임베디드 장치를 위한 동적 서비스 연결 프레임워크

염 귀 덕[†] · 이 정 금[††]

요 약

최근 IT서비스 분야에서는 서비스들간의 자유로운 융·통합이 이슈로 부각되고 있으며 이를위해 많은 소프트웨어 개발 벤더들은 SOA(Service-Oriented Architecture)기반의 통합 서비스 플랫폼을 제시하고 있다. 그러나 이러한 서비스 플랫폼들은 주로 엔터프라이즈 어플리케이션을 대상으로 하며 대규모의 복잡한 시스템에만 적재되어 운용되고 있다. 즉, 서비스 융·통합의 관점은 상위 수준의 서비스 어플리케이션에 초점을 맞추고 있으며 하위 수준의 임베디드 소프트웨어 분야는 고려되지 않고 있는 실정이다. 기존 서비스들과 최근 도입되고 있는 임베디드 소프트웨어 기술 기반의 서비스들과의 융·통합을 위해서는 장치에 대한 서비스화를 지원하는 기술이 필요하다. 본 논문에서는 임베디드 장치의 서비스화를 지원하는 SOA기반의 확장 가능한 구조를 가진 동적 서비스 연결 프레임워크를 제시한다. 또한 임베디드 보드 기반의 로봇 장치에 구현한 프레임워크를 적재하여 외부의 이기종 서비스나 장치들과의 다양한 상호작용을 데모 시나리오를 통하여 보여준다.

키워드 : 임베디드 장치, 동적 서비스 연결 프레임워크, SOA, 융·통합

A Dynamic Service Binding Framework for Embedded Devices

Yeom, Gwyduk[†] · Lee, Jeonggeum[††]

ABSTRACT

In this paper we present a translation lookaside buffer (TLB) system with low power consumption for embedded processors. The proposed TLB is constructed as multiple banks, each with an associated block buffer and a corresponding comparator. Either the block buffer or the main bank is selectively accessed on the basis of two bits in the block buffer (tag buffer). Dynamic power savings are achieved by reducing the number of entries accessed in parallel, as a result of using the tag buffer as a filtering mechanism. The performance overhead of the proposed TLB is negligible compared with other hierarchical TLB structures. For example, the two-cycle overhead of the proposed TLB is only about 1%, as compared with 5% overhead for a filter (micro)-TLB and 14% overhead for a same structure without continuos accessing distinction algorithm. We show that the average hit ratios of the block buffers and the main banks of the proposed TLB are 95% and 5% respectively. Dynamic power is reduced by about 95% with respect to with a fully associative TLB, 90% with respect to a filter-TLB, and 40% relative to a same structure without continuos accessing distinction algorithm.

Key Words : Embedded Devices, Dynamic Service Binding Framework, Service Oriented Architecture, Convergence/Integration

1. 서 론

기존에는 장치를 제어하기 위한 소프트웨어 개발은 벤더(vendor)에 고유한 제어 기술이 적용되거나 특정 서비스 플랫폼을 이용한 제어 환경 구축이 주류를 이루는 추세였다. 이는 유사한 기능을 제공하는 제품들간에 상호운용을 어렵게하고 이를 지원하기 위한 부가적인 기술을 별도로 개발해야 하는 단점이 있다. 또한 하드웨어와 소프트웨어에 대한 이원화된 시각 차로 인해 하드웨어는 하드웨어 분야대로 소프트웨어는 소프트웨어 분야대로 각각 양극화된 기술적 접근이 이루어지고 있다. 실제로 대부분의 분야에서 장치는 장치일 뿐 이를 소프트웨어나 서비스로 인식하는 경우는 드물다.

하드웨어 장치에 대한 서비스 개념의 도입은 빠르게 변화해가는 IT분야에서의 기술적 낙관론을 지원할수 있는 중요한 배경이 될 수 있다. 이는 최근 이슈로 제시되고 있는 신기술들간의 융·통합을 지원하기 위하여 장치의 서비스화가 기술적 문제점을 극복할 수 있는 중요한 역할을 차지할 수 있음을 의미한다.

서비스들간의 융·통합은 이기종 서비스들간에 자유로운 서비스 교환이 가능한 구조를 의미하며 이는 유비쿼터스 환경을 구축하는데 필수적인 기술적 요소이다[1]. 그 한 예로 엔터프라이즈 어플리케이션 환경에서는 상이한 구조의 서비스들간에 자유로운 서비스 교환이 가능하기 위해SOA(Service-Oriented Architecture)기반의 웹 서비스 기술이 이를 지원

† 정 회 원 : 서울시립대학교 Bk21 연구교수
†† 정 회 원 : Alticast Corporation Senior Engineer
논문접수 : 2006년 11월 6일, 심사완료 : 2007년 3월 26일

하고 있다. 그러나, 임베디드 어플리케이션 환경에서는 이를 지원하기 위한 표준화된 플랫폼 기술이 대중화되지 않고 있으며, 앞에서 제시한 장치의 서비스화에 대한 개념의 도입이 이루어지지 않고 있다. 최근 홈 네트워크 분야에서 장치들간의 상호운용이 이슈로 대두되면서 장치들간의 융·통합을 지원하기 위한 서비스 플랫폼들이 개발되고 있는 실정이다. 장치의 서비스화에 있어서는 UPnP[2]기술에서 웹 서비스[3] 기술을 기반으로 하는 UPnP 2.0[4]이 등장할 예정이다.

본 논문에서는 표준 서비스들간의 융·통합 지원과 장치에 대한 서비스화를 위해 컴포넌트 기반의 동적 서비스 연결 프레임워크를 제시한다. 본 프레임워크에서는 다양한 표준 서비스 플랫폼의 지원을 위해 표준화된 인터페이스를 정의 및 구현하도록 하며 동적 컴포넌트 교환 기술을 이용해 실행 중 동적으로 서비스 플랫폼을 추가 및 삭제 할수 있으며 다양한 서비스를 제공한다. 하드웨어를 제어하는 장치 서비스의 경우 지정된 인터페이스의 구현보다는 표준 C++ 클래스의 구조를 지원함으로써 기존 제어 객체의 수정 작업을 최소화하고 객체 구현의 유연성을 최대한으로 보장한다. 또한 임베디드 보드 기반의 로봇 장치에 구현한 프레임워크를 적재하여 외부의 이기종 서비스나 장치들과의 자유로운 상호작용을 실험을 통하여 보여준다.

본 논문의 구성은 다음과 같다. 2장에서는 프레임워크에 대한 관련연구들을 살펴보고 3장에서는 프레임워크의 아키텍처를 제시한다. 4장에서 프레임워크의 주요 기술 구조를 설명한다. 그리고 5장에서는 프레임워크를 탐재한 실험의 주요 결과를 소개한다. 마지막 6장에서 결론으로 마무리한다.

2. 관련연구

본 연구는SOA(Service-Oriented Architecture)의 모델을 기본구조로 하여 하드웨어 자원의 서비스화에 초점을 두고 있다. SOA는 서비스라 불리는 분할(decomposition)된 어플리케이션 단위들을 약결합하여(loosely-coupled) 하나의 완성된 어플리케이션으로 만드는 아키텍처[5]로서 하드웨어 자원도 하나의 서비스로 바라보고 있다.

SOA를 구현하고 있는 대표적인 사례로는 ESB(Enterprise Service Bus)[6, 7]가 있으며 웹 서비스를 이용해 하드웨어 장치들에 대한 운용 메커니즘을 제공하는 사례로 DP4WS(Device Profile for Web Service)[4]가 있다.

ESB는 비즈니스 내에서 서비스, 어플리케이션, 그리고 자원을 연결 및 통합하는 미들웨어이다. 기존의 통합방법에 대한 새로운 방법론이며, 서로 다른 언어와 프로그래밍 모델에서 개발된 소프트웨어들의 통합과 연결을 가능케하여준다. 여기에는 웹 서비스와 메시징기반의 전송 및 라우팅도 포함된다. ESB의 상위계층에는 사용자 상호작용 서비스(user interaction service), 어플리케이션 서비스(application service),정보 서비스(information service), 프로세스 서비스(process service), 그리고 커뮤니티 통합 서비스(community integration service)의 다섯 개의 서비스 계층 계층화되어 있어서 어떠한 형태의 통합도 가능케하는, 특히, ESB의 범용 연결 계층(universal connectivity layer)은 기업간 통합의 규모와 범위를 더욱 확장하였다[4].

DP4WS[4]는 서비스를 임베디드화한 대표적인 기술적 사례로 장치와 서비스의 개념을 논리적으로 동일한 영역으로 추상화하고 있다. DP4WS는 웹서비스의 확장 스펙을 이용해 장치의 서비스 운용환경을 정의하는 기술로서 기존의 UPnP 1.0 스펙[2]을 참조하여 웹서비스 기반의 UPnP 모델을 구현하고 있으며, 향후 UPnP 2.0 모델로 정의될 예정이다.

DP4WS는 크게 디바이스 웹 서비스의 정의, 발견, 제어 및 이벤트 처리 등으로 구성되며, 암호화된 메시지 정보를 주고받는다. 디바이스 웹 서비스 정의는WSDL(Web Service Description Language)[8]를 사용해 정의되며, WS-Discovery [9]를 통해 디바이스의 웹 서비스를 검색하게 된다. 또한 검색된 디바이스의 웹 서비스는 WS-Metadata[10]스펙을 통해 서비스의 메타 데이터 정보를 주고받으며 SOAP(Simple Object Access Protocol)[11]을 통해 제어된다. WS-Eventing [12]스펙을 사용하여 디바이스의 이벤트 메커니즘을 구현할 수 있으며 SOAP의 MTOM(Message Transmission Optimization Mechanism)[13] 스펙을 통해 대용량의 데이터를 전송할 수 있다.

ESB와 DP4WS는 SOA 기반의 서비스 어플리케이션들의 통합과 장치 서비스들의 웹 서비스화에 초점을 두고 있지만 이를 지원하기 위해서 수동으로 해당 서비스 또는 장치들의 시스템 상에 이를 지원하기 위한 확장 모듈을 추가로 설치하거나 신규 개발을 해야하는 문제점을 가지고 있다. 또한, 웹 서비스 플랫폼을 기본 플랫폼으로 사용하므로 이기종 서비스 플랫폼과의 상호운용성이 어렵다.

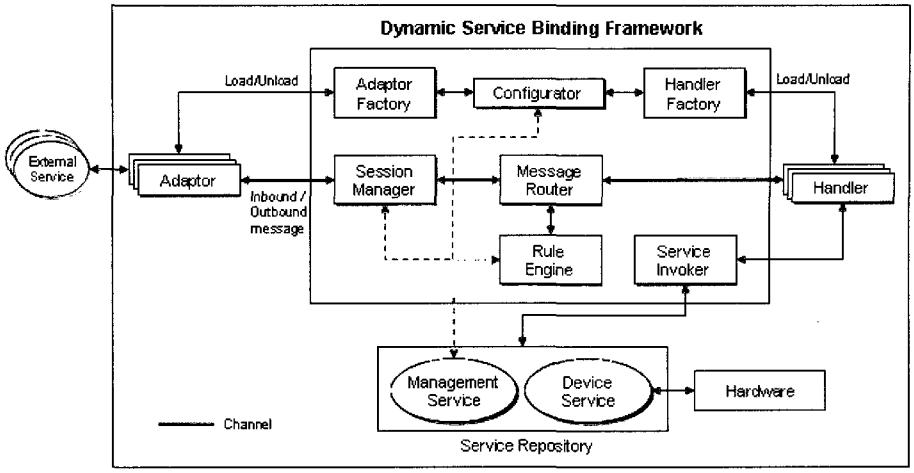
본 논문에서는 동적 서비스 연결 프레임워크를 통해 기존 SOA플랫폼들이 가지는 정적인 형태의 서비스 구조의 문제점들을 보완하고 이기종 플랫폼과의 상호운용성을 보장할 수 있는 프레임워크를 제시한다.

3. 프레임워크 아키텍쳐

본 장에서는 하드웨어 제어를 위해 사용되는 로컬 객체들을 다양한 형태의 표준 서비스로 표현하고 이를 관리하는 동적서비스 연결 프레임워크를 제시한다. 동적 서비스 연결 프레임워크는 SOA의 기본구조를 기반으로 장치의 서비스화에 초점을 두고 있으며, 서비스화 과정에 있어 동적으로 서비스 확장구조를 지원하므로 실시간으로 하드웨어의 자원을 사용할 수 있다.

3.1 동적 서비스 연결 프레임워크 아키텍쳐

동적 서비스 연결 프레임워크(DSBF: Dynamic Service Binding Framework)는 내부의 장치 제어 객체를 다양한 표준 서비스로 제공하기 위해 확장이 용이하고 실시간 서비스 지원이 가능한 구조를 지원한다. (그림 1)은 동적 서비스 연결 프레임워크의 구조를 보여주고 있다.



(그림 1) 동적 서비스 연결 프레임워크 아키텍처

프레임워크에서는 다양한 표준 서비스들의 지원을 위해 IAdaptor와 IHandler라는 두 가지 형태의 인터페이스를 제공한다. 이는 서비스 플랫폼 벤더나 그룹이 해당 서비스 기술의 스펙을 IAdaptor와 IHandler의 인터페이스에 맞춰 구현하도록 하여 프레임워크가 다양한 표준 서비스 플랫폼들을 지원할 수 있도록 한다. IAdaptor와 IHandler인터페이스에 따라 구현된 어댑터와 핸들러는 AdaptorFactory와 HandleFactory를 통해 동적으로 적재 또는 제거될 수 있도록 구현되었으며 실시간으로 다양한 서비스 플랫폼을 지원할 수 있다[14, 15].

어댑터(Adaptor)와 핸들러(Handler) 사이의 연결을 위해 채널기반의 메시지 전달 구조를 지원한다. 즉, 어댑터를 통해 전달된 메시지 정보는 논리적 개념의 전송 채널을 통해 메시지 라우터(Message router)를 거쳐 핸들러에게 전달되게 된다. 채널은 메시지 컨텍스트 객체로써 메시지를 운반하는 통로의 역할을 한다. 채널은 하드웨어의 운용분야에 따라 단일 채널 또는 다중 채널로 운용될 수 있다. 채널은 채널 풀(pool)을 통해 재사용이 가능한 구조로 설계되었다.

메시지 라우터는 어댑터로 전달된 메시지를 설정된 연결 테이블(mapping table)에 있는 정보를 기준으로 해당 핸들러에게 전달한다. 연결 테이블은 메시지의 유형 정보와 연결 테이블에 등록된 처리자(handler)의 정보를 정의하고 있으며 설정관리자(Configurator)를 통해 실시간적으로 갱신된다. 또한 프레임워크의 설정에 따라 메시지 처리 규칙 엔진(Rule Engine)에 정의된 규칙이 반영될 수 있는데, 이는 특정 장치로부터의 전달되는 메시지를 차단거나 변형하는 작업 등을 수행할 수 있도록 한다. 메시지 처리 규칙 엔진에서 사용되는 규칙 정보는 메시지에 대한 비교 조건과 처리 순서를 정의하고 있어 프레임워크로 하여금 가변적인 메시지 처리를 지원할 수 있도록 한다.

프레임워크에서 포함하고 있는 장치 서비스 객체는 GCC_XML[16]그룹에서 제시하고 있는 인터페이스 정의기술을 기반으로 구현된 동적 적재 라이브러리의 구조를 가지며 어댑터와 핸들러와 같은 과정으로 해당 적재자(Loader)를 통해 메모리에 적재된다.

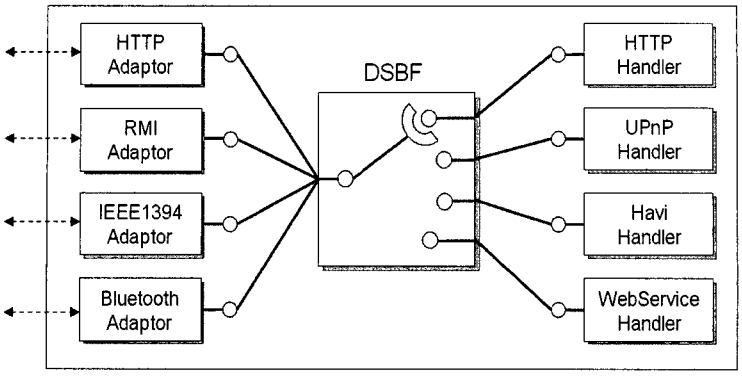
동적 서비스 연결 프레임워크에 대한 관리 인터페이스는 웹 기반의 관리 콘솔을 제공하며 이는 서비스 저장소(Service Repository)내에 관리 서비스(Management Service)를 통해 내부 프레임워크내의 주요 모듈들을 제어하거나 정보를 제공받을 수 있도록 한다.

3.2 어댑터와 핸들러 아키텍처

동적 서비스 연결 프레임워크는 표준 서비스 플랫폼들의 지원을 위해 어댑터와 핸들러 구조를 제시한다. (그림 2)는 어댑터와 핸들러의 연결구조를 보여주고 있다. 어댑터와 핸들러는 각각 프레임워크에서 제시하는 표준 인터페이스 IAdaptor와 IHandler를 구현한다. 어댑터의 경우 특정 서비스 기술의 메시지를 인식하고 이를 통해 동적 서비스 연결 시스템이 요구하는 메시지 포맷으로 변환하는 역할을 수행하며, 핸들러는 어댑터로부터 전달된 메시지 정보를 분석하고, 이를 정의된 표준 기술 스펙에 따라 처리하는 작업을 수행한다. 예를 들어 UPnP 핸들러의 경우 UPnP 기술 스펙에서 제시하고 있는 메시지 처리 구조를 정의하고 메시지 라우터로부터 전달된 UPnP메시지를 정의된 규칙에 맞춰 처리한다.

어댑터는 IAdaptor인터페이스를 상속받아 해당 서비스 플랫폼의 메시지를 수신하는 모듈과 송신하는 모듈로 구성된다. 수신하는 모듈의 경우 운영체제로부터 해당 서비스 플랫폼이 사용하는 프로토콜 인터페이스를 통해 전달되는 메시지 정보를 읽어들여 동적 서비스 연결 프레임워크가 제시하는 표준 메시지로 변환하는 역할을 한다. 송신하는 모듈의 경우에는 프레임워크로부터 전달된 결과 메시지를 해당 서비스 플랫폼이 사용하는 프로토콜로 변환하여 전송하는 역할을 정의한다.

핸들러의 구조는 프레임워크가 제시하는 표준 메시지를 전달받아 분석하는 모듈과 분석된 결과를 처리하는 모듈로 구성된다. 분석 모듈의 경우 메시지가 정의하고 있는 명령정보를 해석(de-serialize)하고 이를 처리 모듈로 넘기게 된다.



(그림 2) 어댑터와 핸들러의 아키텍처

처리 모듈에서는 표준 서비스 플랫폼의 규칙에 따라 해당 메시지가 요구하는 작업을 수행하며 수행된 결과는 다시 직렬화(serialize)되어 프레임워크로 전달되게 된다.

4. 아키텍처상의 이슈

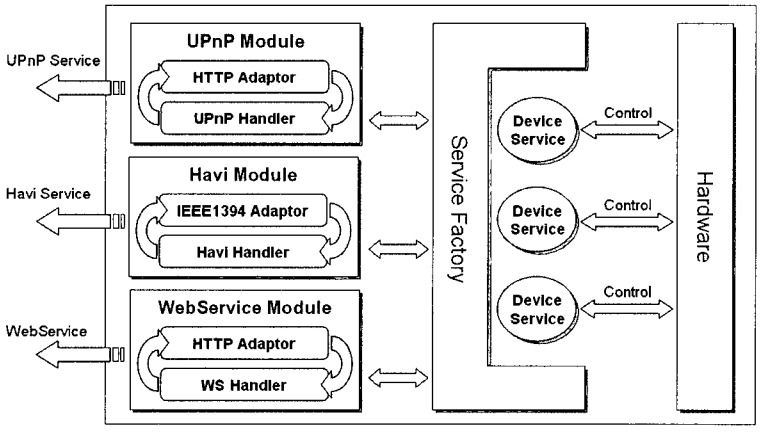
동적 서비스 연결 프레임워크는 장치 종속적인 서비스를 다양한 표준 서비스 미들웨어 플랫폼의 서비스로 제공할 수 있는 구조를 지원한다. 이는 하드웨어 장치로 하여금 SOA 기반의 서비스로 인식하도록 하며 다른 표준 서비스 장치들과의 상호운용을 지원한다. 표준 서비스 미들웨어 플랫폼은 동적 서비스 연결 프레임워크상에서 어댑터와 핸들러의 형태로 구현되며 이는 설정 관리자를 통해 확장가능한 형태로 배포된다. 동적 서비스 연결 프레임워크는 장치 서비스를 SOA기반의 서비스로 제공하기 위해 어댑터, 핸들러, 그리고 장치 서비스들간의 유기적인 관계를 확장가능한 구조로 제공한다.

4.1 하드웨어 제어 객체의 표준 서비스화

동적 서비스 연결 프레임워크에서는 장치 제어 객체와 표준 서비스들의 동적 연결을 통해 하드웨어 제어 객체를 표준 서비스로 제공한다. 이를 위해 프레임워크에서는 어댑터와 핸들러를 통해 표준 서비스 플랫폼을 구현하고 있으며 이들을 실행중인 시스템상에서 동적으로 연결할 수 있는 구조와 장치 제어 객체들을 실시간적으로 추가할 수 있는 구조를 지원하고 있다. (그림 3)은 동적 서비스 연결 프레임워크가 이러한 구조를 기반으로 외부의 장치나 서비스들에게 특정 서비스 플랫폼이 내장된 서비스로 인식되는 것을 보여준다. 즉, 외부의 장치나 서비스가 내부 하드웨어 장치에 대한 정보를 알지 못해도 해당 하드웨어 장치가 제공하는 기능을 이용 또는 제어할 수 있도록 한다. 예를들어 동적 서비스 연결 프레임워크가 HTTP Adaptor와 WebService Handler를 적재하게 되면 내부의 장치 서비스를 웹 서비스[3]로 배포할 수 있게 하며 HTTP Adaptor(UPnP[2]는 메시지 전송을 위해 HTTP를 사용한다.)와 UPnP Handler의 적재를 통해서는 장치 서비스를 UPnP 장치 서비스로 배포될 수 있다. 또한 하나의 장치 서비스가 여러 개의 표준 서비스로 인식되기 때문에 하드웨어의 적용분야와 적응 환경을 확대할 수 있다.

4.2 메시지 처리

동적 서비스 연결 프레임워크에서는 메시지 라우터를 통해 어댑터로부터 전달된 요청 메시지를 특정 핸들러에게 전달한다. 메시지 라우터는 적재 유·무가 동적으로 변경되는



(그림 3) 장치 서비스의 표준 서비스화

핸들러의 특성을 지원하면서 다양한 메시지 전달 규칙을 적용할 수 있다.

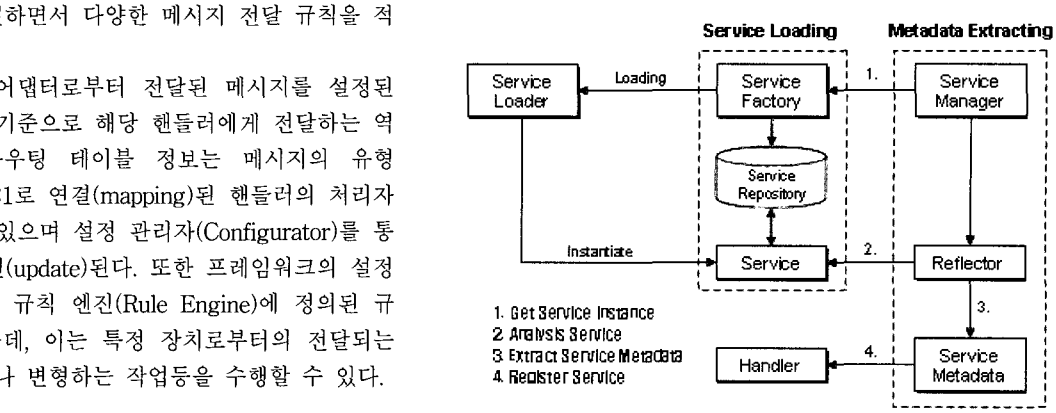
메시지 라우터는 어댑터로부터 전달된 메시지를 설정된 연결 테이블 정보를 기준으로 해당 핸들러에게 전달하는 역할을 한다. 이때 라우팅 테이블 정보는 메시지의 유형(Type)정보와 이에 1:1로 연결(mapping)된 핸들러의 처리자의 정보를 정의하고 있으며 설정 관리자(Configurator)를 통해 실시간적으로 갱신(update)된다. 또한 프레임워크의 설정에 따라 메시지 처리 규칙 엔진(Rule Engine)에 정의된 규칙이 반영될 수 있는데, 이는 특정 장치로부터의 전달되는 메시지를 차단시키거나 변형하는 작업등을 수행할 수 있다.

4.3 실시간 모듈 관리

동적 서비스 연결 프레임워크는 어댑터와 핸들러를 관리하기 위해 각각 AdaptorFactory와 HandlerFactory를 제공한다. 이들은 동적 라이브러리 형태(.so. .dll)로 제공되는 어댑터 모듈과 핸들러 모듈을 동적으로 적재하는 기능을 제공한다. 적재된 어댑터 모듈과 핸들러 모듈은 각 Factory에 내장된 저장소(Repository)를 통해 저장되며, 프레임워크의 초기화 단계에서 활성화된다[14, 16].

활성화된 어댑터와 핸들러는 관리자 콘솔 또는 업데이트 관리자(Update Manager)를 통해 수동 또는 자동으로 새로운 어댑터와 핸들러로 교체 가능하다. 이때 기존 어댑터와 핸들러는 새로운 요청 메시지를 받지 않고 기존에 처리중인 메시지만을 처리한 뒤 저장소로부터 제거된다. 동시에 새로운 어댑터와 핸들러는 새로운 메시지를 받아 처리하므로 외부의 서비스 사용자는 계속적인 서비스 이용이 가능하게 된다.

어댑터와 핸들러는 이러한 실시간 모듈 관리를 위해서 (그림 4)와 같이 설치 단계에서 제거 단계에 이르는 생명주기(Life Cycle)를 가진다. 생명 주기는 시작(Starting)상태, 활성화(Active) 상태, 대기(Wait)상태, 그리고 중지(Stopping)상태로 나뉘어진다. 시작 상태는 모듈 운영시 요구되는 설정 정보와 기본 값을 초기화 한다. 활성화 상태는 Adaptor와 Handler의 기능을 수행하는 상태로 새로운 메시지를 받거나 요청된 메시지를 처리한다. 대기 상태는 현재 진행중인 작업이 있을 경우 이를 처리한 뒤 더 이상의 신규 작업을 수행하지 않는 좀비(zombie) 상태를 유지한다. 정지 상태는 현재 모듈이 처리중인 작업과 사용중인 자원을 모두 처리 또는 소멸 시킨 뒤 저장소로부터 제거된다.



(그림 5) 장치 서비스 변환 및 배포

4.4 장치 서비스 변환과 배포

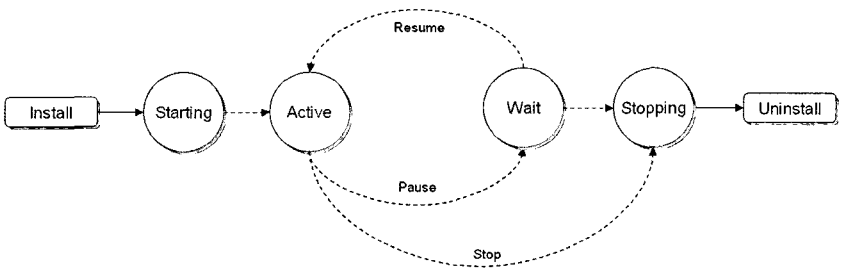
동적 서비스 연결 프레임워크에서는 새로운 하드웨어 제어 객체가 정의되었을 때 서비스 관리자를 통해 이를 동적으로 적재하여 외부의 장치 또는 서비스들에게 배포한다. 하드웨어 제어 객체는 GCC_XML[16]그룹에서 제시하고 있는 인터페이스 정의 기술을 기반으로 구현된 동적 적재 라이브러리로서 규격화된 인터페이스의 정의에 상관없이 표준 C++ 클래스의 구조를 따르는 모든 일반 객체를 지원한다. (그림 5)는 장치 서비스의 변환과 배포 과정을 보여주고 있다.

적재된 장치 서비스는 다양한 형태의 표준 서비스로 배포하기 위하여 Reflector를 통해 장치 서비스의 객체 인터페이스 정보를 추출, 이를 배포하고자 하는 각 핸들러들에게 전달하게 된다. 장치 서비스의 인터페이스 정보를 넘겨받은 각 핸들러들은 해당 플랫폼 이 정의하고 있는 메카니즘에 따라 신규 서비스를 등록하고 배포하는 과정을 수행한다.

서비스 배포과정에서 사용되는 Reflector는 Seal 프로젝트[17]의 반사(Reflection) 기술을 이용하여 GCC_XML[16]을 통해 추가된 객체의 인터페이스 정보를 적재된 상태의 장치 서비스로부터 추출하는 기능을 구현하고 있으며 이를 기반으로 장치 서비스에 대한 동적 호출을 처리한다.

5. 실험 및 결과

본 장에서는 동적 서비스 연결 프레임워크의 구현 및 실제 하드웨어 장치에 적재하여 실험한 내용과 결과에 설명한다.



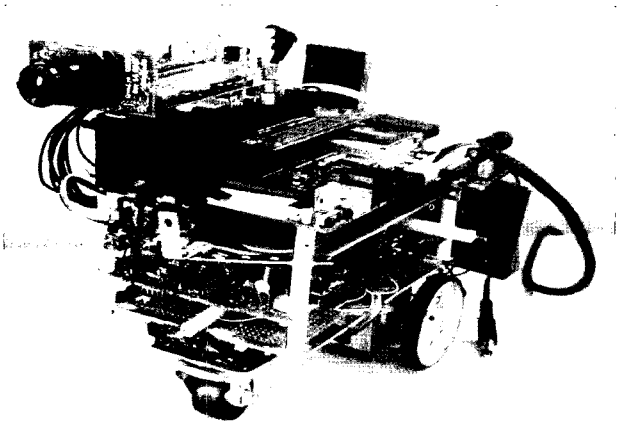
(그림 4) 어댑터와 핸들러의 생명주기

5.1 실험환경

동적 서비스 연결 프레임워크의 구현 환경은 표준 C++ 언어를 기반으로 임베디드 리눅스(Embedded Linux)와 WinCE .NET에서 운용될 수 있는 형태로 구현되었다. XML 파싱(parsing)을 위한 Xerces-C라이브러리와 쓰레드(thread) 처리를 위한 POSIX Thread라이브러리가 사용되었다. 실제로 동적 서비스 연결 프레임워크를 적재할 디바이스로 임베디드 보드 기반의 로봇 디바이스를 사용하였으며, 동적 서비스 연결 프레임워크에 새로운 모듈들을 제공하기 위한 배포 서버(deploy server)를 구현하였다. 로봇 디바이스에 사용된 임베디드 보드의 사양은 <표 1>과 같다. 로봇이 제공하는 서비스로는 전 방향 운행 서비스와 영상 감시 서비스로 각각 모터 구동 제어와 웹캠 제어를 처리한다. (그림 6)은 실험에 사용된 로봇 장치를 보여주고 있다.

〈표 1〉 임베디드 보드 사양

하드웨어 사양	운영체제 사양
X-Hyper255B - Intel XScale PXA255 - 64MB SDRAM - 32MB Flash - 10Base-T CS8900A - PCMCIA 1 slot	Embedded Linux - kernel 2.4.18 WinCE .NET - 4.2



(그림 6) 프레임워크가 탑재된 로봇 장치

5.2 실험 내용 및 결과

본 논문에서 구현된 동적 서비스 연결 프레임워크는 임베디드 보드기반의 로봇 장치에 적재되어 실험되었다. 실험은 장치의 서비스화 성공 여부와 시스템의 구조상 동적 확장 기능의 수행 여부를 확인하고 다양한 플랫폼 환경에서 장치들간의 상호운용성을 검증하는데 초점을 두었다. 실험에 사용된 프레임워크는 UPnP와 웹 서비스(Web service) 표준 서비스 플랫폼을 지원하며 이를 위해HTTP 어댑터, UPnP 핸들러, RMI 어댑터, 그리고 Web Service 핸들러가 사용되었다. 실험의 주요 내용은 동적 서비스 연결 프레임워크가 탑재된 로봇장치가 UPnP, 웹 서비스 플랫폼을 탑재한 다른 이 기종의 장치와 동적으로 상호운용이 가능한지를 확인하는 것이다.

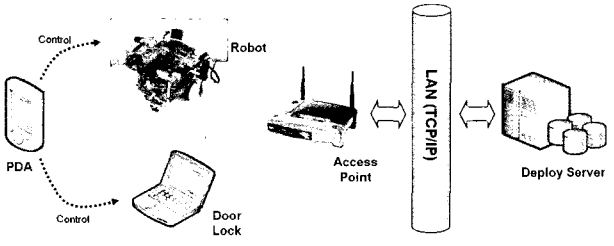
동적인 상호운용은 로봇의 동적 서비스 연결 프레임워크가 UPnP, Web service를 처리할 수 있는 핸들러를 내장하지 않고 작동하는 도중에 다른 서비스 플랫폼을 지원하는 장치를 발견하였을 때 이를 처리하기 위한 핸들러를 외부의 배포 서버로부터 다운받아서 실시간으로 상호 운용하는 것을 의미한다.

이를 위하여 본 실험에서는 크게 내부 망에서의 로봇 제어와 외부 망에서 내부 망내의 로봇을 제어하는 것으로 나누어 실험하였다. 첫번째 실험은 내부 망에서 로봇이 UPnP 제어 포인트(Control Point)를 내장한 장치를 인식하고 이 장치와 상호운용하기 위하여 UPnP 핸들러를 외부의 배포서버로부터 배포를 받아 지역망 상의 다른 UPnP 제어 포인트로부터 제어되는 기능을 실험하였다. 두번째 실험에서는 로봇에 내장된 동적 서비스 연결 프레임워크가 자신의 서비스 및 내부 망에서 발견한 장치들의 서비스들을 웹 서비스로 변환하여 외부 망 상의 웹 서비스 클라이언트에게 제공되어 제어되는 과정을 실험하였다. 로봇을 통해 제공되는 웹 서비스들은 내부 망에서 발견된 다른 장치들의 서비스들이 동적으로 변환되어 웹 서비스로 제공된다.

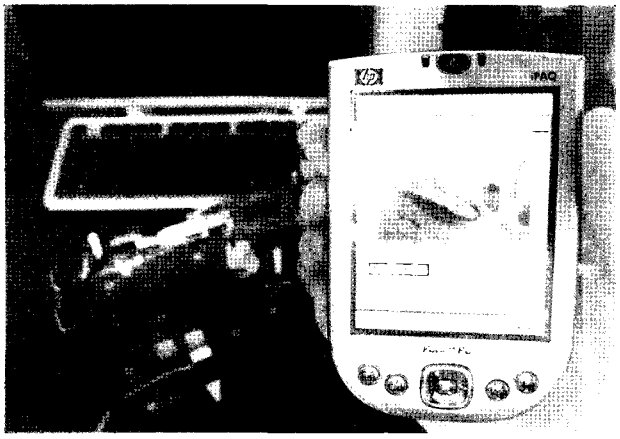
(그림 7)은 실험상에 사용된 장치들의 데모 환경을 보여주고 있다.

각 실험은 초기 로봇 장치가 실험에 제시된 서비스 플랫폼을 지원하지 않는다고 가정하고 실행 중 동적으로 해당 서비스 플랫폼을 적재하여 서비스를 제공하는 방식과 실행 중 동적으로 운영중인 서비스 플랫폼의 지원 모듈들을 교체하는 방식으로 진행되었다.

동적 서비스 적재 기능의 실험에서는 내부 PDA장치로 하여금 내장된 UPnP 제어 어플리케이션을 통해 로봇이 제공하는 이동 서비스와 영상 정보 전송 서비스를 정상적으로 수행하는 것을 확인하였다. 외부망에서는 노트북 상에 미리 제공되는 웹 서비스 어플리케이션을 통해 내부 무선망에 위치한 로봇의 서비스들의 제어를 정상적으로 수행하였다. 또한, 로봇 장치 상에 동작하고 있는 UPnP모듈을 제거한 뒤 실험 도중에 동적으로 UPnP모듈을 추가했을 때도 정상적으로 서비스가 운용됨을 확인하였다. (그림 8)은 로봇이 제공하는 영상 정보 서비스를 UPnP를 지원하는 PDA를 통해 제공되는 모습을 보여주고 있다.



(그림 7) 실험 환경



(그림 8) 영상정보 서비스 수행 화면

실험을 통해 제시되었던 가정 환경상에서 동적 서비스연결 프레임워크는 동작 중에 실시간으로 내부에 존재하는 다른 장치들의 서비스를 인식하고 이들 장치들과의 상호운용에 필요한 각종 서비스 모듈들을 동적으로 활성화하여 내부 환경에 존재하는 다양한 장치들과의 상호운용성을 보장하는 것을 확인하였다.

6. 결 론

본 논문에서 제시하고 있는 동적 서비스 연결 프레임워크는 기존의 장치로 하여금 다양한 표준 서비스 플랫폼을 지원할 수 있는 확장 가능한 동적 관리 구조를 제공한다. 따라서 동적 서비스 연결 프레임워크를 내장한 기존의 장치들은 외부의 다른 서비스 또는 다른 장치들과의 상호운용이 자유롭게 이루어질 수 있다. 이는 이기종의 장치 또는 서비스와의 상호운용을 위해 별도의 추가적인 기술 개발이 요구되지 않음을 의미한다. 단지 동적 서비스 연결 프레임워크에서 제시하고 있는 인터페이스에 맞춰 구현된 서비스 플랫폼 모듈의 추가만이 요구된다. 또한 기존 장치에 동적 서비스 연결 프레임워크를 적재하는데 있어 기존 장치의 서비스 객체가 표준 C++의 클래스 구조를 가진다면 소스 코드의 수정없이 프레임워크 내에 추가할 수 있다.

장치의 사용환경에 따라 자유롭게 서비스의 지원구조를 변경할 수 있는 동적 서비스 연결 프레임워크의 구조는 임베디드 장치 영역 외에도 다양한 서비스 분야에서 범용적으로 이용될 수 있다. 예를 들어 무선 통신 분야에서 다양하게 존재하는 무선 통신 서비스를 소프트웨어 계층에서 통합하려는 기술인 SDR(Software-Defined Radio)[18]을 대치해서 사용될 수 있으며 가정의 다른 벤더들의 가전기기들간에 자유로운 상호운용을 보장할 수 있다. 이처럼 동적 서비스 연결 프레임워크는 차세대 IT 서비스 분야에 융·통합을 지원하는 최적의 솔루션이다.

참 고 문 헌

[1] Tadashige Iwao, Satoshi Amamiya, Guoqiang Zhong, and Makoto Amamiya, "Ubiquitous computing with service adaptation using peer-to-peer communication framework", Distributed Computing Systems, The Ninth IEEE Workshop, 2003.

[2] UPnPTM Device Architecture, http://www.upnp.org/download/UPnPDA10_2000- 0613.htm

[3] Mule's Architecture Guide, http://mule.codehaus.org/Architecture+Guide.

[4] A Technical Introduction to the Devices Profile for WebServices, http://msdn-microsoft.com/library.default.asp?url=/library/enus/dnwebsrv/html/deviceprofile-techoverview.asp

[5] Martin Keen, 'Implementing a SOA using an Enterprise Service Bus', IBM, 2004.

[6] Mike Gilpin, What Is An Enterprise Service Bus?, Integration Landscape, 2004.

[7] Nigel Thomas, 'ESB Technology & Innovation: Extending Web Services with Asynchronous Message Delivery and Intelligent Routing', SpiritSoft, 2004.

[8] W3C Note, WSDL(Web Services Description Language) 1.1,http://www.w3c.org/TR/WSDL/, Mar. 2001.

[9] John Beatty, Gopal Kakivaya, Devon Kemp and Canon Thomas Kuehnel, Web Services Dynamic Discovery (WS-Discovery), MSDN, 2004.

[10] WS-Metadata Exchange1.1, https://www.sdn.sap.com Aug. 2006.

[11] W3C, http://www.w3c.org/TR/NOTE-SOAP-20000508/, May. 2000.

[12] Web Services Eventing, http://www-128.ibm.com/developerworks/library/specification/ws-eventing/

[13] SOAP Message Transmission Optimization Mecha-nism, http://www.w3.org-/TR/soap12-mtom/

[14] Ning ENG, 'S-Module Design for Software Hot-Swapping', 1999.

[15] F.Ning,'S-Module Design for software Hot Swapping Technology', Technical Report SCE-99-04, Systems and Computer Engineering, Carleton University, May. 1999.

[16] GCC_XML, http://www.gccxml.org/

[17] S. Roiser, "The SEAL C++ Reflection System", CHEP'04, Interlaken, Switzerland, Sept. 2004.

[18] Peter G. Cook, 'Overview and Definition of Radio Software Download for RF Reconfiguration in a Technical and Regulatory Context', Base Station Working Group, Apr. 2002.

염 귀 덕

e-mail : kdyeom@venus.uos.ac.kr

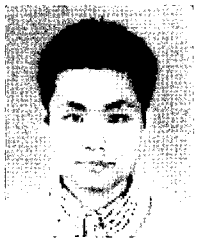
1991년 부경대학교 응용수학과(이학사)

1996년 연세대학교 전자계산전공 (공학석사)

2006년 건국대학교 컴퓨터공학전공 (공학박사)

2006년 12월~현재 서울시립대학교 Bk21 연구교수

관심분야 : 웹 서비스, 임베디드 미들웨어, 유비쿼터스 컴퓨팅



이 정 금

e-mail : stutle@hotmail.com

2004년 건국대학교 정보통신대학컴퓨터공학 전공(공학사)

2006년 건국대학교 컴퓨터공학전공 (공학석사)

2006년 5월~현재 Alticast Corporation Senior Engineer

관심분야 : 유비쿼터스 컴퓨팅, 임베디드 미들웨어, 데이터 방송