Journal of the Korean Institute of Industrial Engineers
Vol. 26, No. 3, pp. 227-237, September 2000.

능동형 DB를 이용한 워크플로우 프로세스의 자동 실행†

배준수[1] · 김영호[2] · 강석호[2]

[1]LG-EDS 시스템(주) 제조, 엔지니어링 사업부 / [2]서울대학교 산업공학과

Automatic Enactment of Workflow Processes using Active Databases

Joonsoo Bae[1] · Yeongho Kim[2] · Suk-Ho Kang[2]

A workflow management system is a software system to assist designing processes, controlling and managing the execution of the designed processes. One emerging trend in many recent information systems is the provision of process management functions. In this paper, we propose a method of designing processes for automatic process execution directly from process modeling. First of all, the concept of block is presented which is to define a nested process model. A block is the minimum unit that can specify the relationships of process components, i.e., tasks. A general process can be defined by a combination of the blocks defined in this paper. An algorithm is developed to transform a general flat process model into a nested model. We identify basic types of blocks and build ECA (Event-Condition-Action) rules for each of the basic types. This allows us to automate the execution of the process model by using the active features of active databases.

1. 서 론

본 논문은 업무 프로세스(또는 워크플로우) 관리를 위한 소프트웨어 시스템 개발의 새로운 접근법을 제시한다. 기업의 업무 프로세스는 그 프로세스의 목표를 달성하기 위해 필요한 여러 단위업무로 구성되고, 이 업무들은 어떤 비즈니스 규칙(로직 또는 룰)이 규정하고 있는 절차를 준수하면서 진행된다. 이때 업무 프로세스는 일반적으로 조직을 구성하는 여러 부서 또는 업무 담당자를 거쳐 완성된다. 그런데 최근 기업의 업무 프로세스는 두 가지 관점에서 크게 변하고 있다. 첫 번째 변화는 내부적인 것으로 고객의 다양한 요구사항을 수용하기 위해 차별화된 업무 프로세스를 제공해야 한다는 점에서 온다. 또 다른 변화는 기업 외부에서 발생하는데, 기업간 제휴나 협력업체와의 긴밀한 협조 체제 구축 등 지리적, 조직적으로 유리된 기업간의 공동 작업이 기업 경쟁력 제고의 발판으로 부각되고 있다는 것이다. 이 두 가지 변화는 필연적으로 업무 프로세스를 복잡하게 만든다. 이와 더불어 프로세스 처리의 정확도와 신속도가 경쟁력을 결정하는 핵심 요인으로 인식되고 있다.

한편, 정보시스템은 기존에 개별 업무만을 처리하던 것에서 업무 프로세스를 통제하고 제어할 수 있는 보다 진보된 기능을 제공하는 방향으로 발전하고 있다. 그런데 전술한 변화는 프로세스를 관리할 수 있는 기술과 도구의 필요성을 더욱 증가시키고 있다. 워크플로우 관리 시스템은 복잡한 업무 프로세스를 정의, 관리, 실행하는 도구(Hollingsworth, 1994)로서 이러한 문제에 대한 새로운 해결책을 제시한다. 최근에는 기존의 정보시스템을 이 워크플로우 시스템과 연결하고자 하는 시도가 많아지고, 특히 워크플로우와 데이터베이스를 결합하여 대량의 워크플로우 정보를 효과적으로 관리하는 것에 관심이 집중되고 있다.

본 논문에서는 능동형 데이터베이스(active database)를 이용하여 프로세스 설계(design)와 실행(enactment)을 결합하는 방법론을 제안한다. 기존의 워크플로우 관리 시스템에서는 프로세스를 설계하고, 이를 워크플로우 관리 시스템의 엔진이 이해할 수 있는 형태로 번역하여 데이터베이스에 저장한 다음, 엔진은 이 저장된 정보를 토대로 실제 프로세스 진행을 통제한다. 이런 개념을 채택한 시스템에서 프로세스 실행의 가장 중요한 역할을 담당하는 부분은 워크플로우 관리 시스템의 엔진이다. 그런데 능동형 데이터베이스는 ECA 규칙을 처리하는 기능을 내장하고 있으므로, 이를 이용하면 기존 시스템의 핵심인

† 본 연구는 과학재단 특정기초연구비(97-02-00-09-01-3)에 의하여 지원되었으며, 지원에 감사드립니다.

엔진을 대체할 수 있다. 즉, 데이터베이스에 저장된 프로세스 정보에서 ECA 규칙을 직접 유도하고, 이를 데이터베이스 내에서 바로 실행한다는 것이다. 이는 엔진은 없이 프로세스 모델과 데이터베이스만으로 프로세스 통제를 가능하게 하므로 프로세스 기반 응용 프로그램을 보다 쉽게 구현하고 사용할 수 있게 한다. 이를 가능하게 하기 위하여 본 연구에서는 프로세스 구성 요소의 행동 양식을 명확히 규정할 수 있으면서 의미를 가지는 최소한의 단위인 블록 개념과 그 유형을 제안하였다. 복잡한 프로세스를 중첩적(nesting)으로 모듈화하고 설계할 수 있고, 이를 처리하기 위해 일반적인 프로세스 모델에서 블록을 추출하여 중첩 프로세스 모델로 재구성하는 알고리듬을 개발하였다. 블록의 유형별로 대응되는 능동형 데이터베이스의 ECA 규칙을 정의하고, 블록으로 재구성된 중첩 프로세스 모델을 ECA 규칙으로 변환하여 데이터베이스가 직접 실행할 수 있게 한다.

논문의 구성은 다음과 같다. 2절에서는 프로세스 모델링과 관련된 기존 연구를 살펴보았으며, 3절에서는 프로세스 모델의 특징과 블록의 유형을 정리하였다. 4절은 일반적인 프로세스 정의에서 블록을 자동으로 찾아내고, 이를 중첩 모델로 변환하는 알고리듬을 설명한다. 5절은 블록 정의에서 ECA 규칙을 유도하여 능동형 데이터베이스가 자동으로 흐름을 통제할 수 있도록 하는 방법을 설명하였다. 6절은 제안한 방법론의 작동 예를 설명하였고, 마지막 절은 결론이다.

2. 기존 연구

워크플로우는 "컴퓨터에서 자동으로 실행할 수 있도록 표현된 비즈니스 프로세스"로 정의되며, 워크플로우 관리 시스템은 "이 워크플로우를 완전하게 정의하고, 정의된 워크플로우의 실행과 순서를 통제하며, 프로세스 전체를 관리하는 소프트웨어 시스템"을 말한다(Hollingsworth, 1994). 최근 10년 동안 프로세스 모델링 및 표현 언어 개발(Dogac *et al.*, 1997), 프로세스 분석과 검증 방법(Al-Ahmari *et al.*, 1999), 웹(Web) 또는 에이전트 기반 분산 워크플로우(Nadoli *et al.*, 1993), 프로세스의 동적 변화 지원(Kumar *et al.*, 1999), 실시간 트랜잭션 처리(Georgakopoulos *et al.*, 1996) 등 여러 선행 연구가 있었다.

이 가운데에서 프로세스 모델링 방법을 다룬 연구가 본 연구와 가장 관련이 많으므로 이 분야의 연구 내용을 세분하면 다음과 같다.

- 프로세스 정의 언어 : 스크립트(script) 형식을 빌어 프로세스의 통제와 데이터의 흐름을 정의하는 프로세스 표현 언어를 말한다. 이것은 컴퓨터 프로그래밍 언어와 유사하지만, 사용자가 이해하기 어렵고 모델의 타당성을 검증하기 위한 형식적인 기초가 부족한 것이 단점이다.
- 네트워크 표현법 : 플로우차트나 페트리네트(Petri-net)(van der Aalst, 1998)같이 그래프를 이용하여 프로세스를 표현하는 전통적인 방법이다. 이 방법은 사용자가 프로세스 모델의 의미를 쉽게 이해할 수 있다는 점 때문에 많이 사용되지만 모델의 타당성을 검증할 형식적인 기초를 충분히 제공하지 못하는 것이 단점이다.
- 논리 이론 기반 방법론 : Temporal logic, CTL(Computation Tree Logic)(Attie *et al.*, 1996)과 같은 논리 이론을 이용하여 프로세스를 정의하는 방법이다. 이 방법은 모델 검증의 이론적인 바탕을 가지고 있지만 프로세스 모델에서 실행 알고리듬의 도출이 어렵다.
- ECA 규칙 활용 : ECA 규칙을 이용하여 프로세스를 정의하고 이를 바탕으로 프로세스의 흐름을 자동으로 통제할 수 있는 방법이다. 그러나 규칙의 의미를 시각적으로 나타내기가 어렵기 때문에 사용자가 쉽게 이해하거나 직접 관리할 수 있는 표현이 아니다. 또 규칙 자체를 생성하는 것이 매우 어렵다.

본 논문에서는 위 방법 중 네트워크 기반 표현법과 ECA 규칙을 결합한다. 즉, 사용자는 사용이 가장 편리한 전자의 방법으로 프로세스를 모델링하고, 시스템은 실행 자동화가 가능한 후자를 이용하여 프로세스 흐름을 통제한다. 이는 네트워크를 이용한 프로세스 표현을 ECA 규칙으로 변환하는 것이 필요함을 의미한다.

워크플로우 분야에서 능동 규칙을 적용한 기존 연구가 몇몇 존재한다. Dayal(1990)이 가장 먼저 워크플로우 시스템에 능동 규칙의 적용을 시도하였다. Casati(1996)는 워크플로우에서 필요한 개념적 수준의 다양한 규칙을 유도하고, 이들을 7가지 그룹으로 나누었다. 또한 Geppert(1998)는 워크플로우 수행중에 발생하는 모든 사건을 논리 이론에 바탕을 둔 사건 이력(event history)에 기록하여, 이 정보를 이용하여 워크플로우를 실행하고자 시도하였다. 이것을 위하여 규칙으로 구성된 중재자(broker)라고 하는 일종의 워크플로우 엔진을 구현하였다. 이런 기존 연구는 순수하게 ECA 규칙만을 이용하여 프로세스를 모델링하는 시도이다. 그러나 ECA 규칙으로 프로세스를 표현하는 것은 전술한 바와 같이 쉬운 일이 아니므로, 이를 일반적으로 적용하려면 프로세스를 먼저 단순화하는 것이 필요하다. 기존의 연구에서는 프로세스 모델을 단순화하는 일반적인 방법을 제시하지 못하여 ECA 규칙으로 프로세스를 정형화하는 데는 한계가 있었다. 본 연구는 블록 개념을 이용하여 프로세스 표현을 간단한 모델로 변환하고 이로부터 자동으로 ECA 규칙을 유도한다는 점이 기존 연구와 가장 큰 차이점이다.

한편, 능동형 데이터베이스 시스템(Paton, 1998)은 사용자나 응용 시스템의 외부 간섭이 없이 특정한 상태를 인식하고, 반응할 수 있다. 능동적인 기능은 일반적으로 규칙으로 표현되는데, 전문가 시스템(expert system)에서 생성규칙(production rule)이라고 일컬어지는 사건-조건-행동(ECA) 모형으로 기술된다. 여기서, 사건이란 데이터베이스 내용의 변경을 의미하고,

조건이란 데이터베이스의 질의에 해당하며, 행동은 데이터베이스의 다른 변경을 유발하는 명령어 집합이다. 능동형 데이터베이스 시스템은 사건(event)을 감지할 수 있고 조건(condition)을 검사할 수 있는 능력을 가지고 있다. 만약 하나의 사건이 발생하고 조건이 참이면 거기에 해당하는 행동(action)을 실행한다.

3. 프로세스 모델과 블록

프로세스는 특정 목적을 수행하기 위해서 모인 태스크(활동)의 집합과 이 태스크간의 관계로 정의할 수 있고, 프로세스 모델은 이 프로세스를 이해, 실행, 분석, 개선하기 위해 태스크와 이들의 관계를 명확하게 표현한 것이라 정의할 수 있다. 이 절에서는 일반적인 프로세스 모델의 특징과 본 연구에서 제시하는 블록 개념을 설명한다.

3.1 프로세스 모델

프로세스 모델은 어느 태스크가, 어떤 순서로, 누구에 의해, 어떤 일을 수행해야 하는지를 정의하고, 또 태스크 수행을 위한 입력 조건과 수행 결과에 대한 정보를 가지고 있다. 프로세스 모델에는 그 구조적 측면과 함께 자원, 태스크 내용, 수행 시간 등의 정보가 포함되는데, 본 논문의 주 대상이 되는 구조적 요소를 중심으로 살펴보면 다음과 같은 특징이 있다(Schlenoff *et al.*,1996). 프로세스는 그 구성 요소로 여러 개의 태스크를 가진다. 각 태스크는 프로세스의 목표를 달성하기 위한 단위기능을 수행한다. 또한, 프로세스가 처리하는 일의 범위가 명확히 정의되어 있다. 이는 프로세스는 반드시 시작 태스크와 종료 태스크를 가지며, 각 태스크의 입력과 출력이 명확히 정의되어야 한다는 의미도 포함한다. 태스크와 태스크 사이에는 이들의 시간적인 실행 우선순위를 나타내는 선후 관계가 존재할 수 있다. 인접한 선후 관계를 가지는 태스크 사이에는 여러 가지 분기와 병합 조건을 나타내는 상관 관계(inter-task dependency)가 존재할 수 있다. 프로세스와 태스크는 프로세스가 진행되면서 발생하는 사건에 따라 그 상태가 변화하는 이산현상 시스템으로 프로세스 모델은 이런 상태 변화를 암시적(implicit)으로 표현한다.

일반적으로 프로세스 구조는 그래프 형식을 이용하여 표현한다. 이를 프로세스 정의 그래프라 한다. 대부분의 기존 워크플로우 관리 시스템은 일원적인 프로세스 설계 방식을 채택하고 있는데, 이 방식은 프로세스를 처음부터 끝까지 하나의 네트워크로 표현하는 것을 말한다.

본 연구에서는 중첩 프로세스 개념을 이용하여 워크플로우를 모델링한다. 중첩 프로세스 개념은 프로세스 모델링의 편의성과 모델의 확장성에 강력한 이점을 제공하는데, 이에 대한 설명과 구체적 논의는 본 논문 저자의 선행 연구인 (Kim *et al.*, 2000)을 참조할 수 있다. 본 논문에서는 중첩 프로세스 개념을 구현하는 방법으로 블록을 이용하는 것을 제안한다. 블록은 이 논문의 핵심 개념으로 다음 절에서 더 자세히 알아보기로 한다.

3.2 블 록

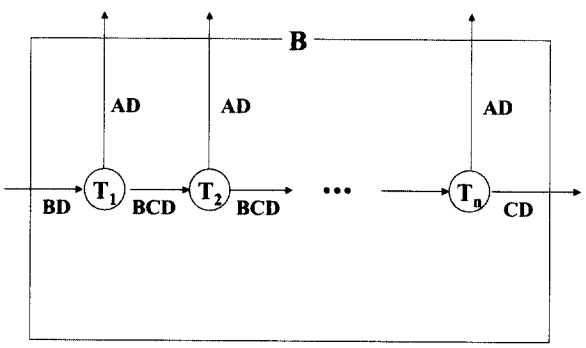
블록은 워크플로우 구성 요소들의 행동 양식을 명확히 규정할 수 있으면서 의미를 가지는 최소한의 단위로 정의한다. 이는 복잡한 프로세스를 간단하게 표현할 수 있는 일종의 템플리트(패턴) 역할을 한다. 워크플로우에서 블록이라는 용어는 Dogac *et al.*(1997)의 분산환경에서 워크플로우 스케줄링을 구현한 연구에서 프로세스 흐름을 구분하기 위해 처음 사용하였다. 본 연구에서는 ECA 규칙을 유도하기 위해 블록 개념을 이용한다.

한편, 본 연구에서는 블록을 정의하기 위해 ACTA(actions의 라틴어) 형식론(Casati *et al.*, 1996; Chrysanthis *et al.*, 1990)을 사용한다. ACTA 형식론은 원래 데이터베이스에서 트랜잭션간의 상관관계를 정의하는 것으로, 부록 1과 같이 미리 정의된 7개의 의존관계를 이용한다. 본 연구에서는 이 ACTA 형식론을 차용하여 태스크간의 상관관계를 표현한다.

프로세스 모델을 살펴보면 직렬, 병렬, 반복의 세 가지 유형의 패턴이 반복적으로 나타나면서 프로세스를 구성하는 것을 알 수 있다. 부록 2에 블록의 유형들을 ACTA 의존관계를 이용해 정의하였다. 여기서는 몇 가지 블록의 정의를 그림으로 알기 쉽게 설명하였다.

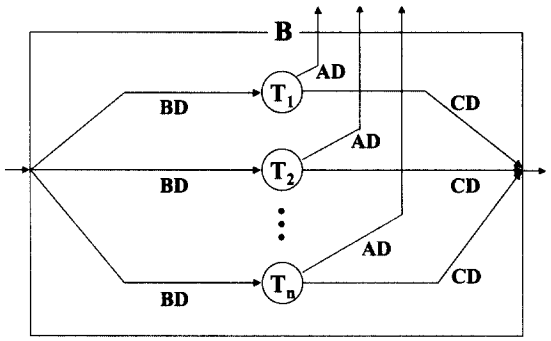
먼저, 직렬블록은 업무 흐름의 분기나 병합이 없이 태스크가 일렬로 연속되는 경우이다. <그림 1>은 직렬블록을 나타낸 것으로 블록 B의 시작은 첫 번째 태스크 T_1이 시작하고, T_1의 완결은 다시 T_2를 시작한다. 이런 식으로 계속 진행되어, 마지막 태스크인 T_n의 완결은 블록 B의 실행을 완결짓는다. 만약 여러 태스크 중에 하나가 중단되면, 전체 블록이 중단된다.

병렬블록은 업무 흐름에 분기와 병합이 있는 것으로 다시 다음과 같이 네 가지로 나누어진다.



Serial Block : No split and No merge

그림 1. 직렬블록.

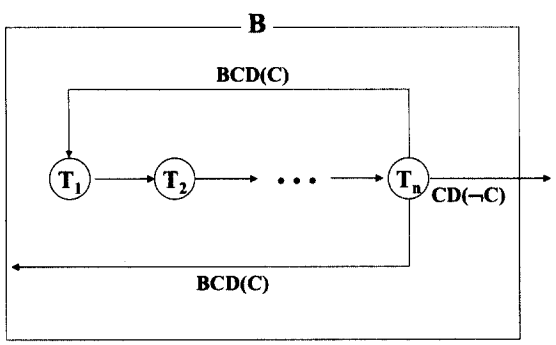


AND-parallel Block

그림 2. AND-병렬블록.

AND-병렬블록은 <그림 2>와 같이 병렬 실행에 참여하는 태스크가 모두 동시에 실행되는 것을 의미한다. 즉, 블록 B의 시작이 블록 안에 있는 모든 태스크 T_i (i = 1, 2, ···, n)를 병렬적으로 시작시킨다. 그리고 이 태스크들이 모두 완결되어야 B도 완결된다. 만약 하나의 태스크라도 중단되면, 전체 블록이 중단된다. 이 외에도 병렬블록에는 OR-병렬블록, ALT-병렬블록, CON-병렬블록이 있으며, 모든 병렬블록 유형은 구조적으로 동일하다. 따라서 각 분기 유형에 대한 명세를 네트워크 구조에 추가하여 정의하거나 특수한 기호를 이용해야 한다. 마지막 세 병렬블록에 대한 상세한 의미와 그림은 Bae, J. S.(2000)을 참조할 수 있다.

마지막으로 반복블록은 조건에 따라 분기하는 CON-병렬블록과 유사하나, 독특한 특징을 가지고 있다. 즉, <그림 3>과 같이 업무 흐름이 진행되는 도중에 특정 태스크에서 특정 조건이 만족되는 사건이 발생하면 이미 진행했던 태스크로 흐름이 되돌아가 분기의 결과가 유방향 사이클을 생성한다. 따라서 이 반복 여부를 검사하는 T_n과 반복이 시작되는 T_1만이 반복블록과 의존관계를 가진다. T_1과 T_n 사이에 있는 태스크들은 반복블록 이외의 블록(직렬 혹은 병렬)에 의해 제어를 받으므로 반복블록과는 의존관계를 가지지 않는다.



Iterative Block

그림 3. 반복블록.

4. 프로세스 모델의 변환

본 장에서는 블록을 이용하여 일반적인 프로세스 정의 그래프를 중첩 프로세스 모델로 자동으로 변환하는 방법을 설명하였다. 이것은 사용자는 블록을 인식하지 않고 시스템이 자동으로 블록을 처리하기 위해 필요한 것이다.

4.1 프로세스 구조의 전처리

본 논문에서 제안하는 방법을 사용하려면 주어진 프로세스 정의 그래프가 다음의 두 전제 조건을 만족해야 한다.

- 분기와 병합은 반드시 서로 쌍을 이루어야 한다.
- 분기와 병합의 쌍이 서로 교차할 수 없다.

위의 두 조건은 프로세스에서 블록을 형성하기 위한 기본 전제 조건으로 그래프가 가져야 할 성질을 의미한다. 이러한 성질은 블록을 이용하여 프로세스 중첩 모델로 자동 변환하기 위해서 필요한 제약 조건이다. 그러나 일반적으로 사용자가 정의한 프로세스 모델에서는 이런 조건을 만족하지 않는 경우가 자주 있는데, 본 연구에서는 프로세스 모델의 의미에 변화를 수반하지 않고서 사용자가 정의한 프로세스 모델이 위 조건을 모두 만족하도록 변환할 수 있는 메커니즘을 제안하였다. 따라서, 위의 전제 조건은 본 연구의 프로세스 자동 실행 방법론에 있어 처리 가능한 프로세스 모델에 대한 제약을 가하는 것이 아니라, 블록 이용 중첩 모델 생성의 편의를 위해 수행하는 사전 처리 과정을 의미한다고 할 수 있다.

첫 번째 조건은 프로세스에서 분기가 이루어지면, 반드시 이에 대응되는 병합이 있어야 하고, 그 역도 성립해야 함을 의미한다. 병합(분기)이 서로 다른 노드에서 발생하면서 이에 대응되는 분기(병합)는 같은 노드에 겹칠 경우가 있는데 이는 위의 조건을 만족한 것으로 본다. 또, 이 조건을 만족하는 프로세스는 반드시 하나의 태스크에서 시작하고 하나의 태스크에서 끝나야 한다. 시작 또는 종료 태스크가 여러 개 있는 프로세스 모델은 각각 직전과 직후에 실제로는 아무런 일도 하지 않는 가공의 노드(dummy node)를 삽입하는 전처리를 함으로써 이 조건을 만족하도록 할 수 있다.

두 번째 조건은 프로세스가 벌집구조(honeycomb)를 포함하지 않아야 한다는 것이다. 벌집구조는 <그림 4>와 같은 구조를 가지는 프로세스를 말한다. 예를 들어 (a)가 표현하는 프로세스를 살펴보면, 노드 1, 2, 3이 하나의 분기-병합을 이루고 있고, 노드 2, 3, 4는 또 다른 하나의 분기-병합을 구성하고 있다. 이때 노드 2와 3은 두 개의 서로 다른 분기-병합의 쌍에 동시에 속하는 노드가 되는데, 이를 교차노드라고 한다. 이런 교차노드가 형성하는 경로를 교차경로라고 하고, 교차경로를 가지는 프로세스 구조를 벌집구조라고 한다. 벌집구조를 제거하기 위해 아래와 같은 순서로 전처리를 수행한다.

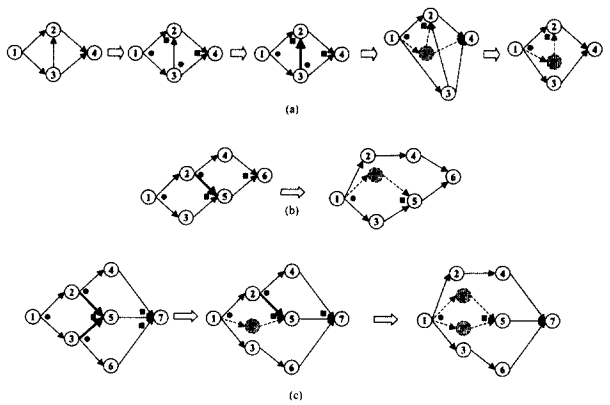


그림 4. 벌집구조 전처리 예.

1. 프로세스 정의 그래프에서 분기와 병합이 일어나는 곳을 탐색한다. 분기가 일어나는 노드에 ● 를 표시하고, 병합이 일어나는 노드에는 ■ 를 표시한다.
2. 앞에서 탐색한 분기와 병합의 쌍을 모두 조사하여 두 개의 서로 다른 분기-병합의 쌍에 속하는 교차노드와 이러한 교차노드로 구성된 교차경로를 찾는다.
3. 교차경로의 시작 노드 N_i를 복사하여 가공 노드(dummy node) N_i'을 생성하고, N_i'의 입력 및 출력 아크도 N_i와 동일하게 복사한다.
4. N_i의 출력 아크 가운데 단계 2에서 찾은 교차경로 상에 있는 아크를 지우고, N_i'의 출력 아크에 대해서는 2에서 찾은 교차경로 상에 있지 않는 아크를 지운다.

예를 들어, (a)에서 노드 3과 2가 형성하는 교차경로를 생각해 보자. 위 알고리듬에 의해 먼저 분기와 병합이 일어나는 곳을 찾아서 표시하고, 교차노드를 탐색하여 노드 3과 2가 형성하는 교차경로를 찾는다. 다음 단계는 교차경로의 시작 노드 3을 복사하여 가공 노드 3′을 새로이 생성하고, 노드 3과 같은 방식으로 가공 노드 3′의 아크를 생성한다. 새로 생성한 가공 노드와 아크는 점선으로 표시하였다. 마지막 단계에서 시작 노드 3의 출력 아크 중 교차경로 상에 있는 아크(3->2)를 제거하고, 가공 노드 3′은 교차경로 상에 있지 않은 출력 아크(3′->4)를 제거한다. 이제 이 구조는 두 번째 조건을 만족한다. 나머지 (b)와 (c)는 보다 복잡한 벌집구조이지만 같은 전처리 방법으로 해결될 수 있다.

벌집 구조를 제거하는 전처리 과정은 워크플로우에서 다루는 프로세스가 분기되어 흐르는 경우 서로간의 간섭 없이 독립적으로 진행할 수 있도록 함으로써 통제를 단순화하는 의미가 있다.

이같이 전처리에 의해 변환된 프로세스 모델을 이용하여 프로세스를 실행하는 것을 생각해보자. 가공 노드에 해당되는 태스크는 실제로는 존재하지 않는다. 그러므로 워크플로우를 실행할 때 가공 노드를 만나면 워크플로우 엔진은 바로 이 가공 노드를 복사한 원래 노드를 실행한다. 또, 어떤 노드가 복사되어 가공 노드가 생성된 경우 원래 노드의 상태가 변하면 즉시 변화된 상태를 가공 노드에 적용하여 이 두 노드는 항상 같은 상태가 되도록 한다.

여기서 중요한 점은 전처리가 프로세스 구조에 의미상의 변화를 초래하지 않아야 한다. 즉, 원래 그래프에서 발생 가능한 모든 경로와 순서가 변환된 그래프에서도 그대로 유지되어야 한다. 앞서 제안한 전처리 과정은 어떠한 경우에도 이를 만족함을 쉽게 알 수 있다.

4.2 블록 탐색 알고리듬

블록 탐색 알고리듬은 프로세스 모델에서 블록을 찾는 것으로, 네트워크 탐색 알고리듬(Ahuja *et al.*, 1993)에 블록을 판별하는 기준을 추가하여 이를 반복 적용한 것이다. 이 수정된 알고리듬은 <그림 5>와 같이 진행된다.

가장 먼저 반복블록부터 탐색한다. 반복블록은 유방향 사이클을 이루므로, 노드의 위상적 순서(topological ordering)를 생성하는 알고리듬을 이용하여 사이클을 찾는다. 이 알고리듬은 사이클을 포함하는 그래프에서 반복의 시작노드와 종료노드를 찾는 것이다. 이 두 노드가 형성하는 경로를 반복블록으로 등록한 다음, 종료노드에서 시작노드로 이어지는 아크를 그래프에서 제거하여 하나의 사이클을 없앤다. 이 과정을 더 이상 사이클을 찾을 수 없을 때까지 반복한다. 반복블록이 모두 제거되면 4.1절의 전처리 과정을 수행한다.

이제 너비우선탐색(breadth first search algorithm)을 변형한 네트워크 탐색 알고리듬을 이용하여 직렬블록과 병렬블록을 교대로 탐색하는 과정을 반복한다. 먼저 직렬블록은 연속적인 두 태스크의 각 입력노드(indegree)와 출력노드의 수(outdegree)가 각각 1인 노드들로 구성되므로 쉽게 탐색할 수 있다. 직렬로 연결된 노드를 탐색하여 하나의 직렬블록으로 압축하면, 병렬블록이 새로이 나타난다.

병렬블록 탐색은 탐색을 진행하면서 탐색을 실시한 노드에 탐색이 이루어졌음을 나타내는 마킹을 해둔다. 현재 탐색하고

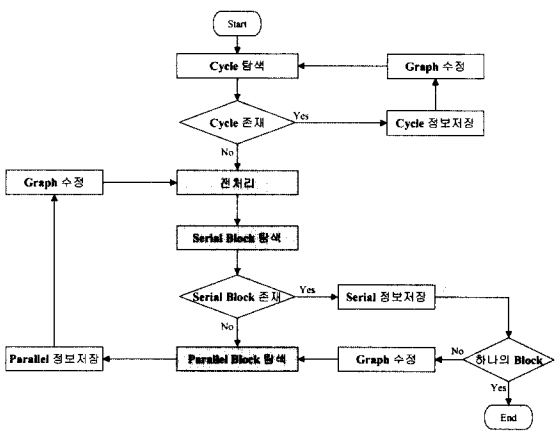


그림 5. 블록 탐색 알고리듬.

있는 노드에 대해서 그 직후행노드를 검사하여 이미 마킹이 되어 있으면, 이는 병렬블록이 하나 존재함을 나타낸다. 이것은 전단계에서 직렬로 연결된 태스크들을 탐색하여 하나의 블록으로 만들었고, 알고리듬이 너비우선탐색을 하기 때문이다. 현재 노드의 직선행 및 직후행 노드 사이에 있는 모든 노드로 하나의 병렬블록을 형성하면 새로이 직렬블록이 생성된다. 여기서 찾은 병렬블록은 현재 탐색하고 있는 프로세스가 가진 여러 분기-병합 쌍 가운데 가장 내부에 존재하는 것이다. 그리고 이 알고리듬은 그래프의 위상(topology)만을 이용하므로 병렬블록의 종류를 세부적으로 구분하려면 프로세스 모델의 분기 조건을 참고해야 한다.

위의 두 과정을 적용할 때마다 그래프는 점점 축소되어 결국 가장 상위의 하나의 블록으로 묶이게 되고 알고리듬은 종료된다. 블록 탐색 알고리듬은 그래프의 아크 수가 m일 때, 그 복잡도(complexity)가 $O(m^2)$인 매우 효율적인 알고리듬이다 (Bae, J. S., 2000).

<그림 6>은 블록 탐색 알고리듬의 적용 예제를 보여주고 있다. 먼저, (a)에서 반복블록을 탐색하여 이를 등록하고 그래프에서 사이클을 제거한다. 이제 (b)와 같이 직렬블록을 탐색하여 B_1 블록으로 묶고((c) 그래프), 병렬블록을 탐색하는 과정을 수행한다. (c)의 그래프에 대해 너비우선탐색을 실시하면서 탐색한 노드를 차례대로 마킹해 나간다. 태스크 T_6를 탐색할 때 직후행노드인 T_9가 이미 마킹되어 있으므로, T_6의 직선행노드 T_3과 직후행노드 T_9 사이에 있는 모든 노드(T_5, T_6, T_7)는 병렬블록을 이룸을 알 수 있다. 이 병렬블록을 B_2 블록으로 묶은 다음 같은 방법으로 알고리듬을 계속 적용하면, 결국 전체 프로세스는 (g)에서 보는 바와 같이 하나의 블록 B_5로 묶인다.

4.3 중첩 프로세스 구성

4.2절의 블록 탐색 알고리듬의 결과를 역으로 추적하면 블록 트리 구조를 생성할 수 있다. 예를 들어 <그림 6>의 결과는 <그림 7>의 (a)와 같은 트리로 표현된다. 일반적으로 트리는 이를

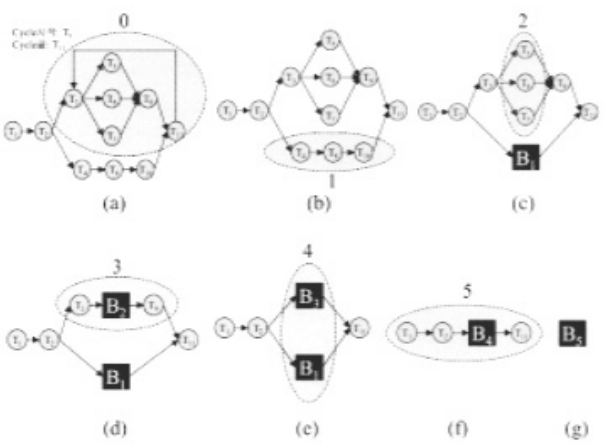


그림 6. 블록 탐색 알고리듬 적용 예.

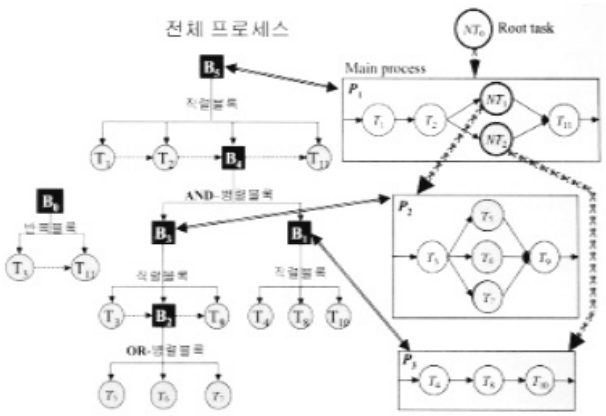


그림 7. 블록 트리 구조.

구성하는 노드간의 부모와 자식 관계로 설명된다. <그림 7>의 (a)에서 가장 상위 수준 블록 B_5는 4개의 자식 노드(T_1, T_2, B_4, T_{11})를 가지고 있다. 블록 B_4는 다시 블록 B_1과 B_3를 자식 노드로 가지고 있다. 이는 결국 Kim *et al.*(2000)이 제시한 중첩 프로세스 모델과 동일한 구조를 가지는 것으로, 일반적인 프로세스 모델에 블록 탐색을 적용하여 중첩 프로세스 모델로 변환할 수 있음을 보여 준다.

중첩 모델은 프로세스를 간단한 트리 형태로 표현하기 때문에 트리를 구성하는 노드간의 실행 순서를 자동으로 제어하는 것이 쉽다는 장점이 있다. 이와 더불어 본 논문에서는 중첩 프로세스 모델이 앞에서 정의한 블록 단위로 구성된다. 즉, 트리를 구성하는 모든 노드는 블록이거나 또는 더 이상 분해할 수 없는 단위 태스크라는 것이다. 따라서 각 블록의 유형에 대해 그 흐름을 자동으로 통제할 수 있는 방법이 있다면 전체 프로세스 또한 자동으로 통제할 수 있을 것이다. 이에 대해서는 다음 절에서 상세히 논의한다.

5. ECA 규칙의 유도

이 절에서는 앞에서 설명한 블록의 정의로부터 ECA 규칙을 유도하여, 이 규칙을 능동형 데이터베이스가 자동으로 처리하게 하는 방법을 설명한다.

5.1 규칙의 유도

본 논문에서 제시하는 워크플로우 시스템의 구조는 <그림 8>과 같다. 이는 크게 두 부분으로 나누어지는데, 하나는 워크플로우 엔진에 해당하는 능동형 데이터베이스이고, 다른 하나는 데이터베이스 바깥에 존재하는 응용 프로그램인 클라이언트가 있다. 능동형 데이터베이스는 데이터베이스의 능동적 기능을 이용하여 프로세스 실행중 발생하는 사건을 감지하고

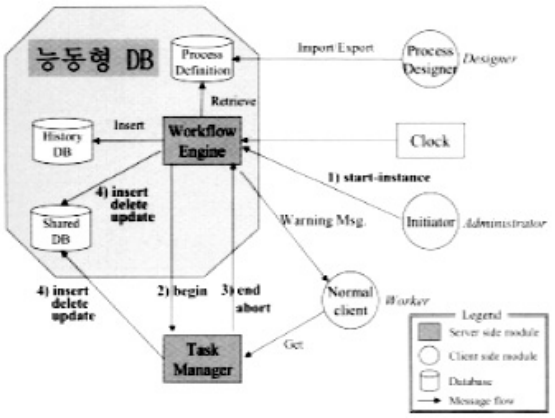


그림 8. 워크플로우 시스템의 구조.

이를 처리하는 워크플로우 시스템 엔진의 역할을 한다. 이를 위해서 능동형 데이터베이스가 감지하고 처리해야 하는 사건이 미리 정의되어 있어야 한다. 본 연구에서는 앞에서 설명한 블록의 개념을 활용하여 필요한 사건을 정의하고, 이것을 바탕으로 ECA 규칙을 유도한다.

능동형 데이터베이스는 감지한 사건에 대해 적절한 행동을 취하는데, 이 때 관련된 태스크의 상태가 바뀌게 된다. 따라서 우선 태스크의 상태 전이 모델을 이해할 필요가 있다. 본 연구에서는 <그림 9>에서 보는 것처럼 Rusinkiewicz *et al.*(1995)의 연구에서와 유사한 태스크 상태 전이 다이어그램을 사용한다.

워크플로우에서의 태스크 상태 변화는 발생한 사건에 의해 결정된다. 사건은 능동형 데이터베이스를 기준으로 그 발생 위치에 따라 외부사건과 내부사건 두 가지로 구분된다.

외부사건은 데이터베이스 외부에서 입력되는 사건이나 또는 외부로 전달되는 사건을 말한다. 외부사건의 종류는 다음과 같은 것이 있다.

- start-instance (인스턴스-시작) : 클라이언트가 프로세스를 처음 시작하는 사건이다. 능동형 데이터베이스가 이 사건을 감지하면 프로세스의 첫 번째 태스크를 "Ready" 상태로 만든다.

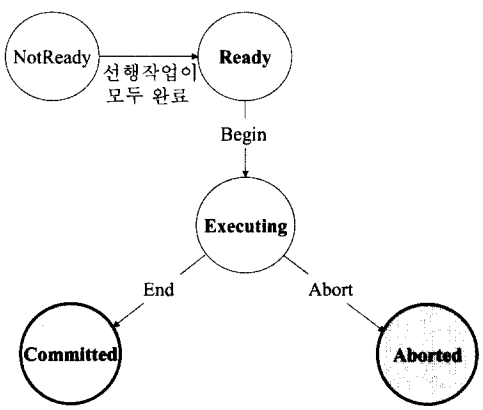


그림 9. 태스크 상태 전이 다이어그램(Rusinkiewicz *et al.*, 1995).

- begin (시작) : 능동형 데이터베이스가 상태가 "Ready"인 태스크를 해당 작업자에게 할당하는 사건이다. 데이터베이스 내부에서는 할당된 태스크의 상태를 "Executing"으로 만들어 준다.
- end(종료) : 클라이언트 사용자가 할당된 작업을 완료했음을 알리는 사건이다. 능동형 데이터베이스는 해당 태스크의 상태를 "Committed"로 바꾼다.
- abort (중단) : 클라이언트 사용자가 할당된 작업을 완료하지 못하고 중단된 것을 나타내는 사건이다. 능동형 데이터베이스는 그 태스크의 상태를 "Aborted"로 바꾼다.

각 사건에 대해 능동형 데이터베이스는 규칙을 이용하여 해당되는 반응을 실행한다.

두 번째는 내부사건으로 데이터베이스 내부에서 발생한다. 전술한 외부사건이 발생하면 외부사건을 관리하는 데이터베이스 테이블에 레코드 하나가 추가되고, 이것을 바탕으로 테이블에 결부되어 있는 규칙이 실행된다. 즉, 외부사건이 데이터베이스 내부 사건으로 변환되어 프로세스 진행이 통제되는 것이다. 내부사건은 ACTA 형식론으로 표현된 블록의 정의대로 프로세스를 진행한다. 본 논문에서는 블록의 정의를 ECA 규칙으로 변환함으로써 블록의 실행을 자동화하고자 한다. 그러나 부록2의 ACTA 형식론으로 표현된 블록의 정의를 살펴보면 블록의 종류에 따라 의존관계의 정의가 다르므로 종류별로 그에 대응되는 규칙을 생성해서 적용해야 한다.

한편, 관계형 DBMS에서는 트리거(trigger)를 이용하여 ECA 규칙을 구현한다. 트리거란 데이터베이스에 저장되어 있는 일종의 프로그램으로서, 어떤 사건이 일어나는 경우에 그 사건을 감지하여 자동적으로 실행되는 프로시저를 말한다. 여기서 트리거가 감지하는 사건은 데이터베이스 테이블에 데이터를 삽입, 갱신, 삭제하는 것이다. 트리거의 구조는 ECA 규칙의 사건, 조건, 행위에 해당하는 3가지로 구성되어 있다. 즉, 첫 번째는 트리거를 실행시키는 사건을 정의하는 "트리거시키는 명령문"이고, 두 번째는 트리거를 실행하기 위해서 만족되어야 하는 조건(논리)을 정의하는 "트리거 조건"이다. 마지막은 트리거시키는 명령문이 발생되고 트리거 조건이 참(True)으로 평가된 경우, 실행될 SQL 명령문과 PL(Procedural Language)/SQL 코드를 포함하는 프로시저(PL/SQL 블록)를 정의하는 "트리거된 명령문"이다.

부록 3은 각 블록 종류별로 유도한 규칙들과 이들을 저장하는 테이블을 정리한 것이다. 규칙 처리와 관련된 테이블은 WF_EVENT, WF_BLOCKINST, WF_TASKINST의 세 개이다. WF_EVENT는 클라이언트가 발생시키는 외부사건 정보를 저장하는 테이블이고, WF_BLOCKINST는 진행중인 블록의 상태 정보를 저장하며, WF_TASKINST는 각 태스크의 정보 및 상태를 저장하고 있다. 이 세 테이블은 각각 자신이 "어떤 경우에 어떤 규칙을 실행한다"라는 정의를 가지고 있다. 먼저 모든 블록에서 공통으로 필요한 6개의 규칙이 있다. 이들 가운데 외부

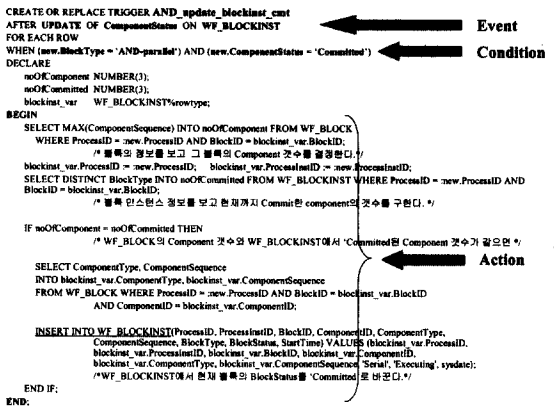


그림 10. ECA 규칙 구현 예제.

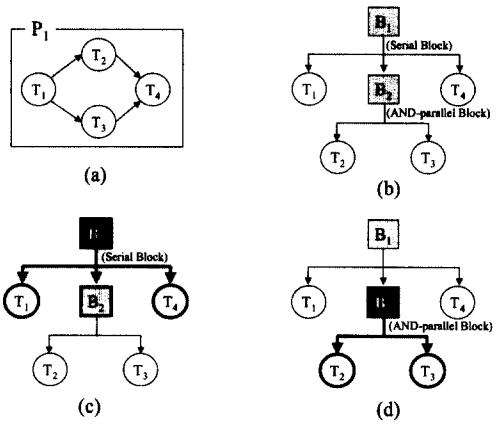


그림 11. 규칙 적용 예제 프로세스.

사건을 처리하기 위한 규칙(R1, R3, R5)은 WF_EVENT 테이블에 저장되어 있고, 외부로 작업 내용을 전달하는 규칙(R2)과 태스크의 상태 변화를 블록에 전달하는 규칙(R4, R6)은 WF_TASKINST 테이블에 저장되어 있다. 다른 규칙(R7~R25)은 모두 블록 종류별로 의존관계에 대응되어 유도한 것이다. 이 규칙들은 모두 블록 내에서 발생하는 사건을 처리하기 위한 것이므로 WF_BLOCKINST 테이블에 정의되어 있다.

예를 들어 R11: AND_update_blockinst_cmt의 구현에 대해서 자세히 살펴보기로 하자(<그림 10> 참조). 규칙 R11은 AND-병렬블록에서 하나의 구성 태스크가 완료되었을 때 실행하는 규칙이다. 이 규칙은 AND-병렬블록의 구성 태스크와 블록 사이의 CD 의존관계에 대응되는 것이다. <그림 8>에서 능동형 데이터베이스 외부에 있는 일반 클라이언트가 할당된 태스크를 완료하면, 외부사건 end가 발생한다. 규칙 R11은 이 사건이 변환된 내부 사건에 대해 해당 블록의 상태정보를 변경하기 위하여 실행된다. 먼저 트리거 문법에 의해 WF_BLOCKINST 테이블에 있는 하나의 레코드 정보(ComponentStatus)가 갱신(UPDATE)되는 것이 "사건"이 된다. 능동형 데이터베이스는 이 사건을 감지하여 조건을 검증한다. 이 경우에는 새로이 변경되는 블록의 종류가 AND-병렬블록이면서 구성요소의 상태가 'Committed'로 바뀌는 것이 "조건"이며, CON-병렬블록에서 사용하는 것과 같은 조건은 없다. 마지막으로 사건이 발생하여 조건이 만족된 경우에 수행해야 되는 명령문들을 모아 놓은 "행위"가 있다. 이곳에서는 WF_BLOCK(프로세스 정의에 사용된 블록의 정의를 저장하는 테이블)에서 블록 구성요소의 개수와 WF_BLOCKINST에서 현재 블록 인스턴스의 'Committed'된 구성요소의 개수가 같으면 그 블록의 상태(BlockStatus)를 'Committed'로 바꾼다. 다른 규칙들도 이와 유사하게 정의된다 (Bae, J.S., 2000).

6. 작동 원리 및 예제

위에서 생성한 ECA 규칙이 워크플로우 시스템의 실행에 적용되는 방식을 <그림 11>의 예를 이용하여 설명하였다.

(a)와 같이 4개의 태스크로 구성된 단순한 프로세스를 생각해 보자. 먼저 블록 탐색 알고리듬은 (b)와 같은 블록 트리를 생성한다. 여기서 B_1은 직렬블록이고, B_2는 AND-병렬블록이다. 이 블록 트리에서 규칙을 이용하여 프로세스를 실행시키는 방법을 (c)와 (d)에서 단계적으로 자세히 살펴본다. 먼저 (c)는 전체 프로세스에 해당하는 블록 B_1을 실행한다. 이것은 T_1, B_2, T_4로 구성된 직렬블록이므로 직렬블록을 실행하는 규칙이 차례대로 적용된다. 여기서 중간에 (d)와 같이 B_2 블록을 만나면 T_2, T_3를 실행하기 위해 AND-병렬블록에 대한 규칙이 실행된다.

전체 프로세스에 대해 start-instance 사건이 클라이언트에서 발생하면, <그림 12>의 순서대로 실행된다. 능동형 데이터베

1. Start-instance : Client가 실행 -> WF_EVENT에 저장
2. R1 수행 : 최상위 블록(B_1)을 결정(Serial Blokc) -> WF _BLOCKINST에 INSERT
3. R7 수행 : 첫 번째 태스크(T_1)를 결정 -> WF_TASKINST에 INSERT
4. R2 수행 : 작업내용을 client에게 전달
5. End : Client 작업 수행 후 -> WF_EVENT에 저장
6. R3 수행 : 태스크의 상태를 바꿈 -> WF_TASKINST를 UPDATE (Committed)
7. R4 수행 : 블록의 진행 상태를 바꿈 -> WF_BLOCKINST를 UPDATE (Committed)
8. R8 수행 : 다음 작업(블록 B_2) 결정 -> WF_BLOCKINST에 B_2를 INSERT
9. R10 수행 : 수행태스크(T_2, T_3) 결정 -> WF_TASKINST에 INSERT (T_2, T_3)
10. R2 수행 : 작업내용을 client에게 전달
11. End : Client 작업 수행 후 전달 -> WF_EVENT에 저장
12. R3 수행 : 태스크의 상태를 바꿈 -> WF_TASKINST를 UPDATE (Committed)
13. R4 수행 : 블록의 진행 상태를 바꿈 -> WF_BLOCKINST를 UPDATE (Committed)
14. R11 수행 : 블록(B_2)의 상태를 바꿈 -> WF_BLOCKINST를 UPDATE (Committed)
15. R8 수행 : 다음 작업(T_4) 결정 -> WF BLOCKINST에 INSERT

그림 12. 규칙 적용 예제.

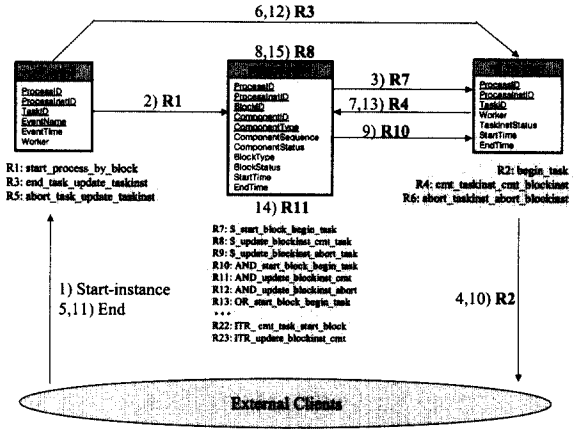


그림 13. 규칙을 처리하는 데이터베이스 테이블간의 관계.

이스는 먼저 전체 프로세스에 해당하는 블록 B_1을 실행한다. 이것은 (c)에서와 같이 T_1, B_2, T_4로 구성된 직렬블록이므로 직렬블록을 실행하는 규칙이 차례대로 적용된다. 예제 프로세스 전체를 실행하기 위한 나머지 과정도 쉽게 이해할 수 있다.

프로세스를 자동 실행하기 위한 규칙을 처리하는 데 사용되는 데이터베이스 테이블 사이의 관계와 관련 규칙을 <그림 13>에 표시하였으며, 규칙 처리는 이러한 관계에 따라 처리된다.

7. 결 론

본 논문에서는 프로세스 모델링과 프로세스의 실행을 결합하는 방법론으로 능동형 데이터베이스의 ECA 규칙을 이용하는 것을 제안하였다. 먼저, 중첩 프로세스 모델을 구성하는 기본 단위가 되는 블록의 유형을 정의하였다. 그리고 프로세스 정의 그래프에서 블록을 자동으로 추출하는 알고리듬을 제시하였으며, 이를 이용하여 블록으로 구성되는 중첩 프로세스 모델 재구성 방법을 제시하였다. 이는 결국 능동형 데이터베이스가 중첩 프로세스 모델을 자동으로 실행할 수 있도록 하기 위한 것이다. 즉, 각 블록에 대한 ECA 규칙을 구현하였으며, 중첩 프로세스 모델 실행시 해당되는 블록의 ECA 규칙을 호출하여 자동으로 프로세스를 통제하는 것을 가능하게 하였다. 마지막으로 제안한 방법의 작동 원리를 간단한 예제를 이용하여 보였다.

본 논문에서 제안한 방법은 다음과 같은 장점이 있다. 첫째, 프로세스 표현의 일반적 방법인 네트워크 기반 방법론과 프로세스의 자동 실행이 가능한 규칙 기반 방법론을 결합하여 양쪽의 장점을 모두 살릴 수 있다. 둘째, 블록 개념을 이용하여 중첩 프로세스 모델을 설계할 수 있어 계층적, 구조적 프로세스 모델링이 가능하다. 셋째, 능동형 데이터베이스의 ECA 규칙으로 프로세스 실행 엔진을 대신함으로써 태스크 흐름을 통제하기 위한 별도의 응용프로그램이 필요하지 않게 된다. 넷째, 블록을 유형별로 정형화한 것은 프로세스를 관리할 필요가 있는 응용 프로그램의 개발을 쉽게 할 수 있음을 의미한다. 다섯째, 최근의 데이터베이스관리시스템은 모두 능동적 기능을 제공하므로, 데이터베이스가 사용되는 환경에서는 어디서나 프로세스 관리 기능을 손쉽게 설치하여 활용할 수 있고, 프로세스의 적용 환경 변화나 적용 규칙의 변경 또한 손쉽게 할 수 있다. 이와 더불어 서로 다른 조직이라도 같은 데이터베이스를 사용하면 워크플로우 관리를 쉽게 통합할 수 있고 따라서 상호운용성(interoperability)이 증가한다는 장점도 있다.

추후 연구과제로는 에이전트(agent) 기술을 활용하여 블록을 구현함으로써 제안한 개념을 분산환경에서 적용하는 것과 프로세스 실행 과정과 결과를 모니터링하는 기능을 추가하는 것 등이 있다.

참고문헌

van der Aalst, W. M. P. (1998), The Application of Petri Nets to Workflow Management, *The Journal of Circuits, Systems and Computers*, 8(1), 21-66.

Al-Ahmari, A. M. A., and Ridgway, K. (1999), An Integrated modeling method to support manufacturing systems analysis and design, *Computers in Industry*, 38, Issue 3, 225-238.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993), Network Flows: Theory, Algorithms, and Applications, Prentice Hall, New Jersey.

Attie, P., Singh, M., Emerson, E., Sheth, A., and Rusinkiewicz, M. (1996), Scheduling Workflows by Enforcing Intertask Dependencies, *Distributed Systems Engineering Journal*, 3(4), 222-238.

Bae, J. S. (2000), Automatic Enactment of Workflow Process using Active Databases, PhD thesis, Seoul National University.

Casati, F., Ceri, S., Pernici, B., and Pozzi, G. (1996), Deriving Active Rules for Workflow Enactment, *Proc. of 7th int. conf. on Database and Expert Systems Applications '96, Zurich, Switzerland, Lecture Notes in Computer Science, Springer-Verlag*, 94-110.

Chrysanthis, P. K., and Ramamritham, K. (1990), ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, ACM Press, Atlantic City, NJ, May 23-25, 194-203.

Dayal, U., Hsu, M., and Ladin, R. (1990), Organizing Long-running Activities with Triggers and Transactions, *Proc. ACM SIGMOD, Hector Garcia-Molina and H. V. Jagadish (Ed.)*, ACM Press, Atlantic City, NJ, May 23-25, 204-214.

Dogac, A., Gokkoca, E., Arpinar, S., Koksal, P., Cingil, I., Arpinar, B., Tatbul, N., Karagoz, P., Halici, U., and Altinel, M. (1997), Design and Implementation of a Distributed Workflow Management System: METUFlow, *In: Advances in Workflow Management Systems and Interoperability, Dogoc, A., L. Kalinichenko, M. T. Ozsu, A. Sheath (Eds.)*, NATO Advanced Study Institute, Istanbul, 61-66.

Georgakopoulos, D., Hornick, M. F., and Manola, F. (1996), Customizing Transaction Models and Mechanisms in a Programmable Environment Supporting Reliable Workflow Automation, *IEEE TKDE*, 8(4), 630-649.

Geppert, A., Tombros, D., and Dittrich, R. (1998), Defining the Semantics of Reactive Components in Event-driven Workflow Execution with Event Histories, *Information Systems*, 23(3/4), 235-252.

Hollingsworth, D. (1994), Workflow Management Coalition Specification: The Workflow Reference Model, *Technical Report TC00-1003*, Issue 1.1, Workflow Management Coalition, Brussels, Belgium, 29.

Kim, Y., Kang, S. H., Kim, D. S., and Bae, J. S. (2000), WW-flow: A Web-based

Workflow Management System Supporting Nested Processes, *IEEE Internet Computing*, 4(3), 55-64.

Kumar, A., and Zhao, J. L. (1999), Dynamic Routing and Operational Controls in Workflow Management Systems, *Management Science*, 45(2), 253-272.

Miller, J., Palaniswami, D., Sheth, A., Kochut, K., and Singh, H. (1997), WebWork: METEOR's Web-based Workflow Management System, *Journal of Intelligent Information Management Systems*, 185-215.

Nadoli, G., and Biegel, J. E. (1993), Intelligent Manufacturing- Simulation Agents Tool, *ACM Transactions on Modeling and Computer Simulation*, 3(1), 42-65.

Paton, N. W. (1998), Active Rules in Database Systems, Springer-Verlag, New York.

Rusinkiewicz, M., and Sheth, A. (1995), Specification and Execution of Transactional Workflows, *in the Modern Database Systems: The Object Model, Interoperability, and Beyond, W. Kim (Ed.)*, Addison-Wesley, New York, 592-620.

Schlenoff, G., Juntilla, A., and Ray, S. (1996), Unified Process Specification Language: Requirements for Modeling Process, NIST.

부 록

1. ACTA 형식론의 의존관계 표현

의존관계	이 름	정 의
t_j CD t_i	Commit Dependency	(Commit $t_j \rightarrow$ Commit t_i) $\wedge$ (Commit $t_i <$ Commit t_j)
t_j CAD t_i	Commit-on-Abort Dependency	(Abort $t_j \rightarrow$ Commit t_i) $\wedge$ (Commit $t_i <$ Abort t_j)
t_j AD t_i	Abort Dependency	(Abort $t_j \rightarrow$ Abort t_i) $\wedge$ (Abort $t_i <$ Abort t_j)
t_j ACD t_i	Abort-on-Commit Dependency	(Abort $t_j \rightarrow$ Commit t_i) $\wedge$ (Commit $t_i <$ Abort t_j)
t_j BD t_i	Begin Dependency	(Begin $t_j \rightarrow$ Begin t_i) $\wedge$ (Begin $t_i <$ Begin t_j)
t_j BCD t_i	Begin-on-Commit Dependency	(Begin $t_j \rightarrow$ Commit t_i) $\wedge$ (Commit $t_i <$ Begin t_j)
t_j BAD t_i	Begin-on-Abort Dependency	(Begin $t_j \rightarrow$ Abort t_i) $\wedge$ (Abort $t_i <$ Begin t_j)

2. 블록의 유형과 ACTA 형식론을 이용한 블록 표현 ($T' \subseteq T = \{ T_i \mid 1 \leq i < n \}$)

블록유형	의존관계	설 명
직렬블록	T_1 BD B	블록의 시작은 첫 번째 태스크를 시작한다.
	$\forall T_i \in T'$, T_{i+1} BCD T_i	i 번째 태스크의 완결은 i+1 번째 태스크를 시작한다.
	B CD T_n	마지막 태스크의 완결은 전체 블록을 완결짓는다.
	$\exists T_i \in T$, B AD T_i	어떤 태스크도 중단되면 전체 블록이 중단된다.
AND-병렬블록	$\forall T_i \in T$, T_i BD B	블록의 시작은 모든 태스크를 시작한다.
	$\forall T_i \in T$, B CD T_i	모든 태스크가 완결되어야 전체 블록도 완결된다.
	$\exists T_i \in T$, B AD T_i	어떤 태스크도 중단되면 전체 블록이 중단된다.
OR-병렬블록	$\forall T_i \in T$, T_i BD B	블록의 시작은 하나 이상의 태스크를 시작한다.
	$\exists T_i \in T'$, B CD T_i	어떤 태스크라도 완결되면 전체 블록도 완결된다.
	$\forall T_i \in T$, B AD T_i	시작한 태스크들이 모두 중단된다면 블록도 중단된다.
ALT-병렬블록	$\exists T_1$ BD B	블록의 시작은 첫 번째 태스크를 시작한다.
	$\forall T_i \in T'$, T_{i+1} BAD T_i	i 번째 태스크의 중단은 i+1 번째 태스크를 시작한다.
	$\exists T_i \in T$, B CD T_i	어떤 태스크라도 완결되면 전체 블록도 완결된다.
	B AD T_n	마지막 태스크가 중단된다면 전체 블록도 중단된다.
CON-병렬블록	$\exists T_i \in T$, T_i BD(C_i) B	블록의 시작은 조건 C_i가 참인 태스크 T_i를 시작한다.
	$\exists T_i \in T$, B CD T_i	T_i가 완결되면 전체 블록이 완결된다.
	$\exists T_i \in T$, B AD T_i	T_i가 중단되면 전체 블록도 중단된다.
반복블록	B BD T_n	반복 조건을 테스트하는 태스크 T_n의 시작은 블록을 시작한다.
	T_1 BCD(C) T_n	반복 조건을 테스트하는 태스크 T_n이 완결되고 반복 조건이 참이면 반복할 태스크 T_1을 시작한다.
	B CD($\neg$C) T_n	반복 조건을 테스트하는 태스크 T_n이 완결되고 반복 조건이 거짓이면 블록이 완결된다.

3. 블록의 유형과 유도된 ECA 규칙

블록유형	의존관계	ECA 규칙 ID	규칙 이름	테이블
공 통	N/A	R1	start_process_by_block	WF_EVENT
		R2	begin_task	WF_TASKINST
		R3	end_task_update_taskinst	WF_EVENT
		R4	cmt_taskinst_cmt_blockinst	WF_TASKINST
		R5	abort_task_update_taskinst	WF_EVENT
		R6	abort_taskinst_abort_blockinst	WF_TASKINST
직렬블록	BD	R7	S_start_block_begin_task	WF_BLOCKINST
	BCD, CD	R8	S_update_blockinst_cmt_task	WF_BLOCKINST
	AD	R9	S_update_ blockinst_ abort_task	WF_BLOCKINST
AND-병렬블록	BD	R10	AND_start_block_begin_task	WF_BLOCKINST
	CD	R11	AND_update_blockinst_cmt	WF_BLOCKINST
	AD	R12	AND_update_blockinst_abort	WF_BLOCKINST
OR-병렬블록	BD	R13	OR_start_block_begin_task	WF_BLOCKINST
	CD, CAD, AD	R14	OR_update_blockinst_cmt	WF_BLOCKINST
ALT-병렬블록	BD	R15	ALT_start_block_begin_task	WF_BLOCKINST
	CD	R16	ALT_update_blockinst_cmt	WF_BLOCKINST
	BAD	R17	ALT_update_blockinst_abort	WF_BLOCKINST
CON-병렬블록	BD	R18	CON_start_block_begin_task	WF_BLOCKINST
	CD	R19	CON_update_blockinst_cmt	WF_BLOCKINST
	AD	R20	CON_update_blockinst_abort	WF_BLOCKINST
반복블록	BCD	R21	ITR_ cmt_task_start_block	WF_BLOCKINST
	CD	R22	ITR_update_blockinst_cmt	WF_BLOCKINST