Journal of the Korea Academia-Industrial cooperation Society
Vol. 15, No. 1 pp. 449-459, 2014

http://dx.doi.org/10.5762/KAIS.2014.15.1.449

Z/Object-Z 사용한 2+1 View 통합 메타모델의 정형 명세와 명확성 검사

송치양[1*]
[1]경북대학교 컴퓨터정보학부

A Formal Specification and Accuracy Checking of 2+1 View Integrated Metamodel Using Z and Object-Z

Chee-Yang Song[1*]
[1]**Division of Computer Information, Kyungpook National University**

요 약 시각적인 클래스 모델로 기 제시된 2+1 View 통합 메타모델은 비정형적인 명세에 기인하여 명확하게 모델의 구문을 표현하지 못하고 있으며, 또한 그 모델의 정확성을 보장할 수 없다. 본 논문은 Z와 Object-Z를 사용해서 2+1 View 통합 메타모델의 구문적 의미를 정형적으로 명세하고, Z/Eves 툴을 통해 메타모델의 정확성을 검사하는 것이다. 정형 명세는 클래스 모델과 Z/Object-Z간의 변환규칙을 적용해서 2+1 View 통합 메타모델의 구문과 정적 시멘틱에 대해 Z와 Object-Z 스키마로 각각 표현한다. 메타모델의 검사는 Z 스키마 명세에 대해 Z/Eves 도구를 사용하여 구문, 타입 검사 그리고 도메인 검사를 수행하여 메타모델의 정확성을 입증한다. 이로서, 2+1 View 통합 메타모델 메타모델의 Z/Object-Z 변환을 통해 구조물의 구문적 의미를 명확하게 표현할 수 있으며, 또한 그 메타모델의 정확성을 검사할 수 있다.

Abstract The proposed 2+1 view integrated metamodel defined formerly with a graphical class model can not be guaranteed the syntactic clarity and accuracy precisely for the metamodel due to the informal specification. This paper specifies the syntactic semantics formally for the 2+1 view integrated metamodel using Z and Object-Z and checks the accuracy of the metamodel with Z/Eves tool. The formal specification is expressed in Z and Object-Z schema separately for syntax and statics semantics of the 2+1 view integrated metamodel, which applying the converting rule between class model and Z/Object-Z. The accuracy of the Z specification for the metamodel is verified using Z/Eves tool, which can check the syntax, type, and domain of the Z specification. The transformation specification and checking of the 2+1 view integrated metamodel can help establish more accurate the syntactic semantics of its construct and check the accuracy of the metamodel.

Key Words : 2+1 View Integrated Metamodel, Z, Object-Z, Formal specification, Model checking, Z/Eves

1. 서론

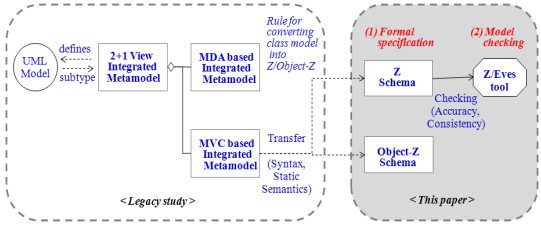
소프트웨어의 개발 과정에서 생성되는 산출물들에 대한 재사용성을 향상시키기 위하여 모델 중심의 MDA (Model Driven Architecture)[1]에 초점을 두어 독립성과 모듈성이 강한 컴포넌트 구축을 통한 재사용에 대한 연구가 가속화되어 가고 있다[2]. 비즈니스 가치가 중요시되는 서비스 지향 시스템 개발에서 가변적 비즈니스 프로세스에 따라 고정적 컴포넌트들을 가지고 어플리케이션을 구축하는 추세이다. 컴포넌트 기반 개발 패러다임에 의한 소프트웨어 모델링 과정에서 모델의 생성을 극대화하기 위한 방안으로, [3]에서 UML(Unified Modeling

본 논문은 2013학년도 경북대학교 학술연구비에 의해 연구되었음.
*Corresponding Author : Chee-Yang Song(Kyungpook Univ.)
Tel: +82-54-530-1453 email: cysong@knu.ac.kr
Received September 3, 2013 Revised September 25, 2013 Accepted January 9, 2014

Language)[4] 모델을 대상으로 개발 프로세스(모델링 단계: MDA) 및 MVC(Model View Controller) 패턴에 의한 2+1 View(MDA/MVC+Architecture) 메타모델을 정의하여 MDA(Model Driven Architecture)(PIM/PSM) 방식에 의한 컴포넌트 모델링 접근을 제시한다. 여기서, 2+1 View 통합 메타모델은 MDA(process view)와 MVC에 의한 각각 두 개로 구성, 제시되었다.

MDA 기반 통합 메타모델은 개발 프로세스 관점에서 요구분석, 기본 설계(PIM: Platform Independent Model), 상세 설계(PSM: Platform Specific Model) 그리고 구현의 개발 단계로 구축하여 컴포넌트를 계층적으로 모델링할 수 있도록 정의되었다. MVC 기반 통합 메타모델은 어플리케이션의 아키텍처 관점에서 MVC 패턴에 의해 시스템을 GUI(Graphic User Interface Model: View)/ 비즈니스 로직(Controller)/ DB(Data Base: Model) 모델로 분리하여 모델링을 수행토록 제시했다. 이 메타모델들은 개발 단계 및 MVC 방식에 따라 수준별 분리된 모델링 요소들을 가지고 이들을 구조화시켜서 UML의 시각적인 클래스 모델로 표현하고 있다. 따라서, 비정형적인 클래스 모델(class model)로 정의된 2+1 View 통합 컴포넌트 메타모델은 이 구조물(construct)이 가진 구문(syntax)과 의미(semantic)를 명확하게 표현하고, 제시된 메타모델의 정확성을 검증이 필요하다.

본 논문은 2+1 View 통합 메타모델의 구문적 및 정적 의미를 명확히 표현하고, 메타모델의 정확성을 입증하기 위해, 정형적 언어인 Z[5-6]와 Object-Z[7]를 사용하여 MVC 기반 통합 메타모델을 명세하고, Z-Eves Tool[8-9]을 사용하여 정확성을 검사한다. 정형 명세는 클래스 모델과 Z간의 변환규칙[[10-12] 그리고 클래스 모델과 Object-Z간의 변환기법[13-17]을 각각 적용하여, MVC 기반 통합 메타모델의 구문과 정적 의미(static semantics)에 대해 Z 및 Object-Z 스키마(schema)로 각각 변환, 명세한다. 즉, Z 및 Object-Z를 사용해서 MVC 기반 통합 메타모델의 구문에 대해 기본 타입을 선언하고, 각 클래스를 대상으로 요소 기반으로 스키마를 정의하고, 또한 클래스간 상호 관련성을 대해 관계 기반으로 스키마를 명세한다. 이 메타모델의 정적 의미(적형성: well-formedness Properties 등)는 스키마내 술어 부문에 술어논리(predicate logic)로 명세한다. 메타모델의 정확성 검사는 Z/Eves 도구를 사용해서 기 명세된 Z 스키마를 입력하고, 구문 및 타입의 검사(type checking) 그리고 도메인 검사를 수행하여 메타모델의 구문적 불일치를 검사한다. Fig. 1은 본 연구의 접근 모델로서 어떻게 2+1 View 통합 메타모델이 정형적으로 명세되고 검증되는지의 과정을 보여준다. 회색 박스 영역이 본 논문에서 수행하고자하는 부분이다. Object-Z 외에 Z를 추가한 이유는 Z 명세에 기반하여 Object-Z로의 변환 명세 및 상호간 비교분석이 용이하고, 검증위한 지원도구 사용에 있다.



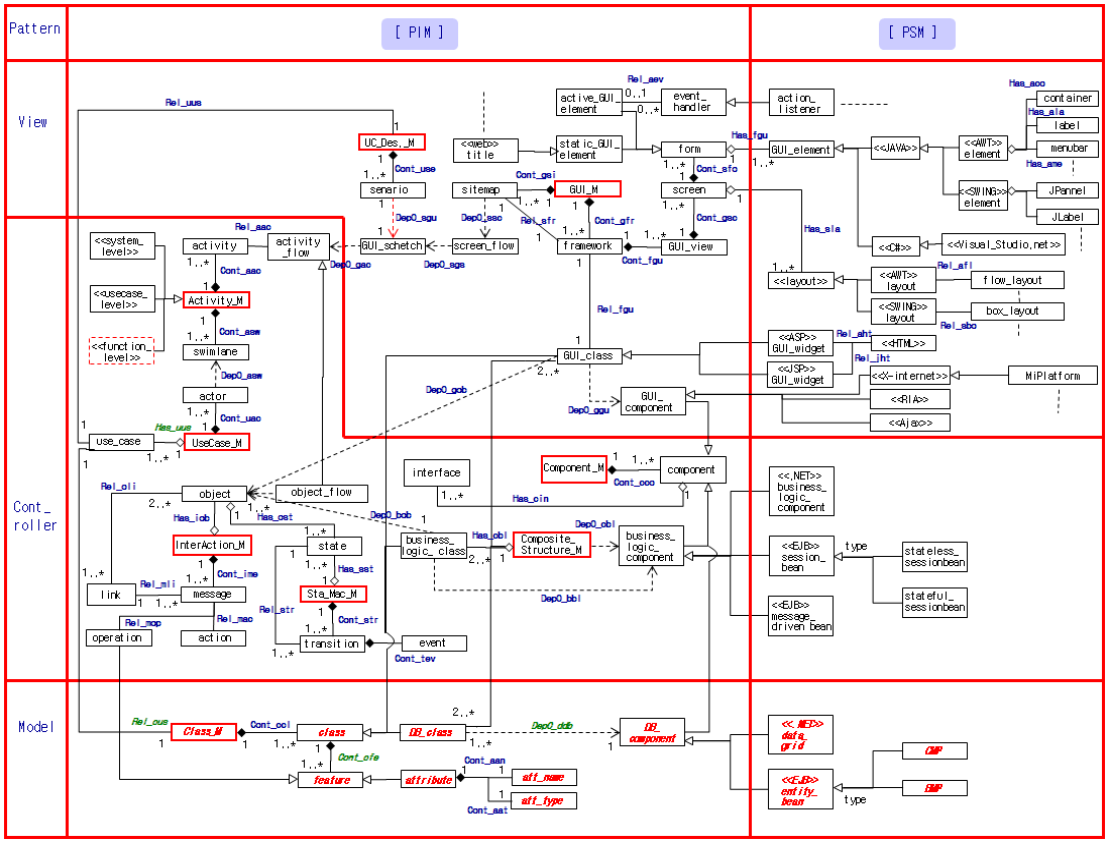
[Fig. 1] Checking procedure for the accuracy of 2+1 View Integrated Metamodel

2. 관련 연구

2.1 2+1 View 통합 메타모델

본 논문에서 다루고자 하는 모델 명세 및 검사의 대상은 클래스 모델로 그래픽하게 표현된 2+1 view 통합 메타모델[3]이다. [3]에서 MDA 기반의 PIM/PSM 모델의 계층적 접근과 MVC 디자인 패턴을 적용된 2+1 view 통합 메타모델을 제시한다. 2+1 view meta-architecture 접근 모델은 한 개의 meta-architecture view와 두개의 metamodel modeling view로 되어있다. meta-architecture view는 메타 아키텍처 차원의 계층적 인프라 위상(topology)을 표현한다. MDA 기반 통합 메타모델은 4개의 개발 단계별로 계층화된 UML 모델들을 대상으로 PIM/PSM의 MDA 기준에 부합하도록, 각 모델이 가진 모델링 요소들을 가지고, process view 기반 계층적 통합 메타모델을 정의했다. MVC 기반 통합 메타모델은 Fig. 2와 같이, MVC 패턴에 의해 구성된 UML과 GUI 모델들을 대상으로 그 모델링 요소들을 가지고 PIM과 PSM 방식으로 분할하여 MVC view 기반 계층적 통합 메타모델을 제시했다.

본 논문은 두 개 통합 메타모델 중에 이해성을 돕기 위해 클래스(모델링 요소)의 수가 MDA 통합 메타모델보다 적은 MVC 기반 통합 메타모델을 대상으로 Z와 Object-Z로 명세하고, 모델을 검사한다. 메타모델 구조물의 정형적 명세를 위해 Fig. 2에 클래스들간의 관계들에 관계명을 추가했다. 이때, 두 클래스간에 상속 관계인 경우에는 Z에서 상속을 표현할 수 있으므로 관계명을 부착하지 않았다. 또한 누락된 클래스들간의 다중성(Multiplicity, Cardinality)에 대해 보완했다.



[Fig. 2] MVC-based Integrated Metamodel [3]

2.2 Z와 Object-Z

Z[5,6]는 범용적 ISO(International Standard Organization) 표준 언어로서 증명(proof)을 위한 좋은 기능을 제공하는데, 구조적이고 수학적인 이론에 기반을 둔 언어 즉, 집합론에 기반을 둔 정형적 명세 언어이다. Z의 구조물[18]은 타입과 스키마로 구성되는데, 스키마는 선언부(signature 혹은 declaration part)와 술어부(predicate part)에 의해 그 구성요소와 제약사항을 명세한다. Z는 풍부한 모델 검사 기능을 지원하나, 소규모 응용 시스템의 정형명세에 적합하며, 객체 개념을 표현하지 못한다. 그 이유는 Z가 클래스에 대해 속성(state)과 기능(operation)을 각각 분리된 스키마로 명세하고, 객체간 복합(composite) 관계를 표현하지 못하며, 또한 클래스간 상호통신을 위한 인터페이스(interface)를 표현하지 못함에 기인한다. Z를 확장한 언어들로 Object-Z, ZEST 및 MOOZ 등이 있다.

Object-Z[7]는 중.대규모 응용 시스템의 정형 명세에 적합하도록 Z를 객체지향 개념으로 확장한 객체 지향적인 정형 명세 언어이다. 즉, Object-Z는 Z를 'class' construct로 확장하여 UML처럼 속성(attribute)과 오퍼레이션(operation)을 캡슐화하여 표현하고, 객체간의 상속이 가능하다. 또한, Object-Z는 semantics를 잘 표현함으로서 UML class diagram보다 좀더 우수한 표현력을 갖는다. 반면, Z에 비해 사용 환경의 제약에 따른 다양한 모델 검사기를 지원하지 못하는 제약을 안고 있다.

2.3 클래스 모델의 Z/Object-Z 명세 및 검증

MVC 기반 통합 메타모델은 클래스 모델의 표기법에 의해 표현되어 있으므로 클래스 모델과 Z/Object-Z간의 변환 명세 및 검증에 관계된 연구들을 분석한다. 클래스 모델에 대한 Z로의 변환은 모델링 메소드(클래스 다이어그램)와 정형 언어(Z/Object-Z)간의 통합을 의미한다. "Z 혹은 Object-Z를 사용하여 클래스 모델을 명세한다." 의미는 절차적 프로그램 언어처럼 일이 어떻게 수행되는지를 표현하는 것이 아니라, 모델을 구성하는 요소들을 위한 클래스와 모델에 가해지는 불변의 제약조건들을 스키마 형태로 명세하는 것이다[15]. 즉, Z의 경우, 클래스

에 대해 속성과 오퍼레이션을 독립된 스키마로 표현되며, Object-Z 경우, 스키마가 클래스와 동일한 구조로 표현할 수 있으므로 두 상이한 구조물들간 상호 정보의 손실없이 직접적 변환이 가능하다.

2.3.1 Z 기반 클래스 모델의 명세 및 검증

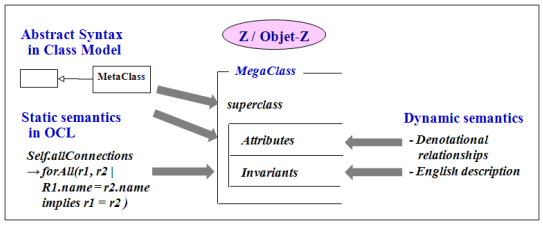
클래스 모델의 정형화된 명세 및 검사(checking)을 위한 기존 연구에서, UML 클래스 모델의 정형적 명세 언어인 Z로의 변환, 명세 및 검증에 대해서는 체계화되어 있다[10-12]. 이들의 두 구조물간 변환 규칙은 클래스 다이어그램이 갖고 있는 요소(Element), 관계(Relationship), 다중성(Multiplicity) 및 제약사항(Constraint)을 고려하여 Z 언어로의 변환을 정하고 있다. 또한, [12, 15]에서 클래스 모델내 포함된 각 클래스들 및 그들간의 관계에 대한 Z 스키마로의 명세 순서와 방법을 잘 정의하고 있다. 이에 기반해서, Z 명세의 방법은 먼저 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, 이어서, 클래스들간의 관계명에 기반하여 각 관계별로(Relationship Name)별로 스키마를 정의한다. 한편, 클래스 모델내 존재하는 정적 의미 혹은 속성들(liveness, soundness 등)을 위한 불변적(Invariant) 제약사항(Constraint)들은 요소를 정의한 스키마의 술어부에 명세한다. 즉, 통합 메타모델의 추상적 구문(abstract syntax: element, relationship, property, multiplicity/ class model 표현)과 정적 의미(static semantics: Well-formedness property - invariant, constraint / OCL(Object Constraint Language) 명세)가 Z 스키마(Basic/Type declaration, Schema for element, Schema for relationship)로 변환, 명세된다[18].

한편, Z 스키마 명세의 검증을 위한 Z를 지원하는 모델 검사기들로서는 대표적인 Z/Eves, 그 외 ZTC 및 ZEST 등이 있는데, 이들은 모델의 Z 스키마 명세에 대해 구문, 타입 검사 및 도메인 검사를 제공한다. [18]에서 메타모델이 아닌 응용시스템의 모델 차원에서, 휘처 모델을 대상으로 Z로의 변환 규칙을 정립하고, 이 규칙에 의거 Z로 명세하고, Z/Eves를 통한 모델의 검사를 보였다. [19,20]에서는 그들 논문에서 제시한 메타모델에 대해 Z로 변환 명세하고 Z/Eves 사용의 모델 검증을 제시했다. 본 연구에서는 [11-12,19]에서 제시한 변환 규칙들을 적용해서 MVC 기반 통합 메타모델의 Z로 명세 및 검증을 수행한다.

2.3.2 Object-Z 기반 클래스모델의 명세 및 검증

클래스 모델의 Object-Z로의 변환 연구[13-17]들에서, UML 클래스 모델의 구조물에 대해 메타모델에 기반한 변환 명세 및 지원 도구를 지원한다. [13]에서 UML class constructs의 구문적 구조 및 시멘틱에 대한 정형 명세, Wizard tool을 통한 검증을 보여준다. [14]에서 Object-Z를 사용한 UML 메타모델의 정형명세를 제시했다. 즉, Fig. 3과 같이 UML 구조물(클래스 모델 등)인 추상적 구문, 정적 의미 및 동적 의미(dynamic semantics)를 캡슐화(encapsulation)하여 단일 정형 명세 언어인 Object-Z로의 변환 명세를 정의했다. [13]에서, 객체를 Object-Z class를 사용해서 클래스의 속성과 오퍼레이션을 하나의 스키마로 그룹화하여 표현한다. 또한, 클래스들간의 관계 유형들에 대해서도 상속(inheritance), 연관(association), 일반화(generalization) 등의 Object-Z 변환을 제시했다. 또한, 클래스 모델의 시멘틱에 대한 Object-Z 변환은 응용 모델 차원에서 그 적용사례를 통해 보여준다.

한편, Object-Z를 지원하는 모델 검사기들로 LaTeX 기반의 Unix 플랫폼에서만 동작하는 Wizard, Z를 주로 지원하고 부분적(파싱과 프린팅)으로 Object-Z를 지원하는 CZT, 그리고 그래픽 툴 형태로 개발된 Mac 플랫폼에서만 동작하는 TOZE[24]가 있다. 이러한 사용 환경의 제약으로 본 고에서는 MVC 기반 통합 메타모델의 Object-Z 스키마 명세에 대한 모델 검사는 다루지 않는다.



[Fig. 3] Model for converting Class model into Z/Object-Z [14]

3. MVC 기반 통합메타모델의 정형명세

2+1 View 통합 메타모델의 Z와 Object-Z로 정형적 명세를 위하여, Fig. 2의 MVC 기반 통합 메타모델을 대상으로 단원 2.3.1와 2.3.2에 제시된 클래스 모델과 Z[3,12,18]/Object-Z[13-15]간의 변환규칙을 각각 적용하여, Z와 Object-Z로의 정형적 명세를 기술한다. Z와 Object-Z 언어를 사용해서 통합 메타모델의 구조물이 가진 구문과 정적 의미에 대해 정형적으로 명세한다. 본 고

에서는 구축된 Z와 Object-Z 명세를 기술함에 있어 기 명세된 일부를 보여준다. 이러한 정형적 명세를 통해서, 그 모델의 의미를 명확하게 함으로서, 구축 모델에 대한 속성들(구문적 일관성 등)의 분석 및 검증이 가능해진다.

3.1 MVC 통합 메타모델의 Z 정형 명세

Z 명세의 방법은 메타모델내에 있는 요소들에 대해 기본 타입(basic type)과 자유 타입(free type)들을 선언하고, 변환 규칙에 의해서 메타모델내 구성 요소(클래스)들에 대한 스키마를 정의하고, 그리고 관계상에 존재하는 각 관계명(relationship name)별로 스키마를 정의한다[11]. 이때, 정적 시멘틱들을 위한 불변적 제약사항들은 스키마의 술어부 혹은 별도의 'theorem'으로 명세한다. Z 명세의 구축 결과로서, 기본타입으로 feature_ID외 49개, 자유타입으로 stateless_sessionbean외 3개이며, 요소 기반 class외 83개 스키마 명세, 그리고 관계 기반 Cont_cfe외 45개의 스키마를 명세하였다.

3.1.1 Z 기본타입 선언

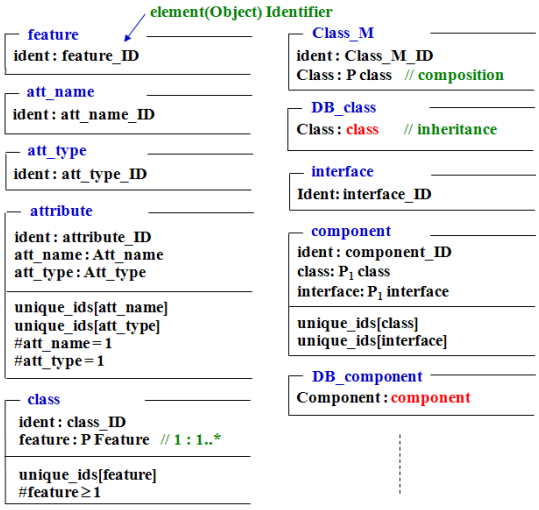
기본 타입들의 선언은 기본 타입과 자유 타입으로 기술된다. 기본 타입의 선언(Fig. 9)은 MVC 기반 통합 메타모델에 포함된 모든 요소들(단, 상속 관계를 갖는 자식 요소를 갖지 않는 최하위 단말 요소는 제외)에 대해서 각 요소별로 기본 타입을 명세한다. 이때, 각 클래스 대상의 기본 타입 선언에는 특정한 순서가 없으나, 하부 클래스에서 상위 클래스로 선언해 나가는 것이 Z 스키마 명세의 순서와 일치하기에 바람직하다. 자유 타입의 선언(Fig. 9)은 상속(inheritance) 관계들중에 role 타입들을 대상으로 선언한다.

3.1.2 Z 스키마 명세

Z 스키마 명세는 MVC 기반 통합 메타모델의 구문을 명세하기 위해 메타모델내에 포함된 모든 요소 그리고 관계들에 기반하여 분리된 Z 스키마들로 명세를 한다. 따라서, 클래스 수 및 관계 수 만큼의 스키마들이 생성된다. 아울러, 메타모델의 정적 시멘틱인 적형성에 대해 Z 스키마로 명세한다.

가. 요소 기반 Z 스키마 명세

요소 기반 스키마 명세는 메타모델을 구성하는 각 요소(클래스)를 대상으로 이 요소를 Z 스키마로 명세하고 이 요소에 직접 관계하는 요소들의 관계를 부분적으로 명세한다. 메타모델내 하나의 요소는 요소 식별자로서 하나의 스키마로 정의한다. 따라서, MVC 기반 통합 메타모델을 구성하는 각 클래스별로 스키마를 정의하며, 그 일부를 나타낸 것이 Fig. 4이다.



[Fig. 4] Element-based Z schema specification for MVC-based integrated metamodel

메타모델내 많은 요소(클래스)들간의 스키마의 선언 순서는 상향식 접근방식(bottom-up approach)으로 클래스 모델 구조상 하부 요소에서 상위 요소를 대상으로 스키마를 선언해 나간다. 예를 들어, Fig. 4에서 'class' 요소의 스키마 정의는 'DB_class' 스키마의 명세 이전에 선언되어야 한다. 그 이유는 메타모델 상에서, 'class'와 'DB_class'는 상속관계로서, 자식 클래스가 부모 클래스로부터 상속을 받아야 하기 때문에, 먼저 부모 클래스가 스키마로 선언되어야 한다.

각 스키마는 스키마 명, 변수들로 구성되는 선언부(signature part) 그리고 제약조건(다중성 등)의 semantics를 표현하는 술어부(predicate part)로 명세한다. 가령, 'attribute' 스키마의 술어부에 기술한 "#att_name = 1"은 불변적 제약사항의 속성을 표현한다. 이것은 attribute의 객체 수가 1개 이어야 함을 의미한다. 아울러, 요소들간의 관계(association, aggregation, composition, dependency 등)에 대해서 요소 기반 스키마에서 부분적으로 명세된다. 클래스간 연관 관계의 경우, 어느 클래스를 스키마로 선언해도 무방하다. 집합 및 복합 관계의 경우, 두 요소를 각각 스키마로 정의한 후 whole(parent) 요소의 스키마내 선언부에 part(child) 요소의 스키마를 포함하여 명세한다. 예를 들어, 'attribute' 클래스는 하부 클래스로 'att_name'과 'att_type'을 복합 관계로 포함하고 있기에, 'attribute' 스키마의 술어부에 포함시키고, 술어부에 유일

식별자 속성과 다중성을 표현한다. 이때, 포함되는 요소들은 'attribute' 스키마 이전에 별도의 스키마들로서 선언이 되어야 한다. 상속 관계를 갖지 않는 스키마 즉, 기본 타입으로 선언된 스키마들에 대해서는 그 선언부에 가령, "ident: feture_ID"으로 명세하여 이 스키마에 대한 유일성을 나타낸다. 의존 관계의 경우, 통상 의존하는 스키마를 먼저 스키마로 명세한다.

나. 관계 기반 Z 스키마 명세

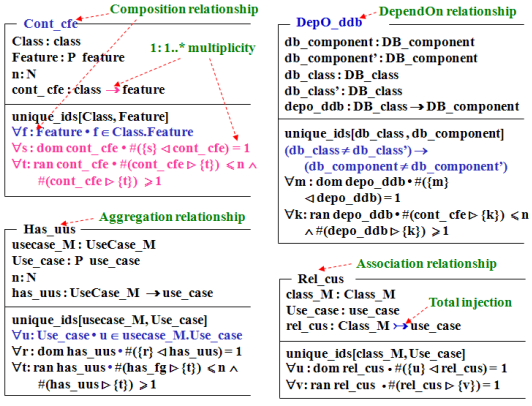
요소 기반 스키마 명세와 달리, 관계 기반 스키마 명세는 두 요소간의 관계별로 각기 스키마로 정의한다. 이 스키마는 클래스간 관계들과 다중성(multiplicity)을 명확하게 구분하여 정의된다. MVC 기반 통합 메타모델(Fig. 3) 내에 있는 라벨(label)이 부착된 모든 관계들에 대해 단원 2.3.1[11,12,18]에서 제시한 관계 기반 스키마 변환 규칙을 적용해서 명세의 일부를 나타낸 것이 Fig. 5이다.

Fig. 2에서 메타모델상의 관계 명칭은 관계의 유형 및 관계하는 두 개의 이름을 약칭하여 부여했다. 즉, 연관은 Rel_,, 집합은 Has_,, 복합은 Cont_ 및 의존은 DepO_로 명칭을 부여했다. 이때, 상속의 경우 라벨을 부착하지 않았는데 요소 기반 스키마 명세에서 표현되었기에 추가적으로 명세하지 않아도 된다. 이것은 두 요소간의 관계에 초점을 두어 Z 스키마로 명세하므로 별도의 명세 순서가 없이 상향식 접근으로 기술하면 된다.

Fig. 5에서, 'Rel_cus' 스키마는 Class_M과 use_case 요소간의 연관 관계를 명세한다. 연관 관계는 이 스키마의 시그네쳐부에 두 요소를 변수로서 선언하면 된다. 이 요소들은 요소 기반 스키마 명세에서 각각 스키마들로 정의되어야 한다. 다음으로 연관의 특수한 형태인 집합(aggregation) 관계를 명세한 'Has_uus" 스키마는 UseCase_M과 use_case 요소간의 관계를 표현한다. 즉, "UseCase_M 요소가 use_case를 가진다" 라고 명세하면 된다. 이것은 'Has_uus" 스키마의 술어부에 집합 관계를 "∀u: Use_case • u ∈ usecase_M.Usecase"와 같이 표현한다. 두 요소간 관계하는 다중성은 선언부(Power set: 객체 2개 이상, 관계 함수)와 술어부(dom: 정의역-부모 요소, ran: 치역-자식 요소)에 표현한다. 관계함수의 예로서, "1(A) : 0..1(B)"이면, 다중 수에 따라 "A ⤔ B"의 Total injection 함수로 표현한다. 복합 관계는 집합 관계에 정의역(whole)의 다중 수가 1이라는 제약(자식 클래스가 부모 클래스를 한 개만 가짐)을 추가해서 표현한다. Fig. 5에서 "Has_uus"와 "Cont_cfe"가 동일하게 표현되었는데, 집합 관계인 "Has_uus"의 정의역이 1이기 때문이다. 의존 관계는 한쪽의 변경에 다른 쪽에 영향을 주는 관계로 "DepO_ddb" 스키마에서 이를 변경 전과 변경 후의 일치 여부로 표현한다.

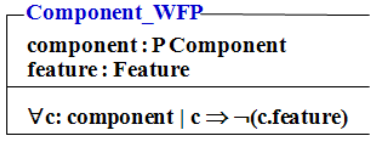
다. 적형성 속성의 명세

정적 의미를 표현하는 적형성 속성(Well-formedness property)은 메타모델내 요소와 이들간의 관계들에 대한 불변적 제약사항을 의미한다. 클래스 모델로 표현된 메타모델내에 표현되지 않은 이외의 수많은 제약사항들을 갖고 있다.



[Fig. 5] Relationship-based Z schema specification for MVC-based integrated metamodel

본 논문에서는 메타모델의 요소와 관련된 적형성 규칙들을 속성(property)으로 정의한다. 예로서, Fig. 2에서, component 모델(Component_M)의 component와 interface 요소(메타모델상 비 표현)가 갖는 적형성 속성을 대상으로 명세한다. 'component' 요소에 대한 적형성 속성으로서 " component는 어떤 'feature'도 가지지 않는다."에 대한 Z 표현을 Fig. 6에서 보여준다. 이 속성은 Z/EVES tool을 통해서 그 정확성이 검증되어진다.



[Fig. 6] Specification of well-formedness property for component element

또 다른 적형성 속성으로, 'interface' 요소가 갖는 속성은 다음과 같은 것이 있다.

- interface는 operation만을 갖는다.
- interface내에 정의된 모든 feature들은 public이다.

이들에 대한 Z 명세는 별도의 스키마로 Fig. 7과 같이 기술한다.

Interface_WFP

interface : P Interface
feature : P Feature
operation : P Operation

{interface} ⊲ operation = ∅
∀f: feature • feature ∈ interface ⇒ feature = public

[Fig. 7] Specification of well-formedness property for interface element

3.2 MVC 통합메타모델의 Object-Z 정형명세

Z 명세 방법은 요소 기반과 관계에 기반해서 Z 스키마를 명세하나, Object-Z 명세의 방법은 객체 패러다임을 표현할 수 있도록 Object-Z의 'Class' construct를 사용해서 속성과 오퍼레이션을 캡슐화하여 클래스간 관계가 반영된 요소 기반의 클래스 구조물로 메타모델을 변환, 정의한다[14]. 클래스 요소별로 정의되는 스키마에 클래스들간의 관계(상속, 연관), 인터페이스(클래스의 오퍼레이션 목록), 다중성 그리고 구문과 시멘틱(OCL 등)를 통합하여 표현한다. 반면, 단원 3.1에서 Z 명세의 경우는 속성 스키마와 오퍼레이션 스키마가 별도로 작성되며, 요소와 관계가 분리된 스키마 명세하며, 인터페이스를 명세하지 못한다. 본 고에서 정형 명세의 대상이 Fig. 2의 메타모델로 각 클래스의 명칭만 제공되고, 각 클래스가 가진 속성과 오퍼레이션이 없는 관계로 Object-Z와 Z의 스키마 명세상에 그 차이가 뚜렷하게 나타나지 않는다. 즉, 클래스의 오퍼레이션이 있어야 속성과 통합된 형태로 인터페이스를 포함해서 Object-Z 스키마로 정의하여 보여줄 수 있다. 또한, 스키마내에서 상속받은 스키마로부터 객체를 생성하여 'self' 예약어를 사용해서 그 인스턴스(instance)를 참조할 수 있다.

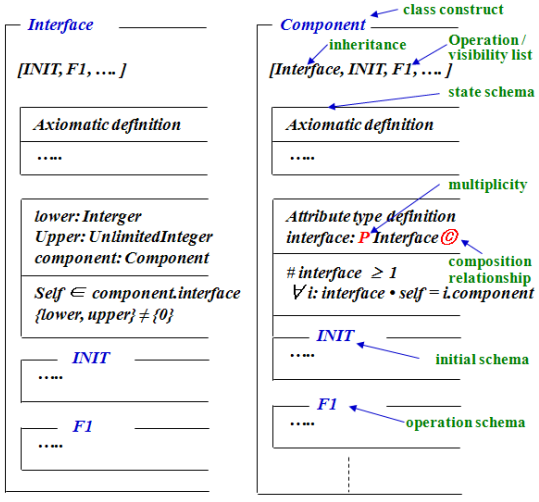
결국, Object-Z의 명세는 타입(기본, 자유) 선언과 클래스 스키마들로 구성하여 정의한다. 이때, 클래스별로 명세되는 각 Object-Z 클래스의 내부는 상속 부모 클래스(존재시), 인터페이스 목록(visibility list), 상수 정의(공리정의), 상태 스키마, 시작 스키마, 오퍼레이션 스키마들로 구성된다. 이때, 내부 스키마들은 이 클래스의 스키마내에 중첩된 형태로 표현된다.

3.2.1 Object-Z 기본타입 선언

Z와 동일한 방법으로 primitive type(원시 타입)을 위한 Object-Z의 기본 타입들의 선언은 기본 타입과 자유타입으로 기술된다. 기본 타입의 선언 대상은 후속하여 정의되는 각 스키마에 대해 그 정체성을 선언한다.

3.2.2 Object-Z 스키마 명세

Object-Z 스키마 명세는 MVC 기반 통합 메타모델의 구문을 명세하기 위해 메타모델내에 포함된 모든 요소별 분리된 Z 스키마들로 요소간 관계를 포함해서 명세한다. 즉, 메타모델내 하나의 클래스는 요소 식별자로서 하나의 Object-Z 클래스로 명세한다. MVC 통합 메타모델에 대한 Object-Z 명세의 그 일부를 나타낸 것이 Fig. 8이다. Fig. 8에서 'component' 요소의 경우, 속성 및 오퍼레이션이 메타모델상에 존재하지 않기에 임시적 명칭으로 부여하였다.



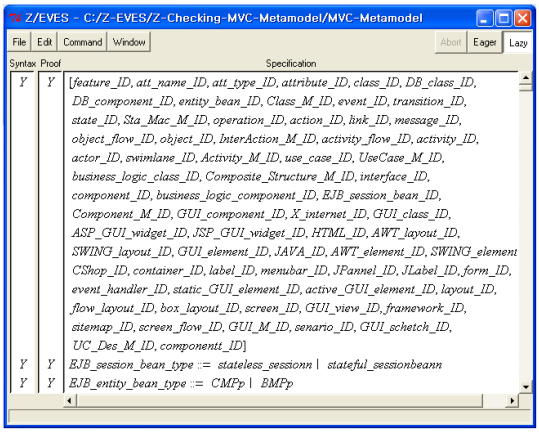
[Fig. 8] Object-Z schema specification for MVC-based integrated Metamodel

4. Z 명세의 모델 검사

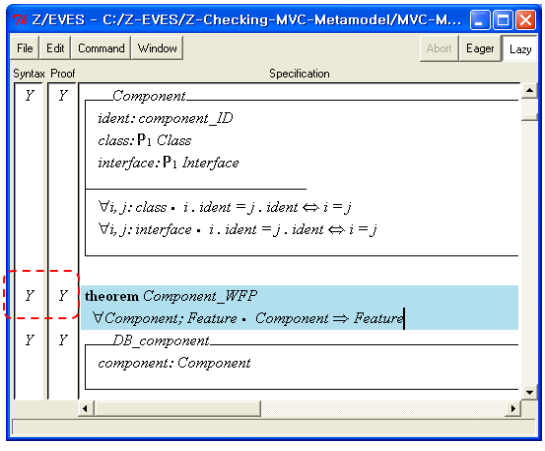
MVC 기반 통합 메타모델의 구조적 및 의미적 정확성을 검증하기 위해, Z 스키마로 명세하였는데, Z 명세의 정확성 검사가 필요하다. 즉, 명세된 Z 스키마에 대해 Z/Eves Tool(V2.0)을 사용해서 메타모델의 구문 검사(syntax checking), 가변 범위 검사(variable range checking) 및 일치성 타입 검사(consistency type checking)를 해야 한다. Z 명세의 검증 툴 중의 하나로 널리 사용되는 Z/Eves Tool의 분석 기능은 타입 검사, 스키마 확장(schema expansion), 사전 조건 계산(precondition calculation), 도메인 검사, 명세 애니메이션(specification animation) 그리고 범용 이론 증명(general theorem

proving)을 제공한다[18].

본 고에서는 이 툴의 Z 편집기를 사용해서 Z 명세를 입력하고, 이론 증명기를 실행하여 메타모델의 구문 검사, 타입 검사, 그리고 도메인 검사를 수행한다. 타입 선언, 요소 기반 및 관계의 기반 스키마 명세가 Z/Eves 툴에서 한 개의 파일로 작성, 검사된다. Z/Eves 툴에 기 명세된 Z 스키마에 대해 수정 표현, 일부 미 표현되는 경우가 발생하였다. 이것은 Z 언어의 표현력(명제, 술어논리)에 대해 Z/Eves에서 완전히 지원치 못하는 제약이 있어서, 가능한 명세 정보의 손실없이 변환, 표현하여 검사를 하였다. Z/Eves 툴의 타입(기본타입은 상단 부분, 자유 타입은 하단 부분) 검사에 대한 실행결과인 Fig. 9는 왼쪽 상단에 표시된 'syntax'는 구문과 타입 검사의 결과를 보여주는 것으로 'Y'로 나타나면 타입간 일치성이 있음을 나타낸다. 'proof'는 타입의 도메인 검사의 결과를 보여주는 것으로 'Y'로 나타나면, 그 명세가 정확함을 의미한다.



[Fig. 9] Syntax checking result for type of Z specification



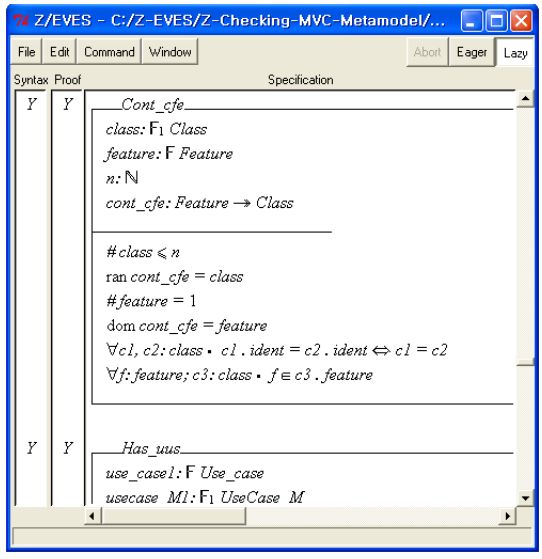
[Fig. 10] Syntax checking result for element-based Z schema specification

Fig. 10은 요소 기반 Z 스키마 명세에 대한 모델 검사의 결과를 보여준다. Fig. 10에서 'component' 스키마는 선언부에 식별자 선언 및 복합 관계로 하부 'class'와 'interface' 스키마를 포함, 명세하고, 이 포함 요소들의 유일성을 술어부에 명세했다.

하단에 표시된 "theorem Component_WFP"는 기 명세한 Fig. 6의 "component는 어떤 'feature'도 가지지 않는다" 적형성 속성을 theorem 예약어 통해 이 불변적 제약을 명제 논리로 표현, 입력하여 reduction('prove by reduce')에 의해 이 이론이 정확함을 보였다.

관계 기반 Z 스키마 명세의 모델 검사의 결과를 나타낸 것이 Fig. 11이다. Fig. 11에서, 'Cont_cfe' 스키마는 'class'와 'feature' 요소간 복합 관계에 대해 'syntax'와 'proof' 부문이 "Y" 의 결과를 보임으로서 Z 명세가 정확함을 알 수 있다. 'proof' 부문의 검사는 reduction의 'prove by reduce'를 수행하여 도메인 에러가 없다면 proof 된다.

결과적으로, Z 명세의 검사를 통해, MVC 기반 통합 메타모델의 구문적 타입 및 도메인 검사에서 오류가 없음을 입증할 수 있었다.



[Fig. 11] Syntax checking result for relationship-based Z schema specification

5. 비교 평가

이전에 MVC 기반 통합 메타모델의 복잡한 구문적 및 정적 시멘틱에 대해 Z와 Object-Z를 사용해서 정형적으

로 명확히 명세하고, Z/Eves 툴을 통해 메타모델의 정확성을 검증하였다. 본 장은 정형 명세와 검사 측면에서 제시 방법과 기존 방법들간의 특성을 비교 평가하고, 그 한계점을 다룬다.

5.1 기존 방법과의 비교평가

먼저, 시각적 모델의 정형 명세를 위한 접근 특성 및 변환방법의 측면에서 기존 방법과 제시 방법간의 상이점을 비교분석을 나타낸 것이 Table 1이다. 변환 방법의 제시 혹은 정립에 초점을 둔 것으로 [11]은 클래스 모델의 Z로 변환, [14]는 클래스 모델의 Object-Z로 변환, 휘처 모델의 Z 변환이 있다. 기존 변환 방법을 적용한 모델의 정형 명세와 검증에 중점을 둔 [19]와 [20]이 있다. 제시 방법을 포함한 기존 방법들은 클래스 요소 및 관계, 다중성에 기반하여 정형명세를 위한 방법의 제시 혹은 적용을 다룬다. 제시 방법은 복잡한 메타모델의 전체 구조를 명확히 명세할 수 있도록 기 제시된 스키마들간의 선언 순서 규칙을 적용해서 스키마들을 명세함으로서 구조적 오류를 제거하였다.

다음은 Z 명세와 Z/Eves 툴에 의한 모델의 검사 측면에서 기존 연구들과의 비교평가를 보여주는 것이 Table 2이다.

Table 2에서, [14]가 클래스 모델의 구문적/의미적 측면에서 Object-Z로 체계적이고 명확한 변환, 명세 기법이 우수하나, 명세된 Z 스키마에 대한 검사 도구를 통한 접근을 제시하고 있지 않다. 반면, 제시 방법은 정형 명세와 모델 검사에서 양호하다고 할 수 있다.

[Table 1] Comparison with approach characteristics and translation method

Evaluation factor	ClassModel-Z [11]	ClassModel-Object-Z [14]	Framework [19]	FeatureModel-Z [18]	Proposed method
Purpose	- Transformation technique between constructs - Formal specification	- Transformation technique between constructs - Formal specification	- Formal specification of model - model checking	- Transformation technique between constructs - Formal specification and checking	- Formal specification of model - model checking
Formal language / Validation tool	- Z - None	- Object-Z - None	- Z - Z/Eves	- Z - Z/Eves	- Z, Object-Z - Z/Eves
Target model / Model level	- Class model - App. model	- Class(UML) model - Metamodel, App. model	- Embeded Framework - Metamodel	- Feature model - Metamodel, App. model	- 2+1 View integrated metamodel - metamodel
Converting technique / Relationship-based multiplicity / Specification order	- Element- and Relation-based specification - Partial applying - No applying	- Element-based specification - Partial applying - No applying	- Element- and Relation-based specification - Applying - Applying	- Element- and Relation-based specification - Applying - Applying	- Element-(Object-Z) and Relation-based (Z) specification - Applying - Applying
Object	- No support	- Support	- No support	- No support	- Support

[Table 2] Evaluation for Z/ Object-Z specification and model checking

Evaluation factor		ClassModel-Z [11]	ClassModel-Object-Z [14]	Framework [19]	FeatureModel-Z [18]	Proposed method
Expressive power of model specification	**Syntax**	0.75	1.00	0.75	0.75	0.75
	Static semantics	0.25	1.00	0.25	0.25	0.75
	Dynamic semantics	0.00	1.00	0.00	0.00	0.00
Property of model checking	**Syntax**	0.00	0.00	0.75	0.75	0.75
	Semantics	0.00	0.00	0.00	0.00	0.75
Accuracy for specification and checking of metamodel		0.50	0.75	0.75	0.75	0.75
Total		1.50	3.75	2.50	2.50	3.75

[Legend] 1.00(very high), 0.75(good), 0.50(medium), 0.25(low), 0.00(very low)

5.2 한계점

제시 방법의 한계점은 다음과 같이 들 수 있다.

- 메타모델 모델의 각 요소 및 관계에 내재된 의미적 속성인 동적 시멘틱에 대한 변환 명세 및 검사 방법을 지원하지 못한다.
- Z 명세의 Z/Eves 툴을 통한 모델 검사시, 툴 지원의 제약에 따른 Z 명세서 대비해서 완전히 일치된 검증을 못했다.
- Object-Z 명세의 지원 툴들의 사용 환경의 제약으로 Object-Z 스키마에 대한 검사를 수행하지 못하였다.

5. 결 론

기 제시된 2+1 View 통합 메타모델은 시각적 표현의 클래스 모델로 되어 있어, 이 모델의 구문적 및 정적 의미에 대한 명확한 명세와 정확성에 문제를 갖고 있었다. MVC 기반 통합 메타모델을 Z와 Object-Z로 명세하고, Z/Eevs를 사용해서 메타모델의 정확성을 보였다. Z와 Object-Z 명세는 MVC 기반 통합 메타모델을 대상으로 기존의 변환 규칙을 적용해서 기본타입, 요소와 관계 기반의 스키마들을 정의했다. Z 스키마 명세의 모델 검사는 Z/Eevs 툴을 사용해서 구문, 타입 검사 및 도메인 검사를 수행하여 메타모델의 구문적 구조가 정확함을 보였다. 특히, 메타모델에 내재된 적형성 속성에 대해 Z로 명세하고, Z/Eevs 툴에서 이 속성을 만족해야 하는 이론을 증명하였다. 이로서, 2+1 View 통합 메타모델의 구조물을 명확화하고 정확성을 입증했다. 향후 연구로서, 2+1 View 통합 메타모델의 동적 시멘틱에 대한 정형적 명세와 검증 그리고 Object-Z 명세에 대한 검증 도구를 통한 검사가 필요하다.

References

[1] Object Management Group, MDA Guide Version 1.0.1, 2003.
http://www.omg.org/docs/omg/03-06-01.pdf

[2] Y. S. Lim, M. Kim, S. N. Jeong, and A. M. Jeong, "A New Component Model and Framework for Reuseof Components", Korean Institute of Inform ation Scientists and Engineers, Vol.34, No.12, 2007. 12.

[3] C. Y. Song and E. S. Cho, C, "A PIM/PSM Component Modeling Technique based on 2+1 View Integrated Metamodel", Korea Information Processing Society, Vol.16-D, No.3, pp.381-394, 2009. 6.

[4] Object Management Group, Unified Modeling Language: Infrastructure V2.1.1, 2007.
http://www.omg.org/docs/formal/07-02-4.pdf.

[5] J.M. Spivey. "The Z Notation: A Reference Manual," 2nd Ed. Prentice Hall, 1992.

[6] J. Woodcock, J. Davis, "Using Z: Specification, Refinement and Proof, International Series in Computer Science," Prentice-Hall, http://www.usingz.com/text/online, 1996.

[7] G. Smith, "The Object-Z Specification Language", Kluwer Academic Publishers, 1999.

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

[9] M. Saaltink, The Z/EVES 2.2 Mathematical Tool kit. ORA Canada Technical Report TR-03-5493-05c, June 2003.

[10] B. France, J. M. Bruel, M. M. Larrondo-Petrie, and M. Shroff, "Exploring The Semantics of UML Type Structures with Z", in Proc. 2nd IFIP Workshop on Formal Methods for Open Object-Based Distributed Systems (FMOODS), Canterbury, UK, 1997.

[11] 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.

[12] Chee-Yang Song, “A Metamodel-Based Modeling Mechanism for Hierarchical Design in UML", Thesis for the Degree of Doctor, 2003. 7.

[13] S. K. Kim, D. Carrington, "Formalizing the UML Class Diagram Using Object-Z", «UML»’99 − The Unified Modeling Language LNCS, Vol.1723, pp.83-98, 1999.

[14] S. K. Kim, D. Carrington, “A Formal Denotational Semantics of UML in Object-Z”, the special Issue of the journal of l'Objet, To appear (2000)
http://citeseerx.ist.psu.edu/showciting?cid=2216695

[15] S. K. Kim, D. Carrington, "A Formal Mapping between UML models and Object-Z Specification ", The University of Queensland, Australia, Technical report 00-03, 2000.

[16] S. K. Kim, D. Carrington, “A Formal Model of the UML Metamodel: the UML State Machine and its Integrity Constraints”, Dept. of Computer Science and Electrical Engineering, University of Queensland, Australia. 2002.

[17] M. Bittner, F. Kamm¨uller, "Translating Fusion/UML to Object-Z", Formal Methods and Models for Co-Design, MEMOCODE '03. Proceedings. pp.49-50,

June 2003.
DOI: http://dx.doi.org/10.1109/MEMCOD.2003.1210087
[18] C. Y. Song, E. S. Cho, and C. J. Kim, "A Formal Specification and Checking Technique of Feature model using Z language", Korea Society of Computer Information, Vol. 15, No. 1, pp.123-136, 2013. 4.
DOI: http://dx.doi.org/10.9708/jksci.2013.18.1.123
[19] E. S. Cho, C. J. Kim, and C. Y. Song, "A Formal Specification of Reusable Framework of Embedded System,", Korea Information Processing Society," Vol.17-D, No. 5, pp. 431-442, 2010. 12.
DOI: http://dx.doi.org/10.3745/KIPSTD.2010.17D.6.431
[20] Chee-Yang Song, Eun-Sook Cho, and Chul-Jin Kim, "AN INTEGRATED GUI-BUSINESS COMPONENT MODELING METHOD FOR THE MDD- AND MVC-BASED HIERARCHICAL DESIGNS", IJSEKE, Vol.21, No.3, pp.447-490, June 2011.

송 치 양(Chee-Yang Song) **[정회원]**

- 1985년 2월 : 한남대학교 전자계산학과 (공학사)
- 1987년 2월 : 중앙대학교 전자계산학과 (공학석사)
- 2003년 8월 : 고려대학교 컴퓨터학과 (이학박사)
- 1990년 5월 ~ 2005년 9월 : KT 중앙연구소 책임연구원
- 2005년 10월 ~ 2008년 2월 : 상주대학교 소프트웨어공학과 조교수
- 2008년 3월 ~ 현재 : 경북대학교 컴퓨터정보학부 부교수

<관심분야>
CBD, MDA, Service-oriented modeling, Formal specification