UML 2.0 프로파일링을 이용한 FORM 아키텍처 모델링

(Modeling FORM Architectures Based on UML 2.0 Profiling)

양 경 모 † 조 윤 호 †† 강 교 철 †††
(Kyungmo Yang) (YoonHo Jo) (Kyo Chul Kang)

요 약 소프트웨어 제품 생산 라인(Software Product Line) 공학은 새로운 소프트웨어 개발 패러다임으로 각광받고 있다. SPL에 FORM(Feature-Oriented Reuse Method) 방법론을 적용하면, 휴대전화나 디지털TV 같이 공통점이 많은 제품군의 다양한 소프트웨어를 휘처 모델링을 통해 만들어진 재사용 가능하고 유연한 컴포넌트를 조합하여 생산해 낼 수 있다. 한편, MDA(Model Driven Architecture) 방법론은 PIM(Platform Independent Model) 을 통해 다양한 개별 플랫폼을 위한 소프트웨어를 생산할 수 있게 하는 새로운 기술을 제공한다. 위 두 가지 방법론의 장점을 조합하면 공통점을 공유하면서 다양한 플랫폼에서 동작하는 제품군의 소프트웨어를 생산하는데 도움이 된다.

이 논문에서는 FORM 방법론과 MDA 방법론을 조합하기 위해 먼저, 프로파일링 기법을 통해 UML2.0을 확장하여 FORM 아키텍처와 Parameterized Statechart 모델링이 가능하게 한다. 다음으로, 휘처가 휘처 모델과 Parameterized Statechart사이에서 일관성 있게 element의 형태로 위치하고 있는지 검증하는 일관성 규칙을 제공한다. 몇 가지 규칙은 FORM 아키텍처와 Parameterized Statechart 사이의 일관성을 검사하기 위해 고안되었다.

마지막으로, 엘리베이터 시스템의 사례연구를 통해 이 논문에서 제안하는 모델링 기법과 일관성 검사 법칙의 유효성을 제시한다.

키워드 : 휘처 지향 재사용 방법론, 소프트웨어공학, 제품 생산라인 공학, 모델 지향 아키텍처

Abstract The Software Product Line (SPL) engineering is one of the most promising software development paradigms. With Feature-Oriented Reuse Method (FORM), reusable and flexible components can be built to aid the delivery of various software products such as mobile phone and digital TV applications based on commonalities and variabilities identified during Feature modeling. Model Driven Architecture (MDA) is also an emerging technology which supports developing software products to work on different platforms with platform independent models (PIM). Combining advantages of these two approaches is helpful to build a group of software products which share common Features while working on various platforms.

As first step to combine FORM with MDA, we extend UML2.0 with profiles by which FORM architectures and parameterized Statecharts can be modeled. Secondly, we provide rules to examine whether Features are allocated at positions of elements of Statecharts consistently between a Feature model and a parameterized Statechart. Some rules are designed to check the consistency between FORM architectures and parameterized Statecharts.

A case study on an elevator control system is provided to demonstrate the feasibility of our modeling approach and consistency checking rules.

Key words : FORM(Feature-Oriented Reuse Method), UML, MDA, Parameterized Statechart, SPLE(Software Product Line Engineering), Software Engineering

† 정 회 원 : 삼성전자 DM개발사업부
kyungmo01.yang@samsung.com
†† 학생회원 : 포항공과대학교 정보통신대학원
lucio.red@postech.ac.kr
††† 종신회원 : 포항공과대학교 컴퓨터공학과 교수
kck@postech.ac.kr
논문접수 : 2008년 4월 14일
심사완료 : 2009년 4월 20일

정보과학회논문지 : 소프트웨어 및 응용 제36권 제6호(2009.6)

1. 서 론

PLM 방법론과 MDA 방법론을 비교해보면 두 방법론이 서로를 보완해주는 측면이 있다는 것을 알게 된다. 우선 MDA[1-3] 방법론은 목적(Target) 플랫폼에 독립적인 PIM을 모델링함으로서 실행코드를 자동화된 도구를 통해 얻어내는 특징이 있다. 즉, 모델링 언어를 프로그래밍 언어처럼 사용할 수 있게 한다. 하지만 도메인의 가변성을 PIM에 표현할 수 없으므로, 도메인에서 요구되는 비슷한 기능을 하는 다양한 제품을 효율적으로 생산하기에는 적합하지 않다. FORM 방법론은 FODA[4]에서 제안한 휘처 모델로서 특정 도메인의 가변성을 식별하고, 그것을 반영한 재사용 가능한 컴포넌트를 만듦으로써, 재사용 컴포넌트의 조합으로 개별 제품을 빠르게 생성해 낼 수 있는 특징이 있다. 또한 플랫폼에 독립적인 모델로서 자산 개발 시 이를 바탕으로 유연한 컴포넌트를 만든다. 하지만 컴포넌트를 만들 때, 코드를 개발자가 작성해야 하고, 가변적인 부분에는 매크로를 사용해야 하는 등의 단점이 있다.

FORM방법론의 네 가지 관점의 아키텍처를 바탕으로 변환규칙을 통하여 PIM을 만들고, 이를 바탕으로 MDA 방법론을 이용하여 실행 코드를 생성할 수 있다면, 고생산성과 고품질의 제품 생산을 기대할 수 있을 것이다. 따라서 본 연구에서는 FORM 아키텍처를 PIM으로 변환할 수 있게 하는 선행 연구를 수행하려고 한다.

2. 관련연구

2.1 UML로 소프트웨어 아키텍처를 표현하기 위한 연구

표 1에서 볼 수 있듯이 UML을 확장한 아키텍처들은 행위모델을 명세할 수 있는 메커니즘을 갖고 있다. 하지만 행위모델과 아키텍처모델과의 일관성을 위한 연구는 찾아보기 어려웠고, 가변성을 표시하기 위한 연구 또한 없었다.

표 1 UML을 이용하여 아키텍처를 표현하려는 연구 비교

연구 논문	제안한 아키텍처에 행위 모델을 명세할 수 있게 하는 연구의 여부	행위 모델과의 일관성을 위한 규칙 제시	가변성을 명세 여부
[5]	◐	Ø	Ø
[6]	◐	Ø	Ø
[7]	● Hybrid Statechart profile을 제시하여 행위 모델 명세	Ø	Ø
[8]	◐	Ø	Ø
[9]	◐	Ø	Ø
[10]	◐	Ø	Ø
[11]	● ArchProtocol, ArchBehavior으로 stereotyping하여 행위 모델 명세	Ø	Ø

(지원 : ● 어느 정도 지원 : ◐ 지원하지 않음 : Ø)

본 논문에서 제안하려고 하는 아키텍처는 휘처모델에서 찾아낸 가변성을 명세할 수 있고, 행위명세 또한 가변성을 명세할 수 있어야 하며, 아키텍처와 행위명세에 일관성을 검사할 수 있도록 한다.

2.2 Statechart에 관한 연구

[12-15]를 살펴보면 Statechart에 가변성을 명세 하려는 노력은 거의 찾아볼 수가 없다. 있더라도 가변적인 휘처를 할당했을 뿐 휘처 모델이 갖는 의미를 반영하려는 연구는 없었다. 본 연구에서는 UML State machine을 확장하여 가변성을 명세할 수 있도록 하고, 가변성이 명세된 State machine이 휘처 모델의 의미를 제대로 반영하고 있는지 검사할 수 있는 규칙을 제시하며, 해당 모델이 도달가능 한지를 알아볼 수 있는 메커니즘을 제안하고자 한다.

3. Parameterized Statechart에 관한 연구

3.1 Parameterized Statechart[1]와 휘처모델간의 일관성규칙

이 절에서는 Parameterized Statechart가 휘처 모델의 의미에 맞도록 가변성이 명세 되었는지의 여부를 판별할 수 있는 일관성 규칙을 제안한다. 일관성 규칙을 소개하기 앞서, 규칙에 쓰일 유용한 함수들을 먼저 소개하도록 한다.

표 2 일관성 규칙을 위한 유용한 함수들

함수	설명
Derived_require (fm,x)	휘처 x가 선택되면 같이 선택되어야 하는 가변적인 휘처들의 집합
Derived_mutex (fm,x)	휘처 x가 선택되면 배제되어야 하는 가변적인 휘처들의 집합
RVPS (statechart,x)	선택된 휘처 x에 의해서 variability가 제거된 state들의 집합
Feature(x)	state x에 할당되어 있는 휘처 들의 집합
Type(f1,f2)	휘처 f1,f2의 관계유형을 리턴 ('consist-of' or 'gen-spec')

• Rule 1. Parameterized Statechart와 휘처 모델의 일관성 규칙

① For all fa,fb ∈Feature and sa,sb ∈ state,
if fa ∈ Feature(sa) and fb ∈ Feature(sb) and Type(fa,fb)='consist-of', then sb ∈ sa.substate

② For all fa,fb ∈ Feature and sa,sb ∈ state,
if fa ∈ Feature(sa) and fb ∈ Feature(sb) and

1) 휘처모델에 가변성을 표현하기 위해서 Statechart에 parameterization을 적용한 것을 의미한다.

Type(fa,fb)='gen-spec' then sa.transition ⊂ sb.transition and sa.state ⊂ sb.state

③ For all fa, x ∈ Feature and sa ∈ state,
if fa ∈ derived_require(fm,x) and fa ∈ Feature(sa)
then sa ∈ RVPS(s,x)

④ For all fa,x ∈ Feature and sa ∈ state,
if fa ∈ derived_mutex(fm,x) and fa ∈ Feature (sa)
then {sa} ∩ RVPS(s,x) = Ø

Rule 1의 1번, 2번 규칙은 휘처 모델의 Gen-Spec[2] 관계와 Consist-of[3] 관계가 Parameterized Statechart에서도 계속 유지되는지를 검사하는 규칙이다. 3번 규칙은 특정 제품에 포함되어야 하는 휘처 집합이 같은 제품에 포함되어야 하는 Parameterized Statechart의 요소들과 같은 지를 알아보는 규칙이다. 4번 규칙은 특정 제품에 포함되지 않아야 하는 휘처 집합이 같은 제품에 포함되지 않아야 하는 Parameterized Statechart의 요소들과 같은지를 알아보는 규칙이다.

3.2 Parameterized Statechart의 도달성 검사를 위한 연구

Parameterized Statechart의 도달성이란 휘처 모델에 존재하는 모든 휘처 구성에 따라서 Parameterized Statechart의 가변성을 제거해 보았을 때, 각각의 Statechart가 위에서 제안한 도달성 규칙을 만족할 경우를 뜻한다. 본 논문에서는 도달성 검사를 하기 위해서 도달성 트리를 만들고, 트리를 분석하여 정의한 도달성을 만족하는지를 검사한다.

그림 1에서는 도달성 트리를 만들기 위한 알고리즘이 제시되었다. 가변성이 제거된 Statechart를 인자로 받아서 도달성 트리를 형성한다. 해당 Statechart에는 발생되어야 하는 내부 이벤트가 있을 경우, 해당 이벤트가 따로 명세 되어 있다고 가정한다. RT(도달성 트리)는 상태 구성과 그 관계를 저장할 수 있는 자료 구조이고, ST, IE, SIE, OIE, QUEUE는 원소를 저장할 수 있는 집합형태의 자료구조이다. 14행부터 26행에서는 검사하고자 하는 전이의 조건이 만족되었을 경우, 전이에 명세된 이벤트가 내부 이벤트가 아니거나, 내부 이벤트일 경우 바로 앞 스텝에서 발생되었는지 여부를 확인하여 해당 전이가 활성화 되었다고 가정한다. 활성화 될 때 발생되는 내부이벤트를 SIE에서는 제거하고, OIE에는 할당한 후, 다음 스텝의 상태 구성 노드(SC')를 RT에 추가한다. 내부 이벤트가 발생되지 않아서 전이가 활성화 되지 않는 경우는 검사할 필요가 없다. SIE에 해당 내부 이벤트가 제거되지 않을 것이므로 검사를 마친 후, SIE를 확인함으로써 내부 이벤트가 발생하였는지, ST를 확인함으로써 내부 이벤트를 명세한 전이가 활성화 되었는지 여부를 알 수 있기 때문이다(<GUARD-EXP>는 해당 전이에 명세된 이벤트+조건 표현식을 의미한다).

그림 2는 도달성 검사를 위한 예제 Statechart이다. 주어진 Statechart에는 내부 이벤트가 {b,h,f} 3가지가 있다. 하지만 {C2,C6,C8} 상태가 활성화된 경우에 내부 이벤트가 발생이 되지 않아서 상태 전이가 이루어 질 수 없게 된다. 또한 {C1,C5} 상태가 활성화 된 경우 서로의 조건이 교착 상태에 빠져서 다음 상태로 전이할 수 없게 된다.

• 도달성 트리를 만드는 알고리즘

```
BuildingRT{
 1   Statechart 의 디폴트 상태 구성(State Configuration)인 SC0 를 구성한다.
 2   SC0 를 도달성 트리인 RT 에 할당한다.
 3   Statechart 의 모든 전이(Transition)를 ST 에 할당한다
 4   내부 이벤트가 명세된 모든 전이를 IE 에 할당한다.
 5   발생해야 하는 모든 내부 이벤트를 SIE 에 할당한다.
 6   디폴트 전이에서 발생하는 내부이벤트를 OIE 에 할당하고 SIE 에서는 제거한다.
 7   (SC0, OIE)를 QUEUE 에 할당한다.
 8   WHILE (QUEUE 가 비어있지 않을 경우)
 9     QUEUE 에서 (SC,OIE)쌍을 하나 꺼낸다.
10     SC 에서 시작하는 모든 전이들을 PT 에 할당한다.
11     IF (PT 가 비어있지 않을 경우)
12       FOR (PT 의 각각의 전이(T 라고 하자)에 대해서)
13         IF (T 의 조건이 true 일 경우)
14           IF ( (T 가 IE 에 속해있지 않을 경우) or (T 가 IE 에 속해 있고, 필요한
내부 이벤트가 OIE 에 있을 경우))
15             T 를 PT 와 ST 에서 제거한다.
16             T 로 인해 이동되는 다음 상태 구성 SC'을 얻는다.
17             OIE 를 비운 후, T 로 인해 발생하는 내부 이벤트를 OIE 에 넣고 SIE
에서는 제거한다.
18             IF (SC'이 RT 에 이미 존재할 경우)
19               SC'을 old 상태 구성 노드로 만든다
20             ELSE
21               SC'을 new 상태 구성 노드로 만든다
22               QUEUE 에 (SC', OIE)를 할당한다.
23               SC'을 SC 의 자식 노드로서 RT 에 할당한다.
24             SC 로부터 SC'로 arc 를 연결하면서 <GUARD-EXP>로 이름 붙인다.
25             ENDIF
26           ENDIF
27         ELSE    //(T 의 조건이 false 일 경우)
28           IF (T 가 IE 에 속해 있지 않을 경우)
29             SC'을 'null' 노드로 만든다.
30             SC'을 SC 의 자식 노드로서 RT 에 할당한다.
31             SC 로부터 SC'로 arc 를 연결하면서 <GUARD-EXP>로 이름 붙인다.
32           ENDIF
33         ENDIF
34       ENDFORLOOP
35     ENDIF
36   ENDWHILE
37 }
```

그림 1 도달성 트리를 만들기 위한 알고리즘

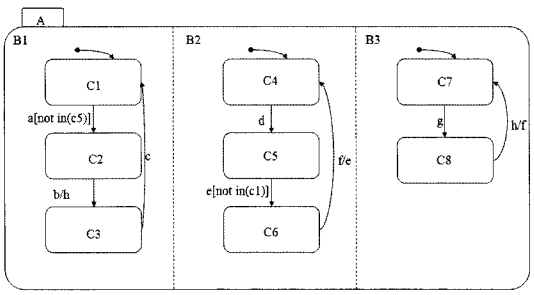

그림 2 도달성 검사를 위한 예제 Statechart

2) Generalization-Specification의 약어로 휘처모델에서의 상속관계를 말한다.(is-a관계)

3) Consist-of는 말 그대로 부모와 자식 휘처들의 관계가 Consist-of인 경우를 말한다.(has-a관계)

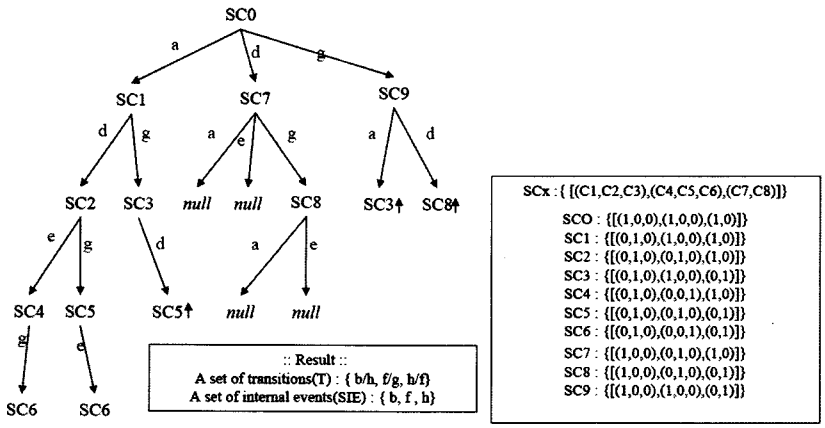

그림 3 그림 2 Statechart의 도달성 트리

그림 3은 제안한 알고리즘에 의해서 그림 2의 도달성 트리를 나타낸 것이다. 여기서 SCx는 상태 구성을 나타낸 것으로 예를 들어{[(1,0,0),(1,0,0),(1,0)]}은 {C1,C4,C7}이 활성화 된 상태를 의미한다. 디폴트 구성인 SC0로부터 시작해서 외·내부 이벤트와 조건에 의해서 상태 전이를 하는 것을 볼 수 있다. 이상의 결과에서 제안한 알고리즘을 이용하면 내부 이벤트의 발생여부와, 모든 전이의 활성화 여부를 검사하여 앞서 정의한 도달성의 여부를 판단할 수 있게 된다.

그림 4에서는 필수 휘처를 제외하고 가능한 휘처 구성들의 집합이 나타나 있다. FC와 FK는 mutex관계를 맺으므로 같은 제품에 들어갈 수 없는 휘처들이다.

그림 5는 도달성이 만족된 Parameterized Statechart를 나타냈다. {FC,FD}가 선택되었을 때는 물론이고, {FC}, {FD}, {FF}, {FD,FK}, {FF,FJ}, {FE,FH}, {FD,FC,FE,

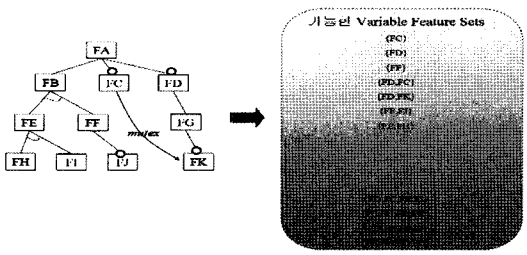

그림 4 휘처 모델과 가능한 휘처 구성들의 집합

FI}, {FE,FH,FD,FK}, {FD,FC,FF,FJ}, {FE,FH,FD,FC} 등의 휘처 구성이 선택되었을 경우에도 도달하지 못하는 상태가 없음을 확인할 수 있다. 그림 5를 보면 전이가 그림 2에 비하여 많이 추가되었다. 도달성을 만족하는 Parameterized Statechart를 설계하기 위해서는 휘처 구성에 따른 변화를 고려해야 함을 알 수 있다.

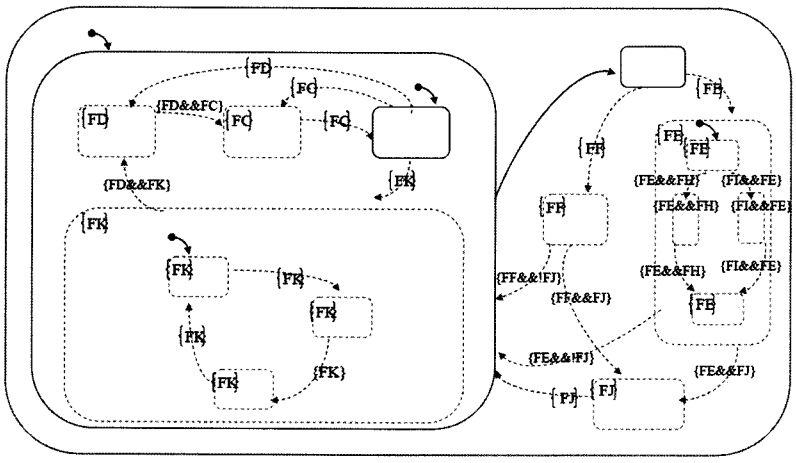

그림 5 도달성이 만족된 Parameterized Statechart

4. UML 2.0 기반의 FORM 아키텍처 모델링을 위한 연구

4.1 FORM 아키텍처를 위한 메타모델의 프로파일

메타 모델로 FORM[16-18]아키텍처를 사례화할 경우, 메타 모델이 갖는 요소간의 관계를 유지하면서 원하는 타입의 엔티티나 커넥터를 추가함으로써 원하는 관점의 아키텍처를 사례화 할 수 있게 된다. 또한, 메타모델 레벨에서 UML2.0의 다른 표준 모델들과의 연관관계와 일관성 규칙을 정의하면, 서브타이핑 된 아키텍처도 그 성질을 상속받아서 사용할 수 있게 되는 장점이 있다. 기존의 네 개의 관점의 아키텍처뿐만 아니라 새로운 관점의 아키텍처를 추가하려고 할 때, 손쉽게 서브타이핑을 할 수 있고, 케이스 툴로 구현할 때에도 유연하게 확장을 할 수 있게 된다.

그림 6을 살펴보면 FORMEntity는 BasicComponents의 Component를 상속받는다. Classifier로서 인스턴스화가 가능하고, BehavioredClassifier와 StructuredClassifier를 상속받았으므로 각각 행위 명세와 구조적인 명세를 포함할 수 있다. EncapsulatedClassifier이므로 포트를 갖고 캡슐화를 통해 정보은닉을 할 수 있다.

FORMInterface는 Interface를 상속받으므로 자신이 필요로 하거나, 공급해 줄 수 있는 오퍼레이션과 속성을 명세할 수 있고, BahavioredClassifier를 상속받으므로 행위명세 또한 포함할 수 있다. FORMConnector는 Connector를 상속받음으로써 StructuredClassifier의 property를 연결하여 내부구조를 형성시켜준다. 그리고 FORM 아키텍처에서는 커넥터에도 구조 명세와 행위 명세를 할 수 있게끔 제안되어 있기 때문에 StructuredClassifier와 BehaviorClassifier를 상속받음으로써 구조적인 명세와 행위 명세를 포함할 수 있게 하였다.

그림 7은 메타모델 프로파일에서 스테레오 타입 사이의 관계를 나타낸 것이다. FORMInterface와 FORMEntity, FORMConnector는 FORMOperation, FORMProperty를 가짐으로써 구조적인 명세에 가변성을 표시할 수 있다. 또 Parameterized된 ASADAL Statechart를 가짐으로써 행위명세에 가변성을 표시할 수 있게 되었다. FORMRole과 FORMPort는 FORMInterface를 타입으로 가짐으로써 FORMInterface에 포함된 구조적인 명세와 행위 명세를 이용할 수 있게 되었다. FORMPort는 FORMEntity에 속해 있으면서, FORMEntity가 캡슐화의 개념을 가질 수 있도록 한다. FORMRole과 FORMPort는 서로 연결관계를 가짐으로써 커넥터와 엔티티를 연결해 줄 수 있게 한다.

4.2 FORM 아키텍처의 프로파일

위에서 제안한 메타모델을 바탕으로 FORM아키텍처의 프로파일을 제안한다. FORM아키텍처에는 4가지 아키텍처가 존재하는데, 여기서는 프로세스아키텍처를 중심으로 FORM 아키텍처의 사례화를 살펴본다.

특정한 관점의 아키텍처를 생성하기 위해서는 메타모델의 요소들을 서브타이핑 해야 하는데 여기서 그림8은 프로세스아키텍처를 생성하기 위해서 메타모델의 요소들을 서브 타이핑한 프로파일을 보이고 있다. 메타모델의 FORMEntity에서 Process, Transient Process, Multiple Process, Transient Multiple Process, Data Object가 서브 타이핑된 것을 알 수 있다. FORMPort와

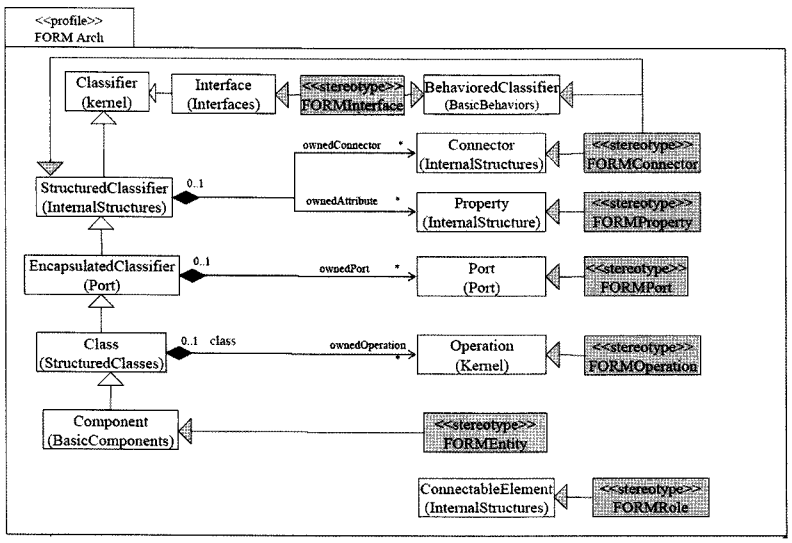

그림 6 FORM 아키텍처를 위한 메타모델의 프로파일1

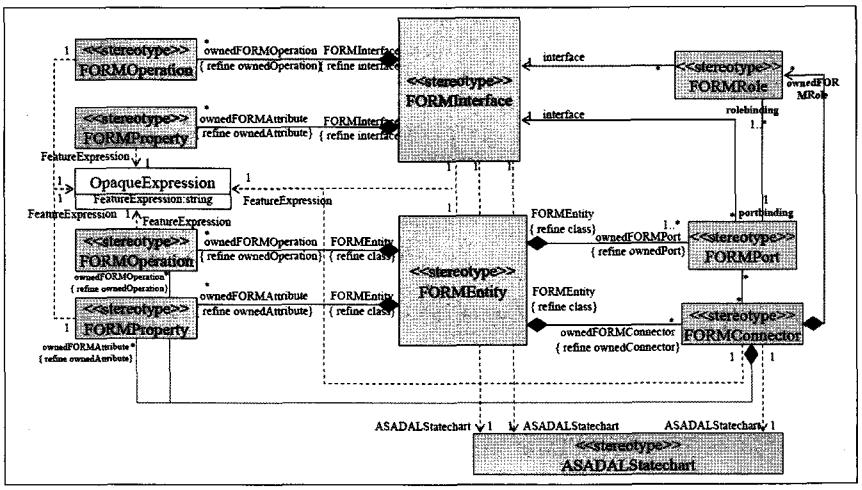

그림 7 FORM 아키텍처를 위한 메타모델의 프로파일2

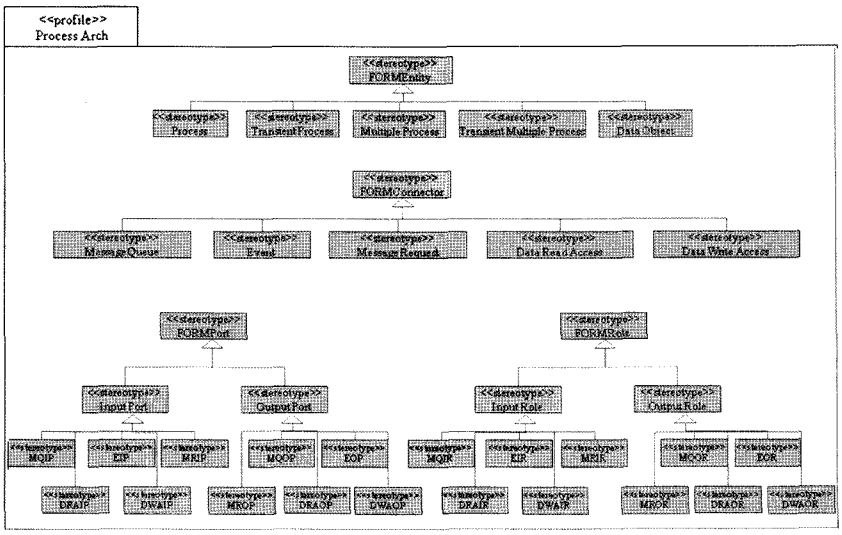

그림 8 프로세스 아키텍처 요소들의 서브타이핑

FORMRole은 각각 입력과 출력으로 나누어지고, 각각의 Connector에 대응되는 포트와 롤(Role)로서 서브 타이핑 된 것을 알 수 있다.

그림 9는 그림 8에서 서브 타이핑된 요소들 사이의 관계를 나타낸 것이다. 이것은 메타 모델에서 정의된 요소들 사이의 관계를 상속 받으면서 프로세스 아키텍처로서 지켜져야 할 추가적인 관계들이 나타나 있다. 메타모델과 같이 FORMEntity를 상속받은 요소들은 FORMConnector를 상속받은 커넥터를 합성할 수 있다. 하지만 FORMEntity를 상속받은 요소 중 Data Object는 FORMPort를 상속받은 요소 중 DWAOP(Data Write Access Output Port)와 DRAIP(Data Read Access Input Port)만을 합성할 수 있다. 이는 Data Object를 접근하기 위해서는 Data Write Access Connector 또는 Data Read Access Connector만을 이용할 수 있고, 따라서 그와 대응되는 포트만을 가져야 하는 프로세스 아키텍처의 제한사항에서 비롯된 것이다. FORMPort를 상속받은 요소들과 FORMRole을 상속받은 요소들이 서로 대응되는 요소 사이에만 각각의 portBinding 관계와, roleBinding관계를 가지는 것도 프로세스아키텍처에서 허용하는 바인딩만을 가능하게 하기 위함이다.

다음은 FORM 아키텍처를 설계할 때 지켜야 하는 문법적 제한사항에 대해서 알아본다.

• FORM 아키텍처의 문법적 제한사항

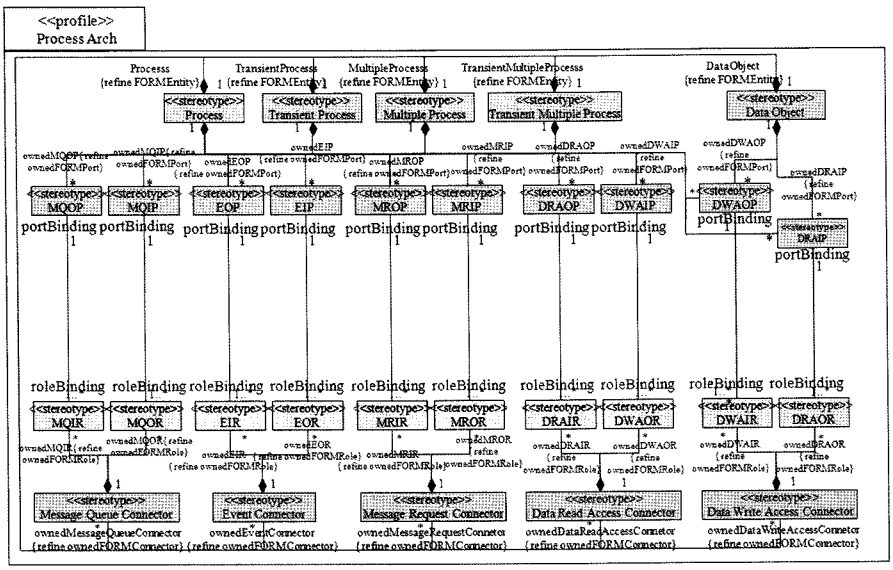

그림 9 프로세스 아키텍처 프로파일 요소들 사이의 관계

① For all e1,e2 ∈ FORMEntity, if e1≠e2 then e1.name≠e2.name

② For all e ∈ FORMEntity and f ∈ Feature and c ∈ FORMConnector, if f ∈ Feature(e)[4] and c ∈ e.FORMConnector and f.type = 'optional' or f.type = 'alternative', then f ∈ Feature(c)

③ For all e,e1 ∈ FORMEntity, ∃c ⊂ FORMConnector such that e $\xrightarrow{C^*}$ e1[5]

④ For all c ∈ FORMConnector, n(c.FORMRole) = n(c.FORMEntity)

⑤ For all e ∈ FORMEntity, n(e.FORMPort) = n(e.FORMConnector)

FORMEntity가 유일한 이름을 가지지 않는다면 아키텍처가 모호하게 설계될 수 있다. 또한 가변성이 명세된 엔티티가 있을 때, 그와 연결된 FORMConnector가 가변성이 명세 되어 있지 않다면, 제품 생성시 엔티티에 연결되어 있지 않은 커넥터가 존재할 수 있게 된다. 규칙 ④를 지키지 않는다면 하나의 커넥터로 엔티티의 입/출력FORMPort와 연결 가능하게 되므로 의미 없는 모델링이 된다.

4.3 FORM아키텍처와 Parameterized Statechart와의 일관성 규칙

• Rule2-1. Parameterized Statechart와 FORM 아키텍처 사이의 일관성 규칙 1

그림 10은 아키텍처 요소에서 이벤트가 전달되는 모습을 나타낸 것이다. 엔티티에서 출력 포트로(1번), 출력 포트에서 입력 롤로(2번), 입력 롤에서 커넥터로(3번), 커넥터에서 출력 롤로(4번), 출력 롤에서 입력 포트로(5번), 입력 포트에서 엔티티로(6번) 이벤트가 전달될 수 있어야 한다. 그러므로 다음과 같은 규칙이 필요하다.

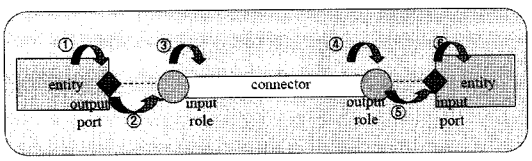

그림 10 아키텍처 요소에서 이벤트의 전달

표 3 FORM 아키텍처와 Parameterized Statechart와의 일관성 규칙 정의를 위해 유용한 함수들

함수 이름	설명
Statechart (element)	아키텍처 요소에 속해있는 Statechart를 반환한다.
GeneratedEvent (statechart)	해당 Statechart에서 발생시키는 이벤트들의 집합을 의미한다.
WaitingEvent (statechart)	해당 Statechart에서 기다리고 있는 이벤트들의 집합을 의미한다.

① For all e ∈ FORMEntity, p ∈ Outputport, se,sp ∈ ASADALStatechart
if se ∈ statechart(e), sp ∈ statechart(p), p ∈ e.FORMPort

4) 휘처(x) : x ∈ FORMEntity or x ∈ FORMConnector일 때, 휘처(x)의 의미는 x에 embedded 되어 있는 휘처들의 집합을 뜻한다.

5) $\xrightarrow{C^*}$의 의미는 집합c의 FORMConnector 들을 조합하면, x로부터 y까지 연결될 수 있다는 의미이다.

then generatedevent(se) ∩ waitingevent(sp) ≠ Ø

② For all p ∈ Outputport, r ∈ Inputrole, sp,sr ∈ ASADALStatechart

if sp ∈ statechart(p), sr ∈ statechart(r), r ∈ p.rolebinding

then generatedevent(sp) ∩ waitingevent(sr) ≠ Ø

③ For all r ∈ Inputrole, c ∈ FORMConnector, sr,sc ∈ ASADALStatechart

if sr ∈ statechart(r), sc ∈ statechart(c), r ∈ c.FORMRole

then generatedevent(sr) ∩ waitingevent(sc) ≠ Ø

④ For all c ∈ FORMConnector, r ∈ Outputrole, sc,sr ∈ ASADALStatechart

if sc ∈ statechart(c), sr ∈ statechart(r), r ∈ c.FORMRole

then generatedevent(sc) ∩ waitingevent(sr) ≠ Ø

⑤ For all r ∈ Outputrole, p ∈ Inputport, sr,sp ∈ ASADALStatechart

if sr ∈ statechart(r), sp ∈ statechart(p), p ∈ r.portbinding

then generatedevent(sr) ∩ waitingevent(sp) ≠ Ø

⑥ For all p ∈ Inputport, e ∈ FORMEntity, sp,se ∈ ASADALStatechart

if sp ∈ statechart(p), se ∈ statechart(e), p ∈ e.FORMPort

then generatedevent(sp) ∩ waitingevent(se) ≠ Ø

위의 의미는 아키텍처에서 통신하려는 두 요소 E1, E2가 있다면 반드시 같은 이름의 보내려는 이벤트와 받으려는 이벤트가 E1, E2에 각각 명세 되어 있어야 한다는 것이다.

• Rule2-2. Parameterized Statechart와 FORM 아키텍처 사이의 일관성 규칙 2

아키텍처 요소에 포함되어 있는 Statechart들은 각각 동시적으로 동작하고 있다. 따라서 AndLine으로 구분된 하나의 Statechart로 생각할 수 있을 것이다. Rule2-2에서는 이벤트가 실제로 발생되고 소비되는지에 대한 동적인 규칙을 정의하려고 한다.

① 아키텍처에 존재하는 Statechart 들의 내부 이벤트들이 모두 발생되어야 한다.

BuildingRT(arch).SIE = Ø

※ BuildingRT()의 'SIE'는 발생되지 못한 Internal Event 들의 집합이다.

② 아키텍처에 존재하는 statechart 들의 전이들이 모두 활성화 되어야 한다.

BuildingRT(arch).ST = Ø

※ BuildingRT() 의 'ST'는 Statechart에 명세된 Transition들 중에 활성화 되지 못한 Transition들의 집합이다.

Rule 2-1에 의해 아키텍처 요소 사이의 통신을 위한 이벤트가 명세 되어있는지를 검사했어도 실제로 해당 이벤트가 생성되지 않거나 또는 소비되지 않는다면 아키텍처 사이의 통신은 이루어 지지 않을 것이다. 이를 검사하기 위해서 3.2절에서 제안한 도달성 검사 알고리즘을 이용한다. 규칙①은 내부 이벤트가 모두 발생되어야 함을 의미한다. 아키텍처 요소간의 통신을 위한 이벤트 역시 내부 이벤트이므로 규칙①을 만족하면 통신에 필요한 이벤트가 모두 생성됨을 알 수 있다. 규칙②는 이벤트들이 모두 소비되어 상태 전이를 일으킬 수 있는지 여부를 알아보는 것이다.

• Rule2-3. Parameterized Statechart와 FORM 아키텍처 사이의 일관성 규칙 3

표 4 Rule 2-3을 정의하기 위한 유용한 함수들

함수 이름	설명
Required_operation (statechart)	해당 Statechart의 전이와 상태에 명세된 오퍼레이션의 집합
Required_attribute (statechart)	해당 statechart의 전이와 상태에 명세된 속성의 집합
Arch_operation (arch)	아키텍처의 엔티티, 커넥터, 인터페이스에 명세된 오퍼레이션의 집합
Arch_attribute(arch)	아키텍처의 엔티티, 커넥터, 인터페이스에 명세된 속성의 집합
RVarch(arch)	휘처 선택에 의해서 아키텍처의 가변성을 제거한 아키텍처를 반환한다.

아키텍처 요소에 Statechart가 포함되어 있을 때, Statechart가 사용하는 오퍼레이션이나 속성은 해당 아키텍처에 명세 되어있는 오퍼레이션이나 속성만을 이용해야 한다. Rule 2-3에서는 Statechart가 사용하는 오퍼레이션이나 속성이 아키텍처의 부분집합임을 검사하는 규칙을 제안한다.

① required_attribute(statechart) ⊆ arch_attribute(arch)

② required_operation(statechart) ⊆ arch_operation(arch)

③ required_attribute(RVPS(statechart)) ⊆ arch_attribute(RVarch(arch))

④ required_operation(RVPS(statechart)) ⊆ arch_operation(RVarch(arch))

Statechart와 아키텍처 사이의 속성과 오퍼레이션의 집합 관계가 부분 집합(⊆)인 이유는 다음과 같다. 아키텍처의 모든 속성이나 오퍼레이션이 특정한 상태에 접근되거나 활성화될 필요는 없을 것이다. 특정 상태에 상

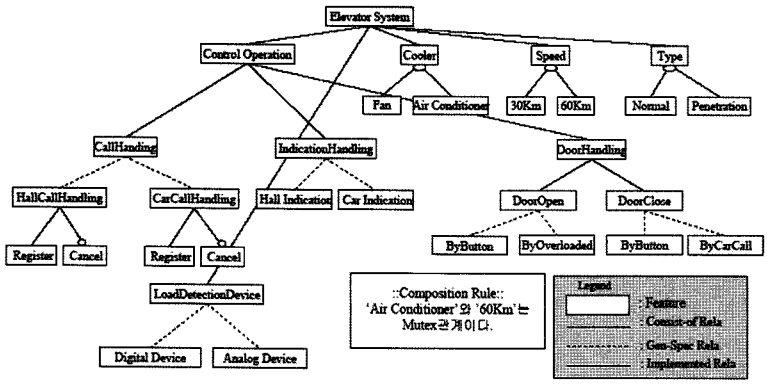

그림 11 엘리베이터 시스템의 휘처모델

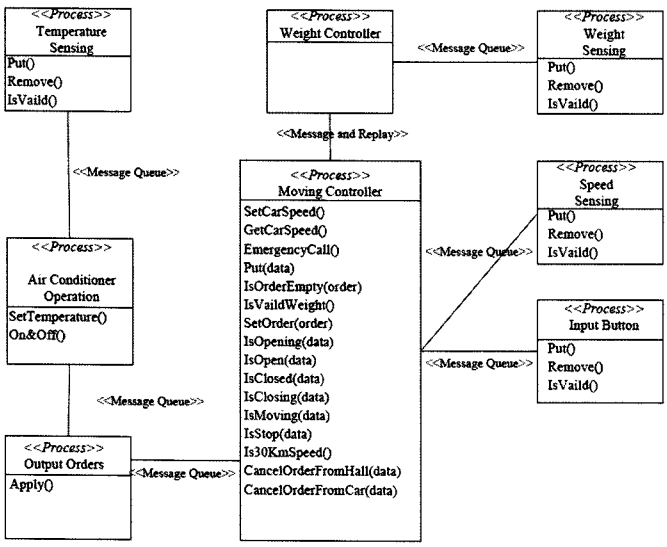

그림 12 휘처 선택에 따라 사례화된 프로세스 아키텍처모델

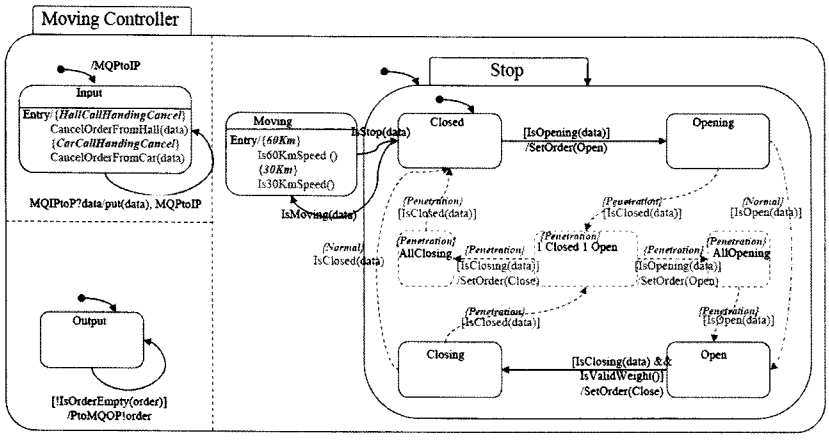

그림 13 가변성이 제거되지 않은 Statechart

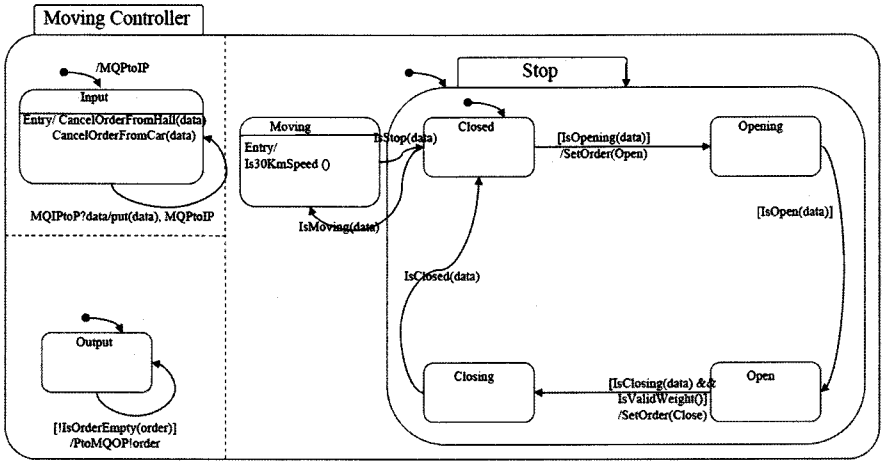

그림 14 휘처 선택에 따라 가변성이 제거된 Statechart

관없이 접근되거나 활성화할 필요성이 있는 속성이나 오퍼레이션이 존재할 것이고, 그것들은 Statechart에 명세 될 필요가 없기 때문이다.

5. 사례연구

5.1 아키텍처 모델과 Parameterized Statechart의 일관성 검증의 예

그림 11의 휘처 모델에서 명령취소와 속도등의 가변성을 가지고 있는데, 아래의 휘처 선택 조합에 대해서 일관성 검증의 예를 보이도록 하겠다. 선택된 휘처의 집합은 '(HallCallHanding) Cancel, (CarCall Handling) Cancel, Normal, Air-Conditioner, 30Km'이라고 하자. 이에 따라 사례화된 아키텍처 모델과 Statechart의 모습은 다음과 같다.

• Rule2-1을 바탕으로 하는 일관성 검사

위의 표를 바탕으로 Rule 2-1에서 정의한 규칙을 검증해 보겠다.

인접한 두 아키텍처 요소의 Statechart사이에 서로

표 5 : Statechart들이 발생시키고 기다리는 이벤트 정리표

각 아키텍처 요소의 Statechart	발생시키는 이벤트	기다리는 이벤트
Input Button	PtoMQOP!data	EnvToP?data
Message Queue Output Port	MQOPtoIR!data	PtoMQOP?data
Message Queue Input Role	MQIRtoC!data	MQOPtoIR?data
Message Queue Connector	MQCtoOR!Get()	MQIRtoC!data MQORtoC
Message Queue Output Role	MQORtoC MQORtoIP!data	MQCtoOR!data MQIPtoOR
Message Queue Input Port	MQIPtoP!data MQIPtoOR	MQORtoIP?data MQPtoIP
Moving Controller	MQPtoIP PtoMQOP!order	MQIPtoP?data

표 6 Rule2-1검증 결과

인접한 아키텍처 요소의 Statechart		Generatedevent(앞) ∩ Waitingevent(뒤)	검증 결과
(앞)	(뒤)		
Input Button	Message Queue Output Port	PtoMQOP	OK!
Message Queue Output Port	Message Queue Input Role	MQOPtoIR	OK!
Message Queue Input Role	Message Queue Connector	MQIRtoC	OK!
Message Queue Connector	Message Queue Output Role	MQCtoOR	OK!
Message Queue Output Role	Message Queue Input Port	MQORtoIP	OK!
Message Queue Input Port	Moving Controller	MQIPtoC	OK!

표 7 Input Button으로부터 Moving Controller까지 이벤트·데이터 전달과정 시나리오

State Configuration	활성화된 전이(Transition)	발생한 이벤트	설명
C0	Default 전이	MQPtoIP EnvToP!data	- 초기 상태 - 외부 환경으로부터Input Button으로 데이터가 전송 - Moving Controller에서 입력 포트로 이벤트 전달(data요청)
C1	MQPtoIP EnvToP?data MQIPtoP?data	MQIPtoOR	- 입력 포트에서 출력 롤로 이벤트 발생(data요청)
C2	[IsValid(data)] MQIPtoOR	PtoMQOP!data MQORtoC	- Input Button으로부터 출력 포트로 이벤트 발생 (data전송) - 출력 롤로부터 커넥터에 이벤트 발생 (data 요청)
C3	PtoMQOP?data MQORtoC	MQOPtoIR!data	- 출력 포트로부터 입력 롤에 이벤트 전달(data전송)
C4	MQOPtoIR?data	MQIRtoC!data	- 입력 롤로부터 커넥터로 이벤트 전달(data전송)
C5	MQIRtoC?data	MQCtoOR!data	- 커넥터로부터 출력 롤로 이벤트 전달(data전송)
C6	MQCtoOR?data	MQORtoIP!data	- 출력 롤로부터 입력 포트로 데이터 전달(data전송)
C7	MQORtoIP?data	MQIPtoP!data	- 입력 포트로부터 Moving Controller에 이벤트 전달
C8	MQIPtoP?data	MQPtoIP	- Moving Controller에 data가 전달됨. Put()명령으로 data저장 - **통신성공!**

약속한 이벤트가 모두 존재하므로 Input Button으로부터 Moving Controller까지의 통신을 위한 이벤트 명세가 옳게 되었음을 알 수 있다.

• Rule2-2를 바탕으로 하는 일관성 검사

Rule2-1이 정적인 검사였다면, Rule2-2를 통한 검사는 동적인 검사이다. 이번 차례에서는 도달성 트리중 일부 시나리오를 통해서 Input Button으로부터 Moving Controller까지 이벤트가 올바로 전송되는 과정을 보이도록 하겠다.

이는 도달성 트리중 일부이며, 엘리베이터 시스템 환경에서 이벤트가 데이터와 결합하여 발생하고, 그것이 Moving Controller까지 도착하는 시나리오를 나타낸 것이다. Rule 2-2인 'BuildingRT(arch).SIE = Ø'인 것을 확인했고, 따라서 내부 이벤트가 모두 발생되었음을 알 수 있다. 순서는 C0부터 C8까지 상태 구성(State Configuration)을 차례로 나타내었다. 이로서 엘리베이터 프로세스 아키텍처의 Statechart들이 서로 통신할 수 있음을 알 수 있다.

• Rule2-3을 바탕으로 하는 일관성 검사

Moving Controller 프로세스와 Moving Controller Statechart와의 관계에서 Rule 2-3을 검사해 보겠다.

사례연구에서 처음 피처를 선택했을 때 가변성을 제거한 것을 그림 12에서 볼 수 있는데, 본래 가변성을 제거하지 않고 있던 명세에서 Is60kmSpeed()오퍼레이션이 빠진 것을 볼 수 있고, 또 그림 14의 Statechart에서도 Is60kmSpeed()가 빠져있는 것을 볼 수 있다. 그러므로 이는 Rule 2-3을 만족하므로 일관성이 유지된다고 결론 내릴 수 있다.

6. 결 론

본 연구에서는 FORM 아키텍처를 UML2.0으로 모델링 할 수 있도록 프로파일을 제시하였다. 또한 UML2.0의 행위모델중의 하나인 State machine을 확장하여 가변성을 명세할 수 있는 Parameterized Statechart 프로파일을 제시하였다. 끝으로 FORM 아키텍처와 Parameterized Statechart사이의 일관성이 유지되었는지 검사하는 규칙을 제안하였다.

본 연구는 다음과 같은 의의를 지닌다. 첫째, 고 생산성, 고품질의 제품 생산 라인을 구축하기 위해 재사용 방법론 중의 하나인 MDA방법론과의 접목을 시작하였다는 것이다. MDA방법론의 표준 명세 언어인 UML2.0을 이용하여 FORM 아키텍처를 모델링 할 수 있게 되었고, UML의 행위 모델을 사용할 수 있게 되었다. 둘째, Statechart에 가변성을 표현할 수 있게 되었다. 따라서 휘처모델에서 식별한 가변성을 표현할 수 있게 되었다. 따라서 휘처 모델에서 식별한 가변성을 명세할 수 있으며, 일관성 규칙에 의해서 휘처가 올바르게 할당되었는지 검사할 수 있다. 셋째, Parameterized Statechart와 그를 포함하고 있는 아키텍처 사이의 일관성을 세가지 측면에서 검증할 수 있다.

향후 연구 방향은 다음과 같다. 우선 UML2.0이 지원하는 다른 행위 명세들(e.g. 시퀀스 다이어그램, 액티비티 다이어그램)과의 연결이 필요하다. 이것은 아키텍처의 의미를 좀더 풍부하게 해줄 것이다. 또한 UML2.0으로 모델링 된 네 가지 관점의 FORM아키텍처를 MDA 방법론에서 제시하는 형식의 PIM으로 변환할 수 있는 변환 규칙을 정의해야 할 것이다. Statechart에 도메인

의 가변성을 모델링 하는 것은 모델의 복잡도를 굉장히 증가시키는 결과를 낳으므로 Parameterized Statechart의 복잡도를 낮추거나 효과적으로 관리할 수 있는 연구가 필요할 것이다. 마지막으로 가변성이 명세된 모델로부터 실행코드를 생성할 수 있는 자동화 도구개발도 필요하다.

참 고 문 헌

[1] Anneke Kleppe, Jos Warmer, Wim Bast, "MDA Explained: The Model Driven Architecture--Practice and Promise," Addison-Wesley, 1998.

[2] Stephen J. Mellor, Kendall Scott, Axel Uhl, Dirk Weise, "MDA Distilled (Principles of Model-Driven Architecture)," Addison-Wesley, 2004.

[3] http://www.omg.org/mda/

[4] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis(FODA) Feasibility Study," Technical Report CMU/SEI-90-TR-21, Pittsburgh, PA., Software Engineering Institute, Carnegie Mellon University, 1990.

[5] Jorge Enrique Pérez-Martínez, Almudena Sierra-Alonso, "UML 1.4 versus UML 2.0 as Languages to Describe Software Architectures," pp. 88-102, EWSA 2004.

[6] Apostolos Zarras, Val´erie Issarny, Christos Kloukinas, Viet Khoi Nguyen, "Towards a Base UML Profile for Architecture Description," Proceedings of ICSE 2001 Workshop for Describing Software Architecture with UML, IEEE Computer Society, Toronto, Ontario, Canada, pp. 22-26, May 2001.

[7] Kirsten Berkenk¨otter, Stefan Bisanz, Ulrich Hannemann, Jan Peleska, "HybridUML Profile for UML 2.0," SVERTS, workshop hold in conjunction with UML, 2003.

[8] C. Hofmeister, R. L. Nord, D. Soni, "Describing Software Architecture with UML," Proceedings of the First Working IFIP Conference on Software Architecture, 1999.

[9] Miguel Goulão, Fernando Brito e Abreu, "Bridging the gap between Acme and UML for CBD," Specification and Verification of Component-Based Systems (SAVCBS'03), Workshop at ESEC/FSE 2003, pp. 75-79, Sep 2003.

[10] Petri Selonen, Jianli Xu, "Validating UML Models Against Architectural Profiles," ESEC/FSE 2003, Helsinki, Finland, Sep 2003.

[11] Sunghwan Roh, Kyungrae Kim, Taewoong Jeon, "Architecture Modeling Language based on UML 2.0," Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04), 2004.

[12] David Harel, "Statecharts: A Visual Formalism For Complex Systems," Science of Computer Programming, pp. 231-274, 1987.

[13] P C Masiero, J C Maldonado, I G Boaventura, "A reachability tree for statecharts and analysis of some properties," Information and Software Technology, Volume 36, 1994.

[14] Zs. Pap, I. Majzik1, A. Pataricza, A. Szegi, "Completeness and Consistency Analysis of UML Statechart Specifications," Proc. of IEEE Design and Diagnostics of Electronic Circuits, 2001.

[15] [Book] Hassan Gomaa, "Designing Software Product Lines with UML : From Use Cases to Pattern-Based Software Architectures," Addison-Wesley, pp. 169-204, 2004.

[16] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," Annals of Software Engineering, pp. 143-168, May 1998.

[17] Kyo C. Kang, Jaejoon Lee, and Patrick Donohoe, "Feature-Oriented Product Line Engineering," IEEE Software, Vol.9, No.4, pp. 58-65, July/August 2002.

[18] 채원석, "소프트웨어 아키텍쳐 모델 기술 및 분석을 위한 메타 모델", 석사 학위 논문, 포항공과대학교 대학원, 2001.

양 경 모

2006년 포항공과대학교 컴퓨터공학과 졸업(학사). 2008년 포항공과대학교 컴퓨터공학과 졸업(석사). 2008년~현재 삼성전자 DM사업부. 관심분야는 소프트웨어 재사용, 제품라인 공학

조 윤 호

2008년 명지대학교 컴퓨터공학과 졸업 2008년~현재 포항공과대학교 정보통신대학원 석사과정. 관심분야는 소프트웨어 공학, 소프트웨어 재사용, 제품 라인 공학

강 교 철

1973년 고려대학교 통계학과 졸업(학사) 1974년 콜로라도 대학교 산업공학 졸업(석사). 1982년~1984년 미시간대학교 Visiting Professor. 1984년~1985년 (미국)벨 통신 연구소 Technical Staff. 1985년~1987년 AT&T 벨 연구소 Technical Staff. 1987년~1992년 카네기멜론 대학교 Project Leader. 1992년~현재 포항공과대학교 교수. 2000년~2001년 카네기멜론대학교 Visiting Scientist. 관심분야는 소프트웨어 공학, 실시간 내장형 시스템, 소프트웨어 재사용, 제품라인 공학