디자인 패턴을 사용한 객체지향 워크플로우 관리 시스템 엔진 개발

이승일, 심재용, 한동수

한국정보통신대학원대학교
{gaialee, jaeyong7, dshan}@icu.ac.kr

Using Design Patterns in the Development of Object-Oriented Workflow Management System Engine

SeungIl Lee, JaeYong Shim, DongSoo Han
Information and Communications University

요 약

jFlow는 OMG에 의해서 표준으로 제정된 워크플로우 관리 시스템이다. 이 표준안은 객체지향 패러다임으로 설계되어져 있으나 효과적인 표현 방법을 사용하지 않아서 제정된 시스템에 대한 이해가 용이하지 않다 본 논문에서는 표준안인 jFlow의 내용을 수용하는 객체지향 워크플로우 관리 시스템 설계와 구현에 있어서 디자인 패턴이 효율적인 접근 방법임을 보인다. 이를 위해서 본 논문에서는 디자인 패턴을 이용하여 설계되고 구현하고 있는 한우리/TFlow 워크플로우 관리 시스템을 간단히 소개하고 프로세스 생성과 변경등에서의 jFlow와 한우리/TFlow 시스템의 클래스 구성방식을 비교 분석한다.

1. 서 론

객체 단위의 작업과 객체 사이의 메시지 전달의 특성을 가지는 객체지향 패러다임은 단일 시스템뿐만 아니라 분산환경의 시스템 구축에 커다란 유연성을 제공한다. 비즈니스 업무의 자동화를 목적으로 하는 워크플로우 관리 시스템(Workflow Management System)도 객체지향 패러다임으로 그 표준안이 OMG(Object Management Group)에 의해 정해 졌다. 그러나 OMG의 표준으로 제정된 jFlow의 표준안[3]의 내용은 효과적인 표현 방법을 택하고 있지 않아서 객체지향적인 내용이 잘 드러나지 않는다. 따라서, 그 표준을 기반으로 객체지향적인 워크플로우 관리 시스템을 구현하기가 쉽지않다

재사용 가능한 객체지향 소프트웨어를 만드는 일은 매우 힘든 작업이다. 또한 비슷한 문제 해결을 위하여 매번 비슷한 노력을 들이는 것은 소프트웨어 공학적으로 볼 때 매우 비효율적이다. 객체지향을 연구하는 단체에서는 이러한 문제를 효율적으로 해결하기 위한 방법으로 디자인 패턴(Design Patterns)에 대한 연구를 진행해 왔다. 디자인 패턴은 자주 발생하는 문제에 대한 객체지향적인 해결을 추상화 시켜 놓은 것이다 따라서, 디자인 패턴을 이용하면 복잡한 시스템도 단순한 몇 개의 핵심이 되는 디자인 패턴으로 설계할 수 있다[4,5,7,8,9].

본 논문에서는 워크플로우 관리 시스템의 표준안인 jFlow를 기반으로 하고 디자인 패턴을 사용하여 연구 개발되고 있는 한우리/TFlow 워크플로우 관리 시스템[1] 소개를 통해서 객체지향 워크플로우 관리 시스템의 개발 방법에 디자인 패턴이 효과적인 접근 방법임을 보이려고 한다.

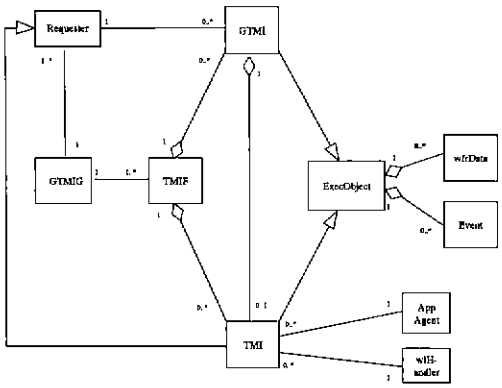
본 논문의 2장에서는 CORBA를 기반으로 설계되어진 개방형 워크플로우 관리 시스템인 한우리/TFlow 소개하고, 3장에서는 jFlow가 제안하고 있는 방안을 분석하고 이를 기반으로 하는 객체지향적인 시스템을 구축하기위해서 한우리/TFlow 시스템이 사용하고 있는 디자인 패턴을 알아본다. 마지막으로 4장에서는 본 논문의 결론과 앞으로의 연구 과제에 대하여 논하고 글을 맺는다.

2. 한우리/TFlow 시스템

한우리/TFlow는 OMG의 BODTF(Business Object Domain Task Force)에서 Workflow Management Facility Specification의 표준으로 채택된 Joint Workflow의 명세서(specification)를 모두 수용하는 CORBA 기반의 개방형 워크플로우 관리 시스템이다[1].

그림 2.1는 한우리/TFlow 시스템의 주요 CORBA IDL 인터페이스(interface)를 보여 준다 그림 2.1에 보이는 여러 IDL 인터페이스 중에 한우리/TFlow 시스템의 핵심이 되는 것은 GTMIG(Global Task Manager Instance Generator), TMIF(Task Manager Instance Factory), GTMI(Global Task Manager Instance), TMI(Task Manager Instance) 이다 이 네 개의 인터페이스를 이용하여 구현된 클래스들이 프로세스와 해당 프로세스를 구성하는

단위업무를 생성하고 관리하는 워크플로우 관리 시스템의 엔진을 구성하게 되는 것이다. GTMIG는 GTMI와 TMI의 생성을 제어하고 워크플로우 엔진 관리자 역할을 하는 인터페이스다. GTMIG가 GTMI와 TMI 생성을 제어하지만 직접적으로 생성 하지는 않는다. GTMI와 TMI의 실질적인 생성은 Object Factory 역할을 하는 TMIF가 담당하게 된다. GTMI는 한 프로세스의 템플릿(template) 정보를 가지며 이를 바탕으로 필요한 TMI의 생성을 TMIF에게 요청하고 생성된 TMI를 관리하는 역할을 하는 인터페이스다 마지막으로 TMI는 GTMI의 한 단위(step)로 단위 업무를 제어 하기 위한 인터페이스다.



[그림 2.1] 한우리/TFlow 시스템의 주요 CORBA IDL 인터페이스

3. 워크플로우 디자인 패턴

비즈니스 업무의 주요 특성은 매우 동적이며 한 프로세스의 수행 시간이 길다는 것이다. 비즈니스 업무가 동적이라는 의미는 업무의 수행 환경에 따라서 업무의 흐름이 자주 바뀔 수 있다는 것이다. 또한 비즈니스의 업무의 수행 시간은 일반적으로 수 일에서 수 주, 많게는 년 단위의 수행 시간이 필요한 특성을 가지고 있다. 따라서, 업무의 자동화를 위한 시스템인 워크플로우 시스템은 동적이고 장기간 지속되는 업무의 특성을 만족시킬 필요가 있다.

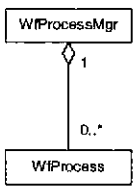
본 장에서는 이러한 업무의 두 가지의 주요 특성을 만족시키기 위해서 jFlow가 제안하고 있는 방안을 알아보고 jFlow의 표준을 수용하여 개발되고 있는 한우리/TFlow 시스템의 설계에 디자인 패턴이 어떤 식으로 사용되었는가에 대하여 알아본다

3.1 프로세스의 생성과 변경

비즈니스 업무는 지속적으로 변한다 지속적으로 변화는 업무를 지원하기 위해서 업무의 변화가 생길 때마다 그에 해당하는 새로운 프로세스 템플릿을 수정하고 수정된 템플릿의 인스턴스(instance)를 생성하는 것은 상당히 비효율적일 것이다 따라서, 워크플로우 시스템에서는 동적으로 프로세스에 변화를 주는 메커니즘이 마련되는 것이 좋다 객체지향의 관점에서는 이미 잘 알려진 디자인 패턴을 사용함으로써 이러한 문제를 쉽게 해결할 수 있다.

프로세스 정의(Process definition) 과정[2]을 통해서 정해진 프로세스의 인스턴스가 동적으로 변하는 경우는 크게 두 가지로 구분할 수 있다 그것은 프로세스 수준(process level)의 변화가 필요한 경우와 프로세스를 구성하는 단위 업무 수준(task level)의 변화가 필요한 경우이다

jFlow는 그림 3.1과 같은 클래스 구조를 갖추고서 동적으로 프로세스가 변해야 하는 요구사항에 대한 해결책[3]을 제시하고 있다 프로세스 정의는 관리자 역할을 하는 WfProcessMgr 객체에 encapsulate 되어 있다. 이 것은 특정 워크플로우 프로세스의 템플릿을 나타낸다. 이 관리자가 워크플로우 프로세스의 인스턴스를 생성하고 초기화한다. 이 모델은 프로세스의 타입(type) 클래스와 인스턴스 클래스를 분리하여 하나의 타입으로 여러 가지의 customize 된 인스턴스를 생성할 수 있도록 하고 있다 즉, 프로세스의 타입 클래스와 인스턴스 클래스를 분리하여 프로세스 타입의 변경을 하지 않는 상태에서 프로세스의 수정을 할 수 있도록 하였다

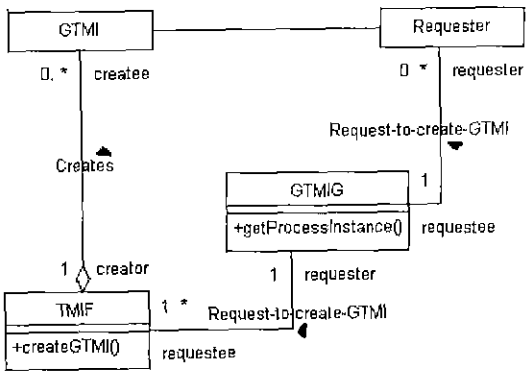


[그림 3.1] jFlow 프로세스 모델

한우리/TFlow는 CORBA를 기반으로 하는 분산 시스템이다 한우리/TFlow 시스템에서는 분산 시스템에서의 프로세스를 담당하는 객체와 해당 프로세스를 구성하는 단위 업무를 담당하는 객체를 생성하는 인터페이스를 하나로 하여 분산 객체 생성을 용이하게 하였다. 이런 구조 때문에 jFlow와는 달리 프로세스 생성과 프로세스에 대한 템플릿 정보를 분리하여 각기 다른 클래스에 분산 시켰다 프로세스 생성에 대한 책임은 GTMIG 클래스와 TMIF 클래스가 담당하고 프로세스에 대한 템플릿 정보는 GTMI 클래스가 담당한다.

현재 GTMI가 가지고 있는 템플릿에 대한 인스턴스는 Factory Method[5]를 이용하여 생성하도록 하였다. 생성된 프로세스를 구성하는 단위 업무도 같은 메커니즘을 이용하여 생성된다 GTMI와 TMI의 인스턴스 생성에 Factory Method를 사용한 이유는 생성을 요구하는 객체가 구체적으로 생성되는 객체의 정보를 가지고 있지 않아도 되도록 하기 때문이다. 그림 3.2는 분산시스템을 위해 Factory Method를 확장하고 이렇게 확장된 디자인 패턴으로 GTMI를 생성하는 방법을 보여 주고 있다 확장된 Factory Method에서는 여러 개의 시스템에 분산되어 있는 Factory 관리를 위한 Factory Manager 클래스를 두고 있다. GTMIG가 Factory Manager 클래스이고 TMIF가 Factory 클래스다 이런 방법으로 Factory Method를 확장한 이유는 Factory 클래스를 분산된 여러 시스템에 두고 각 시스템의 부하를 고려하여 새로운 객체를 생성하기 위해서다.

분산 환경의 구조적 특성을 이용하기 위해서 프로세스 생성과 프로세스 템플릿을 각각 다른 클래스에 나누어 설계된 한우리/TFlow 시스템은 생성되어 진행중인 프로세스를 동적으로 변화 시키기 위해서 몇 가지의 유용한 디자인 패턴을 사용하고 있다



[그림 3.2] 한우리/TFlow의 프로세스 생성 IDL 다이어그램

프로세스 수준의 변화는 GTMI의 프로세스 관련 정보를 가지는 필드(field)를 변경해야 하고 이 변경에 대한 행동을 취해야 한다. Type object 디자인 패턴[7]은 템플릿 클래스와 인스턴스 클래스를 분리하여 동적으로 전체 인스턴스에 대한 변경을 줄 수 있기 때문에 프로세스 수준의 변화에 대한 요구 조건 해결에는 Type object 디자인 패턴을 사용하고 있다 단위 업무 수준의 변경은 단위 업무의 attribute와 behavior에 대한 변화가 있는 경우다. variable state 패턴[4]을 이용하면 클래스의 attribute에 대해서 동적인 변화를 줄 수 있고 Decorator 패턴[5]을 이용하면 상속(inheritance)을 이용하지 않고 클래스의 operation에 대한 변화를 동적으로 줄 수 있기 때문에 단위 업무 수준의 변화는 variable state 패턴과 Decorator 패턴을 사용하여 설계 및 구현하고 있다.

3.2 Persistence

워크플로우 관리 시스템이 관리하는 프로세스는 일반적으로 긴 수행 시간 동안 지속된다 따라서, 실행 시간이 긴 프로세스는 하드웨어나 소프트웨어의 failure에 대해서 많은 영향을 받게 된다. 따라서, 프로세스의 상태 유지에 대한 persistence 문제가 발생하게 된다.

프로세스의 persistence에 대한 문제에 대하여 jFlow는 CORBA의 persistence service인 POS(Persistent Object Service)[6]의 사용에 대하여 초점을 맞추어 간단히 설명되어 있다. 즉, persistence를 해결하기 위한 객체 지향적인 설계 내용이 워크플로우 관리 시스템 표준안에는 들어 있지 않다.

Memento 디자인 패턴[5]을 사용하면 생성되어 수행중인 객체의 상태를 저장하고 그 객체에 문제가 발생했을 경우에 시스템의 오버헤드 없이 이전 상태로 되돌릴 수 있기 때문에 한우리/TFlow 시스템에서는 프로세스의 persistence 문제 해결에 Memento 디자인 패턴을 사용하고 있다

4. 결론 및 향후 연구 과제

jFlow는 객체지향 패러다임으로 설계되어 있지만 디자인 패턴 같은 효과적인 표현 방법을 사용하지 않아서 그 내용 파악이 어렵고 제안 내용 자체의 부족함을 찾아 볼 수 있다 그러나 한우리/TFlow 시스템은 다섯 가지의 디자인 패턴을 사용하여 OMG 표준안인 jFlow의 내용을 수용하고 비즈니스 업무의 두 가지 중요한 특징을 포용하도록 시스템을 설계하고 구현함으로써 jFlow가 가지고 있는 전술한 문제점을 해결할 수 있었다

한우리/TFlow 시스템을 디자인 패턴을 사용하여 설계하고 구현하는데 얻은 이점은 크게 두 가지로 볼 수 있다. 먼저, 시스템 설계가 요구 사항을 만족시키는 디자인 패턴을 선택하는 것으로 끝남으로써 시스템 구축이 단순화되고 용이해졌다 두 번째로는 이렇게 설계되고 구현되는 시스템은 기능적으로 모듈화 됨으로써 확장성(extendibility)과 이해도(readability)가 높아져 시스템 관리와 변경이 용이해 졌다는 것이다 결과적으로 복잡하고 대규모의 객체지향 시스템을 이해하고 설계하며 구현하는데 있어서 디자인 패턴의 이용은 아주 효과적인 접근 방법임을 확인할 수 있었다

한우리/TFlow 시스템의 설계와 구현에 있어서 비즈니스 업무의 두 가지 주요 요구 사항에 대한 해결에만 디자인 패턴이 사용되었다 한우리/TFlow 시스템은 트랜잭션(transaction) 특징을 가지는 워크플로우 시스템이며 오류 처리는 워크플로우 관리 시스템의 안정성(reliability)을 높이는데 반드시 구현되어야 할 부분이기 때문에 트랜잭션 처리와 오류 처리를 위한 워크플로우 디자인 패턴에 대한 연구를 앞으로 진행할 예정이다.

5. 참고 문헌

[1] Dongsoo Han, Jaeyong Shim, *Hanuri/TFlow : A Distributed Transactional Workflow System Supporting Multi-Workflow Types*, Submitted to ACM Group 99

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

[3] OMG BODTF RFP #2 Submission, *Workflow Management Facility*, Revised Submission, OMG Document Number. bom/98-06/07, July 4, 1998

[4] Kent Beck, *Smalltalk Best Practice Patterns*, Prentice Hall, 1996

[5] Erick Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns. Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995

[6] Robert Orfali, Dan Harkey, Jeri Edwards, *Instant CORBA*, John Wiley & Sons, 1997

[7] Robert C Martin, Dirk Riehle, Frank Buschmann, editors, *The Type Object Pattern*, chapter 4 of Pattern Languages of Program Design 3, Software Patterns Series, Addison-Wesley, 1997

[8] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, *A System of Patterns*, John Wiley & Sons, 1998

[9] Dragos-Anton Manoescu, Ralph E Johnson, *A Proposal for a Common Infrastructure for Process and Product Models*, OOPSLA'98 Mid-year Workshop on Applied Object Technology for Implementing Lifecycle Process and Product Models, July 1998, Denver, Colorado