JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 26, No. 1, pp. 183-193, Jan. 2025

Check for updates

LLM과 3D UI를 적용한 Polylith 소프트웨어 개발 방법 개선 방안 연구

이 주 경[1] · 김 태 훈[2*]
[1]서강대학교 메타버스전문대학원 메타버스 비즈니스 박사과정
[2]서강대학교 메타버스전문대학원 교수

Research the Improvement of Polylith Software Development Method Using LLM and 3D UI

Jookyoung Lee[1] · Taehoon Kim[2*]

[1]Postgraduate Ph.D. program, Sogang University Graduate School of Metaverse, Seoul 04107, Korea
[2]Professor, Sogang University Graduate School of Metaverse, Seoul 04107, Korea

[요 약]

본 연구는 Polylith 소프트웨어 개발 아키텍처를 대규모 언어 모델(LLM)과 3D 사용자 인터페이스(3D UI)를 통합하여 개선하는 방안을 탐구한다. Polylith 아키텍처는 모듈화 및 컴포넌트 재사용성을 증진하기 위해 설계되었으며, 여기에 최신 AI 기능과 상호작용 시각화를 접목하였다. LLM을 활용하여 자동 코드 생성, 리팩토링 제안, 효율적인 코드 문서화 등을 구현함으로써 복잡한 시스템 개발의 과제를 해결한다. 동시에 3D UI를 통해 복잡한 시스템 구조를 시각적으로 표현하여 개발자가 컴포넌트 간의 의존성과 상호작용을 더 쉽게 이해할 수 있도록 지원한다. 제안된 접근 방식은 개발자 간 협업을 강화하고, 소프트웨어 유지보수성을 향상시키며, 대규모 프로젝트의 확장성을 지원하는 것을 목표로 한다. 3D UI는 직관적인 시스템 탐색과 상호작용을 가능하게 하며, LLM 통합은 최적화된 성능과 의사결정을 보장한다. 이 혁신적인 프레임워크는 Polylith 아키텍처를 현대 개발 요구사항에 맞게 재정의함과 동시에 모듈형 및 확장 가능한 애플리케이션에 적합한 원활하고 몰입감 있는 효율적인 소프트웨어 엔지니어링 환경을 구축한다.

[Abstract]

This study investigates the enhancement of the Polylith software development architecture through the integration of Large Language Models (LLMs) and 3D User Interfaces (3D UIs). Originally designed to improve modularity and component reuse, the Polylith architecture is extended with state-of-the-art AI capabilities and interactive visualization. By employing LLMs, the study introduces automated code generation, refactoring recommendations, and efficient code documentation, addressing challenges in managing complex system development. Simultaneously, 3D UIs are incorporated to visually represent intricate system structures, aiding developers in comprehending dependencies and interactions across components.

The proposed approach aims to foster collaboration among developers, enhance software maintainability, and support scalability for large-scale projects. The 3D UI facilitates intuitive navigation and interaction with system components, while the LLM integration ensures optimized performance and informed decision-making. This innovative framework not only redefines the Polylith architecture to meet contemporary development demands but also establishes a seamless, immersive, and efficient software engineering environment suitable for modular and scalable applications.

색인어 : Polylith 아키텍처, LLM, 3D UI, 모듈형 소프트웨어, 협업 효율성

Keyword : Polylith Architecture, LLM, 3D UI, Modular Software, Collaboration Efficiency

http://dx.doi.org/10.9728/dcs.2025.26.1.183

Received 04 November 2024; **Revised** 11 December 2024
Accepted 27 December 2024

***Corresponding Author; Taehoon Kim**

Tel: +82-2-705-8065
E-mail: taehoonkim@sogang.ac.kr

Ⅰ. 서 론

1-1 Polylith 아키텍처 개요

Polylith 아키텍처는 1985년에 소개된 소프트웨어 개발 방법론으로, 모듈식 설계와 컴포넌트 재사용성을 강조했다. 이 아키텍처는 다양한 도구와 프로그램을 안전하게 통합하고, 데이터 교환 및 동기화 문제를 해결하는 것을 목표로 한다[1]. Polylith의 핵심 개념은 '소프트웨어 버스'라는 추상적인 분리 계층을 도입하여 시스템 컴포넌트 간의 상호작용을 관리하는 것이다[2].

Polylith 아키텍처는 개발자가 기능적 요구사항을 인터페이스 요구사항과 분리하여 구현할 수 있게 해준다. 이를 통해 한 실행 환경(예: 분산 네트워크)에서 개발된 애플리케이션을 다른 환경(예: 공유 메모리 멀티프로세서)에서 자동화된 기술을 통해 쉽게 재사용할 수 있다[2]. Polylith 시스템은 성능 저하 없이 이러한 유연성을 제공하는데 향상된 실행 시간 시스템 서비스, 개발 도구, 그리고 인터페이스 방법론으로 구성된다. 이 시스템은 환경 인스턴스를 정확하고 신속하게 지정할 수 있는 문법으로 표현되며, 이 언어로 작성된 어서션의 컴파일과 실행을 통해 다양한 프로그램과 도구를 안전하게 통합할 수 있다[1].

개발자는 모듈 상호 연결 언어(MIL)를 사용하여 애플리케이션 구조를 지정하고, Polylith는 이 명세를 사용하여 패키징(스텁 생성, 소스 프로그램 적응, 컴파일, 링킹 등의 정적 인터페이싱 활동)을 안내한다. 런타임에는 소프트웨어 버스 추상화 구현이 메시지 전달, 이름 서비스, 시스템 재구성 등을 지원할 수 있다[2]. 본 연구의 배경은 그동안 폭 넓게 활용되지 못했던 Polylith 아키텍처에 LLM과 3D UI를 적용하여 전문가부터 초보자까지 쉽게 작업을 할 수 있는 여건을 마련할 수 있도록 하기 위함이다.

1-2 Polylith 아키텍처가 확산되지 못한 이유

Polylith는 기존 아키텍처 패턴에 익숙한 개발자들에게 복잡하고 학습 난이도가 높은 개념으로 인식되어 직관적으로 이해하는 데 어려움을 주었으며, Polylith가 제공하는 이점 또한 충분히 설득력 있게 전달되지 않아 많은 개발자들이 기존 방식으로도 유사한 결과를 얻을 수 있다고 판단하게 되었다. 이러한 인식의 문제 외에도, 명확한 설명과 실제 적용 사례의 부족으로 인해 Polylith의 실제 활용 방법을 이해하기 어려웠으며, 많은 개발 환경과 도구들이 Polylith 아키텍처를 지원하지 않음에 따라 기존 워크플로우와의 통합도 쉽지 않았다. 게다가 일부 개발자들은 Polylith의 추상화 계층이 성능 저하를 초래할 수 있다는 우려를 표명하기도 했다. 이러한 요인들로 인해 Polylith 아키텍처는 초기에 기대했던 만큼 폭 넓게 채택되지 못했다[3].

1-3 LLM과 3D UI를 활용한 Polylith 아키텍처의 발전 필요성

Polylith 아키텍처는 모듈화와 재사용성을 극대화하는 소프트웨어 개발 방식으로, 컴포넌트, 베이스, 프로젝트라는 구조화된 요소를 사용하여 복잡한 시스템 개발에 적합한 접근 방식을 제공한다. 그러나 현대의 소프트웨어 개발 환경에서는 인공지능과 사용자 경험의 중요성이 날로 증가하고 있어, Polylith 아키텍처에 대규모 언어 모델(LLM)과 3D 사용자 인터페이스(UI)를 적용하는 것이 필요하다. LLM을 Polylith 아키텍처에 결합함으로써, 코드 생성 및 자동화, 인터페이스 설계 지원, 문서화 개선, 리팩토링 제안 등의 이점을 얻을 수 있다. 특히, LLM의 프롬프트 체이닝 기능을 활용하면 복잡한 태스크 수행이 가능한 LLM 앱 개발을 효과적으로 지원할 수 있다[4]. 또한, LLM을 통해 Polylith의 각 컴포넌트에 대한 단위 테스트 및 통합 테스트 케이스를 자동으로 생성할 수 있어, 개발 프로세스의 효율성과 코드 품질을 크게 향상시킨다.

3D UI를 Polylith 아키텍처에 통합하는 것은 사용자 경험을 혁신적으로 개선하고, 복잡한 시스템 구조를 직관적으로 시각화하는 데 큰 도움이 될 수 있다. 3D UI는 Polylith의 컴포넌트, 베이스, 프로젝트 간의 관계를 공간적으로 표현함으로써, 개발자들이 시스템 구조를 더 쉽게 이해하고 관리할 수 있게 한다. 특히 대규모 프로젝트에서 시스템의 복잡성을 관리하는 데 큰 이점을 제공할 수 있다. 또한, 3D UI를 통해 사용자들은 Polylith 구조 내에서 직접 컴포넌트를 조작하고 연결할 수 있어, 프로토타이핑과 시스템 설계 과정을 더욱 직관적이고 효율적으로 만들 수 있다.

LLM과 3D UI의 결합은 Polylith 아키텍처의 강점을 더욱 강화하고, 개발자 경험을 크게 개선할 수 있다. LLM이 제안한 코드나 구조 변경 사항을 3D UI를 통해 즉시 시각화하고 검토할 수 있게 되어, 의사결정 과정이 더욱 빨라지고 정확해질 수 있다. 이러한 통합은 Polylith 아키텍처를 현대적인 소프트웨어 개발 요구사항에 더욱 적합하게 만들어, 복잡한 시스템 개발에서의 생산성과 품질을 크게 향상시킬 수 있는 잠재력을 가지고 있다[5].

1-4 연구범위 및 목표

이 연구의 범위는 Polylith 아키텍처의 확장성과 발전 가능성을 탐구하는 데 중점을 두었다. 기존 Polylith 아키텍처가 백엔드 중심의 모듈형 설계에 국한되었던 한계를 극복하고, LLM과 3D UI 기술을 통합함으로써 프론트엔드까지 확장하는 방향을 제안한다. 이를 통해 더 나은 사용자 경험과 개발 효율성을 달성하고, 전체 소프트웨어 시스템의 모듈화 및 유지보수성을 크게 향상시키고자 한다. 연구의 주요 목표는 다음과 같다. 첫째, LLM을 활용한 코드 자동 생성 및 테스트 최적화 방법을 탐구하여 개발 속도를 높이는 방안을 제시하는 것이다. 둘째, 3D UI를 도입함으로써 개발자들이 시스템의 복잡성을 보다 직관적으로 이해하고, 시스템 설계를 시각적으

로 관리할 수 있도록 돕는 방법을 연구한다. 셋째, LLM과 3D UI의 통합이 가져올 수 있는 실제적인 개발 생산성 및 협업 증진 효과를 예측하는 것이다. 이 연구를 통해 Polylith 아키텍처의 현대적 재해석과 그 잠재적 활용 가능성을 규명하고, 복잡한 소프트웨어 시스템 개발에 있어 새로운 패러다임을 제시하는 것을 목표로 한다.

II. 본 론

2-1 문헌연구

1) Polylith[6]

Polylith은 소프트웨어 아키텍처로서 기능적 사고를 시스템 규모로 적용하여 단순하고 유지보수 가능하며, 테스트와 확장이 용이한 백엔드 시스템을 구축하는 것을 목표로 한다. 기존의 모놀리식(Monolith) 혹은 마이크로서비스(Microservices) 아키텍처가 지닌 한계들을 해결하고자, Polylith는 작은 모듈형 단위로 시스템을 구성하는 "모듈형 모놀리식(modular monolith)"이라는 접근 방식을 채택하고 있다. 이러한 방식은 시스템의 복잡성을 줄이고 코드 재사용성을 극대화하며, 개발자에게 효율적인 작업 환경을 제공한다. Polylith는 독립적인 기능 블록을 조합하여 시스템을 구성함으로써 복잡한 시스템을 효과적으로 관리할 수 있다.

Polylith 아키텍처의 핵심 구성 요소로는 컴포넌트(Component), 베이스(Base), 프로젝트(Project)가 있다. 컴포넌트는 시스템의 주요 기능을 구현하는 고수준의 모듈로, 독립적으로 존재하면서 다른 컴포넌트나 기반에서 호출될 수 있다. 예를 들어 사용자 관리, 주문 처리와 같은 도메인 로직을 컴포넌트로 정의할 수 있다. 베이스는 외부 세계와의 상호작용을 담당하는 모듈로, REST API나 명령줄 인터페이스 등의 형태로 구현되어 있다. 이러한 기반은 직접적인 비즈니스 로직을 포함하지 않으며, 모든 로직을 컴포넌트에 위임한다.

마지막으로 프로젝트는 컴포넌트와 기반을 조합하여 특정 기능을 배포 가능한 형태로 만드는 단위이다. Polylith는 시스템을 작은 단위로 나누어 개발하면서도, 필요에 따라 전체를 하나로 묶어 배포할 수 있는 유연성을 제공한다.

Polylith 아키텍처에서 중요한 특징 중 하나는 모든 코드를 한 곳에서 관리하는 모노레포(monorepo) 전략으로 코드의 동기화를 쉽고 일관성 있게 유지할 수 있도록 도와준다. 각 컴포넌트는 단일 코드베이스 내에서 작업되며, 모든 팀원이 동일한 코드를 참조하게 되어 코드의 일관성을 유지할 수있다. 중복 코드 작성을 줄이고, 유지보수를 간소화할 수 있다. 또한 컴포넌트 간의 인터페이스가 명확하게 정의되어 있어, 각 모듈 간 결합도가 낮아짐으로써 코드 변경에 따른 영향 범위를 최소화하고 재사용성을 극대화할 수 있다. 이런 구조적 장점은 특히 대규모 개발 프로젝트에서 큰 이점을 제공한다.

Polylith의 또 다른 핵심 개념은 레고 블록 방식의 조립이다. Polylith의 컴포넌트는 레고 블록처럼 서로 독립적이고 조합 가능하게 설계되어 있어, 개발자가 필요에 따라 각 기능을 자유롭게 재배열할 수 있다. 특정 서비스에 새로운 기능을 추가하거나 기존 기능을 재사용하려는 경우, 컴포넌트를 단순히 추가하거나 다른 서비스에서 가져와 사용할 수 있다. 개발자는 동일한 기능을 여러 서비스에서 반복적으로 구현할 필요 없이, 이미 존재하는 컴포넌트를 재사용함으로써 개발 속도를 높이고 시스템의 일관성을 유지할 수 있어 복잡한 시스템에서 개별 모듈의 수정과 확장을 용이하게 만들어 준다.

Polylith는 개발 환경과 배포 환경의 분리를 통해 개발 경험을 향상시킨다. 일반적인 마이크로서비스 아키텍처에서는 각 서비스가 독립적으로 배포되기 때문에 개발자가 모든 서비스를 일관되게 관리하기 어려운 경우가 많지만 Polylith은 모든 컴포넌트와 기반을 하나의 개발 프로젝트에서 관리하고, 이 개발 프로젝트를 통해 전체 시스템을 한 곳에서 작업할 수 있게 한다. 개발자는 코드 변경 사항을 빠르게 확인하고 피드

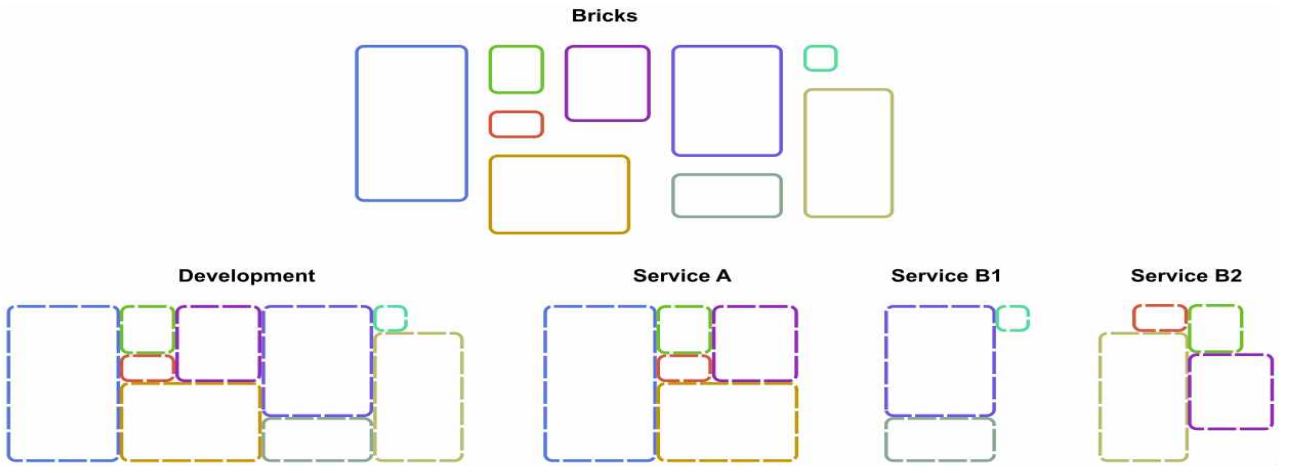


그림 1. 코드블록을 활용한 다양한 코딩 방법
Fig. 1. Various coding methods using code blocks

백을 받을 수 있으며, 개발의 생산성과 품질을 높이는 중요한 요소가 된다. 이런 접근 방식은 코드 수정 시 발생하는 오류를 조기에 발견하고 해결할 수 있는 빠른 피드백 루프를 제공하며, 복잡한 시스템에서도 개발의 일관성을 유지하게 한다.

Polylith 아키텍처는 배포의 유연성을 제공한다. 각 프로젝트는 다양한 컴포넌트와 기반을 포함하여 배포 가능한 아티팩트로 정의된다. 이러한 구조는 개발과 배포의 결합을 최소화하여, 개발 단계에서는 전체 코드를 하나의 프로젝트로 작업하고, 배포 단계에서는 필요에 따라 다양한 조합으로 서비스를 구성할 수 있는 유연성을 제공한다. 동일한 컴포넌트를 여러 배포 아티팩트에서 사용할 수 있어, 시스템 확장이나 변경이 필요한 경우에도 기존 코드를 재사용하여 빠르고 간편하게 대응할 수 있다. Polylith은 마이크로서비스와 유사하게 각 컴포넌트를 독립적으로 배포할 수 있지만, 모놀리식과 같은 단일 개발 경험을 제공하여 개발 생산성을 높인다.

Polylith은 기존 소프트웨어 아키텍처의 단점을 보완하고 개발 생산성을 높일 수 있는 혁신적인 접근 방식을 제공한다. 모놀리식의 개발 편의성과 마이크로서비스의 확장성을 결합한 형태로, 코드 재사용과 독립적 배포가 가능하며, 모듈화된 개발 환경을 통해 개발자의 경험을 향상 시킨다. 컴포넌트 기반 구조를 통해 각 기능을 독립적으로 개발하고 유지보수할 수 있어, 대규모 시스템에서도 일관된 품질을 유지할 수 있다. Polylith 아키텍처는 특히 빠르게 변화하는 비즈니스 요구사항에 유연하게 대응해야 하는 현대의 소프트웨어 개발 환경에서 큰 가치를 제공한다. Polylith은 개발자들에게 더 높은 생산성과 더 나은 개발 경험을 제공하며, 복잡한 소프트웨어 시스템의 관리와 확장을 보다 효율적으로 만들 수 있다.

2) 레고 블록 개념을 적용한 Polylith 아키텍처

최초 발표 이후 2018년에 소프트웨어 엔지니어 James Kronk는 Polylith 레고블록 개념을 적용한 아키텍처를 발표했는데, Polylith 아키텍처가 개발자에게 제공할 수 있는 새로운 개발 경험과 시스템 구조 방식을 중점 개념으로 하고 있다. 소프트웨어 시스템을 레고 블록처럼 구성하는 Polylith 아키텍처의 기본 개념을 소개하면서, 이를 통해 단순하고 유지보수가 용이하며 테스트 가능한 시스템을 구축할 수 있음을 강조한다. James는 Polylith 아키텍처의 세 가지 핵심 요소인 컴포넌트(Component), 베이스(Base), 라이브러리(Linrary)를 사용하여 개발자들이 시스템을 유연하고 모듈화된 방식으로 구성할 수 있는 방법을 제시하였다(그림 2).

기존 Polylith 아키텍처는 소프트웨어 개발에서 모놀리식, 마이크로서비스, 서버리스 아키텍처와 비교하여 고유한 장점을 제공해왔지만, 일부 한계도 가지고 있었다. 개선된 Polylith 아키텍처는 이러한 한계를 극복하면서 Polylith의 장점을 더욱 확장한 형태로 설계되었다. 이 개선된 아키텍처는 클로저(Closure) 언어와의 결합을 통해, 개발자들이 각 컴포넌트를 독립적으로 관리하고 재사용성을 높일 수 있는 방식으로 진화하였다.

기존의 Polylith 아키텍처는 컴포넌트를 모듈화하여 시스템 전체를 하나의 통합된 코드베이스로 관리하는 접근 방식인데 개발자에게 모놀리식의 단순성뿐만 아니라 마이크로서비스의 확장성을 제공하는 특징을 가집니다. 그러나 기존의 구조는 개발 환경과 배포 환경 사이의 갭이 존재하여, 배포 시점에서의 유연성과 운영 비용 최적화 측면에서 일부 개선의 여지가 있었다. 개선된 Polylith 아키텍처는 이러한 갭을 효과적으로 메우기 위해 하나의 개발 환경에서 모든 컴포넌트를 결합하여 마치 단일 모놀리식 시스템처럼 개발과 테스트를 진행할 수 있는 기능을 제공한다. 개발자는 코드 격리와 캡슐화를 유지하면서도 단일 시스템의 일관된 개발 경험을 유지할 수 있다. 개발 시 모든 컴포넌트가 하나의 환경에서 연결되어 작동하기 때문에, 코드 변경의 영향 범위를 보다 쉽게 파악할 수 있으며 테스트 주기 역시 간소화된다.

개선된 Polylith 아키텍처는 컴포넌트 간의 재사용성을 극대화함으로써 효율적인 시스템 확장을 가능하게 한다. Polylith의 독창적인 상징 링크 구조(symbolic link structure)는 기존의 마이크로서비스 아키텍처처럼 각 서비스가 완전히 분리되는 방식보다 더 단순하면서도 강력한 구조를 제공한다. 이런 구조를 통해 각 컴포넌트를 별도의 서비스로 배포할 수도 있고, 필요에 따라 단일 애플리케이션으로 결합해 배포할 수도 있는 유연성을 가지게 된다.

개선된 Polylith 아키텍처가 제공하는 유연한 배포 옵션을 활용하여 운영 비용을 절감하고 컴포넌트의 재사용성을 극대

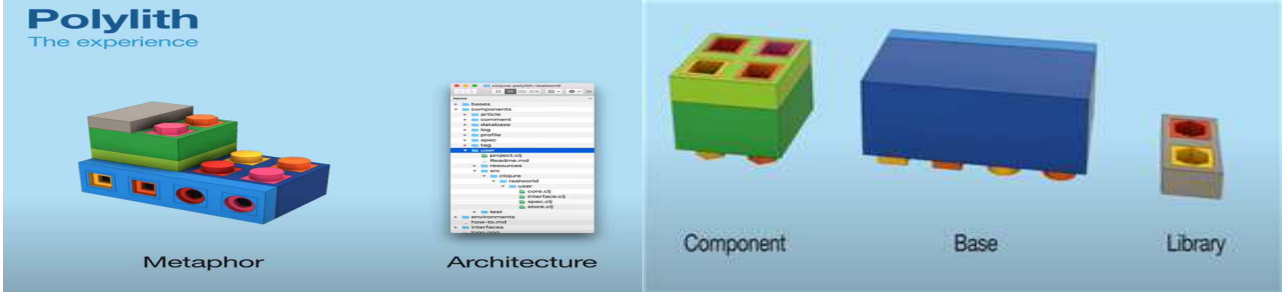


그림 2. Polylith: 레고와 유사한 블록을 기반으로 하는 소프트웨어 아키텍처
Fig. 2. Polylith: A software architecture based on LEGO-like blocks

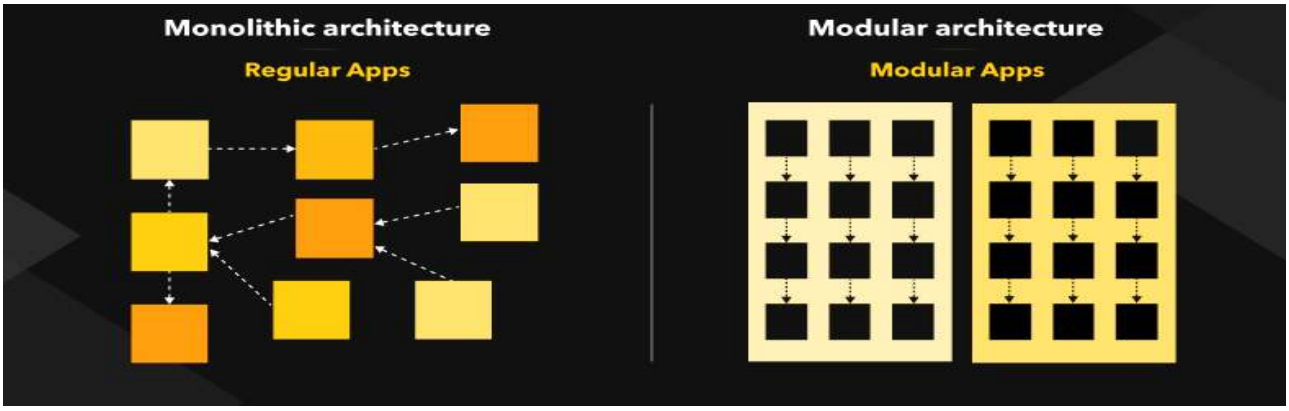


그림 3. 모놀리식 아키텍처와 모듈형 아키텍처 소프트웨어 설계 비교
Fig. 3. comparison of monolithic architecture vs Modular architecture software design

화함으로써 유지보수 및 확장의 효율성을 높일 수 있다.

결과적으로, 개선된 Polylith는 기존의 모놀리식, 마이크로서비스, 서버리스 아키텍처가 가진 한계를 보완하며 개발 및 운영 경험을 한층 더 발전시킬 수 있는 가능성을 제시했다[7].

3) 모듈형 소프트웨어 개발

모듈형 소프트웨어 개발은 현대 소프트웨어 엔지니어링에서 필수적인 패러다임으로 자리 잡고 있으며, 여러 가지 이론적 근거에 의해 뒷받침된다. 첫째, 모듈화는 복잡한 시스템을 더 작은, 독립적인 모듈로 분할함으로써 코드의 가독성과 유지보수성을 향상시킨다[8]. 각 모듈은 특정 기능을 수행하도록 설계되어 있어 개발자들이 시스템의 특정 부분에 집중할 수 있게 하며, 코드의 이해와 수정 과정을 단순화한다. 이러한 구조는 특히 대규모 프로젝트에서 유리하며, 개발자들이 개별 모듈을 독립적으로 개발하고 테스트할 수 있도록 지원한다[9]. 둘째, 모듈형 소프트웨어 설계는 코드 재사용성을 촉진한다. 잘 정의된 모듈은 다른 프로젝트에서도 쉽게 재사용될 수 있어 개발 시간을 절약하고 소프트웨어의 전반적인 품질을 개선한다[10]. 새로운 기능을 구현할 때 기존의 검증된 모듈을 활용함으로써 가능하며, 결과적으로 개발 프로세스의 효율성을 높인다. 또한, 모듈화는 시스템 확장성을 지원하여 비즈니스 요구사항 변화에 유연하게 대응할 수 있게 한다[11]. 셋째, 모듈형 아키텍처는 병렬 개발을 가능하게 하여 여러 팀이 동시에 작업할 수 있도록 한다. 각 모듈이 느슨하게 결합되어 있어 한 모듈의 변경이 다른 모듈에 미치는 영향을 최소화함으로써 협업을 용이하게 하고 충돌을 줄이고[12] 생산성을 높이고 개발 주기를 단축시켜 시장 출시 시간을 단축하는 데 기여한다. 병렬 개발은 특히 빠르게 변화하는 기술 환경에서 경쟁력을 유지하는 데 필수적이다. 넷째, 모듈형 설계는 디버깅과 테스트 과정을 용이하게 한다. 문제가 발생했을 때, 개발자는 문제를 특정 모듈로 격리하여 그 부분만 집중적으로 수정할 수 있다[13]. 오류의 근본 원인을 빠르게 식별하고 목표 지향적인 수정이 가능하도록 하며, 각 모듈이 독립적으로 테스트될 수 있어 전체 시스템의 신뢰성을 높인다. 모듈형 개발 접근 방식은 소프트웨어 유지보수 비용을 절감하고 시스템 안정성을 강화하는 데 중요한 역할을 한다.

마지막으로, 모듈형 소프트웨어 개발은 변화하는 요구사항에 대응하기 위한 유연성을 제공한다. 각 모듈은 독립적으로 설계되어 있어 새로운 요구사항이 발생했을 때 특정 모듈만 교체하거나 업데이트하면 되므로 전체 시스템에 미치는 영향을 최소화할 수 있다[13]. 유연성은 기업이 빠르게 변화하는 시장 환경에서 경쟁력을 유지하고 고객 요구에 신속히 대응할 수 있도록 돕는다.

모듈형 소프트웨어 개발은 현대 소프트웨어 엔지니어링에서 지속적으로 주목받고 있으며, 앞으로도 그 중요성은 더욱 커질 것이다.

4) 거대 언어 모델(LLM)과 코딩

거대 언어 모델(LLM)의 발전은 소프트웨어 개발 분야에 혁신적인 변화를 가져오고 있다. LLM은 자연어 처리(NLP) 기술을 활용하여 코드 생성, 자동 완성, 디버깅 지원 등 다양한 소프트웨어 개발 작업을 수행할 수 있는 능력을 갖추고 있다. 이러한 기능은 개발자들이 복잡한 코드를 보다 효율적으로 작성하고 오류를 줄이는 데 큰 도움을 준다. LLM은 간단한 자연어 설명을 통해 코드 스니펫을 생성할 수 있어 개발 시간을 단축하고 생산성을 높인다[14]. 특히 비전문가가 프로그래밍에 접근할 수 있는 장벽을 낮추어 소프트웨어 개발의 민주화를 촉진할 수 있다[15]. LLM의 또 다른 중요한 기능은 실시간 코드 제안 및 자동 완성이다. 개발자가 코드를 작성하는 동안 문맥에 맞는 제안을 제공하여 오류를 사전에 방지하고 코드 품질을 향상시킨다. LLM은 함수 이름이나 변수명을 입력할 때 적절한 완성을 제안하고, 구문 오류를 미리 감지하여 수정할 수 있도록 도와준다. 즉각적인 피드백 메커니즘은 개발자들이 보다 빠르고 정확하게 코드를 작성할 수 있게 하며, 전체 개발 프로세스의 효율성을 크게 향상시킨다.

디버깅과 오류 처리에서도 LLM은 강력한 도구로 자리 잡

고 있다. LLM은 오류 메시지를 해석하고 잠재적인 해결책을 제안함으로써 개발자들이 문제를 신속하게 해결할 수 있도록 지원한다. 런타임 오류가 발생했을 때, 개발자는 오류 메시지를 LLM에 입력하여 문제의 원인과 가능한 해결책에 대한 통찰을 얻을 수 있다[16].

복잡한 시스템에서 문제를 격리하고 수정하는 데 유용하며, 결과적으로 소프트웨어의 안정성과 신뢰성을 높이는 데 기여한다.

LLM은 소프트웨어 아키텍처와 설계에서도 중요한 역할을 할 수 있다. LLM은 모듈형 시스템 설계를 지원하여 확장 가능하고 유지보수하기 쉬운 소프트웨어 아키텍처를 구축하는 데 도움을 준다. 성능 최적화와 모범 사례 준수를 보장하기 위해 아키텍처 설계 패턴을 제안하고 구성 요소 간의 관계를 제시할 수 있다[16].

5) 소프트웨어 개발에서의 3D 사용자 인터페이스(UI)

3D 사용자 인터페이스(UI)는 소프트웨어 개발 분야에서 혁신적인 변화를 이끌고 있으며, 사용자 경험을 획기적으로 향상시키는 데 기여하고 있다. 최근 연구에 따르면, 3D UI는 전통적인 2D 인터페이스에 비해 사용자 참여도를 최대 37%까지 증가시킬 수 있으며, 복잡한 정보 구조를 탐색하는 능력도 42% 향상시킬 수 있다. 3D UI 인터페이스는 사용자가 가상 환경에서 물리적 세계와 유사한 방식으로 상호작용할 수 있도록 하여 몰입감을 높이고 직관적인 사용자 경험을 제공한다. 3D 가상 쇼룸과 같은 새로운 개념은 사용자가 디지털 제품 및 서비스를 보다 현실감 있게 탐색하고 상호작용할 수 있는 기회를 제공한다. 3D UI의 기술적 구현은 복잡하지만, 소프트웨어 개발의 새로운 가능성을 열어준다. 강력한 3D 엔진인 Unity와 Unreal Engine이 이러한 인터페이스의 구현을 지원하며, 게임 개발에서 주로 사용되던 기술이지만 이제는 일반적인 소프트웨어 디자인에도 적용되고 있다[17].

3D UI 엔진은 AI 및 머신러닝 알고리즘과 결합되어 사용자 상호작용 데이터를 분석하고 최적화된 3D UI 요소를 자동으로 생성한다. AI 시스템은 사용자의 3D 환경 내 상호작용 히트맵을 분석하여 UI 요소의 배치를 조정함으로써 사용성 및 발견 가능성을 향상시킬 수 있다. 또 가상 현실(VR), 증강현실(AR), 혼합 현실(MR)과 같은 첨단 기술과 결합되어 더욱 발전하고 있다. 비디오 게임과 같은 상업적 응용 분야에서 널리 사용되며, 사용자 경험을 더욱 풍부하게 만든다. 연구자들은 3D 제스처 인식 정확도를 개선하고, 다양한 제스처를 신뢰성 있게 인식할 수 있는 새로운 알고리즘을 개발하고 있다[18]. 게임에서 복잡한 상호작용을 지원하는 데 중요한 역할을 하며, 사용자들이 보다 자연스럽고 직관적으로 시스템과 상호작용할 수 있도록 돕는다. 마지막으로, 3D UI의 발전은 단순히 시각적 개선에 그치지 않고, 디지털 정보와의 상호작용 방식을 근본적으로 변화시키고 있다. 데이터 시각화를 개선하고 보다 직관적이고 몰입감 있는 사용자 경험을 창출하여 디지털 환경을 물리적 세계와 더욱 가깝게 만든다[19].

6) 객체 지향 프로그래밍(OOP)과 Polylith

객체 지향 프로그래밍(OOP; Object-Oriented Programming)은 프로그램을 객체라는 단위로 구조화하여 데이터와 해당 데이터를 처리하는 코드를 묶어 다루는 프로그래밍 패러다임이다. OOP의 주요 특징으로는 캡슐화, 상속, 다형성, 추상화가 있으며, 이를 통해 모듈화된 코드 작성, 재사용성, 유지보수성, 확장성을 제공한다. Java, C++, Python, C#, Ruby 등 많은 현대 언어들이 OOP를 지원하며, 이를 통해 대규모 시스템 개발 시 체계적이고 효율적인 프로그램 설계와 구현이 가능하다[20].

Polylith 아키텍처는 Git을 사용한 전통적인 객체지향 프로그래밍(OOP) 방식과 비교했을 때, 모듈화와 재사용성, 의존성 관리 측면에서 더욱 체계적이고 효율적인 접근법을 제공한다. 전통적인 OOP 방식에서는 시스템이 클래스와 객체를 중심으로 설계되고, 전체 프로젝트 단위에서 빌드 및 테스트가 이루어지는데, 시스템의 복잡성이 증가할 경우 순환 의존성과 같은 문제들이 발생하기 쉬우며, 코드의 모듈화와 재사용성은 주로 라이브러리나 프레임워크 수준에서 제한적으로 이루어진다. 반면, Polylith는 시스템을 작고 독립적인 컴포넌트 단위로 분해하여 관리하며, 이러한 컴포넌트는 명확하게 정의된 의존성 구조를 갖는다. 컴포넌트 기반의 접근법은 더욱 세밀한 수준의 모듈화와 코드 재사용성을 가능하게 하며, 동일한 컴포넌트를 쉽게 다른 프로젝트에 통합하거나 재사용할 수 있다. 또한 Polylith는 모노레포(monorepo) 구조를 사용하여 전체 시스템을 하나의 저장소에서 통합 관리하는 방식을 채택하며, 이를 통해 협업과 코드베이스 관리가 용이하다. 개발 과정에서는 변경된 컴포넌트만 선택적으로 빌드하고 테스트할 수 있기 때문에 개발 속도와 효율성이 향상된다. Polylith의 접근법은 시스템의 모듈성을 높이고, 의존성을 체계적으로 관리하며, 코드 재사용성을 극대화함으로써 대규모 시스템이나 마이크로서비스 아키텍처에서 특히 큰 이점을 제공한다[21].

2-2 LLM과 3D UI를 적용해 개선된 Polylith 아키텍처

1) 개념적 개요 및 아키텍처

- **개선된 Polylith 아키텍처 개요와 주요 구성 요소**

그림 4에 제시된 Polylith 아키텍처의 주요 구성 요소는 Integration Layer, LLM Integration Layer, Enhanced UI Layer, Core Polylith Layer, 그리고 Python Foundation Layer로 구성되어 있다. 각 레이어는 시스템의 다양한 기능을 담당하며, 이를 통해 전체적인 소프트웨어 개발 과정이 모듈식으로 이루어진다. Integration Layer는 메시지 큐와 이벤트 매니저를 포함하고 있어 모듈 간 통신을 비동기적으로 수행할 수 있다. LLM Integration Layer는 LLM Orchestrator와 Analysis Cache를 통해 코드 분석, 문서화, 최적화 작업을 수행하고, AI 기반의 자동화 기능을 제공한다.

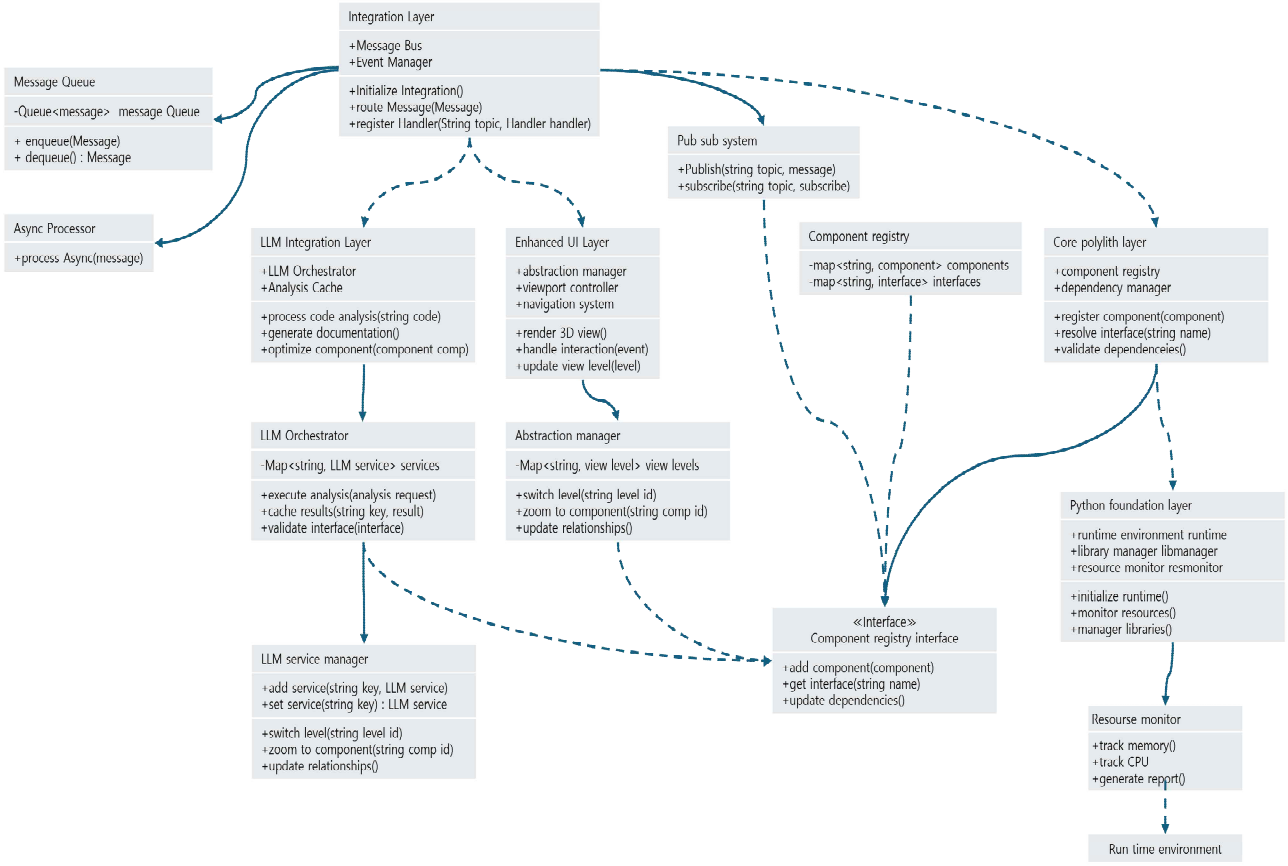


그림 4. LLM과 3D UI를 적용해 개선된 Polylith 아키텍처
Fig. 4. Improved Polylith architecture with LLM and 3D UI

Enhanced UI Layer는 3D UI를 렌더링하고, 사용자가 시스템의 복잡한 관계를 시각적으로 이해하고 탐색할 수 있도록 한다. Core Polylith Layer는 컴포넌트의 등록과 의존성 관리를 담당하며, Python Foundation Layer는 시스템의 런타임 환경을 관리하고, 자원 모니터링을 통해 시스템 성능을 최적화한다.

• 통합 레이어의 역할과 메시지 흐름

Integration Layer는 시스템의 중심적인 통신 허브로서, 다양한 모듈 간의 메시지 전달을 담당한다. 이 레이어는 메시지 큐를 사용하여 메시지를 비동기적으로 관리하며, 이벤트 매니저를 통해 각 모듈이 필요한 주제에 대해 구독하거나 메시지를 발행할 수 있게 한다. 메시지 큐(Message Queue)는 메시지를 큐에 추가하거나 삭제하는 기능을 제공하며, 비동기 프로세서(Async Processor)를 통해 메시지를 처리한다. 이 과정에서 Pub/Sub 시스템이 작동하여 메시지가 적절한 구독자에게 전달되도록 한다. 이러한 비동기 메시징 모델은 시스템 내의 결합도를 줄여주며, 각 모듈이 독립적으로 작동하면서도 필요할 때 정보를 교환할 수 있도록 한다. 따라서 모듈이 추가되거나 제거될 때 시스템의 다른 부분에 미치는 영향을 최소화할 수 있다.

• LLM 통합과 AI 기반 기능

LLM Integration Layer는 시스템 내에서 AI 기반 기능을 제공하는 중요한 역할을 한다. 이 레이어는 LLM Orchestrator를 통해 코드의 분석과 최적화 작업을 수행하며, Analysis Cache를 사용하여 반복적인 요청에 대해 효율성을 높인다. LLM Orchestrator는 여러 LLM 서비스를 관리하며, 분석 요청을 처리하고 그 결과를 캐싱하여 후속 요청에 대해 빠르게 응답할 수 있다. LLM Service Manager는 다양한 LLM 서비스를 추가하거나 설정할 수 있도록 하여, 필요에 따라 최적의 AI 모델을 사용할 수 있게 한다. 개발자가 특정 코드를 최적화하고자 할 때 LLM Orchestrator는 이를 분석하고 최적화된 코드 스니펫을 제안할 수 있다. 이러한 AI 통합은 소프트웨어 개발 과정에서 개발자의 효율성을 높여주며, 복잡한 코드 구조의 이해를 돕는다.

• 향상된 UI와 사용자 인터랙션

Enhanced UI Layer는 사용자와 시스템 간의 상호작용을 시각적으로 제공하는 중요한 레이어이다. 이 레이어는 Abstraction Manager를 포함하고 있으며, 시스템의 전체적인 구조를 3D로 시각화하여 개발자가 쉽게 이해할 수 있도록 돕는다. Abstraction Manager는 시스템, 모듈, 컴포넌트 레

벨 간의 전환을 관리하며, 사용자가 특정 컴포넌트를 선택하고 확대하여 세부적인 정보를 확인할 수 있게 한다. 또한 Viewport Controller와 Navigation System을 통해 사용자는 UI 내에서 자유롭게 이동하고, 시스템의 복잡한 관계를 탐색할 수 있다. 예를 들어, 사용자가 특정 모듈을 클릭하면 해당 모듈의 의존성 관계가 3D UI 상에서 시각적으로 강조되며, 모듈 간의 상호작용을 더 명확하게 이해할 수 있게 한다. 이러한 UI의 시각적 표현은 개발자가 시스템의 구조적 복잡성을 줄이고, 더 직관적으로 이해할 수 있도록 도와준다.

• Python Foundation Layer와 리소스 모니터링

Python Foundation Layer는 시스템의 런타임 환경을 제공하며, 라이브러리 관리와 리소스 모니터링을 담당한다. 이 레이어는 Runtime Environment, Library Manager, 그리고 Resource Monitor로 구성되어 있으며, 각각 시스템의 안정적인 실행을 지원하는 중요한 역할을 한다. Resource Monitor는 CPU 사용량, 메모리 사용량, 디스크 사용량 등을 지속적으로 추적하여 시스템의 성능을 모니터링하고, 필요할 때 경고를 제공한다. 시스템의 CPU 사용량이 특정 임계치를 초과하면 Resource Monitor는 개발자에게 경고를 보내거나 시스템의 상태를 시각적으로 표시하여 문제를 해결할 수 있도록 도와준다. 이러한 모니터링 기능은 시스템의 성능 최적화를 위해 매우 중요하며, 안정적인 소프트웨어 실행을 보장하는 데 기여한다.

제안된 Polylith 아키텍처는 다양한 레이어와 모듈이 서로 협력하여 완성된 Polylith 스타일의 시스템을 구성한다. 각 레이어는 특정한 기능을 담당하며, 독립적으로 작동하면서도 다른 레이어와 통합되어 전체적인 시스템의 일관성을 유지한다. 통합 레이어는 모듈 간의 통신을 관리하고, LLM 통합 레이어는 AI 기반의 코드 분석과 최적화를 제공하며, UI 레이어는 사용자와의 상호작용을 시각적으로 지원한다. Python Foundation Layer는 이러한 모든 작업이 원활하게 이루어지도록 시스템의 런타임 환경을 관리하고 모니터링한다.

2-3 시각적 개발을 위한 3D UI 구현 방안

레고 블록 형태의 3D UI를 활용한 시각적 개발 방안은 개발자들이 복잡한 시스템을 쉽게 이해하고 구성할 수 있도록 돕는 혁신적인 방법으로 개발자가 시스템의 각 모듈을 레고 블록처럼 물리적으로 조립하고 조작할 수 있는 환경을 제공한다. 물리적 접근은 각 모듈의 역할과 상호작용을 시각적으로 표현하여, 개발자가 시스템 전체 구조를 직관적으로 파악할 수 있도록 한다. 각 기능 모듈을 색상과 크기, 형태로 구분하여 블록 형태로 시각화함으로써, 시스템의 구성 요소들이 서로 어떻게 연결되고 의존하는지를 명확히 이해할 수 있다. 사용자는 3D 공간에서 블록을 직접 드래그하거나 조립할 수 있으며, 이를 통해 물리적 인터페이스를 통한 개발의 직관성을 극대화할 수 있다.

3D UI는 창의적이고 직관적인 개발 경험을 제공한다. 사용자가 모듈을 블록처럼 구성할 때, 각 블록은 특정 기능을 나타내며, 다른 블록과 연결되는 방식으로 시스템의 기능적 구조를 형성한다. 데이터 처리 모듈을 파란색 블록으로, 사용자 인터페이스 모듈을 빨간색 블록으로 구분하고, 이들이 상호작용하는 방식을 시각적으로 표현함으로써, 개발자는 전체 시스템의 구조를 빠르게 파악하고 이해할 수 있다. 또한, 사용자는 특정 모듈을 클릭하면 LLM(거대 언어 모델)이 해당 모듈의 기능을 자연어로 설명해주거나 코드 개선에 대한 제안을 제공함으로써, 개발자가 모듈의 목적과 동작을 더 깊이 이해할 수 있도록 한다.

3D UI는 협업 환경에서도 큰 강점이 있다. 각 팀원은 자신의 아바타로 3D 개발 환경에 접속하여 실시간으로 작업을 수행할 수 있으며, 모듈을 직접 조립하거나 변경 사항을 다른 팀원과 공유할 수 있다. 한 팀원이 특정 모듈을 수정하면 그 변경 사항이 즉시 시각적으로 반영되어 다른 팀원들이 이를 쉽게 인식하고 협업할 수 있다. 3D UI 기반의 멀티유저 환경은 팀 내의 소통을 원활하게 하며, 실시간으로 시스템의 변화를 추적하고 조정할 수 있도록 한다. LLM은 이러한 협업 과정에서도 중요한 역할을 한다. LLM은 각 팀원의 작업 내용을 분석하고, 모듈 간의 의존성이나 변경 사항의 영향을 자동으로 평가하여 최적의 해결책을 제안한다. 팀 전체의 협업 효율성을 높이고, 복잡한 시스템에서 발생할 수 있는 오류를 줄이며, 시스템의 일관성을 유지하는 데 도움을 줄수 있다.

더 나아가, 이러한 3D UI 환경은 게임화된 요소를 도입하여 개발 과정에 재미를 더할 수 있다. 특정 모듈을 성공적으로 최적화하거나 새로운 기능을 추가했을 때 레고 블록이 빛

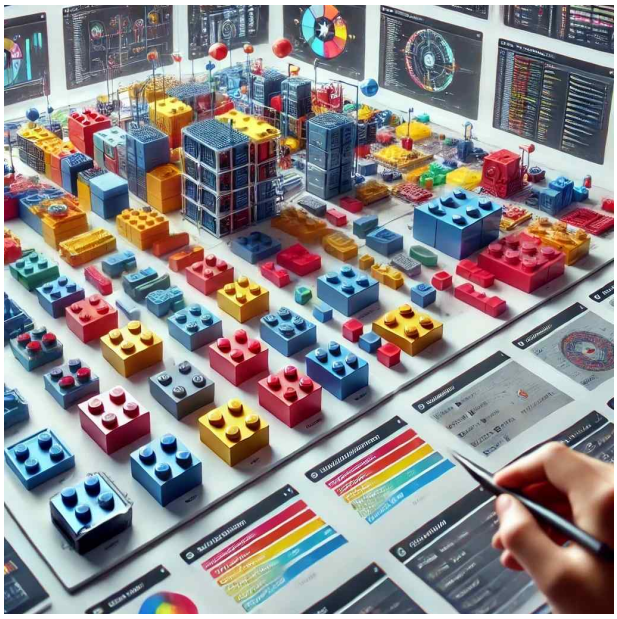


그림 5. 3D UI를 적용한 Polylith 구현 개념도
Fig. 5. Conceptual diagram of Polyllith implementation with 3D UI

나거나 애니메이션 효과를 줄 수 있다. 이와 같은 피드백은 개발자에게 성취감을 주며, 반복적인 개발 작업에서 동기를 부여하는 역할을 한다.

개발 과정에서 발생하는 문제를 해결하기 위해 블록을 재배치하거나 새로운 블록을 추가하는 등, 문제 해결 과정을 물리적으로 시각화하는 방식은 개발자들이 시스템의 문제를 더 창의적으로 해결하도록 유도한다. 게임화한 접근은 개발자들이 더욱 몰입하고 창의적으로 문제를 해결하는 데 큰 기여를 할 수 있다.

2-4 활용 가능성 및 해결 과제

1) Polylith 아키텍처의 활용 가능성

Polylith 아키텍처는 시스템을 모듈화하여 각 구성 요소를 독립적으로 개발하고 유지보수할 수 있는 환경을 제공한다. 모듈화는 개발자들이 시스템을 쉽게 이해하고 관리할 수 있도록 돕고, 새로운 요구사항을 손쉽게 통합할 수 있는 이점을 제공한다. 각 모듈의 역할이 명확하고 독립적으로 개발되므로 코드 품질이 향상되고 유지보수가 용이해진다. 팀 간의 협업을 촉진하여 개발 속도와 일관성을 유지할 수 있으며, 비즈니스 요구에 빠르게 대응할 수 있는 유연한 환경을 제공한다.

2) LLM과 3D UI 통합으로 인한 개발자 인지 이해도 향상

LLM과 3D UI의 통합은 복잡한 시스템을 직관적으로 이해하고 관리하는 데 도움을 준다. LLM은 코드의 분석과 설명을 제공하여 개발자가 코드의 의도와 동작을 쉽게 파악할 수 있도록 하고, 3D UI는 모듈 간의 관계를 시각적으로 표현해 명확한 이해를 지원한다. 개발자는 시스템 구조를 빠르게 파악하고 문제를 창의적으로 해결할 수 있으며, 코드 변경의 영향을 실시간으로 분석하여 시각적으로 제공함으로써 더 나은 의사결정을 내릴 수 있다.

3) 모듈형 구성 요소의 재사용성과 확장성

레고 블록 형태의 3D UI와 LLM 기반 Polylith 아키텍처는 모듈형 구성 요소의 재사용성과 확장성을 극대화한다. 각 모듈은 독립적으로 개발 및 유지보수될 수 있으며, LLM은 모듈의 재사용 가능성을 평가하고 최적의 조합을 제안해 개발 효율성을 높인다. 3D UI는 모듈 간의 관계를 시각적으로 표현하여 개발자가 최적의 재사용 및 확장을 계획할 수 있도록 돕고 대규모 시스템에서 복잡한 기능을 효율적으로 관리하고 유지보수하는 데 유용하다.

4) 3D UI 환경을 통한 협업 개선

3D UI 환경은 개발자 간의 실시간 협업을 지원하여, 시스템 변경 사항을 빠르게 공유하고 중복 작업을 방지한다. 3D UI는 팀원들이 동일한 시각적 정보를 기반으로 작업할 수 있도록 하여 협업의 투명성을 높이고, 역할 분담을 효율적으로 조정하는 데 기여한다. LLM은 실시간 모듈 상태 분석과 개선 사항 제안을 통해 협업 효율성을 극대화하며, 코드 리뷰 과정도 자동화해 시스템의 품질을 높인다.

5) 모듈형 코드 생성에서 LLM의 복잡성 처리

LLM은 코드 간의 복잡한 의존성을 분석하여 최적화된 모듈형 코드를 생성하고, 코드 일관성 및 오류 방지를 지원한다. 개발자는 반복 작업을 줄이고 창의적 문제 해결에 집중할 수 있다. LLM은 특히 대규모 프로젝트에서 복잡성을 관리하고 협업을 지원하는 중요한 도구로 작용한다.

6) 3D UI 구현의 기술적 도전과제

3D UI 구현에는 높은 그래픽 처리 능력과 사용자 경험(UX) 설계가 필요하다. 그래픽 최적화와 성능 개선, 효과적인 UI 디자인, 네트워크 안정성 및 동기화 문제 해결이 중요한 과제이다. 다양한 입력 장치와의 호환성을 고려한 인터페이스 설계도 필요하다. 도전과제를 해결하면 3D UI는 개발자들에게 직관적이고 효율적인 개발 환경을 제공하며, 시스템 개발과 유지보수의 효율성을 높일 수 있다.

Ⅲ. 결 론

본 연구는 Polylith 아키텍처와 LLM(거대 언어 모델), 그리고 3D UI를 결합한 소프트웨어 개발 방식을 제안하여 개발자 경험을 혁신적으로 개선하는 방안을 탐구하였다. Polylith 아키텍처를 기반으로 모듈형 구성 요소를 재사용하고 확장성을 극대화함으로써, 시스템의 유지보수성과 개발 속도를 크게 향상시킬 수 있는 가능성을 제시하였다. LLM과 3D UI를 통합하여 복잡한 시스템의 구조와 의존성을 직관적으로 시각화하고, 자연어 기반의 설명과 최적화 제안을 통해 개발자들이 더 효율적이고 창의적으로 문제를 해결할 수 있는 환경을 구축을 제안했다. 이를 통해 팀원 간의 실시간 협업을 원활하게 하고, 시스템 개발의 복잡성을 관리하며, 개발 과정에서의 인지적 부담을 줄이는 데 기여할 수 있다.

향후 연구에서는 개발자 인터페이스의 향상을 위해 몇 가지 방향성을 제시할 수 있다. 첫째, 3D UI의 구현에서 가상현실(VR)과 증강현실(AR) 기술을 도입함으로써 개발자들이 시스템을 더욱 몰입감 있게 이해하고 조작할 수 있는 환경을 구축할 수 있다. 몰입형 기술은 복잡한 시스템 구조를 더욱 직관적으로 탐색하고, 개발자가 시스템의 동작을 시각적, 공간적으로 이해하는 데 도움을 줄 것이다. 둘째, LLM의 성능을 더욱 강화하여 개발자의 의도를 보다 정교하게 이해하고 맞춤형 제안을 제공하는 방향으로 발전시킬 수 있다. 각 개발자의 작업 스타일과 선호도를 학습하여 개인화된 코드 제안 및 최적화 솔루션을 제공하는 기능을 추가함으로써, 개발자의 생산성과 만족도를 높일 수 있다. 마지막으로, 협업 도구와의 통합성을 강화하여 개발자들이 다른 팀원들과 더 효과적으로

소통하고 협력할 수 있는 환경을 조성하는 것이 중요하다. 이를 통해 다양한 팀 구성원들이 복잡한 프로젝트에서도 효율적으로 협업할 수 있도록 돕고, 시스템 개발의 전반적인 품질을 향상시킬 수 있을 것이다.

감사의 글

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 메타버스 융합대학원의 연구(IITP-2023-RS-2022-00156318)와 문화체육관광부 및 한국콘텐츠 진흥원의 2023년도 문화기술 연구개발사업(RS-2023-00219237)으로 수행된 연구로서, 관계부처에 감사드립니다.

참고문헌

[1] J. Purtilo, “Polylith: An Environment to Support Management of Tool Interfaces,” ACM SIGPLAN Notices, Vol. 20, No. 7, pp. 12-18, July 1985. https://doi.org/10.1145/17919.806822

[2] J. M. Purtilo, “The POLYLITH Software Bus,” *ACM Transactions on Programming Languages and Systems*, Vol. 16, No. 1, pp. 151-174, January 1994. https://doi.org/10.1145/174625.174629

[3] Hacker News. Polylith is a Functional Software Architecture at the System Scale (gitbook.io) [Internet]. Available: https://news.ycombinator.com/item?id=30697724.

[4] Kakao Enterprise. [IT TREND] How to Develop AI Apps in FMOps, LLM Era [Internet]. Available: https://tech.kakaoenterprise.com/196.

[5] Microsoft Learn. Important Concepts and Considerations for Developers Building Generative AI Solutions [Internet]. Available: https://learn.microsoft.com/ko-kr/azure/developer/ai/gen-ai-concepts-considerations-developers.

[6] Polylith. Polylith - GitBook [Internet]. Available: https://polylith.gitbook.io/polylith.

[7] YouTube. Polylith: A Software Architecture Based on LEGO-Like Blocks [Internet]. Available: https://youtu.be/Y3FfLq8QATY?si=sz8b2UkiVqhMO-a0.

[8] NIIT. Unlocking the Power of Modularity in Software Engineering [Internet]. Available: https://www.niit.com/india/Unlocking-the-Power-of-Modularity-in-Software-Engineering.

[9] Institute of Data. What Is Modularity in Software Engineering [Internet]. Available: https://www.institutedata.com/blog/modularity-in-software-engineering/.

[10] Gwentech. The Advantages of Modular Software and Programming [Internet]. Available: http://gwentechembedded.com/the-advantages-of-modular-software-and-programming/.

[11] Harrison Clarke. Benefits of Modular Architecture: Moving from Monolithic to Modular [Internet]. Available: https://www.harrisonclarke.com/blog/benefits-of-modular-architecture-moving-from-monolithic-to-modular.

[12] Maciej Kaszubowski. The Benefits of Modular Software Design [Internet]. Available: https://mkaszubowski.com/2020/06/02/modular-software-design-benefits.html.

[13] X. Gao, “The Application and Benefits of Modular Software Development in Automatic Train Supervision Systems,” *International Journal of Engineering And Science*, Vol. 14, No. 6, pp. 34-38, June 2024.

[14] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, “A Survey on Large Language Models for Code Generation,” arXiv:2406.00515, 2024. https://doi.org/10.48550/arXiv.2406.00515

[15] Unimedia Technology. Transforming Software Development: The Power of Large Language Models Explained [Internet]. Available: https://www.unimedia.tech/power-large-language-model-transforming-software-development/.

[16] Comet. The Future of Software Engineering: LLMs and Beyond [Internet]. Available: https://www.comet.com/site/blog/the-future-of-software-engineering-llms-and-beyond/.

[17] Softwarevates. Integrating 3D Modeling into Software Development for Enhanced User Interface Design [Internet]. Available: https://softwarevates.com/2024/10/03/integrating-3d-modeling/.

[18] H. Kharoub, M. Lataifeh, and N. Ahmed, “3D User Interface Design and Usability for Immersive VR,” *Applied Sciencs*, Vol. 9, No. 22, 4861, 2019. https://doi.org/10.3390/app9224861

[19] A. Yeo, B. W. J. Kwok, A. Joshna, K. Chen, and J. S. A. Lee, “Entering the Next Dimension: A Review of 3D User Interfaces for Virtual Reality,” *Electronics*, Vol. 13, No. 3, 600, February 2024. https://doi.org/10.3390/electronics13030600

[20] S.-I. Choi, “A Study on Teaching the Object Oriented Programming Language,” *The Journal of the Korea Institute of Electronic Communication Sciences*, Vol. 11, No. 8, pp. 751-758, 2016. https://doi.org/10.13067/JKIECS.2016.11.8.751

[21] M. Salah Uddin, K. Jahid Hasan, and R. Tasnima, “A Comparative Analysis of the Rise of Git-Based Distributed Collaborative Hosting Platforms: Survey, Performance

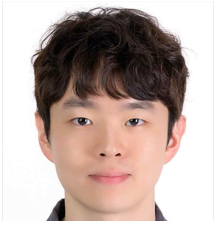
Test, and Comparison," *Asian Journal of Engineering and Applied Technology*, Vol. 12, No. 2, pp. 29-33, November 2023.https://doi.org/10.51983/ajeat-2023.12.2.3969

이주경(Jookyoung Lee)

2001년 : 육군3사관학교 (기계공학사)
2009년 : 아주대학교 경영대학원 (석사-E-비즈니스)

2001년~2020년: 육군
2021년~현 재: 국방기술진흥연구소(KRIT)
2023년~현 재: 서강대학교 메타버스전문대학원 메타버스 비즈니스 전공 박사과정
※관심분야 : 메타버스(Metaverse), 증강 및 혼합현실(AR / XR), 합성훈련환경((Synthetic Training Environment)

김태훈(Taehoon Kim)



2018년 : 서강대학교 (공학사 & 문학사, 컴퓨터공학 & 신문방송학)
2021년 : 서강대학교 대학원 (공학박사, 컴퓨터공학)

2021년~2024년: LG AI 연구원
2024년~현 재: 서강대학교 메타버스전문대학원 조교수
※관심분야 : 멀티모달 인공지능(Multimodal AI), 컴퓨터 비전(Computer Vision), 생성형 인공지능(Generative AI)