서버 기반 컴퓨팅을 활용한 씬-클라이언트 아키텍처 설계 및 구현

(Design & Implementation of Thin-Client Architecture using Server Based Computing)

송 민 규*
(Min-Gyu Song)

요 약 네트워크 및 컴퓨터 기술의 발전에 힘입어 컴퓨팅 서비스 방식에도 상당한 변혁이 일어나고 있다. 1960년대 메인프레임으로 출발하였던 컴퓨터 시스템은 1980년대 출시된 PC를 거쳐 이제는 서버 기반의 컴퓨팅 패러다임이라 할 수 있는 씬-클라이언트(Thin-Client)로 진화하고 있다[1]. 씬-클라이언트 컴퓨팅 방식에서 네트워크는 애플리케이션 전달을 위한 플랫폼으로서 그 역할을 수행하며 클라이언트는 원격에서 서버 상의 애플리케이션을 실행할 수 있다. 또한 네트워크에 접속된 컴퓨팅 자원을 공유하는 것도 가능하다[2]. 씬-클라이언트 아키텍처 구현을 위한 한 방법으로 본 논문에서는 컴포넌트와 분산 컴퓨팅 기술을 제시하였고 그를 위한 기술로서 COM(Component Object Model)과 PYRO(PYthon Remote Objects)를 활용하였다. 본 논문에서는 씬-클라이언트의 개념과 원리를 시작으로 그를 구현하기 위한 기술적 응용에 대해 논의할 것이다. 그리고 이를 기반으로 씬-클라이언트의 아키텍처를 설계 및 구현하고자 한다.

핵심주제어 : 서버 기반 컴퓨팅, 씬-클라이언트, COM(Component Object Model), PYRO(PYthon Remote Objects)

Abstract In the field of computing service, there is a copernican revolution indebted to development of network & computer technology. Computer system, which is set to mainframe in the 1960's, is advancing torwards to the paradigm of server based computing, so-called thin-client[1]. In thin-client computing, network is the platform which is responsible for transfer of application, so that client execute application installed on server. It is also possible that each system share the computing resource connected with network[2]. In this paper, we suggest component & distributed computing technology as a measn for the implementation of thin-client architecture hence, make the best use of COM(Component Object Model) and PYRO(PYthon Remote Objects). We talk about the concept and mechanism of thin-client at the beginning, and propose the design of network architecture for the implementation thin-client.

Key Words : Server Based Computing, Thin-Client, COM(Component Object Model), PYRO(PYthon Remote Objects)

1. 서 론

지난 수십 년간 컴퓨터 및 네트워크 기술은 비약적인 발전을 이룩하였다. 1960년대 메인프레임으로 시작된 컴퓨터의 역사는 1980년대의 PC (Personal Computer)를 거쳐 현재 네트워크 기반의 시스템으로 발전해 나가고 있다. 컴퓨팅 서비스

* 한국천문연구원 전파천문연구부

방식에 있어서도 기존의 클라이언트/서버 방식에서 씬-클라이언트 형태의 서버 기반 컴퓨팅(Server Based Computing)으로 확대되어 가고 있다. 서버 기반의 컴퓨팅에서 작업 수행에 필요한 애플리케이션과 데이터는 서버에 저장되며 사용자는 네트워크 상에서 이를 호출함으로써 원하는 작업을 수행하게 된다[1]. 이는 1990년대 말 대두되었던 네트워크 컴퓨터의 개념과 일맥상통하는 것으로서, 네트워크 기술의 발전에 힘입어 서버 기반 컴퓨팅은 점차 현실화되고 있다. 네트워크 컴퓨터 패러다임에서 클라이언트는 네트워크를 경유하여 서버 상의 애플리케이션을 실행하며 그에 대한 결과 또한 얻을 수 있어야 한다. 이는 기존에 데스크탑에서 수행되던 애플리케이션이 향후 분산 애플리케이션 형태로 진화할 것이라는 것을 의미하며 이 과정에서 웹 브라우저는 HTML 문서를 출력하는 수준을 넘어 애플리케이션 실행 플랫폼으로 업그레이드되어질 것으로 예상되고 있다[2]. 단일 시스템에 국한되던 애플리케이션을 네트워크를 경유하여 웹 브라우저에서 실행할 수 있도록 하는 기술에는 ActiveX,자바 애플릿, 플래쉬 등이 있는데 본 논문에서는 컴포넌트와 분산 객체 기술을 Active Scripting 형태로 설계함으로써 씬-클라이언트의 가능성에 대해 조망해보고자 한다.

본 논문은 다음의 순서에 따라 구성되었다. 서론에 이어 2장에서는 서버 기반 컴퓨팅과 씬-클라이언트의 개념 및 원리에 대해 살펴볼 것이다. 3장에서는 그에 연관된 기술 및 문제점을 소개한 후 이를 극복할 수 있는 방안에 대해 기술하고자 한다. 4장에서는 씬-클라이언트 기반의 컴퓨팅 구현에 있어 기존 단점을 극복할 수 있는 COM (Component Object Model), PYRO(PYthon Remote Object) 기술의 활용에 대해 알아보고자 하며 5장에서는 이를 기반으로 클라이언트에서 원격의 애플리케이션을 실행할 수 있는 씬-클라이언트 아키텍처를 제작하고자 한다. 그리고 마지막 결론 부분에서 이러한 논의 및 전개를 기반으로 씬-클라이언트의 가능성을 조망함으로써 본 논문을 마무리하고자 한다.

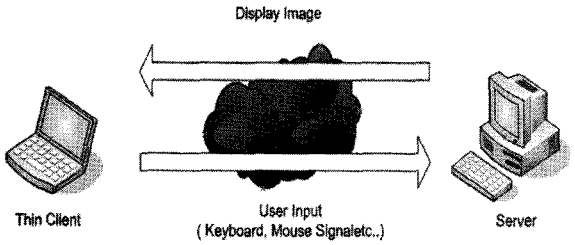
2. 씬-클라이언트의 개념 및 원리

본 절에서는 씬-클라이언트의 근간이라 할 수 있는 서버 기반 컴퓨팅에 대해 먼저 살펴보고자 한다. 애플리케이션과 데이터가 서버에 저장되는 서버 기반의 컴퓨팅을 통하여 씬-클라이언트의 구성 및 메커니즘에 먼저 접근하고자 한다.

2.1 씬-클라이언트

1981년 IBM에서 출시한 PC(Personal Computer)는 컴퓨터 시스템 활용에 있어 말할 수 없는 변화를 야기시켰다. CPU, HDD, 메모리 등 별도의 디바이스를 탑재한 PC의 등장으로 사용자들은 전산실이라는 제한된 환경에서 관리자의 승인을 받아 시스템을 활용하던 한계를 극복하게 되었다. 하지만 PC의 경우 세 가지 크나큰 단점을 가지고 있다. 먼저 초기 도입 비용이 크다는 것이 그 첫 번째이고 시스템 유지 관리, 소프트웨어 업그레이드와 같은 TCO(Total Cost of Ownership)등이 크다는 것이 두 번째, 그리고 데이터 보안에 취약하다는 것이 그 세 번째에 해당한다[3]. 일반 사용자의 경우 PC 상에서 수행하는 작업의 상당수가 웹 서핑, 메일 작성, 문서 작업 등이라는 점에서 기존의 PC는 오버 스펙이라 할 수 있다. 또한 대기업의 경우 한해 시스템 유지 관리 비용은 날이 갈수록 증가하는 추세에 있으며 주요 정보의 유출로 인하여 각 기업체에서는 이를 방지하기 위한 보안 강화에 역점을 기울이고 있는 상황이다. 이에 따라 대두된 것이 서버 기반의 컴퓨팅으로서 네트워크 기술에 발전에 따라 구현이 가능하게 되었다.

서버 기반의 컴퓨팅에서 애플리케이션과 데이터는 기존의 클라이언트/서버 방식과 달리 서버에 저장된다. 클라이언트는 서버 측으로 사용자 입력을 전송하고 서버로부터는 디스플레이 이미지를 전송받는다. 이를 통하여 마치 로컬에서와 같이 원하는 작업을 수행하며 관련 프로토콜로서 VNC (Virtual Network Computing), RDP(Remote Desktop Protocol), ICA(Independent Computing Architecture) 등이 활용되고 있다[4]. 서버의 애플리케이션을 업그레이드함으로써 네트워크 상의 모든 클라이언트는 최신의 애플리케이션을 활용할 수 있으며 이러한 측면에서 서버 기반 컴퓨팅은 유지, 보수의 측면에서 최적의 선택이라 할 수 있다. 서버 기반 컴퓨팅의 개략도를



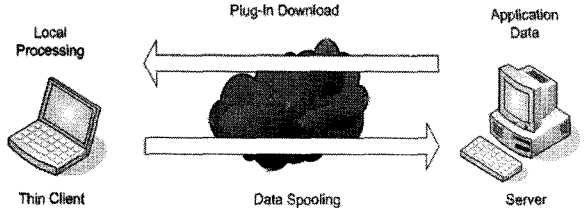
(그림 1) 서버 기반 컴퓨팅의 메커니즘

도시하면 위 그림 1과 같다.

2.2 플러그-인 기반의 컴퓨팅

서버 기반의 컴퓨팅에서 클라이언트의 입력신호는 서버로 전달된다. 모든 작업 및 연산 처리는 애플리케이션, 데이터가 저장된 서버에서 이루어지며 서버의 디스플레이 이미지가 클라이언트로 전달되었다. 입력 신호와 디스플레이 이미지가 송·수신되는 서버 기반 컴퓨팅과는 달리 플러그-인 기반의 컴퓨팅에서는 실행 가능한 애플리케이션이 클라이언트로 전달된다. 일반적으로 이는 웹 애플리케이션의 형태로서 실행되며 서버와의 통신에 기반하여 사용자는 원하는 작업을 수행하게 된다. 모든 연산 처리가 전적으로 서버에 의존하는 서버 기반 컴퓨팅과 달리 플러그-인 기반의 컴퓨팅에서 클라이언트는 연산의 일부분을 담당하며 그를 위한 임의의 애플리케이션이 웹 브라우저와 상호작용하며 설치 운용된다[5]. 이에 관련된 네트워크 기술로 ActiveX, 자바 애플릿, 플래쉬 등이 있으며 이를 통하여 클라이언트는 연산을 처리할 수 있는 애플리케이션을 웹 브라우저 상에서 실행하는 것이 가능하다.

이러한 플러그-인 기술 기반의 컴퓨팅 개념을 그림으로 간략히 요약해보면 아래와 같다.



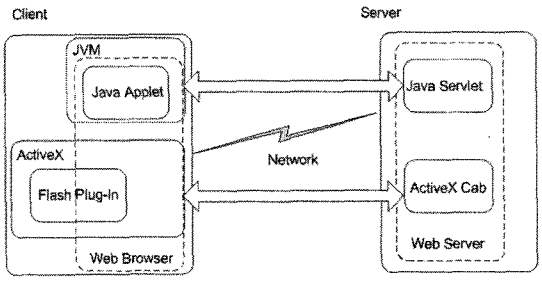
(그림 2) 플러그-인 기술을 활용한 네트워크 컴퓨팅

3. 기존 기술 개괄 및 문제점

본 절에서는 씬-클라이언트를 설계 및 구현함에 있어 관련있는 기술에 대해 살펴보고 해당 기술들의 문제점에 대해 알아보고자 한다. 또한 현재의 단점과 문제점을 극복할 수 있는 방안에 대해서도 간략히 논의하기로 한다.

3.1. 기존의 연관 기술 및 문제점

네트워크를 통하여 애플리케이션이 접근되고 클라이언트가 그를 호출하며 작업을 수행하는 네트워크 기반의 컴퓨팅에서 인터프리터 엔진의 역할은 매우 중요하다. 서버 시스템을 통하여 애플리케이션을 실행하기 위해서는 클라이언트에서도 그에 연관된 서버와의 통신, 객체 호출 등을 수행할 수 있어야 한다. 이는 클라이언트 측의 인터프리터 엔진을 통하여 구현 가능하며 일반적으로 웹 브라우저에 내장된다. 사용자 인터페이스이자 애플리케이션이 구동하는 공간으로서 웹 브라우저는 사용자의 입력을 받아 이를 서버로 전송하고 서버의 응답을 사용자에게 반환한다. 스마트-클라이언트에서 웹 브라우저는 단순히 HTML 페이지를 나타내는 수준을 넘어 인터프리터 엔진을 기반으로 애플리케이션이 호출되고 실제 실행되는 인터페이스로 업그레이드하고 있다. 따라서 사용자들은 인터넷에 연결되어 있다면 언제, 어디서든지 작업을 수행하는 것이 가능하다[6]. 이러한 개념에서 네트워크는 애플리케이션을 전달하는 플랫폼으로서의 역할을 수행하며, 그에 유사한 기능으로 활용되었던 것이 ActiveX, 자바 애플릿, 플래시에 해당하는 것이었다. 씬-클라이언트 컴퓨팅 관련 이러한 컴포넌트 기술의 동작 메커니즘을 도시해보면 아래와 같다.



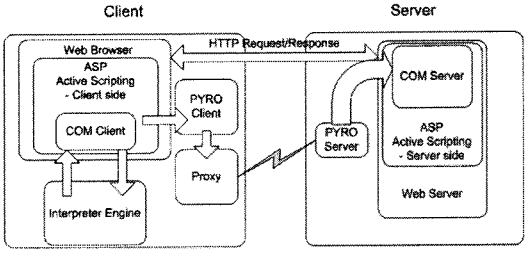
(그림 3) 컴포넌트 기술의 동작 메커니즘

ActiveX, 자바 애플릿 등의 기술을 통하여 웹 브라우저의 기능은 애플리케이션 실행 플랫폼으로 확장되었으며 웹 브라우저 하나만으로 사용자는 원하는 작업을 수행하는 것이 가능하게 되었다. 하지만 이러한 장점, 편리성에도 불구하고 각 기술은 활용에 있어 상당한 문제점을 가지고 있는데 클라이언트 자원에 대한 임의 접근 및 설치가 그에 해당한다 할 수 있다. ActiveX를 먼저 살펴보면 서버로부터 전달되는 ActiveX 컨트롤 형태의 애플리케이션이 클라이언트의 리소스에 영향을 주며 설치되는데 이로 인하여 사용자 시스템이 악성코드, 바이러스 등에 노출되는 문제점을 야기시켰고 이는 이는 보안의 측면에서 크나큰 취약성으로 나타나게 되었다. 물론 클라이언트의 자원에 접근하며 직접 설치되는 것이 단점으로만 작용하는 것은 아니다. 클라이언트에 애플리케이션이 직접 설치되기 때문에 서버와의 빈번한 통신을 지양할 수 있으며 웹 상에서 마치 데스크탑 애플리케이션을 사용하는 것 같은 작업 환경을 구축할 수 있다. 하지만 악성코드, 바이러스로 인한 피해가 기하급수적으로 증가하고 보안의 중요성이 갈수록 증가되는 추세에 있어 사용자의 시스템 자원 접근을 방조하는 것과 같은 기존 ActiveX의 특성은 이상적인 서버 기반 컴퓨팅과도 거리가 있으며 보안의 측면에서 반드시 지양되어져야 할 것이다[3].

자바 애플릿의 경우는 다양한 플랫폼 지원은 물론 클라이언트의 자원에 직접 접근하는 대신 웹 브라우저에 한정되는 특성으로 보안의 측면에서 보다 안정적이다. 하지만 애플릿 구동을 위하여 각 클라이언트 시스템에 JVM(Java Virtual Machine)이 설치되어야 하는 것은 크나큰 단점으로서 웹 브라우저에 기반하여 각종 애플리케이션을 활용 및 원하는 작업을 수행코자 하는 스마트-클라이언트의 부합되지 않는다[6].

3.2. 문제점 해결 방안

기존 기술의 이러한 문제점을 극복하기 위하여 본 논문에서는 서버 기반 컴퓨팅과 분산 컴퓨팅을 결합한 새로운 형태의 씬-클라이어트 컴퓨팅 구현을 위한 아키텍처를 제안하고자 한다. 클라이언트 자원에 대한 접근을 지양함은 물론, 별도의 소프트웨어의 설치 없이 기본적으로 웹 브라우저만을 가지고 원하는 작업을 수행할 수 있도록 하는 것이 본 논문에서 설계하고자 하는 스마트-클라이언트 아키텍처이다. 이를 통하여 보안의 측면에서 보다 안정적인 시스템을 구축할 수 있음은 물론 웹 브라우저=클라이언트의 등식이 성립되는 진정한 서버 기반 컴퓨팅을 구현하는 것이 가능하다. 이러한 요소에 입각하여 씬-클라이언트의 아키텍처를 간략히 도시해보면 그림 4와 같다.



(그림 4) 씬-클라이언트 아키텍처

그림에서도 알 수 있듯이 클라이언트에서 활용하게 될 애플리케이션은 COM 기반의 클래스 객체로 제작하였으며 그에 대한 사용자 요청은 ASP 페이지에 포함되어 클라이언트로 전달되도록 하였다[7]. 애플리케이션이 기본적으로 중앙의 서버에 위치하기에 클라이언트에 인터프리터 엔진을 제외한 임의의 모듈 및 프로그램이 설치되는 것을 방지할 수 있으며, 웹 브라우저만으로도 애플리케이션을 실행하며 원하는 작업을 수행할 수 있다. 네트워크 상의 분산된 시스템에서 COM 클라이언트 / COM 서버 간의 통신을 구현하기 위하여 분산 컴퓨팅 미들웨어인 PYRO(PYRO Remote Objects)를 활용하였고, 이를 통하여 네트워크 상의 자원 및 서비스에 효율적으로 접근할 수 있도록 하였다.

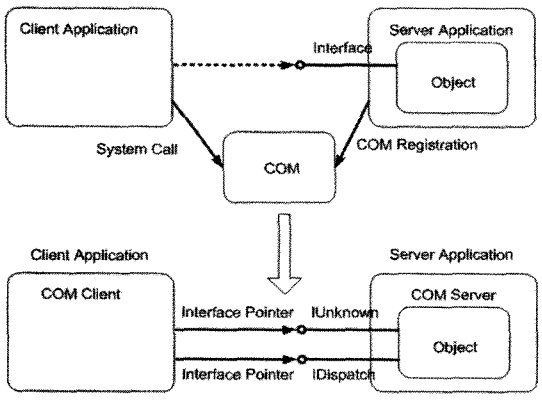
4. 스마트 아키텍처 설계를 위한 제반 기술

스마트 아키텍처를 설계함에 있어 본 논문에서는 COM(Component Object Model)과 PYRO PYthon Remote Object)를 활용하고자 한다. 이를 통하여 서버 기반의 분산 컴퓨팅을 만족시킬 수 있는데 본 절에서는 이러한 기술의 기능 및 원리에 대해 살펴보고자 한다.

4.1 COM(Component Object Model)

COM(Component Object Model)은 윈도우즈 상에서 각 소프트웨어 간 원활한 통신을 구현하기 위한 마이크로소프트사의 컴포넌트 기술이다. 초기 OLE(Object Linking & Embedding)의 형태로 출발한 COM은 임의의 프로그램이나 시스템을 이루는 컴포넌트들이 상호 작용할 수 있도록 통일된 인터페이스를 제공하며 서로 다른 언어로 작성된 프로그램 및 시스템 간 통신을 구현할 수 있다. 또한 시스템 환경에 통합된 소프트웨어를 효율적으로 구축하는 것이 가능하다.

COM은 객체와 인터페이스를 정의하는데 필요한 규칙과 추상적인 개념을 정의한다. COM기반의 클래스 객체는 IUnknown 또는 IDispatch 인터페이스를 통하여 자신의 메소드, 프로퍼티를 외부에 제공하며 워드, 파워포인트, 엑셀 등의 오피스와 인터넷 익스플로러, 파일 탐색기와 같은 윈도우 프로그램이 이러한 COM 서버로 시스템에 등록된다[7]. COM 기반으로 구현된 클래스 객체는 그 자체가 하나의 애플리케이션으로서 타 시스템이 호출할 수 있는 함수, 즉 메소드를 포함한다. 이런 메소드 중 하나를 발생시키려면, 클라이언트는 메소드를 포함한 인터페이스를 가리키는 포인터를 얻어와야 하며 IUnknown 또는 IDispatch 인터페이스를 경유하여 원하는 객체의 기능을 호출한다. 그림 5에 이에 대한 대략적인 메커니즘을 나타내었다.



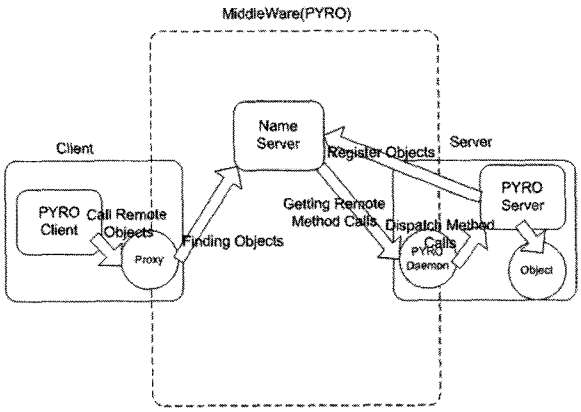
(그림 5) COM의 동작 메커니즘

COM객체는 그 구성 및 역할에 따라 COM 서버와 COM 클라이언트로 분류할 수 있다. 구현 또는 활용하고자 하는 메소드, 속성을 가진 객체가 COM 서버가 되고 그에 접근하고자 하는 객체가 COM 클라이언트가 된다. COM 클라이언트는 IDispatch 인터페이스를 경유하여 COM 서버에 구현된 메소드를 호출하며 사용자는 이를 통하여 원하는 연산을 수행하는 것이 가능하다.

4.2 PYRO(PYthon Remote Objects)

PYRO는 PYthon Remote Objects의 약자로서 네트워크 상의 여러 시스템에 분산된 객체 간의 통신을 구현하는 미들웨어라 할 수 있다. 객체 지향 형태의 RPC를 제공하며, 시스템과 언어에 독립적인 분산 객체 기술인 CORBA, 자바의 RMI(Remote Method Invocation)와 더불어 분산 객체 구현을 위한 기술로 널리 활용되고 있다[8].

PYRO를 활용함으로써 얻을 수 있는 가장 큰 이점으로 원격 객체에 대한 투명한 접근 및 메소드 호출이 있으며, 클라이언트 측의 Proxy, 서버 측의 네임 서버를 통하여 사용자는 원격 시스템에 위치한 객체를 마치 로컬에서와 같이 활용하는 것이 가능하다. PYRO에서 네임 서버는 시스템에 존재하는 모든 객체의 이름과 위치를 알고 있는 중앙 데이터베이스(central database)에 해당한다. PYRO 서버가 이들 객체의 위치를 네임 서버에 등록함으로써, 클라이언트는 호출하고자 하는 객체의 위치정보인 URI(Universal Resourcer Identifier)를 얻게 되고 그에 관련된 호출은 해당 객체로 포워딩된다. 이에 관련된 메커니즘을 그림으로 나타내면 아래와 같다.



(그림 6) PYRO 구성 및 동작 원리

5. 컴포넌트 기반의 분산 컴퓨팅 구현

본 절에서는 이전 절에서 살펴본 COM (Component Object Model)과 PYRO(PYthon Remote Object)를 바탕으로 분산 컴퓨팅을 설계하기 위한 아키텍처에 대해 논의하기로 한다. COM 기반으로 애플리케이션을 구현함으로써 보다 효율적으로 통합된 시스템에 접근할 수 있도록 하였으며 PYRO를 접목시킴으로써 위치에 관계없이 네트워크 상의 어디에서라도 애플리케이션의 기능을 손쉽게 활용할 수 있도록 하였다. 이러한 두 가지 메커니즘을 근거로 씬-클라이언트 컴퓨팅을 구현하기 세부적 방안에 대해 살펴보고자 한다.

5.1 시스템 구현을 위한 기본 아키텍처

본 논문에서 구현하고자 하는 씬-클라이언트 컴퓨팅에서 애플리케이션 및 프로그램은 각 사용자 PC가 아닌 중앙의 서버에 위치한다. 따라서 각 클라이언트는 원격에서 서버 상의 애플리케이션을 실행할 수 있어야 함은 물론, 그에 대한 결과를 적절한 형태로 얻을 수 있어야 한다. 기존에 클라이언트에서 실행하던 애플리케이션 및 프로그램은 서버 상으로 일원화되며 본 논문에서는 그를 위한 한 방법으로 COM을 제안하고자 한다. COM에서 모든 애플리케이션, 프로그램은 다수의 메소드를 가진 클래스 객체로 구현되어 시스템에 일목요연하게 등록될 수 있다. 따라서 개발자의 입장에서 일정한 형태에 따라 통일된 애플리케이션을 제작할 수 있으며, 관리자로서도 통합된 애플리케이션을 보다 체계적으로 관리할 수 있다.

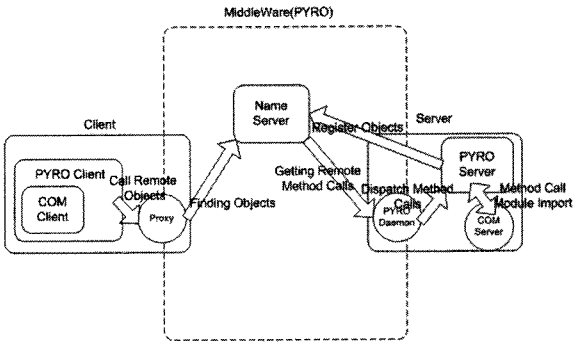
COM은 일반적으로 클라이언트와 서버가 단일 시스템에 위치하는 로컬 서버의 형태로서 본 논문에서 구현하고자 하는 분산 컴퓨팅과는 상치되는 측면도 있다. 물론 리모트 서버의 일환으로 DCOM을 염두에 둘 수도 있지만 이 경우 네트워크 상의 시스템이 모두 윈도우 OS에 종속적이고 설정 또한 윈도우 설정 창을 통해서만 가능하기에 권장할만한 옵션은 아니라 생각한다. 이에 따라 본 논문에서는 클라이언트 시스템과 서버 시스템 사이에 미들웨어를 두어 클라이언트가 네트워크 상에서도 서버 상의 애플리케이션을 활용할 수 있도록 하였으며 PYRO를 통하여 이러한 메커니즘을 구현할 수 있도록 하였다.

클라이언트에 구현된 COM 클라이언트는 미들웨어 PYRO에서 제공하는 Proxy를 통하여 네임 서버에 등록된(원격의 서버 상에 구현된) 객체를 마치 로컬에서와 같이 호출할 수 있다. COM 서버의 경우는 연산 처리를 위한 각종 메소드, 속성을 가지는 클래스 객체로서 PYRO 서버를 통하여 네임 서버에 등록된다. 이를 통하여 사용자는 네트워크 상의 어디에서라도 서버 상의 애플리케이션 객체를 호출하는 것이 가능하게 된다.

클라이언트 시스템에서 마치 로컬 상의 객체를 다루는 것 같은 위치 투명성을 구현하는 것이 분산 컴퓨팅의 골격이라 할 수 있으며 우리는 그를 위한 방법으로 PYRO를 활용하였다. 본 논문에서는 이를 씬-클라이언트 컴퓨팅에 적절히 활용하고자 하며 그에 대한 시스템 아키텍처를 그림 7의 다이어그램으로 나타내었다.

5.2. 분산 컴퓨팅 구현 및 테스트

본 섹션에서는 상기 부분에서 설계한 분산 컴퓨팅 아키텍처를 기반으로 실제 구동하는 애플리케이션의 한 예에 대해 살펴보기로 한다. 이를 위하여 네트워크를 경유하여 서버의 애플리케이션 패키지를 호출하는 클라이언트를 구현하였고 그에 대한 응답을 통하여 설계된 아키텍처를 검증하였다. 이를 위하여 서버 상에서 임의의 모듈 안에 사칙연산을 수행하는 애플리케이션을 클래스 형태로



(그림 7) 분산 컴퓨팅 구현을 위한 아키텍처 설계

작성하였고 이를 아래와 같이 시스템 영역에 구성하였다.

```
import sys
from win32com.server.exception import COMExc
  eption
import win32com.server.util

class NAServer:
        _reg_clsid_='{34D7C9EB-0541-4E4F-94D6-
        952B79310F95}'
        _reg_desc_='Testing Network Application
        Server'
        _reg_progid_='PythonServer.NAServer'
        _reg_class_spec_='comserver.NAServer'
        _public_methods_=['mul','add','sub','dev']

        def __init__(self): pass
        def mul(self, arg1, arg2): return
       arg1*arg2
        def add(self, arg1, arg2): return
       arg1+arg2
        def sub(self, arg1, arg2): return
        arg1-arg2
        def div(self, arg1, arg2): return arg1/arg2

def Register(pyclass=NAServer):
        from win32com.server.register import
        UseCommandLine
        UseCommandLine(pyclass)
.....
```

(그림 8) COM 기반의 클래스 객체를 통한 애플리케이션 구성

상기 과정을 거쳐 작성된 애플리케이션 객체는 COM 서버로서 시스템 상에 등록되고 타 클라이언트에서 접근할 수 있다. 네트워크 상의 타 시스템에서도 접근할 수 있도록 해당 클래스 객체는 PYRO 서버에도 이식되어야 하며, 아래와 같은 과정을 거쳐 네임 서버에 등록된다.

PYRO 미들웨어를 통하여 네임서버에 등록된 애플리케이션 객체는 네트워크 상의 타 시스템에서 접근 가능하여야 하며 이를 위하여 PYRO 클라이언트의 네임서버 접속이 선행되어져야 한다. 네임

```
import Pyro.naming
import Pyro.core
from Pyro.errors import PyroError,NamingErr    or
import NASmod

class    NAServer(Pyro.core.ObjBase,    NASmodN
AServer):
    pass

def main():
    Pyro.core.initServer()
    daemon = Pyro.core.Daemon()

    locator = Pyro.naming.NameServer
    Locator()
    print 'searching for Name Server...'
    ns=locator.getNS(host='16*.***.***.*
    **', port = 9090)
    daemon.useNameServer(ns)
    try:
        ns.unregister('NAServer')
    except NamingError:
        pass

    NASmod.Register()
    daemon.connect(NAServer(),'NAServer')
.....
```

(그림 9) 애플리케이션 객체의 네임 서버 등록

서버를 통하여 접근하고자 하는 객체의 URI정보를 획득하며 해당 객체에 대한 Proxy가 클라이언트에 생성된다. 이후 클라이언트는 Proxy를 통하여 네임서버에 등록된 서버의 애플리케이션 객체에 접근하는 것이 가능하며, 객체에 대한 메소드 호출을 로컬 상에서와 같이 수행하게 된다. 그에 대한 프로그램의 예를 클라이언트에서의 서버 측 프로그램 호출 결과와 함께 그림 7에 나타내었다.

6. 결 론

컴포넌트 기술과 분산 컴퓨팅의 발전은 소프트웨어 개발 및 활용에 있어서도 크나큰 변혁을 초래하였다. 단일 시스템 내에서 독립된 형태로 동작

```
import Pyro.naming, Pyro.core
from Pyro.errors import NamingError
locator=Pyro.naming.NameServerLocator()
print 'Searching Name Server...'
ns=locator.getNS(host='1**.***.***.***',
port = 9090)

print 'finding object'
try:
        URI=ns.resolve('NAServer')
        print 'URI:', URI
except NamingError,x:
        print 'Could\'t find object, nameserver
says:',x
        raise SystemExit

NAServer=Pyro.core.getProxyForURI(URI)

  ......

  ========================
========

Results..

Searching Name Server...
finding object
...
Request of Thin Client for our Server Application
is received successfully!!
>>>
```

(그림 10) 분산 객체 미들웨어를 통한 애플리케이션 객체 접속 및 호출

하던 기존 방식 하에서 애플리케이션은 그 자체가 서버이자 클라이언트였으며 외부에 독립된 상태에서 다수의 기능과 애플리케이션을 갖춘 Fat-Client 형태로 발전하였다. 필요한 기능을 자체적으로 구현하고 다수의 애플리케이션을 제공하는 특성으로 인하여 시스템 사양은 높아질 수 밖에 없으며 소프트웨어 유지 관리 과정에서 많은 번거로움이 발생하였다. 네트워크 기술의 발전으로 네트워크는 예전의 그 단순한 물리적 네트워크에서 벗어나 애플리케이션이 전달되고 구동시킬 수 있는 플랫폼으로 진화되었으며 단일 시스템 환경에서는 불가능하던 컴퓨팅 구현이 가능하게 되었다.

사용자 시스템에 없더라도 타 시스템에 구현된 애플리케이션을 호출함으로써 필요한 기능을 수행할 수 있게 되었고 이는 비로소 서버 기반의 컴퓨팅, Thin-Client의 형태로 나타나게 되었다. 서버 기반의 컴퓨팅에서 모든 애플리케이션과 데이터는 서버에 위치하며 클라이언트는 네트워크를 통하여 필요한 기능을 호출한다. 본 논문에서는 이러한 씬-클라이언트 컴퓨팅을 구현하기 위한 하나의 가능성으로서 컴포넌트 기술인 COM과 분산 컴퓨팅 미들웨어인 PYRO를 접목하였다. 이를 통하여 일목요연한 서버 애플리케이션 제작 및 구성을 수행하였고 네트워크를 통한 원활한 애플리케이션 호출이 가능하도록 하였다. 그리고 언급하였듯이 간략한 예제를 통하여 애플리케이션 구현의 가능성에 대해 진단해보았다.

시스템 구현에 있어 본 논문에서 서술한 방식의 한계점을 논의하자면 일단 윈도우즈 기반의 시스템이라는 점을 언급하고자 한다. 서버 애플리케이션 작성 및 구성에 있어 윈도우즈 기반 컴포넌트 기술인 COM을 활용한 바, 개발 언어인 파이썬과 미들웨어 PYRO 선택에 있어서도 일차적으로는 윈도우즈 환경을 기반으로 하였다. 하지만 리눅스의 경우, 별도의 라이브러리 작성 또는 CORBA 활용을 통하여 본 논문의 컴포넌트를 대체할 수 있으며 본 논문에서 설계한 아키텍처를 접목함으로써 리눅스 버전의 파이썬, PYRO 기반의 네트워크 컴퓨터 설계가 가능할 것이라 생각한다.

추가적인 보완점으로 Active Scripting을 통한 웹 브라우저 상에서의 애플리케이션 활용을 들 수 있다. 본 논문의 경우에는 서버 애플리케이션 의 제작과 사용자 호출 그리고 그를 통한 결과 도출에 위주로 논문을 전개하였지만, 이러한 메커니즘을 웹 페이지로 이식시킴으로써 사용자 편의 증대 및 인터페이스 통합 효과이라는 훌륭한 효과를 거둘 수 있을 것으로 생각한다. 또한 Active Scripting을 통한 애플리케이션 임베딩이웹 브라우저로 통합되는 네트워크 컴퓨터 구현을 위한 훌륭한 대안이기에, 향후 연구 목표로 부족함이 없다 생각한다.

참 고 문 헌

[1] White Paper, "Thin-Client/Server Computing", Citrix Systems, Inc, 1998.

[2] White Paper, "Remote Desktop Protocol (RDP) Features and Performance", Microsoft Corporation, 2000.

[3] J.Neih, S.J. Yang, and N.Novik. " A Comparison of Thin Client Computing Architectures", Technical Report CUCS-022 -00, Department of Computer Science, Columbia University, 2000.

[4] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper, "Virtual Network Computing", pp.Mobile Computing, 33-38, 1998.

[5] Eric Ly, Distributed Java applets for project management on the Web, IEEE Internet Computing, 1997.

[6] Andrej Volchkov, Server-Based Computing Opportunities, IT Pro, 2002.

[7] Mark Lutz, David Ascher, Learning Python, 2001.

[8] pyro.sourceforge.net

송 민 규 (Min-Gyu Song)

- 정회원
- 2001년 2월 : 강원대학교 전기학과 (공학학사)
- 2003년 2월 : 강원대학교 전자공학과 (공학석사)
- 2002년 12월 ~ 현재 : 한국천문연구원 전파천문연구부 연구원
- 관심분야 : 초고속네트워크, 분산 컴퓨팅, 리눅스 프로그래밍