소프트웨어 설계 원칙을 게임 시스템 개발에 적용해 본 사례 연구

이상인[1], 노영주[2]
[1]한국공학대학교 컴퓨터공학과 석사과정
[2]한국공학대학교 컴퓨터공학과 교수
370603@naver.com, yrho@kpu.ac.kr

A Case Study on the Application of Software Design Principles in the Development of Game System

Sang-in Lee[1], Young-ju Rho[2]
[1]Dept. of Computer Engineering, Tech University of Korea
[2]Dept. of Computer Engineering, Tech University of Korea

요 약

본 연구는 소프트웨어 아키텍처의 설계 원칙을 분산 시스템에 적용하는 것을 목표로 한다. 특히 SOLID 원칙을 중심으로 소프트웨어 개발 과정을 분석하고 Unity 기반의 퍼즐 게임을 개발하여 실질적 적용 가능성을 검증하였다. 연구 결과, 설계 원칙을 준수한 소프트웨어는 기존 대비 높은 정확도와 안정성을 보였다. 본 연구는 분산 시스템 설계 및 소프트웨어 개발 교육에서 활용할 수 있는 사례를 제공하고자 한다.

1. 서 론

4차 산업혁명 시대를 맞아 소프트웨어는 점점 더 복잡해지고 있다. 이러한 환경에서 효과적이고 유연한 소프트웨어 설계는 필수적이다. 소프트웨어 아키텍처 원칙을 적용함으로써 코드의 품질과 유지보수성을 높이는 방법을 모색할 필요성이 대두되고 있다.

본 논문에서는 소프트웨어 아키텍처 설계 원칙, 특히 SOLID 원칙을 적용하여 분산 시스템 구현 사례를 개발하는 것을 목표로 하였다. 이를 통해 아키텍처 설계의 실용성을 증명하고, 이를 기반으로 소프트웨어 개발에 필요한 핵심 역량을 강화하고자 한다.

2. 관련 이론

2.1 소프트웨어 아키텍처 개요

소프트웨어 아키텍처는 소프트웨어를 설계할 때 사용하는 큰 그림 같은 개념이다. 건물을 지을 때 필요한 구조와 설계도와 비슷하다고 볼 수 있고, 프로그램이 어떻게 구성되고, 여러 기능이 어떻게 상호작용할지를 결정하는 “기본 설계도” 역할을 한다.

2.2 SOLID 원칙

SOLID 원칙은 객체지향으로 프로그램을 설계할 때 따를 것이 권장되는 매우 기본적인 원칙들의 집합이다. 각 원칙을 간략히 소개하면 다음과 같다[1].

- Single Responsibility Principle(단일 책임 원칙): 클래스는 하나의 책임만 가져야 하고, 클래스가 바뀌는 이유도 오직 하나여야 한다는 원칙이다.
- Open/Closed Principle(개방/폐쇄 원칙): 클래스는 확장에는 열려 있어야 하지만, 수정에는 닫혀 있어야 한다는 원칙이다.
- Liskov Substitution Principle(리스코프 치환 원칙: 자식 클래스는 부모 클래스의 역할을 대체할 수 있어야 하며, 자식 클래스를 부모 클래스처럼 사용할 수 있어야 한다는 원칙이다.
- Interface Segregation Principle(인터페이스 분리 원칙): 하나의 커다란 인터페이스보다는, 작은 인터페이스 여러 개로 분리하는 것이 좋다는 원칙이다.
- Dependency Inversion Principle(의존성 역전 원칙): 높은 수준 모듈은 낮은 수준 모듈에 의존해서는 안 되며, 둘 다 추상화된 인터페이스에 의존해야 한다는 원칙이다.

2.3 디자인 패턴

디자인 패턴은 소프트웨어 개발에서 자주 발생하는 문제를 해결하기 위해 반복적으로 사용되는 설계 방식을 의미한다[2]. 즉, 디자인 패턴은 특정 상황에서 검증된 “코딩 해결책”을 제공한다. 이러한 패턴을 사용하면 코드가 더 이해하기 쉽고 재사용 및 확장할 수 있는 구조를 갖추게 된다. 이러한 유용성을 확보하기 위해 기법에 SOLID 원칙이 적용되며 개발된다.

디자인 패턴은 크게 생성 패턴, 구조 패턴, 행위 패턴으로 나누어진다;

- 생성 패턴 : 객체 생성에 관련된 문제를 해결하고 객체 생성 과정을 단순화해 주는 패턴이다.

- 구조 패턴 : 객체 간의 관계를 설정하여 더 큰 구조를 형성하고, 객체 간 결합도를 낮추어 코드의 유연성을 높이는 패턴이다.
- 행위 패턴 : 객체 간 소통과 책임을 분배하여, 상호작용을 효율적으로 처리하는 방법을 제공하는 패턴이다.

3. 소프트웨어 아키텍처 구현

3.1 선행 연구 분석: SOLID 원칙에 관한 심층적 탐구 및 사례 연구.

"산업용 사물인터넷 기반 분산 서비스의 재구성을 위한 소프트웨어 참조 아키텍처 설계[3]"라는 논문을 읽고 분석해 보았다. 여기에서는 사물인터넷의 소프트웨어 아키텍처 설계 방식인 SOLID 원칙, 디자인 패턴 등이 언급되고, 이렇게 설계된 모듈 간 상호작용으로 사물인터넷이 작동한다고 강조한다[4]. 하지만, 선행 논문에서는 소프트웨어 아키텍처 방식의 장단점이 명확하게 언급되지는 않았다[5].

3.2 게임 개발: Unity 엔진과 C#을 활용하여 퍼즐 게임 소프트웨어를 설계.

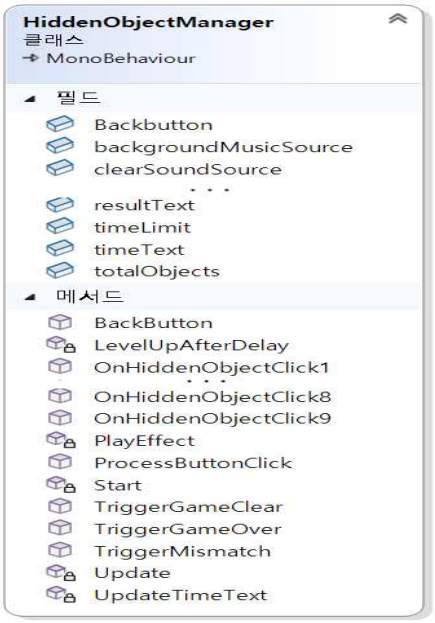
소프트웨어 아키텍처를 게임 소프트웨어 개발에 적용하면, 어떠한 이점이 있을지 생각하면서 퍼즐 게임을 개발하였다. 퍼즐 게임으로 개발한 이유는 유년기에 퍼즐을 좋아했기에 해당 게임 도메인에 익숙했고, 퍼즐의 직관적인 설계가 소스 코드 분석에 도움이 되기 때문이다. 기본 UI(로그인 화면, 레벨 선택 화면)와 게임 플레이 기능(레벨 1부터 20)을 디자인하고, 코딩을 통해 본인의 의도대로 게임을 구현했으며, 디버깅도 여러 번 해보았다. 개발한 소프트웨어의 소스 코드는 4,840줄이고, 실행 파일의 크기는 약 160MB 이다.

3.3 코드 최적화: 소프트웨어 아키텍처 원칙을 코드에 적용하여 유지보수성을 강화.

개발이 완료된 소프트웨어의 SOLID 원칙, 디자인 패턴을 미적용했을 때와 부분 적용했을 때, 완전히 적용했을 때, 이렇게 3가지를 테스트해 보았다.

3.3.1 소프트웨어 아키텍처 원칙 미적용

[그림 1] 아키텍처 미적용 UI 예시



[그림 2] 아키텍처 미적용한 코드의 다이어그램

게임 플레이를 담당하는 소스 코드는 아키텍처 원칙을 적용하지 않았다. [그림 1]처럼 숨은그림찾기의 경우 제한 시간 이내에 숨겨진 그림을 전부 찾아야 한다. 이럴 때, 반응 속도와 퍼포먼스가 중요하다. 이것을 아키텍처 원칙 적용 시 코드 라인 수가 늘어나고 게임 콘텐츠의 속도가 대략 0.3초 저하되는 현상이 일어난다.

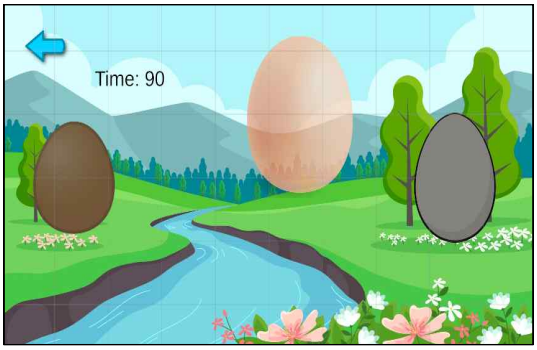
[그림 2]는 숨은그림찾기 기능을 담당하는 코드의 클래스 다이어그램으로 원칙을 미적용한 프로그램이다.

3.3.2 소프트웨어 아키텍처 원칙 부분 적용

[그림 3]은 알 깨기 게임의 진행 장면이다. 알이 생성되고 파괴되는 장면을 구현하려면 디자인 패턴 중 생성 패턴에 해당하는 '싱글턴'을 적용해야 한다. '싱글턴'이란 클래스의 인스턴스가 단 하나만 생성되도록 제한하는 패턴을 뜻한다.

저자가 의도한 대로 작동하려면, 알은 3개가 생성되고, 각 알은 순서대로 파괴해야 [그림 4]처럼 정상적으로 게임이 클리어된다. 만약에, '싱글턴'을 적용하지 않는다면, 알이 한 지점에서 중복으로 생성되거나 여러 지점에서 무한정 생성될 수 있다.

하지만, 게임의 반응 속도가 중요하기 때문에 오버헤드로 인한 속도의 손실을 차단하기 위하여 싱글턴 외에 다른 패턴은 적용하지 않았다.



[그림 3] 아키텍처 부분 적용 UI 예시

[그림 4] 알 깨기 게임의 클리어 장면

```
Unity 스크립트 | 참조 1개
public class SceneTransitionHelper : MonoBehaviour
{
    public static SceneTransitionHelper instance;

    Unity 메시지 | 참조 0개
    void Awake()
    {
        // 싱글톤 패턴을 통해 중복 인스턴스 방지
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(gameObject); // 씬 전환 시에도 유지
        }
        else
        {
            Destroy(gameObject); // 중복 인스턴스 파괴
        }
    }
```

[그림 5] 싱글턴을 적용한 소스 코드 일부

[그림 5]는 '싱글턴'을 적용한 코드의 예시이다. 이것을 통해 알이 같은 지점에서 중복 생성을 방지하고, 무한정 생성되는 것도 막아주는 역할을 한다. 또, 알이 파괴되는 장면을 구현해 준다.

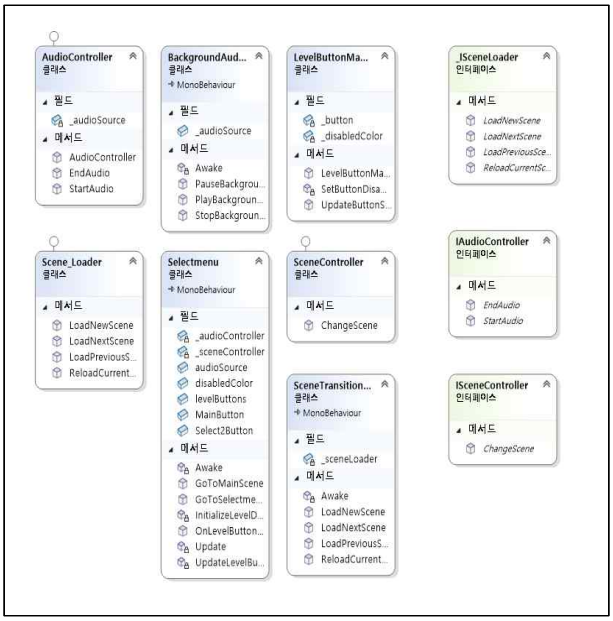
3.3.3 소프트웨어 아키텍처 원칙 완전 적용

아키텍처 원칙을 전부 적용한 소스 코드는 로그인 창, 레벨 선택 창, 배경음악 및 효과음 등을 담당하는 부분이다. 이 부분은 이미지와 음악을 변경하는 등 유지보수가 자주 필요하며, 유지보수를 쉽게 하려면 가독성이 좋아야 한다. 또한, 동일 소프트웨어 내 다른 기능과 충돌을 피하려면 기본적인 UI는 아키텍처 원칙을 되도록 준수해야 한다.

[그림 6] 레벨 선택 UI (아키텍처 완전 적용)

[그림 7]은 [그림 6]의 기능을 담당하는 코드의 클래스 다이어그램이다. 여기에 적용된 아키텍처 원칙의 일부를 설명하면, 각각 오디오, 버튼, 화면 전환을 담당하는 기능은 SRP를 준수하고 있다. 그리고 오디오와 화면 전환 로직을 캡슐화해서 OCP도 준수하고 있다. 또, 오디오와 화면을 담당하는 기능은 각각 상위 클래스의 명세를 정의하고 있고, 이것은 LSP도 역시 준수하고 있다. 게다가, 오디오 관리와 화면 전환을 담당하는 함수(메서드)가 각각의 인터페이스로 분리되어 있으므로 ISP도 준수한다.

마지막으로, 레벨 선택(Selectmenu) 클래스는 오디오와 화면 구현 인터페이스의 상위 인터페이스에 의존하며, 이를 통해 DIP도 준수한다고 볼 수 있다.



[그림 7] 아키텍처 적용한 코드의 다이어그램

4. 결론

4.1 연구 결과

Unity 엔진을 사용하여 분산 시스템 사례로 퍼즐 게임을 개발하였다. 개발된 소프트웨어는 사용자 경험을 개선하고 코드의 가독성을 높였고, SOLID 원칙을 적용함으로써 소프트웨어의 유연성과 재사용성

이 크게 향상되었으며, 예상보다 높은 정확도와 안정성을 달성하였다.

본 연구는 소프트웨어 아키텍처 설계 원칙이 분산 시스템 구현에 효과적으로 적용될 수 있음을 보여주고 있다. 연구를 통해 개발된 게임은 소프트웨어 설계의 중요성을 실증적으로 보여주는 사례로 활용될 수 있다.

4.2 기대효과

본 연구는 다음과 같은 학문적 및 실용적 가치를 제공할 것으로 기대된다.

- 소프트웨어 설계 원칙을 이해하는 데 필요한 사례 연구 자료 제공.
- 실질적 설계 원칙의 적용을 통해 소프트웨어 개발 교육에 기여.

4.3 향후 연구

이러한 소프트웨어 아키텍처를 게임 소프트웨어 외에도 인공지능(AI), 빅데이터, 네트워크 등에도 적용하는 방안을 탐구하면서 아키텍처의 장단점을 명확하게 도출할 것이다.

참 고 문 헌

[1] Harmeet Singh, "Effect of SOLID Design Principles on Quality of Software," *International Journal of Scientific & Engineering Research*, Vol. 6, Issue 4, 2015.

[2] James O. Coplien, "Software Design Patterns: Common Questions and Answers," *AT&T Bell Laboratories*, 1994.

[3] 황병훈 외, "산업용 사물인터넷 기반 분산 서비스의 재구성을 위한 소프트웨어 참조 아키텍처 설계," *고등기술연구원*, 2019.

[4] 강성원, "소프트웨어 아키텍처 설계의 근본 원리들", *Journal of Software Engineering Society*, 2010

[5] 고석하, “소프트웨어 아키텍처의 구성요소에 대한 포괄적 모델”, *한국정보시스템학회*, 2012