공학석사 학위논문

지도교수 황 기 태

UPnP와 Jini 미들웨어, OSGi 게이트웨이를 기반으로 하는 홈네트워킹

Home Networking based on OSGi gateway and the middlewares of UPnP and Jini

2003年 12月

한 성 대 학 교 대 학 원

컴퓨터 공학과

컴퓨터 공학 전공

강 성 민

공학석사 학위논문

지도교수 황 기 태

UPnP와 Jini 미들웨어, OSGi 게이트웨이를 기반으로 하는 홈네트워킹

Home Networking based on OSGi gateway and the middlewares of UPnP and Jini

위 논문을 공학석사 학위논문으로 제출함

2003年 12月 日

한 성 대 학 교 대 학 원

컴퓨터 공학과

컴퓨터 공학 전공

강 성 민

강성민의 공학석사 학위논문을 인정함

2003年 12月 日

심사위원장 정 인 환

심사위원 김 영 웅

심사위원 이 민 석

목 차

표 목 차

그 림 목 차

1. 서 론

1.1 연구배경

인터넷 환경이 성장함에 따라 홈 네트워킹에 대한 관심이 높아지고 있다. 가정을 디지털 네트워크로 연결하는 홈 네트워크 기술은 가정내의 통신, 방송, 가전, 정보기기 등을 유무선 네트워크로 상호 연결하여, 가정의 내·외부에서 언제 어디서나 원하는 단말기로 원하는 정보를 주고받을 수 있는 환경을 제공하는 것을 목적으로 한다[1].

홈 네트워킹의 중요 기술로는 네트워크를 구성하는 물리적 매체(Physical Media)와 네트워크 인터페이스를 정의하는 네트워킹 솔루션, 각 기기들의 제어를 위한 미들웨어 솔루션 그리고 전체 홈 네트워크를 제어하고 외부와의 인터페이스를 제공하는 홈 게이트웨이 솔루션기술이 있다.

네트워킹 솔루션의 형태는 크게 유선과 무선으로 분류된다. 유선형태의 대표적인 것으로는 HomePNA (Home Phoneline Networking Alliance), IEEE1394, USB(Universal Serial Bus) 및 전력선 통신(Power Line Communication)[2-5] 등을 들 수 있으며, 무선형태로는 Bluetooth, HomeRF(Home Radio Frequency), WPAN(Wireless Personal Area Network) 및 IrDA(Infrared Data Association)[6-8] 등이 있다.

미들웨어로는 HAVi(Home Audio and Video Interoperability standard), UPnP (Universal Plug and Play), Jini, HWW(Home Wide Web)[9-11] 등 이 있으며, HAVi를 제외한 대부분의 미들웨어들은 기기간의 통신을 위해 TCP/IP 프로토콜을 하부에 사용하고 있고, HAVi는 IEEE1394를 기반으로 설계된 별도의 프로토콜 스택을 사용한다.

미들웨어 기술은 현재 다양한 단체들의 표준들이 혼재하면서 우위를 차지하기 위한 경쟁이 치열하며, 최근에 들어와 단체 표준간 호환성을 확보하여 경쟁력 우위를 얻기 위한 기술개발이 시작되고 있는 단계이다.

이로 인해, 구축된 홈 네트워크 상에 여러 미들웨어가 혼재하게 되며, 이종 미들웨어 간의 상호운용에 관한 연구가 필요한 실정이다.

외부 액세스 망과 홈 네트워크를 연동시키기 위한 홈 게이트웨이는 각 하드웨어 제작업체에서 미들웨어 종류에 따라 독자적으로 개발되어 왔다. 그러나 이종 미들웨어가 혼재하는 홈 네트워크 상에서는 각 미들웨어에 따른 각각의 홈 게이트웨이가 필요하며 이종의 미들웨어를 사용하는 기기 간의 상호 운용이 어렵거나 불가능하였다. 이와 같은 호환성의 문제를 해결하기 위해 통신, 가전, 컴퓨터 등 각 분야의 세계적인 업체들이 모여 지난 1999년 3월 설립한 비영리 민간단체인 OSGi(Open service gateway initiative)[12] 에서 홈 게이트웨이 서비스의 표준 안을 정의 하였다. OSGi 에서는 홈 게이트웨이를 위한 표준 Java API Spec 1.0 을 2000년 5월에 발표 하였으며, 현재는 버전 3.0 이 발표되어 있다.

1.2 연구 목적

본 논문에서는 홈 네트워크 미들웨어 중 현재 활발히 연구되어지고 있는 UPnP, Jini 미들웨어를 이용하여 홈 네트워크 모델을 구성하고, OSGi 기반의 홈 게이트웨이를 설계하며, UPnP에 대해서 하나의 클라이언트로 여러 UPnP 디바이스를 제어할 수 있는 UUCA(Unified UPnP Control Agent)를 설계, 구현하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서 관련연구로 OSGi와 Jini, OSGi 에 대해 살펴보며, 3장에서 UPnP와 Jini로 구성된 홈 네트워크에 대해서, 4장에서 OSGi 게이트웨이와의 연동을 위한 Jini 제어번들과 UUCA 번들의 설계 및 구현에 대해 기술한다. 5장에서는 OSGi 프레임워크와 번들의 설치와 실행에 대해 기술하며, 6장에서 결론 및 향후 연구과제에 대해 논한다.

2. 관련 연구

2.1 UPnP

2.1.1 UPnP 개요

UPnP는 Microsoft사가 주도하여 UPnP forum에서 제안된 미들웨어로 홈 네트워크의 미들웨어 분야에서 구현 및 표준화 속도가 빠르고, 멤버도 급속도로 늘어나고 있다. UPnP는 표준 TCP/IP와 이미 검증된 웹 기술을 기반으로 홈 네트워크 기기간의 제어 모델을 구현하여 하드웨어, 소프트웨어, 운영체제에 무관하게 동작이 가능하고, HTML을 이용하여 간단하게 사용자 인터페이스를 제공하고 있다[13].

현재 Intel, Allegro Software, Lantronix, Metro Link, Siemens 등 여러 회사에서 UPnP SDK를 개발하고 있으며, Intel에서는 UPnP SDK for Linux 1.0.4 버전을 배포하였다.

다음은 UPnP가 가지고 있는 특징들이다.

① **매체 및 기기 독립성** : UPnP는 전화선, 전력선, 이더넷, RF나 IEEE1394 등 어떠한 매체에서도 동작할 수 있다.

② **플랫폼 독립성** : 디바이스 제조업체는 OS나 프로그램 언어에 구애받지 않고 UPnP 디바이스를 제작할 수 있다.

③ **인터넷 기반 기술** : UPnP 기술은 IP, TCP, UDP, HTTP나 XML기반으로 구성된다.

④ **UI 컨트롤** : UI(User Interface)나 웹 브라우저를 이용하여 기기들을 제어할 수 있는 구조로 되어있다.

⑤ **확장성** : UPnP 디바이스들은 기본적인 기기 구조 위에서 제조업체들이 각각의 부가기능을 제공할 수 있다.

2.1.2 UPnP 구성요소

UPnP는 그림 1과 같이 사용자에게 서비스를 제공하는 댁내망의 장치인 디바이스와 이들을 제어하는 컨트롤 포인트의 두 요소로 구성된다.

① 컨트롤 포인트(Control Point)

컨트롤 포인트는 일반적인 클라이언트의 역할을 하며 디바이스를 제어하고, 디바이스로부터 서비스를 제공받는다. 디바이스를 제어 또는 서비스를 받기 위해 디바이스를 찾는 과정과 디바이스의 기술문서(device description)를 가져오는 과정을 통해 디바이스의 세부정보를 얻고, 그 세부정보를 통해 디바이스를 제어하고, 디바이스로부터 서비스를 제공받는다. TV 리모콘, PC, PDA, 다른 정보가전기기를 제어하거나 서비스를 이용할 필요가 있는 정보가전기기가 컨트롤 포인트 역할을 한다.

② 디바이스(Device)

TV, 비디오, 오디오, 냉장고 등 일반적인 정보가전기기가 디바이스 이며, UPnP 디바이스는 서비스와 장치 내에 내포(nested) 디바이스를 가지고 있을 수 있다. 예를 들어, VCR 디바이스는 비디오 재생에 관련된 서비스와 전원 서비스를 가진다고 하고, TV 디바이스는 채널 서비스와 전원 서비스를 가지고 있다고 하자. 이때, TV/VCR 일체형 디바이스의 경우에는 TV의 서비스를 가지고 있는 내포 디비이스와 VCR 서비스를 가지고 있는 내포 디바이스, 그리고 TV/VCR 일체형 디바이스에 필요한 서비스로 구성될 수 있다.

이러한 디바이스의 모든 정보는 각 디바이스 마다 가지고 있어야 하는 XML 디바이스 기술 문서 안에 들어 있으며, 이 기술문서는 컨트롤 포인트에 제공된다.

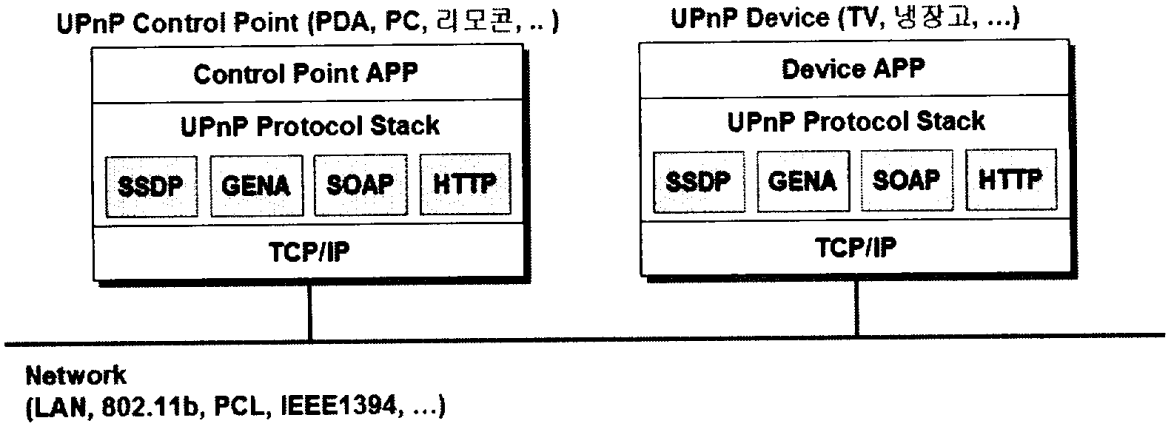

그림 1 UPnP 기본 구성

2.1.3 UPnP 프로토콜 스택

UPnP는 그림 2와 같이 TCP/ IP 기술을 바탕으로 현재 인터넷에서 사용되고 있는 HTTP(HyperText Transfer Protocol), SSDP(Simple Service Discovery Protocol), GENA(General Event Notification Architecture), 그리고 XML(eXtensible Markup Language), SOAP(Simple Object Access Protocol) 등의 프로토콜을 이용하여 기기의 정보를 수집하고 기기를 제어한다.

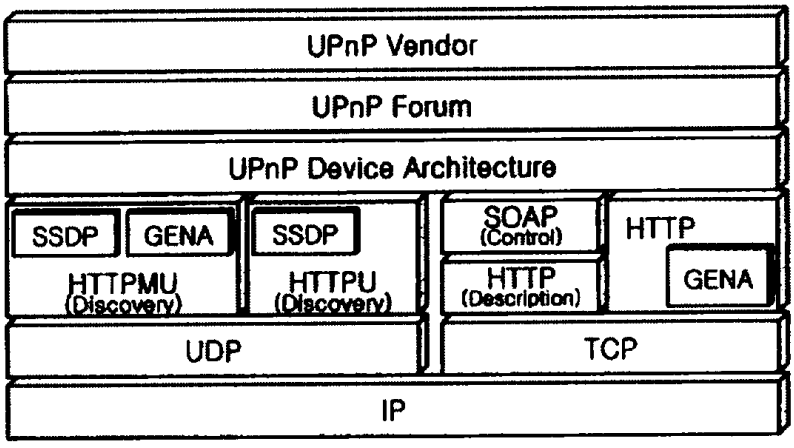

그림 2 UPnP 프로토콜 스택

① UPnP 프로토콜

UPnP 포럼의 UPnP 디바이스 아키텍처 문서에는 UPnP를 구현하는데 사용되는 최상위 계층 프로토콜이 정의 되어 있다. UPnP 포럼 실무 위원회에서는 디바이스 아키텍처에 기반하여 VCR, 식기세척기, TV 등 정보 가전기기들과 여러 디바이스에 따른 정보를 정의하고 있다. 이에 따라 디바이스 제조업체들은 자신들의 디바이스의 모델 이름이나 URL 등과 같은 데이터를 정의 한다.

② TCP/IP

TCP/IP 스택은 다른 UPnP 프로토콜의 기초 역할을 한다. UPnP는 표준인 TCP/IP 프로토콜을 기반으로 사용하기 때문에 다양한 네트워크 매체에 사용할 수 있으며 여러 디바이스 제조업체들 사이의 상호운영을 가능하게 한다.

③ HTTP, HTTPU, HTTPMU

UPnP의 모든 프로토콜은 HTTP나 변형들을 기반으로 한다. HTTPU는 TCP 대신 UDP 상에서 메시지를 전달하기 위한 HTTP의 변형이다. 이 프로토콜은 SSDP에 의해 사용된다. HTTP와 유사한 이러한 프로토콜이 사용하는 메시지 포맷은 멀티캐스트를 지원해야 하고 메시지를 전달할 때 신뢰성을 위한 오버헤드가 없어야 한다.

④ SSDP(Simple Service Discovery Protocol)

IETF (Internet Engineering Task Force)에서 제안한 TCP/IP 네트워크에서 서비스를 발견하기 위한 프로토콜로 매우 간단한 공지(advertisement) 와 검색을 제공한다. 서비스를 검색하려는 클라이언트에서 멀티캐스트로 검색을 하면 서비스를 제공하는 서버에서 유니캐스트로

응답을 하게 된다. 서비스를 제공하는 서버가 네트워크에 참여할시 자신의 서비스를 멀티캐스트로 공지한다.

기존 HTTP 헤더에 3개의 새로운 헤더(Object-Class, Alt-Locations, Request-ID)를 추가하였으며 PC 주변 기기(팩스, 영상 스케너, 인쇄기, ...) 를 위해 제안되었다.

⑤ GENA(General Event Notification Architecture)

IETF에서 제안하였으며, TCP/IP를 통한 HTTP 및 멀티캐스트 UDP를 사용하여 통보(notifications)를 송수신하는 기능을 제공하기 위하여 정의되었다. 또한 GENA은 이벤트 실행을 위하여 가입자 및 통보 발행자의 개념을 정의한다. 각각의 UPnP 제어 메시지는 실행 파라미터들의 집합으로 구성된 SOAP 메시지이고, 응답 또한 변수의 상태나 리턴 값을 포함하는 SOAP 메시지 이다.

⑥ SOAP(Simple Object Access Protocol)

SOAP은 웹상의 객체들을 액세스하기 위한 마이크로소프트의 프로토콜이다. 이 프로토콜은 HTTP를 사용하여 인터넷에 텍스트 명령어를 보내기 위해 XML 구문을 쓴다. SOAP은 인터넷을 통한 RPC 기반 통신의 표준이 되고 있다.

⑦ XML(eXtensible Markup Language)

W3C 정의에 의한 XML은 웹의 구조화된 데이터형식을 정의하고 있다. SGML의 부분집합이라고 할 수 있으며 SGML보다 훨씬 간결화 된 문서를 표현하는 표준이다. XML은 어떤 구조화된 데이터 형식도 텍스트 파일로 바꿀 수 있는 방법을 제시한다. XML은 태그(tage)와 속성(attribute)을 사용하는 측면에서 HTML과 많은 부분 유사하다. 사실상 의미상으로는

이 둘 간의 태그와 속성은 상당히 이질적이지만 이 둘이 사용하는 문법적 의미 내에서는 해석이 가능하다. 이러한 특징들은 다양한 문서 형식에 대한 개발 스키마(schema)로 사용되는데 유리한 점을 가지고 있다. XML은 디바이스와 서비스, 제어 메시지나 이벤트들을 기술하는데 사용된다.

2.1.4 UPnP 동작 과정

UPnP는 컨트롤 포인트와 디바이스 사이의 통신을 지원한다. 표준 인터넷 프로토콜들을 기반한 그림 3과 같은 주소할당(Addressing), 감지(Discovery), 기술문서전송(Description), 컨트롤(Control), 이벤트(Eventing), 표현(Presentation)의 6단계 과정으로 동작한다.

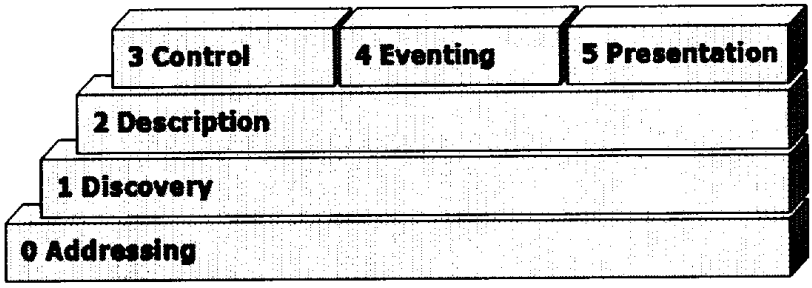

그림 3 UPnP 네트워킹 단계

⓪ 주소할당 (Addressing)

주소할당은 디바이스와 컨트롤 포인트가 IP를 부여받는 과정으로 DHCP(Dynamic Host Configuration Protocol) 서버로부터 IP를 얻거나 네트워크에 DHCP 서버가 동작하지 않으면 Auto IP를 이용하여 IP를 할당받는다. Auto IP는 이미 할당된 사설 주소 집합에서 지능적으로 IP 어드레스를 선택하는 방법을 정의하고 있다. Auto IP를 사용하면 디바이스는 169.254/16 범위 내에서 IP 주소를 자동으로 선택하게 되고 선택된 주소가 사용 중인지 아닌지를 검사한다. 만약 주소가 다른 디바이스용으로 사용

중이면, 다른 주소를 선택하여 다시 검사하게 된다. Auto IP를 사용하여 IP를 할당 받은 경우에는 주기적으로 네트워크 상에 DHCP 서버가 존재하는지를 검사하여 DHCP 서버가 감지되면 DHCP 서버로부터 IP를 다시 할당 받는다.

① 감지 (Discovery)

감지 과정은 SSDP에 의해 이루어지는데, 디바이스가 네트워크에 접속되면, SSDP는 디바이스가 가지고 있는 서비스를 네트워크 상의 컨트롤 포인트에게 통보한다. 반면 컨트롤 포인트가 네트워크에 접속하게 되면 SSDP는 네트워크 내에 관심있는 디바이스들을 검색한다.

디바이스 및 서비스 타입, XML 디바이스 기술 문서를 지시하는 URL 과 같은 디바이스나 서비스가 가지고 있는 필수 요소들이 감지 과정에서 교환되는 정보들이다.

② 기술문서전송 (Description)

컨트롤 포인트가 디바이스와의 정보교환 및 디바이스를 제어하고, 디바이스의 서비스를 사용하기 위해서는, 디바이스의 상세정보가 필요하다. 감지 메시지에 포함되어있는 URL로부터 디바이스의 기술문서를 다운받아서 디바이스의 상세정보를 얻을 수 있다.

UPnP 디바이스의 기술문서는 XML로 표현되고, 디바이스 모델 번호나 이름, 시리얼 번호, 제조사 이름, 제조사 URL 등과 같은 정보와 내포 디바이스나 서비스의 정보를 포함한다. 그림 4와 그림 5는 디바이스 기술문서와 서비스 기술문서의 예이다.

```
<?xml version="1.0" encoding="utf-8" ?>
- <root xmlns="urn:schemas-upnp-org:device-1-0">
  - <specVersion>
      <major>1</major>
      <minor>0</minor>
    </specVersion>
    <URLBase>http://172.19.4.7:59978/</URLBase>
  - <device>
      <deviceType>urn:schemas-upnp-org:device:BinaryLight:1</deviceType>
      <friendlyName>Light (KKANG74-HOME)</friendlyName>
      <manufacturer>Intel Corporation</manufacturer>
      <manufacturerURL>http://www.intel.com</manufacturerURL>
      <modelDescription>Software Emulated Light Bulb</modelDescription>
      <modelName>Intel CLR Emulated Light Bulb</modelName>
      <modelNumber>XPC-L1</modelNumber>
      <modelURL>http://www.intel.com/xpc</modelURL>
      <UDN>uuid:9cd3605c-f55b-404e-a0b4-99ea9bed7ad5</UDN>
    - <iconList>
      - <icon>
          <mimetype>image/png</mimetype>
          <width>32</width>
          <height>32</height>
          <depth>8</depth>
          <url>/icon.png</url>
        </icon>
      </iconList>
    - <serviceList>
      - <service>
          <serviceType>urn:schemas-upnp-org:service:SwitchPower:1</serviceType>
          <serviceId>urn:upnp-org:serviceId:SwitchPower.0001</serviceId>
          <SCPDURL>_SwitchPower.0001_scpd.xml</SCPDURL>
          <controlURL>_SwitchPower.0001_control</controlURL>
          <eventSubURL>_SwitchPower.0001_event</eventSubURL>
        </service>
      + <service>
      </serviceList>
    </device>
  </root>
```

그림 4 디바이스 기술문서의 예

```
<?xml version="1.0" encoding="utf-8" ?>
- <scpd xmlns="urn:schemas-upnp-org:service-1-0">
  - <specVersion>
      <major>1</major>
      <minor>0</minor>
    </specVersion>
  - <actionList>
    - <action>
        <name>GetStatus</name>
      - <argumentList>
        - <argument>
            <name>ResultStatus</name>
            <direction>out</direction>
            <relatedStateVariable>Status</relatedStateVariable>
          </argument>
        </argumentList>
      </action>
    - <action>
        <name>SetTarget</name>
      - <argumentList>
        - <argument>
            <name>newTargetValue</name>
            <direction>in</direction>
            <relatedStateVariable>Target</relatedStateVariable>
          </argument>
        </argumentList>
      </action>
    </actionList>
  - <serviceStateTable>
    - <stateVariable sendEvents="yes">
        <name>Status</name>
        <dataType>boolean</dataType>
      </stateVariable>
    - <stateVariable sendEvents="no">
        <name>Target</name>
        <dataType>boolean</dataType>
      </stateVariable>
    </serviceStateTable>
  </scpd>
```

그림 5 서비스 기술문서의 예

③ 컨트롤 (Control)

컨트롤은 컨트롤 포인트에서 디바이스의 제어를 위한 것으로 SOAP(Simple Object Access Protocol)를 사용하여 XML 형태의 컨트롤 메시지만을 전달하는 기법을 사용한다. 디바이스의 서비스에 대한 URL정보는 디바이스 기술문서에 포함되어 있으며 서비스 제어 URL에게 SOAP

제어 메시지를 송신한다. 제어 메시지에 대한 응답으로 서비스는 기능 수행에 따른 특별한 값을 리턴하거나 오류코드를 리턴한다.

④ 이벤트 (Event)

이벤트는 컨트롤 포인트와 디바이스 간에 이벤트처리를 위한 프로토콜로 GENA를 사용한다. UPnP 서비스 기술문서에는 서비스 액션 리스트와 상태변수 리스트가 포함되어 있으며, 서비스는 변수의 변화가 생겼을 경우 이 변화를 컨트롤 포인트에 알려야 하며, 컨트롤 포인트는 이러한 메시지를 수신해야 한다. 이벤트 메시지는 이러한 상태변수에 대한 이름과 현재 값을 포함한다.

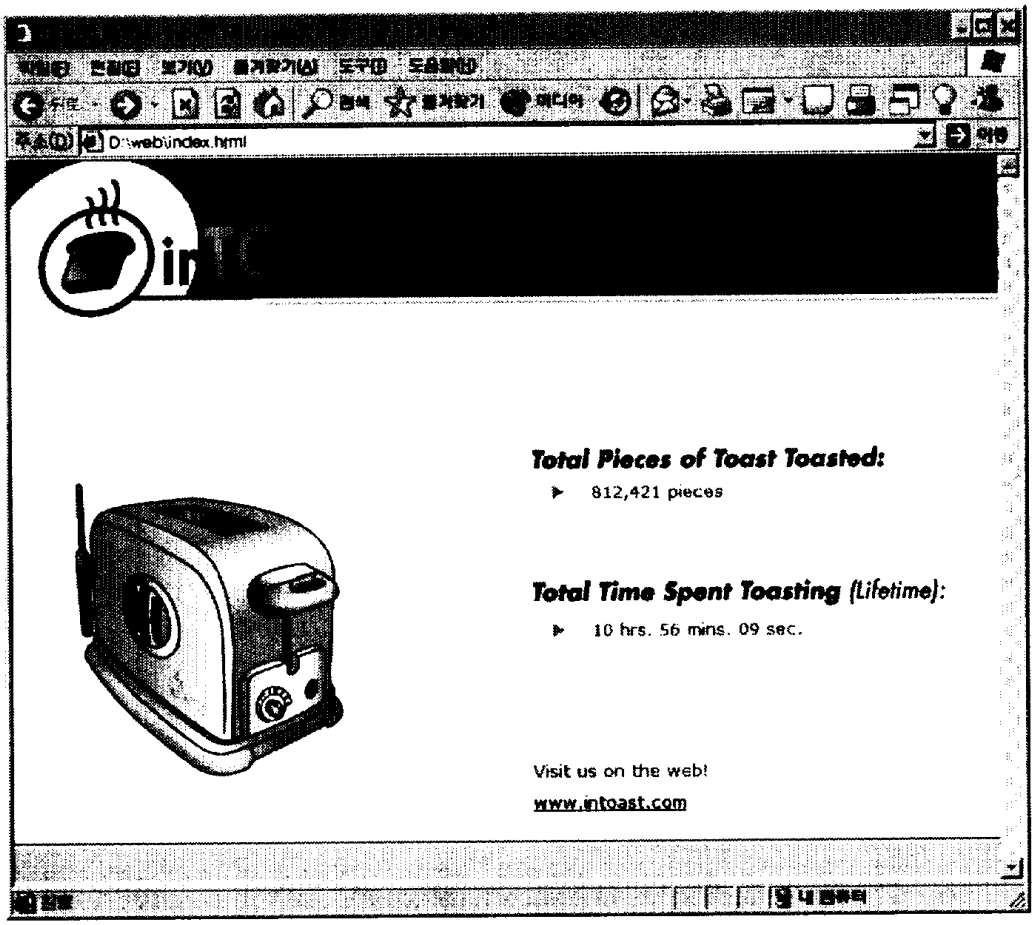

그림 6 프레젠테이션 페이지의 예

⑤ 프레젠테이션 (Presentation)

디바이스가 프레젠테이션에 대한 URL을 가지고 있을 경우 URL로 접근하여 웹페이지를 로드 할 수 있다. 프레젠테이션을 위한 웹페이지는 디바이스 제조업체가 제공하는 웹페이지로 디바이스 제조업체가 제공하는 방법에 따라 디바이스를 제어하고, 디바이스의 상태를 확인 할 수 있다. 그림 6은 UPnP 토스트 디바이스가 제공하는 프레젠테이션 페이지의 예이다.

그림 7은 UPnP 동작과정을 도시한 그림이다.

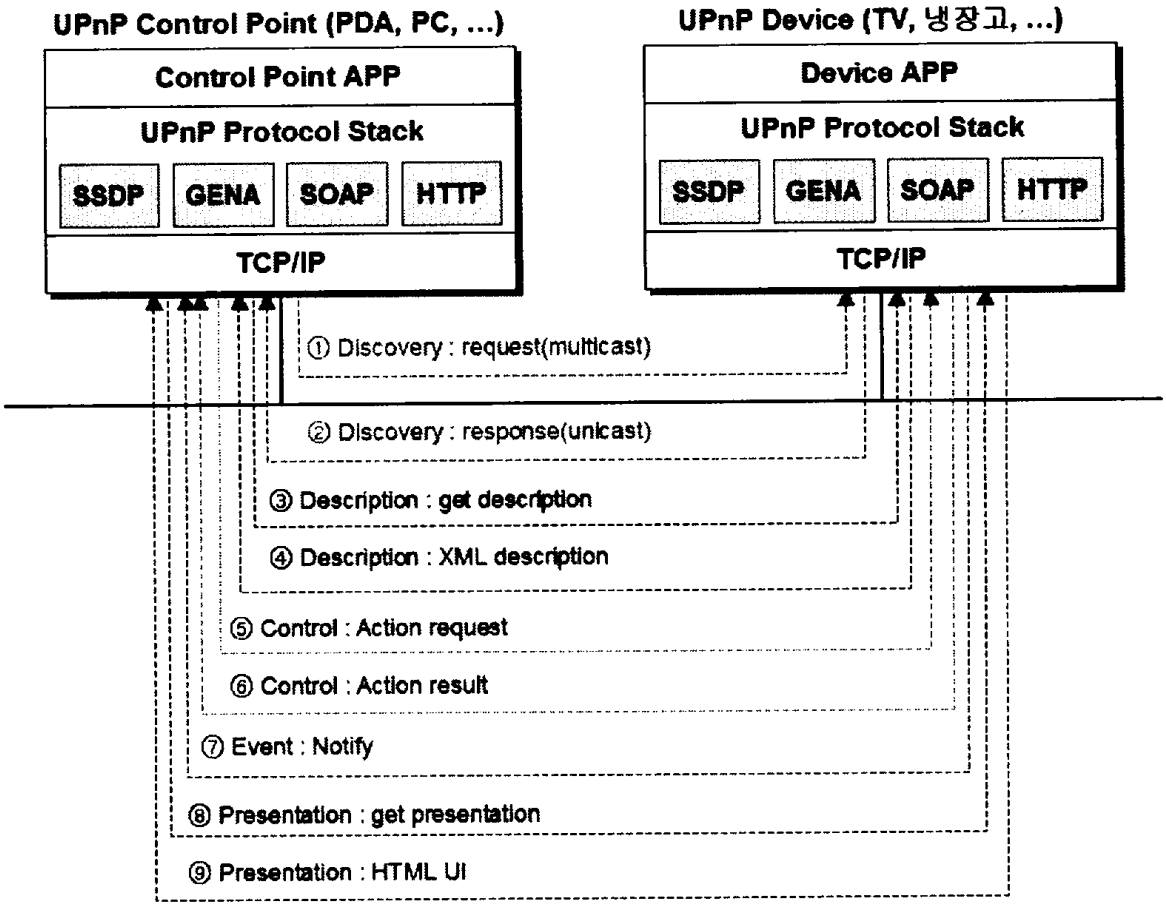

그림 7 UPnP 동작 과정

2.2 Jini

2.2.1 Jini 개요

Jini는 Sun Microsystems사가 제안한 기술로서 자바를 기반으로 다양한 네트워크 솔루션으로 연결된 장치들을 동적으로 상호 작용하게 하는 분산 미들웨어 기술이다. Jini는 네트워크 상에서 플러그 앤 플레이 기능을 제공하며, 모든 요소를 서비스로 처리하는 "서비스 기반" 구조를 활용한다.

실행 시에 네트워크 상에 분산되어 있는 구성 요소들을 탐지하여 동적으로 시스템을 구성하는 구조이다. JVM(Java Virtual Machine)을 도입함으로 인해 하드웨어에 독립적이며, 단순한 네트워크 구성을 제공한다. Jini는 RMI(Remote Method Invocation) 통신 방법을 기반으로 하고 있다[14].

2.2.2 Jini의 구성요소

Jini 시스템은 3가지 구성요소로 이루어진다.

① Jini 서비스(Jini Service)

디스크, 프린터, 팩스, 디스플레이와 같은 디바이스, 어플리케이션 또는 유틸리티와 같은 소프트웨어, DB 등 서비스를 제공하는 요소

② Jini 클라이언트(Jini Client)

서비스가 제공하는 서비스를 사용하고자 하는 요소로서 TV 리모콘, PDA, PC 등과 같다.

③ 룩업 서비스 (Lookup Service)

서비스와 클라이언트간의 중계역할 및 검색을 위한 요소로 Jini 서비스와 직접 통신할 수 있는 프록시 코드를 Jini 서비스로부터 등록 받아

보관하고 있다가 클라이언트의 서비스 요청이 있으면 해당 서비스의 프록시 코드를 클라이언트에 전송하여 프록시 코드를 통해 Jini 서비스와 Jini 클라이언트가 직접 통신이 가능하도록 하는 역할을 한다.

위 세 가지 구성요소들은 그림 8과 같이 네트워크에 연결된다. Jini 서비스에서 제공하는 프록시(Proxy)의 실행코드가 객체 직렬화(Object Serialize), 정렬화(Marshalling)되어 룩업 서비스에 일차적으로 등록되고 클라이언트가 필요시에 룩업 서비스로부터 다운된다. 프록시 코드는 RMI(Remote Method Invocation)를 이용하여 다른 가상머신 상에서 실행중인 객체의 메서드를 호출 할 수 있게 된다.

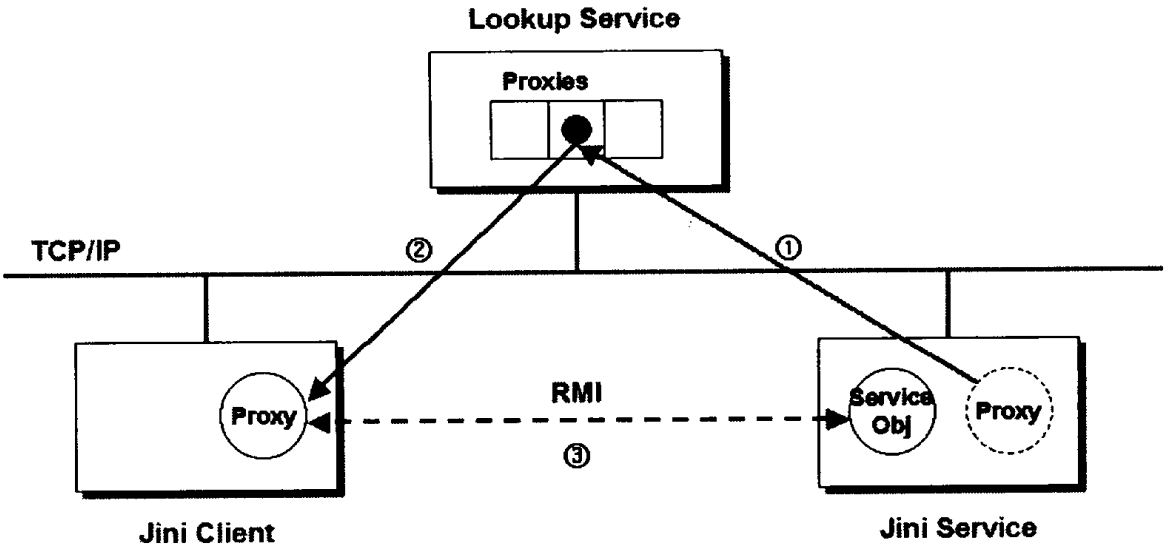

그림 8 Jini의 기본 구조

2.2.3 Jini 동작 과정

Jini의 동작의 핵심은 감지(Discovery), 조인(Join), 룩업(Lookup) 이다.

감지는 서비스가 등록할 룩업 서비스를 찾기 위한 과정이다. 조인은 서비스가 룩업 서비스에 자신의 프록시를 등록하는 과정이고 룩업은 클라이언트가 룩업 서비스에 등록된 서비스의 인터페이스 타입을 통해 서비스를 찾고 서비스의 프록시를 얻어오는 과정이다.

① 감지(Discovery) 과정

서비스는 클라이언트에게 서비스를 제공하기 위해 룩업서비스에 자신의 프록시를 등록하는 과정을 거쳐야 한다. 이를 위해 룩업 서비스를 찾아야 한다. 룩업 서비스의 위치를 이미 알고 있는 경우 유니캐스트를 사용하여 TCP 연결을 맺고, 위치를 모르는 경우는 UDP Multicast Request를 통해 룩업 서비스로부터 룩업 서비스의 위치를 응답 받고 TCP 연결을 맺는다.

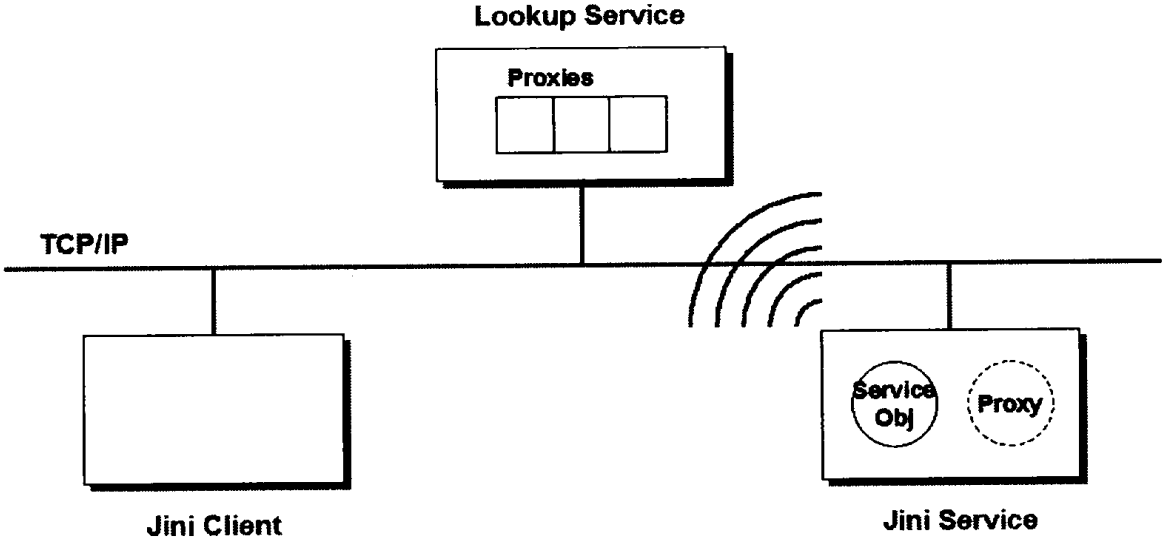

그림 9 Jini의 Discovery 과정

② 조인(Join) 과정

룩업 서비스는 Jini의 디폴트 포트인 4160번 포트로 서비스의 요청을 대기하고 있다가 요청이 있으면 요청한 서비스에게 registrar 객체를 전달한다. registrar 객체는 룩업 서비스의 프록시 역할을 하는 객체로, 이 registrar 객체를 통해 서비스는 자신의 서비스 프록시를 룩업 서비스에 등록 한다. 실제 이 과정은 RMI를 통해 이루어진다.

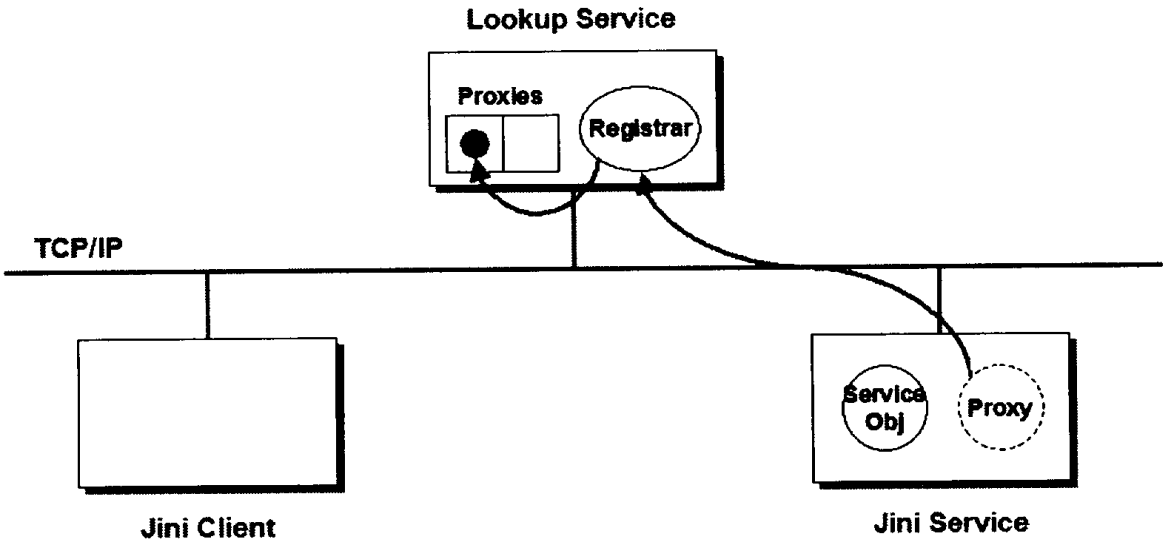

그림 10 Jini의 Join 과정

③ 룩업(Lookup) 과정

서비스를 사용하는 클라이언트도 서비스를 사용하기 위해서는 우선 룩업 서비스를 찾는 Discovery 과정을 거쳐야 한다. 메커니즘은 서비스와 룩업 서비스의 사이의 Discovery와 동일하다.

룩업 서비스는 서비스 요청과 마찬가지로 클라이언트에 registrar를 넘겨준다. 클라이언트는 이 registrar를 통해 원하는 서비스를 검색하고, 서비스가 제공하는 프록시 실행코드를 다운받는다.

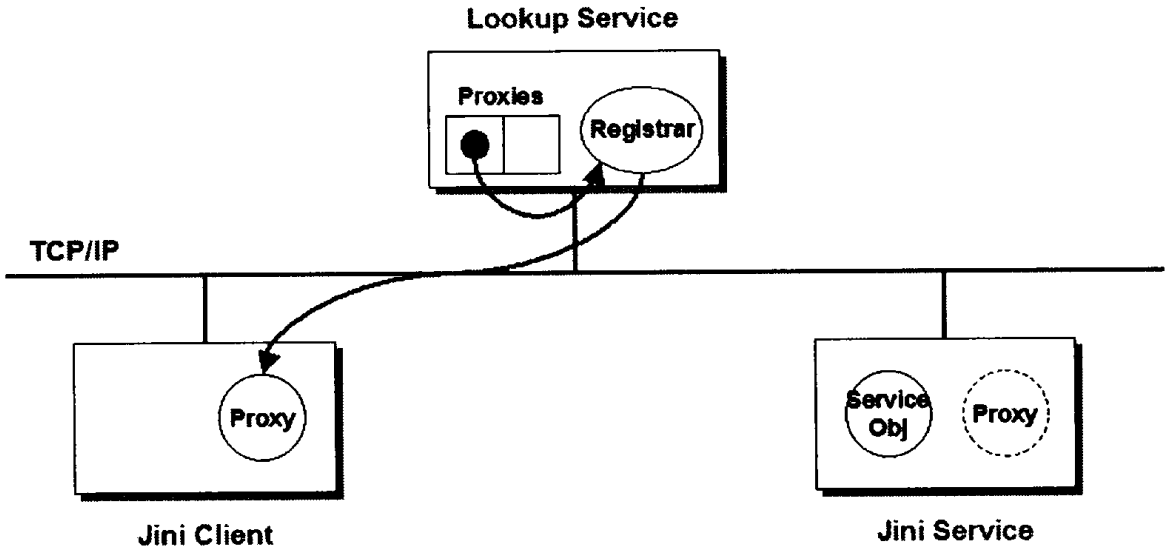

그림 11 Jini의 Lookup 과정

④ 서비스 호출 과정

룩업 서비스로부터 서비스의 프록시를 다운받는 클라이언트는 그림 12와 같이 서비스 프록시를 통해 RMI 방식을 이용하여 서비스와 직접 통신하여 서비스를 제공 받게 된다.

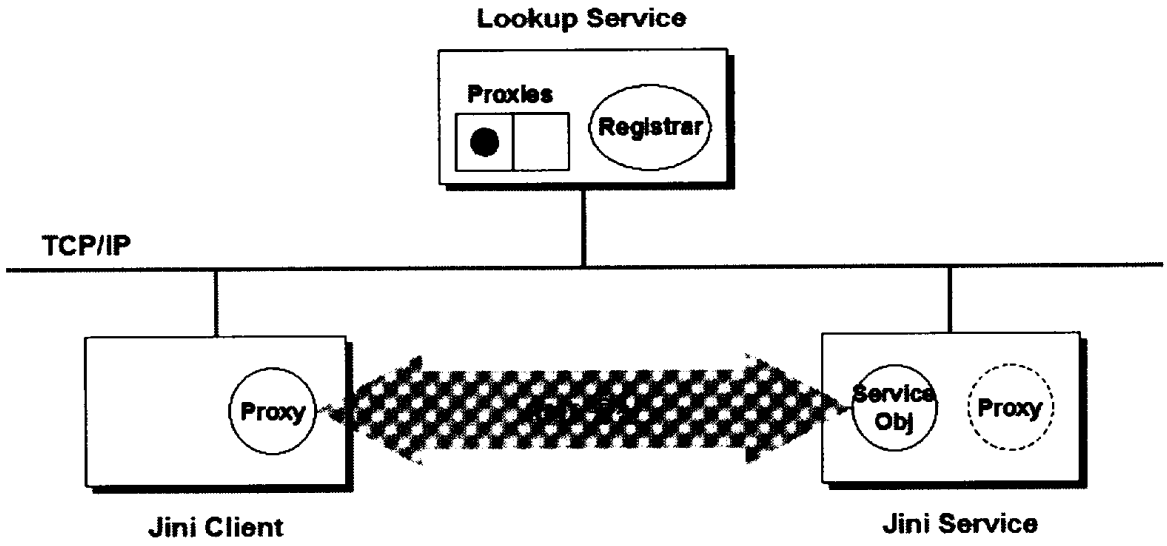

그림 12 Jini의 서비스 호출과정

2.3 OSGi

2.3.1 Home Gateway

홈 게이트웨이는 다양한 홈 네트워크 기술과 외부의 초고속 액세스 망 기술을 연결시켜 주는 장치이다. 홈 네트워크의 상호운용 서비스를 효과적으로 제공하기 위해서는 홈 게이트웨이가 필요하다. 홈 게이트웨이의 주요 기능은 네트워크로 연결된 PC 및 정보가전기기 들을 통합 관리, 제어하는 역할과 내부 네트워크와 외부 인터넷 망의 연결 인터페이스를 제공하고 내부의 기기를 제어할 수 있는 모듈을 제공하는 일이다.

현재 홈 네트워크 시장은 다양한 유·무선 네트워크 기술을 기반으로 다양한 홈 게이트웨이 및 홈 네트워킹 미들웨어 들이 난립하고 있으며, 서로 다른 하드웨어 플랫폼, OS 등 수많은 서비스 개발환경이 존재한다. 홈 게이트웨이의 중요성을 인식한 많은 가전기기 업체, 통신기기 업체, 컴퓨터 기기 업체, 원격 제어 서비스 업체들은 홈 게이트웨이 표준을 만들고 있다.

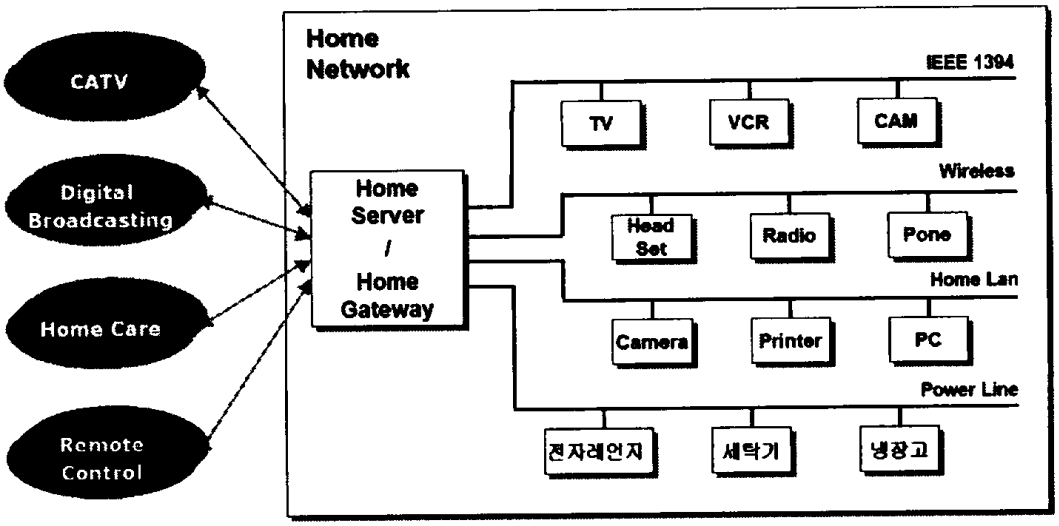

그림 13 홈 네트워크 구성도

2.3.2 OSGi 개요

OSGi(Open Service Gateway Initiative)는 WAN상의 통신 서비스를 가정 내의 LAN과 정보 가전기기 간에 연결하여 정보를 교환하기 위한 개방형 게이트웨이의 규격을 제정하기 위해 설립된 위원회이다. 1999년 3월에 설립되었다. 2000년 5월 스펙 1.0을 공식 발표하였으며 현재 스펙 3.0이 발표되었다. 그림 14는 OSGi 게이트웨이로 구성 가능한 홈 네트워크 모델을 보여준다.

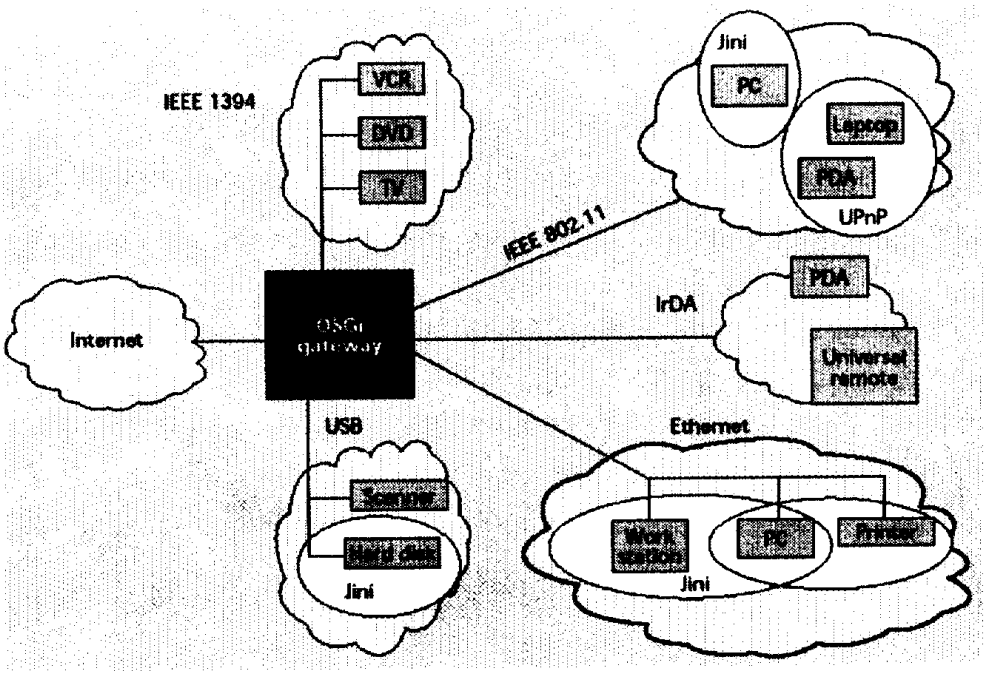

그림 14 OSGi 게이트웨이 구성도

2.3.3 OSGi Framework

OSGi 서비스의 핵심은 프레임워크(Framework)이다. 프레임워크는 번들(Bundle)이라 불리는 확장 가능하고 다운로드 가능한 자바 어플리케이션을 관리하고 지원하는 자바 플랫폼이다[15].

그림 15의 구조를 갖는 프레임워크는 디바이스들의 번들을 설치할 수

있으며, 더 이상 사용하지 않을 경우 제거할 수 도 있다. 설치된 번들은 서비스로 등록되며, 프레임워크의 관리 하에 다른 번들과 공유될 수 있다. 또 프레임워크는 번들의 업데이트를 동적으로 관리하고, 번들과 서비스 사이의 의존성을 관리한다.

프레임워크는 번들이 프레임워크 서비스 레지스트리를 통해 서비스를 등록하고, 서비스들의 상태에 대한 정보를 받고, 장치의 현재 기능에 맞는 서비스를 찾는 기능을 제공한다.

프레임워크는 이러한 패러다임을 지원하기 위한 메커니즘을 번들 개발자에게 제공한다. 이 메커니즘들은 프로그래밍 모델과 함께 개발자가 빠르게 목적을 이룰 수 있도록 간단하게 디자인 되었다.

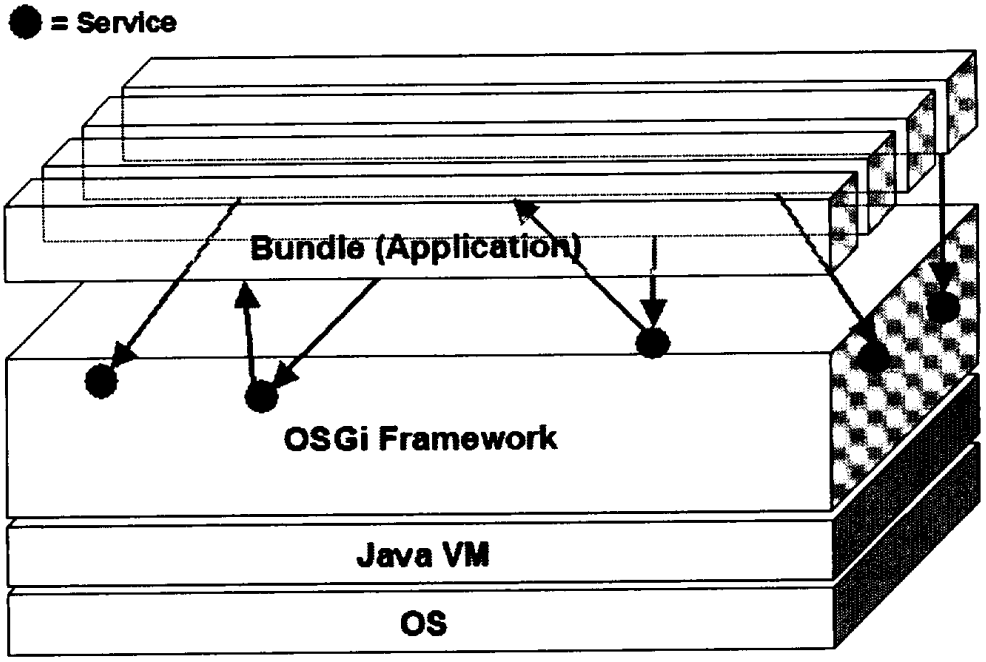

그림 15 OSGi 프레임워크

2.3.4 Bundle

번들은 OSGi 환경에서의 구동되는 유일한 어플리케이션이다. 즉 프레임워크에서 구동되는 유일한 어플리케이션이다. 번들은 자바 클래스들과 여러 리소스들로 구성되어있으며 다른 번들에게 기능을 제공하는 서비스도 포함할 수 있다. 번들은 그림 16과 같은 JAR(Java ARchive)파일 형태로 구성된다.

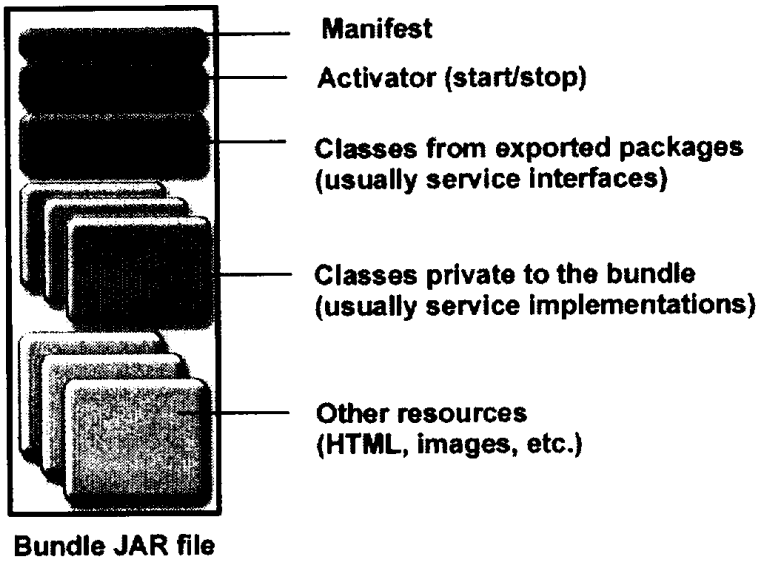

그림 16 번들 JAR 파일의 구조

JAR 파일은 파일의 내용 명세와 번들에 대한 정보를 담고 있는 Manifest 파일을 포함한다. 그림 17과 같은 Manifest 파일은 프레임워크가 번들을 설치하고 활성화시키기 위해 필요한 정보와 다른 번들과 서비스를 공유하기 위한 정보 등을 담고 있다. 번들이 서비스를 다른 번들에게 제공하기 위해서는 Export-package 항목에 패키지의 이름을 명시해야 하며, 다른 번들의 서비스를 이용하기 위해서는 이용하기 위한 패키지 이름을 import-package 항목에 명시해야 한다. 번들은 프레임워크에서 실행되기 위해 BundleActivator라는 특별한 인터페이스를 구현하여야 한다.

```
Manifest-Version: 1.0
Bundle-Vendor: IBM Pervasive Computing
Bundle-Version: 2.1.2
Bundle-Activator: com.ibm.osg.service.http.HttpBundleActivator
Bundle-Copyright: Licensed Materials - Property of IBM. (C) Copyright IBM Corp.
                  1999, 2002. All Rights Reserved.
                  IBM is a registered trade mark of IBM Corp.
Build-Information: Release smf built Wed Jul 16 2003 13.33.27
                   CDT on bwbld000.austin.ibm.com
Bundle-DocUrl: http://www.ibm.com/pvc
Created-By: 1.3.1 (IBM Corporation)
Import-Service: org.osgi.service.log.LogService
Bundle-Name: HttpService
Bundle-ContactAddress: pervasive@us.ibm.com
Export-Service: org.osgi.service.http.HttpService
Export-Package: com.ibm.osg.socket; specification-version=1.0, com.ibm.osg.socket.https
DynamicImport-Package: javax.net
Bundle-Description: IBM HttpService (HTTP/1.0 Servlet 2.1 Engine).
Import-Package: org.osgi.service.http; specification-version=1.1,
                org.osgi.service.log; specification-version=1.0,
                org.osgi.util.tracker; specification-version=1.1,
                org.osgi.framework; specification-version=1.0,
                com.ibm.osg.socket; specification-version=1.0,
                com.ibm.osg.socket.https,
                com.ibm.pvc.resman; specification-version=1.0.0,
                javax.servlet; specification-version=2.1,
                javax.servlet.http; specification-version=2.1,
                org.osgi.service.cm; specification-version=1.0
```

그림 17 Manifest 파일의 예

번들은 그림 18과 같은 6가지의 상태변화를 가지며 각 상태는 다음과 같은 의미를 가진다.

① INSTALLED

번들이 프레임워크에 설치된 상태.

② RESOLVED

번들이 필요로 하는 모든 자바 클래스들과 네이티브 코드(Native Code)를 사용할 수 있는 상태.

③ STARTING

번들이 시작되고 있는 상태, BundleActivator의 start 메서드가 실행되고

있는 상태이며 시작과정이 완료되지 않은 상태.

④ STOPPING

번들이 정지되고 있는 상태, BundleActivator의 stop 메서드가 실행되고 있는 상태이며 완전히 정지과정이 완료되지 않은 상태.

⑤ ACTIVE

번들이 완전히 시작되고 프레임워크 상에서 활성화된 상태.

⑥ UNINSTALLED

번들이 프레임워크에서 제거된 상태.

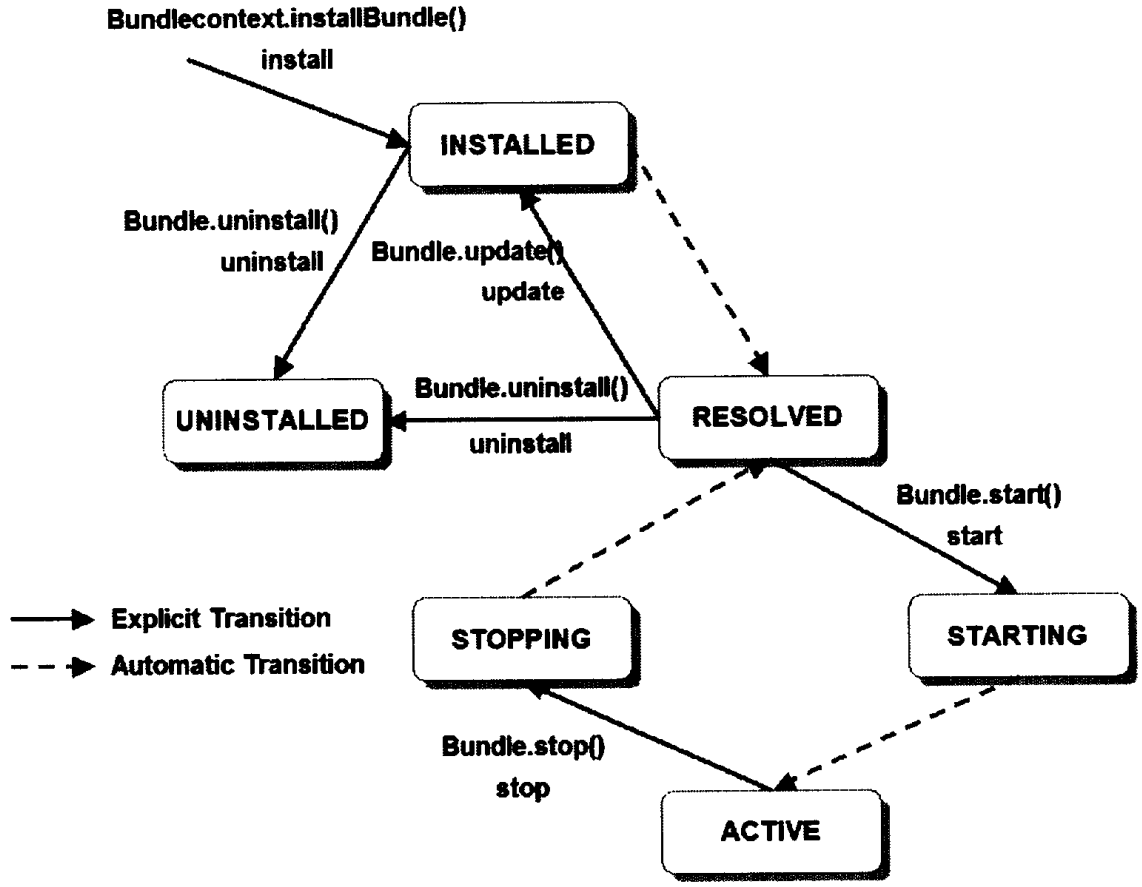

그림 18 번들의 상태 변이도

3. UPnP & Jini 홈 네트워크 구성

OSGi 기반의 홈 게이트웨이를 설계하기 전에 먼저 UPnP, Jini 미들웨어를 이용하여 그림 19와 같은 홈 네트워크 모델을 구성한다.

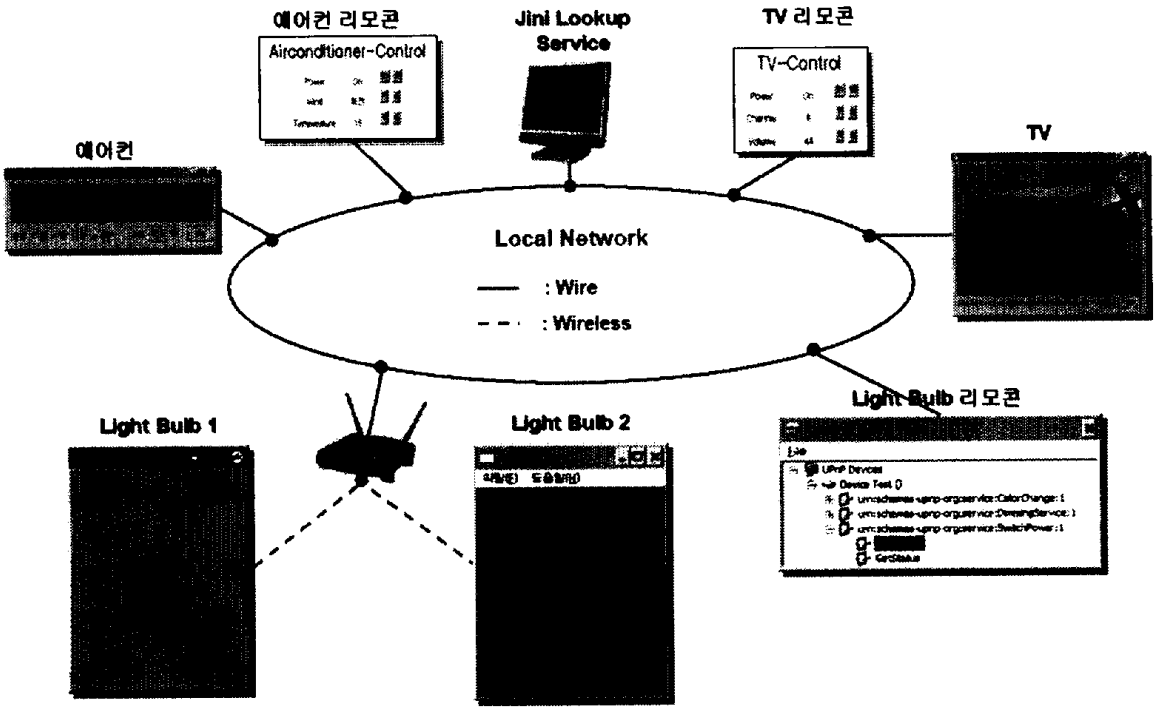

그림 19 UPnP & Jini 홈 네트워크

그림 19의 UPnP, Jini 로 구성된 홈 네트워크는 게이트웨이 없이 내부 네트워크 만으로 구성하였으며, 각 Jini 디바이스와 UPnP 디바이스를 에뮬레이션 하기 위한 에뮬레이터가 설치된 PC와 PDA로 구성된다. Jini는 TV와 에어컨 디바이스, UPnP는 Light Bulb 디바이스를 에뮬레이션 하였으며 홈 네트워크를 구성하기 위한 시스템의 사양과 각 디바이스의 개발 사양은 표 1, 표 2와 같다.

표 1 개발사양

에뮬레이션	기기	미들웨어	API/SDK	개발 툴
Light Bulb1	PDA	UPnP	Intel UPnP SDK	eMbedded Visual C++
Light Bulb1	PC	UPnP	Intel UPnP SDK	Visual Studio .NET
Light Bulb 리모콘	PC	UPnP	Intel UPnP SDK	Visual Studio .NET
TV	PC	Jini	JDK 1.4	JBuilder 7.0
에어컨	PC	Jini	JDK 1.4	JBuilder 7.0
TV 리모콘	PC	Jini	JDK 1.4	JBuilder 7.0
에어컨 리모콘	PC	Jini	JDK 1.4	JBuilder 7.0

표 2 시스템 사양

디바이스		사 양
PDA	Model	Compaq IPAQ H3950
	OS	PocketPC 2002
	Network	AirGate Wireless Lan-11Mbps
PC	Model	Pentium-IV 1.7GHz
	OS	Windows XP
	Network	WLAN USB 11Mbps or 3com ethernet 100Mbps

3.1 Jini device emulator

Jini 디바이스를 구현하기 위해서는 먼저 Jini 디바이스가 제공하는 서비스의 종류에 따라 인터페이스를 구현한다. 예를 들어 프린터의 디바이스인 경우 프린터의 기능을 정의한 인터페이스를 구현한 프록시를 룩업 서비스에 등록 한다. 프린터를 사용하고자 하는 클라이언트는 프린터 인터페이스를 구현한 서비스를 룩업 서비스로부터 검색하여 서비스에 대한 프록시를 가져와 프린터를 사용 할 수 있다. 본 논문에서 구성한 네트워크에는 Jini

를 사용하여 TV와 에어컨을 프로그램으로 에뮬레이션한 에뮬레이터와 클라이언트로 구성서된다. TV 에뮬레이터는 기본적인 TV의 기능인 전원, 채널, 볼륨 제어기능을 에뮬레이션하며, 에어컨 에뮬레이터는 전원, 온도, 풍향 제어를 에뮬레이션한다.

3.2 UPnP device emulator

UPnP 디바이스 에뮬레이터와 클라이언트는 Intel에서 제공하는 SDK를 이용하여 구현되었다. Intel SDK는 그림 20처럼 UPnP에서 사용되는 모든 프로토콜과 미니 웹서버, XML 파서를 제공한다.

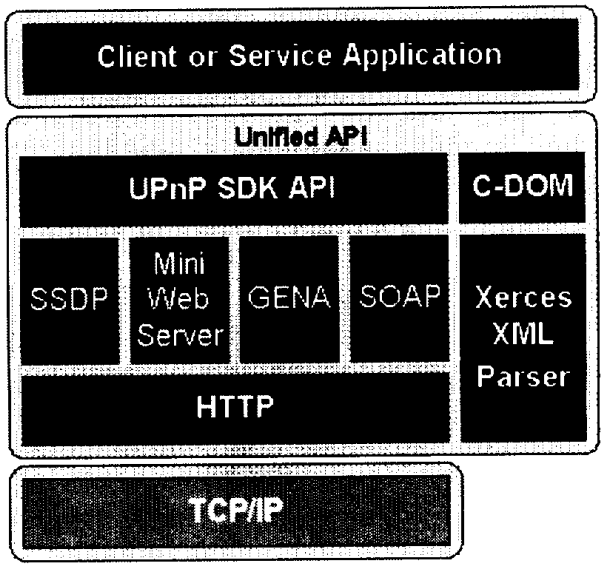

그림 20 UPnP SDK 구조

Light Bulb 에뮬레이터는 전구의 색상과 밝기를 변화시키는 서비스를 제공하며, 유·무선 네트워크 환경과 운영체제에 따른 성능 비교를 위해 무선 네트워크 카드를 탑재한 PDA(WinCE) 버전과 PC(windows XP)버전으로 구현되었으며, 에뮬레이터를 제어하기 위한 컨트롤 포인트를 구현하였다.

4. 번들 구현

앞에서 구현된 네트워크를 OSGi 게이트웨이와 연동하기 위해서는 각 Jini 디바이스와 UPnP 디바이스를 제어하는 클라이언트를 OSGi 프레임워크에서 구동될 수 있도록 번들화 하는 작업이 필요하다. 그림 21은 두 클라이언트를 게이트웨이를 통하여 외부에서 제어할 수 있도록 만든 모델이다.

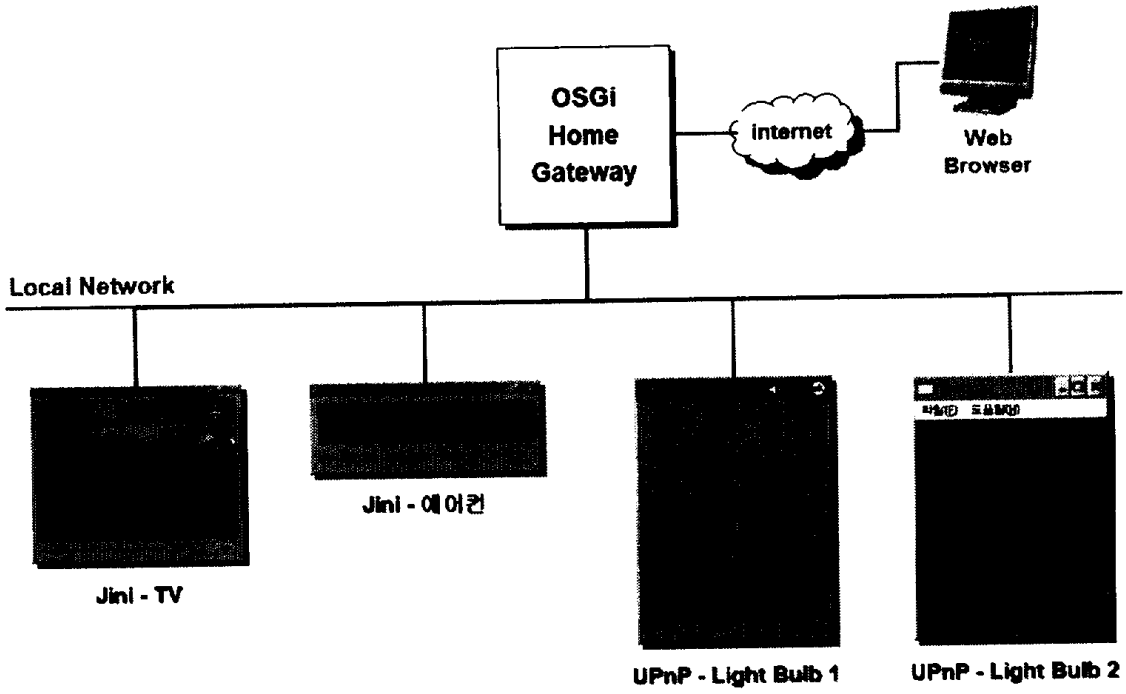

그림 21 UPnP & Jini 홈 네트워크와 OSGi 게이트웨이

4.1 Jini 제어 번들

Jini가 OSGi 환경에서 실행되기 위해서는 Jini 클라이언트를 번들로 만드는 일 이외에 Jini 실행환경에 필요한 RMI 데몬, 룩업서비스 등을 OSGi 프레임워크에서 실행될 수 있도록 번들로 만들어주는 작업이 필요하다. 번들화 되어 실행시 OSGi 프레임워크 밖에서 실행할 경우와 큰 차이점은

없지만 프레임워크에서 제어되어 다른 번들과 같은 라이프 사이클을 갖게 된다. Jini 제어 번들은 표 3과 같은 주요 파일들로 이루어진다.

표 3 Jini 제어 번들의 주요 파일

파일 이름	기 능
meServlet.java	HttpServlet을 상속받은 서블릿 클래스
TV_Activator.java	번들 엑티베이터 클래스
TV_ServiceInterface.java	서비스 인터페이스
TV_Service.java	서비스 클래스
TV_ServiceProxy.java	서비스 프록시 클래스

Jini 디바이스를 OSGi를 통해 외부에서 웹기반으로 제어하기 위한 방법으로는 Jini 디바이스의 서비스 타입과 인터페이스에 따라 동적으로 웹페이지 UI(User Interface)를 구성하여 제어하는 방법과 디바이스가 제공하는 최적화된 웹페이지 UI를 이용하여 제어하는 방법이 있다. 전자가 이상적인 방법이나, 본 논문에서는 후자의 방법으로 Jini 제어 웹페이지를 생성하였다. Jini 번들은 TV를 제어하기 위한 번들과 에어컨을 제어하기 위한 번들, 2개의 번들로 구성되며, 역할은 두 디바이스가 룩업서비스에 등록해 놓은 프록시를 통해 각 디바이스를 제어할 수 있는 UI를 가진 웹페이지를 만들어내는 서블릿을 생성하여 동적으로 OSGi 게이트웨이의 HTTP 번들에 등록하는 역할을 한다.

번들 작성 후 JAR 파일의 내용과 번들에 대한 내용을 기술하고 있는 그림 22와 같은 manifest 파일을 작성 한다. manifest 파일을 만들고 나서 manifest 파일과 번들의 모든 클래스 파일, 리소스 파일을 JAR 파일로 패킹한다.

그림 23은 번들이 OSGi 프레임워크에서 실행되기 위해 필요한 BundleActivator 코드의 일부이다.

```
Manifest-Version: 1.0
Bundle-Vendor: uqiq.hansung.ac.kr
Bundle-Version: 1.0
Bundle-Activator: TV_Activator
Bundle-Copyright: Division of Computer System Engineering, Hansung University
Bundle-DocUrl: uqiq.hansung.ac.kr
Created-By: kkang74[kkang74@hansung.ac.kr]
Bundle-Name: TV Client Servlet
Import-Service: org.osgi.service.http.HttpService, org.osgi.service.log.LogService
Import-Package: javax.servlet; specification-version=2.1,
                javax.servlet.http; specification-version=2.1,
                org.osgi.framework; specification-version=1.0,
                org.osgi.service.http; specification-version=1.1,
                org.osgi.service.log; specification-version=1.0,
                org.osgi.util.tracker; specification-version=1.0,
Bundle-ContactAddress: kkang74@hansung.ac.kr
Bundle-Description: A servlet for Jini TV
```

그림 22 Jini 제어 번들의 Manifest 파일

```
import org.osgi.framework.*;
import org.osgi.service.http.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TV_Activator implements BundleActivator {

    BundleContext bc = null;
    private ServiceReference httpRef = null;
    HttpService httpSvc = null;
    meServlet ser = null;

    public TV_Activator() {}

    public void start(BundleContext context) throws Exception {
        bc = context;
        System.out.println("Start jini TV Client ");
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
        ser = new meServlet();
        servletRegister();
    }

    public void stop(BundleContext context) {
        httpSvc.unregister("/tv");
        httpRef = null ;
        httpSvc = null ;
    }

    public void servletRegister() {
        httpRef = bc.getServiceReference("org.osgi.service.http.HttpService");
        httpSvc = (HttpService)bc.getService(httpRef);
        if (httpSvc != null) {
            try {
                HttpContext hc = new servletTest();
                httpSvc.registerServlet("/tv", ser, null, hc );
            } catch (Exception ex) {}
        }
    }
}
```

그림 23 Jini 번들의 Activator 클래스

그림 24는 게이트웨이를 통해 외부에 제어 웹페이지를 보여주기 위한 서블릿 생성 코드의 일부이다.

```
import javax.servlet.*;
import javax.servlet.http.*;

public class meServlet extends HttpServlet {
    PrintWriter out;
    ServiceRequest sr = null;
    List songList = null;
    TV_ServiceInterface TVService = null;

    public meServlet() {
        sr = new ServiceRequest();
    }
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        TVService = sr.getTVService();
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        int v = TVService.getVolume();
        int c = TVService.getChannel();
        String p;
        response.setContentType("text/html; charset=euc-kr");
        out = new PrintWriter(response.getOutputStream(), true);
        out.println("<html>");
        out.println("<head>");
        out.println("<title>TV-Control</title>");
        out.println("</head>");
        out.println("<body>");
        ....
        ....
        out.println("</body>");
        out.println("</html>");
        out.flush();
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html; charset=euc-kr");
        String Pdata= "";
        out = new PrintWriter(response.getOutputStream(), true);
        if ( (Pdata = request.getParameter("Pdata")) != null) {
            if (Pdata.equals("powerOn")) TVService.setPower(true);
            else if (Pdata.equals("powerOff")) TVService.setPower(false);
            else if (Pdata.equals("channelUp")) TVService.incChannel();
            else if (Pdata.equals("channelDown")) TVService.decChannel();
            else if (Pdata.equals("volumeUp")) TVService.incVolume();
            else if (Pdata.equals("volumeDown")) TVService.decVolume();
        }
        doGet(request, response);
    }
}
```

그림 24 Jini 제어 서블릿 클래스

4.2 UUCA(Unified UPnP Control Agent) 번들

UPnP 디바이스를 제어하는 클라이언트 즉, 컨트롤 포인트를 번들 화하여 OSGi 프레임워크에서 실행되도록 구현한다. 표 4는 UUCA 번들을 구성하는 파일 중 주요 파일이다.

표 4 UUCA 번들의 주요 파일

파일 이름	기 능
NativeClass.java	JNI Call을 위한 클래스 정의
meServlet.java	HttpServlet을 상속받은 서블릿 클래스
XmlConstants.java	XML 태그 및 속성에 대한 상수값 정의
UPnPCP_Activator.java	번들 엑티베이터 클래스
UPnPControlPoint.java	컨트롤 포인트 클래스
UPnPDevice.java	디바이스 클래스
UPnPService.java	서비스 클래스
UPnPAction.java	서비스 액션 클래스
UPnPArgument.java	액션 매개변수 클래스
UPnPStateVariable.java	디바이스 상태변수 클래스
upnpcontorlpoint.dll	C언어로 작성한 컨트롤 포인트 라이브러리

앞에서 구현한 컨트롤 포인트를 코드의 재사용을 위해 DLL(Dynamic Linking Library) 형태로 변환하였다. OSGi 번들은 자바 코드로 구현되어야 하므로 컨트롤 포인트의 C 코드와 번들 자바 코드 사이에 JNI(Java Native Interface)를 사용하여 구현한다.

JNI란 자바에 대한 native 프로그래밍 인터페이스라고 할 수 있다. JNI는 자바 코드에서 C/C++, assembly에서 제공하는 라이브러리나 작성한 프로그램을 호출할 수 있는 기능을 제공한다. 그리고 Invocation API를 이용해서 JVM을 native 프로그램 내에 포함시킬 수 도 있다. JNI는 주로 다음과 같은 경우에 필요하게 된다.

① 표준 자바클래스 라이브러리에서 필요한 기능이 제공되지 않을 경우

② 이미 필요한 기능을 native 프로그램이나 라이브러리로 작성해둬서, Java로 다시 작성하기엔 비효율적일 경우

③ 전체 구조는 Java로 구성했으나 어떤 부분에서는 성능의 효율성 문제로 Java로 작성하는 것 보다 native 프로그램으로 작성하는 것이 바람직할 경우

JNI를 이용해서, native 프로그램에서 Java의 객체를 쉽게 사용할 수 있다. 객체의 생성, 갱신, 접근 등의 작업을 모두 수행할 수 있다. native 프로그램에서 자바 메소드도 쉽게 호출할 수 있다. 요구되는 인자를 넘겨줄 수 있고, 또 메소드 수행이 완료되었을 때 리턴값도 돌려 받을 수 있다. JNI를 이용하면 Java의 장점들을 native 프로그램에서도 사용할 수 있다. 그 대표적인 것으로, catch와 throw를 들 수 있다. 또, JNI 함수를 호출하게 되면 객체의 정보를 받아볼 수 있다.

결과적으로, JNI는 native 프로그램과 Java 프로그램을 묶어주는 역할을 한다. 다음 그림은 JNI가 C와 Java를 어떻게 묶어주는지를 보여주는 그림이다.

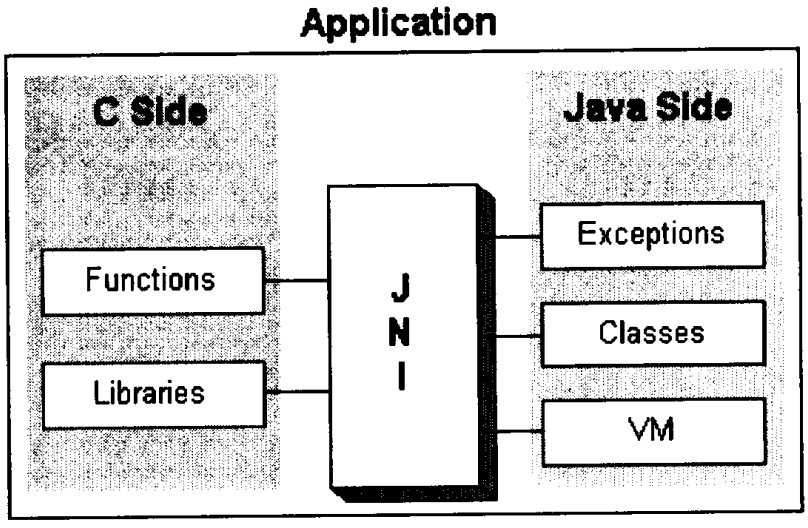

그림 25 JNI의 역할

그림 26, 27, 28은 JNI를 사용하기 위해 UUCA 번들에서 구현된 코드의 일부이다.

```
package cse.nansung.upnpcontrolpoint;

public class NativeClass {
    public native void NativeStart();
    public native void SayHelloWorld();
    public native void NativeInvoke(String SCPDURL, String actionName,
    String arg);
    public native boolean NativeGetUpdate(UPnPCommonControlPoint cp);
    public native void registCallBack(UPnPCommonControlPoint cp);
    public native String NativeGetDevicesLocationURLs();
    public native String[] NativeGetSCPDURL(String deviceURL);
    public native String[] NativeGetActionName(String SCPDURL);
    public native String[] NativeGetVariableAtt(String SCPDURL,
    String variableName);
    public native String[] NativeGetStateVariable(String SCPDURL);

    NativeClass(){
    }

    static {
        System.loadLibrary("UPnPControlPoint");
    }
}


public class UPnPControlPoint {
    ...
    private NativeClass n = new NativeClass();;
    ...

    public UPnPControlPoint(DocumentBuilder db) {
        ...
        n.NativeStart();
        ...
    }
    ...
    public void invokeAction(String scpd, String actionName, String argName, String a){
        String arg = "<name>" + argName + "</name><value>" + a + "</value>";
        n.NativeInvoke(scpd, actionName, arg);
    }

    public NativeClass getNativeClass() {
      return this.n;
    }
    ...
}
```

그림 26 Native Call을 위한 자바 코드

그림 26의 NativeClass는 공유 라이브러리(UPnPControlPoint.dll)를 실행 시 로드한다. UPnPControlPoint 클래스는 NativeClass의 인스턴스를 생성하고 공유 라이브러리의 함수를 이용하여 컨트롤포인트의 기능을 사용하

게 된다.

```
#include <jni.h>
#ifndef _Included_cse_hansung_upnpcontrolpoint_NativeClass
#define _Included_cse_hansung_upnpcontrolpoint_NativeClass
#ifdef __cplusplus
extern "C" {
#endif

JNIEXPORT void JNICALL Java_cse_hansung_upnpcontrolpoint_NativeClass_NativeStart
  (JNIEnv *, jobject);

JNIEXPORT void JNICALL Java_cse_hansung_upnpcontrolpoint_NativeClass_NativeInvoke
  (JNIEnv *, jobject, jstring, jstring, jobjectArray);
JNIEXPORT jboolean JNICALL
Java_cse_hansung_upnpcontrolpoint_NativeClass_NativeGetUpdate
  (JNIEnv *, jobject);
JNIEXPORT jobjectArray JNICALL
Java_cse_hansung_upnpcontrolpoint_NativeClass_NativeGetDevicesLocationURLs
  (JNIEnv *, jobject);
JNIEXPORT jobjectArray JNICALL
Java_cse_hansung_upnpcontrolpoint_NativeClass_NativeGetSCPDURL
  (JNIEnv *, jobject, jstring);
JNIEXPORT jobjectArray JNICALL
Java_cse_hansung_upnpcontrolpoint_NativeClass_NativeGetActinName
  (JNIEnv *, jobject, jstring);
JNIEXPORT jobjectArray JNICALL
Java_cse_hansung_upnpcontrolpoint_NativeClass_NativeGetVariableAtt
  (JNIEnv *, jobject, jstring, jstring);
JNIEXPORT jobjectArray JNICALL
Java_cse_hansung_upnpcontrolpoint_NativeClass_NativeGetStateVariable
  (JNIEnv *, jobject, jstring);
#ifdef __cplusplus
}
#endif
#endif
```

그림 27 Native Call을 위한 C 헤더 파일

본 논문에서 구현한 UUCA(Unified UPnP Control Agent) 번들은 모든 UPnP 디바이스를 제어할 수 있도록 설계된 UPnP 번들이다. Jini 디바이스가 서비스 타입과 인터페이스의 내용으로 디바이스가 제공하는 기능을 알 수 있는 것에 비해 UPnP 디바이스는 디바이스에 대한 상세내용을 XML 문서로 기술하기 때문에 XML 문서를 파싱하여 각 디바이스에 적합한 UI를 가진 웹페이지를 만들어내는 서블릿을 생성하기가 Jini에 비해 용이하다. UUCA는 네트워크에서 감지된 모든 UPnP 디바이스의 XML 기술문서를 분석하여 통합 제어할 수 있는 웹페이지를 만들어내는 서블릿을

생성하고 생성된 서블릿을 OSGi 게이트웨이의 HTTP 번들에 등록하는 역할을 한다.

```
JNIEXPORT jstring JNICALL
Java_cse_hansung_upnpcontrolpoint_NativeClass_NativeGetDevicesLocationURLs
(JNIEnv *env, jobject jobj) {
    struct XmlList *xml = makeXML();
    struct XmlList *tmpXml = xml;
    jstring rValue;
    ...
    while(tmpXml != NULL) {
        if(tmpXml->value != NULL) {
            i += sprintf(url+i, "%s", tmpXml->value);
            tmpXml = tmpXml->Next;
        }
        else
            break;
    }
    ...
    rValue = (*env)->NewStringUTF(env, url);
    free(url);
    return rValue;
}

JNIEXPORT void JNICALL Java_cse_hansung_upnpcontrolpoint_NativeClass_NativeStart
(JNIEnv* env, jobject o) {
    ...
    CreateThread(NULL, 0, ControlPointThread, (LPVOID)dw, 0, &cp);
}

JNIEXPORT void JNICALL Java_cse_hansung_upnpcontrolpoint_NativeClass_NativeInvoke
JNIEnv *env, jobject o, jstring URL, jstring actName, jstring arg) {
    char* SCPDURL;
    struct UPnPService* service;
    char* actionName;
    char* argument;
    struct invokeArgument* invokeArg;
    SCPDURL = (*env)->GetStringUTFChars(env, URL, 0);
    actionName = (*env)->GetStringUTFChars(env, actName, 0);
    argument = (*env)->GetStringUTFChars(env, arg, 0);
    service = getServiceObj(SCPDURL);
    invokeArg = getinvokeArgument(argument);
    UPnPInvoke_SetTarget(service, NULL, NULL, actionName, invokeArg);
}


JNIEXPORT void JNICALL
Java_cse_hansung_upnpcontrolpoint_NativeClass_NativeGetUpdate (JNIEnv *env,
jobject o, jobject obj) {
    jmethodID mid;
    jclass cls = (*env)->GetObjectClass(env, obj);
    if(p_deviceListChange) {
        p_deviceListChange = 0;
        mid = (*env)->GetMethodID(env, cls, "update", "()V");
        (*env)->CallVoidMethod(env, obj, mid);
    }
}
```

그림 28 Native Call을 위한 C 코드

모든 디바이스를 제어하기 위해 컨트롤 포인트에서는 특정 디바이스가 아닌 모든 디바이스를 감지하고 감지된 디바이스를 리스트로 유지한다. 그리고 리스트의 디바이스의 기술문서를 분석하여 각 디바이스에 적합한 웹 페이지를 생성하도록 디자인되었다.

UUCA는 그림 29와 같은 구조를 가진다. 컨트롤 포인트는 네트워크에서 감지된 모든 UPnP 디바이스의 리스트를 유지하고, 그 리스트를 XML 문서로 구성하여 JNI를 통해 자바 코드에 넘긴다. UPnP Manager는 XML문서를 분석하여 디바이스 리스트를 구성하고 디바이스의 정보를 이용하여 서블릿을 생성해낸다. XML 문서는 각 디바이스의 Device Description Document와 Service Description Document에서 그림 30과 같은 필요한 정보만을 축약하여 구성된다.

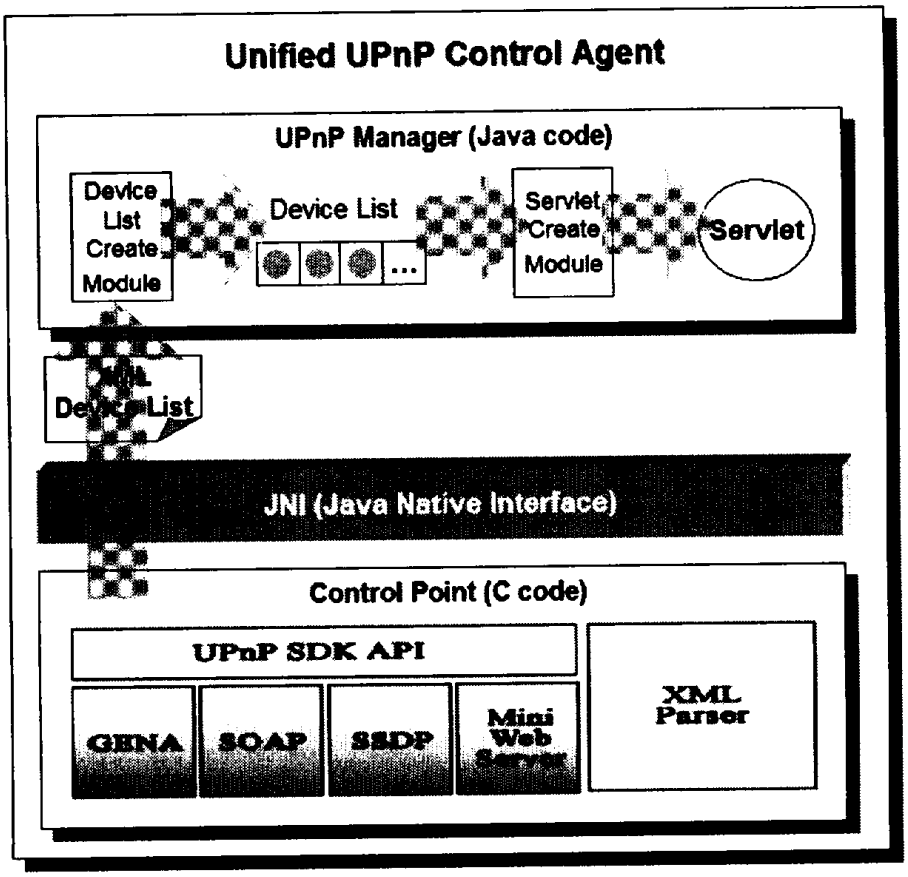

그림 29 UUCA 구조

```
<?xml version="1.0" encoding="utf-8" ?>
- <deviceList>
  - <device LocationURL="http://192.168.1.102:8085/" deviceType="urn:schemas-upnp-
      org:device:BinaryLight:1" FriendlyName="Device Test ()">
    - <service SCPDURL="http://192.168.1.102:8085/ColorChange/scpd.xml"
        controlURL="http://192.168.1.102:8085/ColorChange/scpd.xml" serviceId="urn:upnp-
        org:serviceId:ColorChange.0001" serviceType="urn:schemas-upnp-
        org:service:ColorChange:1">
      - <action name="GetColor">
          <argument name="ResultStatus" direction="out" relatedStateVariable="ColorStatus" />
        </action>
      - <action name="SetColor">
          <argument name="newColor" direction="in" relatedStateVariable="ColorStatus" />
        </action>
      - <variable name="ColorStatus" dataType="string" min="(null)" max="(null)" step="(null)"
          numAllowedValue="4">
          <allowedValue val="Yellow" />
          <allowedValue val="Blue" />
          <allowedValue val="Green" />
          <allowedValue val="Red" />
        </variable>
      </service>
    + <service SCPDURL="http://192.168.1.102:8085/DimmingService/scpd.xml"
        controlURL="http://192.168.1.102:8085/DimmingService/scpd.xml" serviceId="urn:upnp-
        org:serviceId:DimmingService.0001" serviceType="urn:schemas-upnp-
        org:service:DimmingService:1">
    + <service SCPDURL="http://192.168.1.102:8085/SwitchPower/scpd.xml"
        controlURL="http://192.168.1.102:8085/SwitchPower/scpd.xml" serviceId="urn:upnp-
        org:serviceId:SwitchPower.0001" serviceType="urn:schemas-upnp-
        org:service:SwitchPower:1">
    </device>
  + <device LocationURL="http://192.168.1.200:59477/" deviceType="urn:schemas-upnp-
      org:device:BinaryLight:1" FriendlyName="Light (KKANG74-HOME)">
  </deviceList>
```

그림 30 XML 디바이스 리스트

4.2.1 디바이스 감지, 제거시 UUCA 동작 과정

그림 31은 새로운 디바이스가 감지되거나 디바이스가 네트워크에서 제거될 때, UUCA의 동작 과정을 보여준다.

디바이스는 네트워크에 새롭게 연결되거나 제거될 때, UPnP 프로토콜에 따른 메시지를 컨트롤 포인트에게 보낸다. C코드로 구현된 UUCA의 컨트롤 포인트는 네트워크 소켓을 생성하고 생성된 소켓을 감시한다. 소켓에 디바이스로부터 감지 또는 제거에 따른 메시지가 감지되면 Descovery(), Remove() 함수가 호출되고 이 함수들은 UPnP 표준에 따른 감지, 기술문서전송 등을 수행한 후 컨트롤 포인트가 유지하고 있는 deviceList를 갱신하는 일을 수행한다. 컨트롤 포인트는 이와 같이 네트워크상의 UPnP 디

바이스들을 리스트로 유지하는 일을 한다. UUCA의 자바 코드에서는 주기적, 또는 deviceList가 필요시 deviceList의 갱신여부를 JNI를 통해 확인하고 갱신시 C 코드로부터 새로운 XML형태의 deviceList를 JNI를 통해 받아 자바코드의 deviceList를 갱신한다. 서블릿은 사용자의 디바이스 접근 요청이 있을시 자바 코드로부터 deviceList를 통해 서비스한다.

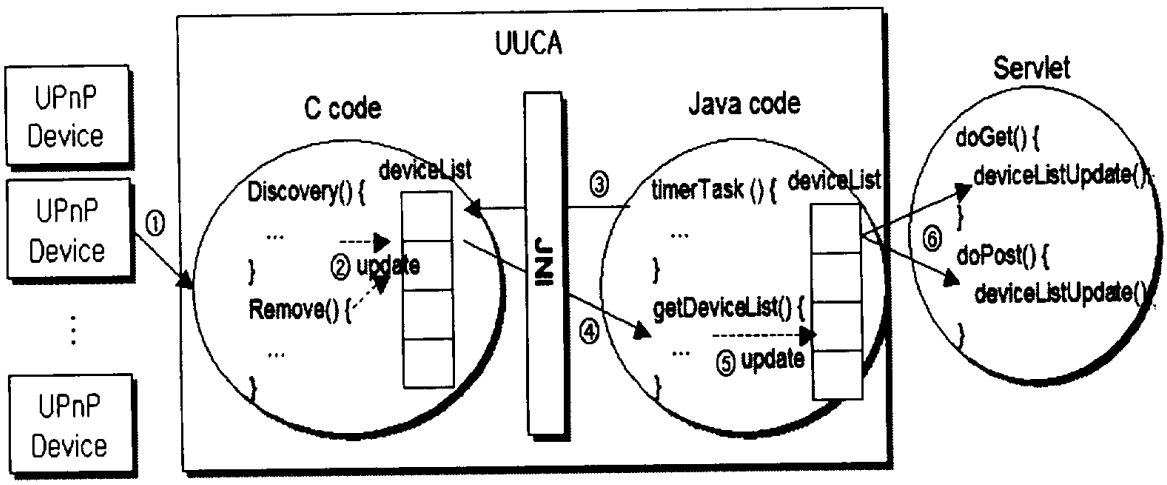

그림 31 디바이스 감지, 제거시 UUCA 동작과정

4.2.2 디바이스 Action 수행시 UUCA 동작 과정

서블릿에서 사용자가 서비스의 액션을 수행할 시 그림 32와 같은 과정으로 수행된다.

사용자의 서비스 액션 수행은 자바 코드의 invoke()를 호출하고 invoke()는 JNI를 통해 C 코드의 UPnPInvoke()를 호출한다. UPnPInvoke()는 파라미터로 넘어온 디바이스 정보, 서비스 정보, 액션 정보, 액션 매개변수를 이용하여 SOAP 메시지를 생성하여 해당 디바이스에게 전송한다. 디바이스는 SOAP 메시지에 의거하여 액션을 수행하고 수행한 결과를 SOAP 메시지 형태로 컨트롤 포인트에 전송한다. 컨트롤 포인트가 결과 메시지를 받으면 UPnPResponseSink()가 수행되고 JNI를 통해 자바 코드의 response()가 호출된다.

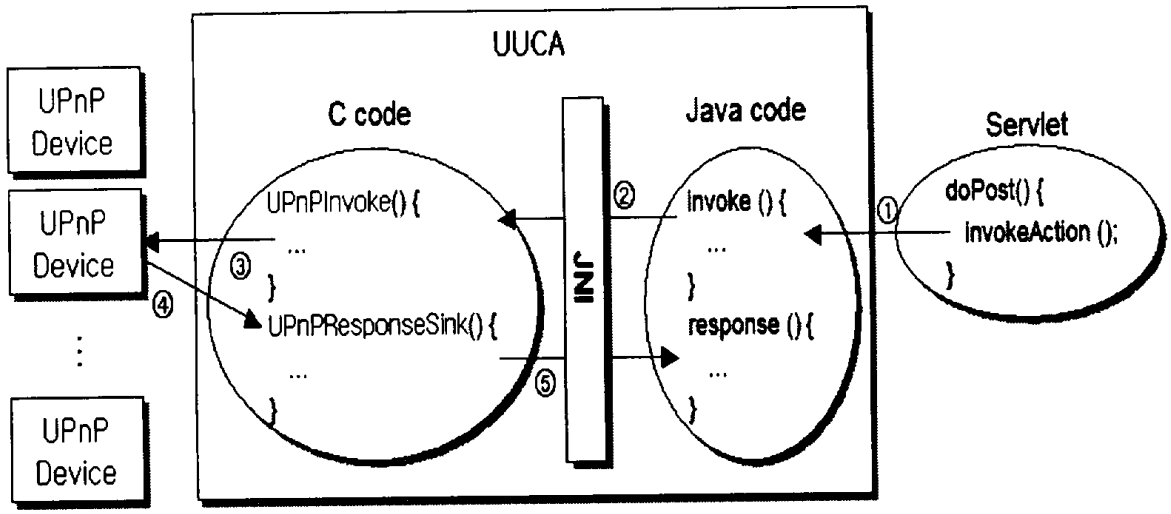

그림 32 디바이스 Action 수행시 UUCA 동작과정

4.2.3 UUCA의 동시성 제어(Concurrency control)

웹 애플리케이션은 단일 사용자를 위한 시스템이 아니다. 서블릿은 여러 클라이언트들의 요청을 동시에 처리할 수 있어야 한다. 본 논문에서 서블릿은 단일 스레드 서블릿이 아닌 다중 스레드 서블릿으로 구현되었다.

구현된 서블릿은 여러 스레드에 의해 실행될 수 있으므로 동시성 제어 문제가 발생한다. 그림 31, 32의 deviceListUpdate(), invokeAction() 메소드들을 동기화하는 방법으로 다중 스레드 서블릿이 가지는 동시성 제어 문제는 해결된다.

또한 컨트롤 포인트가 deviceList를 갱신하는 중에 deviceList를 서블릿에서 읽어 가는 경우에도 동시성 제어 문제가 발생한다. 이 문제는 deviceList에 대한 접근을 제어하여 문제를 해결하였다.

4.3 서블릿의 HTTP 번들 등록 과정

그림 33은 번들이 자신의 서블릿을 HTTP 번들에 등록시키는 방법을 설명한다. 번들 들은 OSGi 프레임워크에서 Start()에 의해 활성화될 때 자신이 다른 번들에게 제공할 서비스를 OSGi 인터페이스를 통해 프레임워크의 서비스 레지스트리에 등록 한다. 그림 31의 HTTP 번들은 서블릿을

등록받는 ResterServlet이라는 서비스를 프레임워크의 서비스 레지스트리에 등록하게 된다.

Jini 번들과 UUCA 번들은 인터넷을 통해 웹페이지를 통해 디바이스를 제어하는 역할을 하기위해 먼저 웹페이지를 생성하는 서블릿을 만들고, 생성된 서블릿을 등록시키기 위해 RegisterServlet 서비스를 OSGi 인터페이스를 통해 프레임워크의 서비스 레지스트리에서 찾는다. 서비스 레지스리에 RegisterServlet 서비스가 존재하면 그 서비스를 가져와 자신의 서블릿을 서비스를 통해 HTTP 번들에 등록한다.

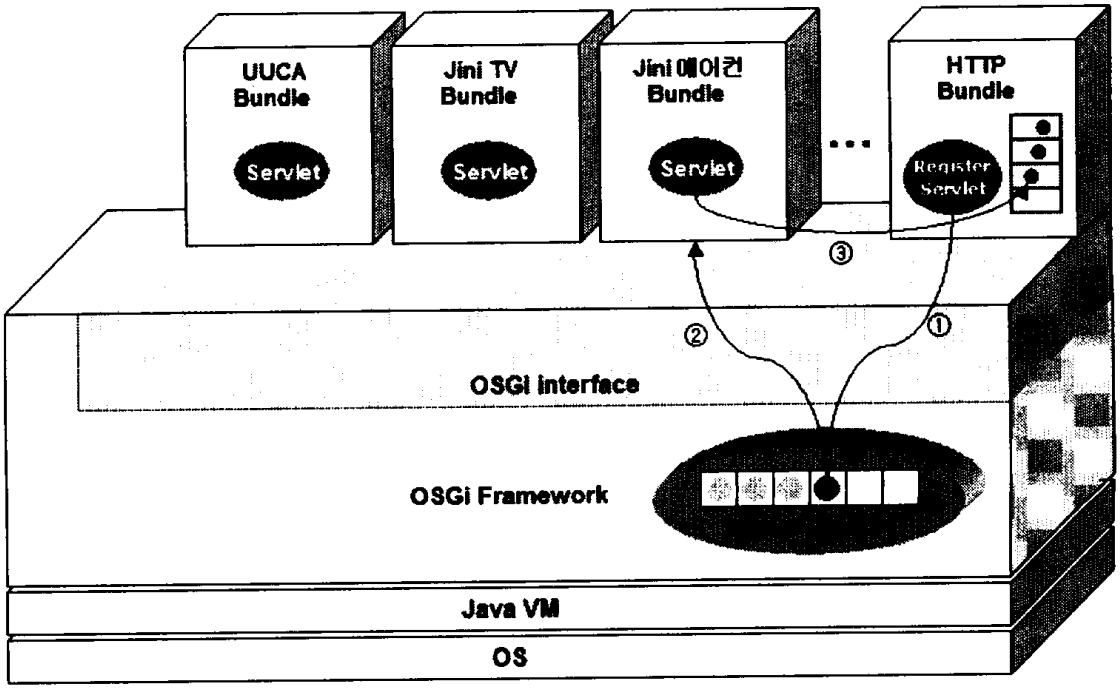

그림 33 서블릿 등록 과정

5. 실험 및 분석

5.1 OSGi 프레임워크 및 번들 설치

본 논문에서는 OSGi 프레임워크로 IBM에서 제공하는 SMF(Service Management Frame)[19] 를 사용하였다. SMF는 OSGi 서비스 플랫폼 3.0을 구현하여, 번들의 설치, 제거, 실행 등 번들 관리기능을 제공한다. 웹서버 및 주요 번들 관리 서비스를 제공하여 게이트웨이 기능과 번들 관리를 위한 인터페이스를 제공한다.

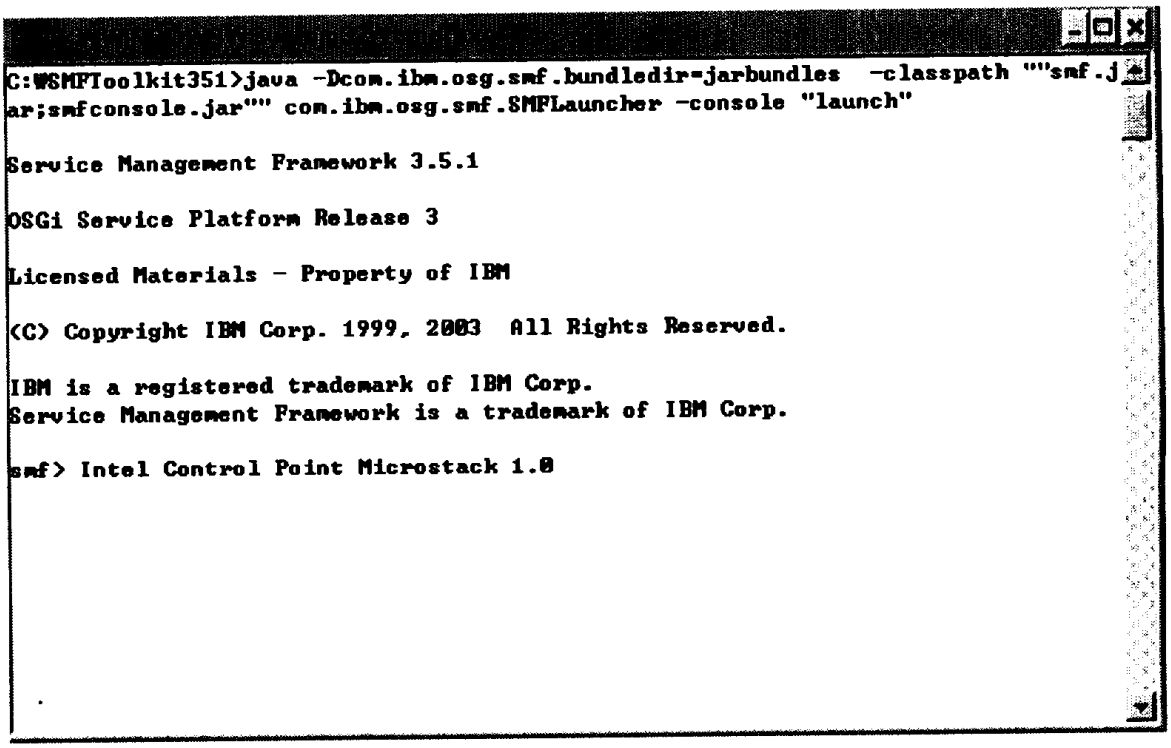

그림 34 smf 실행화면

번들의 설치, 실행, 정지, 제거 등 번들 관리를 위해서는 SMF 시스템 번들에서 제공하는 서비스를 그림 36과 같이 웹브라우저를 통해 사용한다. 웹브라우저의 주소입력창에 "http://localhost:portNum/smfdmin" 을 입력하면 그림 35와 같은 로그인 화면이 열린다.

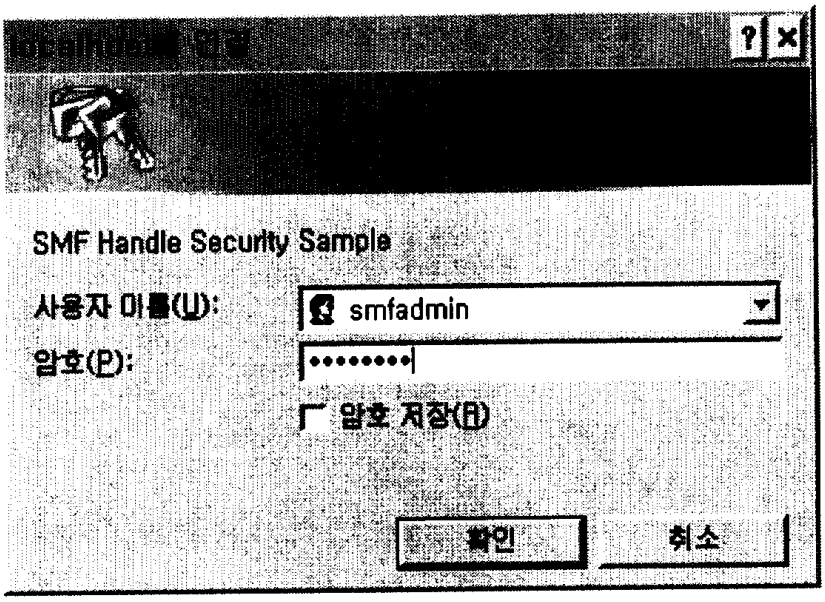

그림 35 번들 관리를 위한 smfadmin 로그인

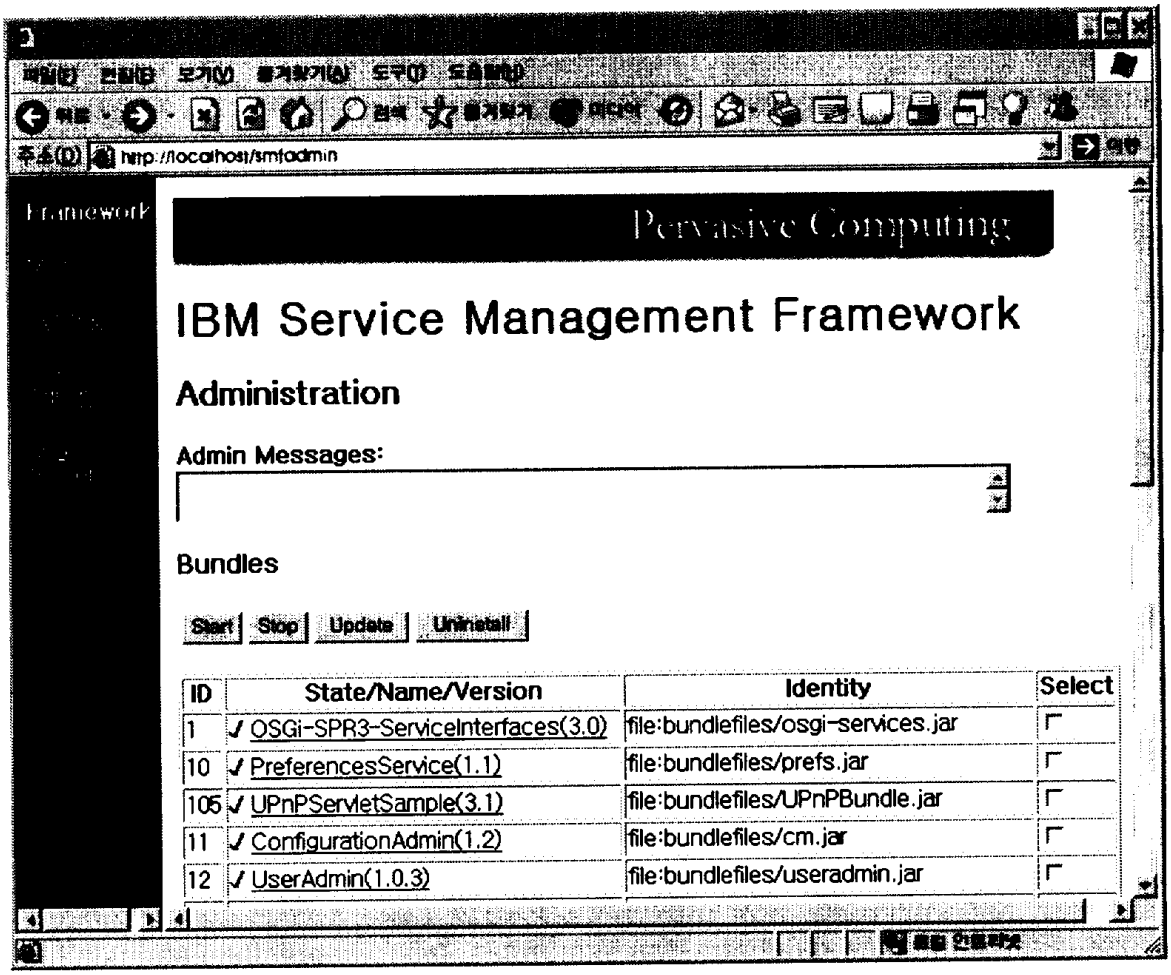

그림 36 프레임워크의 번들 상태

SMF의 초기 아이디와 패스워드는 아이디 : smfadmin, 패스워드 : password 이며, 패스워드를 재설정하기 위해서는 URL에 "http://localhost:portNum/password" 를 입력하여 설정 할 수 있다.

아이디와 패스워드를 입력하여 로그인이 성공하게 되면 그림 36과 같이 설치된 번들의 상태를 보여주는 화면이 열린다. 이 화면에는 번들의 아이디, 현재의 상태, 경로, 번들에 대한 설명, 그리고 번들 관리에 필요한 Install, Start, Stop, Update, Uninstall 버튼이 있다.

새로운 번들을 설치하기 위해서는 그림 37과 같이 설치할 번들의 위치와 파일이름을 Bundle URL 입력란에 입력하고 Install 버튼을 누른다. 번들이 성공적으로 설치되거나, 설치가 실패하게 되면 Admin Messages 란에 상세 메시지가 출력된다.

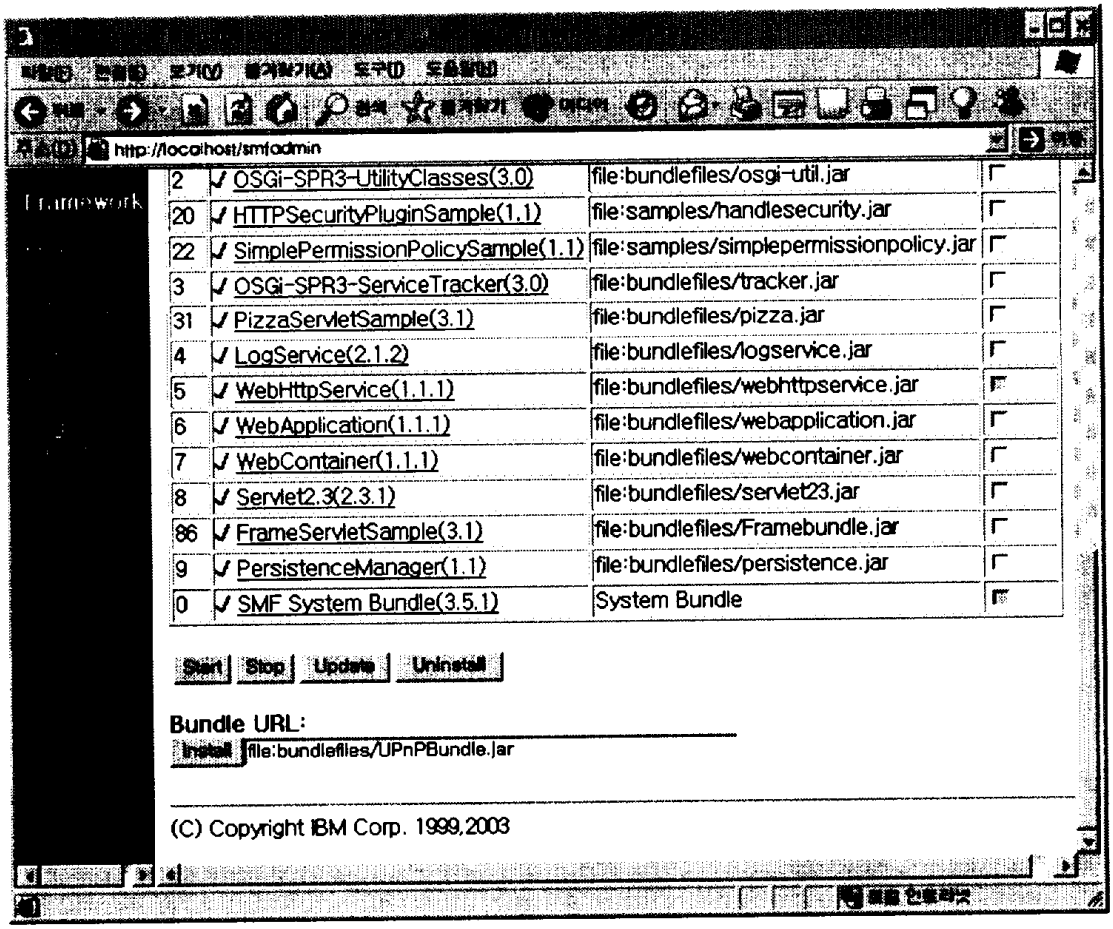

그림 37 번들의 설치

그림 36의 화면에서 설치된 번들을 클릭하면 그림 38과 같은 화면을 볼 수 있는데 이 화면은 선택한 번들이 import 하고 있는 서비스, export 하는 패키지 등 번들의 상세 정보를 볼 수 있으며, 번들에 필요한 환경을 설정 할 수 있다.

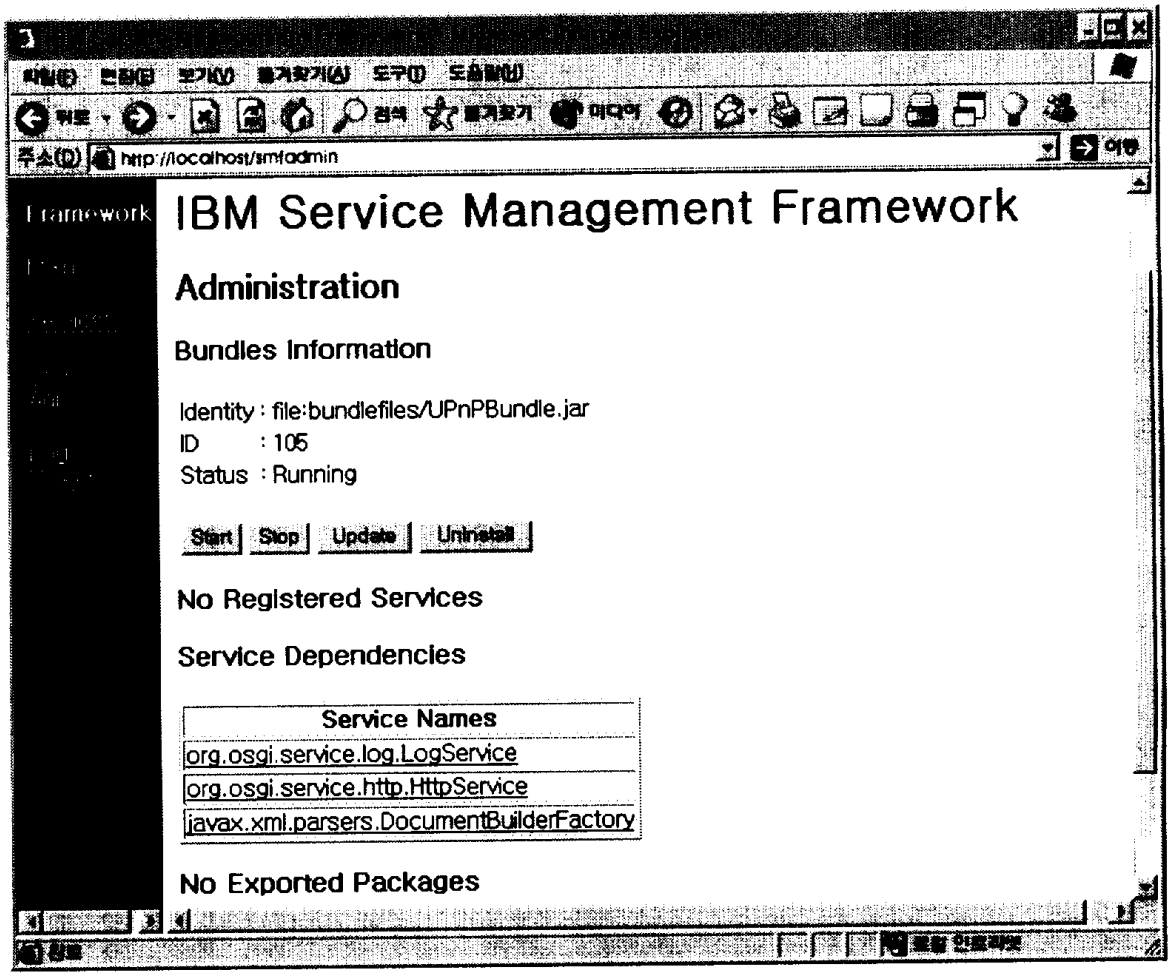

그림 38 번들의 세부 상태 제어

5.2 Jini 디바이스 제어

그림 39는 외부 네트워크에서 OSGi 게이트웨이를 통해 내부 네트워크의 Jini TV와 Jini 에어컨을 제어하는 모습이다. 외부에서 사용자는 웹브라우저에 주소입력창에 OSGi 게이트웨이의 IP주소와 Jini 번들의 서블릿 패스(http://210.222.25.59/jini) 를 입력하여 Jini 제어 페이지를 열고 Jini 디바이스를 제어한다.

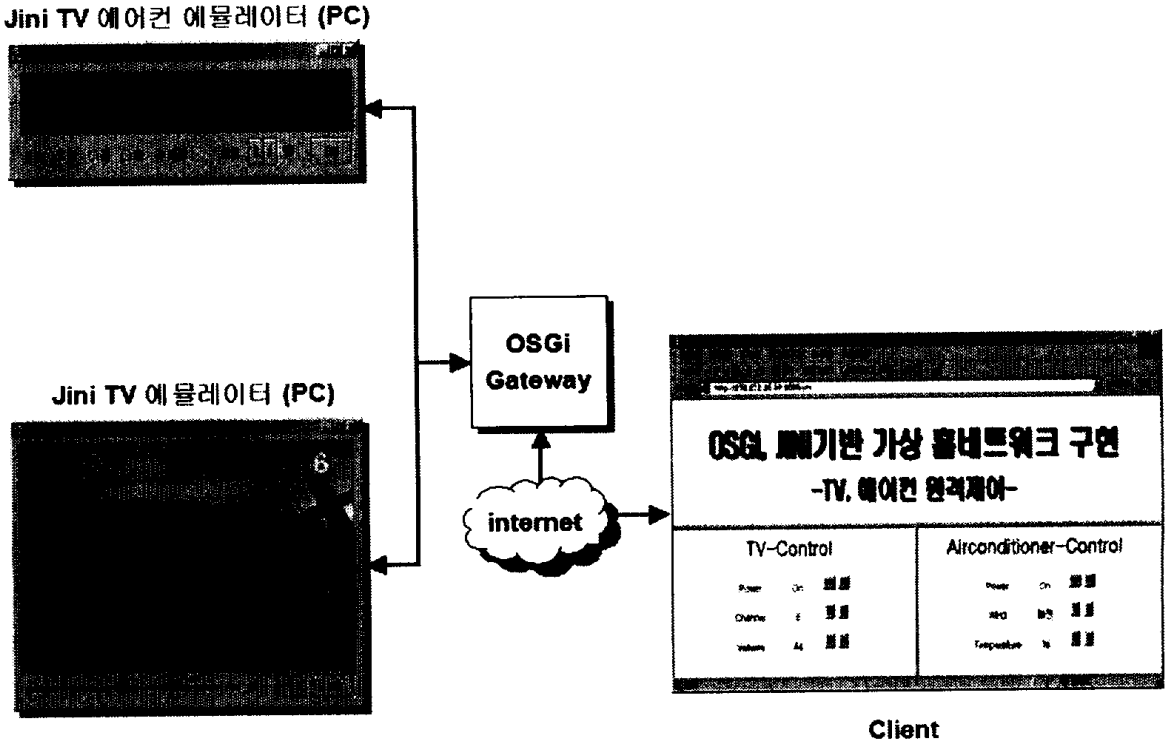

그림 39 OSGi를 통한 Jini 디바이스 제어

5.3 UPnP 디바이스 제어

그림 40은 UPnP 디바이스를 제어하는 화면이다. Jini와 마찬가지로 사용자는 웹브라우저에 주소입력창에 OSGi 게이트웨이의 IP주소와 UPnP 번들의 서블릿 패스(http://210.222.25.59/upnp) 를 입력하여 Jini 제어 페이지를 열고 Jini 디바이스를 제어한다.

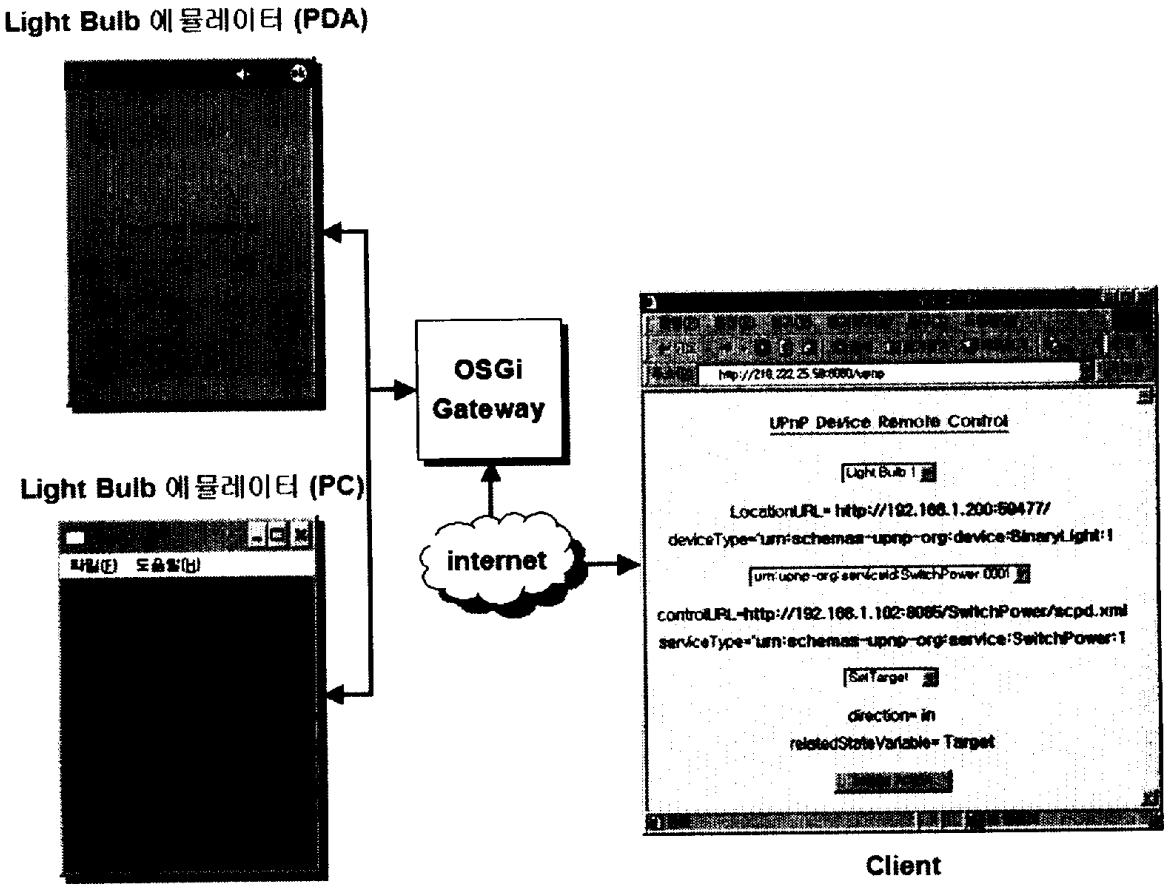

그림 40 OSGi를 통한 UPnP 디바이스 제어

6. 결론 및 향후연구

현재 홈 네트워킹 시장에는 다양한 홈 게이트웨이 및 홈 네트워킹 미들웨어 들의 난립으로 수많은 서비스 개발환경이 존재한다.

본 논문에서는 이와 같은 홈 네트워크의 이질성을 해결하기 위해 제안된 OSGi 게이트웨이와 미들웨어 솔루션 중 대표적인 기술인 Jini와 UPnP를 이용한 홈 네트워크를 구성하였다. Jini 디바이스와 UPnP 디바이스를 에뮬레이션하기 위한 UPnP Light Bulb, Jini TV, 에어컨 디바이스를 에뮬레이터로 구현하고 각 디바이스를 제어하기 위한 클라이언트를 번들로 구현하였다.

OSGi 프레임워크에서 구동되는 번들의 구성은 다음과 같다. UPnP 클라이언트 번들, Jini 클라이언트 번들(TV, 에어컨)로 크게 2가지로 구성되며 Jini 환경을 위한 RMI 번들, 룩업서비스 번들이 필요하다.

하나의 번들로 모든 UPnP 디바이스에 대해 제어서비스를 제공하기 위해 UUCA(Unified UPnP Control Agent) 번들을 설계, 구현하였다. UUCA는 모든 UPnP 디바이스를 네트워크에서 감지하고 감지된 디바이스를 리스트로 유지한다. 그리고 리스트의 디바이스의 기술문서를 분석하여 각 디바이스에 적합한 웹페이지를 생성하도록 디자인되었다.

실행을 통하여 구현된 Jini와 UPnP 디바이스가 정상적으로 동작함을 확인하였으며, 각 번들도 OSGi 스펙에 따라 정상적인 동작을 보였다. UUCA 또한 정상적으로 작동하였다.

현재 Jini와 UPnP 미들웨어를 이용한 시스템들이 개발되고 있으나 Jini와 UPnP의 성능에 대한 객관적인 평가와 분석 데이터의 부재로 시스템 개발 시 미들웨어를 선택하는데 혼란이 따르고 있다.

향후 연구과제로 구현된 두 미들웨어 응용을 성능측정에 적합하도록 보완하고 미들웨어의 성능을 측정하기 위한 모듈의 작성이 요구된다. 또한

Jini와 UPnP의 성능과 특성을 정성적, 정량적으로 비교 분석하여 시스템 구축시 미들웨어를 선택하는데 지침을 줄 수 있는 객관적인 데이터를 얻기 위한 연구가 필요하며, Jini에 대해서 하나의 번들로 모든 Jini 디바이스를 제어할 수 있는 통합 제어 번들 연구가 필요하다.

OSGi는 서비스를 공유하므로서 번들 간의 정보공유 및 제어가 가능하다. 이는 이기종 미들웨어간의 정보공유 및 제어 또한 가능함을 의미한다. OSGi의 이러한 특성을 이용하여 Jini, UPnP 간의 상호운용 방안에 대한 연구가 필요하며 본 논문에서 구성한 단순한 디바이스 제어 서비스에서 진보하여 파일 전송, 교육서비스, 영상, 음성, 엔터테인먼트와 같은 실질적으로 생활에 필요한 서비스와의 연동에 대한 연구가 요구된다.

參 考 文 獻

[1] 과학기술부, "국가기술지도 I 정보,지식,지능화 사회구현 3권", 2002

[2] HomePNA, http://www.homepna.org

[3] 전호인, "IEEE 1394 기술 및 표준화 동향" Nov.2000

[4] "Universal Serial Bus Specification Revision 1.1", Compaq, Intel, Microsoft, NEC, 1998, 3, 23

[5] 현덕화,유인협,박병석 , "전력선을 이용한 통신기술의 동향" , 전기저널, 2000,9

[6] Bluetooth, http://www.bluetooth.org

[7] HomeRF, http://www.homerf.org

[8] irDA, http://www.irda.org

[9] HAVi, "Home Audio/Video Interoperability," 2002. http://www.havi.org

[10] UPnP, "Universal Plug and Play Forum," 2002. http://www.upnp.org

[11] Sun, Jini Connection Technology. Sun Microsystems, 2000 http://www.sun.com/jini,

[12] OSGi, http://www.osgi,org.

[13] http://www.upnp.org/resources.htm

[14] http://www.sun.com/jini/specs

[15] OSGi,"OSGi Service Gateway Specification - Release 3.0" http://www.osgi.org, 2003.

[16] http://www.osgi.org/resources/spec_download.asp

[17] W. Keith Edwards, "Core JINI", Prentice-Hall, 2001.

[18] Henry Wong, Scoot Oaks, "JINI in a Nutshell", O'Reilly, 1999.

[19] http://www-306.ibm.com/software/wireless/smf/toolkit.html

[20] http://www.ietf.org/

[21] Dong-Sung. Kim, "Design and Implementation of Home Network Systems Using UPnP Middleware for Networked Appliances", IEEE Transaction on Consumer Electronics, Vol 48, No. 4, Nov. 2002

[22] Peter M. Conrcoran, "Home Network Infrastructure for Handheld/Wearable Appleances", IEEE Transactions on Consumer Electronics, Vol 48, No.3 Aug, 2002

[23] E. Topalis, "A Novel Architecture for Remote Home Automation e-Services on an OSGi Platform via High-Speed Internet Connection Ensuring OoS Support by Using RSVP Technology", IEEE Transactions on Consumer Electronics, Vol 48, No.4 Aug, 2002

[24] Takeshi Saito, "Home Gateway Architecture and Its Implementation", IEEE Transaction on Consumer Electronics, Vol 46, No. 4, Nov. 2000

[25] Roli G. Wendorft "Remote Execution of HAVi Applications on Internet-Enabled Device" IEEE Transaction on Consumer Electronics, Vol 47, No. 3, Nov. 2001

[26] Peter M. Conrcoran, "User Interface Technologies for Home Appliances and Networks", IEEE Transactions on Consumer Electronics, Vol 44, No.3 Aug, 1998

[27] G. G. Richard III, "Service and Device Discovery: Protocols and Programming", McGraw-Hill, 2002.

[28] 강성민, 황기태 "유비쿼터스 컴퓨팅의 미들웨어 테스트 응용 구현" 한국통신학회 2003년 추계학술대회 5-17, p106.

[29] 강성민, 황기태 "개방형 서비스 게이트웨어(OSGi)와 JINI 기반의 가상 홈 네트워크 구현", 한성대학교 공학연구센터 공학 연구 Vol.1 August 2003.

ABSTRACT

Home Networking based on OSGi gateway and the middlewares of UPnP and Jini

Kang, Sung-min
Department of Computer Engineering
Graduate School, Hansung University

As the network technology grows rapidly, home networking has a great attention by researchers and developers. During the past decade, many middleswares have been proposed and developed for home networking to support home control, bulk data transmission, and audio/video streaming.

In this thesis, a home networking system has been constructed with the technologies of Jini, UPnP, and OSGi, and a proof system has been implemented, which emulates UPnP devices, Jini devices, and clients that request services from them. In that implementation, Jini and UPnP clients have been developed as OSGi Bundles to be run over the OSGi framework.

A key result of this thesis is the design and implementation of UUCA(Unified UPnP Control Agent). UUCA is a kind of UPnP Control Point with a shape of OSGi Bundle which controls all UPnP devices connected to home network. In order to control all UPnP devices, UUCA discovers all UPnP devices using discovery protocol and maintains a list of them. Also UUCA reproduces servlets which helps to have an access to internal UPnP device. It is called by the servlets which give connection to the home network to external web users.

Finally, this thesis shows that the home network system, which is constructed with Jini devices, UPnP devices, and OSGi gateway, works well through real executions.