The Transactions of the Korea Information Processing Society
TKIPS Vol.14, No.10, pp.804~812

eISSN: 3022-7011
https://doi.org/10.3745/TKIPS.2025.14.10.804

Semi-Automatic Code Generation through Analyzing Natural Language Requirement Specifications with Generative AI

Yejin Jin† · Janghwan Kim†† · R. Young Chul Kim†††

ABSTRACT

Recently, generative AI tools have been increasingly utilized in the field of software development. However, the reliability of AI-generated outputs still cannot be guaranteed. In particular, current AI-based code generation often lacks a clear reflection of requirements and sufficient traceability of the design process. To address these limitations, we propose a mechanism that partially applies generative AI and applies metamodel-driven model transformation across the software process, from natural language requirements to skeleton code generation via UML design. We define transformation rules that are automatically applied on a Metamodel Transformation Engine. This approach may provide an automated development process that ensures the consistency of natural language requirements, the traceability of design artifacts, and the quality of the generated code. In future research, we aim to compare human-written code with AI-assisted code through the proposed mechanism. The objective is to verify which approach ensures higher software quality and, by doing so, enhances the reliability of AI-generated code while supporting the development of high-quality software.

Keywords : Natural Language Requirement Specifications, UML Design, Code Generation, Generative AI

생성형 AI를 활용한 자연어 요구사항 분석 기반의 반자동 코드 생성

진 예 진† · 김 장 환†† · 김 영 철†††

요 약

최근 생성형 AI 도구는 소프트웨어 개발 현장에서 많이 사용되고 있다. 그러나, 아직은 AI 기반 생성 결과에 대한 신뢰성을 보장할 수 없다. 또한, 현재도 AI 기반 코드 생성은 요구사항의 명확한 반영과 설계에 대한 추적이 부족하다. 이를 해결하기 위해, 소프트웨어 프로세스인 자연어 요구사항으로부터 UML 설계 및 스켈레톤 코드 생성까지 생성형 AI를 부분적으로 적용하고, 메타모델 기반 모델 변환하는 메커니즘을 제안한다. 이를 위해 변환 규칙 프로그래밍을 Metamodel Transformation Engine을 통해 자동 변환을 수행한다. 이는 자연어 요구사항의 정합성, 설계의 추적성, 그리고 코드의 품질을 확보할 수 있는 자동화된 개발 프로세스를 제공하고자 한다. 이를 통해 생성형 AI 코드의 신뢰성을 보완하고, 고품질의 소프트웨어를 개발하고자 한다. 향후 연구로, 이러한 메커니즘 통해 인간 중심의 코드와 AI가 부분적으로 접목된 코드를 비교하고자 한다. 그 결과로 어떠한 코드가 더 좋은 품질을 갖는 지 확인하는 것이 목표이다.

키워드 : 자연어 요구사항 스펙, UML 설계, 코드 생성, 생성형 AI

1. 서론

최근 소프트웨어 개발 현장에서는 생성형 AI 도구의 활용이 빠르게 확산되고 있다. 대부분의 개발자는 이러한 AI 도구를 통해 초기 코드 베이스를 자동으로 생성하거나 코드 자동 완성, 오류 수정 등을 수행한다[1]. 개발에 소요되는 시간과 비용을 줄이고자 한다. 그러나, 현재의 AI 코드는 확률적 내부 알고리즘을 기반으로 생성된다. 따라서 코드 생성 과정을 개발자가 명확히 검증하기 어렵고, 코드 품질이 일정 수준 이상으로 보장되기 어렵다. 이러한 한계점은 개발자가 AI의 코드가 요구사항이 충분히 반영되었는 지를 반복적으로 확인하게 한다. 또한, 개발자가 이를 모두 해석하고 수정하게 한다. 따라서, 기존 연구에서는 소프트웨어 개발 생명 주기를 기반으로, 생성형 AI를 부분적으로 적용하는 메커니즘을 제안하였다

※ 본 연구는 2025년도 문화체육 관광부의 재원으로 한국콘텐츠진흥원(과제명: 인공지능 기반 대화형 멀티모달 인터렉티브 스토리텔링 3D장면 저작 기술 개발, 과제번호: RS-2023-00227917, 기여율: 100%) 지원과 한국연구재단의 4단계 두뇌한국21사업(과제명: 초분산 자율 컴퓨팅 서비스 기술 연구팀, 과제번호: 202003520005)의 지원을 받아 수행된 연구임.
† 준 회 원 : 홍익대학교 소프트웨어공학연구실 박사과정
†† 준 회 원 : 홍익대학교 소프트웨어공학연구실 박사수료
††† 정 회 원 : 홍익대학교 소프트웨어융합학과 교수
Manuscript Received : July 22, 2025
Accepted : August 13, 2025
* Corresponding Author : R. Young Chul Kim(bob@hongik.ac.kr)

[2]. 해당 메커니즘은 자연어 요구사항 분석, UML 설계 생성, 그리고 메타모델링을 통한 스켈레톤 코드 생성으로 진행된다. 그 절차는 다음과 같다: 1) 자연어로 작성된 소프트웨어 요구사항 명세서에서 기능적 요구사항을 분류한다. 2) 기능적 요구사항 문장을 전처리한다. 3) 촘스키의 구문 구조 분석 이론을 통해 문장의 구조와 형태소를 분석한다. 4) 필모어의 의미론적 분석 이론을 통해 명사에 의미적 역할을 부여하여 문장의 의미를 분석한다. 5) 분석된 문장에서 중복되는 요소들은 제거한다. 6) 요구사항 분석 결과와 UML 설계의 요소를 매핑한다. 7) UML 설계를 생성한다. 8) 코드를 생성하기 위해 UML 설계 데이터를 저장한다. 9) 메타모델링을 통해 모델 변환을 수행한다. 10) 모델 변환을 통해 실행 가능한 스켈레톤 코드를 생성한다. 이러한 연구는 자연어 요구사항을 개발에 반영하기 위한 접근으로, 스켈레톤 코드까지를 목표로 한다. 생성된 스켈레톤 코드는 요구사항을 반영한 설계를 토대로 생성된다. 구조화된 스켈레톤 코드가 요구사항을 정확히 반영함으로써 개발 초기의 품질을 개선한다. 결과적으로 코드의 내부 품질 향상에도 기여할 것이다. 또한, 더 높은 수준의 요구사항 정합성을 확보할 수 있을 것으로 기대한다. 그러나 기존 연구에서는 스켈레톤 코드를 생성하기 위해 메타 모델에 대해서만 언급한다. 그 구조와 작동 방식에 대해 구체적으로 다루지 않는다. 따라서, 본 연구는 이를 발전시켜, Metamodel Transformation Engine의 구조 및 구체적 변환 절차를 제안한다. 이를 통해 누구나 동일한 변환 규칙을 적용할 수 있도록 한다. 따라서, 향후 자동화 가능성을 위한 규칙 기반의 엔진을 제공한다.

2장에서는 관련 연구로서, UML 메타모델링 기반 코드 발생 연구를 설명한다. 3장에서는 본 연구의 메커니즘을 제안하고, 4장에서는 사례 연구를 통해 메커니즘을 설명한다. 마지막으로 5장에서는 결론 및 향후 연구에 대해 언급한다.

2. 관련 연구

기존 연구[3]의 모델 기반 소프트웨어 개발(MDD)는 코드 자동 생성을 위해 xUML을 사용한다. xUML은 실행이 가능한 UML 설계를 의미하며 많은 연구에서는 모델 기반 코드 생성을 보다 효율적으로 수행하기 위해 xUML을 사용한다. 이러한 xUML의 설계의 기반이 되는 요소가 UML 메타모델이다. 그림 1은 기존 UML 메타모델 중 Class 다이어그램에 대한 메타모델이다. 대부분의 연구는 그림 1과 같이 하나의 UML 모델을 모델 변형 자동화에 사용한다[4]. 따라서, 설계 모델 간 통합이 어렵고, 여러 메타모델과 그에 따른 모델 변환 규칙이 추가적으로 필요하다. Fig. 1의 메타 모델을 다른 메타 모델에 따라 정의된 대상 모델로 변환하는 과정이 필요하다. Fig. 2는 모델 변환 과정에 대해 보여준다.

ATL(ATLAS Transformation Language)은 모델을 변환하는 규칙을 명세하기 위한 엔진으로 사용된다. 변환 과정은 메

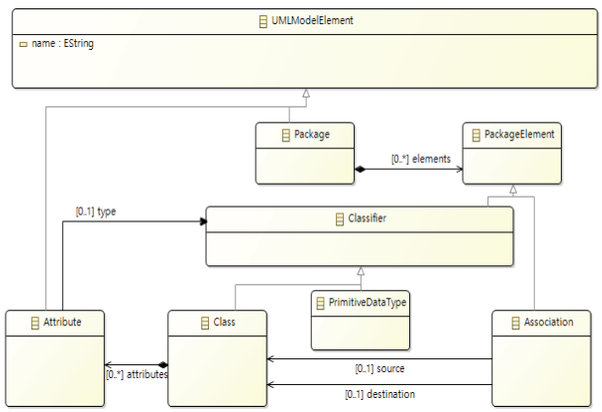

Fig. 1. Class Diagram Metamodel

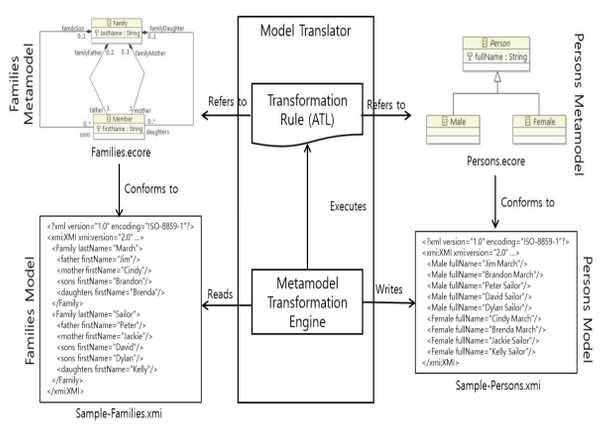

Fig. 2. Metamodel-based Model Transformation using ATL[4]

타모델 정의, 모델 생성, ATL 변환 규칙 작성, 그리고 모델 변환 실행의 단계를 포함한다. 변환될 소스 모델과 생성될 대상 모델의 구조를 정의하는 메타모델을 생성한다. 그리고, 정의된 메타모델에 따라 실제 소스 모델 인스턴스를 생성한다. 이후 ATL을 사용하여 소스 메타모델의 요소들과 대상 메타모델 요소들의 변환 규칙을 작성한다. 마지막으로 ATL 변환 엔진을 사용하여 작성된 규칙에 따라 소스 모델을 대상 모델로 자동 변환한다. ATL은 Eclipse 플랫폼 기반의 통합 개발 환경을 제공하며, 구문 강조 표시, 디버거와 같은 표준 개발 도구를 통해 변환 규칙 작성을 지원한다. ATL을 이용한 메타모델 기반의 모델 변환은 모델을 변환하는 작업을 자동화하여 개발 시간과 노력을 크게 절감하고 생산성을 향상시킨다. 또한, ATL 변환 규칙은 명확하게 정의되어 변환된 모델이 일관된 품질과 정확성을 유지하도록 돕는다. 그러나 ATL을 이용한 메타모델 기반의 모델 변환은 몇 가지 한계점을 가진다. 새로운 사용자는 ATL의 학습 난이도가 높다. 특히 복잡한 변환 논리를 표현하기 위해서는 ATL 언어의 숙련도가 요구된다. 매우 복잡하거나 비정형적인 변환 로직의 경우, ATL 규칙을 작성하는 것은 어렵고, 복잡한 ATL 변환 규칙은 디버깅이 어려울 수 있다. 또한, 소스 또는 대상 메타모델의 변경은 해당

ATL 변환 규칙의 수정으로 이어질 수 있어, 메타모델의 잦은 변경은 유지보수 비용을 증가시키는 요인이 된다.

따라서, 우리는 메타 모델간의 변환에 ATL 변환 규칙을 사용하지 않고, 보다 직관적이고 유연한 방식인 규칙 기반의 알고리즘을 사용한다. 본 연구에서는 소스 모델의 구조를 탐색하고, 사전에 정의된 규칙에 따라 대상 모델로의 변환을 단계적으로 수행한다. 이 방식은 코드 형태로 변환 로직이 명시적으로 표현되기 때문에 디버깅과 유지보수가 용이하다. 또한, 프로젝트의 변동성에 따라 변환 규칙을 유동적으로 수정할 수 있다. 메타 모델 구조 변경에 대해서도 빠르게 적응할 수 있어 확장성 측면에서 유리하다.

3. 생성형 AI를 활용한 자연어 요구사항 분석 기반의 반자동 코드 생성

본 연구는 자연어 요구사항으로부터 실행 가능한 스켈레톤 코드를 자동으로 생성하는 것을 목표로 한다. 이를 위해 소프트웨어 개발 프로세스에 따라 요구사항 분석, 설계, 코드 생성의 단계를 거친다. 1) 요구사항 분석에는 기능적 요구사항만을 분류하여 사용한다. 또한, 두 가지의 언어학을 사용하여 체계적으로 분석하며, 이를 자동화하는 시도로 LLM을 사용한다. 2) 설계 생성 단계에서는 분석된 요구사항 결과를 바탕으로 PlantUML을 사용하여 UML 다이어그램을 생성한다. 3) 마지막으로, 코드 생성 단계에서는 UML 다이어그램의 메타모델과 코드 AST 모델의 메타모델을 만든다. 그리고 메타 모델 변환 엔진을 통해 메타 모델간의 데이터 변환을 수행한다. 본 메커니즘은 소프트웨어 공학적 방식에 AI를 부분적으로 사용한다. 또한, 기존 연구에서 설명되지 않은 메타 모델 변환 기법에 대해 상세히 기술하여 자동화 프로세스를 구축할 수 있도록 한다.

3.1 생성형 AI를 적용한 요구공학

우리는 소프트웨어 요구사항 명세서(SRS)로부터 기능적 요구사항을 분석한다. Fig. 3은 자연어 요구사항 분석을 위한 4가지의 절차를 보여준다.

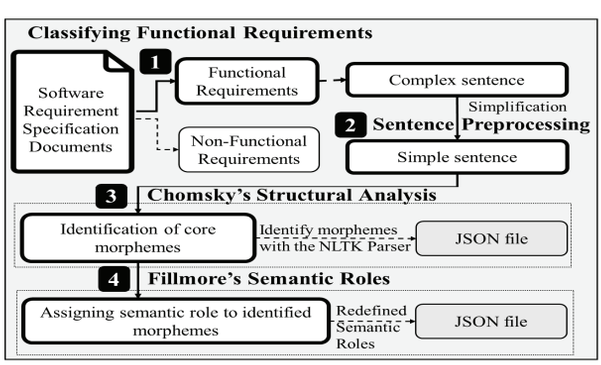

Fig. 3. Procedure for Requirements Analysis

1) SRS 문서로부터 기능적 요구사항 분류

개발자들은 자연어로 작성된 소프트웨어 요구사항 명세서의 기능적 요구사항을 기반으로 개발을 수행한다. 소프트웨어 요구사항 명세서는 이해 관계자들의 의사소통에 따라 고객의 니즈를 파악하여 작성된다. SRS 문서는 시스템 개요, 기능적 요구사항, 비기능적 요구사항 등을 포함한다. 개발자들은 이러한 소프트웨어 요구사항 명세서를 수동으로 읽고, 기능적 요구사항을 분류해야 한다. 따라서, 본 연구에서는 지도 학습을 통한 분류로 기능적 요구사항을 자동으로 분류한다.

2) 요구사항 문장 전처리

자연어 요구사항은 많은 절과 연결 접속사들로 이루어져 있다. 복잡하게 연결된 문장은 자연어 분석에 매우 어렵다. 서로 대등한 절이 연결된 문장은 중문이라고 하고, 주절과 종속절로 이루어진 문장은 복문이라고 한다. 또한, 하나의 절로만 이루어진 문장은 단문이라고 한다. 따라서 여러 개의 절로 이루어진 중문과 복문을 복잡한 문장으로 간주하고, 이를 단문으로 분해한다. 중문의 경우, 'and', 'but'과 같은 등위 접속사를 기준으로 두 개의 절을 개별 문장으로 분리한다. 복문의 경우, 'when', 'if'와 같은 종속 접속사가 포함된 종속절과 주절을 개별 문장으로 분리한다. 새로운 문장으로 분리할 때, 주어, 동사, 목적어가 누락 될 수 있다. 따라서 기존의 문장의 단어를 해당 위치에 추가한다.

3) 문장 구조 및 형태소 분석

문장의 구조를 해석하기 위해 촘스키의 구문 구조 분석 이론을 사용한다[5]. 촘스키의 이론은 언어의 구조에 대한 기초와 규칙을 설명한다. 문장은 하나 이상의 절로 구성되며, 절은 구, 단어, 형태소 등 더 작은 단위로 세분화된다. 많은 연구에서는 문장을 시각화할 수 있는 파서를 사용한다. Fig. 4는 문장의 구조를 보여준다.

시각화된 문장은 Root 노드부터 시작되며 트리 구조로 나타난다. S는 Sentence로 하나의 문장을 의미하고, 이들이 각각 NP (Noun Phase), VP (Verb Phase)로 나뉜다. 문장의 구는 세분화되어 형태소로 나뉜다. 따라서 품사 태깅을 통해 각 단어와 품사를 분류할 수 있다. 문장을 가장 작은 단위인 형태

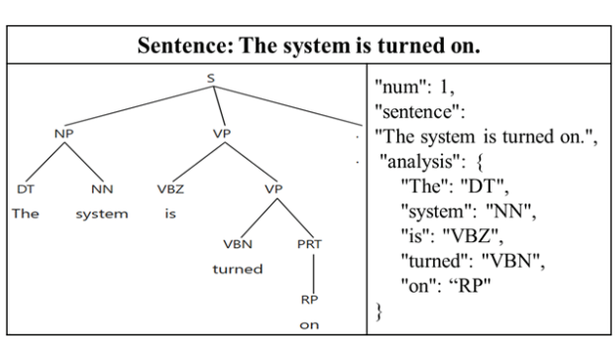

Fig. 4. Sentence Analysis using Chomsky's Theory

소로 나누고 품사 태그를 통해 핵심 형태소인 명사, 동사, 형용사를 구분한다. 그리고, 분석된 단어와 품사를 하나의 쌍으로 연결하여 JSON 형식으로 저장한다.

4) 문장 의미 분석

촘스키의 이론으로 문장의 구조를 파악하지만, 의미를 알 수는 없다. 따라서, 필모어의 Semantic Roles 이론으로 형태소의 품사에 따라 문장의 의미를 분석한다. 필모어의 이론은 동사를 기준으로 각 명사의 관계를 분석한다[6]. 따라서, 우리는 품사 태깅을 바탕으로 VB로 시작하면 동사, NN으로 시작하면 명사로 분류한다. Table 1은 필모어의 이론의 일부를 보여준다. 필모어의 sematic role 이론은 많은 도메인에서 각 특징에 맞게 재정의되어 사용된다. 따라서 우리는 재정의한 필모어의 역할론을 바탕으로 명사에 역할을 부여한다. 재정의한 필모어의 역할론은 Table 2와 같다.

우리는 총 6개의 의미 역할을 사용한다. 또한, 상황에 따라 Source와 Target은 Actor와 Element와 중복될 수 있다. 하나의 객체가 이벤트의 주체이자, 행동을 수행하는 주체가 된다면, 이는 Actor이자 Source가 된다. 또한, Source는 Action을 받는 Target과 짝을 이루어야 한다.

3.2 요구사항 데이터 기반 UML 설계 생성

요구사항의 분석 결과로 의미를 가진 역할들과 동사를 받는다. 본 연구는 Fig. 5와 같이 해당 역할과 동사로부터 UML 설계를 추출하기 위해 각 요소 간의 매핑을 수행한다.

UML 설계로는 정적인 구조를 나타낼 수 있는 클래스 다이어그램, 객체 간의 상호작용을 확인할 수 있는 시퀀스 다이어그램, 그리고 객체의 상태 변화를 확인할 수 있는 상태 다이어그램을 생성한다.

1) 문장 분석 결과 중복성 제거

요구사항의 중복성은 설계 및 코드의 중복으로 이어진다. 중복 코드는 코드의 길이가 늘어나고, 복잡성을 증가시킨다. 따라서, 개발 초기에 중복성을 없애는 것은 매우 중요하다. 자연어 요구사항 분석 시, 중복된 명사와 동사를 식별하여 단일 개체로 처리하는 과정이 필요하다. 우리는 문장마다 의미 분석을 진행하여 Role을 부여하기 때문에 하나의 역할에 대해 중복되는 문장 데이터가 저장된다. 예를 들어, Actor가 중복으로 저장되면, 같은 Class가 두 개 발생한다. 따라서, 중복되는 데이터를 제거하고 다이어그램 생성에 필요한 JSON 파일로 저장한다.

2) UML 설계 요소와 요구사항 분석 결과 매핑

중복성이 제거된 역할과 생성될 UML 설계는 3가지이다. 각 다이어그램에 필요한 요소를 정의하고, 이를 Table 3과 같이 규칙 기반으로 요구사항 결과와 매핑한다.

매핑은 소프트웨어 설계 과정에서 요구사항의 구조적, 행위적, 상태적 정보를 일관성 있게 반영하기 위해 설정된다. Actor는 요구사항 내의 행위를 수행하거나 트리거하는 역할이다. 클래스 다이어그램의 Class와 매핑되며 시퀀스 다이어

Table 1. Part of Original Fillmore's Semantic Roles

Roles	Definition
Agent	The doer or instigator of the action denoted by the predicate
Experiencer	The living entity that experiences the action or event denoted by the predicate
Source	The location or entity from which something moves
Instrument	The medium by which the action or event denoted by the predicate is carried out

Table 2. Refined Fillmore's Semantic Roles

Roles	Definition
Actor	The entity that is the main subject of the event.
Object	The entity affected by the event.
Element	The entity that indicates a property of the actor.
Instrument	The tool used to perform the action
Source	The entity performing the action.
Target	The entity receiving the action.

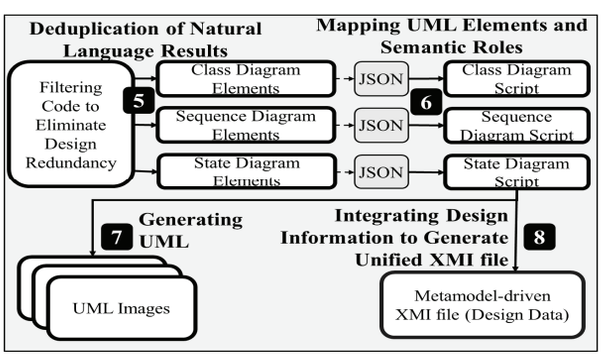

Fig. 5. Procedure for Generating UML Design based on Requirements Data

Table 3. Mapping with Semantic Roles and UML Elements

Roles	Class Diagram	Sequence Diagram	State Diagram
Actor	Class	Sending Object	-
Object		Receiving Object	Object
Element	Attribute	-	-
Instrument	Function's Parameter	Message's Parameter	Event's Parameter
Source	-	Sending Object	-
Target	-	Receiving Object	-
Dynamic Verb	Function	Message	Event
Adjective	-	-	State

그램에서 메시지 송신 객체로 표현된다. Object는 시스템 요소로 클래스 다이어그램의 클래스, 시퀀스 다이어그램에서는 수신을 받는 객체로 매핑된다. Element는 객체의 내부 속성을 의미하며, 클래스 다이어그램에서 속성으로 매핑된다. Instrument는 행위에 사용되는 도구나 수단을 의미하여 함수, 메시지, 이벤트의 매개변수로 표현된다. Source와 Target은 행위의 방향성을 나타내며, 시퀀스 다이어그램의 메시지를 송수신하는 객체로 표현된다. Dynamic Verb는 동사의 한 종류로서 행위를 의미하며 함수, 메시지, 이벤트로 매핑된다. 이는 요구사항의 동작 요소를 나타낸다. 마지막으로 Adjective는 상태적 성질을 의미하며 Object, 즉 시스템이 가질 수 있는 상태를 표현한다.

3) UML 설계 이미지 생성

매핑되는 데이터는 텍스트 형태로 저장되며, UML 생성을 위한 스크립트 형태로 구성할 수 있다. 다이어그램 생성을 위해 PlantUML의 문법을 사용한다. Fig. 6와 같이 매핑된 요구사항 데이터를 바탕으로 다이어그램 생성 스크립트를 정의하고, 이미지로 생성된다.

4) 메타모델 생성 및 데이터 저장

재정의된 UML 설계 메타모델을 바탕으로 요구사항 결과와 매핑된 다이어그램 요소를 저장한다. 메타모델 관련 연구에서는 세 가지 UML 설계에 고유한 메타모델이 존재한다[7]. 따라서, 각 설계 메타모델에 대해 별도로 데이터를 생성하고 입력한 뒤 변환해야 하는 번거로운 절차가 요구되었다. 우리는 이러한 과정을 단순화하기 위해, 모든 정보를 하나의 데이터 파일로 병합하여 정보를 저장하고, 이를 지원하는 통합 설계 메타모델을 설계하였다. Fig. 7은 Class, Sequence, State Digram을 통합적으로 표현할 수 있는 설계 메타모델을 보여준다.

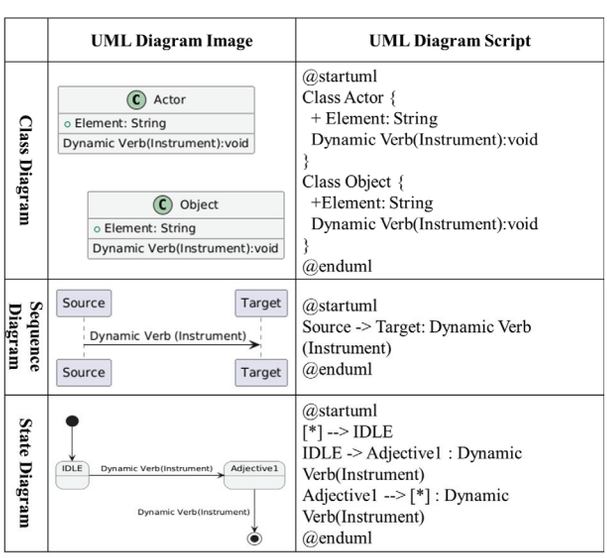

Fig. 6. Generating UML Image from UML Script

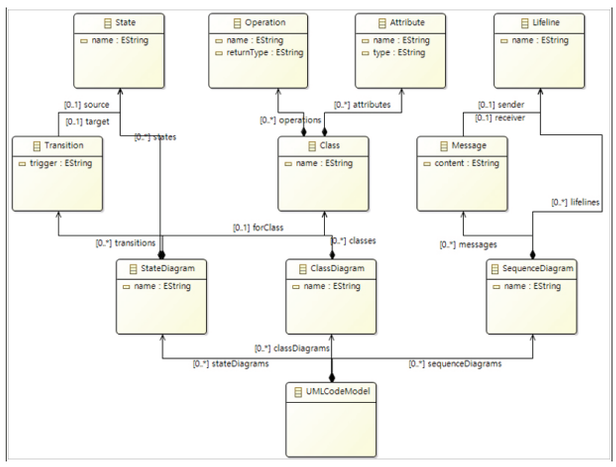

Fig. 7. UML Unified Design Metamodel

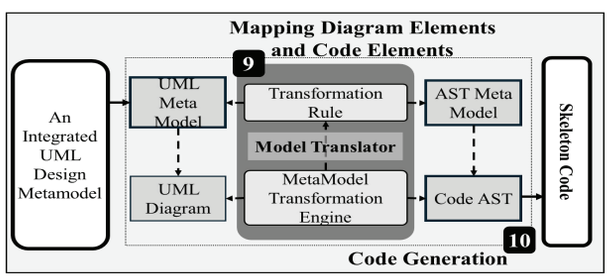

Fig. 8. Generating Code using Model Translator

통합하여 재정의된 하나의 메타모델은 메타모델 변환 규칙을 사용하여 모델 간의 의미적 연결 유지가 가능하다. 설계 메타모델의 최상위 루트노드는 'UMLDesignModel'이다. 각각의 UML 설계의 상위 노드들이 상속될 수 있도록 한다. UML 메타모델 구조를 기반으로 XMI 형식의 ecore 파일로 저장한다.

3.3 메타모델링 기법을 통한 스켈레톤 코드 생성

다이어그램 설계가 생성되면, 이를 바탕으로 실행이 가능한 스켈레톤 코드를 생성한다. Fig. 8은 메타모델링 기법을 통해 설계데이터로부터 코드를 생성하는 과정을 보여준다.

1) 룰 기반의 메타모델 데이터 트렌지션을 위한 엔진

본 연구에서는 메타 모델링을 지원하는 eclipse를 기반으로 Java 코드를 생성한다. Java 소스 코드는 컴파일 과정에서 AST(Abstract Syntax Tree) 형태로 내부가 표현되며, 이는 프로그램의 문법적 구조를 계층적으로 나타내는 트리이다. 본 연구에서는 JavaParser 도구를 활용하여 Java 코드를 AST 형태로 파싱하고, 이를 기반으로 코드 메타 모델을 정의한다[8]. Fig. 9는 코드 메타모델을 보여준다.

메타모델은 클래스, 변수, 메서드, 파라미터와 같은 핵심 구성 요소를 포함한다. 각 노드는 속성과 관계를 가지며, 전체 소스 코드를 구조적으로 표현할 수 있도록 설계되었다. 코드 메타모델은 Ecore 기반으로 정의되어있다. AST 구조에

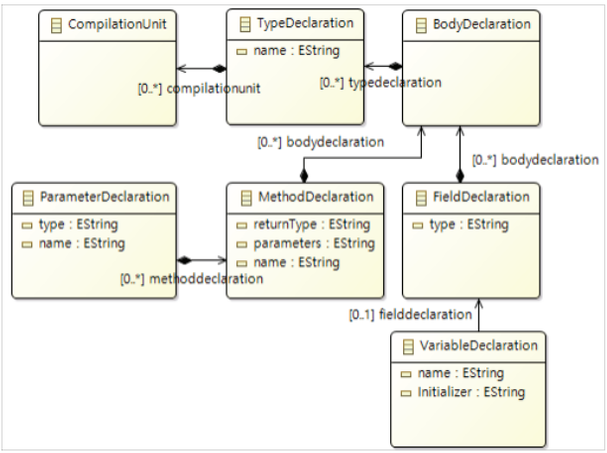

Fig. 9. Code Metamodel

서 클래스는 TypeDeclaration, 변수는 FieldDeclaration의 VariableDeclaration, 메서드는 MethodDeclaration에 포함된다. 트리 구조에 따라 1:M 관계로 이루어져 있으며, FieldDeclaration의 경우에만 1:1 관계를 갖는다.

2) 모델 변환 엔진을 통해 스켈레톤 코드 생성

본 연구는 ATL을 사용하지 않고, 사용자 정의 규칙 기반의 알고리즘을 통해 UML 설계 메타모델과 Java 코드 메타모델 간의 데이터 변환을 수행한다. 변환 알고리즘은 규칙에 따라 UML 요소를 AST 구조로 변환하며, 각 요소의 속성과 관계를 탐색하여 대상 모델의 노드를 구성한다. UML의 Class는 AST Code의 TypeDeclaration으로 변환되어 클래스로 변환한다. 특히, UML Class 내의 Attribute는 Java FieldDeclaration으로, Operation은 MethodDeclaration으로 상세화된다. 각 FieldDeclaration은 VariableDeclaration을 포함하여 실제 변수명과 초기화 정보를 표현하고, MethodDeclaration은 ParameterDeclaration을 통해 메서드 파라미터를 표현한다. 또한, UML의 행동 모델링 요소인 Sequence와 StateMachine은 직접적인 코드 요소로 매핑되기보다는 MethodDeclaration 내의 복합적인 조건문, 메서드 호출 등 Java AST의 Statement 노드를 통해 구현 논리로 변환된다. 이러한 매핑 규칙에 따라 Java 추상 구문 트리(AST)의 메타모델을 기반으로 xmi 데이터 파일이 생성된다. 최종적으로 UML의 xmi 데이터로부터 코드 xmi 데이터가 발생하며 스켈레톤 Java 소스 코드로 변환된다.

4. 사례 연구

본 연구는 전통적인 소프트웨어 개발 프로세스에 AI 기법을 단계별로 접목하여 요구사항 분석부터 코드 생성까지의 과정을 자동화하고자 한다. 엘리베이터 시스템로 이를 설명한다. 우선, SRS 문서로부터 기능적 요구사항을 추출한다. 해당 자연어 요구사항들은 문장 전처리, 구조 분석, 의미 분석 과정을 거친다. 분석 단계의 결과를 JSON 형식으로 저장한다. 문장 분석의 마지막 절차인 의미 분석 단계에서는 동일한 역할을 수행하는 요소를 통합하여 중복성을 줄인다. 이후, 정제된 의미 정보들을 기반으로 다이어그램 데이터가 생성되며, 해당 데이터를 활용하여 PlantUML 스크립트를 자동 작성한다. 이는 UML 다이어그램으로 시각화된다. 또한, 다이어그램의 정보는 사전에 정의된 Ecore기반 메타 모델 구조에 따라 XMI 형식으로 변환된다. XMI 파일은 모델 변환 도구를 통해 코드 생성을 수행한다.

Step 1. SRS 문서로부터 기능적 요구사항 분류

개발자는 자연어 요구사항에서 소프트웨어 요구사항 명세서를 읽고 분류 학습을 통해 기능적 요구사항를 분류한다.

Step 2. 요구사항 문장 전처리

기능적 요구사항을 문장 단위로 입력받아 문장을 단문화한다. 문장 전처리 자동화를 위해 GPT의 Few-shot 프롬프팅을 사용한다. Fig. 10의 FR-001이 중문, FR-002가 복문이다. 이를 각각 전처리하는 모습을 보여준다.

Step 3. 문장 구조 및 형태소 분석

단문화된 문장은 문장의 형태소로 나뉜다. 핵심 형태소인 명사(NN)과 동사(VB)를 구분하기 위해 JSON 형식으로 저장한다. 본 연구에서는 문장 구조 분석을 자동화하기 위해 NLTK (Natural Language Toolkit) 라이브러리를 사용한다. Fig. 11은 "The users presess the button."이라는 문장의 분석 결과를 JSON 형식으로 저장하는 것을 보여준다.

Step 4. 문장 의미 분석

분석된 주요 명사와 동사는 user, button, press이다. press를 기준으로 문장의 의미를 분석하면 Fig. 12와 같다.

Step 5. 문장 분석 결과 중복성 제거

문장마다 명사에 대해 의미역을 부여하기 때문에 Fig. 13과 같이 중복되는 결과를 갖는다. 따라서, UML을 생성하기 전에

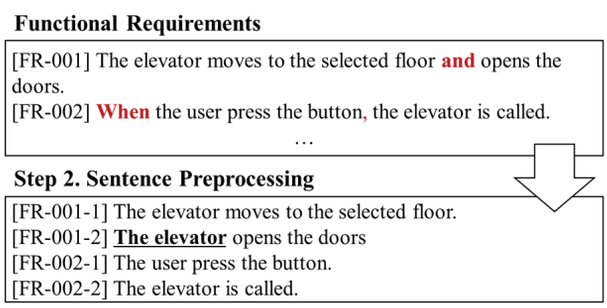

Fig. 10. Sentence Preprocessing

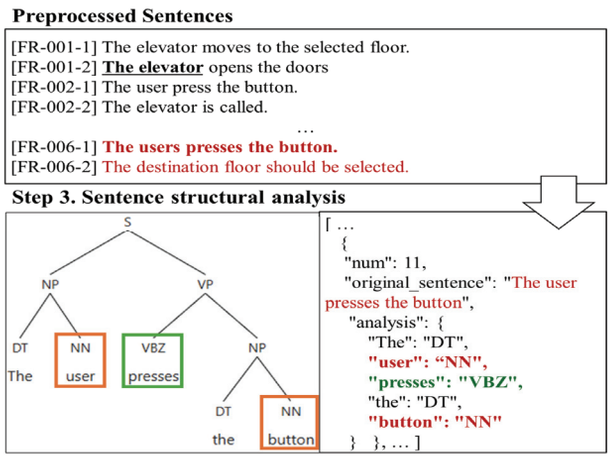

Fig. 11. Sentence Structural Analysis

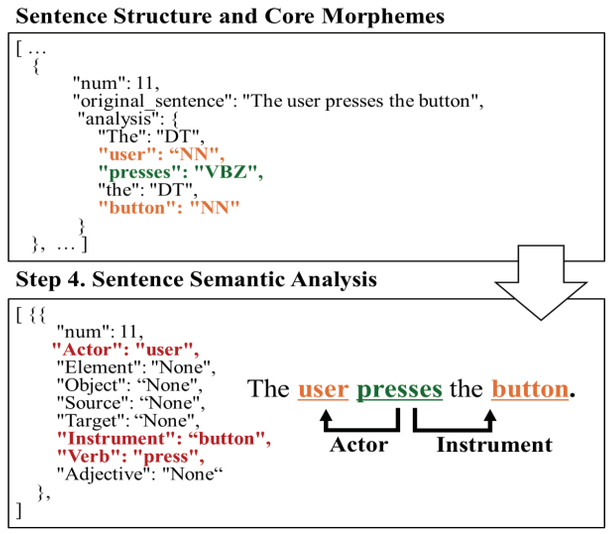

Fig. 12. Sentence Semantic Analysis

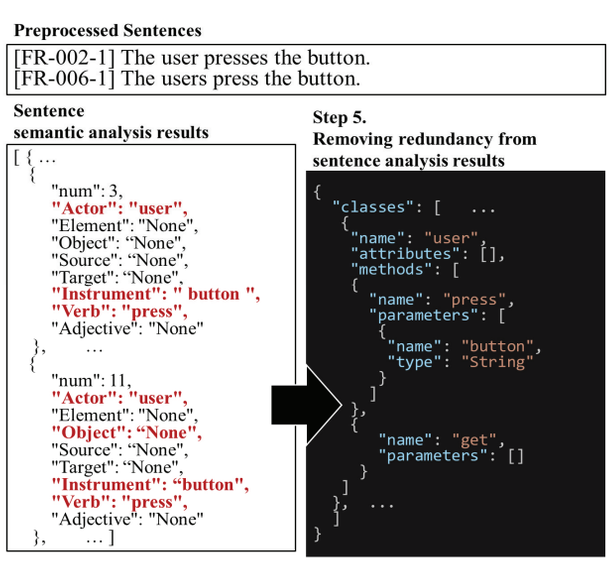

Fig. 13. Removing Redundancy from Sentence Analysis Results

데이터를 정제해야 한다. UML 설계에 필요한 요소들로 JSON 파일 형식을 정한다. 중복되는 데이터를 제거하기 위해 set 형태로 저장 후 JSON에 입력되도록 한다. 이를 통해 중복되는 요구사항이 있더라도, 중복을 제거할 수 있다.

Step 6. UML 설계 요소와 요구사항 분석 결과 매핑

중복이 제거된 결과는 각각 UML 요소와 매핑된다. Fig. 14는 시퀀스 다이어그램에 대한 매핑을 보여준다.

Step 7. UML 설계 이미지 생성

모든 데이터에 대한 매핑이 완료되면, PlantUML 스크립트 문법에 맞춰 다이어그램을 생성한다. Fig. 15는 다이어그램 스크립트와 이미지를 보여준다.

Step 8. 메타모델 생성 및 데이터 저장

자연어로부터 추출한 모델 요소를 표현하기 위해 Ecore 기

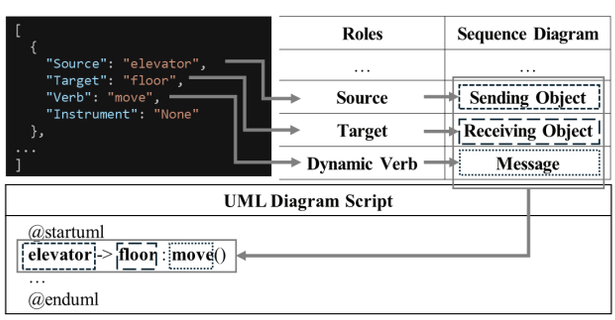

Fig. 14. Mapping UML and Requirements Analysis Results

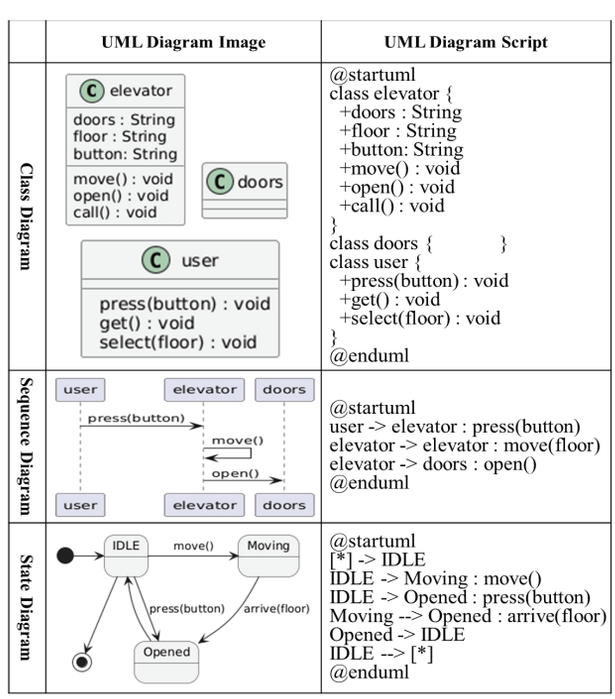

Fig. 15. Generating UML Images

반의 메타모델을 설계한다. Fig. 16는 UML 다이어그램에 대한 메타모델로, Class, Sequence, StateMachine의 세 가지 구성으로 나뉜다. Model 클래스는 시스템 전체를 의미하며, 여러 개의 클래스 다이어그램, 시퀀스 다이어그램, 상태 다이어그램을 포함한다. Class 요소는 이름, 속성(Attribute), 연산(Operation)으로 구성된다. Sequence 요소는 Lifeline과 Message로 구성된다. 또한, StateMachine 요소는 상태(State)와 전이(Transition)로 구성된다.

Step 9. 룰 기반 메타모델 데이터 변환을 위한 엔진

메타모델은 EMF(Eclipse Modeling Framework) 기반 도구들과 호환된다. 우리는 설계로부터 코드를 생성하는 모델 변환을 위해 코드 메타모델을 생성한다. 그 구조는 Fig. 17과 같다.

코드로 생성하기 위해 우리는 코드 메타모델과 UML 메타 모델의 요소들의 변환 규칙 알고리즘을 정의하였다. CompilationUnit을 최상위 요소로 하여 단일 Java 소스 파

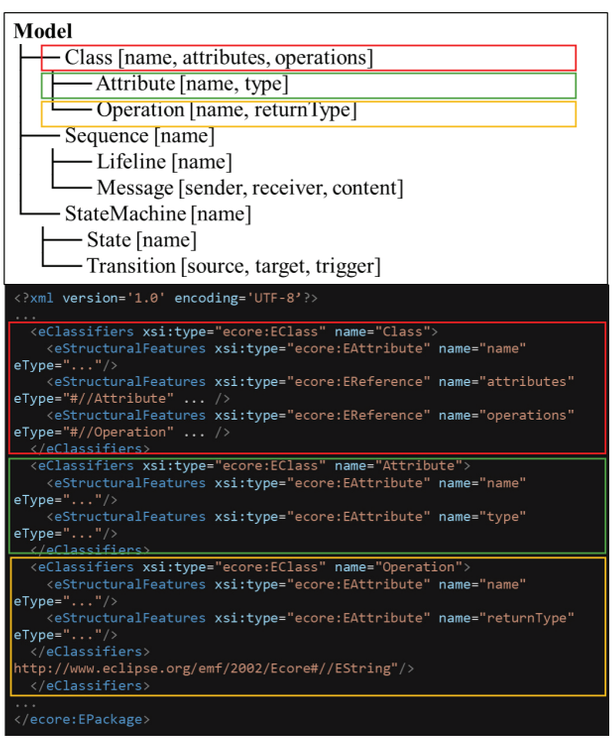

Fig. 16. Generating Metamodel and Saving Data

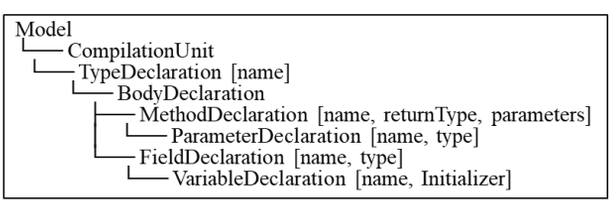

Fig. 17. Code Metamodel Structure

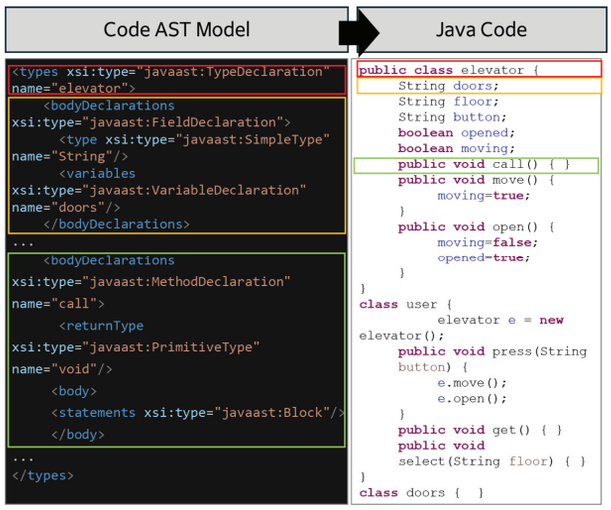

Fig. 18. Generating Java Code from Code AST Model

일의 구조를 나타내며, 그 안에 TypeDeclaration으로 표현된 여러 클래스(elevator, user, doors)를 포함한다. 각 TypeDeclaration은 BodyDeclaration을 통해 계층적으로 관리하며, 이는 다시 MethodDeclaration과 FieldDeclaration으로 구체화된다. MethodDeclaration은 메서드 이름, 반환 타입, 그리고 ParameterDeclaration을 포함하여 정의한다. 그리고, FieldDeclaration은 필드의 타입과 함께 Variable Declaration을 포함하여 실제 변수 이름과 초기화 식을 표현한다.

Step 10. 모델 변환 엔진을 통해 스켈레톤 코드 생성

정의된 Fig. 18은 변환 규칙을 통해 생성된 xmi 데이터 파일과 Java 코드 결과를 보여준다. UML과 Java AST 코드의 데이터 변환을 통해 실행 가능한 스켈레톤을 생성할 수 있다.

5. 결론

본 연구는 인공지능과 소프트웨어 공학을 통합하여 코드를 자동 생성하는 방안을 탐색한다. 특히 소프트웨어 개발 생명주기의 주요 단계인 요구사항 분석부터 설계, 그리고 코드 생성까지의 과정에 AI 기술을 부분적으로 접목하는 방법론을 제시한다. 요구사항 분석 단계에서 AI를 활용하여 요구사항의 중복성을 제거한다. 이를 기반으로 설계를 생성한다. UML 설계는 코드를 생성하기 위한 데이터가 저장된다. 우리는 메타모델링을 통해 UML 설계로부터 코드 모델 변환을 수행한다. 이를 통해 코드 생성를 생성한다. 기존 연구에서는 스켈레톤 코드 생성을 위한 메타모델 자체에 대한 언급은 있었으나, 그 구조와 구체적인 작동 방식에 대한 상세한 설명이 부족했다. 본 연구는 이를 해결하기 위해, 기존 연구를 발전시켜 '메타모델 변환 엔진(Metamodel Transformation Engine)'의 구조와 구체적인 변환 알고리즘을 제안한다. 이는 누구나 동일한 변

환 규칙을 적용할 수 있는 표준화된 프레임워크를 제공하며, 개발 프로세스 자동화를 위한 규칙 기반 엔진의 가능성을 제시한다.

향후 연구는 본 연구에서 제시된 AI 접목 방안을 소프트웨어 개발의 모든 단계로 확장하는 것을 목표로 한다. 이를 통해 다음과 같은 세 가지 유형의 개발 방식을 비교 분석할 계획이다: 1) 인간 기반의 코드는 기존의 개발자 중심 수동 코딩 방식이다. 2) 부분적 AI 적용 코드는 본 연구에서 제안한 바와 같이, SDLC의 특정 단계에 AI가 부분적으로 적용하는 방식이다. 3) 완전히 AI 기반 코드는 전반적인 개발 과정이 AI에 의해 주도되는 방식이다. 이러한 비교 분석을 통해 각 개발 방식의 장단점을 명확히 식별하며, 고품질의 소프트웨어를 개발하기 위한 최적의 방법론을 검증하고자 한다. 소프트웨어 개발의 효율성과 품질을 동시에 극대화할 수 있는 새로운 AI 기반 개발 패러다임을 정립하는 데 기여할 것으로 기대한다.

References

[1] Stack Overflow, AI: 2024 Stack Overflow Developer Survey [Internet], https://survey.stackoverflow.co/2024/ai.

[2] Y. J. Jin, K. D. Kim, D. Y. Yoo, and R. Y. C. Kim, "Adopting generative AI in each phase of software life cycle for software development approach," in *Proceedings of the Annual Conference of the Korea Information Processing Society (KIPS)*, Vol.32, No.1, pp.401-404, 2025.

[3] W. Y. Kim, H. S. Son, Y. B. Park, B. H. Park, C. R. Carlson, and R. Y. C. Kim, "Automatic MDA (Model-driven architecture) transformations for heterogeneous embedded systems," in *Proceedings of the SERP 2008 Conference*, Vol.2, pp.409-414, 2008.

[4] B. Park, B. Jeon, R. Y. C. Kim, and J. H. Lee, "Metamodeling for automatic test case generation," *International Information Institute Information*, Vol.18, No.6, 2015.

[5] N. Chomsky, "Syntactic structures," USA: Mouton de Gruyter, 2002.

[6] C. J. Fillmore, "The case for case," New York: HR&W, 1968.

[7] H. S. Son, W. Y. Kim, R. Y. Kim, and H. G. Min, "Metamodel design for model transformation from Simulink to ECML in cyber physical systems," in *Proceedings of the International Conference on Computer Graphics, Animation and Games*, pp.56-60, 2012.

[8] Eclipse Foundation, ASTParser [Internet], https://help.eclipse.org/latest/index.jsp?topic=/org.eclipse.jdt.doc.isv/reference/api/org/eclipse/jdt/core/dom/ASTParser.html.

진 예 진

https://orcid.org/0009-0000-4996-2436
e-mail : yejin_jin@g.hongik.ac.kr
2023년 홍익대학교 소프트웨어융합학과 (학사)
2025년 홍익대학교 소프트웨어융합학과 (석사)
2025년～현 재 홍익대학교 소프트웨어융합학과 (박사)
관심분야 : Software Engineering, Natural Language-based Code Generation, Text_to_Emotion Recognition

김 장 환

https://orcid.org/0000-0003-0185-4406
e-mail : lentoconstante@hongik.ac.kr
2019년 아이다호주립대학교 컴퓨터과학과 (학사)
2022년 홍익대학교 소프트웨어융합학과 (석사)
2022년～현 재 홍익대학교 소프트웨어융합학과 박사
관심분야 : AI Software Testing, Software Engineering, Reinforcement Learning Software Validation

김 영 철

https://orcid.org/0000-0002-2147-5713
e-mail : bob@hongik.ac.kr
2000년 일리노이공대(IIT) 소프트웨어공학과 (박사)
2001년～현 재 홍익대학교 소프트웨어융합학과 교수
관심분야 : Scenario Based Validation for AI Reinforcement Learning, Metaverse(VR/AR), SE