Journal of Korea Multimedia Society Vol. 7, No. 11, November 2004(pp. 1588-1596)

디자인 패턴을 이용한 네트워크 턴 게임 API 설계 및 구현

김종수[†], 김태석[††]

요 약

현재 인터넷에서 서비스되고 있는 실시간 네트워크 게임을 개발하는 경우, 개발 인원과 시간이 많이 투입되는 프로젝트이기 때문에, 게임 제작 시 기존에 작성되어 있는 코드를 재사용이 가능하도록 설계하는 것은 중요한 일이다. 코드의 재사용을 극대화하기 위해서 연구되고 있는 분야가 디자인 패턴과 관련한 분야인데, 보다 효율적인 게임 제작을 위해 잘 정의된 디자인 패턴을 이용한 다양한 설계기법과 그 적용 예가 개발자에게 제공된다면, 보다 나은 게임 API(Application Programming Interface)를 개발할 수 있고, 또한 API를 바탕으로 한 게임 개발 전용 프레임워크의 개발이 가능하다. 본 논문에서는 2종류의 네트워크 턴 게임 설계와 구현에 있어서, GoF(Gang of Four)가 제안한 디자인 패턴들을 적용하였다. 이를 통해서 네트워크 턴 게임 제작에 있어서 기존에 개발된 게임 API를 효과적으로 재사용하는 효율적인 설계 기법을 제안한다.

Design and Implementation of the Network Turn Game for API Using Design Patterns

Jong-Soo Kim[†], Tai-Suk Kim[††]

ABSTRACT

Developing a real time network game that is serviced on the internet requires a lot of work, time and manpower. Therefore, it is important to design a game in such a way so that existing design codes could be used. The area which is being studied to maximize the reusability of the existing design code is related to design patterns. If the developers are given various design techniques and application examples that use well defined design patterns, they will be able to make better game API(Application Programming Interface) and to invent a framework for game development based on this API as well. In this paper, the design patterns that the GoF(Gang of Four) proposed have been applied to develop and implement two kinds of network turn games. In the process, efficient design techniques will be proposed for the effective reuse of the existing game API.

Key words: Network Game(네트워크 게임), Design Patterns(디자인 패턴), UML

1. 서 론

인터넷 인프라 발전에 힘입어 다양한 온라인 네트워크 게임의 수요가 급증하고 있다. 통신망이 고속화, 대용량화 되면서, 기존에는 거의 불가능할 것으로 예상되었던 실시간 전략 시뮬레이션 게임들도 제작되고 있다.

객체 지향적으로 게임 소프트웨어를 설계하고 개발하기 위한 기법은 회사의 중요 자산이기 때문에 철저하게 보안되고 있다. 게임 개발은 분석, 설계, 구현, 테스트, 리팩토링과 같이 몇 단계로 나누어지는데, 각각의 단계는 모두 중요하므로, 문서 작업도 철

※ 교신저자(Corresponding Author) : 김태석, 주소 : 부산시 부산진구 엄광로(가야 산24번지)(614-714), 전화 : 051)890-1707, FAX : 051)890-1724, E-mail : tskim@deu.ac.kr
접수일 : 2004년 5월 17일, 완료일 : 2004년 6월 3일
[†] 준회원, 동의대학교 영상 미디어센터 PM연구원
(E-mail : seatree@deu.ac.kr)
[††] 종신회원, 동의대학교 공과대학 소프트웨어공학과 교수
※ 본 논문은 2004년도 동의대학교 교비지원(2004AA159)에 의해 수행되었음.

저히 보안되고 있다. 이러한 게임 제작 기술의 보안성 때문에, 게임 개발 기술에 적용되는 객체 지향 설계기법의 객관적인 평가가 이루어지지 못하고 있다[1].

본 논문에서는 사용자가 자신의 순서를 기다려서 게임하는 네트워크 턴 게임 개발에 있어서, 객체 지향 패러다임을 적용하여 이와 유사한 게임 제작에 필요한 API를 제작하였다. 소프트웨어 개발에 있어서 객체 지향 패러다임을 적용하는 이유는 여러 가지가 있지만, 분산된 개발을 가능하게 하고, 기존에 개발된 코드의 재사용이 쉽다는 장점이 있기 때문이다. 그렇지만 네트워크 게임 설계에 있어서 유연하고 재사용이 가능한 설계를 처음부터 정확하게 하기란 매우 어렵다. 일반적으로 애플리케이션의 설계는 많은 시간이 투자되며, 설계를 완성하기까지는 설계를 여러 번에 걸쳐서 수정 및 보완해야 하는 경우가 대부분이다. 또한 객체 지향 설계에 경험이 많은 사람들은 좋은 설계를 할 수 있지만, 초보자들이 애플리케이션 제작에 있어 객체 지향 개념을 적용하기란 쉽지 않다.

본 논문의 목적은 게임 개발자들이 네트워크 턴 게임을 설계하고 구현하는데 있어서 효과적으로 사용할 수 있는 GoF의 디자인 패턴을 제시한다. 그래서 이미 검증된 디자인 패턴들로 만들어진 게임 API들이 애플리케이션의 추가적인 개발이 있을 시에 효과적으로 재사용할 수 있음을 보인다.

2. 네트워크 게임 설계의 관련 연구

본 논문에서 구현된 네트워크 게임의 설계에 있어 가장 큰 구조는 클라이언트/서버 기반의 다계층 구조(multi-tier architecture)다. 이 구조는 일계층(one-tier)이나 이계층(two-tier) 구조에 비해 광범위한 데이터를 읽기 쉽게 만들어 주고, 네트워크를 통한 접근을 쉽게 하고, 데이터 은닉을 용이하게 한다는 장점이 있다[2].

애플리케이션 설계의 기본 구조는 그림 1과 같은 MVC(Model/View/Controller) 디자인 패턴을 적용하였다. MVC는 기존에 개발된 클래스의 유연성과 재사용성을 증대시킨다. MVC에서 뷰와 모델간에 자료를 요청하고 보내는 프로토콜을 만들어 종속성을 없앴으며, 뷰는 외형이 모델의 상태를 반영하도록 작성하였다[3].

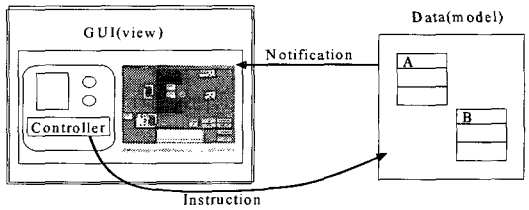


그림 1. MVC(Model/View/Controller) 디자인 패턴

일반적으로 게임 소프트웨어 설계에 객체지향 패러다임이 가지고 있는 장점을 최대한으로 적용한다는 것이 쉽지 않다. 특히 절차적인 프로그래밍 기법에 익숙해져 있는 개발자의 경우는 객체지향언어를 가지고 절차적으로 프로그래밍하는데 익숙해져 있어, 객체지향언어의 장점인 작성된 코드의 재사용성과 유용성의 장점을 살리기 힘들다[4].

특히 소수의 개발 인력이 시스템을 설계하는 경우는 시스템 설계에 대한 지식이 많이 없으므로 정확한 설계를 하기란 매우 어렵다. 이러한 문제점을 해결해 줄 수 있는 대안이 최근에 많이 연구되고 있는 디자인 패턴을 이용하는 방법이다. 디자인 패턴을 이용한 효율적인 설계는 아키텍처의 재사용이 쉽다는 장점을 가지고 있다.

현재도 시스템 개발에 있어서, 여러 가지 다양한 디자인 패턴 개발과 적용에 대한 연구가 이루어지고 있지만, 본 논문에서는 여러 가지 다양한 디자인 패턴들 중에서 지금까지 검증된 디자인 패턴이라고 할 수 있는 GoF(Gang of Four)가 제안한 디자인 패턴을 위주로 설계 최적화 기법을 제안한다. 본 논문의 주된 연구 범위는 GoF가 제안하고 있는 생성패턴(Creational Patterns), 구조패턴(Structural Patterns), 행위 패턴(Behavioral Patterns)이다[5]. 디자인 패턴을 사용함으로써, 애플리케이션의 구조를 간단하면서 이해하기 쉽게 만들 수 있다.

3. GoF의 디자인 패턴 고찰

현재 학계에 발표된 여러 가지 디자인 패턴들은 객체지향 시스템 설계 문제를 해결하는 일반적인 설계 개념을 설명하고 개념에 맞는 이름을 제공한다. 게임 소프트웨어를 객체 지향적으로 설계하기 위해 사용될 수 있는 여러 가지 패턴들이 있지만, 이번 연구의 대상은 GoF가 제안한 디자인 패턴을 어떻게 네트워크 게임 개발에 적용할 것인가 하는 것이다.

GoF가 제안한 여러 가지 디자인 패턴들이 있지만 본 논문에서는 3개의 큰 디자인 패턴 분류인 생성, 구조, 행위패턴에 속하는 다음의 패턴들을 적용한다.

생성패턴은 객체의 생성 방식을 결정하는 포괄적인 방법을 제공하는 패턴으로 클래스를 정의하여 객체를 생성하고 구조화하는 방법과 캡슐화하는 방법을 제시한다. 카드를 사용하는 네트워크 게임 구현에 있어서, 적용될 수 있는 생성 패턴에는 추상화 팩토리(Abstract Factory), 빌더(Builder), 팩토리 메소드(Factory Method), 싱글톤(Singleton) 패턴이 있다.

구조 패턴은 더 큰 구조를 형성하기 위해 클래스와 객체를 어떻게 합성하는가와 관련된 패턴이다. 구조 패턴은 상속 기법을 이용하여 인터페이스나 구현된 객체를 합성한다. 여러 가지 구조 패턴들이 있지만, 이번 연구에서 주로 관심이 되었던 패턴들은 어댑터(Adapter), 데코레이터(Decorator) 패턴이다.

행위패턴은 객체의 행위를 조직화, 관리, 연합하는데 사용되는 객체의 내부 결합에 초점을 맞춘 패턴으로 객체간의 기능을 배분하는 일과 같은 알고리즘 수행에 주로 이용되며, 객체나 클래스에 대한 유형을 정의하는 것이 아니라 객체들 간의 연동에 대한 유형을 제시한다. 게임 설계에 적용될 수 있는 여러 가지 행위 패턴이 있지만, 게임 제작에 Chain of Responsibility, 상태(State)와 같은 패턴들이 응용될 수 있다.

4. 네트워크 턴 게임 제작을 위한 API 설계

본 논문에서는 네트워크 게임 설계에 있어서, GoF가 제안한 디자인 패턴을 다양하게 적용하였다. 그리고 객체지향 패러다임이 가지고 있는 분산 개발과 소프트웨어 재사용이라는 측면에서 네트워크 게임을 구성하는 최적의 설계기법을 제시한다.

4.1 네트워크 턴 게임 설계에서 생성 패턴 적용

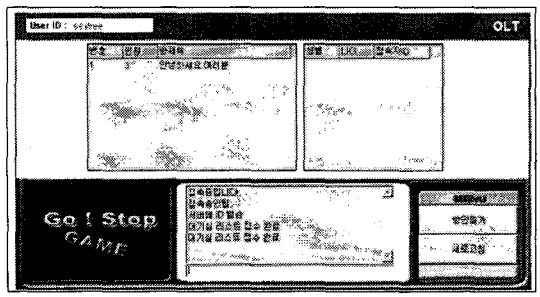
카드류를 사용하는 네트워크 게임에서 일반적인 게임은 사용자가 게임 서버에 로그인하여 대기실에 참여한 후, 게임이 진행되고 있는 방을 찾거나 새로 만들어서 다른 사용자와 같이 게임을 진행한다. API 설계에 사용될 수 있는 여러 가지 생성 패턴이 있지만, 우선적으로 관심 대상은 MVC 디자인 패턴에 따른 각각의 클래스들이 어떻게 재사용되어 질 수 있는가 하는 것으로 뷰와 관련된 GUI(Graphical User Interface) 설계, 프로토콜의 정의와 관련된 컨트롤러의 설계, 데이터와 관련된 모델 설계부분이다.

4.1.1 Abstract Factory 패턴 적용

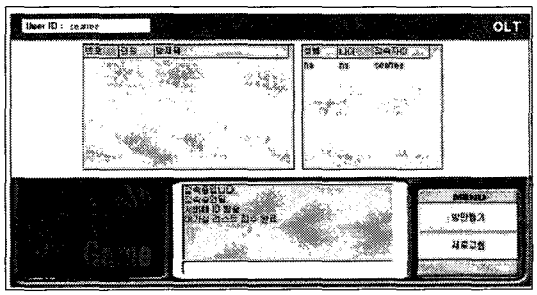
아래의 그림 2는 일반인들에게 잘 알려진 카드를 사용하는 게임인 일명 "고스톱"과 "훌라"라고 불리는 게임의 대기실 GUI 설계 예를 보여 준다. 두 그림을 비교해 보면, 게임에서의 대기실은 거의 같은 GUI 디자인 패턴을 가지고 있다는 것을 알 수 있다.

그림 2의 (a)와 (b)를 살펴보면, 클래스를 얼마나 효율적으로 설계하느냐에 따라서 충분히 클래스의 재사용과 같은 편의성을 제공해 줄 수 있다는 가능성을 보여 주고 있다.

그림 3의 추상화 팩토리(Abstract Factory) 패턴이 두 개 또는 여러 개의 GUI 디자인에서 연관된 객체의 클래스들 중의 하나를 반환하고자 할 때 사용할 수 있다. 추상화 팩토리는 여러 팩토리들 중의 하나를 반환하는 팩토리 객체로 사용되는데, 추상화 팩토리가 적용된 실례로 컴퓨터시스템에서 다중 "룩앤필(look and feel)"을 지원하는 것과 같이 게임 제작에 있어서 그림 2의 (a)와 (b)도 다중 룩앤필로 해석될 수 있다.



(a) 고스톱 게임 대기실 GUI 설계



(b) 훌라 게임 대기실 GUI 설계

그림 2. 두 게임의 대기실 GUI 설계 비교

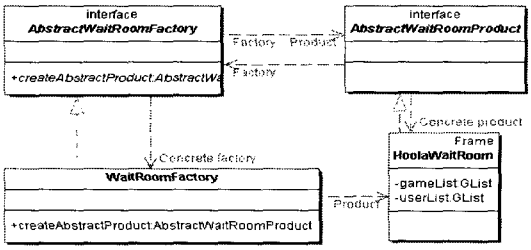


그림 3. 추상화 팩토리(Abstract Factory) 패턴

여기서 팩토리는 사용하는 시스템이 그림 2의 (a)와 같으면 (a)와 관련된 대기실 GUI, (b)와 같으면 (b)와 관련된 GUI를 반환할 수 있음을 볼 수 있다. 즉 두개의 클라이언트 측 GUI는 추상화 팩토리 패턴이 적절하게 적용될 수 있음을 볼 수 있다.

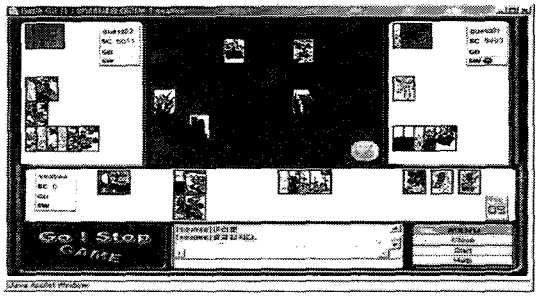
추상화 팩토리 패턴을 사용함으로써, 추후 추가될 수 있는 또 다른 애플리케이션을 쉽게 추가 할 수 있다. 그리고 특히 여러 개의 애플리케이션을 한 클라이언트에게 서비스하는 경우에 같은 자원을 공유할 수 있어서 자원의 효율을 극대화 할 수 있다. 대기실의 GUI를 구성하고 있는 여러 가지 컴포넌트들 중에서 같은 역할을 하는 컴포넌트도 추상화 팩토리 패턴이 적용되었다.

4.1.2 Builder 패턴 적용

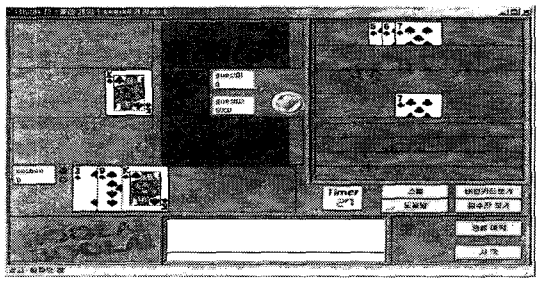
게임 룸 설계는 게임에 참여하는 사용자들에게 얼마만큼 재미있는 요소를 제공해 줄 것인가 하는 것이 중요한 부분이다.

그림 4는 각각 다른 게임의 게임룸 설계 예를 보여준다. 각각은 카드를 사용하는 게임이지만, 그림 (a)의 카드와 (b)의 카드는 형태와 모양이 틀리고, 게임에서 사용되는 카드의 개수도 다르다. 그리고 게임에 참여하는 사용자의 경우도 그림 (a)의 게임룸은 3명, 그림 (b)의 게임룸은 2명에서 5명이 참여할 수 있다. 그리고 카드가 위치하는 장소도 서로 다른 것을 알 수 있다.

그림 4에서 보는 바와 같이 두 개의 게임 룸의 설계는 앞에서 본 그림 2와 많은 차이가 있지만, 객체지향 설계 기법의 기본이 되는 추상화의 개념을 적용해 보면, 카드나 화투와 같이 다른 형으로 나타나지만, 게임에서 같은 역할을 하는 클래스가 있다는 것을 알 수 있다. 게임룸의 설계에서도 마찬가지로 그림 3에서 보았던 추상화 팩토리 패턴을 사용할 수 있다.



(a) 고스톱 게임룸 GUI 설계



(b) 홀라 게임룸 GUI 설계

그림 4. 두 게임의 게임룸 GUI 설계 비교

GUI를 구성하고 있는 요소 중에서 추가적으로 적용이 가능한 패턴은 그림 5의 빌더(Builder) 패턴인데, 추상화될 수 있는 화투나 카드에 적용할 수 있다.

Card 클래스를 구현하는데 있어서, 각각의 카드는 별도의 객체로 취급될 수 있다. 카드의 구성은 일반적인 카드(StandardTrump)인 1에서 9까지의 숫자를 나타내는 카드와 10에서 13의 숫자를 나타내는 그림 카드가 종류별로 4개 있고, 조커(Joker)라고 불리는 특수한 형태의 카드가 있다.

카드의 구현에 빌더 패턴을 사용함으로써, 여러 개의 객체로 생성되어야 하는 카드는 각각을 클래스로 관리할 경우 클래스의 개수가 많아 질 수 있으므로, 복잡한 객체를 생성하는 방법과 표현하는 방법을 정의하는 클래스를 StandardTrump, Joker와 같이

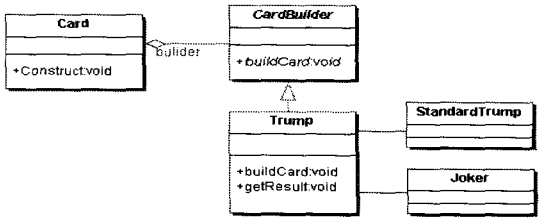


그림 5. 빌더(Builder) 패턴 다이어그램

별도로 분리하여 동일한 구축 공정에서 서로 다른 형태의 카드를 생성할 수 있다는 장점이 있다.

4.1.3 Factory Method 패턴 적용

게임룸을 구성하는 객체 중에서, 게임에 참여하는 사용자 클래스의 정의 시에 고려해 볼 수 있는 패턴은 그림 6과 같은 팩토리 메소드(Factory Method) 패턴이다.

일반적으로 게임에 참가하는 User 객체는 3종류가 있다. 각각은 게임의 진행을 맡게 되는 유저와 게임에 참여하는 유저, 게임을 관람하는 유저다. Users 객체가 여러 개의 User 객체를 관리한다. 게임 진행 시 각각의 객체들은 같은 메소드를 사용하여 게임을 진행하더라도 다른 행동양식을 해야 한다. 그림 6에서 보듯이 팩토리 메소드 패턴을 적용한 Playing 인터페이스를 수행하는 User 객체는 게임 내에서 각각 다른 행동을 한다. 이렇게 함으로써 또 다른 형태의 사용자 추가되어야 하는 경우에도, 새로운 클래스의 생성 없이 새로운 객체를 추가할 수 있다는 장점이 있다.

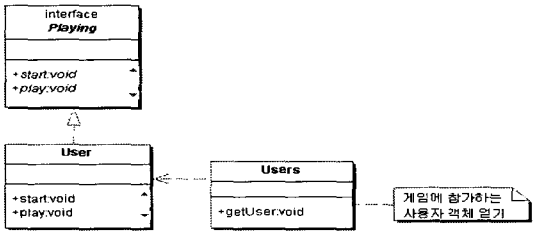


그림 6. 팩토리 메소드(Factory Method) 패턴

4.1.4 Singleton 패턴 적용

네트워크 게임에서 게임 서버는 여러 명이 공유하는 자원을 효율적으로 분배해주고, 게임 참가자 간에 커뮤니티를 형성하게 해준다는 측면에서 중요한 기술이다. 그리고 안정성 있는 서버의 구현과 재미있는 게임 요소를 위한 인공 지능 추가라는 면에서 항상 개선의 여지가 많은 부분이다.

게임 서버의 설계에서도 추상화 팩토리 패턴을 적용할 수 있는 부분이 많이 있다. 그리고, 추가적으로 생각해 보아야 할 디자인 패턴은 그림 7과 같은 싱글톤(Singleton) 패턴이다. 만약 다른 형태의 카드 게임을 같이 관리하는 게임 서버의 구성이 필요하다고 가정한다면, 이 게임 서버는 컴퓨터 자원의 상당한

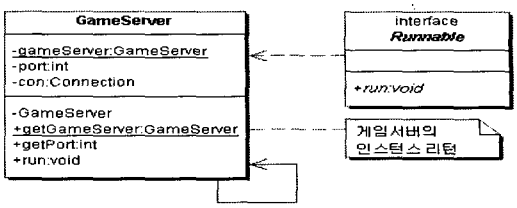


그림 7. 싱글톤(Singleton) 패턴

부분을 차지하게 된다. 이러한 이유로 게임 진행을 총괄하는 서버를 구성하는 GameServer 클래스의 인스턴스는 하나일 필요가 있다.

다른 클래스에서도 GameServer의 객체를 생성해서 참조할 수 있지만, 메모리를 사용하는데 대한 부담이 증가한다. 이러한 문제는 GameServer 클래스 설계에 그림 7과 같이 싱글톤 패턴을 적용함으로써 해결될 수 있다.

4.2 네트워크 턴 게임 설계에서 구조 패턴 적용

구조 패턴은 더 큰 구조를 형성하기 위해 어떻게 클래스와 객체를 합성하는가와 관련된 패턴이다. 자바의 어댑터(Adapter) 클래스가 구조 패턴을 적용한 예다. 본 논문에서 연구 대상이 된 구조 패턴은 어댑터(Adapter), 데코레이터(Decorator) 패턴이다. 어댑터 패턴은 다양한 형태의 Card클래스 구현에 적용 가능하고, 데코레이터 패턴은 카드를 꾸미기 위한 클래스 구현에 적용 가능하다. 각각의 패턴들은 구체적으로 다음과 같이 적용되었다.

4.2.1 Adapter 패턴 적용

자바 언어를 사용함으로써, 객체 지향적 소프트웨어의 설계와 구현에 기본적으로 어댑터(Adapter) 패턴을 사용하고 있다고 보아야 한다. 그림 8은 Card 클래스 구현에 어댑터 패턴이 적용된 예다.

추상클래스인 CardAdapter는 이미 구현된 add(), insert(), remove(), setImage()와 같은 메소드를 가지고 있다. 이것을 상속받는 KoreanCard와 Trump 클래스는 CardAdater 클래스와는 행동양식이 다를 수 있는데, CardAdapter를 상속받아서 자신에 맞는

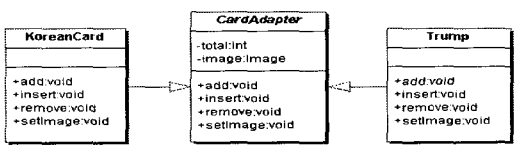


그림 8. 어댑터(Adapter) 패턴

속성과 행동 양식을 가지도록 구현할 수 있다.

이러한 방법을 사용함으로서, KoreanCard와 Trump외에 새롭게 정의되는 Card 클래스가 생기더라도 CardAdapter를 이용할 수 있고, 새로운 Card 클래스가 정의하고 있지 않는 속성과 행동양식을 코드의 수정 없이 사용할 수 있다는 장점이 있다.

4.2.2 Decorator 패턴 적용

게임 그래픽 구현과 관련된 여러 가지 객체들이 있지만, 카드 객체는 게임 객체를 구성하는 중요한 요소이므로, 다양한 구현 방법이 필요하고 새로운 기능을 추가할 필요가 자주 발생할 수 있다. 또한 카드 외의 컴포넌트에도 스크롤링과 같은 행위나 테두리와 같은 속성을 추가해 줄 필요가 생길 수 있다.

새로운 기능의 구현을 위해 상속과 같은 일반적인 방법을 사용할 수 있다. 즉 이미 존재하는 클래스를 상속받고, 또 다른 클래스로부터 테두리를 상속받아서 이 서브클래스들의 인스턴스들이 테두리, 이미지, 애니메이션을 가질 수 있도록 하는 방법이 가능하지만 같은 의미를 가지는 여러 개의 클래스가 생기므로 별로 유용한 방법이라고 할 수 없다.

카드의 구현에서 이것을 해결하기 위해서 그림 9와 같은 데코레이터(Decorator) 패턴을 적용해 볼 수 있다. 위의 그림에서 보면, Component 클래스는 객체를 위한 추상 클래스로서 draw() 메소드를 가지고 있다. CardDecorator 클래스는 draw() 메소드에 대한 요청을 자신이 갖고 있는 하위 클래스에 전달하는 기능만 가지고 있다. CardDecorator 클래스의 서브클래스들은 draw() 메소드를 확장하여 경계와 이미지 그리고 애니메이션 기능을 구현한다.

데코레이터 패턴을 카드 클래스 구현에 사용함으로서, Card 객체에 새로운 서비스를 동적으로 추가하는 방법이 가능하며, 기존의 Card 객체에는 어떠한 영향도 주지 않게 된다. 또한 구현된 속성과 행동 양식이 쉽게 제거될 수도 있다. Card 클래스의 예와 같이, 실제로 상속에 의해 서브클래스를 계속 만드는 방법은 효율적인 자원의 이용이라는 면에서 실질적이지 못한 경우가 많다. 왜냐하면 다양한 종류의 카드 클래스 구현에 너무 많은 수의 독립된 상속이 필요하기 때문이다. 이것을 해결하기 위해서 데코레이터 패턴이 사용될 수 있다.

4.3 네트워크 턴 게임 설계에서 행위 패턴 적용

네트워크 게임 API 설계에 관심이 되었던 패턴들은 Chain of Responsibility, Iterator, Observer, State 패턴이다. 이러한 패턴을 적용해서 구현한 애플리케이션은 각각이 다음과 같이 적용될 수 있다.

Chain of Responsibility 패턴은 Card 객체를 효율적으로 분배하기 위해 적용 가능하고, State 패턴은 클라이언트와 서버 간에 프로토콜을 전송하기 위한 패킷 객체를 만드는데 적용 가능하다. 각각의 패턴들은 구체적으로 다음과 같이 적용되었다.

4.3.1 Chain of Responsibility 패턴 적용

메시지를 보내는 객체와 이를 받아 처리하는 객체들 간의 결합도를 없애기 위한 패턴으로 클라이언트는 처리를 요청하고 이 처리는 실제 서비스를 제공하는 객체를 만날 때까지 정의된 연결 고리를 따라서 계속 전달된다.

그림 10의 Chain of Responsibility 패턴의 개념은 메시지 송신 측과 수신 측을 분리하는 것으로, 이 요청을 처리하는 기회를 다른 객체에 분산하는 것이다. 화면 입출력에서 각 GUI 컴포넌트가 일으키는 이벤트를 처리할 때 사용할 수 있다.

또한, 이 패턴은 게임룸에서 각각의 카드 위치와 고유한 아이디가 부여된 Card 객체를 가지는 User 객체를 결정하는데 사용되었는데, 각각의 사용자에

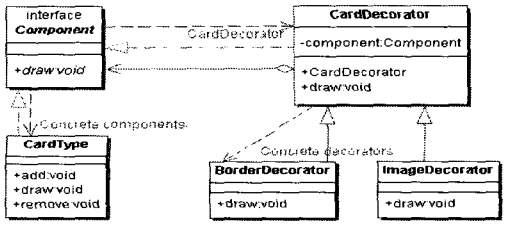


그림 9. 데코레이터(Decorator) 패턴

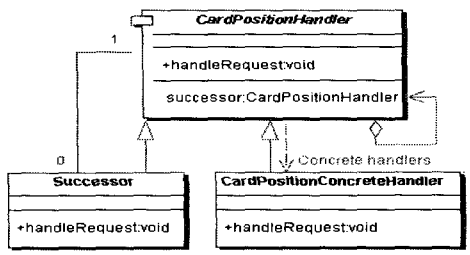


그림 10. Chain of Responsibility 패턴

게 카드를 배분하는 게임 초기 시에 사용되었다. 그리고 사용자가 새로운 카드를 가져올 때, 카드를 소유할 수 있는 user 객체는 현재 게임에 참여하고 있는 User 클래스의 인스턴스 user1, user2, user3, user4, user5 중의 하나가 된다. 카드가 어느 사용자한테 갈 것인가 하는 것은 현재 자기 차례를 나타내는 플래그(flag)가 참으로 설정된 사용자다. 이러한 설계를 채택함으로써, 사용자가 가변적으로 변할 수 있는 턴 게임에서 카드 객체와 유저 객체의 결합도를 없앨 수 있다.

4.3.2 State 패턴 적용

클라이언트 측과 서버 측에서 TCP/IP 프로토콜을 사용하여 자료를 주고받을 때, 자바에서 지원하는 다양한 스트림 모델을 사용할 수 있다. 본 시스템에서는 입출력을 다루기 위한 스트림 모델로 Object InputStream, ObjectOutputStream을 사용하였다.

클라이언트와 서버는 그림 11에서 보는 바와 같이 직렬화된 Result 클래스의 객체를 주고받는다. State 인터페이스를 수행하는 Result클래스의 개수는 클라이언트와 서버가 자료를 주고받기 위해 필요한 프로토콜 개수와 동일하다.

패킷의 송신과 수신은 Command 클래스가 가진 parseCommand() 메소드가 담당하고, MVC 패턴에 따른 클라이언트와 서버간의 프로토콜 정의는 Command 클래스가 담당하였다. 이 방법은 외형인 뷰와 프로토콜의 송수신에 따른 컨트롤러의 개발을 분산할 수 있는 장점이 있다. 또한 데이터베이스 관리 모듈에서도 State 패턴이 적용되었다.

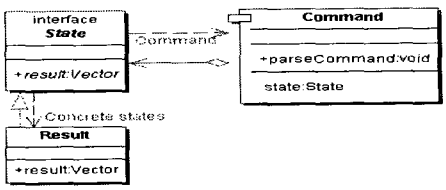


그림 11. State 패턴

5. 디자인 패턴을 이용한 게임 구현

본 절에서는 본 논문에서 제안된 설계기법을 통해서 설계된 네트워크 턴 게임을 구현 환경을 설명하고 구현된 서버와 클라이언트의 테스트 예를 보여준다.

5.1 서버 측 웹 서버와 게임 서버 실행

회원으로 등록한 사용자에 한하여 네트워크 게임을 할 수 있도록 웹 서버를 실행한다. 현재 사용되고 있는 웹 서버는 다양한 종류가 있다. 어떠한 웹 서버를 선택하느냐 하는 것은 서버의 플랫폼에 많이 좌우된다. 웹 서버는 인터넷 사용자들에게 홈페이지로 구성된 사이트를 홍보하고, 여러 명의 회원들을 모집하여 커뮤니티가 형성되도록 하는 역할을 한다.

웹 브라우저를 이용한 사용자들이 애플릿(Applet)을 통하여 서버에 소켓 연결할 수 있도록 하기 위해서 게임 서버를 실행 시켜야 한다. 각각의 게임 서버는 다른 서버에서 실행될 수도 있고, 같은 서버에서 실행될 수도 있다. 그림 12는 게임 서버에 여러 명의 사용자가 접근했을 때의 상태를 보여준다. 게임을 하기위해 커뮤니티를 형성하고 정보를 조회하기 위한 대기실은 최초로 로그인한 사용자가 있을 때, 서버에서 생성된다.

서버의 구현에 사용된 언어는 서버의 확장성을 고려하여 플랫폼에 독립적인 자바 언어를 사용하였다. 본 시스템에서 구현된 두개의 게임 서버는 클라이언트와의 통신을 위해서 정의된 프로토콜 구현에서도 많은 유사성을 보였다. 각각의 서버는 클라이언트가 요청한 자료를 효율적으로 분배해주고, 회원 정보와 같은 자료를 데이터베이스에 기록하고 읽어오는 역할을 한다. 게임 서버의 설계는 클라이언트에게 안정적이고 지속적인 서비스를 해 줄 수 있도록 충분히 고려되어야 한다. 특히 많은 클라이언트 관리 쓰레드(Thread)가 서버의 일정자원을 지속적으로 요청하게 되는데, 교착상태(Deadlock)에 의해 게임진행을 할 수 없는 상태가 발생할 수 있다. 이러한 문제는 서버 측 자원의 동기화를 통해서 해결할 수 있다.

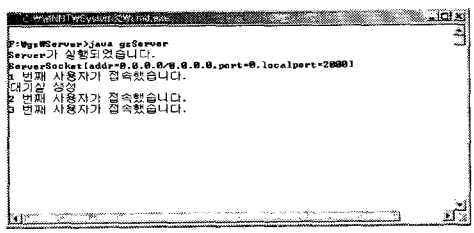


그림 12. 인터넷을 통한 사용자 로그인

5.2 클라이언트 측 처리 결과에 따른 메시지 확인

애플리케이션 개발 시, 코드의 구현에 많은 반복

작업이 필요하고, 구현한 코드가 정확하게 작동하는지 확인하는 작업이 필요하다. 게임 서버의 개발은 도스창과 같은 곳에서 구현된 코드의 정확성을 검증한다.

클라이언트 테스트에 웹 브라우저가 가지고 있는 자바 콘솔을 사용할 수도 있지만, 브라우저를 통한 실행결과 확인은 많은 시간이 소요되기 때문에, 그림 13과 같이 JDK가 제공하는 애플릿뷰어(Appletviewer)를 사용해서 확인하였다.

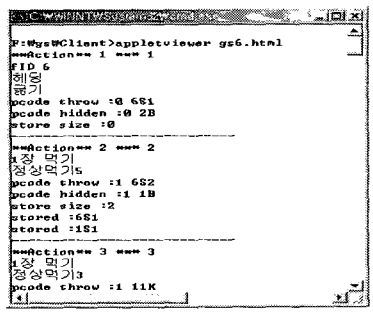


그림 13. 클라이언트의 액션에 따른 처리 결과 확인

6. 결 론

본 논문에서 카드를 사용하는 두개의 게임을 대상으로 GoF의 디자인 패턴을 적용하여 구현된 두개의 네트워크 게임을 통해서, 새로운 게임을 개발하거나 기존에 개발된 게임 소프트웨어를 추가적으로 개발하기 쉬운 최적의 설계 기법을 제안하였다. 또한 게임 API 설계의 여러 부분에서 이미 검증된 다양한 디자인 패턴이 적용될 수 있다는 것을 알 수 있었다. GoF의 생성패턴, 구조패턴, 행위패턴들은 네트워크 게임 API 설계에 있어서 유용하게 적용되었다.

먼저 객체의 생성에 관련하는 생성 패턴은 여러 종류의 네트워크 게임 제작에서 특성에 맞는 다양한 GUI의 구현이나 게임 룸 설계와 종류가 많고 가변적일 수 있는 카드 객체를 구성하고, 실행 시에 새로운 카드의 삽입이나 삭제가 필요한 부분과 게임의 진행을 총괄하는 서버의 전체 자원을 효율적으로 관리하기 위해 적용할 수 있었다.

그리고 전체 설계의 구조를 형성하기 위해 사용된 구조 패턴은 카드 객체가 일으키는 이벤트의 구현, 게임의 각 컴포넌트들이 일으키는 이벤트의 구현, 서버의 플랫폼이 다를 경우와 원시 API를 참조해야하는 클래스 설계 그리고 메모리 소모가 많은 여러 가지 그림을 처리해야 할 경우 적용할 수 있었다.

마지막으로, 객체간의 기능 배분과 구현에 사용된 행위 패턴은 화면 입출력에서 각 GUI 컴포넌트가 일으키는 이벤트 처리나 클라이언트와 서버 프로그램에서 실시간으로 생성되는 Rooms 객체와 Users 객체를 생성하고 필요한 정보를 삽입, 수정, 삭제하는 곳에서 이용될 수 있었고, 또 다양한 객체가 발생하는 이벤트를 처리나 네트워크 연결을 통한 패킷의 송수신에 적용할 수 있었다.

게임 소프트웨어 설계를 프로그램 구현과 같이 진행할 수 있지만, 이러한 방법은 결과적으로 시간과 비용이란 문제를 해결하지 못하고 문제점만 가중시키는 경우가 많다. 소프트웨어의 구현에 따르는 문제를 해결하기 위해서 디자인 패턴을 적용함으로써, 시스템의 구현에서도 분산작업과 재사용에 효과적이라는 것을 알 수 있었다.

참 고 문 헌

[1] 김종수(2003) "웹을 기반으로 한 JAVA 네트워크 게임 시스템의 설계와 구현", 부산외국어대학교 대학원, 석사학위논문.

[2] 김종수, 권오준, 김태석 "네트워크 기반 다자간 아바타 채팅 시스템의 구현", 한국멀티미디어학회 춘계학술발표논문집, pp.171-174, 2003.

[3] Sun microsystems, *sun educational services Advanced Java Programming SL-300, SunSoft Press, 2000*(400 pages)

[4] Sun microsystems, *sun educational services Java Programming SL-275, SunSoft Press, 2000*(720 pages)

[5] Erich Gamma, Richard Helm Ralph Johnson, Hohm Vissides 공저 "GoF의 디자인 패턴", Pearson education Korea(440 pages)

[6] Sun microsystems, *sun educational services Java Programming SL-110, SunSoft Press, 2000*(615 pages)

[7] 김종수, 이종민, 김태석 "생성 패턴을 사용한 네트워크 기반 게임 API 설계", 한국멀티미디어학회 추계학술발표대회논문집, pp.669-674, 2003.

[8] Soon-Kak Kwon, Jong-Soo Kim, Tai-Suk Kim, "An Implementation Avatar Chatting

System for Network-Based Multi-User Environment" EALPIIT2003 National University of Mongolia, Ulaanbaatar, Mongolia July 6-9, 2003 pp.119-123, 2003.

[9] GRAIG LARMAN 원저 "UML과 패턴의 적용" 홍릉과학출판사(705 pages)

[10] 최성 "게임 산업과 기술 전망", 정보처리학회지 특집 게임 기술, pp.11-23, 2002.

[11] Flower, M., and Scott, K. 2000. *UML Distilled.* Reading, MA.: Addison-Wesley.

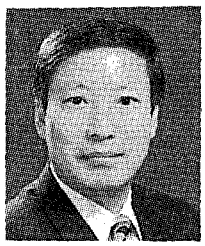
[12] Gemstone Corp., 2000. A set of architectural patterns at http://www.javasuccess.com.

[13] Harrison, N., 1998. Patterns for Logging Diagnostic Messages. Pattern Languages of Program Design vol. 3. Reading, MA.: Addison-Wesley.

[14] Kruchten, P. 2000. *The Rational Unified Process-An Introduction.* 2nd edition. Reading, MA.: Addison-Wesley

김 종 수

1992년 2월 부경대학교 냉동공학과(학사)
1998년 12월 On Line Technology 대표
2003년 2월 부산외국어대학교 컴퓨터공학과 석사
2003년 3월~현재 동의대학교 소프트웨어공학과 박사과정
2003년 4월~현재 동의대학교 영상 미디어센터 PM연구원
관심분야 : 네트워크 게임 제작과 설계, Web 프로그래밍



김 태 석

1981년 경북대학교 전자공학과 졸업(공학사)
1989년 일본 KEIO대학 이공학부 계산기과학전공(공학 석사)
1993년 일본 KEIO대학 이공학부 계산기과학전공(공학 박사)
1993년 일본 국제전신전화연구소(KDD) 기술고문
1993년 일본 KEIO대학 이공학부 객원연구원
1994년~현재 동의대학교 소프트웨어공학과 교수
관심분야 : 정보시스템, 기계번역, 인터넷비즈니스