논문 2021-1-7 http://dx.doi.org/10.29056/jsav.2021.06.07

클라우드네이티브 애플리케이션 구축을 위한 마이크로서비스 식별 방법

최옥주*, 김유경**†

Identification of Microservices to Develop Cloud-Native Applications

Okjoo Choi*, Yukyong Kim**†

요 약

최근 주목받고 있는 마이크로서비스는 독립적으로 개발될 뿐만 아니라 독립적으로 실행 및 배포가 가능하다는 장점 때문에, 클라우드 컴퓨팅 환경에서 보다 유연한 확장과 효율적인 협력을 보장할 수 있다. 이러한 영향으로 최근 마이크로서비스지향 애플리케이션 환경으로의 전환이 급격히 증가하고 있다. 마이크로서비스의 도입을 위해서는 무엇보다 모노리식 아키텍처로 구축된 단일 애플리케이션의 구성요소를 마이크로서비스 단위로 식별하는 문제가 선결되어야 한다. 본 논문에서는 레거시 시스템으로부터 마이크로서비스 식별의 문제를 알고리즘 기반으로 해결하기 이한 접근방법을 제안한다. 코드의 메타정보를 이용하여 그래프를 생성하고 클러스터링 알고리즘을 적용하여 마이크로서비스 후보를 추출한다. 추출된 마이크로서비스 후보에 대해 메트릭을 이용하여, 모듈화 품질을 평가한다. 또한 제안된 식별 방법의 효과를 검증하기 위해 벤치마크를 위해 많이 사용되는 공개 소프트웨어의 코드를 이용하여 후보 서비스를 도출하고, 메트릭을 이용하여 모듈화 수준을 평가한다. 결과적으로 좀더 작은 단위의 마이크로서비스로 식별해 내면서 모듈품질을 향상시키는 결과를 확인할 수 있다.

Abstract

Microservices are not only developed independently, but can also be run and deployed independently, ensuring more flexible scaling and efficient collaboration in a cloud computing environment. This impact has led to a surge in migrating to microservices-oriented application environments in recent years. In order to introduce microservices, the problem of identifying microservice units in a single application built with a single architecture must first be solved. In this paper, we propose an algorithm-based approach to identify microservices from legacy systems. A graph is generated using the meta-information of the legacy code, and a microservice candidate is extracted by applying a clustering algorithm. Modularization quality is evaluated using metrics for the extracted microservice candidates. In addition, in order to validate the proposed method, candidate services are derived using codes of open software that are widely used for benchmarking, and the level of modularity is evaluated using metrics. It can be identified as a smaller unit of microservice, and as a result, the module quality has improved.

한글키워드 : 마이크로서비스, 모노리식 아키텍처, 서비스 식별, 결합도, 클러스터링, 모듈화

keywords : microservice, monolith architecture, service identification, coupling, clustering, modularity

* 한국과학기술원 전산학부
** 숙명여자대학교 기초공학부
† 교신저자: 김유경(email: ykim.be@sookmyung.ac.kr)

접수일자: 2021.05.31. 심사완료: 2021.06.18.
게재확정: 2021.06.20.

1. 서 론

최근 비즈니스 환경 변화의 주기가 짧아지면서 고객의 요구사항이나 시장 변화에 신속하게 대응하기 위해서 스피드와 유연성을 담보로 하는 클라우드 네이티브(cloud-native) 개념이 주목받고 있다. 클라우드 네이티브 애플리케이션 개발은 클라우드 컴퓨팅 기술을 기반으로 애플리케이션을 구축하고 실행 및 개선하는 접근방식이다. 클라우드 네이티브는 클라우드 컴퓨팅 모델을 사용하는 것을 전제로 설계된 시스템과 서비스이기 때문에, 마이크로서비스 아키텍처(Microservice architecture, MSA)나 데브옵스(DevOps), 애자일(Agile) 방법론과 같은 현대적인 애플리케이션 배포 방법을 기반으로 개발된다 [1].

마이크로서비스는 서비스가 독립적으로 개발되어, 독립적으로 실행 및 배포가 가능하다는 장점이 있다. 따라서 실행이 가능한 업무 단위인 마이크로서비스 블록으로 나누어 RestFull API와 같이 단순한 방법으로 서비스 간 상호 통신이 가능하고 연계할 수 있으므로, IaaS 환경에서 보다 유연한 수평적 확장과 개발자 간의 보다 효율적인 협력이 가능해진다고 할 수 있다. 이러한 영향으로 최근 많은 기업들이 초기 모노리식 아키텍처(Monolithic architecture)에서 마이크로서비스 지향 애플리케이션 환경으로 전환하기 위해 진지한 노력을 기울이고 있다.

그러나 기존의 소프트웨어 시스템에서 마이크로서비스로 마이그레이션하는 것은 굉장히 복잡한 작업이며, 모노리식 아키텍처로 구축된 단일 애플리케이션의 구성요소를 식별하는 문제에 직면하게 되는데, 분할의 기준이나 분할 옵션의 평가 기준이 명확하지 않기 때문에, 개발자의 경험과 노하우에 크게 의존할 수 밖에 없는 상황이다. 본 논문에서는 레거시 시스템으로부터 마이크로서비스 식별의 문제를 알고리즘 기반으로 해결하고자 한다. 이를 위해, 코드의 메타정보를 이용하여 그래프를 생성하고 클러스터링을 통해 마이크로서비스 후보를 추출한다.

본 논문의 구성은 다음과 같다. 2장에서는 마이크로서비스의 개념과 함께 마이크로서비스 식별과 관련된 기존의 연구를 기술한다. 3장에서는 추출 모델, 커플링기반 가중치 계산과 클러스터링 알고리즘을 제시하고, 4장에서는 제안된 모델의 유효성을 검증하기 위한 실험 및 평가결과가 제시된다. 마지막으로 5장에서는 결론 및 향후 연구 방향에 대해 논의한다.

2. 관련 연구

시스템을 독립적인 하위 시스템으로 분해하는 것은 소프트웨어공학 분야에서 수년간 진행되온 작업이다. Parnas는 최초로 시스템을 모듈로 분해할 때 사용해야 하는 기준을 체계적으로 제안하였다 [2]. Parnas 이후 여러 연구에서 몇가지 접근방식이 제안되었으며, 최근에는 클라우드 네이티브 특히 마이크로서비스 아키텍처로 인해 시스템 분해가 다시 관심을 받고 있다.

[3]에서 저자들은 마이크로서비스 동향을 파악하기 위해, 다양한 규모의 42개 기업에서의 서비스 컴퓨팅 관행을 조사하였다. 특히 서비스 규모와 복잡성이 검토되었는데, 서비스의 크기는 매우 다양하지만, 100 LOC 미만인 매우 작은 서비스 또는 10,000 LOC 이상인 매우 큰 서비스와 같은 극단적인 형태는 실제로 거의 발생하지 않았음을 보였다. 또한 서비스들은 상대적으로 매우 간결하고 국한된 업무에 전용되는 형태로 이용되고 있다는 사실을 발견하였다. 모노리식 아키텍처는 소프트웨어의 모든 구성요소가 한 프로젝트에 통합되어 있는 형태이다. 하지만 일정 규모 이상의 서비스, 혹은 많은 개발자가 투입되는

프로젝트에서 이런 통합된 형태의 모노리식 아키텍처는 한계를 보이기 때문에, 최근에는 모노리식 아키텍처를 MSA로 마이그레이션하는 시도가 많아지고 있다.

MSA 마이그레이션을 위해서는 모노리식에서 마이크로서비스를 추출하는 것이 필수적인 요구사항이 된다. [4]의 연구에서는 클라이언트와 서버 구성요소, 데이터베이스 테이블 및 비즈니스 영역 간의 종속성 그래프를 구성하여 대규모 단일 기업 시스템에서 마이크로서비스를 식별하기 위한 체계적이지만 수동적인 접근 방식을 제시한다.

모노리식 코드 베이스에서 마이크로서비스를 식별하기 위한 연구로서 주목할 만한 시도는 ServiceCutter이다 [5]. 이 연구에서는 그래프 절단(Cutting)을 통해 구조화된 서비스 분해를 지원하는 도구를 구현하였다. 분해될 시스템의 구조에 대한 표현과 결합 기준은 문헌과 업계 노하우에서 추출한 16가지 기준의 카탈로그를 기반으로 한다. 그러나 ServiceCutter는 모노리식 소프트웨어의 코드로부터 필요한 아키텍처 정보를 마이닝하거나 구성할 수 있는 수단을 제공하지는 못하기 때문에, 카탈로그를 기반으로 하는 특정 예상 모델을 통해서만 소프트웨어 산출물을 제공할 수 있다는 한계가 있다.

[6]에서는 레거시 코드에 정적분석(Static analysis) 기술을 적용하여 도메인 용어(Terms)를 추출하고, 토픽 모델링(Topic modeling)을 통해 서비스를 식별하였다. 토픽 모델은 소프트웨어 문서들에 산재해 있는 토픽들을 추출하는 모델링 방법으로, 이 연구에서는 도메인 용어들에 대응하는 시스템 토픽을 식별하는 방법을 적용하였다.

전통적으로 소프트웨어 분해나 모듈화와 관련되어 소프트웨어 메트릭이나 저장소 마이닝 또는 정적 분석과 같은 여러 기술들이 사용될 수 있다. 본 논문에서는 이런 기존의 분해 기법들의 여러 장점들을 적용하여 마이크로서비스 식별을 위한 접근방법 및 설계 원리를 제공하고자 한다.

3. 마이크로서비스 식별모델

본 논문에서 제안하는 서비스 식별모델은 모노리식 소프트웨어의 코드를 정적분석하여 각 모듈 또는 클래스와 같은 시스템의 단위들 간에 이루어지는 호출관계를 그래프로 표현하고, 그래프의 클러스터링 과정을 통해 후보 마이크로서비스를 도출하는 과정으로 구성되어 있다.

먼저 그래프 생성 단계에서는, 모노리식 코드의 호출관계 정보를 마이닝 하여, 가중치를 갖는 그래프(Weighted graph) G를 생성한다. 그림 1과 같이 표현되는 그래프 $G=(V, E)$에서 정점 $v_i \in V$는 모듈 (또는 클래스)를 나타낸다. 본 논문에서는 클래스 단위와 유사한 정도의 크기로 구성된 소프트웨어의 구성 단위를 모두 클래스라고 명명하기로 하고, 이것을 그래프의 정점으로 표현하기로 한다. 그래프의 간선 $e_j \in E$은 방향을 갖지 않으며 가중치 함수에 의해 산출된 가중치가 부여된다. 가중치 함수는 커플링 전략에 따라 인접한 간선들 사이의 결합이 얼마나 강하게 이루어 졌는지를 나타낸다. 가중치가 클수록 결합이 더욱 강하다는 것을 뜻한다.

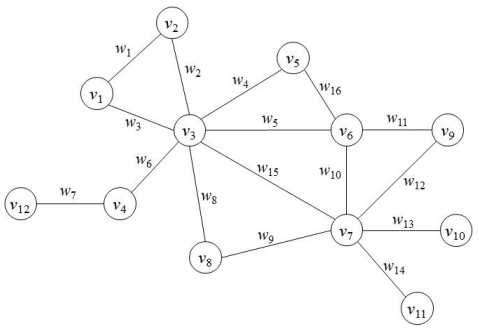

그림 1. 가중치 그래프 G = (V, E)
Fig. 1. A weighted graph G

다음 단계에서는 그래프를 분할하기 위해 클러스터링 알고리즘을 사용한다. 그래프 정점의 클러스터링을 통해, 그래프는 여러 조각으로 분할되며, 각 조각은 마이크로서비스 후보가 된다.

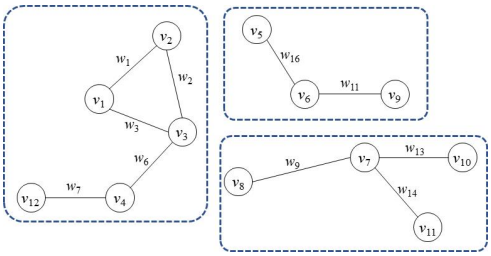

그림 2. 마이크로서비스 후보군 생성
Fig. 2. Creating microservice candidates

그림 2와 같이 클러스터링으로 모노리식에 대한 마이크로서비스 후보군이 생성된다. 각 클러스터는 연결된 컴포넌트(Connected components)로 구성된 그래프로서 각각 마이크로서비스 후보로 정의되며, 따라서 도출된 부분그래프의 개수가 권장 마이크로서비스의 개수라고 할 수 있다.

3.1 커플링 전략에 따른 가중치 계산

가중치 함수는 커플링 전략에 따라 인접한 간선들 사이의 결합이 얼마나 강하게 이루어 졌는지를 나타내며, 논리적인 결합 강도를 고려하여 정의한다. 논리적 결합의 개념은 일반적인 모듈 설계의 목표에서 찾아 볼 수 있다. 설계 목표로서 단일 책임 원칙(Single Responsibility Strategy)은 모듈이나 클래스와 같은 소프트웨어의 구성 단위가 하나의 책임을 갖도록 정의되어, 변경하게 되는 주된 원인이 하나만 있어야 한다고 규정하고 있다 [7]. 따라서 같은 이유로 변경이 이루어지는 소프트웨어 요소들은 하나의 단위로 모아져야 한다. 또한 마이크로서비스는 실행 시간의 의존성을 최소화 해야 하므로, 강력한 모듈 경계를 적용해야 한다. 강력한 모듈 경계와 단일 책임 원칙을 통해 시스템에 변경이 일어나는 경우, 변경할 모듈만 찾으면 되고 제한된 수의 모듈만 이해하면 된다는 장점이 있다. 이런 관점에서 보자면, 단일 책임 원칙이 적용되도록 마이크로서비스가 정의되어야 하며 동시에 함께 변경이 이루어지는 클래스 파일과 같은 소프트웨어 구성요소들이 동일한 마이크로서비스로 정의되어야 한다. 이러한 근거로 커플링 전략에 따른 가중치가 계산된다.

논리적 결합 강도를 측정하기 위해 모노리식와 해당 모노리식의 변경 기록에 있는 각 변경 이벤트를 다음과 같이 정의한다. 모노리식 M의 변경 기록을 H_M이라고하면, 변경기록은 변경 이벤트들의 시퀀스라고 할 수 있으므로 $H_M = (h_1, h_2, ..., h_n)$로 정의한다. H_M에 있는 각 변경 이벤트는 $h_i = (C_i, t_i)$가 되며, 여기에서 C_i는 해당 이벤트에 의해 변경이 이루어진 클래스 파일과 같은 소프트웨어 구성 요소들의 집합이고, t_i는 해당 변경 이벤트가 발생한 타임스탬프이다. 그러면 한 변경 이벤트 h_i에 의해 변경이 이루어진 임의의 두 클래스 $c_1, c_2 \in C_i$에 대해, $\gamma_{h_i}(c_1, c_2)$는 두 클래스가 함께 변경되었는지를 나타내는 것으로 이벤트 h_i에 의해 함께 변경이 되었다면 1로, 그렇지 않다면 0으로 값을 할당한다. 그러면 두 클래스의 논리적 결합 강도 LC(Logical coupling)는 다음 (식 1)과 같이 정의할 수 있다.

$$LC(c_1, c_2) = \sum_{h_i \in H_m} \gamma_h(c_1, c_2) \quad \cdots\cdots\cdots \text{(식 1)}$$

그래프 생성 과정에서 계산된 LC 값을 이용하여 그래프의 간선들에 대한 가중치를 계산한다. 그래프의 정점이 클래스를 표현하게 되므로, 임의의 두 정점 $v_1, v_2 \in V$을 연결하는 간선 $e_i \in E$은 $e_i = (v_1, v_2)$이고, 가중치는 $w(e_i) = LC(v_1, v_2)$로 계산한다.

3.2 클러스터링 알고리즘

생성된 모노리식 그래프로부터 마이크로서비스 후보군을 추출하기 위해서는 그래프를 분할하고, 부분그래프들(subgraphs)로 정의해야 한다. 이 과정은 모노리식 그래프에서 간선들을 제거하여, 가장 강력하게 연결된 간선들만 남겨 놓는 방식을 사용한다. 가중치 그래프에서 간선의 가중치를 이용하여 간선의 일부를 제거하는 방법으로는 최소 신장 트리(Minimum spanning tree)를 생성하는 과정으로 생각해 볼 수 있으므로, 크루스칼 알고리즘(Kruskal's algorithm)이나 프림의 알고리즘(Prim's algorithm) 등의 적용이 가능하다. 본 논문에서는 가장 강력하게 연결된 간선만을 남겨 놓아야 하므로, 최대 신장 트리(Maximum spanning tree, MST)를 생성하기 위해 사용되는 크루스칼 알고리즘에서 최소의 가중치를 선택하는 대신 최대의 가중치를 선택하는 것으로 수정하여 적용해 볼 수 있다.

그래프 분할은 삭제할 간선을 결정할 때 간선의 가중치를 고려해야 하는데, 본 논문에서 제시하는 모델은 최대신장트리(MST)를 생성해야 한다. 따라서, 두 클래스 정점 사이의 간선에 부여된 가중치가 크다는 것은 두 클래스가 동일한 서비스 후보에 속한다는 것을 의미하므로, 결과적으로 삭제할 간선은 가중치가 낮은 간선들이 우선적으로 고려된다.

간선의 삭제가 항상 그래프의 분할을 보장할 수 있도록 하려면, MST만을 고려하기로 한다. 이것은 MST의 간선이 하나씩 삭제될 때마다, MST가 두 개의 구성요소로 분할이 된다는 것을 의미한다.

클러스터링의 목표는 높은 가중치를 갖는 간선으로 연결된 클래스 노드들이 동일한 그래프 분할에 남도록 하는 것이다. 따라서 그래프 G에 대한 MST(G)는 3.1에서 계산된 가중치 $w(e_i)$가 가장 큰 주요 간선들을 유지하게 된다.

표 1의 알고리즘에서, while 반복 단계에서 결합이 가장 낮은 에지는 MST에서 삭제되고 나머지 연결된 구성 요소를 탐색하여 반복하게 된다. 분할이 수행된 후 남아 있는 간선들은 clusterReduction 함수로 전달되어 비정상적으로 큰 클러스터가 없는지 확인하는 과정을 거친다. clusterReduction은 클러스터 크기가 매우 큰 경우, 간선의 가중치가 가장 큰 클래스 노드를 따로 분할을 하고 해당 클래스를 삭제했을 때 남게 되는 노드들의 그룹으로 다시 분할을 하는 방법으로 조절을 한다. 이렇게 하면, 연결된 구성 요소 내부의 내부 결합 수준을 높게 유지하면서 클러스터 크기를 줄일 수 있다.

표 1. 클러스터링 알고리즘
Table 1. Clustering algorithm

```
1: function Cluster(E, s)
2: for e in E do
3:     E_MST <- Krusal(e)
4: endfor
5: while E_MST is not empty do
6:     E_MST <- delete e' ∈ E_MST
7:     Search(E_MST)
8: endwhile
9: clusters <- clusterReduction(E_MST, s)
10: return clusters
11: endfunction
```

4. 실험 및 분석

본 논문에서 제시된 마이크로서비스 식별모델을 설명하기 위하여, 관련된 연구에서 벤치마크를 위해 널리 사용되고 있는 JPetStore[1] 온라인 펫샵을 이용한다. JPetStore는 계정관리, 카테고리 관리, 쇼핑카트 관리, 주문 관리로 이루어져 있다. JPetStore는 4개 패키지에 39개 클래스로

1) https://github.com/mybatis/jpetstore-6/

구성되어 총 4,932 LOC 규모이다.

각 소프트웨어 컴포넌트 단위에 대한 모듈화의 품질을 평가하는 두 메트릭을 이용하여, 식별된 마이크로서비스의 품질평가 결과를 비교하였다. 모델 1은 대상 소프트웨어인 JPetStore에서 정의한 39개 클래스 각각을 하나의 마이크로서비스로 대응하여 정의한 것이다. 모델 2는 제안된 식별 방법으로 정의된 마이크로서비스로서 전체 39개 클래스로부터 48개 마이크로서비스를 식별하였다. 모듈화 품질 평가 메트릭은 [8]에서 정의한 SM(Coupling based Structural Metrics)와 AM(Architectural Metrics)를 이용하였다. 다음 표 2는 사용된 메트릭들이다.

표 2. 평가 메트릭
Table 2. Quality Metrics

SM	MII	Module Interaction Index
	NC	Non-API Function Closedness Index
	APIU	API Function Usage Index
	IDI	Implicit Dependency Index
AM	CDI	Cyclic Dependency Index
	LOI	Layer Organization Index
	MISI	Module Interaction Stability Index
	NTDM	Normalized Testability-Dependency Metric

각 서비스들에 대해 계산된 메트릭 값의 평균을 비교하여 그래프로 작성하였다. 가장 이상적으로 잘 설계된 모듈의 경우 메트릭 값은 1에 가까워진다. 평가 결과에 따르면, 그림 3의 그래프 상에서, 모델 1보다 모델2의 메트릭 값의 분포가 좀 더 이상적인 경우로 나타난다. 온라인 펫샵 애플리케이션은 이미 모듈화가 잘 되어 있는 잘 정의된 클래스들로 구성되어 있으므로, 제안된 식별 방법과의 차이가 두드러지게 나타나지는 않고 있으나, 크기를 줄이는 것뿐만 아니라, 모듈화 품질이 향상되었다고 볼 수 있으므로, 제안된 식별 방법을 적용하는 것이 좀 더 잘 설계된 마이크로서비스를 구현하기 위해 유용할 수 있다.

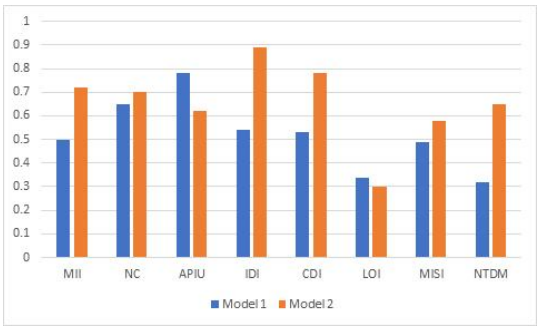

그림 3. 메트릭 값의 비교
Fig. 3. Comparison of metric values

최근의 연구에서는 마이크로서비스를 식별하기 위해 비정형적인 시나리오 모델이나, 데이터 흐름도를 이용하고 있다 [9, 10, 11]. 또한 설계단계에서 마이크로서비스를 식별하기 위해 요구사항 명세서로부터 기능적 분해를 적용하기도 한다 [12]. 이러한 방법들은 MSA 도출을 위해 수동적인 설계(manual design)과정을 거치게 되는데, 모노리식 소프트웨어를 마이크로서비스로 분해하는 것은 매우 복잡한 작업이기 때문에, 새로운 자동화된 접근방식이 필요하게 된다 [13]. 이런 연구와 비교해 본다면, 본 논문에서 제안한 방법은 개발된 코드의 메타정보를 기반으로, 그래프 생성과 클러스터링 알고리즘을 적용하기 때문에 자동화가 가능한 체계적인 접근 방식으로 적절한 추상화 수준의 서비스를 식별하는데 명확하고 객관적이라고 할 수 있다.

5. 결 론

최근 주목받고 있는 클라우드 네이티브 개념

은 클라우드 컴퓨팅 모델을 사용하는 것을 전제로 설계된 소프트웨어 시스템으로 데브옵스나 애자일 방법론, MSA와 같은 현대적인 애플리케이션 배포 방법을 기반으로 하고 있다. MSA 구성요소인 마이크로서비스는 독립적으로 개발될 뿐만 아니라 독립적으로 실행 및 배포가 가능하다는 장점 때문에, IaaS 환경에서 보다 유연한 확장과 효율적인 협력을 보장할 수 있다. 이러한 영향으로 최근 마이크로서비스 지향 애플리케이션 환경으로의 전환이 급격히 증가하고 있다.

마이크로서비스의 도입을 위해서는 무엇보다 모노리식 아키텍처로 구축된 단일 애플리케이션의 구성요소를 마이크로서비스 단위로 식별하는 문제가 선결되어야 한다. 본 논문에서는 레거시 시스템으로부터 마이크로서비스 식별의 문제를 알고리즘 기반으로 해결하기 위한 접근 방법을 제안하였다. 코드의 메타정보를 이용하여 그래프를 생성하고 클러스터링 알고리즘을 적용하여 마이크로서비스 후보를 추출하였다. 또한 제안된 식별 방법의 효과를 검증하기 위해 온라인 펫샵의 구현된 코드를 이용하여 모듈화 평가 메트릭을 계산하였다. 크기를 줄여서 좀 더 작은 규모의 마이크로서비스 단위로 식별해 내면서 모듈품질을 향상시키는 결과를 확인할 수 있다.

향후 연구과제로서, 코드의 메타정보를 얻기 위한 입력물로서 클래스 보다 더 세분화된 단위를 고려하여 MSA 의 구조적 모듈화와 정확성을 좀 더 향상 시킬 수 있는 접근 방법을 모색하고자 한다. 또한 좀 더 규모가 크고, 다양한 사례에 적용하여, 보다 객관적인 비교 검증이 이루어져야 할 것이다.

참 고 문 헌

[1] David Taibi, Kari Systä, “A Decomposition and Metric-Based Evaluation Framework for Microservices”, Proceedings of the International Conference on Cloud Computing and Service Science, pp.133-149, June 4, 2020. DOI: https://doi.org/10.1007/978-3-030-49432-2_7

[2] D. L. Parnas, “On the criteria to be used in decomposing systems into modules”, Communications, ACM, Vol. 15, no. 12, pp. 1053-1058, 1972. DOI : https://doi.org/10.1145/ 361598.361623

[3] G. Schermann, J. Cito and P. Leitner, “All the services large and micro: Revisiting industrial practice in services computing”, Proceedings of the International conference on service orineted computing, pp.36-47. LNCS volume 9586, April 26, 2016. DOI: https://doi.org/10.1007/978-3-662-50539-7_4

[4] A. Levcovitz, R. Terra, and M. T. valente, “Towards a technique for extracting microservices from monolithic enterprise systems”, Proceedings of the Workshop on Software Visualization, Evolution and Maintenance, pp.97-104, May 2016. https://arxiv.org/abs/1605.03175

[5] M. Gysel, L. K¨olbener, W. Giersche, and O. Zimmermann, “Service Cutter: A systematic approach to service decomposition”, Proceedings of the European Conference on Service-Oriented and Cloud Computing, pp. 185 - 200, August 25, 2016. DOI : https://doi.org/10.1007/978-3-319-44482-6_12

[6] Miguel brito, Jacome Cunha, Joao Saravia, “Identification of microservices from monolithic applications through Topic modeling”, Proceedings of the Annual ACM Symposium on Applied Computing, pp.1409-1418, March 2021. DOI : https://doi.org/10.1145/3412841.3442016

[7] Robert C. Martin, “Agile Software Development, Principles, Patterns, and Practices”, Prentice-Hall, 2002. ISBN-10: 0135974445

[8] S. Sarkar, G. M. Rama, Avinash C. Kak, “API-based and Information-Theoretic Metrics for Measuring the Quality of Software Modularization”, IEEE Transactions on Software Engineering, vol. 33, no. 1, pp.14 - 32, Jan. 2007. DOI: https://doi.org/10.1109/TSE.2007.256942
[9] M. J. Amiri, “Object-aware identification of microservices”, Proceedings of the IEEE International Conference on Services Computing, pp253-256, 2018. DOI: https://doi.org/10.1109/SCC.2018.00042
[10] L. Baresi, M. Garriga, and A. De Renzis, “Microservices identification through interface analysis”, Proceedings of the European Conference on Service-Oriented and Cloud Computing, pp. 19-33, Sep. 2017. DOI: https://doi.org/10.1007/978-3-319 -67262-5_2
[11] S. Li, et al., “A dataflow-driven approach to identifying microservices from monolithic applications”, Journal of Systems and Software, vol. 157, article no. 110380, 2019. DOI: https://doi.org/10.1016/j.jss.2019.07.008
[12] S. Tyszberowicz, et al., “Identifying microservices using functional decomposition”, Proceedings of the International Symposium on Dependable Software Engineering: Theories, Tools, and Applications, pp.50-65, 2018. DOI: https://doi.org/10.1007/978-3-319-99933-3_4
[13] C. Schröer, F. Kruse, and J. M. Gómez, “A Qualitative Literature Review on Microservices Identification Approaches”, Proceedings of the Symposium and Summer School on Service-Oriented Computing, 2020, DOI: https://doi.org/10.1007/978-3-030-64846-6_9

저 자 소 개

최옥주(Okjoo Choi)

2008.2 숙명여자대학교 컴퓨터과학과 박사
1990.8-1996.3 LG전자생산기술원 주임원구원
1996.7-2009.8 한국오라클 수석컨설턴트
2009.9-현재 한국과학기술원 전산학부 연구부교수(산학협력중점교수)
<주관심분야> 데이터사이언스, 소프트웨어 품질, 프로젝트 관리

김유경(Yukyong Kim)

2001.8 숙명여자대학교 컴퓨터과학과 박사
2005.9-2006.8 UC Davis, Post-doc.
2006.9-2013.9 한양대학교 컴퓨터공학과 연구교수
2018.2-현재 숙명여자대학교 기초공학부 교수
<주관심분야> 웹서비스 QoS 평가, SOA 기반 IoT 신뢰 평가, 소프트웨어 품질평가