CORBA 명명 서비스를 이용한 객체지향 캐싱시스템

홍 성 준† · 김 영 재† · 한 선 영††

요 약

현재의 캐싱 시스템은 멀티미디어와 같은 방대한 자료를 처리하기에는 대역폭과 지연 문제가 발생한다. 그러므로 이 문제를 해결하기 위해서 본 논문은 분산 환경에서 멀티미디어 응용에 적합한 객체 지향적인 캐싱 시스템을 설계 및 구현하였다. 본 객체지향 캐싱 시스템은 세가지 특징을 가진다. 첫째는 CORBA 명명 서비스를 이용함으로써 URN(Uniform Resource Name) 기반의 객체에 대한 투명성을 제공한다. 둘째는 자바 애플릿을 이용함으로써 웹과 CORBA의 통합 환경을 제공한다. 그리고 셋째는 서버들간의 부하분담(load balancing)을 지원한다.

Object-Oriented Caching System by using CORBA Naming Service

Sungjune Hong† · Youngjae Kim† · Sunyoung Han††

ABSTRACT

Recently, the caching system is not well suited for multimedia applications due to bandwidth and latency overhead. To solve these problems, this paper describes the design and implementation of the Object-Oriented caching system suited for multimedia applications on a distributed environment. The Object-Oriented caching system has three key features: CORBA Naming Service, WWW/CORBA integration, and load balancing. The Object-Oriented caching system provides the transprency of object location for users based on URN(Uniform Resource Name) by using CORBA Naming Service. In addition, it provides the integration of WWW and CORBA by using Java applet. Furthermore, this system reduces the load of each servers by load balancing.

1. 서 론

최근 네트워크의 확산과 더불어 기존의 클라이언트/서버 환경을 발전시켜 지리적으로 분산시키면서 독립적인 기능을 수행하거나 필요할 때 협력하는 RPC나 DCE같은 분산 시스템이 보편화되고 있다. 그러나 이러한 방법은 분산 환경하에서 다양한 하드웨어와 응용 소프트웨어, 운영체제, 데이터베이스 등을 일관되게 연결하지 못하고 있다. 이러한 상황을 해결하기 위해 나온 방법이 OMG(Object Management Group)[1]의 객체 기술에 근거한 통합 기술인 CORBA (Common Object Request Broker Architecture)[1]이다. 이것은 실제로 구현된 또는 구현되는 언어에 상관없이 각 시스템들을 연결·통합할 수 있는 방법을 제공하는 미들웨어이다. 그리고 이러한 다양한 서비스 중에 CORBA 명명 서비스(Naming Service)는 마치 전화번호부의 White Page같은 객체를 이름으로 찾는 것을 의미한다. 이러한 CORBA 명명 서비스는 URN[15]기반으로 사용자가 객체를 이름으로 요구할

† 준 회 원: 건국대학교 컴퓨터공학과
†† 정 회 원: 건국대학교 컴퓨터공학과
논문접수: 1997년 10월 6일, 심사완료: 1997년 12월 29일

수 있다.

CORBA 서비스와 더불어, 현재 인터넷에서는 데이타의 신속한 처리를 위하여 인터넷 캐싱 메카니즘이 도입되었다. 그 이후에 "Single point of failure"의 문제를 해결하기 위해서 Harvest 캐싱 시스템 또는 Squid 캐싱 시스템[2] 등이 등장하게 되었고 현재 운용 중에 있다. 그러나 이러한 캐싱 시스템은 멀티미디어와 같은 방대한 자료를 처리하기에는 대역폭과 지연 문제 때문에 적합하지 못하다. 그러므로 이런 문제를 해결하기 위해서 CgR(Caching goes Replication)[3]이라는 개념이 제시되었고 CgR 구현을 위해서 WLIS(Web Location and Information Service)[3]라는 명명 서비스가 제안 되었다. 그리고, 상용으로 Cisco의 웹 캐싱 제품 등이 소개되고 있다. 그러나 WLIS나 웹 캐싱 제품은 URL(Uniform Resource Location) 기반으로 특정 웹 서버의 부하분담을 목적으로 개발되었지만, URL 기반에서 벗어나지 못하고 있다. 그러므로, 이러한 문제점을 해결하기 위해 본 논문에서는 분산된 PS(Primary Server) 또는 RS(Replicated Server)에 각각 객체가 등록되어 있을 경우, 사용자가 원하는 객체의 서버(PS 또는 RS) 위치를 몰라도 원하는 객체를 찾아주는, 즉 URL이 아닌 URN(Uniform Resource Name) 기반으로 객체를 검색할 수 있는 CORBA의 명명 서비스를 이용하여 CgR을 구현하는 캐싱 시스템 설계 및 구현에 관해서 논한다.

더불어 이러한 이기종 간의 분산 환경에서의 기본 환경인 웹과 CORBA환경의 통합이 요구되므로, 본 논문은 웹 환경하에서 동적인 분산 시스템 구축을 위한 개발 언어인 자바를 이용하여 구현하였다. 이것은 자바가 제공하는 유연성, 이기종 플랫폼에서 잘 동작하는 이식성으로 인해서, CORBA가 제공하는 분산 객체 하부구조에서 많은 서비스를 웹에서 이용할 수 있다.

본 논문은 2장에서 관련 연구로 CORBA, CgR, WLIS 그리고 웹과 CORBA의 통합 기술에 대해서 기술하고, 3장에서는 객체지향의 캐싱 시스템의 설계에 대하여, 4장은 구현을, 5장에서는 개발 결과, 그리고 마지막으로 6장에서 결론을 맺는다.

2. 관련 연구

2.1 CORBA

분산 시스템이 가지는 여러가지 문제를 해결하기 이해 OMG(Object Management Group)[6]에서 분산 객체 컴퓨팅 표준 구조로서 CORBA(Common Object Request Broker Architecture)[1]를 제안 하였다. CORBA는 분산 환경하에서 여러 이질적인 시스템을 통합하는 미들웨어의 한 종류다. 그리고 객체지향 개념을 기본으로 한다. 기존의 객체지향형의 프로그램에서는 메소드의 호출과 이에 대한 호출 처리가 한 프로세스에 존재한다. 네트워크를 기준으로 메소드 호출과 호출 처리가 클라이언트와 객체 구현(Object Implementation)으로 양분되어 있다. 이렇게 나누어진 클라이언트쪽 코드를 스텁이라고 하고 객체 구현 쪽을 스켈톤이라 하여 ORB(Object Request Broker)를 통해 CORBA 객체의 호출이 이루어진다.

2.2 CgR과 WLS

CgR(Caching goes Replication)[3]은 단순 수동적 캐싱과 데이터의 능동적 반영(mirroring)을 한데 묶은 일종의 능동적 캐싱 스킴이다. CgR의 핵심 구성은 PS(Primary Server), RS(Replicated Server)이다. PS는 기존의 프락시 서버와 같은 의미를 가지고, RS는 PS와 일관성(consistency)을 유지하면서 복제(replicated)된 데이터를 가지고 있는 서버를 의미한다.

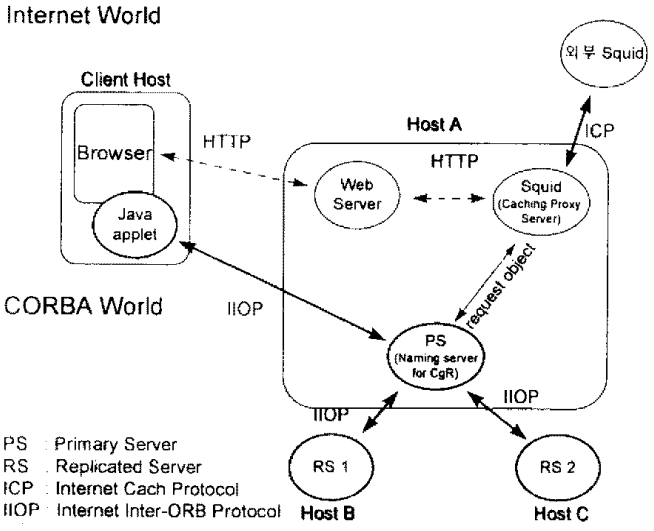
웹 상에서 CgR을 효율적으로 수행하기 위해서는, 주어진 어떤 PS(Primary Server)의 데이터를 복제해서 가지고 있는 RS(Replicated Server)를 찾아내기 위한 새로운 서비스가 필요하다. 이 서비스를 간단히 Web Location and Information Service 또는 WLIS[3]라고 하고, 이 서비스를 제공하는 서버를 WLIS 서버라고 명칭 한다. WLIS 서버는 일종의 "Application-level Name Server"이다.

2.3 CORBA와 웹 통합[8]

자바를 이용한 웹과 CORBA의 통합을 보여주고 있다. 웹 서버 내에 HTTP 서버와 CORBA 서버가 동시에 존재해 사용자가 HTTP 서버를 통해 HTML 문서와 애플릿을 다운 로드하면 웹 클라이언트는 클라이언트 프로그램인 애플렛(Applet)과 웹 서버내의 CORBA 서버의 CORBA IIOP(Internet Inter-ORB Protocol) 통신을 통해 객체들에 대해 서비스를 받는다.

3. 객체지향 캐싱 시스템의 설계

본 논문은 웹과 CORBA의 통합된 환경과 객체지향 캐싱 시스템을 통해서 투명하게 클라이언트의 요구를 처리할 수 있도록 (그림 1)과 같이 설계하였다. 본 논문은 객체 서비스를 요구하는 클라이언트, 요구된 객체를 서비스 해주는 서버 프로그램(PS)이 있다. 그리고 부하분담(load balance)을 위해 RS를 두어 클라이언트가 요구한 객체에 대해서 투명성이 보장되고 PS 또는 RS 서버들간의 부하분담을 해주는 객체지향 캐싱 시스템을 구현하였다.



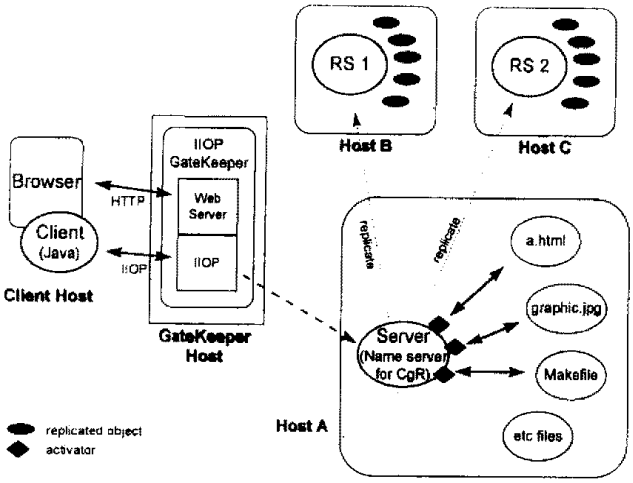
(그림 1) 객체지향 CORBA 캐싱 시스템의 설계
(Fig. 1) Design of Object-Oriented CORBA Caching System

(그림 1)은 CORBA 환경과 웹 환경의 통합을 고려하여 CORBA 명명 서비스를 이용해서 구현된 캐싱 시스템의 설계를 보이고 있다. (그림 1)에서 *Internet World*는 기존의 웹 환경을 말하고 있다. 이것은 *Host A*의 기존 웹 서버와 Squid라는 캐싱 서버가 있고 클라이언트인 브라우저가 존재한다. *CORBA World*측에서의 *Host A*의 *PS*(Primary Server)인 Naming 서버와 *Host B, C*에 있는 *RS* 그리고 CORBA 클라이언트 프로그램인 *Client Host*의 자바 애플릿이 CORBA 환경에서의 캐싱 시스템이다.

본 논문은 이와 같이 캐싱 시스템의 설계를 3가지로 나누어 설계하였다. 요구된 객체에 대한 서비스를 해주는 Naming 서버, 웹과 CORBA 통합을 위한 자바 클라이언트 그리고, 적절한 서버(PS, RS)를 선택해주어 부하분담을 해주는 기능이 있다.

3.1 CORBA 명명 서비스를 이용한 CgR을 위한 Naming 서버

CORBA의 명명 서비스는 (그림 2)에서 *Host A*에 해당한다. 그리고 이 CORBA 캐싱 서버 프로그램이 객체들에 대해 명명 서비스를 해주는 것이다.



(그림 2) 객체지향의 캐싱 시스템의 구현
(Fig. 2) Implementation of Object-Oriented Caching System

처음 CORBA 캐싱 서버 프로그램을 수행 했을 때, 맨 처음에는 캐싱 서버의 캐싱 데이타의 디렉토리와 그 서브 디렉토리의 모든 파일에 대해 파일 이름과 파일의 위치 디렉토리 정보가 객체화 되고 그 각각의 객체들은 Naming Context에 등록된다. 여기서 Naming Context란 각 객체에 대해 구체적인 정보를 가지고 있는 저장소를 의미한다. 그리고 네트워크를 잘 쓰지 않는 시간에 PS(Primary Server)가 갖고 있는 데이타를 RS(Replicate Server)들에게 복제(replicate)한다. 이렇게 캐싱 데이타의 정보가 객체로 등록이 된 다음 CORBA 캐싱 서버는 브라우저를 통해 Client Host의 자바 클라이언트가 객체 서비스를 요구받게 되면 Naming 서버 프로그램은 먼저 요구된 객체에 대해서 PS와 RS중에 대역폭(bandwidth)을 검사하여 대역폭이 가장 큰 서버(RS or PS)를 선택한다. 선택된 서버 프로그램은 클라이언트가 요구했던 객체를 invoke한 다음 해당 객체의 참조가 자바 클라이언트로 넘겨준

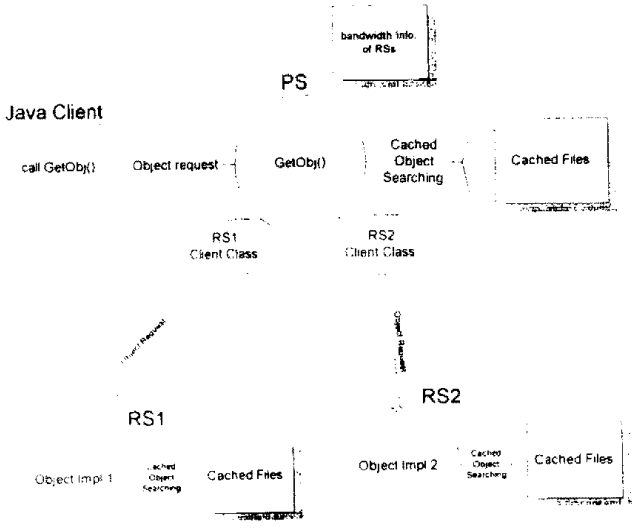
다. 자바 클라이언트 프로그램은 넘겨 받은 객체의 참조를 통해 투명하게 객체를 가져온다.

그러므로 설계된 객체지향 캐싱 시스템은 투명성 있는 URN[15] 기반의 명명 서비스 즉, 객체가 어느 서버에 있는지 그리고 어느 디렉토리에 있는지 몰라도 그 객체의 이름만으로 객체에 대한 서비스를 해줄 수 있다.

3.2 자바를 통한 웹과 CORBA의 통합

자바를 통한 웹과 CORBA의 통합은 (그림 2)의 *Client Host*에서 자바 클라이언트를 사용해서 해결한다. *Client Host*는 맨 처음 HTTP 프로토콜을 통해 웹 서버에 있는 html 문서와 자바 애플릿을 브라우저쪽에 다운로드한다. 다운로드된 CORBA용 자바 클라이언트는 *gatekeeper*를 통해 서버 프로그램과 IIOP (Internet Inter-ORB Protocol) 통신 할 수 있다. 이렇게 연결이 된 다음 자바 애플릿에서 원하는 객체의 이름을 주었을 때 서버 프로그램에게 그 객체의 이름이 전달된다. 그러면 서버 프로그램은 원하는 객체의 참조를 자바 클라이언트 프로그램에게 준다. 자바 클라이언트는 이 참조를 가지고 원하는 객체를 투명하게 얻을 수 있다.

3.3 부하분담(load balance)을 위한 RS(Replicate Server)



(그림 3) 부하분담을 위한 RS의 디자인
(Fig. 3) The Design of RS for Load Balance

요구하는 객체에 대해 많은 사용자들이 한 서버로 몰리는 것을 분산화 시키는 방법으로 PS(Primary Server)에 들어오는 사용자들에 대해 가장 대역폭이 큰 RS(Replicated Server)를 선택해 그 RS로 연결해 주는 부하분담 기능이 있도록 한다.

본 논문에서는 (그림 3)와 같이 PS가 네트워크가 사용되지 않는 밤시간을 이용하여 캐싱 서버가 갖고 있는 데이타를 RS인 RS1, RS2에게 복제 해준다. 자바 클라이언트에서 들어온 객체의 요구에 대해 가장 큰 대역폭을 가진 RS를 PS가 선택한다. 즉 대역폭이 적은 RS가 대신 객체 요구에 대한 서비스를 처리해 주게 한다. 이렇게 웹 사용자는 객체가 어느 서버에 있는지 모르는 투명성을 가지며 같은 객체들을 복제해 줌으로써 부하분담을 해줄 수 있다.

4. 객체지향 캐싱 시스템의 구현

본 논문에서는 (그림 3)과 같이 객체지향 캐싱 시스템을 구현하기 위해 IDL을 정의하였다. 정의된 IDL을 가지고 (그림 4)에서처럼 IDL을 컴파일해 나온 스텁 코드와 스켈톤 코드를 생성하고 CORBA 프로그램을 구현한다.

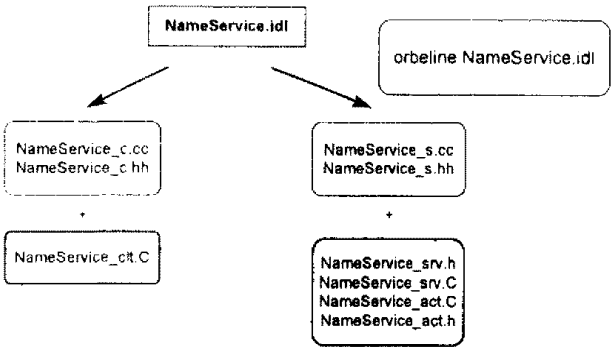
```
inferface NameService
{
        any    GetObj( );
};
```

(그림 4) 캐싱 시스템의 IDL 정의
(Fig. 4) Definition of Caching System IDL

4.1 CORBA 명명 서비스 IDL 정의와 IDL 컴파일

구현된 캐싱 시스템의 명명 서비스 IDL은 (그림 4)와 같이 정의한다.

(그림 4)에서 정의된 NameService.idl을 가지고 Visibroker에서 제공하는 IDL 컴파일러인 orbeline으로 컴파일 하면 *NamingService* 스텁 코드와 Naming Service 스켈톤 코드가 나온다. 여기서 _c가 클라이언트 스텁 코등고 _s가 서버의 스켈톤 코드가 된다.



(그림 5) 구현 소프트웨어의 구성
(Fig. 5) Implementation software configuration

IDL 컴파일러에 의해서 생성된 CORBA의 스텁과 스켈톤 코드는 수정하지 않고 해당 클래스로부터 상속 받아 명명 서비스 기능에 대한 소스 코드를(그림 5)의 하단 부분인 .C파일과 .h파일을 코딩 한다.

(그림 6)은 *NameService_srv.h*에서 정의된 것으로 캐싱 서버가 가지고 있는 캐시 데이타의 정보를 정의한 *NameService_impl* 클래스이다. 캐싱 데이타의 정보인 파일의 이름과 파일이 저장된 디렉토리가 정의되어 있고 객체를 자바 클라이언트가 실행 시킬 *GetObj*( ) 메소드 등이 있다.

```
include "NameService_s.hh"
class NameService_impl:public virtual _sk_ NameService
{
public:
        CORBA::Any* GetObj();
        char* GetFileType();
        void InsertFileInfo(char* name, char* path);
        NameService_impl(char* name) :
                _sk_ NameService (name)( )
        ~ NameService_impl();
private:
        char* file_name;
        char* file_path;
};
```

(그림 6) 등록 객체정보 클래스
(Fig. 6) Registered Object Information Class

4.2 CORBA 명명 서비스의 서버측 프로그램

(그림 7)은 CORBA 명명 서비스의 서버측 프로그램의 소스 코드이다. 우선 *ORB_innt* 메소드를 가지고 ORB를 초기화시키고 서버 프로그램을 활성화(activate)하는 BOA(Basic Object Adaptor)가 초기화된다. 서버 프로그램은 클라이언트가 요구할 객체에 대해 활성화를 준비한다. 준비가 끝나 서버 프로그램은 클라이언트 요구가 있을 때 원하는 객체에 대한 참조를 Any형 객체 포인터를 넘긴다.

원래 Name context에 등록된 서비스 객체들은 서비스를 위해 항상 떠워져 있어야 한다. 그러나 이것은 그 많은 객체들을 띄우게 되면 메모리의 낭비뿐만 아니라 서비스 객체가 많을 때는 다운이 되는 경우도 있어 좋지 않은 방법이다. 그래서 이 논문에서는 활성자(activator)를 사용해 클라이언트에 요구하는 객체를 찾아 활성화 해주어 시스템의 부하를 크게 줄였다.

```
int main(int argc, char* const* argv) {
        CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
        CORBA::BOA_var boa = orb->BOA_init(argc, argv);
        ...
        CORBA::ActivationImplDef* pDef = new ActivationImplDef( "NameService",
            dirp->d_name, id, (CORBA::Activator_ptr) new NameServiceActivator(
            dirp->d_name, ".") );
        act_def[i] = pDef;
        boa->obj_is_ready( NULL,  act_def[j] );
        boa->impl_is_ready();
}
```

(그림 7) 서버 main 소스코드
(Fig. 7) Source Code of Server main

4.3 CORBA 명명 서비스의 클라이언트쪽 프로그램

CORBA 자바 클라이언트 애플릿의 구현은 (그림 8)에서처럼 *init* 메소드에서 *TextField*와 *Button*등 자바 AWT를 사용해 화면에 그려준 다음 ORB를 초기화 시키고 서버의 객체와 연결(bind)한다. 그리고 사용자가 "*RUN*"을 클릭 했을 때 *action* 메소드 부분이 실행된다. 그 다음 서버의 메소드인 *GetObj*( )를 실행하여 클라이언트가 요구한 객체의 참조를 가져와 화면에 출력해준다.

```
import java.awt.*;
public class NameService_applet extends java.applet.Applet {
 public void init() {
      setLayout(new FlowLayout());
      add(new Label(" Object Name "));
      add(_nameField = new TextField("The object name here", 45));
      add(_running = new Button(" RUN "));
      add(_result = new TextArea("This area is result of requested
            Caching Object", 20, 60));

      try {
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(this);
            NameService_object = NameServiceHelper.bind(orb, "run" );
```

```
        } catch(org.omg.CORBA.SystemException e) {
        }
    }

    public boolean action(Event ev, Object org) {
      if(ev.target == _running) {
        try {
            org.omg.CORBA.Any ret;
            ret = NameService_object.GetObj();
              _result.setText( ret.extract_string() );
        } catch(org.omg.CORBA.SystemException e) {
        }
      }
    } // end of action
}
```

(그림 8) 클라이언트 소스코드
(Fig. 8) Source Code of Client Program

(그림 9)은 브라우저에서 클라이언트 애플릿을 부르는 *ClientApplet.html* 문서를 나타낸 것이다. 애플릿 tag를 보면 파라메터로 *USE*_ORB_LOCATOR를 사용하였는데 이것은 *gatekeeper*를 이용 ORB와 통신을 15000 포인트로 한다는 의미이다.

```
<html>
<body bgcolor=white>
<center>
<h1>Java Client Applet using Caching Object Name Service </h1> <br>
다음은 CORBA의 Name Service를 이용하여 Caching Object에대해 사용자가 투명하게<br>
객체의 위치등을 몰라도 객체서비스를 받을수 있는 메카니즘을 구현하였다.
</center>

<br><hr><br>
<center>

<applet code=NameService_applet.class width=500 height=400>
  <param name=USE_ORB_LOCATOR value=15000>
  <h2>This would be a Really Cool VisiBroker for Java Applet,
  but you are not running a Java enabled browser ...</h2>
</applet>

</center><hr>

</body>
</html>
```

(그림 9) ClientApplet.html 문서
(Fig. 9) ClientApplet.html File

4.4 부하분담을 위한 모듈

(그림 10)은 부하분담을 위한 PS에서의 각 모듈에 대한 설명이다. 클라이언트가 처음 구현 객체인 GetObj() 메소드를 호출한다. PS에 있는 GetObj() 메소드는 적당한 RS를 선택하기 위해 ChooseRS() 메소드를 호출하여 N개의 RS에 대해 대역폭 정보를 알아내고 *BandwidthCheck*() 메소드로 대역폭을 비교하여 대역폭이 적은 RS를 선택한다. 이렇게 RS가 선택되면 클라이언트가 *GetRSObj*() 메소드를 호출한다.

```
CORBA::Any* NameService_impl::GetObj()
{

        Name_RS = ChooseRS();
        ret = NamingObject->GetRSObj(Name_RS);

}

CORBA::String ChooseRS()
{
        for(n_RS = 1; n_RS > RSCOUNT; ++n_RS) {
                Open_Bandwidth_Info();
                BandwidthCheck(n_RS);
        }
}
```

(그림 10) 부하분담을 위한 PS쪽 method
(Fig. 10) The PS part for Load Balance

(그림 11)은 이렇게 호출된 GetRSObj() 메소드는 RS가 복제하고 캐시 데이타 중에서 해당 객체의 정보 즉 데이타의 파일과 디렉토리를 가지고 파일을 열고 읽는다. 이렇게 읽은 파일을 캐릭터(Character) 포인터형으로 가리키게 되고 이것을 CORBA의 Any형으로 PS로 넘겨주게 되고 이것을 PS가 클라이언트로 넘겨준다.

이렇게 함으로 Any형으로 객체에 대한 참조로서 클라이언트가 메모리에 있는 것을 가져올 수 있다.

```
CORBA::Any* NameService_impl::GetRSObj()
{
    char temppath[255];

    sprintf(temppath, "%s/%s", file_path, file_name);
    FILE *fp = fopen(temppath,"rb");

    char* file_ptr = (char *)malloc(current);
    fread(file_ptr, current, 1, fp);
    TypeCode_ptr temp_tc = new TypeCode(CORBA::tk_array, current, CORBA::_tc_char, 0);
    CORBA::Any* temp_any = new CORBA::Any(temp_tc, (void*)file_ptr, 0);

    return temp_any;
}
```

(그림 11) 부하분담을 위한 RS쪽 메소드
(Fig. 11) The RS part for Load Balance

5. 개발 결과

먼저 CORBA 환경이 되기 위해서는 분산 디렉토리 서비스인 *osagent*를 실행한다(osagent &). 그리고 웹과의 연동을 위해 *IIOP gatekeeper*를 두어 Visibroker

의 자바 애플릿과 객체 서버와 IIOP 통신을 하게 해 준다. 여기에서 주의할 것은 브라우저의 HTML의 애플릿 태그 중에 *port*와 *gatekeeper*를 띄울 때 서로 *port* 번호가 맞아야 한다는 것이다(Gatekeeper-port 15000 &).

5.1 캐싱 시스템 서버 실행

캐싱 시스템 서버 프로그램을 (그림 12)과 같이 실행하면 지정된 디렉토리의 모든 파일들에 대한 정보(파일, 이름, 파일이 있는 디렉토리)를 가진 객체들에 *osagent*에 등록이 되고 클라이언트의 요구를 기다린다.

(그림 12)에서처럼 서버 프로그램을 실행하면 등록된 객체들을 화면이 출력되고 서비스할 준비가 되었다고 화면에 "*BOA obj is ready*"와 "*BOA imple is ready*" 2개의 메시지를 화면에 출력한 다음 클라이언트쪽의 요구를 기다린다.

```
[oxen] server&
****** 캐시 데이타에 대한 등록된 객체들 *****
0 : NameService.idl
1 : Makefile
2 : NameService.java
3 : WLIS_applet.class
4 : WLIS_clt.class
5 : ClientApplet.html
6 : server
7 : client
8 : frog.jpg
9 : plane.bmp
10 : ClientApplet.html
11 : a.html

***** BOA obj is ready ******
***** BOA imple is ready ******
```

(그림 12) CORBA 서버쪽의 실행
(Fig. 12) Running CORBA Server Program

(그림 13)은 자바 클라이언트가 a.html 객체를 요구했을 때 캐싱 시스템 서버 프로그램의 반응이다.

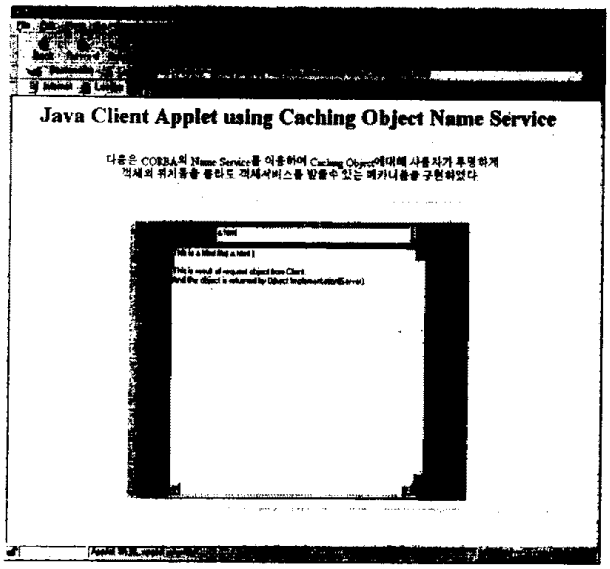
(그림 13)는 (그림 12)에서 서버 프로그램이 실행하고 난 뒤 클라이언트의 요구 즉 a.html 객체를 요구했을 때 서버 프로그램쪽의 반응이다. 맨 먼저 클라이언트가 요구한 객체의 정보 즉 이름과 디렉토리를 출력한다.

```
There come a request from Client.
Object is activated by activator.

** Naming Server GetObj function **

Requested Object Name : ./a.html
```

(그림 13) a.html 객체 요구에 대한 서버처리 출력
(Fig. 13) Result of Requested Object(a.html)

5.2 캐싱 객체 서버의 클라이언트 실행 프로그램

클라이언트 자바 애플릿을 실험하기 위해 (그림 14)처럼 ***a.html*** 객체를 요구할 때 화면에 나타나는 결과를 보여준다. (그림 14)는 자바 클라이언트 애플릿에서 그림 상단에서 보는 것처럼 객체 이름을 ***a.html***라고 삽입하고 "*RUN*" 버튼을 누르면 해당 객체를 서버 프로그램에게 요구하고, 요구된 객체의 참조를 서버 프로그램으로부터 받아 그 참조를 가지고 화면에 출력하는 과정이다.



(그림 14) 클라이언트가 자바 애플릿을 통해 a.html 객체 요구
(Fig. 14) a.html Object request with Java applet

5.3 기존 캐싱 시스템과의 비교 및 평가

여기서는 CgR 기반 캐싱 시스템 구현을 위한 명명 서비스 프로그램을 바탕으로 기존 WLIS와 특징을 비교 평가 해보았다.

5.3.1 평가기준

CgR 기반 캐싱 시스템을 구현하는 맥락에서 평가되어야 하는 주된 속성은 객체에 대한 위치 투명성, 부하분담 그리고 웹과 CORBA 연동 등이 고려될 수 있다.

5.3.2 기존 캐싱 시스템과의 비교

〈표 1〉은 Squid라는 인터넷 캐싱 서버, WLIS 그리고 본 캐싱 시스템과의 특징을 비교하였다. 객체에 대한 위치 투명성 특징은 본 캐싱 시스템이 CORBA의 명명 서비스가 URN을 기반으로 하고 있음을 말한다. 웹과의 연동 특징은 본 캐싱 시스템의 클라이언트 부분이 자바의 애플릿으로 구현되었기 때문에 자바를 이용한 웹과 CORBA의 연동이라는 것을 의미한다. 부하분담 특징은 본 CgR 기반의 캐싱 시스템이 CORBA의 naming을 기본으로 부하분담을 지원하는 것을 말한다.

〈표 1〉 기존 캐싱 시스템과의 특징 비교
〈Table 1〉 Comparison with existing caching system

종류 / 특징	Squid	WLIS	본 캐싱 시스템
위치 투명성(URN)	없음	없음	있음
부하분담	없음	응용 프로그램 수준에서 제공	CORBA의 명명 서비스를 이용, 제공
웹과 CORBA의 연동	없음	없음	자바용 ORB를 사용 연동

6. 결 론

본 연구에서는 웹과 CORBA가 통합된 환경에서 CORBA 명명 서비스를 이용한 CgR 기반의 캐싱 시스템을 개발하는데 목표가 있다. CgR이란 기존 캐싱 서버들의 불필요한 대역폭의 낭비와 지연 문제를 보완하기 위해 나온 캐싱 기법이다. 또한 CORBA 명명 서비스는 URN 기반으로 객체를 찾을 수 있는 메카니즘으로 분산된 환경에서 그 응용 분야가 광범위하며 CORBA와 웹의 통합에 관한 연구는 국내외적으로 활발한 연구가 진행 중에 있다. 본 구현 시스템인 캐싱 시스템은 URN 기반의 CORBA 명명 서비스를 이용하여 객체에 대한 위치 투명성을 제공할 수 있도록 했으며, 클라이언트를 자바 애플릿을 이용하여 개발함으로써 웹과 CORBA의 통합 및 성능 향상을 꾀하였다. 더욱이 RS를 두어 한 서버에 오는 부하를 분담하는 부하분담 기능을 제공한다. 본 시스템의 핵심은 CgR을 위한 CORBA 명명 서비스와 클라이언트의 자바 애플릿을 사용함으로써, URL이 아닌 URN 방식의 접근이 가능하고 자바를 이용한 CORBA와 웹 연동 메카니즘을 이용할 수 있다. 향후에 국제 표준 적용 또한 심도 있게 다루어져야 할 것이다.

참 고 문 헌

[1] CORBA OMG site, 〈URL:http://www.omg.org/〉.

[2] Squid Caching Proxy, 〈URL:http://squid.nlanr.net/Squid/〉.

[3]WLIS(Web Location and Information System), Michael Baentsch, 〈URL:http:www.uni-kl.de/AG-Nehmer/GeneSys/WLIS/〉.

[4] Visigenic Programmer's guide version 2.0, Visigenic Software Inc, 1997.

[5] Visigenic Reference guide version 2.0, Visigenic Software Inc, 1997.

[6] Jon Siegel, "CORBA Fundamentals and Programming," John Wiley & Sons Inc, pp.1-111, 1996.

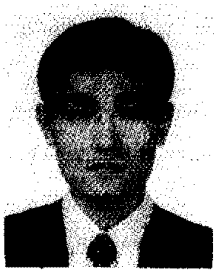
[7] Thomas J. Mowbray, "CORBA Design Pattern," Hohn Wiley & Sons Inc, 1997.

[8] Orfali Harkey, "Client/Server Programming with JAVA and CORBA," Willy Computer Publishing, 1996.

[9] David Flanagan, "Java in a Nutshell," O'Reilly & Associates Inc, 1996.

[10] World Wide Web Journal, Fourth International World Wide Web Conference Proceedings, 1995.

[11] Gundavaram, "CGI Programming on the World Wide Web," O'Reilly & Associates Inc, 1996.
[12] Michael Morrison, "Java Unleashed," Sams.net Publishing, 1996.
[13] Deitel & Deitel, "Java How to Program," Prentice-Hall Inc, 1997.
[14] Michael Baentsch, "Introducing Application-level Replication and Naming into today's Web," Fifth International World Wide Web Conference, Paris, France, 1996.
[15] K. Sollins, L. Masinter, "Functional Requirements for Uniform Resource Names: Internet RFC 1737," http://www.w3.org/pub/WWW/Addressing/rfc1737.txt, December 1994.
[16] 제1회 인터넷 운영 워크샵 자료집, 한국 인터넷 협회, 1997.
[17] 제2회 인터넷 운영 워크샵 자료집, 한국 인터넷 협회, 1997.

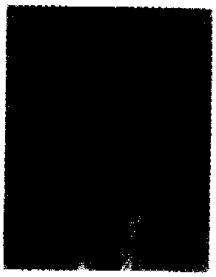


홍 성 준

1991년 경원대학교 전자계산학과 졸업(공학사)
1993년 건국대학교 전자계산학과 졸업(공학석사)
1993년~1996년 한국통신 서울전자교환운용연구단 전임연구원
1996년~현재 건국대학교 컴퓨터공학과 박사과정 재학중
1996년~현재 건국대학교 컴퓨터공학과 시간강사
※관심분야:컴퓨터 네트워크, 멀티미디어 시스템, 차세대 인터넷 프로토콜

김 영 재

1996년 2월 관동대학교 전자계산학과 졸업(공학사)
1998년 2월 건국대학교 컴퓨터공학과 졸업예정(공학석사)
1998년 3월 건국대학교 컴퓨터공학과 박사과정 입학
관심분야:CORBA, Internet Caching, Network Management



한 선 영

1977년 서울대학교 계산통계학과 학사
1979년 한국과학기술원 전산학석사
1988년 한국과학기술원 전산학박사
1981년 3월~1998년 현재 건국대학교 전산과 교수
1989년 1월~1990년 1월:Univ. of Maryland, Dept of Computer Science Visiting Associate Processor(미국)
1991년 1월~1998년 현재 금융결제원 자문교수
1990년 9월~1998년 현재 한국과학기술원 인공지능연구센터 참여교수
1992년 1월~1998년 현재 개방형컴퓨터통신연구회 이사, TG-VT 의장
1991년 7월~1993년 12월 한국정보과학회 정보통신연구회 부위원장
1990년 2월~1998년 현재 ISO/IEC JC1/SC21 WG8 위원장(국내 위원회)
1995년 1월~1998년 현재 개방형 컴퓨터 통신 연구회 TG-WWW 의장
1995년 6월~1997년 건국대학교 산업기술연구소 정보통신연구센터 소장
관심분야:Real Time CORBA, Internet Caching, 차세대 인터넷 프로토콜