한국시뮬레이션학회 논문지
Vol. 18, No. 4, pp. 81-94 (2009. 12)

플러그인 기법을 이용한 임베디드 시스템의 재사용 향상 기법

김철진[1] · 이숙희[2] · 조은숙[3†]

A Reusability Enhancement Technique of Embedded System using Plug-In Method

Chul Jin Kim· Sook Hee Lee· Eun Sook Cho

ABSTRACT

Research of reusability and variability design for embedded system development is insufficient. An embedded system should be designed to support new devices. If extensibility of embedded system is not considered, it is difficult to reconstruct. Currently, the development productivity and reusability of embedded system are very poor, and this will be cased about problems of increasing maintenance and development cost, and decreasing system quality such as software crisis. In this paper, we present framework of embedded system that address those problems of embedded system. We suggest a plug-in technique, based on reusability framework, which can support various devices dynamically. Also, we propose a dynamic Meta model which is base on plug-in technique.

Key words : Reusability, Framework, PlugIn, Dynamic customization, Embedded system

요 약

임베디드 시스템 개발에 있어서 재사용성이나 가변성 설계에 대한 연구가 미흡한 상태이다. 임베디드 시스템의 특성 상 새로운 디바이스를 지원할 수 있도록 설계되어야 하는데, 확장성이 고려되지 않은 임베디드 시스템의 경우 재구축해야 하는 어려움이 있다. 현재 임베디드 시스템의 개발 생산성 및 재사용성이 매우 낮으며, 이런 흐름은 소프트웨어 위기와 같이 임베디드 시스템의 유지보수 비용 증가, 개발비 증가, 시스템의 품질 저하 문제에 직면하게 될 것이다. 본 논문에서는 이와 같은 임베디드 시스템의 문제를 극복하기 위한 기반을 제공하고자 임베디드 시스템의 재사용성 프레임워크를 제시한다. 이 프레임워크를 기반으로 다양한 디바이스를 동적으로 지원할 수 있는 플러그인 기법을 제안하며, 플러그인 기법의 기반이 되는 동적 메타모델를 제시한다.

주요어 : 재사용, 프레임워크, 플러그인, 동적 커스터마이제이션, 임베디드 시스템

1. 서 론

현재 임베디드 소프트웨어 개발에 있어서 소프트웨어 공학적인 기법들이 미흡하게 적용되고 있다. 예를 들면, 임베디드 소프트웨어 개발 방법론, 임베디드 소프트웨어 개발 프로세스, 임베디드 운영체제, 임베디드 미들웨어, 임베디드 소프트웨어 테스팅, 그리고 SoC 모델링 기법 등 다양한 기술들이 연구되고 있으나, 임베디드 시스템의 재사용을 위한 체계적인 공정이나 기법 연구가 미흡한 상태이다[4,16].

홈 네트워크 시스템의 경우, 가정에 존재하는 다양한 디바이스 장치나 프로토콜 등에 대한 가변성이 반드시 고려되어 설계 되어야 한다. 이렇게 해야 임베디드 소프트웨어의 재사용성이나 개발 생산성이 향상될 수 있다[13,21]. 그러나 현재 개발되어 있는 홈 네트워크 시스템에서는 이러한 부분이 미흡한 상태로 설계 및 개발되고 있다[4].

* 이 논문은 2009학년도 인하공업전문대학 교내연구비지원에 의하여 연구되었음.
2009년 8월 21일 접수, 2009년 12월 7일 채택
[1] 인하공전 컴퓨터시스템과
[2] 서경대학교 인터넷정보학과
[3] 서일대학 소프트웨어학과
주 저 자 : 김철진
교신저자 : 조은숙
E-mail; cjkim@inhatc.ac.kr

홈 네트워크 시스템의 가장 큰 문제점 가운데 하나는 디지털 기기(디바이스) 간의 상호 운영성 문제이다. 즉 디지털 디바이스간의 상호 운영성이 보장되어야 한다. 이를 해결하기 위한 방안으로 홈 네트워크 미들웨어가 개발되어 있지만, 아직 실용화하기에는 미흡한 수준에 있다. 두 번째 문제점으로는 홈 네트워크 시스템은 동시에 다양한 가전 디바이스를 제어하기 위한 메커니즘이 미흡하다. 즉, 하나의 이벤트로부터 서로 다른 디바이스를 동시에 제어할 수 있는 동시 제어 메커니즘이 제공되어야 한다. 특히 이 부분은 소프트웨어적으로 처리해야 하는데, 이를 지원하기 위해서는 다양한 디바이스들에 대해 동적으로 행위를 제어할 수 있는 동적 커스터마이제이션 기법이 고려되어야 한다. 세번째 문제점으로는 홈 네트워크 시스템 제품 계열을 볼 때 특정 시스템 마다 적용되는 프로토콜이나 디바이스 유형, 서비스 형태가 공통적인 부분과 특정 제품에 특화된 부분이 존재한다. 이러한 부분을 효율적으로 지원하기 위해서는 공통된 부분은 재사용 단위 컴포넌트로 설계해서 개발해야 하고, 가변적인 부분은 동적으로 플러그-인 되어 처리될 수 있도록 설계되어야 한다. 그러나 대부분의 솔루션들은 제품별로 각각 설계되고, 개발되어 있는 형태를 취하고 있기 때문에, 동일한 디바이스나 서비스에 대한 재사용 비율이 매우 낮은 실정이다.

본 논문에서는 이러한 임베디드 시스템의 문제점을 해결하기 위해 임베디드 소프트웨어의 재사용 측면에서 가변성 부분의 재사용성을 향상시킬 수 있는 재사용 프레임워크를 제안하며, 이를 기반으로 하는 임베디드 시스템 설계 기법을 제안한다. 임베디드 시스템 중에 홈 네트워크 시스템은 도메인 특성 상 다양한 디바이스를 지원해야 하기 때문에 동적으로 다양한 디바이스를 지원할 수 있는 플러그 인 기법을 제안한다. 플러그인 기법은 새로운 디바이스가 추가될 경우 서비스를 임베디드 시스템 내로 동적으로 추가할 수 있는 기법이다.

본 논문에서 제안하는 재사용 프레임워크를 기반으로 임베디드 소프트웨어를 개발할 경우, 임베디드 시스템의 재사용성을 향상시킬 수 있으며, 이를 통해 다양한 도메인의 요구사항을 충족시킬 수 있을 것이다.

본 논문의 구성은 다음과 같다. 2장에서는 임베디드 시스템 개발의 한계점, 기존 동적 커스터마이제이션 기법, 임베디드 시스템 가변성 설계의 한계점에 대해 알아본다. 3장에서는 플러그인 기법을 위한 동적 메타모델을 제안한다. 4장에서는 동적 커스터마이제이션을 위한 재사용 프레임워크와 이 프레임워크를 기반으로 하는 플러그인 기법을 제안한다. 5장에서는 기존 설계 방법과 본 논문에서 제시한 기법에 대해 사례를 들어 실험하고, 재사용성 결과에 대해 분석한다.

2. 관련 연구

2.1 동적 커스터마이제이션

동적 커스터마이제이션 기법에 관련된 연구로서 “Using Component Composition for Self-customizable Systems” [20]와 “Component Interface Pattern”[19]은 모두 복합 컴포넌트를 개발할 때 커스터마이제이션을 하기 위한 기법들을 다루고 있기 때문에 컴포넌트 내부의 행위에 대한 커스터마이제이션 기법은 부분적으로 다루고 있다. 기법[20]은 컴포넌트들 간에 예상하지 못한 변경을 하기 위해 요구되는 인터페이스에 대한 정의를 하여 동적으로 커스터마이제이션이 되도록 하기 위한 기법이다. 기법[19]는 컴포넌트를 조합하기 위해 컴포넌트의 인터페이스에 제공 인터페이스와 요구 인터페이스 뿐만 아니라 구조적 설명(Structural Description)과 행위적 명세(Behavior Specification)을 통해 표현한다. 구조적 설명은 컴포넌트 인터페이스의 서비스 접근 포인트인 채널(Communication Channel)를 이용하여 표현하며 행위적 명세는 패트리 넷(Petri net)을 이용하여 컴포넌트들 간의 연결과 협업을 명세한다. 기법 [20]과 [19]는 모두 복합 컴포넌트를 동적으로 조합하기 위한 명세 기법 및 패턴을 제시하고 있으며 단일 컴포넌트 및 임베디드 소프트웨어의 가변부를 커스터마이제이션하기 위한 기법은 제시하지 못하고 있다.

Componentware에서는 컴포넌트 개발의 여러 단계들을 조합하여 다양한 형태의 프로세스를 구성할 수 있도록 프로세스 패턴들을 제공한다[17]. Componentware는 가변성 설계와 관련된 기법으로는 가변성 정의 장치인 “Import Interface” 정의 기법에 대해서만 제시하고 있다. 따라서 컴포넌트 내부 메시지 흐름에 대한 동적 제어 장치가 없기 때문에 메시지 호출 순서의 추가, 변경, 삭제가 불가능하다. 그리고, 컴포넌트는 하드웨어 컴포넌트처럼 재사용성이 높고 동적 바인딩이 용이하기 위해서는 블랙박스 개념의 컴포넌트 설계 기법이나 가변성 설계 지침이 구체적으로 반영되어 개발되어야 하는데 이 방법론에서는 단지 인터페이스 정의 수준에 머무르고 있다.

CoPAM(Component-Oriented Platform Architecting Method)[2]은 제품 공학(Product Engineering)을 위한 개발 프로세스로서, 플랫폼 엔지니어링(Platform Engineering)과 제품 엔지니어링 2개의 서브 프로세스로 구성되어 있으며, 플랫폼 공학에서는 여러 개의 재사용 컴포넌트들로

구성된 플랫폼을 개발하기 위한 프로세스를 정의하고 있으며, 제품 공학 프로세스에서는 이러한 플랫폼을 이용한 제품을 개발하기 위한 업무들을 정의하고 있다. 이 방법론은 플랫폼 엔지니어링 프로세스에서 여러 도메인에 공통된 부분과 가변적인 부분들을 컴포넌트로 추출하기 위한 활동에 초점을 두고 있다. 따라서, 컴포넌트 내부 설계 기법이나 컴포넌트 내부 메시지 흐름의 가변성 설계와 관련된 구체적인 지침은 제시하고 있지 않다. 다만, 가변성 메커니즘의 유형에 대해서만 언급하고 있다. 본 연구에서 컴포넌트 자체의 범용성과 재사용성 향상에 초점을 두고 제시하는 컴포넌트 내부 메시지 흐름의 동적 변경을 위한 설계 기법을 제시하고 있지 않다.

2.2 임베디드 시스템의 가변성

임베디드 시스템은 하드웨어와 소프트웨어 요소가 함께 결합된 형태이기 때문에 크게 하드웨어 부분과 소프트웨어 부분으로 구별할 수 있으며, 더 상세하게 구분하면 그림 1과 같이 소프트웨어, 하드웨어 플랫폼, 네트워크, 입출력 장치로 크게 구분할 수 있다[4].

이러한 임베디드 시스템 구성 요소들 중에 가변적으로

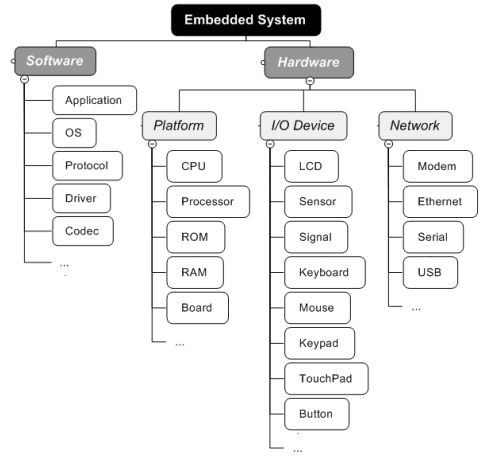


그림 1. 임베디드 시스템 구성 요소

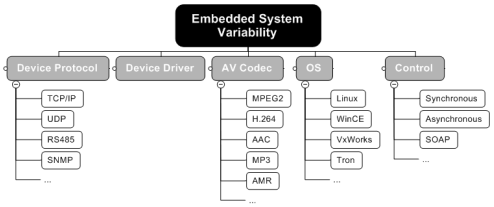


그림 2. 임베디드 시스템의 가변부

변경 가능하도록 구성할 수 있는 요소는 소프트웨어[7]이며 임베디드 소프트웨어를 구성하는 OS(Operation System), 디바이스 드라이버, 코덱(Codec), 프로토콜(Protocol) 등이 임베디드 시스템의 가변부가 될 수 있다.

다양한 도메인 과제 개발 시 특정 요구사항의 변경이 잦은 영역을 가변부(Variation Point)라고 정의하며[3,6], 재사용을 향상시켜 주기 위해서는 이러한 가변부를 특정 요구사항에 대해 변경이 가능하도록 제공해야 한다. 임베디드 시스템의 재사용성을 향상시켜주기 위해 그림 2와 같이 디바이스 프로토콜, 디바이스 드라이버, AV 코덱, OS, 제어(시그널)를 가변적으로 변경할 수 있도록 제공해야 한다.

임베디드 시스템의 가변부를 제공하기 위한 기법으로는 선택 기법과 플러그 인 기법을 이용할 수 있다[1]. 선택 기법은 가변부에 대해 선택 가능한 기능을 설계하여 임베디드 시스템 외부에서 선택하는 기법이다. 선택기법은 매개변수화(Parameterization)에 의해 가능하다[5]. 플러그인 기법은 임베디드 시스템 내부로 변경 기능(디바이스 드라이버, 등)을 입력하여 임베디드 시스템 내부의 기능을 변경하는 기법이다. 본 논문에서는 동적으로 커스터마이제이션하기 위한 플러그인 기법과 프레임워크를 제안한다.

2.3 메타 모델 (Meta Model)

메타모델이란 어떠한 개념들을 구성하는 주요 요소들과 그들 간의 관계를 보여주는 개념 맵이다. UML에서는 MOF(Meta Object Facility)를 통해 메타모델을 정의하고 있는데, 객체지향 모델 작성에 사용되는 UML 모델의 필수 요소와 문법, 구조를 정의하는 메타 모델로 제시하고 있다. 또한 MDA에서는 MOF를 CWM(Common Warehouse Metamodel)이나 UML(Unified Modeling Language)의 메타모델에 대한 공통 모델로 제공하고 있다[22].

메타모델은 특정 개발 플랫폼에 종속되지 않은 형태로 제공되기 때문에 메타 모델에 기반을 두고 모델을 개발할 경우 플랫폼에 따른 다양한 형태의 모델들을 확장하여 개발할 수 있을 뿐만 아니라 모델의 재사용성이 매우 향상되게 된다.

2.4 UML의 상태도 (State Machine Diagram)

본 논문에서 동적 메타모델을 설계하기 위해 UML의 상태도를 활용한다. UML의 상태도는 상태가 변할 때 객체들의 동적인 행위를 나타내며 여러 기능(Use Case)들 사이에 한 객체가 수행하는 기능을 나타내고자 할 때 사

용된다. 상태도는 상태를 통해 한 객체 내에서 이루어지는 수행 과정을 묘사한다[9].

상태도는 활동(Activity)을 갖는 상태, 객체를 처음 동작하도록 하는 시작 상태(Start State), 수행의 끝을 의미하는 종료 상태(Stop State), 그리고 상태 간의 활동 방향을 나타내는 전이(Transition)로 구성된다. 상태는 그림 3과 같이 상태명이 포함되어 있는 라운드 박스(Round Box)로 나타내며 수행할 수 있는 활동(Activity)을 정의한다. 전이는 하나의 상태에서 다음 상태로 이동하는 것을 의미하며 화살표로 나타낸다. 전이 화살표 위에 상태 변화의 사건 및 행위를 정의할 수 있다.

3. 플러그인 기법을 위한 동적 메타 모델

본 장에서는 플러그인 기법의 기반이 되는 메타 모델을 설계하여 플러그인 기법에 대한 절차 및 구성 요소들 간의 관계를 분석한다.

메타모델을 통해 임베디드 시스템에서 요구되는 구성 요소를 파악할 수 있다. 플러그 인 기법은 설정 측면과 서비스 측면으로 구분하여 설계되며 상태도(State Machine Diagram)을 이용하여 메타모델을 설계한다.

플러그 인 기법은 임베디드 시스템 내부에서 변경 요구사항을 제공할 수 없을 경우, 임베디드 시스템 외부에서 변경 기능(디바이스, 프로토콜)을 시스템 내부로 추가하는 기법이다.

플러그인 기법에 의한 가변부 설정은 그림 4과 같이 변경하려고 하는 기능을 요구 인터페이스(Required Interface)를 통해 입력 설정(setup(Plug-Ins))한다. 입력되는 단위는 어댑터, 인터페이스, 클래스와 같이 메타정보가 아닌 실제 기능을 포함한 객체(플러그인스(Plug-Ins))이다. 재사용 프레임워크는 플러그인스를 동적 전개기(HotDeployer)로 전달하여 서비스를 제공할 수 있도록 활성화(Instantiation, Activation) 시킨다.

플러그인 기법의 의해 설정된 메타 정보는 그림 5와 같이 XML 형태의 구조를 가지며 가변부의 어댑터부터 클래스, 함수까지의 설정 정보를 포함하고 있다. 가변부의 메타정보는 가변부 식별자('<Variability Name="_VARIABILITY_NAME_" >'), 가변성 어댑터 명('<Adapter Name="_ADAPTER_NAME_">'), 가변부의 기능을 제공하는 클래스 명('<Class>'), 클래스의 행위 명('<Behavior>'), 실행 환경의 컨텍스트 정보('<Context>')로 구성된다. 가변부 식별자는 가변부 사용 영역에서 가변부를 호출하기

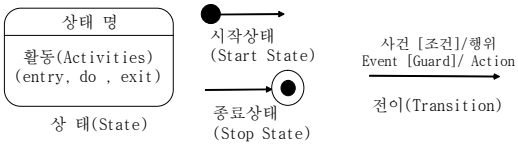


그림 3. 임베디드 시스템 구성 요소

```
...
<Variability Name = "출입통보">
    <Used-By> CE Device Protocol Adapter </Used-By>
    <Used-By> A/V Codec Adapter </Used-By>
</Variability>
<Adapters>
<Adapter Name="CE Device Protocol Adapter">
    <Class>RS485 Device</Class>
    <Behavior>control</Behavior>
    <Context></Context>
    <Class>RF Device</Class>
    <Behavior>control</Behavior>
    <Context></Context>
</Adapter>

<Adapter Name="A/V Codec Adapter">
    <Class>H.263</Class>
    <Behavior>streaming</Behavior>
    <Context></Context>
</Adapter>
</Adapters>
...
```

그림 5. 가변부 메타정보

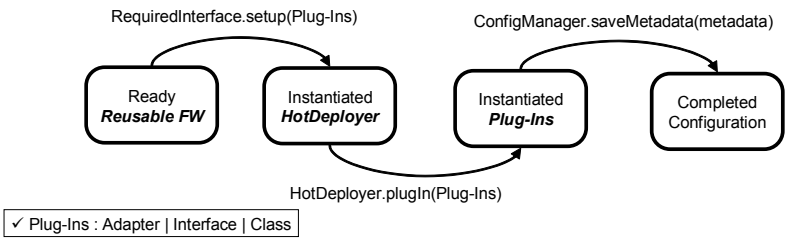


그림 4. 임베디드 시스템의 가변부

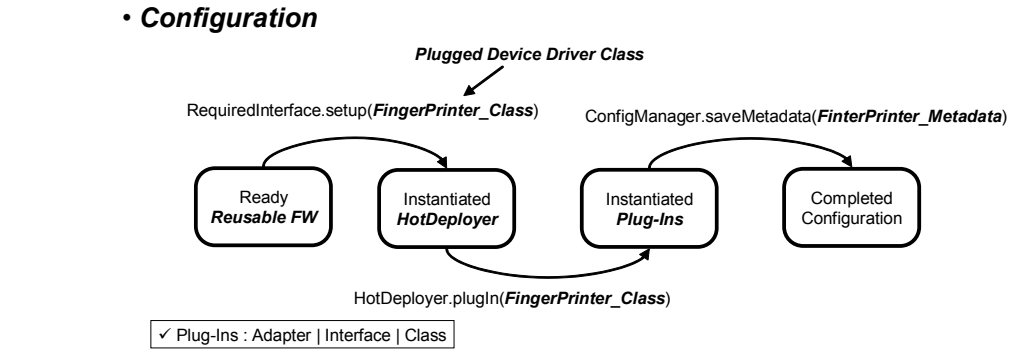


그림 6. 동적 드라이브 설정 메타모델

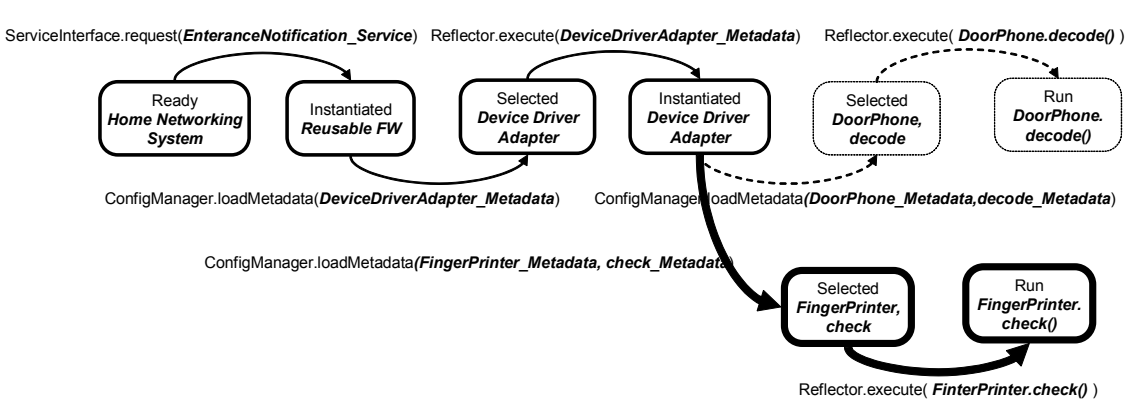


그림 7. 동적 서비스 메타모델

위해 사용되는 가변부 식별자로서 가변부 사용 영역의 코드에 정의 한다. 가변성 어댑터 명은 재사용 프레임워크의 구성요소로서 가변부의 클래스들 중에 동적으로 특정 클래스로 변경하기 위한 중계 역할을 한다. 클래스 명은 어댑터를 통해 선택할 수 있는 클래스를 나타내며 행위 명은 선택된 클래스의 행위(오퍼레이션, 함수)를 나타낸다. 이 메타 정보는 서비스 시점에 호출되어 설정된 가변부의 기능을 제공할 수 있다.

홈 네트워크 시스템은 한 기능에 대해 다양한 디바이스가 연결될 수 있으므로 그림 6과 같이 동적 전개기(HotDeployer)로 현관 지문인식기의 드라이브를 추가(plugIn (FingerPrinter_Class))하여 기존의 열쇠나 도어락(Door Lock)에 의한 현관 출입 기능을 변경할 수 있다.

플러그인 기법에 의한 서비스의 동적 메타모델은 그림 7와 같이 출입 서비스가 도어폰에서 지문인식기로 변경될 경우, 디바이스 드라이브 어댑터(Device Driver Adapter)가 설정 메타정보를 기반으로 지문인식기로 출입서비스를 호출한다. 새로운 기능(디바이스, 프로토콜)이 플러그인 된 이후의 서비스는 메타정보를 기반으로 리플렉터(Reflector.execute(FinterPrinter.check())에 의해 수행된다.

4. 동적 커스터마이제이션을 위한 플러그인 기법

4.1 동적 커스터마이제이션을 위한 재사용 프레임워크

임베디드 시스템의 가변부 처리 메커니즘은 가변부 사용 클래스들이 재사용 프레임워크를 통해 가변부를 접근한다. 임베디드 시스템의 가변성에 해당하는 디바이스 프로토콜, 디바이스 드라이버, AV 코덱, OS, 제어(시그널)를 가변적으로 접근할 수 있는 메커니즘을 제공한다.

임베디드 시스템을 위한 재사용 프레임워크의 구성은 그림 8와 같다. 임베디드 시스템의 가변부들을 중계하는 어댑터와 가변부 처리를 지원하기 위한 핵심 클래스들로 구성된다. 재사용 프레임워크의 어댑터는 임베디드 시스

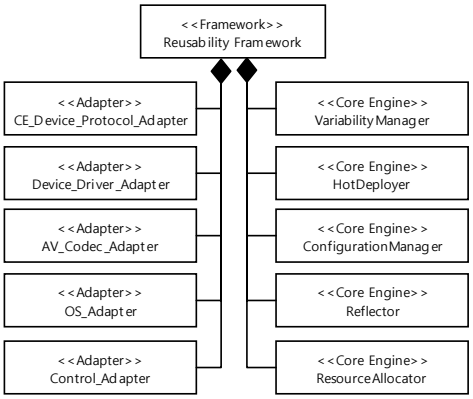


그림 8. 임베디드 시스템을 위한 재사용 프레임워크 구성 요소

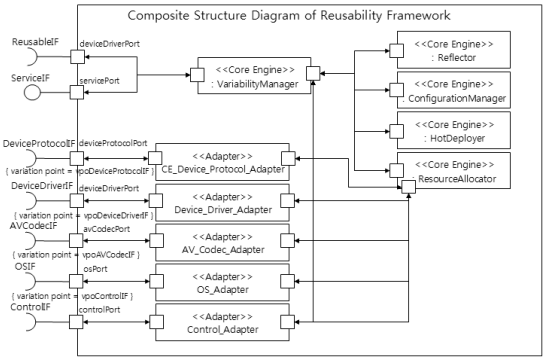


그림 9. 홈 네트워크 임베디드 소프트웨어를 위한 재사용 프레임워크 내부 구조

```
Object result = VariabilityManager.execute( _VARIABILITY_NAME_ , _PARAMETER_) ;
{ variation point = vpo_VARIABILITY_NAME_ }
{ variation point = vpo_PARAMETER_ }
_VARIABILITY_NAME_ : 가변부 식별자
```

그림 10. 가변부 사용 영역에서의 가변부 관리기 실행 코드

템의 가변성인 디바이스 프로토콜, 디바이스 드라이버, AV 코덱, OS, 제어 가변성에 대한 대행 역할을 수행한다. 가변부 처리를 위한 핵심 클래스는 가변성 관리기(Variability Manager), 리플렉터(Reflector), 설정 관리기(Configuration Manager), 동적 전개기(Hot Deployer), 그리고 자원 할당기(Resource Allocator)로 구성된다.

가변부의 변경을 제공하기 위해 재사용 프레임워크는 인터페이스를 제공한다. 서비스 인터페이스는 재사용 프레임워크의 요구 인터페이스(Required Interface)를 통해 설정된 서비스를 제공한다. 임베디드 시스템의 가변성에 대한 요구 인터페이스는 다양한 가변 기능으로 설정될 수 있으며, 이러한 설정은 재사용 요구 인터페이스(Reusable Interface)를 통해 설정된다.

임베디드 시스템의 가변부에 대한 재사용 프레임워크는 어댑터와 핵심 클래스를 통해 동적으로 서비스를 제공할 수 있다. 이러한 재사용 프레임워크의 어댑터와 핵심 클래스들의 내부 구조는 그림 9과 같다. 임베디드 시스템의 가변성에 대한 어댑터들은 재사용 요구 인터페이스를 통해 어떤 기능을 사용할지 설정하며, 재사용 프레임워크의 가변성 관리기는 설정 및 서비스 호출에 대한 대행 역할을 수행한다. 가변성 관리기에 의한 설정은 리플렉터, 설정 관리기, 그리고 동적 전개기를 통해 이루어지며, 설정된 서비스는 가변성 어댑터들에 의해 제공된다. 임베디드 시스템의 자원에 대한 관리는 자원 관리기에 의해 설정되며 가변성 어댑터에 따라 다르게 설정된다. 예를 들면, 운영체제 가변 어댑터가 설정하는 운영체제(Window CE, VxWorks, 등) 에 따라 자원 관리기는 서로 다른 자원(메모리, 등)을 할당한다.

가변성의 표기는 VPM(Variation Point Model)[8]기법을 활용하여 그림 9에서 "{variation point = vpoDevice ProtocolIF}"와 같은 방식으로 표기한다.

재사용 프레임워크의 내부 구성 요소들에 대한 특징은 다음과 같다.

- **가변성 관리기**

가변성 관리기는 가변부를 사용할 수 있도록 중계하는 역할을 하며 재사용 프레임워크의 인터페이스 역할을 한다. 가변부를 사용하는 영역에 의해 특정 가변부를 호출하게 되면 가변성 관리기는 재사용 프레임워크 내의 리플렉터, 설정 관리기, 동적 전개기, 자원 할당기, 그리고 가변성 어댑터들과 상호 작용을 통해 가변부의 특정 기능을 동적으로 호출한다.

가변부 사용 영역에서는 가변부를 사용하기 위해 그림 10과 같이 가변성 관리기를 통해 가변부를 호출한다. 가변부 호출 시 가변부에 대한 식별자('_VARIABILITY_NAME_')와 입력 데이터('_PARAMETER_')를 전달하여 가변부 가변성 관리기가 가변부를 식별하여 요구하는 기능을 호출한다. 가변부 사용 영역에서는 식별자에 의해서 호출하기 때문에 재사용 프레임워크 내부에서 어떤 가변성 어댑터나 클래스로 변경되더라도 전혀 영향을 받지 않는다.

- **설정 관리기**

설정 관리기는 가변부 사용 영역에 의해 사용될 가변부의 메타정보를 관리한다. 설정 관리기는 가변부의 가변부 식별자를 통해 가변부 메타정보를 호출하며, 이러한

```
public class  _UsingVariability_ { // 가변부 사용 클래스
  …
  Object result = VariabilityManager.execute( _VARIABILITY_NAME_ , _PARAMETER_) ;
   // { variation point = vpo_VARIABILITY_NAME_ }
   // { variation point = vpo_PARAMETER_ }
  …                                          _VARIABILITY_NAME_ : 가변성 메타정보 식별자
}                                            _PARAMETER_ : 전달 데이터 (예) 객체 배열
-------------------------------------------------------------------------------------------------
public class  VariabilityManager {   // 가변성 관리기 클래스

  // 설정 관리기에 의해 가변부 메타정보 로딩

  Object result = Reflector.delegate( _ADAPTER_NAME_ , _PARAMETER_ ) ;
    // { variation point = vpo_ADAPTER_NAME_ }
    // { variation point = vpo_PARAMETER_ }
   return result;
}                                                  _ADAPTER_NAME_ : 가변성 어댑터 식별자
-------------------------------------------------------------------------------------------------
public class _ADAPTER_NAME_ {  // 가변성 어댑터 클래스

  // 가변부 클래스 및 행위 메타정보 로딩

  Object result = Reflector.execute( _CLASS_NAME_ , _BEHAVIOR_NAME_ , _PARAMETER_ ) ;
   // { variation point = vpo_CLASS_NAME_ }
   // { variation point = vpo_BEHAVIOR_NAME_ }
   // { variation point = vpo_PARAMETER_ }
  return result;
}                              _CLASS_NAME_ : 가변성 클래스 식별자
                               _BEHAVIOR_NAME_ : 가변성 클래스 내의 특정 오퍼레이션 식별자
```

그림 11. 메타정보 기반의 가변부 선택 및 실행 코드

가변부 식별자는 가변부 사용 영역에서 정의하여 가변성 관리기를 통해 설정 관리기에 전달한다. 그림 10과 같이 가변부 사용 영역에서 전달된 가변부 식별자를 기반으로 설정 관리기는 가변부의 상세한 가변부 메타정보를 얻는다. 설정 관리기는 XML 기반의 메타정보를 관리하기 때문에 동적인 메타정보 관리가 가능하며, 이러한 설정 관리기는 홈 네트워크 시스템의 특성상 도메인에 따라 다양 디바이스로 변경해야 하는 요구사항을 만족시킬 수 있는 도구이다.

- **메타정보 저장소**

메타정보 저장소는 가변부의 메타정보를 포함하고 있는 저장소로서 XML 기반의 가변부 정보를 관리한다(그림 5). 본 논문에서는 제안하는 메타정보는 다중의 어댑터를 통해 다중의 클래스와 다중의 행위를 호출할 수 있는 메타정보를 정의할 수 있다. 이러한 메타정보의 특징은 본 논문이 다양한 기능을 동적으로 변경할 수 기반을 제공한다.

- **리플렉터**

리플렉터는 가변부의 메타정보를 통해 물리적인 가변성 어댑터나 가변성 클래스를 호출하기 위한 도구로서 동적으로 클래스를 호출할 수 있도록 하기 위해 리플렉션(Reflection) 기능을 기반으로 한다. 리플렉션은 메타 형태의 클래스 명(String 타입)과 행위 명(String 타입), 그리고 입력 파라메터(Object Array 타입)를 제공하여 물리적인 클래스의 기능을 호출할 수 있는 메커니즘이다. 이러한 리플렉션 메커니즘은 표준 개발 플랫폼(J2EE, .NET) 에서 제공되고 있으며[11,15], 본 재사용 프레임워크의 리플렉터는 이러한 메커니즘을 커스터마이즈하여 가변부의 메타정보를 처리할 수 있는 기능을 제공한다. 이와 같이 본 재사용 프레임워크의 리플렉터는 가변부 클래스의 다양한 행위뿐만 아니라 다양한 인터페이스의 클래스를 동적으로 변경할 수 있도록 한다.

그림 11에서와 같이 리플렉터는 가변성 어댑터 식별자(_ADAPTER_NAME_)를 이용하여 물리적인 가변성 어댑터를 실행한다(Reflector.delegate(…)). 또한 가변성 어댑터는 리플렉터를 통해 가변성 클래스를 실행한다(Reflector.execute(_CLASS_NAME_,…)). 가변부를 변경하고자 할 때는 가변부 메타정보 만 변경하면 되며, 그림 11의 가변부 사용 구조는 전혀 영향을 받지 않는다. 단지 전달되는 파라메터의 타입은 고려할 여지가 있다. 주로 객체(Object) 배열(Array) 형태로 파라메터를 구성할 수 있다.

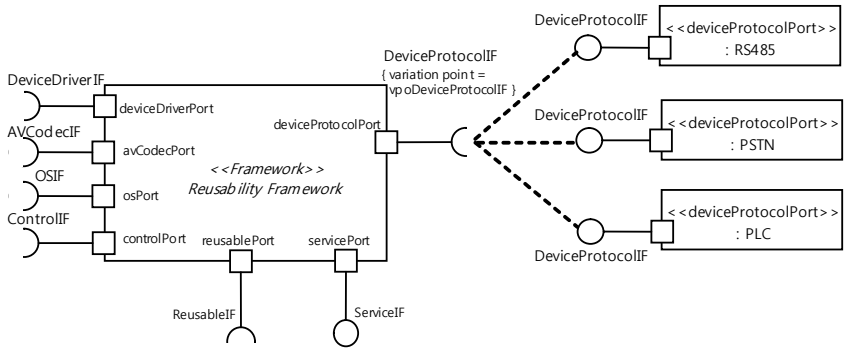


그림 12. 가변성 어댑터에 의한 가변부 구조

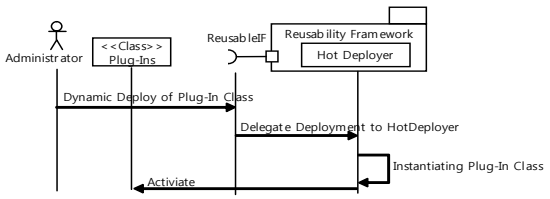


그림 13. 동적 전개기에 의한 가변부 클래스 플러그 인

대부분의 홈 네트워크 시스템의 가변성은 이러한 구조로 가변부를 제공할 수 있으며 디바이스 프로토콜이나 디바이스 드라이버 클래스에 대한 변경도 이러한 구조를 통해 가능하다.

- **가변성 어뎁터**

가변성 어댑터는 가변부를 대행하는 중계자 역할을 하며, 선택된 가변성 어댑터는 메타정보에 정의된 클래스를 호출한다.

재사용 프레임워크의 가변성 어댑터는 다양한 가변 처리를 지원할 수 있도록 요구 인터페이스로 정의되며, 이러한 요구 인터페이스에 맞는 클래스를 통해 가변부를 제공한다. 그림 12은 홈 네트워크 시스템의 가변성인 디바이스 프로토콜, AV 코덱, 운영체제 가변성에 대한 가변부 설계 구조를 나타낸다. 홈 네트워크 시스템은 다양한 디바이스 프로토콜을 통해 다양한 디바이스를 지원하기 때문에 디바이스에 맞는 프로토콜인 RS485 나 PSTN(Public Switched Telephone Network), PLC(Power Line Communication) 등을 가변적으로 선택할 수 있도록 제공해야 한다. 이러한 디바이스 프로토콜을 제공하는 클래스들은 가변성 요구 인터페이스인 'DeviceProtocolIF'를 만족하도록 설계되어야 한다. AV 코덱과 운영체제 가변성도 이와 동일하게 요구 인터페이스를 구현한 가변부를 설계한다.

어댑터는 메타정보가 변경되면 물리적인 클래스도 변경되어 다른 기능을 제공하도록 한다. 기존 연구와 다르게 본 논문에서는 어댑터를 통해 기능을 제공하는 클래스를 동적으로 변경할 수 있으며, 어댑터를 변경하여 가변부를 사용하는 클래스에서 전혀 다른 기능을 호출할 수 있다.

- **동적 전개기**

동적 전개기는 홈 네트워크 시스템 내부에서 가변성을 제공할 수 없는 경우 시스템 외부에서 요구하는 기능 클래스를 시스템 내부로 플러그하기 위한 도구이다. 플러그의 개념은 시스템 패키지 내에 요구하는 클래스를 포함하는 것이 아니라 시스템 운영 환경에 맞게 객체가 생성(Instantiate)되는 것을 의미 한다.

그림 13과 같이 재사용 인터페이스 통해 디바이스 프로토콜 가변성을 시스템 외부에서 제공해야 하는 경우에는 재사용 프레임워크의 동적 전개기에서 요구하는 가변부 클래스('Plug-Ins')의 객체를 초기화하여 현 운영되는 시스템에서 가변적으로 접근할 수 있도록 한다.

- **자원 할당기**

자원할당기는 임베디드 시스템의 운영 환경에 변경에 따른 자원 할당을 지원하기 위한 도구로서 홈 네트워크 시스템의 경우 주로 운영체제 변경에 따른 메모리나 지원 전압의 변경을 설정한다.

4.2 동적 플러그인 기법

가변부 플러그 인 기법은 임베디드 시스템 내부에서 가변 요구사항을 제공할 수 없을 경우에 임베디드 시스템 외부에서 가변 클래스를 시스템 내부로 플러그-인 하는 기법이다.

재사용 프레임워크 기반의 가변부 플러그 인 기법의 설정 및 서비스에 대한 프로세스는 다음과 같다. 그림 14

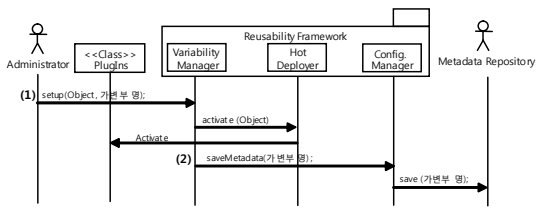


그림 14. 동적 전개기에 의한 가변부 클래스 플러그 인

의 (1)~(2) 단계는 가변성 설정 단계이며, 그림 15의 (3)~(7) 단계는 가변성 서비스 단계이다.

(1) 가변성 어댑터 및 클래스 플러그-인

그림 14과 같이 객체(Object) 타입을 입력 매개변수로 하는 동적 전개기 기능에 외부 가변성 어댑터 및 클래스를 입력한다(setup(Object, 가변부명)). 재사용 프레임워크 내부에서 동적으로 플러그 인 된 가변성 어댑터 및 클래스를 활성화(Activation) 시킨다.

(2) 가변부 메타정보 설정

그림 14과 같이 새로 플러그인 되는 객체에 대해서는 메타정보가 정의되어 있지 않으므로 새로운 클래스 메타정보를 설정한다. 설정되는 메타정보는 신규 어댑터 및 신규 클래스, 오퍼레이션에 대한 식별자가 된다.

(3) 서비스 요청

사용자는 그림 15와 같이 임베디드 시스템의 서비스를 요청한다. 요청 서비스가 가변부를 포함하고 있으면, 그림 15와 같이 재사용 프레임워크를 호출한다(execute()). 임베디드 소프트웨어의 가변부 사용 클래스에 정의되는 가변부 식별자는 가변부가 임베디드 소프트웨어 내부 클래스(하드웨어 드라이버) 인지, 외부 클래스 인지를 고려하지 않고 호출할 수 있다. 서비스를 제공하는 내부 클래스와 외부 클래스 모두 시스템 운영 시점에 활성화 만 되어 있다면 호출될 수 있다.

(4) 재사용 프레임워크(가변성 관리기) 호출

가변부를 사용하는 클래스는 가변부를 호출하기 위해 그림 15와 같이 재사용 프레임워크의 가변성 관리기를 호출한다(execute()). 가변성 관리기는 가변부 식별자를 기반으로 가변부의 메타정보를 호출하며 리플렉터를 통해 가변성 어댑터를 호출한다. 이때 호출되는 어댑터는 재사용 프레임워크 내의 홈 네트워크 임베디드 시스템을 위한 가변성 어댑터이거나 외부 어댑터 일 수 있다.

(5) 메타정보 호출

가변부 식별자를 통해 메타정보 저장소 내의 메타정보를 호출한다. 가변부 식별자는 가변부가 변경되더라도 가변부 사용 클래스에서 사용되는 동안 변경되지 않는다. 호출되는 메타정보는 가변부를 나타내는 어댑터, 클래스, 행위(오퍼레이션), 컨텍스트 정보와 같이 가변부 선택 기법과 동일하지만 임베디드 시스템의 외부 클래스에 대한 메타정보이다. 가변부 플러그 인 기법을 위한 가변부의 메타정보는 가변부의 식별자를 기준으로 단일 클래스 및 다중 클래스를 호출하며 임베디드 시스템 내부와 외부 클래스를 조합하여 구성할 수 있다.

(6) 가변성 어댑터 선택 및 실행

가변성 관리기는 호출된 메타정보를 기반으로 가변부를 연결하는 어댑터를 선택 및 실행한다. 어댑터는 가변부의 클래스에 대한 인터페이스를 통해 접근하므로 다른 클래스로 변경 되더라도 가변부를 사용하는 다른 영역은 영향을 받지 않는다. 플러그 인 기법인 경우 메타정보가 외부 어댑터로 설정되어 있다면 전혀 다른 클래스 타입(인터페이스)으로 변경하는 경우이다.

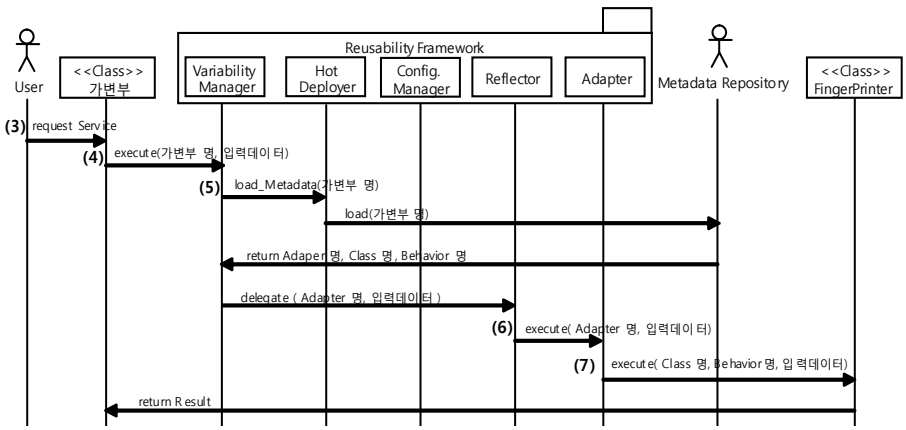


그림 15. 가변성 서비스 순차도

(7) 가변 클래스 선택 및 실행

가변성 어댑터는 메타정보에 선택된 임베디드 시스템의 내부 및 외부 클래스의 행위를 호출한다. 다중 클래스 호출 시 임베디드 시스템 내부 인터페이스를 따르는 클래스를 호출하거나 외부 어댑터에 의해 외부 클래스를 호출할 수 있다. 입력 및 결과 데이터는 재사용 프레임워크를 통해 전달된다.

5. 사례 연구

사례 연구는 홈 네트워크 시스템에서 방문자가 현관의 도어폰(Door Phone)을 눌렀을 때 방문 통보를 가정의 다양한 가전 디바이스(월패드(Wall Pad), 주방폰, 셋탑, 핸드폰, 등)에 알리는 기능이다. 이 때 가정의 다양한 가전 디바이스는 개발되는 도메인에 따라 다르게 적용될 수 있기 때문에 가변적인 처리가 요구된다. 따라서, 기존의 객체지향 기법으로 설계한 사례 2가지와 본 논문에서 제안하는 재사용 프레임워크 기반의 플러그인 기법을 이용하여 설계한 사례를 비교 검증한다. 본 논문에서 제시하는 홈 네트워크 임베디드 시스템의 재사용 프레임워크의 재사용성을 검증하기 위해 기존의 설계 사례와 비교하여 재사용성이 향상됨을 증명한다.

5.1 플러그인 기법의 동적 메타 모델 설계

그림 16는 플러그인 기법에 의해 가변부를 설정하기 위한 설계로서 메타모델과의 관계성을 모여주고 있다. 선택기법과 다르게 플러그인 기법은 외부에서 새로운 서비스를 정의하여 내부로 플러그인 하는 기법이다. 따라서 그림 16에서와 같이 서비스 클래스와 설정정보를 입력하면 서비스 클래스가 동적 전개기에 의해 활성화된다.

본 사례는 출입통제를 도어폰에서 지문인식기로 변경하고자 할 경우 지문인식을 위한 서비스를 플러그인하면 재사용 프레임워크에서 활성화시켜 주는 사례이다. 본 사례에서도 플러그인 기법의 동적 메타모델이 동적설계를 위한 중요한 메타정보를 제공함을 파악할 수 있다.

플러그인 기법에 의한 서비스 설계는 선택기법의 서비스와 동일하며, 단지 설정 방법에 차이가 있다. 또한 시스템 내부 클래스가 아닌 외부 클래스에 대한 서비스를 요청하는 것이 차이가 있다.

5.2 플러그인 기법을 이용한 동적 설계

기존의 객체지향 기법으로 설계한 사례와 본 논문에서 제안하는 재사용 프레임워크를 사용하여 설계한 사례를 비교 검증한다.

그림 17은 홈 네트워크 솔루션의 출입 방문 시 'Visiting' 클래스와 가전 디바이스 제어 프로토콜인 'RS485' 클래스와 관계를 통해 주방폰에 방문자가 도착했음을 통보한다(sendSignal()). 본 사례는 'Visiting' 클래스와 'RS485' 클래스가 직접적으로 결합되어 설계되므로 확장성이 높지 않다.

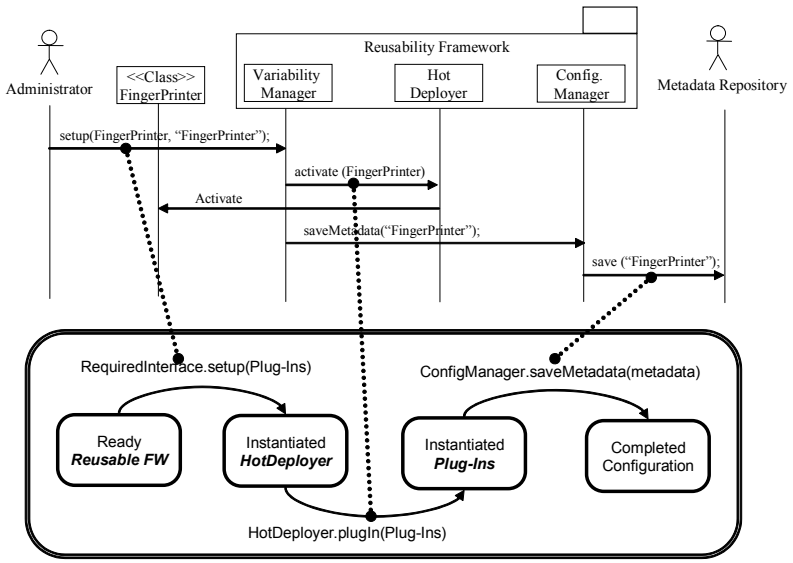


그림 16. 가변부 설정 동적 설계(플러그인 기법)

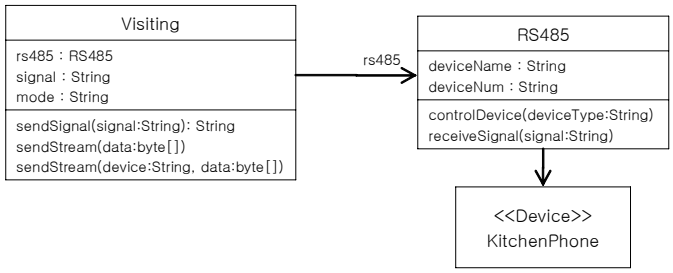


그림 17. 출입통보 설계 사례 1

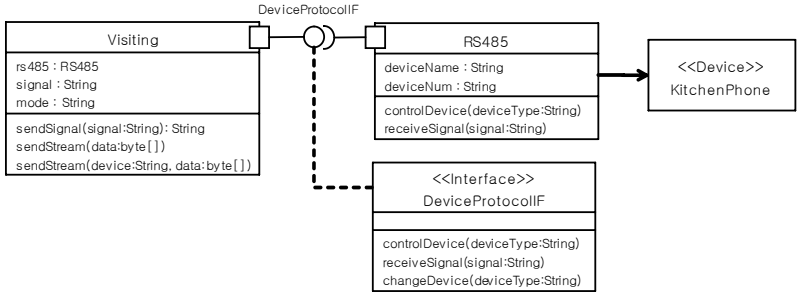


그림 18. 출입통보 설계 사례 2

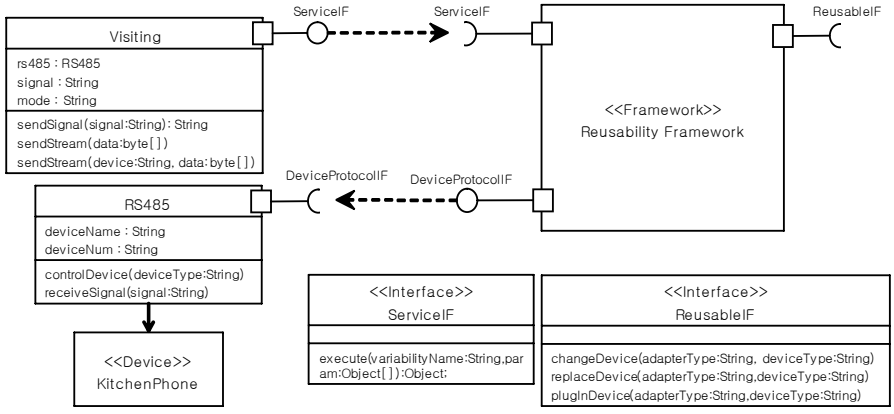


그림 19. 출입통보 설계 사례 3(Reusability Framework 이용)

출입통보 설계 사례 2는 그림 18와 같이 'Visiting' 클래스와 가전 디바이스 제어 프로토콜인 'RS485' 클래스와 직접 연결되지 않으며 인터페이스를 통해 연결되도록 설계되었다. 본 설계에서는 다른 프로토콜로 변경이 가능할 수 있으나, 다중의 프로토콜로는 확장이 불가능하다. 홈 네트워크 시스템에서 출입통보가 특정 시점에 하나의 디바이스에만 서비스를 제공하는 것이 아니라, 가변적으로 다양한 디바이스에 서비스를 제공할 수 있어야 한다. 따라서 다중의 프로토콜을 제공해 줄 수 있어야 한다.

본 논문에서 제안하는 재사용 프레임워크를 이용한 설계 사례는 그림 19과 같으며 'Visiting' 클래스와 가전 디바이스 제어 프로토콜인 'RS485' 클래스가 'Reusability Framework'를 통해 연결된다. 'Visiting' 클래스는 디바이스 제어 프로토콜 'RS485' 클래스와 직접 연결되지 않는다.

그림 20과 같이 'Reusability Framework'을 통해 다양

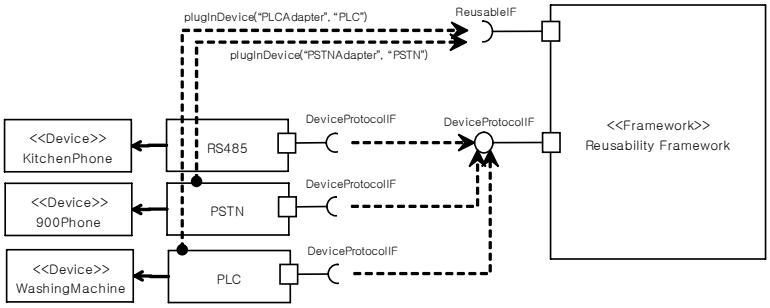


그림 20. Plug-In 기법을 통한 다양한 디바이스 프로토콜 지원

표 1. ISMS 정책 테이블

Quality attribute	Sub-Characteristic			
Functionality	Suitability	Accuracy	Inter-operability	Security
Reliability	Maturity	Fault tolerance	Recoverability	
Usability	*Understandability*	Learnability	Operability	Attractiveness
Efficiency	Time behavior	Resource utilization		
Maintainability	Analyzability	*Changeability*	Stability	Testability
Portability	Adaptability	Installability	Co-existence	*Replaceability*

표 2. 메트릭 측정 결과

Understanding				
Survey #	U1	U2	U3	U4
NM	28	28	28	28
NP(Case1)	13	12	12	10
NP(Case2)	12	14	12	14
NP(PlugIn)	9	10	12	11

Changeability			
Survey #	C1	C2	EC3
NM	28	28	28
NP(Case1)	4	5	3
NP(Case2)	6	5	7
NP(PlugIn)	12	17	16

Replaceability				
Survey #	R1	R2	ER3	ER4
NM	28	28	28	28
NP(Case1)	3	6	6	5
NP(Case2)	6	3	7	9
NP(PlugIn)	17	16	14	14

Extensibility				
Survey #	EE1	EE2		
NM	28	28		
NP(Case1)	2	3		
NP(Case2)	2	4		
NP(PlugIn)	18	17		

Generality				
Survey #	EG1			
NM	28			
NP(Case1)	3			
NP(Case2)	6			
NP(PlugIn)	17			

NM : Number of Measurements
NP : Number of Positive Responses

<Understandability>
U1.1 When the variability of component is designed using this technique, can you understand this technique easily ?
Yes( ) No( )
U1.2 When the variability of component is designed using the existing method, can you understand the existing method easily ?
Yes( ) No( )
U2.1 When the component is customized using this technique, can you understand the variability ?
Yes( ) No( )
U2.2 When the component is customized using the existing method, can you understand the variability ?
Yes( ) No( )
U3.1 When the component is customized using this technique, do exist the required data for variability ?
Yes( ) No( )
U3.2 When the component is customized using the existing method, do exist the required data for variability ?
Yes( ) No( )
U4.1 When the component is customized using this technique, can you understand input and output ?
Yes( ) No( )
U4.2 When the component is customized using the existing method, can you understand input and output ?
Yes( ) No( )

그림 21. 이해성 조사 리스트

한 디바이스 프로토콜을 지원할 수 있다. 'RS485', 'PLC', 'PSTN' 프로토콜 클래스는 'DeviceProtocolIF' 인터페이스를 따르며, 'Reusability Framework'에서 다양한 신규 프로토콜인 PSTN, PLC, 등을 플러그인 하여 새로운 디바이스 프로토콜을 동적으로 제공할 수 있다.

5.3 평가

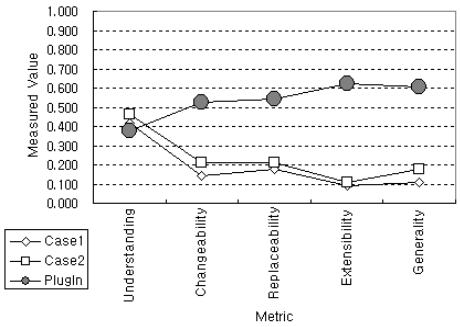
재사용성을 검증하기 위해 기존 설계 기법과 본 플러그인 기법을 적용한 설계를 대상으로 조사한다. 조사 대상자는 3가지 설계 모두 참여한 개발자를 대상으로 조사되었다. 조사 대상자는 기존 설계 기법에 대한 경험을 가지고 있으며, 본 검증의 독립성과 객관성을 보장하기 위해 본 논문에서 제시하는 기법 개발에는 참여하지 않은 개발자를 대상으로 하였다. 조사 리스트는 소프트웨어 품질 측정 메트릭인 ISO 9126[10]을 이용하였다.

표 1에서 보는 것과 같이 소프트웨어 품질 측정 메트릭[10,12,14] 중에 재사용성과 관련된 메트릭으로서 이해성(Understandability), 변경성(Changeability), 교체성(Replaceability), 확장성(Extensibility), 일반성(Generality)를 기본으로 하여 다음과 같은 리스트를 통해 재사용성을 조사하였다.

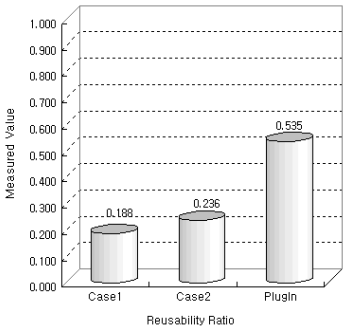
그림 21의 이해성 조사 리스트처럼 변경성, 교체성, 확장성, 그리고 일반성 조사 리스트도 작성되었다. 위의 조사 리스트는 ISO 9126의 메트릭 스펙을 기반으로 긍정적 응답(Yes) 과 부정적 응답(No)을 유도할 수 있도록 정의하였다. 각각의 측정 메트릭에 의해 조사된 결과는 다음과 같다.

표 2는 그림 21의 조사 리스트를 기반으로 조사된 결과이며, 조사 리스트에 대해 긍정적인 대답과 부정적인 대답에 대한 통계적인 수치를 나타낸다. 이해성을 예로 들면, 질문 리스트는 U1, U2, U3, U4 이며, 각 질문에 대한 질문 참여수(NM)는 각 질문 별로 28회씩, 사례1 설계(Case1)에 대한 긍정적 대답 수(NP)는 질문 별로 13, 12, 12, 10회, 사례2 설계(Case2)에 대한 긍정적 대답 수(NP)는 질문 별로 12, 14, 12, 14회, 본 논문에서 제시한 플러그인 기법 설계(PlugIn)에 대한 긍정적 대답 수(NP)는 질문 별로 9, 10, 12, 11회로 조사되었다. 나머지 메트릭에 대해서도 동일한 방법으로 조사 결과가 수집되었다.

5가지 메트릭에 대한 측정 결과는 그림 22 (1)과 같이 그래프로 나타낼 수 있다. 긍정적 대답 수에 대한 결과를 설계 사례별(Case1, Case2, PlugIn)로 비교하며, 이해성 메트릭의 측정 결과는 기존 기법에 비해 본 기법의 이해



(1) Measurement by Metrics



(2) Average Measurement for Design Cases

그림 22. 재사용성 메트릭 측정 그래프

성이 어려움을 나타낸다. 변경성의 측정 결과는 기존의 기법이 거의 변경하기 위한 가변성을 제공하고 있지 않기 때문에 많은 차이를 보이고 있음을 알 수 있다. 교체성, 확장성, 일반성도 기존 설계 기법에 비해 재사용성이 향상될 수 있음을 알 수 있다.

그림 22의 (2)는 5가지 메트릭의 측정 결과에 대한 평균 그래프로서, 설계 사례1의 재사용성 수치는 0.188, 설계 사례2는 0.236, 그리고 본 논문에서 제시한 플러그인 기법을 통한 설계의 경우 재사용성 수치가 0.535로 기존 설계 기법에 비해 재사용성이 2배이상 향상될 수 있음을 판단할 수 있다.

6. 결 론

임베디드 시스템의 재사용 프레임워크를 기반으로 동적 커스터마이제이션을 지원할 수 있는 플러그인 기법에 대해 알아보았다. 플러그인 기법은 임베디드 시스템의 다양한 추가 디바이스를 지원할 수 있는 특징을 제공하며 재사용성 향상의 핵심 기술이라고 할 수 있을 것이다. 사례연구를 통해 다양한 기존 설계에 비해 재사용성이 향상될 수 있음을 증명하였다. 또한, 플러그인 기법 설계의 기반이 되는 동적 메타모델을 제시하였다. 향후에서는 재사용 프레임워크를 모바일(Mobile) 서비스에 활용하여 모바일 서비스 개발의 재사용성이 향상됨을 연구하고자 한다.

참 고 문 헌

1. 김철진, 김수동, “컴포넌트 행위 커스터마이제이션 기법”, 「한국정보과학회논문지(B)」, 제30권 제3.4호 2003년4월.
2. America P., Obbink H., and Ommering R., F. V. D. Linden, “CoPAM: A Component-Oriented Platform Architecting Method Family for Product Family Engineering”, The First Software Product Line Conference(SPLC), Kluwer International Series in Software Engineering and Computer Science, Denver, Colorado, USA, p.15, 2000.
3. Anastasopoulos M. and Gacek C., Implementing Product Line Variabilties, Technical Report IESE Report No. 089. 00/E, Version 1.0, Fraunhofer Institute for Experimental Software Engineering (IESE), November 2000.
4. Axel J., Modeling Embedded System and SOCs, Mogan Kaufmann, 2004.
5. Becker M., “Generic Components: A Symbiosis of Paradigms”, 2nd International Symposium on Generative and Component-Based Software Engineering(GCSE'00), Erfurt, October 2000.
6. Coplien J., Hoffman D., and Weiss D., “Commonality and Variability in Software Engineering”, IEEE Software, pp. 37-45, November 1998.
7. David E. S., An Embedded Software Primer, Addison Wesely, 1999.
8. Diana L. and Gomaa H., “Modeling variability in software product lines with the variation point model”, Science of Computer Programming 53, 2004.
9. Fowler M., and Scott K., UML Distilled: Applying the Standard Object Modeling Language, Addison-Wesley, 1998.
10. ISO/IEC JTC1/SC7 N2419 “DTR 9126-2: Software Engineering-Product Quality Part 2-External Metrics”, 2001.
11. Java Developer Network, http://java.sun.com.
12. Jeffrey S. P., “Measuring Software Reusability”, IEEE Software, 1994.
13. Jose C., “Next-Generation Object-Oriented Software Analysis and Design Methodology”, at URL: http://www.hpl.hp.com/fusion/ma_961007.html, 1996.
14. Kim S. D. and Park J. H., “C-QM: A Practical Quality Model for Evaluating COTS Components”, IASTED, SE 2003.
15. Microsoft Developer Network, http://msdn2.microsoft.com.

16. Raj K., Embedded Systems: Architecture, Programming and Design, McGraw Hill, 2004.
17. Rausch A. “Software Evolution in COMPONENTWARE Using Requirements/Assurances Contracts”, Proceedings of the 22th International Conference on Software Engineering, 06/2000.
18. Ready J. and Howard D., “Structuring Real-Time Application Software Part1”, VMEbus Systems, pp. 33-45, April, 1991.
19. Silva R. P., “Component Interface Pattern”, Procs. Pattern Languages of Program, 1999.
20. Sora I., Verbaeten P. and Berbers Y., “Using Component Composition for Self-Customizable Systems”, Workshop on Component-Based Software Engineering, ECBS 2002, April 8-11, 2002, Lund, Sweden.
21. Szyperski C., Component Software: Beyond Object-Orie nted Programming, Addison-Wesley, 2002.
22. UML Specification v2.0, OMG,Inc., January, 2006.

김 철 진 (cjkim@inhatc.ac.kr)
1996 경기대학교 전자계산학과(학사)
1998 숭실대학교 컴퓨터공학부(공학석사)
2004 숭실대학교 컴퓨터공학부(공학박사)
2004 가톨릭대학교 컴퓨터정보공학부 강의전임교수
2004～2009 삼성전자 책임연구원
2009 ~ 현재 인하공업전문대학 컴퓨터시스템과 교수

관심분야 : CBD, Component Customization, Embedded Software, SOA

이 숙 희 (oleesh@skuniv.ac.kr)
1979 숙명여자대학교 독문학과 졸업(학사)
1982 동국대학교 경영대학원 정보처리학과 졸업(석사)
1991 성균관대학교 대학원 통계학과 졸업(박사)
1987～1993 동신대학교 전자계산학과 교수
1993년～현재 서경대학교 인터넷정보학과 교수

관심분야 : 객체지향 소프트웨어 개발 방법론, 소프트웨어 테스트, 컴포넌트기반 소프트웨어공학

조 은 숙 (escho@seoil.ac.kr)
1993 동의대학교 전산통계학과 졸업(이학사)
1996 숭실대학교 대학원 컴퓨터학과(공학석사)
2000 숭실대학교 대학원 컴퓨터학과(공학박사)
2000～2005 동덕여자대학교 정보학부 강의전임교수
2005～현재 서일대학 소프트웨어과 교수

관심분야 : CBSE, Embedded Software, Service-Oriented Computing, SOA