ACTA 형식론에 기반한 워크플로우 패턴추출
(Workflow Pattern Extraction based on ACTA Formalism)

이 우 기 † 배 준 수 †† 정 재 윤 †††
(Wookey Lee) (Joonsoo Bae) (Jae-yoon Jung)

요 약 워크플로우 관리 시스템은 다양성과 복잡성이 커지고 있는 비즈니스 프로세스 관리의 해결대안으로서 부각되고있다. 본 연구에서는 주어진 프로세스 흐름을 세 가지의 패턴 즉, 반복블록, 직렬블록 및 병렬블록 등으로 단순화하고 노드의 위상적 순서(topological ordering)를 생성하는 알고리즘을 이용하여 사이클을 찾아 반복블록을 제거하고, 다음으로 직렬 및 병렬블록을 반복적으로 검색하는 수로분기 알고리즘을 포함하는 새로운 모델을 제시한다. 그리고 ACTA 형식론에 기반하여 각 블록을 ECA규칙으로 변환하여 사건(event)을 감지하는 워크플로우 시스템으로의 구현 방안을 제시한다. 그 모델의 결과를 컴퓨터가 수행할 수 있도록 만들어주는 과정 즉, 비즈니스 프로세스를 모델에 입각하여 통제하는 데에 사건-조건-처리(ECA)규칙을 사용한다. 유형별로 ECA 규칙에 입각한 통제 논리를 설계하였으며, 이것은 규칙기반 워크플로우 관리시스템의 기초가 될 수 있다. 또한 본 연구의 결과가 현행 DBMS들의 능동형 규칙(active rule)에 적용될 수 있도록 구체적 대안을 제시하였다.

키워드 : 워크플로우, 블록, 수로분기, ECA 규칙

Abstract As recent business environments are changed and become complex, a more efficient and effective business process management are needed. This paper proposes a new approach to the automatic execution of business processes using Event-Condition-Action (ECA) rules that can be automatically triggered by an active database. First of all, we propose the concept of blocks that can classify process flows into several patterns. A block is a minimal unit that can specify the behaviors represented in a process model. An algorithm is developed to detect blocks from a process definition network and transform it into a hierarchical tree model. The behaviors in each block type are modeled using ACTA formalism. This provides a theoretical basis from which ECA rules are identified. The proposed ECA rule-based approach shows that it is possible to execute the workflow using the active capability of database without users' intervention.

Key words : Workflow, Block, Water-Branch, ECA

1. 서 론

최근 기업들은 비즈니스를 수행함에 있어서 내외적으로 변화의 압력을 크게 받고 있다. 첫 번째는 고객의 비중이 커지면서 그 요구의 다양성을 수용하기위한 내부적 변화 압박이 점증하는 것이다. 이를 위해 기업은 별도의 회사를 만드는 식의 투자보다는 비즈니스의 처리과정 소위, 비즈니스 프로세스를 다양화하는 대안을 찾고 있다. 또 하나의 변화요인은 기업 외부에서 발생하고 있는데, 업체간 동맹관계나 전략적 제휴 및 e-비즈니스 등 기업 외적인 변화가 커지고 있는 것이다. 이러한 외적요인은 다른 회사의 비즈니스 프로세스와의 관련성까지 고려하도록 만들고있다. 이러한 내적 외적인 변화는 한편으로 새로운 비즈니스 프로세스를 생성시킬 뿐만 아니라, 비즈니스 프로세스의 복잡도를 증가시키고 있다[1]. 따라서 효과적이고 효율적인 프로세스 관리를 도와주는 기술과 도구의 필요성이 더욱 커지고 있는 것이다. 이로 인해 최근 많은 정보 시스템들이 독립된 업무의 단순한 관리를 넘어서 프로세스 관리와 같은 고급기능을 제공하려 노력하고 있다[2,3]. 이러한 문제에 대한 새로운 해결책으로서 워크플로우 관리 시스

· 이 논문은 2004년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2004-003-D00477).
† 종신회원 : 성결대학교 컴퓨터학부 교수
wook@sungkyul.edu
(Corresponding author임)
†† 정 회 원 : 전북대학교 산업정보시스템학과 교수
jsbae@hanmir.com
††† 정 회 원 : 서울대학교 자동화시스템연구소
dodgerss@ara.snu.ac.kr
논문접수 : 2004년 10월 4일
심사완료 : 2005년 8월 24일

템(WFMS: WorkFlow Management System)은 복잡한 업무 프로세스를 정의, 관리, 실행하는 도구로서 부각되고 있다.

우선 비즈니스 프로세스의 예제를 그림 1에서 살펴보기로 하자. 이것은 신용카드 회사의 입회심사 업무 프로세스로서 '가입서식 기재(application form filling)', '서류 전자화(form scanning)'와 같은 여러 개의 활동들로 이루어져 있다. WFMS는 비즈니스 논리를 표현하기 위해 이와 같이 도식적인 모델을 빈번하게 사용한다. 이는 여러 활동들의 선후행 관계 및 직렬 혹은 병렬과 같은 구조적인 관계의 표현에 유리하기 때문이다. 또한 업무 수행자, 관련 문서, 필요한 응용 프로그램 등과 같이 각 활동의 상세한 사양도 포함할 수 있다.

WFMS는 전형적으로 프로세스 모델을 생성시키는 프로세스 설계 도구와 그 모델을 실행시키는 워크플로우 엔진을 가지고 있다. 대부분의 기존 WFMS에서는 프로세스 모델이 워크플로우 엔진에 의해서 이해될 수 있는 형태로 변환된다. 이런 종류의 시스템에서는, 워크플로우 엔진이 프로세스 모델을 실행하고 통제하는데 핵심적인 역할을 수행한다. 본 논문에서는 워크플로우 프로세스의 구현을 위한 새로운 모델을 제시한다. 그 모델의 결과를 컴퓨터가 수행하도록 만들어주는 과정 즉, 비즈니스 프로세스를 모델에 입각하여 통제하는 데에 사건-조건-처리(ECA) 규칙을 사용하는 방법을 제안한다. ECA 규칙은 기존의 프로세스 모델에서 유도되고 데이타베이스의 능동적인 능력에 의해 실행된다.

논문의 구성은 다음과 같다. 제2장에서는 프로세스 모델링과 관련된 기존 연구를 살펴보며, 제3장에서는 프로세스 모델의 특징과 블록의 유형을 정리한다. 제4장은 주어진 프로세스 정의에서 블록을 찾아내 구조화하고, 이를 중첩 모델로 변환하는 알고리즘을 설명한다. 제5장은 블록 정의에서 ECA 규칙을 유도하여 능동형 데이타베이스로써 프로세스 흐름을 통제할 수 있도록 하는 방법을 설명하며, 제6장은 제안한 방법론의 작동 예를 설명하고, 마지막 제7장에서 결론을 맺는다.

2. 배경연구

워크플로우는 "컴퓨터에서 실행할 수 있도록 표현된 비즈니스 프로세스"로 정의되며, 워크플로우 관리 시스템은 "이 워크플로우를 정의하고, 정의된 워크플로우의 실행과 순서를 통제하며, 프로세스 전체를 관리하는 소프트웨어 시스템"을 말한다[4]. 최근 10년 동안 프로세스 모델링 및 표현 언어 개발[5,6], 프로세스 분석과 검증 방법[7,8], 웹(Web) 또는 에이전트 기반 분산 워크플로우[9], 프로세스의 동적 변화 지원[10], 실시간 트랜잭션 처리[11] 등의 다양한 연구가 있었다. 여러 선행 연구가 있었다.

프로세스(process)란 특정한 결과를 수행하기 위하여 설계된 구조화된 활동(activity)들이다[12]. 이 구조화된 활동을 표현하기 위하여 프로세스 실행 언어는 크게 두 가지 패러다임을 사용한다. 첫 번째는 유방향 그래프를 이용한 모델링이다. 유방향 그래프 방법은 사용자가 인터페이스를 통하여 프로세스를 설계하기 용이하여, 대표적인 프로세스 관리 컨소시엄인 WfMC(Workflow Management Coalition)의 XPDL 표준을 비롯하여 대부분의 프로세스 설계 도구에서 사용된다. 또 다른 방법은 블록 기반의 프로세스 모델링이다. 이 방법은 프로그래밍 언어로 표현하기가 적합하여, BPML이나 BPEL4WS, WSCI 등의 웹 프로세스 실행 언어에서 대표적으로 사용된다[13]. 유방향 그래프 기반의 프로세스 모델은 사용자 설계 용이성에서는 뛰어나지만, 프로세스 전체 개별 액티비티의 예외처리만 가능할 뿐, 프로세스의 블록에 대한 개념이 없고 특정 영역에 대한

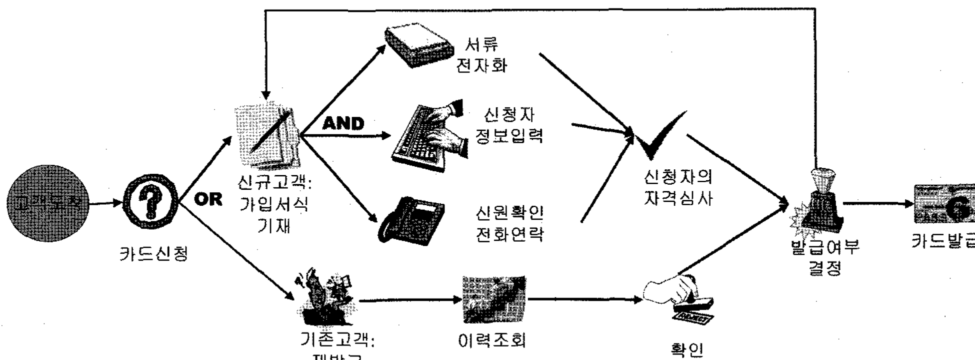


그림 1 비즈니스 프로세스 예제

구분이 불가능하여 부분적인 트랜잭션 관리나 예외처리를 적용하기가 힘들다. 본 논문에서는 이러한 제약을 극복하기 위하여, 그래프 기반의 프로세스 모델로부터 블록 기반의 패턴을 찾아내어, 명확한 규칙에 의해 실행될 뿐만 아니라, 영역별 트랜잭션 처리 및 관리가 가능한 워크플로우 실행 방법을 제안하고 구현하였다.

워크플로우를 자동으로 실행하기 위하여, 규칙 기반을 사용하는 여러 연구들이 있었다[14-16]. 본 연구에서는 특히 ECA 규칙을 이용하여 워크플로우를 실행하는 방법을 제안하였다. ECA 규칙은 본래 수동적인 데이타 저장소를 능동적으로 작동하게 하기 위하여 데이타베이스분야에서 사용된 강력한 메커니즘이다[17]. ECA 규칙은 워크플로우 실행에 적용되어, 워크플로우 시스템이 규칙기반 시스템(Rule-based System)으로 작동하게 하여, 실행시간 변경이 가능한 동적인 워크플로우와 비정형 워크플로우를 지원할 수 있다[18]. 본 연구에서는 기존의 워크플로우를 정의하는데 가장 보편적으로 사용되는 유방향 그래프 기반의 워크플로우 프로세스를 ACTA 형식론에 기반하여 ECA 규칙을 통해 자동실행하는 방법론을 제시한다. 이를 위하여 본 연구에서는 그래프 기반의 프로세스로부터, 블록을 이용한 워크플로우 패턴을 식별하는 알고리즘을 제시하였다.

본 연구에서 다루는 유방향 그래프 기반의 프로세스 모델은 각 활동들(tasks)과 의존관계(transitions), 실행주체(performers), 실행할 내용(work contents)을 정의해주며, 또한 각 활동의 입출력 조건(input and output conditions)을 정해주는 것이다[11]. 본 연구에서는 작업실행 순서를 결정하는 선후행 관계와 직렬 혹은 병렬 흐름을 나타내는 프로세스 모델의 구조적인 측면을 다룬다. 그 밖의 속성들 예컨대, 실행주체, 소요자원, 구체적인 작업 내용 및 실행 시간 등[4]은 본 논문에서는 고려하지 않는다. 프로세스 모델의 구조적인 측면은 유방향 그래프에 의해 표현될 수 있으며 이는 프로세스 정의 네트워크(process definition network)라고 부른다[1,19].

그림 2는 프로세스 정의 네트워크의 예제이다. 둥근 노드는 구성 태스크를 나타내고, 노드를 연결하는 아크는 선후행 관계를 나타낸다. 예를 들어 태스크 T2는 태스크 T3와 T4보다 선행한다. 어떤 태스크들은 직렬로 연결되고 어떤 것은 병렬로 연결된다. 이 그림에서 T4, T8, T10은 직렬로 연결의 예제이다. 병렬인 경우는 T3가 분기되어 T5, T6, T7으로 나누어지고 T9으로 병합된다. WFMS에서 일단 프로세스가 실행되면, 특정 태스크의 모든 선행 태스크가 완료되어야만 그 태스크를 시작할 수 있다. 태스크의 상태는 프로세스 실행 시에 변경되어서 어떤 것은 실행되고 있고, 어떤 것은 이미 완료되었거나 실행을 기다릴 수도 있다. 이와 같은 상태 변경 모델은 제3장에서 자세히 설명하기로 한다.

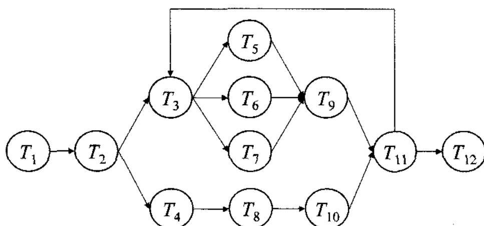


그림 2 프로세스 정의 네트워크

프로세스 모델의 도식적인 표현은 사용자가 그 의미를 쉽게 이해하고 검증하는 할 수 있는 시각적인 도구가 된다. 그러나 이 모델의 한계는 이것이 컴퓨터가 직접 읽을 수 있는 형태가 아니라는 점이며, 따라서 WFMS에서 실행되기 전에 컴퓨터가 읽을 수 있는 언어로 변환과정이 요구된다.

3. 블록을 이용한 워크플로우 패턴의 분류

블록(block) 개념은 본 연구의 기본적인 출발점이다. 비즈니스 프로세스는 크게 보면 일정한 몇 개의 패턴으로 구분할 수 있는데, 각 패턴은 특정한 단위로 단순화할 수 있다. 그 단순화된 패턴을 본 논문에서는 블록(block)이라고 부른다. 블록은 워크플로우 프로세스 구성 요소들의 행동양식을 명확히 규정할 수 있으면서 의미를 가지는 최소단위로 정의된다[5]. 논문[5]에서는 분산 환경에서 동시에 실행되는 워크플로우를 위한 WFMS에서 블록개념을 처음 도입하여, 블록 구조의 워크플로우 정의 언어와 워크플로우 스케줄링 방법을 제안하였다. 본 연구에서의 차이점은 프로세스 모델에서 블록의 구조적 탐색 및 ECA 규칙과의 대응관계를 밝힌 점 및 그 규칙의 DB관점에서의 적용에 있다. 비즈니스 프로세스에서는 프로세스 모델에서 발견할 수 있는 행동양식은 그림 3과 같이 반복, 직렬, 병렬 패턴으로 분류할 수 있다. 본 논문에서는 이러한 패턴의 조합으로 이루어지는 네트워크에 한정한다. 예컨대, 그림 2의 예제는 그림 3에 나타난 일련의 패턴으로 구분할

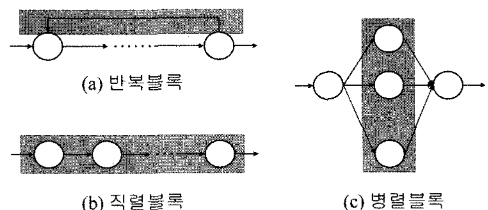


그림 3 패턴의 분류

수 있다는 것을 알 수 있다.

첫째, 반복블록이란 그림 3의 (a)와 같이 사이클을 의미한다. 이 패턴은 어떤 태스크가 반복적으로 수행될 때 나타나는 것이다. 반복 블록은 시작과 종료 노드, 노드를 연결하는 반복 아크에 의해 정의된다. 그림 2의 경우, 반복블록은 T11에 시작해서 T3에서 끝난다. 반복블록의 정의는 반복이 필요한 때를 나타내는 반복 조건을 포함하는데, 이것은 반복블록의 시작 노드에 정의된다.

둘째, 직렬블록은 그림 3의 (b)에 나타나 있는 것과 같이 태스크 흐름에 있어 반복이나 분기, 병합이 없는 일련의 흐름이다. 태스크 패턴은 연속적으로 연결된 두 개 이상의 태스크를 가지고 있고, 각 태스크는 단지 하나의 선행 태스크와 하나의 후행 태스크를 가진다. 그래서 모든 태스크는 연속적으로 실행되어야 한다. 하나의 태스크가 성공적으로 완료되어야 비로소 후행 태스크가 시작될 수 있다.

마지막으로 병렬블록은 그림 3의 (c)에 나타나 있는 것과 같이 하나의 노드가 2개 이상으로 분지되어 평행하게 진행되다가 하나의 노드로 병합되는 흐름을 나타낸다. 이 패턴은 더 세부적으로 나누어져 AND-, OR-, POR-, COR-병렬의 4가지로 구분된다. AND병렬 패턴은 모든 구성 태스크가 동시에 실행된다. 모든 태스크의 성공적인 완료는 다음 태스크를 시작시킨다. 만약 하나의 태스크가 실패하면, 전체 프로세스가 실패한다. 이 마지막 제한이 OR병렬 패턴에서는 완화된다. OR병렬에서는 하나의 태스크가 성공하면 다음 태스크가 시작한다. POR병렬 패턴은 모든 태스크가 우선순위를 가지고 있어서 가장 높은 우선순위를 가지는 태스크가 먼저 실행된다. 만약 이 태스크가 실패하면 다음 높은 우선순위를 가지는 태스크가 시작된다. 반면에 하나의 태스크가 성공하면 모든 다른 태스크는 무시되며 후행 태스크가 시작된다. COR병렬 패턴은 분기에 어떤 조건이 있어서 그 조건을 만족하는 태스크만 실행된다. 이 패턴은 하나의 구성 태스크만 실행하도록 조건을 설정하여 배타적 OR 분기를 나타낼 수 있다.

이상의 병렬 패턴들은 의미상으로 다르지만, 같은 도식적인 구조를 가진다. 이것은 도식적인 객체인 노드와 아크는 오로지 태스크의 분기와 병합 관계만 다룰 수 있기 때문이다. 구체적인 병렬 패턴을 구분하는 의미는 분기 노드나 병합 노드에 명세한다.

4. 블록검색 알고리즘

본 장에서는 주어진 프로세스 모형으로부터 블록을 탐색해 내는 알고리즘을 개발하는 문제를 다룬다. 여기서 사용된 기호는 다음과 같다. 프로세스 정의 네트웍(G)은 노드전체집합(N) 및 아크집합(A)에 대하여, 개별노드 v, $v' \in N$, 출발노드(s) 그리고 v에 대하여 선행 및 후행노드 $pred(v)$, $succ(v)$ 또한 수면높이 $w(v)$ 및 수면높이의 집합 $w\text{-}list$ 등으로 표현된다.

그림 4는 이 알고리즘의 전체적인 흐름을 나타낸다. 둥근 사각형은 중요 단계를 나타내고 각각에 대해서 자세히 설명하기로 한다.

가장 먼저 프로세스 네트워크에서 반복블록부터 탐색하는데, 아래와 같다.

PROCEDURE Iterative-block-detection (**in** G, **out** (the start node, the end node))

QUEUE := $\{s\}$;

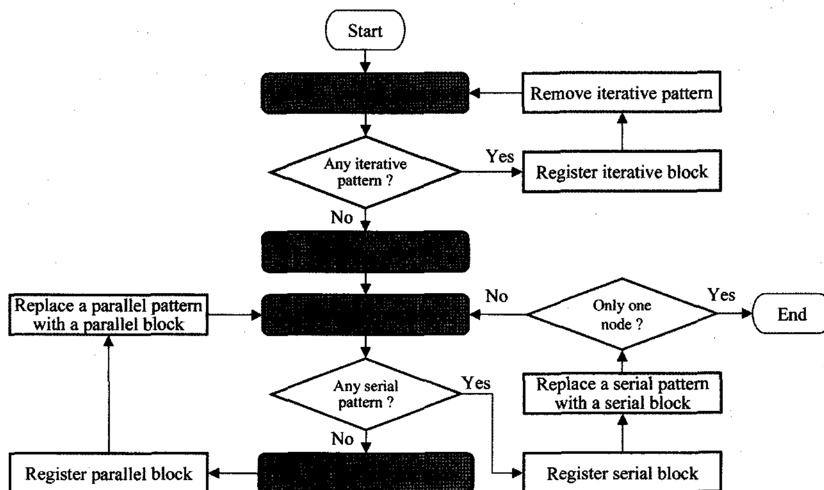


그림 4 블록 탐색 알고리즘

```
    while (QUEUE ≠ ϕ) do
        let v be the first element of QUEUE;
        remove v from QUEUE;
        mark v;
        for (all v′ ∈ succ(v)) do
            if (v′ is marked) then return (v, v′);
            if (all pred(v′) are marked) then append v′ to QUEUE;
        end
    end
    return null;
end Iterative-block-detection
```

반복블록은 유방향 사이클을 이루므로, 노드의 위상적 순서(topological ordering)를 생성하는 알고리즘을 이용하여 사이클을 찾는다. 이 알고리즘은 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.

다른 종류의 블록을 탐색하기 전에, 알고리즘은 다음과 같은 "수로분기(Branch-water)" 절차를 수행해야 한다.

```
PROCEDURE Branch-water (in G, out w-list)
    for all v, do w(v) := 0.0;
    w(s) := 1.0;  w-list := {1.0};  QUEUE := {s};
    while (QUEUE ≠ ϕ) do
        let v be the first element of QUEUE;
        remove v from QUEUE;
        mark v;
        for (all v′ ∈ succ(v)) do
            w(v′) := w(v′) + w(v) / |succ(v)|;
            if (w(v′) ∉ w-list) then add w(v′) to w-list;
            if (all pred(v′) are marked) then append v′ to QUEUE;
        end
    end
end Branch-water
```

이 단계는 각 노드에 숫자를 할당한다. 여기서 프로세스 정의 네트워크를 연결된 파이프라인으로 생각하고, 물이 파이프라인으로 부어진다고 생각한다. 물이 파이프라인으로 흘러 가면서 네트워크에서 분기가 되기도 하고 병합이 되기도 한다. 물의 높이인 "수면높이(water-level)"라고 하는 숫자가 각 노드에 할당된다. 네트워크의 시작 노드에 먼저 시작 수면높이를 할당한다. 이 숫자는 물이 파이프라인을 흘러가면서 네트워크의 모든 아크로 전파된다. 만약 한 노드의 수면높이가 r이라고 하고 이 노드가 k개의 노드로 분지가 된다면, 분지된 노드에 r/k가 할당된다. 즉, 각 노드의 수면높이는 그 노드의 선행 노드들의 수면높이를 합한 것과 같다.

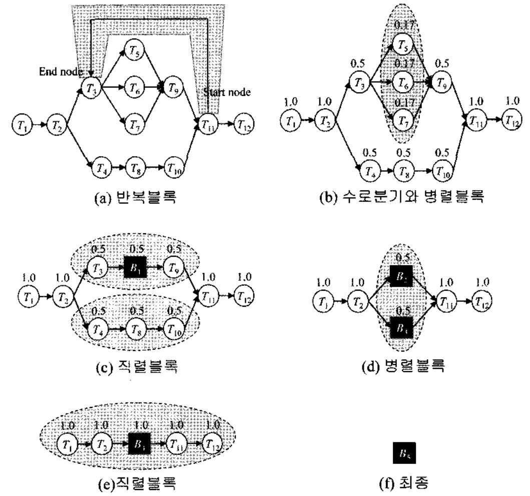


그림 5 블록 탐색 알고리즘의 적용 예제

이러한 수면높이 알고리즘을 이용하여 프로세스 네트워크에서 단계적으로 블록을 찾을 수 있다. 예를 들어 그림 5(b)을 살펴보면, 각 노드 위에 수면높이를 할당하였다. 블록 검색은 가장 낮은 수면높이를 가지고 있는 노드들로부터 탐색할 수 있다. 그림에서는 병렬패턴(T5, T6, T7)이 가장 먼저 탐색된다. 수면높이 절차는 다음 두 단계에서 이용되는데, 다음 단계는 직렬패턴 탐색과 병렬패턴 탐색을 번갈아 가면서 진행한다. 다음 의사코드는 직렬블록 탐색 절차를 나타낸다.

```
PROCEDURE Serial-block-detection (in G, out (SB, w-list))
    b := min(w-list);  LOOP := T;  QUEUE := {s};
    while (LOOP = T) do
        if (QUEUE = ϕ) then return null;
        let v be the first element of QUEUE;
        remove v from QUEUE;
        if (w(v) = b && |succ(v)| = 1 && |pred(succ(v))| = 1) then
            LOOP := F;
            SB := SerialFrom(v);
            w(SB) := w(v);
        else append succ(v) to QUEUE;
    end
end Serial-block-detection
```

이 절차는 직렬패턴에 속한 노드 집합을 찾아 준다. 먼저 G의 시작 노드를 찾은 다음 첫째직렬블록을 찾을 때까지 계속한다. 직렬블록은 연속적인 두 태스크의 각 입력노드(indegree)와 출력노드의 수(outdegree)가 모두 하나인 노드들로 구성되므로 쉽게 식별할 수 있다. 모두 같은 수면높이를 갖는 노드들로 이루어진 직렬패

턴을 찾아서 하나의 직렬블록으로 압축한다. 우리의 알고리즘은 이 직렬블록을 등록하고 그 블록을 하나의 노드로 바꾸어서 그래프를 변경한다. 새 직렬블록의 수면높이는 그 구성 노드가 가지는 수면높이와 같다. 더 이상의 직렬블록이 없으면, 알고리즘은 다음과 같은 병렬블록 탐색 단계로 진행한다.

```
PROCEDURE Parallel-block-detection (in G, out (PB, w-list))
    b := min(w-list);  LOOP := T;  QUEUE := {s};
    while (LOOP = T) do
        let v be the first element of QUEUE;
        remove v from QUEUE;
        if (w(v) = b) then
            LOOP := F;
            PB := succ(pred(v));
            w(PB) := w(b) * |succ(pred(v))|;
            update w-list;
        else append succ(v) to QUEUE;
    end
end Parallel-block-detection
```

병렬블록 탐색도 가장 깊숙이 있는 병렬패턴을 탐색하기 위해 최소 수면높이를 이용한다. 만약 최소 높이를 가지는 노드를 발견하면 이것은 병렬패턴이 존재한다는 것을 의미한다. 왜냐하면 최소 수면높이와 관련된 직렬패턴은 이미 모두 탐색되었기 때문이다. 탐색된 노드와 병렬관계에 있는 모든 노드들을 탐색하여 병렬블록으로 등록한다. 이 알고리즘은 그래프의 위상(topology)만을 이용하므로 병렬블록의 더 자세한 구분인 AND-, OR-, POR-, COR-병렬블록을 알기 위해서는 분기와 병합 조건을 참조하여야 한다. 알고리즘은 다시 직렬블록 탐색을 진행한다.

5. ECA 규칙의 유도

5.1 의존관계(Dependency Relation)

프로세스 통제의 기본적인 메카니즘은 구성 태스크의 상태전이로 설명될 수 있다. 그림 6은 본 논문에서 사용된 상태전이 모델이다[5,6,9,20]. 특정 시점의 태스크 상태는 'Not-Ready', 'Ready', 'Executing', 'Committed', 'Aborted' 중에 하나이다. 모든 태스크는 'Not-Ready' 상태로 초기화 되고, 그것의 모든 선행 태스크가 완료되면 'Ready' 상태가 된다. 각 태스크는 실행 전에 반드시 만족해야 되는 선행조건을 가질 수 있다. 만약 모든 선행조건이 만족되면 그 태스크는 시작하고 상태를 'Executing'로 바꾼다. 선행조건이 없는 태스크는 즉시 시작한다. 모든 태스크는 'Committed'와 'Aborted' 둘 중 하나의 상태로 끝난다. 그림에서 아크의 'Begin' 과 'Commit'는 상태전이를 일으키는 사건을 나타낸다.

블록은 3절에서 언급된 대로 여러 개의 태스크로 구성되기 때문에, 하나의 태스크 상태는 다른 태스크와 그것이 속한 블록의 상태에 영향을 미친다. 태스크와 블록간에 상호관계를 나타내기 위해 각 블록 유형별로 의존관계를 정의한다.

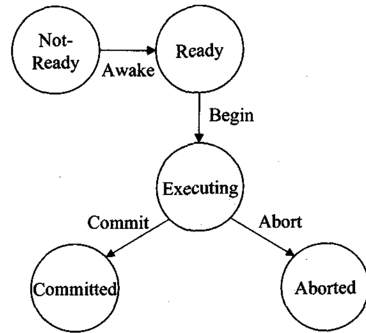


그림 6 태스크 처리의 상태 전이도

직렬블록은 모든 태스크가 일렬로 이루어진 것으로 블록이 시작하자마자 첫번째 태스크가 시작하는 한다. 선행 태스크가 완료되면 다음 태스크가 실행되기 시작한다. 마지막 태스크가 완료되면 전체 블록이 완료된다. 블록을 실행할 때, 하나의 태스크가 중단되면 전체 블록도 중단된다. 이 경우에 이미 완료된 선행 태스크는 다시 원래 상태로 되돌려 져야 한다. 이러한 태스크를 위하여 원래 상태로 되돌리는 추가적인 복원태스크를 정의한다. 반복블록은 비슷하게 작동하지만 특정 조건이 만족될 때까지 반복한다는 것이 다른 점이다. AND-병렬블록은 모든 구성 태스크가 동시에 실행된다. 만약 모든 태스크가 성공적으로 완료되면 블록이 완료된다. 만약 태스크 중에 하나라도 중단되면, 이미 완료된 태스크에 대응되는 복원태스크가 실행되고 난 다음 블록이 중단된다. OR-병렬블록은 모든 태스크가 동시에 시작한다. 그러나 어느 태스크 하나라도 완료하면 전체 블록이 완료되고 모든 태스크가 중단되면 블록도 중단된다. POR-병렬블록과 COR-병렬블록은 완료되고 중단되는 데는 OR-병렬블록과 동일하다. 단지 다른 점은 블록이 시작될 때 모든 태스크가 시작하지 않는다는 것이다. POR-병렬블록은 미리 정의된 순서대로 태스크가 실행되고, COR-병렬블록은 특정 조건을 만족시키는 태스크만 실행된다는 것이다. 본 논문에서는 블록이 계층적으로 조직되고 하나의 블록이 다른 블록의 구성 태스크로 취급될 수 있다. 그래서 블록과 복원태스크도 일반 태스크와 함께 의존 관계를 가지는 것으로 생각한다. 상태전이에 대한 상관관계를 나타내기 위해 ACTA(actions의 라틴어) 형식론을 사용한다. ACTA 형식론은 원래 데이타베이스에서 트랜잭션간의 상관관계를 정의하는 것이다[21,22].

표 1과 같이 미리 정의된 9개의 의존관계를 이용한다. 하나의 사건에 의해 상태전이가 일어나기 때문에 태스크 사이의 의존관계는 사건 사이의 관계로 정의될 수 있다. 예를 들어 't_j CD t_i'는 t_j and t_i 사이의 commit dependency를 나타내고, 의미하는 것은 '전트랜잭션 t_i가 commit된 다음 후트랜잭션인 t_j가 commit 되어야 한다'는 의존관계를 나타낸다. 나머지 의존관계도 비슷한 방법으로 설명된다. 기호 ⇒은 논리적인 결론을 의미하고, 기호 <은 두 사건간의 시간적인 순서를 나타낸다. 예를 들어 $e < e'$은 사건 e가 사건 e'보다 먼저 발생한다는 것을 의미한다. H는 워크플로우 실행 중에 이미 발생한 모든 사건들의 집합을 의미한다. 즉 $e \in H$은 사건 e가 이미 발생했다는 것을 의미한다. 의존관계는 한 태스크의 상태 전이가 다른 태스크의 상태전이에 어떻게 영향을 미치는지를 표현하는데 사용된다. 이렇게 얻어진 의존관계에서 ECA 규칙을 기계적으로 유도해 낼 수 있다. 이것은 다음 절에서 다룬다.

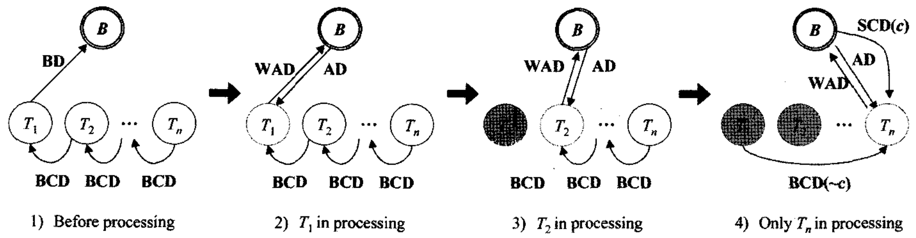
5.2 블록표현을 위한 ACTA 형식론

각 블록 유형에 대해 ACTA 형식론을 이용하여 의존관계를 표현할 수 있다. 하나의 블록 유형은 구조와 의미에서 다른 블록 유형과 다르기 때문에, 의존관계의 표현도 달라질 수밖에 없다. 각 블록 유형에 대한 의존관계는 그림 7에 제시되어 있다. 예를 들어, 그림 7(c)에 나와 있는 AND-병렬블록에 속한 의존관계를 살펴보자. 이 블록에서는 몇 개의 태스크가 동시에 처리된다. 블록이 처리되기 전에, 블록과 각 태스크는 begin dependency(BD)를 가지는데, 블록이 시작하자마자 모든 태스크가 바로 시작된다는 것을 의미한다. 이것에 대한 것은 그림의 1)번 단계에 나타나 있다. 2)번 단계는 블록내의 모든 태스크가 시작된 후의 의존관계를 보여준다. 블록은 모든 태스크와 commit dependency (CD)를 가지고 있다. 그래서 어떤 하나의 태스크도 완료되지 않으면 블록은 완료되지 않는다. 또한 abort dependency(AD)는 어떤 태스크가 중단하더라고 블록이 중단되도록 만드는 것을 나타낸다. 만약 블록이 중단되면, 아직 완료되지 않은 모든 태스크들은 weak abort dependency(WAD)에 의해서 중단된다.

이 그림에서 태스크 T_i는 새로운 태스크 유형인 복원태스크(compensation task) CT_i와 연결되어 있다. 만약 T_i가 시작하면, CT_i는 내부적으로 초기화된다. 만약 한 태스크가 COMMITTED 되었는데, 그것이 포함된 AND-병렬블록이 중단되었다고 가정해 보자. 이 경우에는 태스크가 완료한 것은 전체 프로세스의 성공적인 완료와는 아무 상관없으므로 그 태스크의 완료를 원상 복구 시켜야 하는 것이다. CT_i가 바로 이러한 원상 복구 역할을 수행한다. 태스크와 복구태스크 간의 begin-on-commit(BCD) 의존관계는 원래 태스크가 완료되어야만 복구 태스크가 시작할 수 있다는 것을 나타낸다. 하나의 태스크가(예를 들어 태스크 T_1) 완료하였다면 그 의존 관계는 그림 7의 세번째 그림 (c)에 나타나 있다. 이제 T_1은 더 이상 블록의 상태에 영향을 미치지 않는다. 그러나 복원 태스크 CT_1는 블록과 begin-on-abort dependency(BAD)와 commit-on-abort dependency

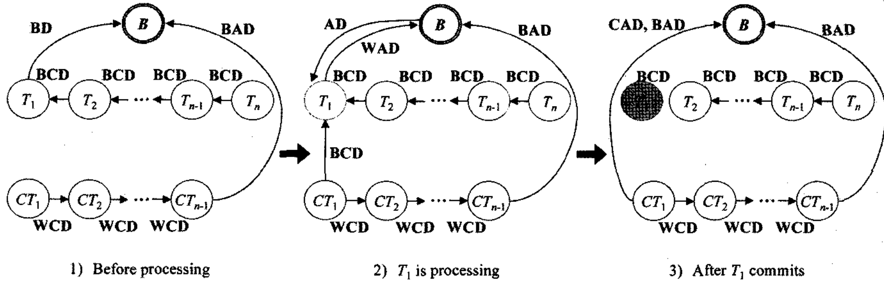
표 1 ACTA 형식론의 의존관계 표현

Dependency Relation	Name	Definition	Description
t_j CD t_i	Commit Dependency	$(Commit\ t_j \in H) \Rightarrow ((Commit\ t_i \in H) \Rightarrow (Commit\ t_i < Commit\ t_j))$	If both t_i and t_j commit, then the commitment of t_i precedes the commitment of t_j.
t_j SCD t_i	Strong-Commit Dependency	$(Commit\ t_i \in H) \Rightarrow (Commit\ t_j \in H)$	If t_i commits, then t_j commits.
t_j AD t_i	Abort Dependency	$(Abort\ t_i \in H) \Rightarrow (Abort\ t_j \in H)$	If t_i aborts, then t_j aborts.
t_j WAD t_i	Weak-Abort Dependency	$(Abort\ t_i \in H) \Rightarrow (\neg(Commit\ t_j < Abort\ t_i) \Rightarrow (Abort\ t_j \in H))$	If t_i aborts and t_j has not yet committed, then t_j aborts.
t_j BD t_i	Begin Dependency	$(Begin\ t_j \in H) \Rightarrow (Begin\ t_i < Begin\ t_j)$	Task t_j cannot begin executing until t_i has begun.
t_j BCD t_i	Begin-on-Commit Dependency	$(Begin\ t_j \in H) \Rightarrow (Commit\ t_i < Begin\ t_j)$	Task t_j cannot begin executing until t_i commits.
t_j BAD t_i	Begin-on-Abort Dependency	$(Begin\ t_j \in H) \Rightarrow (Abort\ t_i < Begin\ t_j)$	Task t_j cannot begin executing until t_i aborts.
t_j CAD t_i	Commit-on-Abort Dependency	$(Abort\ t_i \in H) \Rightarrow (Commit\ t_j \in H)$	If t_i aborts, then t_j commits.
t_j WCD t_i	Weak-Begin-on -Commit Dependency	$(Begin\ t_j \in H) \Rightarrow ((Commit\ t_i \in H) \Rightarrow (Commit\ t_i < Begin\ t_j))$	If t_i commits, t_j can begin executing after t_i commits.

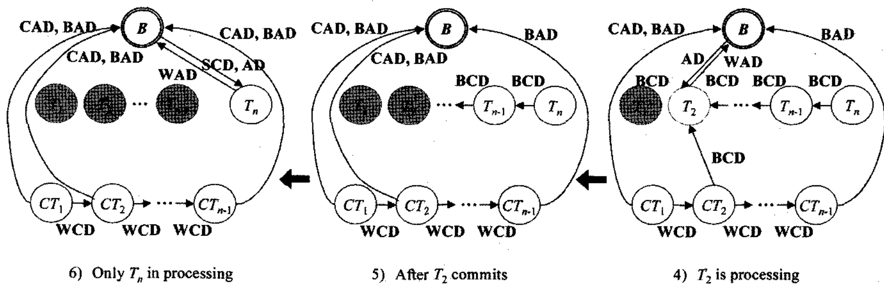


1) Before processing 2) T_1 in processing 3) T_2 in processing 4) Only T_n in processing

(a) 반복블록

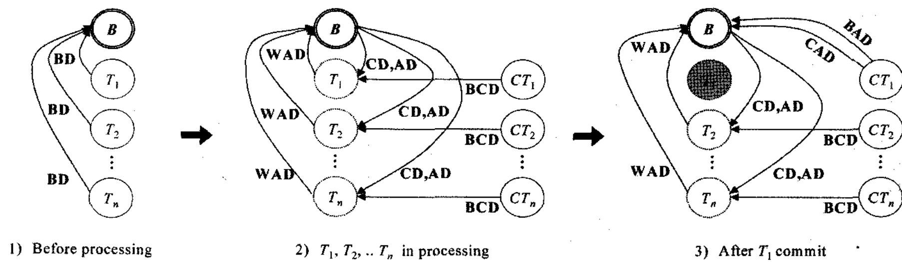


1) Before processing 2) T_1 is processing 3) After T_1 commits

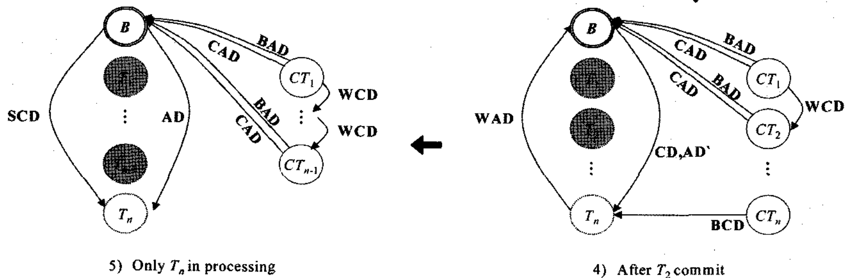


6) Only T_n in processing 5) After T_2 commits 4) T_2 is processing

(b) 직렬블록

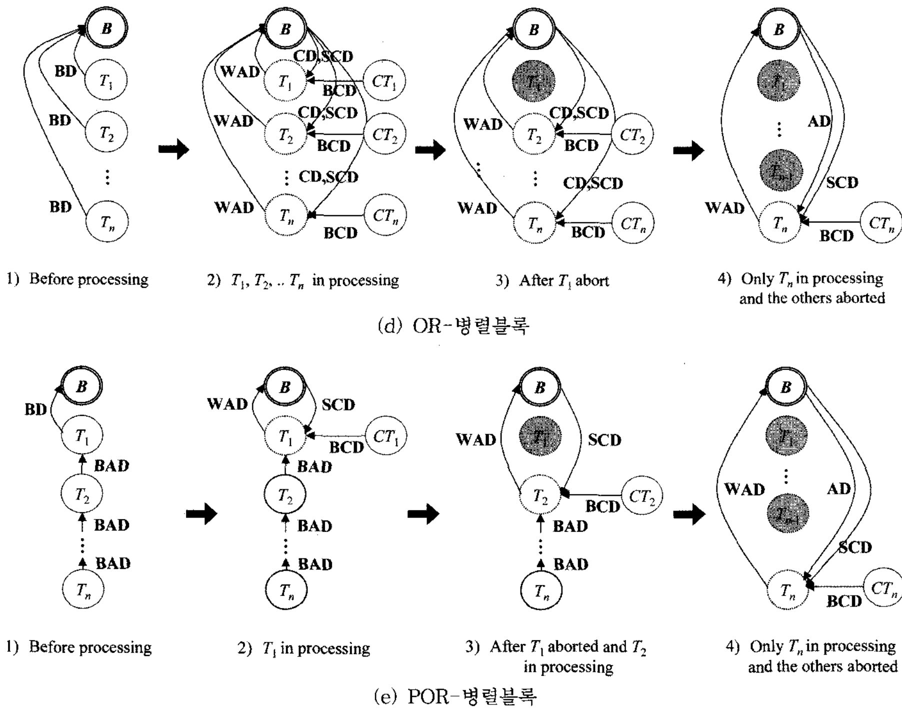


1) Before processing 2) T_1, T_2, .. T_n in processing 3) After T_1 commit



5) Only T_n in processing 4) After T_2 commit

(c) AND-병렬블록



(d) OR-병렬블록

(e) POR-병렬블록

그림 7 블록 유형별 ACTA 형식론

(CAD) 관계를 설정한다. 따라서 블록이 중단하게 되면 복원 태스크도 영향을 받게 된다. 이와 같이 태스크가 완료되면 의존관계가 수정된다.

결국 모든 태스크가 완료되면 블록은 그림 7 (c)의 5)번 그림에 있는 것과 같이 strong-commit dependency (SCD)에 의하여 즉시 완료된다. 만약 블록이 이 과정에서 중단되면, 완료된 태스크에 대한 복원 작업을 시작시킨다. 복원작업의 순서는 완료된 태스크의 순서와 반대로 이루어 진다. 이것은 4), 5) 번 그림에 나와 있듯이 begin-on-commit dependency (WCD)에 의해서 표현된다.

여기서는 모든 태스크가 복원 가능하다고 가정한다. 그러나 현실 세계에서는 이것이 항상 가능한 것은 아니다. 무엇인가를 물리적으로 변경하거나 상태를 바꾸는 경우에는 원래의 상태로 되돌리는 것은 불가능하다. 예를 들어 문서에 도장을 찍거나 우편을 부치거나 한 것은 다시 원상태로 되돌릴 수 없다. 또한 워크플로우 시스템이 응용 시스템을 부르는 경우에도 복원 작업을 하기가 상당히 힘들다. 그래서 본 논문에서 고려한 복원은 워크플로우 데이타베이스 내의 태스크 상태를 복원하는 것에 한정하였다. 그러나 복원이 불가능한 태스크인 경우에는 워크플로우 시스템은 복원되어야 한다는 내용을 응용 시스템과 그 수행자에게 알려주는 일을 해야 한다.

블록에 대한 복원은 조금 더 복잡하다. 왜냐하면 루트노드를 제외한 모든 블록은 다른 블록에 내포되어 있기 때문이다. 내포된 블록의 복원태스크는 재귀적으로 구성요소에 대한 복원을 부른다. 예를 들어 AND-병렬블록 A가 구성요소 {a1, a2, B, a3}을 가지고 블록 B는 serial block으로서 구성요소 {b1, b2}을 가진다고 하자. 현재 a1, B가 COMMITTED 상태이고 a2는 EXECUTING이고 a3는 READY 상태인데, A가 중단되었다고 해보자. 그러면 구성요소 a1과 B의 복원태스크가 시작되어서 a1과 B를 중단시켜서 a1과 B의 상태는 ABORTED로 바뀐다. 동시에 COMMITTED 상태인 B의 구성요소 b1과 b2의 복원태스크가 동작하여 b1과 b2의 상태를 ABORTED로 바꾸게 된다.

5.3 ECA 규칙의 유도

ACTA 형식론으로 나타내어진 의존관계에서 각 블록 유형별로 ECA 규칙이 만들어진다. ECA 규칙은 사건(event), 조건(condition), 처리(action)으로 구성되어 있다. 규칙을 생성하는 것은 이와 같은 세가지 요소를 찾는 것과 같다. ECA 규칙은 DBMS의 내장 프로시져로 구현되는데, 그림 8은 Oracle 8i로 구현된 ECA 규칙이다. 일종의 프로그램인 내장 프로시져는 ECA 규칙과 같이 사건, 조건, 처리의 세가지 구성요소로 이루

Event
```
CREATE OR REPLACE TRIGGER Commit_block_by_all_tasks
AFTER UPDATE OF ComponentStatus ON WF_TASKINST
FOR EACH ROW
```

Condition
```
WHEN (new.BlockType = 'AND-parallel') AND (new.ComponentStatus = 'Committed')
```

Action
```
DECLARE
    noOfComponent       NUMBER(3);
    noOfCommitted       NUMBER(3);
    blockinst_var       WF_BLOCKINST%rowtype;
BEGIN
    blockinst_var.ProcessID := :new.ProcessID;
    blockinst_var.ProcessInstID := :new.ProcessInstID;
    SELECT MAX(ComponentSequence) INTO noOfComponent FROM WF_BLOCK
        WHERE   ProcessID = :new.ProcessID AND
                BlockID = blockinst_var.BlockID;
                        /* Get the number of block components from WF_BLOCK. */
    SELECT MAX(ComponentSequence) INTO noOfCommitted FROM WF_BLOCKINST
        WHERE   ProcessID = :new.ProcessID AND
                BlockID = blockinst_var.BlockID;
                /* Get the number of committed components from WF_BLOCKINST. */
    IF noOfComponent = noOfCommitted THEN
                                /* Check if those two numbers are equal. */
        UPDATE  WF_BLOCKINST SET BlockStatus = 'Committed'
        WHERE   ProcessID = :new.ProcessID AND
                ProcessInstID = :new.ProcessInstID AND
                TaskID = :new.TaskID;
                        /* Update the status of the current block into 'Committed'. */
    END IF;
END;
```

그림 8 ECA 규칙의 구현 예제 (R15의 경우)

어져 있다. 사건은 "시작시키는 명령문"인데, 내장 프로시져가 자동으로 감지할 수 있어야 한다. 이러한 사건으로는 테이블 명령어로서, insert, delete, update 연산 같은 것이 있다. 조건은 "시작 조건"으로서 처리부분을 시작하기 위해 만족되어야 하는 논리식을 의미한다. 처리는 "호출된 명령문"으로서 실행되어야 할 SQL 명령문이나 프로그램을 의미한다.

그림 8의 ECA 규칙 예제는 R15(commit_block_by_all_tasks)를 구현한 것으로, AND-병렬블록의 CD와 SCD 의존관계를 구현한 것이다. AND-병렬블록에서 하나의 태스크가 완료되면 데이타베이스 테이블(WF_TASKINST)에 그 정보가 기록되고 동시에 이 update 연산이 R15 규칙을 시작시킨다. 이 규칙의 사건 부분은 WF_TASKINST의 ComponentStatus에 대한 update라고 정의되어 있다. 능동형 데이타베이스는 이 사건을 감지하고 조건 부분을 검사한다. 조건 부분은 블록 유형이 AND-병렬블록이어야 하고 갱신된 ComponetStatus가 'Committed'이어야 한다는 것이다. 결국 처리 부분에서 블록의 구성 태스크 중 'Committed' 상태인 것의 갯수가 블록의 구성요소 갯수와 같으면 블록의 상태(BlockStatus)를 'Committed'로 바꾼다. 다른 규칙들도 비슷하게 구현될 수 있고 표 2에 요약되어 있다.

6. 시스템 구현 및 응용사례

본 논문에서는 WFMS 프로토타입을 구현하였다. 그림 9는 그 시스템의 간단한 구조를 나타낸다. 능동형 데이타베이스는 종래의 워크플로우 엔진의 역할을 수행하고 있다. 내장 프로시져를 가지고 있는 테이블이 능동형 데이타베이스에 저장되어 있어서, 이러한 테이블에 발생하는 사건을 감지하여 블록 유형과 태스크 상태를 고려한 관련 규칙을 찾아서 시작시킨다. 특정 프로세스를 실행시킬 때, 데이타베이스는 외부 사건 관리자를 통하여 사용자 및 응용 프로그램과 통신한다. 이때 사용하는 외부 사건은 아래와 같다.

- START(Process Instance): 클라이언트가 프로세스 인스턴스를 시작시킬 때 발생하는 사건
- DELIVER(Task): 단위 태스크를 사용자나 프로그램과 같은 업무 수행자에게 할당하는 사건
- SUCCESS(Task): 클라이언트가 태스크를 성공적으로 완료한 것을 나타내는 사건
- FAIL(Task): 사용자나 응용 프로그램이 태스크를 완료하지 못한 것을 나타내는 사건

본 시스템에서 규칙 처리와 관련된 테이블은 WF_EVENT, WF_BLOCKINST, WF_TASKINST의 세 개이다. WF_EVENT는 클라이언트가 발생시키는 외부 사건 정보를 저장하는 테이블이고, WF_BLOCKINST

표 2 블록 종류(패턴)별 ECA 규칙

Block type		Dependency	Event	Condition	Action	ID
Common		EXTERNAL	START(PI)		Begin B	R1
		EXTERNAL	SUCCESS(t_i)	Begin $t_i \in H$	Commit t_i	R2
		EXTERNAL	FAIL(t_i)	Begin $t_i \in H$, Commit $t_i \notin H$	Abort t_i	R3
		INTERNAL	Begin t_i	$t_i = B$	Begin B	R4
		INTERNAL	Commit B	$B = t_i$	Commit t_i	R5
		t_i WAD B	Abort B	Abort $B \in H$, Commit $t_i \notin H$	Abort t_i	R11
		ct_i BAD B ct_i BCD t_i	Abort B	Abort $B \in H$, Commit $t_i \in H$	Begin ct_i	R13
Serial		t_1 BD B	Begin B	Begin $B \in H$	Begin t_1	R6
		B AD t_i	Abort t_i	Abort $t_i \in H$	Abort B	R9
		t_{i+1} BCD t_i	Commit t_i	Commit $t_i \in H$	Begin t_{i+1}	R12
		ct_i WCD ct_{i+1} ct_i BCD t_i	Commit ct_{i+1}	Commit $ct_{i+1} \in H$, Commit $t_i \in H$, Abort $B \in H$	Begin ct_i	R14
		B SCD t_n	Commit t_n	Commit $t_n \in H$	Commit B	R16
Parallel	AND-parallel	t_i BD B	Begin B	Begin $B \in H$	Begin t_i	R7
		B CD t_i B SCD t_n	Commit t_i	$\forall i$ (i=1,2,...,n), Commit $t_i \in H$	Commit B	R15
		B AD t_i	Abort t_i	Abort $t_i \in H$	Abort B	R9
		ct_i WCD ct_{i+1} ct_i BCD t_i	Commit ct_{i+1}	Commit $ct_{i+1} \in H$, Commit $t_i \in H$, Abort $B \in H$	Begin ct_i	R14
	OR-parallel & COR-parallel	t_i BD B	Begin B	Begin $B \in H$	Begin t_i	R7
		B AD t_i	Abort t_i	$\forall j$ (j =1, ..., n), Abort $t_j \in H$	Abort B	R10
		B SCD t_i	Commit t_i	Commit $t_i \in H$	Commit B	R16
		t_i BD(c_j) B	Begin B	Begin $B \in H$, c_j=True	Begin t_i	R8
	POR-parallel	t_1 BD B	Begin B	Begin $B \in H$	Begin t_1	R6
		t_{i+1} BAD t_i	Abort t_i	Abort $t_i \in H$	Begin t_{i+1}	R18
		B SCD t_i	Commit t_i	Commit $t_i \in H$	Commit B	R16
		B AD t_n	Abort t_n	$\forall j$ (j=1,...,n), Abort $t_j \in H$	Abort B	R10
Iterative		t_1 BD B	Begin B	Begin $B \in H$	Begin t_1	R6
		t_{i+1} BCD t_i	Commit t_i	Commit $t_i \in H$	Begin t_{i+1}	R12
		B AD t_i	Abort t_i	Abort $t_i \in H$	Abort B	R9
		B SCD(c) t_n	Commit t_n	Commit $t_n \in H$, c= True	Commit B	R17
		t_1 $BCD(\neg c)$ t_n	Commit t_n	Commit $t_n \in H$, c= False	Begin t_1	R19

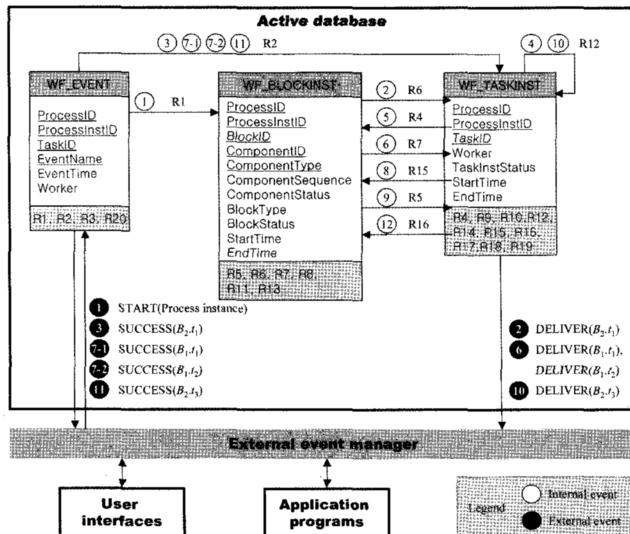


그림 9 WFMS 프로토타입 시스템 구조와 ECA규칙 적용 순서

는 진행중인 블록의 상태 정보를 저장하며, WF_TASKINST는 각 태스크의 정보 및 상태를 저장하고 있다. 규칙을 처리하는 동안 이러한 테이블은 서로 상호작용한다. 앞 부분의 ECA 규칙 예제에서 처리부분은 WF_BLOCKINST를 변경시킨다. 이것은 다시 이 테이블에 정의된 다른 규칙을 시작시킨다. 이와 같이 사건 발생 탐지와 연속적인 규칙 실행이 프로세스를 실행시킨다. 각 테이블 마다 필요한 ECA 규칙은 그림 9의 개별 테이블 하단에 열거되어 있다.

또한 그림 9는 그림 10(a)에 있는 예제에 규칙을 적용하는 순서를 보여준다. 이 예제는 본 논문에서 제시된 접근 방법이 어떻게 작동하는지를 보여준다. 블록 탐색 알고리즘을 적용하여 하나의 병렬블록과 하나의 직렬블록을 찾아낸 다음 네트워크는 그림 10(b)와 같은 트리 형태로 변환된다. 이러한 프로세스 모델이 데이타베이스에 저장되고 나면, 그 정의에 해당하는 프로세스 인스턴스를 실행하고자 하는 권한 있는 사용자가 사용할 수 있다.

프로세스 인스턴스를 시작하고 나면, 먼저 트리의 최상위 노드(B2)를 선택한다. 이 노드는 직렬블록이므로 직렬블록에 해당하는 규칙이 적용된다. 이 직렬블록은 (c)와 같이 B2.t1, B2.t2, B2.t3 (각각 T1, B1, T4에 대응)로 구성되어 있다. 이 블록을 실행하면서 (d)와 같이 다른 블록 B1을 만나면 B2.t1, B2.t2(각각 T2, T3에 대응)를 실행하기 위해 AND-병렬블록에 대한 규칙이 실행된다.

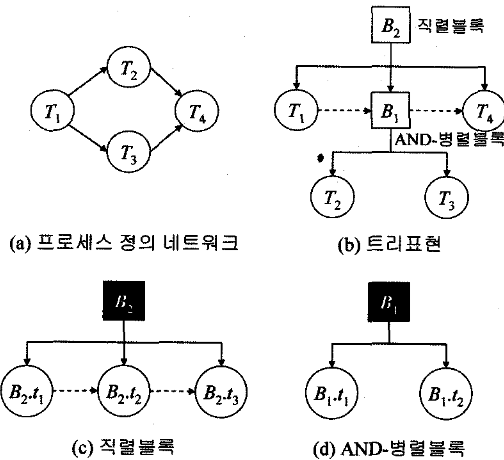


그림 10 규칙 적용 예제 프로세스

7. 결 론

본 논문에서는 능동형 데이타베이스를 이용하여 비즈니스 프로세스를 실행시킬 수 있도록 ECA 규칙에 바탕을 둔 WFMS를 제안하였다. WFMS에서 ECA 규칙을 사용하는 기존의 접근법은 예외사항 처리를 위한 것이나 특정 업무를 수행하는 규칙이 대다수였다. 이것은 기존의 접근법들이 프로세스 통제를 위한 일반적인 방법으로 사용되지 않았다는 의미이다. 본 논문에서 제시한 방법은 기존의 워크플로우 엔진을 전적으로 대체할 수 있다. 이러한 규칙 기반의 워크플로우 실행을 통하여 기존의 유방향 그래프로 설계된 프로세스의 실행을 명확하게 하였고, 기존의 그래프 기반의 프로세스에 적용하기 힘든 구역 기반의 트랜잭션 관리 기법을 ACTA 형식론을 이용하여 적용하는 방법을 제안하였다.

본 논문의 독창적인 내용은 다음과 같다. 첫째, 프로세스 흐름을 몇 개의 패턴으로 나누어 주는 블록의 구조적 적용을 제안하였다. 블록은 프로세스 모델을 나타내는 기본적인 단위이며 ECA규칙을 찾는 도구이다. 둘째, 프로세스 정의 네트워크에서 블록을 찾아내는 알고리즘을 개발하였다. 셋째, 블록을 이용하여 일차원적인 네트워크를 계층적인 트리 형태로 변환하였다. 이것은 프로세스를 통제할 때에 모듈화를 할 수 있도록 도와주었다. 마지막으로 각 블록 유형별로 ACTA 형식론을 이용하여 통제 논리를 설계하였다. 이것은 ECA 규칙의 이론적인 근거가 되었다. 전체적으로 본 논문에서 제시한 방법은 규칙기반 WFMS의 기초가 될 수 있다. 최근의 DBMS들은 능동형 규칙(active rule)을 트리거 혹은 내장 프로시져 방식으로 지원하므로 이 방법론은 현재 대다수의 DBMS에 설치되는 데 무리가 없다.

추후 연구과제로는 다음과 같은 것이 있다. 첫째, 현재 접근법에서 제시한 규칙은 프로세스 모델의 구조적인 측면만을 다루고 있지만 향후 업무 영역별로 독특한 규칙을 추가하여 확장할 수 있다. 둘째, 블록 유형을 더욱 일반화 시킬 수 있다. 예를 들어 병렬패턴에 있는 하나의 태스크가 다른 병렬패턴에 연결되는 등의 일반적 네트워크 문제를 고려할 수 있다. 셋째, 제시된 방법의 효율성을 측정하는 문제로서 동시에 몇 개의 워크플로우가 실행되면서 통제될 수 있는지를 평가하는 연구가 필요하다.

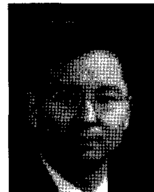
참 고 문 헌

[1] W.M.P. van der Aalst, "Process-Oriented Architectures for Electronic Commerce and Interorganizational Workflow," Information Systems, vol. 24, no. 8, pp. 639-671, Dec. 1999.

[2] G. Mentzas, C. Halaris, and S. Kavadias, "Modelling Business Processes with Workflow systems: An Evaluation of Alternative Approaches," Int'l J. Information Management, vol. 21, no. 2, pp. 123-135, Apr. 2001.

[3] G. Shegalov, M. Gillmann, and G. Weikum, "XML-enabled workflow management for e-services across heterogeneous platforms," The VLDB Journal, vol. 10, pp. 91-103, 2001.
[4] D. Hollingsworth, "The Workflow Reference Model," Technical Report WFMC-TC-1003, 1.1, Workflow Management Coalition, Brussels, 1994.
[5] A. Dogac, E. Gokkoca, S. Arpinar, P. Koksal, I. Cingil, B. Arpinar, N. Tatbul, P. Karagoz, U. Halici, and M. Altinel, "Design and Implementation of a Distributed Workflow Management System: METUFlow," Proc. NATO Advanced Study Institute (ASI) Workshop on Workflow Management Systems and Interoperability, pp. 61-66, Aug. 1997.
[6] M. Rusinkiewicz and A. Sheth, "Specification and Execution of Transactional Workflows," Modern Database Systems: The Object Model, Interoperability, and Beyond, Addison-Wesley, New York, pp. 592-620, 1995.
[7] P.C. Benjamin, C. Marshal, and R.J. Mayer, A Workflow Analysis and Design Environment (WADE), 1999.
[8] A.M.A. Al-Ahmari and K. Ridgway, "An Integrated Modeling Method to Support Manufacturing Systems Analysis and Design," Computers in Industry, vol. 38, no. 3, pp. 225-238, Apr. 1999.
[9] J. Miller, D. Palaniswami, A. Sheth, K. Kochut, and H. Singh, "WebWork: METEOR's Web-based Workflow Management System," J. Intelligent Information Systems, vol. 10, no. 2, pp. 185-215, 1998.
[10] A. Kumar and J.L. Zhao, "Dynamic Routing and Operational Controls in Workflow Management Systems," Management Science, vol. 45, no. 2, pp. 253-272, 1999.
[11] C. Schlenoff, A. Knutilla, and S. Ray, Unified Process Specification Language: Requirements for Modeling Process, NIST, 1996.
[12] T.H. Davenport, Process Innovation: Reengineering Work through Information Technology. Harvard Business School Press, Boston, MA, 1993.
[13] R. Shapiro, "A Comparison of XPDL, BPML and BPEL4WS," Cape Visions, 2002.
[14] A.D. Lucia, R. Francese, G. Tortora, "Deriving workflow enactment rules from UML activity diagrams: a case study," Proc. IEEE Symp. on Human Centric Computing Languages and Environments, 2003, pp.211-218, 2003.
[15] G. Kappel, S. Rausch-Schott, W. Retschitzegger, "Coordination in Workflow Management Systems -A Rule-Based Approach," Lecture Notes in Computer Science, No.1364, pp.99-120, 1998.
[16] J. Meng, S.Y.W. Su, H.Lam, A. Helal, "Achieving dynamic inter-organizational workflow management by integrating business processes, events and rules," Proc. 35th Hawaii Int'l Conf. System Sciences, pp.10-, 2002.
[17] N. H. Gehani, H. V. Jagadish, O. Shmueli, "Event specification in an active object-oriented database," Proc. ACM SIGMOD Int'l Conf. Management of Data, San Diego, CA, pp.81-90, 1992.
[18] A. Goh, Y.-K. Koh, D.S. Domazet, "ECA rule-based support for workflows," Artificial Intelligence in Engineering, vol.15, no.1, pp.37-46, 2001.
[19] W.M.P. van der Aalst and A.H.M. ter Hofstede, "Verification of Workflow Task Structures: A Petri-Net-Based Approach," Information Systems, vol. 25, no. 1, pp. 43-69, 2000.
[20] C. Hagen and G. Alonso, "Exception Handling in Workflow Management Systems," IEEE Trans. Software Eng., vol. 26, no. 10, pp 943-958, Oct. 2000.
[21] F. Casati, S. Ceri, B. Pernici, and G. Pozzi, "Deriving Active Rules for Workflow Enactment," Proc. 7th Int'l Conf. Database and Expert Systems Applications, pp. 94-110, 1996.
[22] P.K. Chrysanthis and K. Ramamritham, "ACTA: The SAGA Continues," Database Transaction Models for Advanced Applications, Morgan Kaufmann Publisher, San Manteo, Calif., pp. 354-397, 1995.

이 우 기
서울대학교 산업공학과 학사(1987). 서울대학교 대학원 산업공학과 석사(1993) 박사(1996), CMU MSE 과정수료(2000) UBC 교환교수(2002-2003). 1996년~현재 성결대학교 컴퓨터학부 부교수, 한국경영과학회 이론분야 최우수논문상(2004) 관심분야는 DW, IR, Web



배 준 수
서울대학교 산업공학과 학사(1993). 서울대학교 대학원 산업공학과 석사(1995) 박사(2000). 2004년~현재 전북대학교 산업정보시스템공학과 조교수. 관심분야는 Workflow, BPM, Enterprise Transformation, Internet Application

정 재 윤
서울대학교 산업공학과 학사(1999), 서울대학교 대학원 산업공학과 석사(2001), 박사(2005). 현재 서울대학교 자동화시스템연구소 연구원, 관심분야는 BPM, 전자거래, 기업정보시스템