韓國컴퓨터情報學會 論文誌
第14卷 第4號, 2009. 4.

2009-14-4-4-2

비즈니스 서비스 식별을 위한 변형 순차패턴 마이닝 알고리즘

이 정 원*

Adapted Sequential Pattern Mining Algorithms for Business Service Identification

Jung-Won Lee*

요 약

SOA를 도입하는 하향식(top-down) 방법은 온톨로지를 기반으로 서비스를 분석하고 설계하는 서비스 모델링 단계를 핵심으로 봄으로써 SOA의 장점을 가장 잘 반영할 수 있는 방법으로 권장되고 있다. 그러나 대부분의 기업들은 하향식 방법이 최상이라는 것을 알면서도 기업 이윤 창출에 단기적인 효과가 드러나지 않고 도입 초기에 개발 시간과 비용이 증대되므로 이를 꺼리게 된다. 특히 잘 정의된 컴포넌트 시스템을 이미 사용하고 있는 경우에 더욱 그러하다. 따라서 본 논문에서는 기존의 잘 정의된 컴포넌트 시스템을 최대한 이용할 수 있는 상향식 (bottom-up) 서비스 식별 방법을 제안한다. GUI는 직접 사용자의 입력을 받아 들여 이벤트를 발생시킨다는 점에 착안하여 이벤트의 경로를 연결하면 비즈니스 프로세스에 근사시킬 수 있다. 따라서 컴포넌트와 상호작용하는 GUI의 이벤트 수를 기준으로 핵심 GUI를 선정하고 핵심 GUI로부터 연결되는 이벤트 경로를 대상으로 기존의 순차패턴 마이닝 알고리즘을 변형하여 사용자의 서비스 사용 패턴을 추출한다. 실험결과 추출된 이벤트 패턴에 응집도를 적용하여 다양한 크기의 비즈니스 서비스를 식별할 수 있음을 보였다.

Abstract

The top-down method for SOA delivery is recommended as a best way to take advantage of SOA. The core step of SOA delivery is the step of service modeling including service analysis and design based on ontology. Most enterprises know that the top-down approach is the best but they are hesitant to employ it because it requires them to invest a great deal of time and money without

• 제1저자 : 이정원
• 투고일 : 2009. 02. 27, 심사일 : 2009. 03. 20, 게재확정일 : 2009. 03. 25.
* 아주대학교 정보통신대학 전자공학부 교수
※ 본 연구는 지식경제 프론티어기술개발사업의 일환으로 추진되고있는 지식경제부의 유비쿼터스컴퓨팅및 네트워크원천기반기술개발사업의09C1-T3-10M 과제로 지원된 것임.
※ 이 연구는 2008학년도 아주대학교 교내연구비 지원(20083770)에 의하여 이루어졌음.

it showing any immediate results, particularly because they use well-defined component based systems. In this paper, we propose a service identification method to use a well-defined components maximally as a bottom-up approach. We assume that user's inputs generates events on a GUI and the approximate business process can be obtained from concatenating the event paths. We first find the core GUIs which have many outgoing event calls and form event paths by concatenating the event calls between the GUIs. Next, we adapt sequential pattern mining algorithms to find the maximal frequent event paths. As an experiment, we obtained business services with various granularity by applying a cohesion metric to extracted frequent event paths.

▸ Keyword : 서비스-지향 구조(Service-Oriented Architecture), 순차 패턴(Sequential Patterns), 서비스 식별(Service Identification), 서비스 모델링(Service Modeling)

Ⅰ. 서 론

서비스-지향 구조(SOA)는 느슨하고 협업을 중심으로 하는 서비스 집합을 중심으로 소프트웨어 개발을 촉진시키는 개발 패러다임이다 [1]. 엔터프라이즈 수준에서 SOA를 도입하는 전략은 크게 하향식(top-down), 상향식(bottom-up), 그리고 혼합식(hybrid or meet-in-the-middle) 접근방식으로 나눌 수 있다 [2]. 하향식 방법은 SOA의 장점을 가장 잘 반영할 수 있는 방법으로서 "분석 최우선(analysis first)" 접근이다. 많은 기업들은 하향식이 SOA의 장점을 가장 잘 살릴 수 있는 전략임을 알면서도 기업 이윤 창출에 단기적인 효과가 드러나지 않고 SOA 도입의 초기단계에서 개발 시간과 비용이 증대되므로 이 방식을 꺼리게 된다. 특히 기존에 잘 동작하던 컴포넌트 기반의 시스템을 SOA로 전향하는 경우엔 더욱 그러하다. 상향식은 온톨로지를 기반으로 한 비즈니스 모델로부터 출발한 것이 아니라 레거시 시스템(응용 프로그램 단위)으로부터 서비스를 도출해 나가는 방식으로서 특별히 서비스-지향 분석 및 설계 단계를 반영할 수 있는 절차와 방법이 고려되고 있지 않다. 따라서 비용과 시간이 많이 소모되는 하향식을 채택하는 모험보다는 이미 잘 정의된 컴포넌트 기반의 시스템의 구조를 크게 변화시키지 않으면서, 즉 최대한 재사용하면서 서비스 모델을 도출해 내면서 SOA의 장점을 도입할 수 없을까에 좀 더 관심을 가질 수밖에 없다.

최근 많은 기업과 공공 기관은 SOA 도입의 장점을 탐색하는 단계에서 벗어나 이제 파일럿 프로젝트를 통해 구체적인 SOA 도입 전략을 세우고 있다. 그러나 그들이 기존의 컴포넌트들을 단순히 SOA 구현 도구 위에 서비스들을 옮기는 것으로 SOA를 도입한다면 다음과 같은 문제점이 발생한다.

첫째, 서비스가 단순히 컴포넌트 그 자체 혹은 컴포넌트의 그룹으로 정의될 위험이 있다. 컴포넌트는 기능 단위인 반면, 서비스는 그 자체로 비즈니스 가치를 창출할 수 있는 단위로 식별되어야 한다 [2]. 예를 들어 'invoice processing'은 서비스가 될 수 있지만, 'transform XML documents to native format'은 기능 단위라고 볼 수 있다. 따라서 단순한 기능단위의 조합이 아닌 단독으로 비즈니스 목적을 달성할 수 있는 단위로서 서비스를 식별해야 한다.

둘째, SOA에서 추천하는 이상적인 서비스 레이어를 따르기 어렵다. SOA에서는 서비스 추상화를 위한 레이어를 비즈니스 서비스, 애플리케이션 서비스, 그리고 이를 중재하는 오케스트레이션(orchestration) 서비스 레이어로 나누고 있다 [2]. 이렇게 서비스의 의미를 계층적으로 나눔으로써 재사용성과 서비스 합성 능력을 증강시킬 수 있게 된다. 그러나 컴포넌트 자체를 바로 서비스로 정의하는 것은 세 레이어중 애플리케이션 서비스 레이어만을 고려하는 것과 같다.

따라서 기존의 컴포넌트 시스템을 크게 파괴하지 않는 범위내에서 서비스-지향 분석과 설계를 반영할 수 있는 방법이 필요하다. 서비스-지향 분석의 시작은 온톨로지를 사용한 비즈니스 프로세스를 고려하는 것이다. 비즈니스 프로세스는 특정 목적을 달성하기 위해 상호 연관된 작업들로 정의된다 [3]. 한편, 컴포넌트들 사이의 상호 연관성은 이벤트 호출에 의해 트리거 된다. 이벤트 호출은 명시적이기도 하고 내부에 숨겨져 있기도 하다. 명시적인 컴포넌트간의 이벤트 호출은 메뉴나 버튼과 같은 GUI(Graphical User Interface)상의 그래픽 요소에 의해 사용자의 요청을 받아 들여 관련된 컴포넌트를 실행시키는 경우를 말한다. 숨겨진 이벤트 호출은 GUI의 상호작용과는 상관없이 한 컴포넌트에서 직접 다른 컴포넌트의 기능을 호출하는 경우를 말한다. 따라서 하나의 GUI(이제부터, GUI는 하나의 윈도우나 화면 단위를 일컫는 용어로 사용한다.)는 그래픽 요소를 사용하는 다수의 명시적인 이벤트 호출과 또 그 명시적인 호출이 내부적으로 직접 컴포넌트들을 호출하는 암시적인 호출을 포함하게 된다. 예를

들어 사용자가 'invoice processing'을 위한 버튼을 클릭했다고 하자. 이 명시적인 호출에 의해 다음과 같은 일련의 컴포넌트들 - 'poll network folder for invoice', 'retrieve electronic invoice', 'transform electronic invoice to XML document', 그리고 'check validity of invoice document, if valid, end process' - 을 암시적으로 호출하게 된다. 즉, GUI들 사이의 숨겨진 호출을 넘어 명시적인 호출에 초첨을 두고 모두 연결하면 컴포넌트 단위라기 보단 'invoice processing'과 같은 서비스 단위를 추출할 수 있다.

대부분의 GUI는 일반적으로 많은 컴포넌트들과 상호작용한다. 하나의 GUI로부터 다른 GUI로의 이벤트 호출은 하나의 경로로 생성되고 이벤트 경로를 통하여 시스템을 사용하는 사용 패턴을 추적할 수 있다. 따라서 특정 목적을 달성하기 위한 이벤트 경로의 패턴은 특정 목적을 해결하기 위한 비즈니스 프로세스에 근사한(approximated) 것으로 간주할 수 있다. 따라서 본 논문에서는 컴포넌트의 GUI정보를 이용하여 비즈니스 프로세스에 근사한 이벤트 경로를 추출하고 이를 기반으로 변형된 순차패턴 마이닝 알고리즘을 설계하여 상향식 서비스 식별 방법을 제안한다. 먼저 핵심이 되는 GUI를 선정하고 핵심 GUI로부터 출발하는 이벤트 호출을 연결하여 이벤트 경로를 만든다. 다음으로 순차패턴 마이닝 알고리즘을 적용하여 GUI 클러스터들간의 최대 공유 이벤트 경로를 발견하고 공유정도에 따라 서비스 후보를 결정한다. 실험을 위해 실제 대기업에서 운용되고 있는 대형의 잘 정의된 컴포넌트 시스템인 MIS(Management Information System)를 대상으로 본 논문에서 제안한 알고리즘을 적용해 보았다. 수치화를 위해 응집도 메트릭을 정의하여 적용해 본 결과 각 서비스의 응집도의 분산값이 커지면서 쉽게 서비스를 식별할 수 있었다.

II. 관련 연구

1. 서비스-기반 개발 방법

잘 모델링된 서비스는 SOA의 재사용성과 유지보수성을 결정하므로 SOA 개발 프로세스단계에서 서비스를 식별하는 것은 매우 중요하다. 서비스 식별이란 기존의 시스템 혹은 비즈니스 도메인으로부터 재사용 자원인 서비스를 식별하는 것으로 비즈니스 도메인을 고려하여야 하고, 재사용성을 위해 비즈니스 서비스 모델과 적합하면서 기존 객체나 컴포넌트보다 큰 입자로 서비스를 식별해야 한다. 기존의 서비스 식별에 관한 연구는 서비스-기반 개발 방법론으로 SODA(Service Oriented Development of Applications)[4], SOUP(Service Oriented Unified Process)[5], SOMA(Service Oriented Modeling and Architecture)[6], SOAD(Service Oriented Analysis and Design)[6]를 들 수 있으며 다음 표 1과 같이 정리하였다.

표 1. 서비스 개발 방법
Table 1. Service-Based Development

방법론	특징
SODA [4]	• CBD, 분산시스템 개발, SOA의 공통 요소를 뽑아 적용 • SOA를 구현하기 위한 서비스 설계, 개발,조합에 관한 원칙 정의 • (문제점) 서비스 단위나 세부 절차에 대하여 구체적으로 명시하지 않음
SOAD [6]	• OOAD, EA, BPM 같은 기존의 모델링 개념들을 SOA에 적용시킴 • 하향식과 상향식을 연결하여 기존시스템의 통합을 위한 아이디어 제시 • (문제점) 세부적인 절차나 활동을 명시하지 않음
SOUP [5]	• RUP와 XP의 특징만을 모아서 SOA에 적용시킨 방법론 • 소프트웨어 개발 6단계를 정의하고, 단계별 활동을 정의 • (문제점) 비즈니스 프로세스, 서비스 규약(choreography)이나 서비스 저장소 같은 SOA에 대한 요구사항을 지원하지 않음
SOMA [6]	• 기존 IBM의 SOAD를 참조하여 확장된 방법론 • 서비스식별, 명세, 실현 세단계로 구성 • 하향식과 상향식 방식을 통합하여 정의 • (문제점) 서비스 식별에 대한 구체적인 활동이 제시되지 않음.

각 방법론에서 서비스를 식별하는 방법은 하향식과 상향식 그리고 이 둘을 혼합한 절충형 방식이 있는데 대체로 하향식 방법론에 대해서 언급하고 있고, 언급하더라도 구체적인 단계 및 단계별 수행 내용을 명시하고 있지 않다.

2. 컴포넌트-기반 개발 방법

CBD(Component-Based Development)에서 컴포넌트의 재사용을 통한 서비스로의 접근을 시도하는 연구가 있다. [7]은 RAS (Reusable Asset Specification)와 SOA에 의해 명세화 되는 컴포넌트의 다양한 측면을 이용하는 맞춤형 모델을 제시하고 있다. 그러나 [7]에서 제시한 서비스 모델은 컴포넌트의 재사용 가능한 단위나 크기에 초점을 둔 것이 아니라 컴포넌트와 관련된 문서나 명세서에 초점을 맞추고 있다. [8]은 헬스케어 도메인 모델을 이용하여 컴포넌트 모델을 domain, service, connector와 같은 세가지 부분으로 분리

를 시도하였다. 그러나 [7]과 [8] 모두, 하향식 접근이거나 개발 프로세스에만 초점을 두고 실질적인 서비스 식별에 대한 방법이 제시되어 있지 않다.

기존 객체지향에서 사용되는 결합도(coupling)는 공유 인스턴스 변수를 가지는 메소드 쌍의 수 또는 메소드에 의해 참조되는 인스턴스 변수로 정의하고, CBD에서 컴포넌트 식별에 사용되는 결합도는 클래스 간의 메시지 호출 수 및 메소드 호출 유형과 방향에 따른 의존성 특성을 고려한다 [9,10,11,12]. CBD에서의 응집도(cohesion) 연구는 대체적으로 데이터, 메시지 등 컴포넌트나 객체의 데이터 명세에 의해 결정되지만, 서비스는 비즈니스 요구 사항, 즉 사용자의 행위를 모델링하여 이를 기반으로 식별되어야 하므로 데이터 명세와는 다른 방법으로 결합도 및 응집도를 고려해야 한다.

따라서, 본 논문에서는 사용자의 비즈니스 프로세스를 GUI간의 이벤트 상호작용으로 판단하기 위한 상향식 및 알고리즘적인 접근을 제안하고, 이를 기반으로 응집도 메트릭을 제공함으로써 식별된 서비스 후보 집합의 크기(granularity)를 조정하여 원하는 크기의 서비스를 찾아 내는 것을 목표로 한다.

3. 순차패턴 마이닝 알고리즘

본래 순차 패턴 마이닝 알고리즘[13]은 데이터베이스에서 사용자-정의 최소 지지도를 만족하는 트랜잭션의 시퀀스들 가운데 최대 시퀀스를 찾는 알고리즘으로 O(n)의 시간 복잡도를 갖는다. 연관 규칙[14]과는 달리 트랜잭션의 발생 횟수만이 아닌 발생 순서(sequence)를 고려한다는 점에서 본 논문에서 찾고자 하는 이벤트의 순차를 고려 할 수 있다. 이 때 트랜잭션 시퀀스는 '우유', '기저귀', '맥주' 와 같이 고객이 구입한 물건에 시간 개념을 부과한 것이다. 따라서 방대한 고객정보 데이터베이스에서 각각의 고객이 구입한 물품들이 시간에 의해 시퀀스를 형성하고 이 정보들을 기반으로 순차 패턴 마이닝 알고리즘을 적용하여 고객들이 구매하는 물품과 그 순서 정보를 알아 낼 수 있다. 이러한 정보는 개인화(personalization)나 추천시스템(recommendation system)과 같이 개인의 구매성향을 파악하는데 이용되고 있다.

다음 예를 통하여 보다 구체적으로 순차 패턴 마이닝 알고리즘의 동작을 살펴 본다. 표 2는 5명의 고객의 시간에 따른 트랜잭션 시퀀스에서 최대 시퀀스(밑줄 친 항목)를 찾아내는 프로세스를 보여 주고 있다. 이때, 사용자 정의 지지도는 40%이상으로 설정해 놓은 것으로 적어도 2명 이상의 고객 시퀀스에 나타난 항목만이 남게 된다. 빈발항목-4까지 진행하면 최대 빈발항목으로서 〈1 2 3 4〉, 〈1 3 5〉 그리고 〈4 5〉가 생성된다. 이는 모집단인 고객별 트랜잭션 시퀀스에서 사용자 정의 40%이상의 지지도 이상을 만족하는 빈발 경로가 된다. 따라서 고객별 트랜잭션 시퀀스들 중에, 이 세가지 시퀀스가 가장 의미 있는 경로로 판단된다.

이와 같이 순차패턴마이닝 알고리즘을 적용하여 XML문서간 공유 구조를 추출하는 연구를 선행한 바 있다 [15]. 즉, 고객의 트랜잭션 시퀀스는 XML의 레벨별 엘리먼트의 경로 표현으로 보고 XML문서들간의 최대 유사 경로를 구할 수 있었다. 이를 본 논문에서는 고객의 트랜잭션 시퀀스 정보를 제시할 GUI의 이벤트 경로 표현과 의미적으로 연관 지을 수 있다. 즉, 사용자 정의 최소 지지도를 만족하는 항목이 살아남듯, 두 GUI간의 공유 경로만을 남긴다면 최종적으로 모든 GUI들 간의 이벤트 공유정도를 파악하고 이에 응집도 메트릭을 적용하여 어떤 GUI에 다른 GUI들이 얼마나 종속되어 있는지, 아니면 관계가 없는지 등을 수치화 할 수 있다. 본 논문에서는 사용자 정의 최소 지지도 개념이 40%와 같은 사용자 설정 값이 아니라 두 GUI 상호작용에 있어서 공유 이벤트가 존재한다는 조건으로 변형된다.

표 2. 순차패턴 마이닝 알고리즘
Table 2. Sequential Pattern Mining Algorithms

고객별 트랜잭션 시퀀스	빈발-1 항목 및 지지도		빈발-2 항목 및 지지도		빈발-3 항목 및 지지도		빈발-4 항목 및 지지도	
			〈1 2〉	2				
			〈1 3〉	4				
〈{1 5} {2} {3} {4}〉	〈1〉	4	〈1 4〉	3	〈1 2 3〉	2		
〈{1} {3} {4} {3 5}〉	〈2〉	2	〈1 5〉	3	〈1 2 4〉	2		
〈{1} {2} {3} {4}〉	〈3〉	4	〈2 3〉	2	〈1 3 4〉	3	〈1 2 3 4〉	2
〈{1 3 5}〉	〈4〉	4	〈2 4〉	2	〈1 3 5〉	2		
〈{4 5}〉	〈5〉	4	〈3 4〉	3	〈2 3 4〉	2		
			〈3 5〉	2				
			〈4 5〉	2				

III. 전처리

이 장에서는 순차패턴 마이닝 알고리즘의 핵심적인 자료로 사용될 이벤트 경로를 추출하기 위한 전처리(preprocessing) 방법을 제시한다.

1. GUI 분석

GUI들 간의 이벤트 호출을 추출하기 위해서는 GUI를 구성하는 요소들을 분석하고 이중 상호작용에만 관계되는 이벤트만을 고려해야만 한다. 대부분의 잘 정의된 컴포넌트 시스템의 GUI 화면은 그림 1과 같이 분류해 볼 수 있다.

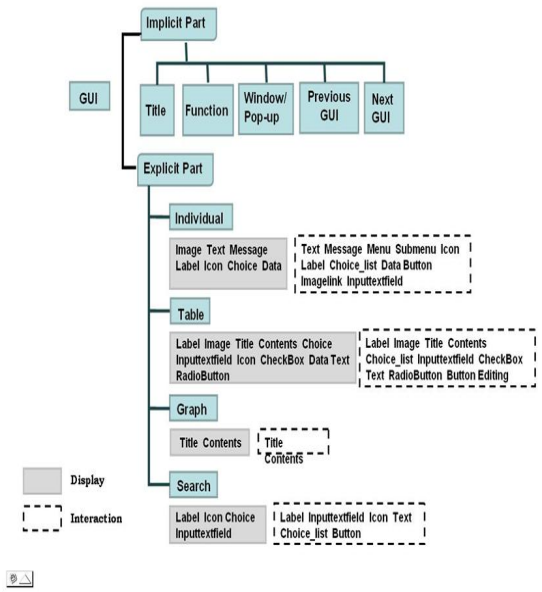


그림 1. GUI 구성요소 분류
Fig. 1. Classification of GUI Component

GUI 이벤트를 발생시키는 부분은 명시적인 부분(Explicit Part)으로 테이블, 그래프, 검색 그리고 그 외 부분으로 나누어지고 각각에 그림, 텍스트, 메시지, 아이콘, 라디오 버튼 등을 통해 사용자 입력에 의한 이벤트가 발생된다. 암시적인 부분(Implicit Part)는 GUI의 제목이나 기능, 윈도우창/팝업창 여부, 이전 화면과 다음 화면에 대한 전반적인 정보를 표시한다. 이벤트로 정의된 GUI 이벤트 요소들에 이름을 붙이고 상위 카테고리의 이름도 역시 접합하여 체계적인 시리얼 번호를 명명한다. 이벤트 시리얼 번호는 '이벤트를 발생하는 GUI화면번호(R#)-명시적 부분 또는 암시적 부분(Imp/Exp)-그에 따른 이벤트 발생 유형'을 나타낸다. 예를 들어 'R1-Exp-Tb-Int-Ic' 는 이벤트를 발생시키는 R1화면의 명시적 부분(Exp) 중 테이블 영역(Tb)에서 인터렉션을 일으키는(Int) 아이콘(Ic)에 의해 발생되는 이벤트임을 나타낸다.

2. GUI 이벤트 흐름도 구축

사용자의 GUI 이벤트 패턴을 분석하기 위해 GUI 이벤트 흐름도를 작성한다. GUI 이벤트 흐름도는 GUI 각 화면은 클래스로, 이벤트는 링크로 표시하여 GUI 화면 별로 발생되는 출력 이벤트와 입력 이벤트를 나타낸 것이고, 링크 정보는 GUI 이벤트의 시리얼 번호를 명시한다. 다음 그림 2는 UML의 클래스 다이어그램을 이용하여 GUI 이벤트 흐름도를 작성한 것이다.

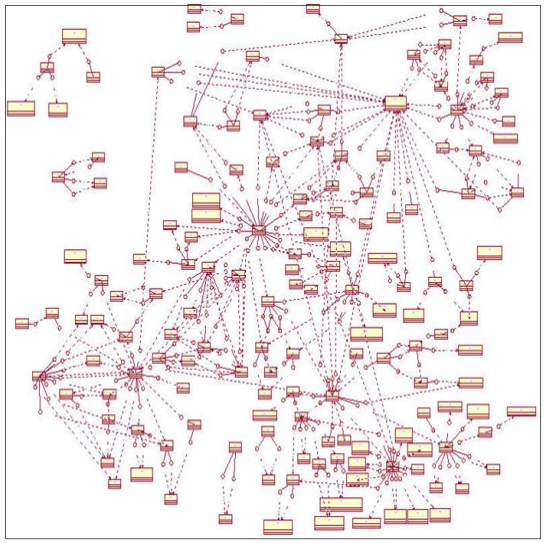


그림 2. GUI 이벤트 흐름도
Fig. 2.GUI Event Flow

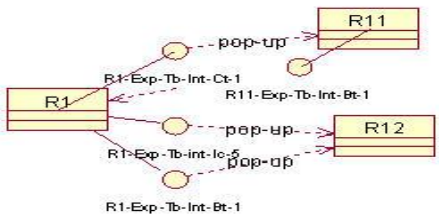


그림 3. GUI 이벤트 흐름도 일부
Fig. 3. A Part of a GUI Event Flow

GUI 이벤트 흐름도를 통해서 가장 많은 이벤트를 발생시키는 GUI화면에 대한 정보와 GUI화면 간 발생하는 이벤트 관계 정보를 시각적으로 알 수 있다. 앞의 그림 3은 R1, R11, R12라는 세 개의 GUI화면 간의 이벤트 흐름을 예시한다.

IV. 변형된 순차패턴 마이닝 알고리즘

이 전 장에서 순차패턴 마이닝 알고리즘의 전처리로 GUI의 이벤트 흐름도가 구축되었다. 이 장에서는 GUI간의 관련도를 평가하기 위해서 순차패턴 마이닝 알고리즘의 5단계가 어떻게 변형되어야 하는지 설명하고 '핵심 GUI 선정', '변형 및 서비스 후보 식별', 그리고 '공유 이벤트 경로 발견 및 최대 공유 이벤트 경로 추출'을 위한 알고리즘을 제안한다.

1. 변형 개념

본래 순차 패턴 마이닝 알고리즘은 정렬(sort), 빈발 항목 발견(litemset), 변형(transformation), 빈발 시퀀스 구하기(sequence), 그리고 최대 빈발 시퀀스 구하기(maximal) 단계로 총 5단계로 구성된다. 본래의 순차 패턴 마이닝 알고리즘은 몇 가지의 시퀀스를 방대한 데이터베이스로부터 찾는데 비해 변형 알고리즘은 구축된 GUI 이벤트 흐름도를 구성하는 이벤트 경로를 대상으로 한다. 이렇게 변형 순차 패턴 마이닝 알고리즘에서 본래의 알고리즘으로부터 변형된 개념 및 동작을 요약하면 다음 표 3과 같다.

- **1단계 (정렬-GUI 이벤트 흐름도 구축)** : 앞서 전처리 과정에서 구축한 GUI 이벤트 흐름도를 대상으로 개별 GUI로부터 시작하여 이벤트의 연결이 단절될 때까지를 연결하여 이벤트 경로를 얻는다. 경로 표현을 구성하는 화면 번호는 자연스럽게 호출 순서로 정렬된다. 즉, 본래의 알고리즘에서는 고객의 ID와 고객이 발생시킨 트랜잭션의 시간에 의해 데이터베이스가 정렬되지만, 변형 알고리즘에서는 흐름도를 구성하는 경로는 GUI의 호출 순서에 의해 정렬된다.
- **2단계 (빈발 항목 찾기-핵심 GUI 선정)** : 흐름도를 구성하는 모든 경로들 중, 시작 GUI, 즉 루트 GUI를 선정함으로써 다수의 GUI와 경로를 그룹핑 한다. 그림 2에서 보듯이 하나의 GUI에 링크가 집중되어 있는 것은, 분명 주요 GUI라는 증거이다. 따라서 본 논문에서는 하나의 GUI로부터 시작되는 이벤트의 수를 파악하여 일정 임계치가 넘는 이벤트를 가지는 GUI를 핵심 GUI로 선정한다.
- **3단계 (변형-변형 및 서비스 후보 식별)** : 대부분의 시스템들이 빠른 계산을 위해 데이터를 정수로 변환하듯이 본래의 알고리즘은 고객의 트랜잭션 시퀀스를 정수로 변형한다. 변형 알고리즘 역시 추출된 경로 표현이 정수로 대치되는데 이 때, 앞 단계에서 핵심 GUI를 선정하는 데에 사용된 이벤트 정보는 숨겨진 GUI번호로만 이루어진 경로로 변형된다. 또한 변형후, 핵심 GUI를 중심으로 경로

표 3. 순차패턴 마이닝 알고리즘의 변형된 개념
Table 3. Adapted Concept of Sequential Pattern Mining Algorithms

본래 순차 패턴 알고리즘		변형된 순차 패턴 알고리즘	
정렬 (Sort)	데이터베이스는 고객 ID와 트랜잭션 시간으로 정렬된다. 따라서 본래의 트랜잭션 데이터베이스는 객-시퀀스로 구성된 데이터베이스로 변경된다.	GUI 이벤트 흐름도 구축(Sort)	GUI를 노드로, 이벤트 (calling and called)를 링크로 정의하여 이벤트 흐름도를 구축한다. 따라서 본래의 GUI 정보들은 GUI-이벤트로 구성된 흐름도로 변경된다.
빈발항목 찾기 (Litemset)	최소지지도를 만족하는 빈발항목을 찾는다.	핵심 GUI 선정 (Litemset)	각 GUI의 호출 이벤트수만을 고려하여 일정 임계치를 만족하는 핵심 GUI를 선정한다. 빈발항목은 본래의 알고리즘과는 달리, 변형후에 찾는다.
변형 (Transformation)	고객 시퀀스는 빠른 검색을 위한 형태로 변경된다.	변형 및 서비스 후보 식별 (Transformation & Identification of Service Candidates)	모든 추출된 경로들이 정수로 교체되는데 이때 링크 정보를 제거하고 GUI 번호만으로 재명명된다. 또한 변형후, 핵심 GUI를 중심으로 클러스터(서비스 후보)를 생성하고 각 GUI 클러스터간에 공유하는 GUI들이 빈발 항목이 된다.
시퀀스화 (Sequence)	빈발항목 1, 2, …, n 시퀀스를 최소지지도에 입각하여 찾는다.	공유 이벤트 경로 발견 (Sequence)	길이 1, 2, …, n의 빈발 경로를 공유 여부에 따라 찾는다.
최대 빈발 시퀀스 추출 (Maximal)	발견된 빈발 시퀀스 중 최대 빈발 시퀀스를 발견한다.	최대 공유 이벤트 경로 추출 (Maximal)	공유 경로로 판단된 경로 중 최대 공유 경로를 추출한다.

상에 있는 GUI들을그룹핑하여 핵심 GUI수만큼의 클러스터(서비스 후보)를 생성한다. 이 때, 각각의 GUI 클러스터간에 공유하는 GUI들이 길이 1의 빈발 항목(large 1-path)으로 선택된다. 본래의 알고리즘에서는 사용자가 정의한 최소 지지도(minimum support)를 만족시키는 고객의 트랜잭션들이 빈발 항목이 된다. 그러나 변형 알고리즘에서는 다수의 핵심 GUI 그룹을 형성하고 각 그룹내의 이벤트 경로들 사이에 공유하는 GUI가 있다면 이 GUI가 빈발 항목으로 선택된다.

- **4단계 (시퀀스화-공유 이벤트 경로 발견) :** 본래의 알고리즘에서 시퀀스화 단계가 핵심이다. 길이를 증가시키면서 최소 지지도에 못 미치는 빈발 항목들을 계속 해서 추려나가게 된다. 변형 알고리즘에서도 길이 1의 핵심 GUI 정보를 기초로 확장하여 다시 길이 2의 빈발 후보 경로(large candidate 2-path)를 만든다. 길이 2의 빈발 후보 경로는 두 GUI 그룹에 존재하지 않으면 제거하여 길이 2의 빈발 경로(large 2-path)로 채택된다. 여기에서 다시 길이 3의 후보 항목(large candidate 3-path)을 만들고 두 그룹에 존재하지 않는 경로는 제거한다. 이와 같은 과정을 반복하여 더 이상 빈발 후보 항목이 발생하지 않을 때까지 수행한다.
- **5단계 (최대 빈발 시퀀스 추출-최대 공유 이벤트 경로 추출) :** 길이 1에서 n까지의 공유 경로들 중 포함관계가 있는 중복 경로를 제거하면 유일한 두 그룹간의 최대 공유 이벤트 경로가 남게 된다. 이 단계만이 유일하게 본래의 알고리즘과 차이가 없다. 단지 입력이 되는 빈발 항목이 트랜잭션의 시퀀스인가 공유 경로인가에만 차이가 있다.

2. 알고리즘

변형 순차패턴 마이닝 알고리즘의 첫 단계인 'GUI 이벤트 흐름도 구축 단계'는 이미 3.2절 전처리 단계에서 수행하였다. 따라서 알고리즘은 2단계인 '핵심 GUI 선정', 3단계인 '변형 및 서 비스 식별', 그리고 4단계와 5단계를 합쳐서 '공유 이벤트 경로 발견 및 최대 공유 이벤트 경로 추출'을 위한 세 가지 알고리즘을 제시한다.

2.1. 2 단계 : 핵심 GUI 선정

GUI 이벤트 흐름도는 'R#'와 같은 GUI 번호와 링크정보에 의해 각기 구별되는 이벤트들을 가지고 있다. 특별히, 하나의 GUI로부터 출력하는 이벤트의 수가 일정 임계치를 넘으면 그 GUI를 핵심 GUI (core GUI)로 선정한다.

다음 그림 4는 핵심 GUI를 선정하는 알고리즘이다.

먼저 n 개의 GUI인 Rn에 대하여 다른 GUI를 호출하는 이벤트를 'calling event'로, 다른 GUI에 의해서 호출되는 이벤트를 'called event'라고 했을 때, 'calling event'와 'called event'가 둘 다 없으면 고려대상 GUI에서 제거하고 'called event'만 있으면 TG(Terminal GUI) 스택에 저장한다(①). 또한 두 이벤트가 다 존재할 때, RG스택에 저장한다. 저장되는 그 R_i의 'calling event' 인 T_i의 수를 세어 CE_i에(Event Counter of R_i) 기억한다(②). 여기에서 R_i, RG_i, T_i^j, 그리고 CEi에의 i는 모두 같은 GUI 번호를 의미한다. 다음으로 핵심 GUI를 선정하기 위한 임계치를 'delta'로 각 R에 대한 CE의 최대값과 최소값의 합에 일정 비율(α)을 곱하여 정의한다(③). 이때 α는 핵심 GUI의 개수를, 즉 GUI 그룹의 수를 조정하는 비율이 된다. 물론 'delta'를 평균값으로 고정시켜 사용할 수도 있다. 마지막으로 RG 스택을 대상으로 delta 이상의 'calling event'를 가지는 GUI만을 Core 스택에 저장한다(④).

2.2. 3 단계 : 변형 및 서비스 후보 식별

전처리에서 준비된 GUI 이벤트 흐름도를 핵심 GUI를 루트로 하여 단말 GUI에 이르기까지 깊이-우선 탐색(Depth-First Search) 방법으로 탐색하면서 GUI 번호와 링크 이름을 접합함으로써 이벤트 경로를 만든다. 앞 단계에서 각 GUI의 calling과 called 이벤트의 수를 세는 데에 링크 정보를 이용하였다. 이제 계산의 간편화와 GUI에 집중하기 위해서 이벤트 경로를 변형한다.

예를 들어

'R1.Exp-Tb-Int-Ct-1.R11.Exp-Tb-Int-Ct-5.R23.⊥'(여기에서 ⊥은 R23이 단말 GUI임을 의미)는 '1.11.23.⊥'으로 변형된다.

또 다른 경로

'R1.Exp-Gr-Int-Tt-1.R11.Exp-Tb-Int-Ct-5.R23.⊥' 역시 '1.11.23.⊥'로 변형된다. 전자의 경로와 후자의 경로가 같은 1, 11, 23번의 GUI를 가지지만 서로 다른 이벤트 상호작용으로 연결되었으며 이 단계에서는 GUI 클러스터를 생성하고 각 클러스터내에서 공유되는 GUI 정보를 파악하는 데에 집중하기 위해 링크정보를 무시하는 것이다. 따라서 변형된 이벤트 경로인 '1.11.23.⊥'은 {1, 11, 23}번 GUI들이 하나의 클러스터를

```
procedure SelectingCore (R1..n: GUIs, TR1..m : calling events of R)
returns Core: set of core GUIs
begin
  for (i=1; i <= n; i++) do
  begin
    if Ri has no calling then
      if Ri has no called event then
①       discard Ri;                          // Ri is a dangling GUI
      else push(i, TG);                      // TG is a set of terminal GUI
    else
      { push (i, RG);                        // save the candidate of core GUI at RG stack
②     for (j=1; j <= m; j++)  do
        if Tj is an calling event of Ri then
          increment CEi;}                    //count the number of calling events of Ri
  end                                        // as the ratio of the number of output event
③ delta = (max(CE1..CEn) + min(CE1..CEn)) x α;  // α is user-defined ratio
  while (!is_empty(RG)) do
    begin
      k := popup(RG);          // R#
④     if CEk > delta then
        push(k, Core);     // selecting GUI over the threshold as Core
    end
end
```

그림 4. 핵심 GUI 집합 선정 알고리즘
Fig. 4. Algorithm for Selecting a Set of Core GUIs

이루고 이 클러스터는 서비스 후보가 될 수 있다. 다음 그림 5는 '변형 및 서비스 후보 식별'을 위한 알고리즘이다. 알고리즘은 크게 세 개의 하부 단계로 구성된다. 먼저 앞 단계에서 선정된 핵심 GUI를 중심으로 클러스터를 생성하고 각 클러스터별 이벤트 경로($Path_i$)를 생성한다(그림 5의 ①). 이때, 이벤트 경로에서 링크정보는 제거하고 순수한 GUI번호만을 남긴다(③).

```
procedure FindCommonGUI (Core: core GUIs, R1..n: GUIs, TR1..m : events)
returns L1Bi·Qk : all large 1-paths between GUIs, Path1..k: all event path expressions
begin
  for (i=1; i <= length(Core); i++) do
  begin
    MakeCluster (Corei, i, R1..n, TR1..m);
①   push(event paths generated from Clusteri, Pathi); // event paths generated from the Corei
  end
  for (i=1; i <= length(Core); i++) do
  begin
    PEBi := Pathi;                          // event paths according to each cluster i
    for (k=1; k <= length(Core) ; k++) do   // find alll arge1-paths L1Bi·Qk
    begin
      if (i ≠ k) then
      { PEQk := Pathk;                      // event path expressions of another cluster to be compared
②       foreach R in PEQk do                // R is a GUI
          if (R ∈ PEBi) then                // minimum support 100%
            insert R into L1Bi·Qk; }        // 1-large itemset between PEBi and PEQk (commonGUI)
    end
  end
end

procedure MakeCluster (InitCore, Cindex, R1..n, TR1..m )
returns ClusterCindex;
begin
  InitGUI := InitCore;
③ for (j=1; j <= m; j++) do
    if exists(initGUI.TinitGUIm.Rk) and (Rk ∉ initGUI) then  // exists an event from ith GUI to kth GUI
      push (Rk, ClusterCindex);                              // clustering kth GUI with ith core GUI
end
```

그림 5. 변형 및 서비스 후보 식별을 위한 알고리즘
Fig. 5. Algorithm for Transformation and Identification of Service Candidates

다음 표 4는 한 예로서 만약 어느 시스템의 핵심 GUI가 5개로 선정되었다면, 5개의 클러스터가 형성되고 클러스터가 포함하는 GUI 번호를 적은 것이다.

표 4. 예: 5개의 클러스터의 구성 GUI 번호
Table. 4. Example: 5 Clusters and their GUIs

	GUI 화면
Core1	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 27 28 29 30 31 32 33 34 35 36 43 46 47 48 50 51 52 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
Core2	16 17 24 25 35 36 46 47 48 50 51 52 63 64 66 68 74
Core3	15 16 17 24 25 35 36 46 47 48 50 51 52 53 63 64 66 67 76
Core4	9 10 16 24 25 26 27 28 35 36 37 38 40 41 45 46 47 48 49 50 51 52 53 54 55 63 64 66 67 74 75
Core5	16 24 25 26 27 28 35 36 37 38 40 42 43 44 45 46 47 48 49 50 51 52 63 64 66 67 75

첫 번째 클러스터는 1부터 74까지 62개의 GUI가 내포되어 있는 것을 알 수 있다. 다음으로, 클러스터 갯수만큼 반복하여 각 클러스터내의 구성 화면의 이벤트 경로들이 i번째 기준 경로(PE_{Bi}) 집합이 되고 다시 $i \neq k$ 인 임의의 k번째 클러스터의 이벤트 경로(PE_{Qk})와 비교하여 PE_{Qk}를 구성하는 임의의 GUI가 기준 경로 PE_{Bi} 집합에 포함되는 GUI라면 공유 이벤트 경로 $L_1^{Bi \cdot Qk}$;에 삽입한다(그림 5의 ②). 이때 $L_1^{Bi \cdot Qk}$ 는 길이 1의 빈발 항목으로서 기준 클러스터 i와 비교 클러스터 k 간의 공유 GUI 번호를 의미한다. 다음 그림 6은 첫 번째 클러스터의 PE_{B1}과 두 번째 클러스터 PE_{Q2}와의 공유 GUI 집합인, $L_1^{B1 \cdot Q2}$를 구하는 과정을 보여준다. 상단의 1번부터 5번은 클러스터의 번호이며 가운데 번호를 둘러싼 큰 상자(음영진 상자)가 클러스터 1과 2(C1, C2)의 공유 GUI들이다.

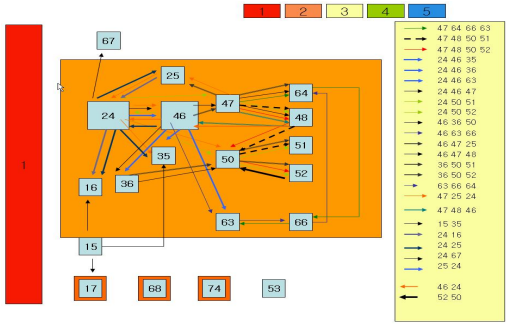


그림 6. 예: 두 클러스터의 공유 GUI
Fig. 6. Example: Common GUIs between Two Clusters

이러한 과정으로 (C1, C3), (C1, C4), (C1, C5), (C2, C1), (C2, C3)… 등과 같은 반복으로 각 클러스터간의 공유 경로를 구하게 된다. 즉, n개의 클러스터가 구해졌다면, n(n-1)개의 공유 GUI 집합이 만들어질 것이다. n(n-1)/2가 아닌 이유는 기준 클러스터가 있기 때문이다.

2.3. 4와 5 단계: 공유 이벤트 경로 발견 및 최대 경로 추출

본래의 알고리즘은 최소지지도를 만족하는 항목을 '빈발항목(large itemset)'이라고 표현한다. 또한 항목이 하나인 경우 1-빈발항목(L_1)으로, 항목-항목으로 시퀀스를 이루기 시작하면 연결된 항목의 개수에 따라 2-빈발항목(L_2), 3-빈발 항목(L_3)으로 명명한다. 본래의 알고리즘은 1-빈발항목을 기반으로 길이를 늘려 나가면서 최대 빈발 경로를 찾는데 O(n) 시간 복잡도를 보장한다. 본 논문에서도 아무리 GUI를 많이 갖는 시스템이라도 O(n) 시간 안에 최대 공유 이벤트 경로를 찾아낼 수 있도록 적용한다. 다음 그림 7은 공유 이벤트 경로($L_1,\ldots,L_n$)와 최대 공유 경로를 구하는 알고리즘이다. (여기에서 for 반복문이 두 개 존재하여 O(n)을 보장하지 않는 것처럼 보이나, 기존의 알고리즘과는 달리 n(n-1)번의 클러스터 비교 반복문이 추가되기 때문에, 실질적인 빈발항목 발견 부분은 O(n)으로 계산된다.)

먼저, 모든 L_1 을 초기 대상으로 하여, 길이를 늘여 나가면서 새로운 후보 경로 C_2를 생성한다(①). 예를 들면 다음 표 5는 L_1 경로를 토대로 길이를 확장해 나가면서 L_2를 구하는 과정을 보여 주고 있다.

표 5. 길이-2의 공유 이벤트 경로 찾기
Table 5. Finding 2-Shared Event Paths

L_1	C_2	L_2
⟨2⟩ ⟨3⟩ ⟨12⟩ ⟨13⟩ ⟨14⟩ ⟨16⟩ ⟨17⟩ ⟨18⟩ ⟨19⟩ ⟨21⟩ ⟨22⟩ ⟨23⟩	⟨2,3⟩,⟨2,12⟩,⟨2,13⟩,⟨2,14⟩, ⟨2,16⟩,⟨2,17⟩,⟨2,18⟩,⟨2,19⟩, ⟨2,21⟩,⟨2,22⟩,⟨2,23⟩,⟨3,2⟩, ⟨3,12⟩,⟨3,13⟩,⟨3,14⟩,⟨3,16⟩, ⟨3,17⟩,⟨3,18⟩,⟨3,19⟩,⟨3,21⟩, ⟨3,22⟩,⟨3,23⟩,⟨12,2⟩,⟨12,3⟩, **⟨12,13⟩,⟨12,14⟩**,⟨12,16⟩,⟨12,17⟩, ⟨12,18⟩,⟨12,19⟩,⟨12,21⟩,⟨12,22⟩, ⟨12,23⟩,⟨13,2⟩,⟨13,3⟩,⟨13,12⟩, ⟨13,14⟩,⟨13,16⟩,⟨13,17⟩,⟨13,18⟩, ⟨13,19⟩,⟨13,21⟩,⟨13,22⟩,⟨13,23⟩, ⟨14,2⟩, ⟨14,3⟩, ⟨14,12⟩,⟨14,13⟩, ⟨14,16⟩,⟨14,17⟩,⟨14,18⟩,⟨14,19⟩, ⟨14,21⟩,⟨14,22⟩,⟨14,23⟩,⟨16,2⟩, ⟨16,3⟩,⟨16,12⟩,⟨16,13⟩,⟨16,14⟩, **⟨16,17⟩,⟨16,18⟩**,⟨16,19⟩,⟨16,21⟩, ⟨16,22⟩,⟨16,23⟩,⟨17,2⟩,⟨17,3⟩, ⟨17,12⟩,⟨17,13⟩,⟨17,14⟩,⟨17,16⟩, **⟨17,18⟩**,⟨17,19⟩,⟨17,21⟩,⟨17,22⟩, ⟨17,23⟩,⟨18,2⟩,⟨18,3⟩,⟨18,12⟩, ⟨18,13⟩,⟨18,14⟩,⟨18,16⟩, ⟨18,17⟩,⟨18,19⟩,⟨18,21⟩, ⟨18,22⟩,⟨18,23⟩,⟨19,2⟩,⟨19,3⟩, ⟨19,12⟩,⟨19,13⟩,⟨19,14⟩, ⟨19,16⟩,⟨19,17⟩,⟨19,18⟩, **⟨19,21⟩**,⟨19,22⟩,⟨19,23⟩,⟨21,2⟩, ⟨21,3⟩,⟨21,12⟩,⟨21,13⟩,⟨21,14⟩, ⟨21,16⟩,⟨21,17⟩,⟨21,18⟩, ⟨21,19⟩,⟨21,22⟩,⟨21,23⟩,⟨22,2⟩, ⟨22,3⟩,⟨22,12⟩,⟨22,13⟩,⟨22,14⟩, ⟨22,16⟩,⟨22,17⟩,⟨22,18⟩, ⟨22,19⟩,⟨22,21⟩,**⟨22,23⟩**,⟨23,2⟩, ⟨23,3⟩,⟨23,12⟩,⟨23,13⟩,⟨23,14⟩, ⟨23,16⟩,⟨23,17⟩,⟨23,18⟩, ⟨23,19⟩,⟨23,21⟩,⟨23,22⟩	⟨12,13⟩ ⟨12,14⟩ ⟨16,17⟩ ⟨16,18⟩ ⟨17,18⟩ ⟨19,21⟩ ⟨22,23⟩

```
procedure FindMaximalPaths (L1^B·Q : all large 1-path expressions between clusters,
                            Path1..k:eventpathexpressionsaccordingtoeachcluster)
returns ML^B·Q ;  // all maximal frequent paths between PE_B and PE_Q
begin
  for (i=1; i <= the number of clusters; i++) do
  begin
    PE_Bi := Path_i;
    for (j=1; j <= the number of clusters; j++) do
    begin
      if (i ≠ j) then
      { PE_Qj := Path[j];
        for (k=2; L_k-1^Bi·Qj ≠ ∅; k++) do
        begin          // same as apriori-generate function of sequential patterns [12]
          C_k = New candidate-paths generated from L_k-1^Bi·Qj;
          foreach event path expressions in PE_Bi and PE_Qj do //pruning
  ①         if (C_k ∈ PE_Bi and C_k ∈ PE_Qj) then          //exists on the both
              L_k^Bi·Qj = C_k;
          end
          length = k;          // longest length of large paths L_k^Bi·Qj
        end
        for (m = length; m >= 1; m--) do    // maximal phase
          foreach m-large paths, L_m^Bi·Qj do
  ②         delete all sub-paths of L_m^Bi·Qj;
          end
        ML^Bi·Qj = { remained large paths in L^Bi·Qj }
      }
    end
  end
end
```

그림 7. 공유 이벤트 경로 및 최대 공유 경로 발견을 위한 알고리즘
Fig. 7. Algorithm for Finding Common GUIs and Maximal Shared Event Paths

먼저 빈발 경로 L_1을 토대로 길이 하나를 확장하여 순서쌍을 구하면 C_2를 구할 수 있다. 물론 확장은 L_1에 있는 항목만을 대상으로 한다. 여기에서 다시 두 클러스터에 존재하는 경로(진하게 표시된 항목)이면 남기고 그렇지 않으면 제거(pruning)한다. 남은 경로는 L_2가 된다.

다음으로, 표 6은 L_3을 찾는 과정을 보여준다. L_3을 찾는 과정에서 확실히 검색 및 비교 시간을 줄일 수 있다는 것을 알 수 있다. AprioriAll 알고리즘을 통하여 길이 2의 공유 경로부터는 L_2의 마지막 GUI와 첫번째 GUI가 동일한 경로만을 가지고 확장한다. 따라서 표 6에서 L_2를 C_3로 확장되는 엘리먼트는 〈16, 17〉 그리고 〈17, 18〉로 유일하며 이를 확장하면 〈16, 17, 18〉 이 된다. 그리고 이 경로가 두 클러스터에 존재한다면 L_3가 된다. 이제 L_3 경로가 하나밖에 없어 이제 후보 경로를 만들 수 없으므로 공유경로를 찾는 루프는 중단되고 다음 최대 공유 경로를 찾는 단계로 넘어간다.

그림 7의 ②와 같이 모든 공유 경로를 탐색하며 부분 경로가 되는 공유 경로를 제거함으로써 최대 공유 경로를 구할 수 있다. 본 논문에서 '최대(maximal)'의 의미는 다른 경로에 포함되지 않음을 의미한다. 예를 들어 paths = {1.2.3.4, 1.2, 2.4, 1.3.4} 가 있다면 최대 경로는 {1.2.3.4}가 된다. 다음 표 7은 구해진 모든 공유 경로, $L_{1..3}$에서 최대 공유 경로만을 밑줄로 표시하였다.

표 6. 길이-3의 공유 이벤트 경로 찾기
Table 6. Finding 3-Shared Event Paths

L_2	C_3	L_3
〈12,13〉,〈12,14〉, 〈16,17〉,〈16,18〉, 〈17,18〉,〈19,21〉, 〈22,23〉	〈16, 17, 18 〉	〈16, 17, 18 〉

표 7. 최대 공유 이벤트 경로 찾기
Table 7. Finding Maximal Shared Event Paths

L_1	L_2	L_3
〈2〉,〈3〉,〈12〉,〈13〉,〈14〉, 〈16〉,〈17〉, 〈18〉,〈19〉,〈21〉, 〈22〉,〈23〉	**〈12, 13〉** **〈12, 14〉** 〈16, 17〉 〈16, 18〉 〈17, 18〉 **〈19, 21〉** **〈22, 23〉**	**〈16, 17, 18〉**

3.서비스 식별

이제 두 핵심 클러스터(서비스 후보)들 간의 최대 공유 경로가 준비되었다. 따라서 서비스 후보에서 서비스를 식별하기 위해서는 수치화가 필요하고 본 논문에서는 이를 위해 다음과 응집도 메트릭을 적용한다.

$$Coh(Cp,Cq)=\frac{1}{T}\sum_{i=1}^{T}\left(\frac{1}{2\times L(EP_i)-1}\sum_{k=1}^{L(EP_k)}V(E_k)\right) \quad \text{식(1)}$$

식(1)에서 T는 기준 클러스터의 총 경로의 수이고 EP_i는 i번째 클러스터의 이벤트 경로 집합, $L(EP_i)$는 그 집합의 총 경로의 수, $V(E_k)$는 특정 경로의 어느 GUI와 기준 경로의 GUI가 서로 동일한 GUI가 있으면 1, 매치되는 GUI가 기준 경로에 없으면 0, 만약 경로자체가 동일하면 2라는 가중치를 주는 함수이다. 이렇게 두 클러스터간의 응집도를 계산하여 일정 임계치를 넘으면 하나의 서비스로 식별될 수 있다. 임계치를 얼마로 설정하느냐에 따라 서비스의 크기를 조정할 수 있다.

V. 적용사례 및 분석

1. 적용

변형된 순차패턴 마이닝 알고리즘을 국내 기업에서 실제 사용되고 있는 MIS(Management and Information System)에 적용하여 서비스 식별을 시도해 보았다. 이 엔터프라이즈 수준의 시스템은 컴포넌트 개발 방법론에 의해 잘 정의된 13개의 상위의 컴포넌트들로 구성 되어 있으며, 컴포넌트는 129개의 GUI화면을 통해 상호작용 한다. 13개의 컴포넌트는 기능적으로 독립이 가능한 단위로 설계 되어 있으나, 정확히 129개의 컴포넌트가 배타적으로 GUI에 할당 되는 것이 아니라 매우 복잡하게 GUI와 연결되어 있었다. 즉, 컴포넌트의 독립성은 보장하도록 개발되었으나, 하나의 화면이 여러 개의 컴포넌트와 상호 작용하고 있는 엔터프라이즈 애플리케이션이다. 적용 과정은 다음과 같다.

- 핵심 GUI 선정 알고리즘에 의해 129개의 GUI들 중 호출 이벤트 수가 최대 19개, 최소 0개의 화면이 있었고, a=50%으로 볼 때, 10개 이상의 호출 이벤트를 가지는 5개의 핵심 GUI가 선정되었다.
- 변형 및 서비스 후보 식별 알고리즘에 의해 핵심 GUI를 중심으로 연결된 모든 GUI를 묶어 클러스터를 형성하였다. 앞의 표 4가 그 결과이며 이 5개의 클러스터가 서비스 후보 단위들이 된다.
- 마지막으로 공유 이벤트 경로 발견 및 최대 공유 경로 추출 알고리즘을 적용하여 각 클러스터를 구성하고 있는 GUI들이 가지는 이벤트 경로 집합을 대상으로 공유 경로를 발견하고 최대 공유 경로들만 남긴다. 다음 표 8은 클러스터 1에 대한 나머지 클러스터들과의 최대 공유 이벤트 경로를 나타낸 것이다.

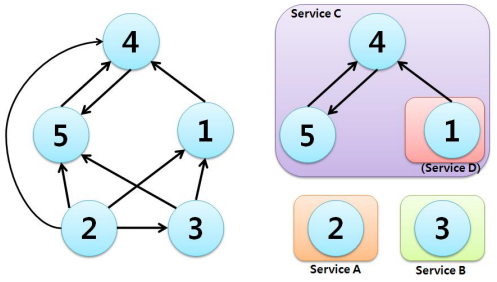
모든 클러스터들간의 최대 공유 이벤트 경로를 구하고 4.3절에서 제시된 응집도 메트릭을 적용하면 다음 표 9와 같은 결과를 얻는다. 세로의 클러스터(Core)가 기준을 의미하고 가로는 비교 클러스터를 의미한다. 예를 들면 기준 클러스터 4에 대해 비교 클러스터 3은 77.1%의 응집도를 보인다. 이 결과를 토대로 각각의 서비스 후보(클러스터)에 만약 80%와 90%이상의 응집도를 갖는 서비스 후보들을 묶어 서비스로 식별하였을 때, 다음 그림 8과 같은 결과를 얻을 수 있다. 각 노드는 클러스터 번호를, 에지 성분은 응집도 수치에 의해 임계치가 넘으면 그린다. 화살표 방향은 포함관계가 된다. 즉 '2 → 5'의 의미는 표 9에서 보는 바와 같이 2번 클러스터가 90.2%로 상당부분 5번 클러스터에 포함되는 것으로 본다.

표 8. 1번 클러스터의 최대 공유 이벤트 경로
Table. 8. Maximal Shared Event Paths of 1st Cluster

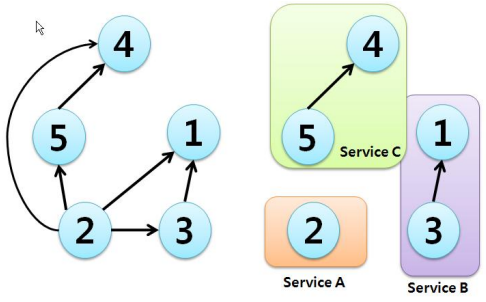
	Core2	Core3	Core4	Core5
Core1	68, 17, 74, ⟨52,50⟩, ⊤ (24,46,35) (24,46,26) (24,46,63) (25,24,16) (46,24,16) (24,46,25) (46,24,35) (47,48,46) (48,46,24) (24,46,47,64) (24,46,47,48) (46,24,50,51) (46,24,50,52) (46,63,66,64) (47,25,24,46) (47,25,24,35) (47,48,50,51) (47,48,50,52) (47,64,66,63)	53, (15,17) (15,16) (15,35) (24,16) (24,25) (24,67) (25,24) (46,24) (52,50) (24,46,35) (24,46,36) (24,46,63) (24,46,47) (24,50,51) (24,50,52) (46,47,48) (46,36,50) (46,63,66) (46,47,25) (46,47,64) (46,47,48) (36,50,51) (36,50,52) (63,66,64) (47,25,24) (47,48,46) (47,64,66,63) (47,48,50,51) (47,48,50,52)	(52,50) (28,27) (9,74) (9,35) (10,9) (27,75) (27,16) (46,47,48) (24,46,35) (24,46,63) (25,24,46) (47,48,46) (9,10,64) (27,28,75) (27,50,52) (27,46,24,35) (25,24,50,51) (25,24,50,52) (10,64,66,63) (27,46,24,25) (27,46,24,35) (27,46,24,67) (27,46,24,16) (27,46,36,50) (27,46,47,25) (24,46,36,50,51) (24,46,36,50,52) (27,46,63,66,64) (24,46,47,48,50) (27,46,47,48,50) (24,46,47,64,66) (27,46,47,25,24) (46,47,48,50,51) (46,47,48,50,52) (46,47,64,66,63) (46,47,25,24,35) (46,47,25,24,50) (46,47,25,24,67) (46,47,25,24,16)	43, (28,27) (27,35) (46,24,50) (46,24,67) (46,24,16) (24,46,35) (24,46,36) (24,46,63) (24,46,47) (25,24,46) (25,24,35) (25,24,67) (25,24,16) (64,66,63) (47,48,46) (27,28,75) (27,46,24,35) (25,24,50,51) (25,24,50,52) (27,46,24,25) (46,47,48,50,51) (46,47,48,50,52) (27,46,36,50,51) (27,46,36,50,52) (27,46,63,66,64) (27,46,63,66,64) (27,46,47,48,50) (27,46,47,64,66) (27,46,47,,25,24)

표 9. 최대 공유 이벤트 경로 찾기
Table 9. Finding Maximal Shared Event Paths

Coh	Core1	Core2	Core3	Core4	Core5
Core1		100%	97.9%	64.3%	62.7%
Core2	63%		77.1%	44.7%	49.3%
Core3	75%	90.2%		43.5%	52%
Core4	83%	92.7%	77.1%		98.7%
Core5	73.3%	90.2%	81.3%	87.1%	



(a) 임계치 = 80% 일 때 식별된 서비스



(b) 임계치 = 90% 일때 식별된 서비스

그림 8. 응집도 임계치 조정에 따른 서비스 식별
Fig. 8. Service Identification by Regulating the Threshold

먼저 그림 8(a)는 응집도 80% 이상의 클러스터들의 응집도를 도식화 한 것이다. 2번 클러스터는 모든 나머지 클러스터에 포함되는 것으로 보아 이를 하나의 서비스 단위로 식별한다면 재사용성(reusability) 및 합성능력(composability)이 매우 높은 서비스가 될 것이다. 따라서 하나의 서비스 A로 식별하고 3역시 두 클러스터에 합성되고 있으므로 서비스 B로 식별한다. 2와 3과 연관된 화살표들을 모두 제거하고 나면 서비스 C의 관계만 남게 되므로 이를 하나의 서비스로 식별하게 된다. 만약 1번 클러스터를 독립적인 서비스 D로 식별해도 좋으나, 단독으로 사용되었고 어디에 재사용, 혹은 합성된 사실이 없으므로 후보로 남긴다. 반면 그림 (b)는 90% 이상의 응집도를 보이는 클러스터의 관계를 도식화 하였다. (a)와는 달리 1—〉 4, 3—〉 5로 가는 화살표가 제거되어 훨씬 식별이 용이해 졌다.

임계치를 얼마로 보느냐에 따라 서비스 식별이 조금씩 달라지기는 했지만, 두 경우 모두 2를 하나의 서비스로, 그리고 4와 5를 하나의 서비스로 바라보고 있다. 여기에 1과 3은 GUI가 상호작용하고 있는 컴포넌트의 분리 및 병합 문제에 따라 별도로 서비스로 보거나 아니면 합성 서비스로 정의할 수도 있다. 만약 좀 더 많은 클러스터를 대상으로 하고 싶다면, '핵심 GUI 선정' 알고리즘에서 delta 값을 좀 낮추어 서비스 후보의 수를 늘릴 수 있다.

2. 분석 결과

본 논문에서 제안한 GUI를 기반으로 하는 비즈니스 서비스 식별 방법의 타당성을 보이기 위해 다음과 같은 실험을 병행하였다.

(정량적 비교) : 본 논문에서 제안한 GUI를 대상으로 하지 않고 컴포넌트 자체의 결합도에 의한 그룹핑을 시도하였다. 다음 그림 9는 A~S까지 총 13개의 컴포넌트를 대상으로 결합도를 계산한 결과 분포를 보인다. 5~50%까지 값이 집중되어 있으며 각 결합도의 분산값이 173정도로 본 논문에서 제안한 방법의 분산값이 880인데 비하여 1/5정도로 줄게 된다. 이는 각 컴포넌트간의 결합도가 평균 26%에 집중되어 있어 서비스가 한 덩어리로 식별되는 결과를 보였다. 즉, 그림 9의 그래프 결과에 따라 컴포넌트를 그대로 서비스화 하는 경우와 같이 평균에 밀접하게 결합도가 분포되어 있는 경우, 기준 결합도를 정하는 데 있어 다양한 값을 정할 수 없고 이것은 곧 어떠한 컴포넌트를 서로 컴포지션하여 새로운 서비스로 식별할 것인가에 대한 판단 기준이 모호함을 의미한다. 그러나 본 논문에서 제시한 방법으로 서비스를 식별할 경우(표 9 참조), 분산 880으로, 컴포넌트를 바로 서비스로 식별할 때보다 모호함이 없이 다양한 크기로 서비스 식별이 가능함을 알 수 있다.

(정성적 비교) 비즈니스 전문가의 시스템 파티셔닝 : 본 실험에서 적용한 동일한 MIS를 SOA 전문가이자 비즈니스 전문가가 직접 컴포넌트 시스템의 시스템 명세서를 보고 컴포넌트를 하향식으로 그룹핑한 결과 세 개의 서비스로 시스템을 파티셔닝 하였다. 컴포넌트 번호를 맞추어 본 결과 본 논문에서 제안한 결과와 거의 일치함을 보였다.

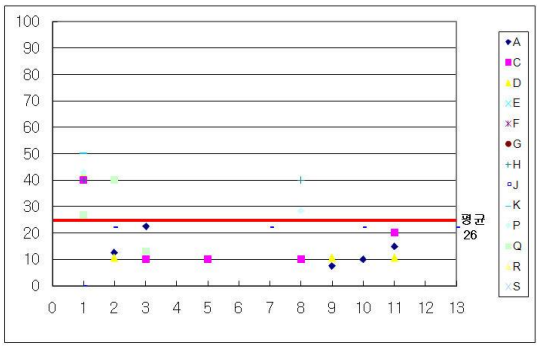


그림 9. 기존 컴포넌트간 결합도
Fig. 9. Coupling between Legacy Components

VI. 결론

본 논문에서는 GUI정보를 이용하여 비즈니스 프로세스에 근사한 이벤트 경로를 추출하고 이를 기반으로 변형된 순차패턴 마이닝 알고리즘을 설계하여 상향식 서비스 식별 방법을 제안하였다. 기존의 컴포넌트 그룹핑을 통해서 서비스 단위를 식별하는 것보다 5배로 분산값이 커지면서 서비스를 식별하기 용이해 졌으며 이는 비즈니스 및 SOA 전문가의 하향식 방법의 서비스 식별과 일치함을 보였다.

기존의 서비스 식별방법들이 대부분 프로세스에 머물러 구체적인 수치화로의 시도가 드물었다. 그러나 본 논문에서 제안한 알고리즘을 토대로 GUI와 컴포넌트의 수가 매우 큰 대형 시스템이라고 하더라도 빠른 시간안에 서비스 식별을 해 낼 수 있으며 파라미터 값을 조정하면서 식별된 서비스의 타당성을 검토할 수도 있다. 그러나 앞으로, 그 GUI들이 가지는 컴포넌트들을 어떠한 방식으로 분리하고 합병할 것인가는 앞으로 해결해야 할 문제이다.

참고문헌

〔1〕 V. Kapoor, "Services and Automatic Computing: A Practical Approach for Designing Manageability," In Proc. of the 2005 IEEE International Conference on Service Computing, Vol.2, pp.41-48, July 2005.

〔2〕 T. Erl, "*Service-Oriented Architecture: Concepts, Technology, and Design*", Prentice Hall, 2005.

〔3〕 *http://en.wikipedia.org/wiki/Business_Process*

〔4〕 Gregg Kreizman, "*How to Build a Business Case for Service-Oriented Development of Applications in Government,*" Gartner. Industry Research, Sept. 2005.

〔5〕 Kunal Mittal, "*Service Oriented Unified Process (SOUP),*" IBM Journal, 2005.6.

〔6〕 Ali Arsanjani, "*Service-Oriented Modeling and Architecture : How to identify, specify, and realize services for your SOA,*" IBM developer Works, Nov. 2004.

〔7〕 Soojin Park, SooyongPark, VijayanSugumaran, "Extending reusable asset specification to improve software reuse", in Prooceddings of SAC, pp. 1473-1478, 2007.

〔8〕 S. C. Chu, "From Component-based to Service Oriented Software Architecture for Healthcare," Enterprise Networking and Computing in Healthcare Industry 2005, HEALTHCOM, in Proc. of 7th International Workshop, pp. 96-100. 2005.

〔9〕 Chidamber S.R., Kemerer, C.F., "A metrics suite for object oriented design," IEEE Trans. Software Engineering, Vol. 20. pp. 476-498, 1994.

〔10〕 Hyung Ho Kim and Doo Whan Bae, "Component Identification via Concept Analysis", Journal of Object Oriented Programming, 2001.

〔11〕 임근, "객체모델을 이용한 컴포넌트 설계 및 검색 프로토타입," 한국컴퓨터정보학회 논문지, 제11권, 제6호, 27-33쪽, 2006년 12월.

〔12〕 임든, 이기영, "컴포넌트 모델구축을 위한 클래스 코드 자동생성 방법," 한국컴퓨터정보학회 논문지, 제13권, 제5호, 69-76쪽, 2008년 9월.

〔13〕 R. Srikant and R. Agrawal. "Mining Sequential Patterns," In Proc. of the Int'l Conf. on Data Engineering(ICDE), pp. 3-14, Mar. 1995.

〔14〕 R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association Rules", In Proc. of the 20th Int'l Conf. on Very Large Databases, pp. 487-499, 1994.

〔15〕 이정원,이기호, "XML 공유 구조 발견을 위한 변형 순차패턴 마이닝 알고리즘," 한국정보과학회 2002년도 가을 학술발표논문집, 제29권, 제2호(Ⅰ), 1-3쪽, 2002년 10월.

저 자 소 개

이 정 원

2003년 이화여자대학교 컴퓨터공학 (박사)

2003년~2006년 이화여자대학교 컴퓨터학과 BK교수, 전임강사(대우)

2006년~현재 : 아주대학교 전자공학부 조교수

관심분야 : SOA, 유비쿼터스 컴퓨팅, 임베디드 소프트웨어