ObjectPeerWork : 공유 객체 모델 기반의 피어투피어 어플리케이션 개발을 위한 프레임워크

(ObjectPeerWork : Framework for the Development of Peer-to-Peer Applications based on Shared Object Model)

강 운 구 † 왕 창 종 ††
(Un-Gu Kang) (ChangJong Wang)

요 약 본 논문에서는 공유 객체 모델 기반의 P2P(Peer-to-Peer) 애플리케이션을 개발하기 위한 프레임워크인 ObjectPeerWork를 설계 및 구현한다. 공유 객체 모델은 자원 관리 기능들을 자원에 포함시킴으로써 관리를 위한 컴퓨팅 파워의 저하를 막고, 보안 문제를 개선함으로써 공유 자원에 대한 신뢰도를 향상시킬 수 있다. 또한, 공유 객체 모델은 분산 컴포넌트 기반의 요청 중계 관리자 및 모듈 컨테이너를 통하여 확장이 가능한 모델이다. 이러한 공유 객체 모델에 기반한 ObjectPeerWork는 일반적인 P2P 모델의 문제점들을 개선하여 기업 내 정보시스템 구축과 컴퓨팅 파워의 분산 및 자원의 효율적인 관리를 가능하게 하는 프레임워크이다.

Abstract In this paper, we describe the design and implementation of ObjectPeerWork, which is a framework for the development of shared object model-based P2P(Peer-to-Peer) applications. The shared object model can prevent the computing power decrease on the way of resource management by incorporating the resource management function into resources themselves, and raise reliability on shared resources by improving the security problems. Also this model assures expandability by means of distributed component-based request broker manager and module container. The ObjectPeerWork based on this shared object model is a framework which makes the implementation of the enterprise information system possible, and makes distribution of the computing power and efficient resource management possible by improving the weakness in the general P2P model.

1. 서 론

인터넷과 PC의 발달은 정보의 분산과 공유를 가속화하였으며, 기존 클라이언트-서버 컴퓨팅 환경은 분산 다중 서버 환경으로 변화되었다. 또한 정보의 원천이 웹에서 데이타베이스, 그리고 개인 PC까지 확대되었고, 검색 포털의 한계, 즉 정보 탐색 범위의 제한, 부정확성, 불충분한 개인 욕구 만족, 포맷의 다양성 등으로 인해 사용자들은 다른 대안을 찾기 시작했고, 사용자 스스로 컴퓨터의 자원과 서비스를 시스템 간의 직접적인 교환을 통해 상호 공유하는 P2P(Peer-to-Peer) 컴퓨팅이 주목받게 되었다[1,2].

한 예로 MP3 음악 파일을 공유하기 위한 냅스터 서비스는 중앙의 서버에는 실질적인 파일을 가지고 있지 않고 인터넷상의 PC에 분산된 파일의 정보만을 제공하며 공유는 개인 PC간에 이루어지도록 하는 모델로 대중적인 인기를 끌고 있다[1,3].

그러나 이와 같은 P2P 서비스의 문제점은 인증되지 않은 불특정 다수를 대상으로 함으로써 데이타의 신뢰도를 보장할 수 없으며, 중앙 관리가 불가능하기 때문에 정보에 대한 저작권 문제 등에 적극적으로 대응할 수 없다[4].

또한 MP3와 같이 중복이 많고 보편화된 데이타가 아

† 정 회 원 : 가천길대학 뉴미디어과 교수
ugkang@gcgc.ac.kr
†† 종신회원 : 인하대학교 전자계산공학과 교수
cjwangse@dragon.inha.ac.kr
논문접수 : 2001년 7월 5일
심사완료 : 2001년 9월 22일

는 특정한 자원에 대해서는 자원을 가지고 있는 PC에 네트워크 부하가 가중될 수 있으며 PC가 꺼져있을 경우에는 자료를 전송 받을 수 없다는 문제점이 있다. 또한 현재 단순한 파일에 기반 한 공유 서비스만을 제공하고 있어 바이러스 배포의 새로운 경로가 되고 있으며 이러한 문제들로 인해 기업 내 정보시스템에서 P2P를 기반 한 서비스 구축에는 적합하지 않은 것으로 평가되고 있다. 그리고 이와 같은 문제점 이외에도 P2P 기반 네트워크를 이용해 보다 효율적인 자원 공유가 가능한 표준화된 모델의 필요성이 대두되고 있다[1,5,6].

본 논문에서는 P2P 방식이 가지는 장점을 살리면서 위와 같은 문제점들을 해결할 수 있는 공유 객체 모델을 제안하고, 공유 객체 모델에 기반 한 P2P 애플리케이션 개발 프레임워크인 ObjectPeerWork(Object based Peer-to-peer applications development frame Work)를 설계 및 구현한다.

제안한 공유 객체 모델은 파일 혹은 URL이 아닌 객체(Object)를 단위로 관리함으로써 분산된 네트워크상의 컴퓨터들간에 다양한 형태의 응용 및 관리가 가능하며 기업 내 정보시스템에서도 활용할 수 있다. 또한 P2P 네트워크를 보다 효율적으로 운영하기 위한 서비스 모델과 하드웨어, 오프라인 연계 서비스, 컨텐츠 등 다양한 자원을 공유할 수 있는 운영 환경을 제공한다.

2. P2P 기술 분야 고찰

이 장에서는 P2P와 관련된 기술 분야에 대해 고찰한다. P2P와 관련된 연구 분야는 초기 단순한 파일 공유에서 출발하여 현재는 메타데이타, 성능, 신뢰성, 책임성, 보안, 상호운용성 등 광범위한 분야로 확장되었다[7]. 이러한 P2P 연구 분야는 다시 객체관리 및 상호운용성 관련, 프로토콜 관련, 서비스 관련, 네트워크 관련 연구로 구분할 수 있다[1,5].

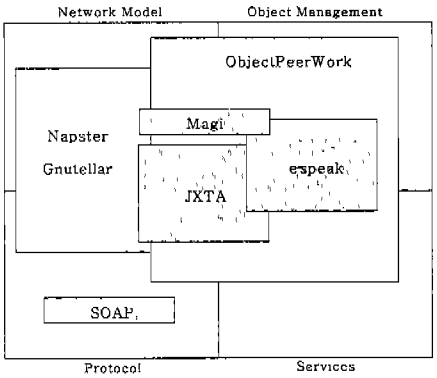

그림 1 P2P 관련 연구 분야

[그림 1]은 현재 연구되고 있는 P2P 분야의 기술들을 네트워크, 객체관리, 프로토콜, 서비스에 대한 연관 관계를 보여주고 있다.

2.1 객체관리 모델

객체관리 모델은 단지 객체만을 관리하는 순수 객체관리 모델과 프로토콜 기반 객체관리 모델, API 기반 객체관리 모델로 구분할 수 있다[5,8].

순수 객체관리 모델은 초기 P2P 모델에 적용된 방식으로 냅스터, 누텔러 등에 채택되어 P2P의 가능성을 보여준 모델이다. 공유 자원이 파일로 한정되며 모든 관리와 책임을 피어에서 처리하도록 했으며 아키텍처적인 접근이 아닌 애플리케이션 모델이므로 관리에 있어 많은 문제점과 한계를 가지고 있다. 냅스터는 일종의 하이브리드 방식인 중계 서버(Broker Mediated Server) 모델이며, 누텔러는 순수 P2P 환경에서의 자원 공유를 위한 브로드캐스팅 방식의 순수(Pure) P2P 모델이다[1,9].

프로토콜 기반 객체관리 모델은 상호운용성에 중심을 둔 모델로 프로토콜을 통한 상호운용성을 제공한다. 대표적인 모델에는 Magi가 있는데 인터넷 비즈니스를 위한 메세징과 서로 다른 크로스 플랫폼에 있어서의 배포를 지원하는 프레임워크 구조로, 구현 및 프로토콜의 표준 기반 위에 상호운용성을 지원하는 오픈 아키텍처이다. 하지만 컨테이너의 명세 변경 시 모든 피어에 대한 일관성 유지가 어려우며 공유되는 자원에 대한 구체적인 모델이나 형태에 대해서는 언급이 없으며 단지 HTTP를 통해 서블릿 엔진을 이용한 이벤트 처리에 대해서만 구현이 되어 있다[5,10,11].

API 기반의 객체관리 방식은 현재 연구되고 있는 대부분의 P2P 아키텍처 관련 연구들로써, 대표적인 연구로는 HP의 e-speak와 Sun의 JXTA가 있다. e-speak는 웹 기반 서비스의 개발, 배포, 동적 상호작용을 위해 설계된 개방형 플랫폼으로 일종의 서비스 컨테이너라 할 수 있다. e-speak 엔진을 통해 등록된 서비스들은 API(JESI)를 통해 구현된 애플리케이션에서 접근이 가능하다. [그림 2]는 e-speak의 구조로 API 라이브러리를 통해 자원을 관리하며 e-speak 코어가 공유자원 운영에 대한 전반적인 중계를 담당하는 구조이다[6].

e-speak의 특징은 P2P 서비스의 대상이 되는 자원을 다양하게 지원할 수 있다는 것과 B2B를 대상으로 한 P2P 모델이라는 점이다. 또한 P2P 네트워크의 구성 모델 및 공유객체의 교환에 대한 네트워크 객체 모델(Network Object Model)을 제안하고 있다. 반면 JXTA

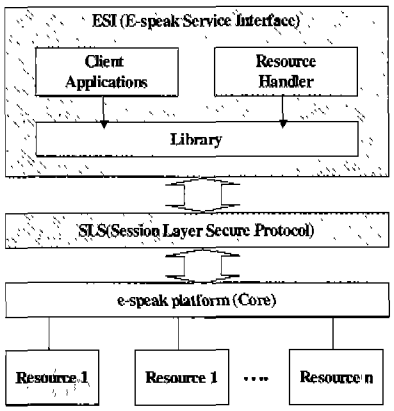

그림 2 e-speak 구조

는 P2P 애플리케이션 용 플랫폼으로 웹 기반 프로그래밍 언어이자 컴퓨팅 플랫폼이라고 할 수 있다. [그림 3]은 JXTA의 구조로 JXTA 기본 서비스 및 애플리케이션에 API를 기반으로 확장이 가능한 프레임워크 모델로 되어 있음을 알 수 있다.

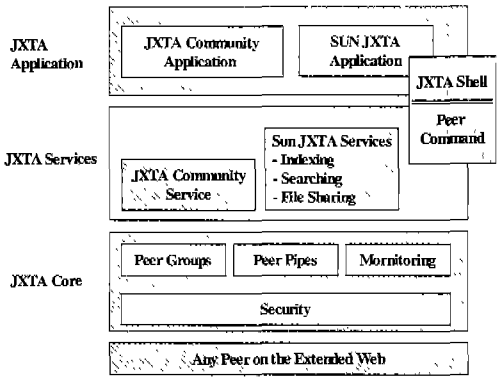

그림 3 JXTA 구조

2.2 순수 P2P 네트워크 모델

순수 P2P 모델은 이름 그대로 피어와 피어 만으로 이루어진 모델이다. P2P 본래의 취지를 가장 잘 반영하고 있지만 현실적으로는 한계가 많은 모델로 공유자원에 대한 검색 및 성공적인 자원 이용에 성능 저하가 불가피하다. 이론적으로는 이 모델의 피어들이 서로 상대에 대한 공유 자원을 검색할 경우 피어의 개수가 n 이라고 한다면, 총 검색 횟수는 n*(n-1)이 되므로 피어 증가에 따라 기하 급수적인 성능 저하를 가져올 수 있다. 이러한 문제점을 보완한 모델이 누텔러로, 누텔러 프로토콜을 통해 효과적인 자원 검색 방식을 제공한다 [1,4].

2.3 하이브리드 모델

이 모델은 순수 P2P 모델에 여러 가지 목적으로 피어와 역할이 다른 서버를 두는 방식으로 중계 서버 모델(Broker Mediated Server Model) 이라고도 한다[1].

대표적으로 냅스터를 들 수 있으나 냅스터는 중앙의 검색 서버만을 두고 있기 때문에 부가 기능이 미약하고 검색 및 회원 급증에 따른 네트워크 및 서버의 과부하 문제를 안고 있는 모델이다. [그림 4]는 중계 서버 기반 P2P 모델의 기본 구조로 중앙의 서버를 통해 로그인과 검색서비스를 운영하는 단순한 구조이다.

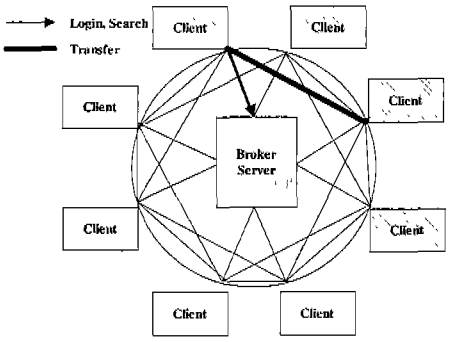

그림 4 중계 서버 모델

3. 공유 객체 모델 제안

이 장에서는 기존 P2P 모델에서의 여러 가지 문제점들을 해결하기 위해 공유 자원을 자원과 자원을 다루는 기능을 모듈로 묶어 객체화하여 관리함으로써 유연성과 확장성을 극대화할 수 있는 모델을 제안한다.

3.1 공유 객체 모델의 자원

공유 객체 모델에서 다루고 있는 자원은 P2P 네트워크 상에 분산되어 있는 파일 및 하드웨어적인 환경과 온라인 혹은 오프라인과 연계된 서비스를 말하며 다음과 같은 공통적인 특징을 가지고 있다.

- 무수히 많은 자원이 있다.
- 자원들은 서로 신뢰할 수 없고 서로 다른 개인에 의해 소유되고 관리된다.
- 자원들은 어느 순간 사용할 수 없는 상태가 될 수 있는 속성을 내재하고 있다.
- 서로 다른 자원은 각기 다른 보안 및 정책을 요구한다.

- 자원들은 이질적인 성질을 가진다.
- 자원들은 서로 다른 환경의 네트워크에 연결되어 있다.
- 자원을 관리하는 방법이 서로 다르다.
- 자원들은 대부분 물리적으로 서로 떨어진 공간에 위치한다.

자원이 가진 이러한 특징들로 인해 다양한 분산 자원을 공유하기 위해서는 자원에 대한 체계적인 분류가 필요하며 이러한 분류를 통해 표준화된 인터페이스 모듈을 제공할 수 있게 된다. 본 논문에서는 표준화된 인터페이스 모듈을 제공하기 위해 공유자원을 [그림 5]와 같이 앞에서 나열한 자원의 특징에 따라 오프라인, 하드웨어, 컨텐츠, ASP(Application Service Provider), 파일공유 등 5개의 범주로 분류하였다.

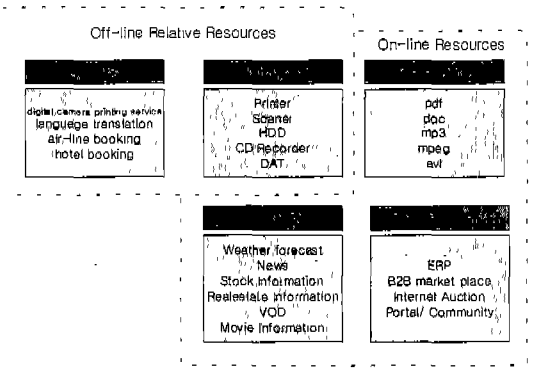

그림 5 공유 객체 모델 자원 범주

이렇게 분류된 자원들은 각각의 유형별로 자원을 처리할 수 있는 표준화된 인터페이스 모듈을 통해 관리하게 된다. 자원에 대한 인터페이스 모듈은 모듈 컨테이너에 위치하며 새로운 자원 유형의 확장 시 컨테이너에 추가해 서비스 할 수 있다.

3.2 공유 객체 클래스

공유 객체 클래스(SOC: Shared Object Class)는 공유 객체 모델의 핵심 요소로 [그림 6]과 같이 공유하고자 하는 자원, 플러그인이 가능한 함수, SOC에 대한 객체 기술서(Object Descriptor)로 구성된다.

자원을 SOC라고 하는 새로운 객체로 재정의 하는 이유는 앞에서 정의한 것처럼 자원의 종류가 다양하고 각각의 운영환경이 상이하며 관리방법 또한 다양하기 때문이다. 따라서 자원을 관리하기 위한 기능들을 자원에 포함시켜 함께 배포함으로써 자원 관리를 위해 컴퓨팅

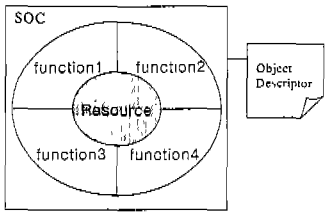

그림 6 SOC의 구조

파워가 저하되는 것을 막고 자원에 대한 보안 문제를 해결해 신뢰도를 향상시킬 수 있다.

SOC는 객체직렬화(Serialization)가 가능한 클래스로서 서비스의 대상이 되는 자원은 SOC에 첨부 될 수도 있으며 원격지에 위치할 수도 있다. 따라서 자원에 대한 위치 정보는 객체 기술서(Object Descriptor)에 객체의 접근 정보를 표시하는 URI(Uniform Resource Identifier)로 기술된다. SOC 에서 URI를 통해 자원을 관리하는 이유는 자원의 소유주 및 작성자, 컴퓨터에 대한 정보를 유지하면서 컴퓨팅 및 네트워크 환경에 따라 자원을 분산 운영하기 위함이다.

객체 기술서는 SOC에 대한 설명으로서 XML로 표현되며 표준화된 자원관리를 위한 정보와 현재 SOC를 구성하고 있는 메소드에 대한 정보를 가지고 있다. [그림 7]은 객체 기술서 중 헤더에 해당하는 부분의 XML DTD를 정의한 것이다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
xmlns="http://www.ObjectPeerWork.net/schema/header"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
<xsd:annotation>
<xsd:documentation>ObjectPeerWork XML request
header</xsd:documentation>
</xsd:annotation>
<!-- used to specify how messages should be routed to a
destination -->
<xsd:complexType name="addressType">
<xsd:sequence>
<xsd:element name="messageID" type="xsd:String"/>
<xsd:element name="sessionID" type="xsd:String"/>
<xsd:element name="serviceName" type="xsd:String"/>
</xsd:sequence>
<xsd:attribute name="encoding" type="xsd:string"/>
</xsd:complexType>
<xsd:element name="route">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="to" type="addressType"/>
<xsd:element name="from" type="addressType"/>
</xsd sequence>
</xsd:complexType>
</xsd:element>
```

그림 7 객체 기술서 메시지 헤더 XML DTD

4. ObjectPeerWork 설계 및 구현

이 장에서는 3 장에서 제안한 공유 객체 모델을 기반으로 한 P2P 애플리케이션 개발 프레임워크인 ObjectPeerWork의 전체 구조를 계층적으로 설계하고, 각 계층에 속하는 모듈들을 설계 및 구현한다.

4.1 설계 개요

ObjectPeerWork의 목적은 P2P 네트워크 환경에서 특정운영체제, 프로토콜, 데이타베이스 등에 영향을 받지 않고 파일의 공유를 포함한 다양한 서비스(예를 들면 네트워크, 저장장치, 프린터, 오프라인 연계서비스 등)를 객체기반의 자원 개념으로 다룰 수 있는 방법을 제공하는 것이다. 이를 위해 ObjectPeerWork는 자원을 공유 자원 클래스(SOC: Shared Object Class)라고 하는 새로운 형태의 클래스로 정의하고 읽기, 수정, 재 배포 및 모니터링지원, 보안을 비롯한 다양한 부가기능을 제공한다. 또한 ObjectPeerWork는 분산 P2P 서비스 운영 프레임워크 및 네트워크 모델인 향상된 중계 서버 기반 P2P 모델이며 여러 애플리케이션에서 ObjectPeer Work를 지원할 수 있는 API를 제공한다.

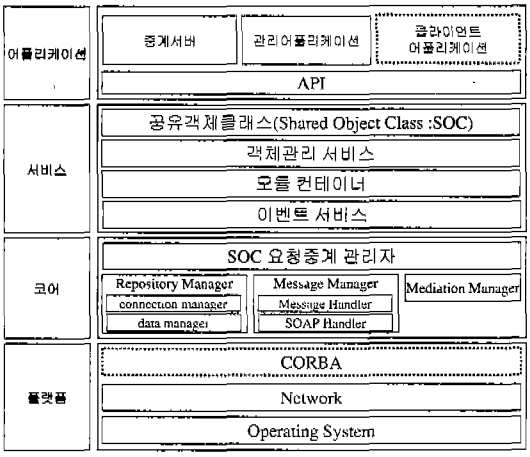

그림 8 ObjectPeerWork 구조

ObjectPeerWork는 [그림 8]과 같이 SOC와 SOC를 관리하기 위한 객체관리자(Object Manager) 및 애플리케이션을 위한 API를 제공함으로써 향상된 중계 서버 기반 P2P 모델의 운영을 위한 중계 서버 및 관리 서버 그리고 클라이언트용 애플리케이션의 개발 및 운영이 가능한 개방형 아키텍처이다. 실제적인 SOC의 교환은 ObjectPeerWork 코어에 위치한 SOC 요청 중계 관리자(Request Broker Manager)가 담당하게 되고 메세징 서브 시스템인 메시지 관리자 (Message Manager)와 파일시스템, 데이타베이스 등의 운영을 담당하는 저장소 관리자(Repository Manager)를 통해 서비스가 운영된다. 또한 제안된 모델은 객체들의 상호운용성을 위해 RMI over IIOP를 통해 미들웨어로 CORBA(Common Object Request Broker Architecture)를 지원하며 다양한 하드웨어와 운영체제에서의 손쉬운 프로그래밍 환경을 제공하기 위해 Java를 기반으로 ObjectPeerWork API를 제공한다.

4.2 객체 관리 서비스

객체 관리 서비스는 실질적으로 SOC를 관리하는 부분으로 SOC의 생성부터 소멸까지를 담당하는 부분이다. 객체 관리 서비스가 제공하는 주요 기능은 SOC에 대한 생성, 등록, 수정, 삭제 및 등록 해제, 복제, 이동 등이며 객체 기술서의 작성도 이 부분에서 이루어진다. 또한 SOC에 포함되어 있는 자원을 어플리케이션에서 사용할 수 있도록 해주며 모듈 컨테이너(Module Container)를 통해 제공되는 부가 기능들을 사용할 수 있다. 물론 SOC에 포함된 메소드를 사용할 수 있도록 해주는 부분도 객체 관리 서비스의 역할이다.

[알고리즘1]은 공유자원을 입수한 클라이언트가 SOC를 사용하기 위한 알고리즘으로 SOC에 자원이 포함되어 있는 경우와 원격지 참조의 경우를 처리하는 과정이다.

알고리즘 1 SOC를 통한 공유자원 처리

```
begin Method
  select SOC;
  open SOC descriptor;
  if ACL exist then Vaildate Permission;
       if ValidUser then
         read resource type;
         case {
            embeded : open Resource;
            remote  :
                 send request message to remote SOC Application;
                 if available then request resource;
                    open remote resource;
                 else
                    add resource to mytodolist at Management
                    Application;
                 end if
         }
       end if
  end if
  if additional function requered then
       if remote function then
          reference remote method;
       end if
       use additional functions;
  end if
  if usage charge required then
       send usage to resource owner;
  end if
end Method;
```

4.3 모듈 컨테이너

모듈 컨테이너(Module Container)는 객체 관리 서비스에서 SOC를 처리하기 위한 부가 기능이 위치하는 곳으로 기존 P2P 모델의 문제점들을 해결 할 수 있는 구조이다. 모듈 컨테이너는 객체 관리 서비스에서 SOC를 관리하기 위한 모듈이 위치하는 컨테이너 구조로서, 컨테이너에 기본적으로 제공되는 모듈은 사용자 인증 및 권한 정보를 관리하는 ACL(Access Control List) 모듈, SOC에 포함된 원격 자원을 관리하기 위한 원격 자원 관리자, 메타데이타 관리자 및 SOC 내장 혹은 원격 메소드를 관리하는 함수 관리자(Function Manager)가 있다. [그림 9]는 모듈 컨테이너 내에 위치한 ACL 모듈의 자바 프로그래밍 인터페이스이다.

```
Public interface ACLManagernterface
{
    public static String registerUser(UserProfile up) throws
    SOInvocationException,
    public static boolean unregisterUser (AuthInfo authInfo,
    String accountName)
                throws SOInvocationExcception;
    public static boolean setUserProfile(AuthInfo authInfo,
    UserProfile up)
                throws SOInvocationException;
    public boolean addDescription(AuthInfo authInfo,String
    accountName,AttributeSet as)
                throws SOInvocationException;
    public static String[] getAllUsers() throws
    SOInvocationException;
}
```

그림 9 ACL 모듈의 인터페이스

4.4 이벤트 서비스

ObjectPeerWork에서 자원 공유 및 원격 피어간의 통신은 기본적으로 메시지 기반이다.

이벤트 서비스(Event Service)는 비동기적으로 메시지를 전송하기 위한 구조를 제공한다. 이것은 P2P 네트워크의 특성상 동기화 보다는 비동기화 된 요청/응답 구조가 적합하기 때문에 비동기(Asynchronous) 컴퓨팅을 지원하기 위해서 필요한 부분으로 메시지의 브로드캐스팅을 하기 위해서도 필요한 부분이다.

ObjectPeerWork는 비동기 방식과 동기화 방식을 함께 사용하며 P2P의 장점과 중계 서버 모델의 장점을 최대한 활용하고 있다. 이벤트 서비스는 ObjectPeer Work 코어의 메세징 서브 시스템인 메시지 관리자와 유기적으로 운영되며 푸쉬(Push) 및 풀(Pull) 모델을 통해 애플리케이션 객체와 이벤트를 주고 받는다.

4.5 SOC 요청 중계 관리자

SOC 요청 중계 관리자(Request Broker Manager)는 ObjectPeerWork의 중심에 위치한 코어 모듈로서 원격 피어의 SOC 요청을 받아들이고 애플리케이션과 커뮤니케이션하기 위한 인터페이스로서 ObjectPeerWork 애플리케이션 사이의 모든 ObjectPeerWork 메시지를 교환한다.

알고리즘 2 SOC 요청 중계 관리자의 ObjectPeerWork 메시지 처리 알고리즘

```
begin Method
    read MessageHeader;
    case MessageType is SOC
          if MessageHeader is incomming then
            send SOC to Object Manager Service;
          create PUSH event;
            else if MessageHeader is outggoing then
                        calculate routing information;
                        insert routing information to
                        MessageHeader;
                        send Message to outgoing queue;
          end if
      case MessageType is Repository then
            send Message to Repository Manager;
      case MessageType is SOCManager then
              read mediation server location;
              caclutate routing information;
            send Message to Mediation Manager;
end Method
```

SOC 요청 중계 관리자는 파일시스템, 데이타베이스, 메모리DB와 같은 자료저장소를 관리하기 위한 저장소 관리자, 메세징 서브 시스템인 메시지 관리자, 중계 서버와의 메세징을 위한 중계 서버 관리자로 구성된다. [알고리즘 2]는 ObjectPeerWork 메시지를 처리하기 위한 요청 중계 관리자의 동작 과정이다

[그림 10]과 같이 ObjectPeerWork 기반의 애플리케이션들은 JRMP(Java Remote Method invocation Protocol)를 통해 메시지를 전달하고 CORBA 애플리케이션들은 RMI-IIOP를 통해 메시지를 전달한다. 메시지 핸들러에 의해 바인딩 된 메시지들은 도착 큐(incomming queue)에 쌓이게 되고 SOC 요청 중계 관리자는 ObjectPeerWork 이벤트 서비스에 푸쉬 이벤트를 보내 애플리케이션에서 메시지를 참조할 수 있도록 제공한다. 반대로 애플리케이션에서 원격 피어 클라이언트로 보내는 메시지는 풀 이벤트를 통해 전송 큐(outgoing queue)에 저장되고 이들은 다시 메시지 핸들러에 의해 브로드캐스팅 되거나 목적지 클라이언트까지 전달되게 된다.

ObjectPeerWork 자원 관리자는 ObjectPeerWork 애플리케이션에서 각종 데이타를 관리하는 모듈로 애플리케이션과 동일한 컴퓨터나 원격지에 위치 할 수도 있다.

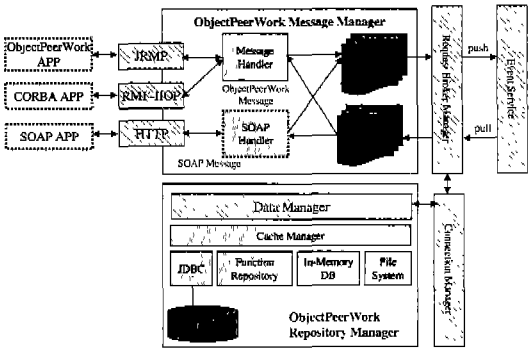

그림 10 ObjectPeerWork 요청 관리자

애플리케이션 에서는 연결 관리자(connection manager)와 데이타 관리자(data manager)를 통해 데이타베이스, 함수 저장소, 메모리DB 및 파일 시스템에 접근하게 된다.

4.6 향상된 중계 서버 P2P 모델

순수한 P2P 네트워크 환경의 문제점을 해결하고 보다 다양한 서비스를 제공하기 위해 본 논문에서는 향상된 중계 서버 P2P 모델(Advanced Broker Mediated P2P Model)을 제안한다. 본 모델은 다양한 기능을 가지는 중계 서버(Mediation Server)를 두어 기존 P2P 서비스의 문제점을 해결하면서 순수 P2P의 장점을 살릴 수 있도록 설계하였다.

특히 공유 자원에 대한 정보를 중앙의 서버에서 관리하는 것이 아니라 분산된 중계 서버에서 관리하므로 사용자 증가에 따른 네트워크 부하 및 자원 데이타베이스의 거대화에 따른 비효율성을 줄 일수 있다. 제안된 모델은 운영의 효율성 극대화는 물론 네트워크 과부하를 최대한 줄일 수 있도록 설계되었으며 [알고리즘 3]은 이러한 목적을 달성하기 위한 중계 서버 분산 알고리즘으로, 중계 서버의 운영 중 자원 검색 요청에 대한 처리 과정을 보여 준다.

이러한 분산 알고리즘을 기반으로 한 중계 서버 간 질의(Query) 브로드캐스팅 전략은 다음과 같다. [그림 11]은 중계 서버의 구조이며 각 요청을 보내는 클라이언트의 배치 순서와 서버 테이블의 관계를 도식화 한 것이다. [그림 11]에서 Nm은 호스트의 IP를 의미하며, 오른쪽의 개수는 각 레벨의 생성 요청 수를 의미한다. 수행 절차는 애플리케이션에서의 요청과 함께 요청 객체(request object)가 생성되고 level0일 때 자신을 복제해 N2 호스트로 이동한다. level1일 때 N1과 N2에 요청을 복제하고 N3와 N4 호스트로 각각 이동한다. level3일 때는 N1, N2, N3, N4 의 요청 객체가 자신을

알고리즘 3 분산 중계 서버 검색 운영 알고리즘

```
begin Method
    search query on local database;
    if not found then
        broadcast query to all alive Mediation Servers;
        create asynchronous receiver thread;
        receive query results;
        if not exist Mediation Server on same subnet then
            regist this server to Mediation Server list;
            broadcast server information on subnet;
        end if
    else
        return found result;
        if client request more result then
            broadcast query to all live Mediation Servers;
            create asynchronous receiver thread;
        end if
    end if
end Method
```

복제하고 N5, N6, N7, N8로 이동을 하게 된다. 이렇게 레벨이 증가하게 될 때마다 2의 배수로 증가하게 되고 모든 중계 서버에 대해 요청이 전달 될 수 있다.

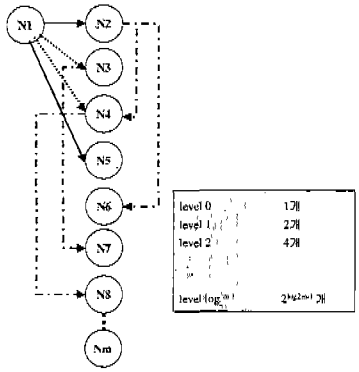

그림 11 분산네트워크에서의 단계별 요청 전달

5. 실험 및 고찰

실험은 ObjectPeerWork의 운영과 아키텍처의 검증을 위한 애플리케이션의 구현과 향상된 중계 서버 P2P 모델에 대한 성능 평가, 그리고 기존 연구와의 종합 평가로 구성하였다.

5.1 향상된 중계자 서버 P2P 네트워크 성능 평가

실험은 클라이언트의 증가에 따른 정보 검색 시간 및 네트워크 효율성에 대한 측정으로 순수 P2P 모델 및 하이브리드 P2P 모델과 본 논문에서 제안한 향상된 중계 서버 모델을 비교하였다. 다음은 실험을 위한 변수들이다.

C_n : 클라이언트의 개수 (node 수)
M_n : Mediation Server 의 개수
bw : 네트워크 대역폭
P_s : 메시지 패킷사이즈
bw_{rate} : 네트워크 사용에 따른 가중치
N_{start} : 최초 정보 검색 횟수
N_{after} : 최초 정보 검색 후 재 검색 횟수
N_{tot} : 총 검색 횟수
ti_{trans} : request 가 ObjectPeerWork server 에 전송되는 시간
ti_{start} : 최초 정보 검색에 소요되는 시간
ti_{work} : 전체 request 가 수행되는 시간
ti_{due} : 단위 호스트별 등록 소요 시간
ti_{ret} : 처리 결과를 받는데 까지 걸리는 시간
ti_{sum} : 단위 호스트별 처리 결과를 종합하는 시간

실험을 위해서 다음과 같이 각 변수들을 설정 하였다.

- C_n = 50,100,200,500,1000,2000,3000,4000,5000
- M_n = 4
- bw = 256k
- bw_{rate} = 100
- P_s = 10k
- 각호스트를 ObjectPeerWork 서버에 등록하는 시간
  $ti_{reg} = (ti_{start} + ti_{due}) \log_2^m$
- 모든 작업 request 가 ObjectPeerWork server 에 전송되는 시간
  $ti_{all} = n(1/2 + 1/4 + 1/8 + ... + 1/(2^{\log_2^m})) * ti_{trans}$
- 처리결과를 받고 통합하는 시간
  $ti_{final} = (ti_{ret} + ti_{sum}) \log_2^m$

이때 최초 정보검색을 위해 소요되는 커뮤니케이션 횟수 N_{start}는 다음과 같다.

$$N_{start} = (C_n / M_n + (M_n - 1)) * M_n$$

최초 정보 검색이 이루어진 후 클라이언트는 가장 가까운 중계 서버를 통해 정보 검색이 가능 하므로 N_{after} 는 $N_{after} = C_n$이 된다.

실험에서는 C_n 모두가 자원을 찾기 위해 한번의 검색만을 수행하는 것으로 가정했으며 네트워크 대역폭 bw 는 256k 로, 검색에 필요한 메시지 크기 P_s 는 10k로 설정하고, 순수 P2P 모델의 경우 C_n = 500 인 경우까지만 측정하였다. 또한 대역폭이 초과될 경우 메시지 전송 지연에 따른 성능 저하의 가중치 bw_{rate} 은 100으로 가정하였다.

모델별 수행시간 성능 평가는 동일 환경에서 다음과 같은 수식으로 실시하였다.

- 순수 P2P 모델
  $ti_{work} = Cn * (Cn-1) * ti_{sum} + ti_{final}$
- 단순 중계 서버 모델
  $ti_{work} = Cn * ti_{sum} + ti_{final}$
- 향상된 중계 서버 모델
  $ti_{work} = (C_n * ti_{sum}/M_n / M_n + (M_n - 1)) * M_n + ti_{final}$

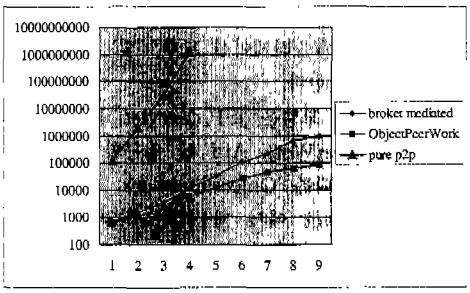

그림 12 네트워크 성능 테스트

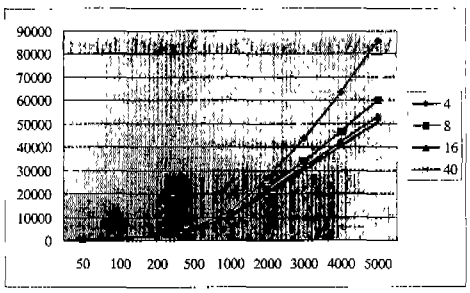

그림 13 중계서버 증가에 따른 성능 테스트

실험 결과 [그림 12]와 같이 C_n 증가에 따라 일반적인 중계 서버 방식은 일정 수준을 넘어서면서부터는 증가폭이 커지는 것을 볼 수 있었으며 제안된 방식은 C_n 증가에 따라 완만한 증가곡선을 나타내었다. 따라서 정보 검색에 대한 성능 향상에 순응한다는 것을 보여 줌으로써 P2P 네트워크 환경에서 네트워크 과부하를 줄일수 있을 것으로 기대 된다.

또 다른 실험으로 중계 서버의 증가에 따른 성능 변화를 측정하기 위해 위의 결과에 중계서버의 수인 M_n을 평균 C_n 의 2%로 유지하고 실험한 결과 [그림 13]과 같은 그래프를 구할 수 있었다.

실험결과 C_n 이 일정 수준 이상이 될 때까지는 M_n의 증가에 따른 성능 향상이 나타나지 않지만 일정 수준 이상이 되면 상당한 성능 향상이 있음을 알 수 있다. 그러나 C_n 이 다시 어느 수준에 이르게 되면 M_n 의 증가에 따른 성능 향상이 둔화되어 적정 수준의 M_n 을 유지할 수 있도록 설정하는 것이 제안된 모델이 풀어야 할 과제라는 것도 알 수 있었다.

5.2 ObjectPeerWork 애플리케이션의 구현

제안된 모델의 검증을 위한 실험으로 중계 서버가 구

현되어 있다는 가정에서 ObjectPeerWork API 와 ACL 함수를 가지고 권한 설정이 가능한 파일 공유 P2P 클라이언트를 개발하였다. ObjectPeerWork API는 [그림 14]와 같이 웹 애플리케이션 모델 및 표준 애플리케이션 모델을 지원한다.

웹 애플리케이션 모델은 JSP, Servlet, EJB session bean 등 ObjectPeerWork API 에 접근할 수 있는 게이트웨이를 통해 다른 ObjectPeerwork 애플리케이션 및 Mediation Server 에서 제공되는 다양한 서비스를 이용 할 수 있다.

표준 애플리케이션 모델은 제공되는 API를 통해 직접 ObjectPeerWork 서비스를 이용 할 수 있다. 본 논문에서는 실험을 위해 표준 애플리케이션 모델을 이용해 개발하였으며, 응용 프로그램은 자바의 스윙 애플리케이션으로 ObjectPeerWork.client.ClientAppIntf Interface를 구현하였다.

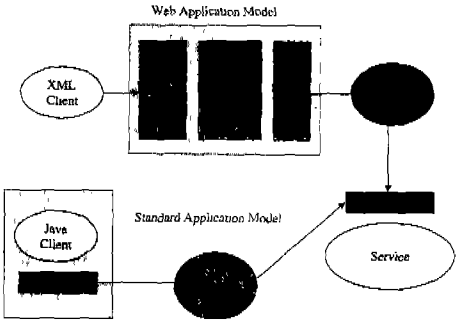

그림 14 SOC 프로그래밍 모델

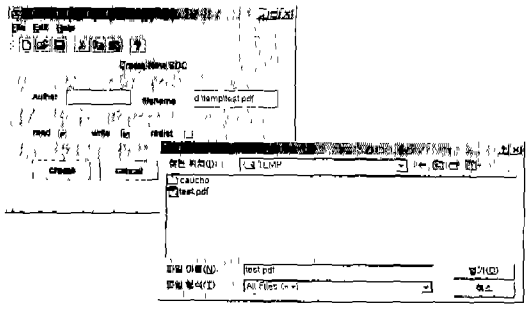

그림 15 권한설정이 가능한 파일공유 P2P 클라이언트

[그림 15]는 구현된 ObjectPeerWork 클라이언트를 사용하여 새로운 SOC를 생성하는 화면으로써 특정 파일에 대해 권한을 설정하고 SOC로 바인딩 하도록 되어 있다.

다음은 [그림 15]의 클라이언트 애플리케이션에서 사용된 SOC 생성과 관련된 코드로 ClientAppInterf 인터페이스를 구현하고 있다.

ClientApp.java

```
import ObjectPeerWork.client.*;

public class ClientAppImpl implements ClientAppInterf {
            SOClass soclass;
            SOConnection connection;
            SODescriptor sodesc;
            public ClientAppImpl(String author, String title, String
            filename) {
                soclass = new SOClass();
                soclass.setAuthor = author.
                soclass.setTitle = title;
                connection = new SOConnection();
                makeSOC(filename);
                setACL(soclass.socId);
                connection.transSOC(soclass,sodesc);
                soclass.close();
                connection.close();
            }
            void setACL(String socid) {
              soclass.soacl.write = true;
        soclass.soacl.read = true;
              soclass.soacl.redist = false;
            }
    void makeSOC(String filename) {
      try {
              File file = new File(filename);
      }
      catch(FileNotFoundException e){
              errlog("error : "+e, socclass.getID());
      }
       try {
            FileInputStream inf = new InputStream(file);
            byte[] contents = new byte(file.length());
          FileInputStream fis = new FileInputStream(inf);
          fis.read(contents);
       }
       catch(IOException e) {
                    errlog("error :"+e, socclass.getID());
          }
       soclass.setSOData();
       soclass.sodata.setName =  filename ;
       soclass.sodata.setSize = file.getSize();
       soclass.sodata.setcontents(contents);
       soclass.write();
    }
}
```

실험 결과 SOC가 정상적으로 생성 및 배포되었으며 배포된 SOC에 대해서는 바이러스 감염과 같은 파일 변화에 대처할 수 있었다. 또한 ACL 함수를 사용해 사용자, 컴퓨터, 네트워크, 읽기, 쓰기에 대해 제어가 가능했다.

[표 1]은 ObjectPeerWork의 실험 결과와 기존 5가지 P2P 모델들을 비교 분석한 도표이다. 비교 항목은 일반적인 P2P 아키텍처에 대한 요구사항을 기준으로 하였다.

표 1 P2P 응용의 비교

구분 \ 모델	Gnutellar	Napster	Magi[IN]	JXTA	e-speak	ObjectPeerWork
객체기반 공유자원 관리	X	X	△	○	○	○
개방형 아키텍처	X	X	○	○	○	○
자원관리를 위한 플러그인 기능지원	X	X	X	△	△	○
프로토콜 의존도	높음	낮음	높음	높음	보통	보통
공유자원의 범위	좁음	좁음	보통	넓음	넓음	넓음
확장성과 투명성	낮음	낮음	보통	높음	높음	높음
API 제공 유무	X	X	△	○	○	○

※공유자원의 범위 : 좁음 : 파일(확장 불가능)
보통 : 파일(부분 확장 가능)
넓음 : S/W, H/W ASP 등 확장성이 유연함

6. 결 론

본 논문에서는 객체기반 P2P 애플리케이션 개발 프레임워크인 ObjectPeerWork를 설계 및 구현하였다. ObjectPeerWork은 공유 객체 클래스(SOC) 라고 하는 클래스를 통해 주어진 자원을 객체로서 관리할 수 있으며 플러그인이 가능한 메소드를 통해 확장이 가능한 P2P 아키텍처이다. 따라서 기업정보시스템 구축에 있어 컴퓨팅 파워의 분산, 자원의 효율적인 분산 운영 관리에 대해 새로운 모델로서 적용이 가능하며, P2P 네트워크 상의 하드웨어 및 오프라인과 연계된 서비스, ASP, 컨텐츠 등 공유 자원의 범위를 확장함으로써 P2P의 새로운 활용 가능성을 제시하였다. 또한 새로운 분산네트워크 관리 알고리즘을 제안해 P2P 네트워크의 보다 효율적인 운영이 가능함을 보였다.

향후 연구 과제로는 중계 서버의 운용 성능 향상을 위한 부분이 지속적으로 연구되어야 하며 보다 동적인 서비스 제공을 위한 다양한 에이전트들을 설정하고 운영 할 수 있는 방안과 기본적으로 제공하는 플러그인 함수를 확대하는 것이다.

참 고 문 헌

[1] Andy Oram, "Peer-To-Peer," March 2001, O'reilly.
[2] ThinkStream, "A Technical Review of the Next-Generation Internet Architecture".
[3] http://www.napster.com.
[4] on the Net Magazine, "네트워크 컴퓨팅의 새로운 패러다임, P2P", 2001.
[5] Endeavor Technology, "Introducing Peer-to-Peer," 2000, http://www.peer-to-peerwg.org.
[6] HP, "e-speak Architectural Specification" Release A.0,http://www.e-speak.com, December 2000.
[7] George Coulouris,Jean Dolli-more, and Tim Kindberg, "Distributed Systems:Concepts and Design," Addison Wesley, 1998.
[8] 송문섭, "분산 객체 시스템에서 지능형 에이전트를 이용한 자원 공유 모델 설계 및 구현", 전북대 대학원 석사학위논문, 2001.
[9] http://gnutella.wego.com.
[10] Gregory Alan Bolcer, Michael Gorlick, Arthur S. Hitomi, Peter Kammer, Brian Morrow, Peyman Oreizy, Richard N. Taylor. Peer-to-Peer Architectures and the Magi Open-Source Infrastructure, December 6, 2000, http://www.peer-to-peerwg.org.
[11] 안은정, "P2P(Peer-to-Peer)를 적용한 인터넷 비즈니스의 가능성에 관한 탐색적 연구", 숙명여대 정보통신대학원 석사학위논문, 2001.
[12] 김경하, 김영학, 오길호, "이동에이전트 시스템기반의 병렬계산을 위한 효율적인 분산방법", 한국정보과학회, 춘계학술발표 논문집, 2000.
[13] "XML Schema," W3C Candidate Recommendation, 2000.
[14] "Resource Cataloging And Distribution System" Keith Moore, Shirley Browne, Jason Cox, and Jonathan Gettler, 1997.
[15] Ian Clarke, "A Distributed Decentralized Information Storage and Retrieval System," 1999.
[16] Project JXTA, "Technical Specification" Version 1.0, http://www.jxta.org, 2001.
[17] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte, D. Winer. "Simple Object Access Protocol (SOAP) 1.1," W3C Note 08 May 2000, http://www.w3.org/TR/SOAP/
[18] OMG, The Common Object Request Broker Architecture and Specification 2.4.2, OMG Document, 2001.

[19] Roy T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, Ph.D. dissertation, Information and Computer Science, University of California, Irvine, 2000.

강 운 구

1993년 인하대학교 산업기술대학원 정보공학과 졸업(공학석사). 1999년 인하대학교 대학원 전자계산공학과 박사과정 수료. 2000년 ~ 현재 (주)이노웹 기술고문. 1994년 ~ 현재 가천길대학 뉴미디어과 부교수. 관심분야는 분산객체 컴퓨팅, 원격교육, 컴퍼넌트 기반 소프트웨어공학, 멀티미디어, P2P 컴퓨팅, 이동 에이전트

왕 창 종

1979년 ~ 현재 인하대학교 전자계산공학과 교수. 1997년 ~ 2001년 인하대학교 사회교육원장. 1992년 ~ 1994년 한국정보과학회 부회장. 관심분야는 소프트웨어공학, 분산객체 컴퓨팅, 지능형웹기반교육시스템