프로세스 정의 도구의 객체지향적 설계 및 구현

황미진[O] 이민규 한동수
한국정보통신대학원대학교
{mjlois, niklaus, dshan}@icu.ac.kr

Object-Oriented Design and Implementation of Business Process Definition Tool

Mijin Hwang [O] Minkyu Lee Dongsoo Han
Information and Communications University

요 약

BPMT는 워크플로우 관리 시스템에서 프로세스의 정의 도구이다. BPMT는 워크플로우 시스템의 종속된 도구이기 때문에 워크플로우 시스템에서 요구하는 사항에 대처하기 위한 방안으로 확장성과 재사용성이 필수적이다. 이러한 확장성과 재사용성을 달성하기 위해 본 논문에서는 객체지향적인 개발 방법을 선택하고 BPMT 에 적합한 디자인 패턴인 Visitor 패턴과 Serializer패턴을 사용하였다. Visitor 패턴을 이용하면 새로운 표현 방법을 추가하고자 할 때 새로운 모듈만 추가한다는 점에서 표현의 확장성을 달성할 수 있다. 또한 Serializer를 이용함으로써 저장매체에 독립적인 입출력을 가능케 하고 하나의 인터페이스를 제공함으로써 객체의 확장성을 달성하였다.

1. 서론

소프트웨어 시스템을 개발하는 데 있어 재사용성, 유지보수와 확장성은 중요한 의미를 갖는다. 객체지향 방식을 활용하여 개발된 소프트웨어의 경우 그 구조가 상속 되기 때문에 유지보수가 용이하다. 따라서 어떤 변화가 가해질 때 발생하는 외부효과(side effect)를 줄일 수 있으며 확장 또한 수월하다[6].

체계적으로 구조화된 객체지향 시스템은 패턴의 반복으로 나타나는 것으로 알려져 있다[1]. 한편 디자인 패턴은 소프트웨어를 개발할 때 발생하는 문제의 해결책을 제시해준다. 따라서 어떤 시스템을 객체지향적으로 설계하고자 할 때 그 시스템에 적합한 디자인 패턴을 이용한다면 좀 더 효율적인 개발이 가능하며 시스템의 변동 사항을 유연하게 처리할 수 있고 근본적으로 재사용성을 높일 수 있다.

워크플로우 관리 시스템(Workflow Management System)도 기존의 다른 시스템과 마찬가지로 개발 과정에서 요구사항의 변경이 빈번하게 발생한다. 특히 워크플로우 시스템의 구성 요소인 BPMT(Business Process Modeling Tool)는 시스템 내부에서 전반적으로 사용되는 프로세스를 모델링 하는 역할을 하기 때문에 이러한 요구에 유연하게 적응할 수 있도록 설계, 구현되어야 한다. 또한 개발 후 발생할 수 있는 여러 문제에 적절히 대처할 수 있는 기반을 갖추는 것이 바람직하다.

본 논문에서는 이러한 객체지향적 방안인 디자인 패턴을 이용하여 현재 개발 과정 중인 워크플로우 시스템 ICU/COWS [3]의 프로세스 정의 도구인 pBuilder (Process Builder)가 새로운 요구에 쉽게 대처할 수 있도록 하는 설계 및 구현에 관하여 논의한다. 본 논문의 구성은 2 장에서는 워크플로우 관리 시스템의 build-time 도구인 프로세스 정의 도구와 디자인 패턴에 대해 간략히 소개하고, 3장에서는 실제로 pBuilder를 설계하는데 디자인 패턴을 어떻게 적용시켰는지 구체적으로 알아본다. 마지막으로 4장에서는 결론 및 향후 연구 과제를 제시한다.

2. 배경

2.1 프로세스 정의 도구(Process Definition Tool)

프로세스 정의 도구는 워크플로우 관리 시스템을 구성하는 요소 중 하나로서 컴퓨터로 처리 가능한 형태로 프로세스의 명세를 작성하는데 사용되는 도구이다. 정의 도구에서 모델링 한 프로세스 정의는 워크플로우 시스템의 여러 제품에 의해 이용되기 때문에 정의 도구는 서로 다른 제품간의 교환이 가능한 형태로 프로세스를 표현할 수 있어야 한다.

2.2 디자인 패턴

재사용이 가능한 객체지향 소프트웨어를 만드는 것은 매우 어려운 작업이다. 또한 비슷한 문제를 해결하기 위해 매번 비슷한 노력을 들이는 것은 매우 비효율적이다. 객체지향을 연구하는 여러 그룹에서는 이러한 문제를 해결하기 위한 방법으로 디자인 패턴에 대한 연구를 진행해왔다. 디자인 패턴은 자주 발생하는 문제에 대한 객체지향적인 해결을 구조적으로 추상화 해 놓은 것이다. 따라서 디자인 패턴을 이용하면 복잡한 시스템도 단순한 몇 개의 핵심이 되는 디자인 패턴으로 설계할 수 있다[1,4].

3. pBuilder의 설계 및 구현

본 장에서는 프로세스를 구성하는 객체 설계 부분과 pBuilder를 설계, 구현하는데 있어 디자인 패턴을 적용시킨 방법을 살펴보고자 한다.

3.1 pBuilder에서의 프로세스 정의 구성

pBuilder의 기능을 크게 살펴보면 새로운 프로세스의 정의와 정의된 프로세스의 저장 및 수정을 들 수 있다. 새로운 프로세스를 정의하는 작업은 여러 개의 단위업무(activity)와 그것들 간의 트랜지션(transition), 관련 데이터(relevant data), 어플리케이션 선언과 할당, 그리고 참여자(participant) 할당으로 이루어진다. 그림 1은 하나의 프로세스를 구성하는 객체들의 클래스 다이어그램이다.

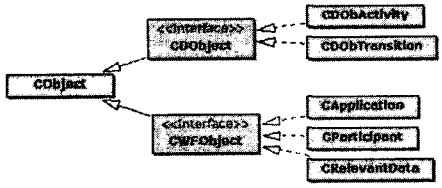

그림 1. pBuilder에서의 프로세스 구성 요소

3.2 프로세스 정의의 표현

여러 객체들로 구성된 프로세스 정의는 워크플로우 시스템의 요구에 따라 XML, WPDL과 같은 여러 가지 형태로 표현될 수 있어야 한다.

동일한 객체를 바탕으로 각기 다른 작업을 해야 할 때 그 객체의 클래스마다 관련 오퍼레이션을 전부 포함시킨다면 각 작업의 기능이 흩어지게 된다. 결국 또 다른 형태로 표현하려면 각 객체의 클래스를 하나하나 수정해주어야 하는 번거로움이 생기게 된다.

이런 경우 Visitor 패턴[1]을 이용하면 수정과 확장이 용이하고 readability 가 향상된다. 많은 디자인 패턴이 존재하지만 Visitor 패턴의 목적 자체가 이미 존재하는 객체에 대해 오퍼레이션을 표현하는데 있기 때문에 해당 객체의 클래스를 변경할 필요 없이 Visitor 만 가지고 새로운 기능을 정의할 수 있다.

그림 2 는 프로세스 정의를 표현하는 부분에 Visitor 패턴을 적용한 모습이다. 그림에서 보듯이 Object - Visitor는 모든 객체 타입을 visit 하는 오퍼레이션들을 선언하고 XMLGeneratingVisitor와 WPDLGenerating -

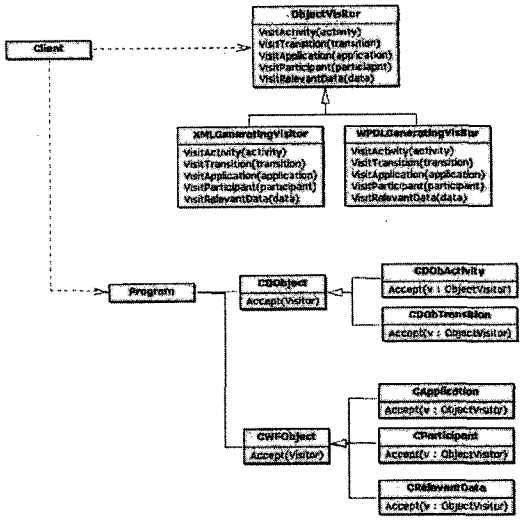

그림 2. Visitor 패턴을 적용한 프로세스 표현

Visitor는 ObjectVisitor 에서 선언된 오퍼레이션을 Visitor의 역할에 맞게 구현한다. 이런 방식으로 이미 구성되어 있는 프로세스 정의 관련 클래스의 내부는 수정하지 않고 Visitor 만 추가함으로써 새로운 기능을 구현할 수 있다. 프로세스 정의는 어떤 한 가지 특정한 형태를 요구하지 않기 때문에 이러한 표현의 확장성이 요구된다.

3.3 프로세스 정의 입출력

BPMT에서 모델링된 프로세스 정의는 워크플로우 시스템의 구동을 위해 일단 어떤 형태로든 저장되어질 필요가 있다. WfMC 표준에서 프로세스 정의의 입출력 형식을 특별히 제한하지는 않지만 현재는 파일이나 관계형 데이터베이스에 저장하는 방식이 일반적이다[5]. pBuilder의 경우 워크플로우 시스템 간의 상호 운용을 위해 파일 입출력과 데이터베이스 입출력 두 가지 모두 지원한다.

그림 3은 Serializer 패턴을 이용하여 입출력 클래스와 프로세스 정의 관련 클래스를 분리한 모습이다.

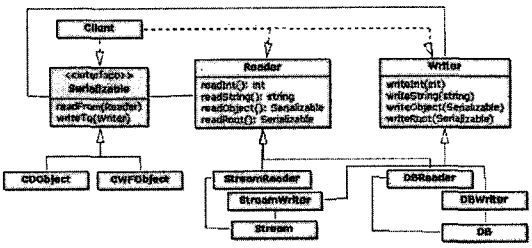

그림 3. Serializer 패턴을 적용한 프로세스 정의 입출력

pBuilder에서는 Serializer 패턴을 적용하여 입출력을 담당하도록 했다. Serializer 패턴의 목적은 어떠한 객체든지 간에 저장 매체에 종속되지 않고 데이터 구조 기반으로 입출력 작업을 가능하게 하기 위함이다.

Serializer패턴은 저장 매체에서 데이터를 읽어들여 객체를 구성하는 부분과 실재하는 객체를 데이터 구조로 저장 매체에 쓰는 두 부분으로 구성된다. Reader/Writer 클래스를 상속받아 새로운 표현 형식에 적합한 newReader/newWriter를 더하는 형태로 확장이 가능하다. 또한 입출력 부분이 Serializable 이라는 인터페이스로 규정되어 있기 때문에 새로운 타입의 객체를 추가하는 작업이 수월하다. 왜냐하면 다른 타입의 객체를 더하고자 할 때 입출력과 관련해서는 그 클래스 내부에 readFrom(Reader), writeTo(Writer)라는 두 개의 함수만 구현하면 되기 때문이다.

Serializer패턴을 사용하지 않고 구현한다면 새로운 객체를 추가하기 위해서는 그 객체가 매체에 저장되는 방법을 스스로 알아야 한다. 이것은 곧 객체 내의 속성마다 일일이 입출력 하는 방법을 기술해줘야 하는 것을 의미한다.

pBuilder를 개발하는데 사용된 프로그래밍 언어인 Visual C++는 기본적으로 특정한 한 객체 CArchive에 대해 바이너리 파일에 한해서 Serializable한 오퍼레이션을 제공한다. 이와는 별도로 주어진 특정 객체가 아니라 프로세스 정의 구성 객체를 ASCII파일과 데이터베이스에 입출력을 가능하게 하기 위해 Serializer 패턴을 사용하였다.

BPMT에서는 CDObject의 객체를 CDObArray로 유지한다. 이 때 CDObArray의 각 객체를 저장하는 코드를 살펴보면 다음과 같다.

```
void CDObArray::WriteTo(Writer *w)
{
    for(int i=0; i < GetSize(); i++)
        GetAt[i]→writeTo(w);
}
```

Serializer패턴을 이용하여 위와 같은 방식으로 구현했을 때 의도했던 객체의 확장성을 제공할 수 있다. CDObject 클래스를 상속 받는 새로운 객체 타입을 추가하려고 할 때 그 객체는 기존의 Serializable 인터페이스를 구현하면 된다. 모든 책임은 추가되는 객체 자체에 국한되기 때문에 새롭게 추가된 객체를 위해 위의 코드를 변경할 필요가 없다. 이러한 특징은 프로그램의 유지보수 측면에서 유용하다고 할 수 있다.

또한 프로세스 정의를 파일이나 데이터베이스가 아닌 네트워크 상에서 전송하고자 한다면 매체에 한정된 코드 부분을 Reader/Writer 형식으로 모듈화 함으로써 간단히 덧붙일 수 있다.

4. 결론 및 향후 연구과제

본 논문에서는 디자인 패턴을 사용하여 객체지향의 장점인 재사용성, 확장성과 명료성을 갖도록 ICU/COWS 의 pBuilder를 설계, 구현하였다.

크게 Visitor 패턴과 Serializer 패턴을 적용하였으며 이 두 패턴을 사용함으로써 객체의 확장성, 표현의 확장성을 보장할 수 있었을 뿐만 아니라 유지보수가 용이함을 알 수 있었다.

향후에는 정의 도구에서 갖춰야 할 기능인 정의된 프로세스의 분석, 검증을 위한 패턴 적용 방안을 연구할 예정이다.

5. 참고 문헌

[1] Gamma et al, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[2] Workflow Management Coalition, *Workflow Management Coalition The Workflow Reference Model*, Document number TC00-1003, January 19, 1995.

[3] Dongsoo Han, Jaeyong Shim, Chansu Yu, *ICU/COWS: A Distributed Transactional Workflow System Supporting Multiple Workflow Types*, IEICE Transactions on Information and Systems Vol. E83-D, No. 7, July 2000.

[4] 이승일, 심재용, 한동수, " 디자인 패턴을 사용한 객체지향 워크플로우 관리 시스템 엔진 개발", 한국정보과학회 ' 99 가을 학술발표논문집(I), 제26권 1호,pp537-539, 1999.

[5] Workflow Management Coalition, *Workflow Management Coalition Interface 1: Process Definition Interchange Process Model*, Document number WfMC TC-1016-P, October 29, 1999.

[6] Roger S. Pressman, *Software Engineering: A practitioner's approach 4 th ed.*, McGRAW -HILL, 1997.