클라이언트-서버 프로그래밍을 지원하기 위한 C++ 언어의 확장

(Extending C++ Language to Support Client-Server Programming)

류 경 동 † 이 강 우 †† 김 형 주 †††

(Kyoung-Dong Ryu) (Kang-Woo Lee) (Hyoung-Joo Kim)

요 약 분산프로그램을 작성하는 것은 아직 매우 복잡한 일로 여겨지고 있다 특히 현재 각광 받고 있는 클라이언트-서버 컴퓨팅환경에서 수행될 수 있는 프로그램을 손쉽게 작성하기 위한 툴이 시급한 실정이다.

본 논문에서는 이러한 분산프로그래밍의 복잡성을 완화하기 위하여 기존의 C++언어를 확장하여 쉬운 클라이언트-서버 프로그래밍을 지원하도록 하는 방법을 제시하고, 이를 구현한 확장된 C++언어 시스템(CSC++)의 구성과 기능에 대해 소개한다. 확장된 C++언어는 간단한 키워드 추가를 통하여, 클래스뿐만 아니라,클래스 계층구조에 의한 상속도 클라이언트 측에 유효하게 함으로써 클라이언트-서버 프로그래밍에서 분산에 대한 투명성과 재사용성을 증대시키게된다.

Abstract Programming in client-server environment is more complex than that of centralized computing environment.This paper provides a methodology to reduce the complexity of programming in client-server environment, through C++ language extension.The extended C++ language makes not only 'classes in the server'accessable remotely from the client, but also 'class hierarchy in the server' meaningful to the client.The proposed C++ language extension increases transparency and reusability of codes in client-server style programming by introducing a few additonal keywords.

1. 서 론

컴퓨터의 발전에 따라 컴퓨팅 환경과 프로그램의 수행환경에 많은 변화가 생겨났다. 특히, 네트워크 기술의 발전에 따른 통신의 고속화와 PC, 웍스테이션등의 성능의 비약적인 발전이 새로운 컴퓨팅 환경인 분산 컴퓨팅 환경을 구성하도록 하였다. 종래에는 하나의 중앙 대형 컴퓨터에서 모든 작업이 이루어지는데 비해,현재는 여러대의 고성능 PC, 웍스테이션들이 고속의 네트워크를 통해 연결되어 분산 작업이 가능하고, 또한 컴퓨팅 환경 구성의 비용도 절감할 수 있게 하였다. 이에 따라 프로그램도 단순히 하나의 컴퓨터에서 수행되는 형태에서 여러대의 컴퓨터에서 분산 수행되는 분산 프로그램의 형태로 변이되게 되었다[6]. 그 중에서도 클라이언트-서버 형태의 분산 프로그램의 개발이 아주 활발하게 이루어지고 있다. 클라이언트 프로그램 모듈은 주로 사용자 기기에서 수행되며, 사용자 측면의 여러기능을 수행하고, 내부적으로 서버 프로그램에 여러 서비스를 요구하여 이용한다. 서버 프로그램 모듈은 비교적 고성능의 기기에서 수행되며, 여러 개의 클라이언트에 공통된 서비스를 제공하고, 데이타베이스와 같은 공유된 자원에의 접근을 제공한다.

클라이언트-서버 컴퓨팅 환경은 여러가지 새로운 프로그램들---특히 DBMS---을 클라이언트-서버형태 프로그램으로 작성되게 할 뿐아니라, 기존의 중앙집중식으로 개발되었던 많은 프로그램들도 클라이언트-서버 형태 프로그램으로 다시 설계/개발되도록 하였다.

그러나 클라이언트-서버 프로그램의 작성은 아직 중앙집중식 프로그램에 비해 어려운 것으로 여겨지고 있

† 비 회 원 : 메릴랜드대학교 전산학과
†† 학생회원 : 서울대학교 전산과학과
††† 종신회원 : 서울대학교 컴퓨터공학과 교수
논문접수 : 1995년 5월 18일
심사완료 : 1996년 2월 16일

다. 클라이언트-서버 프로그램의 가장 큰 특징은 클라이언트 프로그램과 서버 프로그램 사이에 서비스 요구와 결과 전송이 네트워크를 통해 이루어져야 한다는 것이다. 그러나 소켓(socket)과 같은 트랜스포트 수준(transport layer)의 통신 API의 사용은 많은 프로그램 작성자에게 익숙치 않으므로, 분산 환경에서의 통신에 대한 세부적인 조작을 프로그래머가 정확히 한다는 것은 큰 어려움이 되어 클라이언트-서버 프로그램 개발의 생산성을 떨어뜨리는 한 요소가 된다.

프로그래밍 언어 차원에서 클라이언트-서버 프로그램을 지원하는 것은 위 문제를 해결하는 좋은 방법이 될 수 있다. 지금까지도 일반적인 분산 프로그램의 작성을 위해 새로이 여러 언어들이 연구되고 있고, 발표되고 있다[3,8]. 하지만, 위에서 설명한 클라이언트-서버의 특성을 잘 지원하여 주고, 기존의 복잡성을 완화 시킬 수 있는 기능을 기존의 프로그래밍 언어를 확장하여 개발하면,그 유용성과 사용 편이성은 더욱 증대될 것이다. 본 논문에서는 기존의 프로그래밍 언어를 클라이언트-서버 프로그래밍을 지원할 수 있도록 확장하여, 분산 프로그램의 작성이 중앙 집중식 프로그램의 작성과 거의 동일한 느낌을 줄 수 있도록 한다.

본 논문에서는 C++언어[12]를 확장하여, 한 프로그램(서버 프로그램) 내부에서 정의된 클래스[1]를 다른 기기 또는 다른 프로세스로서 수행되는 프로그램(클라이언트 프로그램)에게 사용가능하게 하는 클래스 export 방법에 의해 C++언어를 이용한 클라이언트-서버 프로그래밍을 지원한다. 특히, C++언어의 클래스 계층구조를 통한 상속과 클래스 export와의 관계를 깊이있게 연구하여, 클래스 계층구조에 의한 상속이 클래스 원격 접근자에게도 유효할 수 있도록 한다.

클라이언트-서버 프로그래밍을 지원하기 위해 C++언어를 채택한 이유는 다음과 같다.

* 현재 가장 널리 쓰이는 시스템 프로그램 언어 중의 하나다. 뿐만 아니라, 많은 응용 프로그램들도 C++언어로 작성되고 있다.
* 객체지향 프로그래밍 언어로써 재사용성, 확장성등 여러 장점들을 제공한다.
* 클래스(class)란 추상적 데이타 타입(abstract data type)과 메쏘드(method)에 의한 데이타 접근이 분산환경을 프로그래머에게 투명(transparent)하게 할 수 있는 좋은 통로가 될 수 있다.

1) 객체지향 언어에서의 클래스로서, 자료구조와 이를 접근/조작하기 위한 인터페이스를 가진다.

본 논문의 주요 관심대상은 단순한 투명한 원격 객체의 인터페이스 사용에 국한된 것이 아니라, 서버 쪽에서 구성된 클래스 계층구조의 여러가지 의미(semantic)를 투명하게 클라이언트에서도 보이게 하기 위한 노력에 있다. 일반적으로 객체 지향 프로그램 언어(특히 C++)를 이용하여 응용 프로그램을 작성할 때,사용자들은 단순히 클래스가 제공하는 인터페이스 뿐만을 이용하여 프로그램을 작성하는 것이 아니라, 여러 클래스가 서로 관계를 맺는 클래스 계층구조 개념도 함께 사용하여 프로그램을 작성한다[2]. 그러므로 객체 지향 개념을 사용한 프로그래밍을 할 때, 보다 더 투명하게 클라이언트에서 서버의 객체를 사용하려면, 투명한 원격 함수 호출을 위한 인자 전달 방법과 naming 에 대한 노력 뿐만 아니라, 서버 객체가 속한 클래스의 계층구조의 의미 역시 클라이언트에서도 사용할 수 있어야 한다. 본 논문에서는 서버에서의 클래스 계층구조 의미를 원격의 클라이언트에서도 투명하게 사용하려 할 경우, 발생 가능한 문제점을 지적하고, 이에 대한 해결책을 제시한다.

결과적으로 이 확장된 언어는 프로그램 작성자에게 클라이언트-서버 프로그램 개발의 난이도를 기존의 중앙집중식 프로그램의 개발과 거의 같게 하여, 클라이언트-서버 프로그램의 작성을 용이하게 하고, 또한 중앙집중식으로 이미 개발된 프로그램들을 손쉽게 클라이언트-서버 프로그램으로 변환할 수 있게 해줌으로써 클라이언트-서버 컴퓨팅 환경에서의 시스템 개발에 많은 도움이 된다. 실제로 이 논문에서 제시한 방법들은 SRP(SNU Relational DBMS Platform)을 클라이언트-서버형태로 재구성하는 데 이용되었다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 원격 객체에 대한 메쏘드 호출 기법과 클래스 계층구조를 중심으로 원래의 C++언어에서 제공되는 기능들을 클라이언트와 서버로 분산된 환경에서도 가능하게 하는 세부적인 방법들에 대해 설명한다. 3장에서는 실제로 이를 구현한 CSC++(Client-Server Style C++ Programming Lanugage) 시스템에서 확장한 C++언어와 시스템의 구성에 대해 설명하고, 간단한 프로그램 작성 예를들어 그 투명함과 작성 용이함을 보일 것이다. 4장에서 관련된 여러 연구와 비교하여 그 차이점을 논하고, 5장에서 결론을 맺는다. 마지막으로 CSC++언어를 이용한 클라이언트-서버 프로그램 작성 예와 그 처리 및 실행과정을 부록에 실었다.

2) C++ 언어의 경우에는 하위 클래스의 객체 주소는 상위 클래스의 객체형의 포인터 변수에 assign 될 수 있는 사실을 들 수 있다

2. 원격 메쏘드 호출 기법을 이용한 C++언어의 확장 방법

2.1 클래스 단위의 분산 메카니즘

C++언어의 클래스는 C언어의 함수보다는 추상적인 개념이다. C 언어에서는 흔히 RPC에 의해 함수가 원격 접근 가능한 단위가 되지만, C++에서의 RPC사용은 부자연스럽다. 이에 클래스 자체를 원격접근 가능한 광역적인 단위로 확장하여 분산수행을 가능하도록 한다. 이를 위해 클라이언트 프로그램이 원거리에 존재하는 서버의 클래스 라이브러리를 마치 클라이언트의 주소 영역 내에 있는 클래스 라이브러리와 동일한 방법으로 사용하게 하는 기능이 존재하여야 하는데, 이 절에서 이에 필요한 여러 기법을 제시한다.

C++ (혹은 객체지향) 언어에서 클래스 라이브러리의 기능을 사용하는 방법은 자신이 원하는 작업을 수행하는 클래스로부터 객체를 생성하여, 그 객체가 제공하는 인터페이스를 사용하는 것이다. 이런 계산환경에서 클라이언트가 서버의 클래스 라이브러리를 사용하는 자연스러운 방법으로 다음의 방법을 들 수 있다.

객체 생성: 클라이언트가 자신의 원하는 클래스로부터 객체를 생성시킬 때, 서버의 해당 클래스로부터 객체를 하나 생성시키고, 클라이언트에는 서버에서 생성된 객체를 대리하는 객체[3]를 생성시킨다. 물론 이러한 작업을 해당 객체를 사용하는 클라이언트에게는 보이지 않고, 이 대리 객체를 클라이언트 응용이 생성시킨 객체로 한다.

멤버 함수 호출: 클라이언트가 생성된 객체에 특정 멤버 함수를 호출하면, 대리 객체는 이 호출에 사용된 여러가지 인자들을 이용하여, 자신이 대신하고 있는 서버의 해당 객체를 원격 멤버 함수 호출을 하고, 그 결과값을 서버의 객체로 부터 받아 반환 한다.

객체 소멸: 클라이언트의 대리 객체는 자신이 대리하고 있는 해당 서버의 객체를 소멸시키고, 대리 객체 자신을 소멸시킨다.

이러한 방식의 분산수행을 지원하기 위해서 *대리 클래스(proxy class)*를 도입한다. 대리 클래스란 서버가 외부로 export한 클래스 라이브러리의 각 클래스에 대응되는 클래스로서 서버의 클래스 라이브러리를 사용할 클라이언트에 위치한다(그림 1 참조). 대리 클래스는 public 멤버 함수로 그에 대응되는 서버 클래스와 같은 public 멤버 함수 인터페이스[4]를 갖게하고 protected 멤버로 서버 객체 식별자를 갖게 하여 클라이언트에서 서버 클래스의 사용을 대리하게 한다. 대리 클래스는 실제로 원격 메쏘드 호출을 통해 서버 클래스의 해당 객체에 해당 멤버 함수를 호출하게 된다. 특히 C++언어에서 클라이언트의 대리 클래스를 이용한 투명한 서버클래스 접근은 다음의 방법으로 구현된다.

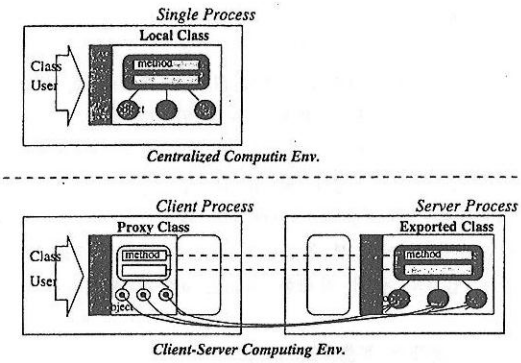


그림 1 대리 클래스에 의한 위치 투명성.

객체 생성: 대리 객체가 생성될 때, 자동적으로 호출되는 생성자 함수를 이용한다. 생성자 함수에서 해당 서버 클래스의 객체를 생성하는 요구를 서버로 보내, 생성된 서버 객체의 식별자를 대리 객체에 기록한다. 이때 호출되는 생성자 함수에 인자가 있는 경우는 서버로 객체생성 요구를 보낼 때 이를 함께 전송하여 서버에서 해당 객체를 생성할 때, 사용도록 한다.

대리 객체에 대한 멤버 함수 호출: 클라이언트 응용이 생성된 대리 객체에 대해 멤버 함수를 호출하면, 전달된 인자들과, 자신이 대신하는 해당 서버객체의 식별자, 그리고 해당 멤버 함수 식별자를 함께 패킷화한 다음, 서버로 전송하여 수행하게 하고, 그 결과를 서버로부터 받아 호출자에게 반환 한다. 서버에서는 클라이언트로부터 멤버 함수 호출요청을 받으면, 함께 전달된 패킷으로부터 객체의 식별자, 호출할 멤버 함수 식별자, 그리고 인자들을 역마샬링(demarshalling)하여, 멤버 함수 호출을 수행한다.

객체 소멸: 대리 객체의 소멸자 함수가 불리면, 먼저 서버 객체 식별자를 인자로 하는 객체 소멸 요구를 서버로 보내어, 서버로 하여금, 서버 객체 식별자에 해당하는

3) *대리 객체(proxy object)*라 부르자.

4) 이 멤버 함수에 서버클래스의 public 생성자 함수와 소멸자 함수 들도 포함된다.

객체를 소멸하게 한다.

위와같은 방법으로 대리 클래스/객체를 사용하면, 클라이언트 응용으로 하여금,서버 클래스 라이브러리를 로칼하게 사용하는 것과 같은 투명성을 제공할 수 있다. 이러한 방법은 C++언어를 확장하여 분산 프로그래밍 언어로서 제공하는 기존의 여러 연구에서도 유사하게 사용되는 기법이다[11, 9, 5]. 그러나 객체지향 언어를 사용한 클래스 라이브러리는 각 클래스가 독립적으로 존재하는 경우는 거의 없고, 대부분 복잡한 클래스 계층구조로 구성되어 있다. 그러므로 단순히 앞에서 설명한 방법만으로는 클라이언트 응용에게 보다 완전한 투명성을 주기에는 많은 문제점이 있다. 즉, 위 방법은 한 클래스에 대한 사용에는 투명성을 줄 수 있지만, 서버 클래스의 계층구조로부터 생성되는 여러가지 의미에 대한 투명성을 제공하지 않는다. 본 논문에서 제안한 기법이 기존의 관련 기법과의 좀더 자세한 비교는 4절을 참조하기 바란다.

2.2 클래스 계층구조의 처리

원격접근을 가능하도록 명시한 클래스[5]의 대리 멤버 함수 호출가능만을 이용하여 서버의 클래스 라이브러리를 export 시켜, 클라이언트로 하여금 사용하게 하면, export 되는 클래스 라이브러리내의 클래스들의 계층구조에 의한 상속을 클라이언트 응용에게 마치 서버 클래스 라이브러리를 로칼하게 사용하는 듯한 투명성을 제공하지 못하게 된다.

예를들어 C++ 언어에서는 클래스 B가 클래스 A의 하위 클래스라 할때, 클라이언트에 단순히 클래스 A, B를 각각 독립적으로 export 한 경우에는, 클래스 B의 객체의 주소가 클래스 A 형의 포인터 변수에 assign 될 수 없어, 그 만큼 투명성이 저하되기 때문에, 클라이언트에서도 클래스 B 가 클래스 A의 하위 클래스라는 의미가 보존되어야 한다. C++의 클래스 계층구조에 인해 다음과 같은 상황에서 새로운 문제점들이 발생하게 된다.

* 서버에서 어떤 상위 클래스부터 상속을 받아 구성된 하위 클래스만을 export 한 경우.
* 서버에서 하나의 클래스 계층구조에 속하여 IS-A 관계를 가지고 있는 클래스들을 동시에 export하는 경우.
* 클라이언트에서 exported 클래스로부터 지역적으로 하위 클래스를 구성하려는 경우.

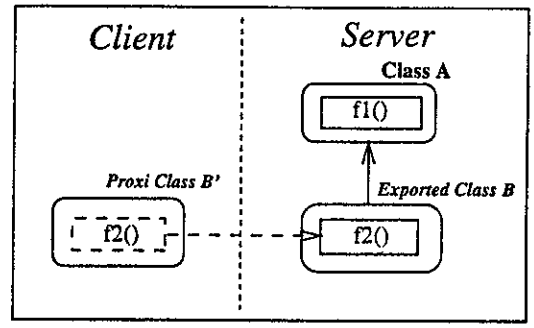
5) 앞으로 exported 클래스라 하자

그러면 각 경우에 관해 그 문제점과 해결 방법을 제시한다.

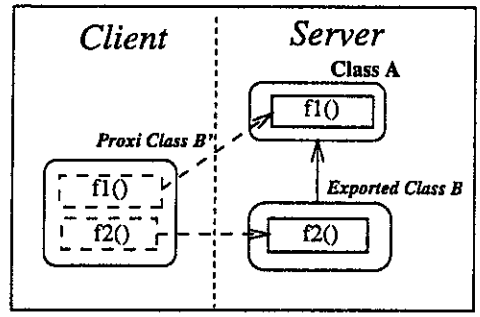
Exported 클래스가 상위클래스를 가지는 경우

Exported 클래스가 단순한 독립적인 클래스가 아니라, 상위 클래스를 가지는 하위 클래스일 수가 있다(그림 2 에서 클래스 B 는 클래스 A 의 하위 클래스이다.).

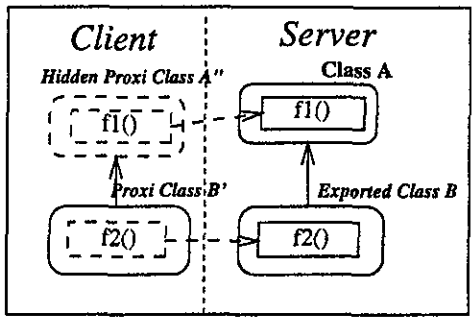
이런 경우에는 export 된 클래스의 인터페이스 외에도 상위 클래스로 부터 상속받은 인터페이스에 대한 고려가 있어야 한다. Exported 클래스만을 대상으로 클라이언트에 대리 클래스를 구성하면, 서버의 상위 클래스로부터 상속받은 멤버 함수를 클라이언트에서는 사용하지 못하게 된다. 예를 들어 설명하면 그림 2(a)에서 서버에서는 클래스 B가 클래스 A의 하위 클래스로서 멤버 함수 f1()을 상속받아, 클래스 B의 객체 b에 대해 b.f1()이 호출 가능함에도 불구하고, 클라이언트에서는 대리 객체를 통해 이를 호출할 수가 없다. 이는 대리 클래스의 구성시, 서버에서 존재하는 exported 클래스의



(a) Silly Proxi Class



(b) Interface Flattening



(c) Implicitly Exporting Base Class

그림 2 계층구조에 속한 하위 클래스의 export의 해결 방법

클래스 계층구조를 고려하지 않음으로 생기는 문제이다. 이러한 문제에 관한 해결방법을 몇가지 제시한다.

1. 인터페이스 Flattening 방법

첫번째 방법은 export 시킬 대리 클래스의 인터페이스에 단순히 해당 exported 클래스의 멤버 함수들과 클래스가 상위 클래스로 부터 상속받은 모든 멤버 함수들을 함께 추가하는 방법이 있다. 즉 exported 클래스 B를 위해 대리 클래스 B'를 구성할 때, B의 멤버 함수 f2() 뿐만 아니라, A로 부터 상속받는 f1()도 B'의 멤버 함수로 추가시키는 것이다.

이렇게 하면, 상속받은 멤버 함수의 호출은 가능해지지만, 각 클래스의 멤버 함수단계에서 그 상속여부를 결정하고, 다중 상속 문제도 해결할 수 있는 Proxy 생성기를 구성해야 하므로 그 구현이 복잡해지게 된다. 또한 하나의 상위 클래스로부터 구성된 여러개의 하위 클래스가 export 된 경우, 상위 클래스의 하나의 멤버 함수에 대해서, export 된 하위 클래스 모두에 대리 멤버 함수를 각각 따로 구성해야 하는 불합리한 점이 생긴다.

2. 암시적 상위클래스 Exporting 방법

두번째 방법은 export 될 클래스외에도 해당 클래스의 모든 상위 클래스도 함께 export 시켜 클라이언트에 해당 대리 클래스를 별도로 추가해서 만드는 것이다. 클라이언트에서는 이들 export된 대리 클래스들 간에 서버에서와 동일하게 클래스 계층구조를 구성하여, 상속받은 인터페이스에 대한 접근이 클라이언트 내부적으로 결정되도록 한다[6].

6) 함께 export 되는 상위 클래스들은 은닉 대리 클래스라 부른다.

그림 2(c)에서 보듯이. 서버내에서 클래스 B만 export 한 경우라도, 인터페이스의 상속 여지가 있는 클래스 A도 암시적으로 export 된 것으로 간주하여, 이에 대해서도 대리 클래스 A''를 클라이언트에 구성한다. 또한 서버와 같게 대리 클래스 A'', B'간에도 상속 관계를 가지도록 클래스 계층구조를 구성하게 된다. 그러면, B의 객체 b에 대한 A로부터 상속받은 f1()의 호출 b.f1()은 클라이언트에서 대리 클래스 B'에 대해, b'.f1()에 의해 대행되게 된다. 이 때의 A에서 상속받은 인터페이스의 제공은, 클라이언트의 대리클래스 계층구조에 의해 자동적으로 해결된다.

그러나 클래스 A''는 서버에서 명시적으로 export 한 클래스가 아니므로, 그 객체의 생성이 허용되어서는 안된다. 그래서, 이를 지원하기 위해 클래스 A를 가지고 클래스 A''를 구성할 때, *public* 생성자 함수와 소멸자 함수를 protected 멤버 함수로 전환하게 된다.

Export 되는 클래스들이 서버내에서 클래스 계층구조를 이루고 있는 경우

서버에서 export 된 클래스들이 독립적인 것들이 아니라, 클래스 계층구조에 의해 관련지어진 것들일 수 있다. 즉 상위클래스와 하위클래스가 동시에 export 될 수가 있다. 이 때 마찬가지로 클라이언트에서도 이들 대리 클래스간에 클래스 계층구조를 구성하도록 하여야 한다 (그림 3 참조).

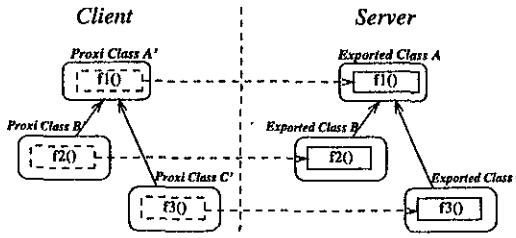


그림 3 클래스 계층구조상의 여러 클래스의 export

대리 클래스간의 계층구조에 의해, 상위 클래스의 인터페이스를 상속 받아 하위 클래스의 객체에의 호출이 가능하게 된다.

이때, 아래의 코드에서 나타나는 polymorphism 문제도 해결할 수 있어야 하는데, 이것도 대리 클래스 계층구조에 의해 자연스럽게 수용된다.

A' a',

```
B′ b′;

a′.f1(&b′);          // 하위 클래스 객체를 인자로 쓸 경우
                     // f1(A′* ptr)
```

그러나 하위 클래스 B′의 객체 생성시 앞에서 밝힌 기본 메카니즘의 대리 생성자 함수의 개념을 그대로 이용하면 생성자 함수 호출의 상위로의 전파가 클라이언트와 서버에서 이중으로 일어나는 문제가 발생하게 된다.

원격 메쏘드 호출을 지원하는 기본 메카니즘에 의하면, exported 객체의 생성은 대리 클래스의 생성자 함수에 의해 이루어진다. 즉, 클라이언트 측에서 대리 클래스의 객체를 하나 생성하면, 그 때 호출되는 대리 클래스 생성자 함수는 이 대리 객체의 초기화 기능을 할 뿐만 아니라, 서버의 exported 클래스에 객체 생성을 원격으로 요구하게 된다. 그림 3에서 보면, exported 클래스 B의 객체를 생성하기 위해 클라이언트에서 대리 클래스 B′의 객체를 생성한다. 그러면, 이 대리 객체가 생성되면서 생성자 함수가 호출되는데, 생성자 함수는 해당 클래스가 상위 클래스를 가지는 경우 먼저, 그 상위 클래스의 생성자 함수를 호출하여 상속받은 영역에 대해 초기화를 수행하게 된다. 즉 A′()이 먼저 호출되고, B′()이 호출된다. 이 때 발생하는 문제점은 전파되어 호출되는 상위 클래스의 생성자 함수 A′()도 대리 클래스 생성자 함수라는 점이다. 그러므로, A′()도 exported 클래스 A의 객체 생성을 원격으로 요구하게 되어 의도하지 않은 객체를 서버에 생성시키게 된다. 다시 말하면, 생성자 함수 호출의 상위 전파가 클라이언트의 대리 클래스 계층구조와 서버의 exported 클래스 계층구조 양쪽에서 발생된다.

대리 클래스 계층구조의 상위전파에 의한 불필요한 객체 생성을 막기위하여, 클라이언트에서는 생성자 함수 상위 전파가 강제적으로 상위 클래스의 무기능 생성자 함수(dummy contructor)를 호출하여 이루어지도록 하는 것이다. 여기서 무기능 생성자 함수란 대리 클래스의 다른 생성자 함수와는 달리 원격 객체 생성 기능이 없고, 단지 클라이언트에서 생성자 함수 상위 전파에만 대응하도록 구현되는 것이다[7]. 이 무기능 생성자 함수를 A′(OID)라 하면, 이번엔 B′의 객체 생성시 A′()가 상위 전파에 의해 호출되는 것이 아니라, A′(OID)가 호출되어 서버에 클래스 A의 객체 생성을 원격으로 요구하지 않게 된다. 결과적으로 상위 클래스를 가지는 클래스의 원격 객체 생성은 그 생성자 함수 호출이 서버에서만 이루어지므로, 하나의 클래스 계층구조에 속하는 여러 클래스를 export 하는 경우도 수용할 수 있게된다.

Exported 클래스에 대해서 클라이언트에서 지역적으로 하위 클래스를 구성하려는 경우

제공되는 클래스를 사용하는 방법은 직접 객체를 생성하여, 멤버 함수 호출에 의해 사용할 수 도 있지만, 그 클래스를 상위 클래스로 하여 하위 클래스를 구성하여 사용할 수도 있다. 그러면, 서버에 존재하는 exported 클래스로부터, 하위 클래스를 클라이언트에서 지역적으로 구성하는 것은 어떨까?

이것도 대리 클래스에 의해 제공될 수 있다. 사용자는 exported 클래스에서 하위 클래스를 구성하지만, 실제로는 대리 클래스의 하위 클래스로 구성된다(그림 4). 이때의 하위 클래스는 exported 클래스로 부터 private 멤버를 상속받아 객체의 멤버영역을 구성할 수가 없다. 즉 대리 클래스는 exported 클래스의 private 멤버에 대한 정보는 전혀 가지고 있지 않으므로 하위 클래스가 이를 실제로 상속받아 멤버영역을 구성할 수가 없다.

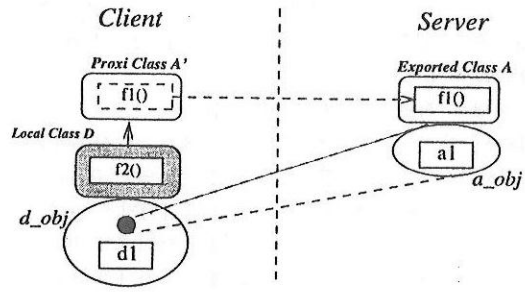


그림 4 클라이언트에서 exported 클래스로 부터의 하위 클래스 작성(조합 객체의 구성)

이 문제는 exported 클래스에 객체를 만든 후, 하위 클래스의 객체가 이를 가리키도록(서버 객체 식별자에 의해)하여 같은 기능을 할수 있도록 한다[8].

그림 4를 보면, exported 클래스 A로 부터 클라이언트에서 하위 클래스를 구성하였다. 즉, 클라이언트에 있는 대리 클래스 A′로 부터 하위 클래스 D가 구성되어 있다. 그러면, 개념적으로 클래스 A의 멤버 a1과 클래스

7) 이 생성자 함수는 인자로 객체 식별자 타입을 가져서 다른 사용자 정의 생성자 함수와 구분된다.

8) 본 논문에서는 이것을 조합객체(fragmented object) 방법이라 부른다

D에서 추가된 d1 두 개의 멤버영역으로 구성된 객체가 클라이언트에 생성되어야 한다. 하지만, 실제로 서버에 존재하는 클래스 A의 객체 멤버 영역에 대한 정보를 클라이언트에서는 알 수가 없다 이 문제는 클라이언트에서 새로이 정의된 클래스 D가 클래스 A의 멤버영역 대신 이 멤버영역을 가지는 클래스 A의 객체를 가리키는 서버 객체 식별자를 클래스 A'로 부터 상속받아 가짐으로써 해결된다.

하위 클래스 D의 객체 생성은 다음과 같이 이루어진다. 먼저 하위 클래스 D의 객체를 생성하면, 실제 새로이 정의된 멤버 d1의 영역과 상속받은 객체 식별자 영역이 할당된 d_obj가 생기고, 이 클래스의 생성자 함수가 불린 후, 생성자 함수 상위전파에 의해 클래스 A'의 생성자 함수가 호출되게 된다 이 때, 로칼 하위 클래스 생성자 함수가 대리 클래스 생성자 함수로 상위 전파될 때는 이전에 설명한 대리 하위 클래스 생성자 함수의 상위 전파와 달리 무기능 생성자 함수를 부르지 않고, 원래의 생성자 함수를 부르게 된다. 그 이유는 개념적으로 상위 클래스가 되어있는 export 클래스에 객체가 생성되어 상속을 대신하여야 하기 때문이다. 즉 export 객체 서버에 a_obj가 생성되고, 클라이언트에 있는 d_obj의 객체 식별자 멤버가 이를 가리키게 된다.

n개의 exported 클래스를 부모로 하여 로칼하게 하위 클래스를 생성하였으면, n+1개의 객체가 모여, 사용자에게는 하나의 객체로 보여지게 된다. 상속받은 멤버는 해당 클래스로부터 상속받은 멤버 함수에 의해서만 호출된다는 가정하에서만, 이 조합객체에 대한 멤버 함수 호출이 모두 유효하게 되므로, protected 멤버사용은 제한되어야만 한다.

앞에서 설명한 세가지 메카니즘을 이용하여, 서버내의 클래스 계층구조가 클라이언트에서 투명하게 같은 의미를 가질수 있고, 클래스와 객체의 실제 위치에 관계없이 객체생성은 물론, 하위 클래스의 구성도 사용자는 로칼하게 존재하는 것과 별 차이없이 할 수 있게 되었다.

3. 언어적 확장 및 구현 : CSC++

언어의 구문적 확장은 분산 환경에서 필요로하는 최소한으로 이루어지고,클래스와 클래스 계층구조등이 분산된 환경에서도 기존과 거의 같은 기능을 할 수 있도록 의미적 확장이 이루어 진다(투명성의 제공) 또한 본 연구에서 앞에서 제안된 방법을 실제로 구현하여 확장된 C++언어 시스템을 구현하였다(CSC++). 이 절에서는 C++언어의 확장을 위해 추가된 키워드들을 살펴보고, CSC++로 프로그램을 작성하는 간단한 예를 들어, 클라이언트-서버 프로그램이 어떻게 손쉽게 작성되고, 수행되는 지 보일 것이다.

3.1 언어적 확장

앞장에서 설명한 메카니즘을 제공하기 위하여 여러가지 키워드가 확장된다(표 1 참조). 특히, 추가된 키워드들은 클래스의 원격성을 지시하기 위한 것, 멤버 함수의 원격호출을 위한 인자 전달에 관한 것 그리고 서버 프로세스를 접근하기 위한 정보에 대한 명세를 위한 것 세가지 범주로 나누어 진다.

표 1 추가된 키워드와 의미

추가된 키워드	종 류	의 미
[export]	클래스 export	특정 클래스를 export 할 것을 명시
[string]	인자 타입	문자열 타입임을 명시
[array n]	인자 타입	고정 길이 배열타입임을 명시
[array $n]	인자 타입	가변 길이 배열타입임을 명시
[in]	인자 전달 방향	서버로만 전달하는 인자
[out]	인자 전달 방향	클라이언트로만 전달하는 인자
[inout]	인자 전달 방향	서버로 전달되고 되돌아오는 인자

먼저 클래스 작성자는 그 클래스가 외부에 보여질 것인지 아닌지를 선택할 수 있어야 한다 그리하여 보여질 클래스들만 [export]란 키워드로 명시하게 된다. 이 export 지정에 의해 단순히 그 클래스만이 원격으로 접근 가능하게 되는 것이 아니라, 그 상위 클래스에서 상속받은 인터페이스도 원격으로 사용 가능하도록 앞장에서 제공한 방법들을 이용하여 지원하게 된다.

서버 작성자에게 export될 클래스의 작성이 지역적인 클래스 작성과 최대한 동일할 수 있도록,분산환경에서 기존의 C++ 에서의 의미를 대부분 그대로 수용하지만, 인자전달이 네트워크를 통해 일어나므로, 전송을 위해 인자의 타입에 대한 명시적인 지시가 필요하고, 또한 이때 전송량의 최소화를 위해 인자의 사용형태를 지시할 수 있도록 해야한다. [array i] 키워드는 인자가 배열인 경우 그 타입이 단순한 포인터 타입과 다름을 명시하고 또한 전체 배열의 길이를 명시하여 정확한 인자 전송이 일어날 수 있도록 한다. 고정 크기의 배열인 경우 그 길이를 상수로 명시해야 하고, 가변 크기인 경우에는, 해당 배열의 크기를 값으로 가지는 인자 번호를 추가하여 [array $3] 의 형태로 표기한다.마찬가지로 char 포인터

와 스트링은 그 표현상으로 전혀 구분이 안되므로, 인자 전송시 구분할 수가 없다. 그러므로 스트링 타입도 [string] 키워드로 명시하게 된다.

이 외에도 인자 전송의 효율성을 위한 인자 사용형태 지시를 위한 키워드로 [in], [out], [inout]이 있다. 실제로 멤버 함수 내부에서 값을 이용하기만 하고, 변화시키지 않는 경우는 [in] 이란 키워드로 이를 지시함으로서, 원격 메쏘드 호출시 서버에 이 인자를 보내기만 하고, 결과 패킷에는 넣지않아 클라이언트로 돌아오지 않게 함으로써, 인자 전송에 효율성을 주게 된다. [out] 은 반대로 멤버 함수 내부에서는 그 값을이용하지 않고 단순히 메쏘드 호출자에 값을 넘기기 위한 인자인 경우에 사용하게 된다. [inout] 은 두 가지 용도를 다하는 참조에 의한 호출인 경우 사용하게 된다.

3.2 CSC++ 을 이용한 프로그램 작성 및 수행 과정

확장된 C++언어를 이용하여 클라이언트-서버 프로그램을 단순한 단일 수행 프로그램처럼 손쉽게 작성할 수 있다. CSC++을 이용한 프로그램의 작성 및 수행의 예는 부록에 상세히 서술하였다.

작성 및 수행과정을 간단히 살펴보면 다음과 같다. CSC++ 로 작성된 프로그램은 Proxi 생성기에 의해 해석되어 자동적으로 서버 메인 모듈을 구성하고, 또한 해당하는 대리 클래스를 작성하여 클라이언트 사용자에게 제공하게 된다. 클라이언트 프로그램 작성자는 이 대리 클래스와 관련 화일들을 이용하여 마치 로칼한 클래스 라이브러리를 이용하듯 클라이언트 프로그램을 작성하면 된다. 이렇게 작성된 클라이언트-서버 프로그램의 실행은 먼저 자동으로 생성된 서버 메인 모듈을 해당 기기에서 수행시키고, 클라이언트 기기에서는 작성된 클라이언트 프로그램을 수행시키면 된다. Proxi 생성기에 의해 자동으로 생성된 서버 메인 모듈, 서버 스텁 클래스 모듈, 대리 클래스 모듈들은 원격 메쏘드 호출 라이브러리를 이용해 상호 연관되어 작동하게 된다(그림 5).

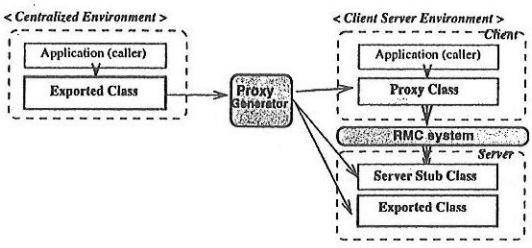


그림 5 CSC++ 시스템의 구성과 사용형태

4. 관련 연구와의 비교

용이한 분산 프로그래밍을 위한 기존의 언어적 확장 연구 결과들과 상용화된 도구 개발, 그리고 분산 객체 호출 표준화 노력은 지금까지 여러 연구자 들에 의해서 진행되어 왔다. 이러한 기존의 연구 결과와 본 논문에서 제안한 C++ 언어 확장의 근본적인 차이점은 앞서 설명한 바와같이 C++ 언의 클래스 계층구조의 이해이다. 기존의 대부분의 연구는 객체 자체를 마치 자신의 주소영역에 있는 것과 같은 투명성으로 다룰 수 있는 연구에 주로 집중되어 왔다.

그러나 본 연구에서는 개개의 객체 사용의 투명성 뿐만 아니라, 클래스 계층구조의 의미도 원격지에 투명하게 보일 수 있는 가에 대한 연구에 집중하였다. 현재 상용의 많은 원격 프로시져 호출 패키지들(Sun RPC, DCE RPC등)이 분산프로그래밍 도구로써 사용되고 있다[2,10,4]. 이러한 RPC 패키지들은 고유의 자료형을 갖고 있어, 좋은 함수 호출 투명성을 제공하지 못하는 단점을 갖고 있어[13], 실제 사용에 있어서는 많은 어려움이 따른다. 또한 RPC의 프로세져 개념은 C++ 언어와 같은 객체지향 개념의 언어와 함께 사용할 경우에는 두 개념에 불일치로 인하여 보다 많은 불편함이 따른다.

Argus는 MIT에서 만들어진 분산 프로그래밍 언어이다[8]. 이 언어의 프로그래밍적 요소는 기존의 CLU언어를 그대로 이용하고 있다. 분산 프로그램은 여러 개의 가디안(Guardian)들로 구성된다. 가디안은 클래스와 유사하게 하나의 독립적 모듈 단위이며, 핸들러(handler)에 의해서만 접근가능 하다. 하지만 핸들러의 호출은 값에 의한 호출만 가능하고, 가디안은 데이타 추상화 개념은 지원하지만, 객체지향 개념이 아니므로 가디안간의 상속같은 기능은 제공하지 않는다. 이 시스템은 특히 수행의 원자성(Atomicity)을 보장하며, 중첩된 트랜잭션을 지원한다. 분산 수행시의 트랜잭션의 원자성과 직렬성(serializability)보장이 시스템의 주 관심대상이라 할 수 있다. 하지만 이 시스템은 분산의 여러 정보를 사용자에게 이용하게 하는 대신, 사용상의 투명성은 저하된다. 이 시스템은 클라이언트-서버 형태의 트랜잭션 기능을 포함한 응용 프로그램(OLTP 응용등)을 구성하기에는 적합하지만, DBMS같은 시스템 프로그램 자체를 구성하기에는 부적합하다.

Avalon/C++은 C++언어를 확장하여 만든 것으로, Camelot을 기반으로 분산된 원자성 객체(atomic object)에 대한 트랜잭션의 개념을 지원하며, 주 목적은 분산, 결함허용 응용프로그램을 지원하는 것이다[7]. 이

시스템은 트랜잭션을 언어 내부적으로 지원하는 것을 주 관심사로 하고 있어, 그 부하가 너무 과중하고, 또한 C++언어의 투명성있는 사용에 관한 지원이 약하여, 이 언어를 이용하여 클라이언트-서버 시스템 프로그램을 투명성 있게 작성하기 힘들다.

Arjuna 는 객체지향 프로그래밍 시스템으로 분산된 환경에서 지속성(persistency)있는 객체의 무결성(integrity)을 보장하기 위해 중첩된 원자성 트랜잭션(nested atomic transaction)을 지원한다[11]. Arjuna도 C++언어를 확장한 형태로 본 논문의 확장된 C++언어와 유사한 기능을 가진다. 하지만, 다음과 같은 몇가지 이유들로 클라이언트-서버 DBMS같은 분산 시스템 프로그램을 작성하는 데는 부적합하다. 먼저, Arjuna 에서의 클래스 계층구조 내의 특정 클래스를 export 하는 경우에는 앞서 설명한 인터페이스 *flattening* 기법을 사용하여 서로 클래스 계층구조로 연관된 여러 클래스를 export 하는 경우에는 해당 클래스들의 계층구조를 통한 상속의 의미가 없어진다. 그리고, 하나의 서버 프로세스가 여러 클래스를 export 할 수 없는 단점을 갖는다. 이것은 하나의 클래스를 export 시키는 경우, 반드시 해당 클래스를 사용을 전담하는 서버 프로세스가 생성되기 때문이다. 즉, n개의 클래스를 export 하는 경우에는 n개의 서버 프로세스가 생성되게 된다. 그러므로 한 서버에서 여러 클래스를 export 시켜 서비스를 제공하는 클라이언트-서버 시스템 프로그램을 구성하기에는 부적합하다. 또한, Arjuna에서는 멤버 함수 인자로 원격 클래스 타입을 가지는 객체등을 허용하지 못하는 제약점을 갖고 있다. 이에 비해 본 연구에서 제한 방법과 구현된 CSC++언어는 하나의 서버가 여러 클래스를 export 시켜, 동시에 해당 클래스의 객체 호출 서비스를 제공할 수 있어, 클라이언트-서버 구조의 시스템 프로그래밍에 보다 용이하고, Arjuna에 비해 인자에 대한 제약이 적다.

Electra 는 RMC(Remote Method Call)방법을 이용해 분산을 지원하는 객체지향형 툴이며, C++ 클래스 라이브러리, 스텁생성기(stub generator) 등에 의해 제공된다[9]. Concert/C[1]는 C언어를 확장하여 프로세스의 조작과 분산을 언어 내부적으로 지원한다. 또한 RPC를 언어 내에서 투명하게 지원하며 비동기적 통신 기능도 포함하고 있다. CSC++ 시스템은 Concert/C와 그 이용에서 유사성을 많이 가지나, 객체지향 언어인 C++언어를 확장하여 객체지향 개념에서의 여러 장점들을 지원하게 하는 특성을 가진다.

분산환경에서의 객체관리를 표준화하기 위하여 CORBA(Common Object Request Broker Architecture and Specification)라는 명세를 OMG에서 제시하였다[5]. 여기서 객체지향 시스템들에서 원격 객체를 접근하는 표준적인 방법을 제시하고 있다. 인터페이스의 명세는 CORBA-IDL 언어를 사용하도록 하고 있는데 C++과 유사한 면을 많이 가지고 있다. CORBA 의 경우에는 이미 만들어진 객체를 원격지에 사용할 수 있는 기능을 제공하는 반면, 클라이언트에서 동적으로 자신이 원하는 작업을 수행할 수 있는 객체를 생성시키는 방법은 제공되어 있지 않다. 전반적인 CORBA 는 응용 프로그램에서의 분산 객체의 투명한 사용에 주로 중점을 둔 반면, CSC++언어에서는 클라이언트가 원하는 작업을 수행하는 객체를 동적으로 생성하여, 원하는 작업을 수행한 후, 다시 해당 원격 객체를 소멸시키는 기능을 제공한다. 뿐만 아니라, CORBA 역시 앞서 언급한 바와 같이 클래스들의 계층구조에 대한 언급은 없다.

5. 결론 및 향후과제

본 논문에서는 C++언어를 확장하여 분산수행이 가능한 클라이언트-서버 프로그램을 작성할 수 있도록 하는 방법을 제시하고, 또한 이를 구현한 CSC++ 시스템에 대해 간단히 소개하였다.

하위레벨의 툴을 이용한 분산프로그램 작성의 복잡성을 제거하고, C++언어로 구현되는 여러 프로그램들이 쉽게 클라이언트-서버화 될 수 있도록 하기위해 기존의 C++ 구문에 키워드를 추가하여 로칼한 클래스 라이브러리가 클라이언트-서버 형태로 수행 가능한 모듈로 변환되게 하여 줌으로써, 프로그래머에게 분산환경에 대한 투명성을 주고자 하는 것이 본 논에서 제안하고 구현한 언어시스템의 주 목적이다. 이를 위해 RPC기법을 C++의 객체의 메쏘드 호출에 응용한 원격 메쏘드 호출 기법을 정의하였고, 원격 객체의 생성, 원격 객체의 소멸을 사용자에게 투명하게 하기 위해 생성자 함수와 소멸자 함수라는 특수한 멤버함수를 이용하였다. 이로써 원격 객체에 대한 모든 조작이 사용자에게는 로칼한 클래스 객체에 대한 조작과 동일하게 되었다.

C++언어에는 단순한 클래스개념을 넘어 클래스 계층구조에 의한 상속을 지원하고 있다. 클래스 계층구조에 의한 상속은 단순한 클래스 export방법을 이용하면 그 의미가 사라지게 되는데, 이를 보완하기 위해 무기능 생성자를 이용한 암시적 상위클래스 export기법을 제안하고 구현하였다.

현재 존재하는 여러 분산프로그래밍 시스템, 특히 C++ 언어를 확장한 시스템들에서는 언어적으로 트랜잭

션 개념의 지원에 중점을 두고 있어, 클래스 계층구조의 처리를 지원하고 있지 않거나, 인자 타입에서의 제약이 강하여 투명성이 떨어지는 데 비해 본 연구에서는 이를 지원하는 방법을 제안하고 구현하므로써 본래 C++의 기능을 분산환경에서도 충분히 활용할 수 있게 하였다.

현재 CSC++ 시스템 프로토타입은 원격 메쏘드 호출 시스템과 Proxy 생성기로 구성되어 있는데, 유닉스(Unix)용으로 소켓을 사용하도록 되어있다. 현재는 윈도우즈(Windows)용을 구현하고 있는 중이다. 이것이 완성되면, 현재 아주 널리 쓰이는 윈도우즈-클라이언트, 유닉스-서버 형태의 이기종간 클라이언트-서버 프로그램의 작성이 손쉽게 이루어 질 수 있을 것으로 기대된다.

부 록

A CSC++ 을 이용한 프로그램 예

본 장에서는 CSC++언어를 사용한 클라이언트-서버 프로그램 작성 예를 들고, Prxoy 생성기에 의해 생성되는 모듈들과 이를 클라이언트와 서버측에서 사용하는 방법에 대해 설명하도록 하겠다.

A.1 서버 원격 클래스의 작성

먼저 원격으로 제공할 서비스들을 클래스들로 구성한다. 이때 클래스들은 서로간에 계층구조를 가져서 상속의 특성을 이용할 수 있다. 구성된 클래스들 중 클라이언트 즉, 원격 클래스 사용자에게 직접 객체 생성 및 접

```
[export] class SeqFile {
public
   SeqFile();
   ~SeqFile();
   int create([string] char devName[], int perm =0600);
   int open([string] char devName[], int mode =O_RDWR);
   int close();
   int read(int offset,[array 1024][out] char* bufp, int nbytes, int whence);
   int write(int offset,[array 1024][in] char* bufp, int nbytes, int whence);
private
   int fDesc;
};

[export] class FixedSeqFile   public SeqFile {
public
   FixedSeqFile(int recLen);
   ~FixedSeqFile();
   int read(int recID,[array 1024][out] char* bufp, int noOfRecs);
   int write(int recID,[array 1024][in] char* bufp, int noOfRecs);
   int copyfrom(FixedSeqFile& fileObj);
   int recLength() const;
private
   int recLen;
};

[servername "FileSrv"]
[serverid   100]
[versionid    1]
[serverloc  10000, "147 45 10 109"]
```

그림 6 서버 원격 클래스의 작성 예: SeqFile.eh

근을 허용할 클래스들에 대해서만 export를 명시한다. 작성된 클래스들의 멤버 함수 구성시에는 인자에 관한 키워드들을 적절히 사용하도록 한다. 그러면 원격 클래스를 작성한 간단한 예를 보도록 하자.

그림 6의 예를 보면 SeqFile과 FixedSeqFile 두 개의 클래스를 [export] 란 확장된 키워드로 명시하여 동시에 export 하고 있다. 이 경우 단순히 두 클래스가 독립적으로 원격 접근 가능해 질 뿐아니라, 본문에서 설명한 것과 같이 클래스 계층구조에 의한 상속도 유효해진다. 즉, 클라이언트에서도 FixedSeqFile 클래스의 객체에 상속받은 멤버 함수 open, close 등을 호출할 수가 있게된다. SeqFile 클래스의 경우는 생성자 함수, 소멸자 함수, create, open, close, read, write 7개의 멤버 함수가 인터페이스가 되어 객체를 조작하는 데 사용될 수 있으며, FixedSeqFile 클래스는 생성자 함수, 소멸자 함수, read, write, copyfrom, recLength 6개의 직접 정의된 멤버 함수와 상위 클래스 SeqFile 로부터 상속받은 3개의 멤버 함수를 합한 9개의 멤버 함수가 인터페이스가 된다. SeqFile 클래스의 create 와 open 에서 devName 인자를 [string] 으로 지정하여 문자열 타입임을 지정하고 있다. 각 클래스의 read, write 멤버 함수를 보면 버퍼를 인자로 사용하고 있는데, [array] 키워드를 이용하여 배열임을 나타내고, 배열의 크기를 지정하여 인자 전송시 전송량을 결정할 수 있도록 하고 있다. read 멤버 함수의 버퍼 인자는 출력용이므로 [out] 을 지정하여 함수내에서 내용을 받아오는데 이용함을 명시하였고, write 멤버 함수에서는 반대로 [in] 을 지정하여 버퍼의 값을 함수내에서 이용하기만 하도록 하고 있다 FixedSeqFile 클래스의 copyfrom 멤버 함수를 보면 그 인자로 원격 클래스의 객체를 받도록 하고 있는데, 이 점이 다른 시스템(Arjuna)이 가지는 인자에 대한 제약을 완화시켜 exported 클래스 작성의 투명성을 증대시키는 것이다. 자동적으로 생성되는 서버 메인 모듈에 대한 명세도 여기에 표현되어 있다. 이 클래스들을 원격 제공하는 서버의 이름은 FileSrv 로 명시되어 있다. 서버번호와 버젼번호도 [serverid], [versionid] 키워드에 의해 유일한 값으로 지정되어 있으며, 서버가 수행되는 기기의 IP 주소와 통신을 위한 포트번호도 [serverloc] 에 의해 주어져 있다 멤버 함수들의 구현은 로칼한 클래스 멤버 함수와 동일하게 하면 된다.

A.2 Proxy 생성기를 이용한 여러모듈들의 생성 및 수행

CSC++로 작성된 클래스 헤더 화일 SeqFile.eh 를 Proxy 생성기 cscpp로 처리하면 그림 7과 같이 9개의

화일이 자동적으로 생성된다. 이들 화일들은 서버용 모듈과 클라이언트용 모듈로 나뉘게 된다. 먼저 서버용 모듈들은 서버 정보 명세모듈(FileSrv.gfn, FileSrv.srv, FileSrv.def), 서버 클래스 스텁 모듈(SeqFile_S.hh, SeqFile_S.C), 서버 메인 모듈(FileSrv_main.C)로 구성된다. 이 화일들을 C++ 컴파일러로 컴파일한 후 서버 메인 모듈을 지정된 기기에서 수행시키면 해당 클래스들이 원격으로 서비스된다.

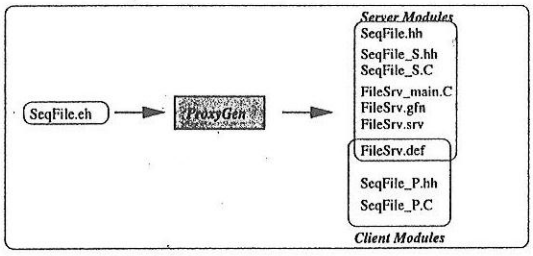


그림 7 Proxi 생성기에 의해 자동 생성된 화일들

클라이언트용 모듈로는 서버 정보 정의 화일(FileSrv.def)과 대리 클래스 모듈(SeqFile_P.hh, SeqFile_P.C)로 이루어 진다. 클라이언트 응용프로그램 작성자는 exported 클래스와 동일한 인터페이스를 갖는 대리 클래스에 대해 로칼한 프로그램을 작성하듯이 프로그래밍 하면 된다. 다음은 클라이언트 응용 프로그램의 간단한 예이다. (fileapp.C)

```
#include "SeqFile_P.C"
int main()
{
   . . .
   SeqFile sf(100);
   sf.create("app_file1");
   . . .
   sf.write(1, in_buff, 1);
   . . .
   sf.close();
   . . .
}
```

작성된 클라이언트 응용 프로그램은 컴파일하여 대리 클래스 정의화일과 링크하여 사용한다. 클라이언트 응용 메인 모듈을 클라이언트 기기에서 수행시키면, 내부적으로 원격 메쏘드 호출 시스템을 이용하여 해당 서버를 동적으로 접근하고, 객체의 생성과 멤버 함수호출을 원격으로 수행하게 된다.

참 고 문 헌

[1] J. S. Auerbach, A. P. Goldberg, G. S. Goldszmidt, A. S. Gopal, M. T. Kennedy, J. R. Rao, and J.R. Russell, "Concert/C: Language for Distributed Programming," In *Proceedings of the Winter 1994 USENIX Conference*, pp. 79-96, Jan. 1994.

[2] A. D. Birrell and B. J. Nelson. "Implementing Remote Procedure Calls," *ACM Trans. Comput. Syst.*, 2(1):39--59, Feb. 1984.

[3] A. P. Black., "The Eden Programming Language," Technical Report 85-09-01, Dept. of Computer Science, University of Washington, 1985.

[4] X. Corporation., "Courier: The Remote Procedure Call Protocol," Technical Report Xerox System Integration Standard 038112, Xerox Corporation, 1981.

[5] O. M. Group., "The Common Object Request Broker: Architecture and Specification," Technical Report OMG Document Number 91.12.1, OMG, Dec. 1991.

[6] J. Hwett, editor. *Client-Server Computing -Commercial Strategies*, Ovum Ltd., 1990.

[7] R. A. Lerner. "Reliable Servers: Design and Implementation in Avalon/C++," Technical Report CMU-CS-88-177, School of Computer Science, Carnegi Mellon University, Sept. 1988.

[8] B. Liskov. "Distributed Programming in Argus," *ACM Trans. on Programming Languages and Systems*, 31(3):300--312, Mar. 1988.

[9] S. Maffeis. "Object-Oriented Distributed Programming with ELECTRA," Technical Report 93.17, Computer Science Dept., University of Zurich, Mar. 1993.

[10] B. Nelson. "Remote Procedure Call," Technical Report CSL-81-9, Xerox Palo Alto Research Center, 1981.

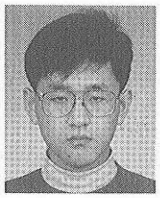
[11] S. K. Shrivastava, G. N. Dixon, and G. D. Parrington. "An Overview of the Arjuna Distributed Programming System," *IEEE Software*, June 1991.

[12] B. Stroustrup, editor. *The C++ Programming Language*. Addison Wesley, 1986.

[13] W. E. Weihl. "Remote Procedure Call," In S. Mullender, editor, *Distributed Systems*, chapter 4. ACM Press (Frontier Series), 1989.

류 경 동

1993년 2월 서울대학교 계산통계학과(이학사). 1995년 2월 서울대학교 계산통계학과 전산과학전공(이학석사). 1995년 3월 ~ 현재, Univ. of Maryland 전산과 박사과정. 관심분야는 분산 시스템, 객체지향 시스템.



이 강 우

1991년 2월 서울대학교 계산통계학과(이학사). 1993년 2월 서울대학교 계산통계학과 전산과학전공(이학석사). 1993년 3월 ~ 현재, 서울대학교 전산과학과 박사과정, 관심분야는 트랜잭션 시스템, 객체지향 시스템,

김 형 주

1982년 2월 서울대학교 컴퓨터공학과 졸업. 1985년 8월 Univ. of Texas at Austin, 전자계산학 석사. 1988년 5월 Univ. of Texas at Austin, 전자계산학 박사. 1988년 5월 ~ 9월 Univ. of Texas at Austin, Post-Doc. 1988년 9월 ~1990년 12월 Georgia Institute of Technology, 조교수. 1991년 1월 ~ 현재 서울대학교 컴퓨터공학과, 조교수. 관심분야는 객체지향 시스템, 사용자 인터페이스, 데이타베이스.