홈네트워크 장치 제어를 위한 OSGi 및 UPnP 미들웨어 기술

강동오 · 강규장 · 이전우<한국전자통신연구원>

1. 서 론

디지털과 정보 통신 기술의 발달로 인하여 가정 내의 장치들이 홈네트워크로 연결되고 있다. 가까운 미래에는 집안의 모든 가전 기기와 조명 및 전기 장치들이 홈네트워크를 통하여 연결되어 제어되며, 다양한 서비스가 제공되는 디지털 홈이 구축될 것으로 전망된다 [1, 2, 3]. 이렇게 가정 내의 홈네트워크 장치가 보편화되어 서비스 제공자에 의한 다양한 서비스 제공이 가능하게 됨으로 인하여 홈네트워크 기술에 대한 관심이 매우 커지고 있는 추세이다.

홈네트워크 장치들은 네트워크을 통한 통신에 의해서 제어되고, 또한 여러 제조사들의 제품이 혼재하여 상호 동작하여야 하므로, 상호 호환성과 상호 운용성을 가지고 동작하는 것이 매우 중요하다. 응용 프로그램 개발은 하부의 통신 프로토콜에 무관하게 미들웨어 API를 이용하여 작성되고, 장치 간의 통신은 미들웨어가 담당하는 형태로 개발된다. 따라서, 홈네트워크 미들웨어는 장치 상호 운용성을 제공하여 각 장치들의 기능을 서로 사용하고 제어할 수 있게 한다. 이러한 홈네트워크 미들웨어에는 Jini, UPnP (Universal Plug and Play), Havi (Home Audio Video Interoperability), Lonworks 등이 있다.

가정 내에 있는 홈네트워크 장치들에 여러 서비스 제공자들이 다양한 서비스를 제공하기 위해서는 표준화된 서비스 플랫폼이 필요하다. 공개된 표준을 통하여 서비스 제공자들은 표준화된 서비스를 제공함으로 서비스 간의 상호 기능 운용이 가능하다. 이 때 서비스 제공자에 의하여 제공되는 서비스에는 홈네트워크 장치들을 이용한 서비스를 제공할 수 있다. 따라서, 표준 서비스 플랫폼에는 홈네트워크 장치와 통신하고 제어하는 구조가 필요하다. 또한, 홈네트워크 미들웨어와 연계를 통하여 홈네트워크 미들웨어를 사용하는 홈네트워크 장치와 통신하고 제어할 수도 있어야 한다.

본 논문에서는 홈네트워크 장치의 제어를 위한 미들웨어 기술 중에서 대표적인 공개 표준 서비스 플랫폼인 OSGi (Open Service Gateway Initiative)와 홈네트워크 미들웨어인 UPnP 기술에 대하여 설명한다. 본 논문의 구성은 다음과 같다. 먼저, 2장에서는 OSGi에 대한 소개와 장치 접근 구조, 홈네트워크 미들웨어와 연계 방법에 대하여 기술한다. 3장에서는 UPnP에 대한 소개 및 조명 제어 UPnP 표준에 대하여 기술한다. 끝으로, 4장은 논문 내용을 요약한다

2.. OSGi에서의 홈네트워크 장치 제어 기술

2.1 OSGi 소개

OSGi (open services gateway initiative)는

다양한 WAN (wide area network) 표준들과 홈네트워크 표준들 사이에서 이들을 연결하고 서비스를 전달하기 위한 국제 공개 표준으로 OSGi 연합에 의해 제정되었다.

OSGi 연합은 개방형 규격을 개발하고 이러한 규격에 근거한 제품 및 서비스의 사용 촉진을 목표로 1999년 Ericsson, SUN, IBM 등의 15개 회사가 모여 창립하였으며 2004년 현재 60여개 업체가 회원으로 참여하고 있고 2003년 4월에 OSGi 서비스 플랫폼 릴리즈 3.0을 발표하였다.

OSGi는 다른 산업계 표준들과 상호 밀접한 관계를 가지고 있으며 그림 1에서와 같이 광역 네트워크와 지역 네트워크를 연결하는 환경에서 사용될 수 있고 그림 2에서와 같은 솔루션을 가질 것이다.

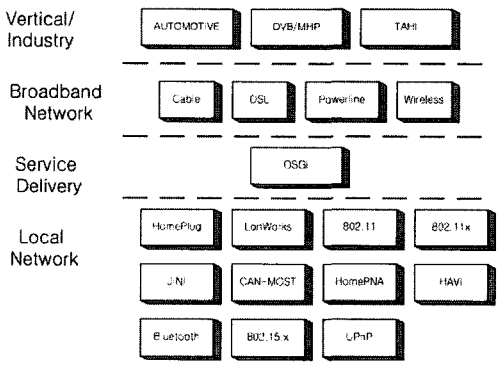

그림 1. 다른 표준들과의 관계

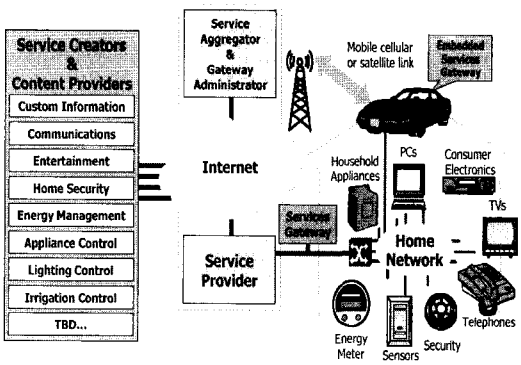

그림 2. OSGi 종단간 솔루션

OSGi 규격은 서비스 전달과 관리를 위한 프레임워크와 OSGi 번들에 대해서 정의하고 있다.

OSGi 번들은 배포될 수 있는 하나의 단위(Unit)로 인터페이스(Interface), 구성 정보(Configuration Information), 실행 가능한 코드 및 디지털 서명 등이 하나의 JAR 파일로 패키지화된다. 그러므로 써드 파티(3rd Party) 개발자에 의해 제공되는 서비스 번들은 배포될 수 있는 하나의 응용 솔루션이 될 것이다.

OSGi를 기반으로 하는 서비스 번들의 전달, 실행 및 관리 구조는 그림 3과 같이 나타낼 수 있다.

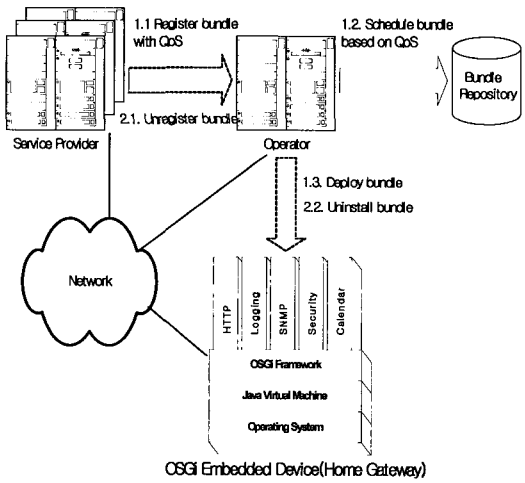

그림 3. 서비스 전달, 실행 및 관리

홈네트워크의 응용에서 OSGi가 관리 가능한 서비스를 제공하기 위해서 응용 서비스 소프트웨어를 저장하고 수행할 수 있는 게이트웨이 장치를 필요로 한다.

OSGi 서비스 플랫폼은 PC와 같은 범용 목적의 컴퓨터, 홈 게이트웨이와 같은 특별한 목적의 장치, 셋탑박스, 미디어 서버 또는 오퍼레이터 네트워크에 있는 가상의 리소스가 될 것이다. 서비스 플랫폼은 동적인 소프트웨어 업데이트, 원격 제어, 원격 관리, 원격 진단, 데이터 교환, 및 신뢰할 수 있는 보안적 기능 등을 가진다.

OSGi 규격은 현재 세 번의 업그레이드가 있었다. OSGi 릴리즈 1.0은 OSGi 규격의 기반이 되는 프레임워크와 로그 서비스, HTTP 서비스 및 디바이스 매니저로 구성된다. 프레임워크는 응용 서비스들이 하나의 자바 가상 머신을 공유할 수 있도록 하고 번들의 라이프 사이클, 자바 패키지, 리소스, 번들간의 의존성 등을 관리하며, 번들간의 상호 협업을 위해 서비스 레지스트리를 가지고 있다. 로그 서비스는 프레임워크에서 발생하는 이벤트를 기록하고, 읽을 수 있도록 하는 서비스를 제공하며 HTTP 서비스는 서비스 플랫폼을 웹을 통해 접근할 수 있도록 한다. 그리고 디바이스 매니저는 동적인 드라이버 다운로딩 모델을 제공한다.

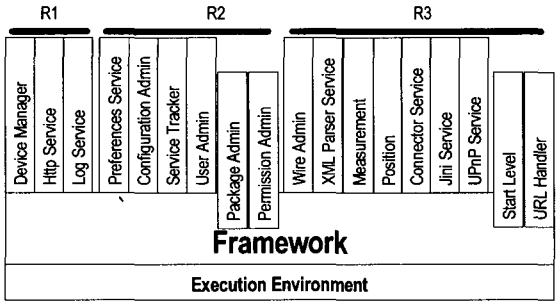

그림 4. OSGi 규격 릴리즈

2001년 10월에 발표된 릴리즈 2에서는 릴리즈 1의 기능에 프레임워크가 등록하는 패키지 관리 서비스, 퍼미션 관리 서비스가 추가되었고, 그 이외에 사용자 관리를 위한 사용자 관리 서비스, 동적인 구성정보 관리를 위한 구성 관리 서비스 등이 추가되었다.

2003년 4월에 발표된 릴리즈 3에서는 릴리즈 2의 기능에 프레임워크가 등록하는 시작 레벨 관리 서비스, URL 핸들러가 추가되었고, 그 이외에 Jini 네트워크를 수용하기 위한 Jini 서비스, UPnP 네트워크를 수용하기 위한 UPnP 서비스 등이 추가되었다. 특히 릴리즈 3에서는 OSGi가 SOHO용 게이트웨이 뿐만 아니라 차량용 게이트웨이 지원을 위해 연결 관리 서비스, IO 커넥터, 포지션 등과 같은 서비스 들이 추가 되었다.

OSGi 릴리즈의 변화 및 그 내용에 대해 〈그림 4〉에 나타내었다.

2005년도에 3천 5백만 가정이 브로드밴드 네트워크에 접속될 것으로 예상되고 그 중 75% 정도가 홈 네트워크를 구축할 것으로 예상하고 있다. 홈 네트워크에 적용될 수 있는 홈 플랫폼으로는 Ciaolab 홈스테이션, Cisco CiscoWorks 2000 서비스 관리 솔루션, 프랑스 텔레컴 및 톰슨 멀티미디어 홈서비스 플랫폼, MetaVector 오피스, 보안 응용 및 에너지 관리, 스마트홈(SmartHome) 등이 있다.

정보와 오락 기능을 겸비한 인포테인먼트 플랫폼(Infotainment Platform)으로서의 자동차 개발이 시도되고 있으며, 이들 대부분이 OSGi를 채택하고 있다. 자동차/인포테인머트 플랫폼의 예로는 유럽에서 진행되고 있는 3GT 인포테인먼트 플랫폼이 있다.

유럽 파트너쉽 프로젝트들은 협동 환경을 단순화시키고 가정과 차량을 통합하는 관점으로 OSGi 서비스 플랫폼을 정의하고 있으며 그밖에 홈오토메이션/헬스케어의 예로는 알레토(Alleato) 헬스케어, BSH 네트워크 홈 가전, E2-홈 스톡홀름, e-PASTA 홈 네트워킹, 홈디렉터 홈파일럿, 홈넷투런(HomeNet-2Run), 인하우스 두이스버그(InHaus Duisburg), 인터컴포넌트웨어(InterComponentWare) 대화형 헬스 기록 라이프 센서, 텔리아(Telia) 커넥티드 홈 및 월풀(WhirlPool) 홈 솔루션 라인 등이 있다.

2.2 OSGi 시스템 구조 및 기본 개념

OSGi 소프트웨어는 서비스 번들의 실행 환경인 프레임워크와 프레임워크에 설치되는 소프트웨어 컴포넌트인 번들로 구성되는데 그림 5와 같고 서비스 플랫폼으로 불린다. 서비스 번들은 서비스 객체를 포함하지 않는 라이브러리와 같은 번들과 여러 개의 서비스 객체로 구성 되는 번들의 형태로 만들어진다. 기

본적으로 프레임워크는 자바 가상 머신 위에서 동작하고 서비스 번들은 프레임워크, 자바 가상 머신 및 운영체계에 걸쳐서 동작된다.

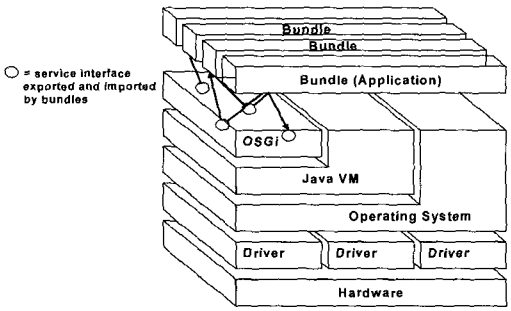

그림 5. OSGi 소프트웨어 실행 환경

OSGi 환경에서 서비스는 유용한 기능을 가진 객체이고 자바 프로그래밍 언어로 구현되는데 개발하는 과정은 다음과 같다.

단계1: 서비스가 해야 할 기능을 정의하는 인터페이스 정의 단계

단계2: 정의된 기능이 어떻게 동작되는지 클래스로 구현하는 단계

서비스를 개발할 때 인터페이스를 정의하는 것과 이를 구현하는 것으로 구분하여 만드는 이유는 구현된 클래스 객체에 변화가 생기더라도 이 서비스를 사용하는 입장에서 동일한 접근 인터페이스를 제공하기 위해서이다. 또한 하나의 서비스 인터페이스에 대해 여러 개의 구현이 존재할 수 있다.

OSGi 환경에서 번들은 서비스를 패키지화하는 단위이고 각각은 자신의 생명주기를 가지고 있으며 프레임워크로 설치되기 위한 인터페이스를 가지고 있다. 번들은 표준 자바 아카이브인 JAR 파일로 만들어지며 서비스 인터페이스, 이를 구현한 클래스, 그림, HTML 페이지, 데이터 등의 리소스로 구성된다. OSGi 환경에서 번들은 서비스를 배포하고 관리하기 위한 유일한 방법이다.

OSGi 환경에서 번들은 installed, resolved, active, uninstalled, starting 및 stopping과 같은 여섯 가지 상태를 가지는데 그림 6과 같다.

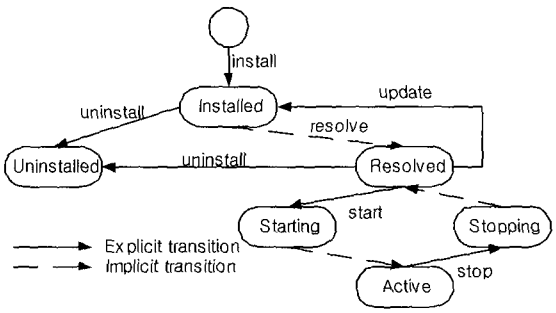

그림 6. 번들의 생명주기

프레임워크와 번들 간의 의사소통을 위해서 표준적인 인터페이스가 정의되는데 그 하나는 번들의 매니페스트이고 다른 하나는 번들 액티베이터(activator)이다. 번들 매니페스트는 JAR 파일의 표준 구성 요소로서 체크섬 (checksum) 및 서명과 같은 메타 정보와 OSGi에서 정의하는 매니페스트 헤더 정보로 구성된다. 번들 액티베이터는 번들에 의해 구현되고 번들이 기동 및 중지 될 때 수행할 기능을 구현하고 있다. 그리고 번들 액티베이터는 프레임워크에 의해 인식되기 위해 OSGi에서 정의하는 표준 API를 구현해야 한다.

OSGi 환경에서 프레임워크는 모든 번들의 실행환경으로서 번들의 생명주기를 관리하고 번들 간의 의존성을 해결하며 번들 상호간에 클래스와 리소스가 공유되도록 한다. 그리고 내부에 서비스 레지스터리를 가지고 있어 번들에 의해 등록되는 서비스를 관리한다. 프레임워크는 또한 이벤트 메커니즘을 가지고 있는데 번들의 상태가 변경될 때, 서비스가 등록 및 해지될 때, 프레임워크가 기동할 때 및 오류를 일으킬 때 이와 관련된 이벤트를 공지하여 리스너를 가지고 있는 다른 객체가 적절한 조치를 취할 수 있도록 한다.

프레임워크는 막후에서 동작되는 것으로 자신의 인터페이스를 외부에 공개하지 않고 번들 컨텍스트를 두어 이것을 통해서 번들이 프레임워크 기능에 접근할 수 있도록 하는 프락시와 같은 객체를 가지고 있다.

OSGi의 상호 협동 모델은 그림 7과 같이 번들 및 서비스 간의 상호 협동 작업에 의해 수행되는데 번들에 의해 패키지를 제공하는 엑스포팅 (exporting)과 이 패키지를 인용하는 임포팅 (importing) 그리고 서비스 레지스터리를 통해 서비스를 등록 하는 것과 등록된 서비스를 사용하는 것에 의해 실현 된다.

패키지를 엑스포팅 및 임포팅 하는 것은 번들의 매니페스트 헤더에 다음 예와 같이 선언함으로써 실현된다.

- Export-Package: come.acme.foo; specification-version=2.0, com.acme.bar
- Import-Package: com.acme.foo; specification-version=1.0, com.acme.bar

이와 같이 선언이 되면 프레임워크는 패키지를 사용하는 번들이 기동되기 전에 패키지를 제공하는 번들을 찾고 관련 패키지가 정상적으로 제공되는지를 확인한다.

번들이 실행시 필요한 클래스를 로딩 하기 위해 찾는 순서는 시스템 클래스와 클래스패스 (CLASS-PATH)에 명기된 클래스를 먼저 찾고 다음으로는 프레임워크로 엑스포트 된 패키지를 찾으며 마지막으로 자신의 번들에서 클래스를 찾는다.

OSGi 환경의 상호 협동 모델은 정적인 패키지 의존성과 동적인 서비스 의존성이라는 특징을 가지고 있다. 패키지 의존성은 번들을 개발하는 시점에서 번들의 매니페스트 헤더에 상호 의존 관계를 명시함으로 정적으로 결정되고 서비스 의존성은 번들이 기동되고 난 후 프레임워크 서비스 레지스터리에 서비스를 등록하고 해지함으로서 동적으로 결정된다. 이러한 서비스 의존성은 등록 및 해지 시 발행하는 이벤트를 활용하여 실행 중에 동적으로 커스터마이징이 가능하다.

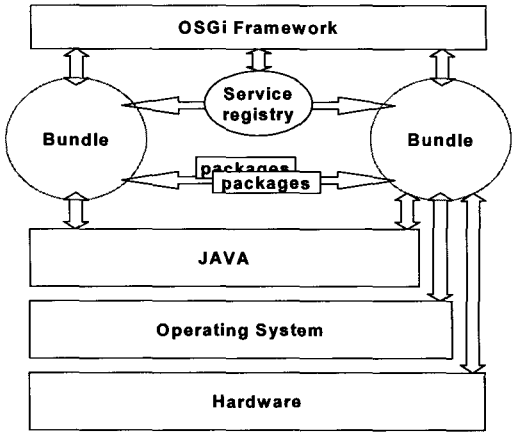

그림 7. OSGi 상호 협동 작업

2.3 장치 접근 (device access) 구조

OSGi 서비스 플랫폼은 사용자가 서비스 가입을 추가하거나 취소하고, 새롭게 설치된 서비스와 관련된 입출력 장치를 찾고 드라이버가 하드웨어를 연결할 수 있는 서로 다른 벤더들의 서비스와 디바이스가 만나는 지점이다. OSGi 환경에서 서비스 번들의 실행 환경인 프레임워크가 실행중일 때 이러한 활동들은 동적으로 일어나며, 이러한 유연성은 장치와 관련된 부분을 구성하기 힘들게 한다. 이러한 이유로 OSGi 명세에서는 장치 접근에 대한 구조를 제공하는데 이것은 OSGi 환경에서 존재하는 장치의 자동화된 탐지와 부착을 조정하고 새로운 장치의 핫플러깅 (hot-plugging) 및 언플러깅 (unpluging) 그리고 필요에 따라 드라이버를 다운로드하고 설치하는 것을 용이하게 한다. 그러나 장치 접근 구조는 어떤 특정한 장치나 네트워크 기술을 규정하지 않으며 특정한 장치를 탐지하는 방법에 대해 명기하지 않는다. 이 구조는 단지 서로 다른 벤더에 의해 제공되는 장치의 부착에 대한 메커니즘에 초점을 맞추고 있다.

장치 접근 구조는 백그라운드에서 동작하는 장치

관리기의 행동을 정의하는 것으로 자바 가상 머신에서 서비스 번들이 객체로 표현되는 디바이스와 통신할 수 있도록 하는 구조를 제공한다. 그러나 장치 접근 구조는 저수준의 디바이스 드라이버를 만드는 것을 의미하지 않으며 일반적으로 네이티브 드라이버는 사전에 설치 및 설정되어 있다고 가정한다.

장치 관리기는 디바이스 서비스, 드라이버 서비스, 드라이버 탐색 서비스, 드라이버 선택 서비스 등의 행동을 조율하는 기능을 가지고 있으며, 디바이스 서비스의 등록을 탐지하고 이 디바이스에 적합한 드라이버 서비스를 드라이버 탐색 서비스 및 드라이버 선택 서비스를 통해 적합한 드라이버를 찾고 선택된 드라이버 서비스를 이용해 정제된 디바이스 서비스를 설치한다.

장치 접근 구조에서의 소프트웨어 구조는 그림 8과 같다.

자바 프로그래밍 언어로 만들어진 응용 프로그램은 물리적 장치에 접근하기 위해 간접적으로 몇 가지 층을 통해야 한다. 디바이스 드라이버를 통해 하드웨어와 통신할 수 있는 기반 운영체계와 통신하기 위해서 응용 프로그램은 JNI (java native interface)을 경유해야 한다.

일반적으로 장치 접근 구조는 하나의 장치 관리기, 하나의 드라이버 선택 서비스 및 여러 가지의 디바이스 서비스, 드라이버 탐색 서비스, 드라이버 서비스를 가지고 있으며 다음과 같은 역할을 수행한다.

■ 디바이스 탐지 (device detection): 네이티브 드라이버의 도움으로 장치 접근 구조는 어떤 디바이스가 연결되었는지 탐지하고 그 기능을 나타내는 디바이스 서비스를 등록한다.

■ 디바이스 개량 (device refinement): 장치 관리기는 드라이버 서비스를 참고하여 디바이스 서비스를 개량하고 새로운 디바이스 서비스를 등록한다. 이 과정에서 장치 관리기는 드라이버 탐색 서비스를 통해 드라이버를 찾고 드라이버 선택기를 통해 드라이버를 선택한다.

■ 재구성 (reconfiguration): 디바이스 개량 과정에서 어떤 드라이버 서비스도 발견되지 않으면 장치 관리기는 다른 방식으로 디바이스를 표현하는 디바이스 서비스를 등록할 수 있다.

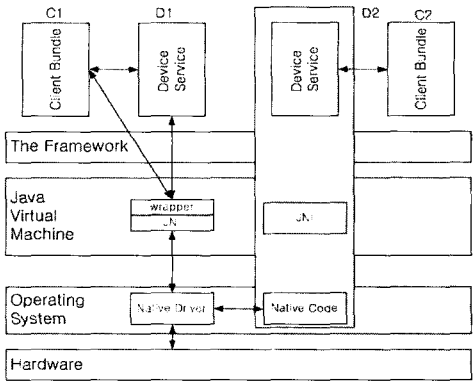

그림 8. 하드웨어 장치에 대한 접근

2.3.1 디바이스 서비스

디바이스 서비스는 물리적인 장치에 의해 제공되는 기능을 서비스 인터페이스를 통해 표현하는 것으로 OSGi 장치 접근 구조에서 정의된 표준 API를 반드시 구현해야 한다. 예를 들어 병렬 포트와 관련된 디바이스 서비스를 정의하려면 스트림 입출력 기능을 표현하는 서비스 인터페이스를 정의하고 OSGi 표준 API인 Device 인터페이스를 구현하면 OSGi 환경에서 디바이스 서비스로 인식이 된다.

장치 접근 구조에서 디바이스 서비스의 등록과 해지는 물리적인 장치의 부착 및 탈착과 관계된다. 예를 들어 ZIP 디스크를 병렬 포트에 연결하면 병렬 포트의 기능에 대한 디바이스 서비스가 등록되고 병렬 포트를 사용하는 클라이언트 번들이 있으면 설치된 디바이스 서비스를 통해 사용할 수 있다. 또한 연결을 해지하면 이 디바이스 서비스가 등록 해지 되고 프레임워크에 의해 발생하는 서비스 등록 해지 이벤트를

감시함으로서 클라이언트 번들은 적절한 다른 행동을 할 수 있다.

클라이언트의 관점에서 보면 같은 장치에 대해 다른 수준으로 접근을 요구할 수 있다. 예를 들어 ZIP 드라이버가 연결될 경우 대부분의 응용 프로그램은 병렬 포트를 통해 비트 스트림을 바로 전달하기 보다는 파일을 저장하고 지우기를 원할 것이다. 따라서 ZIP 드라이버 디바이스 서비스와 같은 상위 레벨의 추상화를 선호 할 것이다. 이러한 경우 병렬 포트 디바이스 서비스는 ZIP 드라이버 디바이스 서비스로 개량 될 수 있다. 만약 두 개의 디바이스 서비스가 등록되어 있으면 클라이언트 번들은 같은 장치에 대해 서로 다른 수준으로 장치에 접근할 수 있다.

디바이스 서비스를 등록할 때 DEVICE_CATEGORY라는 이름으로 적어도 하나의 속성은 가지고 등록 되어야 한다. 디바이스 서비스를 등록할 때 사용되는 이러한 속성은 디바이스의 특징에 대해 정의하고 디바이스 개량 과정에서 기반 정보로 사용된다

디바이스 개량을 위해 현재 존재하는 디바이스 서비스를 평가하고 다른 디바이스 서비스를 등록하는 책임은 드라이버 서비스에게 있다.

2.3.2 드라이버 서비스

장치 접근 구조에서 드라이버라는 용어는 운영체계의 저수준 디바이스 드라이버를 의미하지 않는다. 저수준 디바이스 드라이버는 OSGi에서 다루는 영역을 벗어나는 것으로 여기서는 기반 드라이버 (base driver)로 언급한다. 일반적으로 기반 드라이버는 장치를 탐지하는 책임을 가진다. 예를 들어 기반 드라이버는 장치가 병렬 포트에 연결되었는지를 확인하기 위해 주기적으로 폴링을 하고 연결 확인이 되면 병렬 포트 디바이스 서비스를 등록할 수 있다.

OSGi 장치 접근 구조에서 드라이버 서비스는 장치 관리기의 제어 하에서 존재하는 디바이스 서비스를 개량하여 가장 적합하거나 다른 수준의 디바이스 서비스를 등록하는 역할을 수행한다. 그리고 하나 이상의 드라이버 서비스를 가지고 있는 번들을 드라이버 번들이라고 한다.

드라이버 번들은 드라이버 서비스를 프레임워크에 등록해야 하고 이때 발생하는 이벤트는 장치 관리기가 새로운 드라이버 서비스가 등록되었음을 인식 하는데 사용된다. 드라이버 서비스는 DRIVER_ID 속성 값을 가지고 등록 되어야 하고 장치 관리기는 같은 드라이버 번들의 복사본이 중복해서 설치되는 것을 방지하기 위해 이 값을 사용한다. 일반적으로 DRIVER_ID는 중복을 방지하기 위해 구현 회사의 도메인 이름 역순으로 시작하여 만들어진다.

드라이버 서비스가 등록 해지 될 때 이 드라이버 서비스에 의해 등록된 모든 디바이스 서비스는 무효 상태가 되어야 하고 장치 관리기는 이런 모든 무효 디바이스 서비스를 관리하고 다시 부착 시켜야 한다.

구현된 드라이버 서비스가 OSGi 환경에서 드라이버 서비스로 인식되기 위해서는 장치 접근 구조에서 정의하는 표준 API인 Driver를 구현해야 한다.

2.3.3 디바이스 카테고리

디바이스 카테고리는 디바이스 서비스와 드라이버 서비스 사이에서 통신하는데 필요한 규칙과 인터페이스를 규정하는데 같은 디바이스 카테고리의 디바이스 서비스와 드라이버 서비스만이 유일하게 통신할 수 있고 협동할 수 있다.

장치 접근 구조는 드라이버 서비스에 의한 디바이스 서비스의 부착에 대해서 명세하고 개별 디바이스 카테고리에 대해서는 규격화 하지 않는다.

디바이스 카테고리는 USB, IEEE1394, Jini, UPnP, Salutation, CEBus, Lonworks와 같은 특정한 디바이스 기술과 연관 된다. 디바이스 카테고리의 목적은 모든 디바이스 서비스를 합의된 인터페

이스로 사용하기 위한 것이다. 예를 들어 벤더 A의 USB 드라이버 서비스는 USB 버스에 연결된 벤더 B의 디바이스 서비스를 제어할 수 있다.

디바이스 카테고리를 정의할 때 다음과 같은 요소를 포함하고 있어야 한다.

- 디바이스 카테고리에 소속된 모든 디바이스가 반드시 구현해야 하는 인터페이스: 기본적인 디바이스와 어떻게 통신할 것인가에 대한 규칙 설계
- 서비스 등록 프로퍼티 및 데이터 타입
- 디바이스 카테고리에 대한 매칭 값의 범위

2.3.4 드라이버 탐색 서비스

드라이버 탐색 서비스는 특정한 디바이스 서비스에 필요한 드라이버 번들을 찾고 다운로드 할 수 있는 정보를 가지고 있다. 이러한 탐색 과정에서 사용되는 정보는 디바이스와 함께 등록되는 DEVICE_CATEGORY 및 다른 속성 값들이다.

결국 장치 접근 구조에서 새로운 번들의 설치에 대한 결정은 장치 관리기에 의존하고 어떤 번들이 설치되고 어디서 다운로드 받을지에 대한 결정은 드라이버 탐색 서비스에 의해 의존한다.

OSGi 환경에서 여러 개의 드라이버 탐색 서비스 설치가 가능하고 장치 관리기는 이들 모두를 사용할 수 있고 DRIVER_ID 값을 기반으로 중복 되는 것을 관리한다.

2.3.5 드라이버 선택 서비스

드라이버 선택 서비스의 목적은 적합한 드라이버 번들 중에서 최상의 드라이버 서비스를 선택하는데 있어 커스터마이징을 허용하기 위한 것이다.

장치 관리기는 드라이버 선택 서비스에 대한 디폴트 알고리듬을 가지고 있다. 그러나 이 알고리듬이 불충분하고 오퍼레이터에 의해 커스터마이징이 요구되면 드라이버 선택 서비스를 제공하는 번들이 프레임워크에 설치될 수 있다. 이 서비스는 디바이스 서비스에 대한 최상의 매치를 이루는 것을 선택하기 위해 최종적으로 장치 관리기에 의해 사용되어야 한다.

드라이버 선택 서비스는 OSGi 실행 중에 하나만 동작해야 하고 장치 관리기에 의해 하나만 선택된다.

2.3.6 장치 관리기 (device manager)

장치 접근 구조는 백그라운드에서 실행되는 장치 관리기에 의해 제어된다.

장치 관리기는 디바이스 서비스, 드라이버 서비스, 드라이버 탐색 서비스, 드라이버 선택 서비스 등의 행동을 조율하는 기능을 가지고 있으며, 디바이스 서비스의 등록을 탐지하고 이 디바이스에 적합한 드라이버 서비스를 드라이버 탐색 서비스 및 드라이버 선택 서비스를 통해 적합한 드라이버를 찾고 선택된 드라이버 서비스를 이용해 정제된 디바이스 서비스를 설치한다.

장치 관리기는 서비스 리스너 (service listener)를 통해 디바이스가 등록되는 것을 탐지하고 탐지된 디바이스 서비스에 적합한 드라이버 번들을 찾기 위해 드라이버 탐색 서비스를 사용하여 현재 설치되어 있지 않은 드라이버 서비스를 찾는다. 드라이버 탐색 서비스에 의해 다수의 드라이버 서비스가 등록되면 장치 관리기는 그림 9와 같은 디바이스 부착 알고리듬을 사용하여 최상의 매치를 이루는 드라이버 서비스를 드라이버 선택 서비스를 통해 결정하고 개량된 디바이스 서비스를 부착하게 된다.

디바이스 부착의 예를 구체적으로 설명하면 그림 10과 같다. 그림 10에서 S1 및 S2는 디바이스 서비스이고 DM은 장치 관리기이다. 그리고 DL1 및 DL2는 드라이버 탐색 서비스이고 D1, D2 및 D3는 드라이버 서비스이다. 디바이스 부착에 대한 과정을 살펴보면 다음과 같다.

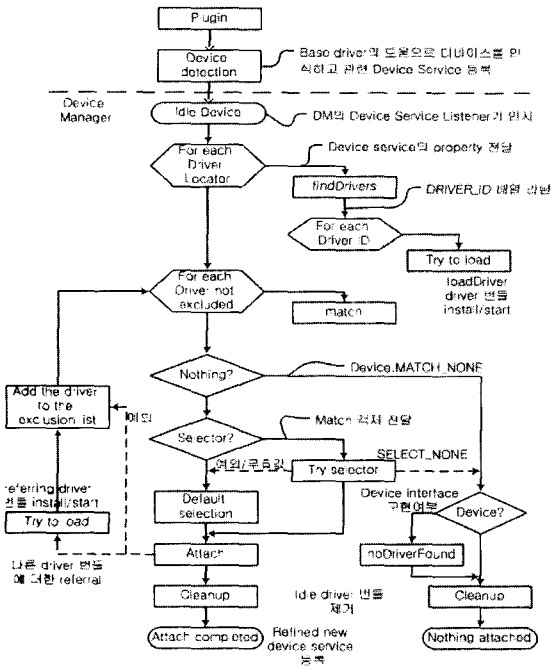

그림 9. 디바이스 부착 알고리듬

a. 디바이스 서비스 S1이 프레임워크에 등록되면 DM은 새로운 디바이스 서비스의 등록을 탐지하고 현재 존재하는 두 개의 드라이버 탐색 서비스에 의뢰하여 어느 번들이 디바이스 서비스 S1을 개량할 수 있는지 확인한다. DL2가 가능한 드라이버 탐색 서비스라는 결과를 수신한다.

b. DM은 DL2가 이 드라이버를 다운로드 하도록 요청하고 이것을 프레임워크에 설치 및 기동 시킨다.결과적으로 새로운 드라이버 서비스, D3가 등록된다.

c. DM은 각각의 드라이버 서비스에 대해 디바이스 서비스 S1을 갱신할 수 있는 드라이버 서비스인지 질의 한다. D2가 최상의 매칭을 이룬다는 결과를 수신한다.

d. DM은 내부적으로 가지고 있는 드라이버 선택 서비스를 통해 D2를 선택하고 갱신된 새로운 디바이스 서비스 S2를 등록한다. 이 과정을 새로운 디바이스 서비스 S2에 대해서도 반복한다.

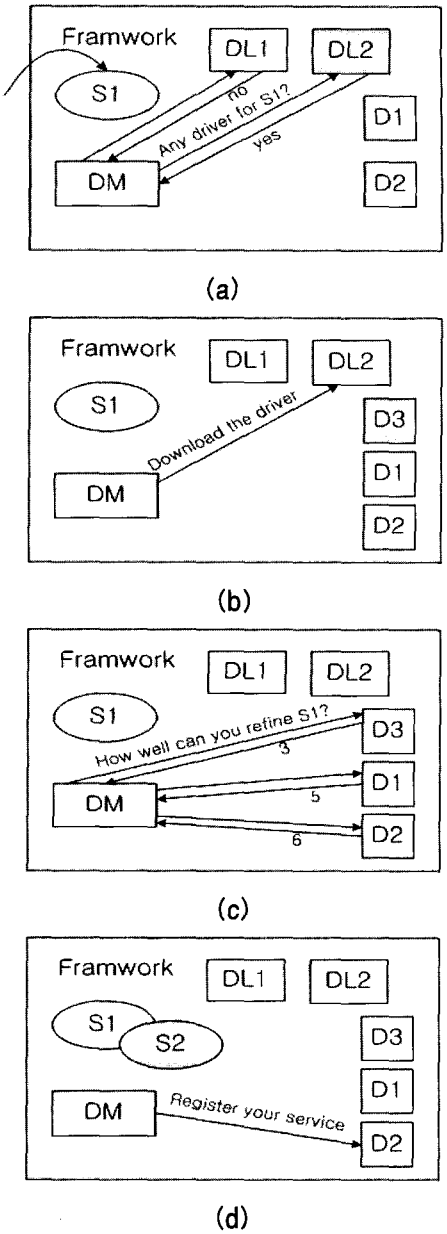

그림 10. 디바이스 부착 예

2.4 Jini 및 UPnP 연계를 통한 장치와 서비스의 발견 및 제어 구조

외부의 서비스 제공자(Service Provider)들이 홈 게이트웨이를 통하여 댁내의 가전이나 장비들에 특정

의 서비스를 제공하게 하기 위하여는 홈네트워크 미들웨어 표준을 홈서버의 개방형 서비스 프레임워크에서 구현하여 지원하여야 한다. 개방형 서비스 플랫폼인 OSGi에서는 이러한 홈네트워크 미들웨어를 번들의 형태로 구현하여 이러한 기능을 지원한다. 즉, 서비스 제공자들은 자신이 제공할 특정의 서비스를 OSGi의 서비스 번들로 제작하여 홈서버에 다운로드하는데, 이 서비스 번들들이 홈네트워크 장치들을 서비스에 이용한다면 OSGi 플랫폼에 있는 홈네트워크 미들웨어의 번들의 OSGi 서비스를 사용하여 이를 홈네트워크 장치들과 통신하고 제어하게 된다. OSGi는 홈네트워크 미들웨어인 Jini와 UPnP와 연계를 통하여 Jini와 UPnP를 미들웨어로 가지는 홈네트워크 장치와 상호 동작한다. 이를 위하여 Jini Driver Service specification과 UPnP Device Service Specification이 존재한다.

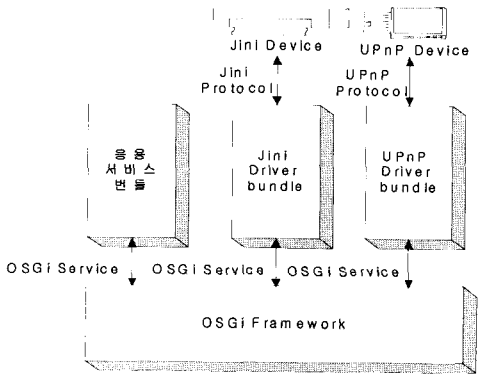

그림 11. OSGi와 Jini, UPnP와의 연계 구조

2.4.1 Jini Driver Service Specification

Jini는 Sun Microsystems사가 개발 하였으며, Lookup Server를 두어 네트워크 상의 서비스 발견을 지원한다. Jini는 새로이 사용되고자 하는 서비스가 Lookup Service를 찾는 기능 (Discovery), Lookup Service에 서비스 기능 등록 (Join), 서비스를 이용하고자 하는 클라이언트의 발견 요청 (Lookup) 및 서비스 요청 기능(Invocation), Timeout 을 이용한 서비스 이용 기능 (Lease) 및 이벤트 기능을 제공한다.

Jini와 OSGi의 연계를 위한 Jini Driver Service는 3절에서 다룬 장치 접근 구조를 따르고, Jini lookup service와 OSGi 서비스를 처리한다. 따라서, OSGi번들은 Jini network와 OSGi service platform 사이의 bridge 기능을 하게 되어 OSGi-to-Jini export와 Jini-to-OSGi import 기능을 수행한다.

Jini service를 impot하거나 export하기 위하여는 Jini lookup service를 Jini network에서 discovery하여야 한다. Jini driver는 이러한 lookup service를 찾아 OSGi framework 상에 proxy object를 만들어 net.jini.core.lookup.ServicRegistar OSGi 서비스로 등록한다.

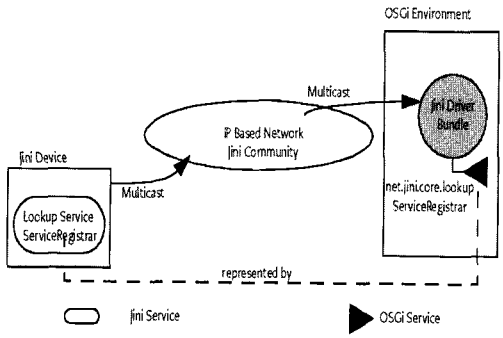

그림 12. Jini lookup service discovery[6]

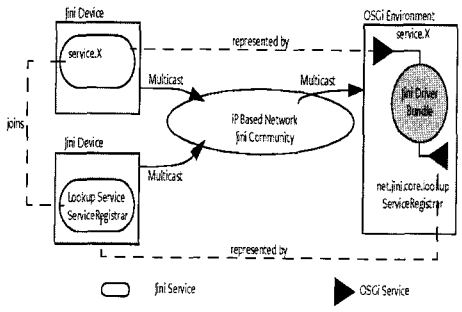

그림 13. Jini service import 구조[6]

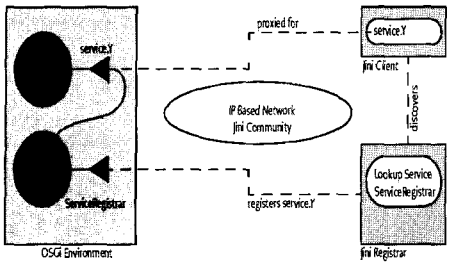

그림 14. Jini service export 구조[6]

Jini driver service는 이 lookup service에 있는 Jini network 상의 Jini service를 OSGi 상의 OSGi Device service로 등록하여 다른 OSGi 번들이 사용할 수 있게 한다. 이 때 OSGi Device service의 DEVICE_CATEGORY는 jini의 값을 가진다.

Jini driver service는 OSGi Device service를 Jini network 상의 lookup service에 등록함으로 Jini network 상의 Jini 장치들이 OSGi service platform 상의 OSGi Device service를 Jini device로 인식하고 사용할 수 있게 한다. 이 때 OSGi Device service의 DEVICE_CATEGORY는 jini의 값을 가진다.

2.4.2 UPnP Device Service Specification

UPnP (Universal Plug and Play)는 마이크로소프트사가 제안한 미들웨어로서 기존의 IP 네트워크와 HTTP 등의 프로토콜을 사용하여 홈네트워크 상의 기기들이 제공하는 서비스를 자동으로 발견하고, 사용할 수 있도록 한다. UPnP에 대한 보다 자세한 내용은 3장을 참조하기 바란다.

UPnP (Universal Plug and Play)의 경우는 Java 기술을 바탕으로 하지 않기 때문에 OSGi에서는 UPnP Device service에 대한 표준을 3절에서 설명한 장치 접근 표준과는 별도로 제정하였다. UPnP Base Driver는 UPnP network 상의 UPnP 장치를 discovery하여 OSGi framework 상의 UPnP Device service로 등록한다. 또, OSGi framework 상의 UPnP Device service를 UPnP network 상에 advertise하여 UPnP network 상에서 UPnP device처럼 동작할 수 있도록 한다.

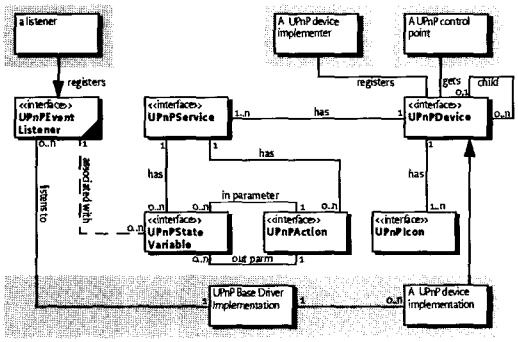

그림 15. UPnP Device service specification 구성도[6]

표 1. OSGi UPnP Device service 표준 인터페이스

인터페이스	기능
UPnPDevice	Represents an UPnP device
UPnPService	Represents an UPnP service.
UPnPAction	Represents an UPnP action.
UPnPStateVariable	an UPnP state variable as declared in the device's service state table (SST).
UPnPIcon	Represents an UPnP icon.
UPnPEventListener	UPnP Events are mapped and delivered to applications according to the OSGi whiteboard model.

UPnP Device service specification은 UPnP stack에서 정의한 UPnP entity들을 표 1과 같은 interface로 정의한다. 각 OSGi UPnP device service는 표 1과 같은 인터페이스들을 구현하여야

하고, DEVICE_CATEGORY는 UPnP의 값을 가진다. UPnP network 상의 UPnP device의 statevariable의 event에 가입 (subscribe)하여 event를 수신하고자 하는 OSGi 번들이 UPnPEventListener interface를 구현하여 이를 OSGi framework에 OSGi service로 등록하면, UPnP base driver는 UPnP network 상의 UPnP device에서의 event가 발생 하는 경우에 UPnPEventListener의 notifyUPnPEvent method를 이용하여 가입한 번들에 이 event를 전달하는 whiteboard model을 사용한다.

3. UPnP에서의 홈네트워크 장치 제어 기술

3.1 UPnP 소개

홈네트워크 미들웨어는 각종 홈네트워크 가전 및 장비 제조업체가 공통으로 사용할 수 있는 제어 및 통신 프로토콜로 각 장치들 간의 상호 운용성을 가지게 한다.

UPnP는 로컬 PC에서 제공되던 Plug and Play와 기능을 네트워크 상에서 제공하기 위하여, Microsoft가 주축이 되어 구성된 UPnP Forum (http://www.upnp.org)에서 제정된 미들웨어 규약으로, IP 기반의 네트워크에서 사용된다.

기존에 설치된 IP 네트워크와 많은 회원 회사로 인하여 UPnP는 주요한 홈네트워크 미들웨어가 될 것으로 추측된다.

UPnP는 IP 네트워크에서 장치들이 서로 동적으로 네트워크에 진입하고 각각의 기능과 존재를 다른 장치와 공유하여 제어하는 미들웨어이다. 기본적으로 네트워크에 진입하게 되면 UPnP 장치들은 addressing, discovery, description, control, event들을 수행하게 된다.

Addressing은 UPnP 장치에 IP 주소를 부여하는 기능으로 DHCP와 Auto IP 표준을 채택한다. Discovery는 네트워크에 진입한 장치들의 존재를 발견하거나 알리는 기능을 수행하고 description은 장치들의 기능을 XML을 이용하여 서로 알리는 기능이다. Control은 description을 통하여 알려진 각 장치의 기능을 서로 이용하는 것으로 client/server 모델을 기본으로 한다. Event는 각 장치에서 일어나 변화를 수신 가입 (subscription)한 다른 장치에 알리는 기능이다.

장치의 기능을 공유하고 발견하기 위하여는 peer-to-peer discovery model을 사용하여 XML 문서를 통한 SSDP (Simple Service Discovery Protocol)을 이용한다.

UPnP는 UPnP 장치를 표현하는 추상적인 개념으로 device, service, action, statevariable 등을 이용한다. Device는 UPnP 장치 자체를 지칭하는 것으로 각 device는 몇 개의 service들로 구성된다.

각 device는 또 내장된device를 가질 수도 있는데, 이 때 가장 최상위의 device를 root device라고 하고 다른 장치에 내재된 device를 embedded device라고 한다. 각 service는 action과 statevariable로 구성된다.

Action은 실제 UPnP 장치들이 수행하는 기능을 의미하고 statevariable은 각 service의 상태 정보를 나타내는 것으로 action의 매개변수 (argument)와 연관성을 가지고 statevariable의 변화는 event 기능을 통하여 변화에 관심 있는 네트워크 상의 다른 장치에 GENA(General Event Notification Architecture) 프로토콜을 통하여 통지한다.

이러한 추상적인 device, service, action, statevariable들은 XML 문서에 저장되어 SSDP를 통하여 네트워크 상의 다른 장치들과 공유하게 되어 각 장치들은 네트워크 상의 다른 장치들의 기능을 알게 된다.

네트워크 상의 각 장치들이 서로의 기능을 사용하고 제어하기 위해서는 client/server model을 이용

한다. 이 때도 비교적 가벼운 XML을 바탕으로 한 SOAP (Simple Object Access Protocol)을 이용한다. 이 때 service를 제공하는 server에 해당하는 장치는 device라고 client를control point라고 한다. control point는 다른 device의 action을 invoke하기 위하여 입력 매개변수 (input argument)를 SOAP을 통하여 전달하고 server인 device는 그 결과를 SOAP을 통하여 control point에 전달한다.

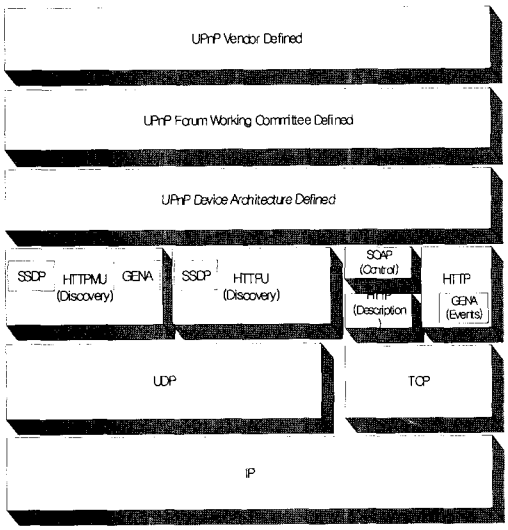

그림 16. UPnP Protocol Stack[9]

UPnP Forum은 steering commitee, working commitee, member로 구성된다. Working commitee에서는 각 UPnP device의 service, statevariable과 action 등을 정하고 이는 각 장치의 DCP (Device Control Protocol)로 확정된다. 각 장치의 DCP는 각 제조사들이 특정 장치를 제조할 때 따라야 하는 표준이 된다. 본 논문에서는 Lighting Control DCP에 대하여 소개한다.

3.2 Lighting Control DCP

UPnP Working Commitee에서 제정한 조명 제어 DCP로 BinaryLight와 DimmableLight 두가지 있다[10, 11]. BinaryLight는 on/off 제어가 가능하고 DimmableLight는 dimming 기능이 추가되어 있다.

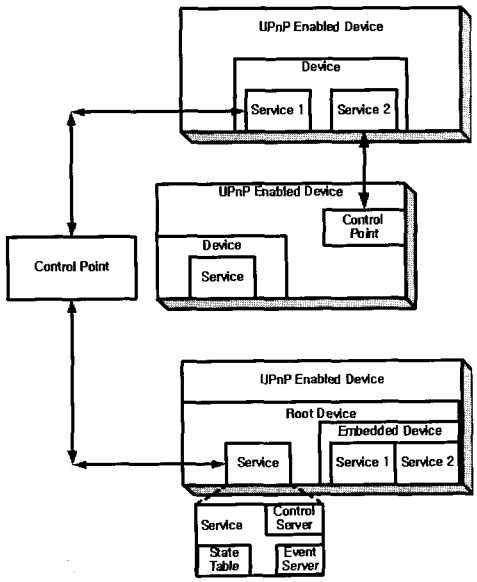

그림 17. UPnP entities[9]

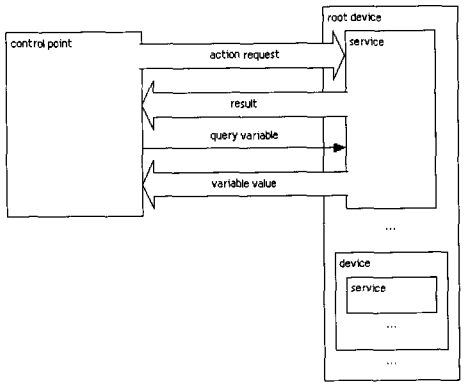

그림 18. UPnP Control 구조

즉, BinaryLight Device는 SwitchPower service를 가지고 있고, DimmableLight는 SwitchPower service외에도 Dimming service를 가진다. SwitchPower service와 Dimming service는 직렬

형태로 연결되어져 있어 power가 off되어 있으면 dimmer 기능은 동작하지 않는다. UPnP Control Point는 BinaryLight와 DimmableLight의 각 서비스의 UPnP action을 SOAP을 통하여 사용하여 조명장치를 제어할 수 있다.

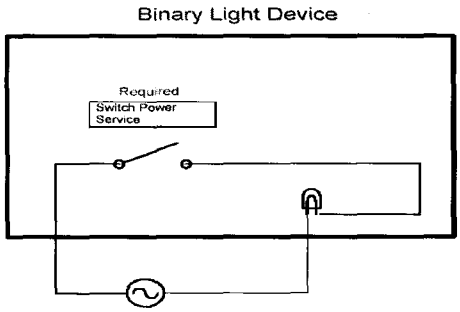

그림 19. BinaryLight Device 개념도[11]

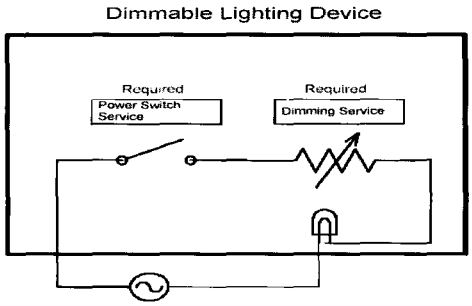

그림 20. DimmableLight Device 개념도[11]

SwitchPower Service는 조명원을 on/off하는 기능을 담당한다. 이 UPnP service의 statevariable과 UPnP action은 다음과 같다.

표 2. SwitchPower Service의 statevariable들

Statevariable	Req. or Opt.[1]	Data type	Evented
Target	R	Boolean	No
Status	R	Boolean	Yes

[1]R = Required O=Optional

- Target: 원하는 조명의 상태를 나타내는 statevariable로 0이면 off, 1이면 on을 의미한다.
- Status: 실제 조명 장치의 상태를 나타낸다.

표 3. SwitchPower Service의 action들

Actions	Req. or Opt.[1]
SetTarget	R
GetTarget	R
GetStatus	R

[1]R = Required O=Optional

- SetTarget: 원하는 Target값을 지정하는 action이다.
- GetTarget: 디버깅을 위하여 사용하는 action이다.
- GetStatus: 조명 장치의 On/Off 상태를 알려준다.

Dimming Service는 dimmer 기능을 제공하고 ramping 동작을 지원한다.

이 UPnP service의 statevariable과 UPnP action은 다음과 같다.

표 4. Dimming Service의 statevariable들

Statevariable	Req. or Opt.[1]	Data type	Evented
LoadLevelTarget	R	ui1	No
LoadLevelStatus	R	ui1	Yes
OnEffectLevel	O	ui1	No
OnEffect	O	string	No
StepDelta	O	ui1	Yes
RampRate	O	ui1	Yes
IsRamping	O	Boolean	Yes
RampPaused	O	Boolean	Yes
RampTime	O	ui4	No

[1]R = Required O=Optional

- LoadLevelTarget:원하는 조명 수준으로 0부터 100값을 가진다.
- LoadLevelStatus: 실제 장치의 조명 수준으로, 원하는 레벨과 모든 시간에 일치하지는 않는다.
- OnEffectLevel: Power가 켜질 때 초기의 LoadLevelTarget 값을 지정할 때 사용한다.
- OnEffect: Power가 켜질 때 초기의 LoadLevelTarget값을 어떻게 지정할 것인가를 나타낸다. OnEffectLevel은 OnEffectLevel값을 LoadLevelTarget값으로 지정하고, Lastsetting은 off할 때의 마지막 값을 지정하고, default는 미리 지정된 값을 지정한다.
- StepDelta: StepUp이나 StepDown action에서 LoadLevelTarget을 한번에 변화시키는 양을 나타낸다.
- RampRate: RampUp이나 RampDown 명령에서 1초에 변화하는 양을 나타낸다.
- IsRamping: 현재 ramping이 진행되면 1의 값을 가진다.
- RampPaused: 현재 ramping이 pause되었다면 1의 값을 가진다.
- RampTime: StartRampToLevel 명령에서 LoadLevelTarget에 도달하는 시간을 나타낸다.
- SetLoadLevelTarget: 원하는 조명 정도를 지정한다. 이 값에 따라 조명 장치는 원하는 조명 정도로 장치를 조절한다.
- GetLoadLevelTarget: 현재의 지정된 조명 정도를 알려준다.
- GetLoadLevelStatus: 현재 실제 장치의 조명 정도를 알려준다.
- SetOnEffectLevel: Power가 on될 때 사용되는 OnEffectLevel 값을 지정한다.
- SetOnEffect: Power가 on될 때의 조명 정도를 어떻게 결정할 것인가를 지정한다.
- GetOnEffectParameters: 현재 지정된 OnEffectLevel과 OnEffect 값을 알려준다.
- StepUp: LoadLevelTarget 값을 StepDelta만큼씩 증가시킨다. LoadLevelTarget가 최대치에 도달하면 최대치 100을 유지한다.
- StepDown: LoadLevelTarget 값을 StepDelta만큼씩 감소시킨다. LoadLevelTarget가 최소치에 도달하면 최소치 0을 유지한다.
- StartRampUp: RampRate에 의하여 지정된 속도로 LoadLevelTarget을 증가시킨다. LoadLevelTarget가 최대치에 도달하면 ramping을 멈춘다. IsRamping은 1이 되고, RampPaused는 0이 된다.
- StartRampDown: RampRate에 의하여 지정된 속도로 LoadLevelTarget을 감소시킨다. LoadLevelTarget가 최소치에 도달하면 ramping을 멈춘다. IsRamping은 0이 되고, RampPaused는 1이 된다.
- StopRamp: ramping 동작을 멈춘다. IsRamping은 0이 되고, RampPaused는 0이 된다.
- StartRampToLevel: 현재의 조명 정도에서 원하는 조명 정도까지 RampTime 동안 ramping시킨다. Ramping이 시작되면 RampTime이 줄어든다. RampTime이 0이 되면 ramping을 멈춘다. IsRamping은 1이 되고, RampPaused는 0이 된다.
- SetStepDelta: StepDelta 값을 지정한다.
- GetStepDelta: 현재의 StepDelta 값을 알려준다.
- SetRampRate: RampRate 값을 지정한다.
- GetRampRate: 현재의 RampRate 값을 알려준다.
- PauseRamp: ramping을 멈춘다. RampPaused가 1이 된다.
- ResumeRamp: ramping 동작을 다시 시작한다. RampPaused가 0이 된다.

- GetRampPaused: RampPaused 값을 알려준다.
- GetRampTime: RampTime을 알려준다.
- GetIsRamping: IsRamping 값을 알려준다.

표 5. Dimming Service의 actions들

Actions	Req. or Opt.
SetLoadLevelTarget	R
GetLoadLevelTarget	R
GetLoadLevelTarget	R
SetOnEffectLevel	O
SetOnEffect	O
GetOnEffectParameters	O
StepUp	O
StepDown	O
StartRampUp	O
StartRampDown	O
StopRamp	O
StartRampToLevel	O
SetStepDelta	O
GetStepDelta	O
SetRampRate	O
GetRampRate	O
PauseRamp	O
ResumeRamp	O
GetRampPaused	O
GetRampTime	O
GetIsRamping	O

[1]R = Required O=Optional

4. 결 론

디지털과 정보 통신 기술의 발달로 인하여 가정 내의 가전 및 전기 장치들이 홈네트워크로 연결되어 가까운 미래에 디지털 홈 서비스가 등장할 것으로 예상된다. 이러한 서비스를 위하여 홈네트워크 장치의 제어를 위한 미들웨어는 다양한 제조사들이 만든 홈네트워크 장치에 상호 운용 기능을 제공한다. 본 논문에서는 홈네트워크 장치를 제어하기 위한 미들웨어 기술인 OSGi와 UPnP에서의 장치 제어 구조에 대하여 살펴 보았다.

OSGi는 서비스 제공자가 가정 내의 홈네트워크 장치에 서비스를 제공하는 공개 표준 서비스 플랫폼으로 본 논문에서는 OSGi의 장치 접근 구조와 홈네트워크 미들웨어와의 연계 방법에 대하여 기술하였다.

UPnP는 IP 네트워크 상에서 장치들 간의 addressing, discovery, control, event 등의 기능을 제공한다. UPnP forum의 working committee에서는 각 UPnP 장치의 DCP를 제정하여 특정 장치의 표준을 지정한다. 본 논문에서는 lighting control DCP를 홈네트워크 장치 제어의 예로 기술하였다.

OSGi와 UPnP는 각각 대표적인 서비스 공급 플랫폼과 홈네트워크 미들웨어로 홈네트워크 장치들에 서비스를 제공하고, 상호 운용성을 제공한다. 서비스 공급자는 OSGi와 UPnP를 바탕으로 하여 홈네트워크 장치를 제어하는 다양한 서비스를 제공하여 디지털 홈 서비스나 스마트 홈 서비스를 가정의 사용자에게 제공할 수 있다.

참 고 문 헌

[1] 이전우, 배창석, "홈서버 플랫폼 기술", 정보과학회지, 제19권, 제4호, pp. 7-15, 2001. 4.

[2] 배창석, 유진호, 강동오, 강규창, 이전우, "홈서버 구현 기술", 정보통신기술, 제16권, 제2호, pp. 38-58, 2002. 11.

[3] 박윤경, 강규창, 강동오, 유동완, 이전우, "디지털 홈 서비스 미들웨어 및 응용 서비스 기술", 한국통신학회지, 제 20권, 제6호, pp. 56-65, 2003. 6.

[4] John Barr, "Managed Services for the New Millennium", OSGi World Congress CD, September 24~27, 2002, Stockholm, http://www.osgi.org.

[5] BJ Hargrave, "OSGi Specification Evolution", OSGi World Congress CD, September 24~27, 2002, Stockholm,

http://www.osgi.org.
[6] OSGi "OSGi R3 specification", http://www.osgi.org.
[7] Kirk Chen, Li Gong, "Programming Open Service Gateways with Java Embedded ServerTM Technology", 2002, ADDISON-WESLEY.
[8] Pavin Dobrev, David Famolari, Christian Kurzke, Brent A. Miller, "Device and Service Discovery in Home Networks with OSGi", IEEE Communication Magazine, August, 2002.
[9] Understanding Universal Plug and Play, white paper, Microsoft Corp., 2000.
[10] BinaryLight:1 Device Template Version 1.01, http://www.upnp.org
[11] DimmableLight:1 Device Template Version 1.01, http://www.upnp.org
[12] SwitchPower:1 Service Templae Version 1.01, http://www.upnp.org
[13] Dimming:1 Service Template Version 1.01, http://www.upnp.org

◇ 저 자 소 개 ◇

강동오(姜東吾)
1969년 1월 16일생. 1994년 연세대학교 전자공학과 학사. 1996년 한국과학기술원 전기 및 전자공학과 석사. 2001년 한국과학기술원 전기 및 전자공학과 박사. 2001년~현재 한국전자통신연구원. 현재 차세대PC 및 홈네트워크 프레임워크 기술 개발 과제 수행 중. 주관심 분야는 홈네트워크 미들웨어 기술, 분산제어, 지능제어.

강규창(姜奎昌)
1971년 6월 7일생. 1994년 경북대학교 전자공학과 학사. 1997년 경북대학교 전자공학과 석사. 1997년 ~ 2000년 국방과학연구소 종합시험단에서 도플러 레이다 신호 분석기 및 영상 계측 기법 개발. 2001년 ~ 2003년 한국전자통신연구원 컴퓨터소프트웨어 연구소에서 홈서버 및 홈게이트웨이 개발. 현재 개방형 홈네트워크 프레임워크 및 컨텍스트 인식기술 개발 과제 수행 중. 관심분야: 미들웨어 기술, 모바일 애플리케이션, 서비스 플랫폼 기술.

이전우(李銓佑)
1961년 6월 26일생. 1983년 경북대학교 전자공학과 학사, 1985년 경북대학교 전자공학과 석사, 1998년 경북대학교 전자공학과 박사, 1985년~현재 한국전자통신연구원, 현재 차세대PC 및 홈네트워크 프레임워크 기술 개발 과제 수행 중. 주관심분야는 차세대PC 서비스 플랫폼 및 미들웨어, 컨텍스트 인식.