韓國IT서비스學會誌
第12卷 第3號
2013年 9月, pp.421-432
http://dx.doi.org/10.9716/KITS.2013.12.3.421

Linked Observer 패턴을 적용한 안드로이드 게임 프레임워크 설계 및 구현*

서 문 석**

Design and Implementation for Android Game Framework Using the Linked Observer Pattern*

Moon-Seog Seo**

■ Abstract ■

The changes in the game platform is appeared since the advent of smart-phones. Apple's iOS or Google's Android platform is gradually expanding their territories in the mobile game area. Android's platform fragmentation and the absence of game development framework act as barriers for game developments. In the development of game applications on the smart-phone, the standardized development procedures under game framework can achieve the productivity improvement. In this paper, we propose a new design pattern suitable for the design of a framework for new areas of application through the expansion of existing framework. The design and implementation of the Android game development framework is presented by taking advantage of existing design patterns and a proposed pattern. Also we propose the standardized development procedure of game applications under the proposed framework.

Keyword : Android, Framework, Design Pattern, Mobile Game

논문투고일 : 2013년 04월 25일 논문수정완료일 : 2013년 09월 01일 논문게재확정일 : 2013년 09월 03일
* 본 논문은 2013학년도 세한대학교 교내연구비 지원에 의해 작성되었습니다.
** 세한대학교 컴퓨터응용기술학과

1. 서 론

스마트폰이 등장한 이후 게임 플랫폼의 변화가 감지되고 있다. 닌텐도 DS나 플레이스테이션 포터블과 같은 전통적인 모바일 게임 플랫폼이 시장의 선도적 위치를 위협받고 있는 상황에서 애플의 iOS나 구글의 안드로이드 같은 스마트폰 플랫폼이 모바일 게임 영역에서 차지하는 비중은 점차 확대되고 있다. 특히 플랫폼의 개방성, 애플리케이션의 유통 혁신으로 수많은 애플리케이션들이 개인 또는 조직 단위로 생성, 보급 및 판매가 가능한 안드로이드는 게임 플랫폼으로써 충분한 장점을 가지고 있다. 그러나 다양한 버전이 공존하는 플랫폼의 파편화 및 표준화된 안드로이드 게임 전용 프레임워크의 부재는 다양하고 신속한 게임 개발의 장애 요인으로 작용하고 있다[4].

프레임워크는 애플리케이션을 관리하고 애플리케이션이 활용할 수 있는 토대를 제공하는 것으로 애플리케이션의 생애주기 동안 호출되어야 하는 API(Application Programming Interface)들을 제공한다. 스마트폰에서 실행되는 게임 애플리케이션의 경우에도 애플리케이션의 생애주기 동안 필요로 하는 API들이 정의된 게임 프레임워크 하에서 표준화된 개발 절차에 따라 개발됨으로써 생산성 향상을 도모할 수 있다. 안드로이드는 일반 애플리케이션 개발을 위한 프레임워크를 제공하고 있으나 게임 애플리케이션의 개발에는 부족한 점이 있다. 이를 위해 기존 프레임워크를 확장하여 게임 개발에 적합한 프레임워크의 제공이 필요하다[5].

프레임워크의 설계에는 제어의 역전(Inversion of Control) 기능, 환경설정 주입(Configuration Injection) 처리 기능과 같은 프레임워크가 해결하고자하는 공통 문제들의 해결을 위해 다양한 디자인 패턴들이 활용되고 있다. 이에 본 논문에서는 기존 프레임워크의 확장을 통해 새로운 영역의 애플리케이션 개발을 위한 프레임워크의 설계에 적용할 수 있는 디자인 패턴으로 Linked Observer 패턴을 제시하고 이를 적용한 안드로이드 게임 프레임워크를 설계 및 구현하였다. 또한 구현한 프레임워크를 기반으로 하는 정형화된 게임 애플리케이션의 개발 절차를 설정하고 이에 따라 예제 게임 프로그램을 개발해 봄으로써 제안한 디자인 패턴이 게임 프레임워크의 구현에 적용 가능한지 검증하고 구현한 게임 프레임워크의 실효성을 분석해 보고자 한다.

2. 관련 연구

안드로이드기반 게임 프레임워크의 설계를 위해 프레임워크의 정의, 핵심 기능, 구성요소 및 설계 방법 등에 대해 알아보고 기존에 제시된 다양한 디자인 패턴 중 프레임워크의 설계에 적용 가능한 디자인 패턴들에 대해 살펴본다.

2.1 프레임워크

프레임워크의 사전적 의미는 복잡한 문제를 해결하거나 서술하는데 사용하는 기본 개념구조로 정의할 수 있으며 특정 범주에 속한 다수의 애플리케이션 생성에 공통으로 활용되는 사고체계라 할 수 있다. 소프트웨어 개발 측면에서는 공동으로 사용하는 라이브러리 혹은 공통 개발구조, 공통 인터페이스로 정의할 수 있다. 프레임워크 내에는 모든 기능이 아닌 핵심 기능 혹은 기본적인 실행흐름만을 제공하고 이를 기반으로 정해진 표준에 따라 확장(Extend) 및 실현(Implement)을 통해 특정 범주에 속한 다양한 애플리케이션들을 개발하게 된다. 프레임워크에는 프레임워크의 구성요소, 각 구성요소가 다루는 내용, 갖춰야할 형식, 구성요소 사이의 관계 및 사용 예제를 담고 있어야 한다. 소프트웨어 분야에서 애플리케이션 프레임워크는 반제품 형태의 구현체로 실체화되며 개발자가 따라야 하는 가이드라인 제시, 개발 범위 설정 및 다양한 도구의 지원 등을 포함한다[1, 2].

소프트웨어 개발자들이 애플리케이션 개발에 프

레임워크를 사용함에 따라 얻을 수 있는 이점들로는 모듈화(Modularity), 재사용성(Reusability), 확장성(Extensibility) 등이 있으며 이를 통해 유지보수의 편이성 확보 및 애플리케이션 개발기간의 단축과 같은 기대효과를 얻을 수 있다[5].

- 모듈화 : 프레임워크는 구현의 상세 내역을 인터페이스 뒤로 캡슐화 함으로써 모듈화를 강화한다. 이는 설계와 구현의 변경에 따른 충격을 국지화하여 소프트웨어의 품질을 향상시킨다.
- 재사용성 : 프레임워크에 의해 제공되는 인터페이스 및 클래스들을 새로운 애플리케이션들을 생성하기 위해 재사용될 수 있는 일반적인 컴포넌트들로 정의함으로써 재사용성을 강화한다. 반복적인 응용프로그램 개발과정 중 중복되는 공통 솔루션들의 재생성을 피할 수 있으며, 특정 범주 내의 애플리케이션에 대한 지식을 강화할 수 있다. 프레임워크 컴포넌트들의 재사용은 생산성 증대, 품질개선, 성능, 신뢰도 및 상호운용성의 증대를 가져올 수 있다.
- 확장성 : 프레임워크는 프레임워크 내에서 호출되는 명시적 콜백(Callback) 메서드들을 제공함으로써 확장성을 강화할 수 있다. 이는 애플리케이션들로 하여금 자신의 인터페이스를 확장 가능하도록 해준다. 콜백 메서드는 구조적으로 특정 문맥상의 애플리케이션에 의해 요구되는 변형들로부터 인터페이스와 애플리케이션 영역 내의 행위들을 분리시켜 애플리케이션의 새로운 서비스나 특성들을 적시에 변경할 수 있도록 보장한다.

프레임워크가 갖춰야할 핵심 기능 및 특성들로는 제어의 역전, 애플리케이션 생애주기 정의 및 주입된 환경설정 주입의 처리 등이 있다.

- 제어의 역전 : 프레임워크의 실행시점 구조는 제어의 역전에 의해 특징지어진다. 이러한 구조는 애플리케이션 처리방식을 프레임워크의 외부 이벤트에 대한 분배처리 기능(Dispatching Mechanism)을 통해 호출되어지는 이벤트 핸들러 객체의 구현을 통해 이루어진다. 이벤트가 발생하면 프레임워크의 Dispatcher는 이벤트별로 사전 등록된 애플리케이션의 이벤트 핸들러 객체의 콜백 메서드를 호출함으로써 대응한다. 제어의 역전은 외부의 이벤트를 개별 애플리케이션이 처리하는 것이 아니라 프레임워크가 이에 대한 대응으로 호출할 애플리케이션의 특정 메서드 그룹을 결정하도록 해준다. 이러한 제어의 역전은 GUI(Graphic User Interface) 애플리케이션 개발 영역에서 흔히 볼 수 있으며 대표적인 것으로 마이크로소프트의 MFC(Microsoft Foundation Class) 기반 애플리케이션 개발을 들 수 있다[12].
- 환경설정 주입 처리 : 프레임워크를 기반으로 다양한 애플리케이션의 동적 생성 처리를 위해서는 각각의 애플리케이션이 갖는 차이에 대한 정보를 환경 설정 파일로 관리하고 이를 프레임워크에 제공함으로써 프레임워크와 결합하는 개별 애플리케이션의 생성을 완성한다. 이를 위해 프레임워크는 주입된 환경 설정 정보의 처리 능력을 보유하여야 한다.
- 애플리케이션의 생애주기 정의 : 동일 영역의 애플리케이션들은 실행 시작 후부터 종료까지 정해진 행위들을 수행하는 생애주기를 정의할 수 있다. 프레임워크는 이러한 생애주기를 추상 클래스의 API로 반영하고 개별 애플리케이션들은 해당 API들을 실현함으로써 애플리케이션의 처리를 완성한다.

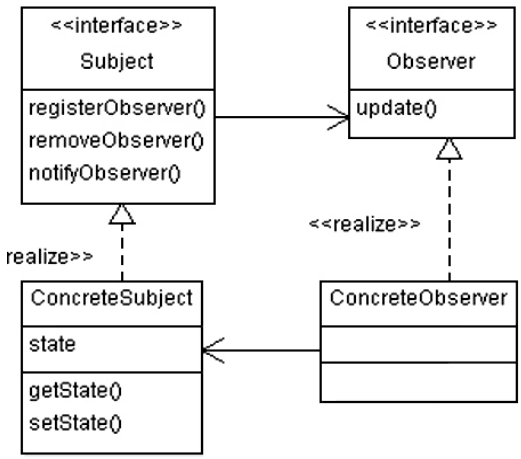
프레임워크는 그들을 확장하기 위해 사용되는 기술에 의해 분류되어 질 수 있으며 White box 프레임워크와 Black box 프레임워크로 구분할 수 있다. White box 프레임워크는 확장성을 구현하기 위해 객체지향 언어의 상속과 동적 바인딩 특성에 의존한다. 이에 따라 응용프로그램 개발자들은 프레임워크의 내부구조에 대해 많은 지식을 갖

도록 요구받는다. 비록 White box 프레임워크가 폭넓게 사용되고는 있으나 프레임워크의 세부 상속 구조와 강 결합된 애플리케이션을 생산하는 경향이 있다. 반면 Black box 프레임워크는 객체 구성과 위임을 통해 프레임워크 내에 삽입될 수 있는 컴포넌트를 위한 인터페이스들을 정의함으로써 확장성을 지원한다. 기존 기능들은 특정 인터페이스에 부합하는 컴포넌트들을 정의하거나 이러한 컴포넌트들을 Strategy 패턴을 이용하여 프레임워크 내에 병합시킴으로써 재사용되어진다. Black box 프레임워크는 개발자로 하여금 광범위한 잠재 유스 케이스에 대응 가능한 인터페이스와 콜백 메서드들을 정의하도록 요구함에 따라 개발하기 더 어려워지는 경향이 있다[12].

2.2 디자인 패턴

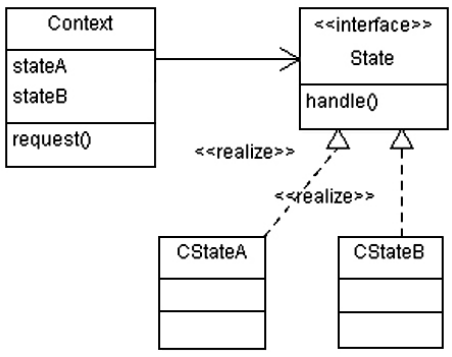
프레임워크의 이점들을 극대화하고 핵심 기능요소의 설계를 위해 지켜야할 설계 원칙으로는 의존성 역전 원칙(Dependency Inversion Principle)을 들 수 있다. 이는 구조적 소프트웨어 설계의 전형인 상위레벨 모듈이 구성요소인 하위레벨 모듈에 의존하는 것과 반대로 하위 구상 모듈들이 상위 추상 모듈에 의존하는 반대 방향의 의존 관계를 갖도록 설계하여 하위 모듈의 변경이 상위 모듈에 전파되지 않도록 하는 것을 말한다. 이러한 설계 원칙에 입각하여 프레임워크의 제어의 역전 기능 설계에 적용할 수 있는 디자인 패턴으로는 Factory 패턴, Dependency Injection 패턴, Reactor 패턴, Observer 패턴 등이 있다[13]. 디자인 패턴은 기능 확장과 재사용이 용이한 소프트웨어 설계 방법 및 아이디어의 집합으로 정의할 수 있으며 마틴 파울러(Martin Fowler)의 Analysys Patterns, 데이빗 헤이(David C. Hay)의 Data Model Patterns 및 GoF(Gang of Four)의 디자인 패턴 등 많은 패턴들이 제시되어 있다. 여기서는 게임 프레임워크 설계에 적용 가능한 주요 디자인 패턴들에 대해서 살펴보고자 한다[8, 14].

- Observer 패턴 : 한 객체의 상태가 변하면 그 객체에 의존 하는 다른 객체들한테 변경 내용이 통지되어 자동으로 내용이 갱신되는 1 : N의 의존성을 갖는 패턴으로 정의 된다([그림 1] 참조). N : M의 의존성을 갖는 경우에 적용 가능한 Reactor 패턴과 유사하나 이벤트의 순차적 처리가 가능한 게임 프레임워크에는 Observer 패턴의 적용이 유리할 것으로 판단된다.



[그림 1] Observer 패턴의 클래스 다이어그램

- State 패턴 : 게임 상태, 캐릭터의 상태에 따라 해당 객체의 행동을 바꾸기 위해 적용 가능한 패턴으로 마치 해당 객체의 클래스가 바뀌는 것과 같은 결과를 얻을 수 있다([그림 2] 참조).



[그림 2] State 패턴의 클래스 다이어그램

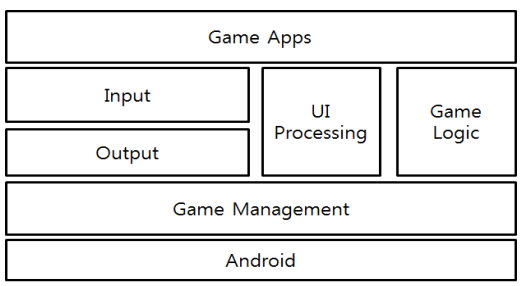
• Singleton 패턴 : 해당 클래스의 객체가 하나만 만들어지고 어디서든지 그 객체에 접근 가능하도록 설계하는 패턴으로 게임 애플리케이션에서는 사운드, 그래픽 등과 같은 다양한 구성요소들이 하나의 객체 내에서 관리되어야 할 필요가 있으며 이를 위해서는 Singleton 패턴의 적용이 가능하다[6].

3. Linked Observer 패턴 기반 게임 프레임워크 설계

본 장에서는 구현하고자하는 하는 안드로이드 기반 게임 프레임워크의 아키텍처 및 게임 애플리케이션의 생애주기를 정의하였으며 기존 프레임워크를 기반으로 하여 새로운 프레임워크를 설계하는 경우에 적용 가능한 Linked Observer 패턴(LOP)을 제시하고 또한, LOP를 적용한 안드로이드 게임 프레임워크의 설계 내역을 제시하고자 한다. 설계 명세는 UML(Unified Modeling Language)을 사용하여 표시한다[11].

3.1 안드로이드 게임 아키텍처 및 생애주기

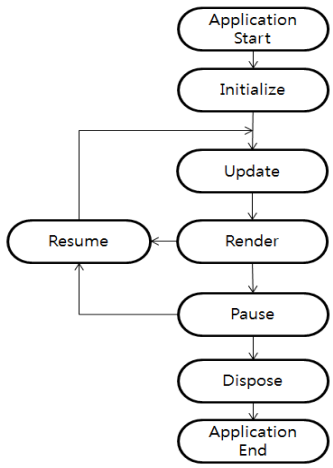
안드로이드 게임 프레임워크의 아키텍처는 안드로이드 플랫폼이 제공하는 안드로이드 애플리케이션 프레임워크를 확장하여 상위 계층에 게임 프레임워크를 구성하며 이를 활용하여 게임 애플리케이션들이 구현된다. 게임 프레임워크의 구성요소는 [그림 3]과 같다.



[그림 3] 게임 프레임워크의 아키텍처

• UI(User Interface) Processing 모듈 : 객체 생성, 닫기, 일시 정지 및 재개 등의 게임 뷰, Scene 및 Character 등의 생애주기 내 콜백 메서드를 정의하며 게임 애플리케이션에서 이를 실현한다.
• Game Logic 모듈 : 입출력 이벤트에 따라 스레드에서 애플리케이션의 생애주기 내 메서드들을 호출하며 필요한 수학 및 물리함수를 구현한다.
• 입출력 관리 모듈 : 사용자 입력 추적, 보조 저장장치로부터 자원을 읽거나 저장하여 게임 애플리케이션에 공급, 사운드 및 그래픽 처리에 대한 모든 처리를 담당한다.
• 프레임워크 통합관리 모듈 : 위의 모듈들을 상호 연결하며 부가적인 기능들이 제공된다.

게임 프레임워크에 기반 한 애플리케이션의 생애주기는 기본적으로 안드로이드 애플리케이션의 생애주기 API들을 활용하고 별도로 게임 애플리케이션의 처리를 위해 추가된 행위들은 게임 로직을 실행하는 스레드에 의해 호출 되며 처리 흐름은 [그림 4]와 같다. 게임 애플리케이션은 프레임워크에서 제공되는 API들을 자신의 애플리케이션에서 실현함으로써 해당 생애주기 동안 자신만의 실행을 처리한다. 각 메서드들의 처리내역은 다음과 같다.



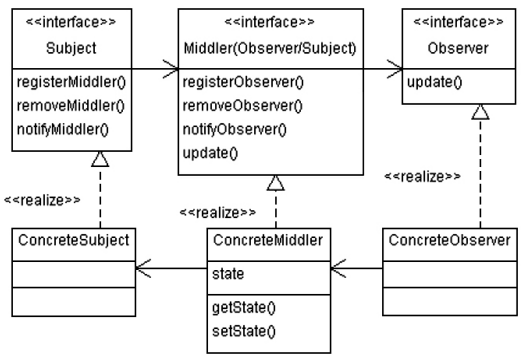
[그림 4] 게임 애플리케이션 생애주기

- Initialize : 객체 생성 시 호출되는 메서드로 객체의 구성요소들을 생성하고 초기화한다.
- Update : 입력에 따른 객체의 상태를 갱신한다.
- Render : 변화된 객체의 상태에 따라 해당 객체의 이미지를 드로잉 한다.
- Resume : 객체의 재활성화 시 호출된다.
- Pause : 객체가 정지되기 바로 직전 호출되며 객체의 상태 정보가 저장된다.
- Dispose : 종료 시점에 호출된다.

3.2 Linked Observer 패턴(LOP)

기존 애플리케이션 프레임워크와 연동하여 새로운 게임 프레임워크를 정의하기 위해서는 안드로이드 프레임워크의 제어의 역전 기능을 연결하여 게임 프레임워크에서도 제어의 역전 기능이 수행되도록 하여야 한다. 이를 위해서는 Observer 패턴의 상호 연결이 필요하며 이를 Linked Observer 패턴으로 새롭게 정의할 수 있다.

입력된 외부정보는 Subject의 구상객체에 등록된 Middler의 구상객체로 notifyMiddler() 메서드를 통해 전달되고 다시 Middler에 등록된 Observer의 구상 객체들에 Middler 구상객체의 notifyObserver() 메서드에 의해 통지되어질 수 있다. 이러한 LOP는 기존 프레임워크를 확장하여 새로운 프레임워크를 설계해야하는 경우 일반적으로 적용되어질 수 있다([그림 5] 참조).



[그림 5] Linked Observer 패턴의 클래스 다이어그램

3.3 게임 프레임워크 설계

게임 프레임워크는 안드로이드 플랫폼에서 제공하는 애플리케이션 프레임워크를 확장하여 설계하며 프레임워크의 핵심 기능인 제어의 역전 기능, 정의된 생애주기의 처리 및 구성요소들의 관리 업무가 반영되어 있어야 한다.

3.3.1 Singleton 패턴을 적용한 GameManager 클래스

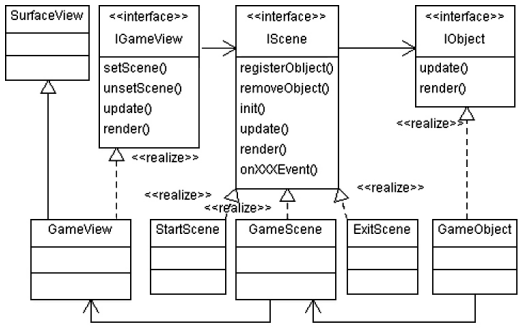
게임 애플리케이션의 실행동안 게임 화면이나 캐릭터들에 의해 참조되어지는 비트맵, 사운드 등의 게임 리소스와 게임 뷰 클래스들의 객체들은 하나의 객체만이 활성화되어야 하기 때문에 이들을 관리하는 GameManager 클래스를 정의하고 GameManager 객체 생성에 Singleton 패턴을 적용한다. 게임 애플리케이션은 실행 후 종료 시까지 GameManager 객체를 통해 필요한 자원을 참조한다.

3.3.2 LOP를 적용한 게임 프레임워크

안드로이드 액티비티(Activity)의 뷰는 게임 시나리오 상에서 정의되는 다양한 화면(Scene)들로 구성되며, 각 Scene들은 해당 Scene의 구성요소들 즉, 배경, 캐릭터, 아이템 등으로 이루어진다. 뷰를 통해 입력된 이벤트 정보는 Scene과 플레이어 등으로 전파되며 해당 객체에 정의된 이벤트 핸들러들에 의해 필요한 처리가 이루어진다. 이러한 객체들 간의 의존 관계를 설계에 반영하기 위해 LOP의 적용이 가능하다.

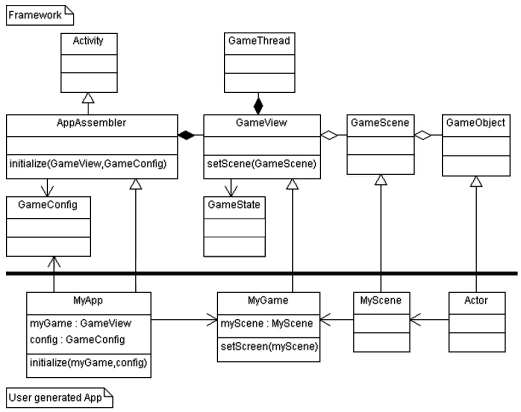
[그림 6]은 LOP를 적용한 게임 프레임워크의 핵심 클래스 다이어그램이다. LOP에서 ConcreteSubject 클래스에 해당하는 클래스는 GameView 클래스로 안드로이드에서 제공하는 SurfaceView를 확장하고 IGameView 인터페이스를 실현하여 구현된다. LOP에서 Middler 인터페이스에 해당하는 IScene 인터페이스는 시작화면(StartScene), 게임화면(GameScene) 및 종료화면(ExitScene) 등 다양한 화면들로 구현된다. GameView 클래스에

는 화면들에 이벤트 정보 전달이 가능하도록 화면들을 등록, 해제할 수 있는 setScene(). unsetScene() 메서드를 제공한다. GameView에 설정된 해당 화면에는 LOP의 ConcreteObserver에 해당하는 GameObject 클래스의 객체들이 등록되어 있으며 사용자의 이벤트는 IScene의 onXXXevent()의 핸들러 메서드를 통해 GameObject의 이벤트 핸들러까지 전파되어 처리된다. 액티비티의 생애주기에 추가하여 게임 프레임워크에서 추가된 Scene과 GameObject의 생애주기 API로 정의된 update(), render()등의 메서드들은 프레임워크 내에 추가된 스레드에 의해 호출된다. 결국 LOP를 구성하는 클래스들과 게임 로직을 구현하는 스레드에 의해 프레임워크의 핵심 기능인 제어의 역전 기능이 수행된다.



[그림 6] LOP를 적용한 클래스 다이어그램

전체 게임 프레임워크의 구조에는 안드로이드 액티비티를 확장한 AppAssembler 클래스가 정의되며 이는 게임 애플리케이션의 실행 진입점 역할을 담당한다. AppAssembler의 initialize() 메서드를 통해 게임 생애주기를 실현한 GameView 클래스의 객체를 게임 구성정보와 더불어 애플리케이션 내에 반영함으로써 사용자의 이벤트에 대해 프레임워크를 경유한 처리가 가능하다. 구성정보의 관리를 위해 GameConfig 클래스가 정의되며 이는 게임 애플리케이션의 초기화에 이용된다. 안드로이드의 액티비티를 상속받은 AppAssembler는 UI(User Interface) 스레드에 의해 실행되는 반면 Game Logic의 주요 처리를 담당하는 메인 스레드는 프레임워크 내의 GameView 객체 내에서 시작된다. 기본적으로 메인 스레드 내에서 UI 스레드에서 발생하는 이벤트를 처리하게 되는데 이때에 스레드간의 메시지 전달 메커니즘은 핸들러를 활용한다. 전체 게임 프레임워크의 클래스 다이어그램은 [그림 7]과 같다.



[그림 7] 게임 프레임워크 주요 클래스 다이어그램

4. 게임 프레임워크의 구현

프레임워크 구현에 있어 제어의 역전 기능, 환경설정 주입 처리 및 애플리케이션의 생애주기별 API의 처리와 같은 요구사항들을 반영하고 새롭게 제시한 디자인 패턴인 LOP의 적용을 위해서는 객체지향 언어의 선택이 바람직하다[7]. 이에 자바 언어를 활용하여 게임 프레임워크를 구현하고 이를 활용한 게임 애플리케이션의 구현 알고리즘을 제시하고자 한다.

4.1 프레임워크의 구현

게임 프레임워크의 구현은 기본적으로 컨테이너의 형태로 구현될 수 있지만 Startable Code인 액티비티를 확장한 AppAssembler 클래스를 프레임워크 내에 배치하여 실행의 주도권을 쥔 Active 라이브러리 형태로 구현 가능하다. [그림 8]에서

는 LOP Middler의 구상 클래스 형태로 동작하는 GameScene 클래스의 구현 코드를 보인다. 이는 IScene 인터페이스를 실현한 다수의 클래스 중 하나로 게임 시나리오를 구성하는 전체 Scene 중 핵심을 차지하는 Scene 중의 하나이다. Scene의 구성요소인 GameObject들을 ArrayList로 관리하며 GameObject들은 ①, ②의 메서드들에 의해 Scene에 등록되거나 폐기된다. 등록된 Game-Object들은 ③, ④ 메서드들과 같은 통지 메서드에 의해 변경 내용이 반영되어진다.

```
public class GameScene implements IScene {
GameManager app;
private ArrayList<GameObject> m_list;
private Actor actor, actor1;
public GameScene() {
            super();
            app = GameManager.getInstatnce();
            m_list = new ArrayList<GameObject>(); }

public void registerObject(GameObject object) {.....①
            m_list.add(object); }

public void removeObject(GameObject object) {.....②
            int i = m_list.indexOf(object);
            if(i >= 0)   m_list.remove(i);   }
public void init() {
          actor = new

Actor(app.getBitmap(R.drawable.soldier_comes),this);
        actor.initSpriteData(200, 200, 10, 13, 100, 0); }

public void update() {.....③
        long GameTime = System.currentTimeMillis();
        for(int i = 0; i < m_list.size(); i++){
        GameObject spr = (GameObject) m_list.get(i);
        spr.update(GameTime); }}

public void render(Canvas canvas) {.....④
        for(int i = 0; i < m_list.size(); i++){
        GameObject spr = (GameObject) m_list.get(i);
        spr.render(canvas);   } ... }
```

[그림 8] GameScene 알고리즘

4.2 프레임워크를 활용한 게임애플리케이션 구현

구현된 게임 프레임워크를 활용하여 게임 애플리케이션을 생성하는 경우에는 정형화된 절차에 따라 개발이 가능하다. 먼저 기획된 시나리오 상에 필요한 Scene들을 구현하는 작업으로 Scene은 사용자 이벤트에 따라 전환이 이루어지며 초기화면, 종료화면, 도움화면 등을 포함한다. GameScene은 Scene을 구성하는 구성요소(배경화면, 캐릭터, 효과 등)도 다양하고 게임 상태 값에 따라 변화하는 요소도 많기 때문에 프레임워크 내에 정의된 클래스를 상속하고 사용자 정의 메서드들을 추가하여 구현한다. 다음으로 GameView를 확장하여 자신의 게임로직을 반영한 MyGame 클래스를 구현한다. 이 클래스에서는 자신의 게임에서 게임 라이프 사이클 동안 처리하고자하는 메소들을 상속을 통해 정의 하여야 하며 해당 메서드 내에서 필요에 따른 Scene을 setScene 메서드를 통해 화면에 표시하여야 한다. 게임 사용자의 입력에 따른 상태정보를 GameState 객체를 통해 MyGame 클래스에 반영하여야 하며 이를 토대로 GameObject 객체들이 렌더링되도록 한다. 구현된 MyGame 객체는 AppAssembler를 상속 구현한 MyApp의 initialize 메서드를 통해 환경 설정정보를 보유한 GameCofig 객체와 함께 프레임워크 내로 주입되고 프레임워크는 주입된 클래스들을 바탕으로 애플리케이션의 실행을 관리한다.

4.3 LOP 기반 프레임워크의 실효성 분석

제시한 LOP가 유의미하기 위해서는 이를 적용하여 구현된 게임 프레임워크가 게임 애플리케이션 개발에 활용되고 개발된 게임이 안드로이드 실행환경에서 에러 없이 실행되어야 한다. 또한 구현된 게임 프레임워크의 실효성 검증을 위해서는 프레임워크를 이용한 애플리케이션의 개발기간, 사용의 편이성 및 활용도 등을 통해 분석해 볼 수 있다.

4.3.1 게임 애플리케이션의 완성도 및 개발기간 단축

프레임워크 기반 게임 애플리케이션의 완성도 측정을 위해서는 Non-Framework 기반의 게임과 기능상으로 동일함을 증명하는 테스트를 수행함으로써 증명할 수 있다. 게임 애플리케이션의 테스트는 게임의 요구명세에 대해 입출력 이벤트 처리 기반으로 테스트할 수 있는 블랙박스 테스트 방식으로 수행하였으며 주요 테스트 케이스들에 대한 실행 결과는 <표 1>과 같다[3].

〈표 1〉 테스트 결과표

테스트 케이스		예상결과	정상처리
Scene 전환 이벤트 처리		StartScene에서 GameScene으로의 전환	○
GameView의 이벤트 처리	Scene 처리	GameScene의 배경 변경	○
	Object 처리	GameObject의 행동 반응	○
게임 레벨 선택 처리		레벨 선택 후 해당 게임 실행	○
비정상 종료 후 재실행		이어서 실행	○
정상종료 처리		정상종료 후 결과 저장	○

또한 동일한 게임 애플리케이션을 프레임워크를 이용하지 않은 경우와 LOP를 기반으로 구현한 프레임워크를 이용하여 개발하는 경우를 게임 개발 기간 측면에서 비교해 봄으로써 LOP 및 게임 프레임워크의 실효성을 검증할 수 있다. <표 2>에서는 [그림 9]와 같은 내용으로 구성된 교육용 게임 애플리케이션의 구현 결과를 비교하였다. 개발기간은 개발자의 숙련도에 따라 다를 수 있으므로 개발하는 애플리케이션의 code size로 예측 가능하다. LOP 게임 프레임워크를 기반으로 구현된 게임 애플리케이션이 Non-Framework 대비 43%의 코드만으로 구현되었음을 확인할 수 있다. 다만, 프레임워크를 이용하는 경우 이를 활용하기 위한 학습기간이 소요되는 문제점이 있으나 이는 한번 학습으로 다수의 게임 개발에 적용이 가능하고 프레임워크 없이 개발을 하는 경우 학습해야하는 API들에 비해 간단하다고 할 수 있다. 리소스가 대부분을 차지하는 Apk 파일 크기는 별다른 차이를 보이지 않았다.

〈표 2〉 code size 비교

	Non-Framework Based	Proposed Framework Based
Code size	970 lines	420 lines
Apk file size	8.54MB	8.32MB

[그림 9] 예제 프로그램

4.3.2 게임 프레임워크의 비교

다수의 안드로이드용 게임 프레임워크가 존재하고 있으나 대부분 상세 설계 내역이 공개되어 있지 않고 개발자를 위한 문서들도 충분하지 않은 실정이다. 모듈화, 재사용성 및 확장성 등과 같은 설계 관점에서 이들 프레임워크 간 비교는 한계가 있어 구현된 결과물에 대한 개발자들의 이용 편이성 및 활용도 등을 중심으로 <표 3>에서와 같이 제안한 프레임워크와 소스가 공개된 안드로이드 플랫폼용 프레임워크들 중 게임 개발 사례가 있는 것들을 중심으로 비교하여 보았다.

〈표 3〉 프레임워크 비교[10]

	개발된 게임수	Orientation	Source Size	Platform	문서화
LibGDX	23	2D,3D	54MB	Cross- Platform	중
AndEngine	22	2D	2.7MB	Android	하
Moai	6	2D	5.0MB	Cross- Platform	중
제안 Framework	-	2D	1.6MB	Android	상

AndEngine의 경우 Android 플랫폼 전용으로 소스코드의 규모가 작아 분석이 용이한 장점이 있으나 문서화의 경우 단순 API만을 제공하고 있는 상황이어서 개발된 예제를 통해 학습이 이루어져야하는 단점이 있다. LibGDX의 경우 풍부한 기능을 제공하고 다양한 플랫폼에서 활용할 수 있는 장점이 있으나 코드의 규모가 커 분석이 어렵고 안드로이드용 라이브러리만으로도 200KB가 넘어 애플리케이션의 규모도 커지는 단점이 있다. Moai 프레임워크는 크로스 플랫폼을 지향하며 비교적 코드 규모가 작아 분석이 용이하나 아직 이를 이용한 게임 개발 건수가 6건으로 적다. 문서화의 경우 API뿐만 아니라 개발 절차를 설명한 문서가 있어 게임 개발 접근이 용이하다. 본 논문에서 구현한 프레임워크의 장점은 설계 내역의 공유를 바탕으로 다수의 프레임워크 개발자들이 협업을 통해 개선해 나갈 수 있다는 점이며 제안한 프레임워크를 활용한 공개된 게임의 개발 내역이 없는 점은 단점으로 지적할 수 있다.

5. 결 론

특정 도메인 내의 애플리케이션 개발을 위해 동일한 부분이 반복적으로 구현되어진다면 이들은 라이브러리나 컴포넌트형태로 개발되어 재사용하는 것이 유리하다. 또한 애플리케이션 개발의 생산성 향상을 위해서는 입력된 이벤트 처리의 주도권을 플랫폼이 담당하고 애플리케이션 개발은 생애주기 동안 해당 이벤트의 처리를 위한 핸들러만을 구현하는 프레임워크 기반 애플리케이션 개발 방법을 채택하는 것이 필요하다.

본 논문에서는 기존 프레임워크를 확장하여 새로운 영역의 애플리케이션 개발 프레임워크를 설계하는 데 적용 가능한 디자인 패턴으로 Linked Observer 패턴을 제시하였고 이를 기반으로 제어의 역전 기능, 게임 애플리케이션의 생애주기 정의 및 처리를 담당하는 안드로이드 게임 개발 프레임워크를 설계 및 구현하였다. 프레임워크가 의미를 갖기 위해서는 이를 기반으로 하는 정형화된 애플리케이션 개발 절차가 제시되고 이에 따라 개발된 애플리케이션들이 플랫폼 상에서 오류 없이 실행되어야 한다. 본 논문에서는 구현된 프레임워크를 토대로 한 게임 애플리케이션 개발 절차를 제시하였으며 LOP 및 프레임워크의 실효성 분석을 위해 게임 애플리케이션을 구현하여 게임의 완성도 및 개발기간을 프레임워크를 사용하지 않고 구현한 경우와 비교해 보았다.

향후 본 논문에서 구현한 프레임워크가 안드로이드 장치의 플랫폼 내에 포함되어 활용되기 위해서는 해당 프레임워크를 기반으로 다수의 게임 애플리케이션들이 개발되어야 하며 이런 과정을 통해 설계의 완전성을 보완하고 게임 애플리케이션 개발 절차를 표준화하는 과정이 필요하다.

참 고 문 헌

[1] 김운용, 최영근, “디자인 패턴의 점진적 통합을 이용한 패턴지향 소프트웨어 개발 방법”, 「정보처리학회논문지」, 제10-D권, 제5호(2003), pp.763-772.

[2] 김종수, 김태석, “디자인 패턴을 이용한 네트워크 턴 게임 API 설계 및 구현”, 한국멀티미디어학회논문지, 제7권, 제11호(2004), pp. 1588-1596.

[3] 서광익, 최은만, “객체지향 소프트웨어를 위한 주요 블랙박스 테스트 기법들의 비교”, 「정보과학회 논문지」, 제33권, 제1호(2006), pp.1-16.

[4] Android Open Source Project, *Android SDK*, http://developer.android.com/sdk/index.html.

[5] Fayad, M. and D. Schmidt, “Object-Oriented Application Frameworks”, *Communications of the ACM*, Vol.40, No.10(1997).

[6] Gamma, E., R. Helm, R. Johnson, and J. Vlissides, *Design Pattern : Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[7] Gosling, J. and F. Yellin, Java Team, *The Java Application Programming Interface Volume1*, Addison-Wesley, 1996.

[8] Kim, J. and T. Kim, "Design of Network-based Game Using the GoF Design Patterns", *Journal of Korea Multimedia Society*, Vol.9, No.6(2006), pp.742-749.

[9] Martin Fowler, *Inversion of Control Containers and the Dependency Injection pattern*, Thought Works at http://www.fowler.com.

[10] Mobile Game Engines, *Compare Abdroid Game Engines*, http://mobilegameengines.com/android/game_engines.

[11] OMG, *UML2.0 Infrastructure Specification*, http://www.omg.org/uml/2.0/infrastructure/PDF.

[12] Roberts, D. and R. Johnson, *Evolving Frameworks : A Pattren Language for Developing Object-Oriented Frameworks*, http://st-www.cs.illinois.edu/users/droberts/evolve.html.

[13] Schmidt, D., "Reactor : An Object Behavioral Pattern for Demultiplexing and Dispatching Handles for Synchronous Events", *Proceedings of the First Pattern Languages of Programs conference in Monticello, Illinois,* 1994.

[14] Schmidt, D., R. Johnson, and M. Fayad, "Software Patterns", *Communications of the ACM*, Vol.39, No.10(1996).

◈ 저 자 소 개 ◈

서 문 석 (msseo@sehan.ac.kr)

현재 세한대학교 컴퓨터응용기술학과 부교수로 재직 중이며, 한국정보통신대학원 공학부에서 정보보안을 전공하였다. 금융결제원, ㈜시큐아이에 근무하였으며 주요 관심분야는 암호이론, 모바일보안, 소프트웨어공학 등이다.