DOI: 10.3745/KIPSTD.2010.17D.6.431

임베디드 시스템의 재사용 프레임워크에 대한 정형명세

조 은 숙[†] · 김 철 진[††] · 송 치 양[†††]

요 약

임베디드 시스템은 하드웨어와 소프트웨어 요소들이 서로 결합된 시스템이기 때문에 설계 시 실시간성(Real-time), 반응성(Reactive), 소규모(Small Size), 경량화(Low Weight), 안전성(Safe), 신뢰성(Reliable), 견고성(Harsh Environment), 저비용(Low Cost) 등의 요소들을 고려하여 설계해야 한다. 그러나 현재 임베디드 시스템 개발에 이러한 요소들을 반영한 설계 기법들이 미비하게 제시되어 있다. 특히 임베디드 시스템 개발에 있어서 프레임워크를 기반으로 임베디드 시스템을 개발하는 형태가 거의 이뤄지고 있지 않다. 이로 인해 현재 개발되어 있는 임베디드 시스템들의 내부 코드들을 살펴보면 시스템 작동과 전혀 상관이 없는 코드들이 무수하게 잔재하고 있으며, 임베디드 시스템 개발에 있어서 재사용성이나 가변성에 대한 고려가 미흡한 실정이다. 따라서 본 연구에서는 임베디드 시스템의 재사용성을 향상시키기 위해 제안했던 재사용 프레임워크 설계에 대한 완전성이나 일관성을 보장하기 위해 Z를 이용하여 프레임워크 설계에 대한 정형 명세 기법을 제시하고자 한다. 또한 Z 언어를 통해 명세한 결과를 Z-Eves Tool을 통해 Z 모델 체킹을 수행하여 프레임워크 설계의 명확성을 보이고자 한다.

키워드 : 임베디드 시스템, 재사용 프레임워크, 가변성, 정형명세

A Formal Specification of Reusable Framework of Embedded System

Eun-Sook Cho[†] · Chul-Jin Kim[††] · Chee-Yang Song[†††]

ABSTRACT

Because embedded system is combined system of hardware and software, we should design by considering elements such as real-time, reactive, small size, low weight, safe, reliable, harsh environment, low cost, and so on. However, those are poorly reflected on current embedded system development. Especially, there is few existed framework-based embedded system development. As a result, there are many internal codes which is not related with system operation in currently developed embedded system, and reusability or variability is not considered into embedded system development. Therefore we propose a formal specification technique using Z language to guarantee completeness or consistency of design of reusability framework proposed for improving reusability of embedded system. Also we assure correctness of framework design by checking Z model through Z-Eves Tool.

Keywords : Embedded System, Reusability Framework, Variability, Formal Specification

1. 서 론

임베디드 시스템이라 함은 특정한 기기에 주어진 작업을 수행하도록 구동시키는 시스템이라 할 수 있다. 첨단 기능이 들어있는 가전제품이나 컴퓨터, 엘리베이터, 공장 자동화 시스템 등등 특정한 기기를 운용할 수 있는 운용체제라면 임베디드 시스템이라 할 수 있다[1, 2]. 임베디드 시스템 가운데 하드웨어를 제외한 나머지 부분을 임베디드 소프트웨어라고 말할 수 있다. 이러한 임베디드 시스템은 기존 시스템과 달리 하드웨어와 소프트웨어 요소들이 서로 결합된 시스템이기 때문에 임베디드 시스템 설계 시 실시간성(Real-time), 반응성(Reactive), 소규모(Small Size), 경량화(Low Weight), 안전성(Safe), 신뢰성(Reliable), 견고성(Harsh Environment), 저비용(Low Cost) 등의 요소들을 고려하여 설계해야 한다[1]. 그러나 현재 임베디드 시스템 개발에 이러한 요소들을 반영한 설계 기법들이 미비하게 제시되어 있다. 특히 임베디드 시스템 개발에 있어서 프레임워크를 기반으로 임베디드 시스템을 개발하는 형태가 거의 이뤄지고 있지 않다. 이로 인해 현재 개발되어 있는 임베디드 시스템

※ 본 논문은 2009년 서일대학학술연구비에 의해 연구되었음.
† 정 회 원 : 서일대학 컴퓨터 소프트웨어과 조교수
†† 정 회 원 : 인하공업전문대학 컴퓨터시스템과 조교수(교신저자)
††† 정 회 원 : 경북대학교 컴퓨터정보학부 조교수
논문접수 : 2010년 7월 9일
수 정 일 : 1차 2010년 8월 18일
심사완료 : 2010년 8월 18일

들의 내부 코드들을 살펴보면 시스템 작동과 전혀 상관이 없는 코드들이 무수하게 잔재하고 있으며, 임베디드 시스템 개발에 있어서 재사용성이나 가변성을 반영하여 설계하는 사례 또한 거의 존재하지 않고 있다. 이러한 흐름이 계속 이어지게 될 경우, 향후 임베디드 시스템은 유지보수의 어려움, 개발비의 증가, 유비보수비의 증가, 시스템 품질 저하 등과 같은 다양한 위기에 직면하게 될 것이다. 따라서 본 연구에서는 임베디드 시스템의 재사용성을 향상시키기 위해 재사용 프레임워크를 제안하고, 아울러, 제안된 프레임워크 메타모델의 정확성을 입증하기 위해, Z를 사용하여 정형적으로 명세하고 Z-Eves Tool을 통해 Z 모델 체킹을 수행하여 프레임워크의 정확성을 제시하고자 한다[3, 4].

본 논문은 다음과 같이 구성된다. 2장에서는 관련 연구로 현재 임베디드 시스템 개발에 있어서의 한계점을 제시한다. 3장에서는 본 논문에서 제시하는 임베디드 시스템의 재사용성 향상을 위한 프레임워크와 이에 대한 정형 명세 기법을 제시한다. 4장에서는 본 논문에서 제시하는 재사용 프레임워크의 정형명세에 대한 검증 결과를 제시한다. 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

2.1 현존 임베디드 시스템 개발의 한계성

현존하는 임베디드 시스템 개발에 있어서 가장 큰 문제점 가운데 하나는 재사용성을 고려하지 않고 설계 및 개발이 진행되고 있다는 것이다. 이는 재사용성의 문제점만 초래할 뿐만 아니라 새로운 유형의 디바이스나 프로토콜에 대한 시스템 확장성 및 상호 연동성의 문제점도 야기시키게 된다. 두 번째 문제점으로는 현존 임베디드 시스템은 동시에 다양한 디바이스들을 제어하기 위한 메커니즘이 미약하게 지원되고 있다. 즉, 하나의 이벤트로부터 서로 다른 디바이스를 동시에 제어할 수 있는 동시 제어 메커니즘이 제공되어야 한다[5, 6]. 특히 이 부분은 소프트웨어적으로 처리해야 하는데, 이를 지원하기 위해서는 다양한 디바이스들에 대해 동적으로 행위를 제어할 수 있는 동적 커스터마이제이션 기법이 고려되어야 한다.

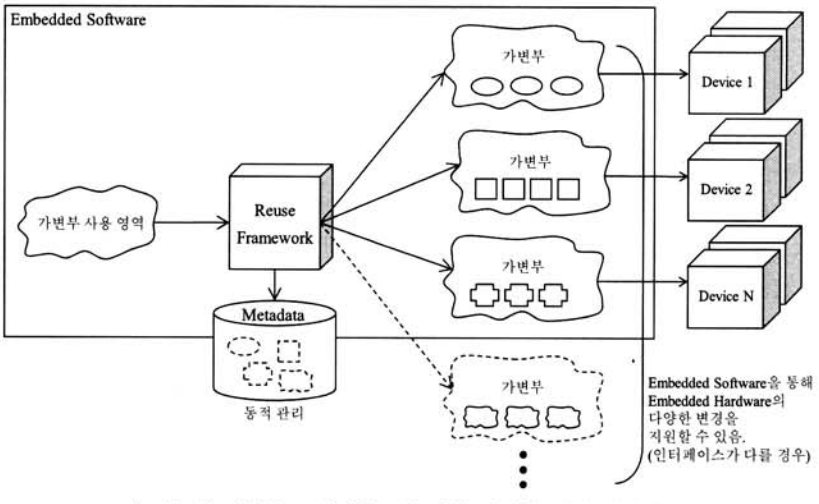
세번째 문제점으로는 임베디드 시스템 제품 계열을 볼 때 특정 시스템 마다 적용되는 프로토콜이나 디바이스 유형, 서비스 형태가 공통적인 부분과 특정 제품에 특화된 부분이 존재한다. 이러한 부분을 효율적으로 지원하기 위해서는 공통된 부분은 재사용 단위 컴포넌트로 설계해서 개발해야 하고, 가변적인 부분은 동적으로 플러그-인 되어 처리될 수 있도록 설계되어야 한다[7-10]. 그러나 대부분의 솔루션들은 제품별로 각각 설계되고, 개발되어 있는 형태를 취하고 있기 때문에, 동일한 디바이스나 서비스에 대한 재사용 비율이 매우 미흡한 실정이다. 본 연구에서는 이러한 부분을 향상시키기 위해 재사용 프레임워크를 제시하고 제시된 프레임워크를 검증할 수 있는 정형명세 기법을 제시하고자 한다.

3. 임베디드 시스템의 재사용 프레임워크 및 정형 명세 기법

이번 장에서는 이전 논문에서 연구된 임베디드 시스템의 재사용 프레임워크의 구조 및 각각의 구성 요소와 처리 메커니즘에 대해 설명한다[11, 12]. 설계된 재사용 프레임워크는 경량(Light Weight)의 임베디드 시스템 개발이 가능하며 개발 시점뿐만 아니라 운영 중에 동적으로 변경할 수 있도록 설계 한다.

3.1 임베디드 시스템의 재사용 프레임워크

임베디드 시스템의 재사용성을 향상시키기 위한 프레임워크는 (그림 1)과 같이 가변부 사용 영역이 재사용 프레임워크



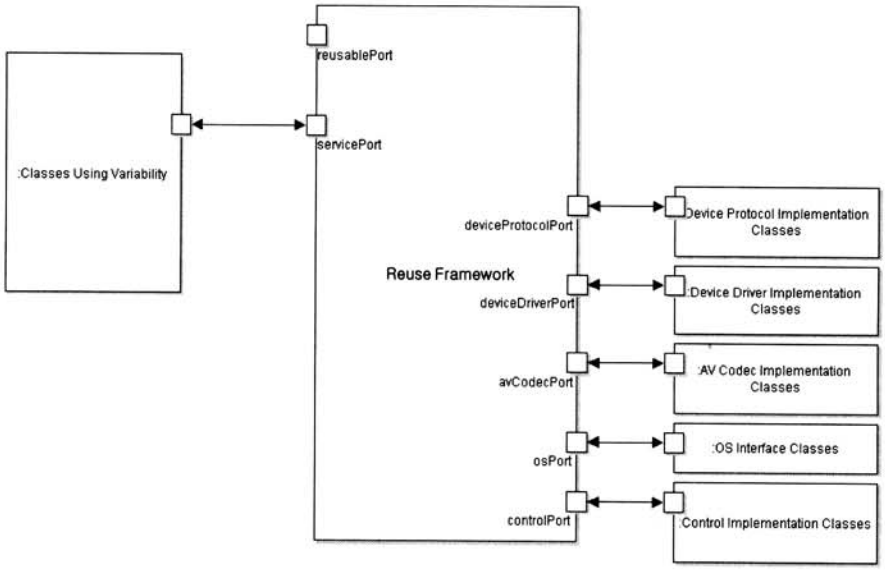
(그림 1) 재사용 프레임워크에 의한 가변부 처리 메커니즘

를 통해 가변부를 사용할 수 있다. 또한 재사용 프레임워크는 단일 인터페이스만을 통해 재사용성을 제공하는 것과 다르게 다양한 인터페이스를 제공할 수 있는 기반을 제공한다.

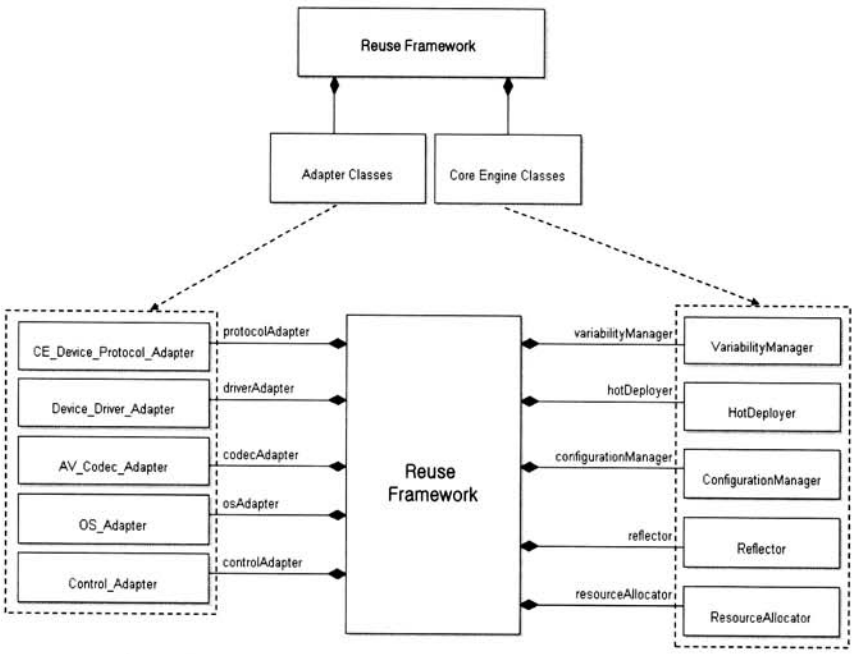
(그림 1)과 같이 재사용 프레임워크는 가변부 사용 영역과 가변부 사이의 중계자 역할을 한다. 가변부 사용 영역은 가변부를 직접 호출하지 않고 재사용 프레임워크를 통해 가변부의 서비스를 호출한다.

임베디드 시스템의 가변부 처리 메커니즘은 (그림 2)와 같이 가변부 사용 클래스들이 재사용 프레임워크를 통해 가변부를 접근한다. 홈 네트워크 시스템의 가변성[13]에 해당하는 디바이스 프로토콜, 디바이스 드라이버, AV 코덱, OS, 제어(시그널)를 가변적으로 접근할 수 있는 메커니즘을 제공한다.

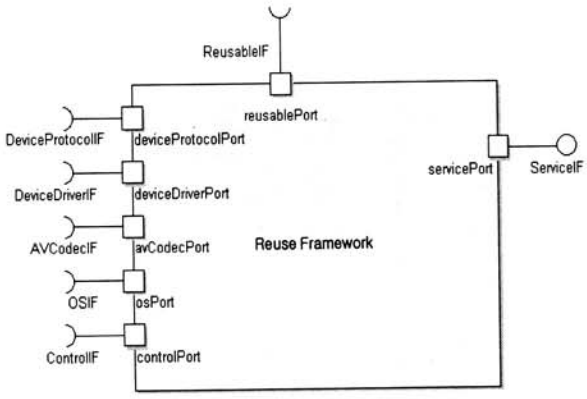
임베디드 시스템을 위한 재사용 프레임워크의 구성은 (그림 3)과 같다. 임베디드 시스템의 가변부들을 중계하기 어댑터와 가변부 처리를 지원하기 위한 핵심 클래스들로 구성된다. 재사용 프레임워크의 어댑터는 홈 네트워크 시스템의 가변성인 디바이스 프로토콜, 디바이스 드라이버, AV 코덱, OS, 제어 가변성에 대한 대행 역할을 수행한다. 가변부 처리를 위한 핵심 클래스는 가변성 관리기(Variability Manager),



(그림 2) 홈 네트워크 임베디드 시스템의 가변부 처리 메커니즘



(그림 3) 임베디드 시스템을 위한 재사용 프레임워크 구성 요소



(그림 4) 임베디드 시스템의 가변부 처리를 위한 인터페이스

리플렉터(Reflector), 설정 관리기(Configuration Manager), 동적 전개기(Hot Deployer), 그리고 자원 할당기(Resource Allocator)로 구성된다.

가변부의 변경을 제공하기 위해 재사용 프레임워크는 (그림 4)와 같은 인터페이스를 제공한다. 서비스 인터페이스('ServiceIF')는 재사용 프레임워크의 요구 인터페이스를 통해 설정된 서비스를 제공한다. 임베디드 시스템의 가변성에 대한 요구 인터페이스는 다양한 가변 기능으로 설정될 수 있으며 이러한 설정은 재사용 요구 인터페이스('ReusableIF')를 통해 설정된다[14-16].

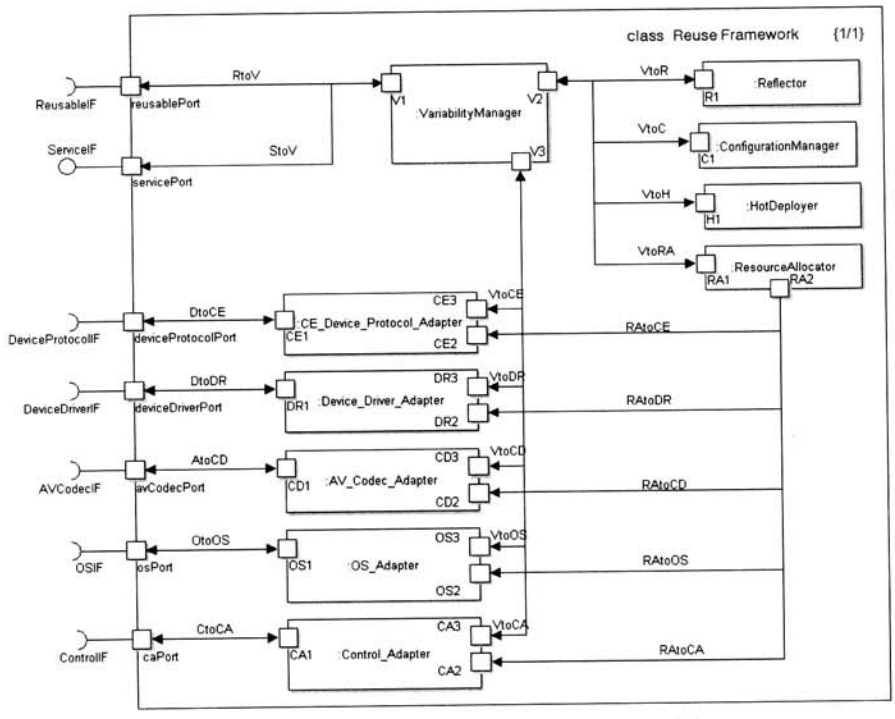
임베디드 시스템의 가변부에 대한 재사용 프레임워크는 어뎁터와 핵심 클래스를 통해 동적으로 홈 네트워크 서비스를 제공할 수 있다. 이러한 재사용 프레임워크의 어뎁터와 핵심 클래스들의 내부 구조는 (그림 5)와 같다. 임베디드 시스템의 가변성에 대한 어뎁터들은 재사용 요구 인터페이스('ReusableIF')를 통해 어떤 기능을 사용할지 설정하며, 재사용 프레임워크의 가변성 관리기는 설정 및 서비스 호출에 대한 대행 역할을 수행한다. 가변성 관리기에 의한 설정은 리플렉터, 설정 관리기, 그리고 동적 전개기를 통해 이루어지며, 설정된 서비스는 가변성 어뎁터들에 의해 제공된다. 임베디드 시스템의 자원에 대한 관리는 자원 관리기에 의해 설정되며 가변성 어뎁터에 따라 다르게 설정된다. 예를 들면 운영체제 가변 어뎁터가 설정하는 운영체제(WindowCE, VxWorks, 등) 에 따라 자원 관리기는 서로 다른 자원(메모리, 등)을 할당한다.

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 어뎁터들과 상호 작용을 통해 가변부의 특정 기능을 동적으로 호출한다.

가변부 사용 영역에서는 가변부를 사용하기 위해 (그림 6)과 같이 가변성 관리기를 통해 가변부를 호출한다. 가변부 호출 시 가변부에 대한 식별자와 입력 데이터를 전달하여



(그림 5) 임베디드 시스템을 위한 재사용 프레임워크 내부 구조

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 가변부 식별자는 가변부 사용 영역에서 정의하여 가변성 관리기를 통해 설정 관리기에 전달한다. (그림 6)과 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 디비아스로 변경해야 하는 요구사항을 만족시킬 수 있는 도구이다.

• 메타정보 저장소

메타정보 저장소는 가변부의 메타정보를 포함하고 있는 저장소로서 XML 기반의 가변부 정보를 관리한다.

(그림 7)에서와 같이 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 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. 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, 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 8)에서와 같이 리플렉터는 가변성 어댑터 식별자를 이용하여 물리적인 가변성 어댑터를 실행한다. 또한 가변성 어댑터는 리플렉터를 통해 가변성 클래스를 실행한다. 가변부를 변경하고자 할 때는 가변부 메타정보 만 변경하면 되며 (그림 8)의 가변부 사용 구조는 전혀 영향을 받지 않는다. 단지 전달되는 파라메터의 타입은 고려할 여지가 있다.

대부분의 임베디드 시스템의 가변성은 이러한 구조로 가변부를 제공할 수 있으며 디바이스 프로토콜이나 디바이스

```
Object result = VariabilityManager.execute( _VARIABILITY_NAME_ , _PARAMETER_ );
```

_VARIABILITY_NAME_ : 가변부 식별자

(그림 6) 가변부 사용 영역에서의 가변부 관리기 실행 코드

```
...

<Variability Name = "_VARIABILITY_NAME_">
    <Used-By> _ADAPTER_NAME_ </Used-By>
</Variability>


<Adapter Name ="_ADAPTER_NAME_">
    <Class> _CLASS_NAME_ </Class>
    <Behavior> _BEHAVIOR_NAME_ </Behavior>
    <Context> _CONTEXT_INFORMATION_ </Context>
</Adapter>
...
```

(그림 7) 가변부 메타정보

• 가변부 사용 클래스

```
Object result = VariabilityManager.execute( _VARIABILITY_NAME_ , _PARAMETER_ );
```

_VARIABILITY_NAME_ : 가변성 메타정보 식별자
PARAMETER : 전달 데이터 (예) 객체 배열

• 가변성 관리기 클래스

```
// 설정 관리기에 의해 가변부 메타정보 로딩
Object result = Reflector.delegate( _ADAPTER_NAME_ , PARAMETER_ );
Return result;
```

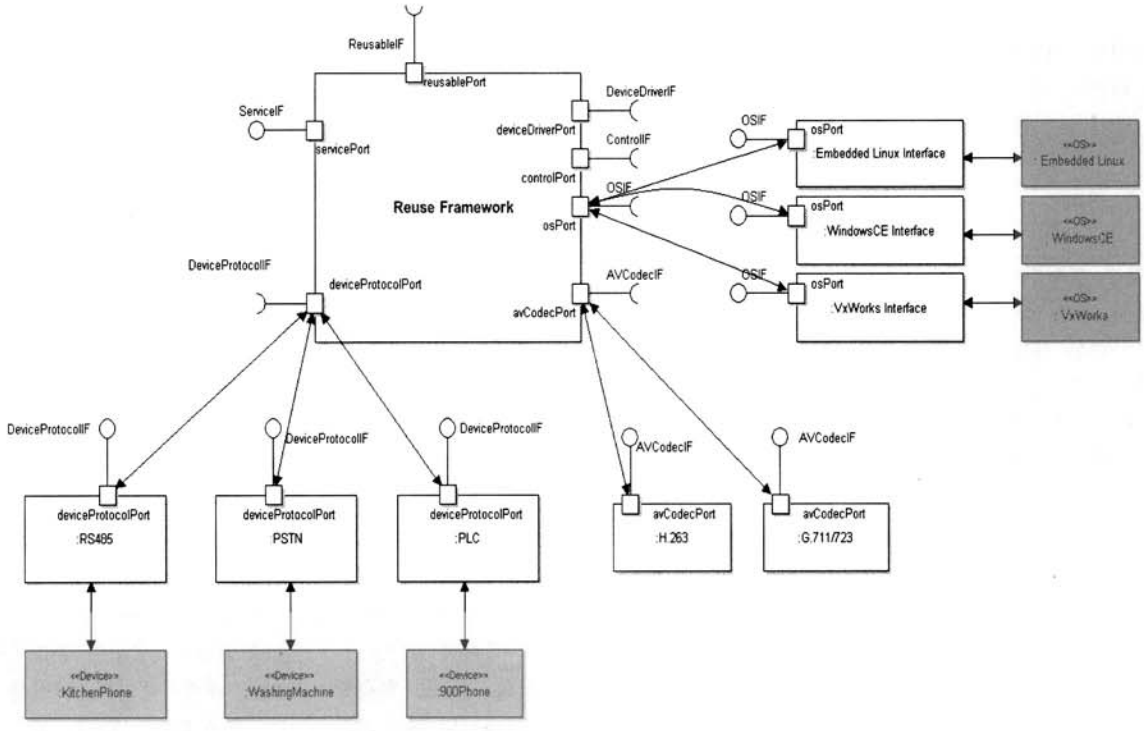
_ADAPTER_NAME_ : 가변성 어댑터 식별자

• 가변성 어댑터 클래스

```
// 가변부 클래스 및 행위 메타정보 로딩
Object result = Reflector.execute( _CLASS_NAME_ , _BEHAVIOR_NAME_ , _PARAMETER_ );
Return result;
```

_CLASS_NAME_ : 가변성 클래스 식별자
_BEHAVIOR_NAME_ : 가변성 클래스 내의 특정 오퍼레이션 식별자

(그림 8) 메타정보 기반의 가변부 선택 및 실행 코드



(그림 9) 가변성 어뎁터에 의한 가변부 구조

드라이버 클래스에 대한 변경도 이러한 구조를 통해 가능하다.

• 가변성 어뎁터

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

재사용 프레임워크의 가변성 어뎁터는 다양한 가변 처리를 지원할 수 있도록 요구 인터페이스(Required Interface)로 정의되며, 이러한 요구 인터페이스에 맞는 클래스를 통해 가변부를 제공한다. (그림 9)는 임베디드 시스템의 가변성인 디바이스 프로토콜, AV 코덱, 운영체제 가변성에 대한 가변부 설계 구조를 나타낸다. 임베디드 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.

• 동적 전개기

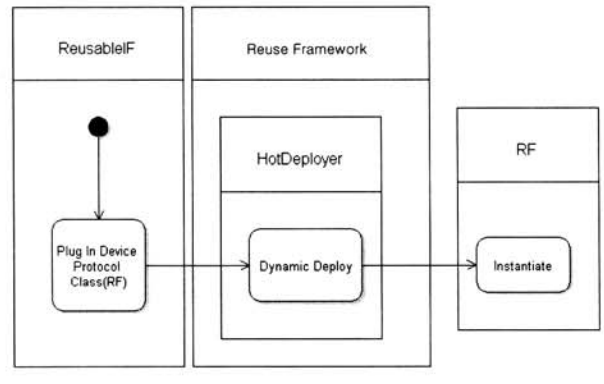
동적 전개기는 임베디드 시스템 내부에서 가변성을 제공할 수 없는 경우 시스템 외부에서 요구하는 기능 클래스를 시스템 내부로 플러그하기 위한 도구이다. 플러그의 개념은 시스템 패키지 내에 요구하는 클래스를 포함하는 것이 아니라 시스템 운영 환경에 맞게 객체가 생성(Instantiate)되는 것을 의미 한다.

(그림 10)과 같이 재사용 인터페이스 통해 디바이스 프로토콜 가변성을 시스템 외부에서 제공해야 하는 경우에는 재사용 프레임워크의 동적 전개기에서 요구하는 가변부 클래스('RF')의 객체를 초기화하여 현 운영되는 시스템에서 가변적으로 접근할 수 있도록 한다.

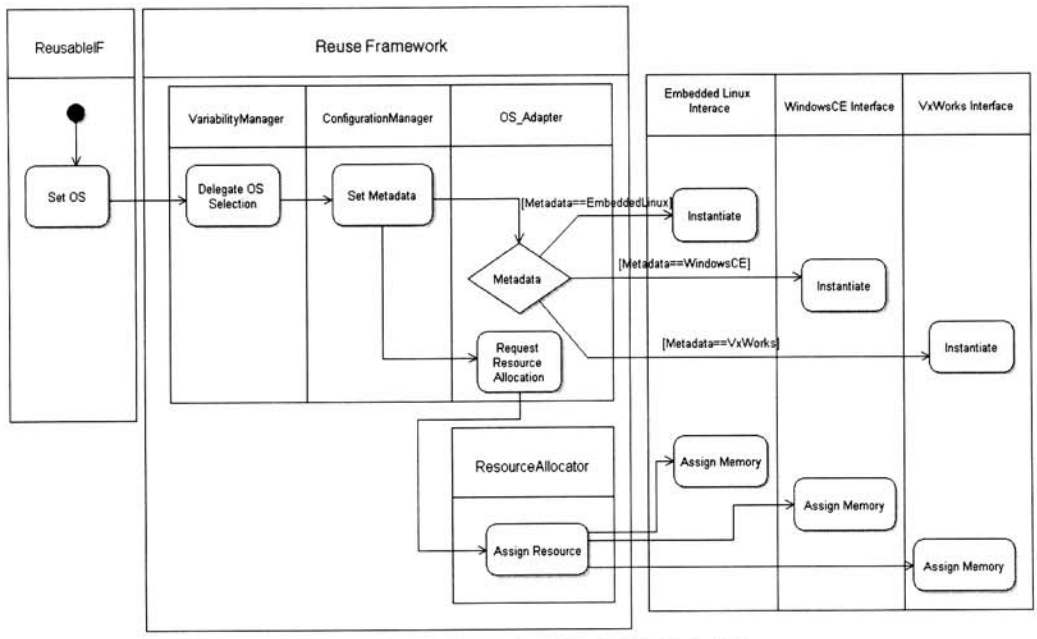
• 자원 할당기

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

(그림 11)의 경우는 임베디드 시스템의 운영 환경이 도메



(그림 10) 동적 전개기에 의한 가변부 클래스 플러그 인



(그림 11) 자원 할당기에 의한 가변성 처리 흐름

인의 요구사항에 따라 Embedded Linux 나 WindowsCE 또는 VxWorks로 변경해야 하는 경우 자원 할당기는 요구 운영 체제에 맞게 임베디드 시스템의 메모리를 설정한다. 이렇게 설정된 자원 정보는 홈 네트워크 시스템이 도메인에 설치(또는 배포)될 때 적용될 수 있다.

3.2 재사용 프레임워크의 정형 명세

비정형적인 클래스 모델(Class Model)로 정의된 임베디드 시스템의 재사용 프레임워크 메타모델은 이 구조물이 가진 구문(Syntax)과 의미(Semantic)를 명확하게 표현하고 제시된 모델의 정확성을 검증하기 위해서, 이 메타모델을 정형적으로 명세하고 이에 대한 모델의 검사가 필요하다[16].

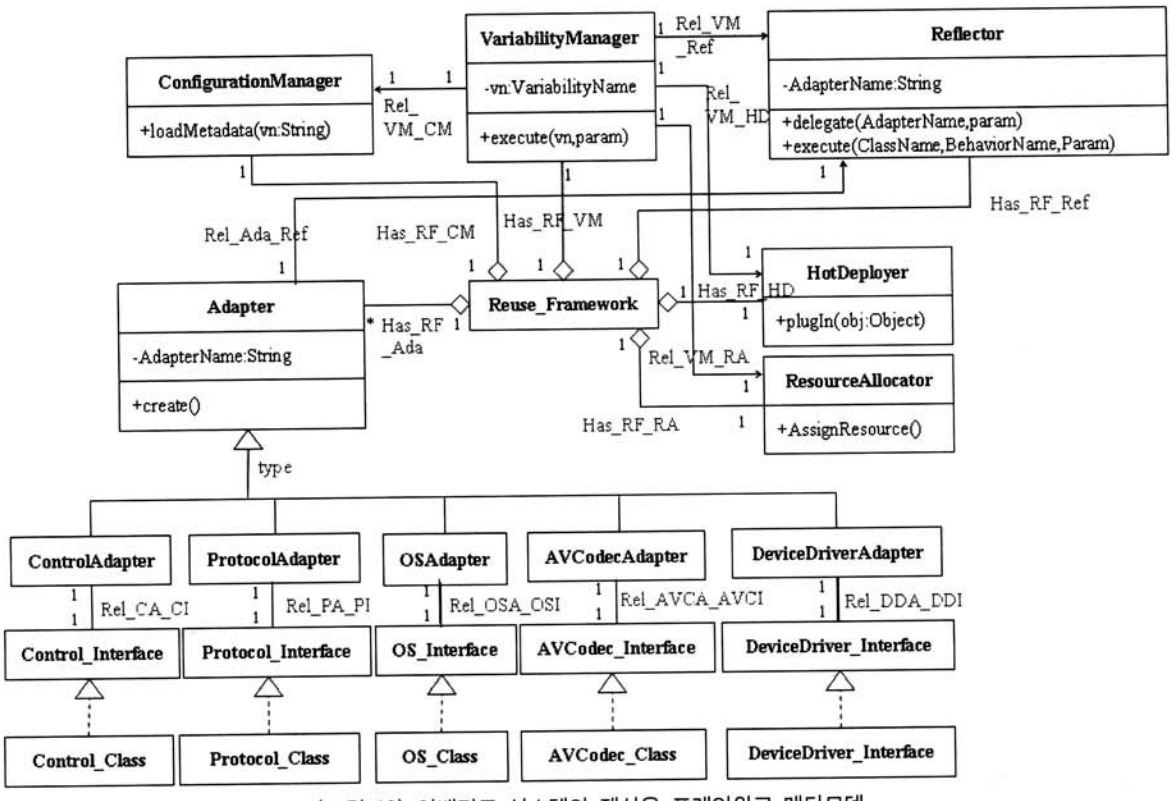
본 절에서는 제시하는 재사용 프레임워크 메타모델의 구문적 및 정적 의미를 명확히 표현하고 모델의 정확성을 입증하기 위해, 정형적 언어인 Z를 사용하여 정형적으로 명세하고 이를 Z-Eves[4] Tool을 통한 Z[3] 명세의 정확성을 검사한다. 먼저, 정형명세는 [3]에서 제시한 클래스 메타 모델과 Z간의 변환규칙을 적용하여, 재사용 프레임워크 메타모델의 구문과 정적 의미(Static Semantics)에 대해 Z 스키마로 명세한다. 즉, 재사용 프레임워크 메타모델의 구문에 대해 Z로 기본 타입을 선언하고 각 클래스를 대상으로 요소 기반으로 Z 스키마를 정의하고, 또한 클래스간 상호 관련성을 대해 관계 기반으로 Z 스키마를 정의한다. 정적 의미는 Z 스키마내에 술어 부문 선언에 술어논리(Predicate Logic)로 명세한다. 이어서, Z에 대한 모델의 작성/편집/체킹을 제공하는 Z/Eves 도구를 사용해서 정의된 Z 명세를 입력하고, 구문 및 타입 체킹(Type Checking) 그리고 도메인 체킹(Domain Checking)을 수행하여 모델의 오류를 검사한다. 이로서, 프레임워크 메타모델 구조물의 아키텍쳐 의미를 명확히 나타낼 수 있고, 모델의 정확성을 보인다.

3.2.1 임베디드 시스템의 재사용 프레임워크 메타모델

임베디드 시스템의 재사용 프레임워크 설계인 (그림 9)의 가변성 어뎁터에 의한 가변부 구조에 기반하여 클래스 모델로 표현된 메타모델로 정의한 것이 이다. (그림 12)에서 재사용 프레임워크는 ConfigurationManager, VariabilityManager, Reflector, HotDeployer, ResourceAllocator 그리고 adapter로 구성한다. Adapter의 종류로서 ControlAdapter, Protocol-Adapter, OSAdapter, AVCodecAdapter 및 DeviceDriverAdapter가 있다.

3.2.2 재사용 프레임워크 메타모델의 Z 정형명세

재사용 프레임워크 메타모델인 그림 12에 대한 정형적 Z 명세를 위하여, [3]에 제시된 클래스 모델과 Z간의 변환규칙을 적용하여, Z로의 정형적 명세를 기술한다. [3]에서 변환규칙은 클래스 다이어그램이 갖고 있는 요소(Element), 관계(Relationship) 다중성(Multiplicity) 및 제약사항(Constraint)을 고려하여 Z 언어로의 변환을 정하고 있다. 그래서, Z 명세의 방법은 메타모델내 포함된 요소들에 대해 기본 타입(Basic Type)과 자유 타입(Free Type)들을 선언한다. 다음으로 변환 규칙에 의해, 메타모델내 구성 요소(클래스)별로 스키마를 정의하고, 또한 관계상에 존재하는 각 관계명(Relationship Name)별로 스키마를 정의한다. 이때, 정적 의미들을 위한 Invariant한 제약사항(Constraint)들은 요소를 정의한 스키마의 술어부에 명세한다. Z 명세의 구축 결과로서, 기본타입으로 Control_Interface외 16개 타입 선언 및 자



(그림 12) 임베디드 시스템의 재사용 프레임워크 메타모델

유타입으로 Adapter Type에 대해 5개를 선언하고, 요소 기반 스키마 명세로 Adapter외 16개를 명세하고, 그리고 관계 기반 스키마 명세로 Rel_VM_CM외 13개를 작성하였다. 본 단락에서는 구축된 Z 명세를 기술함에 있어 일부를 발췌하여 수록한다. 단, 재사용 프레임워크 메타모델상의 실체화 관계에 대해서, 그 실체화 관계(예, Control_Interface와 Control_Class간 등) 그리고 요소(예, Control_Class 등)에 대해서는 Z 명세를 생략하였다. 그 이유는 Z로의 변환 규칙과 Z/Eves에서 지원하지 않기 때문이다.

이러한 정형적 명세를 통해서, 그 모델의 의미를 명확하게 함으로서, 구축 모델에 대한 속성들(구문적 일관성 등)의 분석 및 검증이 가능해진다.

• 기본타입 선언

기본 타입들의 선언은 기본 타입과 자유 타입으로 기술한다. 기본 타입의 선언은 재사용 프레임워크 메타모델에 포함된 모든 요소들에 대해서 각 요소별로 기본 타입을 명세한다. 기본 타입의 선언은 다음과 같다.

−기본 타입

[Class1_f1_ID, Class1_Adp_ID, Class2_Adp_ID, ClassN_Adp_ID, Reflector_ID, Config_Manager_ID, Hot_Deploy_ID, Var_Manager_ID, Var_Class_ID]

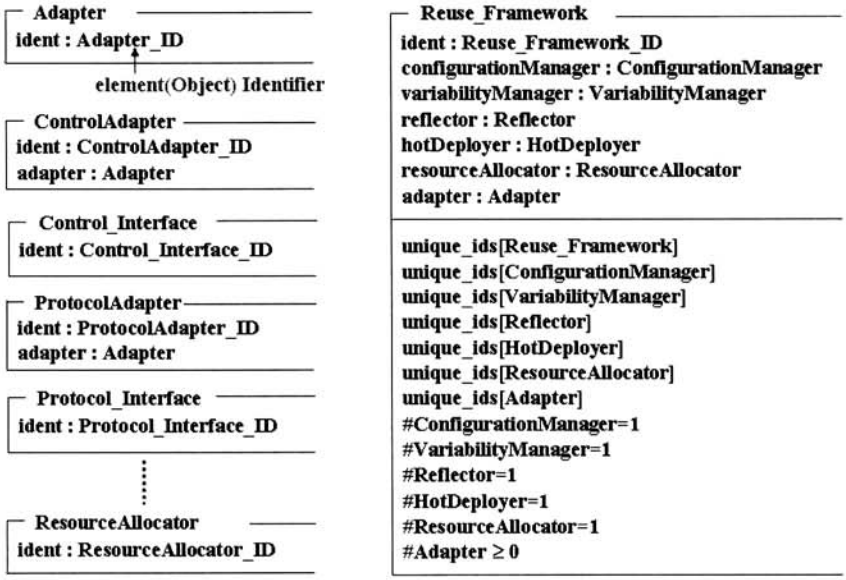
−자유 타입

Class1_f1_type ::= Class1a_f1 | Class1b_f1 | Class1c_f1

• 스키마 명세

스키마 명세는 재사용 프레임워크 메타모델내에 포함된 모든 요소와 관계들에 대해서 Z 스키마로 명세를 한다.

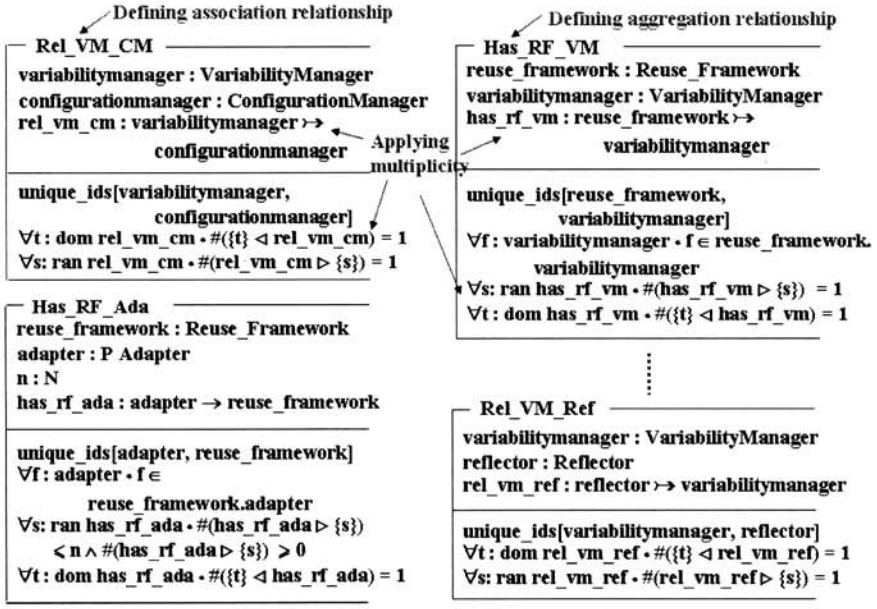
먼저, 요소 기반 Z 스키마 명세는 메타모델내 클래스별로 각각 스키마를 생성한다. 이의 일부를 (그림 13)에서 보여준다. 메타모델내 많은 요소(클래스)들간의 스키마의 선언 순서는 메타모델의 클래스 모델 구조상 하부 요소에서 상위 요소를 대상으로 스키마를 선언해 나간다. 예를 들어, (그림 13)에서 'Adapter' 요소의 스키마 정의는 'ControlAdapter' 스키마 정의 이전에 명세되어야 한다. 왜냐하면, 재사용 프레임워크 메타모델 상에서, 'Adapter'와 'ControlAdapter" 요소간에는 상속관계로서, 자식 클래스가 부모 클래스로부터 상속을 받아야 하기 때문에, 먼저 부모 클래스가 스키마로 선언되어야 하기 때문이다. 각 스키마는 스키마 명, 변수들로 구성되는 선언부 그리고 제약조건의 Semantics를 표현하는 술어부로 명세한다. 즉, 요소들에 대한 선언과 그들 간의 관계는 시그네쳐부(Signature Part, 변수 선언부)에 명세하고, 요소들간의 불변적 제약사항들은 술어부(predicate part)에 정의한다. 가령, (그림 13)에서, 'Reuse_Framework' 스키마의 술어부에 기술한 "#Adapter ≥ 0"은 불변적 제약사항의 속성을 표현한다. 이것은 재사용 프레임워크 구성을 위해 어뎁터 객체 수가 0개 이상이어야 함을 의미한다. 아울러, 요소들간의 관계인 연관(association) 및 집합(aggregation)



(그림 13) 재사용 프레임워크 메타모델의 요소 기반 Z 스키마 명세

에 대해 요소 기반 스키마에서 부분적으로 명세된다.

다음으로 관계 기반 스키마 명세는 두 요소간의 관계별로 각기 스키마로 정의한다. 이 스키마는 클래스간 관계들과 다중성(multiplicity)을 명확하게 구분하여 정의를 해준다. 재사용 프레임워크 메타모델의 관계 기반 스키마 명세의 일부를 나타낸 것이 (그림 14)이다. 재사용 프레임워크 메타모델에서 관계의 명칭 부여는 연관은 Rel_, 집합은 Has_로 약칭하였다. (그림 14)에서 각 관계의 유형별, 즉, 연관 및 집합에 대해 그 명세를 보여주고 있다. 먼저, 연관관계에 대해, 연관 관계를 갖는 'Rel_VM_CM' 관계에서 이 관계에 관련되는 요소는 VariabilityManager와 ConfigurationManager이다. 연관 관계는 이 스키마의 시그네쳐부에 두 요소를 변수로서 선언하면 된다. 이 두 요소는 요소 기반 스키마 명세에서 개별적으로 이전에 스키마로 정의한다. 이 요소에 대한 변수 선언시, 다중성에 의해 단일 set(객체가 1개) 혹은 Power set(객체가 2개 이상)으로 표현된다. 이어서, 집합 관계에 대해서, 가령 'Has_RF_VM' 집합 관계에서, 이 관계에 관련되는 요소는 Reuse_Framework와 VariabilityManager이다. 이 두 요소의 Reuse_Framework와 VariabilityManager를 시그네쳐부에 기술하고, 아울러 다중성에 기반하여 시그네쳐부와 술어부에 정의한다. 특히, 술어부에 집합 관계를 "∀f: variabilitymanager • f ∈ reuse_framework.variability



(그림 14) 재사용 프레임워크 메타모델의 관계 기반 Z 스키마 명세

manager"와 같이 표현한다. 그리고 관계 유형에 대해 dom (정의역)과 ran(치역)에 의해 기 제약사항을 표현한다. 한편, 본 논문에서 생략된 메타모델내 요소와 이들간의 관계들에 대한 불변적 제약사항인 Well-formedness property(적형성 속성)에 대해서 스키마의 술어부에 기술할 수 있다.

4. 실험 및 평가

본 논문에서 제시하는 프레임워크 메타모델의 정확성 및 의미적 타당성을 검증하기 위해 명세한 Z 명세에 대해 정확성 검사가 필요하다. 그 이유는 재사용 프레임워크 메타모델은 Z 스키마로 변환 명세되었고, 명세의 구문적 및 정적 시멘틱 속성이 정확하다고 검사된다면 결국, 재사용 프레임워크 메타모델이 명확하다는 것을 의미하기 때문이다. 이를 위해, Z/EVES Tool을 사용하여 Z 명세에 구문적 구조 속성 등의 정확성 검사가 요구된다. 즉, Z 명세의 syntax checking, variable range checking 및 consistency type checking을 위하여 Z/Eves Tool을 통하여 모델 체킹한다.

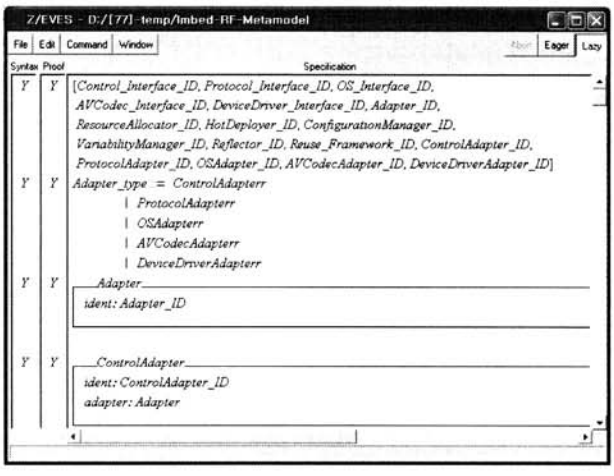
일반적으로 Z로 명세된 것에 대한 검증의 툴로서 Z/Eves가 널리 사용되고 있다. 사용된 Z/Eves Tool은 V2.0을 이용한다. 이 툴을 통한 Z 명세의 분석 기능은 syntax와 type checking, schema expansion, precondition calculation, domain checking, specification animation 그리고 general theorem proving을 할 수 있다. 본고에서는 이 툴에 Z 명세한 것을 입력하여 실행하여 syntax 및 type checking, 그리고 domain checking을 검사한다. 검사의 결과로서 (그림 15)는 타입 선언에 대한 명세에 대한 검사 결과를 보여준다. (그림 15)에서 상단의 syntax와 proof 메뉴를 보면, 그 이하 부문에 각각 "Y"로 실행 결과가 나온 것은 Z의 기본 타입과 자유 타입 선언에 syntax와 proof 속성(property)에 대해 오류가 없음을 의미한다.

다음으로 (그림 16)은 요소 기반 Z 스키마 명세에 대한 부분에 대한 모델 체킹의 검사 결과를 보여주고 있다. (그림 16)에서 왼쪽 상단에 표시된 'syntax'는 syntax와 type checking을 보여주는 것으로 'Y'로 나타나면 타입간 일치성이 있음을 나타낸다. 반면, 'proof'는 domain checking의 결과를 보여주는 것으로 'Y'로 나타나면, 그 명세가 정확함을 의미한다. 이 검사를 위하여, 이전에 Z 명세한 것에 수정 혹은 추가적 속성들의 명세가 요구되기도 하였다. 왜냐하면 Z 언어의 표현력과 이 표현력에 대해 Z/Eves에서 완전히 동일하지 않기 때문에, 스키마의 술어부에 당초에 기술된 일부 술어 명제에 대해서는 정보의 손실없이 달리 변환, 표현하여 검사를 하였다.

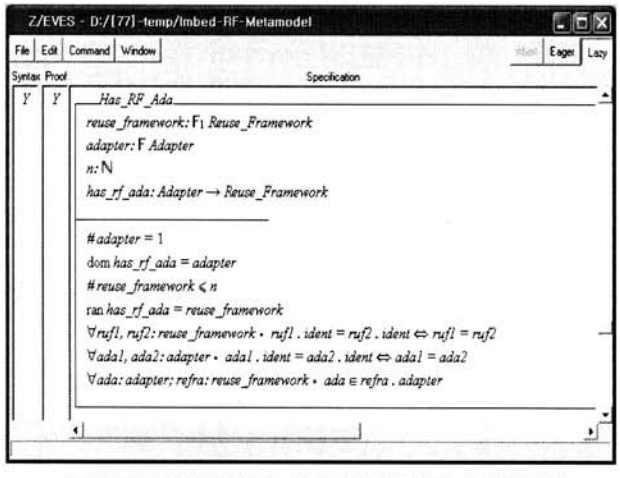
끝으로, (그림 17)은 관계 기반의 Z 스키마 명세에 대한 모델 체킹의 결과를 나타낸다. syntax와 proof 부문이 "Y"의 결과를 보임으로서 명세가 정확함을 알 수 있다.

결국 이와 같은 모델 체킹을 통해, 재사용 프레임워크 메타모델의 구문적 및 정적 시멘틱 측면에서 오류가 없음을 입증할 수 있다.

(그림 16) 요소 기반 Z 스키마 명세의 구문검사 결과



(그림 15) 기본타입 Z 명세의 구문검사 결과



(그림 17) 관계 기반 Z 스키마 명세의 구문검사 결과

5. 결 론

임베디드 시스템은 다양한 도메인의 요구사항에 따라 다양한 디바이스를 지원할 수 있어야 한다. 본 논문에서는 임베디드 시스템의 이러한 가변적인 처리를 지원하기 위해 홈 네트워크 임베디드 소프트웨어 내에 재사용 프레임워크를 제안하여 적용하였다. 홈 네트워크 시스템의 디바이스 변경에 따라 제어, 디바이스 프로토콜, 디바이스 드라이버, 운영체제, AV 코덱 등이 변경되어야 하며 재사용 프레임워크에서는 이러한 가변적인 부분을 인터페이스나 동적으로 처리될 수 있는 메커니즘을 제공한다. 홈 네트워크 시스템의 가변적인 처리 기법은 홈 네트워크 시스템 내의 여러 기능 중에 하나를 선택하는 선택 기법과 홈 네트워크 시스템 외부에서 새로운 기능을 제공하는 플러그 인 기법이 있다. 두 기법은 메타정보를 통해 설정할 수 있으며 개발 시점뿐만 아니라 운영 시점에도 다양한 요구사항을 지원할 수 있다. 아울러, 제시하는 임베디드 시스템의 재사용 프레임워크 메타모델에 대한 Z 언어를 이용한 정형적 명세와 Z/Eevs를 사용한 모델 체킹을 수행하여 이 메타모델이 정확함을 입증하였다. 본 논문에서 제안하는 홈 네트워크 임베디드 시스템 재사용 기법에 대한 사례연구를 통해 재사용성이 향상됨을 증명하였으며, 향후 연구에서는 임베디드 시스템의 재사용성을 향상시킬 수 있는 공통의 재사용 프레임워크 기법을 연구할 것이다.

참 고 문 헌

[1] 홍성수, "RSCA: 분산 로봇 플랫폼에서 임베디드 소프트웨어의 동적 재구성을 지원하는 통합 미들웨어," 한국통신학회지, 21권 10호, 2004.

[2] K. Raj, Embedded Systems: Architecture, Programming and Design, McGraw Hill, 2004.

[3] M. Shroff, R. France, "*Towards Formalization of UML Class Structures in Z*", in Proc. of the COMPSAC '97, Washington DC, pp.11-15, Aug. 1997.

[4] M. Saaltink, The Z/EVES 2.0 Users Guide, TR-99-5493-06A, ORA Canada, 1999.

[5] H. Comma, "A Software Design Method for Real-Time Systems", Communications of ACM, Vol.27, No.7, pp.938-949, Sept. 1984.

[6] J. Ready and D. Howard, "Structuring Real-Time Application Software Part1", VMEbus Systems, pp.33-45, April, 991.

[7] Kang K., "Issues in Component-Based Software Engineering", International Workshop on Component-Based Software Engineering 1999.

[8] Atkinson C., Bayer J., Bunse C., Kamstices E., Laitenberger O., Laqua R., Muthig D., Paech B., Wust J., and Zettel J., Component-based Product Line Engineering with UML, Addison-Wesley, 2001.

[9] 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), Novermber 2000.

[10] Muthig D. and Patzke T., "Generic Implementation of Product Line Components", NODe 2002, LNCS 2591, pp.313-329, 2003, Fraunhofer Institute for Experimental Software Engineering (IESE), Novermber 2000.

[11] 조은숙, 김철진, "임베디드 시스템의 재사용 프레임워크를 위한 정적 메타모델 설계", 한국멀티미디어학회 논문지, pp.231-243, 제12권 제2호, 2009년 2월

[12] 김철진, 조은숙, "재사용성 향상을 위한 임베디드 소프트웨어의 동적 가변성 설계 기법", 한국정보과학회 논문지, pp.30-44, 제36권 제1호, 2009년 1월.

[13] Coplien J., Hoffman D., and Weiss D., "Commonality and Variability in Software Engineering", IEEE Software, pp. 37-45, November 1998.

[14] Gamma E., et al., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[15] D'souza D. F. and Wills A. C., Objects, Components, and Frameworks with UML, Addison-Wesley, 1999.

[16] 송치양, "A Metamodel-Based Modeling Mechanism for Hierarchical Design in UML", Thesis for the Degree of Doctor, 2003. 7.

조 은 숙

e-mail : escho@seoil.ac.kr
1993년 동의대학교 전산통계학과(이학사)
1996년 숭실대학교 컴퓨터학과(공학석사)
2000년 숭실대학교 컴퓨터학과(공학박사)
2000년~2005년 동덕여자대학교 정보학부 강의전임교수
2005년~현 재 서일대학 컴퓨터 소프트웨어과 조교수
관심분야 : CBSE, Embedded Software, Web Service, SOA, Service-Oriented Computing

김 철 진

e-mail : cjkim777@gmail.com
1996년 경기대학교 전자계산학과(이학사)
1998년 숭실대학교 컴퓨터공학부(공학석사)
2004년 숭실대학교 컴퓨터공학부(공학박사)
2004년 가톨릭대학교 컴퓨터정보공학부 강의전담교수
2004년~2009년 삼성전자 책임연구원
2009년~현 재 인하공업전문대학 컴퓨터시스템과 조교수
관심분야 : CBD, Component Customization, Embedded Software

송 치 양

e-mail : cysong@knu.ac.kr

1985년 한남대학교 전산학과(학사)

1987년 중앙대학교 전산학과(이학석사)

2003년 고려대학교 컴퓨터학과(이학박사)

1990년~2005년 한국통신 중앙연구소 책임연구원

2005년~2007년 상주대학교 소프트웨어공학과 조교수

2008년~현 재 경북대학교 소프트웨어공학과 조교수

관심분야 : CBD, MDA, Meta-Model, Formal Specification, IP-TV Service