DEÜ FMD 25(74), 395-404, 2023

Dokuz Eylül Üniversitesi Mühendislik Fakültesi Fen ve Mühendislik Dergisi
Dokuz Eylul University Faculty of Engineering Journal of Science and Engineering

Basılı/Printed ISSN: 1302-9304. Elektronik/Online ISSN: 2547-958X

Yazılım Mühendisliğinde Gözlemlenebilirlik ve İzleme Üzerine Bir Tasarım Şablonu

Observability and Monitoring Design Pattern in Software Engineering

Savaş Takan [1], Fatih Soygazi [2*]

[1] Ankara Üniversitesi, Mühendislik Fakültesi, Yapay Zeka ve Veri Mühendisliği Bölümü, Ankara, TÜRKİYE
[2] Aydın Adnan Menderes Üniversitesi. Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Aydın, TÜRKİYE
*Sorumlu Yazar / Corresponding Author **: fatih.soygazi@adu.edu.tr

Geliş Tarihi / Received: 23.07.2022 *Araştırma Makalesi/Research Article*
Kabul Tarihi / Accepted: 24.09.2023 *DOI:10.21205/deufmd.2023257411*
Atıf şekli/ How to cite: TAKAN, S.,SOYGAZİ,F.(2023). Yazılım Mühendisliğinde Gözlemlenebilirlik ve İzleme Üzerine Bir Tasarım Şablonu. DEUFMD, 25(74),395-404.

Öz

Yazılım geliştirmede gözlemlenebilirlik ve izlemenin rolü oldukça kritiktir ve her geçen gün artmaktadır. Gözlemlenebilirlik ve izlenebilirlik sağlayan mevcut araçlar, harici yapıda bulundukları için sistemin derinliklerine girebilen ve gerekli bilgiye bu yolla ulaşabilen bir yapıda değillerdir. Ayrıca bu araçların özelleştirilmesi ve tek yönlü (içeriden dışarı) bilgi sağladıkları için modifiye edilerek yeniden konfigüre edilmesi zordur. Pek çok araç, yazılım çöktüğünde yazılımın sahiplerine çökme nedenlerini gönderir ancak bu konuda bir standardizasyon bulunmamaktadır. Dolayısıyla bu sistemlerin geri bilgilendirme başarımı tartışma konusudur. Çalışmamızda, yukarıda sözü edilen problemlere çözüm sunmak amacıyla, yazılımdan ayrı tutulup, sisteme entegre edilmeyen bir gözlemleme ve izleme anlayışının yerine, sistemin bir parçası olan ve onunla birlikte yaşayan canlı bir gözlemleme ve izleme tasarım şablonu geliştirilmiştir. Tasarım şablonunun geliştirilmesinde, gözlemlenebilirlik ve izlenebilirlik açısından özetleme mekanizmasından yararlanılmıştır. Bu doğrultuda, yazılım üretim araçlarına ve paradigmalarına uyumu açısından çizge yapısından yararlanılmış ve bu yapı, önerdiğimiz tasarım şablonunun temelini oluşturmuştur. Geliştirdiğimiz tasarım şablonu ve algoritma ile lineer bir karmaşıklıkta sistemdeki verinin güncellenebilmesi sağlanmaktadır. Ayrıca döngüsüz çizge yapısının desteklenmesinin yanı sıra Etiket adı verdiğimiz yapı sayesinde, çizge yapısındaki döngüler desteklenir hale getirilmiştir. Son olarak geliştirdiğimiz yapı gözlemlenebilirlik ve izlenebilirlik açısından blokzinciri veri yapısı ile karşılaştırılmış ve sonuçta geliştirdiğimiz modelin zaman/alan karmaşıklığının daha başarılı olduğu tespit edilmiştir.

Anahtar Kelimeler: *Gözlemlenebilirlik, İzlenebilirlik, Tasarım Şablonu*

Abstract

The importance of observability and monitoring in software development is growing daily. Existing tools that provide observability and monitoring are in an external structure, which prevents them from reaching the required information in the system deeply. In addition, these tools are difficult to personalize, modify, and reconfigure due to their unidirectional (inside-out) data flow. When software crashes, many tools send the software owners the reasons for the crash, but there is no standardization in this regard. Consequently, the feedback performance of these systems is debatable. To address the issues above, we have developed a live observation and monitoring design pattern that is an integral part of the system, as opposed to an observation and monitoring approach that is separate from the software and not integrated into the system. In developing the design template,

blockchain, the most modern and trustworthy technology for observability and monitoring, was utilized by using the hashing mechanism of the blockchain. In this direction, it has been ensured that the blockchain supports the graph structure in terms of its compatibility with software production tools and paradigms, and this structure serves as the foundation of the design pattern we propose. Our proposed design template and the algorithm provides the update of the data stored in the system. Furthermore, the proposed data structure, named Tag, supports the cycles in the graph data structure as well as the acyclic graphs. Finally, the structure we developed was compared to blockchain technology in observability and monitoring. It was determined that our model was superior to the data model in terms of time and space complexity.

Keywords: *Observability, Monitoring, Design Pattern*

1. Giriş

Yazılım geliştirmede gözlemlenebilirlik ve izlemenin rolü oldukça kritiktir ve her geçen gün artmaktadır. Gözlemlenebilirlik ve izleme birbirini tamamlayan, ancak her biri farklı bir amaca hizmet eden iki kavramdır. Kontrol teorisinden kaynaklanan gözlemlenebilirlik kavramı, bir sistemin dış çıktılarını inceleyerek iç durumunu anlama yeteneğini tanımlamak için mühendislik alanında kullanılır [1,2]. Gözlemlenebilirliği kolaylaştırmak, yazılım tasarlarken kritik öneme sahiptir, çünkü yalnızca arızaların nerede meydana geldiğine değil, aynı zamanda neden ve nasıl olduğuna dair eyleme geçirilebilir bilgiler sağlar. Gözlemlenebilirlik, hem yazılım bileşenlerinin kendisini hem de bunlar arasında dolaşan verileri kapsar [3]. Bu bütünsel bakış açısına ulaşmak için, üç tür telemetri verisine vardır: Günlükler (olay kayıtları), metrikler (zaman içinde ölçülen performans verileri) ve izler (sistemdeki veri akışı hakkında bilgi) [2,4–6]. Gözlemlenebilirlik, çok büyük miktarda bilgiyi toplamayı, depolamayı ve analiz etmeyi mümkün kılar ve birden fazla teknoloji söz konusu olduğunda bile geliştiricilere o ortamda neler olup bittiğine dair eksiksiz bir çıktı üretmeyi hedefler.

İzleme ise gözlemlenebilirliğin alt kümesi olarak düşünülebilir. En genel ifade ile izleme, sistemde neyin niçin problem yarattığını yanıtlamak üzere kullanılır. Bu açıdan izleme, günlükleri, metrikleri ve izleri kullanarak sistemin durumunun izlenmesini ve anlaşılmasını ifade eder. Çalışan sistemlerde izleme eylemi, bu sistemlerin normal davranıştan ne zaman sapmaya başladığının algılanmasını ve böylece bilinen bir dizi hata modunun rahatlıkla belirlenmesini sağlar [7]. Bunun yanı sıra izleme, uzun vadeli eğilimleri analiz etmek, gösterge tabloları oluşturmak ve uyarı vermek için de oldukça önemlidir [8]. Ayrıca izleme, uygulamaların nasıl çalıştığının, nasıl büyüdüğünün ve nasıl kullanıldığının bilinmesini sağlar.

Gözlemlenebilirlik ve izlenebilirliğin sağlanmasında bazı zorluklar ortaya çıkabilmektedir. Sürekli değişimin karmaşıklığı, bu zorluklara örnek gösterilebilir. Çünkü eski uygulamaların ve iş yüklerinin, hızla değişen ortamlara adaptasyonu karmaşıklığa yol açabilmektedir [9]. Gözlemlenebilirlik ve izlenebilirlik sağlayan mevcut araçlar, harici yapıda bulundukları, yani sisteme entegre olmadıkları için sistemin derinliklerine girebilen ve gerekli bilgiye bu yolla ulaşabilen bir yapıda değillerdir [10]. Mevcut araçların tümü, standart belli başlı özellikler altında gelmektedir. Dolayısıyla bu araçların özelleştirilmesi zordur. Aynı zamanda mevcut yapılar, tek yönlü (içeriden dışarıya) bilgi sağladıkları için, bu yapıların modifiye edilerek yeniden konfigüre edilmesinde problemler yaşanmaktadır [11]. Bu durum, kullanıcı ile sistem arasındaki iletişimi ve etkileşimi azaltmaktadır. Mevcut araçlar, parametrik olma açısından katı bir yapıdadır ve bu nedenle yeni parametreler oluşturma konusunda problem yaratabilmektedir [12]. Pek çok yazılım, yazılım çöktüğünde yazılımın sahiplerine çökme nedenlerini gönderir ancak bu konuda bir standardizasyon bulunmamaktadır. Dolayısıyla bu sistemlerin geri bildirim başarımı tartışma konusudur [13]. Mevcut araçların çoğu, yazılım geliştirme aşamasında izlenebilirliği amaçlar. Bu nedenle, yazılım sonrasında ortaya çıkabilecek sorunlara adaptasyon konusunda yetersizdir [14, 15].

Çalışmamızda, yukarıda sözü edilen problemlere çözüm sunmak amacıyla, yazılımdan ayrı tutulup, sisteme entegre edilmeyen bir gözlemleme ve izleme anlayışının yerine, sistemin bir parçası olan ve onunla birlikte yaşayan canlı bir gözlemleme ve izleme anlayışı geliştirilmiştir.

Bu anlayış çerçevesinde, kendi başına yazılımın ve dolayısıyla kendisinin problemlerini çözebilen, yazılımı tüm yaşam süresi boyunca gözlemleyebilen ve izleyebilen, yapay zeka ve makine öğrenme algoritmalarıyla adaptif hata toleransına sahip olabilme potansiyeli bulunan, belirli bir standardizasyonu ortaya koyan, parametrik esnekliği arttıran, sade ve çift yönlü iletişim sayesinde kullanıcı etkileşimini artıran yeni bir tasarım şablonu önerilmiştir. Önerdiğimiz tasarım şablonunda gözlemlemenin, *günlük, metrik ve iz* adı altındaki üç özelliği yer almaktadır.

Önerdiğimiz tasarım şablonunun geliştirilmesinde, gözlemlenebilirlik ve izlenebilirlik açısından en güncel ve güvenilir teknoloji olan blokzincirinden yararlanılmıştır. Ancak yazılım üretim araçlarına ve paradigmalarına baktığımızda, bu yapıların çizge veri yapısı üzerinde inşa edildiği görülmektedir. Blokzinciri, gözlemlenebilirlik ve izlenebilirlik açısından her ne kadar iyi bir teknoloji olsa da döngüsel çizge yapısını desteklememektedir. Bu nedenle öncelikle, yukarıda değinilen problemleri çözümlemek üzere, çizge yapısını desteklemesi için blokzinciri teknolojisinde değişiklik sağlanmıştır.

Çalışmamızda öncelikle, konu ile ilgili araştırmalara değinilmiştir. Ardından, önerdiğimiz tasarım şablonu detaylı şekilde açıklanmıştır. Ayrıca geliştirdiğimiz yapı, gözlemlenebilirlik ve izlenebilirlik açısından blokzinciri veri yapısı ile karşılaştırılmış ve sonuçlar ayrıntılı şekilde ortaya konmuştur. Bununla birlikte modelin önemli işlevsellikleri örnek senaryolarla test edilip ispatlanmıştır. Son olarak, geliştirilen model tartışmaya açılarak sonraki çalışmalar için önerilerde bulunulmuştur.

2. İlgili Çalışmalar

Yazılım mühendisliğinde pek çok kişinin çalıştığı projelerde, projedeki verilerin durumu hakkında bilgi edinmek üzere gözlemleme ve izleme metodolojilerinden yararlanılır [16]. İzlenebilirlik ve gözlemlenebilirlik konusundaki mevcut problemlere çözüm amacıyla geliştirdiğimiz tasarım şablonunda blok zincirinin özet yapısı kullanılmıştır. Bunun nedeni, blok zincirinin özet yapısının verideki ve modeldeki değişiklikleri anında farkedip gerekli güncellemeleri yapabilme yeteneğine sahip bir teknoloji sunmasıdır. Dolayısıyla bu bölümde, izlenebilirlik ve gözlemlenebilirlik konusunda blok zinciri odaklı araştırma ve önerilere yer verilmiştir.

Dağıtık defter teknolojisi (Distributed Ledger Technology, DLT) ve blokzincirinin ilerlemesi ile beraber iş süreci modelleri veya işlemlerin (transaction) kayıtlanma açısından ele alınması gerekliliği meydana gelmiştir. Ladleif ve arkadaşları [17], DLT teknolojileri ve blokzincirinin operasyonel semantiğini ele alan bir yaklaşımı bazı örnek durum çalışmaları ile ele almıştır. Bu durum çalışmalarında iş süreçleri hakkında bilgi elde etmek için kayıt mekanizmasının öneminden bahsedilmiştir. Ciccio ve arkadaşları [18], blokzinciri üzerinde iş süreci izleme ile ilgili sıkıntılar üzerinde durmuştur. Bu çalışmada, veri yönetimi ve akıllı kontratlar ile ilgili problemli durumlar açıklanmıştır. Bu problemlerin tümü blokzincirinin veri ve süreç bağımlı yapısı ile ilişkili olup çalışmamızda önerdiğimiz gibi ayrıştırılmış bir veri bazlı çözüm anlatılan sorunların üstesinden gelebilecektir. Gözlemlenebilirlik için önem arz eden bir diğer faktör olan izler, günlüklerin bir tür gösterimidir. Bu yüzden izler için üretilmiş bir veri yapısı günlük verisini tutar. Karumuri ve arkadaşları [19], izler için uygun bir veri yapısı olarak yönlü düz ağaçlardan (directed acyclic graph, DAG) yararlanarak düğümlerin fonksiyon çağrıları ve kenarların düğümler arasındaki nedensel sıralamayı gösterebileceğini ifade etmiştir. Tzanettis ve arkadaşları [10], ortamdan gelen sinyalleri günlükler, ölçümler ve izler olarak sınıflandırmıştır. Sonrasında örnek bir dağıtık IoT uygulaması içerisinde bir orkestrasyon mimarisi üzerinde odaklanmışlardır. Günlükler ve izler dışında dağıtık sistemlerde gözlemlenebilirlik için bir diğer önemli nokta metriklerdir. Farklı uygulamalar için çeşitli metriklerle ölçümlemeye ihtiyaç duyulmaktadır. Örnek olarak Alhamazani ve arkadaşları [20] bulut tabanlı izleme araçları üzerinde farklı metriklerin başarı faktörü olarak nasıl ele alındığını açıklamıştır. Bir bulut ortamındaki hesaplamalı analiz için veri bazlı metrikler, donanımsal kaynak kullanımlarını göz önüne almaktadır. Bu doğrultuda ölçümlenmesi gereken metriklerin CPU hızı, CPU kullanım oranı ve disk verimi gibi metrikler olduğu görülmektedir [20]. Bunların yanı sıra yazılım mühendisliği bakış açısı ile farklı metrikler de tanımlanabilir.

Çalışmamızda, gözlemlenebilirlik unsurları olan günlükler, ölçümler ve izlerin herhangi bir çalışma alanına bağımlılığın ortadan kaldırılarak yazılım mühendisliği perspektifinden esnek bir

yöntemle incelenmesi amaçlanmıştır. Bu doğrultuda, mevcut önerilerden farklı olarak yalnızca tasarım anında değil, tüm yazılım mühendisliği süreçlerinde rahatlıkla bulunabilen kodun içine gömülerek kod ile birlikte yaşayan, mantıksal süreçlerde karar alma mekanizmalarına katılan ve ürün son kullanıcıya teslim edilse dahi yazılımın bir koruyucusu olarak işlev gören yeni bir gözlenebilir ve izlenebilir tasarım şablonu önerilmiştir.

3. Materyal ve Metot

Geliştirdiğimiz model ile blokzinciri arasındaki iki temel fark, modelin veriden ayrılması ve veri ile ilişkilerde değişebilir ve değişemez olma özelliklerinin bulunmasıdır. Modelin veriden ayrılması, verisel tekrarların yanı sıra, bakım ve alan masraflarını azaltmaktadır. Değiştirilemez ve değişebilir veri ve ilişkileri desteklemesi sayesinde döngüsel çizge yapıları modellenebilir hale gelmekte ve böylece daha esnek tasarımlar geliştirilmektedir. Örneğin Şekil 1'de görülebileceği gibi Etiket 2 - Etiket 4 - Etiket 3 bir döngü oluşturmakta; Etiket 2 - Etiket 4 değiştirebilir geçişi sayesinde tasarım modellenebilmektedir.

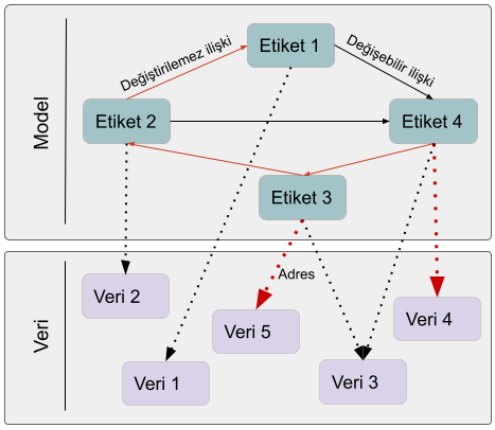
Geleneksel blokzinciri yapısı <*data, prevHash*> şeklindedir. Bu yapıda veri, sadece *data* içerisinde saklanmaktadır. Yeni veri, *prevHash* kullanılarak, bir önceki veriye eklemlenir. Çalışmamızda önerdiğimiz yapıda ise veri üç kısma bölünmüştür. Birincisi *meta*; ikincisi *data*, üçüncüsü ise *tanık çizgesi* (witnessing graph) kısmıdır. Çalışmamızda Etiket olarak adlandırdığımız yapı <*meta, data, witnessing graph, stamp*> şeklinde formüle edilmektedir. Meta, data ve tanık çizgesi yapıları, Etiket yapısı içinde "gösterge" (pointer) olarak tutulmaktadır.

Önerilen modelin meta kısmında değişebilen veriler bulunmaktadır. Böyle bir veri yapısına duyulan ihtiyaç, bir tablonun, kendisi değişmediği halde sahiplerinin değişebilir olmasının sağlanmasına benzetilebilir. Data kısmı ise değişemez veriyi barındırır. Bu analojiye göre tabloyu yapan, meta kısmında değil, data kısmında bulunmalıdır. Çünkü bir tablonun ressamı sabittir, değişmez. Önerilen veri yapısında esneklik sağlayabilmek için, data ve meta birer gösterge olarak modellenmiştir.

Önerilen modelde Etiket'ler veri taşıyabilir veya başka Etiket'lere tanıklık yapabilir. Tanık Çizgesi ise tanıklardan oluşan bir yapıdır. Diğer bir deyişle, içerisinde tanık Etiket'lerini barındırır. Bu tanıklar sayesinde özet fonksiyonu hesaplanır. Tanık Çizgesi sayesinde, Etiket'lere ulaşmak mümkündür. Önerilen veri yapısı, kanıt Etiket'lerin göstergelerini kendi içerisinde tutmaktadır. Bu da onlara kolaylıkla erişimi mümkün kılmaktadır. Ayrıca, blokzincirinden farklı olarak, önerilen yaklaşımda Şekil 1'de de görülebileceği üzere Tanık Çizgesi ile veri ağı birbirinden ayrı tutulmuştur. Tanık Çizgesi, tanık Etiket'lerinden oluşmaktadır. Veri ağındaki veri ise *meta* ve *data* bölümlerinden oluşmaktadır. Diğer bir husus, örneğin bir Etiket'in tanıkları olarak sahip olduğu bazı Etiket'lerin değiştirildiği anlaşılırsa, bu Etiket tanıkların içerisinden çıkarılmaktadır. Bu durum sistemdeki gözlemlenebilirlik ve izlenebilirliği sağlmaktadır. Bir Etiket'in güvenilir olması için en az bir tane tanık Etiket'ine ihtiyaç duyulmaktadır. Bu durumu sağlamayan Etiket'lere Şüpheli Etiket adı verilmektedir.

3.1. Blokzincirinden Esinlenilen Veri Yapısı

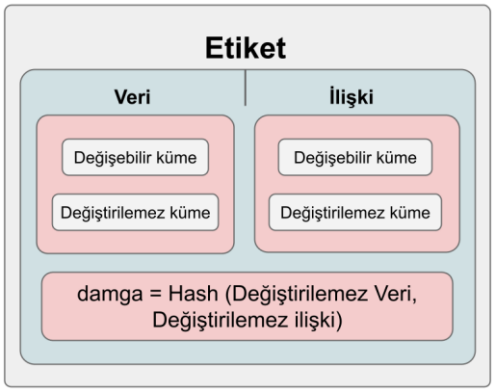
Geliştirdiğimiz modeldeki tüm unsurlar "Etiket" olarak kabul edilir. Etikete ait veri, adres olarak tutulmaktadır. Burada verinin kendisi yerine adresin tutulmasının nedeni, adresin veriden daha az yer kaplaması, tekrarı engellemesi, esnek tasarım ve kolay bakım imkanı sunmasıdır. Etiket adını verdiğimiz modelde, verinin ilişkilerini tutan değişebilir ve değişemez adresler ile birlikte değişebilir ve değişemez verilerin adresleri bulunmaktadır. Modelde her etiket, en az bir veri tutabilmektedir. Değişebilir ilişki ve veriye sahip olduğundan dolayı etiketler üzerinde ekleme, yaratma, değiştirme ve silme işlemleri rahatlıkla gerçekleştirilebilmektedir. Değişebilir ilişkiler, her türlü ilişki yapısını modelleyebilmek amacıyla adlandırılabilmektedir. Bir etiketin silinmesi, değiştirilmesi, yaratılması ve eklenmesi durumunda, ona bağlı olan etiketlerin özet değerleri yeniden hesaplanarak güncellenir. Geliştirdiğimiz etiket ağının genel tasarımı Şekil 1'de gösterilmiştir.



Şekil 1: Etiket ağı

Figure 1: Tag network

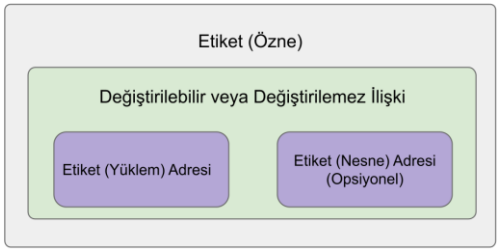
Şekil 1'de görülebileceği gibi, geliştirdiğimiz yapıda model ile veri birbirinden ayrılmıştır. Örneğin geliştirdiğimiz yapıda Etiket 4 ve Etiket 3, aynı anda Veri 3'e bağ oluşturmaktadır. Burada Veri 3'ün iki ayrı kopyasının yerine yalnızca adresleri tutulmaktadır. Bu durum, veri tekrarının yanı sıra bakım ve alan maliyetlerini azaltmaktadır. Ayrıca Etiket 3, Veri 5 adında değiştirilemez bir veriye, Veri 3 adında değiştirilebilir bir veriye sahiptir. Bu durum, geleneksel blok zincirinin blok ve veri ilişkisini kapsayan bir yapı oluşturur. Buradaki tek fark, verinin de ilişkiler gibi değişebilir ve değişemez özelliklerinin bulunmasıdır. Etiketlerde, değişebilir ve değişemez yapıların bulunmasının en temel nedeni, tasarım kolaylığı sunması ve döngüsel çizge gibi pek çok modelin gerçeklenebilmesini sağlamasıdır. Diğer yandan blok zincirinin özetleme yapısında değişebilir ilişkiler ve veriler bulunmadığı için tasarımsal ve verisel manipülasyonlar oldukça zordur. Şekil 2'de, geliştirdiğimiz etiket yapısı genel özellikleriyle gösterilmiştir.



Şekil 2: Etiket yapısı

Figure 2: Tag structure

İlişkiler, iki adet etiket adresinden oluşur. Bunlardan ilki zorunlu, ikincisi ise opsiyoneldir. Opsiyonel ilişki bulunmasının nedeni, bazı ilişkilerin sadece bir adet etiket almasıdır. Modeldeki ilişki yapısı, ilişkinin değişebilir ya da değiştirilemez olması farketmeksizin bu şekilde ifade edilir. Şekil 3'te, ilişki yapısı gösterilmektedir.



Şekil 3: İlişkilerin genel gösterimi

Figure 3: General representation of the relations

Geliştirdiğimiz modelde[1] değişmezlik kontrolü, özet mekanizması ile sağlanmıştır. Burada özet, değişemez ilişkilerin ve verinin özetini ifade eder. İstenildiği zaman özet hesaplanır ve özet kümesine eklenir. Geliştirdiğimiz veri yapısında bu işlem, *lock* fonksiyonu ile gerçekleştirilmektedir. Daha sonra, etikette herhangi bir değişiklik olup olmadığını kontrol etmek için yeni bir özet değeri hesaplanır ve özet kümesinin içerisindeki eski özet değerleri ile karşılaştırılır. Bu andan sonra, etiketin bağlantılarında veya değiştirilemez veride bir

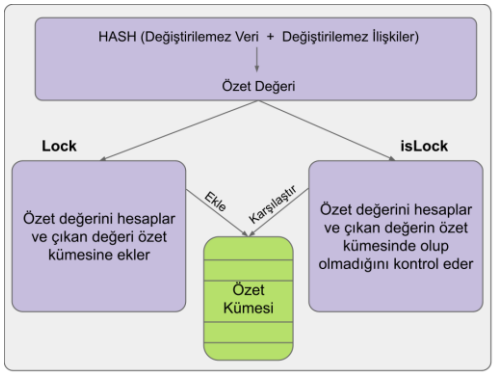
[1] https://github.com/kursuApp/tag/

```
def update(graph, parent, weight_cost, start_edge):
    stack = [start_edge]
    visited = []
    while stack:
        edge = stack.pop()
        visited.append(edge)
        for neighbor_edge in graph.neighbors(edge):
            if neighbor_edge not in visited:
                visited.append(neighbor_edge)
                if edge not in parent :
                    parent[neighbor_edge] = edge
                    weight_cost[neighbor_edge] = weight_cost[edge] + graph[neighbor_edge]['weight']
                    stack.append(neighbor_edge)
def run(graph,start_edge):
    weight_cost = defaultdict(int, {edge: 0 if start_edge else -sys.maxsize for edge in graph.edges})
    weight_cost[start_edge] = 0
    while len(required) > 0:
        update(graph, parent, weight_cost, start_edge)
```

Algoritma 1: Değişikliklerin çizge yapısında güncellenmesi için kullanılan sözde kod

Algorithm 1: The pseudo code used for updating changes in the graph structure

değişiklik gerçekleşirse, farklı özet çıkacağından, verinin değişip değişmediği ve değişmişse bulunduğu pozisyon otomatik olarak tespit edilecektir. Çıkan sonuçlar eşitse yapının değiştirilmediği; sonuçlar eşit değilse yapının değiştirildiği anlamına gelir. Bu işlem, *isLock* fonksiyonu ile gerçekleştirilmektedir. Etiketin *Lock* ve *isLock* fonksiyonlarının genel çalışma prensibi Şekil 4'te gösterilmiştir.



Şekil 4: Lock ve isLock fonksiyonlarının çalışma prensibi

Figure 4: The execution mechanism Lock and isLock functions

Örnek özet bulma formülü aşağıdaki gibidir:

$$hash(\ değişemez\ veri + \sum_{j=0}^{n} değişemez\ ilişki) \quad (1)$$

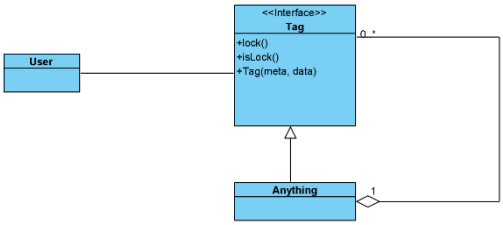
Algoritma 1 sayesinde, düşük karmaşıklık ile değişikliklerin sistemdeki tüm noktalara taşınması sağlanmaktadır. Algoritmada güncellemeler, değişemez ilişkilerin bulunduğu modelin çizgesinde gerçekleşir. Bu algoritma, derinlik ilk arama ile dinamik programlama kullanılarak geliştirilmiştir. Sistem çok yönlü bir çizge olduğundan, bütün geçişleri gezmesi için düğümlerle kenarlar yer değiştirilmiştir. Böylece tüm kenarların gezilebilmesi sağlanmıştır. Bu sayede tüm sistem O(E) karmaşıklığı ile gezilmiş olur. Sonuç olarak lineer bir karmaşıklık ile bütün sistem güncellenmektedir. Algoritma 1, döngüsüz çizge yapısını desteklese de, geliştirdiğimiz tasarım şablonunda yarattığımız Etiket yapısı sayesinde, çizge yapısındaki döngüleri destekler hale getirilmiştir. Örneğin Şekil 1'deki Etiket 2 - Etiket 4 - Etiket 3 arasındaki ilişki döngü oluşturmaktadır. Fakat Etiket 4 ile Etiket 3 ve Etiket 3 ile Etiket 2 arasındaki değişebilir ilişkiler sayesinde, oluşan döngüler problem olmaktan çıkmaktadır.

"Bir etiketin birden fazla veri alabilme özelliği" sayesinde, geliştirdiğimiz Etiket veri yapısı Blokzinciri veri yapısını kapsar hale getirilmiştir.

Geliştirdiğimiz modelde, güven mekanizması bulunmaktadır. Güven mekanizması, modelin içindeki değiştirilemez ilişkilerin toplamından meydana gelir. Güven mekanizması bu yönüyle, etiketlerin sıralanmasını sağlar. Yani, güven değeri yüksek olan Etiket, daha güvenlidir ve üst sıralarda yer alır.

3.1. Blokzincirinden Esinlenen Veri Odaklı Bir Tasarım Şablonu

Şekil 5'te önerdiğimiz tasarım şablonu ana hatları ile gösterilmektedir. Önerdiğimiz tasarım şablonu, var olan kompozit tasarım şablonuna benzese de (yaprak kısmı eksik), burada fonksiyonların ve Etiket arayüzünün işlevleri farklıdır. Sistemde her varlık bir Etiket olarak tutulmakta ve Etiket'ler de Etiket'lerden meydana gelmektedir.



Şekil 5: Gözlemlenebilir ve izlenebilir tasarım şablonu

Figure 5: Observability and monitoring design pattern

Tasarım şablonu açısından, Etiket arabirimini uygulayan her sınıf, eklenebilir ancak değişmez bir özelliğe sahip olacaktır. Şekil 5'teki görselde, Kullanıcı sınıfı, Etiket'in kilit işlevini kullanarak yeni eklemeleri engeller. Burada Etiket sınıfına eklenen Etiket yapıları, tanıkları oluşturur. Bilginin doğasına benzer şekilde eklemlenen Etiket yapıları, ana Etiket yapısının doğruluğunu destekler.

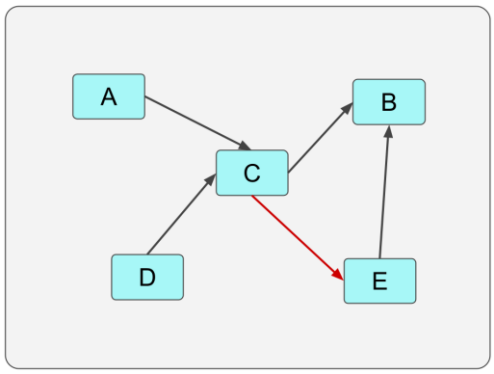
Etiketlerin değişemez ve değişebilir ilişkileri sayesinde, istenen verilerin ve modelin gözlemlenmesi ve izlenmesi sağlanır. Sıralama mekanizması sayesinde, izlenen ve gözlemlenen verilerin önem sırası tayin edilir. Tutarlılık mekanizması sayesinde herhangi bir veri ya da modelde değişiklik gerçekleştiğinde gerekli noktalar bilgilendirilir. Bağlam sayesinde ise iki farklı bölgedeki verinin veya modelin ilişkisel bütünlüğü ve doğruluğu saptanır. Böylece model ve veri ile ilinti herhangi bir noktadaki değişiklik, anında gözlemlenebilir ve izlenebilir. Genel olarak baktığımızda, söz konusu işlevleri gerçekleştiren bir tasarım şablonu bulunmamaktadır. İlerleyen bölümde, geliştirdiğimiz tasarım şablonunun özellikleri açıklanmıştır.

3.2. Özellik: Günlük (Logging) ve İzleme (Tracking)

Günlükler *modelin* değişimini kontrol ederken, izleme ise *verinin* değişimini kontrol etmektedir. Günlükler ve izleme ile kodların durumu hakkında bilgi edinilebilir. Fakat, günlüklerin ve izleme metodolojilerinin bazı eksiklikleri bulunmaktadır. Bunlar, sadece geliştirme zamanında kullanılmaları, sistemden bağımsız olmaları, ilişkisel ve interaktif olmamaları ve bütüncül çıkarsama yapılamamasıdır [21, 22].

Önerdiğimiz gözlemleme ve izleme yönteminde, diğer yöntemlerden farklı olarak izleyiciler, model ve veri ile etkileşim halindedir. Geliştirdiğimiz günlük ve izleme sistemi, rahatlıkla konfigüre edilebileceği için daha sonradan programlanabilir, ilişkiler yaratılabilir ve gerektiğinde sistemde pasif hale getirilebilir. Ayrıca tüm bunlar, yapay zeka ve makine öğrenme algoritmalarına entegre edilerek, otomatik ve adaptif bir yapıya sahip hale getirilebilir. Önerdiğimiz günlük ve izleme mekanizması, yalnızca tasarım anında değil, tüm yazılım mühendisliği süreçlerinde rahatlıkla bulunabilir. Kodun içinde gömülü bulunduğu için kod ile birlikte yaşar, mantıksal süreçlerde karar alma mekanizmalarına katılır, ürün son kullanıcıya teslim edilse dahi yazılımın bir koruyucusu olarak işlev görür.

Şekil 6'da kırmızı ile gösterilen ok, değişebilir ilişkileri ifade ederken, siyah ile gösterilen oklar değişemez ilişkileri ifade etmektedir. Şekil 6'daki A, B, C, D ve E adlı düğümler sınıfları ifade etmektedir. Burada örneğin A düğümünde bir değişiklik meydana geldiğinde B düğümü bunu anında fark edebilecek ve değişikliğin türüne göre pozisyon alabilecektir. Fakat E düğümü bu değişikliği fark edemeyecektir. Çünkü C düğümü ile E düğümü arasındaki ilişki değişebilir olduğu için, özetleme fonksiyonunda hesaplamaya dahil edilmez ve dolayısıyla bu değişiklikten haberdar olamaz. Bu işlem, hem günlük hem de izleme mekanizması için benzer şekilde oluşmaktadır. Burada tek fark, günlükler modeldeki değişime odaklanırken izlemenin verideki değişime odaklanmasıdır.



Şekil 6: Günlük sistemine dair bir örnek

Figure 6: An example for logging system

Verinin, modelin dışında tutulmasından dolayı modelde herhangi bir veri artışı gerçekleşmemektedir. Bu sayede yazılım mühendisliği açısından, veri manipülasyonu, tekrar problemi ve veri bakımı gibi avantajlar elde edilebilmektedir. Aynı zamanda veri ile model birbirinden ayrıldığı için modelleme daha kolay gerçekleşebilmektedir.

Blokzincirindeki veri yapısının arama karmaşıklığı Bağlı Liste yapısı nedeniyle lineer şekilde artar. Diğer yandan Etiket veri yapısı, çizge veri yapısını desteklediği için herhangi bir indeks mekanizması rahatlıkla oluşturulabilir. Buna ek olarak, B Ağaç veya B+ Ağaç gibi bir indeks kullanıldığında, arama karmaşıklığı O(log n)'e kadar düşer.

Değişebilir ve değişemez ilişkiler sayesinde istenilen modelin ve verilerin izlenmesi ve gözlemlenmesi rahatlıkla mümkün hale gelebilir. Sıralama (güven) mekanizması sayesinde ise modelin ve verilerin önemli bölgeleri tespit edilebilir. Tutarlılık mekanizması sayesinde, modelde ve veride meydana gelebilecek herhangi bir değişikliğin anında fark edilebilmesi sağlanır. Son olarak bağlam mekanizması sayesinde ise modeldeki ve verideki ilişkisel bütünlüğe bağlı olarak değişmezlik sağlanabilir.

Sonuç olarak önerdiğimiz tasarım şablonu, çoğu zaman yazılımın ilk aşamasından sonra gereksiz gibi görünen ve genellikle göz ardı edilen, sistemden ayrıştırılmış, interaktif, adaptif ve otomasyon özellikleri bulunmayan, statik günlük tutma ve izleme problemlerine çözüm getirmektedir.

3.3. Özellik: Metrik (Metrics)

Metrikler, belirli bir zaman periyodunda ölçülen veriye dair nümerik gösterimlerdir [3]. Metrikler, bir sistemin kapsamlı biçimde değerlendirilmesi için sistemin genel davranışı hakkında bilgi çıkarmaya yardımcı olur. Metrikler, kullanıcı trafiğine sürekli olarak odaklanmak yerine ihtiyaç duyulduğunda sisteme ait nümerik sonuçlar üretmektedir. Performans bazlı bakıldığında günlükler ve izlere göre avantajlıdır. Sistemdeki trafik arttığında günlükler ve izlerle ilgili işlemlerdeki benzeri orandaki artış yerine metrikler sistem hakkında ilgili ana dair bilgi sunarak trafik yükünden bağımsız bir hizmet sunmaktadır.

Yazılım dünyasında metrikler, belirli değerlerin ölçümünün belirli kriterlere oturtulması ile oluşturulur. Ancak ölçüm sürecinde bazı problemlerle karşılaşılabilmektedir. Bu problemler yaygın olarak yazılımın soyutlama mekanizmasından kaynaklanmaktadır. Soyutlama mekanizması dolayısıyla bir çok veriye ulaşılamaz ve pek çok ölçüm alınamaz. Bu durum, istenilen metriklerin hesaplanamamasına neden olur. Geliştirdiğimiz tasarım şablonuna entegre ettiğimiz ölçüm mekanizması sayesinde, normalde ulaşılamayacak verilerin ulaşılabilir hale gelmesi ve böylece metrik çıktısı üretilebilmesi sağlanır.

Önerdiğimiz yaklaşım kapsamında belirli bir zaman dilimindeki veri değişimi ele alınmaktadır. Örneğin bir veri bölümü normal şartlar altında çok fazla değişime uğramamakla beraber bir anda ani değişimler gözlemleniyorsa bir saldırgan veriye dair işlemler yapmaya çalışıyor olabilir ve bu durumun tespiti için gözlem neticesinde metriklere bağlı bir sinyal üretilmelidir [10]. Benzeri biçimde, çoğunlukla değişime uğrayan bir veri bölümü artık değişime uğramıyorsa yine bir anomali durumu söz konusu olacağından metrik bazlı bir sinyal sistemde elde edilmelidir. Değişime dair önerilebilecek bir diğer metrik, değişebilirlik kapsamıdır. Değişebilirlik kapsamına dair bir metrik, bağlamsal benzerlikler içeren veri bloklarının beraberce veya yakın zamanlarda değiştirilip değiştirilmediğinin tespiti için kullanılır. Özetle, verideki herhangi değişime dayalı bir durumun gözlemlenmesi açısından probleme dayalı gözlemlenebilirlik metrikleri sistemin değerlendirilmesi açısından önem arz etmektedir.

4. Tartışma ve Sonuç

Çalışmamızda, yazılım geliştirmede gözlemlenebilirlik ve izlenebilirliğe yönelik problemlere çözüm sunmak amacıyla, yazılımdan ayrı tutulup, sisteme entegre edilmeyen bir gözlemleme ve izleme anlayışının

yerine, sistemin bir parçası olan ve onunla birlikte yaşayan canlı bir gözlemleme ve izleme tasarım şablonu önerilmiştir. Bu anlayış çerçevesinde, kendi başına yazılımın ve dolayısıyla kendisinin problemlerini çözebilen, yazılımı tüm yaşam süresi boyunca gözleyebilen ve izleyebilen, yapay zeka ve makine öğrenme algoritmalarıyla adaptif hata toleransına sahip olan, belirli bir standardizasyonu ortaya koyan, parametrik esnekliği arttıran, sade ve çift yönlü iletişim sayesinde kullanıcı etkileşimini artıran yeni bir tasarım şablonu geliştirilmiştir. Geliştirdiğimiz tasarım şablonuna, gözlemlenebilirliğin üç temel özelliği olan günlük, izleme ve metrik mekanizmaları entegre edilmiştir.

Önerilen tasarım şablonunun geliştirilmesinde, gözlemlenebilirlik ve izlenebilirlik açısından en güncel ve güvenilir teknoloji olan blokzincirinin özetleme mekanizmasından yararlanılmıştır. Yazılım üretim araçlarına ve paradigmalarına baktığımızda, bu mekanizmaların çizge veri yapısı üzerinde inşa edildiği görülmektedir. Blokzinciri, gözlemlenebilirlik ve izlenebilirlik açısından her ne kadar iyi bir teknoloji olsa da çizge yapısını desteklememektedir. Bu nedenle, çizge yapısını desteklemesi için blok zinciri teknolojisinin veri yapısında değişiklik sağlanmış ve böylece çizge yapısını destekleyen ve aynı zamanda blok zincirini kapsayan yeni bir izlenebilirlik ve gözlemlenebilirlik tasarım şablonu geliştirilmiştir.

İleriki çalışmalarda, önerdiğimiz tasarım şablonu yardımı ile bir bütün olarak örnek uygulamaların gerçeklenmesinin faydalı olacağı düşünülmektedir.

5. Discussion and Conclusion

In our study, a live observation and monitoring design template that is a part of the system and lives with it has been proposed to provide solutions to the problems of observability and traceability in software development, instead of a separate observation and monitoring approach that is kept apart from the software and not integrated into the system. Within this framework, a new design template has been developed that can solve its own problems and thus observe and monitor the software throughout its entire lifecycle, with adaptive error tolerance using artificial intelligence and machine learning algorithms, which establishes a certain standardization, increases parametric flexibility, and enhances user interaction through simple and bidirectional communication. The design template we developed has integrated the three fundamental features of observability, which are logging, tracing, and metric mechanisms.

In the development of our proposed design template, we have used the hashing mechanism of blockchain, which is currently the most up-to-date and reliable technology in terms of observability and monitoring. When we examine software production tools and paradigms, it is observed that these mechanisms are built on graph data structure. Although blockchain is a good technology in terms of observability and traceability, it does not support the graph data structure. Therefore, to support the graph data structure, changes have been made in the data structure of the blockchain technology, and a new observability and monitoring design pattern has been developed that supports the graph data structure and also encompasses the blockchain.

In future work, it is believed that implementing sample applications as a whole using the design template we proposed would be beneficial.

6. Etik kurul onayı ve çıkar çatışması beyanı

Hazırlanan makalede etik kurul izni alınmasına gerek yoktur.

Hazırlanan makalede herhangi bir kişi/kurum ile çıkar çatışması bulunmamaktadır.

Kaynakça

[1] Liu, Y. Y., Slotine, J. J., Barabási, A. L. 2013. Observability of Complex Systems. Proceedings of the National Academy of Sciences, 110(7), 2460-2465.

[2] Sridharan, C. 2018. Distributed Systems Observability: A Guide to Building Robust Systems. O'Reilly Media.

[3] Li, B., Peng, X., Xiang, Q., Wang, H., Xie, T., Sun, J., & Liu, X. 2022. Enjoy Your Observability: An Industrial Survey of Microservice Tracing and Analysis. Empirical Software Engineering, 27(1), 1-28.

[4] Indrasiri, K., Siriwardena, P. 2018. Microservices for the Enterprise. Apress, Berkeley.

[5] Gatev, R. 2021. Observability: Logs, Metrics, and Traces. In Introducing Distributed Application Runtime (Dapr): Simplifying Microservices Applications Development Through Proven and Reusable Patterns and Practices, pp. 233-252, Apress, Berkeley, CA.

[6] Goniwada, S. R. 2022. Observability. In Cloud Native Architecture and Design: A Handbook for Modern

Day Architecture and Design with Enterprise-Grade Examples, 661-676, Apress, Berkeley, CA.

[7] Niedermaier, S., Koetter, F., Freymann, A., Wagner, S. 2019, October. On Observability and Monitoring of Distributed Systems–An Industry Interview Study. In International Conference on Service-Oriented Computing, 36-52, Springer, Cham.

[8] Robinson, W. N. 2006. A Requirements Monitoring Framework for Enterprise Systems. Requirements Engineering, 11(1), 17-41.

[9] Marie-Magdelaine, N. 2021. Observability and Resources Managements in Cloud-Native Environments (Doctoral dissertation, Université de Bordeaux).

[10] Tzanettis, I., Androna, C. M., Zafeiropoulos, A., Fotopoulou, E., & Papavassiliou, S. 2022. Data Fusion of Observability Signals for Assisting Orchestration of Distributed Applications. Sensors, 22(5), 2061.

[11] Robillard, S., & Coullon, H. 2022, March. SMT-Based Planning Synthesis for Distributed System Reconfigurations. In FASE, 268-287.

[12] Bharathi, R., Selvarani, R. 2022. A Machine Learning Approach for Quantifying the Design Error Propagation in Safety Critical Software System. IETE Journal of Research, 68(1), 467-481.

[13] Balke, D. 2022. Consistency and Robustness in an Event-Sourced System. https://es.cs.uni-kl.de/publications/datarsg/Balk22.pdf (accessed Sep. 9, 2022).

[14] Hindle, A. 2010, October. Software Process Recovery: Recovering Process from Artifacts. In 2010 17th Working Conference on Reverse Engineering, 305-308.

[15] Qayum, A., Khan, S. U. R., Akhunzada, A. 2022. FineCodeAnalyzer: Multi-Perspective Source Code Analysis Support for Software Developer Through Fine-Granular Level Interactive Code Visualization. IEEE Access, 10, 20496-20513.

[16] Yusop, N. S. M., Grundy, J., Vasa, R. 2016. Reporting Usability Defects: A Systematic Literature Review. IEEE Transactions on Software Engineering, 43(9), 848-867.

[17] Ladleif, J., Weske, M., & Weber, I. 2019. Modeling and Enforcing Blockchain-based Choreographies. In International Conference on Business Process Management, 69-85.

[18] Ciccio, C. D., Meroni, G., Plebani, P. 2020. Business Process Monitoring on Blockchains: Potentials and Challenges. Enterprise, Business-Process and Information Systems Modeling, 36-51.

[19] Karumuri, S., Solleza, F., Zdonik, S., Tatbul, N. 2021. Towards Observability Data Management at Scale. ACM SIGMOD Record, 49(4), 18-23.

[20] Alhamazani, K., Ranjan, R., Mitra, K., Rabhi, F., Jayaraman, P. P., Khan, S. U., ..., Bhatnagar, V. 2015. An Overview of the Commercial Cloud Monitoring Tools: Research Dimensions, Design Issues, and State-of-the-Art. Computing, 97(4), 357-377.

[21] Cândido, J., Aniche, M., van Deursen, A. 2021. Log-based Software Monitoring: A Systematic Mapping Study. PeerJ Computer Science, 7, e489.

[22] Gujral, H., Lal, S., Li, H. 2021. An Exploratory Semantic Analysis of Logging Questions. Journal of Software: Evolution and Process, 33(7), e2361.