유비쿼터스 환경에서 임베디드 디바이스 서비스 통합을 위한 웹서비스 변환 게이트웨이

신성환° 이정금 이용환 민덕기
건국대학교
omayaro@hotmail.com°, {stutle, yhlee, dkmin}@konkuk.ac.kr

WebService Conversion Gateway for Integrating Embedded Device Service in Ubiquitous Environment

SoungHwan Shin°, JeongGeum Lee, YoungHwan Lee, Dugki Min
Konkuk University

요 약

웹서비스는 분산 개별 서비스들을 통합하기 위한 표준기술로서 최근에는 홈 네트워킹, 임베디드 환경 등 다양한 분야에서 핵심 연동기술로 활용범위가 확산되고 있다. 본 논문에서는 유비쿼터스 환경에서 상호 이질적인 플랫폼이나 디바이스를 기반으로 제공되는 서비스들 간 통합을 용이하게 할 수 있는 웹서비스 변환 게이트웨어 프레임 워크를 제안한다. 본 논문에서 제안한 변환 게이트웨이는 유비쿼터스 환경에서 다양한 플랫폼이나 프로토콜들에 유연하게 적응할 수 있도록 설계되었으며 또한 임베디드 환경에서 요구되는 경량화를 지원할 수 있는 구조를 가지고 있다.

1. 서 론

유비쿼터스 환경이 확산됨에 따라서 분산된 서비스들의 연계 및 통합의 중요성이 커지고 있다. 웹 서비스는 이러한 분산된 서비스들을 연계하고 통합하는 표준기술로서 광대역 통합망(BcN: Broadband convergence Network) 기반의 유무선 통합 응용, 방송/통신 융합, 정보 가전/홈 네트워킹, 임베디드 환경 등 다양한 분야에서 핵심 연동 기술로 그 활용 범위가 빠르게 확산되고 있다. 최근 홈 네트워킹이 발전되면서 생활 주변의 가전기기는 기존의 기기에서 제공하는 기능뿐만 아니라 추가적인 향상된 기능을 요구하고 있으며 이를 위해 상호 이질적인 플랫폼이나 디바이스들에서 제공되는 서비스들 사이의 통합이 요구되어지고 있다. 하지만 이러한 상호이질적인 서비스들 간 통합을 위해서는 무엇보다도 표준 통신 프로토콜, 표준 프로그래밍 인터페이스, 그리고 표준 레퍼런스, 저장소와 같은 표준들을 제공함으로써 상호 운영성이나 연계 통합을 용이하게 할 수 있어야 한다. 최근에 많은 기업이나 연구자들은 인터넷을 통한 상호 이질적이 서비스나 어플리케이션 통합을 용이하게 하기 위해 위의 3가지 표준 문제를 만족시키는 웹서비스 기술을 사용하고 있다.

유비쿼터스 환경에서 웹서비스[11] 기반으로 상호 이질적인 디바이스간 통합을 지원하는 변환 게이트웨이는 우선 기존 디바이스에서 제공되던 서비스들과의 연동이 용이해야 하며 추가적인 새로운 기능을 동적으로 제공할 수 있는 유연한 확장 가능한 구조여야 한다. 두 번째로 변환 게이트웨이는 임베디드 환경에서 수행할 수 있기 때문에 경량화 할 수 있는 구조여야 하며 이를 지원할 수 있어야 한다. 세 번째로 분산 환경에서 흩어져 있는 다양한 서비스들에 대한 통합 프레임워크에는 관련된 리소스에 대한 모니터링뿐만 아니라 관리를 용이하게 할 수 있어야 한다.

본 논문에서는 임베디드를 이용한 유비쿼터스 환경을 구축하기 위하여 이질적인 플랫폼이나 프로토콜을 사용하는 서비스들에 대한 통합을 용이하게 지원하는 웹서비스 기반 변환 게이트웨이 프레임워크를 제안한다. 본 논문에서 제안한 변환 게이트웨이는 임베디드 환경에 맞추어 경량화 되었고 동적으로 다양한 프로토콜을 웹서비스로 제공하기 위한 구조를 가지며 다양한 서비스를 추가적으로 제공할 수 있고 이런 서비스를 분산시킬 수 있는 구조를 가진다. 또한 변환 게이트웨이에 대한 관리뿐만 아니라 환경 설정을 용이하게 할 수 있다.

2. 관련 연구

본 논문에서 제시하고자 하는 WSCG는 유비쿼터스 환경에서 다양한 디바이스에 존재하는 기존의 서비스들을 동일한 서비스 형태로 간주함으로서 상호운용성의 문제를 해결하고자 노력한 것이며 이는 SOA(Service Oriented Architecture)의 사상에 그 근간을 두고 있다[1]. SOA는 상호작용하는 소프트웨어들 간의 느슨한 연결이 되게 하여 소프트웨어 하나하나가 자신의 역할에만 충실하게 하는 아키텍처 스타일을 말한다. 느슨한 연결이 된다면 상호작용하는 소프트웨어들 간의 새로운 소프트웨어의 추가 또는 기존 소프트웨어의 삭제가 용이해지게 된다. SOA는 웹서비스를 이용하여 모든 어플리케이션이 서로를 이용할 수 있도록 하여 커다란 하나의 서비스를 만들 수 있도록 한다. 이런 SOA는 서비스의 관점에서 소프트웨어 아키텍처를 지향하는 기술로 최근 많은 각광을 받고 있으며 시장조사 업체인 가트너 그룹은 2006년까지 전 세계 비즈니스 어플리케이션의 80% 이상이 SOA를 기반으로 개발될 것이라고 전망하고 있다.

이러한 SOA를 따르는 가장 대표적인 미들웨어로서 ESB(Enterprise Service Bus)[3]를 들 수 있다. ESB는 통합 프로젝트의 기술과 경제성을 근본적으로 변화시키기 위하여 SOA의 원리와 Web Service가 가진 잠재 능력을 이용한다. ESB는

B2B 내의 서비스, 어플리케이션, 자원들을 연결 및 통합하는 기능을 수행한다. 이러한 ESB와 관련된 중요한 개념들에는 이질성 극복, 소프트웨어 컴포넌트 사이의 서비스 지향적, 이벤트 지향적인 분야에서 QoS의 보장, 상호운용성 그리고 타 벤더와의 통합을 위한 개방 API의 용이한 대체 등이 있다.

현재 서비스들 확장 추세는 기존에 기업의 비즈니스 서비스에서 디바이스의 서비스로 그 범위가 확장되어 가고 있다. 특히 디바이스 성능의 향상 및 홈 네트워크의 보급은 서비스의 기능적 확장으로 이어지면서 서비스의 임베디드화를 촉진시키고 있다. 이러한 서비스의 임베디드화에 대한 대표적인 사례로 DP4WS[12]가 있다. DP4WS는 기존의 UPnP 1.0 스펙[7]을 참조하여 이를 웹서비스 기반의 UPnP 모델로서 구현하고 있는데 이는 UPnP의 차후 버전으로써 제시되고 있다. DP4WS[6]는 기존의 UPnP 1.0의 기능인 서비스의 정의, 발견, 제어, 이벤트의 처리 등을 웹서비스를 통하여 처리한다. 그리고 WS-Addressing[15], WS-Security[17]등의 웹서비스를 위한 스펙과 첨부(Attachment)를 위한 MTOM(Message Transmission Optimization Mechanism)[8] 스펙 등을 사용하여 선택적으로 추가하여 사용할 수 있는 서비스를 정의하고 있다.

ESB 미들웨어는 본 논문에서 제시한 WSCG와 같이 다양한 서비스간의 이질성을 극복하기 위한 노력을 하고 있다. 하지만 최근 이슈가 되고 있는 임베디드와 홈네트워킹에서는 사용하기 어렵다는 단점이 있다. 그리고 UPnP1.0과 DP4WS는 홈네트워크 통신망 안에서는 아주 유용하지만 외부 네트워크에 연결되어 서비스들을 통합하여 사용하기에는 부족한 점이 많이 있다. 이러한 문제점을 극복하기 위해 본 논문에서 제시한 WSCG는 서비스들 간의 이질성을 극복하는 동시에 이를 임베디드에서 사용할 수 있도록 경량화하고 기존의 서비스들을 통합하여 외부의 네트워크에 서비스들을 제공할 수 있도록 한다.

3. WSCG 아키텍처

본 절에서는 본 논문에서 제안하고 있는 WSCG의 전체적인 아키텍처에 대한 내용을 다룬다. WSCG 아키텍처는 크게 3개의 부분인 SOAP Server, Service Container, Service Conversion Framework로 나뉜다. SOAP Server는 외부로부터 들어오는 웹서비스 요청을 받고나 내부에서 외부로 어떤 웹서비스를 호출할 때 처리하기 위한 서버이다. Service Container는 기존의 홈 네트워크 또는 어떤 컴퓨팅 서비스를 제공할 수 있도록 구성되어 있는 서비스 환경 또는 이 서비스에 추가적인 기능을 제공하기 위한 서비스들을 관리한다. 그리고 Service Conversion Framework 는 SOAP Server와 Service Container를 중재하여 기존 서비스들을 웹서비스로의 변환 및 시스템 운영을 위한 관리를 담당한다.

3.1. 시스템 아키텍처

그림 1은 트랜스포트(Transport), 프로토콜(Protocol), 서비스(Service), 어플리케이션(Application) 등의 4개의 레이어로 나누어진 WSCG의 계층 형 아키텍처 구조이다. 트랜스포트 계층은 웹서비스를 이용하는 클라이언트의 접근을 받아들이는 역할을 수행한다. 이는 내부에 스레드 풀(Thread Pool) 패턴을 이용하여 다수의 사용자의 동시 접근을 가능하게 한다. 프로토콜 계층은 클라이언트로부터 전달되어진 요청 메시지를 분석하여 SOAP 메시지를 클래스화한 WSCG의 내부에서 사용하는 데이터 타입 인 Envelope로 변환하는 역할을 수행한다. WSCG는 시스템 초기화 시에 트랜스포트 계층의 어댑터들과 프로토콜 계층의 핸들러들에 대한 정보를 XML기반 설정파일로부터 읽어서 해당 핸들러와 어댑터들을 생성하고 로딩(Loading) 한다. 서비스 계층은 서비스들을 관리하고 있는 서비스컨테이너들로부터 서비스의 이름, 함수 이름 등의 정보를 받아서 커넥터 및 Service Container의 정보와 함께 저장한다. 클라이언트가 해당 서비스들을 요청할 때 서비스 계층은 서비스 컨테이너에게 서비스들에 대한 호출을 요청하며 그 결과를 받아 클라이언트들에게 전달한다. 어플리케이션 레이어는 실제 어떤 서비스를 제공할 수 있는 서비스 컴포넌트들을 관리하는 서비스 컨테이너가 위치한다. 서비스 컨테이너는 자신이 관리하는 서비스 컴포넌트들의 정보를 커넥터를 통하여 서비스레이어로 전달하고 후에 클라이언트로부터 해당 서비스들에 대한 요청이 있으면 실제 서비스들을 호출하여 그 결과를 반환한다.

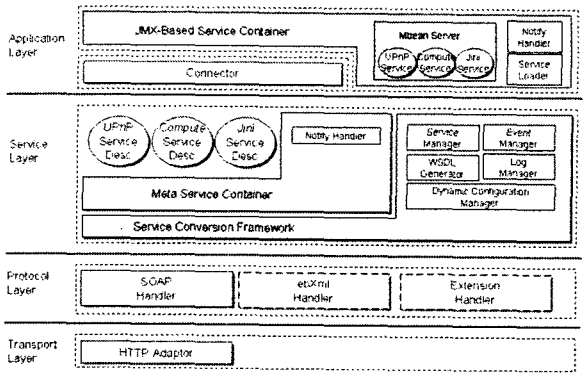

그림 1 WSCG Layered System Architecture

WSCG의 초기화 작업들은 Service Conversion Framework로부터 시작된다. Service Conversion Framework는 처음 XML 형태를 지니는 환경설정 파일로부터 어댑터와 핸들러의 정보를 읽어 들여 해당 서비스 컴포넌트들을 생성한다. 그리고 내부에 서비스의 메타 정보를 저장하는 Meta Service Container를 생성하면 Meta Service Container는 자신이 사용할 수 있는 커넥터 정보를 설정파일로부터 읽어 들인다. 그리고 커넥터가 생성되면 서비스 컨테이너로부터 실제 서비스의 정보를 넘겨받아 그 정보를 저장하고 각 서비스들에 대한 인터페이스를 기술하고 있는 WSDL을 생성한다.

Service Conversion Framework의 초기화 완료 후 클라이언트들은 WSCG에서 관리되는 서비스들에 대한 인터페이스들을 기술하고 있는 WSDL 얻어서 웹서비스 기반으로 해당 서비스들을 호출한다. 클라이언트가 하부 프로토콜로 HTTP 프로토콜 상에서 SOAP 메시지를 사용해 서비스 호출을 한다면 제일 먼저 WSCG의 트랜스포트 계층으로 접근하게 된다. HTTP 어댑터는 인증 및 HTTP 프로토콜과 관련된 정보들을 처리한 후에 메시지를 프로토콜 레이어에 존재하는 특정 핸들러에게 전달한다. 각 핸들러가 수행하는 주요 역할은 특정 메시지 프로토콜에 해당하는 메시지 정보를 파싱(Parsing)해서 처리한 후에 Envelope로 변환하여 서비스 레이어의 Service Conversion Framework로 전달한다. Service Conversion Framework는 서비스 호출에 관련된 로깅을 시작하고 Meta Service Container에게 클라이언트에 의해 요청된 서비스의 정보인 ServiceDesc를 검색 및 실제 서비스들에 대한 호출을 요청한다. Meta Service Container는 해당 서비스들에 관련된 정보를 찾으면 정보에 맞는 커넥터(Connector)를 선택하여 실제 서비스들을 관리하고 있는 Service Container들에게 호출을 요청해서 그 결과를 해당 클라이언트에게 반환한다.

3.1. WSCG 아키텍처 디자인 이슈

WSCG는 유비쿼터스 환경에서 상호 이질적인 플랫폼이나 프로토콜을 사용하는 기존 서비스들을 웹서비스 기반으로 통합하며 이들 서비스들을 임베디드 디바이스와 같은 제한된 경량 컴퓨팅 환경에서 제공할 수 있도록 하고 있다. 기존에 구성되어 있는 서비스들의 컴퓨팅 환경을 통합하여 웹서비스의 환경으로 변환하기 위해 WSCG는 실제 서비스를 관리하는 Service Container와 Service Container에 저장되어 있는 서비스들에 관련된 정보들을 관리하는 Meta Service Container를 분리하여 사용한다. WSCG는 두 컨테이너 사이에 커넥터를 이용하여 통신함으로써 어떠한 Service Container 일지라도 커넥터를 이용하여 이 Service Container에 있는 서비스를 사용할 수 있다. WSCG는 Service Container로써 JMX 기반의 Service Container를 기본적으로 제공하며 Service Container에 서비스 객체들을 생성하여 플러그인(Plug-In)형태로 추가할 수 있다. 이들 서비스 객체들에는 J2EE의 EJB 컨테이너나 서블릿 엔진과 같이 기존의 외부 컨테이너에서 제공되는 외부 서비스 객체들과 연동하기 위한 객체들도 존재할 수 있고 컨테이너 내부에서 새롭게 제공하는 실제 내부 서비스 객체일수도 있다. 특히 경량 컴퓨팅 환경을 요구하는 임베디드 디바이스들은 디바이스 관련 서비스들이나 이들 서비스들을 관리하는 Service Container들을 임베디드 디바이스 외부에 위치시킴으로서 경량화 목적을 달성한다. WSCG에서는 임베디드 컴퓨팅 환경을 위한 JMX 기반의 Service Container나 혹은 이들 Service Container내부에 서비스들을 제공할 수 있기 때문에 경량화를 요구하는 임베디드 컴퓨팅 환경을 만족시킬 수 있다. 또한 기존의 다양한 플랫폼이나 프로토콜을 사용하는 서비스들의 통합을 웹서비스 기반으로 쉽게 통합할 수 있다.

4. 시스템 개발 및 적용

그림 2는 본 논문에서 제안한 WSCG를 홈 네트워크 상황에서 미디어 스트리밍 서비스에 적용해본 결과이다. 위의 적용에서 사용한 구현환경은 WSCG를 구동하기 위한 임베디드 보드의 운영체제는 Linux를 사용하였고 그 위에 자바를 구동하기 위한 JVM으로써 Blackdown java Linux JVM을 사용하였다. 구현에 사용된 자바 SDK는 J2SE1.3을 사용하였다. 그리고 클라이언트의 구현은 JMF(Java Mulitmedia Framework)를 사용하였다.

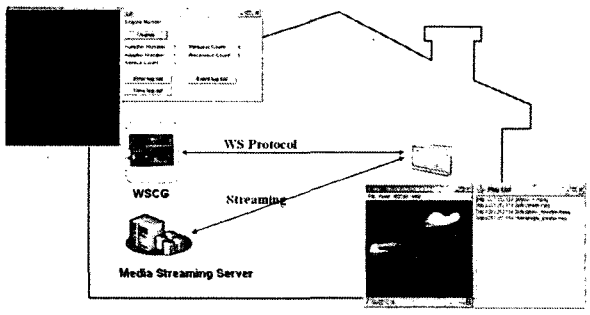

그림 2 WSCG를 홈 네트워크 미디어 스트리밍 서비스 적용

그림 2에서 기존의 미디어 스트리밍을 위한 환경이 이미 홈 네트워크에 적용되어 있다고 가정하자. 홈 네트워크의 새로운 미디어 클라이언트는 WSCG에 접근해 기존 미디어 스트리밍 서비스에 대한 인터페이스를 기술한 WSDL을 다운로드한다. 이러한 WSDL을 기반으로 클라이언트는 미디어 스트리밍 서비스 서버의 위치 및 서버에서 제공하는 스트리밍의 목록을 받아가게 된다. 이들 정보를 기반으로 클라이언트는 직접 메디아 스트리밍 서버에게 접근하고 원하는 메디아의 스트리밍을 요청한 후 서비스를 받는다.

5. 결론

유비쿼터스 환경과 홈 네트워킹 그리고 임베디드를 이용한 수많은 서비스 환경이 생겨나면서 이들의 상호 호환성의 문제가 크게 부각되고 있다. 그리고 이런 문제를 풀기 위하여 많은 곳에서 다각도로 노력을 하고 있다. 본 논문에서는 웹서비스를 이용하여 상호 호환성의 문제를 해결하고 다양한 프로토콜을 쉽게 웹서비스와 결합 할 수 있으며 임베디드의 환경에 맞춰 경량화 되어진 WSCG 시스템을 소개하였다. WSCG의 유연한 아키텍처는 기존에 구축되어 있는 홈 네트워킹 환경이나 다른 많은 환경에서 서비스들을 통합하고 동시에 새로운 서비스를 제공할 수 있는 기반을 만들 수 있다. 또한 임베디드로의 구동 환경을 넓힘으로써 좀 더 다각화된 분야에서 이를 적용시킬 수 있게 되었다.

6. 레퍼런스

[1] Software-Oriented Architecture at IBM Web Site, http://www-306.ibm.com/software/solutions/soa/
[2] The Mission and Future of Integration, Gartner Group, http://www.gartner.com/regionalization/img/gpress/pdf/gartner_exec_report_sample_WEB.pdf
[3] James Pasley, How BPEL and SOA are changing Web services development, Cape Clear Software, 2003, IEEE CNF
[4] Microsoft on the Enterprise Service Bus, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/BTS_2004WP/html/47850cbd-63ed-4370-a467-6bd320636902.asp
[5] Min Luo, Goldshlager, B., Liang-Jie Zhang, "Designing and implementing Enterprise Service Bus (ESB) and SOA solutions", Services Computing, 2005 IEEE International Conference on, Vol2, July 2005
[6] Technical Introduction to the Devices Profile for Web Services, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/ deviceprofile-techoverview.asp
[7] UPnP Standards, http://www.upnp.org/standardizeddcps
[8] SOAP MTOM spec, http://www.w3.org/TR/soap12-mtom/
[9] WebSphere, http://www-306.ibm.com/software/websphere/
[10] WebLogic, http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/ products/weblogic/
[11] Web Services Architecture, http://www.w3.org/TR/ws-arch/
[12] WSDL , http://www.w3.org/TR/wsdl
[13] SOAP, http://www.w3.org/TR/soap/
[14] Web Services Dynamic Discovery (WS-Discovery), http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-discovery1004.pdf
[15] Web Services Addressing (WS-Addressing), http://www.w3.org/Submission/ws-addressing/
[16] Web Services Eventing (WS-Eventing), http://ftpna2.bea.com/pub/downloads/ WS-Eventing.pdf
[17] Web Service Security(WS-Security), http://www-128.ibm.com/developerworks/library/ws-secure/