웹 기반 워크플로우관리시스템의 설계와 개발

강 석 호·김 영 호*

《目 次》

Ⅰ. 서 론
Ⅱ. 관련 연구
Ⅲ. 시스템 구조
Ⅳ. 중첩 프로세스 설계
Ⅴ. 웹 기반 워크플로우관리시스템 기능

Abstracts

본 연구에서는 중첩 프로세스의 설계와 관리가 가능한 웹 기반 워크플로우관리시스템을 설계하고 개발하였다. 워크플로우관리시스템이 제공하는 여러 장점 때문에 이는 최근에 등장한 대부분 기업정보시스템의 핵심 모듈로 취급되고 있다. 이 논문에서는 워크플로우를 효과적으로 설계하는 방법론을 제시하고, 그리고 이기종 분산 업무 환경에서도 사용할 수 있는 웹 기반 워크플로우관리시스템을 설명하고자 한다. 중첩 설계 방안은 복잡한 프로세스를 보다 효율적으로 설계할 수 있고, 이 설계를 바탕으로 효과적으로 프로세스 실행을 관리하는 메커니즘을 개발할 수 있었다. 개발한 시스템의 가장 핵심적인 부분이라고 할 수 있는 워크플로우 엔진은 100% 자바로 개발되었으므로 이기종 분산 환경에서 시스템간의 상호운영성 보장해 줄 수 있다. 그리고, 모든 클라이언트 인터페이스를 애플릿으로 개발하였으므로 웹 브라우저를 사용할 수 있는 어떤 사용자도 사용 장소에 구애받지 않고 시스템에 접근할 수 있다.

Ⅰ. 서 론

워크플로우는 컴퓨터가 통제하고 관리하는 비즈니스 프로세스를 말하며, 워크플로우관리시스템은 이 워크플로우를 정의하고 그에 따라 비즈니스 프로세스를 실행하고 관리하는 소프트웨어이다〈4〉. 최근 주목받고 있는 전사적자원관리시스템 (ERP: Enterprise Resource

* 서울대학교 산업공학과 교수

Planning), 제품정보관리시스템 (PDM: Product Data Management), 공급망관리시스템 (SCM: Supply Chain Management), 고객관계관리시스템 (CRM: Customer Relationship Management), 전자상거래 (EC: Electronic Commerce) 등과 같은 대부분의 기업정보시스템은 이 워크플로우 관리 기능을 포함하고 있다. 이는 이들 기업정보시스템이 지원하는 업무가 대부분 절차에 따라 진행되는 프로세스를 가지고 있기 때문이다.

워크플로우관리시스템은 다음과 같은 장점을 제공하여 업무 생산성을 제고한다. 첫째, 업무 흐름을 자동으로 통제하기 때문에 복잡한 프로세스를 거치는 업무도 손쉽게 처리할 수 있다. 둘째, 정보 및 문서 전달이 전자화 되므로 업무를 신속히 처리할 수 있고, 또 불필요한 종이 문서가 감소된다. 셋째, 데이터를 일관적으로 접근하고 제어하므로 데이터 불일치에 따른 문제나 정보의 부정확성이 감소하고, 따라서 업무 처리 결과의 정확도가 높아진다. 넷째, 업무 진행 현황을 실시간에 투명하게 감독할 수 있다. 다섯째, 그룹웨어 기능을 가지므로 능률적인 공동작업이 가능하다. 이러한 장점 때문에 워크플로우관리시스템은 앞에서 언급한 정보시스템의 핵심 모듈로 취급되고 있다.

이 논문에서는 워크플로우를 효과적으로 설계하는 방법론을 제시하고, 그리고 이를 분산된 업무 환경에서도 효율적으로 이용하기 위해 개발한 웹 기반 워크플로우관리시스템을 설명하고자 한다.

업무 프로세스의 복잡도는 매우 다양하게 나타나는데, 분산 환경에서 처리하는 업무 프로세스의 일반적인 특징은 그 상대적인 복잡도가 높다는 것이다. 여러 부서나 다른 조직간의 협동 작업이 필요한 경우가 많으며, 주변 상황이나 업무의 중간 결과에 따라 프로세스의 진행이 달라지기도 하고, 몇 개의 하위 프로세스가 동시에 병렬적으로 진행되기도 한다. 프로세스를 설계하려면 프로세스의 처음부터 끝까지 프로세스를 정의하는데 필요한 모든 사항을 알고 있어야 한다. 그런데 복잡도가 큰 프로세스의 경우 이를 한 수준에서 모두 설계하는 것은 매우 어렵다. 본 연구에서는 이런 프로세스를 효율적으로 설계하기 위해 중첩 프로세스 (nested process) 개념을 사용할 것을 제안한다. 중첩 프로세스는 직접적인 선후관계가 있는 몇 개의 작업을 그룹화 하여 하나의 단위로 취급하는 것을 말한다. 이는 프로세스의 모듈화 설계를 가능하게 하고, 실행 및 통제 시에도 이 단위별로 모듈화된 관리를 할 수 있다. 또한, 프로세스 설계의 재사용성을 높일 수도 있는데, 이는 자주 사용하는 업무 프로세스 단위를 라이브러리 형태로 관리함으로써 가능하다.

한편, 인터넷 사용 인구가 급속도로 증가함에 따라 많은 정보 시스템들이 웹 인터페이스를 제공하고 있다. 워크플로우관리시스템의 경우에도 기존의 많은 업체들이 웹 기반 시스템을

개발하여 제공하고 있다. 웹 기반 워크플로우관리시스템의 가장 큰 장점은 웹브라우저를 통해서 시스템에 접근할 수 있으므로 시간이나 장소에 구애받지 않고 시스템을 사용할 수 있다는 것이다. 또, 별도의 클라이언트 프로그램을 설치할 필요가 없으므로 시스템의 운용, 유지보수, 수정이 편리하다는 장점도 있다.

웹 기반 워크플로우관리시스템은 분산된 업무 환경에서 프로세스를 관리하기 위해 사용될 것이다. 이때 위에서 언급한 프로세스의 중첩 설계 개념이 매우 유용하게 사용될 수 있다. 즉, 중첩 프로세스 각각을 분산 환경에서 독립적으로 통제와 제어가 가능한 단위로 설계할 수 있다는 것이다. 조직 외부에서 행해지는 작업들을 그룹화 하여 중첩 태스크로 모델링 하여 외부로 전달하면, 외부 조직에서는 상세한 프로세스를 정의하여 실행한 후 그 결과를 전달해 줄 수 있다. 이렇게 함으로써 동일 조직 내에서의 프로세스 관리뿐만 아니라 기업간의 프로세스를 연동하기 위해서도 워크플로우관리시스템을 이용할 수 있다.

이 논문은 다음과 같이 구성된다. 2장에서는 관련된 연구들을 정리하였다. 3장에서는 개발한 시스템의 전체 구조와 각 구성요소들 간의 상호작용을 설명하였다. 4장에서는 모듈화된 프로세스 설계 기능을 자세히 설명하였다. 5장에서는 개발한 웹 기반 워크플로우관리시스템의 기능을 프로세스 디자이너, 워크플로우 엔진, 그리고 클라이언트 인터페이스로 나누어 설명하였으며, 6장은 이 논문의 결론이다.

Ⅱ. 관련 연구

워크플로우관리시스템에 대한 주요 연구 결과를 표준화와 웹 기반 시스템 관련 연구로 나누어 본 연구와의 연관성을 알아보고자 한다.

먼저, 워크플로우의 표준화에 대한 연구로는 이 분야의 표준화 작업을 주도하고 있는 국제단체인 WfMC(Workflow Management Coalition)가 제안한 참조모델이 대표적이라 할 수 있다. 이 참조 모델에서는 워크플로우관리시스템을 프로세스 디자이너, 엔진, 클라이언트, 각종 응용 프로그램간 인터페이스 등의 구성 요소로 정의하고 있다. 본 연구에서 개발한 시스템도 WfMC의 표준 모델과 동일한 구조를 가지고 있다.

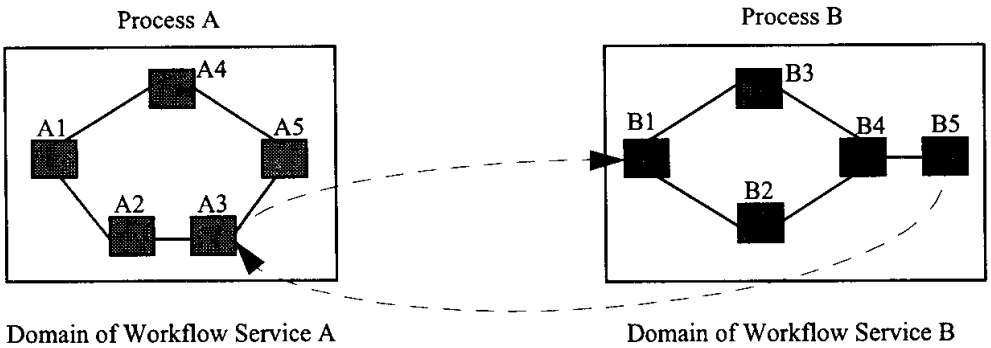
한편, WfMC에서는 이기종 워크플로우관리시스템간의 상호운영성(interoperability)을 위한 네 가지 모델을 제안하고 있는데, Connected Discrete (Chained) 모델, Hierarchical (Nested Subprocesses) 모델, Connected Indiscrete (Peer-to-Peer) 모델, Parallel Synchronized 모델이 그것이다.

Connected Discrete (Chained) 모델에서는 한 프로세스 내에 연결점이 있어서 이를 통해 다른 프로세스의 연결점과 연결된다. 이는 두 개의 프로세스에서 하나의 업무 항목을 전달하는 것을 지원한다. 업무 항목을 전달한 후에는 두 프로세스가 더 이상 동기화될 필요 없이 독립적으로 동작한다.

Connected Indiscrete (Peer-to-Peer) 모델은 두 개 이상의 워크플로우관리시스템에서 프로세스가 완전히 혼합되어 구동되는 모델이다. 각 워크플로우 엔진은 상호작용을 통해 사용자나 관리자의 특별한 통제가 없어도 투명하게 동작하는 것을 목표로 한다. 이 시나리오를 완벽히 구현하기 위해서는 관련된 엔진이 모두 프로세스 정의를 이해할 수 있어야 한다. 그러나 아직 WfMC 등 표준화 단체에서 제안한 표준이 완벽하지 않고 또 완전히 수용되지 않고 있는 문제가 있다.

Parallel Synchronized 모델은 두 프로세스가 사실상 독립적으로 작동하되, 프로세스 사이에 동기화 지점을 지정하여 그 지점에서 상호작용하는 모델이다. 이 모델은 Connected Indiscrete 모델보다는 간단하지만, 이 역시 표준 사양이 완벽하게 정의되고 수용되어야만 가능하다.

본 연구의 중첩 프로세스 개념은 동일 조직 내의 복잡한 프로세스 관리뿐만 아니라 엔진간의 상호운영성을 구현하기 위한 방법으로, Nested Subprocesses 모델과 동일하다. 〈그림 1〉은 이 모델의 예를 보여주고 있다. 프로세스 사이에는 계층적 관계가 있는데, 그림과 같은 경우 프로세스 B는 프로세스 A의 단일 태스크로 캡슐화 된다. 이때 B를 A의 하위 프로세스라 한다. 이 같은 계층적 관계는 여러 단계에 걸쳐서 나타날 수 있는데, 본 연구에서는 중첩 단계가 몇 단계이든지 상관없이 일관성 있게 프로세스를 설계할 수 있도록 하였다. 이 모델에 대해서는 4장에 상세히 설명하였다.



〈그림 1〉 Nested Sub-processes 모델 〈4〉

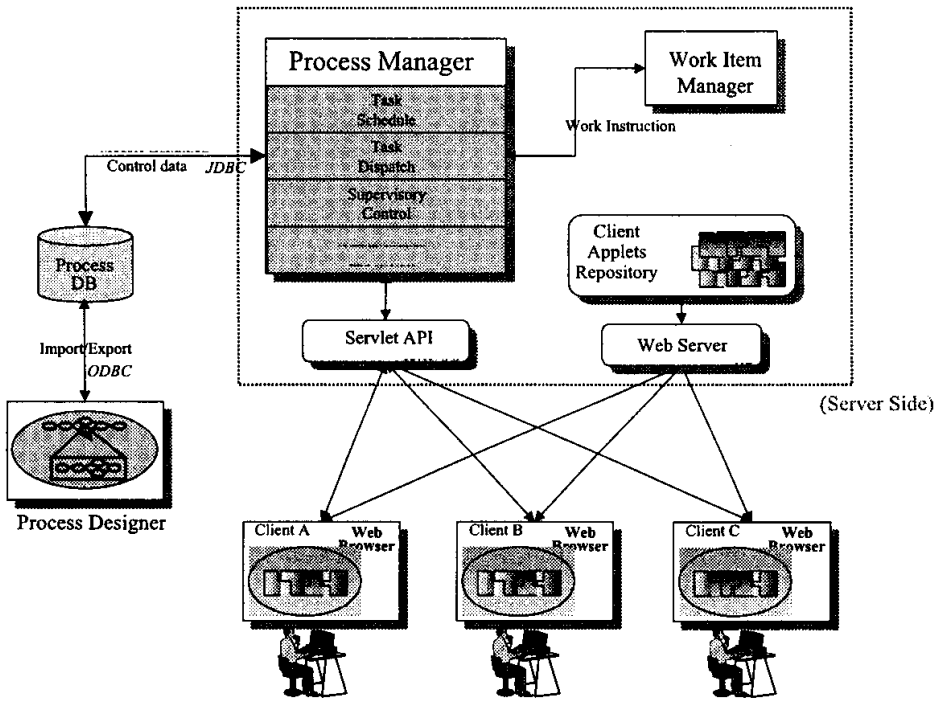
워크플로우 표준화에 대한 또 다른 연구로 PIF(Process Interchange Format) Working Group에서는 프로세스 정보 교환 형식에 대한 표준을 연구하였다〈9〉. 그리고 최근에 IETF(Internet Engineering Task Force)의 Working Group에서는 SWAP(Simple Workflow Access Protocol)이라는 이종 워크플로우 시스템 사이의 상호운영성을 제공하기 위한 표준 프로토콜을 제안하였다〈11〉. SWAP은 비동기적인 프로세스를 시작, 중지, 통제하기 위한 메커니즘을 제공하는 것이 목적이다. 이는 단순, 경량, 확장 가능한 프로토콜로 인터넷상에서 장기간에 걸쳐 진행되는 워크플로우에서 정보를 교환하기 위해 이용될 수 있다.

웹 기반 워크플로우관리시스템에 대한 기술도 인터넷 관련 기술이 발전함에 따라 동반하여 발전하고 있다. Georgia 대학의 LSDIS 연구실에서 개발한 WebWork는 CGI (Common Gateway Interface)를 이용하여 개발한 초보적 시스템이라고 할 수 있다〈10〉. Dartmouth 대학의 DartFlow는 이동 가능 에이전트(mobile agent)와 CGI, 자바 등의 기술을 이용하여 개발한 시스템이다〈13〉. 대표적인 상용 웹 기반 시스템으로는 Staffware 사의 Staffware, Action Technologies 사의 Metro, Ultimus 사의 Ultimus, Fujitsu 사의 i-Flow 등이 있다. Staffware는 자바 기반의 웹 클라이언트를 출시하였는데, 기존의 클라이언트 기능을 모두 구현하고 있으며 그 성능을 개선하고 기능을 확장하였다〈12〉. Metro 또한 웹 기반 클라이언트 프로그램을 제공하고 있는데, 본 연구에서 개발한 시스템과 마찬가지로 프로세스 디자이너(Builder)는 웹으로부터 분리되어 있다〈1〉. 이 시스템에서 업무 할당은 WorkBox를 통한 확인(pull) 방식과 E-mail로 Worklink를 전해주는 전달(push) 방식으로 이루어진다. i-Flow 시스템은 자바를 기반으로 분산 확장 가능한 아키텍처로 설계되었으며, 100% 자바로 된 엔진을 가지고 있다〈5〉. 클라이언트는 디자이너, 단순 클라이언트, 초기화 클라이언트, E-mail 클라이언트로 구분된다. Ultimus는 NT Server에서 작동하는 서버로 Microsoft Transaction Server, COM/DCOM, Java, DHTML 등의 기술을 이용하여 개발되었다〈14〉.

본 연구에서는 이들 기존 시스템의 기능과 구조를 분석하여 시스템 구현에 참조하였다. 기존의 시스템에 비해 본 연구에서 개발한 시스템이 가지는 가장 큰 특징이자 장점은 이기종 분산환경에서 발생하는 복잡한 프로세스를 모듈화 하여 설계하고 관리하는 기능이다. 중첩 프로세스 방식은 인터넷을 통한 이종 워크플로우 시스템간의 상호운영성을 위한 토대가 될 것이다. 또, 플랫폼 독립적인 시스템을 개발하기 위해 클라이언트와 엔진을 100% 자바로 구현하였다.

Ⅲ. 시스템 구조

본 연구에서 개발한 시스템의 전체적인 구조는 〈그림 2〉와 같다.



〈그림 2〉 시스템 전체 구조

〈그림 2〉에서 보는 바와 같이 시스템은 크게 프로세스 디자이너, 워크플로우 엔진, 워크플로우 클라이언트로 구성되어 있다. 프로세스 디자이너는 모듈화된 프로세스의 모델링을 지원하고, 설계된 프로세스 정의를 엔진이 이해할 수 있는 형태로 변환하여 이를 데이터베이스에 저장한다. 신규로 프로세스를 모델링 한 경우에는 프로세스 모델의 오류를 먼저 검증한 후 데이터베이스에 저장한다. 저장된 프로세스 모델은 언제든지 프로세스 디자이너에서 수정하거나 다른 프로세스 모델에 활용될 수 있다. 프로세스 디자이너는 독립 실행형 클라이언트로 개발되었으며, 데이터베이스와의 연결은 ODBC(Open Database Connectivity)를 이용하였다.

워크플로우 엔진은 프로세스 데이터베이스에 저장된 프로세스 정의에 따라 전체 업무 흐름

을 통제하고 감독한다. 그리고 수행 대상 업무를 프로세스 진행 상황에 따라 선택하여 특정 업무담당자의 작업항목관리자(Work Item Manager)에 할당한다. 이 밖에 전체 프로세스 관리, 통계 자료 제공, 단일 프로세스 인스턴스에 대한 모니터링을 수행하기 위한 기능도 제공한다. 엔진 쪽에 있는 작업항목관리자 모듈은 클라이언트에 할당되는 작업 리스트를 관리한다. 프로세스 데이터베이스는 엔진이 워크플로우를 구동하는데 필요한 통제 데이터와 관련 데이터를 저장하여 관리한다. 이들은 모두 NT 서버에서 구동된다. 전술한 바와 같이 엔진은 순수 자바로 구현하였으므로 플랫폼 독립적으로 설치할 수 있다.

워크플로우 클라이언트는 업무 담당자가 실제로 업무를 처리할 때 사용하는 사용자 인터페이스를 말한다. 모든 클라이언트 모듈은 자바 애플릿으로 구현하였다. 사용자는 웹브라우저를 통해 URL을 입력하여 애플릿을 다운로드 받아 태스크 수행, 모니터링, 관리 기능 등의 작업을 수행하는데, 사용자의 역할과 권한에 따라 할 수 있는 업무가 자동으로 차별화 된다.

한편, 클라이언트와 엔진 사이의 통신 메커니즘으로는 HTTP 프로토콜을 사용하는 서블릿(servlet)을 이용하였다. 서블릿은 Java Server Technology의 일환으로 Sun 사의 JavaSoft 디비전에서 제공한 기술이다〈8〉. 이는 현재 CGI의 효과적인 대안으로서 CGI와는 달리 멀티쓰레딩(multi-threading) 방식으로 사용자 요구를 처리한다. 멀티쓰레딩을 사용하기 때문에 여러 사용자로부터 많은 요구가 동시에 들어오더라도 프로세스를 실행하는 것 이상의 부하가 걸리지 않고, 따라서 효과적으로 프로세스를 처리할 수 있는 장점이 있다.

Ⅳ. 중첩 프로세스 설계

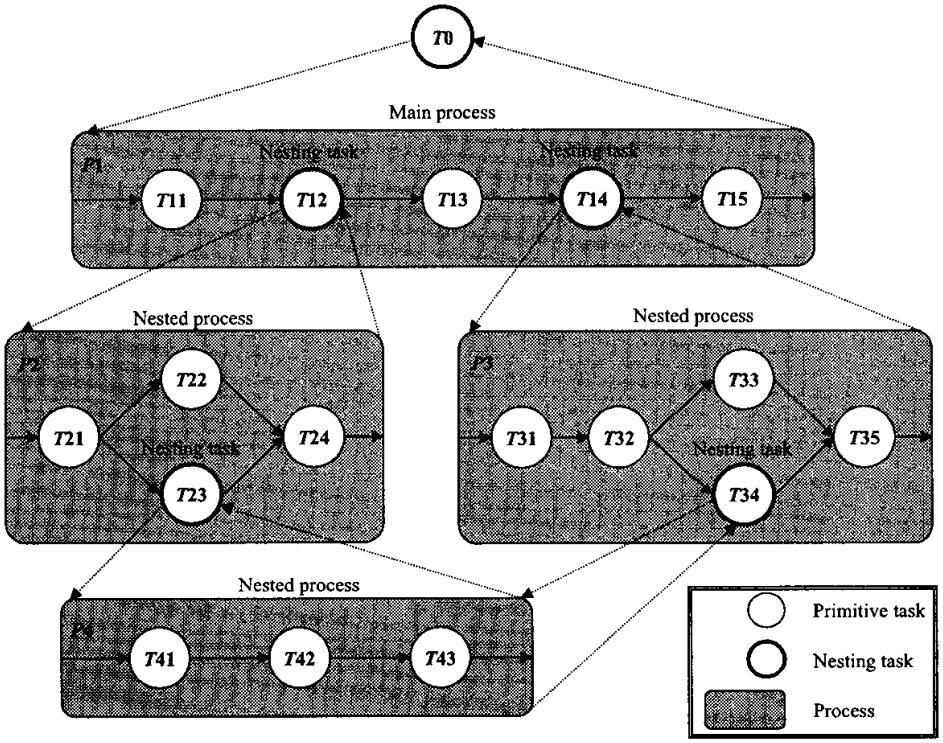
이 장에서는 프로세스의 중첩 설계와 이를 이용한 실행 시 캡슐화 기능, 그리고 중첩설계를 위한 데이터베이스 스키마 및 주요 엔진 구동 메커니즘을 설명하였다.

4.1 중첩 프로세스의 구조

일반적으로 프로세스의 구조는 이를 구성하는 여러 태스크와 이들 간의 순서 관계로 정의된다. 이때, 태스크는 프로세스를 구성하는 요소로서 프로세스의 하위 개념이다. 그러나 중첩 프로세스에서는 태스크가 가장 상위 개념이 되며, 하나의 태스크는 이를 더욱 자세히 기술하는 프로세스를 포함할 수 있다. 태스크는 프로세스를 포함하는 동시에 자기 자신은 다른 프로세스의 구성요소가 될 수 있다. 다시 말하면 태스크는 프로세스의 자식이 될 수도 있고 부모가 될 수도 있으므로, 기존의 방법처럼 프로세스의 하부로만 존재하는 것이 아니라 프로세

스와 같은 수준에서 정의된다.

〈그림 3〉은 중첩 프로세스 구조의 예를 보여주고 있다. 그림에서 동그라미는 태스크를 나타내고, 네모는 프로세스를 나타낸다. 태스크는 다시 두 종류로 나누어지는데, 가는 실선으로 표시된 것은 원시 태스크(primitive task)이고, 굵은 실선은 중첩 태스크(nesting task)이다. 두 태스크를 연결하는 아크는 이들간의 선후관계를 나타낸다. 그림에서 중첩 태스크는 항상 하나의 프로세스로 전개됨을 알 수 있다. 즉, 중첩 태스크는 전개된 프로세스에 의해 더욱 자세히 모델링되는 것이다. 이때 이 프로세스를 중첩된 프로세스(nested process)라 한다. 반면에 원시 태스크는 더 이상 분해할 수 없다.



〈그림 3〉 중첩 프로세스의 구조

그림에서 보는 것처럼 루트 태스크(root task) T0가 가장 상위에 있으면서, 이는 주프로세스(main process)라고 부르는 프로세스 P1으로 전개된다. 프로세스 P1은 다섯 개의 하위 태스크들과 이들간의 선후관계로 정의되어 있다. 이 태스크들 가운데 태스크 T12와 T14는 중첩 태스크로 다시 프로세스 P2와 P3로 자세히 전개된다. 이 같은 중첩은 몇 단계에 걸쳐서 일어 날 수 있으며, 단계의 개수에는 제한이 없다. 한편, 프로세스 P4처럼 동일한 프로

세스가 여러 곳에 사용될 수도 있다.

중첩 프로세스를 실행하는 과정을 간단히 설명하면 다음과 같다. 어떤 프로세스를 나타내는 루트 태스크를 실행하면, 이는 바로 자신의 하부에 있는 주프로세스를 실행한다. 이제 프로세스를 구성하는 태스크들의 선후관계에 따라 프로세스가 진행된다. 프로세스를 수행하던 중에 중첩 태스크를 만나면, 이 태스크는 하부의 프로세스를 시작하고, 이 하부 프로세스가 종료하면 수행 결과를 반환받는다. 최상위의 프로세스가 끝나면 그 결과값을 루트 태스크에 돌려주게 되고, 루트 태스크의 경우에는 다음 태스크가 없으므로 바로 종료된다. 이러한 방식으로 진행할 경우 중첩이 여러 단계에 걸쳐서 일어나도 중첩 태스크나 중첩된 프로세스간의 관계가 항상 동일하므로 일관성을 잃지 않고 작동할 수 있다.

중첩 프로세스 개념을 사용하여 프로세스를 모델링함으로써 다음과 같은 이점을 기대할 수 있다.

- Top-down 방식으로 프로세스를 모델링하므로, 복잡한 프로세스도 쉽게 설계하고, 분석할 수 있다.
- 캡슐화, 다형성, 상속성 등과 같은 객체지향모델의 장점을 프로세스 모델링과 시스템 구현에 차용할 수 있는 이론적 근거를 제공하여 프로세스 모델링과 이를 구동하는 시스템의 개발이 간편해진다.
- 자주 사용하는 프로세스 단위들은 라이브러리 형태로 제공하여 모델의 재사용성이 증가하고, 이는 프로세스 설계에 필요한 노력을 줄여준다.
- 프로세스 하나하나가 분산 환경에서 독립적으로 통제와 제어가 가능한 단위이므로 실행중인 프로세스의 정의를 변경하는 것이 용이하다.
- 사용자의 조직상의 위치나 권한에 따라 프로세스 실행, 감독 등 관리 활동을 다양한 수준에서 할 수 있다.
- 프로세스 설계가 모듈화 되므로 분산된 서로 다른 조직이 하나의 프로세스에 서로 관련되어 있을 때도 쉽게 프로세스의 설계와 실행을 분리하고 통합할 수 있다.
- 이기종 워크플로우 시스템간의 상호운영성을 위한 방안으로 사용될 수 있다.

반면에 대부분의 기존 워크플로우관리시스템은 일원적인 프로세스 설계 방식을 채택하고 있는데, 여기에는 모듈 또는 중첩 개념이 없어 프로세스를 처음부터 끝까지 한꺼번에 설계해야 한다. 따라서 이 방식으로는 위에 나열한 중첩 설계 방식의 이점을 기대하는 것이 매우 어렵다.

4.2 실행 시 캡슐화

중첩 프로세스 모델은 프로세스를 실행 시에 캡슐화하는 것을 가능하게 한다. 하나의 프로세스에 서로 다른 부서나 조직이 개입하여 업무가 진행되는 경우 필요에 따라 중첩된 하위 프로세스를 캡슐화 하여 실행 내용과 결과를 보호해야 한다. 중첩 프로세스 모델에서 상위 수준의 프로세스에서는 하위 프로세스의 입력과 출력만 정의하면 된다. 또, 캡슐화 기능은 전체 프로세스를 완전히 정의하기 힘든 경우에도 프로세스를 정의하여 실행할 수 있도록 해준다. 상위 프로세스의 중첩 태스크의 입력과 출력만 정의한 후, 나머지 상세한 프로세스 구성은 해당 부서나 조직에서 정의하여 실행할 수 있다.

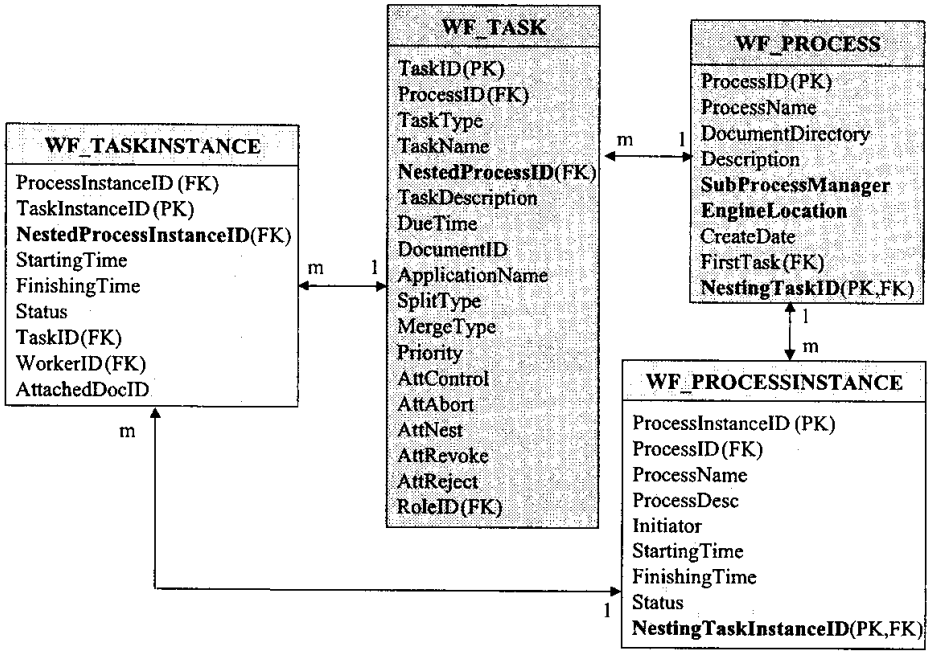
즉, 상위 수준의 중첩 태스크가 하위 프로세스를 호출할 때 이를 관리할 하위 엔진에게 프로세스의 입출력 데이터, 관리자, 작업 기한 등의 정보를 넘겨준다. 이렇게 하기 위해서 중첩 태스크는 담당 엔진의 위치를 알고 있어야 한다. 전달받은 정보를 바탕으로 하위의 엔진은 해당 중첩된 프로세스를 실행시킨다. 이때 하위 프로세스 모델이 준비되지 않았을 수도 있다. 이 경우 상위 엔진이 하위 엔진이 수행할 프로세스 모델을 같이 전달할 수도 있고, 하위 수준에서 자체적으로 설계할 수도 있다. 하위 프로세스가 종료하면 엔진은 상위 엔진에게 그 결과를 반환하게 되고, 상위 엔진은 다음 태스크를 진행한다.

이 같은 실행 시 캡슐화 기능은 기존의 시스템이 제공하지 않는 몇 가지 장점이 있다. 첫째, 특히 분산된 업무 환경에서 독립적인 업무가 맞물려 진행될 때 유용하다. 둘째, 중첩된 프로세스는 다른 엔진에 의해 별개로 진행되므로 시스템의 확장성(scalability)이 높아진다. 셋째, 하위 프로세스 모델의 변경을 부서나 조직의 상황에 맞게 자유롭게 할 수 있다. 이는 하위 프로세스의 국지적 변경이 전체 프로세스에 영향을 미치지 않기 때문이다. 마지막으로 캡슐화 된 하위 프로세스를 모니터링하기 위한 권한을 차별화 할 수 있다. 예컨대 전체 프로세스 모델 관리자는 하위 프로세스의 개략적 진행 상황만을 모니터링할 수도 있고, 적절한 권한이 있는 경우 자세한 내용까지도 관리할 수 있는 것이다.

4.3 중첩 설계를 위한 데이터베이스 스키마

〈그림 4〉는 중첩 설계된 프로세스 정의를 저장하기 위한 프로세스와 태스크의 관계를 나타내는 데이터베이스 스키마이다. 여기서 프로세스(WF_PROCESS) 테이블과 태스크(WF_TASK) 테이블은 프로세스의 정적인 구조를 저장하기 위한 것이고, 프로세스인스턴스(WF_PROCESSINSTANCE) 테이블과 태스크인스턴스(WF_TASKINSTANCE) 테이블은 프로세스가 실행이 되어 진행될 때 발생하는 동적인 정보를 저장하는 테이블이다.

프로세스의 중첩 설계 정보를 저장하기 위해서 태스크 테이블에 중첩 태스크를 더 상세히 기술하는 하위 수준의 중첩된 프로세스를 가리키는 애트리뷰트 NestedProcessID가 있고, 프로세스 테이블에는 NestingTaskID 애트리뷰트에 상위 수준에서 이 프로세스를 가리키는 태스크 정보를 가지고 있다. WF_TASK와 WF_PROCESS는 m : 1 관계를 가지는데, 이는 하나의 프로세스가 여러 개의 중첩 태스크를 포함할 수 있기 때문이다. 실행 시의 정보를 저장하는 WF_TASKINSTANCE와 WF_PROCESSINSTANCE도 같은 이유로 동일한 관계를 가진다.



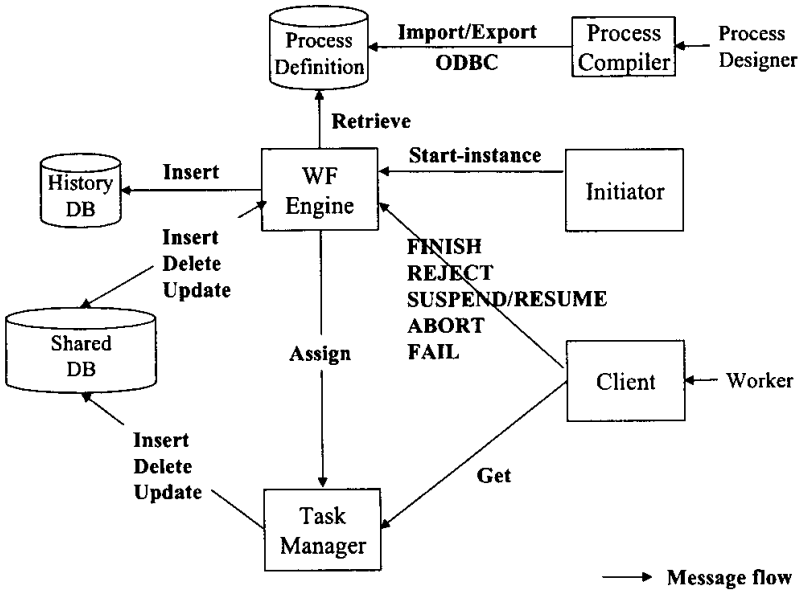
〈그림 4〉 중첩 설계를 위한 데이터베이스 스키마

4.4 태스크와 프로세스의 속성

여기서는 개발한 시스템이 사용하는 태스크와 프로세스의 속성과 행동 양식을 정의한다.

〈그림 5〉는 워크플로우 구성 모듈간에 주고받는 메시지를 보여준다. 크게 서버 측 모듈(사각형으로 표시)과 클라이언트 측 모듈(원으로 표시)로 나눌 수 있다. 여기서 워크플로우 엔진과 일반 클라이언트의 상호작용이 가장 중요하다. 일반 클라이언트가 작업을 마치면 그 결과는 WF 엔진으로 전달되는데, 이 때 결과의 종류에 따라 FINISH, REJECT, SUSPEND, RESUME, ABORT, 또는 FAIL 가운데 하나의 반환 코드 (return code) 메시지가 엔진에

전달된다. 각 코드의 의미는 〈표 1〉에 정리되어 있다.



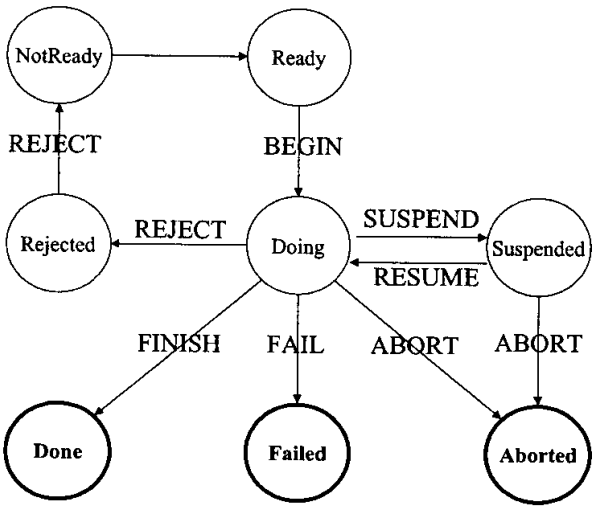
〈그림 5〉 워크플로우 구성 모듈간의 전달 메시지

중첩 모형에서 태스크와 프로세스가 가지는 상태(status)에 대해 알아보자. 중첩 모형에서는 태스크와 프로세스를 똑같이 취급하기 때문에 가질 수 있는 상태도 동일하다. 〈표 1〉과 같이 모두 8가지 상태가 있다. 태스크의 초기 상태는 "NotReady"이고 선행 태스크가 모두 완료되면 "Ready"가 된다. 이때 워크플로우 엔진에서 클라이언트로 작업을 할당(Assign)하면 상태가 "Doing"으로 바뀌고 작업 완료를 기다린다. 작업이 진행된 다음의 태스크 상태는 위에서 설명한 반환 코드에 의해서 결정된다.

〈표 1〉 반환 코드와 태스크 상태

Return Code	태스크 상태	설명
	NotReady	초기 상태
	Ready	선행 태스크가 모두 Done일 때
Assign	Doing	태스크 시작
FINISH	Done	정시 완료
SUSPEND/RESUME	Suspended	일시 정지
REJECT	Rejected	태스크 흐름 반려
ABORT	Aborted	프로세스 전체 취소
FAIL	Failed	태스크 실패 및 대체 태스크 시작

〈그림 6〉은 반환 코드에 따른 태스크 상태의 변이를 나타내는 상태전이다이어그램이다. 작업이 진행 중임을 나타내는 "Doing" 상태에서 반환 코드에 따라 Rejected, Suspended, Done, Failed, 또는 Aborted 상태로 상태가 바뀐다. 태스크의 최종 상태는 Done, Failed, Aborted 중에 하나이다.



〈그림 6〉 태스크 상태전이 다이어그램

한편, 다양한 형태의 업무 흐름을 처리할 수 있도록 하기 위하여 다음과 같은 분기 형태를 정의하고 사용하였다.

- 동시분기(Concurrent Split) : 동시분기를 하는 태스크의 완료는 모든 후행 태스크를 병렬적으로 시작시킨다. 이 모든 태스크들이 완결되어야 동시분기가 끝난다.
- 선택분기(Alternative Split) : 선택분기를 하는 태스크의 완료는 모든 후행 태스크를 병렬적으로 시작시킨다. 이 중 적어도 한 태스크가 끝나면 선택분기가 끝난다.
- 배타적 선택분기(Exclusive OR Split) : 배타적 선택분기를 하는 태스크가 완료된 다음 후행 태스크 중에 하나만 시작된다.
- 조건분기(Conditional Split) : 조건분기를 하는 태스크가 끝나면 조건을 평가하여 그 결과에 따라 하나의 후행 태스크가 선택된다.

V. 웹 기반 워크플로우관리시스템 기능

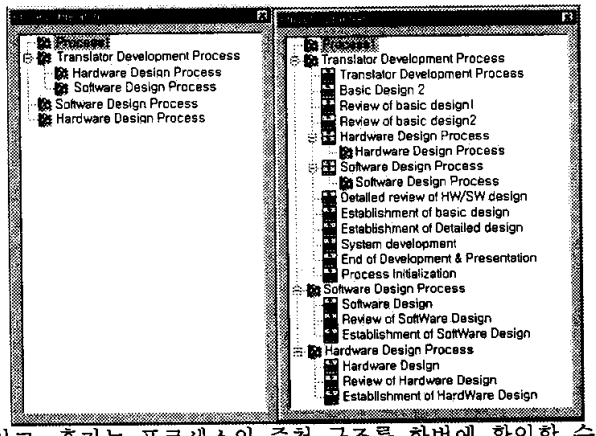
개발한 웹 기반 워크플로우관리시스템의 기능을 프로세스 디자이너, 워크플로우 엔진, 그리고 클라이언트 인터페이스로 나누어 알아보기로 한다.

5.1 프로세스 디자이너

프로세스 디자이너는 워크플로우를 설계할 때 사용하는 도구로 다음과 같은 기능을 제공한다.

프로세스 설계 및 편집: 작업 팔레트와 다이어그램 뷰로 구성되는 GUI (Graphical User Interface) 환경에서 다이어그램 형태로 업무 프로세스를 설계하고, 각 태스크에 대해 그 속성과 분기 조건을 입력할 수 있다.

중첩 설계 및 편집: 트리 형태의 계층 구조를 갖는 프로세스를 설계할 수 있으며, 이를 프로세스 브라우저를 통해 조감할 수 있다. 〈그림 7〉은 프로세스 브라우저로 중첩 프로세스의 계층 구조를 나타낸 모습이다. 두 가지 트리가 제공되는데, 좌측은 태스크와 중첩된 프로세스들을 한 단계로 제공하고, 우측은 프로세스의 중첩 구조를 최하위 수준까지 보여준다. 전자는 다이어그램 뷰의 설계 환경과 연동되어 여러 프로세스를 한 번에 설계할 경우에 효율적



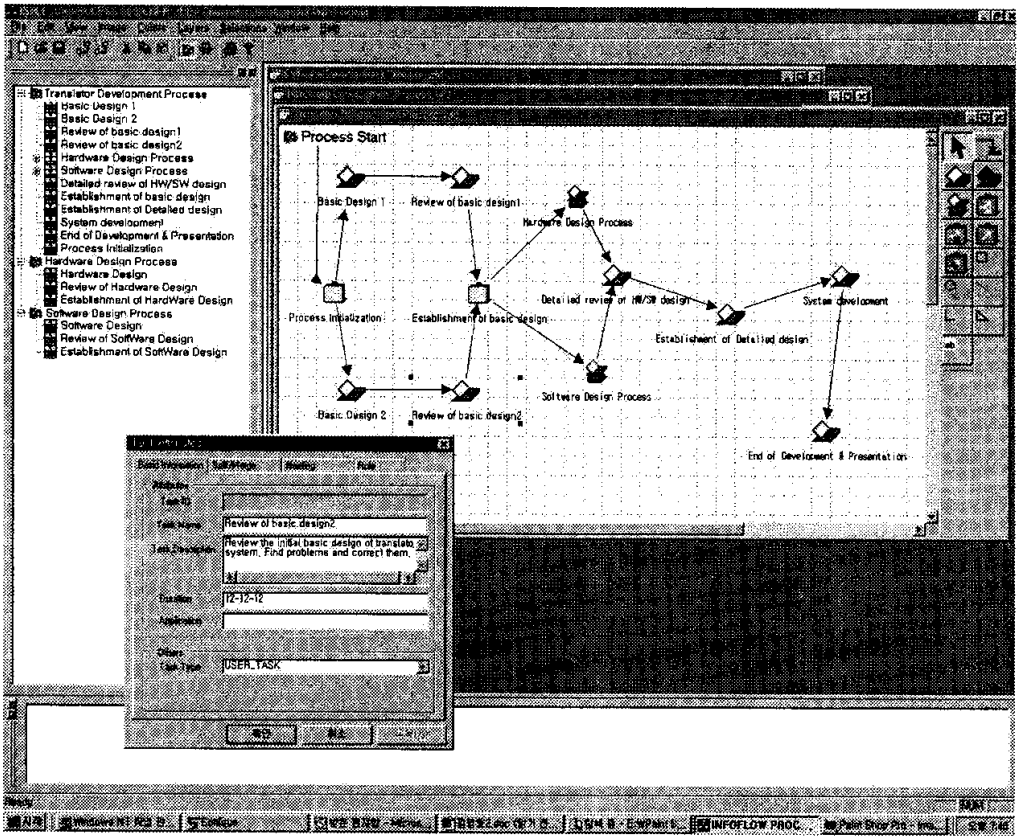
으로 사용될 수 있고, 후자는 프로세스의 중첩 구조를 한번에 확인할 수 있다.

〈그림 7〉 프로세스 브라우저가 제공하는 중첩 프로세스 계층 구조

오류 검증 및 데이터베이스 입출력: 엔진이 프로세스를 정확히 구동하기 위해서는 설계에 오류가 없어야 한다. 프로세스 디자이너는 설계된 프로세스를 데이터베이스에 저장하기 전에

이를 검증하여 기본 속성 누락, 잘못된 태스크간 연결 관계, 태스크 속성과 연결 관계의 불일치 등의 설계 오류를 발견할 수 있다. 프로세스에 오류가 없으면 데이터베이스에 저장할 수 있다. 또, 데이터베이스에 저장된 프로세스를 호출하여 수정하거나 다른 프로세스의 하부 프로세스로 재사용할 수도 있다.

프로세스 디자이너는 Visual C++로 구현하였으며, GUI 개발을 위해서 Stingray 사의 Objective Diagram 라이브러리를 이용하였다. 〈그림 8〉은 프로세스 디자이너를 이용하여 중첩 프로세스를 설계하는 화면을 보여 주고 있다.



〈그림 8〉 프로세스 디자이너를 이용한 중첩 프로세스 설계

5.2 워크플로우 엔진

워크플로우 엔진은 프로세스의 실행 환경을 제공하는 소프트웨어이다. 이는 데이터베이스의 프로세스 정의를 해석하고, 프로세스 인스턴스를 생성하며, 이의 진행을 통제한다. 그리고, 엔진은 로그 데이터, 클라이언트 감독 명령, 시스템 관리 등 워크플로우관리시스템이 제

공하는 부가 서비스에 필요한 데이터를 제공하는 핵심 모듈이다. 엔진은 100% 자바로 개발하였으며, 위의 서비스를 사용하는 클라이언트는 해당 애플릿을 다운로드 받아서 엔진의 서블릿과 요청과 응답을 주고받는다. 엔진의 프로세스 실행 제어 순서는 다음과 같이 여섯 가지 처리로 나누어서 설명할 수 있다.

반환코드 처리: 태스크 수행 결과가 서버로 돌아올 때 반환코드의 종류에 따라 해당되는 처리를 한다.

프로세스 종료 처리: 마지막 태스크의 반환코드가 FINISH이면, 해당 프로세스를 종료하고, 그 프로세스를 중첩하고 있는 태스크의 반환코드를 FINISH로 바꾼다. 루트 태스크가 종료된 경우에는 바로 전체 프로세스가 종료된다.

태스크 분기 처리: 후행 태스크를 찾기 전에 우선 그 태스크의 분기 유형을 확인하고, 각 분기 유형별로 해당되는 후행 태스크 탐색 함수를 호출한다.

후행 태스크 탐색: 이는 태스크 분기 처리에서 호출된다. 각 분기 유형별로 정의된 함수가 있고, 각 함수별로 정해진 루틴에 따라 후행 태스크를 얻어 온다.

후행 태스크 병합 처리: 후행 태스크로 결정된 태스크의 병합 유형을 확인한다. 병합 유형은 분기 유형과 일대일 대응되며, 각 유형별로 해당 함수를 호출한다. AND 병합인 경우에는 병합되는 모든 태스크들이 완료된 다음 후행 태스크를 실행할 수 있다.

중첩 태스크 처리: 이는 다음 알고리즘을 이용하여 중첩 프로세스를 처리한다. 이 알고리즘은 먼저 후행 태스크로 결정된 태스크의 중첩 여부를 확인한다. 중첩 태스크이면 그 태스크가 중첩하고 있는 프로세스를 시작한다. 중첩이 아닌 경우 작업부하 균형 메커니즘을 이용하여 작업자를 선택하고 작업 항목 관리자로 작업을 할당한다. 동시에 자바 메일 API를 사용하여 사용자에게 작업이 할당되었음을 E-mail로 통지한다.

```
void HandlingNestedProcess(TaskInstanceID id)
{
        TaskInstanceID cid;
        ProcessInstanceID pid;
        UserID uid;
        ProcessManager manager;
        If(CheckAttribute(id)) {
                If(CheckDesigned(id)) {
                        If(CheckInThisEngine(id)) {
                                pid=InitiateNestedProcess(id);
                                cid=FindFirstTask(pid);
```

```
                        SetTaskStatus(cid, READY);
                        HandlingNestedProcess(cid);
                }
                Else
                        NotifytoApprEngine(id, manager);
        }
        Else {
                If(CheckTaskType() == Dummy); {
                        AutoFinish(id);
                }
                Else  {
                        AsignTask(id);
                        SetTaskStatus(id, DOING);
                        uid=GetAvailableUser(id);
                        Notify(uid);
                }
        }
    }
}
```

5.3 클라이언트 인터페이스

클라이언트 인터페이스는 초기화 사용자 클라이언트, 일반 클라이언트, 모니터링 클라이언트, 시스템 관리 클라이언트의 네 가지로 분류되는데, 사용자의 역할에 따라 적절한 클라이언트 인터페이스가 제공된다. 사용자는 자신의 계정을 이용하여 로그인하고, 정해진 권한에 따라 각 클라이언트 모듈이 제공하는 기능을 수행할 수 있다. 클라이언트 인터페이스는 애플릿으로 개발하였는데 볼랜드 사의 JBuilder 2.0과 JDK 1.2로 구현하였다. 클라이언트 모듈에 대응되는 서버 측 서블릿은 Sun 사의 JSDK 2.0으로 구현하였다. 데이터베이스는 오라클을 이용하였으며, 오라클 사의 JDBC thin 드라이버를 통해 자바 엔진과 데이터베이스를 연결하였다. 각 클라이언트 인터페이스가 제공하는 기능은 다음과 같다.

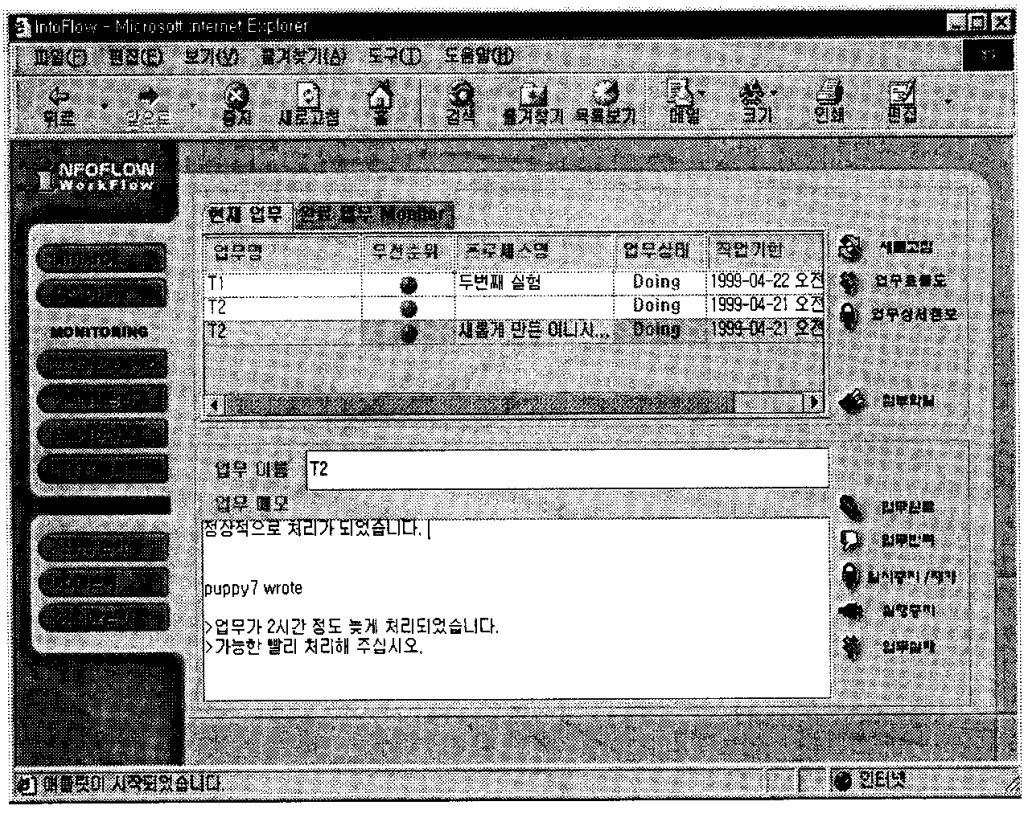
초기화 사용자 클라이언트: 프로세스 정의를 선택하여 해당 프로세스의 인스턴스를 생성하는 애플리케이션이다. 이 때 인스턴스별로 인스턴스 이름, 설명, 작업 기한, 태스크 담당자 등 필요한 데이터를 입력할 수 있다.

일반 클라이언트: 이 클라이언트는 일반 사용자가 워크플로우 엔진과 상호작용하면서 실제 업무를 처리하는 애플리케이션이다. 사용자에게 할당된 업무리스트를 관리하며, 현재 진행하는 태스크에 대한 정보와 관련 데이터를 저장하고, 처리된 태스크의 결과를 서버로 전달하는 기능을 수행한다. 〈그림 9〉는 일반 클라이언트의 실행 화면인데, 현재 사용자에게 할당된 업

무리스트를 보여주고 있다. 또한, 오른쪽에 있는 버튼은 선택된 현재 업무에 대한 상세 정보나, 이 업무가 포함된 프로세스 모델을 조회할 때 사용한다. 메모 입력 기능을 통해 후행 작업자에게 전달할 업무 내용을 기록할 수도 있다. 오른쪽 아래의 버튼들은 태스크의 결과를 엔진으로 전달할 때 사용한다.

모니터링 클라이언트: 이 모듈을 이용하여 진행 중인 프로세스의 상태를 감시하고, 작업 독촉, 일시 중지/재개, 실행 중지 등의 통제 명령을 내리며, 업무 수행 이력에 대한 통계 정보를 열람할 수 있다. 권한이 부여된 사용자만이 이 기능을 이용할 수 있다. 본 연구에서는 중첩을 지원하기 때문에 다양한 수준에서의 모니터링이 가능하다. 줌인-아웃(zoom in/out) 기능을 통해 프로세스 최상위 수준에서 축약된 정보를 열람할 수도 있고, 하위 수준의 상세한 내용도 파악할 수 있다.

시스템 관리 클라이언트: 이는 워크플로우에 관계된 사용자, 역할, 그룹, 권한 등을 추가, 수정, 삭제하는 일을 지원한다. 또, 시스템 사용 환경도 여기서 설정할 수 있다.



〈그림 9〉 일반 클라이언트 실행 화면

Ⅵ. 결 론

본 연구에서는 중첩 프로세스의 설계와 관리가 가능한 웹 기반 워크플로우관리시스템을 설계하고 개발하였다. 이 시스템은 복잡한 프로세스의 중첩 설계를 지원함으로써 보다 효율적으로 프로세스를 설계하고 관리할 수 있으며, 프로세스의 재사용성을 높일 수 있다. 그리고, 이 시스템의 가장 핵심적인 부분이라고 할 수 있는 엔진은 100% 자바로 개발되었다. 따라서 이기종 분산 환경에서 시스템간의 상호운영성을 효과적으로 제공한다. 또, 프로세스 디자이너를 제외한 모든 클라이언트를 애플릿으로 개발하였으므로 웹 인터페이스가 가능한 어떤 사용자도 사용 장소에 구애받지 않고 시스템에 접근할 수 있다.

전술한 바와 같이 워크플로우 관리 기능은 PDM, ERP, CALS, EC 등과 같은 비교적 최근에 등장한 기업정보시템의 근간을 이룬다. 본 연구에서 개발한 웹 기반 워크플로우관리시스템은 이러한 시스템들이 웹 환경에서 운영될 수 있음을 보여주고 있을 뿐만 아니라, 이들 시스템을 효과적으로 통합하기 위해 이용될 수도 있다. 이 때 중첩 설계 기능이 매우 유용하게 사용될 수 있다.

감사의 글

본 연구는 한국과학재단 특정기초연구비 (97-02-00-09-01-3) 지원으로 수행되었으며 지원에 감사를 드립니다.

참 고 문 헌

[1] Action Technologies, Inc., ActionWorks Metro: e-Process Application Platform, http://www.actiontech.com/Products/Metro/Overview/, 1999.

[2] C.Mohan, G.Alonso, R. Günthör, M.Kamath, Exotica: A Research Perspective on Workflow Management Systems, Data Engineering Vol. 18, No. 1, 1995

[3] Dave Rosenberg, Bringing Java to the Enterprise: Oracle on Its Java Server Strategy, IEEE Internet Computing Vol. 2, No. 2, 1998, pp. 52-59.

[4] David Hollingsworth, Workflow Management Coalition Specification: The Workflow Reference Model, WfMC specification, 1994

[5] Fujitsu Software Corporation, i-Flow Architectural White Paper, available online at http://www.i-flow.com/about_iflow/archwhitepaper.pdf , 1999

[6] Graig Schlenoff, Amy Juntilla, Steven Ray, Unified Process Specification Language: Requirements for Modeling Process, NIST, 1996

[7] Gregory Alan Bolcer, Gail Kaiser, SWAP: Leveraging the Web to Manage Workflow, IEEE Internet Computing, Vol. 3, No. 1, 1999, pp. 85-88.

[8] James Duncan Davidson, Suzanne Ahmed, Java Servlet API Specification, Java Software Division of Sun Microsystems, Inc., November 1998.

[9] Jintae Lee, Michael Gruninger, Yan Jin, Thomas Malone, Austin Tate, Gregg Yost, The PIF Process Interchage Format and Framework v 1.1, PIF Working Group, May 24, 1996

[10] John A. Miller, Devanand Palaniswami, Amit P. Sheth, Krys J. Kochut and Harvinder Sing, WebWork: METEOR2's Web-based Workflow Management System. Technical report, University of Georgia, April 1997.

[11] K. Swenson, Simple Workflow Access Protocol (SWAP), Strawman document,Aug. 1998, available online at http://www.ietf.org/internet-drafts/draft-swenson-swap-prot-00.txt.

[12] Staffware PLC, Staffware Functionality, white paper, 1999, available online at http://www.staffware.com/home/products/2000WhitePaper.zip

[13] T. Cai, P. Gloor, and S. Nog. DartFlow: A Workflow Management System on the Web using Transportable Agents. Technical report, Dartmouth College, 1997, available online at http://www.cs.dartmouth.edu/reports/authors/ Cai,Ting.html

[14] Ultimus, Inc., Ultimus Workflow Suite: Product Brief available online at http://www.ultimus1.com/release40/dld/ultbrief.pdf , 1999

[15] Walt Scacchi, John Noll, Process Driven Intranets: Life-Cycle Support for Process Reengineering, IEEE Internet Computing, Vol. 1, No. 5, 1997, pp. 42-49.