이질적인 홈 네트워크 미들웨어 상호 연동성 지원을 위한 사용자 중심의 시나리오 기반 통합 구조

(A Scenario-Based User-Oriented Integrated Architecture for Supporting Interoperability among Heterogeneous Home Network Middlewares)

김 민 찬 † 이 학 진 † 김 성 조 ††
(Min Chan Kim) (Hark Jin Lee) (Sung Jo Kim)

요 약 정보가전제어를 위해 Havi, Jini, LonWorks, UPnP, SLP 등 여러 미들웨어들이 현존하고 있음에도 불구하고, 홈네트워크가 계속 진화함에 따라 다양한 정보가전들에 대해 특화된 새로운 미들웨어들이 계속해서 등장할 것으로 예상된다. 본 논문은 홈네트워크 상에서 이질적인 미들웨어 간의 상호 연동을 위한 통합 구조 방식에 대해 고찰하고, 효율적인 홈오토메이션을 위하여 기존 방식과는 달리 이질적인 미들웨어를 통합하는 시나리오 기반의 사용자 중심 통합 구조를 제안하고 구현한다. 본 논문에서 제안한 HOMI 구조(HOmenetwork Middleware for Interoperability)는 사용자가 서비스 연동 시나리오를 스크립트 방식의 인터프리터 언어인 HOMIL(HOMI Language)를 이용하여 직접 작성하고 변경할 수 있는 인터페이스를 제공한다. HOMI는 이러한 인터페이스를 통하여 기존의 통합 미들웨어 구조와는 다르게 사용자들이 직접 시나리오를 작성하고 구성함으로써 홈오토메이션을 위한 이기종 가전들간 연동의 효율성과 편리성을 향상시켰다. HOMI는 연동 서비스를 시간 문맥, 동기 문맥, 비동기 문맥 등 3가지로 분류하고, 특정 이벤트가 발생하였을 때 문맥을 고려하여 다음 서비스가 수행되도록 지원한다. HOMI는 변경된 시나리오가 댁내의 홈네트워크 환경에 즉각적으로 반영될 수 있도록 함으로써 사용자들이 새로운 시나리오 적용을 위해 새로운 응용을 설치하거나 서버를 갱신하고 재부팅하는 과정 없이 계속적으로 서비스들을 받을 수 있도록 지원한다. 마지막으로 HOMI는 통합 미들웨어를 위한 중앙 집중형 구조에서 발생하는 부하 문제를 Agent들을 여러 장치에 분산할 수 있도록 함으로써 해결하였다.

키워드 : 홈 네트워크, 홈 서버, 홈 게이트웨이, 시나리오

Abstract Although there exist many middlewares such as Havi, Jini, LonWorks, UPnP, and SLP, new middlewares specialized for diverse information appliances are expected to appear continuously as home networks evolve. In this paper, we examine an integrated architecture for supporting interoperability among heterogeneous middlewares under home network, we also propose and implement a scenario-based user-oriented integrated architecture for efficient home automation which is different from existing methods. HOMI(Homenetwork Middleware for Interoperability) architecture proposed in this paper provides interfaces that assist users with designing and modifying desirable scenarios using a script interpreter language HOMIL(HOMI Language). Different from an existing integrated middleware architecture, HOMI improves efficiency and convenience of interoperation between heterogeneous appliances for home automation allowing users to design and organize scenarios through these interfaces. HOMI classifies interoperation services into time context, synchronization context, and asynchronization context and helps to execute next services considering

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터(홈네트워크연구센터) 지원사업의 연구결과로 수행되었음(IITA-2006-C1090-0603-0035)
† 학생회원 : 중앙대학교 컴퓨터공학과
barrios@konan.cse.cau.ac.kr
gausslee@konan.cse.cau.ac.kr
†† 종신회원 : 중앙대학교 컴퓨터공학과 교수
sjkim@cau.ac.kr
논문접수 : 2006년 4월 7일
심사완료 : 2006년 12월 27일

contexts when a specific event occurs. Applying modified scenarios immediately to home network environment, HOMI provides users with seamless services without installing new applications, updating the server, or rebooting in order to adopt new scenarios. Lastly, distribution agents into several devices, we solved the overhead problem occurred in a centralized architecture for integrated middleware.

Key words : Home Network, Home Server, Home Gateway, Scenario

1. 서 론

초고속 통신 기술, 인터넷, 하드웨어 디지털 기술의 발전은 정보가전들의 지능화를 촉진시키고 있다. 이에 따라 정보가전들의 설치를 쉽게 하며 다른 서비스(다른 응용이나 서비스들에 의해 접근되어 사용될 수 있는 소프트웨어 응용이나 장치)를 동적으로 찾아낼 수 있도록 서비스 발견과 상호 작용을 지원하는 다양한 미들웨어(UPnP[1], Jini[2], Havi[3], LonWorks[4], SLP[5] 등)들이 개발되어 왔다.

이러한 미들웨어들은 장치의 발견과 제어라는 공통의 목표를 가지고 있지만 여러 가지 면에서 상호 이질적이다. 이러한 이질성이 나타나게 된 배경은 각각의 기술들이 특정 응용 도메인을 지원하기 위하여 특화되었기 때문이다. 예를 들어, SLP는 엔터프라이즈 네트워크 환경에서 서비스를 제공하기 위하여 확장성에 많은 비중을 두고 있으며, UPnP는 SOHO 환경에 적합하도록 설계되었다. 반면 Havi는 홈 AV 기기간의 상호 연동에 초점을 맞추고 있다.

홈네트워크 미들웨어 간의 이질성은 서비스 개발자로 하여금 각 미들웨어의 특성을 고려하여 같은 서비스를 제공하는 장치들을 위한 응용을 중복해서 개발하도록 하거나, 이들 미들웨어들을 동시에 지원하기 위하여 하나의 큰 응용을 개발하여야만 한다. 이러한 이질적인 홈네트워크 미들웨어는 계속 출현할 것이며, 미래의 홈네트워크 환경에서 이질적인 서비스들과 프로토콜들이 혼재할 것으로 예상되고있다[6,7].

미들웨어들간의 이질성 극복 방안은 홈네트워크 연구에서 꼭 필요한 분야이다. 홈네트워크 기술의 발전을 위해서는 이질성 문제와 함께 댁내의 사용자들이 다양한 홈네트워크 장치들을 효율적으로 활용할 수 있도록 다양한 미들웨어간의 통합을 통한 홈네트워크 서비스 연동지원 방안이 개발되어야 한다. 이러한 서비스 연동은 이기종 홈네트워크 가전들의 혼적없는(seamless) 연결을 지원함으로써 생활의 편리성과 효율성을 제공할 수 있다.

다양한 서비스 제공을 필요로 하는 홈오토메이션 환경에서의 서비스 연동을 위해선 특정 서비스를 시작으로 각기 다른 서비스가 연속적으로 수행되어야 한다. 즉, 특정 문맥(context)에서 특정 서비스가 시작되고, 이 서비스를 시작으로 다른 서비스가 수행되며, 수행된 서비스의 결과와 그 시점에서의 문맥을 기반으로 또 다른 서비스가 실행되어야 한다. 이처럼 연속적으로 수행되는 서비스들이 시나리오를 구성하게 되며, 홈네트워크 내의 가전들의 협업을 통해 새로운 서비스가 제공된다. 본 논문은 편리하고 효율적인 홈오토메이션 환경 지원을 위한 시나리오 기반의 이기종 홈네트워크 미들웨어 통합 구조를 설계하고 구현한다.

본 논문의 구성은 다음과 같다. 제2장에서는 먼저 홈네트워크에서 다양한 장치들간의 상호 연동성 지원을 위해 현재 진행되고 있는 통합형 미들웨어 개발 관련연구들을 소개하고, 그것들의 문제점들을 분석하며, 연구 동기를 제시한다. 제3장에서는 본 논문에서 개발한 시나리오 기반의 사용자 중심 통합 미들웨어 구조의 설계에 대해 자세히 기술한다. 제 4장에서는 설계를 기반으로 구현에 대해 기술하고, 제 5장에서는 우리가 구현한 통합 구조를 테스트한 결과를 설명한다. 마지막으로, 제 6장에서는 결론과 향후 연구방향에 대하여 기술한다.

2. 관련 연구 및 연구 동기

본 논문은 먼저 이질성이 홈네트워크 미들웨어 연동에 어떤 영향을 주는지에 대해 논의한다. 일반적으로 이질성이란 특정 응용 도메인 내에 존재하는 다양한 컴포넌트의 인터페이스와 구조가 서로 다르다는 것을 의미한다[8]. 이질성은 한 시스템 내 여러 부분(예를 들어, 정보의 인코딩 방식, 네트워크 프로토콜, 데이타 포맷 등)에서 존재할 수 있다. 이러한 부분에서 각각 표준화가 이루어 진다면, 이질성으로 인한 문제점이 해결될 수 있으며, 원활한 상호 연동을 통하여 다양한 홈네트워크 사용 시나리오들이 구성될 수 있다. 하지만 TCP/IP등 기존의 네트워크 관련 미들웨어와는 달리 계속해서 새로운 형태의 가전들이 개발되고, 너무나 다양한 장치들과 서비스들이 존재하는 홈네트워크 환경의 특성으로 인하여 홈 네트워크 관련 미들웨어의 표준화는 용이하지 않다. 또한 기존의 서비스나 장치들이 진화하며 특성이 변경될 가능성이 있기 때문에 미래를 예측하기는 더욱더 어렵다. 설사 예측이 가능할 지라도 이질성이 가지고 있는 근본적인 문제를 해결하기에는 한계가 있다. 이러한 문제점은 2.2절에서 논의될 의미론적(semantic) 단계에서 나타나는데, 이러한 문제는 해결하기가 결코 쉽

지 않으며 심지어 찾아내기 조차 어려울 경우도 있다. 그 외 각 미들웨어간 서비스 발견과 호출 메커니즘의 차이에서 발생하는 문법적(syntax) 문제 또한 상호 연동을 위해서는 함께 해결되어야 한다.

2.1 이기종 홈네트워크 미들웨어 간의 연동 방식

현재 연구되고 있는 이기종 홈네트워크 미들웨어 간의 연동방법은 크게 개별 브릿지 방식과 통합 미들웨어 구조 방식 등 두 가지로 분류될 수 있다. 전자는 이기종 미들웨어들을 연동하기 위해 미들웨어간 1:1 브릿지 방식을 이용하여 상호 호환성을 보장하는 방식이다. 톰슨 멀티미디어와 필립스 등에서 UPnP-to-Havi 브릿지[9]를 개발했으며, New Orleans 대학에서도 Jini와 UPnP를 연동하기 위한 연구[10]가 진행되었다. 이 방식의 경우, 특정 두 미들웨어 간의 연동은 용이하지만, 새로운 미들웨어가 출현할 때마다 브릿지의 숫자도 증가하게 되고 연결 관계가 복잡해질 수 있으며, 다양한 미들웨어들을 연동하기 위한 일관된 방법을 제시하지 못하고 있어 확장성에 문제가 있다. 후자의 경우는 다양한 미들웨어들의 상위에 있는 하나의 추상화된 공통 계층을 통하여 각 미들웨어들을 브릿지 하는 구조를 가지고 있다. 이러한 구조는 새로운 미들웨어가 출현하더라도 해당 에이전트만 구현되면 쉽게 다른 미들웨어들과 통합될 수 있다. 와세다 대학에서는 SOAP(Simple Object Access Protocol)[11] 게이트웨이를 구성하여 미들웨어 통합[12]을 시도하였으며, OSGi Alliance[13]나 국내 ETRI[14] 등에서도 이기종 미들웨어간 연동 서비스 지원을 위한 통합 미들웨어에 대한 연구가 진행되고 있다.

2.2 통합 미들웨어 구조 기반 방식

본 논문에서 제시할 모델 또한 통합 미들웨어 구조 기반 방식이며, 다음 네 가지 이슈가 고려되어야 한다.

1) 어떻게 서로 다른 미들웨어들로 구현된 장치가 서로를 투명하게 발견할 것인가?

각 미들웨어들은 서로 다른 서비스 발견(Service Discovery) 메커니즘을 사용하고 있다. 이러한 메커니즘의 차이는 상호 호환되는 서비스를 제공하는 장치라 할 지라도 장치가 기반하는 미들웨어가 다르면 서로 인지할 수 없다.

2) 어떻게 서로 다른 미들웨어로 구현된 서비스들이 서로를 호출할 것인가?

Jini의 서비스 호출 방식은 Java 바이트코드를 이용한 Java RMI(Remote Method Invoke)를 사용하고, UPnP의 경우는 SOAP(Simple Object Access Protocol)을 사용하여 XML 텍스트 스트림을 전송하여 호출한다. 이러한 서비스 호출 메커니즘의 차이는 이기종 미들웨어 연동에서 반드시 해결되어야 할 문제이다. 이 문제의 해결을 위해 먼저 미들웨어간의 문법적인 요소(메소드 이름, 인자의 배치 순서, 각 인자나 리턴 값 타입의 크기)가 보정되어야하고, 각 미들웨어 서비스 호출 메커니즘이 요구하는 호출 방식으로의 변환이 필요하다.

3) 통합 미들웨어 구조에서 이기종간 서비스가 상호 연동이 가능하다는 것을 어떻게 알 수 있을까?

서비스 인터페이스(즉 문법적인 요소)가 같다고 상호 연동되는 서비스가 제공되는 것으로 간주될 수는 없다. 예를 들어, 표 1의 Jini는 스토리지 서비스 제공 메소드인 PutFile 이용하여 새로운 데이타를 파일로 저장한다고 가정하자. 마찬가지로 표 1의 UPnP 스토리지 서비스는 SCPD(Service Control Protocol Description) 안에 PutFile 메소드를 이용하여 파일을 저장한다고 가정하자. 그러나 UPnP 스토리지 서비스는 먼저 사용자 인증 과정을 거친 후에만 새로운 데이타가 파일에 저장되게 된다. 사용자 인증이 실패하면, 에러가 반환된다. 위의 경우 서비스 인터페이스의 문법적(syntax)인 요소는 동일해 보이지만, 의미론적(semantics) 요소의 차이로 인하여 상호 연동을 위해서는 의미론적 요소를 보정하기 위한 방안이 필요하다. 이렇게 문법상으로는 일치하지만 의미상으로는 일치하지 않는 서비스를 상호 연동하기 위한 대표적인 방법으로서 각 서비스(예: TV, MP3, 프린터 등)마다 통합 미들웨어 표준 인터페이스를 정의하고, 그 정의에 따라 각 미들웨어별로 테이블 형태의 변환 브릿지를 구현하는 방식이 사용될 수 있다. 하지만 이러한 단일 표준 인터페이스(single standard interface)에 정적으로 브릿지하는 방식은 새로운 기능이나 장치가 추가될 때마다 이들을 동적으로 반영하기에는 한계가 있다[15].

또한 새로운 장치들이 추가될 때 마다 표준을 정의하고, 표준에 의거하여 새로운 서비스를 개발하는 것은 기존의 많은 사례에서 보듯이 많은 시간이 소요된다. 그 대신 응용 개발자가 하나의 서비스를 개발할 때마다 다른 미들웨어상의 서비스들과 모두 상호 연동될 수 있게 응용을 개발하면 되지만, 이것은 현실적으로 불가능하다. 왜냐하면 응용이 개발된 이후에도 새로운 장치의 출현 등으로 인하여 새로운 인터페이스가 계속 정의될 수

표 1 Jini와 UPnP의 문법

Jini	void PutFile(String file);
UPnP	<action> <name>PutFile</name> <argumentlist> <name>file</name> <relatedStateVariable> newFile </relatedStateVariable> <direction>in</direction> </argumentlist> </action>

있기 때문이다.

4) 홈네트워크 환경에서 이기종 미들웨어 간 연동을 필요로 하는 홈 오토메이션 서비스에 대한 수요는 충분한가?

KISDI(Korea Information Strategy Development Institute)[16]의 조사에 따르면 사용자들의 홈네트워크 사용 형태는 데이타 네트워크, 엔터테인먼트, 홈오토메이션으로 분류될 수 있고, 그림 1)과 같이 홈오토메이션을 원하는 사용자들이 적지 않음을 알 수 있다. 이처럼 홈오토메이션의 잠재적 시장 규모가 매우 큼에도 불구하고, 엔터테인먼트 분야에서는 많은 연구가 진행되고 있는 반면, 홈오토메이션을 지원하기 위한 미들웨어 연구는 거의 이루어지고 있지 않다. 다양한 미들웨어가 공존하는 홈네트워크상에서의 효율적이고 편리한 서비스 제공을 위해서는 먼저 이질적인 홈네트워크 미들웨어 간 상호 연동 기술 개발이 선결되어야 한다. 이러한 상호 연동 기술은 사용자가 원하는 형태로 서비스 시나리오를 변경하는 것을 언제든 가능하게 할 수 있다.

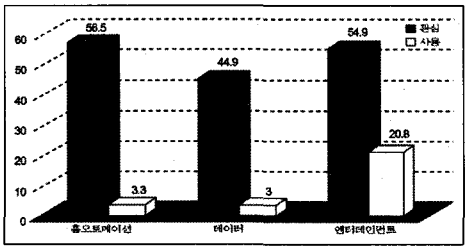


그림 1 홈네트워크 활용 분야 및 관심도

2.3 연구 동기

본 절은 아래와 같이 일반적인 홈네트워크 사용 시나리오 예를 제시하고, 그 시나리오의 분석을 통해 본 연구의 동기와 새로운 통합 미들웨어 구조 설계의 방향을 제시한다.

"A씨의 집은 매일 아침 7시에 알람 시계가 켜지고, 알람 시계가 꺼짐과 동시에 창문이 열리며, 그 다음 커피 포트가 자동으로 물을 끓이기 시작한다. A씨가 샤워를 마치고 나면 대략 7시 30분 정도가 되며, 막 구어진 토스트와 함께 커피를 마신 후, 8시 정도에 출근을 위하여 집을 나선다. A씨가 현관문을 닫은 직후, 창문이 다시 닫히고 보안 서비스가 작동하기 시작한다. 회사에서 근무하는 동안, A씨는 집안에 침입 흔적이 없다는 문자 서비스를 주기적으로 받게 된다."

위의 예를 분석하면, 시나리오가 A씨의 기상시간인 오전 7시를 기준으로 시작됨을 알 수 있다. 7시가 됨과 동시에 알람이 울리며, 알람이 꺼지면 창문이 열린다. 또한 창문이 열림과 동시에 커피포트가 자동으로 켜지게 된다. 위의 시나리오에서 보듯이 홈오토메이션 시나리오가 수행되기 위해서는 특정 시나리오를 수행하기 위한 시작 이벤트가 필요하게 된다. 이러한 시작 이벤트는 시간 또는 다른 어떤 특정한 문맥(context)과 연관될 수도 있다.

이와 같이 홈오토메이션을 위한 홈네트워크 시나리오는 특정 이벤트로부터 시작하여 연속적으로 여러 이벤트가 수행되는 형태를 가진다. 앞서 언급한 시나리오에서 A씨는 자신의 기상 시간을 변경할 수도 있고, 알람이 꺼졌을 때 외부 온도에 따라 창문을 열지 않고 싶어질 수도 있을 것이다. 즉 사용자는 능동적으로 자신의 시나리오를 수정할 수 있어야 하며, 그것은 즉각적으로 홈네트워크 환경에 반영될 수 있어야 한다.

응용 개발자에 의해 개발된 시나리오가 정적인 프로그램 구조를 가지고 있다면, 사용자로부터 시나리오에 대한 새로운 요구사항이 발생할 때 마다 또 다른 새로운 응용이 개발되어야 한다. 이것은 시간이 오래 걸리는 작업이며, 홈네트워크의 효용성과 편리성이 크게 떨어지게 될 것이다.

본 논문에서는 이러한 문제를 해결하기 위한 방안으로서 시나리오 기반의 사용자 중심 통합 미들웨어 구조인 HOMI(HOmenetwork Middleware for Interoperability)를 제안한다. 본 논문은 이 구조를 통해 홈오토메에션에 적합하도록 홈네트워크 서비스 간 연동을 지원하며 사용자의 요구를 즉각적으로 반영할 수 있는 통합 미들웨어 구조를 설계 및 구현한다.

3. HOMI 구조

HOMI는 홈네트워크 서비스의 사용 형태를 고려하여 설계된 시나리오 기반의 사용자 중심 통합 미들웨어 구조이다. HOMI는 2-2절에서 지적되었던 이기종 미들웨어 인터페이스의 차이로 인해 연동 과정에서 발생하는 의미론(semantic)적 문제를 해결하기 위해 최종 사용자가 원하는 연동 시나리오를 간편한 방식으로 쉽게 작성할 수 있도록 HOMIL(HOMI Language)이라는 인터프리터 언어를 제공한다.

HOMI는 다음과 같은 4가지의 주요 모듈로 구성된다.

- Agent Manager
- State Manager
- HOMIL Analyzer
- Context/Event Manager

HOMI는 중앙 집중형 구조에서 발생하는 병목 문제의 해결을 위해 각 Agent를 여러 서버에 분산시킬 수 있다. 또한 모든 프로토콜은 XML을 사용하여 통신하는

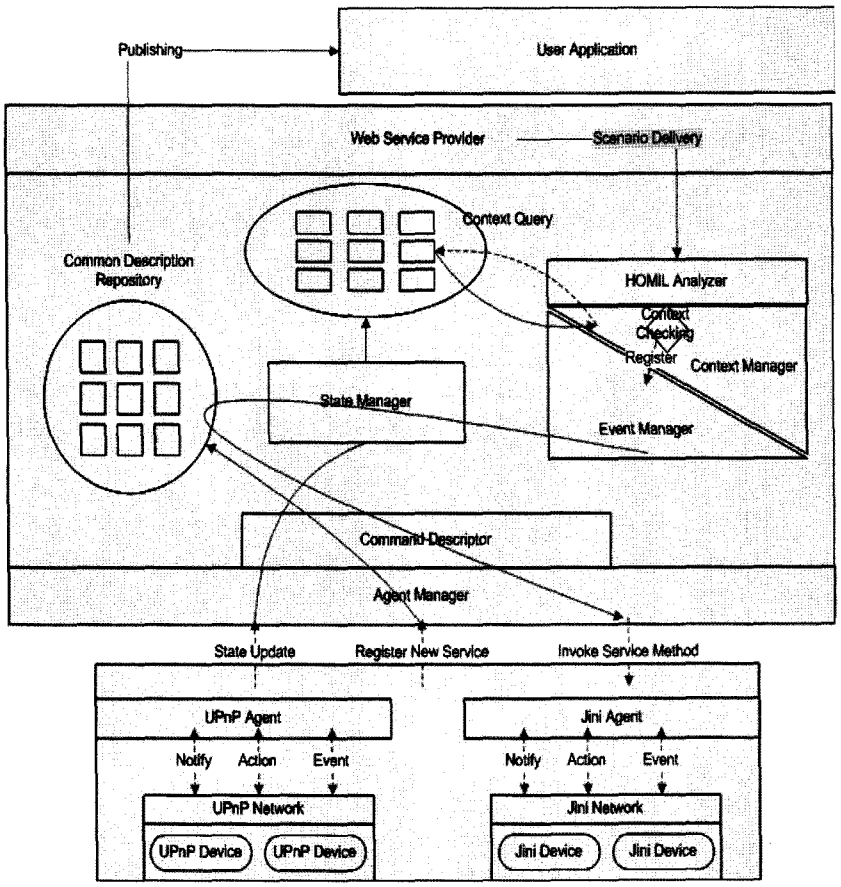


그림 2 HOMI 구조도

개방형 구조를 가지며, 새롭게 정의된 미들웨어에 대한 Agent 개발을 돕기 위하여 C++과 Java기반의 유틸리티 클래스들과 API를 제공한다.

그림 2는 HOMI의 구조도를 보여준다.

3.1 Agent Manager

댁내 가전들의 제어를 위해 사용되는 홈네트워크 미들웨어들은 각기 다른 프로토콜과 수행 메커니즘으로 인해 연동될 수 없다는 것은 이미 언급한 바 있다. 이러한 이질적인 홈네트워크 미들웨어들을 하나로 통합하기 위해서는 서로 다른 미들웨어들을 하나로 추상화 시킬 수 있는 계층이 필요하다.

이를 위해 본 논문에서는 Common Descriptor라는 추상화 계층을 정의하였다. 이 계층은 다양한 미들웨어들 간의 공통 부분만을 포함하기 위하여 댁내 가전에서 필요로 하는 최소한의 요소들만으로 구성된다. 홈네트워크 환경에서 동작하는 가전은 적어도 Service Description, Service Method(Action), Service State 등의 3요소를 가져야 한다. Service Description은 사람이 이해할 수 있는 형태의 서비스 명세이고, Service Method는 서비스가 수행할 수 있는 기능이다. 그리고 가전은 자신의 Service State를 가지고 있어야 하며, 그것을 외부로 알리거나 외부에서 상태를 알 수 있도록 하는 메커니즘을 제공해야 한다. 가전 내부에서든 외부에서든 이러한 구조가 지원 되어야만 다른 가전들과의 연동이 가능하다. 현재 UPnP나 Jini는 모두 이러한 방식을 지원할 수 있는 구조를 가지고 있다.

각 미들웨어 Agent는 네트워크 상에서 서비스를 발견하게 되면, 발견된 서비스들의 이름, 인터페이스 명, 호출 인자명, 그리고 장치의 상태를 분석하여 Common Descriptor의 포맷으로 변환한 후, Agent Manager로 전송한다. Agent Manager는 Common Descriptor 이외에도 Agent들이 지켜야 하는 다른 템플릿들도 정의하고 있다. 이 템플릿들은 모두 XML로 정의되어 있기 때문에 특정 언어나 운영체제에 독립적이다. Agent와 Agent Manager는 TCP/IP를 통해 통신하기 때문에 신뢰성이 보장되며 안정적이다. 또한 네트워크 발전으로

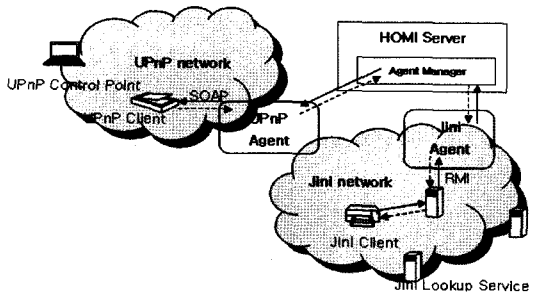


그림 3 HOMI의 분산 Agent 구조

인하여 XML 파싱을 하나의 서버에서 처리하는 것보다 여러 Agent를 통해 처리하는 것이 서버의 부하를 줄이면서 전체적으로 오버헤드를 감소시킬 수 있다. HOMI 구조는 Agent와 Agent Manager를 그림 3과 같이 분리함으로써 중앙 집중형 통합 구조에서 발생하는 부하 집중 문제를 해결할 수 있다.

3.2 State Manager

홈네트워크 내 가전들 간의 연동을 위해 가전들의 상태는 무엇보다 중요한 요소이다. 2.3절에서 소개한 시나리오에서와 같이 특정 장치의 서비스 수행 후 상태가 변경되었을 때, 그것은 다음 시나리오를 시작하기 위한 이벤트로서 역할을 할 수 있다. HOMI에서는 각 미들웨어의 상태 광고 메커니즘[1,2]의 차이로 인하여 발생하는 문제를 해결하기 위해, 각 Agent는 자신이 관리하는 가전들의 상태를 모니터링하고, 그 결과를 HOMI 서버의 State Manager에게 통보한다.

3.2.1 UPnP의 상태 전달 메커니즘

UPnP의 상태 광고는 이벤트를 기반으로 한 Publisher/Subscriber 방식을 취하고 있다. 새로운 서비스를 발견했을 때, UPnP Agent는 발견된 장치의 상태를 모니터링 하기 위해 장치에게 등록(subscription)요청을 하게 되고, 그 결과로 SID(subscription identifier)를 전달받게 된다. 등록 요청이 성공할 경우, UPnP Agent는 등록한 장치의 상태 변화를 통지 받게 되며, 그 결과를 다시 HOMI의 State Manager에게 전달한다.

UPnP Agent의 상태 전달 메커니즘은 다음과 같다(그림 4 참조).

1. UPnP Agent는 모니터링할 장치의 상태 변화를 통보받기(notification) 위해 등록(subscription)을 요청한다.
2. 등록요청이 성공하게 되면, UPnP Agent로 SID가 전달된다.
3. 상태가 변경될 경우, UPnP장치는 자신에게 등록을 요청한 모든 클라이언트들에게 이벤트를 통지한다.
4. 상태의 변화를 통지받은 UPnP Agent는 Agent Manager를 통하여 State Manager에게 변화된 상태값을 통보한다.
5. State Manager는 변화된 상태값을 통하여 상태 변수를 갱신한다.

3.2.2 Jini의 상태 전달 메커니즘

Jini는 상태 전달을 위해 표준으로 정해진 메커니즘이 따로 없다. Jini의 이러한 문제점을 해결하기 위하여 Jini가 지원하는 애트리뷰트(attribute)[2]를 사용하여 장치의 상태 변화가 통지될 수 있게 하였다. 해당 서비스의 애트리뷰트가 변했을 때 발생하는 이벤트를 통해 HOMI의 State Manager에게 장치의 상태 변화를 통지한다.

Jini Agent의 상태 전달 메커니즘은 다음과 같다(그림 5 참조).

1. Jini Agent는 모니터링할 장치의 상태 변화를 통보받기 위해 Jini Lookup Service에게 리스너(Listener)를 등록을 한다.
2. 등록요청이 성공하면, Jini Agent로 serviceChanged Event가 전달된다.

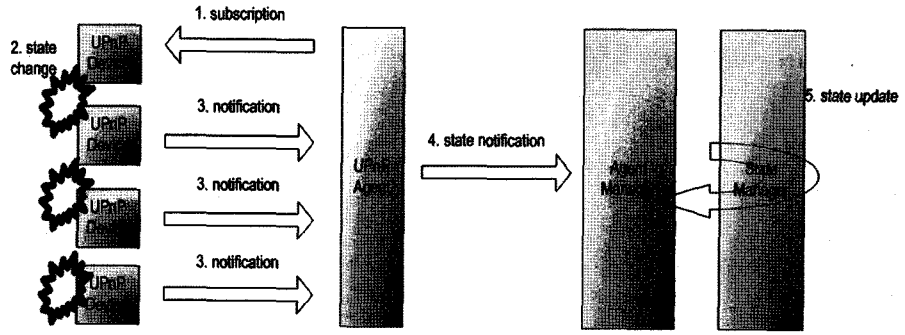


그림 4 UPnP Agent의 상태 전달 메커니즘

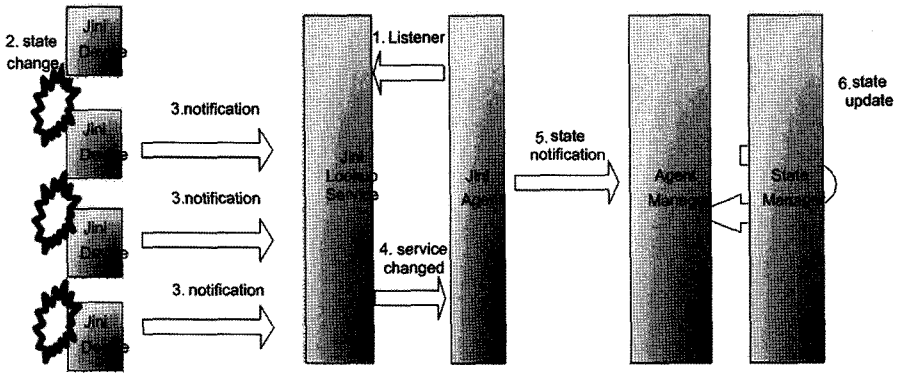


그림 5 Jini Agent의 상태 전달 메커니즘

3. 상태가 변경될 경우, Jini장치는 자신에게 등록을 요청한 모든 클라이언트들에게 이벤트를 통지한다.

4. Jini Lookup Service는 장치변화에 대한 리스너를 등록한 클라이언트에게 서비스가 변경되었음을 알린다.

5. 상태의 변화를 통지받은 Jini Agent는 Agent Manager를 통하여 State Manager에게 변화된 상태값을 통보한다.

6. State Manager는 변화된 상태값을 통하여 상태변수를 갱신한다.

3.3 HOMI Language(HOMIL) Analyzer

유비쿼터스 환경과 달리 홈네트워크 환경에서는 댁내 가전들로 범위가 한정되고, 사용 문맥 또한 유비쿼터스 환경과 같이 다양하지 않기 때문에 홈네트워크 사용 시나리오를 사용자가 직접 구성하는 것은 충분히 가능하다. 이러한 홈네트워크 환경의 특성을 고려하여 사용자가 연동 시나리오를 간편하게 작성할 수 있도록 HOMIL (HOMI Language)이라는 스크립트 방식의 인터프리터 언어를 설계하였으며, 시나리오를 더욱 편리하게 구성할 수 있도록 도와주는 시나리오 빌더가 현재 개발 중에 있다. HOMIL을 이용하면 XML보다 간단한 방법으로 시나리오를 새로 작성하거나 수정할 수 있다. HOMIL은 복잡한 연동 시나리오가 특정 사건을 시작으로 발생하는 일련의 이벤트들의 흐름으로 표현될 수 있도록 하는 이벤트-구동 처리(event-driven processing) 방식의 스크립트 언어이다. HOMIL로 작성된 시나리오는 HOMIL 파서를 통해 XML 형태로 변환되어 HOMI로 전달된다.

표 2 HOMIL 사용예

```
a) execute Aservice.Method1
b) execute Bservice.Method2 when time == 7:00
c) execute Cservice.Method3, Dservice.Method4 if
   Bservice.state1 == Xstate
```

표 2는 HOMIL을 사용한 의사코드이다. a)는 execute 절만 단독으로 사용한 명령문으로서 Aservice의 Method1을 실행하라는 의미이다. b)의 when 조건절은 특정 시간을 의미하며, 조건절의 문장이 참이 될 경우에만 execute절의 문장이 실행된다. c)는 Bservice의 상태가 Xstate가 되면, execute절의 Cservice의 Method2와 Dservice의 Method3가 비동기적으로 실행됨을 의미한다.

3.4 Context/Event Manager

홈오토메이션에서 홈네트워크의 문맥(Context)이란 일련의 연속된 시나리오들의 수행에 영향을 주는 조건이라 정의될 수 있다. 본 논문에서는 문맥을 시간(Time), 동기(Syncronization), 비동기(Asyncronization)로 분류하였다. 예를 들어, "7:00시가 되면 알람이 울린다."와 같은 시나리오는 시간문맥이다.

Context Manager와 Event Manager는 항상 쌍(pair)으로서 동작한다. Context Manager는 시나리오들을 위와 같은 문맥에 맞게 분류하여 Event Manager에게 전달한다. Event Manager는 각 문맥마다 큐를 이용하여 시나리오들을 관리하며, 시나리오의 문맥에 해당되는 조건이 만족되었을 때, 해당 시나리오를 시작(trigger)시키는 역할을 한다. 수행된 시나리오의 결과는 장치의 상태를 변경하게 되며, 변경된 상태로 인하여 State Manager의 상태 변수 테이블이 갱신된다. 이러한 변화는 다음 시나리오를 시작하기 위한 이벤트를 발생시킬 수도 있다.

3.4.1 문맥의 종류

HOMI가 지원할 문맥에는 3가지 종류가 있으며, 이들을 간략히 설명하면 다음과 같다.

1) 시간 문맥

특정 사건이 시간에 의해 시작(trigger)됨을 의미하며, HOMI 내부의 전역(global) 시간 관리 모듈에 의해 관리 및 동작된다. 시간 문맥은 시간 조건절을 이용하여

다음과 같이 HOMIL로써 표현될 수 있다.

• execute Aservice.Method1 when time == 7:00

2) 동기 문맥

동기 문맥은 사건의 시작이 수행 명령문 조건절의 결과가 참이 되었을 경우 동작한다. 동기 문맥은 특정 서비스의 수행 결과가 조건절로 표현되며 다음과 같이 HOMIL로써 표현될 수 있다.

• execute Aservice.Method1
if Bservice.State1 == Xstate and Dservice.State2 == Ystate

3) 비동기 문맥

비동기 문맥은 특정 조건에 의해 수행되는 다양한 서비스들이 순서에 상관없이 병렬 수행됨을 의미하며 다음과 같이 표현될 수 있다.

• execute Aservice.Method1, Bservice.Method2 if Cservice.State1 == Xstate

4. HOMI 구현

4.1 Agent Manager의 구현

1) 이기종 서비스 발견/ 제거

HOMI는 이기종 미들웨어 서비스 발견 또는 제거를 위하여 각 미들웨어별로 Agent를 제공한다. 본 논문에서는 UPnP Agent와 Jini Agent를 C++와 Java를 사용하여 각각 구현하였다. HOMI 서버는 새로운 미들웨어 Agent가 추가될 때 마다 Agent와의 통신을 전담하는 ACT(Agent Communication Thread)를 생성한다. ACT는 네트워크 상에서 전달되는 메시지들을 Agent Manager의 메시지 큐로 전달하며, Agent와 같이 소멸된다. 메시지 큐로 전달된 메시지들은 Agent Manager로 전달되며, 메시지 타입에 따라 해당 루틴이 수행된다. 그림 6은 Agent가 Agent Manager로 메시지를 전달하는 과정을 보여주고 있다.

그림 7은 UPnP네트워크내에 새로운 장치가 설치되었을 때, Agent Manager와 Agent들간의 동작과정을 보여주고 있다. Agent가 새로운 서비스를 발견하게 되면, XML형태의 Common Descriptor를 생성한 다음, ACT를 통하여 Agent Manager에게 전달한다. Agent로부터 새로운 서비스 발견 메시지를 받으면, Agent Manager는 새로운 서비스에 대응하는 Common Descriptor 객체를 생성하고 이를 리스트 형태로 관리한다. 마지막으로, 새로운 서비스를 발견한 Agent에게 UUID를 전송함으로써 새로운 서비스에 대한 Agent와 Agent Manager간의 채널이 생성된다.

2) 이기종 서비스 간의 투명하고 일관된 제어

홈네트워크 환경에 설치되어 있는 모든 이기종 서비스들의 Common Descriptor는 해당 Agent들을 통하여 Agent Manager에게 전달되고, Agent Manager는 이 Common Descriptor를 이용하여 다양한 이기종 미들웨어 서비스들을 하나의 일관된 인터페이스를 통해 제어할 수 있게 된다. 그림 8은 액자 서비스의 사진 색상 제어를 위해 정의된 XML 예를 보여준다.

서비스 실행을 위해 사용자는 Agent Manager가 리

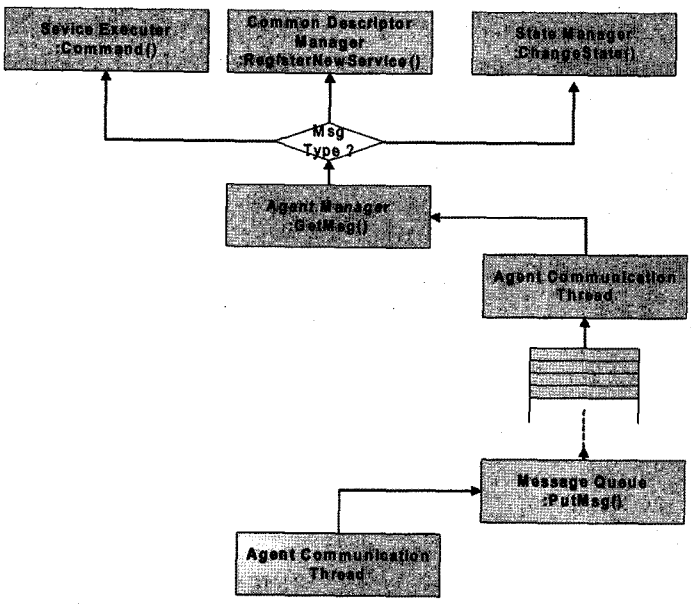


그림 6 Agent로부터 Agent Manager로의 메시지 전달 과정

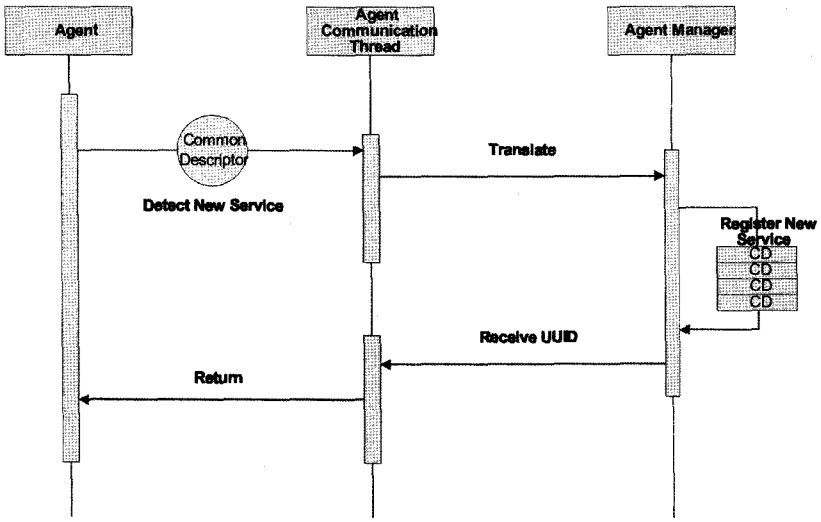


그림 7 새로운 장치가 설치되었을 때의 UPnP Agent 동작 과정

```
<?xml version="1.0"?>
<root xmlns="konan:schemas-homi-org:service">
<serviceuuid>xxxxxx</serviceuuid>
<action>SetColor</action>
    <argumentlist>
        <argument>
            <name>NewColor</name>
            <datatype>int</datatype>
            <value>10</value>
            <direction>in</direction>
        </argument>
        <argument>
            <name>OldColor</name>
            <datatype>int</datatype>
            <value>15</value>
            <direction>out</direction>
        </argument>
    </argumentlist>
</root>
```

그림 8 액자 서비스의 사진 색상 제어를 위한 XML

턴한 UUID를 포함한 메시지를 Agent Manager에게 전달한다. Agent Manager는 요청 메시지를 통해 전달된 UUID를 사용하여 Common Descriptor를 찾고, Common Descriptor내에서 수행할 서비스를 파악한 후, 해당 서비스를 수행하게 된다. 그림 9는 제어명령이 전달되는 과정을 보여준다.

HOMI는 서비스 수행을 추상화 하기 위해 SVCExe (Service Execute)라는 클래스(표 3 참조)를 제공한다.

SVCExe의 멤버변수 m_cdMgr은 제어명령용 XML 문서 생성을 위한 HOMI 라이브러리이다. HOMI는 Agent 개발의 편이성을 제공하기 위해 디버그, 네트워크, XML 파서 등의 라이브러리들을 제공하고 있다. m_uuid는 서비스가 처음으로 HOMI 서버에 등록될 때

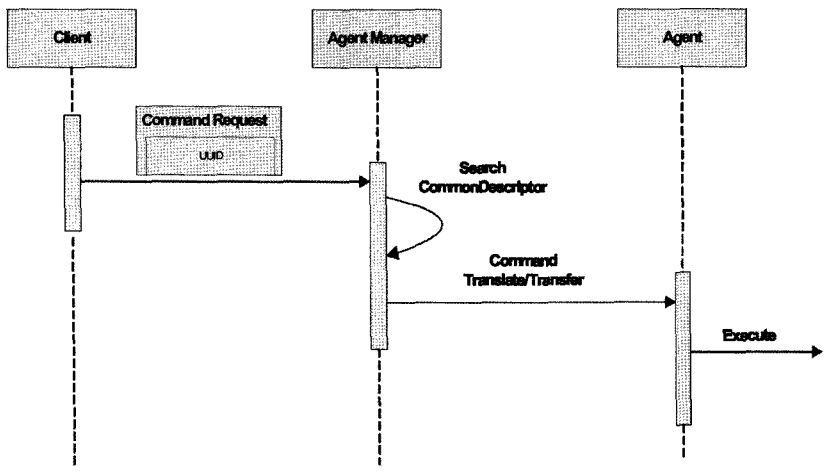


그림 9 제어 명령 전달 과정

표 3 SVCExe 클래스의 객체 구조

SVCExe
-m_cdMgr : CDManager -m_uuid : DWORD -m_pCM : CommandMaker * -m_pMsgHdr : Message * -m_svcName : string
+SVCExe(in uuid : DWORD, in pSvcName : CHAR*) +~SVCExe() +SetAction(in pActionName : BYTE*, in retType : CD_TYPE) +SetArg(in pArgName : BYTE*, in type : CD_TYPE, in pValue : CHAR*, in direction : ARG_DIRECTION) +SetArg(in pArgName : BYTE*, in type : CD_TYPE, in value : INT, in direction : ARG_DIRECTION) +Execute() +GetSvcName() : string

Agent Manager에 의해 생성된 UUID의 값을 가진다. m_commandMaker는 소켓을 통하여 전송할 제어명령을 생성하기 위해 HOMI가 제공하는 라이브러리이다. 마지막으로 m_svcName은 수행될 서비스의 이름을 나타낸다.

3) 이기종 장치 간의 투명한 상태 전달

Agent Manager는 장치들의 상태 변화를 전달 받아 State Manager(3.2절 참조)에게 전달한다. 홈네트워크 미들웨어마다 상태 전달 메커니즘의 차이로 인한 연동의 문제는 각 미들웨어의 Agent를 그림 10과 같은 일관된 XML규약으로 변경한 후, Agent Manager에게 전송함으로써 해결될 수 있다. 이에 따라 State Manager는 미들웨어의 이질성에도 불구하고 항상 일관된 인터페이스를 통해 장치들의 상태를 관리할 수 있다.

Agent Manager는 ACT를 통해 전송된 데이타를 파싱하고 분석한다. Agent Manager는 상태 메시지를 소유하고 있는 Common Descriptor를 찾아 유효성을 검사한 후, 해당 상태 정보를 갱신한다. 이러한 가전의 상태 정보는 연속된 시나리오의 수행에도 이용된다. 따라서 상태정보의 갱신은 단순히 값만을 변경하는 것이 아니라 사용자가 연동하길 원하는 장치의 상태와 현재 갱신된 장치의 상태가 일치하게 되면 다음 서비스를 수행하게 될 이벤트로서도 동작하게 된다.

```
<?xml version="1.0"?>
<root xmlns="konan:schemas-homi-org:service">
<serviceuuid>988021095</serviceuuid>
<statelist>
    <state>
        <name>state</name>
        <datatype>int</datatype>
        <statevalue>1</statevalue>
    </state>
</statelist>
</root>
```

그림 10 상태변화를 알리기 위한 상태 전달 XML

4.2 State Manager의 구현

State Manager는 시스템에 전역적으로 존재하는 객체이며, 각 서비스 별로 생성된 상태 테이블(표 6 참조)을 통해 관리된다. State Manager의 객체 구조(StateMgr)는 아래 표 4와 같이 구성되며, 서비스의 상태 테이블을 찾을 수 있는 키로서 서비스 이름을 이용하여 관리된다(표 5 참조). UpdateList는 시나리오 연동을 위해 정의된 자료 구조로서 서비스의 상태 변수와 이벤트를 시작시키기 위한 시그널 등 2개의 필드로 구성된다. 홈네트워크 상의 디바이스 중에서 상태가 변한 것이 있다면, 상태 변화 메시지가 해당 디바이스의 Agent로부터 Agent Manager에게 전송된다. 상태 변수는 UpdateList에 등록된 변수로서 사용자가 원하는 값과 일치하면, State Manager는 시그널을 생성하여 해당 Event Manager로 통보한다.

State Manager에 정의된 주요 메소드는 다음과 같다.

표 5 서버의 상태 테이블(UpdateList)

서비스 이름	상태 테이블
TV	TV_table
Bulb	Bulb_table

표 6 서비스의 상태 테이블(StateTable)

상태변수	상태 값
채널	23
음량	12

표 4 State Manager의 객체 구조

StateMgr
-m_pThis : StateMgr * = NULL -m_tablePerSvc : TABLE_PER_SERVICE -m_updateList : UpdateList -m_stateTblMutex : pthread_mutex_t
-StateMgr() -~StateMgr() +GetInstance() : StateMgr * +RegisterStateTable(in svcName : string) : STATE_TABLE_HANDLE +RegisterStateVariable(in stateTbl : STATE_TABLE_HANDLE, in varName : string, in value : string, in varType : string) +GetStateSize() : INT +GetStateName(in index : INT) : string +GetStateTbl(in index : string) : StateTable * +SetEvent(in pCond : pthread_cond_t*, in svcName : string, in variableName : string) +UpdateVariable(in svcName : string, in variableName : string, in result : string) +GetValue(in svcName : string, in stateName : string) : string

- RegisterStateTable() : 서비스마다 다양한 상태가 존재하기 때문에 서비스별로 상태 테이블을 생성하여 등록한 후 상태 테이블 핸들을 반환한다.
- RegisterStateVariable() : 상태 테이블 핸들을 인자로 하여 상태 변수의 이름과 값의 쌍을 상태 테이블에 등록한다.
- GetStateSize() : 현재 등록되어 있는 상태 테이블의 개수를 반환한다.
- GetStateName() : 상태 테이블의 이름을 반환한다.
- GetStateTbl() : 서비스 이름을 인자로 하여 상태 테이블을 찾아 이를 반환한다.
- UpdateVariable() : 상태 변수 값을 갱신하며, 이때 이 값이 시그널을 보내기 위한 조건을 만족하면, Event Manager에게 시그널을 전달 한다.
- SetEvent() : 상태 변수가 특정 값과 같을 때 실행할 이벤트를 등록한다.
- GetValue() : 상태 테이블과 상태변수를 인자로 하여 상태 값을 찾는다.

4.3 Context/Event Manager의 구현

Context 클래스는 TimeContext, SyncContext, ASync-Contex를 추상화하기 위한 추상 클래스로서 다음과 같이 Handle이란 추상 메소드만을 포함하고 있다(그림 11 참조).

서브 클래스인 TimeContext, AsyncContext, Sync-Context는 부모 클래스 Context의 Handle 메소드를 구현하고 있으며, Handle 메소드는 각각 자신의 문맥에 맞도록 처리 과정들이 구현되어 있다.

TimeContext는 HOMI의 Utillity 클래스로서 시간정보들을 관리하는 TimerService를 이용하여 자신이 이벤트를 받아야 할 시간을 등록한다. 표 7은 TimeContext의 이벤트 등록 및 처리를 위한 Handle 메소드 의사코드이다.

표 7 TimeContext의 Handle 메소드 의사코드

```
BOOL TimeContext::Handle()
{
    If (!TimerService.SetEvent(&m_timeCond, m_time))
            return TRUE;
            Signal(&m_timeCond, &m_timeMutex);
            return TRUE;
}
```

이벤트가 호출될 시간이 TimerService에 등록되며, 이벤트가 발생하기를 대기하게 된다. 명시한 시간이 되어 이벤트가 발생될 경우, 함수는 대기 상태에서 깨어나 리턴하게 된다.

SyncContext는 특정 서비스의 상태 변화를 알기 위하여 State Manager에게 상태의 등록을 요청한다. SyncContext는 장치의 상태가 변화하였을 경우, 변화된 값과 사용자의 시나리오에서 명시된 값을 비교하여 그 결과에 따라 AsyncContext의 서비스를 실행시키기 위한 시그널을 전달한다. 표 8은 SyncContext의 이벤트 등록 및 처리를 위한 Handle 메소드 의사코드이다. SetResult()는 장치의 변화된 상태값과 사용자의 시나리오에 명시된 값을 사용자가 명시한 비교 연산자를 통하여 비교하는 메소드이다.

AsyncContext는 수행될 서비스들의 리스트를 가지고 있다. TimeContext와 SyncContext를 통하여 모든 조

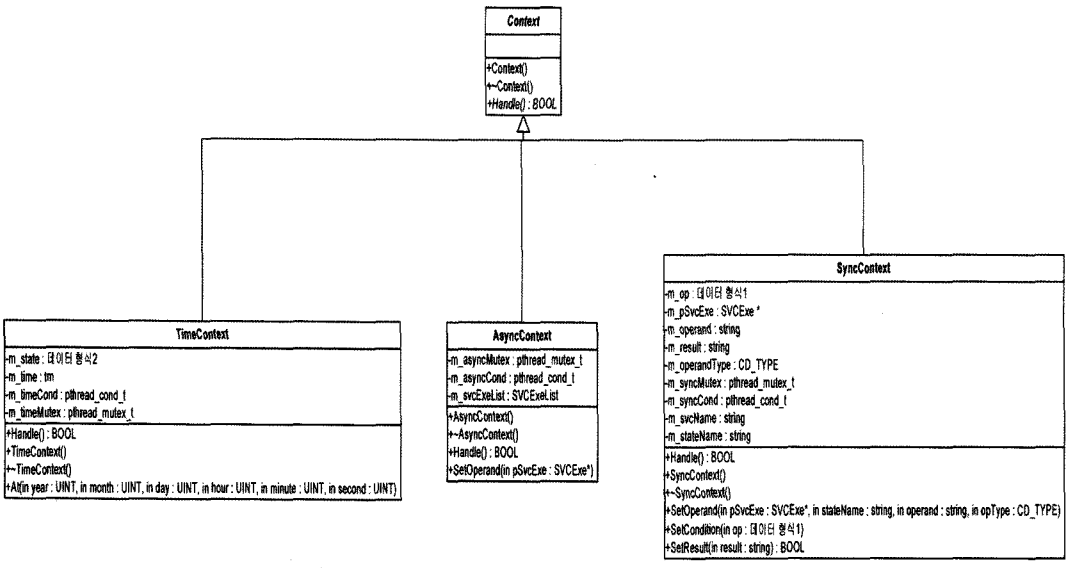


그림 11 Context 클래스와 서브 클래스의 객체 구조

건들이 만족됨을 확인한 후, 표 9와 같이 등록되어 있는 모든 서비스 리스트들을 순차적으로 실행한다. 표 9는 AsyncContext의 이벤트 등록 및 처리를 위한 Handle 메소드 의사코드이다.

표 8 SyncContext의 Handle 메소드 의사코드

```
BOOL SyncContext::Handle()
{
    StateMgr.SetEvent(&m_syncCond, m_svcName,
    m_stateName);
    Signal(&m_syncCond, &m_syncMutex);
    string result = StateMgr.GetValue(m_svcName,
    m_stateName);
    return SetResult(result);
}
```

표 9 AsyncContext의 Handle 메소드 의사코드

```
BOOL AsyncContext::Handle()
{
    INT size = m_svcExeList.size();
    for ( INT i = 0; i < size; i++)
    {
        m_svcExeList[i]->Execute();
    }
    return TRUE;
}
```

표 10 Event Manager의 객체 구조

EventMgr
-m_scenarioList : ScenarioList -m_eventMutex : pthread_mutex_t -m_eventCond : pthread_cond_t -m_eventThread : pthread_t
+EventMgr() +~EventMgr() +RegisterScenario(in pScenario : Scenario*) +_EventHandler(in pArg : void*) : void_* +EventHandling()

표 11 EventManager의 EventHandling 함수의 의사코드

```
void EventMgr::EventHandling()
{
    BOOL isNewScenario;

    while(TRUE)
    {
        Scenario *pScenario;
        isNewScenario = m_scenarioList.GetScenario(&pScenario);
        if ( isNewScenario == FALSE)
        {
            Signal(&m_eventCond, m_eventMutex);
            continue;
        }
        pScenario->Handle();
    }
}
```

마지막으로 Event Manager는 등록되어 있는 시나리오들을 시나리오의 순서대로 큐에 저장하고 읽어오는 역할을 하며 해당 Context 시나리오들의 Handle 메소드를 호출한다. 시나리오 큐에 더 이상의 시나리오가 없으면, Event Manager는 블록되고, 새로운 시나리오의 등록을 기다리게 된다. 표 10과 표 11은 각각 Event Manager의 객체 구조와 Event 처리 함수의 의사코드를 보여준다.

5. 시나리오 테스트 및 평가

5.1 시나리오 테스트

본 논문에서 제안한 시나리오 기반의 사용자 중심 홈 네트워크 미들웨어 통합구조인 HOMI가 사용자들의 요구를 반영하여 시나리오들을 쉽게 변경할 수 있으며, 홈 오토메이션 지원을 위하여 이기종 서비스들간의 연동이 가능함을 보이기 위해 그림 12와 같은 테스트베드를 구축하였다.

먼저 이기종 미들웨어 서비스간 상호연동성을 보이기 위하여 Jini장치로서 원격에서 시간을 셋팅하고 시계의 전원을 on-off 할 수 있는 Jini-Clock과 원격에서 전구를 on-off할 수 있는 Jini-Light를 시뮬레이션 하는 응용들을 개발하였다. UPnP 장치로서는 PXA-255기반의 ARM보드상에 UPnP액자를 개발하고, UPnP 전구는 Intel에서 제공하는 UPnP Network light[17]를 사용하였다.

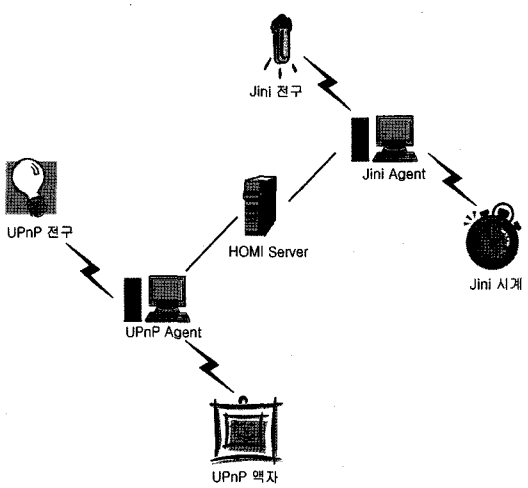


그림 12 테스트를 위한 시스템 구성도

테스트를 위해 시스템은 모두 이더넷을 통하여 연결되었으며 IP위에서 동작한다. 테스트에 사용된 시나리오는 다음과 같다.

"Jini시계가 오전 8:00:00시를 가리키면 UPnP전구와 Jini전구가 켜진다. UPnP전구가 켜지면 UPnP액자에는 기존 사진 대신 저장되어 있는 임의의 새로운 사진이 주기적으로 교체된다."

위의 시나리오를 HOMIL로써 표현하면 표 12와 같다.

표 12 HOMIL로써 표현된 시나리오

```
Execute Light.SetTarget(TRUE), JiniLight.PowerOn()
when time == 8:00
Execute UPnPAngle.AutoOn if Light.state == TRUE
```

그림 13은 위의 시나리오를 XML로 변환환 결과이다.

```
<?xml version="1.0"?>
<root xmlns="konan:schemas-homi-org:service">
<statementlist>
  <statement>
    <asynccontext>
      <service>
        <name>Light</name>
        <uuid>3081</uuid>
        <action>SetTarget</action>
        <returntype />
        <argumentlist>
            <argumenttype>bool</argumenttype>
            <argumentvalue>true</argumentvalue>
        </argumentlist>
        <stateVarName>target</stateVarName>
      </service>
      <service>
        <name>JiniLight</name>
        <uuid>3082</uuid>
        <action>PowerOn</action>
        <returntype>void</returntype>
        <argumentlist />
        <stateVarName>status</stateVarName>
      </service>
    </asynccontext>
    <timecontext>
      <hour>8</hour>
    </timecontext>
  </statement>
  <statement>
    <asynccontext>
      <service>
        <name>UPnPAngle</name>
        <uuid>3083</uuid>
        <action>AutoOn</action>
        <returntype />
        <argumentlist />
        <stateVarName>IsAuto</stateVarName>
      </service>
    </asynccontext>
    <synccontext>
      <service>
        <name>Light</name>
        <uuid>3081</uuid>
        <action>GetStatus</action>
        <returntype />
        <argumentlist />
        <stateVarName />
      </service>
      <operator>equal</operator>
      <operand>
        <type>string</type>
        <value>true</value>
      </operand>
    </synccontext>
  </statement>
<statementlist>
</root>
```

그림 13 XML로 변환된 테스트 시나리오

테스트 시나리오 수행 과정은 다음과 같다(그림 14 참조).

1. 시나리오가 입력되면 시나리오를 분석한다.
2. 분석된 시나리오의 이벤트를 등록한다.
3. 8시가 되면 등록된 이벤트(UPnP 전구 On, Jini 전구 On)를 실행한다.
4. State Manager 로부터 서비스 실행 명령이 전달되면, Common Description Repository에서 서비스를 찾아 해당 Agent 에게 실행을 요청한다.
5. Agent는 Agent Manager로부터 전달 받은 명령을 파싱하여 Agent 에 등록되어 있는 디바이스에게 실행을 요청한다.
6. 디바이스는 실행된 결과로 Agent에게 자신의 상태(UPnP 전구 On, Jini전구 On) 변화를 알린다.
7. Agent Manager는 Agent로부터 온 상태 변화를 State Manager 에게 전구의 상태가 켜짐으로 변화 되었음을 알린다.

UPnP 전구의 상태가 On으로 변함으로써 다음 이벤트인 UPnP 액자의 사진 자동변환 기능을 실행시킨다. 과정은 동일하다.

실험 결과 작성된 시나리오에 따라 이기종 장치들이 상호 작용을 하며 연속적으로 수행됨을 확인할 수 있었다. 또한 웹 서비스를 통하여 사용자가 각 장치들을 원활하게 제어할 수 있음을 확인할 수 있었다.

5.2 평가

본 논문에서는 홈오토메이션의 효율적인 지원을 위하여 홈네트워크 상에서 이질적인 미들웨어 간의 상호 연동에 적합한 통합 구조를 제안하고 구현 하였다. 그 동안 제안된 이질적인 미들웨어 간의 상호 연동을 지원하는 미들웨어들의 특징을 비교하면 표 13과 같다. 본 연구에서 제안한 HOMI, ETRI의 UMB, 와세다의 PCM 등 미들웨어 통합을 위한 접근 방식은 대부분 meta-protocol에 기반 하여 일대다 연결을 통한 상호 연동을 지원 하고 있다. HOMI는 댁내 가전이 각 미들웨어에서 공통적으로 필요로 하는 최소한의 요소들만으로 구성된 추상화 계층을 제공함으로써 일관된 방식으로 가전기기를 표현 하도록 지원한다. 와세다 대학의 PCM처럼 초기 설정 시에 모든 것들이 정적으로 연결되는 것이 아니라 가전기기가 켜짐과 동시에 동적으로 연결되고 꺼지게 되면 바로 연결이 끊어지는 구조를 가지고 있다. 또한 기존의 ETRI나 와세다의 연구와는 달리 본 논문의 홈네트워크 미들웨어 통합 서버는 표준으로 정해지지 않은 디바이스도 시나리오를 통해 지원할 수 있다. 디바이스 간의 직접 데이타 전송을 위해서는 장치들 간에 표준이 정해지고 이를 통해 서로간의 서비스 이용 방법을 알고 있어야 하기 때문에 표준으로 정해지지 않

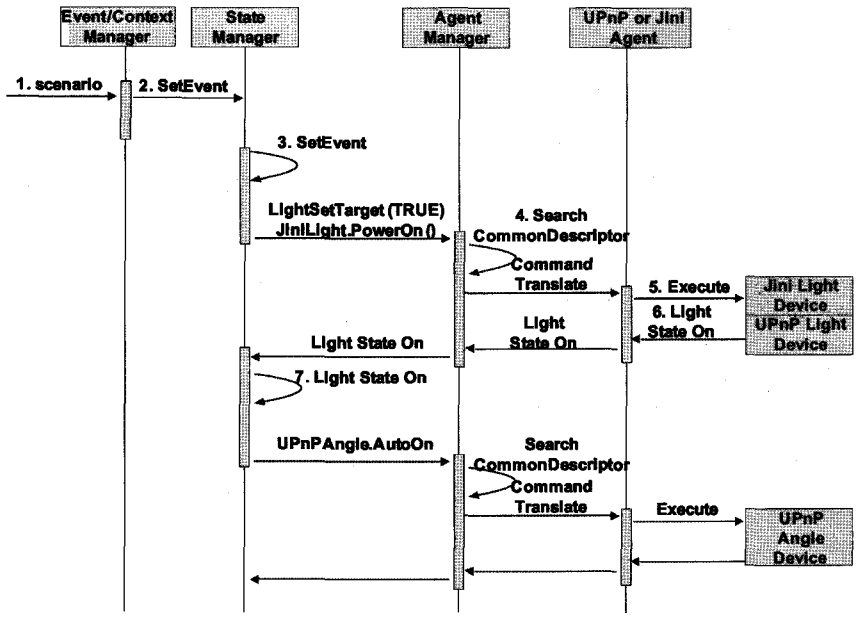


그림 14 XML로 변환된 테스트 시나리오

은 모든 디바이스를 지원하는 HOMI에서는 서버를 통한 데이타 이동은 가능하나 디바이스간의 직접 데이타 이동은 현재 불가능하다. 마지막으로, 기존 방식처럼 사용자로부터 홈오토메이션 서비스에 대한 새로운 요구사항이 발생할 때 마다 또 다른 응용이 개발되어야 하는 것이 아니라, 사용자가 직접 자신이 원하는 홈오토메이션 시나리오를 HOMIL을 통해 작성할 수 있다.

6. 결론 및 향후 연구

홈네트워크의 효율적 활용을 위해서는 미들웨어의 이질성으로 인하여 발생되는 문제가 반드시 해결되어야 한다. 이와 함께 이기종 미들웨어간의 상호 연동을 통해 사용자의 필요에 따라 다양한 홈오토메이션 서비스들이 재구성될 수 있도록 지원하는 것 또한 중요하다.

본 논문에서는 홈네트워크 사용자들의 다양한 홈오토메이션 서비스 요구를 만족시킬 수 있도록 시나리오 기반의 사용자 중심 통합 미들웨어를 설계하고 구현하였다. HOMI는 기존의 통합 미들웨어 구조와는 다르게 사용자들이 직접 시나리오를 작성하고 구성함으로써 홈오토메이션을 위한 이기종 가전들의 연동성을 향상시켰다. HOMI는 이기종 가전들 간의 연동을 위하여 사용자들의 연동 서비스를 시간 문맥, 동기 문맥, 비동기 문맥 등 3가지로 분류하였고, 특정 이벤트가 발생하였을 때 문맥을 고려하여 다음 서비스가 수행되도록 지원하는 구조이다. 이러한 구조는 사용자들에게 자신이 원하는 대로 서비스를 재구성할 수 있는 편리하고 효율적인 환경을 제공한다. 또한 HOMI는 변경된 시나리오가 댁내의 홈네트워크 환경에 즉각적으로 반영됨으로써 사용자들이 새로운 홈네트워크 사용 시나리오를 적용하기 위하여 새로운 응용을 설치하거나 서버를 갱신(update)하고 재부팅하는 과정 없이 계속적으로 서비스들을 받을 수 있도록 지원한다. 마지막으로HOMI는 통합 미들웨어의 중앙 집중형 구조에서 발생하는 부하 문제를 Agent들을 분산시킬 수 있도록 지원함으로써 해결하였다.

표 13 통합 미들웨어 비교

	HOMI	ETRI(UMB)	와세다(PCM)
연결 방법	일대다	일대다	일대다
연결 비용	n-1	n-1	n-1
가전기기 추상화	O	O	X
연결	실행 시에 동적인 연결 가능	실행 시에 동적인 연결 가능	초기 설정 시에 정적으로 연결함
지원 서비스 형태	표준과 상관없이 지원 가능	표준 서비스 지원	표준 서비스 지원
이기종 디바이스간 직접 데이타 전송	X	O	O
사용자 기반 홈 오토메이션 지원	O	X	X

본 연구는 앞으로 더욱 다양한 환경 하에서 홈오토메이션 서비스가 수행될 수 있도록 문맥의 종류를 확장할 필요가 있으며, 확장된 문맥 지원을 위해 HOMIL의 개선도 필요할 것이다. 홈네트워크 환경에서 다양한 시나리오들이 필요할 때마다 생성되어 동작하게 되면 시나리오 간의 충돌이 발생할 수 있다. 이를 효율적으로 해결하기위해 사용자의 서비스 이용 패턴을 기반으로하여 서비스 간의 충돌, 서비스 이용시 발생한 오류 등의 관리에 관한 연구도 필요할 것이다. 또한 이기종 홈네트워크 미들웨어를 통합하기 위하여 중앙 집중형 방식을 사용함으로써 HOMI 서버의 결함 감내가 중요 이슈가 될 수 있다. 이의 해결을 위해 홈 네트워크상의 고사양 가전들을 인지하고 중요한 서비스들을 이러한 가전들로 분산시킴으로써 HOMI 서버가 동작하지 않을 시에도 중요한 서비스들은 계속해서 동작할 수 있는 결함 감내 시스템이 향후 연구되어야 할 것이다. 마지막으로 사용자가 시나리오를 편리하게 작성할 수 있도록 지원하는 시나리오 빌더와 응용 개발자들을 위한 새로운 미들웨어용 Agent의 신속한 개발을 지원하는 새로운 API의 추가 정의가 필요하다.

참 고 문 헌

[1] UPnP Forum. http://www.upnp.org.

[2] Sun Microsystems. Jini Architecture Specification. http://www.sun.com/jini/.

[3] The Havi Organization, Havi Version 1.1 Specification. http://www.havi.org.

[4] Echelon Co., LonTalk Protocol Specification, Ver 3.0, 1994.

[5] E. Guttman, C. Perkins, J. Veizades and M. Day, Service Location Protocol, Ver 2, 1999.

[6] B Rose, "Home Networks: A Standards Perspective," IEEE Communications Magazine, pp. 78-85, Vol. 39, December 2001.

[7] G. O'Driscoll, The Essential Guide to Home Networking Technologies, Prentice-Hall, 2001.

[8] S. Huhns, Service-Oriented Computing, WILEY, 2005.

[9] B. Guillaume, R. Kumar, B. Helmut, and S. Thomas, "Methods for Bridging a HAVi Subnetwork and a UPnP Subnetwork and Device for Implementing said Methods," Thomson Multimedia, 2002.

[10] J. Allard, V. Chinta, S. Gundala, G. Richard III, "Jini Meets UPnP: An Architecture for Jini/UPnP Interoperability," Symposium on Applications and the Internet, pp. 268-275, January 2003.

[11] D. Box, "Simple Object Access Protocol 1.1" available at URL http://www.w3.org/TR/SOAP/.

[12] E. Tokunaga, H. Ishikawa, M. Kurahashi, Y. Morimoto, and T. Nakajima, "A Framework for Connecting Home Computing Middleware," ICDCSW, pp.765-770, July 2002.

[13] OSGI Alliance. http://www.osgi.org/.

[14] K. Moon, Y. Lee, Y. Son, and C. Kim, "Universal Home Network Middleware Guaranteeing Seamless Interoperability among the Heterogeneous Home Network Middleware," IEEE Transactions on Consumer Electronics, Vol. 49, August 2003.

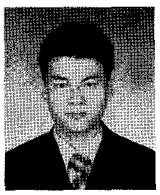
[15] A. R. Ponnekanti and A. Fox. "Application Service Interoperation without Standardized Service Interfaces," Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, pp. 30-40, March 2003.

[16] KISDI, 홈네트워킹 시장 분석 및 발전 전망, 2003.12.

[17] Intel® Software for UPnP Technology. http://www.intel.com.

김 민 찬

2004년 국민대학교 자동차공학과(공학사). 2004년~현재 중앙대학교 컴퓨터공학과 석사과정. 관심분야는 유비쿼터스컴퓨팅, 임베디드 시스템, RTOS



이 학 진

2005년 중앙대학교 컴퓨터공학과(공학사). 2005년~현재 중앙대학교 컴퓨터공학과 석사과정. 관심분야는 유비쿼터스컴퓨팅, 임베디드 시스템, RTOS

김 성 조

1975년 서울대학교 응용수학과(공학사) 1977년 한국과학기술원 전산과(이학석사) 1977년~1980년 ADD(연구원). 1980년~현재 중앙대학교 컴퓨터공학부(교수). 1987년 Univ. of Texas at Austin(공학박사) 1887년~1988년 Univ. of Texas at Austin (Research Fellow). 1996년~1997년 Univ. of California-Irvine (Visiting Professor). 관심분야는 이동컴퓨팅, 임베디드 소프트웨어, 유비쿼터스컴퓨팅