시간 제약을 포함한 워크플로우 모델링 및 검증

정 희 택[†] · 이 도 헌[††] · 김 문 자[†††] · 류 영 철[††††]

요 약

최근에 자동화된 업무처리 시스템으로써, 워크플로우 시스템에 대한 연구가 활발히 이루어지고 있다. 기존 연구에서는 태스크 자체에 시간적 특성을 고려하지 않고 태스크의 상태 변이 특성만을 고려하였다. 본 연구는 태스크의 최소 처리시간과 마감시간을 고려함으로써, 시간 제약을 포함한 워크플로우 모델링 및 검증방안을 제안한다. 이를 위해 첫째, 마감시간을 위배하지 않는 태스크의 철회에 대해 대안태스크를 결정할 수 있는 우선 순위 함수를 제안한다. 우선 순위 함수는 대안태스크에 대한 시간요소, 의미적 호환 수준을 고려함으로써, 가장 적절한 대안태스크를 결정한다. 둘째, 기존 연구에서는 순차 및 병행 종속성만을 고려하였으나, 본 연구에서는 태스크간 종속성을 순차 종속성이외에 다양한 병행 종속성인 동시수행 종속성, 동시시작 종속성, 동시종료 종속성, 간섭된 종속성, 그리고 중첩된 종속성으로 구분하여 제안한다. 셋째, 기술된 워크플로우에 존재할 수 있는 논리적 모순을 발견하기 위해 그래프를 이용한 검증기법을 제안한다.

Modeling and Verification of Workflows with Time Constraints

Hee-Taek Ceong[†] · Do-Heon Lee[††] · Moon-Ja Kim[†††] · Young-Chul Lew[††††]

ABSTRACT

Recently, automated business processing systems, so called workflow systems, have been studied. Even though each task has noticible duration inherently, most of previous models regard it as a simple finite automaton where state transitions occur instantly. In this paper, we propose a modeling method for workflows each of whose component task has minimum processing time and due-date. Firstly, we provide a priority function to choose the proper alternatives when tasks fail before their due-date. The priority function considers the time-relevant factor and the semantic compatibility level to decide the best alternative. Secondly, we identify six different types of execution requirements for tasks with noticible durations. They are serial, parallel, simultaneous start, simultaneous termination, interleaved and nested dependency. Note that previous models deal with only serial execution and parallel execution. Lastly, we also propose graph-based methods to detect logical conflicts in time constraint-based workflow specifications.

1. 서 론

최근 일반 기업에서는 조직 규모의 복잡화와 방대화에 따라 자동화된 업무처리의 요구가 날로 증가하고 있다. 또한, 업무처리 환경의 급격한 변화는 유연성 있는 업무처리 시스템을 요구하고 있다. 그러나, 기존에 자료 처리 시스템은 응용처리 프로그램 자체에 제어 구조를 내포함으로써 환경변화에 적절히 대처할 수 없다. 이러한 배경에 대해 업무흐름 정의와 이들의 수행을 구분한 워크플로우 관리 시스템에 대한 연구가 활발히 이루어지고 있다[1,2,3]. 워크플로우란 자동화된

※ 본 연구는 한국학술진흥재단 '97 신진연구인력 연구장려금과 한국전자통신연구원 '98 위탁과제 연구비에 의해 수행되었음.
† 정 회 원 : 전남대학교 대학원 전산통계학과
†† 정 회 원 : 전남대학교 전산학과 교수
††† 정 회 원 : 한국전자통신연구원
†††† 종신회원 : 한국전자통신연구원
논문접수 : 1998년 4월 9일, 심사완료 : 1998년 11월 23일

업무흐름을 의미한다[1,3]. 조직의 업무 흐름은 특정 목적을 달성하기 위한 태스크(task)들로 구성된다. 태스크는 특정 업무를 수행하기 위한 일련의 연산들로 구성되며 이질적인 처리 개체들(processing entities)에 의해 수행된다. 이와 같은 워크플로우를 컴퓨터와 통신망을 이용하여 체계적으로 관리하는 시스템을 워크플로우 관리 시스템이라 한다[3,4,5].

워크플로우 전체 혹은 워크플로우를 구성하는 태스크들은 최소 소요시간, 마감시간과 같은 시간 제약을 갖는 것이 일반적이다. 예를 들면, 병원에서 환자에 대한 의료서비스 워크플로우를 고려할 수 있다. 워크플로우는 환자의 접수에 의해 시작된다. 접수된 정보를 바탕으로 의사에 의해 진료 태스크가 수행된다. 진료에 따른 검사 태스크로써 임상병리 및 CT(computed tomogram) 검사를 가정할 때, 환자에 대한 임상병리 검사를 시작한 후 CT 검사를 시작한다. 또한, 수술실, 수술일, 그리고 수술 팀을 결정하는 수술일정 태스크를 위해, 해당 검사 태스크들은 지정된 시간 내에 종료되어야 한다. 환자에 대한 수술은 지정된 시간에 시작 및 완료되어야 한다. 또한 수술 범위를 결정하는 동결 조직 검사는 수술 시작 후 시작되고 종료 전에 완료되어야 한다. 제시된 예와 같이, 워크플로우는 다양한 시간 제약 요소를 포함한다.

워크플로우 모델링을 위한 대부분의 기존 연구는, 태스크를 순간적인 상태의 이행(transition)으로 표현하고 있다. 즉, 태스크의 상태를 시작, 완료, 그리고 철회의 3가지 사건으로서 정의하고 태스크간 종속성을 사건들간의 종속성으로 기술하였다[6,7,8,9,10,11]. [8,9]에서는 3가지 사건를 기반으로 특정 태스크의 완료가 다른 태스크의 완료에 의존하는 완료 종속성과 특정 태스크의 철회가 다른 태스크의 철회에 의존하는 철회 종속성을 제시하였다. 또한, [10,11]에서는 3가지 사건들로 기술되는 활성화조건(enabling condition)을 제시함으로써 태스크간 종속성을 검사하고 강요할 수 있는 방안을 제안하였다. 그러나, 사건을 기반으로 이루어진 워크플로우 모델링은 시간 제약적 특성을 고려하지 않음으로써, 시간 제약을 포함한 태스크 기술 및 태스크간 종속성을 기술할 수 없다. 예를 들면, 임상병리 및 CT 검사 태스크가 지정된 시간 제약 내에 완료되어야 한다는 시간 제약을 표현할 수 없다.

한편, 부분적 시간 특성을 고려한 연구로서 [12]에서는 태스크에 대한 종료일(expiration date)과 완료일(commit date)을 제안함으로써 해당 시간에 태스크의 묵시적(implicitly) 종료 및 완료를 가정할 수 있는 방안을 제안하였다. 한편, [13]에서는 각 트랜잭션에 대한 예상 종료 시간(value date)을 정의하고, 그 시간들간에 관련성을 시간 종속성(chronological dependency)으로 정의하였으며, 그리고 기술된 종속성을 강요하기 위한 스케줄 방안을 제안하였다. 태스크의 마감시간만을 고려한 [14]에서는, 마감시간 위배에 따른 철회 비용(escalation cost) 최소화를 달성하기 위해서, 처리 개체의 통계적 정보를 이용하여 마감시간 만족 여부를 예측할 수 있는 방안을 제안하였다. 제안된 연구들은 태스크의 종료 및 완료시간, 또는 마감시간만을 고려하였을 뿐만 아니라 제시한 시간요소들 간의 관련성만을 고려함으로써 다양한 종속성을 표현할 수 없다. 예를 들면, 수술 태스크 수행 중에 시작 및 완료되어야 하는 동결조직 검사 태스크를 표현할 수 없다.

사건을 기반으로 한 연구들과 부분적 시간 특성을 고려한 기존 연구들은, 예로 제시한 바와 같이 워크플로우를 구성하는 시간 제약적 요소를 표현하지 못하며 태스크간 다양한 종속성을 표현할 수 없다. 이러한 이유로 본 연구에서는 워크플로우의 시간제약 특성을 분석하고 이를 기반으로 모델링 방안을 제안한다. 먼저, 워크플로우의 시간 제약적 특성을 모델링하기 위해, 태스크에 내재된 시간 제약적 특성을 분석하고 이를 기반으로 시간 제약적 제어흐름 모델링 방안을 제시한다. 제안된 시간 제약적 제어흐름은 태스크간에 존재할 수 있는 다양한 병행 종속성 요구를 고려한다. 다음으로, 시간 제약적 제어흐름을 포함한 워크플로우에 존재할 수 있는 모순을 정의하고 검증한다. 이는 모델링 된 결과가 요구사항을 모순 없이 포함하고 있고 수행 가능함을 보장하기 위한 필수 요소이다.

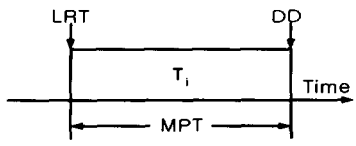
본 연구에서는 시간 제약적 특성을 고려한 워크플로우 모델링 방안 및 일관성 검증기법을 제안한다. 먼저, 태스크에 내재된 시간 제약적 특성과 태스크들간에 존재하는 시간 제약적 제어흐름을 모델링 하기 위한 방안을 2장에서 제시한다. 3장에서는 시간 제약을 포함한 워크플로우에 존재할 수 있는 모순을 정의하고 일관성 검증 방안을 제시한다. 4장에서는 본 연구에 대한 결론을 기술한다.

2. 워크플로우에서 시간제약 모델링

워크플로우는 태스크들로 구성된다. 본 장에서는 먼저, 각 태스크에 내재된 시간 제약을 제시한다. 다음으로, 우선 순위 함수를 제안함으로써, 시간 제약 내에 철회된 태스크에 대한 대안태스크 결정 방안을 제안한다. 마지막으로, 시간 제약적 특성에 근거한 태스크간 종속성들을 구분하고 이를 모델링하기 위한 방안을 제안한다.

2.1 태스크 모델링

시간 종속적 워크플로우에서, 각 태스크는 워크플로우 시작을 기준으로 상대적인 마감시간(due-date)을 갖는다. 또한, 태스크는 처리 개체에 의해 수행되기 때문에 최소 처리시간(minimum processing time)을 갖는다. 이러한 태스크가 갖는 시간 제약은 (그림 1)과 같다. 가장 늦은 준비시간(the latest ready time)은 마감시간을 만족하기 위해 태스크가 최소한 시작되어야 하는 시간으로써, 마감시간과 최소 처리시간에 의해 간단히 계산된다.



LRT : Latest Ready Time(DD - MPT)
DD : Due-Date
MPT : Minimum Processing Time

(그림 1) 시간 제약을 포함한 태스크
(Fig. 1) A task with time constraints

제시된 시간 제약을 포함한 태스크의 수행은 항상 완료될 수 없다. 즉, 처리 개체의 고장과 같은 다양한 이유로 철회될 수 있다. 태스크의 종료는 마감시간과 철회에 의해 3가지 형태로 구분된다. 첫째, 태스크는 마감시간을 위배하지 않고 완료된다. 둘째, 마감시간 전에 처리개체에 의해 태스크가 철회된다. 마지막으로, 마감시간을 초과함으로써 태스크는 철회된다. 태스크의 철회는 워크플로우 전체의 철회를 요구한다. 장기 수행특성을 갖는 워크플로우 특성을 고려할 때, 태스크의 철회에 의한 전체 워크플로우의 철회는 비효율적이다. 예를 들면, CT 검사 태스크의 철회에 의해, 이미 완료된 환자 접수 및 진료 태스크를 취소해야 하고 환자 접수 태스크부터 다시 시작해야 한다. 이러한 비효율성을 최소화하는 방안으로써 대안태스크를 고려할 수 있다. 대안태스크를 수행함으로써, 워크플로우 철회에 필요한 자원의 낭비를 최소화할 수 있다. 이는 [6,7]에서 고려한 보상 태스크에 의한 회복비용을 회피할 수 있다. 대안 태스크는 전진 회복(forward recovery) 방안이 된다.

본 연구는, 마감시간을 초과하지 않는 태스크의 철회에 대한 대안태스크를 고려한다. 철회된 태스크에 대한 대안태스크를 결정하기 위해 두 가지 요소를 고려한다. 첫째, 철회된 태스크의 시간제약과 대안태스크의 시간 요소를 고려한다. 둘째, 철회된 태스크와 대안태스크간의 의미적 호환성을 고려한다. 제시된 두 가지 요소를 종합적으로 고려함으로써 가장 최적의 대안태스크를 결정 할 수 있는 우선 순위 함수를 제안한다.

첫째, 철회된 태스크의 시간제약(즉, 마감시간)을 보장할 수 있는 대안태스크를 선택하기 위해 시간 요소를 고려한다. 이는 대안태스크를 수행시킬 때, 선택된 대안태스크가 철회된 태스크의 시간제약을 보장하도록 하기 위함이다. 대안태스크의 최소처리시간과 철회된 태스크의 철회 시간을 고려할 때, 본래 수행되어야 할 태스크의 철회는 재시도가능 철회(retriable abort)와 재시도불가능 철회(non-retriable abort)로 구분된다. 전자는 마감시간을 초과하지 않고 대안태스크를 수행할 수 있는 태스크의 철회이다. 후자는 마감시간을 위배하지 않았으나 대안태스크를 수행할 수 없는 철회이다. 임의의 태스크 T_i에 대한 대안태스크 T_{i_1}, T_{i_2}, ..., T_{i_n}가 존재할 때, T_i의 철회는 시간간격에 의해 재시도가능 및 재시도불가능 철회로 구분된다. [t(Start$_i$),DD$_i$-Min (MPT_{i_1}, MPT_{i_2}, ..., MPT_{i_n})]에서 발생한 태스크의 철회는 재시도가능 철회이다. 한편, (DD$_i$-Min (MPT_{i_1}, MPT_{i_2}, ..., MPT_{i_n}), DD$_i$]에서 발생한 철회는 재시도불가능 철회이다. 여기서 t(Start$_i$)는 T_i의 시작시간, DD$_i$는 T_i의 마감시간, 그리고 MPT_{i_j}는 대안태스크 T_{i_j}의 최소처리 시간이다. 재시도가능 철회에 따른 대안 태스크를 고려할 때, 가장 짧은 수행시간을 갖는 대안 태스크를 선택해야 한다. 이는, 긴 수행시간을 요하는 대안태스크의 수행 및 철회로 원래 태스크의 마감시간을 만족하지 못할 수 있기 때문이다. 즉, 가장 짧은 수행시간을 갖는 대안태스크를 선택해야 한다.

둘째, 철회된 태스크와 대안태스크의 의미적 호환성을 고려해야 한다. 이는, 철회된 태스크에 대해 임의의 태스크가 대신할 수 없기 때문이다. 의미적 호환성은 철회된 태스크와 대안태스크들 사이의 의미적 호환정도로써, 이를 기반으로 철회된 태스크 목적에 가장 부합하는 대안작업을 수행할 수 있다. 예를 들면, CT 검사 태스크가 철회되었을 때, CT 검사결과에 가장 근접한 검사결과를 보장하는 대안태스크의 수행을 의미한다. 의미적 호환성을 정의하기 위해, 퍼지 집합 이론[15]을 기반으로 의미적 호환 관계를 정의한다. 의미적 호환관계에서 퍼지 집합이론의 적용은, 철회된 태스크와 대안태스크들의 다양한 대치 수준을 표현할 수 있다. 이는 기존 연구[5,8,10]에서처럼 철회된 태스크를 대안 태스크로 대치할 수 있는지 여부를 이분론적으로 규정하는 것보다, 철회된 태스크의 본래 의도를 대안 태스크가 얼마나 충분히 만족시키는지를 정량적으로 명세함으로써 보다 정확한 모델링을 가능하게 한다.

정의 1. 의미적 호환 관계(Semantic Compatibility Relation : SCR)

태스크들의 정의 구역 D와 호환 정도 L에 의해 의미적 호환 관계 SCR은 다음과 같다.

$$SCR \subseteq D \times D \times L$$

호환정도 $L = \{ L_l, ..., L_m \}$의 집합은 전체 순서 집합(totally ordered set)이며, 하한(low bound) L_l과 상한(upper bound) L_m은 가장 큰 비호환성과 가장 큰 호환성을 나타낸다. 중간 정도인 L_2부터 L_{m-1}은 부분적인 호환정도를 나타낸다. 이때 의미적 호환 관계는 다음 두 가지 속성을 갖는다.

- 반사성(reflexivity) : $\forall T_i, < T_i, T_i, L_m > \in SCR$
- 최대-최소 이행성(Max-Min transitivity) :
  $\forall T_i, \forall T_j, \forall T_k$
  $<T_i, T_j, L_P > \in SCR \wedge <T_j, T_k, L_Q > \in SCR$
  $\rightarrow <T_i, T_k, Max\text{-}Min(L_P, L_Q)> \in SCR.$

최대-최소 이행성에서, 먼저, T_i와 T_k간의 호환정도는 T_i와 T_j의 호환정도 L_P를 보장하고 T_j와 T_k간의 호환정도 L_Q를 보장해야 하는 논리곱적 성격을 갖는다. 정량적 호환정도에 대한 논리곱 연산은 퍼지이론에서 최소(Min)연산으로 정의되었다[15]. 즉, T_i과 T_k간의 의미적 호환정도는 $Min(L_P, L_Q)$에 의해 결정된다. 한편, T_i과 T_k간에 또 다른 이행적 특성 $<T_i, T_{j'}, L_{P'}> \in SCR \wedge <T_{j'}, T_k, L_{Q'}> \in SCR$을 갖을 때, 동일한 논리곱적 성격에 의해 T_i과 T_k간의 의미적 호환정도는 $Min(L_P, L_{Q'})$이다. 다음으로, T_i와 T_k간에 둘 이상의 이행적 특성이 존재할 때, 하나의 이행적 특성으로도 T_i과 T_k간의 호환정도를 결정할 수 있다. 즉, 논리합적 성격을 갖는다. 정량적 호환정도에서 논리합 연산은 퍼지이론에서 최대(Max)연산으로 정의되었다[15]. 즉, T_i과 T_k간의 호환정도는 $Max(Min(L_P, L_Q), Min(L_{P'}, L_{Q'}))$이다. 제시된 논리합 및 논리곱적 특성에 의해 의미적 호환관계는 최대-최소 이행성을 갖는다.

제시한 두 가지 특성을 반영하여 마감시간 내에 재시도가능하고 의미적으로 가장 높은 호환 정도를 갖으며 완료를 보장할 수 있는 대안태스크를 결정해야 한다. 이를 다음과 같은 대안태스크 우선 순위 함수로써 결정한다.

정의 2. 대안태스크 우선 순위 함수(Alternative Task Priority Function)

임의의 태스크 T_i의 철회 시간 $-t(abort_i)$-에서 대안태스크 T_{ij}의 우선 순위를 결정하기 위한 함수 $ATPF_{i_j}$는 다음과 같다.

$$ATPF_{i_j} = \omega_1 RTF_{i_j} + \omega_2 SCR_{i \cdot i_j}$$

여기서, ω는 가중치(weight)로서 $\omega_1+\omega_2=1$이다. RTF_{i_j}(Retriable Time Factor)는 T_{ij}가 수행 가능한 시간요소로서 재시도가능 시간 요소이다. MPT_{i_j}는 T_{ij}의 최소 처리 시간, DD_i는 T_i의 마감시간이라 할 때,

$$RTF_{i_j} = \begin{cases} 1 - \dfrac{MPT_{i_j}}{DD_i - t(abort_i)} & MPT_{i_j} \le DD_i - t(abort_i) \\ -\infty & MPT_{i_j} > DD_i - t(abort_i) \end{cases}$$

$SCR_{i \cdot i_j}$는 의미적 호환관계 $< T_i, T_{i_j}, L_k>$에서의 미적 호환정도 L_k 값.

우선 순위 함수에서 시간요소 가중치(ω_1)는 원래 태스크의 시간제약보장에 더욱 중점을 두고 있음을 표현할 수 있다. 즉, 상대적으로 의미적 호환성은 부족하더라도, 시간제약 보장을 만족할 수 있는 대안태스크를 결정할 수 있다. 의미적 호환성 가중치(ω_2)는 의미

적 호환성에 중점을 둔 대안태스크를 결정할 수 있다. 즉, 상대적으로 시간제약에 중점을 두지는 못하지만 철회된 태스크에 가장 근접한 대안태스크를 결정할 수 있다.

제시된 시간요소 및 의미적 호환 관계는, 시간제약을 근간으로 하는 본 연구에 필수 요소이다. 그러나, 제시된 요소 이외의 요소는 필요한 응용에 맞게 채용될 수 있다. 예를 들면, 수행비용을 고려하기 위해 대안태스크의 반복성을 고려할 수 있다. 제시된 우선 순위 함수는 시간제약을 갖는 태스크에 대한 대안태스크 결정 기준으로써 의미를 갖는다. 대안태스크 결정 기준으로써 시간요소 및 의미적 호환관계를 고려하였다.

시간 제약적 특성과 최적의 대안태스크를 결정하기 위한 요소를 포함한 태스크의 정의는 다음과 같다.

정의 3. 시간 제약 태스크(Time constrained task)

시간 제약 태스크는 6가지 요소 (ID, Action, MPT, DD, SCR)에 의해 정의된다. 구성하는 요소 중 ID는 태스크 식별자, Action은 태스크가 수행해야 할 연산들의 집합, MPT는 최소 처리 시간, DD는 태스크의 마감 시간, 그리고 SCR은 의미적 호환 관계를 나타낸다.

[예제 1] 앞서 제시된, 병원에서의 의료서비스 워크플로우를 고려한다. 먼저 워크플로우를 구성하는 태스크가 다음과 같고, 태스크의 활동(action)은 간략한 표현을 위해 생략한다. 각 태스크의 마감시간은 환자의 접수시간에 의해 결정되는 상대적 시간이다.

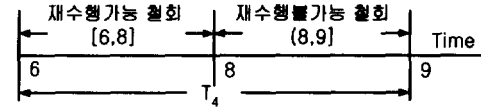
T_1 : (1, 환자 접수 활동, 1, 2, -)
T_2 : (2, 의사 진료 활동, 2, 4, -)
T_3 : (3, 임상병리 검사 활동, 2, 9, -)
T_4 : (4, CT 검사 활동, 3, 9, {<T_4, T_{4_1}, 0.3>})
T_5 : (5, 수술 활동, 4, 13, -)
T_6 : (6, 동결 조직 검사 활동, 1, 12, -)

또한 태스크 T_4의 대안태스크는 다음과 같다.

T_{4_1} : (4_1,X-선 검사 활동,1,-,{< T_{4_1}, T_4, 0.5>, < T_{4_1}, T_{4_2}, 0.4>})

T_{4_2} : (4_2,MRI 검사 활동,1.5,-,{< T_{4_2}, T_{4_1}, 0.2>})

CT 검사 태스크가 철회(7.5) 되었을 때, 두 가지 요소를 고려한다. 첫째, 태스크의 철회가 재시도가능 철회인지 재시도불가능 철회인지 식별한다. CT 검사 태스크 T_4가 [6, 8]사이에 발생했기 때문에 재시도가능 철회이다.



(그림 2) 재시도 및 재시도불가능 철회
(Fig. 2) A retriable and non-retriable abort

둘째, 의미적 호환성을 고려한다. 앞선 기술에서 제시된 T_4와 T_{4_1}간의 호환관계 또는 T_{4_1}와 T_{4_2}간의 호환관계를 기반으로 최대-최소 이행성을 적용함으로써 T_4과 T_{4_2}간의 호환정도를 결정할 수 있다.

$SCR_{4\cdot 4_1}$	T_4	T_{4_1}
T_4	1	0.3
T_{4_1}	0.5	1

$SCR_{4_1\cdot 4_2}$	T_{4_1}	T_{4_2}
T_{4_1}	1	0.4
T_{4_2}	0.2	1

→

SCR_{4_2}	T_4	T_{4_2}
T_4	1	0.4
T_{4_2}	0.5	1

(그림 3) 최대-최소 이행성을 이용한 의미적 호환정도 결정
(Fig. 3) The decision of semantic compatibility relation level using Max-Min transitivity

제시된 두 가지 요소의 우선 순위 함수 적용은 다음과 같다. 의미적 호환 관계에 중점을 둔 가중치 값으로 ω_1= 0.2 그리고 ω_2= 0.8을 가정한다. 이때 다음과 같은 과정을 통해 MRI 검사 태스크 T_{4_2}가 CT 검사 태스크의 철회에 대한 대안태스크로서 결정된다.

$$ATPF_{4_1} = \omega_1 RTF_{4_1} + \omega_2 SCR_{4\cdot 4_1} = 0.30$$

$$ATPF_{4_2} = \omega_1 RTF_{4_2} + \omega_2 SCR_{4\cdot 4_2} = 0.32$$

$$\text{where } RTF_{4_1} = 1 - \frac{1}{9-7.5} = 0.3,.$$

$$RTF_{4_2} = 1 - \frac{1.5}{9-7,5} = 0,$$

$$SCR_{4\cdot 4_1} = 0.3,\ SCR_{4\cdot 4_2} = 0.4$$

2.2 태스크간 종속성 모델링

제시한 태스크 정의를 기반으로 각 태스크간에 존재하는 종속성을 표현해야 한다. 태스크간 종속성은

태스크를 시간 제약을 갖는 인터벌(interval)로 정의함으로써 달성한다. 즉, 시작 시간-t(start)-과 종료시간-t(termination)-을 갖는 인터벌로써 정의한다. 인터벌간 의미 있는 시간 종속 관계는 [16]에서 정의한 시간 관련성을 고려할 수 있다. [16]에서 제안된 시간 관련성은 이미 존재한 데이터에 대한 시간관련성임에 반해, 본 연구는 앞으로 발생할 업무 흐름의 모델링을 위한 종속성 정의가 필요하다. 이런 이유로, [16]에서 제안한 7가지 시간관련성에서 중간 여유시간이 있는 관련성(즉, a before b)을 제외한 6가지 종속성을 고려한다. 한편, 이러한 종속성을 적용함으로써 기존 연구 [4,5,6,7,8,9,10]의 순차 및 병렬 종속성외에 다양한 태스크간 종속성을 모델링 할 수 있다.

제시된 표에 의해 태스크간 종속성을 순차 종속성-SE-과 병행 종속성- PE, SS, ST, IE, 그리고 NE-으로 구분된다. PE는 동시 시작 및 동시 종료의 종속성을 의미하고, SS는 동시 시작 종속성, ST는 동시 종료 종속성, IE는 선행 태스크의 시작 후에 후행 태스크의 시작, 그리고 선행 태스크의 종료 후에 후행 태스크의 종료를 기술하는 간섭된 종속성, 그리고 NE는 IE와 달리 후행 태스크의 종료 후에 선행 태스크의 종료를 기술한다.

제시한 6가지 종속성을 기반으로 워크플로우를 구성하는 태스크간 종속성은 다음 형식을 이용 정의한다.

정의 4. 시간 제약을 포함한 태스크간 종속성 기술형식(Time constrained intertask Dependency Formula : TDF)

TDF는 원자식 이거나 논리곱에 의한 식이다.

① 원자식일 때

$P(T_i, T_j)$

where $P \in \{PE, SS, ST, SE, IE, NE\}$,

$T_i, T_j \in$ 태스크 집합 Ψ.

② 논리곱 식은 재귀적으로 다음과 같이 정의된다.

$F_1 \wedge F_2$

where F_1, F_2는 TDF이다.

[예제 2] 앞서 제시한 의료서비스 워크플로우를 재시된 종속성 기술형식에 의해 기술하면 다음과 같다. 이때, 워크플로우를 구성하는 태스크들은 세 가지 시간적 종속성을 갖는다. 첫째, 임상병리 태스크는 CT 검사 태스크 전에 시작되어야 한다. 둘째, 검사 태스크들은 수술 태스크 전에 종료되어야 한다. 마지막으로, 동결 조직 검사 태스크는 수술 태스크가 수행되는 중에 수행되어야 한다. 이러한 태스크간 종속성을 포함한 워크플로우 정의는 다음과 같다.

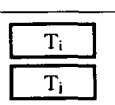
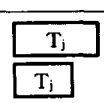
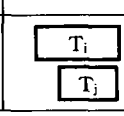
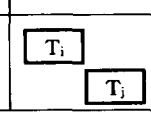
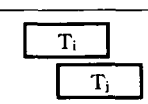
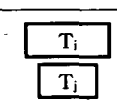
$TDF=SE(T_1, T_2) \wedge SE(T_2, T_3) \wedge ST(T_3, T_4) \wedge SE(T_4, T_5) \wedge NE(T_5, T_6)$.

3. 워크플로우의 일관성 검증

앞서 제시한 태스크간 종속성 요소를 이용하여 기술된 워크플로우는, 임의의 두 태스크간에 요구된 종속성을 정확하게 표현할 수 있다. 그러나, 그러한 종속성 요소들로 구성된 전체 워크플로우가 논리적 모순이 없고 수행 가능함을 가정할 수 없다. 즉, 임의의 두 태스크가 여러 종속성을 통해 갖게되는 종속성들이 의미

<표 1> 태스크간 종속성

<Table 1> Intertask dependencies

	요소 1	요소 2	요소 3	요소 4	요소 5	요소 6
기호	**$PE(T_i, T_j)$: Parallel Execution**	**$SS(T_i, T_j)$: Simultaneous Starts**	**$ST(T_i, T_j)$: Simultaneous Termination**	**$SE(T_i, T_j)$: Serial Execution**	**$IE(T_i, T_j)$: Interleaved Execution**	**$NE(T_i, T_j)$: Nested Execution**
의미	$t(start_i)=t(start_j)$ and $t(termination_i)=t(termination_j)$	$t(start_i)=t(start_j)$ and $t(termination_i)<t(termination_j)$	$t(start_i)<t(start_j)$ and $t(termination_i)=t(termination_j)$	$t(termination_i) \leq t(start_j)$	$t(start_i)<t(start_j)<t(termination_i)<t(termination_j)$	$t(start_i)<t(start_j)<t(termination_j)<t(termination_i)$
그림						

적 모순을 포함함으로써, 수행 불가능한 워크플로우가 기술된다. 올바른 워크플로우를 위해, 의미적 모순이 없고 수행 가능함을 의미하는 일관성을 보장해야한다. 워크플로우에 존재할 수 있는 모순을 검증하기 위해, 첫째 태스크간 종속성 요소들간의 조합에 따른 특성을 분석한다. 둘째, 분석된 결과를 바탕으로 모순을 정의하고 이를 예로 제시한다. 마지막으로, 제시한 모순을 워크플로우 그래프에서 발견할 수 있는 기법을 제안한다.

3.1 태스크간 종속성 요소들의 조합특성

워크플로우를 구성하는 임의의 태스크간 종속성들은 이행적 특성을 갖는다. 즉, 하나의 태스크를 매개로 종속성들의 이행적 조합에 의해 새로운 태스크간 종속성을 산출할 수 있다. 6가지 종속성에 대해, 각각의 조합을 표현하면 <표 2>와 같다.

<표 2> 태스크간 종속성 요소의 이행성
<Table 2> Transitions among intertask dependencies

∧		(T_j, T_k)					
		PE	**SS**	**ST**	**SE**	**IE**	**NE**
(T_i, T_j)	**PE**	PE	SS	PT	SE	IE	NE
	SS	SS	SS	①IE	SE	①IE	②ST (∨IE ∨NE)
	ST	ST	IE	ST	SE	IE	NE
	SE	SE	SE	SE	SE	SE	SE
	IE	IE	IE	①IE	SE	①IE	②ST (∨IE ∨NE)
	NE	NE	ST (∨IE ∨NE)	NE	②ST (∨IE ∨NE)	ST (∨IE ∨NE)	NE

① SE나 IE로 태스크 T_i와 T_k간에 종속성을 정의할 수 있으나, SE는 T_i의 종료 후 T_k의 시작을 기술함에 반해 IE는 T_i의 시작 후 T_k의 빠른 시작을 기술할 수 있음으로 인해 **IE를 선택한다.**
② ①과 유사한 이유로써 빠른 시작을 보장하는 모델링을 위해 SE를 고려하지 않는 조합들을 선택한다.

<표 2>에서 제시된 각 요소간에 조합은 대칭성, 멱등성, 흡수성, 그리고 교환성의 특성을 갖는다. 제시된 특성 중 흡수성의 일부만을 증명하였다. 다른 정리의 증명은 제시된 증명방법과 유사하게 증명할 수 있으며 지면상 이유로 생략한다.

정리 1.
임의의 태스크 T_i와 T_j가 종속성 PE를 만족하면 T_j와 T_i는 종속성 PE를 만족한다. 그 역도 성립한다. 이는 다음과 같이 표현되며 이를 대칭성(symmetric)이라 한다.

$$PE(T_i, T_j) \Leftrightarrow PE(T_j, T_i).$$

정리 2.
임의의 태스크 T_i와 T_j가 종속성 P를 만족하고 T_j와 T_k가 종속성 P를 만족한다면 T_i와 T_k는 종속성 P를 만족한다. 여기서 종속성 $P \in \{PE, SS, ST, SE, IE, NE\}$이다. 이는 다음과 같이 표현되며 이를 멱등성(idempotency)이라 한다.

$$P(T_i, T_j) \wedge P(T_j, T_k) \Rightarrow P(T_i, T_k).$$

정리 3.
1. 임의의 태스크 T_i와 T_j가 종속성 SE를 만족하고 T_j와 T_k가 종속성 P를 만족한다면 T_i와 T_k는 종속성 SE를 만족한다.
2. 임의의 태스크 T_i와 T_j가 종속성 P를 만족하고 T_j와 T_k가 종속성 PE를 만족한다면 T_i와 T_k는 종속성 P를 만족한다.
3. 임의의 태스크 T_i와 T_j가 종속성 PE를 만족하고 T_j와 T_k가 종속성 P를 만족한다면 T_i와 T_k는 종속성 P를 만족한다.

여기서 종속성 $P \in \{PE, SS, ST, SE, IE, NE\}$이다. 이는 다음과 같이 표현되며 이를 흡수성(absorption)이라 한다.

1. $SE(T_i, T_j) \wedge P(T_j, T_k) \Rightarrow SE(T_i, T_k)$
2. $P(T_i, T_j) \wedge PE(T_j, T_k) \Rightarrow P(T_i, T_k)$
3. $PE(T_i, T_j) \wedge P(T_j, T_k) \Rightarrow P(T_i, T_k)$.

증명, $SE(T_i, T_j) \wedge P(T_j, T_k) \Rightarrow SE(T_i, T_k)$임을 보인다. $SE(T_i, T_j)$는 다음을 의미한다.

$$t(termination_i) \le t(start_j) \quad \text{----------(1)}$$

다음으로, 조합될 수 있는 종속성 요소 P 각각을 고려한다.

경우 1) P = PE인 경우, $PE(T_j, T_k)$는 다음을 의미한다.

$$t(start_j) = t(start_k) \quad \text{----------(2)}$$
$$t(termination_j) = t(termination_k) \quad \text{----------(3)}$$

(1)과 (2)에 의해 다음을 얻는다.

$$t(termination_i) \le t(start_k) \quad \text{----------(4)}$$

경우 2) P = SS인 경우, $SS(T_j, T_k)$는 다음을 의미한다.

$t(start_j)=t(start_k)$ ----------(2)

$t(termination_j)<t(termination_k)$ ----------(3)

(1)과 (2)에 의해 다음을 얻는다.

$t(termination_i)\leq t(start_k)$ ----------(4)

경우 3) P = ST인 경우, $St(T_j, T_k)$는 다음을 의미한다.

$t(start_j)<t(start_k)$ ----------(2)

$t(termination_j)=t(termination_k)$ ----------(3)

(1)과 (2)에 의해 다음을 얻는다.

$t(termination_i)<t(start_k)$ ----------(4)

경우 4) P = SE인 경우, $SE(T_j, T_k)$는 다음을 의미한다.

$t(termination_j)=t(start_k)$ ----------(2)

또한 태스크 T_j의 시작 및 종료는 다음을 만족한다.

$t(start_j)<t(termination_j)$ ----------(3)

(1), (2), 그리고 (3)에 의해 다음을 얻는다.

$t(termination_i)\leq t(start_k)$ ----------(4)

경우5) P = IE인 경우, $PE(T_j, T_k)$는 다음을 의미한다.

$t(start_j)<t(start_k)$

$<t(termination_j)<t(termination_k)$ ----------(2)

(1)과 (2)에 의해 다음을 얻는다.

$t(termination_i)<t(start_k)$ ----------(3)

경우 6) P = NE인 경우, $PE(T_j, T_k)$는 다음을 의미한다.

$t(start_j)<t(start_k)$

$<t(termination_k)<t(termination_j)$ --------(2)

(1)과 (2)에 의해 다음을 얻는다.

$t(termination_i)\leq t(start_k)$ ----------(3)

제시된 각 경우의 최종 결과((3) 또는 (4))에 의해 T_i, 와 T_k의 수행은 순차 수행 종속성 즉, $SE(T_i, T_k)$을 갖게된다.

정리 4.

임의의 태스크 T_i와 T_j가 종속성 P를 만족하고 T_j와 T_k가 종속성 Q를 만족한다면 T_i와 T_j는 종속성 Q를 만족하고 T_j와 T_k는 종속성 P를 만족한다. 그 역도 성립한다. 여기서 P, Q ∈ {*PE, SS, ST, SE, IE, NE*}이고 단, P = SE(NE)이고 Q = NE(SE)인 경우는 제외한다. 이는 다음과 같이 형식화되며 이를 교환성(commutativity)이라 한다.

$P(T_i, T_j) \wedge Q(T_j, T_k) \Leftrightarrow Q(T_i, T_j) \wedge P(T_j, T_k)$.

3.2 일관성 검증 기법

많은 종속성 요소들로 구성된 전체 워크플로우는 일관성을 위배할 수 있다. 즉, 워크플로우는, 이행적(transitive) 조합에 의해 둘 이상의 상이한 종속성을 내포하거나, 순환이 존재하는 종속성을 포함할 수 있다. 전자는, 임의의 두 태스크가 서로 상이한 둘 이상의 종속성을 갖음으로써, 그들간에 둘 이상의 종속성을 강요해야 하는 논리적 모순이 존재한다. 후자는, 임의의 태스크에서 자신으로의 종속성을 갖음으로써, 자신에서 자신으로 종속성을 강요하는 논리적 모순이 존재한다. 제시된 모순을 정의하기 위해, 먼저, 종속성 요소들의 이행적 조합에 의해 결정되는 간접 종속성을 정의한다. 워크플로우 정의 상에 기술된 태스크간 종속성은 간접 종속성과 구별하기 위해 직접 종속성이라 한다.

정의 5. 간접 종속성(Indirect Dependency)

임의의 종속성 요소 P. Q. 그리고 R과 $TDF = P(T_1, T_2) \wedge Q(T_2, T_3) \wedge ... \wedge R(T_{n-1}, T_n)$에서, TDF에 기술된 종속성을 갖지 않는 임의의 태스크 T_i와 T_j가 표 2의 조합에 의해 종속성 S가 존재하면 S를 간접 종속성이라 한다.

간접 종속성을 기반으로, 태스크사이에 둘 이상의 종속성을 갖는 모순을 다중 종속 모순, 자신에서 자신으로 종속성을 다음과 같이 자가 종속 모순으로 정의한다.

정의 6. 다중 종속 모순(Multi-Dependency Contradiction)

정의된 워크플로우에서, 임의의 태스크 T_i와 T_j간에 둘 이상의 상이한 직접 및 간접 종속성이 존재하면 이를 다중 종속 모순이라 한다.

정의 7. 자가 종속 모순(Self-Dependency Contradiction)

정의된 워크플로우에서, 임의의 태스크 T_i와 T_i간에 직접 및 간접 종속성이 존재하면 이를 자가 종속 모순이라 한다.

먼저, 다중 종속 모순의 예로써, 다음과 같은 워크플로우 기술을 가정하자.

$IE(T_i, T_j) \wedge IE(T_j, T_k) \wedge SS(T_i, T_k)$ -------(1)

$IE(T_i, T_k) \wedge SS(T_i, T_k)$ ----------(2)

(1)은 IE 기반 멱등성에 의해 (2)와 같은 모순을 발견할 수 있다. 즉, T_i와 T_k간에 상충되는 IE와 SS 종속성을 갖음으로써 다중 종속 모순이 존재한다. 다음으로, 자가 종속 모순의 예로써, 다음과 같은 워크플로우 기술을 가정하자.

$$SE(T_i, T_j) \wedge PE(T_j, T_k) \wedge SE(T_k, T_i) \text{ -----(1)}$$
$$SE(T_i, T_k) \wedge SE(T_k, T_i) \text{ ----------(2)}$$
$$SE(T_i, T_i) \text{ ----------(3)}$$

(1)과 같은 워크플로우 기술은 SE 기반 흡수성에 의해 (2)와 같고, 동일한 이유로 (3)과 같이 모순된 종속성을 발견할 수 있다. 즉, T_i 수행 완료 후 T_i를 수행해야 하는 자가 종속 모순이 발견된다.

예에서처럼 워크플로우에 존재하는 다중 종속 및 자가 종속 모순을 발견하기 위해, 워크플로우 정의를 그래프로 변환한다. 변환된 그래프를 다음과 같이 워크플로우 그래프라 한다.

정의 8. 워크플로우 그래프(Workflow Graph)

워크플로우 정의를 구성하는 태스크 집합 Ψ와 각 태스크사이에 존재하는 종속성 집합 Δ에 대해, 다음 f와 g 함수에 의해 변환된 WG = (V, E)를 워크플로우 그래프라 한다.

$$f : \Psi \rightarrow V \quad g : \Delta \rightarrow E$$

함수 f는 각 태스크를 하나의 정점으로 변환한다. 함수 g는 종속성을 선행 태스크에서 후행 태스크로의 방향성 간선으로 변환하며 해당 종속성을 간선의 레이블로 갖는다.

변환된 워크플로우 그래프에서 제시된 모순을 효과적으로 발견하기 위해, 종속성 PE을 갖는 태스크들은 하나의 정점으로 합병한다. 이는 종속성 PE가 갖는 대칭성에도 근거한다. 워크플로우 정의를 워크플로우 그래프로 변환하기 위한 알고리즘은 다음과 같다.

```
Algorithm  Generation_of_Workflow_Graph
Input : Workflow Definition WD
Output : Workflow Graph *WG

  While( ! EOT( WD ) ) { /* End Of Tasks */
    vtex =( char *) malloc( sizeof( vertex ) );
    vtex.ID = task.ID
    WG.vertices=
            AppendVertex(WG.vertices, vtex);
    task = task->next;
  } /* Assigns a task to a vertex */
  While( ! EOD( WD ) ) {
            /* End Of Dependencies */
    edge=(char *) malloc (s izeof(edge));
    edge.LeftID = LeftTask(dependency);
    edge.RightID = RightTask(dependency);
    edge.Label = Label(dependency);
    WG.edges =
            AppendEdge(WG.edges,edge);
    dependency = dependency->next;
  } /* Generate edge between vertices
              which has a dependency */
  tmp = WG.edge;
  While( ! tmp ) {
    If( tmp.Label == 'PE' ) {
     WG.vertices=Merge(WG.vertices,tmp.
                    LeftID, tmp.RightID);
     rtmp = tmp; tmp = tmp->next;
     WG.edges=
          RemoveEdge(WG.egdes, rtmp );
    }
    else tmp = tmp->next;
  } /* Marges vertices that have
              a label PE into a vertex */
EndAlgorithm
```

<알고리즘 1> 워크플로우 그래프 생성 알고리즘

변환된 워크플로우 그래프에서 다중 종속 모순과 자기 종속 모순은 다음과 같다.

정리 5. 워크플로우 그래프에서 다중 종속 모순(Multi-dependency contradiction on workflow graph)

워크플로우 그래프 WG = (V, E)에서, $v_i \in V$, $v_j \in V$인 v_i와 v_j에 대해, v_i에서 v_j로의 병렬 경로(parallel path)가 존재하고, 각 병렬 경로상의 레이블에 대한 종속성이 서로 상이하면 WG는 모순이 존재한다.

증명. 증명을 위해 다음을 가정한다.

[가정] 임의의 정점 v_i에서 v_j로 병렬 경로가 존재하고 각 병렬 경로상의 레이블에 대한 종속성이 서로 상이하면, WG는 모순이 존재하지 않는다

[단계 1] 정점 v_i에서 v_j로 가는 병렬 경로 P와 P′을 다음과 같이 가정하자[1]. 이때, 경로 P와 P′를 구성하는 정점들의 집합을 각각 $TP=\{v_i, v_{i+1}, v_{i+2}, ..., v_{j-1}, v_j\}$, $TP'= v_i, v_{i'+1}, v_{i'+2}, ..., v_{j'-1}, v_j\}$이라 할 때, 일반성에 위배됨이 없이 $TP \cap TP' = \{v_i, v_j\}$라 가정하자.

$$P = v_i, v_{i+1}, v_{i+2}, ..., v_{j-1}, v_j,$$
$$P = v_i, v_{i'+1}, v_{i'+2}, ..., v_{j'-1}, v_j$$

1) 증명을 간략히 하기 위해 두 개의 경로만을 가정한다. 셋 이상의 경로도 유사하게 증명될 수 있다.

[단계 2] 각 경로 상의 레이블에 대해 표 2를 적용하는 함수 Θ에 의해 다음과 같은 종속성 S와 S'를 결정할 수 있다.

$$\Theta(P) \rightarrow S,\ \Theta(P') \rightarrow S'$$

[단계 3] 가정에 의해 $S \neq S'$이면, v_i와 v_j는 S와 S' 종속성을 강요해야 한다. 즉, S는 태스크 v_i의 시작(또는 종료) ***후에*** 태스크 v_j의 시작(또는 종료)을 강요한다면, S'는 태스크 v_i의 시작(또는 종료) ***전에*** 태스크 v_j의 시작(또는 종료)을 강요할 수 있다. 이러한 이유로 기술된 워크플로우는 수행될 수 없다. 가정이 거짓이므로, 병렬 경로가 존재하고 경로의 종속성 조합 결과가 서로 상이하면 WG에는 모순이 존재한다.

정리 6. 워크플로우 그래프에서 자가 종속 모순(Self-dependency contradiction on workflow graph)

워크플로우 그래프 WG = (V, E)에서, $v_i \in V$인 정점 v_i에 대해, v_i에서 v_i로 가는 순환(cycle)이 존재하면 WG는 모순이 존재한다.

증명. 증명을 위해 다음을 가정한다.

[가정] 임의의 정점 v_i에서 v_i로 가는 순환이 존재하면, WG는 모순이 존재하지 않는다.

[단계 1] 일반성에 위배됨이 없이 v_i에서 v_i로 경로 P를 다음과 같이 가정하자.

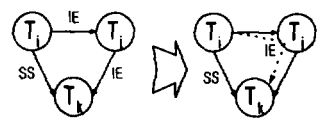
$$P = v_i, v_{i+1}, v_{i+2}, \ldots, v_j, v_i$$

[단계 2] 경로 상의 레이블에 대해 표 2를 적용하는 함수 Θ에 의해 다음과 같은 종속성 S를 결정할 수 있다.

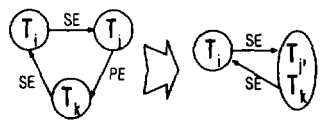
$$\Theta(P) \rightarrow S$$

[단계 3] 종속성 S는 v_i와 v_i간에 종속성 존재를 의미한다. 이는 한 태스크의 시작과 종료가 자신의 시작과 종료에 각각 종속됨을 의미하는 것으로 수행될 수 없는 워크플로우의 정의이다. 즉, 가정이 거짓이므로 WG에 순환이 존재하면 WG에는 모순이 존재한다.

워크플로우 그래프에서 정의된 모순들을 발견하는 알고리즘은 다음과 같다. 한편, 앞서 예시한 모순의 예를 그래프로 표현하면 (그림 4) 및 (그림 5)와 같다. 먼저, (그림 4)에서는 두 태스크간의 종속성이 두 가지 - 직접 종속성 SS와 간접 종속성 IE- 존재함으로써 다중 종속 모순을 발견할 수 있다. (그림 5)에는 합병된 그래프 상에 존재하는 순환을 발견함으로써, 자가 종속 모순을 발견할 수 있다.



(그림 4) 워크플로우 그래프를 이용한 다중 종속 모순
(Fig. 4) Multi-dependency contradiction on the workflow graph



(그림 5) 워크플로우 그래프를 이용한 자가 종속 모순
(Fig. 5) Self-dependency contradiction on the workflow graph

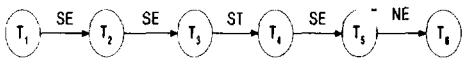
```
Algorithm  Contradiction_Condition_Detection
Input : Workflow Definition WD
Output : Boolean
  Workflow_Graph *WG;
  BoolValue = False;
  Workflow_Path *WP;
  WG=Generation_Of_Workflow_Graph(WD);
  WP = Search_Parallel_Path( WG );
    /* Searches parallel
       path on workflow graph */
  If ( WP != NULL &&
       Detect_Self_Dependency(WP)==True )
    BoolValue = True;
    /*Detects whether or not the
     dependencies of each path are equal */
  If ( Detect_Multi_Dependency( WP, WG )
       == True ) BoolValue = True;
    /* Detects a cycle on workflow graph */
  return BoolValue;
EndAlgorithm

Workflow_Path *Search_Parallel_Path
                ( Workflow_Graph *t_WG)
  Workflow_Path *IDS;
  Vertex SVertex, *AjVertices, AVertex;
  tmpedges = t_WG.edges;
  tmpvertices = t_WG.vertices;
  While( tmpvertices != NULL ) {
    SVertex = SelectVertex( tmpedges );
    /* Select vertex which has
       zero indegree */
    SVertex.path =
     ConcatenatePath(SVertex.path,SVertex);
    /* SVertex.path=SVertex.path+SVertex */
    AjVertices =
     SelectAdjacentVertex(tmpedges,SVertex);
    /* Select adjacent vertices for Svertex */
    While( ! AjVertices ) {
      AVertex = VertexToken(AjVertex);
      AVertex.path = ConcatenatePath
                     (SVertex.path,AVertex);
      If(Outdegree(tmpedges,AVertex)==0){
            IDS=
            ConcatenatePath(IDS,AVertex.path);
            RemoveVertex(tmpvertices,AVertex);
      } /* Generates path */
```

```
        AjVertices =
            RemoveVertex(AjVertices,AVertex)
        /* AjVertices = AjVertices - AVertex */
      } /* Vertex broadcasts
           it's id to adjacent vertices */
      tmpvertices =
          RemoveVertex(tmpvertices,SVertex);
      /* tmpvertices = tmpvertices-SVertex */
      /* Removes the selected vertex */
      tmpvertices = tmpvertices->next;
    }
    return IDS;
End Search_Parallel_Path

Boolean Detect_Self_Dependency
                ( Wrokflow_Path *t_WP )
    Boolean BoolValue = False;
    tmppath = t_WP;
    While( ! tmppath ) {
      While( ! tmppath )
        sTask = TaskToken( tmppath );
        tmppath=RemoveTask(tmppath,sTask);
        If ( sTask ∈ tmppath )
              { BoolValue = True; Break; }
      }
      tmppath = tmppath->next;
    } /* Detects self-dependency
                contradiction */
    return BoolValue;
End Detect_Self_Dependency

Boolean Detect_Multi_Dependency
    (Wrokflow_Path*t_WP,Workflow_Graph t_WG)
    dependency dp, DP;
    Boolean BoolValue = False;

    While( ! t_WP ) {
      tmppath = t_WP;
      While( ! tmppath )
        LeftTask= TaskToken( tmppath );
        tmppath=RemoveTask(tmppath,LeftTask);
        /* tmppath = tmppath - LeftTask */
        RightTask = TaskToken( tmppath )
        tmppath=RemoveTask(tmppath,RightTask);
        dp=SearchLabel
              (t_WG,LeftTask,RightTask );
        DP = GenerateDepedency( DP, dp )
        /*DP = dp∧DP */
      }
      tmppath.dp = DP;
      tmppath = tmppath->next;
    }/*Generates dependencies between tasks*/
    tmppath = t_WP;
    while( ! tmppath ) {
      tmppathnext = tmppath->next;
      If ( tmppathnext == NULL ) break;
      If ( tmppath.dp != tmppathnext.dp ) {
        BoolValue = True;
        break;
      }
      tmppath = tmppathnext;
    } /*Detects multi-dependency contradiction*/
    return BoolValue;
End Detect_Multi_Dependency
```

<알고리즘 2> 자가종속 및 다중 종속 모순 발견 알고리즘

2장과 3장에서 제시한 예제 1과 2를 기반으로, 새로운 종속성 요구를 반영할 때 발생할 수 있는 모순은 다음과 같다.

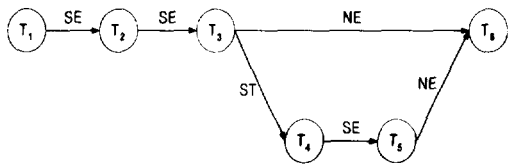
[예제 3] 예제 2에서 제시한 태스크간 종속성 정의를 워크플로우 그래프로 변화하면 (그림 6)과 같다. 이때, 다음과 같은 새로운 종속성 요구를 워크플로우 그래프에 반영하면 다음과 같다.



(그림 6) 워크플로우 그래프로의 표현
(Fig. 6) Representation on workflow graph

[새로운 종속성 요구]

임상병리 검사 태스크 수행 중에 동결조직 검사 태스크를 수행해야 한다.



(그림 7) 새로운 종속성을 반영한 워크플로우 그래프
(Fig. 7) Workflow graph with new dependency

새로운 종속성의 반영은 T_3와 T_6사이에에 다중 종속 모순을 야기한다. 즉, 경로 P_1 =<T_3, T_6>와 경로 P_2 = <T_3, T_4, T_5, T_6>가 존재하고, P_1의 레이블에 대한 종속성은 NE이나 P_2의 레이블에 대한 종속성은 표 2에 의해 SE(ST∧ SE ∧ NE = SE)를 갖는다. T_3와 T_6 간에 서로 상이한 종속성 NE와 SE에 의해, 다중 종속 모순이 발견된다.

4. 결 론

본 연구는 시간 제약을 포함한 워크플로우 모델링 방안을 제안하였다. 먼저, 워크플로우를 구성하는 태스크들의 시간 제약적 특성을 마감시간, 최소 수행시간으로 구분하였다. 둘째, 가장 적절한 대안태스크를 결정하기 위한 우선 순위 함수를 제안하였다. 우선 순위 함수는 대안태스크를 결정하기 위한 기준으로써 시간 요소와 의미적 호환성을 고려하였다. 셋째, 다양한 태스크간 종속성을 기술하기 위해 6가지 종속성을 구분하였다. 6가지 종속성은 기존의 순차 및 병행 종속성

외에도 동시시작 병행 종속성, 동시완료 병행 종속성, 간섭된 병행 종속성, 그리고 중첩된 병행 종속성을 표현할 수 있다. 넷째, 태스크간 종속성을 정의할 수 있는 방안을 제안하였다. 다섯째, 워크플로우 기술상에 존재할 수 있는 모순을 정의하고 이를 발견할 수 있는 기법을 제안하였다. 다중 종속 모순 및 자가 종속 모순을 정의하고 예로 제시하였으며, 이를 방향성 그래프를 이용 발견할 수 있는 기법을 제안하였다.

본 연구에서 제안한 모델링 방안은 기존 연구들과 달리 워크플로우의 시간 제약적 특성을 반영한다. 즉, 완료 및 철회를 근간으로 하는 사건 기반 워크플로우 모델링 방안[6,7,8,9,10,11]과 달리 시간 제약을 포함한 워크플로우 모델링 방안으로써 의미를 갖는다. 또한, 본 연구는 시간적 수행 특성에 근거한 대안 태스크 결정 방안을 제안하였다. 이는 기존 연구[5,6,7]에서 제안하고 있는 대안 태스크 결정 방안 보다 다양한 요구를 반영한다. 즉, 대안 태스크들의 의미적 호환정도를 모델링 할 수 있다. 이는 장기 수행 특성을 갖는 워크플로우의 수행 중에 발생할 수 있는 철회에 대한 융통성 있는 전진(forward) 회복방법으로써 의미를 갖는다. 다음으로, 태스크간에 다양한 병행 종속성을 제시함으로써 동시 시작 및 종료 외에도 동시시작 병행 종속성, 동시완료 병행 종속성, 간섭된 병행 종속성, 그리고 중첩된 병행 종속성을 표현할 수 있다. 이는 기존 연구[6,7,8,9,10,11,12,13,14]에서 태스크간 종속성으로 순차 및 병행 종속성만을 고려함에 반해 본 연구에서는 다양한 병행 종속성을 제시하였다. 즉, 기존 연구에 비해 다양한 제어흐름을 명세할 수 있다. 마지막으로, 그래프를 이용하여 워크플로우의 일관성을 검증할 수 있는 방안을 제안하였다. 이는 기존 연구들이 모델링 요소들만을 제안하였으나 본 연구에서는 워크플로우에 존재할 수 있는 모순을 정의하고 이를 검증할 수 있는 방안을 제안하였다. 즉, 제안된 자가 종속 및 다중종속 발견 알고리즘에 의해 워크플로우에 존재할 수 있는 모순을 발견할 수 있다.

향후 연구에서 시간제약을 포함한 워크플로우의 스케줄링 기법을 개발하고자 한다.

참 고 문 헌

[1] A. L. Scherr, "A new approach to business processes," IBM SYSTEM Journal, Vol.30, No.1, pp.80-98, 1993.

[2] H. Garcia-Molina and K. Salem, "Services for a Workflow Management System," Bulletin of the technical Committee on Data Engineering, Vol. 17, No.1, pp.40-44, 1994.

[3] D. Hollingsworth, "Workflow Management Coalition-The Workflow Reference Model," TC00-1003 issue 1.1, 1994

[4] D. Wodtke, J. Weissenfels, G. Weikum and A. K. Dittrich, "The Mentor Project : Steps Towards Enterprise-Wide Workflow Management," IEEE International Conference on Data Engineering, pp.556-565, 1996.

[5] M. Rusinkiewicz and A. Sheth, "Specification and Execution of Transactional Workflow," *Modern Database Systems : The Object, Interoperability and Beyond*, W. Kim(Ed.), Addison-Wesley, 1994.

[6] P. K. Chrysanthis and K. Ramamritham, "Synthesis of Extended Transaction Models Using ACTA," ACM transaction on Database Systems, Vol.19, No.3, pp.450-491, 1994.

[7] P. K. Chrysanthis and K. Ramamritham, "ACT A : The SAGA Continues," Chapter 10 in Database Transaction Model for Advanced Applications, A. K. Elmagarmid(Ed.), 1992

[8] P. C. Attie, M. P. Singh, A. Sheth and M. Rusinkiewicz, "Specifying and Enforcing Intertask Dependency," VLDB conference, pp.134-145, 1993.

[9] J. Klein, "Advanced Rule Driven Transaction Management," Proc. of the IEEE COMPCON, pp. 562-567, 1991.

[10] D. Georgakopoulos and M. F. Hornick, "A Framework for Enforceable Specification of Extended Transactional Models and Transactional Workflow," International Journal of Intelligent and Cooperative Information Systems, Vol.3, No.3, pp.225-253, 1994.

[11] D. Georgakopoulos, M. F. Hornick and F. Manola, "Customizing Transaction Models and Mechanisms in a Programmable Environment

Supporting Reliable Workflow Automation," IEEE transaction on knowledge and data engineering, Vol.8, No.4, pp.630-649, 1996.

[12] M. E. Rusinkiewicz, A. K. Elmagarmid, Y. Leu and W.Litwin, "Extending the Transaction Model to Capture more Meaning," SIGMOD RECORD, Vol.19, No.1, pp.3-7, 1990.

[13] D. Georgakopoulos, M. Rusinkiewicz and W. Litwin, "Chronological Scheduling of Transactions with Temporal Dependencies," VLDB Journal, Vol.3, No.1, pp.1-28, 1994.

[14] E. Panagos and M. Rabinovich, "Reducing Escalation-Related Costs in WFMSs," *Advances in Wrokflow Management Systems and Interoperability*, A. Doğaç (Ed.), NATO, pp.106-128, 1997.

[15] G. J. Klir, *Fuzzy Sets, Uncertainty, and Information*, Prentice-Hall, pp.65-106, 1988.

[16] J. F. Allen, "Maintaining Knowledge about Temporal Interval," Communication of the ACM, Vol.26, No.11, pp.832-843, 1983.

정 희 택

e-mail : htceong@dbcore.chonnam.ac.kr

1992년 전남대학교 전산통계학과(학사)

1995년 전남대학교 대학원 전산통계학과(이학석사)

1995년~현재 전남대학교 대학원 전산통계학과 박사과정

관심분야 : 워크플로우 시스템, 데이터 웨어하우징, 데이터 마이닝

이 도 헌

e-mail : dhlee@dbcore.chonnam.ac.kr

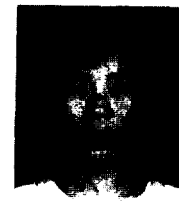
1990년 한국과학기술원 전산학과(석사)

1992년 한국과학기술원 전산학과(석사)

1995년 한국과학기술원 전산학과(박사)

1996년~현재 전남대학교 전산학과 조교수

관심분야 : 데이터마이닝, 워크플로우 시스템, 데이터 웨어하우징, 퍼지 데이터베이스



김 문 자

e-mail : mjkim@etri.re.kr

1984년 전남대학교 계산통계학과(학사)

1986년 전남대학교 대학원 전산통계학과(이학석사)

1985년~현재 한국전자통신연구원 선임연구원

관심분야 : 분산트랜잭션, 워크플로우, 분산시스템, 이동 컴퓨팅

류 영 철

e-mail : lewyc@etri.re.kr

1978년 한양대학교 전자공학과(학사)

1980년 한양대학교 전자공학과(석사)

1983년 금성반도체(주) 컴퓨터사업부

1988년~현재 한국전자통신연구원 선임연구원

관심분야 : 분산 트랜잭션 처리, 트랜잭셔널 워크플로우 관리, CALS/EC, 분산처리