UML 구조 다이어그램과 행위 다이어그램의 일관성 메타검증

(Meta-Validation for Consistency between UML Structural Diagram and Behavioral Diagram)

하 일 규 † 강 병 욱 ††
(Il Kyu Ha) (Byung Wook Kang)

요 약 UML은 객체지향 모델링에 있어서 표준으로 받아들여지고 있다. UML은 풍부한 구성요소를 가지므로 개발하고자하는 시스템을 상세하게 묘사할 수 있지만, 모델링된 다이어그램의 정확성과 일관성은 보장하지 못한다는 결점을 가진다. 따라서 개발프로세스의 초기단계에서 사용자 모델을 검증함으로써 오류를 최소화하는 것이 중요하다. 본 연구에서는 메타-메타모델과 OCL로 표현된 검증규칙을 이용하여 UML structural 다이어그램과 behavioral 다이어그램의 일관성을 검증하는 방법을 제안한다. 일관성은 하나의 요구사항을 가지고 작성된 structural 다이어그램과 behavioral 다이어그램이 일관성있게 작성되었는지를 판단하기 위한 성질이다. 검증의 첫 번째 작업으로서 UML 다이어그램과 그들사이의 관련요소로 표현된 메타-메타모델을 유도하고, 유도된 메타-메타모델로부터 일관성을 검증하기 위한 규칙을 유도하고, 유도된 검증규칙은 검증작업의 자동화를 위해 OCL과 같은 정형적인 언어로 명세한다. 마지막으로 사례모델을 통해 검증규칙의 유용성을 검증한다.

키워드 : UML, OCL, 일관성, 메타모델, 메타검증

Abstract The UML is a widely accepted standard in object-oriented modeling. As the UML is semantically rich, we can describe in detail the system that will be developed, but we cannot guarantee the correctness and consistency of the designed model. Therefore, it is important to minimize the error by verifying user models in an early stage. In this paper, we propose a method for verifying the consistency of UML structural diagrams and behavioral diagrams using OCL verification rules and meta-metamodel. The consistency is a nature for checking whether the structural diagrams and behavioral diagrams are coherently designed according to a specific requirement. First we build meta-metamodels of the structural diagram and behavioral diagram that are described with the UML diagrams and the related elements, we derive rules for verifying the consistency from each meta-metamodels, and then formally specify with the language such as OCL for automatic verification. Finally, we verify the usefulness of the rule through a case study.

Key words : UML, OCL, Consistency, Metamodel, Meta-validation

1. 서 론

UML(Unified Modeling Language)[1,2]는 객체지향 모델링 방법의 통합된 표준 언어로서 업계의 표준으로 사용되어 왔다. UML은 풍부한 구성요소를 가지므로 개발하고자 하는 시스템을 바라보는 관점에 따라 다양하고 상세하게 표현할 수 있지만, UML로 작성된 다이어그램에 대한 정확함은 보장하지 못한다. 정확하지 못한 다이어그램의 영향은 개발 프로세스가 진행될수록 오류가 증폭되므로 개발 초기단계에 정확함의 검증을 통해 오류를 최소화하는 것이 필요하다. 모델의 정확함은 3가지로 구분할 수 있다. 즉 완전성(completeness), 일관성(consistency), 정확성(correctness)이다. 완전성은 사용자가 작성한 다이어그램과 초기의 사용자 요구사항과의 일치여부를 검증하기 위한 성질이다. 초기의 사용자 요구사항이 완전하게 다이어그램에 반영되었을 때 완전성이 있다고 할 수 있다. 일관성은 사용자가 작성한 한 종류의 다이어그램과 다른 종류의 다이어그램사이의 의미

† 정 회 원 : 영남대학교 컴퓨터공학과
ilkyuha@yumail.ac.kr
†† 종신회원 : 영남대학교 컴퓨터공학과 교수
bwkang@yu.ac.kr
논문접수 : 2003년 2월 25일
심사완료 : 2003년 8월 21일

의 일치여부를 검증하기 위한 성질이다. 즉 사용자가 작성하는 9가지의 UML 다이어그램은 하나의 요구사항으로부터 작성된 것이므로 그들이 일관성 있게 작성되었는지를 검증하는 것이다. 정확성은 사용자가 작성한 다이어그램과 UML 표준의 일치여부를 검증하기 위한 성질이다. 본 연구에서는 다이어그램의 정확함에 관한 성질 중 일관성에 초점을 두고 메타-메타모델(meta-metamodel)과 OCL(Object Constraint Language)[3,4]로 표현된 검증규칙을 이용하여 UML structural 다이어그램과 behavioral 다이어그램의 일관성을 검증하는 방법을 제안한다. 검증의 초기단계로서 UML의 9가지 다이어그램의 메타모델(metamodel)을 고려하여 structural 다이어그램과 behavioral 다이어그램의 메타-메타모델을 유도한다. 그리고 유도된 메타-메타모델로부터 structural 다이어그램과 behavioral 다이어그램의 일관성을 검증하기 위한 검증규칙을 유도한다. 유도된 검증규칙은 검증을 자동화하고 모델을 명확하게 이해할 수 있도록 하기 위해 정형적인 형태로 명세한다. 본 연구에서는 UML의 표준모델 제한언어로 사용되고 있는 OCL을 이용하여 규칙을 명세한다. 마지막으로 하나의 사례모델을 통해 유도한 검증규칙의 유용성을 검증한다.

본 논문의 구성은 다음과 같다. 2장에서는 UML 다이어그램의 일관성 검증 방법과 관련한 기존의 연구에 관하여 조사 분석한다. 3장에서는 각 다이어그램의 메타모델을 고려하여 structural 다이어그램과 behavioral 다이어그램의 메타-메타모델을 유도하고 다이어그램간의 관계성을 분석한다. 4장에서는 메타-메타모델로부터 다이어그램간의 일관성을 검증하기 위한 검증규칙을 유도하고, 유도한 검증규칙을 OCL로 정형적으로 명세한다. 5장에서는 예제 모델을 통하여 일관성 검증규칙의 유용성을 검증한다. 마지막으로 6장에서는 결론을 내리고 향후 연구과제를 제시한다.

2. 관련 연구

UML 다이어그램의 일관성과 관련한 연구로는 다이어그램간의 관계분석에 관한 연구[5,6]와 다이어그램의 관계를 기초로 다이어그램간의 일관성을 검증하는 방법에 관한 연구, 그리고 검증작업을 구현하는 방법으로서 검증자동화와 관련한 연구로 나눌 수 있다.

다이어그램간의 분석에 관한 연구[5]는 다이어그램 사이의 관계를 분석하고 그 관계에 바탕을 둔 변환 오퍼레이션을 유도하여 특정 다이어그램으로부터 또 다른 다이어그램으로의 변환을 시도한다. 이러한 변환 오퍼레이션은 모델 검증, 모델 통합, 모델 분리 등에 도움을 줄 수 있다. [6]에서는 개발하고자 하는 시스템을 바라보는 관점(views)에 따라 작성된 다이어그램 사이의 구조적 불일치성을 밝히고 각 관점을 통합한 형태의 모델을 통해 그러한 불일치성을 좀 더 쉽게 밝혀보고자 하는 것이다.

일관성 검증방법에 관한 연구는 메타모델기반 검증방법[7,8], 그래프기반 검증방법[9-11], 시나리오기반 검증방법[12], 제약언어기반 검증방법[13-17]과 같은 것이 있다.

메타모델기반 검증방법은 초기 작업으로서 UML 다이어그램의 메타모델을 유도하고, 나타난 다이어그램 사이의 관계성을 바탕으로 일관성을 검증하기 위한 검증규칙을 유도한다. 유도한 규칙을 사용자가 작성한 다이어그램에 적용하여 일관성을 검증하는 방법이다. 이 방법은 검증한 규칙을 자동화도구에 의해 검증하기 위해 특정한 언어로 변환하게 되는데, 그 언어가 UML 다이어그램의 각 구성요소를 표현하기에 부족함이 있고 나아가 검증규칙을 표현하기에 부족한 면이 있다.

그래프기반 검증방법의 하나로서 [9,10]은 검증하고자 하는 다이어그램을 특정한 형태의 그래프로 변형시키고 그 그래프에 그래프 grammar를 적용하여 일관성을 검증하는 방법이다. [11]은 다이어그램을 구성하고 있는 component와 component사이의 관계를 표현한 개념그래프(conceptual graph)를 사용하여 검증하는 방법으로서 사용자가 작성한 다이어그램을 개념그래프로 변환하고 이들을 검증규칙을 표현한 개념그래프에 적용함으로써 다이어그램사이의 일관성을 밝히는 것이다. 이 방법은 메타모델 대신에 개념그래프를 사용하여 다이어그램의 정확함을 검증하고자 하는 것으로, UML 다이어그램의 각 요소를 개념그래프의 각 요소로 변환하는 규칙과 대응관계를 설정하는 것이 필요하다. 따라서 이 방법은 번거로운 변환작업을 요구한다.

시나리오기반 검증방법[12]은 UML 다이어그램을 정형적으로 명세하는 과정에서 정적인 정보로서 class 정보를 도출하고 동적인 정보로서 시나리오 정보를 도출하여, 정적인 정보인 class 정보가 동적인 정보인 시나리오에 포함되어 있는지를 통하여 정적모델과 동적모델 사이의 일관성을 검증한다. 이 방법은 객체의 정적구조를 표현하는 class의 시나리오 포함여부라는 단지 하나의 규칙에 의해서만 일관성을 검증하므로 완벽하지 못한 면이 있고, 또한 9개 UML 다이어그램간의 구체적인 일관성 검증방법은 제시되어 있지 않다.

제약언어를 이용한 검증방법은 객체지향모델이 정확하게 작성되어야 함을 나타내는 제한조건을 사용하여 모델의 정확함을 검증하는 방법이다. 이 방법은 객체모델을 고유한 명세로 표현하고, 일관성 또는 완전성과 같은 제약조건을 특정한 제약언어로 표현한다. 제약조건은 검증시스템의 검증 알고리즘으로 사용하고, 명세된 객체

모델에 검증 알고리즘을 적용하는 방법을 취한다. [17]에서는 제약언어로서 MCL(Model Constraint Language)을 이용하고, [13-16]에서는 OCL을 이용하고 있으나 다이어그램별 일관성과 정확성의 검증방법은 제시하지 않고 있다. 제약언어를 이용한 검증방법은 객체모델을 고유한 명세로 표현하기 위한 표현규칙이 요구되며, 검증규칙을 표현하기 위한 제약언어가 요구된다.

검증의 자동화와 관련한 연구는 주로 제약언어를 이용하여 모델의 검증규칙을 표현하고 검증하고자하는 모델을 정형적으로 명세하여 검증시스템에 입력함으로써 일관성을 자동검증하고 있다. 이와 같은 연구로는 [13-17]과 같은 것이 있다.

본 연구는 메타-메타모델과 검증규칙을 기반으로 다이어그램의 일관성을 검증하는 방법으로서, 특정적으로 UML의 표준제약언어로 사용되고 있는 OCL을 이용하여 검증한다. 본 연구에서 사용하고 있는 검증방법의 전체구조는 그림 1과 같고 그 검증과정은 다음과 같이 설명할 수 있다. 첫째, 검증의 기초작업으로서 UML 명세로부터 각 다이어그램의 메타모델을 유도하고, 유도한 메타모델을 바탕으로 structural 다이어그램과 behavioral 다이어그램 사이의 관계성을 표현하는 메타-메타모델을 유도한다. 둘째, 메타-메타모델을 바탕으로 structural 다이어그램과 behavioral 다이어그램 사이의 관계요소를 파악하고 일관성을 검증하기 위한 규칙을 유도한다. 셋째, 유도한 검증규칙은 OCL을 이용하여 정형적으로 표현한다. 넷째, 사용자가 작성한 다이어그램을 정형적 형태로 표현하여 OCL로 표현한 검증규칙과 함께 통합된 명세 형태로 검증시스템에 입력한다. 마지막으로 기존의 OCL관련 도구를 이용하여 검증한다.

본 연구에서는 일관성 검증의 각 방법에서 나타나는

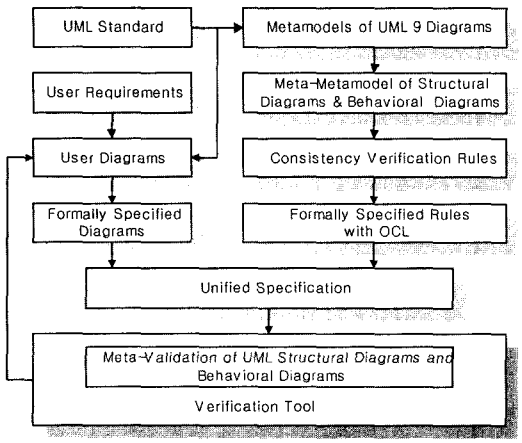


그림 1 메타-메타모델과 검증규칙을 이용한 검증방법의 구조

문제점인 변환작업의 복잡함을 최소화하기 위하여 메타모델에서 검증규칙에 이르는 과정과 사용자 다이어그램을 정형 명세하는 과정을 최소화하고, 시나리오기반 검증방법의 부족한 부분인 정확하고 완전한 일관성 검증규칙을 위하여 메타-메타모델을 유도하여 보다 넓은 시각으로 다이어그램간의 관계를 파악한다. 그리고 메타모델기반 검증방법과 제약언어기반 검증방법의 부족한 부분인 검증규칙의 정확한 표현을 위하여 UML 표준 제약언어인 OCL을 사용한다. 메타-메타모델과 검증규칙 표현언어로서 OCL을 사용함으로써 다음과 같은 효과를 얻을 수 있다. 첫째, UML 표준에 맞게 고안된 언어를 사용함으로써 UML 구성요소와 제한조건을 포함하는 검증규칙을 명세하는 데 보다 적합하다는 이점을 가진다. 둘째, OCL은 기존의 복잡한 정형명세 방법에 비하여 프로그래밍언어 형태로 구성되어 있으므로 검증규칙을 명세하기가 용이하고 이해하기 쉽다. 셋째, OCL은 정형적인 언어로서 문법이 정의되어 있으므로 검증을 위한 자동화가 용이하다는 이점을 가진다. 넷째, 메타-메타모델을 사용함으로써 UML 다이어그램 9개 모두를 고려하여 전체적인 관점에서 일관성 관계를 파악할 수 있다. 다섯째, 9개 UML 다이어그램의 메타모델을 기반으로 메타-메타모델을 작성함으로써 정확성에 바탕을 둔 메타-메타모델을 작성할 수 있다.

3. 메타-메타모델 유도와 관계성 분석

Structural 다이어그램은 시스템의 정적인 면을 모델링하기 위한 다이어그램으로 UML의 9가지 다이어그램 중 class, object, component, deployment 다이어그램이 여기에 속한다. Behavioral 다이어그램은 시스템의 동적인 면을 모델링하기 위한 다이어그램을 말하며 use case, sequence, collaboration, statechart, activity 다이어그램이 여기에 속한다[18]. 이러한 structural 다이어그램과 behavioral 다이어그램의 메타모델을 유도함으로써 다이어그램간의 관계성을 파악할 수 있고, 서로 일관성 관계에 있는 요소를 파악할 수 있다. 일반적으로 메타모델은 다이어그램의 원형(prototype)과 같은 것으로 다이어그램의 기본적 구조를 표현하는 것이며, 메타-메타모델은 메타모델의 메타모델이라고 할 수 있다. UML specification[1]에는 UML 구성요소의 정확한 의미와 구성요소간의 관계를 설명하기 위해 메타모델링 접근방식을 이용한다. 이러한 접근방식은 그림 2와 같이 4단계의 구조로 나타난다. 최상위의 메타-메타모델은 메타모델의 하부구조를 정의하는 모델로서 UML에서는 MOF로 정의되어진다. 본 연구의 structural 다이어그램의 메타-메타모델과 behavioral 다이어그램의 메타-메타모델이 이 단계에 해당된다. 두 번째 단계인 메타모델

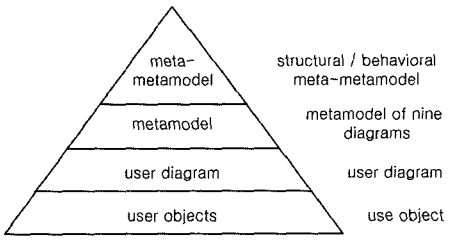


그림 2 메타모델링의 4단계

은 UML의 구성요소의 문법을 정의하는 것으로 UML semantics[19]에서는 'Behavioral Elements', 'Model Management', 'Foundation' package로 나누어 메타모델을 제시하고 있다. 본 연구의 UML 9가지 다이어그램에 대한 메타모델이 이 단계에 해당한다. 세 번째 단계인 사용자 다이어그램(user diagram)은 메타모델을 기초로 사용자가 작성한 다이어그램이다. 네 번째 단계인 사용자 객체(user objects)는 사용자 다이어그램의 하나의 인스턴스이다. 이와 같은 구조 내에서 최상위 단계인 메타-메타모델을 유도함으로써 하위단계인 메타모델간의 관계성을 파악할 수 있고, 관계성 분석을 통해 일관성 관계요소를 파악할 수 있다. 나아가 structural 다이어그램의 메타-메타모델과 behavioral 다이어그램의 메타-메타모델을 통해 사용자가 작성한 다이어그램의 정확성과 일관성을 검증할 수 있다.

메타-메타모델은 메타모델을 기반으로 구성한다. 메타모델은 UML 9개 다이어그램에 대하여 각각 작성되며, 각 다이어그램을 구성하는 구성요소와 그들 사이의 관계를 표현한다. 또 각 메타모델은 관계성을 가지는 다이어그램과의 관계요소를 표현한다. 그림 3은 class 다이어그램의 메타모델을 표현한 것이다. 이렇게 표현된 9개 메타모델을 각각 성격이 유사한 두 가지 부류 즉 Structural 다이어그램과 Behavioral 다이어그램으로 구분하여 각각에 대하여 메타-메타모델을 작성한다. 메타-메타모델에서 각 다이어그램은 하나의 패키지로 표현하고 다이어그램간의 관계요소만을 따로 분리해내어 패키지와 패키지 사이에 배치한다.

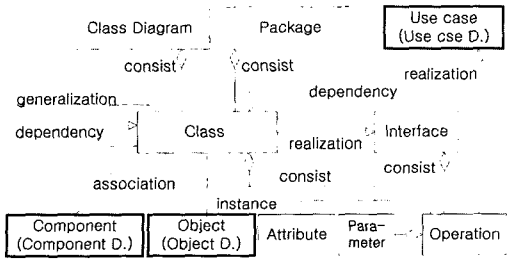


그림 3 Class 다이어그램의 메타모델

Structural 다이어그램의 메타-메타모델은 structural 다이어그램에 속하는 각 다이어그램을 하나의 패키지로 표현하고 class, object와 같은 관계요소를 패키지 사이에 배치한다. 유도한 structural 다이어그램의 메타-메타모델은 그림 4와 같다. 유도한 메타-메타모델에는 class, object, component, node와 같은 관계요소가 중심을 이루고, 요소들 사이에 일반화(generalization), 집합연관(aggregation), 의존(dependency)관계가 있다. Behavioral 다이어그램의 메타-메타모델은 그림 5와 같이 유도한다. Behavioral 다이어그램의 메타-메타모델은 object와 state를 중심으로 callaction, signal, event와 같은 동적인 요소로 구성되어 있다.

이와 같은 메타-메타모델은 다이어그램간의 관계를 이해하는데 도움을 준다. 즉 하위 다이어그램간의 관계성 존재여부와 관계요소를 파악할 수 있다. 위의 메타-

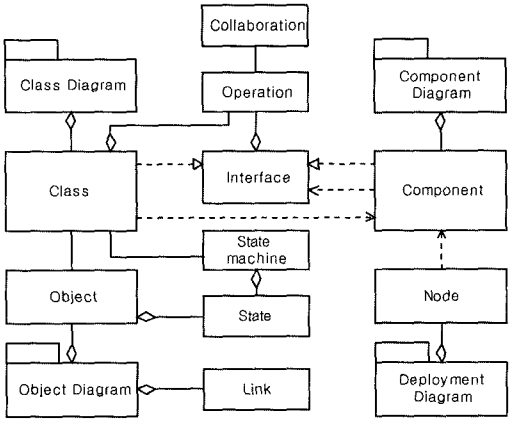


그림 4 Structural 다이어그램의 메타-메타모델

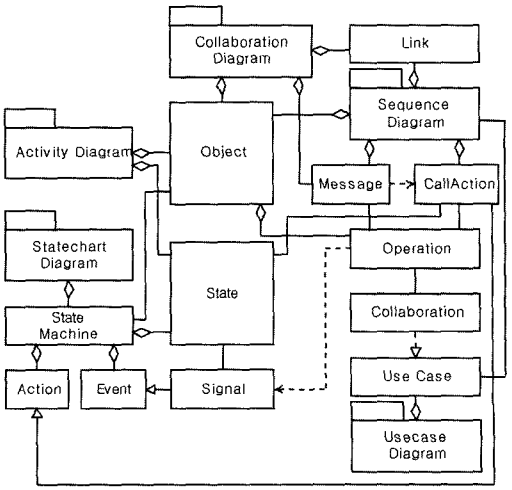


그림 5 Behavioral 다이어그램의 메타-메타모델

메타모델을 기초로 UML의 9가지 다이어그램간의 관계를 표현하면 그림 6과 같다. Structural 다이어그램의 메타-메타모델에서는 class 다이어그램과 object 다이어그램이 관계의 중심을 이루고, behavioral 다이어그램의 메타-메타모델에서는 sequence 다이어그램과 collaboration 다이어그램이 관계의 중심을 이룬다.

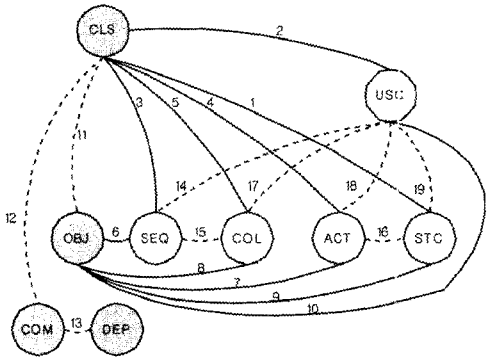
그림 6에서 1은 class 다이어그램과 statechart 다이어그램의 관계를 표현한다. 한 class의 동적인 측면을 모델링 하고자 할 때는 일반적으로 statemachine을 사용한다[18, pp.335]. 따라서 class 다이어그램의 class와 statechart 다이어그램의 statemanine사이에는 연관관계가 있다. Statechart 다이어그램에는 state와 event, action과 같은 구성요소를 포함한다. Event는 상태전이를 촉발시킬 수 있는 자극을 말하며, action은 상태를 변경시킬 수 있는 동작을 말한다[18, pp.333]. Event (callevent)는 class의 operation으로 모델링할 수 있으므로[18, pp.282] statemachine의 event와 class의 operation사이에 연관관계를 가진다.

2는 class 다이어그램과 use case 다이어그램간의 관계를 표현한다. Use case 다이어그램은 시스템의 행동을 모델링하고 조직화하기 위해 사용하는 다이어그램으로 use case와 행위자로 구성된다. Use case 다이어그램은 class의 행동을 모델링하는 것이므로 이에 해당하는 class가 class 다이어그램 내에 존재하여야 한다[18, pp.233]. 따라서 class 다이어그램의 class와 use case 다이어그램의 use case는 연관관계를 가진다.

3은 class 다이어그램과 sequence 다이어그램간의 관계를 나타낸 것이다. Sequence 다이어그램은 시간 진행에 따라 object들이 주고받는 메시지의 순서를 표현한 다이어그램이다[18, pp.246]. 따라서 sequence 다이어그램에 나타나는 object는 object 다이어그램에 나타나야 하고 object의 타입을 정의하는 class 다이어그램에도 나타나야 한다. 따라서 sequence 다이어그램의 object와 class 다이어그램의 class는 연관관계를 가진다. Sequence 다이어그램에서 callaction은 객체의 operation을 invoke한다. 이때 callaction은 아무런 operation을 호출하는 것이 아니라 호출하고자 하는 객체에 존재하는 operation을 호출하여야 한다[18, pp.212]. 따라서 sequence 다이어그램의 callaction에 의해서 발생되는 message와 object 다이어그램의 operation 사이에는 연관관계가 존재하며, class 다이어그램에 정의된 operation과도 연관관계가 존재한다. 두 object 간에 link가 존재한다면 한 객체로부터 다른 한 객체로 메시지를 보낼 수 있다. 따라서 sequence 다이어그램의 두 객체사이의 메시지의 전달에 해당하는 object 다이어그램의 두 객체간의 link가 존재하여야 한다[18, pp.209].

4는 class 다이어그램과 activity 다이어그램간의 관계를 나타낸다. Activity 다이어그램은 statechart 다이어그램의 특별한 경우로서 활동(activity)에서 활동으로 가는 제어흐름을 보여주는 다이어그램이며 state, transition, object 등으로 구성된다[18, pp.260]. Activity 다이어그램에서는 활동의 제어흐름에 따라 관련되는 object를 연관시킨다[18, pp.314]. 따라서 Activity 다이어그램의 object는 object 다이어그램의 object와 연관관계를 가진다고 할 수 있으며, 따라서 class 다이어그램의 class와도 연관관계를 가진다.

5는 class 다이어그램과 collaboration 다이어그램간의 연관관계를 나타낸다. Collaboration 다이어그램은 sequence 다이어그램과 유사한 성격의 다이어그램으로서 sequence 다이어그램은 객체간에 주고받는 메시지의 시간적 순서를 강조할 때 사용하며 collaboration 다이어그램은 객체들의 구조적 조직을 강조할 때 사용하는



CLS : class 다이어그램
OBJ : object 다이어그램
COM : component 다이어그램
DEP : deployment 다이어그램
USC : use case 다이어그램
SEQ : sequence 다이어그램
COL : collaboration 다이어그램
ACT : activity 다이어그램
STC : statechart 다이어그램

———— : structural <-> behavioral
- - - - - - - - - - - - : structural <-> structural or behavioral <-> behavioral

그림 6 다이어그램간의 관계

표 1 Structural 다이어그램과 behavioral 다이어그램간의 일관성 검증요소 I

번호	structural		relationship	behavioral	
	다이어그램	component		component	다이어그램
1	class	operation	dependency	event	statechart
	class	class	realization	statemachine	statechart
2	class	class	realization	use case	use case
3	class	class	correspond	object	sequence
	class	operation	correspond	message(call)	sequence
	class	association	correspond	link	sequence
4	class	class	correspond	object	activity
5	class	class	correspond	object	collaboration
	class	operation	correspond	message(call)	collaboration
	class	association	correspond	link	collaboration

표 2 Structural 다이어그램과 behavioral 다이어그램간의 일관성 검증요소 II

번호	structural		relationship	behavioral	
	다이어그램	component		component	다이어그램
6	object	object	correspond	object	sequence
	object	operation	correspond	message(call)	sequence
	object	link	correspond	link	sequence
7	object	object	correspond	object	activity
8	object	object	correspond	object	collaboration
	object	operation	correspond	message	collaboreation
	object	link	correspond	link	collaboreation
9	object	object	realization	statemachine	statechart
	object	operation	dependency	event	statechart
10	object	object	realization	use case	use case

다이어그램이다. Collaboration 다이어그램은 sequence 다이어그램과 마찬가지로 object, link, message 등을 구성요소로 가지며[18. pp.243], 그림 6의 3과 같은 취지에서 class 다이어그램과 collaboration 다이어그램은 연관관계를 가진다. Structural 다이어그램의 class 다이어그램과 behavioral 다이어그램에 속하는 다이어그램 사이의 관계를 정리하면 표 1과 같다. 1번 관계는 class 다이어그램의 operation이 statechart 다이어그램의 event와 dependency 관계를 가진다는 것을 나타내며 class 다이어그램의 class는 statechart 다이어그램의 statemachine과 realization 관계를 가진다는 것을 표현하고 있다.

그림 6의 6은 object 다이어그램과 sequence 다이어그램간의 관계를 나타낸다. Object 다이어그램은 어느 한 순간에 object와 그들 간의 관계를 묘사하는 다이어그램이다. Object 다이어그램은 class 다이어그램의 하나의 인스턴스와 같으므로[2, pp.80], 그림 6의 3과 같은 취지에서 object 다이어그램의 object, operation, link는 각각 sequence 다이어그램의 object, message, link와 연관관계를 가진다.

7은 object 다이어그램과 activity 다이어그램간의 관계를 나타낸다. 두 다이어그램간의 관계는 그림 6의 4관계와 같은 취지에서 object 다이어그램의 object와 activity 다이어그램의 object 사이에 연관관계를 가진다.

8은 object 다이어그램과 collaboration 다이어그램간의 관계를 나타낸다. 두 다이어그램간의 관계는 그림 6의 5와 같은 취지에서 object 다이어그램의 object, operation, link는 각각 collaboration 다이어그램의 object, message(call), link와 연관관계를 가진다.

9는 object 다이어그램과 statechart 다이어그램간의 관계를 나타낸다. 그림 6의 1과 같은 취지에서 object 다이어그램의 operation, object는 각각 statechart 다이어그램의 event, statemachine와 연관관계를 가진다.

10은 object 다이어그램과 use case 다이어그램간의 관계를 나타낸다. 그림 6의 2와 같은 취지에서 object 다이어그램의 object는 use case 다이어그램의 use case와 연관관계를 가진다.

Structural 다이어그램의 object 다이어그램과 behavioral 다이어그램에 속하는 다이어그램 사이의 관계를 정리하면 표 2와 같다.

표 3 Structural 다이어그램과 behavioral 다이어그램 내의 일관성 검증요소

번호	다이어그램	component	relationship	component	다이어그램
11	class	class	instance	object	object
	class	association	correspond	link	object
12	class	class	dependency	component	component
13	deployment	node	dependency	component	component
14	use case	use case	realization	sequence	sequence
15	sequence	object	correspond	object	collaboration
	sequence	message	correspond	message	collaboration
	sequence	link	correspond	link	collaboraiton

Structural 다이어그램에 속하는 다이어그램과 behavioral 다이어그램에 속하는 다이어그램간의 일관성 관계요소 이외에도 structural 다이어그램에 속하는 다이어그램과 behavioral 다이어그램에 속하는 다이어그램 각각의 내부적인 일관성 관계요소도 존재한다. 정리하면 표 3과 같다.

11은 class 다이어그램과 object 다이어그램간의 관계를 나타내는 것이다. object 다이어그램은 class 다이어그램의 인스턴스와 같다. 따라서 object 다이어그램의 object는 class 다이어그램의 class와 인스턴스관계를 가진다. 또한 link는 association의 하나의 instance이므로[18, pp.209] object 다이어그램의 link와 class 다이어그램의 association 사이에는 연관관계가 존재한다.

12는 class 다이어그램과 component 다이어그램간의 관계를 표현한 것이다. Component 다이어그램은 실행파일, 라이브러리, 테이블, 파일, 문서와 같은 일체의 component와 그들간의 관계를 표현하는 다이어그램이다. Component 속에는 클래스를 포함할 수 있으므로 component 다이어그램의 component와 class 다이어그램의 class는 연관관계를 가진다.

13은 deploymemt 다이어그램과 component 다이어그램간의 관계를 나타낸다. Deployment 다이어그램은 실행 노드와 그 노드에 존재하는 component의 구성을 나타내는 다이어그램으로서 component는 deployment 다이어그램의 node와 연관관계를 가진다.

14는 use case 다이어그램과 sequence 다이어그램간의 관계를 나타내는 것이다. Use case 다이어그램의 Use case는 행동을 명세화하기 위해 사건흐름을 문장으로 설명한다. 사건흐름에는 사건주흐름과 사건예외흐름이 있다. 이때 사건주흐름을 하나의 sequence 다이어그램으로 표현한다[18. pp.224]. 따라서 use case 다이어그램의 use case와 sequence 다이어그램사이에는 연관관계가 존재한다.

15는 sequence 다이어그램과 collaboration 다이어그램간의 관계를 나타낸다. Sequence 다이어그램과 collaboration 다이어그램은 모두 interaction 다이어그램의 일종으로서 객체의 행동을 모델링하기 위한 다이어그램이다. Sequence 다이어그램의 object, message, link는 각각 collaboration 다이어그램의 objet, message, link와 연관관계를 가진다.

그림 6의 16은 activity 다이어그램과 statechart 다이어그램간의 관계를 나타낸다. Activity 다이어그램은 statechart 다이어그램의 특별한 경우로서, activity 다이어그램이 activity에서 activity로 가는 제어흐름을 보여주는 반면 statechart 다이어그램은 state에서 state로 가는 제어흐름을 보여준다[2, pp.129]. Activity 다이어그램에 나타나는 object 전부가 statechart 다이어그램에 나타나는 것은 아니므로 규칙으로 보기 어렵다. 17은 use case 다이어그램과 collaboration 다이어그램간의 관계를 나타낸다. Use case 다이어그램의 use case를 실체화하고자 할 때는 하나의 collaboration을 사용한다[18, pp.219]. 즉 collaboration 다이어그램은 한 use case에서 여러 객체들의 행동을 보고자 할 때 사용된다. 18은 use case 다이어그램과 activity 다이어그램간의 관계를 나타낸다. Activity 다이어그램은 use case와 연관될 수 있다. Use case 내에서 어떤 활동들이 일어나고 그들간의 행동적 의존관계가 어떤 것인지를 파악하기 위해서 사용한다. 19는 use case 다이어그램과 statechart 다이어그램간의 관계를 나타낸다. Statechart 다이어그램은 한 객체가 여러 use case에 걸쳐서 취하는 행동을 설명하기 위해 사용한다. 16, 17, 18, 19는 관계성을 가지기는 하나 규칙을 파악하기 곤란한 관계들이다.

4. 검증규칙의 유도와 정형적 명세

검증규칙은 표 1과 표 2에서 제시된 일관성 검증요소를 기초로 유도한다. 검증규칙은 일관성 관계를 가지는 다이어그램 사이에 관계요소를 적절하게 포함하고 있는가를 판단하는 것이다. 그림 7과 같이 sequence 다이어그램과 object 다이어그램간에는 object와 class를 관계요소로 하여 일관성 관계를 가지며 이때 sequence 다이어그램 내에 존재하는 어떤 object도 object 다이어그램

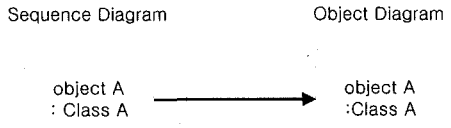


그림 7 다이어그램간 대응요소의 존재여부

내에 존재하여야 한다는 규칙을 얻을 수 있다. 즉, sequence 다이어그램에 나타나는 각 object는 반드시 object 다이어그램에서 해당 object가 존재하여야 한다는 것이다.

따라서 다음과 같은 검증규칙을 유도할 수 있다.

• 규칙11 : Sequence 다이어그램의 각 object에 대응하는 object가 object 다이어그램 내에 존재하여야 한다.

이와 같은 검증규칙을 OCL을 이용하여 정형적으로 표현하면 다음과 같다.

```
Contents1() : Set(Object_b) = self.r_objectb;

Matchparentb() : Set(BehavioralDiagramElement) =
    self.r_behavioraldiagramelement;
Matchparentd() : Set(DiagramElement) =
    self.Matchparentb()->collect(c|c.parent())->flatt
    en->asSet;
Matchchilds() : Set(StructuralDiagramElement) =
    self.Matchparentb()->collect(c|c.childs())->flatt
    en->asSet;
Matchchildob() : Set(ObjectDiagramElement) =
    self.Matchchilds()->collect(c|c.childobj())->flatt
    en->asSet;
Matchchildobj() : Set(Object) =
    self.Matchchildob()->collect(c|c.childobj())->flat
    ten->asSet;

Matchseqd(ob:Object_b): Boolean =
    (self.Matchchildobj()->exists(c|c.object_name =
    ob.object_name));

context SequenceDiagramElement inv rule11:
self.Contents1()->forAll(ob|self.Matchseqd(ob))
```

Contents1()는 sequence 다이어그램으로부터 모든 object를 골라내는 함수이며, Matchseqd() 함수는 골라낸 각 object에 대응하는 object가 object 다이어그램에 존재하는지를 판단하는 함수이다. 그림 6의 6관계에서 sequence 다이어그램의 callaction에 의해서 발생하는 message에 해당하는 operation이 호출되는 객체에 존재하여야 한다. 따라서 다음과 같은 규칙을 유도할 수 있다.

• 규칙12 : Sequence 다이어그램의 각 message(call)에 대응하는 operation이 object 다이어그램의 해당 object내에 존재하여야 한다.

검증규칙12를 OCL로 정형적으로 표현하면 다음과 같다.

```
Contents2() : Set(Message_b) = self.r_messageb;

Matchselobj(me:Message_b) : Set(Object) =
    self.Matchchildobj()->select(c|c.object_name =
    me.endobject_name);
Matchchildope(me:Message_b) : Set(Operation) =
    self.Matchselobj(me)->collect(c|c.childoperation(
    ))->flatten->asSet;
Matchmessage(me:Message_b) : Boolean =
    (self.Matchchildope(me)->exists(c|c.operation_n
    ame = me.operation_name));

context SequenceDiagramElement inv rule12:
self.Contents2()->forAll(me|self.Matchmessage(me))
```

Contents2()는 sequence 다이어그램으로부터 모든 message를 골라내는 함수이며, Matchmessage() 함수는 골라낸 각 message에 대응하는 operation이 object 다이어그램에 존재하는지를 판단하는 함수이다. 그림 6의 6관계에서 sequence 다이어그램에서 객체 사이에 message의 연결관계 즉 link가 존재한다면 object 다이어그램에도 해당 객체간에 link가 연결되어 있어야 한다. 따라서 다음과 같은 규칙을 얻을 수 있다.

• 규칙13 : Sequence 다이어그램의 각 message의 연결관계에 대응하는 link가 object 다이어그램 내에 존재하여야 한다.

검증규칙13을 OCL로 정형적으로 명세하면 다음과 같다.

```
Contents3() : Set(Link_b) = self.r_linkb;

Matchchildlin() : Set(Link) =
    self.Matchchildob()->collect(c|c.childlink())->fla
    tten->asSet;
Matchlink(li:Link_b) : Boolean =
    (self.Matchchildlin() ->
  exists(c|(c.startobject_name = li.startobject_name
    and c.endobject_name = li.endobject_name)
  or (c.endobject_name = li.startobject_name and
    c.startobject_name = li.endobject_name)));

context SequenceDiagramElement inv rule13:
self.Contents3()->forAll(li|self.Matchlink(li))
```

Contents3()는 sequence 다이어그램으로부터 모든 link를 골라내는 함수이며, Matchlink() 함수는 골라낸 각 link에 대응하는 link가 object 다이어그램에 존재하는지를 판단하는 함수이다. 그림 6의 7관계에서 activity 다이어그램과 object 다이어그램의 연관관계를 나타낸다. Activity 다이어그램은 여러 use case에 걸쳐 object가 취하는 행동을 모델링하는데 사용되며, 이때 나타나는 object는 object 다이어그램의 object와 연관관계를 가진다. 따라서 다음과 같은 규칙을 얻을 수 있다.

- 규칙14 : Activity 다이어그램의 각 object에 대응하는 object가 object 다이어그램 내에 존재하여야 한다.

검증규칙14를 OCL로 정형적으로 표현하면 다음과 같다.

```
Contents4() : Set(Object_b) = self.r_objectb;

Matchobject(ob:Object_b) : Boolean =
    (self.Matchchildobj()->exists(c|c.object_name =
    ob.object_name));

context ActivityDiagramElement inv rule14:
self.Contents4()->forAll(ob|self.Matchobject(ob))
```

Contents4()는 activity 다이어그램으로부터 모든 object를 골라내는 함수이며, Matchobject() 함수는 골라낸 각 object에 대응하는 object가 object 다이어그램에 존재하는지를 판단하는 함수이다. 그림 6의 8관계는 object 다이어그램과 collaboration 다이어그램의 관계를 나타낸다. Collaboration 다이어그램은 Interaction 다이어그램의 한 종류로서 객체들의 행동을 모델링하는 다이어그램으로서 특히 객체구조의 조직을 강조한다. Collaboration 다이어그램은 sequence 다이어그램과 같이 object, link, message와 같은 요소를 공통적으로 포함하고 있으며 그 성격이 유사하다. 검증규칙 11, 12, 13과 같은 취지에 의해서 Object 다이어그램이 collaboration 다이어그램과 가지는 일관성 검증규칙은 다음과 같이 유도할 수 있다.

- 규칙15 : Collaboration 다이어그램의 각 object에 대응하는 object가 object 다이어그램 내에 존재하여야 한다.
- 규칙16 : Collaboration 다이어그램의 각 message (call)에 대응하는 operation이 object 다이어그램의 해당 object내에 존재하여야한다.
- 규칙17 : Collaboration 다이어그램의 각 message의 연결관계에 대응하는 link가 object 다이어그램 내에 존재하여야 한다.

검증규칙 15, 16, 17의 각각의 정형적 명세도 검증규칙 11, 12, 13과 같이 유도할 수 있다. 그림 6의 9관계는 object 다이어그램과 statechart 다이어그램간의 관계를 나타낸다. Statechart 다이어그램은 object의 행동을 모델링하는 것으로 object의 statemachine을 보여주며, state에서 state로 가는 제어흐름을 강조한다[18, pp.333]. Statemachine은 객체가 생명주기 동안에 사건이 발생하면 그에 대응해서 옮겨 다니는 상태의 순서를 명세화 한 행동이고, 사건에 대한 응답도 함께 명세화한 것이다. 따라서 statechart 다이어그램의 statemachine과 object 다이어그램의 object 사이에는 연관관계를 가지며 다음과 같은 규칙을 유도할 수 있다.

- 규칙18 : Statechart 다이어그램의 statemachine에 대응하는 object가 object 다이어그램 내에 존재하여야 한다.

검증규칙18을 OCL로 정형적으로 명세하면 다음과 같다.

```
Contents8() : Set(Statemachine_b) =
    self.r_statemachineb;

Matchstatemachine(st:Statemachine_b) : Boolean =
    (self.Matchchildobj()->exists(c|c.object_name =
    st.object_name));

context StatechartDiagramElement inv rule18:
self.Contents8()->forAll(st|self.Matchstatemachine(st))
```

Contents8()은 statechart 다이어그램으로부터 모든 statemachine을 골라내는 함수이며, Matchstatemachine()는 골라낸 각 statemachine에 대응하는 object가 object 다이어그램에 존재하는 지를 판단하는 함수이다. 그림 6의 9는 statechart 다이어그램과 object 다이어그램간의 관계를 나타낸다. Statemachine내의 하나의 Event(callevent)는 연관되는 object의 operation으로 모델링할 수 있으므로 다음과 같은 검증규칙을 유도할 수 있다.

- 규칙19 : Statechart 다이어그램의 statemachine내의 event에 대응하는 operation이 object 다이어그램의 해당 object내에 존재하여야 한다.

검증규칙19를 정형적으로 명세하면 다음과 같다.

Contents9()는 statechart 다이어그램으로부터 모든 event를 골라내는 함수이며, Matchevent()는 골라낸 각 event에 대응하는 operation이 object 다이어그램의 대응 object내에 존재하는 지를 판단하는 함수이다. 그림 6의 10은 use case 다이어그램과 object 다이어그램간의 관계를 나타낸다. Use case 다이어그램은 class의 행동을 모델링하는 것이므로 이에 해당하는 class가

class 다이어그램 내에 존재하여야 한다. 따라서 class의 인스턴스인 object와도 연관관계를 가진다. Use case 다이어그램과 object 다이어그램간에는 다음과 같은 규칙을 유도할 수 있다.

```
Contents9() : Set(Event_b) =
    self.Contents8()->collect(c|c.childevent())->flatt
    en->asSet;

Matchselobj(ev:Event_b) : Set(Object) =
    ev.parentstm()-> collect(s|
    self.Matchchildobj()->select(o|o.object_name =
s.object_name))->flatten->asSet;
Matchchildope(ev:Event_b) : Set(Operation) =
    self.Matchselobj(ev)->collect(c|c.childoperation(
    ))->flatten->asSet;
Matchevent(ev:Event_b) : Boolean =
    (self.Matchchildope(ev)->exists(c|c.operation_na
    me = ev.event_name));

context StatechartDiagramElement inv rule19:
self.Contents9()->forAll(ev|self.Matchevent(ev))
```

• 규칙20 : Use case 다이어그램의 use case에 대응하는 object가 object 다이어그램 내에 존재하여야 한다.

검증규칙20을 OCL로 정형적으로 명세하면 다음과 같다.

```
Contents10() : Set(Usecase_b) = self.r_usecaseb2;

Matchusecase(us:Usecase_b) : Boolean =
    (self.Matchchildobj()->exists(c|c.object_name =
    us.object_name));

context UsecaseDiagramElement inv rule20:
self.Contents10()->forAll(us|self.Matchusecase(us))
```

Contents10()은 use case 다이어그램으로부터 모든 use case를 골라내는 함수이며, Matchusecase()는 골라낸 각 use case에 대응하는 object가 object 다이어그램 내에 존재하는지를 판단하는 함수이다.

Object 다이어그램과 인스턴스 관계를 가지고 있는 Class 다이어그램에 대한 검증규칙도 앞서 유도한 object 다이어그램의 검증규칙과 유사하게 유도할 수 있다. 따라서 그림 6의 1, 2, 3, 4, 5 관계는 각각 9, 10, 6, 7, 8 관계와의 유사성에 의해 검증규칙을 유도할 수 있다.

5. USE를 이용한 규칙의 검증

OCL로 명세한 규칙은 OCL을 처리해주는 관련도구 [20,21,22,23,24]를 이용하여 검증할 수 있다. OCL 관련 도구 중 가장 뛰어난 기능을 가진 도구는 USE[22]로 판단된다. 따라서 본 연구에서는 USE를 이용하여 검증규칙의 유용함을 밝힌다. USE는 검증하고자 하는 다이어그램을 USE 명세 형태로 변형하고 다이어그램에 적용하는 제한자와 검증규칙을 OCL로 표현하여 검증 다이어그램에 적용한다. USE를 이용한 검증의 전체과정은 그림 8과 같이 표현할 수 있다. USE의 이용방법은 크게 validation과 meta-validation으로 구분할 수 있다. Validation은 사용자가 작성한 다이어그램에 constraint를 적용하여 다이어그램의 정확함을 검증하는 것이고, 즉 사용자의 'data'를 검증하는 것이고, mata-validation은 사용자가 작성한 다이어그램에 well-formedness rules를 적용하여 '다이어그램 자체'의 정확함을 검증하는 것이다. Well-formedness rules는 UML 다이어그램의 정확함을 표현하는 정확성 규칙으로서 UML specification에서는 well-formedness rules를 OCL로 표현하고 있다. 본 연구에서는 USE를 meta-validation에 이용하여 앞에서 유도한 검증규칙의 효과를 검증한다.

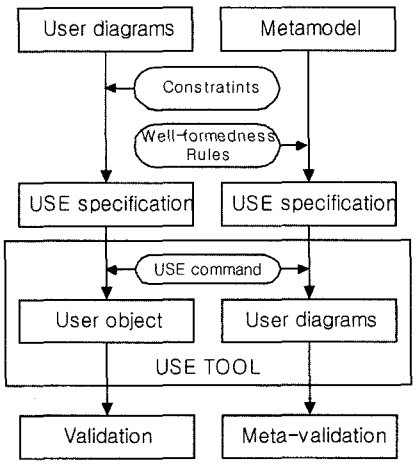


그림 8 USE의 이용도

Meta-validation은 목적에 맞는 metamodel을 구성하여 USE 명세로 표현하여 검증시스템의 입력으로 사용한다. 이때 적용하고자 하는 검증규칙을 well-formedness rule과 같이 표현하여 명세에 삽입한다. 입력된 명세를 기초로 insert, create, set 과 같은 USE 명령어를 이용하여 메타모델의 인스턴스 즉 사용자 다이어그램을 생성한다. 생성된 인스턴스 다이어그램에 well-formedness rule과 같은 검증규칙이 적용되어 다이어그램의 건전성을 검증하게 된다. 본 연구에서는 이러한 검증과정의 초기작업으로서 3장에서 structural 메타-메타모델

과 behavioral 메타-메타모델을 유도하였고 이를 검증의 초기 메타모델로 삼는다. 유도한 메타-메타모델은 USE의 입력으로 사용하기 위해 하나의 통합된 명세로 구성하고 4장에서 유도한 검증규칙을 well- formedness rule와 같은 의미로 USE 명세 속에 포함되도록 한다.

검증규칙의 유용성을 검증하기 위해 다음과 같은 예를 도입한다. 예는 웹기반 학습평가시스템으로 교수자와 학습자가 상호작용하면서 학습할 수 있는 공간을 제공해주는 것으로 교수자가 문제를 출제하면 해당학생들은 문제를 풀이하고 그 결과를 바탕으로 각종 통계지표를 통해 학생과 교수자에게 앞으로의 학습방향에 도움을 주는 시스템이다. 이 시스템의 설계시 작성할 수 있는 object 다이어그램의 예가 그림 9에 나타나 있다.

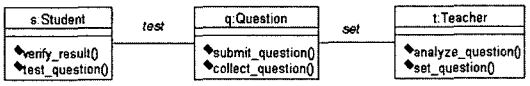


그림 9 예제 시스템의 object 다이어그램

또 그림 9에 대응하는 sequence 다이어그램은 그림 10처럼 나타낼 수 있다.

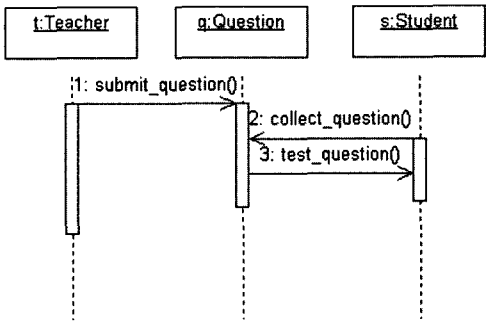


그림 10 예제 시스템의 sequence 다이어그램

USE의 입력명세는 다이어그램의 구성요소를 명세하는 부분과 제한자를 명세하는 부분으로 구성되어 있다. 다이어그램의 구성요소를 모두 하나의 클래스 형태로 표현하고 필요한 attribute와 operation을 표현하게 되어 있다. 다이어그램에 적용하고자하는 검증규칙과 같은 제한자는 OCL 형태로 표현하여 명세의 하단에 삽입한다. USE의 명세구조는 다음과 같다.

```
model model_name
class class_name
attributes
 attributes_name
operation
 operation_name
end
constraints
context verified_element inv rule_name:
verification_condition
```

그림 9와 그림 10 다이어그램의 건전성을 검증하기 위해서는 USE의 meta-validation의 과정에 따라 metamodel의 하나의 인스턴스로 생성하여 검증한다. 즉 structural 메타-메타모델과 behavioral 메타-메타모델의 통합된 메타모델의 하나의 인스턴스가 된다. 인스턴스를 생성하는 명령은 create, insert, set과 같은 명령어가 있다. Create는 인스턴스 모델의 구성요소를 생성하는 명령어이고, insert는 구성요소 사이의 관계를 설정하여주는 명령어이다. Set 명령어는 구성요소의 속성에 값을 설정하여주는 명령어이다. 명령어의 형식은 다음과 같다.

```
!create  component_name:component
!insert  (component_name, component_name) lass_
         name into association_name
!set acomponent_name.attributes_name = 'name'
```

생성된 인스턴스 모델은 그림 11과 같이 표현된다. 인스턴스 모델은 메타모델의 하나의 인스턴스이므로 원본 다이어그램과 달리 object 다이어그램 형태로 표현된다. DiagramElement 객체가 다이어그램의 중심역할을 하며 왼쪽에는 behavioral 다이어그램의 종류로서 그림 10의 sequence 다이어그램이 표현되어 있고, 오른쪽에는 structural 다이어그램의 한 종류로서 그림 9의 object 다이어그램이 표현되어 있다. 객체와 객체사이에는 연관관계가 표시되며 내부적으로 각각의 역할명을 가지게 된다. 역할명은 OCL로 검증규칙을 표현할 때 중요한 역할을 한다.

생성된 인스턴스 모델에 정의한 검증규칙이 적용된다. 그림 9와 그림 10 다이어그램에 일관성 검증규칙을 적용한 결과는 그림 12와 같다. 다이어그램에 검증규칙11에서 검증규칙20까지의 규칙을 적용한 결과 모두 true로 나타났다. 따라서 그림 9의 object 다이어그램과 그림 10의 sequence 다이어그램은 일관성 검증규칙에 만족한다는 결과를 얻을 수 있다.

검증규칙의 유용성을 설명하기 위하여 그림 13과 같은 오류가 있는 sequence 다이어그램을 도입한다. 그림 13은 표시된 부분과 같이 메시지의 표현에 오류가 있다. 검증규칙 12는 sequence 다이어그램의 각 message에 대응하는 operation이 object 다이어그램의 해당 object 내에 존재하여야 한다는 것이다. 즉 s:Student 객체가

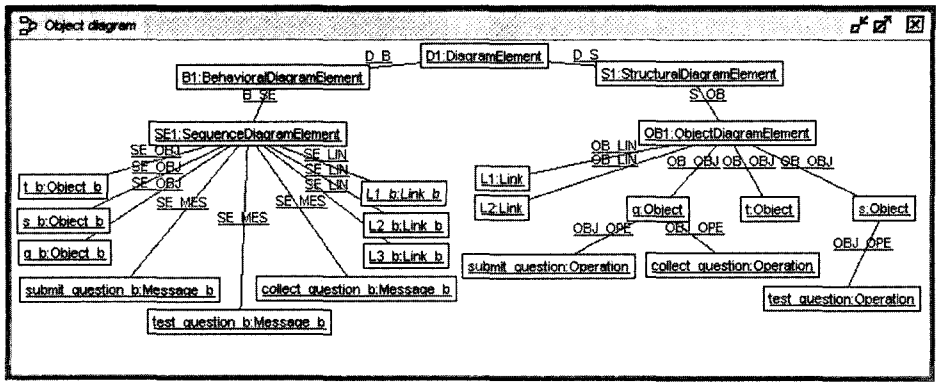


그림 11 생성된 인스턴스 모델

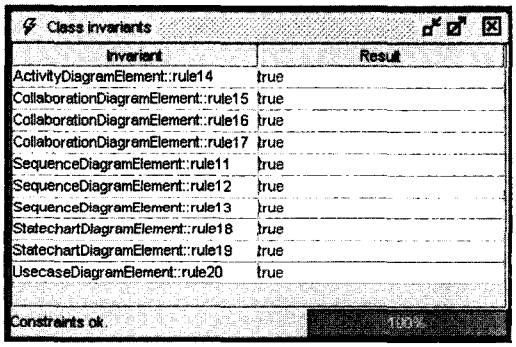


그림 12 검증규칙의 적용결과

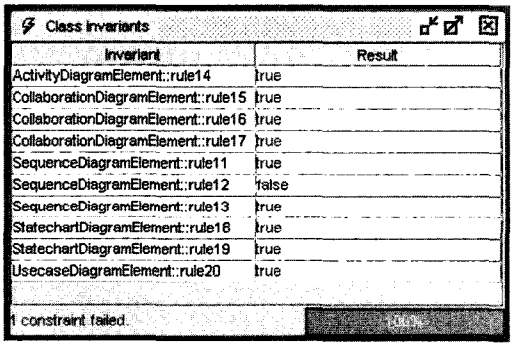


그림 14 오류가 있는 다이어그램의 검증규칙 적용결과

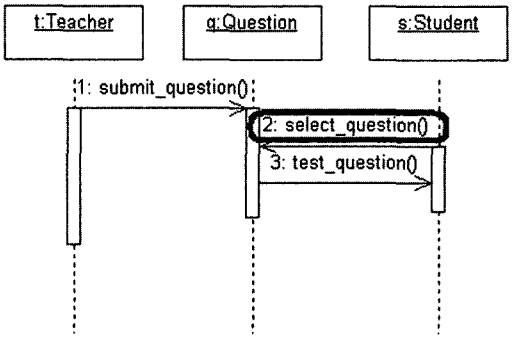


그림 13 오류가 있는 sequence 다이어그램의 예

호출한 select_question() 메시지는 q:Question 객체에 operation으로 정의되어 있어야 한다는 것이다.

따라서 그림 9에는 s:Student 객체에서 호출하는 select_question 메시지를 받는 q:Question 객체에는 select_question과 같은 operation이 정의되어 있지 않으므로 이를 검증하는 검증규칙12는 만족하지 못한 결과가 나와야 한다. 그림 14는 그림 9와 그림 13의 다이어그램에 관하여 검증규칙을 적용했을 경우 나타나는 결과이다.

6. 결론 및 향후 연구방향

본 연구에서는 UML 표준에 의해 작성된 behavioral 다이어그램에 속하는 다이어그램과 structural 다이어그램에 속하는 다이어그램간의 일관성을 검증하는 방법으로서 structural 다이어그램의 메타-메타모델과 behavioral 다이어그램의 메타-메타모델, 그리고 그로부터 유도된 OCL로 표현한 일관성 검증규칙을 이용하여 검증하는 메타검증 방법을 제안하였다. 메타검증을 위해 우선 behavioral 다이어그램과 structural 다이어그램의 메타-메타모델을 유도하였고, 메타-메타모델을 통해 structural 다이어그램과 behavioral 다이어그램간의 관계성을 분석하고 일관성 검증요소를 파악하였다. 그리고 일관성 검증요소를 중심으로 일관성 검증규칙을 유도하였다. 유도된 검증규칙은 좀더 명확한 이해와 검증의 자동화를 위하여 특징적으로 OCL을 사용하여 정형적으로 명세하였고, 마지막으로 정형적으로 명세한 규칙을 OCL 관련도구인 USE를 이용하여 그 유용성을 검증하였다.

USE는 OCL 관련도구로서 가장 뛰어난 기능을 가지는 도구이긴[13] 하지만 그 도구를 이용하여 검증하는

과정에서 이용상의 불편한 점을 발견하였다. 즉 사용자가 작성한 다이어그램의 건전성을 검증하기 위하여 불편한 변환작업을 거쳐야 한다는 것이다. 메타모델과 검증규칙은 한번만 생성하여 검증시스템에 입력해둔다면 불편함이 없으나 사용자가 작성한 다이어그램을 검증하기 위해서는 USE 명령어를 통하여 인스턴스를 발생시켜야하는 여분의 작업이 필요하게 된다. 이러한 문제는 UML 모델링을 지원하는 도구와 모델의 검증을 지원하는 도구가 분리되어 있는 현실에 기인한다. 따라서 향후 연구과제로 이러한 문제점을 해결하기 위해 사용자가 작성한 다이어그램을 여분의 변환과정 없이 검증하기 위한 연구가 필요하다. 또한 제안한 메타-메타모델과 검증규칙을 보다 상세하고 정확하게 표현하여 다이어그램 간의 일관성을 보다 완벽하게 검증할 수 있도록 하는 것이 필요하다.

참 고 문 헌

[1] OMG, OMG Unified Modeling Language Specification Version 1.4, Object Management Group Inc. 2001.

[2] M. Fowler and K. Scott, UML Distilled, Addison-Wesley, 1999.

[3] OMG, Object Constraint Language Specification, OMG Unified Modeling Language Specification Version 1.4, 2001.

[4] J. B. Wormer and A. G. Kleppe, The Object Constraint Language, Addison-Wesley, 1999.

[5] P. Selonen, K. Koskimies and M. Sakkinen, "How to Make Apples from Oraqnges in UML," Proc. of the 34 Hewaii International Conference on System Sciences, 2001.

[6] A. Egyed, "Integrating Architectural Views in UML," Techical Report USCCSE-99-514, 1999.

[7] 정기원, 조용선, 권성구, "객체지향 설계방법에서 오류 검출과 일관성 점검기법 연구", 한국정보처리학회논문지 제6권 제8호. 1999.

[8] 김도형, 정기원, "객체지향 분석과정에서 오류와 일관성 점검 방법", 정보과학회논문지(B) 제26권 제3호. 1999.

[9] A. Tsiolakis and H. Ehrig, "Consistency analysis of UML class and sequence Diagrams using attributed graph grammars," Proc. of Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems, Mar. 2000.

[10] A. Tsiolakis, "Consistency Analysis of UML Class and Sequence Diagrams based on Attributed Typed Graphs and their Transformation," Technical Report 2000/3, Technical University of Berlin, Mar. 2000.

[11] T. Sunetnanta and A. Finkelsteing. "Automated Consistency Checking for Multiperspective Software Specifications," Proc. of the 23rd Internation Conference on Software Engineering, ICSE2001, May 2001.

[12] 조진형, 배두환, "UML 객체지향 분석 모델의 완전성 및 일관성 진단을 위한 시나리오 검증기법", 정보과학회논문지 제28권 제3호, 2001.

[13] M. Richters, "A Precise Approach to Validating UML Models and OCL Constraints," PhD thesis, University Bremen, Logos Verlag, Berlin, BISS Monographs, No.14, 2002.

[14] B. Hnatkowska, Z. Huzar and J. Magott, "Consistency Checking in UML Models," Proc. of 4th International conference on Information Systems Modeling ISM'01, 2001.

[15] M. Richters and M. Gogolla, "Validating UML models and OCL Constraints," Proc. of UML2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, vol. 1939 of LNCS, pp.265-277, Oct. 2000.

[16] P. Bottoni, M. Koch, F. Parisi-Presicce and G. Taentzer, "Consistency Checking and Visualization of OCL Constraints," Proc. of UML2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, vol. 1939 of LNCS, pp.294-308, Oct. 2000.

[17] 김진수, 강권학, 이경환, "제약언어를 이용한 객체 모델 검증시스템", 한국정보처리학회논문지 제3권 제6호, 1996.

[18] G. Booch, J. Rumbaugh and I. Jacobson, The Unified Modeling Language User Guide, Addison-Weslay, 1999.

[19] OMG, UML Semantics, OMG Unified Modeling Language Specification Version 1.4, 2001.

[20] M. Wittmann. "Ein Interpreter fur OCL," Diplomarbeit, Ludwig-Maximilians-Universitat Munchen, 2000.

[21] BoldSoft, 'Modelrun,' 2000. Internet: http://www.boldsoft.com/products/modelrun/index.html

[22] M. Richters, 'The USE tool: A UML-based specification environment,' 2001. Internet: http://www.db.informatik.uni-bremen.de/projects/USE/

[23] H. Hussmann, B. Demuth, and F. Finger, "Modular architecture for a toolset supporting OCL," Proc. of UML2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, vol. 1939 of LNCS, pp.278- 293 Oct. 2000.

[24] IBM, 'OCL Parser,' ver.o.3, Internet: http://www-3.ibm.com/software/ad/library/standards/ocl.html

하 일 규

1992년 영남대학교 전산공학과(학사) 2001년 영남대학교 정보처리교육전공(석사). 2003년 8월 영남대학교 컴퓨터공학과(박사). 1992년~1995년 증권감독원 전산업무실. 관심분야는 UML, OCL, 정형명세기법, 원격교육

강 병 욱

1970년 영남대학교 전기공학과(학사) 1977년 영남대학교 전자공학과(석사) 1994년 경북대학교 전자공학과(박사) 1973년~1976년 영남대학교 전자계산연구소(SE). 1977년~1978년 영남전문대학 전자과 전임강사. 1979년~현재 영남대학교 공과대학 전자정보공학부 교수. 관심분야는 UML, OCL, 정형명세기법, 소프트웨어공학, 원격교육