객체지향적 워크플로우 모델링

강 석 호*·정 재 윤*·김 영 호*

《目 次》

Ⅰ. 서 론
Ⅱ. 비즈니스 프로세스 모델링
Ⅲ. 객체지향 개념의 필요성
Ⅳ. 객체지향 워크플로우
Ⅴ. 프로세스 계층 모델과 접근 권한
Ⅵ. 결 론

워크플로우관리시스템은 업무 프로세스의 자동화를 통해 업무 효율을 제고하고자 하는 시스템이다. 이 시스템은 기업 활동을 지원하는 여러 정보시스템에서 정형화된 업무를 수행하고 관리하는 중추적 역할을 수행한다. 본 논문에서는 효과적인 워크플로우 관리를 위하여 객체지향 프로세스 모델링 방안을 제안한다. 이 방법론은 분산 환경에서 여러 조직이 참여하고, 업무 환경의 변화에 능동적으로 대응할 수 있는 동적 프로세스를 지원하기 위한 것이다. 그리고, 프로세스의 중첩 관계를 이용한 계층 모델을 통해 프로세스 접근 권한을 효과적으로 관리할 수 있도록 한다. 객체지향적 워크플로우 모델링은 프로세스의 유연성과 재사용성을 높일 수 있다. 또, 분산 환경에서 여러 부서나 기업이 참여하는 협력 프로세스 모델링에 효과적이다.

주요어 : 워크플로우관리시스템, 객체지향, 프로세스 계층 모델

Ⅰ. 서 론

워크플로우(workflow)는 컴퓨터가 자동으로 실행하고 관리하는 업무 프로세스(process)를 말하고, 워크플로우관리시스템(WFMS: Workflow Management System)은 워크플로우를 정의하여 실행시키며, 이를 체계적으로 관리할 수 있는 소프트웨어를 의미한다[HOLL95]. WFMS는 업무 프로세스를 효율적으로 관리하기 위하여 사용되는데, 최근에 기업 활동을 지

* 서울대학교 산업공학과

원하는 여러 가지 정보시스템 - 예를 들면, 전사적 자원관리시스템(Enterprise Resource Planning), 제품정보관리시스템(Product Data Management), 공급망관리시스템(Supply Chain Management), 고객관계관리시스템(Customer Relationship Management), 전자상거래(Electronic Commerce) 등 - 의 기반으로서 정형화된 업무를 처리하고 관리하는 중추적 역할을 수행하고 있다.

WFMS를 적용하기 위해서는 비즈니스 프로세스를 모델링 하는 것이 필요하다. 이 모델링의 목적은 실제 프로세스를 시스템이 이해할 수 있도록 워크플로우 명세(specification)를 준비하는 것이다. 이 명세 즉, 프로세스 모델을 통하여, 어떤 활동들이 필요한지, 활동 간의 의존 관계가 어떻게 되는지, 그리고 어떤 역할들이 요구되는지를 알 수 있다. 데이터에 대한 사용자의 관점을 표현하는 데이터 모델링이 효율적인 데이터베이스 응용에 있어 중요한 것과 마찬가지로, 프로세스 모델링은 사용자 관점에서의 업무 처리와 산출물 생성을 컴퓨터가 처리할 수 있는 형태로 표현하여 이를 자동화하는 워크플로우 응용에 있어 매우 중요한 일이다.

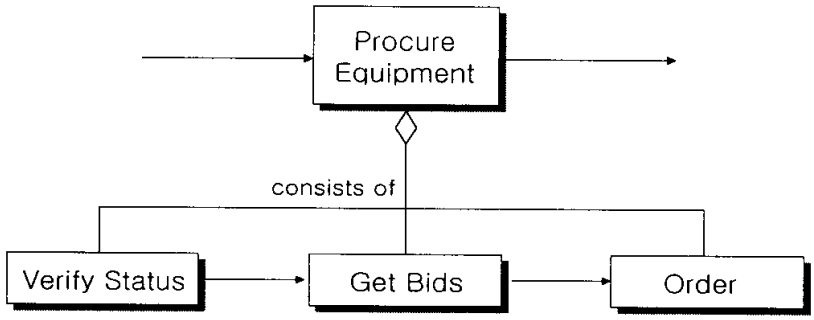
본 논문의 목적은 복잡하고 가변적인 업무 프로세스를 효율적으로 관리하기 위해 객체지향 기법을 이용하는 워크플로우 모델링 방법을 개발하는 것이다. 이 모델에서는 공통된 사항을 상위 수준에 정의하고 하위 수준에서는 이를 상속해서 이용하며, 세부적인 사항 또는 내용의 변경은 하위 수준에서 별도로 정의한다. 또, 중첩의 개념[KIM2000]을 이용한 프로세스 계층모델로 프로세스 모델링을 컴포넌트(component)화 하며, 이를 바탕으로 프로세스 접근 권한을 효과적으로 통제할 수 있는 메커니즘을 제안한다. 이 같은 객체지향 워크플로우 모델은 워크플로우의 다단계 설계, 하위 프로세스의 독립적 관리, 진행 중 프로세스의 동적 변경, 중첩 프로세스의 접근 권한 관리, 분산 서버로 확장 등 프로세스의 재사용성과 유연성을 높일 수 있는 이점이 있다.

Ⅱ. 비즈니스 프로세스 모델링

비즈니스 프로세스는 제품이나 서비스를 산출하기 위해 설계된 업무 활동(work activities)에 대한 설명과 순서를 정의한다. 이 업무 활동은 시간과 공간에 걸쳐 조직의 단계별 목적에 부합하도록 고안되어야 한다. 이를 효과적으로 표현하기 위한 비즈니스 프로세스 모델링에는 여러 가지 방법론이 있는데, 크게 의사소통 기반(communication-based) 방법[WINO97], 인공물 기반(artifact-based) 방법[SMITH91], 활동 기반(activity-based) 방법[GOLD99]의 세 가지 범주로 나눌 수 있다[CICH98].

앞의 두 방법은 사람들 간의 약속(commitments)을 프로세스 모델의 토대로 사용하는데 비해 활동 기반 모델링은 업무 자체의 모델링에 그 초점을 두고 있다. 활동 기반 모델링은 프로세스를 태스크의 집합으로 분해하고, 태스크 간의 의존 관계에 근거하여 그 실행 순서를 결정한다. 예를 들어 〈그림 1〉은 활동 기반 모델링을 이용하여 설비 구입 프로세스를 표현한 것이다. 이 프로세스는 세 개의 활동, 즉, 재정 상태의 검증(Verify Status), 입찰 수행(Get Bids), 주문(Order)으로 구성된다. 화살표는 활동들 사이의 의존관계 또는 순서를 설명하고 있다.

〈그림 1〉 활동 기반 프로세스 모델링 예[WINO97]



활동 기반 모델링 방법은 프로세스와 그 수행 과정, 그리고 단계별 데이터 흐름에 대해 많은 정보를 표현할 수 있다. 그래서, 프로세스 관리자나 담당자들이 프로세스에 관한 전체적인 조망을 가능하게 하고, 인공물과 활동의 상황에 대한 이해를 돕는다. 이러한 모델링 방법의 가장 큰 장점은 생성된 모델을 워크플로우 명세로 쉽게 변환할 수 있다는 것이다[WINO97]. 따라서, 현재 사용되는 많은 워크플로우관리시스템들은 활동 기반 모델링 접근법을 사용하고 있으며, 본 연구에서도 기본적으로 이 방법을 사용하고 있다.

Ⅲ. 객체지향 개념의 필요성

WFMS는 기업 환경에 따라 다양한 종류의 비즈니스 프로세스를 관리할 수 있어야 한다. 프로세스의 규모와 형태에 따라 다양한 기능을 제공하고, 프로세스 참여자의 차별적인 요구를 수렴할 수 있어야 한다. 특히, 여러 부서나 기업이 공동으로 참여하여 수행하는 프로세스의 경우에는 프로세스 설계, 통제, 분석에 있어서 여러 수준에서 관리할 수 있는 워크플로우

모델이 필요하다. 그리고, 복잡한 대규모 프로세스의 경우 프로세스가 진행되는 도중에 프로세스 모델을 변경할 필요가 생기기도 한다. 또, 특정 조직이 관리하는 프로세스의 일부는 다른 조직으로부터 독립적으로 관리할 필요도 발생한다. 이러한 점을 감안하여 WFMS가 만족해야 하는 요구사항을 정리하면 아래와 같다.

- **프로세스 설계의 체계적인 분할과 통합**

대규모 프로세스의 경우 분야별 전문가가 각자 담당하는 프로세스의 부분을 나름대로 전문성을 발휘하여 설계한 다음 이를 통합할 수 있어야 할 것이다. 설계된 부분 프로세스는 다른 프로세스에 재사용되거나 또는 필요에 따라 부분별로 쉽게 수정될 수 있어야 한다.

- **이질적 조직의 협력 프로세스에서 다양한 관리 정책**

  여러 조직이 참여하여 수행하는 프로세스의 경우 부분적으로 독립적인 프로세스를 유지할 필요가 있다. 그리고, 여러 프로세스 간의 의존 관계와 역할에 따라 관리자들에게 차별화 된 프로세스 접근 권한을 설정하고 부여할 수 있어야 한다.

- **장기 프로젝트의 동적 변경과 유연한 대응**

  장기적으로 진행되는 프로젝트의 경우 선행 프로세스의 진행 추이에 따라서 후행하는 프로세스의 흐름을 동적으로 변경할 수 있도록 지원하여 유연하게 대응할 필요가 있다.

- **업무 관련 자료의 공유 문제**

  프로세스가 진행되면서 작성된 여러 가지 정보나 문서들은 필요에 따라서 다른 업무 수행에서 참조될 수 있다. 자료의 공유 여부는 담당자와 그 역할에 따라 결정할 수도 있지만, 부분 프로세스 간의 관계를 고려하여 결정될 수도 있다.

이러한 요구를 반영하기 위하여 우리는 다음에서 설명하는 바와 같이 프로세스를 컴포넌트화 하는 방안을 제시한다. 그리고 이 프로세스의 컴포넌트화를 위한 대안으로 4장에서 설명하는 워크플로우의 객체지향 설계를 제안하는 것이다.

3.1 프로세스의 컴포넌트화

워크플로우에서 다루고 있는 프로세스는 정형화된 업무 프로세스로서 반복적으로 사용된다.. 전체 프로세스 수준에서의 반복 사용뿐만 아니라 부분적으로도 그러한 필요성이 자주 있다. 그리고, 프로세스의 일부는 임의로 변경되거나, 대체되어 사용될 수 있어야 한다. 그러므로, 이런 프로세스의 부분들은 각각 독립적인 역할을 수행할 수 있어야 한다. 이를 위해 프

로세스를 컴포넌트화 해야 할 필요가 있는 것이다.

컴포넌트란 특정 표준에 맞게 설계된 것으로 같은 표준에 의해 설계된 다른 컴포넌트와 결합하여 기능적인 그룹을 이룰 수 있는 것을 말한다. 정형화된 업무 프로세스를 컴포넌트화 함으로써 반복적으로 사용함과 동시에, 적절하게 변경하여 효과적으로 이용할 수 있게 한다. 많은 수의 태스크가 상호 연관된 복잡한 프로세스의 경우 이들을 관리상의 이유, 또는 업무 성격의 유사성 등으로 인해 몇 개의 컴포넌트 프로세스를 조합하여 구성할 수 있다면 여러 가지 이점을 기대할 수 있다. 프로세스를 컴포넌트화 하기 위해서는 다음과 같은 특성들을 보장하여야 한다.

- **독립성 (independency)**

  분리된 프로세스는 상위 프로세스와 독립적으로 자신의 기능을 수행할 수 있어야 한다.
- **재사용성 (reusability)**

  독립된 프로세스는 개별적으로, 또는 다른 프로세스의 하위 프로세스로 재사용 가능해야 한다.
- **대체가능성 (replaceability)**

  기존의 하위 프로세스 대신에 다른 하위 프로세스를 임의로 대체할 수 있어야 한다.
- **가분성 (separability)**

  기존의 복잡한 프로세스에서 의미 있는 하위 프로세스를 쉽게 분리할 수 있어야 한다.

Ⅳ. 객체지향 워크플로우

이 장에서는 프로세스 모델링에 객체지향 접근법을 접목하는 방안을 설명한다. 그리고, 이로부터 얻게 되는 객체지향적 장점을 워크플로우의 관점에서 분석한다. 마지막으로, 객체지향 프로세스 모델의 관리 상의 이점과 의미를 알아본다.

4.1 워크플로우의 객체화

프로세스 정의가 준비되면, 이 정의를 따르는 실질적인 워크플로우 프로세스 인스턴스 - 를 필요할 때마다 반복적으로 생성하여 사용할 수 있다. 프로세스 인스턴스의 실행 결과는 데이터베이스에 저장되어 추후에 분석 자료로 사용한다. 하나의 프로세스 정의를 수정하여 또 다른 프로세스를 정의할 수 있고, 이는 별개의 프로세스 인스턴스를 발생시키면서 다른

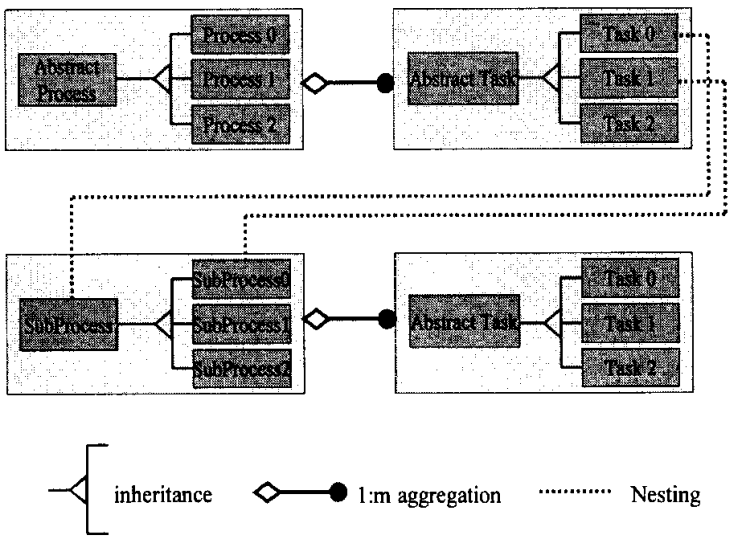
종류의 데이터로 저장된다. 세부적인 형태가 다른 비즈니스 프로세스들은 비록 공통점이 있다고 하더라도 각기 다른 프로세스로 간주하는 것이 지금까지의 프로세스 관리 방법이었다. 그러나, 여러 가지의 세부적인 형태를 가지는 비즈니스 프로세스가 다양하게 존재하고, 계속하여 새로운 형태로 변모해간다면 기존의 방법으로 프로세스 모델을 관리하는 것은 많은 문제를 안고 있다.

이러한 문제를 보완하기 위하여 프로세스 정의를 상속할 수 있는 추상 프로세스형(APT: Abstract Process Type)을 제안한다. 추상 프로세스형은 기존에 개별적인 프로세스들이 각각 여러 개의 인스턴스를 가지며 반복 사용되던 워크플로우들의 공통 부분을 추상화 시킴으로써 프로세스를 보다 효율적으로 관리하기 위하여 사용될 수 있다.

추상 프로세스는 하나 이상의 추상 태스크를 가진다. 추상 태스크는 자식 태스크로 상속되며, 이는 다시 하위 프로세스를 중첩할 수 있다. 다시 말해서 추상 프로세스는 자식 프로세스로 상속할 수 있고, 자식 프로세스는 추상 태스크를 상속 받은 자식 태스크를 가진다. 이 자식 태스크는 하나의 하위 프로세스를 중첩하고 있다. 이 때 중첩되는 하위 프로세스는 또 다른 추상 프로세스나 자식 프로세스이다.

〈그림 2〉와 같이 프로세스는 여러 개의 태스크의 집합으로 표현된다. 이 그림에서 *Abstract Process*는 세 개의 자식 프로세스, *Process0*, *Process1*, *Process2*로 상속되고 있다. *Process0*는 자식 태스크 *Task0*를 가지고 있다. 자식 태스크 *Task0*는 *Abstract*

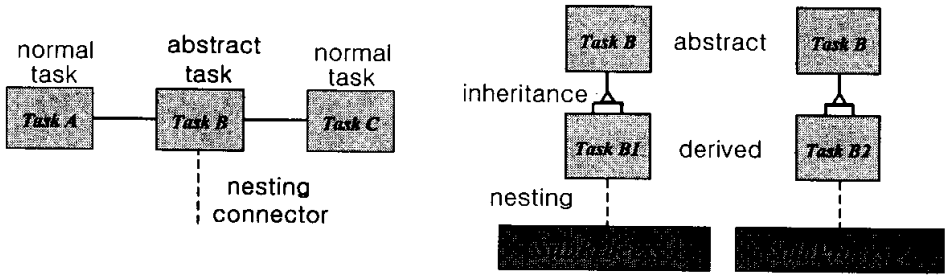
〈그림 2〉 프로세스와 태스크의 상속



*Task*를 상속하고 있고, 추상 프로세스 *Sub Process*를 중첩하고 있다. 이에 비하여, *Process1*의 자식 태스크 *Task1*는 또 다른 자식 프로세스 *Sub Process0*를 중첩하고 있다.

객체지향 워크플로우의 핵심 개념은 상속과 중첩이다. 전술한 바와 같이 상속을 이용하여 다양한 프로세스의 세부적인 사항을 효과적으로 정의할 수 있다. 중첩은 상속받은 프로세스의 자식 태스크를 독립적으로 설계하기 위한 것이다. 〈그림 3〉과 같이 추상 프로세스는 전체의 흐름을 통제하고, 상속 프로세스는 특정 추상 태스크를 어떤 프로세스와 중첩해야 하는지를 저장하고 있다. 이러한 구조를 통하여, 상위 프로세스와 하위 프로세스의 관리가 분리되어, 서로 다른 두 기업이나 부서에 의해 진행되는 분산 환경의 프로세스에 효과적으로 적용될 수 있다. 그리고, 동일한 상위 프로세스를 공유하는 여러 개의 자식 프로세스들이 독립적으로 변경될 수 있도록 지원하고 있다.

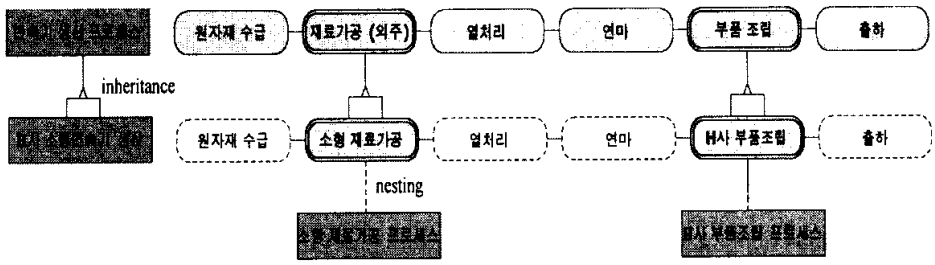
〈그림 3〉 상속 프로세스의 설계



(a) Abstract Process (b) Derived Processes

다음 예를 살펴보자. 기업에서 어떤 제품을 구입하는 구매 프로세스는 거시적으로는 동일한 업무흐름을 갖지만, 제품의 특성에 따라 프로세스의 세부적인 사항들은 조금씩 다르다. 이때 제품에 따라서 개별적인 프로세스로 정의하여 사용하는 것이 아니라, 공통적으로 수행되는 과정은 추상 프로세스를 통하여 정의하고, 특정 제품에 따라 다르게 수행되는 부분을 자식 프로세스에서 세부적으로 정의하여 사용한다. 다시 말하면, 프로세스 중에서 부분적으로 다른 사항들은 여러 개의 하위 프로세스로 모델링 한다. 그리고, 여러 개의 자식 프로세스를 상속하여 해당되는 하위 프로세스를 중첩함으로써, 부모 프로세스의 공통적인 흐름과 하위 프로세스의 차별적인 흐름을 보완적으로 수행하게 된다. 이처럼 객체지향 프로세스의 설계는 다양한 제품의 구매 프로세스와 같이 여러 형태로 변형되는 프로세스들을 정의하고 수행하는 데 효율적으로 사용될 수 있을 것이다.

〈그림 4〉 '변속기 생산 프로세스'의 상속 예제



〈그림 4〉는 추상 프로세스인 '변속기 생산 프로세스'를 상속하여 자식 프로세스 'H사 소형 변속기 생산 프로세스'를 정의한 예를 보여주고 있다. 자식 태스크 '소형 재료가공'은 추상 태스크 '재료가공(외주)'를 상속받았고, 하위 프로세스로 '소형재료가공 프로세스'를 중첩하고 있음을 알 수 있다.

4.2 객체지향적 특징

워크플로우의 객체화는 객체지향 모델링의 일반적인 특징을 가지고 있다. 프로세스를 객체화 하고 상속과 중첩을 통해 자식 프로세스를 설계하는 방법은 추상화(abstraction), 은닉화(encapsulation), 상속성(inheritance), 다형성(polymorphism)과 같은 기본적인 객체지향적 특징을 보여준다[RUMB91]. 워크플로우의 객체화가 얻는 장점을 위의 네 가지 특징의 관점에서 설명하겠다.

- **추상화(abstraction)**

  추상 프로세스형을 통하여 개괄적인 프로세스를 정의하여 관리할 수 있다. 동일한 과정을 거치는 프로세스들을 상위 수준에서 정의하여 공통적인 변경 사항과 관리를 총괄할 수 있다. 세부적인 추가 정의나 변형은 추상 프로세스형을 상속한 자식 프로세스에 중첩된 하위 프로세스에 적용한다. 국소적으로 다양한 형태를 가지는 프로세스를 집합체인 상위 프로세스라는 추상화 된 개념을 통하여 함께 관리하게 된다.

- **은닉화(encapsulation)**

  하위 프로세스를 중첩하여 구현함으로써 개별적인 하위 프로세스 정보를 은닉할 수 있다. 따라서 상위 수준에서의 개념적 설계와 관리가 용이하다. 즉, 프로세스를 다단계의 관리 수준에서 독립적으로 설계하고, 변경하며, 관리할 수 있다.

• **상속성(inheritance)**

기존의 프로세스 정의를 여러 자식 프로세스로 상속함으로써 다양한 형태의 프로세스를 효율적으로 생성하여 관리할 수 있다. 이는 국소적으로 형태가 다른 자식 프로세스들을 공통 집합인 상위 프로세스로 묶어주는 것으로 볼 수도 있다. 즉, 여러 가지 형태로 파생된 프로세스 정의를 하나의 추상 프로세스형을 통하여 통제한다.

• **다형성(polymorphism)**

동일한 객체가 '하나 이상의 형태를 띠는 것', 즉 '같은 명령에 대하여 객체가 서로 다르게 반응하는 것'이 다형성이다. 동일한 상위 프로세스가 어떤 하위 프로세스를 적용하여 실행하느냐에 따라서 실제로는 다른 흐름으로 진행된다. 추상 프로세스형의 인스턴스가 실행시간에 적절한 자식 프로세스의 인스턴스로 전이됨으로써 여러 가지 형태로 수행될 수 있다는 것이다.

4.3 프로세스의 계층적 관리

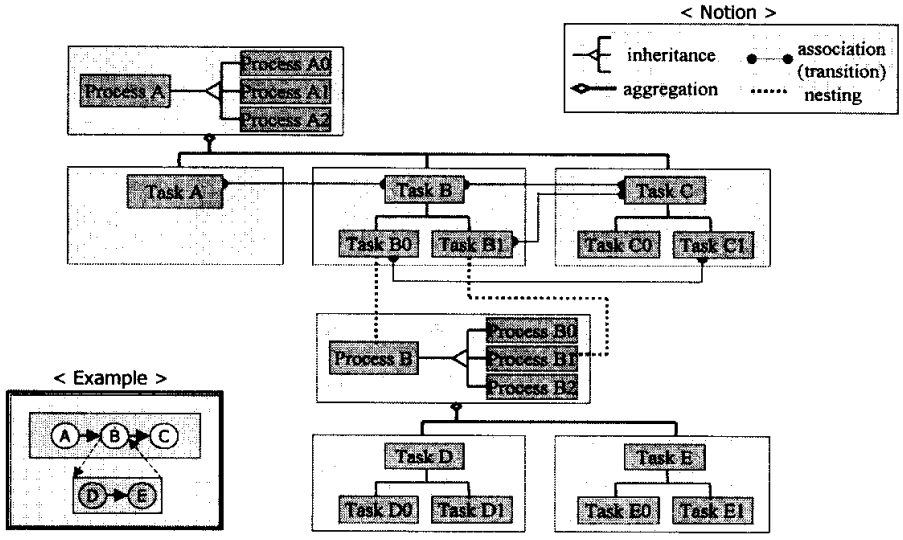
프로세스를 관리하기 위해서는 프로세스 정의를 유지하고 있어야 한다. 그러나 비슷한 종류의 프로세스가 많이 있을 때 이는 워크플로우의 관리 측면에 몇 가지 문제를 야기시킨다. 다음과 같은 경우를 생각해 보자. WFMS에서 실질적인 업무 프로세스는 인스턴스를 생성하여 수행한다. 프로세스 인스턴스가 진행되는 도중에 하위 프로세스를 통하여 전체 프로세스의 흐름을 변경하였다고 해보자. 이는 결국 같은 프로세스 정의에 의해 실행되는 프로세스 인스턴스가 다른 프로세스로 진행되는 결과를 가져온다. 시스템에서 일시적으로 필요한 프로세스 변경을 반영한다는 점에서는 의미가 있다. 그러나, 비슷하지만 실질적으로는 다른 두 개의 프로세스를 수행하였기 때문에 수행된 프로세스 결과를 체계적으로 저장하여 과거 프로세스를 분석하는 등과 같은 관리에는 어려움이 있다는 것이다.

이 같은 프로세스 정의 유지 문제와 프로세스 인스턴스 저장 문제는 워크플로우를 체계적으로 관리하는 것을 힘들게 한다. 프로세스 정의 유지 문제에는 여러 가지 정책이 있을 수 있다. 먼저, 하나의 프로세스 정의만을 유지하는 것을 생각해 보자. 일시적으로 변경된 프로세스 정의를 복구하거나, 또는 변경된 프로세스 정의를 계속 유지하게 될 것이다. 그러나, 이처럼 하나의 정의를 유지하는 방법은 인스턴스의 결과 분석이나 활용에 다른 프로세스 정의를 따르는 인스턴스를 올바르게 반영할 수 없는 문제가 있다. 그렇지 않고, 변경 전후의 두 프로세스 정의를 모두 유지하여 이들이 무관한 것으로 간주하는 것 역시 바람직한 분석 결과를 제공하지 않는다.

워크플로우의 특성상 프로세스가 자주 실행되고, 변경된 프로세스의 결과도 모두 중요하다고 가정하면, 다양하게 실행되는 프로세스들의 총괄적인 관리와 개별적인 관리가 동시에 이루어져야 한다. 객체지향 워크플로우는 이러한 문제를 해결할 수 있다. 즉, 다양한 형태로 확장 가능한 프로세스의 공통된 진행 과정은 추상 프로세스형으로 모델링 하여 상위 프로세스로 정의, 변경, 관리하고, 개별적인 세부 과정들은 자식 프로세스가 중첩하고 있는 하위 프로세스에서 관리하게 한다.

〈그림 5〉는 하나의 상위 프로세스가 유사한 형태의 하위 프로세스들을 가질 수 있는 경우에, 상속을 통하여 자식 프로세스를 정의한 예를 보여준다. 몇 가지 발생 가능한 형태를 자식 프로세스 정의로 미리 설계해 둘 수도 있고, 실행 시에 하위 프로세스를 변경하거나 새롭게 정의할 수 있다. 즉, 프로세스 인스턴스가 진행할 때에는 부모 프로세스로 시작되어 진행 도중에 자식 프로세스로 전이되어 진행될 수도 있고, 초기부터 자식 프로세스가 직접 실행될 수도 있다. 이러한 방법은 세부적으로 정의된 프로세스를 직접 시작 시킬 수 있을 뿐만 아니라, 진행중인 프로세스가 상세한 프로세스로 전이되어 실행될 수 있고, 하위 프로세스를 새롭게 정의하여 사용할 수 있도록 함으로써 프로세스의 유연성과 정형성을 적절히 부여하고 있는 것이다.

〈그림 5〉 프로세스의 상속과 중첩



V. 프로세스 계층 모델과 접근 권한

본 논문에서는 워크플로우의 객체화를 통하여 실행시간에 하위 프로세스를 새롭게 정의하거나 변경할 수 있도록 객체지향 프로세스 모델링을 설명하였다. 특히 중첩 모델을 사용하여 하위 프로세스를 상위 프로세스와 명확히 분리함으로써 실행시간에 유연한 프로세스의 변경을 보장할 수 있다는 것이 중요하다. 이 장에서는 프로세스의 복잡한 중첩 관계를 효율적으로 관리하기 위한 프로세스 계층 모델(Process Hierarchy Model)과 이를 이용한 프로세스 접근 권한 통제 방법을 제안한다.

5.1 프로세스 계층 트리

프로세스는 실행 중에 자식 프로세스를 정의하여, 임의의 중첩 관계를 복잡하게 가질 수 있다. 중첩된 하위 프로세스는 새롭게 정의하거나, 기존 프로세스를 그대로 또는 변형하여 사용할 수 있다. 이러한 프로세스의 상속은 재귀적으로 발생할 수 있고, 프로세스의 중첩 관계가 임의로 형성되고 변경될 수 있다. 이러한 이유로, 프로세스 간의 관계를 관리하기 위한 체계적인 모델이 요구된다. 그리고, 중첩 관계의 두 프로세스 간 접근 권한을 정의하고 관리하기 위한 정책이 필요하다.

프로세스는 실행시간 중에 임의의 상속 과정을 거쳐 다양한 형태의 하위 프로세스를 가지게 된다. 자식 프로세스로 한 번 상속될 때마다, 하나 이상의 중첩 관계가 추가된다. 그런데, 그 하위 프로세스가 또다시 상속될 수 있기 때문에 다단계로 중첩 관계가 형성되어 프로세스 중첩은 계층을 이루게 된다. 이러한 프로세스의 중첩 관계는 실행시간에 형성되고 변경될 수 있기 때문에 이를 체계적인 모델로 관리할 필요가 있다. 본 논문에서는 이러한 중첩 관계를 처리하기 위한 자료구조로써 트리(tree)를 사용하는데, 이를 프로세스 계층 트리(Process Hierarchy Tree)라고 부른다. (그림 6 참조)

프로세스 계층 트리는 실행시간에 추가되거나 변경될 수 있는 자식 프로세스의 중첩 관계를 나타내기 위한 모델이다. 이는 다음 절에서 설명하는 중첩 관계가 있는 프로세스 간에 접근 권한을 설정하는 데 유용하게 사용되어 프로세스 간의 업무흐름(control-flow)과 자료흐름(data-flow)을 통제하는 수단을 제공한다.

5.2 프로세스 간 접근 권한

중첩 프로세스 모델에서는 상 하위 프로세스의 관리자가 다르게 지정될 수 있다. 즉, 상 하

위 비즈니스 프로세스가 다른 조직이나 부서에서 독립적으로 실행될 때 이들 간의 접근 권한을 통제할 필요가 있다는 것이다. 프로세스 상호간의 접근 권한은 연결 모드(coupling mode)로 구분할 수 있는데 연결 모드를 결정하는 주요한 요인들은 〈표 1〉과 같다. 즉, 중첩 관계의 프로세스 사이에 설정할 수 있는 다섯 가지 접근 권한은 프로세스 정의의 변경 권한, 담당자의 변경 권한, 프로세스 통제 권한, 프로세스 모니터링 권한, 데이터의 접근 권한이다.

먼저, **acModifyProcDef**와 **acChangeWorker**는 원래 프로세스 인스턴스가 발생하기 전에 확정되는 사항이지만, 실행시간 중에도 수정이 가능한 사항들이다. 그리고, **acControlProcInst**와 **acMonitorProcInst**는 진행 중인 프로세스 인스턴스를 통제하는 관리자의 권한을 나타낸다. 마지막으로, **acShareDoc**는 프로세스를 수행하는데 필요한 문서들을 중첩 관계에 있는 프로세스 실행자와 공유하도록 할 것인지에 관한 설정이다.

〈표 1〉 중첩 프로세스의 접근 권한

접근 권한	설 명
프로세스 정의 변경 권한 (**AcModifyProcDef**)	실행시간에 중첩 관계의 프로세스 정의를 변경할 수 있는 권한
담당자 변경 권한 (**AcChangeWorker**)	실행시간에 중첩 관계의 프로세스 담당자를 교체할 수 있는 권한
프로세스 통제 권한 (**AcControlProcInst**)	실행시간에 중첩 관계의 프로세스 인스턴스를 통제 - 일시 중지/재개(suspend/ resume), 거부(reject), 취소(abort), 실패(fail), 완료(finish) - 할 수 있는 권한.
프로세스 모니터링 권한 (**AcMonitorProcInst**)	중첩 관계의 프로세스 인스턴스의 진행 상황을 모니터링할 수 있는 권한
데이터 접근 권한 (**AcShareDoc**)	중첩 관계의 프로세스에서 사용하는 문서를 공유할 수 있는 권한

5.3 접근 권한 통제

전 절의 다섯 가지 접근 권한은 중첩 관계에 있는 상위 프로세스와 하위 프로세스가 존재할 때, 양방향으로 설정해야 한다. 즉, 상위 프로세스를 실행하는 관리자가 하위 프로세스의 업무흐름(control-flow)을 통제하거나 자료흐름(data-flow)에 접근할 수 있는 권한을 허락할 것인지, 반면에 하위 프로세스를 실행하는 관리자가 상위 프로세스를 통제하거나 문서에 접근할 수 있게 할 것인지를 구분하여 설정한다. 이 권한은 중첩 관계에 있는 프로세스 i로부터 프로세스 j로의 접근을 허용할 경우에는 1, 불허할 경우에는 0의 값으로 표시되는 다섯

개의 순서쌍 AC_{ij}로 표현한다.

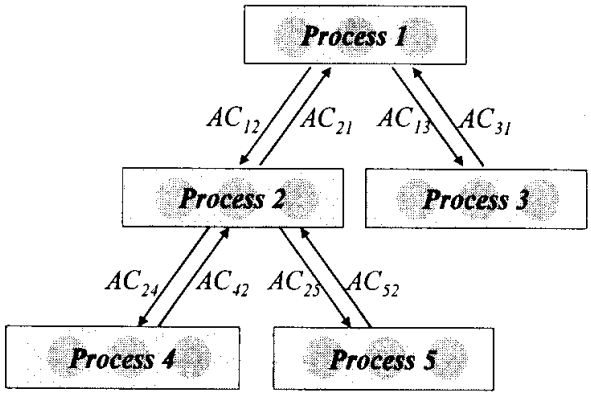
AC_{ij} = ($ac1$, $ac2$, $ac3$, $ac4$, $ac5$)

AC_{ij} = access control from Process i to Process j

$ac1$, $ac2$, $ac3$, $ac4$, $ac5$ = 1 or 0

ac1, **ac2**, **ac3**, **ac4**, **ac5** 는 각각 **acModifyProcDef**, **acChangeWorker**, **acControlProcInst**, **acMonitorProcInst**, **acShareDoc**를 나타낸다. AC_{ij}를 상속 프로세스의 중첩 구조를 표현하고 있는 프로세스 계층 트리에 표시함으로써, 자식 프로세스 내의 여러 프로세스 간에 접근 권한을 총괄적으로 관리할 수 있게 한다.

〈그림 6〉 프로세스 계층 트리와 접근 권한



예를 들면, 〈그림 6〉에서 *Process1*에서 *Process2*의 접근 권한은 대응되는 노드에 표시된 AC_{12}를 통해서 판단하게 된다. AC_{12} = (0, 1, 0, 1, 1)이라고 가정하면, *Process1*의 관리자는 *Process2*의 프로세스에 대하여 $ac1$=0, $ac2$=1이므로 프로세스 정의를 변경시킬 수 있는 권한은 없지만, 프로세스의 작업을 수행할 작업자들은 바꿀 수 있는 권한을 가지게 된다. 그리고 $ac3$=0, $ac4$=1이므로 *Process1*의 관리자는 *Process2*의 프로세스 인스턴스를 일시 중지/재개(Suspend/Resume), 거부(Reject), 취소(Abort), 실패(Fail), 완료(Finish) 시킬 수 있는 권한은 없지만, *Process2*의 프로세스 인스턴스의 진행 과정을 모니터링할 수는 있다. 또한, $ac5$=1이므로 *Process1*의 관리자는 *Process2*에서 사용하는 문서나 자료들을 필요하면 참조할 수 있는 권한을 가진다.

그리고, 직접 중첩되지 않은 프로세스들 사이의 접근 권한도 프로세스 계층 트리를 이용하여 판정할 수 있다. 예를 들어, *Process4*에서 *Process3*으로의 권한은 위와 같은 방법으로,

AC42, ***AC21***, ***AC13***을 순차적으로 탐색함으로써 판단할 수 있다. 만약 ***AC42*** = (0, 1, 0, 1, 1); ***AC21*** = (0, 0, 1, 1, 1), ***AC13*** = (1, 1, 1, 1, 1)이라면,

AC43 = (0×0×1, 1×0×1, 0×1×1, 1×1×1, 1×1×1) = (0, 0, 0, 1, 1)

이 되어 결국 *Process4*의 관리자는 *Process3*의 프로세스 인스턴스의 진행 상황을 모니터링할 수 있는 권한과, *Process3*의 문서를 관리할 수 있는 권한을 가지게 된다.

이러한 정책을 수용할 수 있는 근거는 순차적으로 권한을 확장할 수 있다고 가정하였기 때문이다. 만약, *Process1*의 관리자가 *Process2*를 관리할 권한이 있고, *Process2*의 관리자가 *Process3*을 관리할 권한을 가지고 있다고 가정하면, *Process1*의 관리자는 *Process3*를 관리할 수 있는 권한을 가진다는 것을 전제로 한다.

Ⅵ. 결 론

본 논문에서는 객체지향 워크플로우 모델링을 제안하였다. 이는 분산 환경에서 여러 기업이나 조직이 참여하는 복잡한 비즈니스 프로세스를 효과적으로 관리하고, 실행시간에 일어나는 프로세스 변화에 유연하게 대응할 수 있는 프로세스 모델링 방법이다.

객체지향 워크플로우 모델은 중첩 프로세스를 사용하여 여러 조직이 독립적으로 프로세스를 설계하고 관리할 수 있게 하였으며, 분산 환경에서 여러 조직이 참여하는 유동적인 협업 과정을 워크플로우로 설계하여 관리하는 경우에 더욱 효과적이다. 또, 필요에 따라서는 독립적인 엔진을 사용하여 다중서버 시스템을 구축할 수 있도록 하였다. 그리고, 실행시간에 중첩 관계에 있는 하위 프로세스를 독립적으로 생성, 수정, 교체할 수 있도록 함으로써 동적 프로세스를 지향하였다. 프로세스 정의를 상속하여 여러 형태로 파생된 자식 프로세스를 생성하는 반면에, 추상 프로세스를 통하여 통합하여 수정, 관리할 수 있도록 함으로써 재사용성과 확장성을 증가시켰다. 이러한 프로세스 모델은 컴포넌트화 되어 조직별로 독립된 엔진과 데이터베이스 서버를 구동할 수 있도록 함으로써 시스템의 부하를 분산시키고, 부서 간 정보를 효과적으로 보호할 수 있다.

참 고 문 헌

[CICH98] A. Cichocki, A. Helal, M. Rusinkiewicz, and D. Woelk, Workflow and Process Automation: Concepts and Technology, Kluwer Academic Publication, pp. 4-18, 1998.

[GOLD99] S. Goldmann and B. Kotting, "Software Engineering over the Internet," IEEE Internet Computing, Vol. 3, Issue 4, pp. 93-94, 1999.

[HOLL95] D. Hollingsworth, The Workflow Reference Model, The Workflow Management Coalition Specification, WfMC-TC-1003, Jan. 1995.

[KIM2000] Y. Kim, S.-H. Kang, D. Kim, J. Bae, and K.-J. Ju, "WW-flow: A Web-based Workflow Management System Supporting Run-time Encapsulation," IEEE Internet Computing, Vol. 4, Issue 3, pp. 55-64, 2000.

[RUMB91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, Object-Oriented Modeling and Design, Prentice Hall, 1991.

[SMITH91] J. B. Smith and F. D. Smith, "ABC: A Hypermedia System for Artifact-Based Collaboration," Proceedings of Hypertext '91, ACM Press, 1991.

[WINO97] T. Winograd and R. Flores, Understanding Computers and Cognition, Addison-Wesley, 1997.