CAMUS 시스템에서의 가변적인 서비스 라이프 사이클 관리 기법

정인철, 서영호, 이강우, 김현, 함호상
한국전자통신연구원
e-mail : { jic, yhsuh, kwlee, hyunkim,hsham}@etri.re.kr

Dynamic Service-Lifecycle Management in CAMUS

In-Cheol Jung, Young-Ho Suh, Kang-Woo Lee and Hyun Kim, Ho-Sang Ham
Intelligent Robot Research Division
Electronics and Telecommunications Research Institute
161 Kajong-Dong, Yusong-Gu, Taejon, 305-350, Korea
{ jic, yhsuh, kwlee, hyunkim, hsham}@etri.re.kr

요 약

컨텍스크 기반 미들웨어는 응용 도메인에서 실제 상황을 구현할 수 있다. 전형적으로 실제 위치를 환경에 메핑시킴으로서 실제상황을 가변적으로 변화시킬 수 있다. 또한 실제 위치에 센서와 actuator 를 이용한 서비스의 라이프사이클을 원격지에서도 가변적으로 변경할 수 있다. 이 시스템은 CAMUS[1] 시스템에서 라이프 사이클 관리를 OSGi 기반으로 한 내용을 구현하였다. OSGi 는 홈 네트워크 관련 개방형 시스템 게이트웨이 시스템으로서 OSGi 를 이용함으로써 다양한 유비쿼터스 통신, 가정, 컴퓨터 시스템간의 쉽게 라이프 사이클 관리를 할 수 있는 장점이 있다.

1. 서론

URC (Ubiquitous Robotic Companion) 시스템은 기존 로봇에 네트워크 및 정보기술을 접목한 진형 서비스 로봇의 새로운 개념으로, 언제 어디서나 나와 함께 하며, 나에게 필요한 서비스를 제공하는 네트워크 기반 로봇이다. URC 를 이용하여 서비스 로봇은 센서 네트워크와 리모트 서버를 접근할 수 있다. CAMUS (Context-Aware Middleware for URC system) 시스템은 소프트웨어 프레임워크의 구조를 가지며 URC 환경에서 context-aware 응용을 작성할 수 있다.

- O context 정보를 관리하고 표현하기 위한 UDM (Universal Data Model)
- O 자바 언어를 확장한 ECA (Event-Condition-Action) 룰 기반 프로그래밍 언어
- O 사용자와 위치정보 서비스를 위한 entity 의 위치 정보를 추적
- O 이벤트 기반 응용 프레임워크 개발
- O 여러가지 센서와 legacy 응용을 통합하기 위한 서비스 프레임워크

Figure 1 은 CAMUS 의 컨텍스트 다이아 그램을 나타낸다. Environment 는 실제 위치를 추상화 한 모델이다. 실재 상황에서 environment 는 계층구조를 가진다. 회사는 여러 개의 건물을 가지고 있고, 사무실과 방으로 이루어져 있다. 각 environment 는 유일하게 식별될 수 있도록 계층구조를 가지고 있다. (예 ETRI/building7/room8. Users) Users 는 Tasks 가 실행하는 장소의 environment 에 위치한다.

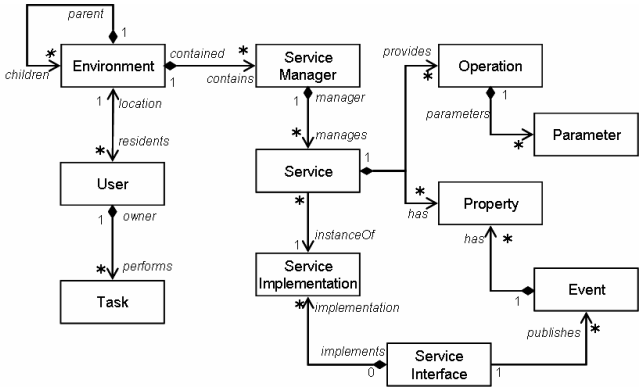

Figure 1. CAMUS Context diagram

예를 들면, conference room 에서 사용자는 conference 에 연결된 task 를 수행한다. 각 Environment 는 zero 또는 하나 이상의 Service Manager 를 가지고 있는다. 이 Service Manager 는 zero 이상의 서비스를 관리한다.

Service 는 environment 하에서 이용 가능한 여러 가지 센서와 actuator 로서 이루어 진 proxy object 의 형태를 가진다. Service 는 Operation 과 Properties 로 이루어 지는데 service client 가 actuator 와 sensor 를 control/sense 한다. 여러 가지 서비스 타입은 Service Interface 를 정의함으로써 설정된다. Service Interface 는 하나 이상의 Service Implementation 에 의하여 구현될 수 있다.

이 논문에서는 CAMUS 시스템에서의 OSGi 기반 서비스 lifecycle manager 를 기술한다. 실제적인 environment 설정은 여러 가지 이유로 변경이 될 수 있다. Home environment 에서 사용자는 구형의 에어컨을 신형으로 대치하거나 침실에 있는 TV 를 거실로 옮길 수 있다.

Service 가 proxy object wrapping device 로 구성되어 있기 때문에 서비스 lifecycle 은 전반적으로 환경의 device 에만 의존된다. 이 점이 CAMUS 가 서비스 lifecycle 을 동적으로 관리할 수 있는 메커니즘을 가지고 있다고 생각된다. 그래서 시스템은 어떤 시점의 외부 환경 설정의 변화를 처리할 수 있다.

더욱이, 대부분의 경우에 많은 환경 변수는 응용 도메인에 위치된다. 예를 들면 회사 도메인에서는 많은 사무실과 방이 구성될 수 있다.

이것을 각 environment 설정에 변화가 있을 때마다 각 위치를 방문하여 service lifecycle 을 local 하게 관리한다는 것이 어렵다. 이 경우에 CAMUS 시스템에서 remote management 가 요구조건이 된다.

이 문제를 해결하기 위하여 이 논문에서는 dynamic and remote service lifecycle management scheme 을 구현하고 설계한 내용을 기술한다. 이 논문의 나머지는 다음과 같다. 2 장에서는 CAMUS 서비스와 OSGi 서비스 사이의 의미상에서 차이점을 기술한다. 3 장에서는 의미상으로 연결되는 CAMUS service lifecycle 을 표현한다. 4 장에서는 CAMUS service lifecycle manager 의 구현과 설계를 기술한다. 5 장에서는 결론을 기술하였다.

2. OSGi 서비스와 CAMUS Service 의 비교

OSGi 서비스는 로칼 OSGi 프레임워크에 있는 bundles 중에서 공유되는 응용 컴포넌트일 동안에 CAMUS 서비스는 remote proxy object 로 네트워크 디바이스나 legacy 응용의 기능을 wrapping 한 것이다. OSGi 서비스와 CAMUS 서비스 사이의 개념상의 차이는 두 개의 시스템이 서로 다른 service lifecycle 을 가지는 결과를 낳는다.

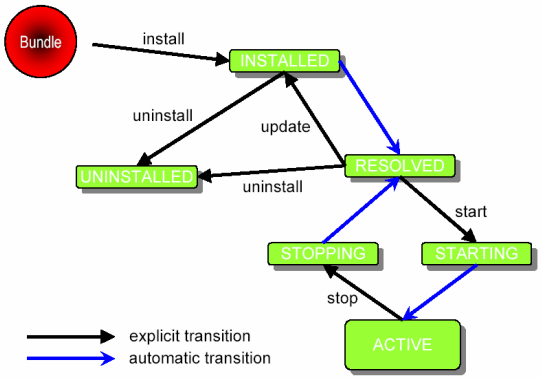

Figure 2. The OSGi bundle lifecycle

(용어상의 혼란을 피하기 위해 CAMUS service 를 'Service'로 사용한다.) Figure 2 은 OSGi bundle 의 lifecycle 을 나타낸다.

OSGi 프레임워크의 경우에, 서비스는 그들이 속한 bundle을 함께 lifecycle을 공유한다. Bundle 이 start 될 때 bundle은 서비스 instance를 생성하고 bundle을 프레임워크 서비스에 등록한다.

Bundle이 stop 될 때는 프레임워크는 자동적으로 모든 서비스를 unregistered 한다. 이 것은 서비스 수준의 lifecycle 관리가 OSGi 프레임워크에서는 가능하지 않다는 것을 의미한다. 또한 추가적인 서비스 instance 도 bundle이 started 되고 난 후에는 추가적인 service instance를 추가하지 못한다. 또한 bundle이 stop 되기 전에 service instance 를 stop 할 수 없다.

그러나 CAMUS 프레임워크에서, Services 는 bundle이 active 상태에 있는 동안에만 생성되거나 삭제된다. 또한 bundle이 install 되고 난 후에 실제 environment로부터 삭제되거나 추가될 수 있다.

이런 이유로 OSGi bundle로부터 서비스의 lifecycle을 분리하기 위해 Service Implementation의 집합으로서 우리의 프레임워크내의 OSGI bundle에 정의에 제한을 둠으로써 해결할 수 있다. 다른 말로 하면 각 Service Implementation에는 active bundle이 삽입되어 있고, unique id를 가진 새로운 Services는 서비스 인수를 passing 함으로서 생성된다. 또한 Services는 전체 bundle을 stop 시키지 않고, 개별적으로 stop 할 수 있다. Figure 3는 CAMUS 서비스 프레임워크에서 Service의 생성 및 삭제를 보여준다. 이 그림에서는 3가지 light 서비스가 생성되며 각기 다른 조명의 집중도를 가지며 그 중에 하나가 'kichen_light' 이다.

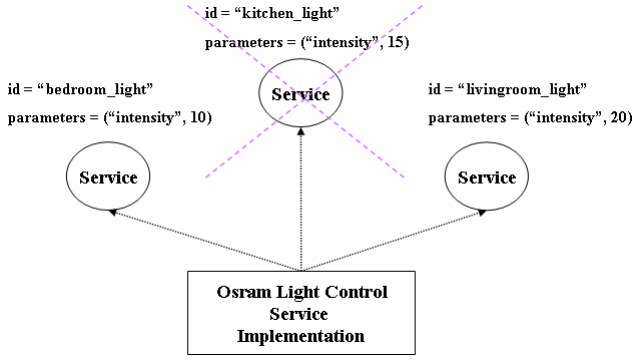

Figure 3. CAMUS *Service Implementation* and CAMUS *Service*

3. CAMUS Service Lifecycle

Service Implementation의 집합으로써 bundle 정의를 구현하기 위해서는 CAMUS 서비스 프레임워크는 Service Bundle, Service Bundle Info, Service Factory와 같은 개념을 구현해야 한다.

- CAMUS Service Bundle

CAMUS Service Bundle은 CAMUS Service Implementation을 가지고 있는 특별한 형태의 OSGi bundle이다. 이 것은 두 가지 이유로 특별하다. 첫 번째, XML file 이름은 'bundle.xml' 파일에 저장되어야 한다. Figure 4에서는 'bindle.xml'의 DTD을 보여준다.

다른 하나는 bundle activator을 가지고 있지 않는 경우이다. 대신에 CAMUS Service 프레임워크 클라이언트 라이브러리로 동작되는 독점적인 bundle activator를 가지고 있다. CAMUS Service Bundle이 OSGi 프레임워크에 설치될 때, Bundle Activator 는 'bundle.xml'을 parsing 하고, Service Bundle Info 을 생성하고 OSGi 프레임워크의 Service Registry에 등록한다.

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT bundle (service+)>
<!ATTLIST bundle id CDATA #REQUIRED>
<!ELEMENT service (implementation, interface, parameters)>
<!ATTLIST service id  #REQUIRED >
<!ELEMENT implementation EMPTY>
<!ATTLIST implementation class>
<!ELEMENT interface EMPTY>
<!ATTLIST interface class  #REQUIRED>
<!ELEMENT parameters (param*)>
<!ELEMENT param EMPTY>
<!ATTLIST param name  #REQUIRED value #REQUIRED>
```

Figure 4. DTD of the bundle.xml in the Service Bundle

● Service Bundle Info

서비스 Bundle Info 는 서비스 bundle 을 OSGi 프레임워크내의 다른 bundle 에게 전파할 때 이용된다. CAMUS 서비스 프레임워크는 OSGi 프레임워크의 시스템 bundle 로서 구현되며 프레임워크의 BundleListener 인터페이스로서 구현된다. 프레임워크는 미리 정의된 이벤트가 설치된 서비스 Bundle 에서 일어날 때마다 CAMUS Service 프레임워크에 전달된다. CAMUS Service 프레임워크는 'BUNFLE STARTED' 라는 이벤트를 받을 때 마다 서비스 registry 에서 새롭게 시작된 bundle 의 Service Bundle Info 객체의 reference 를 찾고서, 서비스 Bundle Info 객체를 사용하여 Service Bundle 객체를 생성한다.

● Service Bundle

Service Bundle 은 CAMUS Service Bundle 의 wrapper 된 객체이다. 이것은 wrapping 된 bundle 의 lifecycle 을 관리한다. 이것은 서비스 bundle 을 통하여 bundle 을 시작하고, 변경하고, uninstall 할 수 있다는 것을 의미한다. CAMUS Service Bundle 은 특별하지만 Figure 2 에서 기술한 바와 같이 OSGi bundle 과 동일하다. Service Bundle 은 bundle 을 install 할 수 있는 API 를 가지고 있지 않다. Bundle 을 생성시에 Service Bundle 객체는 Service Factory 를 생성하여 Service Implementation 을 삽입한다.

● Service Factory

Service Factory 는 CAMUS Service Bundle 의 wrapper 된 객체이다. 서비스 Bundle 는 여러 개의 Service Implementation 을 보유하고 있으며 Service Factory 객체는 Service Implementation 의 개수 가 생성된다. 이것으로 Service 의 lifecycle 을 관리한다. Service Factory 는 CAMUS Service 를 시작하고 중지할 수 있는 API 를 가지고 있는데 각 서비스는 적절한 인수에 의해서 unique id 을 이용하여 관리된다.

CAMUS Services 의 lifecycle 은 전적으로 그들이 속한 CAMUS Service Bundle 에 의존한다. 예를 들면 만일 CAMUS Service Bundle 에 uninstall 되어 있는 경우에 모든 'ACTIVE' CAMUS Services 는 bundle 이 stop 된 위치에서부터 시작되고, Service Factory 는 Service Bundle 과 함께 유지되며 CAMUS Services 와 CAMUS Service Bundle 중에 dependency 를 관리한다.

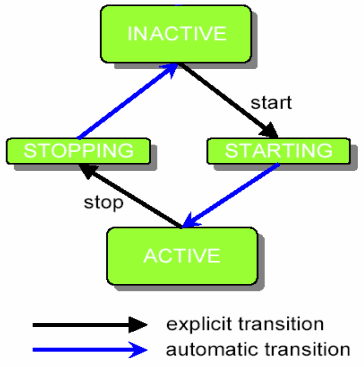

Figure 5. The GAMUS *Service* lifecycle

4. CAMUS Service Lifecycle Manager

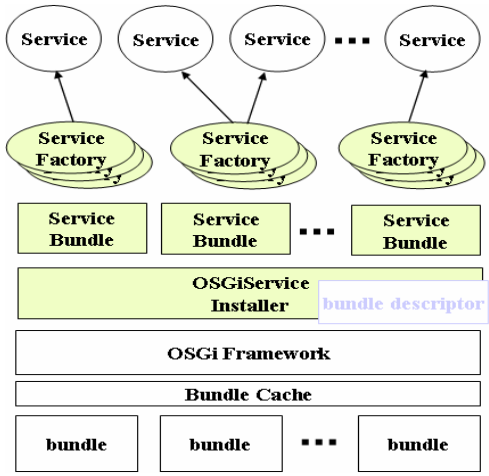

Figure 6. Service Lifecycle Manager Architecture

Figure 6은 CAMUS 서비스 프레임워크의 service lifecycle manager의 기본구조이다. 이 그림에서는 회색으로된 box가 CAMUS 서비스 프레임워크와 대응되는 OSGi Framework의 top에서 실행되는 것이다. 이 서비스는 remote object로 구현되어 있어서 원격으로 조정이 가능하다. 이 서비스는 CAMUS에서 remote object를 접근하는 원격 통신 프로토콜로 Hessian Protocol [4] 을 사용한다. 이 문서에서는 각 시나리오와 관련된 service lifcycle을 가변적으로 관리하는 프레임워크를 기술한다. 이 시스템에서 가변적이라는 의미는 'without needing to restart the system' 이라는 의미이다.

1) Bundle Installation

OSGi 서비스 Installer 객체는 bundle에 url을 기술함으로써 Bundle Repository로부터 download 되고, local Bundle Cache에 install 한다. OSGi installer는 OSGi 프레임워크의 시스템 bundle로서 구현되었다. Service Bundle은 OSGi 프레임워크가 'BUNDLE STARTED" 라는 event를 받으면 Bundle Cache에 설치된다. 그러면 서비스 registry로부터 설치된 bundle의 Service Bundle Info의 reference를 갖고서 Service Bundle Object를 생성하고 Service Factory Object를 생성한다.

2) Bundle Update

Bundle에 새로운 버전에 있는 경우에 Bundle Repository로부터 CAMUS 서비스 bundle을 update하는 API이다. 사용자가 Service Bundle Object를 자체적으로 update 하는 경우에 OSGi Framework는 local Bundle Cache에 있는 bundle을 update 한다. 만일 이용가능하다면 framework는 local bundle cache 에 있는 새로운 것으로 이전 것을 대치한다.

3) Bundle Uninstallation

서비스 Bundle object는 Local Bundle Cache로부터 CAMUS service bundle을 uninstall 한다. CAMUS Service와 CAMUS Service Bundle은 bundle을 uninstall 과정에서 account를 가져야 한다. 이는 Service Factory나 Services와 같은 Service Bundle Object 사이에 dependency chain이 존재하기 때문이다. Service라는 객체는 Service Factory Object에 의존한다. 만일 사용자가 Service Bundle Object를 자체를 삭제하려고 하면 먼저 그들의 Services 를 먼저 순서적으로 삭제하여야 한다.

Uninstall 되어야 하는 bundle과 관련된 모든 객체를 지우고 난 후에 OSGi 프레임워크는 local bundle cache에서 bundle을 uninstall 한 뒤 Service Bundle 객체를 지운다.

4) Service Creation and Removal

Service Factory 객체는 CAMUS Service를 생성하거나 삭제하기 위한 API를 가진다. 만일 사용자가 Service를 생성하기 위해 Service Factory를 요청하는 경우에 unique id 와 인수를 생성하여 wrapping 되는 Service Implementation의 instance를 생성한다. 만일 사용자가 어느 Service를 제거하기 위해 Service Factory 객체를 요청하는 경우에 Service를 stop 시킨다. Service Factory 객체의 내부 상태는 XML 문서의 형태로서 유지, 보수한다.

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT bundle_container (bundle+)>
<!ELEMENT bundle (service_factory+)>
<!ELEMENT service_factory (service+)>
<!ATTLIST service_factory id CDATA #REQUIRED>
<!ELEMENT service (implementation, interface, parameters)>
<!ATTLIST service id  #REQUIRED name #REQUIRED rank CDATA #REQUIRED>
<!ELEMENT implementation EMPTY>
<!ATTLIST implementation class>
<!ELEMENT interface EMPTY>
<!ATTLIST interface class  #REQUIRED>
<!ELEMENT parameters (param*)>
<!ELEMENT param EMPTY>
<!ATTLIST param name  #REQUIRED value #REQUIRED>
```

Figure 7. DTD of the bundle_descriptor.xml

Service가 생성되거나 삭제되는 경우에 관련된 정보는 'bundle_descriptor.xml' 이라는 XML file 에 반영된다. Figure 7는 'bundle_descriptor.xml'의 DTD 그림을 나타낸다. CAMUS Service 프레임워크는 어떠한 이유로 재시작될 때 OSGi Installer는 'ACTIVE' 로 유지하는 모든 객체를 재저장한다.

5. 결론

이 논문에서는 OSGi 기반 CAMUS 서비스 프레임워크를 기술한다. 이 프레임워크는 서비스 lifecycle을 동적으로 가변적으로 관리한다.

먼저, 서비스 lifecycle manager가 context-aware 서비스 프레임워크를 지원하기 위해서는 2가지 요구사항이 존재한다. 하나는 'dynamic management' 이며, 또 하나는 'remote object'이다. Dynamic management는 여러 가지 이유로 가변적으로 변경되어야 하는 프레임워크의 설정에 이용된다. 전형적인 context-aware 응용인 경우에 서비스 프레임워크가 변경될 때 마다 service lifecycle을 지역적으로 관리해야 한다는 것이 부담이 된다. 그러나 이런 변경이 자주 있던, 아니던 간에 매번 각 위치를 방문해야 하는 문제점이 상존한다. 이 문제를 풀기 위해 OSGi 기반 dynamic and remote 서비스 lifecycle을 CAMUS 시스템에 구현하였다.

이런 해결방식은 CAMUS 서비스 개발자가 OSGi bundle의 형태로 CAMUS 서비스를 개발해야 한다는 것을 의미한다. 서비스의 activator를 bundle화 하는 것을 의미하는 것은 아니고, bundle.xml 파일에 삽입되어야 한다. 이러한 구조는 Operator가 대규모의 서비스 프레임워크의 네트워크를 관리할 때 용이하며, remote manager는 중앙에 위치하여 remote service framework의 서비스 lifecycle manager와 통신한다.

참고문헌

[1] H. Kim, Y.-J. Cho and S.-R. Oh, " CAMUS: A Middleware Supporting Context-aware Services for Network-based Robots", IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO) 2005.

[2] OSGi Alliance, OSGi Service Platform The OSGi Alliance Release 3, IOS Press.

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

[4] http://www.caucho.com/hessian/

[5] H. Elzabadani, A. Helal, B. Abdulrazak and E. Jansen, "Self Sensing Spaces: Smart Plugs for Smart Environments ". ICOST 2005, to be held in Montreal, Canada, July 2005.

[6] Zigor Salvador, Alberto Lafuente, and Mikel Larrea, "Jini as a platform for ubiquitous computing", Proceedings of the Simposio sobre Computación Ubicua e Inteligencia Ambiental, UCAmI *2005*, Granada, Spain, Sep 2005.

[7] Zigor Salvador, Raúl Jimeno, Alberto Lafuente, Mikel Larrea, and Julio Abascal, "Architectures for ubiquitous environments", Proceedings of the IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob *2005*, Montreal, Canada, Aug 2005.