는 병행성을 관리하는 기본적인 병행 클래스인 CONCURRENCY를 제공하며, 이 클래스의 인스탄스를 통해 새로운 능동 객체를 생성하는 것이 가능하게 된다.

CONCURRENCY 클래스로부터 상속받은 새로운 클래스는 능동 객체를 생성하는 메소드를 포함하게 된다. 또한 이 클래스는 능동 객체가 다수의 메소드를 실행할 수 있도록 정의하고 있다.

능동 객체로 만들기 위해서는 그림 1과 같이 병행 클래스로부터 상속받아야 한다. 그러나 병행 클래스로부터 상속받는다고 해서 능동 객체가 바로 생성되어지는 것은 아니고, 그림 2와 같이 능동 객체가 가지는 메소드 함수인 make_runnable의 호출이 필요하다. 이 방법은 순차적 프로그래밍 객체를 언어의 속성을 변화시키지 않고 능동 객체를 구현할 수 있도록 고안되었다.

```
class ActiveType : public CONCURRENCY
{
        int i;
    public :
        ActiveType(int);
        void execution(int);
        void scheduler();
}
```

그림 1. 병행 클래스에서 순차 클래스로의 상속
Fig. 1. Inheritance of Sequential Class from Concurrent Class

```
main()
{
    int claim_num;
    ActiveType activeObj;
    activeObj.make_runnable();
    claim_num = activeObj.remote_invoke(
          "execution", "#t #v ", int, 34);
                    :
    activeObj.claim_result(claim_num);
}
```

그림 2. make_runnable의 실행예
Fig. 2. Execution example of make_runnable.

```
void make_runnable()
{
  클라이언트의 IPC 파라메타를 초기화한다.
  통신 포트의 생성 및 초기화
  새로운 프로세스의 생성
  부모 프로세스에서
     결과를 받을 버퍼를 생성한다.
     서버로 부터의 acknowledge을 기다린다.
     서버의 정보를 기록한다.
     return;
  자식 프로세스에서
     서버의 IPC 파라메타를 초기화한다.
     클라이언트에게 acknowledge을 보낸다.
     서버의 객체를 생성한다.
     서버의 scheduler()을 동작한다.
     // scheduler는 무한 루프를 돌고 있다.
}
```

그림 3. make_runnable의 구현 알고리즘
Fig. 3. Implementation algorithm of make_runnable

make_runnable 의 의사 알고리즘(pseudo-algorithm)은 그림 3과 같다. make_runnable 함수는 호출 객체의 통신 포트의 생성 및 초기화를 수행하며 IPC(Inter-Process Communication)을 위한 여러 작업을 하게 된다.

Unix 시스템에서의 시스템 호출 fork() 함수를 호출함으로써 새로운 프로세스를 생성하게 된다. 새로이 생성되어지는 프로세스를 위해 호출 객체의 IPC 정보를 전달하게 된다.

호출 객체(부모 프로세스) 안에서는 서버 객체로부터의 실행 결과를 받을 수 있는 결과 큐(result queue)의 생성 즉, 버퍼(buffer)를 생성하고, 부모 프로세스는 자식 프로세스(서버 객체)로 부터의 인식(acknowledge)을 기다리게 된다. 이러한 인식 메시지를 사용하여 서버의 정보를 기록, 저장하게 된다. 이렇게 통신 포트의 생성 및 초기화를 끝으로 호출 객체는 make_runnable 함수를 종료하게 된다.

자식 프로세스는 IPC 파라메타를 초기화하며 클라이언트의 인식 메시지를 보낸다. 인식 메시지를 받으면 클라이언트간의 통신이 연결되고 서버 객체를 생성하고 서버 객체는 스케쥴러(scheduler)라는 멤버 함수를 수행하게 된다. 스케쥴러는 무한 루프를 돌면서 클라이언트

객체의 요구를 처리하고 결과를 클라이언트에게 보내게 된다.

대리 객체(proxy object)는 지역(local)에 상주하고 있는 객체이며, 이 객체는 원격 객체(remote object)로 표현된 객체이다.

위의 make_runnable에 의해 능동 객체가 생성되며 하나의 능동 객체를 생성하기 위해서 대리 객체에 make_runnable 메소드를 호출하여야 한다. 이것은 하나의 대리 객체가 하나의 능동 객체를 생성할 수 있다는 것을 의미한다.

여러개의 같은 클래스 타입의 인스탄스 객체가 생성되고 그 인스탄스 객체 중에 하나가 make _runnable을 호출하여 하나의 능동 객체를 생성하였다고 가정하면, 각각의 다른 동일한 클래스 타입의 객체가 능동 객체를 생성하기 위해서 make_runnable을 호출하여야 하는데 이것은 프로세스가 차지하는 메모리의 낭비를 가져온다. 이러한 경우, 하나의 능동 객체를 공유하는 새로운 메소드를 가지고 능동 객체에게 연산을 할당하는 방법이 필요할 것이다. 이를 위하여 attach 메소드는 하나의 능동 객체를 공유하여 시스템의 자원 낭비를 최소화하는 방법을 위하여 고안하였다. 그러나, attach 메소드는 동일한 타입을 가지는 많은 인스턴트가 하나의 능동 객체를 사용하게 될 때는 병행성의 효율을 감소시킬 수 있는 단점이 존재한다. 그림 4는 attach의 의사 알고리즘을 보여주고 있다.

```
void attach(server_info)
{
    클라이언트의 IPC 파라메타를 초기화한다.
    통신 포트의 생성 및 초기화
// 새로운 프로세스의 생성을 생성하지 않는다.
    결과를 받을 버퍼를 생성한다.
    서버의 정보를 기록한다.
    return;
}
```

그림 4. attach의 의사 알고리즘
Fig. 4. Pseudo Algorithm of attach

attach 메소드는 같은 타입의 다른 인스탄스 객체에 의해 만들어진 능동 객체의 정보를 알 필요가 있다. 그것을 위하여 attach의 파라메타로 해당 능동 객체의 정보를 알려주어야 한다.

```
main( )
{
  int claim_num1, claim_num2;
  int result;
  ActiveType activeObj1, activeObj2;

  activeObj.make_runnable();
  claim_num1=activeObj1.remote_invoke(
        "execution", " #t #v ", int, 34);
                :
  activeObj2.attach(activeObj1::pid);
  claim_num2=activeObj2.remote_invoke(
        "execution", "#t #v", float, 3.4);
                :
  result=activeObj2.claim_result(claim_num2);
                :
}
```

그림 5. attach 메소드의 사용 예
Fig. 5. Execution example of attach

또한, attach 메소드는 통신 포트의 생성 및 초기화 작업을 하게 된다. 그러나, make_runnable에서 통신 포트의 생성 및 초기화 후에 프로세스를 생성하는 것과는 달리 여기서는 프로세스를 생성하지 않는다. 그리고, attach 메소드는 결과 큐를 생성하고 서버의 정보를 저장한다. 그리고, attach 메소드를 종료하게 된다. 다음의 그림 5는 attach의 사용 예이다.

3. 원격 메소드 호출

원격 객체(remote object) 위에서의 연산의 구현의 기본적 생각은 Amber[12]로 부터 시작되었다.

이것은 지역 객체가 원격 객체에게 메소드를 실행시키게 하고 자신은 자기의 작업을 비동기적으로 수행함으로써 실행 효율을 증대시키며 이 원격 객체에 의해 병행성이 달성된다.

그림 6은 원격 메소드 호출의 과정을 보인 것으로 다음과 같은 단계로 진행된다.

첫째, 병행 클래스가 가지는 메소드인 remote_invoke는 캡슐화 되어 있으며 수행될 메소드의 이름 및 파라메타를 선달하는 과정을 수행하게 된다. remote_invoke 메소드는 수행 결과가 도착할 때까지 기다리지 않고 수행을 마치게 된다.

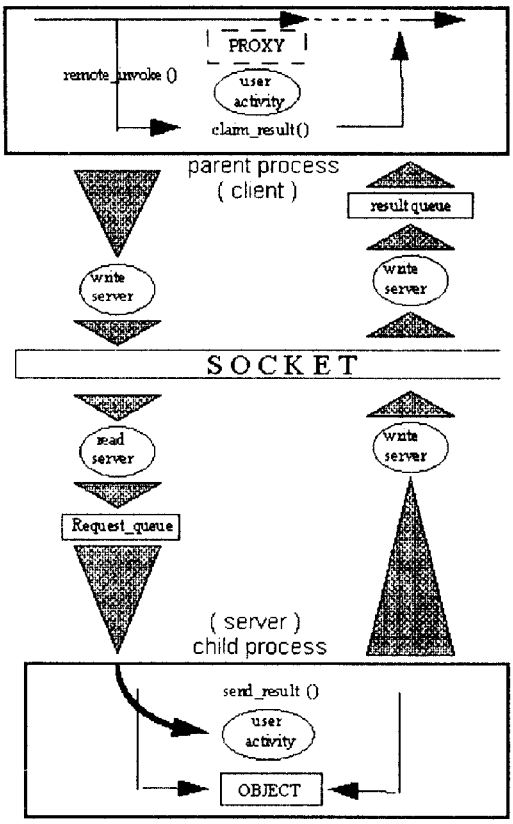


그림 6. 원격 메소드 호출의 흐름
Fig. 6. Flow of remote method invocation

둘째, remote_invoke에서 보내 온 메시지를 능동 객체는 요구 큐에서 읽고, 능동 객체 위에서 메시지를 처리하고 send_result 메소드를 통해서 그 결과를 보내게 된다.

셋째, 능동 객체로부터 보내 온 결과는 결과 큐에 저장되고 결과를 가지고 비동기적으로 진행되는 있는 대리 객체는 결과를 얻어서 계속해서 자신의 작업을 수행하게 된다.

이러한 원격 메소드 호출 과정은 remote_ invoke를 통해서 이루어지는 데 remote_invoke는 그림 7에 원시(source) 코드로 보여진다.

그림 7에서 remote_invoke는 파라메타로 method, type 그리고 파라메타의 값을 얻어서 Request_package라는 데이터 구조를 생성하고 그 데이터 구조에 파라메타로 받은 값을 저장하고 또한

```
int remote_invoke( char *method,
                                char *fmt, ... )
{
   va_list ap;
   char *sval;

   va_start(ap, fmt);
   for( p = fmt; *p; p++) {
       if(*p != '#') {
           continue;
       }
       switch(*++p) {
       case 't' :
           for( sval = va_arg(ap, char*); *sval;
                sval++)
                   :
           Request_package =
                create_request_package();
           claim_num = unique_claim_no();
           set_method_field(Request_package,
                method, method_parameter);
                   :
       case 'v':
                   :
           set_IPC_field(Request_package,
           claim_num, IPC_info);
                   :
   }
   va_end(ap);
   send_package( Request_package );
   return ( claim_num );
}
```

그림 7. remote_invoke 메소드의 원시 코드
Fig. 7. Source code of remote_invoke method

```
Boolean result_ready( int claim_number )
{
   result_queue를 검색한다.
   if(claim_number와 연관된 결과가 도착하였다)
       return TRUE;
   else
       return FALSE;
}
```

그림 8. result_ready의 의사 코드
Fig. 8. Pseudo code of result_ready

claim_num을 저장하여 send_package라는 모듈을 통해서 능동 객체에게 전달하게 된다.

remote_invoke는 독특한 인식자(claim_num)를 대리 객체에게 전달하여 능동 객체의 결과를 이용할 수 있게 된다.

```
Boolean claim_result( int claim_number )
{
  while( TRUE ) {
     if ( claim_number와 연관된 결과가
          도착하였다 )
               return TRUE;
  }
}
```

그림 9. claim_result의 의사 코드
Fig. 9. Pseudo code of claim_result

4. 동기화

순차적인 객체지향 패러다임에서 메소드 호출은 동기적(synchronous)이다. 그리고 객체는 메소드가 호출될 때 활성화되는 수동 객체를 의미한다.

우리는 하나의 객체가 다른 객체에 통신 채널을 통하여 메세지 전달이 완료되었을 때 두 객체는 동기화되었다고 한다.[15] 순차적 C++ 언어에서는 단순한 메소드 호출로 달성되어질 수 있는 반면에, 다중 프로세스를 가지는 병행 객체지향 언어로서의 C++에서는 객체간의 동기화를 위한 고려가 필요하다. 병행적인 객체지향 패러다임에서 병행 객체의 통신 동기화는 병행성을 구현하는데 크게 문제되는 점 중의 하나이다.

동기화를 위한 메시지 전달 방식은 비동기식 메시지 전달(nonblocking send, blocking receive)과 동기식 메시지 전달(blocking send, blocking receive), 비중단 원격 프로시저 호출(nonblocking RPC), 미래 원격 프로시저 호출(future RPC)로 구분할 수 있다. 비동기식 메시지 전달은 송신 명령을 수행하는 객체가 송신 메시지의 도착 여부와 관계없이 작업을 계속 수행하며, 수신 명령을 수행하는 객체는 해당 메시지가 수신될 때까지 수행을 중단하는 방식이다. 동기식 메시지 전달은 송신 명령을 수행하는 객체가 수신 객체에서 메시지를 수신할 때까지 수행을 중단하는 방법이다. 비중단 원격 프로시저 호출은 송신 객체가 수신 객체로부터 응답이 올 때까지 수행을 중단하는 방법이다. 미래 원격 프로시저 호출은 송신 객체가 자신의 응답이 필요할 때까지 작업을 계속할 수 있으며, 어느 순간 응답이 필요하게 될 경우 비중단 원격 프로시저 호출과 같이 행동하게 된다.

본 논문은 이러한 동기화를 위한 방법들은 기존의 C++ 언어가 가지는 동기식 메시지 전달 방법과 remote_invoke 메소드의 특성인 비동기식 메시지 전달 방식을 이용한다.

remote_invoke에서 보내 온 메시지를 능동 객체는 메시지를 처리하고, 그 결과를 대리 객체에게 보낸다. 대리 객체는 결과를 받아들이는 것은 비중단형 원격 프로시저 호출, 미래 원격 프로시저 호출 방법을 가지고 있다.

능동 객체에 바탕을 두는 통신은 메시지 전송을 통해서만 이루어질 수 있고, 어떤 객체가 다른 객체의 데이터를 직접 참조/수정하는 것이 불가능하다. 또한 어떤 객체에 대하여 능동 객체 내에서 어떤 순간에 수행되는 멤버 함수는 최대 하나밖에 있을 수 없기 때문에 객체 안의 데이터에 대한 동기화는 암시적으로 이루어진다. 이런 능동 객체들 사이의 동기화는 선택적 메시지 수신(selective message receiving)을 할 수 있는 스케쥴러에 의해 이루어질 수 있다. 즉, 어떤 객체가 어떤 순간에 자신이 수신할 메시지의 종류를 지정할 수 있다면, 그 객체는 다른 객체가 그런 종류의 메시지를 전송할 때까지 대기 상태에 있다가 그 메시지를 수신한 후에 수행을 계속하여 능동 객체들 사이의 동기화가 이루어질 수 있다.

5. 스케쥴링(Scheduling)

본 논문은 병행성을 위하여 응답 스케쥴링(reply scheduling)과 요구 스케쥴링(request scheduling) 두 가지 방법으로 스케쥴링할 수 있도록 하였다. 응답 스케쥴링은 클라이언트의 요구에 대한 서버의 처리 결과를 가지고 스케쥴링하는 방식이다. 요구 스케쥴링은 클라이언트의 요구를 받아들이거나 처리하는 것을 능동 객체가 통제하도록 하는 스케쥴링 방식이다. 응답 스케쥴링은 remote_invoke, 결과가 도착할 때까지 대기하는 claim_result, 결과의 도착을 검사하는 result_ready, 결과 값을 얻어내는 get_result와 같은 메소드에 의해 수행된다. 요구 스케쥴링은 요구를 능동

객체에게 넘겨주는 get_request, 결과를 대리객체에게 보내는 send_result, scheduler로 스케쥴링할 수 있으며, 표 2와 같다.

표 2. 스케쥴링 전략
Table 2. scheduling strategy

스케쥴링 방법	통제 객체	사용 메소드
응답 스케쥴링	지역 객체 (클라이언트 객체)	remote_invoke, claim_result, result_ready, get_result
요구 스케쥴링	능동 객체 (서버 객체)	get_request, send_result, scheduler

claim_result와 result_ready의 의사 코드를 그림 8과 그림 9에서 보여주고 있다.

```
  template<class T> void send_result( T, int
);
  /*
   * Specification part of CONCURRENCY
class
   */
      class CONCURRENCY {
      Boolean has_split;
      Boolean is_proxy;
      int server_sockfd, client_sockfd;
      int newsockfd;
      struct sockaddr_un cli_addr, serv_addr;
      pid_t pid;
      QUEUE request_queue, result_queue;
  public:
      virtual ~CONCURRENCY();
      virtual void scheduler();
      void make_runnable();
      int remote_invoke( String, String, ... );
      Boolean get_request();
      Boolean get_result();
      Boolean result_ready( int );
      Boolean claim_result( int );
  };
```

그림 10. CONCURRENCY 클래스의 구조
Fig. 10. Structure of CONCURRENCY Class

능동 객체가 생성되면 시동 메소드(start-up)로서 스케쥴러가 실행이 되어 능동 객체의 내부 상태 및 행동을 규정하게 된다.

클라이언트의 요구를 처리하기 위해 요구 큐(request queue)를 가지며, 큐안에 있는 각각의 엔트리는 클라이언트의 원격 호출 요구의 파라메타(parameter)인 claim_number, 응답 주소, 요구 이름(request _name) 등을 갖고 있다. 스케쥴러는 요구 큐(request queue)의 제한 없이 접근 가능하며 큐안에 있는 파라메타 및 요구 이름을 조사하여 선택하고 처리한다. 이러한 스케쥴러는 하나의 프로세스를 가지고 있으며 내부적으로는 순차적으로 메소드의 실행하는 이유는 클래스가 가지는 내부 데이터를 병행적으로 처리할 경우, 동기화를 위해 lock, monitor와 같은 상호배제 동기화를 이용한 일관성을 유지하기 위한 방법이 제시되어야 한다. 능동 객체의 메시지 처리를 내부적으로 순차적인 방법으로 실행시킴으로써 이러한 오류를 방지하며 프로그래머가 작성하여야 하는 스케쥴러를 간결하게 프로그래밍할 수 있도록 만들어 준다. 그러나 객체 내부의 병행성을 주지 않음으로써 진정한 객체지향 언어의 병행성을 충족시키지 못한다는 한계가 존재한다.

6. 구현 환경

객체지향 언어인 C++에 병행성을 제공하기 위해 여러 개의 프로세스를 필요로 한다. 이 시스템은 다중 프로세스 환경에서 UNIX을 기초로 하는 SUNOS 5.3 위에서 구현되었으며, 컴파일러는 SUNOS 5.3에서 동작하는 SPARCcompiler 3.0인 C++을 이용하여 구현하였다.

이 시스템은 interprocess communication(IPC) 부분은 기존의 C 언어가 가지는 라이브러리를 사용하였다.

비동기적인 통신은 Internet domain socket 및 (UDP) datagram message를 사용하여 구현하였다.

7. 병행 클래스의 구성

C++ 언어는 상속성, 추상화의 기본 개념을 가지고 있다. 병행 클래스인 CONCURRENCY는 이러한 상속성 및 추상화의 기본 개념 아래 설계 및 구현되어졌다. CONCURRENCY는 그림 10과 같이 클래스 구조를 보이고 있다.

has_split은 make_runnable과 attach에 의해 참(true)으로 설정되어진다. is_proxy는 make_runnable과 attach에 의해 참으로 설정되어지고 능동 객체에 의해 거짓(false)으로 설정되어진다.

scheduler는 CONCURRENCY 클래스를 상속받는 모든 클래스가 scheduler의 정의가 필요하며 이를 위해 scheduler는 virtual로 선언되어 있다.

send_result는 임의 리턴 타입을 가지는 함수를 수행한 결과를 전달하여야 하므로 파라메타로 받아들이는 타입의 정보를 알 수가 없다. 이를 위해 C++ 언어의 function template을 사용하여 구현하였다.

Ⅳ. 결론 및 향후 과제

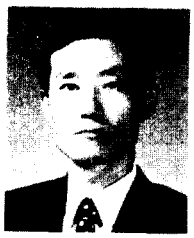
본 논문에서는 C++ 언어에 병행 객체지향 모델을 도입하기 위하여 능동 객체 생성 및 원격 메소드 호출 방식 및 attach 메소드를 사용하였으며, 동기화의 기법으로 비동기식 메시지 전달 및 미래 원격 프로시저 호출 방식을 사용하였다. 객체 내부의 동기화를 위하여 필요에 의한 요구 방식을 이용하여 보다 쉬운 동기 메카니즘을 사용하였다. 위의 방법들은 소프트웨어 공학의 재사용 원칙을 준수시키므로 객체지향 언어가 가지는 장점을 유지하기 위한 방법이었다. 본 논문은 실세계를 보다 정확히 묘사하기 위해서 병행성을 가져야 한다는 논리를 뒤받침 함으로써 C++ 사용자가 병행 객체지향 클래스 라이브러리만을 추가함으로써 보다 효율적인 병행 객체지향 프로그래밍을 할 수 있다는 장점이 있다.

향후 과제로는 서버와 클라이언트간에 파라메타 전달시 타입(type)에 관한 정보를 서로 주고받아야 하는데, 이는 사용자의 부담을 가중시키는 결과를 초래함으로 비효율적이다. 이를 위해 C++ 언어가 제공하지 않은 타입도 수용할 수 있는 ANY 타입의 구현을 위한 연구가 필요하며, 서버 객체 안에서의 메소드를 내부적으로 순차적으로 실행하므로 일관성을 유지할 수 있었지만 진정한 병행 객체지향 언어가 가져야 하는 메소드 내부의 병행성에 관한 메카니즘 연구가 필요하다. 또한 사용자에게 부담을 주는 scheduler의 작성을 자동화하는 방법이 연구 중이다.

참고문헌

1. Gul Agha. Concurrent Object-Oriented Programming. Commun. ACM 33, 9, Sept 1990.
2. Nierstrasz, O.M. Active objects in Hybrid. ACM SIGPLAN Not. 22, Dec. 1987.
3. America, P. Pool-T: A parallel object-oriented language. Object-oriented Concur rent Programming. M. Tokoro and A. Yonezawa, MIT Press, Cambridge, Mass., 1987, pp.199-220.
4. Andrews, G.R., et al. An Overview of the SR Language and Implementation. ACM Trans. Program. Lang. Syst. 10., Jan, 1988.
5. Akinori Yonezawa. ABCL : An Object-Oriented Concurrent System. MIT Press, 1990.
6. Buhr, P.A. , et al. μC++: Concurrency in the object-oriented language C++. Soft. Pract. Exp. 22, 2. Feb. 1992.
7. Michael L. Nelson Concurrency & Object - Oriented Programming. SIGPLAN Notices 26, 10. Octo 1991.
8. Caromel, D. Toward a method of object-oriented concurrent programming. Commun. ACM 36, 9, 1993.
9. Bershad, B.N., et al. PRESTO: A system for object-oriented parallel programming. Softw.Pract. Exp. 18. Aug. 1988.
10. Kafura, D.G. and Lee, K.H. Inheritance in ctor based concurrent object-oriented language. In Proceeding of ECOOP '89 (July 10-14, Nottingham). Cambridge University Press, 1989.
11. Briot, J. P. Actalk : A testbed for classifying and designing actor languages in Smalltalk-80 environment. In Proceedings of the Third ECOOP Conference '89. (July 10-14 1989, Nottingham). Cambridge University Press. 1989.
12. Cristina Videira, Karl J. Lieberherr. Abstracting Process-to-Function Relations in

Concurrent Object-Oriented Applica tion. In Mario Tokoro. Remo Pareschi. editor, Lecture Notes in Computer Science, pages 81-99. ECOOP'94, Springer-Verlag, July 1994.
13. Ellis, M., Stroustrup, T. The Annotated C++ Reference Manual. Addison-Wesley, Reading, Mass., 1990.
14. Murat Karaorman and John Bruno. Introducing Concurrency to a Sequential Language. Commun. ACM 36, 9. Sept, 1993.
15. Grady Booch. Object-Oriented Analysis and Design with Applications. The Benjamin/Cummings Publishing Company, Inc. 1994.
16. Gul Agha. An Overview of Actor Langu- ages. SIGPLAN Notices, 21(10) pages 58-67. Octo, 1986.
17. Stroustrup, Bjarne. The C++ Programming Language. 2nd ed. Addison-Wesley, Reading, Mass., 1991.



李 俊(Joon Lee) 정회원

1979년 : 조선대학교 전자공학과(학사)
1981년 : 조선대학교 대학원 전자공학과 졸업(석사)
1989년~현재 : 숭실대학교 대학원 전자계산학과 박사과정
1982년~현재 : 조선대학교 컴퓨터 공학과 부교수
※주관심 분야 : 운영체제, 병렬처리, 프로그래밍 언어

劉 載 旭(Chae Woo Yoo) 정회원

1976년 : 숭실대학교 전자계산학과 졸업
1978년~1985년 : 한국 과학기술원 전산학과(석사, 박사)
1986년~1987년 : Cornell Univ. 의 Visiting Scientist
1983년~현재 : 숭실대학교 컴퓨터학부 교수
※주관심 분야 : 컴파일러, 프로그래밍 환경, HCI

金 成 植(Sung Keun Kim) 정회원

1993년 : 숭실대학교 전자계산학과 졸업
1995년 : 숭실대학교 대학원 전자계산학과 졸업
1995년~현재 : 숭실대학교 대학원 전자계산학과 박사과정
※주관심 분야 : 병렬처리, 프로그래밍 환경

宋 後 鳳(Hoo Bong Song) 정회원

1956년 : 육군사관학교(학사)
1986년 : 중앙대학교 전자계산학과(석사)
1989년 : 조선대학교 전기공학과(박사)
1971년~현재 : 숭실대학교 컴퓨터학부 교수
※주관심 분야 : 운영체제, 프로그래밍 언어, 컴퓨터 보조 학습