韓國 컴퓨터情報學會 論文誌
第10卷 第6號, 2005. 12.

2005-10-6-4-3

유비쿼터스 환경을 위한 서비스 게이트웨이간의서비스 이동 관리 시스템 개발

이 승 근*

Development of Service Mobility Management Systembetween Service Gateways for Ubiquitous Environment

Seung-Keun Lee*

요 약

유비쿼터스 환경은 가전, 센서, 디바이스 등의 기기들로 연결된 네트워크를 이용해서 사용자에게 다양한 서비스를 통해서 제공될 수 있다. 현재, 홈네트워크 환경과 텔리매틱스등의 분야에서 개방형 서비스 게이트웨이가 이를 위해서 이용되고 있으며, 이 게이트웨이를 통해서 서비스의 설치, 배치 및 실행 환경을 제공한다. 서비스 게이트웨이는 유비쿼터스 환경내의 다양한 디바이스들과 이들 서비스들 간의 중간 연결 역할을 담당하며, 서비스들 간의 상호 작용을 지원한다. 유비쿼터스 환경에서는 사용자와 디바이스들은 이동성의 특징을 가지기 때문에 이를 지원하는 서비스는 서비스 게이트웨이간의 이동성을 지원할 수 있어야 한다. 그러나 현재 이용되고 있는 서비스 게이트웨이에서는 서비스의 배포 및 설치 단위인 번들 코드의 다운로드를 통한 원격 설치를 지원하지만, 실행 중인 서비스의 이동을 지원하지 못한다. 본 논문은 개방형 서비스 게이트웨이 프레임워크들 간의 서비스 이동성을 보장하기 위해서 동작되고 있는 서비스의 이동을 관리할 수 있는 시스템을 개발한다. 서비스 이동 관리 시스템은 OSGi 프레임워크에서 동작될 수 있기 위해서 OSGi 표준 스펙에 맞게 설계 및 구현하며, 이동성을 갖는 서비스에 대한 관리를 가능하게 함으로써 유비쿼터스 환경에서 서비스의 이동성을 보다 효과적으로 지원할 수 있다.

Abstract

Ubiquitous environment can be supported by various service on networked appliance, sensors and devices. In home network and telematics, open service gateway is widely used for ubiquitous environment and present environment for service installation, deployment and execution. Service gateway is a middle layer which is located between service and various devices in ubiquitous environment and offers a unique opportunity for pervasive computing as a potential framework for achieving interoperability between various sensors, home appliances, and networked devices. In ubiquitous environments, these services must support the mobility among service gateway because users and devices has a mobility characteristic. However the OSGi framework supports only a remote installation of a bundle, which is a unit that installs and deploys services. This paper develops a system that can manage bundles for supporting this dynamic bundle's mobility between service gateway. This system we are proposing implements a bundle form which can perform in an OSGi framework as well as manage the mobile services. As a result, mobility in a ubiquitous computing environment will be supported more efficiently.

▸ Keyword : 서비스 게이트웨이(Service Gateway), 임베디드 시스템(Embedded System), 유비쿼터스 컴퓨팅(Ubiquitous Computing)

• 제1저자 : 이승근
• 접수일 : 2005.11.02, 심사완료일 : 2005.12.15
* 인하대학교 컴퓨터정보공학과 박사과정

Ⅰ. 서 론

개방형 서비스 게이트웨이는 센서, 임베디드 컴퓨팅 기기, 가전제품들과 이를 이용한 서비스간의 연결을 지원하며, 이를 개발하고 이용할 수 있도록 하기 위한 개방형 표준 프로그래밍 인터페이스를 제공한다[1][2]. 특히, OSGi는 디바이스에 관한 서비스의 전달, 배치, 관리 등에 초점을 맞추고 있으며 Jini나 UPnP 등의 기술에 관한 연결 서비스를 배치 및 상호 작용이 가능하게 한다. OSGi는 홈, 오피스, 자동차와 같은 로컬 네트워크 상에서 다양한 디바이스들을 연결하며, 관리 가능한 확장성 있는 프레임워크를 제공함으로써, 유비쿼터스 컴퓨팅 환경 구축에 폭넓게 이용되고 있다. 또한, 표준적인 실행 환경과 서비스 인터페이스를 정의함으로써, 서로 다른 다양한 종류의 리소스들로부터 서비스와 디바이스들의 동적인 발견과 협력을 증진시킨다[3][4][8][9].

OSGi 기반의 시스템에서는 새로운 서비스를 분배할 수 있는 구조를 가지고 있으며, 그 구조가 로컬 네트워크상의 구성 요소들만으로 이루어져 있어서 비교적 폐쇄적인 특성을 갖는다. 하지만, 그러한 하나의 OSGi 프레임워크 내에서의 서비스 관리와 분배가 동적으로 수행될 수 있는 반면에, 복수의 프레임워크 간의 이동성을 갖는 응용들에 대한 지원은 부족한 실정이다. 그래서 유비쿼터스 공간에서 복수의 OSGi 프레임워크들 간의 사용자, 디바이스 및 센서들의 이동성에 관한 충분한 고려가 이루어져야 하며, 그에 따른 이동성을 지원하는 서비스에 관한 연구가 필요하다. 본 논문에서는 유비쿼터스 컴퓨팅 환경에서 서비스의 상태 정보를 포함하는 서비스 객체의 OSGi 프레임워크간 자유로운 이동을 위한 OSGi 상의 서비스 관리 시스템을 개발한다. 개발하는 시스템은 OSGi 상의 서비스의 이동을 관리할 수 있는 OSGi 프레임워크의 번들 형태로서 구현되며, OSGi 프레임워크간의 서비스의 자유로운 이동을 지원한다. 따라서 특정 컴포넌트나 사용자를 위한 서비스, 디바이스 드라이버 등 다양한 형태의 개체들의 이동성을 지원할 수 있다.

Ⅱ. 관련 연구

OSGi는 네트워크 환경에서 서비스를 전달하고 배치, 관리하기 위한 표준 명세를 정의하는 비영리 단체이다. 초기에는 홈서비스 게이트웨이에 집중되었지만 최근에는 특정 네트워크 환경에 국한하지 않고 유비쿼터스 환경까지 확장해 가고 있다. 따라서 다양한 임베디드 디바이스와 이를 이용하는 사용자를 위한 서비스 게이트웨이 구축을 목표로 하고 있다[7].

(그림 1)의 OSGi 서비스 플랫폼은 OSGi 프레임워크와 표준 서비스들로 구성된다. OSGi 프레임워크는 이러한 서비스들을 위한 실행 환경을 의미하며 최소한의 컴포넌트 모델, 컴포넌트를 위한 관리 서비스, 서비스 레지스트리 등을 포함한다. OSGi 프레임워크는 번들이라고 불리는 OSGi 컴포넌트를 설치하고, 서비스 등록 및 실행을 위한 프로그래밍 모델을 지원한다.

또한 프레임워크 자신도 번들로 표현되며, 이러한 번들을 시스템 번들이라고 한다. 번들을 위한 호스팅 환경을 제공하는 OSGi 프레임워크의 역할을 정리하면 다음과 같다.

- 번들의 라이프 사이클 관리
- 번들들 간의 상호 독립성 해결
- 서비스들에 관한 레지스트리 관리
- 번들의 상태 변화, 서비스 등록 및 해제, 프레임워크의 동작 등에 관한 이벤트 처리

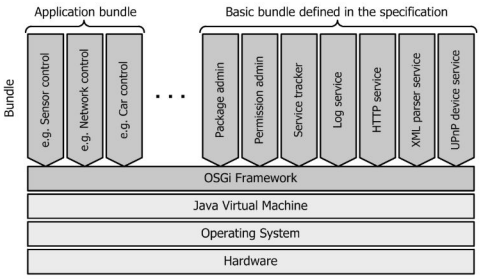

그림 1. OSGi 서비스 플랫폼
Fig 1. OSGi Service Platform

번들들은 서비스 레지스트리에 등록된 서비스를 이용하는 서비스 집합인 동시에 컴포넌트 단위이다. 서비스 구현은 물리적이며 논리적인 단위인 번들을 통해서 프레임워크에 전달되고 배치되어진다. 물리적으로, 번들은 코드, 리소스, Manifest 파일 등을 포함하는 Java Archive (JAR) 파일 형태로 배포된다. 특히, 번들의 manifest 파일은 프레임워크에게 번들 클래스의 실행 경로를 알려주며, 다른 번들들과 공유하게 될 자바 패키지를 선언한다. 또한, 번들의 Activator 클래스에 대한 정보를 갖고 있다. 논리적으로, 하나의 번들은, 운영체제에서의 하나의 프로세스와 유사한 개념으로, 어떤 서비스를 제공하는 서비스 제공자(provider)이거나 실행 시간에 프레임워크 내에 어떤 서비스를 이용하려는 서비스 요청자(requester)이다. OSGi 프레임워크는 번들의 라이프 사이클을 관리하는 메커니즘을 제공하는데, 만약 번들이 인스톨되어 프레임워크 상에서 실행되면 자신의 서비스를 제공할 수 있게 된다. 또한 프레임워크 상에서 실행 중인 다른 서비스를 발견하여 이용할 수 있으며, 프레임워크의 서비스 레지스트리를 통해서 다른 번들들의 서비스들과 묶여질 수 있다. 번들은 OSGi 프레임워크의 서비스 레지스트리에 서비스를 등록할 때, 서비스에 관한 속성-값(attribute-value) 쌍의 형태로 그 서비스의 특성을 함께 저장하는데, 이러한 서비스 특성은 동일한 서비스를 제공하는 여러 서비스 제공자들 사이에서 구분될 수 있도록 사용된다.

OSGi 서비스는 인터페이스와 서비스 구현으로 구성되어 있다. 서비스들 사이의 접근은 인터페이스를 통해 이루어지며 각 서비스는 서비스 레지스트리에 객체 단위로 등록된다. OSGi 프레임워크 상의 서비스를 위한 어플리케이션은 번들을 개발함으로써 이루어진다. 번들 개발을 위한 프로그래밍 모델은 컴포넌트 기반이며, 프레임워크는 어플리케이션 개발자에게 프레임워크와 서비스 간의 인터페이스만을 정의함으로써 일관된 프로그래밍 모델을 제공한다. 실제 구현은 번들 개발자의 몫이며, 서비스 선택과 상호 작용에 관한 결정은 서비스 발견과 실행 시간에 동적으로 실현된다. 이러한 서비스 정의와 구현의 분리는 서로 다른 서비스 제공자들로부터 제공된 서비스들 간의 상호운용을 가능케 한다.

OSGi 프레임워크에서 제공하는 번들의 라이프 사이클 관리와 클래스 로딩에 대한 지원은 다른 번들이나 외부로부터의 코드를 이용할 수 있도록 설계된 구조이다. 이러한 특징으로 인해 번들의 생성부터 소멸까지의 상태 정보가 관리되며, 자바 클래스 파일로부터 새로운 인스턴스를 생성할 수 있는 기능을 제공할 수 있다[6][7].

III. 시스템 설계 및 구현

이 장에서는 OSGi 프레임워크 상에서의 서비스 이동을 관리할 수 있는 OSGi 상의 서비스 이동 관리 시스템을 설계한다. 본 논문에서 제안한 서비스 이동 관리 시스템은 OSGi 표준 스펙에 맞게 번들의 형태로 구현되어, 사용자를 위한 서비스나 디바이스 드라이버 등 다양한 형태의 서비스 개체들의 이동성을 지원한다.

3.1 OSGi 상의 서비스 이동 관리 시스템 개요

서비스 이동 관리 시스템은 이동성을 관리하기 위한 Mobility Interface, 직렬화 및 역직렬화를 처리하는 Service Serializer, SOAP 메시지 송/수신을 위한 SOAP Manager로 구성되며, 전체 시스템은 (그림 2)와 같다.

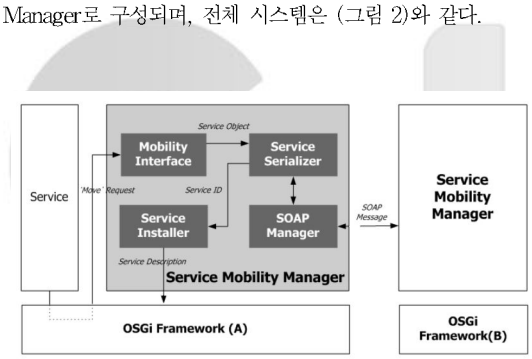

그림 2. OSGi 상의 서비스 이동 관리 시스템
Fig 2. Service mobility management system on OSGi

서비스 이동을 위해서 Service Mobility Manager는 해당 번들의 이동 요청을 받으면 Mobility Interface를 통해서 서비스의 라이프 사이클을 관리한다. 우선 현재 수행되고 있던 서비스와 관련된 모든 작업을 중지시키거나 현재의 서비스 상태를 저장하는 것과 같은, 서비스가 이동하기 전에 수행될 필요가 있는 모든 작업을 할 수 있도록 Service Mobility Manager는 "MOVE" 요청을 해당 서비스에 전달한다. 이러한 과정을 처리하기 위해서 번들은 Mobility Interface를 상속받아 구현되도록 한다. Mobility Interface는 OSGi 프레임워크들간의 서비스 이동을 위해서, 어떤 프레임워크에서 실행되고 있던 서비스의 이동전에

수행되어야 하는 작업들과 서비스의 이동 후에 수행되어야 하는 작업들을 구현해 주는 두 개의 메소드로 구성이 된다. 다음은 프레임워크들간의 이동성이 요구되는 서비스가 구현해야 하는 Mobility Interface의 메소드들이다.

```
interface Mobility {
  // called by Framework before moving
  bool beforeMoving() {};
  // called by Framework after moving
  bool after Moving() {};
}
```

Service Mobility Manager는 서비스가 이동하기 전의 상태 정보를 Service Serializer에 의해 XML로 직렬화한다. 이렇게 직렬화된 XML은 OSGi 프레임워크로부터 얻은 해당 서비스의 등록 정보와 번들의 클래스 위치 정보인 URL 등과 함께 SOAP 메시지로 변환된 후, 네트워크를 통해 전송된다. 이러한 작업을 수행하는 SOAP Manager 컴포넌트는 위와 같이 변환된 SOAP 객체를 전송할 뿐만 아니라 다른 프레임워크로부터 SOAP 객체를 전송 받는 역할을 담당한다.

SOAP 메시지를 전송 받은 OSGi 프레임워크는 그 SOAP 메시지에 포함되어 있는 해당 번들의 위치 정보인 URL를 이용하여 필요한 클래스를 다운로드 한다. 다음으로, Service Mobility Manager의 Service Installer는 해당 서비스의 번들을 다운로드하여 설치하는데, 이때에 서비스의 등록 정보와 직렬화된 XML 정보는 Service Serializer에 의해 역직렬화된 후 설치될 번들의 서비스 상태를 이동하기 전의 서비스 상태 정보로 복원할 수 있도록 이용된다.

3.2 서비스 이동 객체 생성

서비스 이동 관리 시스템은 서비스의 이동성을 지원하기 위해서 서비스를 이동 가능한 객체로 만든다. 다른 OSGi 프레임워크로 서비스를 이동시키기 위해서 Service Mobility Manager는 서비스의 상태를 XML 형태로 직렬화하여 전송하며, 이동된 서비스가 OSGi 프레임워크에서 실행되기 위해서는 전송 받은 XML 형태의 상태 정보를 역직렬화하여 원래의 서비스 상태를 복원시킨다. 이렇게 XML 형태로 변환된 서비스의 상태 정보는 서비스가 수행할 상태 정보를 포함한다. Service Mobility Manager는 서비스 객체를 Service Serializer에게 전달하고, Service Serializer는 리스트 1의 XML 스키마에 의해 서비스의 상태를 XML 형태로 직렬화한다.

[리스트 1] 서비스 직렬화를 위한 XML 스키마

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd=
        "http://www.w3.org/2001/XMLSchema"

targetNamespace="http://hci.inha.ac.kr/ema">
<xsd:complexType name="MObject">
<xsd:sequence>
<xsd:element name="name" type="xsd:string"
        minOccurs="1" maxOccurs="1"/>
<xsd:element name="status" type="xsd:int"
        minOccurs="1" maxOccurs="1"/>
<xsd:element ref="actionScheduler" minOccurs="1"
        maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>
<xsd:element name="actionScheduler">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="action" type="action"
        minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:complexType abstract="true" name="action">
</xsd:complexType>
</xsd:schema>
```

서비스 객체의 직렬화를 위해서 Castor 프로젝트(www.castor.org)의 Marshaller 클래스를 이용하며, 다음의 리스트 2는 서비스의 상태 정보를 저장하여 객체를 직렬화하는 코드의 일부분이다.

[리스트 2] 서비스의 객체 직렬화 코드

```
import java.io.*;
import org.exolab.castor.xml.Marshaller;
import org.exolab.castor.mapping.MappingException;
public String getXML(Serializable obj) throws
MappingException, IOException {
//전달 받은 객체를 XML로 직렬화하여 String 객체로 리턴
StringWriter sw = new StringWriter();
//Marchaller는 자바 객체를 XML로 직렬화
Marshaller marshaller = new Marshaller(sw);
marshaller.setMapping(mapping);
marshaller.marshal(obj);
return sw.toString();
.....
}
```

Service Serializer에 의해 생성된 XML 형태의 서비스의 상태 정보는 SOAP Manager에게 전달되어, SOAP 메시지로 변환된 후 다른 프레임워크로 전달된다. SOAP 메시지에는 서비스의 상태 정보 이외에도 서비스 레지스트리에 등록된 그 서비스의 기술 정보와 그 서비스가 위치하는 URL 정보도 함께 포함된다. 이것은 서비스가 이동하게 될 OSGi 프레임워크에서 해당 서비스의 클래스를 다운로드 받기 위해서 그 URL 정보를 필요로 하기 때문이다.

3.3 서비스의 이동

이동할 필요성이 있는 서비스는 다른 OSGi 프레임워크로 이동해야 하는 경우, Service Mobility Manager에게 "MOVE" 요청을 전달한다. 이렇게 전송을 필요로 하는 "MOVE" 요청에는 그 프레임워크의 URL 정보와 그 서비스의 ID를 포함한다. Service Mobility Manager는 "MOVE" 요청에 포함된 서비스의 ID를 이용해서 beforeMoving() 메소드를 호출한다. 이처럼 "MOVE" 요청을 받은 Service Mobility Manager는 현재 처리 중인 작업을 마친 후 리턴한다.

Service Mobility Manager는 해당 서비스의 상태가 이동 가능한 상태로 전환된 경우, 그 서비스의 객체를 Service Serializer에게 전달하여 서비스의 상태 정보를 XML 형태로 직렬화시킨 후, SOAP Manager를 통해 직렬화된 XML과 함께 서비스의 등록 정보 및 프레임워크의 URL 정보를 포함하여 SOAP 메시지를 생성한다. SOAP Manager는 이동하고자 하는 프레임워크를 확인하고 생성된 SOAP 메시지를 해당 프레임워크로 전송한다. 서비스의 이동이 성공했다면 현재 수행 중이던 서비스를 서비스 레지스트리에서 삭제한다. 알고리즘 1은 서비스 객체의 전송을 위한 알고리즘이다.

[알고리즘 1] 서비스 전송 알고리즘

```
SendingServiceObject(ServiceID)
Begin Proc
  ServiceRef = ServiceManager.ServiceFinder.
               GetServiceRef(ServiceID);
  ServiceDescription = ServiceManager.ServiceFinder.
                       GetServiceDes(ServiceID);
  res= ServiceRef.beforeMoving();
  If res is true then
    ServiceStatus = ServiceSerializer.serializer(res);
    SOAPMessage = SOAPService.makeSOAPMessage
      (URL,serviceStatus, ServiceDescription);
    sendMessage(TargetURL, SOAPMessage);
  End If
End Proc
```

3.4 서비스의 복원

이동되는 서비스를 전송 받는 프레임워크에서 SOAP 메시지를 기다리고 있던 SOAP Manager는 SOAP 객체 내에 포함된 클래스 위치 정보인 URL과 직렬화된 데이터를 수신하고 이를 Service Mobility Manager에게 전달한다. Service Mobility Manager는 전송 받은 객체를 역직렬화하기 전에 Service Installer를 통해 원격지로부터 번들을 다운로드 하여 설치하고, 설치가 성공된 후에 XML을 역직렬화하여 이동 전의 상태로 해당 서비스를 복구시킨다. 마지막으로 해당 서비스의 상태를 실행 가능 상태로 전환한 후 서비스 레지스트리에 등록한다. 알고리즘 2는 서비스의 이동 후에 서비스 객체를 복원하는 알고리즘이다.

[알고리즘 2] 서비스 복원 알고리즘

```
RestoreServiceObject(SOAPMessage)
Begin Proc
  URL = SOAPService.getAttribute(SOAPMessage, URL);
  ServiceDes = SOAPService.getAttribute
              (SOAPMessage, ServiceDes);
  ServiceStatus = SOAPService.getAttribute
              (SOAPMessage, ServiceStatus);
  ServiceClass = ServiceInstaller.getClass(URL);
  ServiceRef = ServiceInstaller.serviceInstalling
              (ServiceClass, ServiceDes);
  ServiceRef.afterMoving(ServiceDeserializer.
              deserializer(ServiceStatus);
End Proc
```

전송 받은 SOAP 메시지를 파싱하여 해당 서비스의 복원에 필요한 정보를 얻는다. 전송 받은 SOAP 메시지에는 XML 객체와 함께 해당 서비스의 등록 정보 및 다운로드 할 URL 정보가 포함된다. 다음의 리스트 3은 SOAP 메시지를 파싱하는 코드의 일부분이다.

[리스트 3] SOAP 메시지 파싱 부분 코드

```
import javax.xml.soap.*;
import javax.servlet.*;
public void doPost(HttpServletRequest req,
                   HttpServletResponse resp)
        throws ServletException, IOException {
String obj, String url, String desc;
MimeHeaders headers = getHeaders(req);
InputStream is = req.getInputStream();
//전송받은 데이터로 부터 SOAPMessage를 생성한다.
SOAPMessage message =
msgFactory.createMessage
                   (headers, is);
SOAPBody body = message.getSOAPBody();
Iterator iter = body.getChildElements
          (envelope.createName("service", "ns1",
          "http://hci.inha.ac.kr/service/ema"));
if (iter.hasNext()) {
  /*객체 직렬화와 URL정보, Description정보를 읽는다.*/
  SOAPElement element = (SOAPElement) iter.next();
  Iterator objElements =
        element.getChildElements(envelope.
        createName("obj"));
  Iterator urlElements =
        element.getChildElements(envelope.
        createName("url"));
  Iterator descElements =
        element.getChildElements(envelope.
        createName("desc"));
  SOAPElement objElement =
        (SOAPElement)objElements.next();
  SOAPElement urlElement =
        (SOAPElement)urlElements.next();
  SOAPElement descElement =
        (SOAPElement)descElements.next();
  obj = XMLEncoder.decode
          (objElement.getValue());
  url = urlElement.getValue();
  desc = XMLEncoder.decode
          (descElement.getValue());
  }
}
```

SOAP 메시지를 파싱하여 얻어진 정보 중 클래스의 다운로드에 이용되는 URL 정보를 Service Installer에게 전달하여 해당 서비스의 클래스 파일을 다운로드하여 OSGi 프레임워크에 설치한다. 이 때, BundleContext의 installBundle 메소드를 이용하여 설치한다. 리스트 4는 해당 서비스의 클래스를 다운로드하여 서비스를 설치하기 위한 코드의 일부분이다.

[리스트 4] 서비스 설치 부분 코드

```
import java.net.*;
import java.io.*;
import org.osgi.framework.BundleContext;
import org.osgi.framework.BundleException;
import org.osgi.framework.Bundle;

public Bundle install(String url) throws
  IOException, ProtocolException,
BundleException {
  BundleContext bc = MAManagerBundleActivator.bc;
```

```
.....
// 소켓을 생성하여 번들 파일이 있는 URL로HTTP연결을 시도
Socket socket = new Socket();
InetSocketAddress sockAddr =
                    new InetSocketAddress(host, port);
socket.connect(sockAddr, Http.TIMEOUT);
.....
String reqStr = getRequestHeader(url);
byte[] reqBytes = reqStr.getBytes();
req.write(reqBytes);
req.flush();
// 리턴받은 번들 파일을 byte[]에 저장
byte [] content = readPlainContent(in);
socket.close();
// 전달받은 파일을 번들 저장소에 파일로 저장
FileOuputStream fout =  new FileOutputStream
                    (new File(BUNDLE_DIR,filename));
fout.write(content);
fout.close();

//번들을 프레임 워크에 인스톨한다.
Bundle bundle = bc.installBundle
            (url, new ByteArrayInputStream(content));
return bundle;
}
```

설치된 서비스는 이동하기 전의 상태로 복원되어야 한다. XML 형태로 변화되어 SOAP 메시지에 포함된 서비스의 상태 정보는 역직렬화 과정을 거쳐서 서비스의 복원에 사용된다. 이 과정에서 직렬화 과정과 마찬가지로 Castor 프로젝트의 Unmarshaller 클래스를 이용하여 서비스의 역직렬화 과정을 수행한다. 다음의 리스트 5는 서비스 객체의 역직렬화하는 코드의 일부분이다.

[리스트 5] 서비스 역직렬화 부분 코드

```
import java.io.*;
import org.exolab.castor.xml.Unmarshaller;
import org.exolab.castor.mapping.MappingException;
public Mobject getMObject(String objXml)
        throws MappingException, IOException {
/* 전달받은 XML 문자열을 MObject의 인스턴스로
Deserialize 한다. */
  Unmarshaller unmar =
                    new Unmarshaller(mapping);
  MObject obj = (MObject)unmar.unmarshal
            (new java.io.StringReader(objXml));
  return obj;
}
```

Ⅳ. 실험 및 평가

이 장에서는 본 논문에서 제안한 OSGi 상의 서비스 이동 관리 시스템의 평가를 위해서 OSGi 프레임워크 상의 음악 서비스를 개발한다. Knopflerfish 1.3.3 상에 서비스 이동 관리자를 설치/운용함으로써, OSGi R3 표준 스펙을 준수하고 관리자의 기능이 적합하게 동작되는지를 실험한다.

4.1 실험 환경 및 시나리오

본 논문에서 제안하는 서비스 이동 관리 시스템을 실험하기 위해서 음악 서비스를 구현한다. 이 음악 서비스는 OSGi 프레임워크 상에서 MP3 파일을 재생하는 서비스이며, MP3 파일과 파일 목록을 포함하는 번들로 설치된다. 이러한 음악 서비스의 이동성을 실험하기 위해서 PDA 단말기와 PC에 구현한 서비스 이동 관리 번들을 OSGi 프레임워크에 설치한다. 음악 서비스의 이동에 관한 예로서 사용자가 PDA 단말기로부터 음악 서비스를 제공받아 음악을 들으면서 댁내에 들어온 경우를 생각해 볼 수 있다. 이때 사용자는 이어폰으로 듣던 음악을 PC 스피커를 통해 듣기를 원할 수 있다.

본 논문의 실험을 위한 음악 서비스는 MP3 음악 파일을 플레이하여 사용자에게 음악을 제공하는 OSGi 프레임워크

상의 서비스로서 MP3 파일과 파일 목록을 포함한다. 이러한 음악 서비스의 이동에 따른 서비스 이동성을 실험하기 위해서 PDA 단말기와 PC에 OSGi 프레임워크인 Knopflerfish 1.3.3과 함께 본 논문에서 구현한 서비스 이동 관리 번들을 함께 설치한다. 앞서 언급한 것처럼, 음악 서비스에 관한 서비스 이동의 한 예로서 사용자가 PDA 단말기로부터 음악 서비스를 제공받아 음악을 들으면서 댁내로 들어온 경우를 생각해 볼 수 있다. 이때, 사용자에게 연속적인 서비스를 제공하기 위해서 음악 서비스는 PDA 단말기에서 PC 상의 OSGi 프레임워크로 이동되어야 할 필요성이 있다. 그리고 PDA 단말기에서 연주 중이던 MP3 파일의 목록과 함께 사용자가 듣고 있던 음악을 이어서 들을 수 있도록 연속적으로 제공해야 한다.

본 논문에서는 윈도우 XP 환경에서의 OSGi 프레임워크인Knopflerfish 1.3.3을 사용하였고, Eclipse 3.0과 Knopflerfish Eclipse Plug-in 0.7을 이용하여 실험에 쓰이는 음악 서비스를 개발한다. 그리고 이 실험에 사용한 PDA단말기는 HP iPAQ Pocket PC h2210 모델로서, 400MHz 인텔 XScale 프로세서와 64MB 램으로 구성된 사양을 갖는다. 그리고 OSGi 프레임워크인 Knopflerfish 1.3.3은 IBM J9 컴파일러를 이용하여 PDA 단말기에 포팅한다.

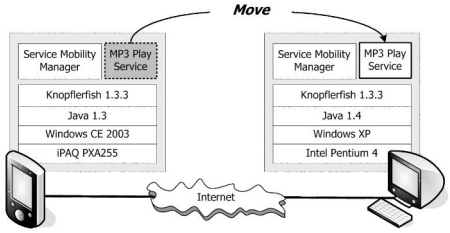

그림 3. 음악 서비스의 이동
Fig 3. the mobility of a music service

4.2 음악 서비스 개발

OSGi 상의 서비스 이동 관리 시스템은 OSGi 프레임워크간의 서비스의 이동성을 지원하기위해서는 서비스 객체를 이동시킨다. 서비스가 이동하기 전에 서비스의 상태 정보는 직렬화 과정을 거쳐서 XML 형태로 생성되고, 네트워크 전송을 위한 SOAP 메시지로 다시 변환된다. (그림 4)는 서비스 이동 전의 시퀀스 다이어그램이다. PDA에서 사용자에게 제공되고 있던 음악 서비스가PC 상의 OSGi 프레임워크로 이동해야 하는 경우, Service Mobility Manager에게 "MOVE" 요청을 전달한다.

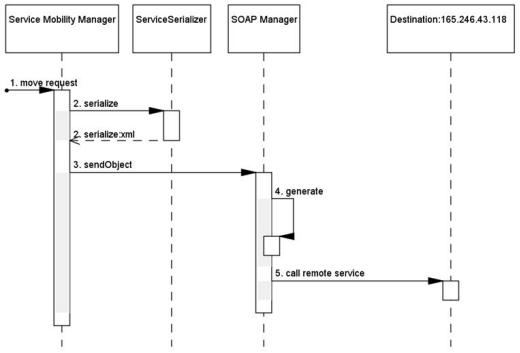

그림 4. 서비스 이동 전의 시퀀스 다이어그램
Fig 4. the sequence diagram before moving service

음악 서비스가 수행하고 있는 현재의 상태 정보를 Service Serializer를 통해서 XML로 직렬화한 형태는 리스트 7과 같으며, MP3 파일 재생 목록과 현재 재생하는 MP3 파일의 offset 정보를 포함한다.

[리스트 7] 음악 서비스를 직렬화한 XML

```
<mplayer>
<description> mp3 music player </description>
<status> 3 </status>
<option>
    <repeat> false </repeat>
</option>
<current>
    <offset> 132 </offset>
    <id> 2 </id></current>
<item id="1">
    <title> pop1 <title>
    <location> music/first.mp3 </location>
</item>
<item id="2">
    <title> dance1 <title>
    <location> music/second.mp3 </location>
</item>
<item id="3">
    <title> classic1 <title>
    <location> music/third.mp3 </location>
</item>
</mplayer>
```

XML로 직렬화된 데이터는 네트워크 전송을 위해서 SOAP Service에 의해서 SOAP 메시지를 생성하고 PC 상의 OSGi 프레임워크로 전송한다.

(그림 5)는 서비스가 이동하기 전에 PDA에서 음악 서비스가 OSGi 프레임워크에서 동작되는 화면이다. [MPlayer] 번들이 인스톨 과정을 거쳐 시작되고, 서비스가 등록되면서 프레임워크는 음악 서비스를 시작한다. first.mp3 파일 재생을 마친 후, 두 번째 곡인 second.mp3 파일이 재생되는 도중에 MPlayer 서비스의 이동을 요청 받은 [Mobile Manager]는 MPlayer 서비스에게 "MOVE" 이벤트를 전달한다. "MOVE" 이벤트를 받은 MPlayer 서비스는 현재의 상태 정보를 저장하고 서비스를 중지한다. [Mobile Manager]는 MPlayer 서비스를 직렬화하여 지정된 URL의 PC로 전송한다.

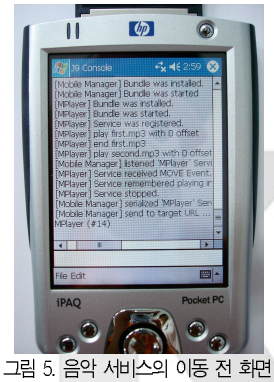

그림 5. 음악 서비스의 이동 전 화면
Fig 5. the shot of the music service's mobility

(그림 6)은 음악 서비스 이동 후의 시퀀스 다이어그램이다. 서비스의 복원을 위해서 전송된 SOAP 객체로부터 그 서비스에 해당하는 클래스의 URL 정보와 함께 서비스의 상태 정보 및 등록 정보를 얻어 낸다. Bundle Installer에 의해 음악 서비스에 해당하는 번들의 클래스 파일을 다운로드 한 후 OSGi 프레임워크에 설치하고, Service Deserializer는 전송 받은 서비스의 상태 정보를 역직렬화 한다. 역직렬화 과정을 거쳐 얻어진 서비스의 상태 정보인 재생 목록과 재생하고 있던 MP3 파일의 offset 정보로 이동하기 전의 서비스 상태로 복원시킨다.

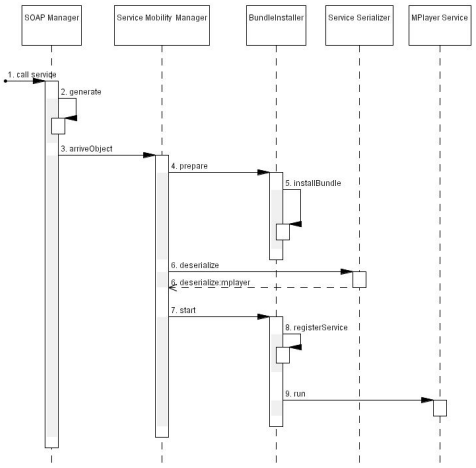

그림 6. 서비스 이동 후의 시퀀스 다이어그램
Fig 6. the sequence diagram after moving service

(그림 7)은 서비스가 이동한 후에 PC에서 음악 서비스가 동작하는 화면이다. PDA로부터 MPlayer 서비스가 전송된 후, [Mobile Manager]는 [MPlayer] 번들을 다운로드 하여 설치한다. MPlayer 서비스를 전송 받은 [Mobile Manager]는 역직렬화 과정을 거쳐 MPlayer 번들을 인스톨한 후, 번들을 시작한다. 시작된 [MPlayer]가 서비스를 등록하면, 이동 전에 중지한 132 offset부터 second.mp3 파일을 재생한다.

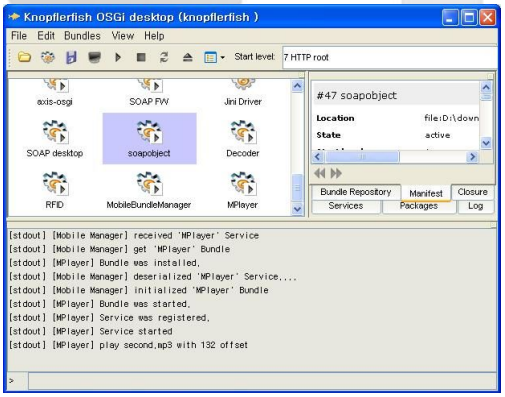

그림 7. 음악 서비스의 이동 후 화면
Fig 7. the shot of the music service's mobility

4.3 평가

본 논문에서 제안한 OSGi 상의 서비스 이동 관리 시스템은 OSGi 프레임워크를 구현한 오픈 소스인 Knopflerfish

1.3.3 상에 설치되어 무리없이 작동하였다. 그리고 이를 통해 구현한 시스템이 OSGi 표준 스펙에 맞게 구현되었는지를 확인하였다.

PDA 상의 OSGi 프레임워크에서 MP3 파일을 플레이하는 서비스가 자신의 서비스 상태를 유지한 채 PC 상의 OSGi 프레임워크로 이동하여 연속된 음악 서비스를 제공해 주었다. 이 실험을 통해서 본 논문에서 제안한

OSGi 상의 서비스 이동 관리 시스템은 서비스의 현재 상태 정보를 함께 이동시킴으로써, 복수 개의 OSGi 프레임워크가 존재하는 유비쿼터스 컴퓨팅 공간에서의 서비스에 대한 이동성을 지원할 수 있음을 보였다.

V. 결론

홈네트워크 등 유비쿼터스 컴퓨팅 환경을 구성하는 OSGi 프레임워크들간의 개체의 이동을 보장하기 위해서는 서비스 이동에 대한 관리가 필요하다. 본 논문에서는 OSGi 프레임워크 상에서 이동이 요구되는 서비스에 관한 번들 형태의 관리 시스템을 제안하였고, 이를 위해서 OSGi 스펙에 맞는 서비스 이동 관리 시스템을 설계하고 구현하였다.

설계한 서비스 이동 관리 시스템은 번들 형태로 구현되어 OSGi 프레임워크에서 동작된다. 서비스 이동 관리 시스템은 OSGi 프레임워크들간의 서비스의 이동을 지원하기 위해서 서비스 객체의 직렬화 과정을 통해 얻어진 서비스의 상태 정보를 서비스의 등록 정보와 함께 SOAP 메시지 방식으로 전송한다. 다른 OSGi 프레임워크로 이동한 서비스는 이전의 서비스의 상태 정보와 등록 정보를 전송받아 역직렬화 과정을 거쳐서 이동전에 수행하고 있던 서비스의 상태로 복원하여 그 서비스가 재시작 될 수 있도록 한다. 이처럼 이동성을 갖는 서비스에 대한 동적 관리를 가능하게 함으로써 서비스의 이동성을 보다 효과적으로 지원할 수 있었다.

향후 지능형 서비스를 제공하기 위해서 상황 정보를 고려한 OSGi 상의 상황 인식 프레임워크에 대한 연구와 서비스의 이동에 따른 보안에 관한 연구가 이루어져야할 것이다.

참고문헌

[1] Open Services Gateway Initiative: http://www.osgi.org.

[2] D. Marples and P. Kriens, "The Open Services Gateway Initiative: An Introductory Overview," IEEE Communications Magazine, Vol. 39, No. 12, pp. 110-114, 2001.

[3] C. Lee, D. Nordstedt, and S. Helal, "Enabling Smart Spaces with OSGi," IEEE Pervasive Computing, Vol. 2, Issue 3, pp. 89-94, 2003.

[4] P. Dobrev, D. Famolari, C. Kurzke, and B. A. Miller, "Device and Service Discovery in Home Networks with OSGi," IEEE Communications Magazine, Vol. 40, Issue 8, pp. 86-92, 2002.

[5] L. Gong, "A Software Architecture for Open Service Gateways," IEEE Internet Computing, Vol. 5, Issue 1, pp. 64-70, 2001.

[6] R. S. Hall and H. Cervantes, "Challenges in Building Service-Oriented Applications for OSGi," IEEE Communications Magazine, Vol. 42, Issue 5, pp. 144-149, 2004.

[7] R. S. Hall and H. Cervantes, "An OSGi Implementation and Experience Report," First IEEE Consumer Communications and Networking Conference, pp. 394-399, 2004.

[8] 윤정섭, "이동환경을 위한 서비스 에이전트 시스템," 한국컴퓨터정보학회논문지, 2004. 9.

[9] 김효남, 박용, "유비쿼터스 컴퓨팅 환경에서 상황인식 미들웨어 설계," 한국컴퓨터정보학회논문지, 2005. 11.

저 자 소 개

이 승 근

1996년 2월 인하대학교 전자계산공학과

1998년 2월 인하대학교 전자계산공학과 공학석사

2000년 2월 인하대학교 컴퓨터정보공학과 박사수료

2000년~2005년 (주)하이켐텍 기술연구소 책임연구원