DOI: 10.3745/KIPSTD.2009.16D.6.935

OSGi 서비스의 효과적인 외부연동을 위한 자바원격서비스호출 프레임워크

최 재 현[†] · 박 제 원[†] · 이 남 용[††]

요 약

최근 다양한 네트워크 기술들을 통합하고 하나의 단일화된 유비쿼터스 네트워크를 형성하여 사용자에게 효과적으로 서비스를 제공하려는 기술적 시도들이 많이 이루어지고 있다. 특히, OSGi Alliance에서 제안된 OSGi는 자바기술을 기반으로 서비스의 동적 재구성을 지원함으로써 다양한 미들웨어 환경 사이에서 서비스간의 상호연동 및 유비쿼터스 네트워크 구성을 위한 핵심 인프라로서 많이 활용되고 있다. 그러나 현재 OSGi 표준에서는 내부에 등록된 서비스를 외부환경에서 공개(publish)하고 호출 및 연동하기 위한 메커니즘을 정의하고 있지 않아, 분산 환경에서 효율적인 서비스의 동적 재구성 및 협업을 달성하는 데에 다소 어려움이 있다. 따라서 본 논문에서는 OSGi 환경에서 등록된 서비스들의 효율적인 서비스의 공개 및 연동을 위한 JARSIO(JAva Remote Service Invocation for OSGi) 프레임워크를 제안한다. JARSIO는 TCP/IP 통신을 기반으로, OSGi 환경에서 동적으로 재구성되는 다양한 서비스들을 원격지에서 자유롭게 호출 및 연동 가능할 수 있도록 지원한다.

키워드 : OSGi, 자바, 유비쿼터스, 원격호출, 서비스

JARSIO : Java Remote Service Invocation for OSGI Framework to Enhance Inter-Operations of Services on OSGi

Choi Jae-hyun[†] · Park Jae-won[†] · Lee Nam-yong[††]

ABSTRACT

Recently, many researches focus on Ubiquitous Network which comprised various networks for effective service provision. In particular, OSGi proposed by OSGi Alliance is preferred for core infrastructure to establish Ubiquitous Network as it supports integration and inter-operation among various service environments, and dynamic configuration of services. However, OSGi is limited to be used only within local service framework, since OSGi specification does not have any considerations for inter-operations between internal and external services. Thus, in this paper we propose JARSIO(JAva Remote Service Invocation for OSGi) framework which enables the inter-operations of dynamic internal OSGi services and other external services. The proposed framework is based on the TCP/IP protocol, and provides effective mechanisms for the inter-operations of the services.

Keywords : OSGi, Java, Ubiquitous, Remote Invocation, Service

1. 서 론

최근 시간과 장소에 구애받지 않고 언제 어디서나 네트워크에 접속할 수 있는 유비쿼터스 환경에 대한 연구가 활발히 진행되고 있다. 이러한 유비쿼터스 환경은 시간, 장소를 초월한 통신환경을 의미하며, 하나의 광대한 단일 네트워크를 형성하여 사용자에게 효과적으로 서비스를 제공하는 것을 목적으로 하고 있다. 그러나 이러한 광대한 단일 네트워크를 실현하기 위해서는 실시간으로 서비스의 설치 및 제거가 이루어지고, 이들 서로간의 협업 및 연동이 편리하게 이루어질 수 있어야 한다. 이러한 동적협업네트워크의 구성을 위해 자바진영을 주축으로 한 OSGi Allience에서 제안한 것이 OSGi(Open Service Gateway Initiative)[1]이다. 이것은 자바기술을 중심으로 다양한 미들웨어 환경 사이에서 서비스의 상호연동 및 유비쿼터스 환경 구축을 위한 핵심 인프라 구축을 위한 기술적 기반을 제공한다[2]. 즉, OSGi는 텔레비전, 냉장고, 조명기기, 계량기 등 다양한 가전제품과 설비 등이 광대한 단일 네트워크 망에 연결될 수 있는 관문역할을 수행함으로써 유비쿼터스 환경 구축을 위한 핵심 기반

※ 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음.
† 준 회 원 : 숭실대학교 컴퓨터학과 박사과정
†† 정 회 원 : 숭실대학교 컴퓨터학과 교수
논문접수 : 2009년 3월 6일
수 정 일 : 1차 2009년 6월 18일, 2차 2009년 7월 29일
심사완료 : 2009년 7월 29일

을 제공해주는 것이다. OSGi는 이를 위해 번들(Bundle)로 일컬어지는 컴포넌트를 기반으로 한 모듈형 구조를 채택하고 있으며, 이를 통해 서비스지향아키텍처(SOA)[3]를 실현함으로써 다양한 서비스들이 동적으로 설치 및 제거되고, 유기적으로 협업할 수 있도록 지원한다.[4-6].

그러나 이러한 OSGi는 내부에 정의된 서비스들 간의 동적 재구성이나 협업만 지원할 뿐, 내부에 정의된 서비스들이 외부의 서비스들과 연계나 협업은 지원하고 있지 않다. 즉, OSGi 환경 내에서는 다양한 서비스들이 등록/설치, 협업/연동될 수 있으나, 외부에 존재하는 서비스들은 OSGi 환경 내에 존재하는 서비스들과 효과적으로 협업 및 연동할 수 없는 문제점이 있다. 이것은 OSGi가 다양한 서비스들이 유기적으로 협업 및 연동되는 유비쿼터스와 같은 분산환경에서 보다 유용하게 활용되는데 있어 다소 걸림돌이 된다.

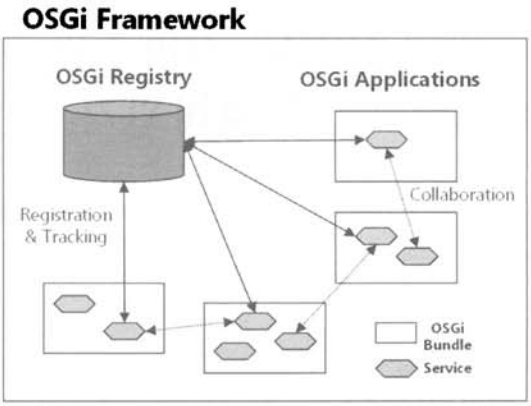
따라서 본 논문에서는 OSGi 환경 내부에 존재하는 다양한 서비스들을 외부에 공개하고, 이를 기반으로 다양한 내외부 서비스들이 효과적으로 협업할 수 있도록 지원하는 자바원격서비스 프레임워크(JARSIO: Java Remote Service Invocation for OSGi)를 제안한다. 이러한 자바원격서비스 프레임워크는 OSGi환경 내부에 등록된 서비스 객체를 서비스 속성 설정을 통해 외부에 공개하고, 외부에서는 이러한 객체에 대한 원격 레퍼런스를 얻을 수 있도록 하여 서비스를 손쉽게 연동할 수 있도록 지원한다.

2. OSGi (Open Service Gateway Initiative)

2.1 OSGi 플랫폼

OSGi는 서로 다른 환경에 있는 다양한 장치들을 효과적으로 통합하고 미들웨어의 상호작용 및 서비스 협업을 위해 만들어진 일종의 게이트웨이이다[1]. 즉, 다양한 서비스들은 OSGi를 통해서 서로 통신할 수 있으며, 이를 통해 다양한 서비스를 통합하고 유기적인 협업환경을 구축할 수 있다. 이러한 OSGi는 초기에는 홈서비스 게이트웨이 분야에서 주로 활용되었지만, 최근에는 동적 서비스 구성 및 유연한 환경을 기반으로 다양한 애플리케이션 플랫폼으로도 활용되고 있다. 이러한 OSGi를 기반으로 실제적인 서비스를 제공하기 위한 구현체를 OSGi 플랫폼이라 하며, OSGi 프레임워크와 OSGi 서비스로 구성된다. 이 중, OSGi 프레임워크는 OSGi 서비스들에 대한 배치 및 실행을 담당하며, OSGi 서비스는 실제적인 서비스의 제공을 담당한다. 특히, OSGi 플랫폼의 가장 큰 강점은 다양한 서비스나 애플리케이션이 다른 컴포넌트나 플랫폼의 재시작 없이 실시간으로 설치, 시작, 정치, 업데이트, 제거를 할 수 있도록 지원한다는 점이다. 이것은 OSGi가 서비스들이 실시간으로 추가/제거/실행되는 유비쿼터스 환경의 핵심기반으로서, 다른 한편으로는 서비스 지향 아키텍처를 기반으로 한 보다 효율적인 애플리케이션 플랫폼 실현을 위한 핵심기반으로서 효과적으로 활용될 수 있도록 해준다.

OSGi는 이러한 유연한 서비스 환경을 구축하기 위해 번

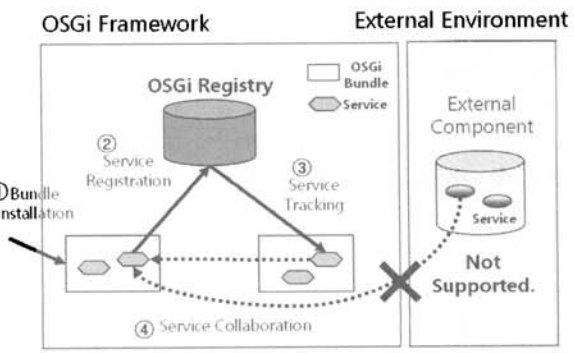


(그림 1) OSGi 프레임워크

들이라고 불리는 OSGi 컴포넌트를 기반으로 서비스를 등록 및 설치하고 실행하는 프로그래밍 모델을 채택하고 있다. 번들은 다양한 서비스를 등록 및 발견하고 사용하기 위해 OSGi 플랫폼에 설치되는 컴포넌트 단위인 동시에 일종의 서비스 집합으로, 서비스의 구현체 및 서비스를 표현하기 위한 메니페스트(Manifest)정보로 구성된다. 번들은 설치, 제거, 갱신, 정지, 실행 등의 라이프사이클을 가지며, 하나의 번들에는 다수 개의 서비스가 포함될 수 있다.

2.2 OSGi 플랫폼 상의 서비스

OSGi 번들은 내부에 다수개의 서비스를 포함할 수 있으며, 이러한 서비스는 OSGi 플랫폼 상의 다른 서비스들에 의해 사용될 수 있다. 번들은 이를 위해 실행 시에 내부에 구현되어 있는 서비스를 OSGi 레지스트리에 등록하여야 하며, 해당 서비스를 사용하고자 하는 서비스는 OSGi 레지스트리에서 등록된 서비스 객체에 대한 참조를 얻음으로써 해당 서비스와 협력할 수 있게 된다. 그러나 OSGi 플랫폼은 동일한 OSGi 플랫폼 내부의 서비스들 간의 협력을 지원할 뿐, 외부의 플랫폼 또는 시스템과 내부에 존재하는 서비스 사이의 협력은 지원하지 않는다. OSGi 플랫폼 상에 HTTP를 기반으로 서블릿 기능을 제공하는 서비스는 존재하지만, 이것은 어디까지나 HTTP를 통한 서비스의 제어 및 모니터링을 목적으로 하고 있는 것으로, 코드레벨에서의 협력을 통한



(그림 2) OSGi 프레임워크상의 서비스의 사용 및 한계

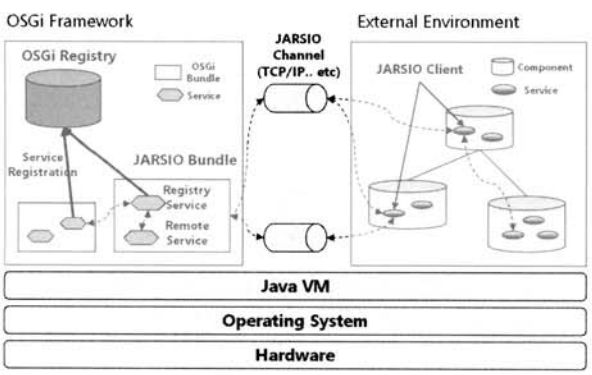
다양한 서비스 연계(상황인지, 지능형 서비스 등)를 위한 서비스는 존재하지 않는다.

이러한 점은 OSGi가 제공하는 동적 서비스의 구성 및 운영의 범위를 제약하는 것으로서, 다양한 서비스가 존재하고 서로 유기적으로 협업하는 OSGi기반 유비쿼터스 환경을 구축하는데 다소 걸림돌이 된다. 따라서 본 논문에서는 OSGi의 동적인 측면을 저해하지 않는 범위 내에서 외부 플랫폼 또는 시스템상의 서비스들과 유기적으로 협업할 수 있는 엔진의 제시를 통해, OSGi 플랫폼을 보다 유비쿼터스 환경에 적합한 환경으로 활용할 수 있도록 한다.

3. OSGi 환경에서의 자바원격서비스호출 (JARSIO:Java Remote Service Invocation for OSGi)

3.1 JARSIO 개요

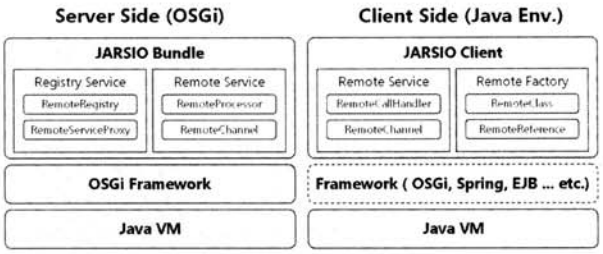
JARSIO는 OSGi 플랫폼 내에 존재하는 서비스를 외부 플랫폼이나 시스템에서 동적으로 원격호출 및 연동하기 위한 프레임워크이다. 즉, OSGi 환경내의 서비스는 JARSIO 프레임워크를 통해서 외부에 공개될 수 있으며, 공개된 서비스는 외부에서 자유롭게 호출하거나 연동할 수 있다. 이를 위해 JARSIO 프레임워크 역시, OSGi 플랫폼에서 동작하기 위한 번들형태로 구현되어 운용되며, 내부적으로 다양한 서비스들에 대한 원격 프록시를 등록하고 관리하기 위한 레지스트리 서비스와, 외부에서 OSGi 내부의 서비스들에 대한 호출을 실제적으로 처리하기 위한 원격서비스로 구성된다. JARSIO는 내부의 서비스들과 외부의 서비스간의 통신 채널을 추상화하며, 이를 통해 JARSIO 메시지를 주고받음으로써 적절하게 연동된다.



(그림 3) JARSIO의 개요

3.2 JARSIO 아키텍처

(그림 4)는 JARSIO의 아키텍처를 보여주고 있다. 서버측은 OSGi 플랫폼 상이 되며, 여기에서는 외부에 제공하고자 하는 서비스를 등록 및 관리하기 위한 구성요소들이 정의된다. 클라이언트 측은, 별도의 애플리케이션 플랫폼 또는 프



(그림 4) JARSIO 아키텍처

레임워크 (Spring, EJB 등)의 활용여부와는 무관한 자바환경이 되며, OSGi 내부의 서비스를 원격으로 호출 및 연동하기 위한 구성요소들이 정의된다.

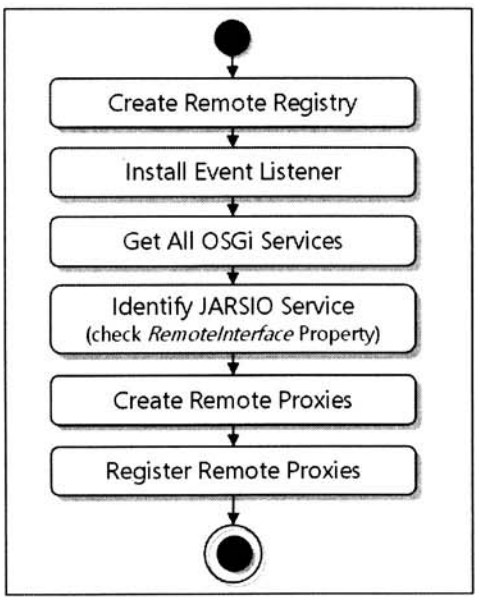
OSGi 플랫폼 내에서 동작하게 되는 서버 측 구성요소들은 OSGi 플랫폼 내의 서비스들에 대한 원격 호출을 지원하기 위한 레지스트리 및 프록시, 그리고 원격 채널 및 프로세서로 구성되며, 이러한 서비스를 사용하기 위한 클라이언트 측 구성요소들은 원격객체를 생성하기 위한 Factory 객체와 실제적인 로컬 호출을 원격 호출로 변환하기 위한 핸들러 및 채널로 구성된다. <표 1>은 JARSIO를 구성하는 구성요소들 및 역할을 설명하고 있다.

〈표 1〉 JARSIO의 구성요소

구분	구성요소	설 명
서버 구성요소	RemoteServiceProxy	OSGi 플랫폼 내에 등록된 서비스를 사용하기 위한 원격객체이다. 이것은 외부에서 들어온 호출을 OSGi 프레임워크에 등록되어 있는 서비스객체에 대한 호출로 변환하여 처리하는 일을 담당하며, 원격호출을 위해 JARSIO 레지스트리에 등록되는 단위이다.
	RemoteRegistry	RemoteServiceProxy들이 등록되는 서비스 레지스트리로 등록된 원격객체에 대한 관리를 수행하며, 외부에서 요청한 서비스객체에 대한 원격객체의 반환을 담당한다.
	RemoteProcessor	원격객체와 OSGi 서비스객체 간에 송수신 되는 메시지를 해석하고 처리하는 일을 담당한다.
클라이언트 구성요소	RemoteClass	OSGi 서비스객체에 접근하기 위한 스텁객체이다. 즉, 외부 서비스객체는 이 스텁객체를 통해 OSGi 서비스객체와 통신한다. 이 과정에서 스텁객체는 문자열로 된 메소드명 사용하여, OSGi 내부에 등록된 서비스객체를 호출한다.
	RemoteFactory	OSGi 서비스에 대한 스텁객체인 RemoteClass의 생성 및 관리를 담당한다. 이것은 OSGi 내부에서 생성된 RemoteRegistry와 통신하며, RemoteClass의 동작을 처리하기 위한 RemoteCallHandler를 생성 및 등록한다.
	RemoteCallHandler	스텁객체인 RemoteClass에 전달된 호출을 OSGi 서비스객체에 전달하고 반환값을 전달받는 일을 담당한다. 즉 로컬호출을 원격호출로 변환하는 일을 수행한다.
공통 구성요소	RemoteChannel	원격객체와 OSGi 서비스객체간의 실제적인 통신을 담당한다. 다양한 통신프로토콜을 추상화할 수 있으며, 기본적으로는 TCP/IP 채널의 구현을 사용한다.

3.3 JARSIO 구동

JARSIO의 서버측 구성요소는 OSGi 플랫폼에서 구동 가능한 번들형태로 구현된다. 이러한 JARSIO 번들은 시작시점에서 원격객체를 등록 관리하기 위한 JARSIO 원격레지스트리를 생성하며, JARSIO 서비스의 등록을 파악하기 위한 이벤트 리스너를 OSGi 프레임워크 내부에 설치하게 된다. 설치된 이벤트 리스너는 OSGi 프레임워크 내에 새로운 서비스가 설치되는 것을 지속적으로 모니터링하게 되며, 새로운 서비스가 설치되는 경우, JARSIO 인터페이스 기술속성인 RemoteInterface 속성의 존재유무에 따라 원격객체 생성하고 이를 JARSIO 서비스레지스트리에 등록하게 된다. 이러한 등록절차는 JARSIO 구동시점에서 기 설치된 모든 OSGi 서비스들에 대해서도 적용된다.



(그림 5) JARSIO의 구동절차

3.4 JARSIO 서비스 등록

일반적으로 OSGi 서비스는 번들의 설치 후 서비스 객체를 생성하고, 필요한 속성 값을 등록 후, 최종적으로 OSGi 레지스트리에 등록된다. JARSIO 서비스 역시 기존의 표준 등록과정을 변형하지 않은 절차를 따른다. JARSIO 서비스 등록은 기존 OSGi 서비스 등록과정에서 JARSIO 서비스 인터페이스 기술속성인 RemoteInterface 속성 값의 설정으로 이루어지며, 속성값이 설정된 서비스에 한해 JARSIO 엔진은 해당 인터페이스에 대한 원격객체를 생성하여 JARSIO레지스트리에 등록하게 된다.

이러한 JARSIO 서비스 등록절차는, OSGi 플랫폼의 서비스 등록절차를 변형시키지 않는 범위 안에서 JARSIO 의 활용을 가능하게 하며, 기존플랫폼 내 표준을 준수함으로써 기존 OSGi 서비스들과의 상호운용성을 보장한다. 또한 JARSIO 서비스가 별도의 서비스가 아닌 하나의 OSGi 서비스로서 체계적인 관리를 받을 수 있도록 한다.

(그림 6)은 화재센서에 대한 일반적인 OSGi 서비스 등록 코드이다. 먼저, 화제센서에 대한 서비스 객체(FireSensor-DeviceImpl)을 생성한 후, 해당 객체를 인터페이스 클래스를 기반으로 레지스트리에 등록하게 된다. 이것은 다른 서비스에서 호출 가능한 원격인터페이스와 내부인터페이스를 구분하기 위한 것이다. JARSIO 역시 이러한 방식을 통해서 원격인터페이스와 내부인터페이스를 구분한다. (그림 7)은 이러한 JARSIO 서비스의 등록과정을 보여준다. 기존의 서비스 등록과정에서 속성 값인 *RemoteInterface*를 이용하여 원격인터페이스를 지정하게 되며, 해당 원격인터페이스에 지정된 메소드들은 외부에 공개되어, JARSIO 라이브러리를 사용해 호출가능하게 된다.

(그림 8)은 JARSIO 가 OSGi 서비스에 대한 원격지원을 수행하게 되는 과정을 보여준다. JARSIO 는 앞서 언급한대로 OSGi 플랫폼에 서비스가 등록되는 것을 모니터링하게 되

```
// Create service object
FireSensorDevice sensor = new FireSensorDeviceImpl();

// Set general properties
Dictionary properties = new Properties();
properties.put("category", "sensor");

// OSGi Service Registration.
context.registerService(FireSensorDevice.class.getName(),sensor, properties);
```

(그림 6) OSGi 서비스의 등록

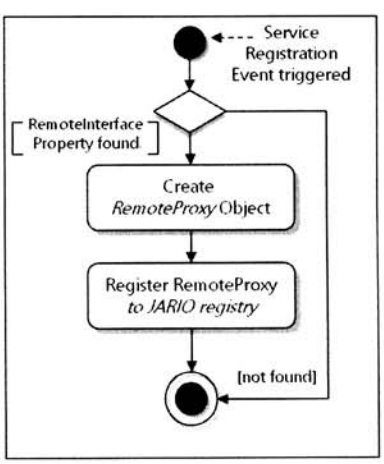
```
// Create service object
FireSensorDevice sensor = new FireSensorDeviceImpl();

// Set general properties
Dictionary properties = new Properties();
properties.put("category", "sensor");

// Set JARSIO Interface property.
properties.put("RemoteInterface", FireSensorDevice.class.getName());

// OSGi Service Registration.
context.registerService(FireSensorDevice.class.getName(),sensor, properties);
```

(그림 7) JARSIO 서비스의 등록



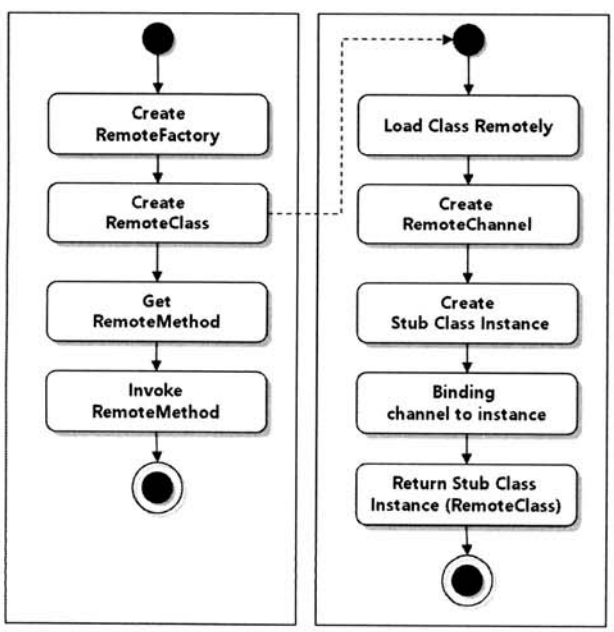
(그림 8) JAESIO 서비스 등록처리 절차

며, 이 과정에서 등록된 서비스에 RemoteInterface 속성이 명시된 경우, 이에 대한 원격 프록시를 생성하게 레지스트리에 등록하게 된다. 원격 프록시에 대한 생성 및 등록이 끝나면 외부에서는 JARSIO 클라이언트 라이브러리를 가지고 자유롭게 OSGi 플랫폼 내의 서비스를 호출 및 연동할 수 있다.

3.5 JARSIO 서비스 사용

등록된 JARSIO 서비스는 다른 외부프레임워크와의 존재유무와는 상관없이 어떠한 자바환경에서도 호출가능하다. 클라이언트 측은 JARSIO 라이브러리를 사용하여 스텁객체를 동적으로 생성할 수 있으며, 이를 통해 OSGi 내부의 서비스를 호출하게 된다. 이를 위해 JARSIO 클라이언트에서는 먼저, OSGi 서비스의 사용을 위해 스텁객체의 생성을 담당하는 RemoteFactory 객체를 생성해야 한다. 그 후, 생성된 RemoteFactory 객체를 이용해서 스텁객체인 RemoteClass 객체를 생성할 수 있으며, 이를 통해 자유롭게 OSGi 프레임워크 내부에 등록된 서비스를 사용할 수 있다. RemoteClass는 JARSIO 서비스를 호출하기 위해서 내부적으로 하나의 RemoteChannel과 바인딩되며, 이를 통해 객체의 호출 및 반환값을 송수신 하게 된다. (그림 9)는 클라이언트에서 JARSIO 클라이언트 서비스의 생성 및 사용과정을 보여준다.

(그림 10)은 실제적으로 OSGi 프레임워크 내에 등록된 서비스 인터페이스 예를 보여주고 있으며, (그림 11)은 이러한 서비스를 JARSIO 서비스로 등록하였을 경우, 이를 JARSIO 클라이언트에서 호출하는 예를 보여주고 있다. JARSIO 클라이언트에서는 스텁객체인 RemoteClass의 RemoteMethod를 호출함으로써 JARSIO 서비스로 등록된 OSGi 서비스를 자유롭게 호출할 수 있다. 이 때, JARSIO 클라이언트 측에서는 별도의 정적 스텁의 생성이나, 코드의 변환 과정을 요구



(그림 9) JARSIO 클라이언트 서비스 생성 및 사용

```
package osgi.device.sensor;

public interface FireSensorDevice {
    public String getStatus();
    public void setMode(String mode);
    public void setLocation(String place, int x, int y);
}
```

(그림 10) OSGi 서비스 인터페이스의 예

```
// Create remote class.
RemoteFactory factory = new RemoteFactory("203.253.23.194");
RemoteClass rc = factory.getRemoteClass("osgi.device.sensor.FireSensorDevice");

if ( rc != null ) {
    // Get remote method handle.
    RemoteMethod getStatus = rc.getMethod("getStatus");
    RemoteMethod setMode = rc.getMethod("setMode");
    RemoteMethod setLocation = rc.getMethod("setLocation");

    try {
        // Invoke remote method of OSGi service
        System.out.println("Status : " + getStatus.invoke());
        setMode.invoke("NORMAL");
        setLocation.invoke("living room", 34, 60);
    } catch (RemoteException e) { }
}
```

(그림 11) JARSIO 클라이언트의 서비스 호출 예

하지 않으며, 코드 레벨에서 동적으로 스텁을 생성 및 운용할 수 있다.

이 과정에서 JARSIO는 내부적으로 원격 OSGi 프레임워크 내의 서비스 클래스의 정보를 원격에서 로드하여 참조할 수 있도록 한다. 이것은 기존의 RMI[7] 나 웹서비스[8]에서 필요하였던 정적타임에 스텁을 생성하는 과정을 없애준다.

JARSIO의 원격로드는 원격지에서 자바가상머신에서 필요로 하는 클래스정보를 네트워크를 통해 수신하는 것이다. 즉, 클래스가 가지고 있는 메서드의 이름 등 클래스 관련 정보를 네트워크를 통해 수신 받고 이를 원격지에서 클래스화 하는 과정으로 요약된다. 하지만 이러한 과정은 스텁 자체를 전송받는 것과는 다르다. JARSIO는 단지 메서드 이름 등의 클래스 형식정보만을 수신하는 것이며, 그 외의 기능들은 내부 공통프레임워크에서 제공하게 된다.

클래스 형식정보는 OSGi 플랫폼의 자바가상머신이 실행과정에서 로드하여 메모리에 적재한 바이트 코드를 기반으로 구성된다. 즉, JARSIO가 별도의 처리를 통해 메모리에 로드하여 사용하는 것이 아니라, 이미 로드한 것을 사용하는 것이기 때문에 시스템적인 오버헤드가 상대적으로 매우 작다. 또한, JARSIO의 클래스 정보추출은 클래스의 구현과는 무관한 형식정보이므로, 추출된 정보의 크기 또한 매우 작아 네트워크 송수신에 따른 오버헤드 또한 크지 않다. 실제로 2~3개의 매개변수 포함한 메소드 50개로 구성된 인터페이스 클래스의 바이트코드 크기는 약 1kbyte 정도로, 이를 네트워크로 전송 시 소요시간은 1Mbps 기준 약 0.008초 정도이며, 이것은 최초 원격클래스를 얻는 과정에서 단 한 번만 수행된다. 따라서 JARSIO는 사용자의 입장에서 충분

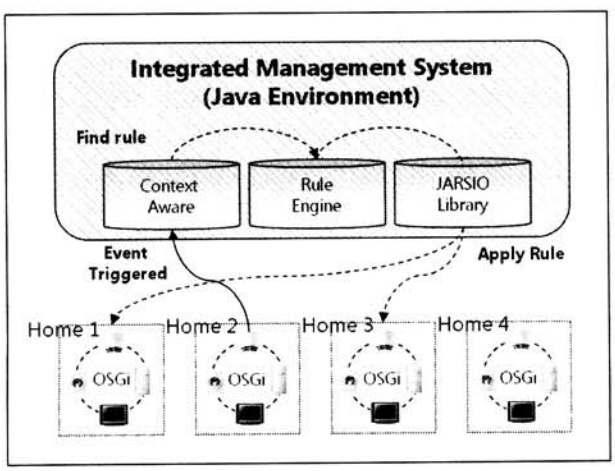
히 수용 가능한 범위 내에서 효과적인 방법을 통해 동적스텁의 생성을 실현하고 있다고 할 수 있다. 그러나 향후연구를 통해 이러한 동적 스텁생성의 효율성 및 오버헤드 문제를 보다 더 개선하도록 할 것이다.

4. JARSIO 사례연구

JARSIO는 OSGi 프레임워크 내의 서비스들에 대해 편리한 접근을 가능하게 하며, 외부환경에서 OSGi 서비스들에 대한 동적 탐색 및 활용을 가능하게 한다. 뿐만 아니라, 다수의 OSGi 기반 환경을 하나의 통합 환경으로 구성 및 관리할 수 있으며, 이러한 환경에서의 상황인지 및 지능형 서비스를 가능하게 한다.

(그림 12)는 이러한 예로, JARSIO를 이용한 통합 지능형 홈 네트워크 환경 구성의 예를 보여주고 있다. 각 가정에는 OSGi 플랫폼을 기반으로 한 홈 네트워크가 구성되며, 이러한 홈 네트워크 및 통합관리 시스템에 JARSIO 서비스가 포함될 수 있다. 이러한 환경이 구성될 경우, 통합관리 플랫폼에서는 각각의 홈 네트워크를 하나의 상황인지 환경으로 인식할 수 있으며, 이에 따른 확장된 지능형 서비스의 제공 및 상황인지 서비스의 제공이 가능하다. 즉, 하나의 지능형 홈에서 발생한 정보는 다른 지능형 홈에게 제공될 서비스를 결정하게 되며, 이러한 메커니즘을 통해 개별 홈 네트워크 환경들은 하나의 단일 유비쿼터스 서비스 환경으로 확장 및 통합된다.

본 사례연구에서는, 이러한 가상 홈 네트워크 통합관리 시스템을 구성하고 지능형 홈의 화재센서의 정보를 JARSIO 서비스를 통해 모니터링 하여, 화재가 발생하였을 경우 물리적으로 인접한 홈에 화재경보를 전달하도록 하는 지능형 홈 방재시스템을 구현하였다. 본 사례연구에서 지능형 홈의 경우, OSGi 플랫폼을 이용해 가상으로 구성하였으며, OSGi 구현체로 Eclipse의 Equnox를 사용하였다. 통합관리 시스템 측은 J2SDK(Java2 Software Development Kit) 1.6.0을 기반으로 한 자바애플리케이션으로 구현하였다.

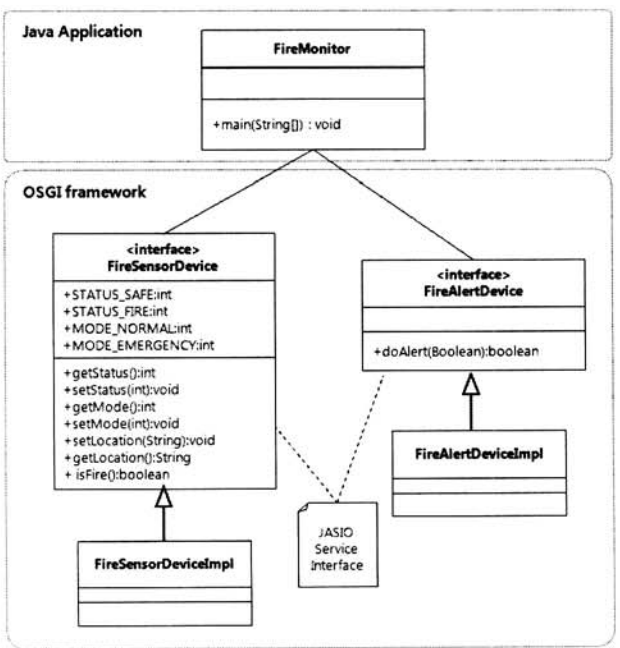


(그림 12) JARSIO를 이용한 통합 홈네트워크의 구성 및 동작

사례연구에서 구축한 지능형 홈 방재시스템은 홈에서의 화재상황을 감지하고, 경고하기 위한 화재센서서비스, 화재경보서비스(OSGi 플랫폼 상에 구현됨) 이를 외부에서 통합관리하기 위한 방재모니터(자바애플리케이션으로 구현됨)로 구성된다. (그림 13)은 이러한 지능형 홈 방재시스템의 클래스다이어그램을 나타내고 있으며, (그림 14)는 위의 시스템에서 실제적으로 가정 내의 OSGi 프레임워크에 설치되는 방재관련 서비스의 등록 및 JARSIO 설정을 나타내고 있다.

지능형 홈 방재시스템의 방재모니터는 JARSIO를 이용하여 총 4개 홈의 화재상황을 실시간으로 감시하고, 화재발생시 물리적으로 인접한 곳에 존재하는 화재경고를 동작시키는 기능을 수행하게 된다. 본 사례연구에서 4개의 홈은, 순서대로 인접해 있음을 가정하였으며, 동일한 로컬컴퓨터상의 서로 다른 4개의 OSGi 프레임워크로 표현하였다.

(그림 15)는 JARSIO를 이용한 방재모니터의 구현을 보여주고 있다. 앞서 설명한 것처럼, JARSIO를 활용할 경우 이러한 통합 OSGi 환경 구성과정에서 별도의 프록시나 스텁



(그림 13) JARSIO를 이용한 지능형 홈 방재시스템의 클래스다이어그램

```
// Create service object and set JARSIO interface property.
FireSensorDevice sensor = new FireSensorDeviceImpl();
Dictionary sensorProperties = new Properties();
sensorProperties.put("RemoteInterface",FireSensorDevice.class.getName());

// Create service object and set JARSIO interface property.
FireAlertDevice alert = new FireAlertDeviceImpl();
Dictionary alertProperties = new Properties();
alertProperties.put("RemoteInterface",FireAlertDevice.class.getName());

// OSGi and JARSIO Service Registration.
context.registerService(FireSensorDevice.class.getName(),sensor, sensorProperties);
context.registerService(FireAlertDevice.class.getName(),alert, alertProperties);
```

(그림 14) OSGi 프레임워크 내 화재센서서비스와 화재경보서비스의 등록 및 JARSIO 설정

```
int N = 4;
RemoteFactory[] home = new RemoteFactory[N];

home[0] = new RemoteFactory("203.253.23.194", 5801);  // Create RemoteFactories.
home[1] = new RemoteFactory("203.253.23.194", 5802);
home[2] = new RemoteFactory("203.253.23.194", 5803);
home[3] = new RemoteFactory("203.253.23.194", 5804);

RemoteClass[] fireSensor  = new RemoteClass[N];
RemoteClass[] fireAlert = new RemoteClass[N];

RemoteMethod[] rmDoAlert = new RemoteMethod[N];
RemoteMethod[] rmIsFire = new RemoteMethod[N];

 // Create RemoteClasses & RemoteMethods.
for ( int i  = 0 ; i < N ; i++ ) {
   fireSensor[i] = home[i].getRemoteClass("osgi.home.device.sensor.FireSensorDevice");
   fireAlert[i] = home[i].getRemoteClass( "osgi.home.device.alert.FireAlertDevice");

   rmIsFire[i] =  fireSensor[i].getMethod("isFire");
   rmDoAlert[i] =  fireAlert[i].getMethod("doAlert");
}

while ( true ) {
   for ( int i  = 0 ; i < N ; i++ ){
      if ( (Boolean)rmIsFire[i].invoke() == true ){              // Check if there is a fire.
         // Fetch fire alert method of neighbors.
         for (int j = -1 ; j < 2 ; j++ )
         {
            if ((Boolean)rmDoAlert[(i+j+N)%N].invoke(true))
               System.out.println("HOME #"+((i+j+N)%N)+" fire alert is activated. \n");
            else
               System.out.println("HOME #"+((i+j+N)%N)+" fire alert is NOT activated. \n");
         }
      }
   }
   Thread.sleep(10000);   // Periodically check.
}
```

(그림 15) JARSIO를 이용한 지능형 홈 방재모니터의 구현

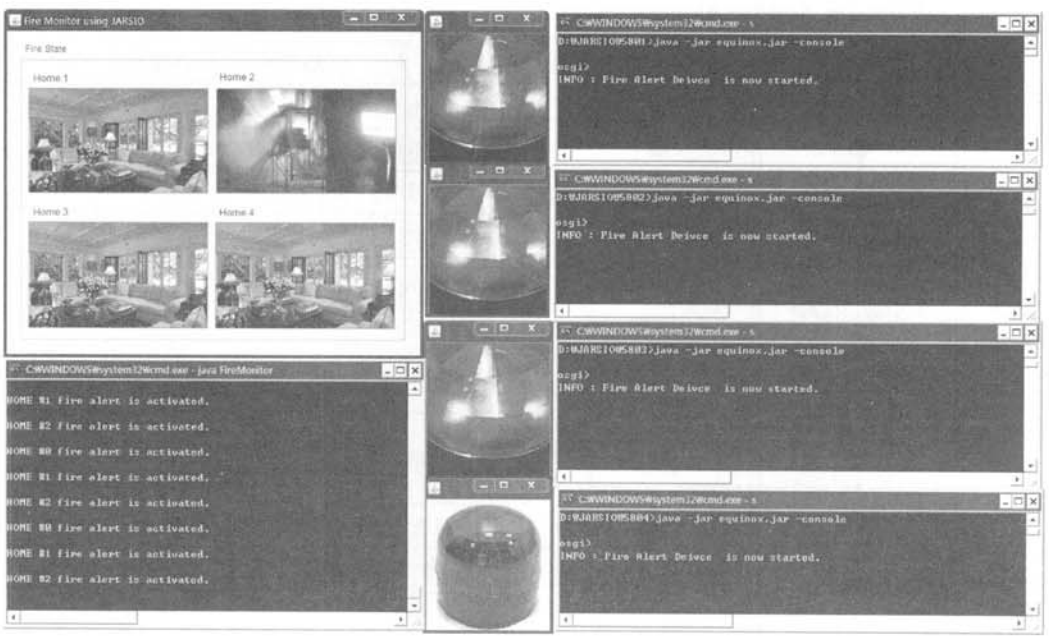
을 생성해야하는 과정은 없으며, 표준 OSGi 플랫폼의 변경이나 별도의 서버의 구동도 필요치 않다. RemoteFactory를 이용하여 스텁객체를 생성하고, 이를 통해 원격메서드에 대한 참조를 얻어 화재센서 및 화재경보 서비스를 접근할 수 있다.

(그림 16)은 이러한 지능형 홈 방재서비스의 구동화면을 보여주고 있다. 방재모니터는 JARSIO를 이용하여 4개의 가상 홈 내부에 대한 상황을 OSGi 서비스를 통해 모니터링할 수 있으며(FireSensorDevice의 isFire 메소드 호출), 화면에서 보여지는바와 같이 두 번째 홈에서 화재가 발생한 경우, 첫 번째 및 두 번째, 그리고 세 번째 홈의 화재경고를 전달할 수 있다. (FireAlertDevice의 doAlert메소드 호출)

5. JARSIO 의 활용 및 평가

JARSIO 는 OSGi 플랫폼 내의 레지스트리에 기반한 다양한 서비스들의 협업 및 연동을 외부 환경에서도 가능하도록 지원한다. 즉, JARSIO를 기반으로 다양한 외부 자바환경들이 OSGi 프레임워크와 융복합될 수 있으며, 이를 통해 보다 강력한 서비스 통합환경을 구축할 수 있다. 현재 이러한 OSGi 외부연계 문제를 기존의 다른 연구들[7-9]에서는 자바 원격메소드 호출(RMI:Remote Method Invocation)[10]기술을 이용해 접근하고 있다.

자바 원격메소드 호출은 원격지에 존재하는 프로시저를 로컬에 있는 것처럼 호출할 수 있도록 지원하는 메커니즘이다. 그러나 이러한 자바원격메소드 호출을 이용해 OSGi 서비스를 호출하기 위해서는 OSGi 플랫폼과는 별도의 레지스트리를 구동하여야 하며, 또한 외부에서 해당 서비스를 호출하기 위한 스텁 및 스켈레톤을 실행시간 이전에 생성 및 배포하여야 효과적으로 동작할 수 있다. 따라서, 실행시간에 동적으로 서비스가 추가 및 갱신되는 OSGi 환경에서 이러한 자바원격메소드 호출을 사용하여 외부연동을 하는 것은 불가능한 일이며, 또한 별도의 레지스트리 구동에 따른 OSGi 플랫폼과의 비통합성 및 부하증가는 유비쿼터스서비스 제공에 있어 다소 문제점으로 작용한다.



(그림 16) 지능형 홈 방재시스템의 실행화면

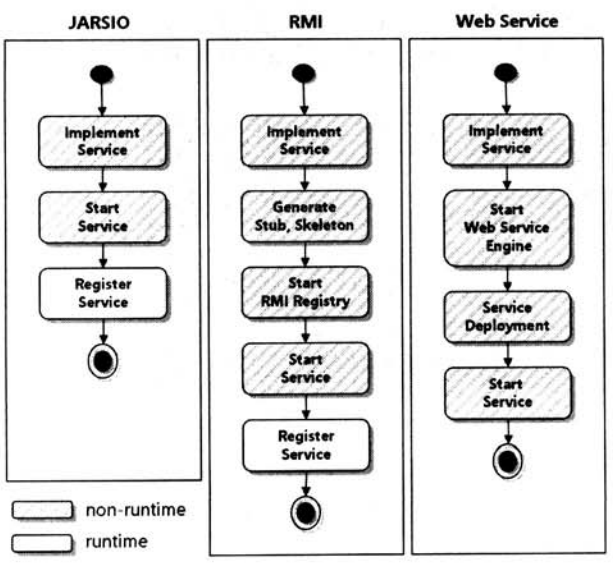
자바웹서비스 기술 또한 OSGi 프레임워크상에서 외부 서비스 연계기술로 사용가능하다. 그러나 자바 웹서비스의 경우에도 WSDL(Web Service Description Language)를 이용하여 동적인 서비스의 발견이나 구성은 가능하지만, 역시 클라이언트 환경에서는 실행시간 이전에 스텁 클래스를 생성하여야 하며, 따라서 서비스의 구성이 동적인 OSGi 환경을 지원하는 것은 불가능하다. 또한 별도의 웹서비스 실현을 위한 엔진을 구동하여야 하며, 내부적으로 XML언어로 표현되는 SOAP(Simple Object Access Protocol) 메시지를 사용하기 때문에 메시지의 파싱 및 처리로 인한 부하증가로 적절한 응답성 및 처리시간을 보장하기에는 다소 어려운 점이 있다.

(그림 17)은 JARSIO를 사용한 서버측에서 OSGi 서비스와 외부환경의 연동절차와 자바원격메소드호출 및 웹서비스를 사용한 외부연동 절차의 편의성에 대한 비교결과를 보여주고 있다.

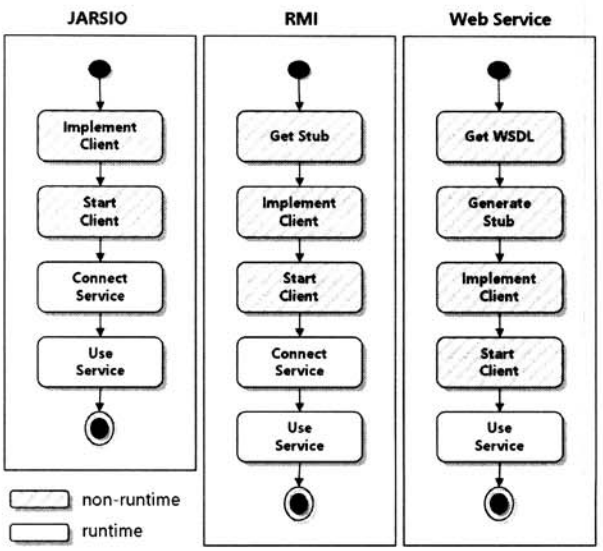
여기서 중요한 것은 자바원격메소드 호출 및 웹서비스는 OSGi 서비스를 외부와 연동하기 위해서 별도의 스텁/스켈레톤이나 서비스의 배치가 비실행시간에 이루어진다는 점이다. 이것은 만약 동적으로 서비스가 추가되거나 갱신되었을 경우, 이러한 추가나 갱신을 지원하기 위해서 서비스를 중단해야하는 결과를 불러일으키며, 따라서 이것은 OSGi 가 가지는 동적 서비스 구성 및 갱신환경을 제대로 지원할 수 없게 된다.

클라이언트 측에서 이루어지는 절차 또한 유사하다. (그림 18)과 같이 자바원격메소드 호출 및 자바웹서비스는 서비스 연동을 위해 비실행시간에 해당 서비스 사용을 위한 스텁을 얻어오거나 생성해야 하기 때문에 동적으로 서비스 갱신이 이루어지는 OSGi 서비스 환경과 연동하기에는 많은 문제점이 있다.

그러나 클라이언트 내부구현에 있어서 JARSIO는 자바웹



(그림 17) 외부연동 절차 비교 - 서버측



(그림 18) 외부연동절차 비교 - 클라이언트측

서비스나 자바원격메소드호출과 달리 서비스호출을 위해 부가적인 객체들을 생성하는 과정이 다소 복잡한 점이 있다. 실제 서비스하나를 사용하기 위해 RemoteFactory, RemoteClass, RemoteMethod의 3개 객체를 정해진 방법에 따라 순차적으로 생성해야하기 때문이다. 따라서 향후 연구를 통해 사용측면에서 보다 편리하게 사용가능한 방법을 연구할 것이다.

<표 2>는 본 논문에서 제시하는 JARSIO 및 자바원격메소드, 웹서비스간의 비교평가를 보여주고 있다. 앞서 설명한 바와 같은 동적 서비스의 지원여부 이외에도 JARSIO는 편의성 측면에도 장점을 지니며, 다른 기술들이 지원하는 대부분의 기능을 제공한다.

〈표 2〉 JARSIO 및 타 연동기술간의 비교평가 표

특성	JARSIO	RMI	Web Service
동적서비스구성	지원	비지원	비지원
연동편의성	높음	낮음	낮음
사용편의성	낮음	높음	높음
시스템통합성	높음	낮음	낮음
스텁생성	필요하지 않음	필요함	필요함
자바환경통합	내부통합	외부연동 (RMI Registry)	외부연동 (WebService Engine)
보안	별도구현에 따름	내부관리객체사용 (RMI Security Manager)	표준메커니즘 적용 (WS-Security 등)
트랜잭션처리	불가능	불가능	가능
타 언어 연동	불가능	불가능	가능
연동메시지 형태	바이트 (자바객체 직렬화)	바이트 (자바객체직렬화)	텍스트 (XML)

6. 결 론

본 논문에서는 OSGi 환경에서 등록된 서비스들의 효율적인 서비스의 공개 및 호출을 위한 자바원격서비스 호출(JARSIO:JAva Remote Service Invocation for OSGi) 프레임워크를 제안하였다. 이것은 동적으로 서비스가 추가 및 갱신되는 OSGi 환경에서 내부의 서비스들과 외부의 서비스들을 보다 효과적으로 연동하기 위한 서비스로, 기존의 자바 기술들이 OSGi 환경에 적용될 수 없는 문제점들을 개선한 것이다. 즉, 서비스간 연동에 있어서 별도의 비실행시간 작업들을(스텁/스켈레톤 생성) 제거하고, 서비스의 공개 및 활용을 모두 실행시간에 수행될 수 있도록 함으로써, 동적 서비스 구성을 지원하고 설정절차의 간편화를 통해 효과적인 연동이 가능하도록 하였다.

이러한 JARSIO를 활용할 경우, 동적 서비스 구성이 가능한 OSGi 플랫폼을 외부 환경과 편리하게 통합하고, 이를 통해 다양한 분산서비스를 연동함으로써 효과적인 홈네트워크 및 분산 유비쿼터스서비스환경을 구성할 수 있을 것으로 기대된다. 또한 분산 애플리케이션 및 엔터프라이즈 응용 솔루션의 통합과정에서 애플리케이션 구조를 단순화하고, 다양한 분산서비스에 대한 접근성 개선에도 크게 기여할 수 있을 것으로 판단된다.

향후 연구로는, 이러한 JARSIO 서비스에 대한 성능향상 및 보안 문제에 대한 추가적인 연구가 진행될 것이며, 또한 이를 응용한 다양한 유비쿼터스통합환경 아키텍처 및 응용 서비스구조에 대한 연구도 진행될 것이다.

참 고 문 헌

[1] The OSGi Alliance, "OSGi Service Platform, Core Specification r4", Aug., 2005. http://www.osgi.org

[2] L. Gong, "A Software Architecture for Open Service Gateways," IEEE Internet Computing, Vol.5, No.1, pp.64-70, 2001.

[3] M.P. Papazoglou and W-J. van den Heuvel, "Service-Oriented Architectures : Approaches, Technologies and Research Issues", VLDB J., Vol.16, No.3, 2007, pp.389-415

[4] Pavlin Dorbrev, David Famolari, Christian Kurzke, Brent A. Miller, "Device and Service Discovery in Home Networks with OSGi", IEEE Communications Magazine, Augest 2002

[5] R.S. Hall, H. Cervantes, "An OSGi implementation and experience report", Consumer Communications and Networking Conference, 2004. CCNC 2004. First IEEE pp.394-399, 5-8 Jan., 2004.

[6] S. Chemichkian, "Building smart services for smart home", Networked Appliances, 2002. Gaithersburg. Proceedings, 2002 IEEE 4th International Workshop on, 2002, pp.215-224.

[7] Kyuchang Kang, Jeunwoo Lee, Hoon Choi, "Extended Service Registry for Distributed Computing Support in OSGi Architecture", Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference.

[8] Lu Yiqin, Yuan Yao, Sun Yingkai, Yang Xiaodong, "An approach to service integration in the OSGi architecture of home networks", Communication Systems, 2008. ICCS 2008. 11th IEEE Singapore International Conference on 19-21 Nov., 2008 Page(s):756-760.

[9] Marquez, J.M., Alamo, J., Ortega, J.A., "Distributing OSGi Services: The OSIRIS Domain Connector", Networked Computing and Advanced Information Management, 2008. NCM '08. Fourth International Conference on Volume 1, 2-4 Sept., 2008 Page(s):341-346.

[10] Sun Microsystem Javasoft Java RMI Team, "Java Remote Method Invocation, Specification", 1997.

[11] W3C Working Group Note, "Web Service Architecture", 2004.

최 재 현

e-mail : uniker80@empal.com

2004년 숭실대학교 컴퓨터학부(공학사)

2006년 숭실대학교 컴퓨터학과(공학석사)

2006년~현 재 숭실대학교 컴퓨터학과 박사과정

관심분야 : 소프트웨어아키텍처, 분산컴퓨팅, SOA, 유비쿼터스

박 제 원

e-mail : jwpark5656@hotmail.com

2006년 숭실대학교 컴퓨터학과(공학석사)

2006년~현 재 숭실대학교 컴퓨터학과 박사과정

관심분야 : 소프트웨어테스팅, 소프트웨어프로세스, 웹서비스, SOA/ESB

이 남 용

e-mail : nylee@ssu.ac.kr

1983년 고려대학교 경영정보학과(석사)

1993년 미시시피주립대학 경영정보학과(경영학박사)

1979년~1983년 국군정보사령부 정보처 정보시스템분석 장교

1983년~1999년 한국국방연구원 군수체계 및 정보체계연구부장

2000년 한국전자거래학회 논문편집위원장

2004년 한국정보통신기술사협회 회장

1999년~현 재 숭실대학교 컴퓨터학과 교수

관심분야 : 소프트웨어테스팅, 시스템엔지니어링 등