Journal of Korea Multimedia Society Vol. 10, No. 8, August 2007(pp. 1060-1067)

Observer 패턴을 적용한 MMORPG의 파티 시스템 아이템 배분 방법

김태석[†], 김신환[††], 김종수[†††]

요 약

인터넷을 이용하는 다양한 게임 장르 중에서 대규모의 게임 유저들이 이용하는 게임 장르인 MMORPG (Massively Multi-player Online Role-Playing Game)를 개발하기위해서는 많은 기술들이 필요하다. 특히 분산 작업의 효율을 높이기 위해서 C++와 같은 객체지향언어가 사용되는데, 대규모의 게임을 만들 때 객체지향개념을 충분히 활용할 수 있는 설계기법이 유용하다. GoF(Gang of Four)의 디자인 패턴에는 소프트웨어 분산 설계에 응용할 수 있는 다양한 패턴이 있는데, 게임 유저들 사이에 커뮤니티를 형성하기 위한 파티 시스템 설계에 Observer 패턴을 이용하면, 필요한 새로운 클래스의 추가나 유지보수를 쉽게 할 수 있다. MMORPG 게임 내에서 파티 사냥 시스템은 게임 이용자들의 커뮤니티를 형성하기 위해 자주 이용되는 중요한 시스템이다. 파티 사냥 시스템에서 중요하게 고려해야 할 사항은 파티 사냥 결과로 얻어지는 획득물과 경험치를 다양한 레벨의 이용자들에게 공평하게 나누어 주는 것이다. 시스템의 유지보수적인 측면을 고려한 파티 사냥 시스템을 구현하기 위하여, 본 논문에서는 GoF의 디자인 패턴 중 Observer Pattern을 이용한 기법을 제안하고, 제안된 기법이 C++언어가 가지는 장점인 동적메모리 할당과 가상 메소드 호출을 이용하여 프로그램 실행 시에 실시간으로 객체를 변경하고 새로운 클래스를 추가하는데 효율적이며, 시스템을 유지보수하는데 장점이 있음을 보인다.

The Item Distribution Method for the Party System in the MMORPG Using the Observer Pattern

Tai-Suk Kim[†], Shin-Hwan Kim[††], Jong-Soo Kim[†††]

ABSTRACT

We need various methods to develop MMORPG that is game genre which many users use among various game genre using Internet. Specially, to heighten efficiency of distributing work, Object-oriented language such as C++ is used and we need design techniques that can take advantage of enough object-oriented concept when making large-scale game. There is various pattern that can apply in software breakup design in GoF's design pattern for these design techniques. If you apply Observer pattern to Party System Design for forming community between game users, you can easily add new class and maintain system later. Party Play is one of the important system that is used to form game users' community in MMORPG games. The main point that must be considered in Party-Play-System is to divide evenly experience value and acquisition that is got by Party-Play among users according to each user's level. To implement Party Play System that consider maintenance of system, in this paper, we propose a method using GoF's Observer-Pattern, showing you that proposed method which has advantage to dynamic memory allocation and to virtual method call can be used usefully to change object to real time at program run and to add new class and to maintain system new.

Key words: Design Patterns(디자인 패턴), Observer Pattern(관찰자 패턴), Network Game(네트워크 게임), Party Play System(파티 사냥 시스템)

1. 서 론

검색엔진이 제공하는 2006년 국내 온라인 게임의 인기 순위를 보면 10권 안에 들어 있는 게임 중 과반수가 MMORPG 게임이다. 게임의 인기를 계속적으로 유지하기 위해서는 게임 내에서 유저들 간의 커뮤니티를 형성해주는 것이 필요한데, 이를 위한 방법의 하나로 대부분의 MMORPG는 파티 사냥 시스템을 지원한다[1].

파티 시스템은 2명 이상의 플레이어들이 파티라고 불리어지는 그룹단위로 협력해서 몬스터를 사냥하거나 퀘스트(Quest)를 수행하는 것을 말한다. 게임 유저들은 특정한 퀘스트를 수행하거나 몬스터를 사냥하기 위해서 파티 시스템을 사용할 수 있고, 그 결과에 따른 경험치와 전리품을 나누어 가진다. 본 논문에서는 파티 사냥 시스템의 효율적인 설계를 위해, GoF의 디자인 패턴 카탈로그 중에서 Observer 패턴의 적용에 대해서 연구하였고, 패턴을 사용한 설계와 구현이 유효한 것을 확인한다[2-4].

2. 관련 연구

본 절에서는 GoF의 디자인 패턴을 연구한 다수의 국내외 문헌을 조사하여 분석하고, 파티 시스템에 경험치 배분에 사용되는 공식의 예를 바탕으로 경험치가 배분되는 방법과 게임에서 사용되고 있는 다양한 파티시스템에 대해서 고찰한다.

2.1 국내외 동향

본 연구와 관련된 국내외 동향을 파악하기 위하여 IEEE의 저널과 논문을 탐색할 수 있는 IEEEXplore라는 검색 사이트를 이용하였다. 관련 문헌 조사를 위하여 검색어로 "design pattern", "network game", "RPG"라는 단어를 사용하였고, 검색결과는 표 1과 같이 나왔다.

표 1. IEEEXplore 검색 사이트를 이용한 관련 자료 조사

검색어	검색 결과	관련 문헌
design pattern	100건	11건
network game	39건	3건
RPG	21건	0건

MMORPG 게임은 국내의 인터넷 인프라가 발전함으로써 새롭게 생긴 게임 분야로서 관련된 연구가 거의 없었고, 조사한 문헌 중 게임과 다른 애플리케이션 설계에 디자인 패턴을 적용한 논문들이 있는 것으로 조사되었다.

코스웨어 시스템을 제작하기 위해 디자인 패턴을 적용한 "Using design patterns in the development of a planner-based courseware system."[5] 에서는 시스템을 구현하기 위한 몇 가지 디자인 패턴을 제안하였다. 첫 번째로 제안된 패턴은 Learning Session을 구현하기 위한 Composite 패턴, 코스웨어를 사용하기 위한 학생과 선생을 구현하기 위한 State 패턴, 학생들의 이력과 시험 결과에 따라 계획과 관련된 Learning Session 설계에 Strategy 패턴, 그리고 학생들이 수업을 받는 과정에 상황에 자유롭게 바꾸기 위한 Iterator 패턴을 제안하였다.

그러나 제안한 설계의 구체적인 구현과 관련되어 어떠한 언어로 구현이 가능한지 제시되지 않았다. 일반적으로 회사에서 사용하는 업무용 애플리케이션은 데이터관리에 많은 비중을 두는데 코스웨어 시스템도 마찬가지라 볼 수 있다. 업무용 애플리케이션을 개발하기위한 언어로는 비주얼 베이직, 파워빌더, 델파이가 있는데 4GL로 분류되지만 객체를 지향하는 언어가 아니다. C++를 사용하여 코스웨어 시스템을 개발할 수도 있지만, 생산성이 떨어지므로 비합리적이라 할 수 있다.

수학의 대수학과 관련하여 디자인 패턴을 연구한 "Specification of design patterns using real-time process algebra (RTPA)."[6] 에서는 대수학과 관련된 분야에서 사용할 수 있는 Abstract Factory와 Observer 패턴을 제안하였다. 그러나 대수학은 수학의 한 분야이고, 근본적으로 디자인 패턴에서 말하는

※ 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 : 2007년 3월 15일, 완료일 : 2007년 6월 13일
† 종신회원, 동의대학교 공과대학 컴퓨터·소프트웨어공학과
†† 대구대학교 정보통신대학 정보통신공학부
(E-mail : namuri@daegu.ac.kr)
††† 정회원, 동의대학교 게임연구소
(E-mail : seatree@deu.ac.kr)
※ 본 논문은 2007학년도 정부(교육인적자원부)의 재원으로 한국대학교육협의회 대학교수 국내교류 연구비 지원에 의한 것임.

객체 지향적으로 설계된 코드의 효율적인 재사용기법과는 크게 관련이 없다고 할 수 있다.

2.2 파티 시스템에 대한 고찰

파티 시스템을 구현함으로서 게임 운영자 측에서 얻을 수 있는 가장 큰 효과는 게임을 통한 커뮤니티 형성이다. 게임 내에 커뮤니티가 형성되면 게임 사용자가 장기간 게임을 하도록 하여, 사용자가 다른 온라인 게임으로 빠져나가는 것을 막을 수 있다[7,8].

이러한 이유에서 게임 개발사측은 파티 시스템을 사용하는 유저들에게 아이템이나 경험치를 더 많이 주는 것이 일반적이고, 혼자서 게임을 할 때는 사용할 수 없는 특별한 기술들을 제공하는 경우도 있다. PC방에서 온라인 게임을 즐기는 유저들을 대상으로 파티플레이에 대해 조사한 내용에 따르면, 온라인 게임을 즐기는 유저들 중에서 23%는 파티 플레이를 자주하는 것으로 조사되었고 32%는 거의 못한다고 조사되었다[9].

파티로 퀘스트를 수행하는 중에 파티원들이 획득하는 경험치의 배분 방법은 게임마다 다르다. 파티 시스템의 경험치 배분 공식의 한 예는 다음과 같다.

파티원과의 경험치 계산법 = X+(n-1)*((2/3)*X) X=원래 경험치 n=게임 안의 플레이어 수

어떤 몬스터를 혼자서 사냥해서 얻을 수 있는 경험치가 3382라고 했을 때, 파티 시스템을 사용하여 사냥할 때 획득할 수 있는 경험치를 계산해 보면, n=2일 때 경험치는 3,382+((2/3)*3,382)=5,638이고, n=3일 때 3,382 + (2*(2/3) *3,382) = 7,891로 점점 증가하고 10명이 동시에 잡았을 경우를 생각해 보면, 23680의 경험치가 각각 분배되어, 같은 시간에 혼자서 사냥하는 것보다 경험치를 많이 얻는다는 것을 알 수 있다. 게임 내에서 파티 플레이가 활성화되면 게임을 시작한지 얼마 되지 않는 초보자들이 다른 유저들의 도움을 받아서 게임을 쉽게 익힐 수 있다.

게임마다 사용자들의 흥미를 유발시키기 위해 다양한 파티 시스템이 사용된다. 악튜러스라는 PC용 게임에서는 특이한 파티 시스템이 있는데, 일반적인 파티 시스템과 MVP(most valuable player), 몰아주기라는 파티 시스템이 있다. MVP 시스템은 마지막으로 몬스터를 잡은 파티원에게 더 많은 경험치를 배분해 주는 것이고, 몰아주기는 파티원 중 특정한 사람에게 경험치를 몰아주는 방법이다.

3. APIs 구현을 위한 GoF의 디자인 패턴 적용

게임을 구성하는 다양한 요소의 설계에 GoF 디자인 패턴을 이용할 수 있다[10-12]. 본 절에서는 파티 플레이 시스템을 구현하기위한 문제 영역을 정의하여, 파티 시스템 설계와 구현에 있어서, 기존의 설계 방법과 이를 개선하기 위해 사용할 수 있는 Observer Pattern에 대해서 살펴본다.

3.1 문제 영역

파티를 처음 만드는 사람을 파티장이라고 부르는데, 파티장은 파티를 만든 후 파티에 참여하고 싶어하는 유저들을 모으게 된다. 파티를 만든 목적에 따라서 전사, 마법사, 궁수, 도적, 광대와 같은 직업을 가진 캐릭터가 필요하고, 퀘스트를 수행하는 목적으로 구성되는 파티는 캐릭터 중에 특정한 캐릭터가 필요한 경우가 있다. 파티에 참여하고자 하는 유저들은 각각 다른 레벨과 전사, 마법사와 같은 다른 직업을 가지고 있다. 게임 서버는 파티에 참여한 여러 캐릭터의 레벨과 사냥의 참여 정도를 판단하여 파티원들에게 경험치를 배분한다. 고려해야 할 사항은 사용자들이 파티 시스템을 사용하여 몬스터를 사냥하고 있는 중에도 파티원이 추가되거나 감소될 수 있다는 것과 파티를 구성하여 몬스터를 사냥하는 중에도 경험치를 획득한 파티원의 상태가 변화될 수 있다는 것이다. 파티 시스템은 이러한 변화를 쉽게 수용할 수 있도록 설계 되어야 한다.

3.2 기존의 연구 방법

파티를 상징하는 Party 클래스와 파티에 참여하고 싶어 하는 전사, 마법사와 같은 클래스를 정의하여 일반적인 다이어그램을 그리면 그림 1과 같은 설계 방법이 있을 수 있다.

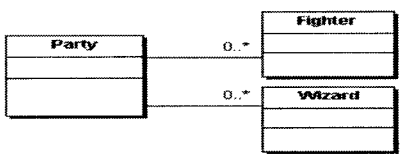


그림 1. 파티 시스템의 일반적인 해법

다이어그램에서, Party 클래스는 파티에 참여하는 Fighter클래스와 Wizard 클래스를 관리한다. 파티원의 수가 일정하고 각자의 상태가 변하지 않는다고 가정하면, Observer 패턴을 사용하지 않는 일반적인 방법으로 설계된 클래스들은 다음과 같은 코드로 구현할 수 있다.

Party.h

```
class Party{
private:
    Fighter fighter;
    Wizard wizard;
public:
    void addPartyMembers(Character& c);
    void getPartyMembers();
    int void calcPoint(int cLevelPoint, int cATime)
    void dividePartyVaule();
};
```

Party.cpp에서 경험치 배분 공식

```
Party::Party(){  //파티원을 얻어오기 위한 생성자
   getPartyMembers();
}
void Party::addPartyMembers(Character& c){//파티 참여자
   List<characters> *chars;
   party.addMember(chars);
}
int void Party::calcPoint(int cLevelPoint, int cATime){
   return (getMosterValue()+(getTotalMembers()-1)*
           (2.0/3)*getMonsterValue() *
           cLevelPoint/getAllLevelPoints()*
           cATime/getTotalAttackTimes();
}
```

Fighter.h

```
#include "Party.h"
class Fighter{
public:
    Fighter();
    void addValue(int value);
    bool checkLevelUp(Character& c);
    void partyBroadcast(char * str);
    void makeParty();  // 파티객체 생성
    Party getParty();  // 생성된 파티 얻기
    void joinParty();  // 파티에 참가
private:
    Party * party;     // 파티 객체
    int level, point;  // 전사 캐릭터들의 레벨과 경험치
};
```

PartyTest.cpp

```
#include "Party"
#include "Character.h"
#include "Fighter.h"
... 생략
void dividePoints(){   // 각각의 파티에서 경험치를 나누기
   List<Party> *partys=getPartyList();//등록된 파티 리스트
   for(partys.First(); !partys.IsDone(); partys.Next()){
      List<Chracter> * chars;
      chars = partys.getPartyUsers();
      for(chars.First(); !chars.IsDone(); chars.Next()){
         Party party = chars.getParty();
         party.dividePartyValue();
      }
   }
}
```

```
/*** PartyTest.cpp 계속 ***/
void main(){  // 프로그램 시작
   User * user = getUser();  //사용자 정보 얻기
   if  loginOk() {              //로긴 확인 후 캐릭터 생성
      createCharacter(user);
   }
   for(;;){
      ... 생략
      dividePoints();
   }
}
```

그러나 파티 사냥 도중에 새로운 유저가 비번하게 가입하고, 탈퇴하는 경우를 고려했을 때, Party 클래스는 어떤 파티원이 나가고 새로 들어왔는지, 들어온 파티원의 경험치 배분 기준이 어떻게 되는지 쉽게 알 수 있어야하고, 추가적으로 얻은 경험치로 인해 파티에 참여한 객체들 중 레벨이 높아진 캐릭터가 있는지 등에 관한 정보를 지속적으로 관찰할 수 있으면 효율적이다. 또한 파티에 참가하는 객체들의 변화를 알고 싶어 하는 객체들에게 변화에 따른 정보를 쉽게 통보할 수 있고, 게임 애플리케이션 내에서 새로운 캐릭터가 추가적으로 생겼을 경우 애플리케이션의 유지보수를 쉽게 할 수 있는 설계 구조면 더욱 효율적이다.

4. Observer 패턴을 적용한 파티 사냥 시스템의 설계방법

본 절에서는 Observer Pattern을 이용하여 파티 플레이 시스템을 효율적으로 설계하기위한 방법과 제안된 방법이 새로운 API의 추가와 유지 보수가 있을 때, 효율적이라는 것을 보인다.

4.1 Observer 패턴을 적용한 설계 및 구현

일반적인 파티 시스템에서 파티에 참여하는 캐릭터는 2명 이상인데, 몬스터를 사냥해서 얻은 경험치의 배분은 각각의 캐릭터들의 레벨의 수치에 따라 전체 경험치에서 적절한 비율로 경험치가 나누어지도록 해야 한다.

그림 2는 파티시스템으로 여러 명의 캐릭터가 파티 시스템을 사용하여 동시에 몬스터를 사냥했을 때, 사냥의 결과로 획득하게 되는 경험치를 캐릭터의 레벨에 따라서 배분할 수 있도록 Observer 패턴이 적용된 설계를 보여준다.

관찰자를 관찰하는 Subject로 Party, Observer로

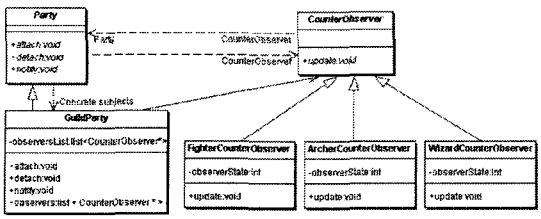


그림 2. 파티의 경험치를 나누기 위한 Observer 패턴

CounterObserver를 두었고, 각각의 파티원들이 관심을 가지는 경험치와 속성은 GuilParty에서 구현하였다.

그리고 사냥이나 퀘스트 수행 결과로 얻어지는 경험치 배분을 위한 클래스로 각각의 캐릭터들은 FighterCounterObserver, ArcherCounterObserver, WizardCounterObserver가 추가되었다. 각각의 클래스들은 다음과 같이 구현될 수 있다.

Party.h

```
class Party{
public:
    virtual void attach(CounterObserver*);
    virtual void detach(CounterObserver*);
    virtual void notify();
};
```

GuildParty.h

```
class GuildParty : public Party{
public:
    virtual int getDividedValue();
    virtual void notify();
private :
    List<CounterObserver*> *observers;
};
```

FighterCounterObserver.h

```
class FighterCounterObserver{
public:
    FighterCounterObserver(GuildParty*);
    virtual void update(GuildParty*);
private:
    GuildParty+ _aParty;  // 등록한 통보자 관리
};
```

다이어그램에서 CounterObserver를 상속받은 FighterCounterObserver, ArcherCounterObserver, WizardCounterObserver가 Party 객체에서 알려주는 정보를 바탕으로 update() 메소드를 이용하여 각각의 경험치를 갱신한다. Observer 패턴을 이용함으로써, 새로운 observer 객체가 GuildParty의 통보를 받기위해서 쉽게 추가될 수 있다. 그리고 객체가 바뀔 때마다 관련된 Observer들이 새로 갱신된다.

4.2 Observer Pattern을 적용한 시스템 평가

MMORPG 게임은 다양한 유저들에게 흥미를 유발하기 위해 여러 가지 직업 시스템을 가진 새로운 캐릭터가 필요한 경우가 많다. 인기 있는 MMORPG의 하나인 메이플 스토리[13]가 가진 직업 시스템은 표 2와 같다.

표 2. 메이플스토리 전직 시스템

직업 / 차수	전사	궁수	마법사	의적
2	파이터, 페이지, 스피어맨	헌터, 사수	위자드(불, 독, 번개, 냉기), 클레릭	어쌔신, 시프
3	크루세이더,나이트,드래곤나이트	레인저, 저격수	메이지(불독, 번개 냉기), 프리스트	허밋, 시프마스터
4	히어로, 팔라딘, 다크나이트	보우마스터, 신궁	아크메이지, 비숍	나이트로드, 섀도어

메이플 스토리는 1, 2, 3, 4차 전직 시스템이 있는데, 이중 4차 전직은 캐릭터의 레벨이 120이상일 때 선택가능한데, 2006년 말에 생긴 새로운 전직시스템이다. 기존의 게임에서 의적이라는 캐릭터가 추가 되었다고 가정하면, 기존의 파티 시스템의 유지보수에 옵저버 패턴을 적용하지 않는 설계는 그림 1의 다이어그램에서 그림 3과 같이 변경될 수 있다.

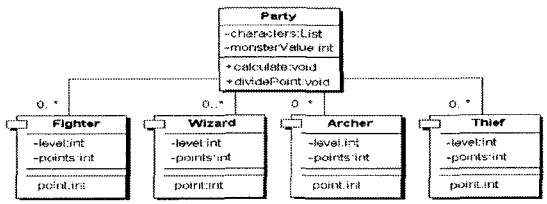


그림 3. 기존 방법에서 도적이 추가된 다이어그램

다이어그램에서 Thief 클래스가 추가되었을 때, 새로운 클래스의 헤더 파일은 다음과 같은 코드를 가질 수 있다.

Thief.h

```
#include "Party.h"
#include "GuildParty.h"
class Thief{
public:
    Thief();
    void addValue(int value);
    void partyBroadcast(char * str);
    void checkLevelUp(int quiredValue);
    ... 생략
};
```

만약 어떤 파티 시스템에서 새롭게 추가된 Thief가 파티에 참가한다고 가정했을 경우, 기존의 파티 시스템의 코드 수정이 필요하다. 파티원을 추가하는 addPartyMember() 메소드에 Thief클래스의 객체를 추가해 주어야하고, 경험치를 배분하는 dividePartyValue() 메소드에서 Thief 객체의 특성에 따른 경험치 배분기준을 가져와서, 파티에 참가한 Thief 클래스 객체의 경험치 계산식에 의해 배분된 경험치를 추가하기 위한 코드가 추가되어야 한다.

Party.cpp

```
void Party::addPartyMembers(Character& c){//파티 참여자
    List<characters> *chars;
    //Thief클래스 객체 추가 위한 코드 추가
    party.addMember(chars);
}
... 생략
void Party::dividePartyVaule(){//경험치 배분 메소드
    Party party = getParty();   // 파티 얻기
    ...생략
    Iterator<Character*>* i = createIterator();
    for(i->First(); !i->IsDone(); I->Next()){
        Character chrc = (Character)(i->CurrentItem());
        if (sizeof(Fighter)=sizeof(chrc)){          //전사
            //각 캐릭터에 대한 처리
        }//추가되는 코드
        else if (sizeof(Thief)=sizeof(chrc)){
            Thief thief = (Thief)chrc;      //의적
            int cLevelPoint  = thief.getLevel();
            int cATime = thief.getMonsterAttackTime();
            thief.setPoint(caclPoint(cLeverPoint,cATime));
        }
    }
}
```

위와 같은 설계와 구현은 게임 내에서 주요 역할을 담당하는 캐릭터 클래스가 또 다른 파티와 관련된 메소드를 구현하게 함으로써, Character 클래스에게 너무 많은 부담을 준다는 단점이 있다. 이러한 방법은 유지보수의 측면을 고려하지 않은 것으로 게임이 업그레이드 될 때, 새로운 종족이나 파티 시스템이 계속 생겨날 수 있다는 가정을 하면, 결과적으로 컴퓨터 메모리를 비효율적으로 사용하게 된다. 그리고, 네트워크의 지연이라든지, 파티를 수행하고 있는 클라이언트와 클라이언트간의 메시지 전달에서 교착상태와 같은 문제를 일으킬 수 있다.

이와 같은 문제는 파티와 관련된 처리를 전문가 클래스를 만드는 것이다. 이 클래스는 파티에 참여하는 캐릭터들을 실시간으로 체크하여, 각 캐릭터의 상태변화가 있으면 실시간으로 통보해주는 역할을 할 수 있다. 이러한 처리를 위해 기존에 사용된 다양한 설계 패턴들이 사용될 수 있지만, Observer 패턴은 실시간으로 객체 변화를 관찰하기 위해 사용될 수 있는 좋은 설계의 예를 보여준다.

본 논문에서도 Observer 패턴의 장점을 활용하기 위해 Character클래스가 구현해야 하는 파티와 관련된 일을 Party 클래스와 Observer 클래스에 위임하는 방식을 검토하여 적용하기로 하였다. 논문에서 제안하는 방식을 적용하면, 클래스 설계가 그림 4와 같이 된다. 다이어그램에서 Fighter, Wizard, Archer, Thief 클래스의 파티 처리를 Party로부터 유도된 GuilParty 클래스와 CounterObserver 클래스로부터 유도된 Observer 클래스들에게 위임하였다.

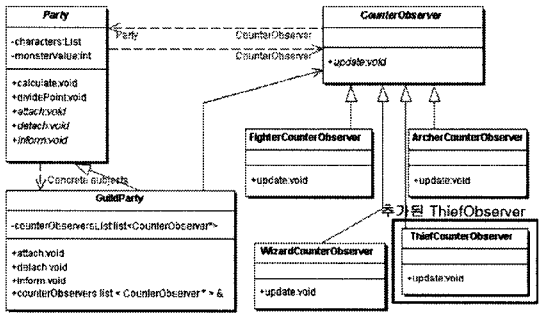


그림 4. 제안된 방법에 의한 Thief Observer 추가

파티에 대한 처리를 다른 클래스에 위임하는 것도 새로운 클래스를 추가해야 하고, 관련된 코드를 수정해야하는 추가적인 비용이 든다. 그러나 추후 발생할 수 있는 게임 애플리케이션이나 파티와 관련된 유지보수에서 기존의 코드에 영향을 덜 미치고 코드를 추가할 수 있다는 장점이 있다. 파티 처리를 위한 Observer를 사용하기 위해 추가되는 비용을 아래 코드에 나타내었다.

Thief.cpp

```
Thief::Thief(){
}
void Thief::partyBroadcast(char * str){
}
...생략
void Thief::joinParyt(Character& c){ //파티에 참가
    List<GuildParty> *guildPartys;
    //파티 처리를 위해 추가되는 코드
    GuildParty gParty = gulidPartys.getSelectedParty();
    observer = new ThiefCouterObserver(gparty, this);
    g.attach(&observer);
}
```

이러한 방식은 기존에 만들어진 Character 클래스에 영향을 미치지 않고, 변화된 파티 시스템을 효율적으로 수용할 수 있으므로, 유지보수에 편리하다는 장점이 있다. 또 레벨업과 같은 변화가 있을 경우에 캐릭

터의 속성 값을 변경하기 위해 Observer클래스의 update 메소드를 수정할 수 있다. ThiefCounterObserver와 관련된 구현은 아래 코드와 같이 될 수 있다.

ThiefCounterObserver.h

```
class ThiefCounterObserver{
public:
    ThiefCounterObserver(GuildParty*, Thief* thief);
    virtual void update(GuildParty*);
private:
    GuildParty+ _aParty; // 등록한 통보자 관리
};
```

ThiefCounterObserver.cpp

```
bool ThiefCounterObserver::isLevelUp(int point){
   ...의적 클래스 레벨업에 대한 처리
   return false;
}
void ThiefCounterObserver::update(GuildParty* gP){
    if (isLevelUp(point)){          //레벨업 체크
        thief->levelUpCeremony();
        thief->checkNextJob();  //다음 전직 체크
        gP.notify();                    // 길드들에게 레벨업 통지
    }
    if (gP == aParty){
        gP.getMVP()
        if (isMVP()) {
            getDroppedItem(droppedItem);
        }
        gP.notify();                    //길드들에게 통지
    }
}
void ThiefCounterObserver::setDivideValue(){
    int value = aParty->getDividedValue();
}
```

GuildParty.cpp

```
int GuildParty::calcPoint(int levelPoint, int aTime){
   ...생략
   return value;
}
void GuildParty::dividePartyVaule(){//경험치 배분 메소드
   getPartyMembers();
   Iterator<Observer*>* i = createIterator();
   for(i->First(); !i->IsDone(); I->Next()){
      Observer observer =
                  (observer)(i->CurrentObserver());
      int levelPoint  = observer.getLevel();
      int aTime = observer.getMonsterAttackTime();
      observer.setPoint(caclPoint(leverPoint, aTime));
   }
}
```

```
/*** GuildParty.cpp 계속 ***/
void GuildParty::notify(){
    divideValue();     //경험치 배분
    ListIterator<CounterObserver> coi(observers);
    for(coi.First(); !coi.IsDone(); coi.Next(){
        coi.CurrentItem()->update(this);
    }
}
```

Thief 클래스의 추가에서 보듯이 게임 시스템 전체에서 기존의 캐릭터와 다른 속성을 가진 새로운 캐릭터가 추가되거나 새로운 파티시스템이 추가될 경우에 Observer Pattern을 이용한 파티 사냥시스템 설계는 새로운 캐릭터가 사냥의 결과로 얻어지는 경험치의 배분 조건을 Observer클래스로부터 유도된 새로운 클래스에 구현하면 되고, GuildParty 클래스에서 이것을 관찰할 수 있도록 attach() 메소드를 사용하여 Observer인 Party를 추가하였다. 이러한 설계 방법은 Character 클래스가 파티 시스템 처리와 같은 기능을 구현하기 위해 비대해지는 것을 방지할 수 있고, 파티 처리를 담당하는 전문가 클래스에게 처리를 위임하게 하여, 유지보수를 쉽게 할 수 있다는 장점이 있다. 또한 객체 지향언어의 큰 장점인 동적 바인딩 기법을 적용할 수 있으므로, 프로그램 실행 시에 객체를 바꾸는 것과 같은 효과를 나타낼 수 있다.

5. 결 론

본 논문에서 파티 시스템의 효율적인 설계를 위하여 GoF의 디자인 패턴 카탈로그에 있는 Observer 패턴을 이용하는 설계방법을 보였다. 일반적인 설계로 파티 시스템을 구현하면, 관련 클래스들의 수정이 자주 일어날 수 있는데, 특히 Character 클래스에 계속되는 코드의 추가로 인해 시스템 내에서 부담하는 역할이 커져서, 결과적으로는 컴퓨터 자원을 비효율적으로 사용하는 결과를 초래하고, 교착상태 발생과 같은 문제점을 일으킬 수 있다는 단점이 있다. 이러한 설계는 추후에 발생할 수 있는 유지보수에도 비효율적인 경우가 많다.

논문에서 보여준 Observer 패턴을 이용한 파티 시스템 설계에서 Party 클래스로부터 유도된 GuildParty 클래스는 파티에 참여하는 Character 클래스들로부터 파티 처리를 위임받은 Observer 객체들의 추가, 삭제가 쉽게 이루어질 수 있도록 해주었고, 몬스터 사냥에 따라 획득한 경험치를 각각의 캐릭터들에게 효율적으로 배분한 후, 경험치 변경에 따른 변화를 관찰할 수 있도록 해주었다. 또한 CounterObserver 클래스로부터 유도된 FighterCounterObserver, ThiefCounteObserver와 같은 클래스들은 파티에 참가하여 얻어지는 경험치를 GuildParty 클래스로부터 전달받아 자신의 정보를 갱신하고, 그 결과로 발생되는 변화를 GuildParty 클래스의 공지를 통해서 각각의 Character 객체에 전달할 수 있었다.

본 논문에서 제안하는 설계방법을 게임 제작자들

이 이용하면 게임 설계를 위한 기존의 설계 방법보다 더 좋은 설계나 아키텍처를 재사용하기 쉬워진다는 장점이 있고, 논문에서 보여준 설계와 코드를 참고로 새로운 게임 개발자들이 Observer 패턴을 더 유용하게 사용할 수 있고, 재사용이 가능한 설계를 선택하도록 도와줄 수 있다.

참 고 문 헌

[1] Soon-Kak Kwon, Jong-Soo Kim, Tai-Suk Kim, "An Implementation Avatar Chatting System for Network-Based Multi-User Environment," *EALPIIT2003*, Proceedings of the 3rd, pp. 119-123, 2003.

[2] 남재욱, 온라인 게임 서버 프로그래밍, 한빛미디어, 서울, 2004.

[3] 김종수, "웹을 기반으로 한 JAVA 네트워크 게임 시스템의 설계와 구현," 부산외국어대학교 석사학위논문, pp. 1-11, 2003.

[4] 김종수, 이종민, 김태석, "생성 패턴을 사용한 네트워크 기반 게임 API 설계," 한국멀티미디어학회 추계학술발표대회논문집, 제6권, 2호(하), pp. 669-674, 2003.

[5] Khalid S. AI-Tahat, Prof. Dr. Tengku M. T. Sembok, and Dr. Sufian Bin Idris. "Using Design Patterns in the Development of a Planner-Based Courseware System" *IEEE Catalogue No. 01CH37239,* 2001.

[6] Vu Nguyen-Cong and Yingxu Wang. "Specification of Design Patterns using Real-Time Process Algebra" *CCECE 2004-CCGEI 2004,* Niagara Falls, May/mai 2004.

[7] Craig Larman, UML과 패턴의 적용, 홍릉과학출판사, 서울, 2003.

[8] Erich Gamma, Richard Helm, Ralph Johnson, Hohm Vissides, GoF의 디자인 패턴, (주) 피어슨 에듀케이션 코리아. 서울, 2003.

[9] http://ragnagate.gamehankook.com/zboard/zboard.php?id=press_new&page=1&divpage=1&ss=on&keyword=파티&no=634.

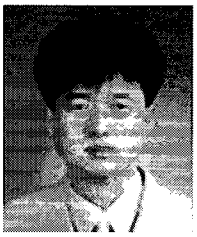
[10] 김종수, 김태석 "Design of Network-based Game using the GoF Design Patterns," 한국멀티미디어학회 논문집, 제9호, pp. 742-749, Jun. 2006.

김 태 석

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

김 신 환

1975년 2월 한국항공대학교 통신공학과(공학사)
1983년 2월 전북대학교 전기공학과(공학석사)
1988년 3월 University of Tokyo 전자공학전공(공학박사)
1983년 3월~현재 대구대학교 정보통신공학부 교수
관심분야 : 영상신신호처리, 영상데이터압축



김 종 수

1992년 2월 부경대학교 학사
2003년 2월 부산외국어대학교 컴퓨터공학과 석사
2006년 2월 동의대학교 소프트웨어 설계 박사
2003년 4월~현재 동의대학교 시간 강사
1998년 12월~현재 On Line Technology 대표
관심분야 : 네트워크 게임 제작과 설계, Web 프로그래밍