

논문 2026-3-10 <http://dx.doi.org/10.29056/jsf.2026.09.10>

시뮬레이션 소프트웨어의 이벤트 로그 위장 복원과 파생 정황 분석

유예동*, 조병현**, 장수영**, 양승호***, 최창범**†

Obfuscation-Resilient Behavioral Similarity Analysis of Simulation Software Using Event Logs

Yea Dong Yoo*, Byeonghyeon Cho**, Sooyoung Jang**, Seungho Yang***, Changbeom Choi**†

요 약

시뮬레이션 소프트웨어 분쟁에서는 소스코드를 확보하지 못해 실행 이벤트 로그만으로 파생 정황을 분석해야 할 수 있다. 그러나 라벨 재명명과 동일 시각의 기록 분할은 라벨 의존 유사도 지표를 무력화한다. 본 논문은 라벨 비의존 특징 상태 시그니처와 헝가리안 할당에 의한 상태 정렬, 약한 이중모사 개념을 참고한 stutter 병합을 제안하고 정의된 변환에 대한 조건부 복원절차와 적용 조건을 제시한다. pyjevsim 대기행렬 모델의 기준 구현 1개와 비교 변형 6개를 평가한 결과, 재명명 변형의 원라벨 종합 유사도는 0.133으로 독립 구현의 0.132와 유사했으나 정렬 후에는 각각 1.000과 0.847로 구분되었다. 또한 재명명·분할·복합 변형의 산출 로그가 정의된 변환과 일치함을 오라클로 확인하였다. 결과는 제안 방법이 정의된 위장 조건에서 은폐된 실행 행동 유사성을 복원하고 파생 정황 분석을 지원할 수 있음을 보여준다.

Abstract

In disputes involving simulation software, source code may be unavailable and derivative indications must be examined from execution logs. Label renaming and same-time record splitting, however, can invalidate label-dependent similarity metrics. This paper combines label-free state signatures and Hungarian assignment with stutter merging inspired by weak bisimulation, and specifies a conditional restoration procedure and its applicability conditions. We evaluated one reference implementation and six comparison variants generated with pyjevsim. The raw-label similarity of the renamed variant was 0.133, nearly identical to 0.132 for the independent implementation; after alignment, the scores became 1.000 and 0.847, respectively. Oracle checks confirmed that the renamed, split, and combined variants matched the defined log transformations. These results show that the method can restore concealed behavioral similarity under the evaluated obfuscation conditions and support technical analysis of possible derivation.

한글키워드 : 소프트웨어 포렌식, 이벤트 로그, 소프트웨어 유사도, 상태 정렬, 약한 이중모사

keywords : Software forensics, Event log, Software similarity, State alignment, Weak bisimulation

* 국립한밭대학교 우주국방융합학과

** 국립한밭대학교 컴퓨터공학과

*** 국립한밭대학교 도시공학과

† 교신저자: 최창범(email: cbchoi@hanbat.ac.kr)

접수일자: 2026.08.19. 심사완료: 2026.08.31.

게재확정: 2026.09.20.

1. 서론

시뮬레이션 소프트웨어는 해당 분야의 시뮬레이션 수행하기 위한 모의논리가 담긴 시뮬레이션 모델 코드와 모델을 연결하고 시뮬레이션을 실행, 관리하고 사용자의 편의성을 주기 위한 코드로 구성되는 특징을 가지고 있다[1]. 소스코드 대조에 의존적인 일반적인 소프트웨어 표절 감정과는 달리 시뮬레이션 소프트웨어는 계산의 정확도를 요구하는 공학 방정식의 동일성 및 거동 특성을 결정짓는 시간적 특성을 반영하는 모의 논리의 특징으로 소스 코드가 제출된다고 하더라도 단순 비교를 통한 유사도 측정에 제한된다. 또한 시뮬레이션 소프트웨어 분쟁에서는 대상 소프트웨어가 실행 파일 형태로만 제출되거나 클라우드 서비스로만 접근 가능한 경우가 빈번하다. 이 경우 실행 중 기록되는 이벤트 로그를 주요 분석 자료로 활용할 수 있다. 이벤트 로그는 시뮬레이션 소프트웨어의 디버깅과 분석을 위하여 시뮬레이션이 실행될 때 출력 혹은 저장되는 자료로 감정인이 통제할 입력 시나리오를 재현할 수 있고, 소프트웨어 내부를 건드리지 않는 정보 수집이 가능하다. 이와 같은 가정에서 시뮬레이션 소프트웨어 감정은 무엇의 유사성인지를 구분하여 확인한다. 이름이 비슷한 경우, 구조가 비슷한 경우, 동일 동작을 수행하는 지, 같은 궤적을 보유하고 있는지 파악해야 한다. 즉, 로그의 라벨 어휘는 명목(이름) 영역을, 로그에서 유추한 전이 그래프는 구조 영역을, 기능적 수렴과 동치 관계는 의미영역을, 이벤트 순서와 타이밍은 실행 영역의 근거를 제시할 수 있다.

본 연구는 시뮬레이션 소프트웨어가 생성하는 이벤트 로그 관찰로부터 네 영역에서 감정평가의 근거를 제시하는 방법을 제안한다. 특히, 이벤트 로그의 라벨 명만 바꾸는 명목 영역을 조작하여 라벨 의존 지표를 무너뜨리는 공격과 시뮬레이션

모델이 가질 수 있는 상태를 여러 연속 이벤트로 나누어 기록하는 공격은 상태 수를 어긋나게 해 로그 간의 일대일 대응을 할 수 없게 만들어 감정을 어렵게 할 수 있다. 본 연구는 라벨 재명명과 기록 분할로 은폐된 실행 행동 유사성을 복원하고, 독립 구현 대조군과의 차이를 분석하여 후속 감정을 위한 기술적 정황을 제공하는 방법이다.

본 논문은 라벨 문자열을 전혀 사용하지 않는 특징 상태 시그니처와 헵가리안 할당으로 두 행동 모델의 상태를 대응시키는 정렬 기법을 제안한다. 또한, 하나의 상태를 여러 연속 이벤트로 쪼개어 이벤트 로그를 수정하는 경우는 약한 이중모사 개념 기반의 stutter 병합을 적용한다. 마지막으로 pyjevsim[2] 환경을 사용하여 소스 수준에서 다양한 변형에 대한 실험결과를 제시한다.

2. 관련 연구

소프트웨어 감정 실무에 있어서 소스코드 수준의 유사도 분석과 관련된 연구는 다양한 분야에서 진행된 바 있다[3-7]. 예를 들어, code2vec 임베딩으로 유사도 감정 도구의 성능을 개선하는 연구[3], 정보기기 소스코드 유사성 분석에서 감정 목적물의 검증 절차를 다룬 연구[4] 등 소스코드 확보를 전제한 감정 기법이 이어지고 있다. MOSS(Measure Of Software Similarity)[5]와 같은 학생의 프로그래밍 과제물에 관련된 표절 탐지에서도 소스 코드 확보를 가정하고 진행하고 있으며, JPlag[6]의 토큰 열 비교, 프로그램 의존 그래프로 의미 보존 변형에 대응한 GPLAG[7]는 모두 소스 코드 또는 그 토큰, 그래프 표현을 입력으로 요구한다.

본 연구는 감정 대상 소프트웨어의 소스 대조가 어려운 상황을 가정하고, 두 대상 소프트웨어

가 실행되었을 때 생성되는 이벤트 로그를 기반으로 한 알고리즘을 제시함으로써 소스 수준 감정 결과를 보완한다.

소프트웨어 저작권 분쟁과 관련하여 프로그램이 동일 소스 코드에서 파생되었는지 확인하기 위한 연구로 소프트웨어 버스마크(Birthmark)가 있다. 소프트웨어 버스마크는 소스 코드의 변수명, 함수명 등이 변경되더라도 바이너리 파일로 변환되었을 때의 구조적인 특징을 분석하는 연구이다[8]. 정적 버스마크는 코드 구조에서, k-gram 계열은 명령어 열에서 정보를 뽑는다[9]. 동적 버스마크는 Windows API나 Java API 호출 정보 실행 시 호출되는 시스템 콜을 사용하여 소스 코드의 유사도 측정 정보로 쓴다[10-12]. 제안된 소프트웨어 버스마크와 관련된 연구는 소프트웨어 코드나 바이너리에 대한 계측을 전제하고 있으나 본 연구는 산출 로그만 확보할 수 있는 상황에서 이벤트 로그 수집과 분석을 통하여 유사도 측정을 할 수 있다는 점에서 다르다.

이벤트 로그로부터 행동 모델을 정의하는 방안에 관한 연구로 프로세스 마이닝 분야에서 유사한 연구가 진행된 바 있다. 프로세스 마이닝은 로그로부터 프로세스 모델을 유추하고[13], 관측된 실제 행동과 모델 부합도를 검사한다[14]. 하지만 프로세스 마이닝 분야는 활동 라벨이 신뢰할 수 있다고 전제하므로 라벨을 적대적으로 재명명하는 위장은 다루지 않는다.

3. 제안하는 이벤트 로그 기반 분석방법

본 연구는 소프트웨어 유사성 분석을 서로 다른 관점으로 분석하여 소프트웨어 감정에 활용할 수 있도록 근거를 도출하는 알고리즘과 이벤트 로그 분석틀을 제시한다. 먼저 소프트웨어에서 생성되는 이벤트들에 관하여 명목 영역(Nominal Domain), 구조 영역(Structural Domain), 의미영역

(Semantic Domain) 및 실행 영역(Execution Domain)으로 구분하여 근거를 도출한다.

명목 영역(Nominal Domain)은 구성 요소, 구성 요소의 인터페이스, 이벤트의 이름과 로그의 라벨 어휘, 곧 식별자 수준의 유사성이다. 구조 영역(Structural Domain)은 상태 전이 그래프의 위상과 상태의 구조적 역할 등 그래프, 계층 구조에 관한 유사성을 판단할 수 있다. 의미영역(Semantic Domain)은 기능과 행위의 동등성을 파악할 수 있으므로 위장 복원의 근거로 활용할 수 있다. 실행 영역(Execution Domain)은 이벤트의 순서와 상대 타이밍 등 시뮬레이션이 실행되었을 때의 이벤트 궤적으로, 두 소프트웨어 간 파생 가능성을 검토하기 위한 실행 행동상의 기술적 정황으로 활용할 수 있다.

본 연구는 앞선 네 영역의 지표를 시뮬레이션 소프트웨어가 생성한 로그로부터 계산하고 그 가중 결합값을 S_{obs} 로 정의한다. 일반적으로 같은 명세를 구현한 소프트웨어의 경우 일반적인 환경에서 저장되는 로그가 유사할 것으로 기대되므로 정상적으로 동작하는 구간의 유사성은 파생의 증거로 삼기 어렵다. 반면 비정상 입력, 경계값, 동시 사건, 용량 초과와 같은 예외 상황 처리의 경우 두 소프트웨어가 유사한 경우 두 소프트웨어 간의 유사도가 높다고 볼 수 있다. 전자를 규정 시나리오(Specified Scenario), 후자를 탐침 시나리오(Probe Scenario)라 하고, 이를 독립 분석하며 S_{spec} 과 S_{probe} 로 정의한다.

S_{spec} 이 높다는 것은 규정 시나리오에 대한 결과가 유사하다는 의미로 의미영역에서 본래 명세대로 동작하고 그 기능이 같다는 측면에서 AFC(Abstraction-Filtration-Comparison) 여과 원칙에 따라 판별 근거에서 배제될 수 있다[15]. 이와 반대로 S_{probe} 가 높다는 것은 명세에서 직접 규정되지 않은 조건에서도 구현 선택이 유사

함을 의미하며, 다른 기술적 증거와 함께 공통 구현 선택 또는 공통 기원의 정황으로 해석할 수 있다.

이벤트 로그에 대한 명목 영역의 일치 경우 저비용으로 일치성을 조정할 수 있기 때문에 구조 영역을 기반으로 분석을 수행한다. 즉, 원 라벨 유사도와 정렬 후 유사도를 분석하였을 때 두 값이 모두 높으면, 이름과 구조가 함께 일치한다고 가정한다. 동일한 이벤트 로그의 유사도가 1이라고 가정하였을 때 원 라벨 유사도가 낮는데 정렬 후 결과가 1에 가까우면 이벤트 로그 내에 라벨 이름만 바뀌었을 뿐 행동은 같다는 내용이며, 정렬 후에도 1에 크게 못 미치면 실질적인 행동 차이가 있는 것으로 해석할 수 있다.

본 연구는 비교 대상 개발자가 원본 기능을 유지하면서 시뮬레이션 로그를 위장하는 것으로 가정하기 때문에 합리적 위장은 행동을 보존하면서 로그의 표면을 바꾸고 라벨 어휘와 기록 입도를 변경한다고 가정한다. 이때 재명명 변형 π 는 라벨 집합 위의 전단사 함수로, 구조, 의미, 실행영역을 보존한 채 명목 영역만 조작한다. 기록 분할 변형 함수 ρ 는 한 이벤트를 같은 논리 시점의 연속 이벤트 k 개로 쪼개 상태천이를 숨긴다. 따라서 본 논문은 유사도 측정을 위하여 표절자의 위장 노력을 π 와 ρ 함수로 정의하고 $S(A, \pi(A))$, $S(A, \rho(A))$, 합성 $S(A, \pi(\rho(A)))$ 를 $S(A, A)$ 수준으로 복원하여 분석을 수행하고 독립 구현 B 에 대해서는 정렬 후 유사도가 위장 변형의 복원 수준에 도달하는지를 비교하여 제안 방법의 과도한 정렬 여부를 분석한다.

이벤트는 시각과 라벨의 쌍 $e = (t, l)$ 으로 정의될 수 있다. 여기서 일반적으로 라벨은 시뮬레이션의 구성 요소, 인터페이스, 이벤트명을 결합한 문자열로 SRV.out:done과 같이 정의할 수 있다. 이벤트 로그는 한 실행에서 수집한 이벤트의 유한한 시퀀스로, 시각을 기준으로 정렬하여 동시 이벤트

의 기록 순서를 보존한다. 본 연구는 시뮬레이션 소프트웨어 로그 집합에서 1차 마르코프 전이 모델을 유추하며 이벤트 로그에 나타난 라벨을 노드로 구성된 관측 행동 모델의 상태, 즉 관측 상태 (observed state)로 정의하고 DEVS 모델의 내부 상태변수를 직접 의미하지 않는다. 구체적으로 가상 시작 상태와 종료 상태를 두고, 각 로그에 대해 시작, 첫 이벤트, 인접 이벤트 쌍, 마지막 이벤트, 종료의 전이 횟수를 센다. 상태 s 의 행 확률 $P(\cdot | s)$ 는 출력 횟수의 비율이고, 전이 (s_1, s_2) 마다 소요 시간 표본의 평균과 표준편차를 수집한다. 이때 이 모델은 구조 영역 분석의 대상이고, 시간 통계는 의미영역 문 판정과 상태를 구분하는 데 사용한다. 본 논문에서는 수치 지표로 4개의 지표를 사용하여 유사도를 추출하기 위한 지표를 산출한다.

n-gram 프로파일은 명목 영역을, 전이 구조는 구조 영역을, 이벤트 순서와 시간 정렬은 실행 영역을 센다. 의미영역은 수치 지표가 아니라 동치 구성으로 측정한다. 이벤트 순서, 시간 정렬 지표는 로그 쌍 단위로 계산되므로 A 와 B 의 로그를 이름 우선으로 대응시킨다. B 의 로그를 파일명으로 색인하고 같은 이름을 가진 A 의 로그를 B 의 순서대로 대응한다.

명목영역의 경우 n-gram 프로파일 로그마다 라벨열 앞뒤에 시작 표지 $\wedge$ 와 종료 표지 $\$$ 를 붙이고, $n \in \{2, 3\}$ 의 연속 n-gram 빈도를 센다. 양측의 모든 로그에서 또는 각 측의 모든 로그에서 누적한 빈도 벡터 v 가 프로파일이고, 유사도는 키 합집합 위에서 계산한 두 프로파일의 코사인 수식 1이다.

$$S_{\text{ngram}}(A, B) = \frac{v_A \cdot v_B}{\|v_A\| \|v_B\|} \quad (1)$$

이후 대응된 로그 쌍 (a, b) 의 라벨열에 단위 비용 편집거리를 수식 2와 같이 계산한다[16]. 동

적 계획 점화식은 수식 2이고, 경계는 $L(i,0)=i$, $L(0,j)=j$ 이며 $\mathbf{1}[\cdot]$ 는 조건이 참이면 1, 거짓이면 0인 지시함수다.

$$L(i, j) = \min \left\{ \begin{array}{l} L(i-1, j) + 1, \\ L(i, j-1) + 1, \\ L(i-1, j-1) + \mathbf{1}[a_i \neq b_j] \end{array} \right\} \quad (2)$$

쌍 유사도는 최대 길이로 정규화한 수식 3과 같이 정의하고, 지표값은 전체 쌍의 평균이다.

$$sim_{edit}(a, b) = 1 - \frac{L(|a|, |b|)}{\max(|a|, |b|)} \quad (3)$$

이벤트 간의 시각의 경우 로그별 최소-최대 정규화 $t_i = (t_i - t_1)/(t_n - t_1)$ 로 $[0, 1]$ 에 맞춘다. 스텝 비용은 라벨 불일치와 정규화 시각차의 가중합으로 수식 4로 계산하고, 누적 최소 비용은 표준 DTW(Dynamic Time Warping)[17] 점화식 수식 5로 계산한다. 경계는 $D(0, 0) = 0$, 나머지 0 행 · 0열은 ∞ 이다.

$$c(e, e') = \mathbf{1}[l \neq l'] + \alpha |\hat{t} - \hat{t}'| \quad (4)$$

$$D(i, j) = c(a_i, b_j) + \min \left\{ \begin{array}{l} D(i-1, j), \\ D(i, j-1), \\ D(i-1, j-1) \end{array} \right\} \quad (5)$$

길이 n, m 인 쌍의 누적 비용 $D(n, m)$ 을 경로 길이 근사 $(n + m)$ 으로 나눈 정규화 거리 d 를 수식 6으로 유사도로 변환하고, 전체 쌍을 평균한다. 쌍대응과 서브샘플은 편집거리와 같은 규칙을 쓰며, 시각 정규화는 서브샘플 후 열의 지속 시간을 기준으로 한다.

$$S_{dtw} = 1 - \min \left(1, \frac{d}{1 + \alpha} \right), \quad d = \frac{D(n, m)}{n + m} \quad (6)$$

구조영역의 경우 전이 구조를 분석한다. 상태별 출력 분포의 Jensen-Shannon Divergence (JSD) [19] 수식 7을 상태 사용 빈도로 가중 평균한 값의 보수가 수식 8이다. 상태 집합은 양측 모델의 합집합이고, 한쪽에만 있는 상태의 발산은 1로 둔다. 본 연구에서는 상태 가중치 w_s 를 양측 모델의 상태 사용 빈도 합에 비례하도록 수식 8과 같이 정의한다.

$$\begin{aligned} JSD(P, Q) &= \frac{1}{2} \sum_k P(k) \log_2 \frac{P(k)}{M(k)} \\ &\quad + \frac{1}{2} \sum_k Q(k) \log_2 \frac{Q(k)}{M(k)}, \\ M &= \frac{1}{2} (P + Q) \end{aligned} \quad (7)$$

$$\begin{aligned} S_{model} &= 1 - \sum_s w_s JSD(P_A(\cdot | s), P_B(\cdot | s)), \\ w_s &= \frac{f_A(s) + f_B(s)}{\sum_x (f_A(x) + f_B(x))} \end{aligned} \quad (8)$$

관측 집합 종합인 S_{obs} 값은 평가자가 결정한 가중치로 결합되어 규정/탐침 시나리오에 따라 S_{spec} , S_{probe} 를 도출한다. 수식 9의 S_{obs} 는 관측 집합 전체 또는 총화된 각 축을 가리킨다. 결측 지표의 경우 남은 가중치로 재정규화한다. 가용 지표 집합을 M 이라 하면 종합은 수식 9와 같다.

$$S_{obs} = \frac{\sum_{m \in M} w_m S_m}{\sum_{m \in M} w_m} \quad (9)$$

라벨이 없는 두 전이 그래프 사이의 대응으로 유사도 측정을 진행하기 위해서는 확률과 시간 특성을 갖는 두 관측 행동 그래프에서 구조적으로 대응 가능한 상태 후보를 찾아야 한다. 본 연구에서는 각 상태를 시그니처로 요약하고, 시그니처 거리가 정의하는 비용 행렬에서 총비용을 최소화하는 일대일 대응을 구한다. 시그니처는

여덟 특징으로 이뤄진다. 출력 확률 분포를 내림차순으로 정렬한 벡터로, 어느 라벨로 가는지 분포의 패턴을 구하고, 자기 루프 확률, 종료 전이 확률, 시작 입력 확률, 전체 출력 전이 상태 비중, 전체 입력 전이 중 상태 비중, 로그 내 정규화 위치의 발생 평균, 출력 전이 평균 소요 시간(횟수 가중)이다. 특징별 거리 d_f 는 분포 모양에는 총변동 거리 $\frac{1}{2} \| \cdot \|_1$, 스칼라에는 절대차, 시간에는 양측 최대 평균 시간으로 정규화한 절대차를 쓴다. 시간 특징은 어느 한쪽이 결측이면 제외한다. 가용 특징 집합을 $F(s, s')$ 라 하면 쌍 비용은 수식 10과 같다.

$$\text{cost}(s, s') = \frac{1}{|F(s, s')|} \sum_{f \in F(s, s')} d_f(s, s') \quad (10)$$

정렬은 비용 합을 최소화하는 일대일 대응으로 수식 11과 같다.

$$M^* = \arg \min_M \sum_{(s, s') \in M} \text{cost}(s, s') \quad (11)$$

최적해를 구하기 위해서는 비용 행렬에 포텐셜 기반 $O(n^3)$ 형가리안 알고리즘을 적용한다. 두 상태 집합의 크기가 다르면 작은 쪽을 행으로 두어 작은 쪽 전부가 대응되고, 남은 상태는 “대응 없음”으로 보고된다. 그 대응으로 소프트웨어 B의 로그를 재라벨링하고 모델을 다시 구성한 뒤 같은 지표를 재계산한 값이 “정렬 후” 유사도이다.

두 비교물의 상태 수가 일치하는 경우 두 이벤트 로그를 정렬하여 유사도 측정하기에 용이하지만 악의적인 목적으로 위장하는 경우 상태수가 다른 경우가 발생할 수 있다. 이때 상태의 개수 차이가 기록 입도에서 온 경우, 한쪽이 하나의 논리 상태를 두 개 이상의 연속 이벤트로 기록했

다면 두 기록은 의미영역에서 동치라고 볼 수 있다. 즉, 논리 시간 관점에서 한 논리 단위 안에서만 나타나는 연쇄는 하나의 상태로 간주할 수 있기 때문에 상태 연쇄 s_1 에서 s_2 로 전이되는 것이 같은 논리 단위의 내부 분해로 판정되려면 세 조건이 필요하다. Δ 를 시작 상태 발 전이를 제외한, 횟수 가중 평균 전이 시간, $\text{In}(s_2)$ 를 s_2 로의 입력 횟수 합, 문턱을 p, r 라 하면 판정은 수식 12와 같다.

$$P(s_2|s_1) \geq p, \quad \frac{\text{In}(s_2 \leftarrow s_1)}{\text{In}(s_2)} \geq p, \quad (12)$$

$$dt(s_1 \rightarrow s_2) \leq r \bar{\Delta}$$

첫째 조건은 후속지배 조건으로 s_1 의 출력이 s_2 로 집중되는가를 판단한다. 둘째는 입력 집중 조건으로 s_2 입력이 s_1 에서 유래하는가를 판단한다. 셋째는 논리적 동시성으로 전이가 논리적으로 동시인지를 판단한다. 세 조건은 각각 조건부 분기, 다중 경로 도달 상태, 실제 소요 시간이 있는 파이프라인 단계를 병합 대상에서 배제하는 가드로 작동한다. 세 개 이상 이어진 사슬은 머리 상태까지 따라가 한 그룹으로 접으며 뭉 로그를 만들 때는 같은 그룹 안의 연쇄만 첫 이벤트로 접고, 원 라벨이 같은 연속 이벤트는 반복으로 보아 보존한다. 따라서 본 논문의 병합은 동일 논리 시각에 발생하고, 외부 분기가 없으며, 선행 상태의 출력과 후행 상태의 입력이 상호 집중된 연쇄에 한정된다. 본 논문에서 제시하는 복원 성질은 이러한 조건을 만족하는 기록 분할 변형에 대한 조건부 성질이다.

4. 사례연구

본 사례연구에서는 위장 복원 성능을 전체 관측

집합의 S_{obs} 로 평가하고, S_{spec} 과 S_{probe} 는 독립 구현에서 발생한 차이의 원인을 해석하기 위한 보조 지표로 사용한다. 본 논문의 이벤트 로그는 이산사건시스템형식론 기반 모델링 및 시뮬레이션 환경인 pyjevsim으로 구현된 시뮬레이터로부터 생성하였고, 정상 흐름, 영 배치, 용량 폭주, 비정상 입력, 동시 사건, 빈 실행, 대량 배치, 공진의 여덟 시나리오를 혼합해 구성하고 규정 시나리오와 탐침 시나리오를 기반으로 이벤트 로그를 생성하였다. 기준물의 로그는 12개, 총 이벤트는 1,928건으로 구성되었다.

표 1. 구현 변형 목록
Table 1. List of Implementation Variants

변형	성격
ported	동일 빌드·동일 시드를 사용한 동일 조건 재실행
original_rep	동일 구현의 재실행 (재실행 변동 포함)
combined	renamed와 refined를 겹친 복합 위장 빌드의 직접 시뮬레이션($\pi \circ \rho$)
renamed	식별자 표를 전부 바꾼 위장 빌드의 직접 시뮬레이션 (π)
refined	완료를 동일 시각의 두 이벤트로 분할 기록하는 빌드의 직접 시뮬레이션(ρ)
independent	같은 명세를 별도로 구현한 독립 DEVS 모델

비교 대상은 표 1의 여섯 변형으로 정하고 실험을 진행하였다. 이벤트 순서 및 시간 정렬 지

표는 로그 쌍 단위로 계산하므로, 두 비교물의 로그를 동일한 입력 시나리오 식별자를 기준으로 대응한다. 본 사례연구에서는 동일한 파일명을 시나리오 식별자로 사용하였다. ported는 동일 빌드와 동일 시드를 사용한 동일 조건 재실행이고, original_rep은 동일 구현을 다른 시드로 재실행한 변형이다. renamed, refined, combined는 위장을 소스 수준에서 적용한 빌드, 곧 식별자 표를 바꿔 구성한 빌드(π), 완료 기록을 동일 시각의 두 줄로 분해하는 빌드(ρ), 두 위장을 겹친 빌드를 같은 워크로드 실행(같은 시드와 난수 스트림)으로 시뮬레이션을 수행하였다. 재실행 변동은 original_rep을 사용하여 측정한다. 이벤트 로그 생성기가 생성한 위장 빌드들의 경우 앞서 정의한 로그 수준 형식 변환 함수 π , ρ 를 기준물 산출에 적용한 결과와 정확히 일치하도록 만들었고, 소스 수준 위장과 로그 수준 변환의 일치 여부는 기준 로그에 π 와 ρ 를 적용한 결과를 오라클로 사용하여 검증하였다. independent는 동일한 명세를 바탕으로 기준 구현과 별도로 작성한 DEVS 모델이다. 표 2는 이산사건시뮬레이션 환경으로 작성한 시뮬레이션 소프트웨어의 로그를 바탕으로 도출한 데이터이다.

사례연구의 수치는 모두 로그를 종류별로 나누지 않고 통째로 하나로 보고 계산한 종합 유사

표 2. 네 지표의 원라벨/ 정렬 후 값(관측 집합)
Table 2. Raw-Label and Post-Alignment Values for the Four Metrics (Observed Set)

Variant	n-gram	Edit	DTW	Transition	S_{obs}
ported	1.000/ 1.000	1.000 / 1.000	1.000 / 1.000	1.000 / 1.000	1.000 / 1.000
original_rep	0.997/ 0.996	0.765 / 0.764	0.923 / 0.925	0.997 / 0.997	0.924 / 0.924
renamed	0.000/ 1.000	0.000 / 1.000	0.667 / 1.000	0.000 / 1.000	0.133 / 1.000
refined	0.655/ 1.000	0.833 / 1.000	0.936 / 1.000	0.743 / 1.000	0.782 / 1.000
combined	0.000/ 1.000	0.000 / 1.000	0.640 / 1.000	0.000 / 1.000	0.128 / 1.000
independent	0.000/ 0.880	0.000 / 0.699	0.660 / 0.900	0.000 / 0.908	0.132 / 0.847

도인 S_{obs} 를 사용하여 계산하고 S_{obs} 를 계산할 때 사용되는 가중치 값으로 본 실험에서는 n-gram 길이를 $n \in \{2, 3\}$, DTW 비용의 시각 차 가중치를 $\alpha = 0.5$, stutter 병합 문턱을 $p = 0.95$, $r = 0.25$ 로 설정하였다. 상태 시그니처의 특징 가중치는 균등으로 두어 가용한 특징값의 산술평균을 쌍 비용으로 사용하였다. 시간 특징은 양측 모두 관측된 경우에만 포함되므로 특징 수는 7 또는 8이다. 지표 결합 가중치는 n-gram 0.25, 편집거리 0.25, DTW 0.20, 전이 구조 0.30이다. 모든 설정값은 파이썬 참조 구현에서 가져온 것으로 본 데이터셋을 만들기 전에 고정되었고 결과를 보고 조정하지 않았다.

renamed는 원본 로그에서 이벤트 이름만 전부 바꾸고 실행 순서와 타이밍은 그대로 둔 변형으로 소프트웨어 A와 행동이 동일함에도 유사도는 0.133까지 떨어졌으며 이는 단순히 로그 내 원라벨 기반 지표만으로는 재명명 변형과 독립 구현을 구분하기 어렵다. 특히, n-gram, 편집거리, 전이 구조는 라벨 문자열을 직접 비교하므로 이름이 전부 바뀌자 모두 0.000이 되었고 이름이 겹치지 않아도 0으로 내려가지 않는 구조로 인하여 DTW만 0.667로 계산되었다. 종합값 S_{obs} 는 0.133은 DTW 유사도 0.667에 가중치 0.20을 곱한 값이다.

개명된 라벨 위에서도 동일 위치에서 stutter 병합이 적용되었다. B측에서 접힌 `Queue.hand:assign←Worker.track:working`은 소프트웨어 A의 로그를 재명명한 연쇄로, 이는 병합 판정이 라벨 문자열이 아니라 전이 특성에 기반하여 작동한 사례로 볼 수 있다. 구조영역 정렬은 표 2와 같이 병합 후 몫 상태 9쌍 전부가 시그니처 거리 0.000으로 대응되었고, 그 대응은 실제 이름 변경 관계와 일치했다. 정렬 후 종합은 1.000으로 원 라벨에서 0.133이던 값이 정렬 후에 1.0이 되었다는 것은 이름만 바뀌었을 뿐

동작은 동일하다는 뜻으로 재명명 위장의 정황이 될 수 있다.

그림 1은 관측 집합에서 기준물 A와 renamed의 전이 그래프를 중첩한 도구 화면이다. 라벨이 전부 달라 A 전용 전이와 B 전용 전이가 분리된 두 구조로 보이지만, 정렬이 이은 파선이 같은 역할의 상태끼리 연결되어 이름은 다르지만, 유사한 전이 구조와 상태 대응 관계가 시각적으로 나타난다.

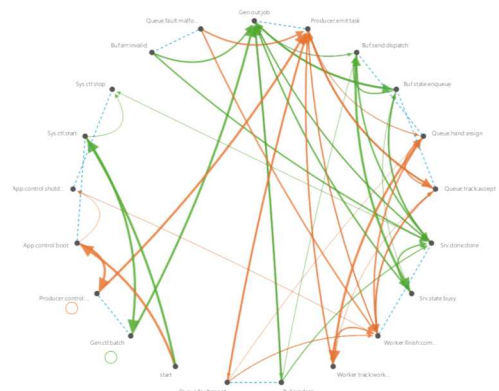


그림 1. 기준 구현과 재명명 변형의 전이구조 및 상태 대응

Fig. 1. Transition Structures and State Correspondences between the Reference and Renamed Variants

refined는 기록 분할로 이벤트 수가 1,928건에서 2,291건으로 증가하고 관측 상태 수가 달라지도록 구성된 변형이다. 라벨 어휘가 대부분 보존되므로 원라벨 종합은 0.782로, 어휘에 기대는 지표들이 0이 되지 않았고, 새로 끼어든 이벤트도 363건으로 전체의 16%에 그친다. 나머지 84%는 원본과 그대로 일치하므로 값이 낮아지지 않았다. stutter 병합의 세 조건은 B측에서 공통 정준 연쇄에 더해 삽입 이벤트 363건이 만든 분할 연쇄 `Srv.done:done←Srv.state:cleanup`을 정확히 검출

해 접었고, 병합 후 정렬에서 1.000을 회복하였다.

combined는 두 위장을 겹친 합성 $\pi(p(A))$ 다. refined의 분할 연쇄에 전면 이름 변경을 적용하고 분할로 생긴 이벤트에도 별도 이름을 부여했다. combined는 renamed와 refined를 함께 적용한 변형이다. 원라벨 종합은 0.128로 여섯 변형 중 가장 낮았다. 이름을 바꿨으므로 어휘에 기대는 세 지표가 0.000이 되었고, 동일 시각의 삽입 이벤트로 DTW 정렬 경로가 달라져, DTW 유사도도 renamed의 0.667보다 낮은 0.640으로 감소하였다. 종합 0.128은 이 0.640에 DTW 가중치 0.20을 곱한 값이다. 위장은 겹쳤지만 복원은 순서대로 이뤄졌다. stutter 병합이 개명된 라벨 위에서 정준 연쇄와 함께 분할 연쇄 Worker.finish:complete←Worker.track:sweep을 동일한 상태로 인식하였고, 이어진 정렬이 뭉치 상태 9쌍 전부를 평균 시그니처 거리 0.000의 교차 대응으로 잇자 1.000이 회복되었다. 세 위장 모두 라벨 정보 없이 복원되었고, 본 실험 조건에서는 복합 위장에 따른 추가적인 회피 효과가 관찰되지 않았다.

본 연구는 정렬이 독립 구현을 과도하게 동일화하지 않는지도 확인하였다. independent는 기준물의 라벨 10종과 어휘를 전혀 공유하지 않는 독립 구현으로, 원라벨 종합이 0.132이고 renamed의 원라벨 0.133과 사실상 같은 값을 가졌다. independent 자신의 Gate.fwd:grant ← Proc.run:exec이 먼저 자체 정준화로 접힌다. 이어진 정렬은 뭉치 상태 9쌍을 평균 시그니처 거리 0.034로 대응시켰다.

renamed에서는 아홉 쌍 모두 거리가 0.000이었지만, independent에서는 평균 0.034가 남았다. 정렬 후 종합도 0.847에 그쳐 복원된 위장들의 1.000과 0.153 차이가 나며 편집거리는 정렬 후에도 0.699로 이름을 맞춰 주어도 이벤트가 일어난

순서 자체가 다르다는 뜻으로 해석할 수 있다.

로그를 종류별로 증화하지 않았는데도 독립 구현과의 차이가 관찰된 이유는 수집한 워크로드에 있다. 비정상 입력, 동시 사건, 용량 폭주처럼 명세가 행동을 정해 두지 않은 조건이 섞여 있었고, 그 구간에서 두 구현이 갈라졌기 때문이다. 관측 집합의 워크로드에 명세가 침묵하는 탐침성 조건(비정상 입력, 동시 사건, 용량 폭주 등)이 섞여 있으므로, 본 사례에서는 증화하지 않은 S_{obs} 에서도 차이가 관찰되었다.

진양성 대조군도 확인하였다. ported는 원라벨 1.000이 정렬 후에도 1.000으로 유지되었다. 양측에서 동일한 정준 연쇄가 같은 뭉치로 잡히고 교차 대응은 없어 대칭 정준화가 파생 신호를 훼손하지 않는다는 것을 확인하였다. original_rep은 재실행 변동으로 원라벨이 0.924인데, 하라분은 편집거리 0.765로 이벤트 순서가 주도하고 정렬 후에도 0.924 그대로였다. 정렬이 파생 신호를 흐트러뜨리지도, 재실행 변동을 인위적으로 지우지도 않는다는 뜻으로 해석할 수 있다.

독립 구현은 규정 축에서 0.904, 탐침 축에서 0.634를 기록했다. 명세가 정해 둔 정상 동작에서는 기준물과 거의 같게 움직이지만 명세가 침묵하는 조건에서는 갈라진다는 뜻이며, 이는 본 독립 구현에서 관찰된 양상이다. 격차를 만든 것은 네 지표 중 둘이다. n-gram 프로파일이 0.998에서 0.417로, 전이 구조가 0.997에서 0.647로 떨어졌다. 나머지 둘은 축이 바뀌어도 거의 움직이지 않았다. 편집거리는 0.692에서 0.633으로, DTW는 0.914에서 0.885로 조금 낮아졌을 뿐이다. 두 축의 차이는 주로 n-gram 프로파일과 전이구조 지표에서 나타났다.

5. 결론

본 논문은 소프트웨어 유사성을 명목, 구조, 의

미 및 실행의 네 영역으로 구분하고, 시뮬레이션 소프트웨어의 이벤트 로그에서 각 영역의 기술적 근거를 도출하는 파생 정황 분석 방법을 제안하였다. 명목 영역의 라벨 재명명은 라벨 비의존 관측 상태 정렬로 대응하고, 기록 입도 차이는 동일 논리 시점의 연쇄에 대한 stutter 병합으로 정규화하였다. 대기행렬 시뮬레이션 사례에서 재명명, 기록 분할 및 복합 위장 변형은 원라벨 유사도를 감소시켰으나, 제안한 정렬과 병합 후 기준 구현과의 유사도가 복원되었다. 반면 동일 명세를 별도로 구현한 독립 구현은 정렬 후에도 1.000에 도달하지 않아 본 사례에서 위장 변형과의 차이가 유지되었다.

본 연구의 복원 성질은 진단사 재명명과 동일 논리 시점의 무분기 분할 연쇄라는 조건에서 확인되었으며, 실험 또한 하나의 대기행렬 모델과 하나의 독립 구현을 대상으로 수행되었다. 향후에는 복수 모델과 독립 구현, 통계적으로 동치인 반복 실행, 대응 거부를 포함한 부분 상태 정렬 및 실행 순서와 시간까지 변경하는 위장에 대한 검증이 필요하다.

본 과제(결과물)는 2026년도 교육부 및 대전광역시 재원으로 대전엑스센터의 지원을 받아 수행된 지역성장 인재양성체계(앵커)(2026-ANCHOR-06-002)와 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구(Grant RS-2025-25438354)의 결과입니다.

참 고 문 헌

- [1] B. P. Zeigler, A. Muzy, and E. Kofman, Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations, 3rd ed. Academic Press, 2019.
doi: 10.1016/C2016-0-03987-6.
- [2] C. Choi, “pyjevsim: A DEVS Modeling & Simulation Environment,” ver. 2.1.1, PyPI/GitHub, 2026. [Software]. Accessed: Aug. 7, 2026. [Online]. Available: <https://pypi.org/project/pyjevsim/2.1.2/> and <https://github.com/eventsim/pyjevsim>.
- [3] T. H. Um, S. M. Hong, J. H. Yang, H. S. Jang, and K. G. Doh, “Enhancing the performance of code-clone detection tools using code2vec,” Journal of Software Assessment and Valuation, vol. 17, no. 1, pp. 31 - 40, 2021.
doi: 10.29056/jsav.2021.06.05.
- [4] S. Nam, D. H. Kim, and K. T. Lee, “Comparison procedure in evaluation analysis of source code comparison on Embedded system,” Journal of Software Assessment and Valuation, vol. 17, no. 2, pp. 31 - 38, 2021.
doi: 10.29056/jsav.2021.12.04.
- [5] S. Schleimer, D. S. Wilkerson, and A. Aiken, “Winnowing: Local algorithms for document fingerprinting,” in Proc. 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD '03), 2003, pp. 76 - 85.
doi: 10.1145/872757.872770.
- [6] L. Prechelt, G. Malpohl, and M. Philippsen, “Finding plagiarisms among a set of programs with JPlag,” Journal of Universal Computer Science, vol. 8, no. 11, pp. 1016 - 1038, 2002.
doi: 10.3217/jucs-008-11-1016.
- [7] C. Liu, C. Chen, J. Han, and P. S. Yu, “GPLAG: Detection of software plagiarism by program dependence graph analysis,” in Proc. 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '06), 2006, pp. 872 - 881.
doi: 10.1145/1150402.1150522.
- [8] H. Tamada, M. Nakamura, A. Monden, and

- K. Matsumoto, "Design and evaluation of birthmarks for detecting theft of Java programs," in Proc. IASTED International Conference on Software Engineering, Innsbruck, Austria, 2004, pp. 569 - 574. ACTA Press. ISBN: 0-88986-381-4.
- [9] G. Myles and C. Collberg, "K-gram based software birthmarks," in Proc. 2005 ACM Symposium on Applied Computing (SAC '05), 2005, pp. 314 - 318.
doi: 10.1145/1066677.1066753.
- [10] H. Tamada, K. Okamoto, M. Nakamura, A. Monden, and K. Matsumoto, "Dynamic software birthmarks to detect the theft of Windows applications," in Proc. International Symposium on Future Software Technology (ISFST 2004), Xi'an, China, 2004. ISBN: ISBN4-916227-17-4
- [11] D. Schuler, V. Dallmeier, and C. Lindig, "A dynamic birthmark for Java," in Proc. 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE '07), 2007, pp. 274 - 283.
doi: 10.1145/1321631.1321672.
- [12] X. Wang, Y. C. Jhi, S. Zhu, and P. Liu, "Behavior based software theft detection," in Proc. 16th ACM Conference on Computer and Communications Security (CCS '09), 2009, pp. 280 - 290.
doi: 10.1145/1653662.1653696.
- [13] W. M. P. van der Aalst, Process Mining: Data Science in Action, 2nd ed. Berlin/Heidelberg, Germany: Springer, 2016. doi: 10.1007/978-3-662-49851-4. ISBN: 978-3-662-49850-7.
- [14] A. Rozinat and W. M. P. van der Aalst, "Conformance checking of processes based on monitoring real behavior," Information Systems, vol. 33, no. 1, pp. 64 - 95, 2008.
doi: 10.1016/j.is.2007.07.001.
- [15] Computer Associates International, Inc. v. Altai, Inc., 982 F.2d 693 (2d Cir. 1992).
<https://law.justia.com/cases/federal/appellate-courts/F2/982/693/137252/>.
- [16] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," Soviet Physics Doklady, vol. 10, no. 8, pp. 707 - 710, 1966.
<https://ui.adsabs.harvard.edu/abs/1966SPhD..10..707L/abstract>
- [17] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 26, no. 1, pp. 43 - 49, 1978.
doi: 10.1109/TASSP.1978.1163055.
- [18] H. W. Kuhn, "The Hungarian method for the assignment problem," Naval Research Logistics Quarterly, vol. 2, nos. 1 - 2, pp. 83 - 97, 1955.
doi: 10.1002/nav.3800020109.
- [19] J. Lin, "Divergence measures based on the Shannon entropy," IEEE Transactions on Information Theory, vol. 37, no. 1, pp. 145 - 151, 1991. doi: 10.1109/18.61115

저 자 소 개



유예동(Yea Dong Yoo)

2003.2. 육군3사관학교 기계공학과 학사
 2018.8. 경희대학교 경영학과 석사
 2018.12.~2019.11. 육본 기참부 지상무인체계기획장교
 2019.12.~2020.11.육본 시험평가단 시험평가장교
 2023.4.~현재 육군 미래혁신연구센터 지향성에너지기술연구장교·첨단드론기술연구장교
 2024.2.~현재 국립한밭대학교 우주국방융합학과 박사과정
 <주관심분야> 국방 모델링 및 시뮬레이션, 소프트웨어 공학



조병현(Byeonghyeon Cho)

2021.3 - 현재 국립한밭대학교 컴퓨터공학과 재학
 <주관심분야> 소프트웨어 공학, 모델링 및 시뮬레이션, 소프트웨어 감정 및 포렌식



장수영(Sooyoung Jang)

2006.2 한국과학기술원 산업공학과 졸업
 2008.8 한국과학기술원 산업및시스템공학과 석사
 2014.2 한국과학기술원 산업및시스템공학과 박사
 2014.3-2017.9 삼성전자
 2017.10-2023.2 한국전자통신연구원
 2023.3-현재 : 국립한밭대학교 교수
 <주관심분야> 소프트웨어공학, 강화학습, 인공지능응용



양승호(Seungho Yang)

2009.2 서울대학교 지구환경시스템공학부 졸업
 2014.8 서울대학교 건설환경공학부 박사
 2015.3-2017.4 한아도시연구소 팀장
 2017.5-2018.3 고양시정연구원 연구원
 2018.6-2021.1 York University 박사후 연구원
 2021.3-현재 : 국립한밭대학교 교수
 <주관심분야> 도시설계, 스마트도시



최창범(Changbeom Choi)

2005.2 경희대학교 컴퓨터공학과 학사
 2007.2 한국과학기술원 전산학 석사
 2014.8 한국과학기술원 전자공학 박사
 2014.9-2021.2 한동대학교 부교수
 2021.3-현재 : 국립한밭대학교 부교수
 <주관심분야> 체계공학, 모델검증, 국방 모델링 및 시뮬레이션