변형된 FP-트리 기반의 적응형 비즈니스 프로세스 마이닝 알고리즘

(An Adaptive Business Process Mining Algorithm based on Modified FP-Tree)

김 건 우 † 이 승 훈 † 김 재 형 †† 서 혜 명 ††† 손 진 현 ††††
(Gun-Woo Kim) (Seung Hoon Lee) (Jae Hyung Kim) (Hyemyung Seo) (Jin Hyun Son)

요 약 기업 간의 경쟁이 심화되고 새로운 비즈니스 가치 창출을 위한 필요성이 증대되고 있는 상황에서, 기업들은 비즈니스 프로세스 관리 기술에 많은 관심을 기울이고 있다. 하지만 비즈니스 분석가와 시스템 개발자간의 이해 정도 및 의견 불일치 등으로 인하여 프로세스가 의도한대로 실행되지 않거나 효율이 떨어지는 프로세스 등이 설계될 수 있다. 이러한 문제점을 해결하기 위하여 비즈니스 프로세스 재설계의 근거로 사용될 수 있는 비즈니스 프로세스 마이닝이 중요한 개념으로 인식되고 있다. 하지만 기존의 프로세스 마이닝에 관한 연구에서는 완성되어 있는 프로세스 로그를 기반으로 워크플로우 기반의 프로세스 모델을 추출하는 단조로운 형태였기 때문에 다양한 형태의 비즈니스 프로세스를 표현하는데 한계가 있었으며, 새로운 프로세스 로그가 추가될 때마다 로그 정보들을 재 스캔해야함으로 프로세스 검출 및 로그 정보 탐색시간이 느려지는 단점이 존재하였다. 본 논문에서는 데이터 마이닝의 연관성 분석에 사용되는 FP-트리를 비즈니스 프로세스에 적합하게 변형하여 추가되는 대량의 프로세스 로그 정보를 재 스캔과정 없이 사용자가 원하는 수준의 프로세스 모델을 검출하도록 지원하는 변형된 FP-트리 기반의 프로세스 마이닝 알고리즘을 제시하고자 한다.

키워드 : 프로세스 마이닝, 데이터 마이닝, 비즈니스 프로세스

Abstract Recently, competition between companies has intensified and so has the necessity of creating a new business value inventions has increased. A numbers of Business organizations are beginning to realize the importance of business process management. Processes however can often not go the way they were initially designed or non-efficient performance process model could be designed. This can be due to a lack of cooperation and understanding between business analysts and system developers. To solve this problem, business process mining which can be used as the basis of the business process re-engineering has been recognized to an important concept. Current process mining research has only focused their attention on extracting workflow-based process model from competed process logs. Thus there have a limitations in expressing various forms of business processes. The disadvantage in this method is process discovering time and log scanning time in itself take a considerable amount of time. This is due to the re-scanning of the process logs with each new update. In this paper, we will presents a modified FP-Tree algorithm for FP-Tree based business processes, which are used for association analysis in data mining. Our modified algorithm supports the discovery of the appropriate level of process model according to the user's need without re-scanning the entire process logs during updated.

Key words : Process Mining, Data Mining, Business Process

· 이 논문은 2007년도 정부(과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(No. R01-2007-000-20135-0)
· 본 연구는 지식경제부 및 정보통신 연구 진흥원의 대학 IT연구센터 육성·지원 사업(IITA-2009-c1090-090 2-0031)의 연구결과로 수행되었음

† 학생회원 : 한양대학교 컴퓨터공학과
gwkim@database.hanyang.ac.kr
shlee@database.hanyang.ac.kr
†† 정 회 원 : 알티베이스 DBMS R&D 개발 본부
canpc815@altibase.com
††† 정 회 원 : 한양대학교 분자생물학과 교수
hseo@hanyang.ac.kr
†††† 종신회원 : 한양대학교 컴퓨터공학과 교수
jhson@hanyang.ac.kr

논문접수 : 2009년 1월 14일
심사완료 : 2009년 12월 28일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제3호(2010.3)

1. 서 론

정보처리, 네트워크, 소프트웨어 등 사회 기반을 형성하는 유형 및 무형의 기술을 정보기술(Information Technology)이라고 한다[1]. 최근의 정보기술은 그 자체로도 산업의 하나로 볼 수 있지만, 다른 산업에 적용되어 기업의 비용을 줄이고 생산성을 향상시키기 위한 효율적인 전략으로 주로 사용되고 있다. 특히 기업 간의 경쟁이 심화되고 새로운 비즈니스 가치 창출을 위한 필요성이 증대되는 상황에서, 기업에서 활용되는 프로세스 흐름의 자동화 및 관련 정보들을 관리하는 비즈니스 프로세스에 많은 기업들은 관심을 기울이고 있다[1-3]. 하지만 비즈니스 프로세스를 설계하고 지원하기 위해 워크플로우 시스템을 도입하는 경우, 비즈니스 분석가와 시스템 개발자간의 이해정도 및 의견 불일치 등으로 인하여 초기부터 완벽한 프로세스를 설계해낸다는 것은 매우 어려운 일이다[2,3]. 이러한 문제점을 해결하기 위하여 기업의 생산성 향상을 위한 비즈니스 프로세스 재설계(Business Process Reengineering)의 근거로 사용되는 비즈니스 프로세스 마이닝이 중요한 개념으로 인식되고 있다[4]. 비즈니스 프로세스 마이닝은 프로세스가 실행될 때 발생하는 프로세스들의 로그(Log) 정보를 수집하여 완성된 형태의 프로세스 모델을 생성하는 것에 그 목적이 있다[1,2,4]. 비즈니스 프로세스 마이닝은 새로운 개념이 아니다. 이전에도 비즈니스 프로세스 재설계를 위한 로그 정보는 시스템 차원에서 제공되어왔다. 그러나 워크플로우 기반의 시스템에서는 실행중인 단편적인 프로세스 정보만을 제공하였으며 단순한 데이터 베이스 필드 수준에 불과했다. 하지만 비즈니스 프로세스 관리 기술이 EIP(Enterprise Information Portal), BAM(Business Activity Monitoring), RTE(Real-Time Enterprise)등 실시간 정보공유 및 모니터링을 지원하도록 시스템이 발전하면서 프로세스 마이닝은 프로세스 분석과 개선을 위한 주된 개발영역으로 간주되고 있다[1-3,5].

비즈니스 프로세스 관리 기법에서 사용되는 비즈니스 프로세스 마이닝은 크게 3가지 기법으로 구성된다. 첫 번째로는 정보 시스템(Information System)의 실행 결과가 기록된 프로세스 로그 정보를 추출하여 프로세스 모델을 찾아내는 프로세스 모델 발견(Discovery)기법이 있으며, 두 번째로는 발견된 프로세스 모델과 프로세스 로그 정보를 비교하여 기업에서 의도했던 프로세스의 목표에 적합한지를 평가하는 적합성 검사 기법(Conformance Checking), 마지막으로는 발견된 프로세스 모델을 변경하여 적합성을 상승시키는 확장(Extention) 기법이 있다[1]. 프로세스 마이닝의 3가지 기법 중에서 첫 번째인 프로세스 모델 발견 기법은 프로세스 평가 및 개선을 위한 2가지 기법인 적합성 기법과 확장 기법의 근간이 되기 때문에 가장 중요한 부분을 차지하게 된다. 그 결과 프로세스 마이닝에 대한 기존 연구에서도 프로세스 발견 기법에 대해 많은 연구가 이루어졌다. 하지만 기존연구에서는 시스템 개발자 위주의 워크플로우 방식에 기반을 두어 비동기적인 시스템을 표현하기위한 그래픽 툴인 페트리넷이 많이 사용되어 왔기 때문에 AND 게이트웨이로 이루어진 Parallel Split & Merge 패턴과 XOR 게이트웨이로 이루어진 Exclusive Choice & Merge 패턴 발견에 대해서만 다루고 있고 OR 게이트웨이를 사용한 Multiple Choice & Merge 패턴 발견을 고려하지 않으므로 다양한 형태의 비즈니스를 표현하는데 한계가 있었다. 또한 완성되어있는 프로세스 로그를 기반으로 워크플로우 프로세스 모델을 추출하는 단조로운 형태였기 때문에 대량의 새로운 프로세스 로그가 추가될 때마다 초기에 기록되었던 로그 정보부터 새로 추가된 로그 정보까지 모두 재 스캔해야 하는 단점이 존재하였다. 그 결과 제한적이며 간단한 프로세스에만 적용이 가능하였고 프로세스 검출 및 로그 정보 탐색시간이 느려지는 결과가 발생하게 되었다.

본 논문에서는 데이터 마이닝의 연관성 분석에 사용되는 FP-트리를 비즈니스 프로세스 마이닝의 프로세스 모델 발견 기법에 적합하도록 변형하여 기존 연구에서 문제점으로 지적되었던 OR 게이트웨이를 사용한 Multiple Choice & Merge 패턴 발견을 지원한다. 또한 실시간으로 추가되는 대량의 프로세스 로그 정보를 기존 로그의 재 스캔 과정 없이 빠르게 알고리즘에 반영하여 좀 더 신뢰할 수 있는 프로세스를 생성하는데 목적이 있다. 하지만 프로세스 마이닝은 데이터 마이닝과는 다른 분야에 속하기 때문에 기존 데이터 마이닝의 방법론을 그대로 적용할 수 없다. 이러한 이유로, 본 논문에서는 비즈니스 프로세스에 적합하게 변형한 FP-트리를 이용한 프로세스 마이닝 알고리즘을 제시한다.

본 논문에서 제안하는 변형된 FP-트리는 대량의 프로세스 로그 정보를 모두 FP-트리 내부에 저장하여 서로 다른 트랜잭션들이 공통으로 포함하는 중복된 태스크들을 삭제하고 재 스캔 과정 없이 빠르게 프로세스 모델을 추출할 수 있도록 한다. 또한 최종적으로 프로세스 모델 검출을 위해 변형된 FP-트리를 표준화된 비즈니스 프로세스 모델 표기법인 BPMN기반의 프로세스 모델로 매핑 하는 방법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구인 프로세스 마이닝 기법과 연구 동향, 데이터 마이닝에서 사용되는 FP-트리 및 직접인과행렬에 대하여 설명하며, 3장에서는 비즈니스 프로세스 모델 검출을 위해

적합하게 변형한 FP-트리에 대하여 설명한다. 4장에서는 변형된 FP-트리 기반의 비즈니스 프로세스 마이닝 알고리즘을 이용하여 프로세스 모델을 검출 하는 방법에 대하여 설명하고 5장에서는 변형된 FP-트리를 이용하여 복잡한 프로세스 모델 검출 방법에 대한 예제를 제시한다. 6장에서는 개발된 알고리즘의 성능평가를 위한 비교실험을 수행하고, 마지막 6장에서는 논문의 결론을 맺는다.

2. 관련 연구

2.1 프로세스 마이닝(Process Mining)

최근의 정보 시스템들은 프로세스의 실행결과가 기록된 상세한 프로세스 로그 정보들을 자체적으로 제공하고 있다[1,3]. 이러한 프로세스 로그들은 프로세스 마이닝을 위한 기반이 된다. 프로세스 마이닝은 정보 시스템으로부터 프로세스의 실행 결과가 기록된 프로세스 로그 정보를 기반으로 수행되며 기존의 데이터 마이닝과는 다르게 프로세스 로그에 기반을 둔 순차적인 프로세스 모델을 찾는 것에 그 목적이 있다.

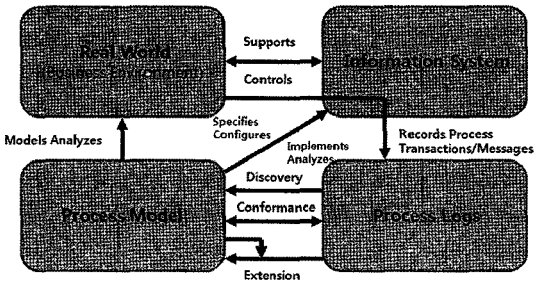

그림 1 프로세스 마이닝

그림 1은 프로세스 마이닝에 대한 간략한 정보를 나타낸다. 실제 비즈니스 환경에서 기업의 목표 달성 및 정책 목적을 실현하기 위하여 설계된 비즈니스 프로세스를 정보 시스템에 적용하게 되면 프로세스 트랜잭션의 결과물인 프로세스 로그를 출력하게 된다. 출력된 프로세스 로그를 바탕으로 비즈니스 재설계의 근거로 사용되는 비즈니스 프로세스 마이닝을 수행하기 위해 서론에서 소개한 바와 같이 프로세스 발견 기법, 적합성 검사 기법, 확장 기법 등을 적용하게 된다. 그 결과 실제 비즈니스 환경과 비교 분석을 통하여 효과적이고 효율적인 프로세스를 검출해 낼 수 있으며 정보 시스템에 적용된 업무 프로세스를 개선할 수 있다.

비즈니스 프로세스 마이닝에서 사용되는 기본적인 프로세스 모델 검출 방법을 살펴보면 다음과 같다.

표 1은 프로세스가 실행되면서 기록된 프로세스 로그

표 1 프로세스 실행시의 출력된 프로세스 로그

Case Identifier	Task Identifier
Case 1	Task A
Case 2	Task A
Case 3	Task A
Case 3	Task B
Case 1	Task B
Case 1	Task C
Case 2	Task C
Case 4	Task A
Case 2	Task B
Case 2	Task D
Case 5	Task E
Case 4	Task C
Case 1	Task D
Case 3	Task C
Case 3	Task D
Case 4	Task B
Case 5	Task F
Case 4	Task D

정보이다. 표 1에 제시된 프로세스 로그정보에 의하면 프로세스 모델은 총 5개의 프로세스 인스턴스 케이스로 구성 되어 있으며 각각의 케이스들을 실행순서대로 표현하면 Case 1은 A->B->C->D, Case 2는 A->C->B->D, Case 3은 A->B->C->D, Case 4는 A->C->B->D, 마지막으로 Case 5는 E->F 순서로 실행된다. 그 결과 Case 1부터 Case 4까지에서는 A, B, C, D 4가지 업무들이 모두 실행되는 것을 알 수 있으며 Case 5인 경우에만 업무 E와 업무 F가 실행되는 것을 알 수 있다. 하지만 Case 1 부터 Case 4까지의 업무별 트랜잭션을 보면 업무 B와 업무 C의 실행 순서가 다른 것을 알 수 있다. 즉 Case 1과 Case 3에서는 업무 B가 실행되고 나서 업무 C가 실행되지만 Case 2와 Case 4에서는 업무 C가 실행되고 나서 업무 B가 실행된다. 하지만 Case 1 부터 Case 4까지의 업무별 트랜잭션에서는 공통적으로 업무 A로 시작하여 업무 D로 끝나기 때문에 업무 B와 업무 C가 서로 병렬적으로 동시에 실행되는 것을 유추할 수 있다. 그러므로 표 1에 제시된 프로세스 로그는 프로세스 시작점으로부터 업무 A 또는 업무 E로 선택 실행되는 프로세스이며 업무 B와 업무 C는 업무 A가 실행되었을 때만 서로 병렬적으로 동시에 실행되는 프로세스임을 알 수 있다. 그 결과 표 1에 제시된 프로세스 로그를 페트리넷 기반의 프로세스 모델로 표현하면 그림 2와 같다.

2.2 프로세스 마이닝 연구 동향

프로세스 마이닝은 1998년에 Agrawal et al.과 Cook and Wolf에 의해 시작되었다. 프로세스 마이닝의, 3가

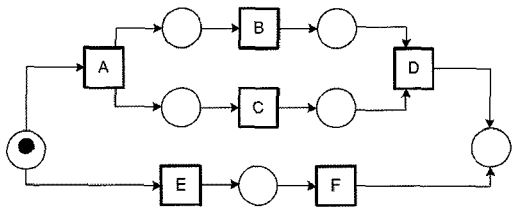

그림 2 프로세스 로그로부터 생성된 프로세스 모델

지 영역 중 프로세스 발견 기법은 가장 연구가 활발히 진행되고 있는 분야이다. 프로세스 마이닝에 분야에 사용되는 기술은 유전자 알고리즘(Genetic algorithm), 맞춤형 알고리즘(custom algorithm), 마코브(Markovian approach), 신경망(Neural network), 클러스터 분석(Cluster analysis)의 5가지 기술이 가장 많이 사용되고 있다[6,7]. 프로세스 마이닝에 대한 논문 중 대부분의 논문에서는 워크플로우를 기반으로 한 맞춤형 알고리즘을 가장 많이 사용하고 있으며, 적은수의 논문에서만 데이터 마이닝 알고리즘이나 순수 컴퓨팅 기술을 사용하고 있다. Agrawal et al.은 1998년에 워크플로우 기반의 시스템에서 얻어지는 로그를 기반으로 프로세스 흐름 그래프(Process Flow Graph)를 구성하는 방법을 제시하였으며 처음으로 방향성 그래프를 프로세스 마이닝에 이용하였다[6]. Cook and Wolf 또한 1998년에 RNet이라 불리는 뉴럴 네트워크, 마코브, 그리고 Ktail이라 불리는 순수 알고리즘(purely algorithmic)의 방법을 이용하여 확률적인 방법을 통하여 워크플로우 프로세스 모델을 개선하는 방법을 제안하였다. 그렇지만 이 방법은 순차적인 구조(sequential structure)의 프로세스밖에 추론할 수 없다는 단점이 있었다. 그래서 이후의 연구에서 Cook과 and Wolf는 동시 실행 프로세스도 추론할 수 있도록 연구를 확장하였다. 그렇지만 이 방법에는 AND 게이트웨이와 XOR 게이트웨이에 대한 판별이 어려워 완전한 프로세스 모델(complete process model)을 검출하는데 어려움이 있었다[7]. Aalst et al.은 2004년에 α-algorithm을 이용하여 선택실행 프로세스를 고려한 워크플로우 모델을 발견하는 연구를 하였으며, Aalst et al.과 Medeiros et al.은 각각 2005년과 2006년에 유전자 알고리즘을 이용하여 선택실행 프로세스 뿐만 아니라 동시실행 프로세스를 고려한 프로세스 마이닝 방법을 연구하였다[8,9]. 유전자 알고리즘을 적용하기 위해 검출되는 프로세스 모델을 워크플로우 기반의 모델링 기법인 페트리넷(Petri Net)을 사용하여 표현하였고 인과행렬(Casual Matrix)를 이용하여 프로세스의 적합도를 평가하였다. 이 방법으로 프로세스 모델 내부의 AND게이트웨이와 XOR 게이트웨이를 판별할 수 있게 되었지만 완성되어 있는 프로세스 로그를 기반으로 프로세스 모델을 검출해야하기 때문에 새로운 로그 정보가 추가 되는 경우 프로세스 검색 시간 및 평가 시간이 매우 높았으며, 데이터양이 증가할수록 알고리즘의 효율성이 급격하게 낮아지게 된다.

2.3 FP-트리(Frequent Pattern Tree)

FP-트리는 데이터 마이닝의 연관 분석에서 사용되는 FP 성장 알고리즘을 위한 자료 구조다[5]. FP-트리는 DB 스캔에 대한 부담을 획기적으로 절감시킴으로써 전체적인 성능을 향상시키고자 제안되었으며 매우 간결한 자료구조 형태이기 때문에 직접 빈발 패턴의 추출이 가능하다[5].

그림 3에서는 FP-트리의 생성 방법을 나타내었다. 먼저 FP-트리에서는 입력된 트랜잭션들을 ID를 부여하여 구분한 후 각 트랜잭션의 아이템들을 사전 순서로 정렬하고, 정렬된 아이템들을 트리에 삽입함으로써 트리의 구축이 시작된다. 그림 3에 제시된 4가지의 그림 중 트랜잭션 ID 1번(i.e., TID=1)이 삽입된 트리를 보면 삽입된 아이템은 뿌리 노드(Root Node)의 자식 노드(Child Node)가 된다. 이후에는 아이템들의 시퀀스(Sequence) 대로 자식노드가 생성되게 된다. 그림 3의 TID=2와 TID=3 그림처럼 트랜잭션 ID가 계속적으로 삽입되는 경우에 만약 같은 노드를 발견하게 되면 해당 노드의 카운트(Count)에 더하고, 다른 노드가 발견될 때까지 같은 작업을 반복한다. 이러한 작업을 모든 트랜잭션에 대해 수행하고 나면 그림 3의 TID=10번과 같은 완성된 FP-트리가 된다. 만약 같은 아이템이 다른 줄기 위 노드에 존재할 경우, 포인터를 이용해 모든 동일한 아이템

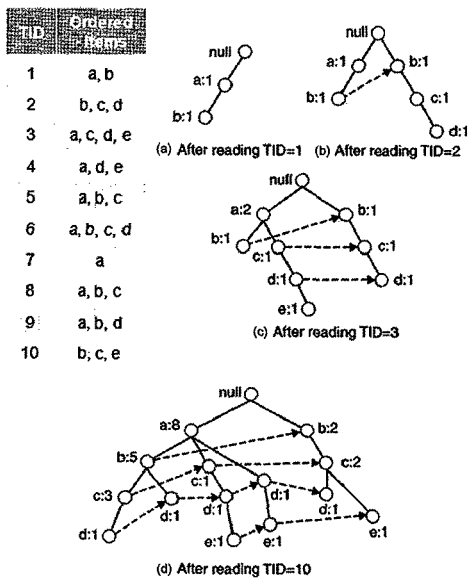

그림 3 FP-트리 표현

이 있는 노드를 연결하게 한다. 이를 통해, 알고리즘의 빠른 데이터 접근을 가능하게 한다. 위에서 설명한 바와 같이, FP-트리는 중복된 입력에 대해 카운트 값만을 유지할 뿐 여분의 공간을 요구하지 않는 매우 간결한 구조를 갖고 있음을 알 수 있다.

2.4 직접 인과 행렬(Direct Causal Matrix)

직접 인과 행렬(Direct Causal Matrix)은 본 논문에서 제안한 알고리즘인 변형된 FP-트리에 프로세스 로그 정보를 적용할 때, 태스크(Task) 간의 인과 관계를 쉽게 계산하기 위해 유지하는 행렬이다[8]. 인과 관계는 프로세스 로그 상에 나타난 중복된 태스크들을 하나의 태스크로 정확하게 검출하기 위한 근거가 된다. 직접 인과 행렬은 그림 4와 같다. 각 행과 열 사이에 직접적인 인과관계가 있을 경우, 행렬 값은 1이 되고, 그렇지 않을 경우는 0이 된다. 'start'와 'end'는 시작 노드(Start Node)와 끝 노드(End Node)에 해당한다. 각 케이스별로 태스크 간의 인과관계를 모두 행렬 상에 표현하게 된다. 직접 인과 행렬은 태스크의 개수에 따라 행렬의 크기가 달라지지만 프로세스 로그의 크기와는 관계가 없는 간결한 형태의 행렬이 된다.

본 논문에서 제시된 변형된 FP-트리 기반의 프로세스 마이닝 알고리즘은 트리를 구축할 때 인과 관계에 대한 정보를 추출할 필요가 있는데 이를 저장하기 위해 직접 인과 행렬이 사용된다. 즉 초기 상태에 트리가 구축된 후에 알고리즘이 진행되면서 앞서 소개한 FP-트리와 같이 차츰 변형되며, 같은 프로세스 태스크가 다른 줄기 위 노드에 존재할 경우에 직접 인과 행렬의 인과관계를 이용해서 중복된 태스크(Duplicated Task)를 제거한다.

Case 1 A→B→C→D
Case 2 A→C→B→D
Case 3 A→B→C→D
Case 4 A→C→B→D
Case 5 E→F

	start	A	B	C	D	E	F	end
start	0	1	0	0	0	1	0	0
A	0	0	1	1	0	0	0	0
B	0	0	0	1	1	0	0	0
C	0	0	1	0	1	0	0	0
D	0	0	0	0	0	0	0	1
E	0	0	0	0	0	0	1	0
F	0	0	0	0	0	0	0	1
end	0	0	0	0	0	0	0	0

그림 4 직접 인과 행렬(Direct Causal Matrix)

3. 비즈니스 프로세스 모델 검출을 위한 변형된 FP-트리의 설계 및 구성

본 논문에서 제시된 비즈니스 프로세스 모델 검출을 위한 프로세스 마이닝 알고리즘인 변형된 FP-트리는 OR 게이트웨이를 사용한 Multiple Choice & Merge 패턴을 지원하여 다양한 형태의 비즈니스 프로세스를 지원하며 실시간으로 추가되는 대량의 프로세스 로그 정보를 기존 로그의 재 스캔 과정 없이 빠르게 알고리즘에 반영하여 좀 더 신뢰할 수 있는 프로세스를 생성하기 위하여 데이터 마이닝의 연관성 분석에 사용되는 FP-트리를 이용하였다. 하지만 비즈니스 프로세스 마이닝은 데이터 마이닝과는 다른 분야에 속하기 때문에 기존 데이터 마이닝에서 사용되던 FP-트리 자료구조 방법론을 그대로 적용할 수 없다. 본 논문에서 제시된 변형된 FP-트리를 사용하기 위해서는 아래와 같은 몇 가지의 제약 조건을 갖는다.

3.1 제약 조건

본 논문에서 제시하는 변형된 FP-트리 알고리즘을 적용하기 위해서는 다음과 같은 제약 조건을 갖는다.

- 프로세스 로그의 모든 트레이스(Trace)들은 '시작 노드(Start Node)'와 '끝 노드(End Node)'를 갖는다. 트레이스는 프로세스 로그의 케이스를 말한다.
- 프로세스 상에는 중복된 태스크(Duplicated Task)가 존재하지 않는다.
- 프로세스 로그를 생성하는 원본은 항상 잘 구성된(Well Structured) 프로세스에 해당한다.

위 제약 조건들이 성립한다고 했을 경우 본 논문에서 제시하는 알고리즘이 올바르게 적용될 수 있다.

3.2 변형된 FP-트리의 설계와 구성

이 절에서는 본 논문에서 제시한 변형된 FP-트리를 구축하는 설계 과정 및 구성 형태에 대하여 설명한다. 변형된 FP-트리를 구축하는 전략은 데이터마이닝에서 연관 분석에 사용되는 FP-트리와 기본적인 구성이 같다. 하지만 이 두 가지 자료 구조 사이에는 명백한 차이점이 몇 가지 존재한다.

먼저 비즈니스 프로세스 모델 검출을 위해 적합하게 변형된 FP-트리는 시작 노드와 끝 노드가 존재한다. 또한 트리를 구축하기 위해 사용되는 입력 로그의 시퀀스들은 정렬되어 있지 않은 상태로 유지된다. 마지막으로 내부 노드의 링크는 양방향 연결 리스트(Doubly Linked List)의 구조로 되어 있다. 그림 5는 표 1과 그림 4의 프로세스 로그를 기반으로 생성된 변형된 FP-트리를 나타낸다. 눈 여겨 볼만한 점은 앞서 기술한 바와 같이 시작 노드와 끝 노드인 'start'와 'end'를 두 개의 뿌리(Root)로 갖는 이중 뿌리 형태의 트리 구조를 갖는 것이다.

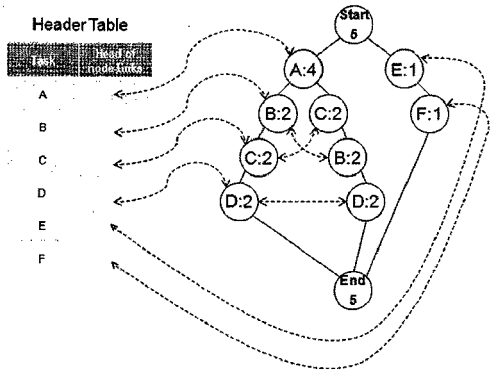

그림 5 변형된 FP-트리

이해를 돕기 위해 실제 데이터를 삽입하는 과정을 그림 6에 나타내었다. 각 트리는 프로세스 로그의 각 케이스가 삽입된 이후의 형태를 나타낸다. 그림 6의 트리는 표기의 어려움을 이유로 헤더 테이블의 노드 링크를 생략한 형태로 표현하였다. 하지만 트리 내의 노드 링크는 그대로 나타내었다.

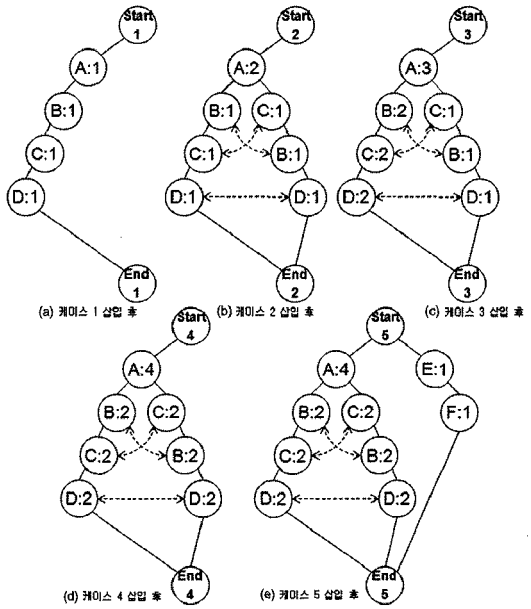

그림 6 변형된 FP-트리의 프로세스 데이터 삽입과정

그림 6을 보면 헤더 테이블의 노드 링크를 확인할 수 있는데, 이는 같은 이름을 갖는 노드에 대한 접근을 용이하게 한다. 변형된 FP-트리의 각 노드들은 태스크의 이름과 각 태스크이 몇 번 노드에 매핑되어 나타났는지를 나타내는 카운터를 포함한다. 그림 6에 나타난 변형된 FP-트리의 구축 과정은 아래와 같은 방법으로 반복적으로 진행된다.

(I) 프로세스 로그의 데이터는 트리 구축을 위해서 기존 로그의 재 스캔 과정 없이 단 한번만 스캔된다. 프로세스 로그의 각 케이스를 읽어 들이는 순서대로 트리 상에 매핑 한다. 첫 번째 케이스의 값인 A, B, C, D를 매핑하면 그림 6의 (i)와 같다. 이 때 한번 씩 매핑되었으므로 카운트 값은 1이 된다. 마지막 태스크가 매핑 되면, 그 줄기는 'end' 노드로 연결된다.

(II) 두 번째 케이스가 입력되면 알고리즘은 이 케이스를 다시 트리 상에 매핑 한다. 이 때, 'start→A'라는 줄기를 갖는 경로가 트리 상에 존재하므로 삽입된 노드 A는 이 줄기 상에 포함된다. 이 때 A는 두 번째 매핑되었으므로 카운트 값은 2가 된다. 하지만 두 번째 케이스의 A 이후의 태스크들은 첫 번째 케이스와 다르므로 다른 경로로 구분된다. 따라서 나머지 태스크들은 A에서 분기되고, 트리 상에 다른 줄기로 매핑 된다. 이 때, 트리 상에 같은 이름을 갖는 노드가 존재하므로 이들은 헤더 테이블에서 시작된 노드 링크에 연결된다. 즉, 헤더 테이블은 새로운 이름을 갖는 노드가 삽입될 때마다 갱신되고, 노드 링크 역시 같은 이름을 갖는 노드인지를 판별하여 링크의 연결을 갱신하게 된다.

(III) (II)의 방법과 마찬가지로 케이스 3과 4가 삽입이 되고, 마지막으로 케이스 5가 삽입이 되면 'start→E'라는 새로운 줄기가 만들어지므로 'start' 노드에서 분기한다. 'start'와 'end' 노드는 케이스가 삽입될 때마다 카운트가 증가한다.

정의 1. (비즈니스 프로세스 모델 검출을 위한 변형된 FP-트리) 변형된 FP-트리는 아래와 같이 정의되는 트리 자료 구조다.

가. 비즈니스 프로세스 모델 검출을 위해 변형된 FP-트리는 'start'와 'end'라고 명명된 두 개의 뿌리 노드를 갖고, 입력 로그의 시퀀스들을 구성하는 태스크들이 노드가 되며, 노드의 헤더 테이블로 구성되어 있다.

나. 각 노드는 몇 개의 필드 정보를 소유하고 있다. 특정 노드에 도달하기 위해 해당 노드가 몇 번 수행되었는지를 나타내는 노드 카운트(node_count), 입력 로그의 시퀀스에 해당하는 태스크의 이름을 저장하는 태스크 이름(task_name), 트리에서 해당 노드의 깊이(depth) 정보를 저장하는 노드의 깊이(node_depth), AND/OR/XOR-split/join 등의 노드 형태를 결정짓는 노드의 타입(node_type) 정보 등으로 구성된다.

다. 노드의 헤더 테이블의 각 엔트리들은 크게 두 가지 필드 정보로 구성된다. 해당 노드의 태스크 이름(task_name)을 저장하고, 해당 태스크 이름이 처음 나타난 노드에 대한 포인터 정보가 이에 해당한다.

정의 1을 바탕으로 변형된 FP-트리의 자료구조를 pseudocode형태로 표현하면 그림 7과 같다.

```
struct node
{
        int node_count;
        int node_depth;
        keytype task_name;
        keytype node_type;
        index pre_nodelink;
        index next_nodelink;
        index[] parentnodes;
        index[] childnodes;
}
```

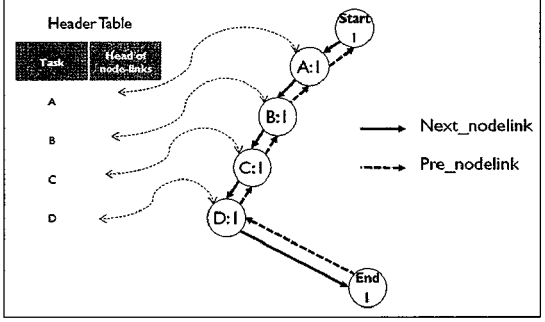

그림 7 변형된 FP-트리의 자료 구조

그림 6에서와 같이 변형된 FP-트리에 실제 데이터를 삽입하는 과정을 나타내는 알고리즘은 다음과 같다.

알고리즘 1 (변형된 FP-트리의 데이터 insert과정)

입력(Input): 이벤트 로그들의 시퀀스

출력(Output): 데이터만 insert된 초기상태의 변형된 FP-트리

메소드(Method): 변형된 FP-트리의 insert과정은 아래와 같은 방법에 따라 구축된다.

```
// c: 각 케이스에 포함된 태스크 중 첫 번째 태스크
// C: 각 케이스에 포함된 태스크 중 c를 제외한 나머지 태스크
// T : 현재까지 생성된 FP-트리
insert ([c|C], T)
{
 Node start, end;
 PatternSet c = Null; C= Null;
 N = childnodes;
 Tree T;

 if (start == null)
 {
  Create_Tree();
  start = c;
  c-->next_nodelink = C;
  node_count++;
 }
 for(int node_depth=0;node_depth< LogLength;
     node_depth++){
  if(N.task_name == c.task_name)
  {
    N.node_count++;
    //업데이트된 노드를 트리에 반영
    updateTree();
  }
  else
  {
    temp = Create_node();
    temp.node_count = 1;
    N = temp.pre_nodelink; //N은 현재 노드의 부모노드
    updateTree();
  }
 }

 //현재 입력중인 시퀀스가 비어있지 않으면 insert함수를
 //재귀적으로 호출
 insert ([c|C], T);
}
```

그림 8 변형된 FP-트리의 데이터 insert과정

1. 이벤트 로그의 각 케이스들을 추출한다.

2. "start/end" 노드를 갖는 초기상태의 FP-트리를 만든다. 각 케이스들에 대해서 insert([c|C], T) 함수가 다음과 같은 방법으로 수행된다. 여기서 c는 각 케이스에 포함된 태스크 중 첫 번째 태스크를 말하고, C는 그 나머지들을 말한다. T는 현재까지 생성된 초기상태의 FP-트리를 말한다. 만약 트리 T가 조건 N.task-name = c. task-name을 만족하는 자식 노드 N을 갖고 있다면, N의 노드 카운트는 1 증가한다; 그렇지 않으면 새로운 노드 N을 생성한다, 그리고 그 노드의 노드 카운트는 1이 된다, 그 노드의 부모 노드는 트리 T의 'start' 노드가 된다, 그리고 그 노드의 노드 링크는 현재까지 트리 T에 존재하는 같은 태스크 이름을 갖는 노드들의 노드 링크와 연결된다. 만약 이벤트 로그의 현재 입력중인 시퀀스 P가 비어있지 않다면 insert([c|C],T) 함수를 재귀적으로 호출한다.

4. 변형된 FP-트리를 이용한 프로세스 검출

이 장에서는 변형된 FP-트리를 이용한 프로세스 검출 기법에 대해서 소개한다. 먼저 알고리즘이 필요한 이유와 적용의 목적에 대해서 설명하고, 프로세스 모델을 검출하기 위한 전략을 상세히 기술한다.

4.1 알고리즘의 목적

'프로세스 상에는 중복된 작업(Duplicated Task)가 존재하지 않는다.'는 말은 이미 이전의 제약 조건에서 언급한 바 있다. 이는 프로세스 추출 과정에서 올바른 결과를 도출하기 위한 것이다. 변형된 FP-트리에 실제 프로세스 데이터 값을 적용하였을 때 트리내부에 중복된 태스크들이 존재할 수 있다. 중복된 태스크들이 나타나는 근본적인 이유는 이벤트 로그의 원본이 되는 시스템의 모델이 분기(Split)를 갖고 있기 때문이다. 따라서 이러한 중복된 태스크들을 제거하는 과정을 거치면 본래의 태스크를 찾는다는 목적에 부합하게 된다. 이 알고리즘의 목적은 결국 중복된 태스크를 제거함으로써 올바른 프로세스 모델을 추출하는 것에 있다. 다음 절에서는 이를 위한 과정을 상세히 기술한다.

4.2 추출 알고리즘(Discovery Algorithm)

3장에서 언급한 제약 조건을 통해 생성된 중복 태스크 제거에 대한 알고리즘이 주된 내용이다. 제거와 동시에 변형된 FP-트리의 구축을 위해서 다음과 같은 알고리즘을 설계하였다.

알고리즘 2 (중복된 태스크 제거)

입력(Input): 데이터만 insert된 초기상태의 변형된 FP-트리

출력(Output): 중복된 태스크가 제거된 중간 상태의 변형된 FP-트리

메소드(Method): 중복된 태스크 이름을 갖는 모든 노드는 다음과 같은 과정에 따라 제거된다.

```
/*중복된 태스크가 존재하는지 판단하여 Merge 함수를 수행해야
 *하는지 검사
 */
checkingMerge()
{
 /*
   *헤더 테이블의 노드링크를 통해 n번째 노드에 도달하였는지
   *검사
   */
  for(nodeCount=0; nodeCount = n ;nodeCount++)
    {
     //중복된 태스크가 없는 경우
     if((n-1)th.task_name = the head of node links.
         task_name)
     {
      exitMerge();
     }
     //중복된 태스크가 있는 경우
     else
     {
      merge(sourceNode, targetNode, T);
     }
    }
}
/*
 * sourceNode : 병합 하는 노드
 * targetNode: 병합 되는 노드
 * T : 현재까지 생성된 FP-트리
 */
merge(sourceNode, targetNode, T)
{
  Node A, B, C;
  A = targetNode.pre_nodelink; //targetNode의 부모
  B = targetNode.next_nodelink; //자식 nodelink
  C = temp_nodelink;

  //직접 인과 관계가 있는지 판단
  if(check_CausalMatrix(A,B))
  {
    //직접 인과 관계가 존재한다면 A와 B의 정보를 C에 복사
    Copy_Info(A,B.C);
  }
}
```

그림 9 변형된 FP-트리에서 중복된 태스크를 검사하는 checkingMerge 및 중복된 태스크를 삭제하는 Merge과정

1. 변형된 FP-트리에 포함된 헤더 테이블을 통해 중복된 태스크를 찾는다. 중복된 태스크를 검색하는 일은 checkingMerge 함수를 통해 헤더 테이블에 기록된 순서대로 순차적으로 수행된다. 만약 헤더 테이블의 노드 링크를 통해 n번째 노드에 도달했을 때, n-1번째 노드가 조건 '(n-1)-th node.task-name = the head of node-links.task-name'를 만족하면, 해당 노드 링크의 태스크는 중복된 태스크를 갖지 않는다. 그렇지 않으면 함수 merge(sourceNode, targetNode, T)가 다음 과정에 따라 수행된다.

2. 중복된 태스크가 존재할 경우, 두 개의 노드에 대해서 병합이 진행된다. 근원 노드(sourceNode)는 함수 merge(sourceNode, targetNode, T)에서 제거되어야 할 노드를 가리킨다. 함수를 통해 제거되면 목표 노드(targetNode)라는 병합의 대상이 되는 노드가 근원 노드의 링크 정보 및 노드 카운트를 물려받게 된다. 근원 노드의 부모 노드를 A라고 하고, 근원 노드의 노드 링크가 가리키는 노드를 B라고 가정한다. 함수 merge (sourceNode, targetNode, T)가 수행되면, 부모 노드에 대한 링크와 노드 링크의 정보가 근원 노드의 자식 노드 C로 복사된다. 이 작업을 수행하기 위해서는 관련 연구에서 언급한 직접 인과 행렬이 필요하다. 직접 인과 행렬의 인과 정보를 통해 자식 노드 C가 물려받을 링크 정보에 직접 인과가 존재할 경우에만 링크를 물려받게 된다. 주의할 점은 중복된 노드가 존재할 경우, 중복 노드는 다수의 노드를 갖는 것이 일반적이며 이는 분기의 경우의 수에 따라 다르다. 따라서 한 번에 두 개의 노드를 대상으로 함수 merge(sourceNode, targetNode, T)가 수행된다. 또한 근원 노드와 목표 노드를 결정하는 기준은 노드의 노드 깊이(Node-depth) 정보를 기반으로 하며, 깊이가 깊은 것이 근원 노드가 된다. 그 이유는 프로세스 로그에서 가장 처음 나타나는 위치가 해당 노드의 원본 위치를 나타내기 때문이다. 이는 이벤트 로그의 기록 방식에 의한 것으로 항상 한 번에 하나의 노드만을 기록하기 때문에 동시에 수행되는 경우에도 어느 하나가 먼저 혹은 늦게 나타나게 된다. 따라서 먼저 나타나는 경우가 발견되면 이것이 해당 노드의 본래 위치가 된다.

3. 검색 단계에서 중복 노드가 검출됨에 따라 함수 merge(sourceNode, targetNode, T)는 모든 중복 태스크에 대해서 반복적으로 수행된다.

알고리즘 2의 설명을 돕기 위해 표 1과 그림 4의 정보를 기준으로 만들어진 변형된 FP-트리에 알고리즘을 적용하는 과정을 보인다.

데이터만 insert된 초기상태의 변형된 FP-트리는 사실상 입력된 프로세스 로그의 형태를 따르기 때문에 이 과정을 통해 중복된 태스크가 제거된 중간 상태의 변형된 FP-트리로 변환하는 과정을 거친다. 위에서 언급한 알고리즘 2를 바탕으로 초기상태의 변형된 FP-트리에서 온전한 형태의 분기, 즉 병렬 구조를 검출할 수 있다. 아래 그림 10은 병렬 구조를 검출하는 과정을 나타낸다.

(I) 헤더 테이블의 태스크들에 대해서 중복된 태스크 검색을 실시한다. A의 경우 노드 링크를 따라 들어가면 하나의 노드만 존재하므로 병렬 구조에서 제외된다. B의 경우 두 개의 노드가 검출되므로 병렬 구조에 속한다.

(II) B 노드를 하나의 유일한 노드로 만드는 과정에서 병렬 구조가 검출된다. 이 과정에서 각 노드의 노드 링크를 유지하는 방법이 병렬 구조의 형태를 결정짓는다. 노드 링크는 차후에 알고리즘을 적용하기 위해 양방향 형태로 유지된다.

(III) 노드의 깊이(Depth)가 낮은 노드를 근원 노드(Source Node)로 둔다. 근원 노드는 목표 노드(Target

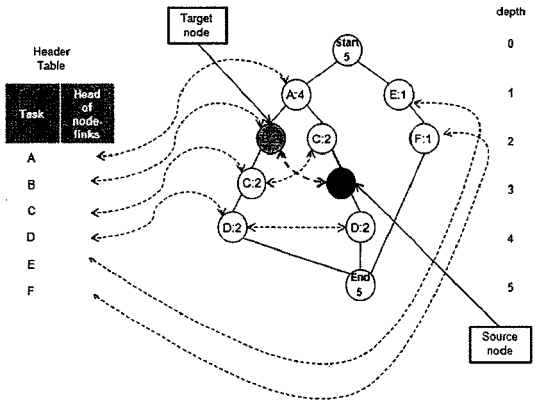

그림 10 중복 태스크 제거(Duplicated Task Removal)

Node)의 병합의 대상이 된다. 즉, 근원 노드는 목표 노드에 합쳐지고, 근원 노드는 트리에서 제거된다. 만약 중복된 노드의 개수가 2개 이상일 경우에는 가장 깊이가 낮은 노드에서 헤더 테이블에 가까운 방향으로 병합이 진행된다. 최종적으로는 깊이가 가장 얕은 노드만 남게 된다.

(IV) 제거되는 근원 노드의 상위 노드에 대한 링크와 노드 링크는 근원 노드의 아래 노드로 전이된다. 전이의 근거는 직접 인과 행렬에 있다. 직접 인과 행렬에 인과 관계가 존재할 경우 링크와 노드 링크는 아래 노드로 모두 전이된다. 위의 경우에는 그림 4의 직접 인과 행렬을 참조했을 때 목표 노드와 노드 D 사이에 인과 관계가 존재하므로 링크와 노드 링크는 노드 D로 전이된다. 위의 과정을 헤더 테이블의 모든 태스크에 대해서 진행하고 나면 그림 11과 같은 형태의 중복 태스크가 제거된 중간형태의 변형된 FP-트리가 생성된다.

알고리즘 2를 통해 알고리즘 1에서 생성한 데이터만 insert된 초기상태의 변형된 FP-트리를 중복된 태스크가 제거된 중간 상태의 변형된 FP-트리로 변환하였다. 이 과정에서 중복된 노드들은 제거되고, 완성에 가까운 형

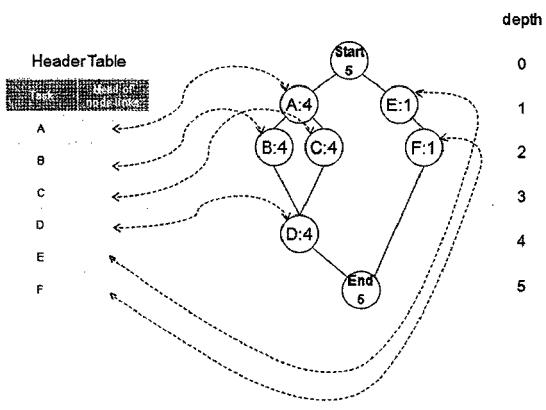

그림 11 중복 태스크가 제거된 중간상태의 변형된 FP-트리

태의 트리가 만들어졌다. 일반적으로 중간 상태의 변형된 FP-트리의 크기는 충분히 작고, 이를 통해 메모리상에 이 자료 구조를 지속적으로 유지하는 것이 가능하다.

본 논문에서 말하는 적응이란 말은 바로 여기에서 나왔다. 본 시스템의 중간 상태의 변형된 FP-트리를 유지하면서 추후에 새로운 프로세스 로그를 입력해서 더 나은 프로세스 모델을 찾는데 활용할 수 있다. 이는 프로세스 마이닝의 적합성 검사의 영역에 활용될 수 있다. 마지막으로 AND 게이트웨이로 이루어진 Parallel Split & Merge 패턴과 XOR 게이트웨이로 이루어진 Exclusive Choice & Merge 패턴 및 기존 연구에서 문제점으로 지적되었던 OR 게이트웨이를 사용한 Multiple Choice & Merge 패턴을 지원하기 위한 분기 노드에 대한 정보를 추가한다.

4.3 분기 타입 결정 알고리즘

이 절에서 소개하는 내용은 노드의 분기 특성을 결정짓는 알고리즘이다. 즉 특정 노드가 어떤 형태의 분기인지를 결정짓는다. 분기 노드의 타입을 결정짓는 방법은 다음 알고리즘을 따른다.

알고리즘 3 (분기 타입 결정)

입력(Input): 중복된 태스크가 제거된 중간 상태의 변형된 FP-트리

출력(Output): 분기노드가 결정된 완성 상태의 변형된 FP-트리

메소드(Method): 분기 노드에 AND/OR/XOR 등의 게이트웨이 타입을 다음과 같은 과정에 따라 부여한다.

```
/*
 * node : 타입을 결정해야 할 노드
 * T : 현재까지 생성된 FP-트리
 */
typeGrant(node, T)
{
  Node A, B, C;
  int n1, n2;

  A = node; //타입을 결정해야 할 노드
  B = node.childnodes[0]; //자식노드
  C = node.childnodes[1];
  .
  .
  .

  n1 = B.node_count % A.node_count;
  n2 = C.node_count % A.node_count;

  if(n1+n1 == A.node_count)
  {
    A.node_type = XOR;
  }
  else if((B.node_count + C.node_count) /
          A.childCount()== A.node_count)
  {
    A.node_type = AND;
  }
  else if((B.node_count + C.node_count) /
          A.childCount() != A.node_count)
  {
    A.node_type = OR;
  }
}
```

그림 12 변형된 FP-트리에서 분기 노드 타입 결정 과정

1. 분기 노드를 검색한다. 두 개 이상의 자식 노드를 갖는 노드는 모두 분기 노드(Split Node)에 해당한다. 또한 두 개 이상의 부모 노드를 갖는 노드를 병합 노드(Join Node)라고 한다. 병합 노드는 분기 노드와 짝(Pair)를 이루게 된다. 검색은 깊이 우선 검색(Depth First Search)에 따라 수행된다.

2. 중복된 태스크가 제거된 중간 상태의 변형된 FP-트리로부터 분기 노드가 검색되면, 함수 typeGrant(node, T)가 수행된다. 이 함수는 분기 노드에 타입을 부여하게 되는데 부여의 근거는 다음 보조 정리를 따르며, 노드 카운트 정보를 활용한다. 노드 카운트는 분기 노드에 타입을 부여하기 위한 근거가 된다. 분기 노드 A가 자식 노드 B, C를 갖고 있다고 가정한다. 노드 A의 타입은 다음 내용을 따르게 된다. n1과 n2는 다음과 같다.

(노드 B의 노드 카운트) % (노드 A의 노드 카운트) = n1

(노드 C의 노드 카운트) % (노드 A의 노드 카운트) = n2

n1과 n2의 합이 A의 노트 카운트와 같다면 노드 A의 타입은 Exclusive-OR가 된다. Exclusive-OR 타입은 분기에 접어들 경우, 반드시 한 쪽만 수행되기 때문에 위의 식이 성립한다.

n1과 n2의 합이 A의 노드 카운트와 같지 않다면, 노드 A는 AND 혹은 OR가 될 수 있는 잠재적인 가능성을 갖는다. 다음 방법을 통해 AND 혹은 OR 타입을 결정할 수 있다.

노드 B의 노드 카운트와 노드 C의 노드 카운트의 합에서 A의 자식 노드의 수를 나눈 몫이 A의 노드 카운트와 같으며 노드 B와 노드 C의 노드카운트 수가 같다면, A는 AND라고 할 수 있다. 만약 노드 B의 노드 카운트와 노드 C의 노드 카운트의 합에서 A의 자식 노드의 수를 나눈 몫이 A의 노드 카운트와 같지 않고 노드 B와 노드 C의 노드 카운트 수가 같지 않으면, A의 타입은 OR가 된다.

분기 노드의 타입을 결정짓는 방법을 간략히 정리하면 표 2의 내용과 같다.

3. 앞서 언급한 바와 같이 병합 노드는 분기 노드와 짝을 이룬다. 만약 split-join 쌍 사이에 또 다른 split-join 쌍의 노드가 존재한다면 함수 typeGrant(node, T)는 재귀적으로 수행된다.

그림 13은 분기노드 타입을 결정하는 과정을 나타낸다.

이를 통해 분기노드가 결정된 완성 상태의 변형된 FP-트리가 생성된다. 최종 완성 상태의 변형된 FP-트리는 곧 프로세스 모델이 되며 이로써 알고리즘의 수행은 종료된다.

표 2 분기 노드 타입 결정

근거	분기 종류
n1 + n2 = 노드 A의 노드 카운트	XOR 분기
(노드 B의 노드 카운트 + 노드 C의 노드 카운트) / 노드 A의 자식 노드의 수 = 노드 A의 노드 카운트 이며 (노드 B의 노드 카운트 ==노드 C의 노드 카운트)인 경우	AND 분기
(노드 B의 노드 카운트 + 노드 C의 노드 카운트) / 노드 A의 자식 노드의 수 != 노드 A의 노드 카운트 이며 (노드 B의 노드 카운트 !=노드 C의 노드 카운트)인 경우	OR 분기

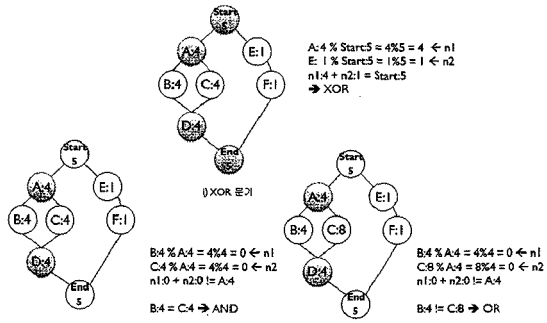

그림 13 변형된 FP-트리에서 분기 노드 타입 결정 과정

4.4 변형된 FP-트리에서 BPMN으로의 매핑

본 논문에서 제시한 변형된 FP-트리는 일반적인 비즈니스 프로세스 모델링 방법으로 매핑할 수 있다. 본 논문에서는 비즈니스 프로세스를 모델링하기 위한 표준 표현인 BPMN(Business Process Modeling Notation)을 사용한다[10].

그림 14는 변형된 FP-트리에서 BPMN으로 매핑한 것을 나타낸다. 4장에서 설명했던 노드의 특성을 이용해서 어떤 종류의 다이어그램이 사용될 것인지를 결정한

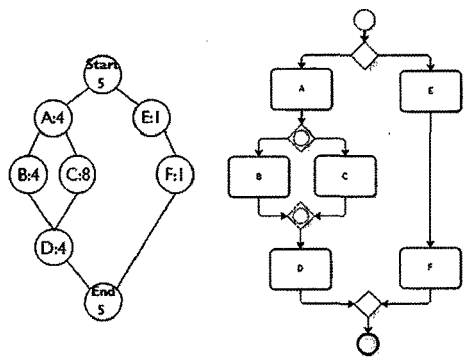

그림 14 변형된 FP-트리에서 BPMN으로의 매핑

다. 그림 8에서는 시작 노드와 끝 노드, 분기와 병합 노드를 변형된 FP-트리의 노드 정보를 이용해 매핑하고 있다. 특히, 노드의 형태를 결정짓는 근거는 이미 4장에서 소개한 바와 같으며, 트리의 노드에 저장된 형태 정보를 이용해 직접적으로 매핑 하는 것이 가능하다. 하지만 BPMN에서 사용되는 모든 다이어그램에 대한 활용도가 아직은 미비하다. 단순히 프로세스에서 태스크의 진행 결과만이 기록되는 프로세스 로그를 통해서는 의미를 갖는 정보(Semantic Information)를 추출하는 것이 매우 어렵다[11].

5. 실 험

본 논문에서 제시한 알고리즘을 평가하고 몇 가지 케이스를 비교하기 위해 자바 프로그래밍 언어로 알고리즘을 구현하였으며 Intel Core2Duo 2.66GHz의 윈도우즈 서버 2008 운영체제에서 실험을 실시하였다. 본 논문에서 제시한 알고리즘을 평가하기 위해 프로세스 일치도 및 프로세스 검출 시간을 기존 연구 중 Medeiros et al.[9]이 제시한 유전자 알고리즘과 비교한 후 제시하였다.

표 3에서 제시한 성능비교를 위한 프로세스 로그 정보는 전체 52개의 로그 수를 가지며 6개의 프로세스 인스턴스 케이스로 이루어져있다. 표 3에서 제시된 프로세스 로그 정보에 대한 직접 인과 행렬은 표 4와 같다.

실험에서는 Case 1부터 Case 4까지의 프로세스 로그 정보를 먼저 적용하고 Case 5와 Case 6의 정보가 차후

표 3 성능 비교를 위한 프로세스 로그 정보

Case Identifier	Task Identifier	Case Identifier	Task Identifier
Case 1	Task A	Case 2	Task E
Case 1	Task B	Case 1	Task H
Case 2	Task A	Case 1	Task J
Case 3	Task A	Case 3	Task E
Case 1	Task C	Case 2	Task F
Case 3	Task B	Case 3	Task F
Case 1	Task D	Case 4	Task D
Case 2	Task B	Case 2	Task G
Case 4	Task A	Case 2	Task I
Case 1	Task E	Case 3	Task F
Case 2	Task D	Case 2	Task J
Case 3	Task D	Case 4	Task E
Case 1	Task F	Case 3	Task G
Case 1	Task F	Case 4	Task F
Case 4	Task B	Case 3	Task I
Case 5	Task A	Case 4	Task G
Case 2	Task C	Case 4	Task H
Case 3	Task C	Case 3	Task J
Case 4	Task C	Case 4	Task J
Case 1	Task G	.....	.....

표 4 성능 비교를 위한 프로세스 로그 정보에 대한 직접 인과 행렬

Case Identifier	Task Identifier
Case 1	A -> B -> C -> D E -> F -> G -> H -> J
Case 2	A -> B -> D -> C E -> F -> G -> I -> J
Case 3	A -> B -> D -> C - E -> F -> G -> I -> J
Case 4	A -> B -> C -> D - E -> F -> G -> H -> J
Case 5	A -> B -> C -> D - E -> F -> G -> I -> J
Case 6	A -> B -> D -> C - E -> F -> G -> H -> J

	Start	A	B	C	D	E	F	G	H	I	J	End
Start	0	1	0	0	0	0	0	0	0	0	0	0
A	0	0	1	0	0	0	0	0	0	0	0	0
B	0	0	0	1	1	0	0	0	0	0	0	0
C	0	0	0	0	0	1	0	0	0	0	0	0
D	0	0	0	0	0	1	0	0	0	0	0	0
E	0	0	0	0	0	0	1	0	0	0	0	0
F	0	0	0	0	0	0	0	1	0	0	0	0
G	0	0	0	0	0	0	0	0	1	1	0	0
H	0	0	0	0	0	0	0	0	0	0	1	0
I	0	0	0	0	0	0	0	0	0	0	1	0
J	0	0	0	0	0	0	0	0	0	0	0	1
End	0	0	0	0	0	0	0	0	0	0	0	0

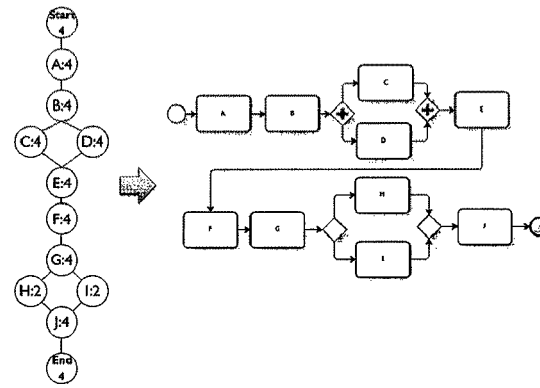

그림 15 표 3의 프로세스 결과

에 업데이트 된다는 전제 하에 프로세스 일치도와 프로세스 검색시간을 비교하였다.

프로세스 일치도는 프로세스 마이닝 알고리즘을 통하여 검출된 프로세스 모델과 실제 프로세스의 차이점을 검사하기 위한 방법으로써 본 논문에서 사용된 프로세스 일치도 식은 다음과 같다[12].

$$\text{프로세스 일치도} = 1 - \frac{\text{업무간잘못연결된수}}{\sum_{i=1}^{n}(\text{업무수}(n)-1)}$$

성능 비교를 위해 제시한 프로세스 로그 정보 중 Case 1부터 Case 4까지의 정보를 이용하여 실험을 진행하였

표 5 Case 1부터 Case 4까지의 정보 적용

프로세스 로그 수/ Case 수	Medeiros et al.[9]		변형된 FP-트리	
	프로세스 일치도	시간(초)	프로세스 일치도	시간(초)
39/4	100%	9.08	100%	0.14

표 6 Case5와 Case 6까지의 정보를 적용

프로세스 로그 수/ Case 수	Medeiros et al.[9]		변형된 FP-트리	
	프로세스 일치도	시간(초)	프로세스 일치도	시간(초)
56/6	83.2%	634.23	100%	0.27

을 때의 결과는 다음과 같다.

표 5는 Case 1부터 Case 4까지의 로그정보를 이용하여 각 알고리즘에 적용했을 때의 프로세스 일치도와 시간에 대해 나타낸 표이다. Case 1부터 Case 4까지의 로그정보를 각 알고리즘에 적용했을 때의 프로세스 일치도는 두 알고리즘 모두 100%의 일치도를 보였다. 하지만 표 6에서 Case 1부터 Case 6까지의 정보를 각 알고리즘에 적용했을 때에는 Medeiros et al.[9]의 알고리즘은 83.2%의 프로세스 일치도를 보인 반면 변형된 FP-트리를 이용한 알고리즘은 100%의 일치도를 보였다. 또한 Medeiros et al.[9]의 알고리즘은 로그 정보 전체를 재 스캔해야 하기 때문에 표 5에서의 시간보다 많은 시간이 소요되었다. 하지만 변형된 FP-트리를 이용한 알고리즘은 표 5에서 구축된 FP-트리에 새로 추가된 로그 정보를 이용하여 재구성하는 방식을 사용하여 적은 시간 안에 프로세스 모델을 검출할 수 있었다.

이처럼 변형된 FP-트리에 기반을 둔 프로세스 마이닝 알고리즘은 비즈니스 프로세스가 주로 사용되는 정보 시스템에서 비즈니스 프로세스 재설계를 위한 목적으로 활용하기 위한 장점을 갖고 있다. 하지만 변형된 FP-트리는 재 스캔 과정 없이 프로세스 로그에서 일어나는 모든 트랜잭션을 FP-트리 내부에 저장 하여야 하기 때문에 많은 메모리를 요구하는 단점이 발생한다[5]. 이러한 단점이 본 논문에서 제안한 알고리즘에 얼마만큼 영향을 주는지 알아보기 위하여 몇 가지 특별한 경우를 선택하여 몇 가지 타입의 프로세스를 선정하여 이벤트 로그를 추출한 후 알고리즘에 적용시키는 실험을 수행하였다.

5.1 트리 내부 노드의 메모리 점유율

실험 이전에 분기 노드가 갖는 자식의 수가 일정 수 이상 많아지면 merge 연산의 횟수가 크게 늘어나고, 많은 수의 메모리를 차지하게 될 것이라 예상하였다. 하지만 이는 충분히 수용할 수 있는 범위의 것으로 생각되었다. 실험에 사용된 프로세스 로그 데이터는 페이지관계상 모두 표현할 수 없기 때문에 표 7에서는 실험에 사용된 프로세스 로그 정보에 대한 직접 인과 관계를 제시 하였다.

표 7 메모리 점유율 비교를 위한 프로세스 로그 정보의 직접 인과 관계

	Case Identifier	Task Identifier
Type A	Case 1	A -> B -> F
	Case 2	A -> C -> F
	Case 3	A-> D -> F
	Case 4	A -> E -> F

	Case Identifier	Task Identifier
Type B	Case 1	A -> D -> J
	Case 2	A -> E -> J
	Case 3	B-> F -> J
	Case 4	B -> G -> J
	Case 5	C -> H -> J
	Case 6	C -> I -> J

	Case Identifier	Task Identifier
Type C	Case 1	A -> B -> F -> I -> J
	Case 2	A -> B -> G -> I -> J
	Case 3	A -> B -> H -> I -> J
	Case 4	A -> C -> J
	Case 5	A-> D -> J
	Case 6	A -> E -> J

	Case Identifier	Task Identifier
Type D	Case 1	A -> B -> F -> L -> N
	Case 2	A -> B -> G -> L -> N
	Case 3	A -> B -> H -> L -> N
	Case 4	A -> B -> I -> L -> N
	Case 5	A -> C -> J -> M -> N
	Case 6	A -> C -> K-> M -> N
	Case 7	A-> D -> N
	Case 8	A -> E -> N

	Case Identifier	Task Identifier
Type E	Case 1	A -> B -> F -> R -> W
	Case 2	A -> B -> G -> R -> W
	Case 3	A -> B -> H -> R -> W
	Case 4	A -> B -> I -> R -> W
	Case 5	A -> C -> J -> S -> W
	Case 6	A -> C -> K-> S -> W
	Case 7	A -> C -> L-> S -> W
	Case 8	A -> D -> M-> T -> W
	Case 9	A -> D -> N-> T -> W
	Case 10	A -> D -> O-> T -> W
	Case 11	A -> E -> P-> T -> W
	Case 12	A -> E -> Q-> T -> W

표 7에서 타입 A는 분기는 하나이지만 분기 노드의 자식 노드 수가 많은 경우를 선정했다. 타입 B는 타입 A와 같은 형태의 분기를 시작 노드가 여러 개 갖는 경우를 선정했다. 타입 C는 A와 같은 형태의 분기를 중첩

되게 설계한 형태를 갖는다. 이는 분기 내에 분기를 포함하는 형태로 되어 있다. 마지막으로 타입 D와 E는 타입 C에 추가적으로 분기를 더하고, 분기당 노드의 수를 확장한 것이다.

본 논문에서 제시하는 알고리즘의 가장 큰 장점인 적합성 검사에 유용한 중복된 태스크가 제거된 중간 상태의 변형된 FP-트리의 유용성을 측정하기 위해 메모리 점유율을 측정한 결과 그림 16의 실험 결과와 같이 메모리상에 상주시키더라도 문제가 없는 것으로 나타났다. 중간 상태의 변형된 FP-트리는 프로세스 로그를 축소된 형태로 저장하는 특성을 갖기 때문에 분기 자체가 노드 개수에 영향을 끼치지 못한다.

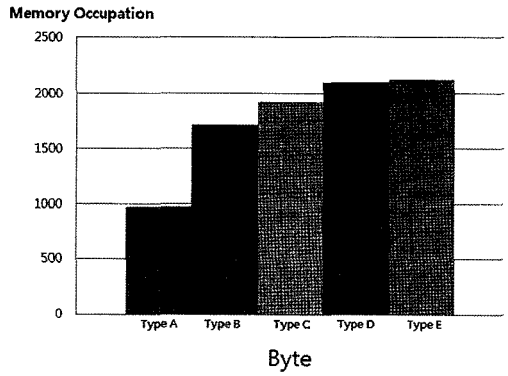

그림 16 메모리 점유율 비교

5.2 Merge 연산 수행 회수 비교

그림 16의 실험 결과를 봤을 때 Type C, Type D 및 Type E와 같이 분기의 중첩이 깊어지거나, 분기의 노드가 많아 질 경우에는 연산 횟수가 기하급수적으로 커지는 문제점이 발견되었다. 일반적인 비즈니스 프로세스의 경우 분기는 한정적이고 그 복잡도가 크지 않지만, 위와 같은 경우가 생성된다면 연산 횟수가 크게 증가하는 문제가 생길 수 있다.

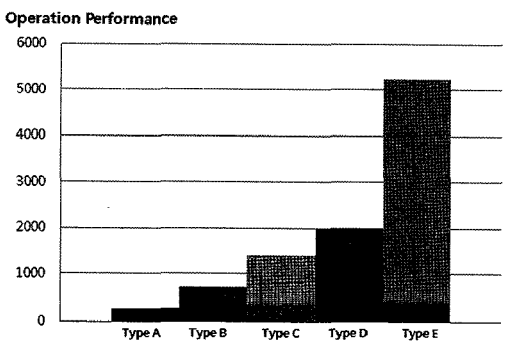

그림 17 Merge 연산 수행 회수 비교

5.3 트리의 Update 회수

분기의 깊이에 따라서 초기 삽입 연산 시의 업데이트 횟수 변화 추이를 살펴보면서, 이를 통해 트리의 생성에 걸리는 부하를 또 다른 방법으로 예측할 수 있다. Merge 연산의 횟수는 트리를 생성한 후 중간 상태의 변형된 FP-트리로 변환하기 위한 작업의 부하이지만 트리의 업데이트 회수를 측정하는 작업은 데이터만 insert된 초기상태의 변형된 FP-트리를 생성하기 위한 작업의 부하를 측정하는 실험이다. 사실상 마지막 분기노드가 결정된 완성 상태의 변형된 FP-트리를 생성하는 일에는 트리의 업데이트 등의 수정이 요구되지는 않고 단지 읽기 작업만이 있으므로 이는 트리의 구조에 변화가 생기는 작업에 비할 바가 아니다.

트리의 업데이트는 주로 카운트 연산에 해당하며 혹은 새로운 노드를 생성하는 일이 이에 포함된다. 분기 발생 노드의 경우 하위 노드의 개수가 많으면 많을수록 카운트 회수뿐 만 아니라 이미 입력되어 있던 노드들에 대해서도 새로운 노드를 생성해야한다. 이에 대해서 노드 링크를 재설정 하는 일을 수행해주어야 하므로 분기의 중첩이 깊거나, 분기의 노드가 많은 Type E와 같은 예제에서는 일정량의 오버헤드가 그림 18과 같이 지속적으로 발생하게 된다.

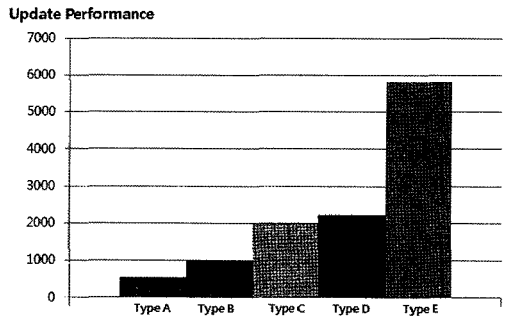

그림 18 Update 수행 회수 비교

6. 결 론

프로세스 마이닝은 실제 비즈니스 프로세스가 실행됨에 따라 생성되는 프로세스 로그를 바탕으로 완성된 비즈니스 모델을 생성하는 방법론이다. 단순 순차 정보인 프로세스 로그를 통해 생성된 모델이 반드시 정답이라고 할 수는 없다. 하지만 프로세스 마이닝을 통해 생성된 비즈니스 모델은 비즈니스 재설계의 근거로 활용될 수 있다. 설계 비즈니스 프로세스에서 생각하지 못했던 과정이나 혹은 불필요하게 중복되게 사용되었던 태스크를 찾아내어 성능 향상을 꾀할 수 있다[8].

본 논문에서 제시한 알고리즘은 잘 구성된 비즈니스

프로세스(Well-Structured Business Process)에서 추출한 프로세스 로그에 대해서 좋은 성능을 보인다. 실제 비즈니스 프로세스에서 빈번하게 발생하는 순차 루프(Sequence Loop)를 다루기 위한 방법을 현재 연구 중에 있다. 루프에 대한 처리는 전처리(Pre-processing)과정을 통해 처리하는 방법을 연구 중이다[11].

제시한 프로세스 마이닝 알고리즘은 대량의 프로세스 로그를 읽어들여 처리하는 것에 유리하며 자료구조를 이용한 알고리즘을 적용하기 때문에 이후에 자료구조의 변경을 통해 알고리즘의 완성도를 꾀하는 것이 용이하다. 또한 일반적인 형태의 프로세스 다이어그램 자체를 생성하기 때문에 어느 형태의 표기 방법으로도 매핑하는 것이 가능하다.

실험을 통해 검증된 사항을 보면 중간 상태의 변형된 FP-트리를 이용해 적합성 검사의 영역에서 활용하는 것이 가능한 것으로 간주된다. 하지만 특정 케이스의 경우 연산의 횟수가 급격히 증가하는 양상을 보이기 때문에 어떤 경우에는 성능의 저하를 가져올 수도 있다.

앞으로의 연구에서는 순차 루프의 검출과 특정 케이스에 의존하지 않는 강건한 알고리즘적 특성을 갖도록 진행될 예정이다. 또한, 현재 자기 루프의 검출은 최종 작업이 진행 중이다. 또한 결과 값에 대한 검증관련 연구가 학계에서 진행 중이기 때문에 이 분야에 대한 고려가 필요한 것으로 생각된다[13-15].

참 고 문 헌

[1] W.M.P. van der Aalst, A.J.M.M. Weijters, "Process Mining: A Research Agenda," *Computers in Industry*, vol.53, no.3, pp.231-244, 2004.

[2] W.M.P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, H.M.W. Verbeek, "Conformance Checking of Service Behavior," *ACM Transactions on Internet Technology (TOIT)*, vol.8, no.3, 2008.

[3] W.M.P. van der Aalst, "Trends in Business Process Analysis: From Verification to Process Mining," *Proceedings of the 9th International Conference on Enterprise Information Systems (ICEIS 2007)*, pp.12-22, 2007.

[4] XIE Yi-wu, LI Xiao-wan, Chen Yan, "The Research on the Usage of Business Process Mining in the Implementation of BPR," *Proceedings of the 2007 IFIP International Conference on Network and Parallel Computing Workshops*, pp.995-1000, 2007.

[5] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without condidate generation," *Proceedings of 2000 ACM SIGMOD Int. Conf. Management of Data(SIGMOD'00)*, Dallas, Tx, pp.1-12, 2000.

[6] A. Tiwari, C.J. Turner, B. Majeed, "A review of business process mining : state-of-the-art and future trends," *Business Process Management Journal*, vol.14, no.1, pp.5-22, 2008.

[7] W.M.P. van der Aalst, "Trends In Business Process Analysis : From Verification to Process Mining," *Proceedings of the 9th International Conference on Enterprise Information Systems (ICEIS 2007)*, pp.12-22, 2007.

[8] W.M.P. van der Aalst and C.W. Günther, "Finding Structure in Unstructured Processes: The Case for Process Mining," *Proceedings the 7th International Conference on Applications of Concurrency to System Design, ACSD 2007*, pp.3-12, 2007.

[9] A.K. Medeiros, A.J.M.M. Weijters, W.M.P. van der Aalst, "Genetic process mining: an experimental evaluation," *Journal of Data Mining and Knowledge Discovery*, vol.14, no.2, pp.245-304, 2007.

[10] Object Management Group/Business Process Management Initiative, "BPMN 1.1: OMG Specification," February, 2008.

[11] A.K. Medeiros, Antonella Guzzo, Gianluigi Greco, W.M.P. van der Aalst, A.J.M.M. Weijters, Boudewijn F. van Dongen, Domenico Sacca, "Process Mining Based on Clustering: A Quest for Precision," *BPM Workshops, LNCS 4928*, pp.17-29, 2008.

[12] S. Chung, S. Kwon, "A Process Mining using Association Rule and Sequence Pattern(in korean)," *Journal of the Society of Korea Industrial and Systems Engineering*, vol.31, no.2, pp.104-111, 2008.

[13] M. Funk, A. Rozinat, A.K. Medeiros, P.H.A. van der Putten, H. Corporaal, W.M.P. van der Aalst, "Semantic Concepts in Product Usage Monitoring and Analysis," *ESR-2008-10, Report of Group of Electronics Systems, Department of Electrical Engineering, TU/e*, 2008.

[14] A. Rozinat, M. Veloso, W.M.P. van der Aalst "Evaluating the Quality of Discovered Process Models," *Proceedings of Induction of Process Models, IPM workshop*, pp.45-52, 2008.

[15] A. Rozinat, R.S. Mans, M. Song, W.M.P. van der Aalst, "Discovering Simulation Models, BETA Working Paper Series," WP 223, Eindhoven University of Technology, Eindhoven, 2007.

김 건 우

2006년 호주 뉴캐슬 대학교 컴퓨터공학과 학사. 2007년 호주 뉴캐슬 대학교 정보공학과 석사. 2008년~현재 한양대학교 컴퓨터공학과 박사과정. 관심분야는 BPM, 데이터베이스, 데이터마이닝, E-Business, RFID

이 승 훈

2009년 한양대학교 전자컴퓨터공학부 학사. 2009년~현재 한양대학교 컴퓨터공학과 석사과정. 관심분야는 BPM, 데이터베이스, 데이터마이닝, E-Business

김 재 형

2007년 한양대학교 전자컴퓨터공학부 학사 2009년 한양대학교 컴퓨터공학과 석사 2009년~현재 알티베이스 DBMS R&D 개발 본부. 관심분야는 Mobile, 데이터베이스, 데이터마이닝

서 혜 명

1989년 한양대학교 생화학과 학사. 1991년 한양대학교 생화학과 석사. 1999년 테네시 멤피스 대학교 신경과학과 박사 2003년 하버드 메디컬 스쿨 박사후 연구원. 2004년~현재 한양대학교 분자생명과학과 부교수. 관심분야는 Bioinformatic, Huntington Disease, Alzhemier Disease, GABAergic Differentiation, Neuroinflammation

손 진 현

1996년 서강대학교 전산학과 학사. 1998년 한국과학기술원 전산학과 석사. 2001년 한국과학기술원 전자전산학과 박사 2002년 한국과학기술원 전자전산학과 박사후 연구원. 2002년~현재 한양대학교 컴퓨터공학과 부교수. 관심분야는 데이터베이스, E-Business, 유비쿼터스 컴퓨팅, 임베디드 시스템