T.C.
TURGUT ÖZAL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK VE BİLGİSAYAR MÜHEDİSLİĞİ
ANABİLİM DALI

TASARIM DESENİ KULLANILARAK GELİŞTİRİLEN YAZILIM İLE KULLANILMADAN GELİŞTİRİLEN YAZILIMIN PERFORMANS ANALİZİ

YÜKSEK LİSANS TEZİ

Hazırlayan
Emre KAZAN

Tez Danışmanı
Yrd. Doç. Dr. Muhammet BAŞTAN

Ankara - 2015

T.C.
TURGUT ÖZAL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK VE BİLGİSAYAR MÜHEDİSLİĞİ
ANABİLİM DALI

TASARIM DESENİ KULLANILARAK GELİŞTİRİLEN YAZILIM İLE KULLANILMADAN GELİŞTİRİLEN YAZILIMIN PERFORMANS ANALİZİ

YÜKSEK LİSANS TEZİ

Hazırlayan
Emre KAZAN

Tez Danışmanı
Yrd. Doç. Dr. Muhammet BAŞTAN

Ankara - 2015

Bilimsel Etik Bildirim Sayfası

Turgut Özal Üniversitesi Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında;

- tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversite veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı beyan ederim.

.../.../2015

İmza

Emre KAZAN

ONAY

Emre KAZAN tarafından hazırlanan "Tasarım Deseni Kullanılarak Geliştirilen Yazılım İle Kullanılmadan Geliştirilen Yazılımın Performans Analizi" başlıklı bu çalışma 16.09.2015 tarihinde yapılan savunma sınavı sonucunda oybirliği ile başarılı bulunarak jürimiz tarafından Elektrik ve Bilgisayar Mühendisliği Anabilim dalında Yüksek Lisans Tezi olarak kabul edilmiştir.

Yrd. Doç. Dr. Muhammet BAŞTAN

Yrd. Doç. Dr. Rıfat ÖZCAN

Yrd. Doç. Dr. Selen PEHLİVAN

ÖNSÖZ

Yüksek Lisans eğitimim boyunca ilminden faydalandığım, insani ve ahlaki değerleri ile de örnek edindiğim, öğrencisi olmaktan onur duyduğum ve ayrıca tecrübelerinden yararlanırken göstermiş olduğu hoşgörü ve sabırdan dolayı değerli hocam, Yrd. Doç. Dr. Mehmet CANTÜRK'e teşekkürlerimi sunarım.

En az Yrd. Doç. Dr. Mehmet CANTÜRK kadar desteklerini esirgemeyen, çalışmalarımızı yönlendiren danışmanım Yrd. Doç. Dr. Muhammet BAŞTAN'a, değerli hocalarım Yrd. Doç. Dr. Selen PEHLİVAN ve Yrd. Doç. Dr. Rıfat ÖZCAN'a şükranlarımı sunarım.

Çalışmalarımız sırasında yardımlarını esirgemeyen pek değerli çalışma arkadaşım, Rifat GÜNEŞ'e teşekkürlerimi sunarım.

Tez yazım aşaması boyunca benimle beraber uykusuz kalan, verdiği moralle çalışmalarımı destekleyen çok değerli eşim, Gamze'ye sonsuz teşekkürler.

ÖZET

KAZAN, Emre. Tasarım Deseni Kullanılarak Geliştirilen Yazılım İle Kullanılmadan Geliştirilen Yazılımın Performans Analizi, Yüksek Lisans Tezi, Ankara, 2015.

Tasarım desenleri (design patterns), yazılım tasarımı sırasında sıkça karşılaşılan, birbirine benzer sorunları çözmek için geliştirilmiş ve işlerliği kanıtlanmış genel çözüm önerileridir. Bu çalışmanın amacı, tasarım desenleri kullanılarak geliştirilen yazılımlar ile kullanılmadan geliştirilen yazılımlar arasındaki performans farkının analiz edilmesidir. Bu amaç doğrultusunda, model-view-controler, model-view-presenter ve proxy desenleri kullanılarak geliştirilen bir yazılım, tasarım desenleri kullanılmadan geliştirilen aynı işlevdeki bir yazılım ile karşılaştırılmıştır. İki yazılım, aynı işlevselliğe sahip üç kullanım senaryosu üzerinde testlere tabi tutulmuş, performans ölçümleri yapılmış ve sonuçlar analiz edilmiştir. Analiz sonuçları, tasarım desenleri kullanılarak geliştirilen yazılımın işlemlere çok daha kısa zamanda cevap verdiğini göstermiştir. Aynı zamanda, tasarım desenleri kullanılarak geliştirilen yazılımın modüler tasarımı ve modüller arasındaki ilişkilerin tanımlanmış olması, yazılımın yeni ihtiyaçları karşılamak amacıyla güncellenmesini kolaylaştırmıştır.

Anahtar Sözcükler

1. Tasarım Deseni
2. Performans Analizi
3. UML Diyagramları
4. Yazılım Geliştirme Süreci

ABSTRACT

KAZAN, Emre. Performance Analysis of a Software Developed with and without Design Patterns: A Case Study, Master Thesis, Ankara, 2015.

Design patterns are functionally-proven general solution proposals developed to solve similar problems frequently encountered during software design process. The goal of this study is to analyze the performance difference between software developed using design patterns and without using design patterns. To this end, a software developed using model-view-controller, model-view-presenter and proxy patterns is compared with a software having the same functionality but developed without using design patterns. The two software were tested on three use-case scenarios; their performances were measured and results were analyzed. The results show that the response time of the software developed using design patterns is much shorter than the one developed without using design patterns. Furthermore, as a result of the modular design and well-defined relations among the modules, it is much easier to update the software to add new functionalities.

Keywords

1. Design Patterns
2. Performance Analysis
3. UML Diagrams
4. Software Development Process

İÇİNDEKİLER

ÜÇÜNCÜ BÖLÜM
İKİ UYGULAMANIN KARŞILAŞTIRILMASI

DÖRDÜNCÜ BÖLÜM
SONUÇ

KISALTMALAR

AOP	:	Aspect Oriented Programming (Bağlam Yönelimli Programlama)
GİB	:	Gelir İdaresi Başkanlığı
GOF	:	Gang of Four
GUI	:	Graphical User Interface (Grafiksel Kullanıcı arayüzü)
GÜVAS	:	Gümrük Veri Ambar Sistemi
HTTP	:	Hyper text transfer protocol (Üstün metin transferi protokolü)
Max	:	Maksimum
MARTE	:	Modeling and Analysis of Real-Time and Embedded Systems
Med	:	Median (Ortanca)
Min	:	Minimum
MVC	:	Model View Controller
MVP	:	Model-View-Presenter
OO	:	Object-Oriented (Nesne Tabanlı)
OOP	:	Object Oriented Programming (Nesne Tabanlı Programlama)
SoaML	:	Service Oriented Architecture Modeling Language
TRT	:	Türkiye Radyo Televizyon Kurumu
URL	:	Uniform Resource Locator (Standart Kaynak Bulucu)
UML	:	Unified Modeling Language (Birleşik Modelleme Dili)

TABLOLAR DİZİNİ

ŞEKİLLER DİZİNİ

GİRİŞ

Tasarım desenleri, yazılımın tasarım aşamasında sıkça karşılaşılan, birbirine benzer sorunları çözmek için geliştirilmiş genel çözüm önerileridir [1]. Tasarım desenleri; daha önce sınanmış, ispatlanmış ve yararı görülmüş yazılım yöntemleri önerdiği için yazılım geliştirme sürecini hızlandırır. Tasarım desenli yazılım geliştirirken karşılaşılacak sorunlar ile ilgili çözümler zaman içinde tasarım desenli yazılım geliştiren yazılımcılar tarafından deneme yanılma yoluyla elde edilmiştir

Tasarım deseni kullanılmadan geliştirilen bir yazılım ortaya çıkan yeni ihtiyaçlardan dolayı işlevselliğini kaybetmiştir. Bu işlevselliğin kaybolmasının en önemli nedenlerinden biri yazılımın teknolojik altyapısının yetersiz olmasıdır. Gelişen teknoloji karşısında eski teknoloji ile tasarım deseni kullanılmadan geliştirilen bir yazılım artık sorunlara çözüm bulamamaktadır. Aynı zamanda yazılımın belirli bir yazılım prensibine göre kodlanmamasından dolayı kod üzerinde yapılacak yapısal değişikliklere cevap verememektedir. Kod üzerinde yapılacak değişiklikler yazılımın farklı noktalarında büyük sorunlara yol açabilmektedir. Bu sorunlar geliştirilen yazılımda yeni entegrasyonların sağlanabilmesi için gerekli olan gereksinimlere cevap verememesine yol açmaktadır.

Bu çalışmadaki amaç tasarım deseni kullanılmadan geliştirilen yazılımın teknolojik altyapısını yenileyerek ve daha sonra ortaya çıkabilecek ihtiyaçları karşılayabilen ve yeni entegrasyonlara cevap verebilen bir yazılım ortaya çıkarmaktır.

Bu amaç doğrultusunda yapılan araştırmalar sonucu tasarım deseni kullanılmadan geliştirilen yazılımlarda ortaya çıkan sorunlara tasarım deseni kullanarak çözüm bulmaya karar verilmiştir. Bu karar neticesinde aynı amacı gerçekleştirmeye çalışan farklı iki yazılım ortaya çıkmıştır.

Tasarım deseni kullanılarak geliştirilen projenin, yazılıma olan etkisini araştırmak ve yazılımlar arasındaki performans farkını görmek için projeler karşılaştırılmıştır. Karşılaştırma sonucunda tasarım deseni kullanılarak geliştirilen

yazılımın yapılan işlemlere çok kısa zamanda cevap vermesi, işlem sayısının fazla olmasına rağmen işlem başına düşen zamanın daha az olduğu görülmüştür. Ayrıca tasarım deseni kullanılarak geliştirilen yazılımın test edilebilirliğinin daha fazla olduğu sonucuna varılmıştır.

Bu tez çalışması dört bölümden oluşmaktadır:

Birinci bölümde; nesne yönelimli programlama, tasarım deseni, tasarım prensipleri ve tasarım desenleri, tasarım desenlerinin yazılıma sağladığı avantajlar, ilişkili çalışmalar ve yeni uygulamanın aşamaları hakkında kısaca bilgi verilmektedir.

İkinci bölümde; bandrol takip sistemi, gereksinim analizi, tasarım deseni seçimi, tasarım aşaması, yazılımın statik mimarisi ve test aşaması hakkında bilgi verilmektedir.

Üçüncü bölümde; geliştirilen yazılımın kullanıcı ara yüzü, performans analizinde kullanılan araçlar ve iki uygulamadan elde edilen istatistikî veriler hakkında bilgi verilmektedir.

Dördüncü bölümde; iki uygulamanın karşılaştırılması sonucu elde edilen veriler analiz edilerek değerlendirme yapılmıştır.

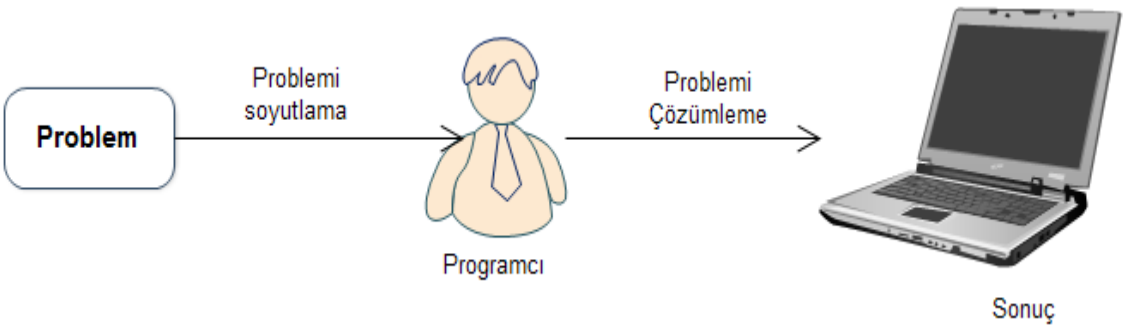
BİRİNCİ BÖLÜM
ÖN BİLGİ

1.1. NESNE YÖNELİMLİ PROGRAMLAMA

Programlar, gerçek hayatta karşılaşılan bir takım problemlerin çözülmesi için bilgisayarların hızlı ve doğru işlem yapabilme yeteneklerinden faydalanmak üzere yazılırlar. Bilgisayarların hız ve kapasiteleri arttıkça geliştirilen programlar da bu hız ve kapasiteden faydalanabilecek şekilde gelişerek değişmektedir. Bu karşılıklı gelişme ve değişme, zaman içerisinde program geliştirme yöntemlerinde de değişikliklere neden olmuş, böylece çeşitli program geliştirme yaklaşımları ortaya çıkmıştır.

Nesne yönelimli programlama (OOP) bir yazılım geliştirme yaklaşımıdır. Bu fikri ilk ortaya atan 1960'larda Alan Kay'dır [2].

İnsanlar günlük hayattaki kavramları, konuşma dili ile anlatırlar. Programcılar, problemle ilgili kavram ve varlıkları, programlama dili ile bilgisayara ifade etmeye çalışırlar. Bunun için tasarım aşamasında, ifade etmeyi sağlayacak modeller oluşturulur. Nesneye yönelik yöntem, bu modellerin oluşturulmasında ve gerektiğinde sistemin güncellenmesinde avantajlar sağlar.



Şekil 1. Nesne yönelimli programlamanın aşamaları

Nesne yönelimli programlamada üç temel yapı üzerine durulur. Bunlar veri soyutlama (encapsulation), kalıtım (inheritance) ve çok biçimlilik (polymorphism) şeklindedir [3].

Nesne yönelimli programlama, yazılım mühendisliği dünyasında artan ihtiyaçlar ve karmaşıklaşan problemler karşısında bazı gereksinimleri karşılamakta yetersiz kalmaya başlamıştır. Nesne yönelimli programlamanın eksik kaldığı durumlarda yardımcı yaklaşımlar olarak tasarım desenleri ortaya çıkmıştır.

1.2. TASARIM DESENİ

1.2.1. Tasarım Deseni Kavramı

Tasarım desenleri, yazılım yaparken sıkça karşılaşılan, birbirine benzer sorunları çözmek için geliştirilmiş genel çözüm önerileridir. Tasarım desenleri; çalışan kod veya algoritma olmayıp sınıfların iyi bir şekilde tasarlanmasını belirten yöntemlerdir. Bu yöntemler uygulamaların esnekliğini ve kalitesini arttırmakla kalmaz aynı zamanda geliştirme sürelerini de azaltır. Bunun yanında uygulamalardaki kodların anlaşılabilirliliği de kolaylaşmış olmaktadır [4]. Tasarım desenleri yazılım geliştirirken karşılaşılan sorunlara esnek, genişletilebilir, yeniden kullanılabilen çözümler getiren ve çeşitli durumlarda sorunların nasıl çözüleceğini gösteren yöntemlerdir [5]. Tasarım desenleri daha çok nesneye dayalı programlama da sınıf ve nesnelerin birbirleriyle olan ilişkilerini açıklamaktadır. Tasarım deseni kullanılarak yazılım geliştirirken karşılaşılacak sorunlar ile ilgili çözümler zaman içinde yazılımcılar tarafından deneme yanılma yoluyla elde edilmiştir.

Yeni uygulamalar; yazılımcılara, uygulamalarda karşılaştıkları problemlere esnek çözümler geliştirmelerine olanak sağlar. Tasarım deseniyle geliştirilmiş uygulamaların daha sonra uygulamanın belli ihtiyaçları karşılayamamasından dolayı yeniden geliştirilmesi veya düzenlenmesi gerekebilir. Bu durumlarda tasarım desenlerini bilen yazılımcıların, yazılımlara gerekli düzenlemeleri yapmaları daha az maliyetli olmaktadır [6]. Tasarım desenine sahip olan uygulamaların da kod

okunabilirliği yüksek olmaktadır. Bu yapıyı barındıran uygulamalar yazılımcılar arasında ortak bir dil oluşturmaya da katkı sağlamışlardır.

Nesne yönelimli yazılım geliştirenler en iyi uygulama örneklerini, tasarım desenleri kullanarak elde etmişlerdir [7].

1.2.2. Tasarım Prensipleri ve Tasarım Desenleri

İyi tasarımın amacı nedir? Neden iyi bir tasarıma sahip olmak istenir? İyi tasarımın amacı ileride karşılaşılacak olası değişiklikleri en kolay şekilde ele alınmasını sağlamaktır. Yazılım sistemlerinde gereksinimler değişmektedir. Gereksinimlerin değişmesinin pek çok değişik nedeni vardır. Projenin başında tam manası ile tespit edilememiş olabilirler. Eksik, yanlış veya yanıltıcı ifade edilebilirler. Sistemin geliştirilmesi süresince ve sistem kullanıma alındıktan sonra da sürekli olarak değişirler [8]. Bu değişiklikler ekip üyelerinin ve kullanıcıların yeni olasılıkları görmelerinden, geliştiricilerin sistemi kavrayışlarının değişmesinden, yazılım sisteminin geliştirildiği ortamın veya iş süreçlerinin değişmesinden kaynaklanabilir. Ancak, gereksinimlerin değişmesinden şikâyet etmek anlamsızdır. Yapılması gereken gereksinimlerdeki değişmelerin önüne geçmekten çok, değişiklikleri daha kolay biçimde ele almamızı sağlayacak bir sistem tasarımına sahip olmaya çalışmaktır. Bu noktada da aklımıza şöyle bir soru gelebilir. Peki, iyi tasarımı, kötü tasarımdan nasıl ayırt edebiliriz? Neyin iyi bir tasarım kararı olduğunu, hangi kararın ise ileride karşımıza çıkabilecek muhtemel değişiklikleri ele almamızı kolaylaştıracağını veya zorlaştıracağını önceden nasıl anlayabiliriz?

1.2.3. Tasarım Desenlerinin Yazılıma Sağladığı Avantajlar

Tasarım desenleri sayesinde yazılımdaki kodların tekrar kullanılabilir olması, sistemin işleyişini çok fazla etkilemeden kolayca sisteme ekleme çıkarma yapılarak yazılıma büyük bir esneklik kazandırmaktadır.

Yazılımcılar kapsamı çok büyük olmayan yazılımları kolaylıkla geliştirebilirler. Bu uygulamalardaki veriler ve kullanıcı sayıları çok fazla

olmadığından yazılımların arka planda nasıl bir kodlaması olduğu çok önemsenmez. Fakat büyük yazılımlarda kod yazmak küçük yazılımlara göre kolay olmaz. Büyük yazılımlarda, yazılımın çalışmasından çok aşağıdaki konular önem sağlamaktadır:

- Uygulamanın çalışma performansı,
- Yazılımın aynı anda çok fazla kullanıcı tarafından kullanıldığındaki performansı,
- Yazılımın güvenliği,
- Yazılımda ileride yapılacak değişikliklerin kodlamada karmaşaya yol açmaması ve yazılımın esnek bir yapıya sahip olmasıdır.

Yukarıdaki özelliklere sahip yazılımların öncelikle yazılımın performansı arttıracak ve kod karmaşıklığını engelleyecek metotlara ihtiyaç duyulacaktır [9].

Tüm yazılım geliştiriciler bir sorun için aynı yöntemi (standart tasarım desenini) kullanırlarsa ve bunu alışkanlık haline getirirlerse, tasarım şablonlarının kullanıldığı herhangi bir projeyi ele aldıklarında kodları kolaylıkla çözebilmektedirler [10]. Bu da genel anlamda geliştirilen yazılımı hem belirli standartlara uygun hale getirir hem performansını artırır hem de projeyi gereksiz kodlamalardan kurtarır.

Yazılım geliştiriciler tasarım desenlerini yeni gereksinimlerin doğabileceği ihtimalini düşünerek sistemi genişleyebilir olarak tasarlamak için kullanılırlar. Sistem değişimlere ve genişlemeye uygun tasarlanmadıysa, ilerde ortaya çıkan gereksinimler sistemin baştan yazılmasını gerektirebilir. Bu durumda yeni yazılımın test sürecinin de tekrar yapılması gerekir. Bu durum yazılımcılara zaman maliyeti olarak döneceğinden sistemi en başta doğru tasarlamak çok önemlidir. Tasarım desenleri, sistemlerin daha rahat, hızlı ve sağlam bir değişebilirlik özelliğine sahip olmasını sağlarlar [11].

Tasarım desenleri, probleme ve probleme özgü tasarıma yüksek seviyede bir bakış açısı ile yaklaşılmasını sağlar.

Bu sayede probleme özgü detaylarla, zamanı geldiğinde ilgilenilir, tasarımın başlangıcında detaylarla zaman kaybedilmesi engellenmiş olur. Bu sayede sistem daha sağlıklı bir biçimde gelişir, detayların sistemde yaratabileceği bağımlılıklar mümkün olduğunca azaltılır [12].

Nesneye dayalı programlamada sınıfların kendi içinde tutarlı, diğer sınıflara olabildiğince az bağımlı olmaları beklenir. Yazılımdaki kod bloklarının tekrar kullanılabilir olmaları yani kısaca esnek olmaları beklenir. Yeni ihtiyaçların, yazılımın diğer kısımlarını en az biçimde etkileyerek yazılıma kolayca dâhil edilmeleri beklenir [13]. Tasarım desenleri, nesneye dayalı programlamanın prensiplerini (abstraction, encapsulation, modularity, hierarchy, polymorphism) doğru bir şekilde uygulanmasını sağlar.

Tasarım şablonları aşağıda yer alan ortak özelliklere sahiptir [14]:

- Edinilen tecrübeler sonunda ortaya çıkmışlardır.
- Tekrar kullanılabilir kalıplardır.
- Ortak kullanılarak daha büyük problemlerin çözülmesine katkı sağlarlar.
- Devamlı geliştirilerek, genel bir çözüm olmaları için çaba sarf edilir.

Kaliteli bir tasarımın önemi çoğunlukla yazılım sistemlerini işletime alınıp bir süre kullanıldıktan sonra ortaya çıkmaktadır. Sistemlerde bakım ve idame süresince karşılaşılan hataların düzeltilmesi, yeni gereksinimlerin eklenmesi veya mevcutlarda yapılacak değişikliklerin ele alınması; sistemlerin esnek bir tasarıma ve mimariye sahip olduklarında oldukça kolay olmaktadır [15]. Yeniden kullanılabilir sistemlerin ve modülerin ortaya çıkartılması uzun vadede yeni sistemlerin geliştirme maliyetlerine de olumlu etkilerde bulunacaktır. Kaliteli bir tasarıma götüren yol ve yordamları; yazılım geliştiricilerin çok iyi etüt ederek günlük geliştirme pratikleri arasına katmaları, tasarım ve mimari çalışmalarını belirli kural ve prensipler etrafında yürütmeleri uzun ömürlü yazılım sistemleri ortaya koymalarını sağlayacaktır [16].

Tasarım desenlerinin temelindeki fikir yazılım sistemlerinin kalitesinin nesnel biçimde değerlendirilip değerlendirilemeyeceğidir [17]. Tasarım desenleri kaliteli olarak nitelendirilen, zaman içerisinde ortaya çıkan değişiklik taleplerini esnek biçimde karşılayabilen pek çok yazılım sisteminin incelenmesi sonucu ortaya çıkmış, rafine edilmiş, yeniden kullanılabilecek şekilde ifade edilmiş bilgi yumağıdır. Bu tür yazılım sistemlerinde olup da, kötü veya başarısız olarak nitelendirilmiş tasarımlarda olmayan, ya da kötü tasarımlı sistemlerde olup da, bu tür başarılı sistemlerin uzak durduğu noktalara odaklanılarak tespit edilmişlerdir. Tasarım desenleri daha önceki çözümlerin, tasarımların ve hali hazırda mevcut bilgi birikiminin yeniden kullanılmasını sağlamıştır. Bu örüntüler zaman içerisinde evrilmiş ve olgunlaşmış çözümlerdir. Probleme sıfırdan başlamayı ve daha önceki ekiplerin karşılaştığı problemlerle tekrardan uğraşmamayı sağlar. Diğer ekiplerin deneyimlerinden faydalanmak mümkün hale gelir. Ekip içinde ortak bir terminolojinin oluşmasını sağlar, ortak bir bakış açısı getirir. Tasarım ve nesne modelleme sürecine bir üst perspektiften bakmayı sağlar. Bu sayede daha ilk dakikadan itibaren gereksiz detaylar ve ayrıntılar içerisinde boğulmanın önüne geçilebilir. Belirli bir süre ve farklı yerlerde kullanıldıklarından olası yeni gereksinimleri ele alırken üzerlerinde değişiklik yapmak daha kolay ve hızlıdır.

Tasarım desenleri, yazılım dünyasındaki problemler için kalıcı ve tekrar edici çözümler sunmaktadır. Tasarım desenleri, nesneler arasındaki ilişkiyi ve bunların nasıl yönetileceği ile ilgili konuları tanımlar [18].

1.3. İLGİLİ ÇALIŞMALAR

Tez kapsamında yapılan çalışmalara faydalı olması açısından referans alınacak çalışmalar bulunmaktadır. Bu çalışmalar uygulama geliştirmede tasarım desenleri hakkında çeşitli bilgiler ve tasarım deseninin yazılıma sağladığı faydalara ilişkin öneriler içermektedir.

Al-Ahmad [19] çalışmasında, endüstride ve akademide tasarım desenlerinin öğretimi sırasında keşfedilen basit yapısı nedeniyle de her uygulamada kolaylıkla kullanılabilecek tasarım desenlerini tanıtmaktadır. Sonuç olarak da bu tasarım

desenlerini otomatize ederek daha ayrıntılı tasarım modelleri oluşturulmasına yardımcı olan bir araç önermektedir.

Nija ve Ronald [20] çalışmalarında ters mühendislikte tasarım deseni kullanarak yüksek performanslı yazılım geliştirmeyi amaçlamışlardır. Yüksek performanslı yazılımlarda kod tasarımlarının anlaşılması için ters mühendiste kullanılan araçların önemini vurgulamışlar. Strateji ve durum desenleri gibi bazı tasarım desenlerinin yapısal olarak aynı, davranış olarak farklı olduğunu belirtmişlerdir. Bu tasarım desenlerindeki hata oranlarını düşürmek için dinamik analizler kullanmışlardır. Nija ve Ronald göre yüksek performans yazılımlar tasarım deseni kullanılarak yazılan yazılımlardır.

Mani, Petriu ve Woodside [21] çalışmalarında, tasarım desenlerini SOA yazılım modelinin mimarisindeki hatalar, tasarımındaki eksiklikler ve uygulamasında ortaya çıkan sorunlarının çözülmesi için önermektedir. Tasarımcı yazılım modellerini ve desenlerini, SoaML ve MARTE profilleri ile genişlettiği UML ile tanımlamıştır. Tasarımcı aynı zamanda bir desenin uygulamasını kullanıcılara göre özelleştirerek tanımlamıştır. Böylelikle desenlerin çeşitli çalışmalar üzerindeki etkilerini kademeli olarak incelemiş ve ortaya koymuştur.

Schmidt [22], çalışmasında sistem geliştirmede ortaya çıkan temel sorunları engellemek ve yazılım kalitesini artırmak için tasarım deseni kullanmıştır. Bu temel sorunlar, geliştiriciler arasındaki mimarideki bilgi paylaşımı, yeni mimari sitillerin ve tasarım paradigmalarının kullanımı ve genellikle kişisel birikimlere dayanan hatalara düşülmesi gibi konuları içermektedir. Bu makale de, tasarım deseni stratejisiyle nesneye dayalı dağınık sistem yazılımların ara yüz iletişimi ve yeniden kullanımı sağlamayı amaçlamıştır. Schmidt makalesinde sezgilerle elde edilen yanıltıcı bilgiler yerine birkaç yıllık tecrübeyle elde edilen önerileri koymayı amaçlamaktadır.

1.4. ÇALIŞMANIN AŞAMALARI

Bu tez çalışmasında tasarım deseni kullanılmadan geliştirilen bir yazılım ile model-view-controler (MVC), model-view-presenter (MVP) ve proxy desenlerini

(pattern) kullanarak geliştirilen bir yazılım arasındaki performans farkının ortaya koyulması amaçlanmıştır. Bu genel amaca ulaşmak adına izlenecek adımlar şu şekildedir:

Tasarım deseni kullanılmadan geliştirilen yazılım nesne tabanlı yaklaşımına göre tasarlanmıştır. Tasarım deseni kullanılmadan geliştirilen yazılım java programlama dili kullanılarak, nesne tabanlı yazılım prensiplerine göre yazılmıştır.

Nesne tabanlı tasarım için gereksinim raporu oluşturulmuştur. Bu yazılımda üç tane kullanım senaryosu belirlenmiştir. Öncelikle kullanım senaryolarıyla ilgili yapılması gerekenler belirlenmiştir. Daha sonra belirlenen kullanım senaryoları için Birleşik Modelleme Dili (Unified Modelling Language) diyagramları çizilmiştir. Son olarak da belirlenen diyagramlara göre yazılım geliştirilmiştir

Tasarım deseni kullanılarak geliştirilen yazılım için tasarım desenleri seçilmiştir. Tasarım desenleri belirlenirken, Gang of Four [16] ve Martin Fowler tarafından geliştirilen tasarım desenleri, tasarım desenleri ile ilgili makale ve yazıları incelenmiştir. Uygulama için üç tane tasarım deseni belirlenmiştir. Bu tasarım desenleri Model-View-Controler, Model-View-Presenter ve Proxy şeklindedir.

Tasarım deseni kullanılarak geliştirilen yazılım için seçilen tasarım desenlerin yazılıma uygulanmıştır. Belirlenen desenler kullanım senaryoları da göz önüne alınarak projenin kodlaması yapılmıştır.

İki uygulamanın sayısal verilere göre karşılaştırılmıştır. Uygulamaların performans ölçümleri yapılarak bir takım sayısal veriler elde edilmiştir. Söz konusu sayısal verilerin elde edilmesi için aşağıdaki araçlardan yararlanılmıştır.

- Jmeter
- Speedy Framework

İKİNCİ BÖLÜM
YAZILIM GELİŞTİRME AŞAMALARI

Bu bölümde yazılım geliştirme aşamaları hakkında bilgi verilmiştir. Yazılım geliştirme aşamalarını, tasarım deseni kullanılarak geliştirilen TRT bandrol takip sistemi üzerinde uygulanmıştır.

2.1. BANDROL TAKİP SİSTEMİ

2.1.1. Bandrol Takip Sisteminin Tanımı

Bandrol takip sistemi, firmaların ithal ettikleri ya da imal ettikleri televizyon, radyo gibi ürünler için kullanmak zorunda oldukları TRT bandrollerinin takibini yapmak için kullanılan bir uygulamadır.

Yurt dışından ithal edilen ya da yurt içinde imal edilen televizyon ve radyo gibi yayın alan cihazların satışından elde edilen karın belirli bir oranı kanun gereği (3093 sayılı TRT Gelirleri Kanunu) TRT'ye verilmek zorundadır. TRT, bu yolla elde ettiği geliri bandrol satışı üzerinden takip etmektedir.

2.1.2. Bandrol Takip Sisteminin Amacı

Bandrol Takip Sistemi'nin amacı; bandrollerin basımından firmalara satışı, satış sonrası tahakkuk ve iadesine kadar olan süreci takip etmektir.

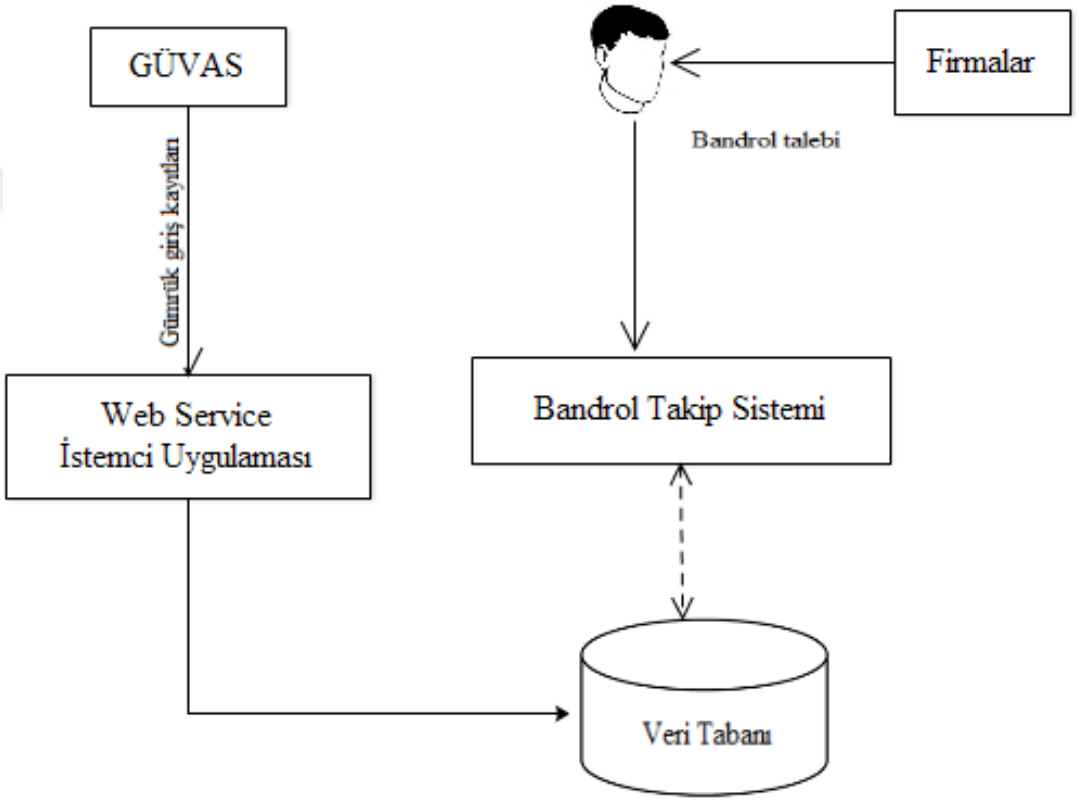
Bu uygulama üzerinden; bandrol talepleri girilebilmekte, yapılan bandrol talepleri onay sürecinden geçirilmekte, talepler neticesinde verilen bandrollerin firmalara teslimleri takip edilebilmekte, bandrollerin ücretlerinin tahakkuk ettirilmesi izlenebilmekte kullanılmayan bandrollerin iade süreçleri yürütülebilmektedir.

2.1.3. Mevcut Uygulamanın Entegrasyon Noktaları

Mevcut bandrol takip sisteminin iş akış diyagramı Şekil 2‘de gösterilmiştir. Gümrük giriş kayıtları Gümrük Sistemi (GÜVAS) üzerinden web servis ile

sorgulanmaktadır. Bunun için geliştirilen ayrı bir istemci uygulaması her ay manüel olarak çalıştırılmaktadır.

Bandrol takibi yapılan firmalar uygulamayı doğrudan kullanmamaktadır. Bandrol taleplerini excel dosyaları şeklinde kurum içi kullanıcılara göndermektedir. Bu kullanıcılar ise gönderilen dosyaları sisteme yüklemektedir.



Şekil 2. Mevcut uygulamanın entegrasyon noktaları

2.2. BANDROL TAKİP SİSTEMİNİN GELİŞTİRME SÜRECİ

İkinci uygulamayı geliştirmeye karar verildiği zaman, aynı işlevselliğe sahip ve hali hazırda kullanılan ilk projeden farkı tasarım deseninin kullanılmasıdır.

Sistem geliştirilirken iki uygulamada ortak olan gereksinim analiz dokümanı oluşturulmuştur. Gereksinim analiz dokümanında belirlenen üç tane kullanım senaryosunun aşamaları belirlenmiştir. Tasarım deseni seçilmiştir. Tasarım desenleri seçildikten sonra Bandrol Takip Sisteminin UML diyagramları çizilmiştir. Çizilen

UML diyagramlarına göre uygulama yazılmıştır. Yeni geliştirilen uygulamayla hali hazırda kullanılan Bandrol Takip Sistemi karşılaştırılmıştır. Bu aşamaların detayları aşağıda belirtildiği gibidir.

2.2.1. Uygulamanın Gereksinim Analizinin Yapılması

Uygulamanın başarıya ulaşması için gereksinimlerin ortaya çıkarılması ve gereksinimlerin öncelikli durumları analiz edilmiştir. Burada gereksinim analiz dokümanına göre üç tane kullanım senaryosu oluşturulmuştur.

2.2.1.1. Bandrol Talep İşlemleri

Kullanıcı, bandrol talebi girmek istediği firmayı kendisine zimmetli firmalar içerisinden seçmelidir. Talep edilen bandrollerin firma yetkilisi tarafından teslim alınacağı il müdürlüğü talep esnasında seçilmelidir. Bandrolleri teslim alacak kişi talep esnasında belirli ise sisteme girilebilmelidir. Firmaların bandrol talepleri, yapısı tasarım aşamasında belirlenecek olan bir dosya ile alınabilmeli ve sisteme yüklenebilmelidir. Kullanıcı bandrol alınacak her bir talep kalemi için bandrol tipini belirlemelidir.

Beyan edilen bandrol matrahı kullanıcıya zimmetli olan firmaların önceki aynı marka ve modele sahip onaylanmış talepleri ile karşılaştırılabilmelidir. Aynı modele sahip kayıt yoksa markaya, o da yoksa cihaz tipine bakılmalıdır. Bulunan kayıtlardan son bir kaç tanesi kullanıcıya görüntülenmelidir. Kullanıcı, sadece kendisine zimmetli olan firmaların geçmişe yönelik taleplerini görebilmelidir.

Bandrol talep kalemleri gümrük giriş beyanname kalemleri ile beyanname numarası ve kalem numarası üzerinden eşleştirmektedir. Firmanın beyan ettiği bilgiler ile GÜVAS'tan elde edilen ilgili gümrük giriş beyannamesi aşağıdaki alanlar bazında kontrol edilmektedir. Gümrük Giriş Beyannamesi'ndeki KDV matrahı ile firmanın beyan ettiği KDV matrahı farklı ise sistem kullanıcıyı uyarmaktadır. Gümrük Giriş Beyannamesi'ndeki ÖTV tutarı ile firmanın beyan ettiği ÖTV tutarı farklı ise sistem kullanıcıyı uyarmaktadır. Talep edilen bandrol

adedi; gümrük giriş beyannamesi ve beyanname kaleminde ithal edilen cihaz sayısından fazla ise sistem kullanıcıyı uyarmaktadır.

Bandrol ücreti, bandrol matrahı ve seçilen bandrol tipine karşılık gelen bandrol oranı kullanılarak hesaplanmaktadır. Bandrol matrahı, ÖTV hariç KDV matrahıdır. İthalat talepleri için bandrol matrahı hesaplanırken gümrük giriş beyannamesinde yer alan (aynı zamanda firma tarafından da beyan edilmiş olan) KDV Matrahı ve ÖTV Tutarı kullanılmaktadır. İmalat talepleri için bandrol matrahı olarak firma tarafından beyan edilen ÖTV hariç KDV matrahı kullanılmaktadır. Bu matrah firma tarafından tahmini satış fiyatıdır.

Bandrol talepleri ya talep bazında tamamen peşin, ya da talep bazında tamamen teminatlı alınmaktadır. Aynı talep içerisinde hem peşin, hem de teminatlı alım yapılmamaktadır. Firmanın nakit hesabı bandrol talep tutarını karşılayamıyorsa ya da yeteri kadar teminatı yoksa bandrol talebi onaylanamamaktadır.

Talepler onaylanmadan önce onay yazısı bastırılabilmektedir. Bandrol talepleri yetki verilen kullanıcılar tarafından onaylanmaktadır. Onay yazısının tarih ve sayısı onay esnasında belirtilmektedir. Bandrol talebi onaylanıncaya kadar güncellenebilmeli ya da silinebilmelidir. Onaydan sonra güncelleme veya silme yapılamamaktadır.

2.2.1.2. Teslimat İşlemleri

Onaylanmış yani teslimata hazır bandrol talepleri teslimatın yapılacağı il müdürlüklerinde görevli kullanıcılar tarafından görüntülenebilmektedir. İl müdürlüğü kullanıcıları hangi tipteki bandrollerden kaç adet teslim edileceğini görüntüleyebilmektedir. İl müdürlüğü kullanıcıları, teslim ettikleri bandrollerin seri no aralıklarını sisteme girmektedir. Teslimatı yapan kullanıcı, teslim tarihini girmelidir. Girilen bu tarihin, onay tarihinden önce olmamalıdır. Bandrolsüz satışların tespitinde kullanıcı tarafından girilen bu tarih dikkate alınmalıdır. Şayet talep anında bandrolleri teslim alacak kişilerin isimleri bildirilmişse bu isimler il müdürlüğü kullanıcıları tarafından görüntülenebilmelidir. Bandrolleri teslim alan

kişinin adı ve soyadı sisteme girilmelidir. İl müdürlüğü kullanıcısı, teslimatı onaylamadığı müddetçe, girdiği verileri (bandrol seri no aralığı ve teslim tarihi) güncelleyebilmelidir. Bandrol teslimatı yapıldığında il müdürlüğü kullanıcısı teslimatı onaylamalı ve teslimat tutanağı sistemden bastırılabilmelidir. Teslimat onaylandıktan sonra, il müdürlüğü kullanıcısı teslimat hakkında herhangi bir veriyi güncelleyememelidir. Teslimat tutanağı alınır. Teslimat görevlilerinin onay gelmeden bandrol vermiş olma ya da ellerindeki bandrolleri el altından piyasaya sürme durumlarının tespiti için teslimat görevlilerinin girdikleri seri no aralığının başındaki bandrol, stoktaki henüz teslim edilmemiş en küçük seri numarasına sahip bandrol değilse bu durum merkeze e-posta yoluyla bildirilmelidir.

2.2.1.3. Ödeme İşlemleri

Firmaların TRT hesabına yatırdıkları paralara ilişkin getirdikleri dekontlar takip edilebilmelidir. Dekontların ödeme teyidinin yapılabilmesi için dekont giriş anında bankanın web sitesine bir dış bağlantı linki verilmelidir. Bu dekontlar bandrol talebi, ceza ve faiz borcu karşılığında kullanılabilmelidir. Dekontlar; dekont numarası ve banka bazında tekildir. Aynı dekont numarasına sahip birden çok dekont girişine izin verilmemelidir.

Belirlenen üç tane kullanım senaryosunun gereksinim analiz dokümanı oluşturulmuştur. Gereksinim analiz dokümanını oluşturduktan sonra uygulamada kullanacağımız tasarım desenleri seçimi yapılmıştır.

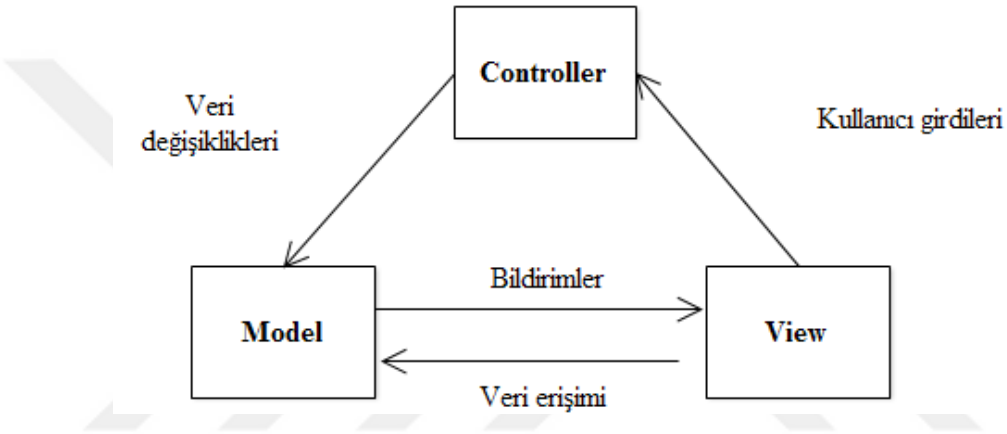
2.2.2. Tasarım Deseni Seçimi

Tasarım desenleri belirlenirken, öncelikle GOF ve Martin Fowler tarafından geliştirilen tasarım deseni örnekleri incelenmiştir. Ayrıca tasarım desenleri ile ilgili makale ve yazıları incelenmiştir. Bu araştırma sonucunda Model-View-Controler, Model-View-Presenter ve Proxy tasarım desenlerinin kullanılacağına karar verilmiştir. Model-View-Controler'ın seçilme nedeni yazılımın farklı amaçlara hizmet eden bölümlerinin birbirine girmesini engellemesi, Model-View-Presenter'ın

seçilme nedeni view katmanı ile servis katmanı arasındaki iletişimi sağlaması, Proxy'nin seçilme nedeni ise işlem yönetimini (Transaction) sağlamasıdır.

2.2.2.1. MVC (Model View Controller)

Model View Controller 1970'lesonunda Norveçli bilim adamı Trygve Reenskaug'un Amerika'daki Xerox laboratuvarlarını ziyareti sırasında ortaya konulmuş mimarisel bir örüntüdür [23]. Kısaca MVC olarak adlandırılır.

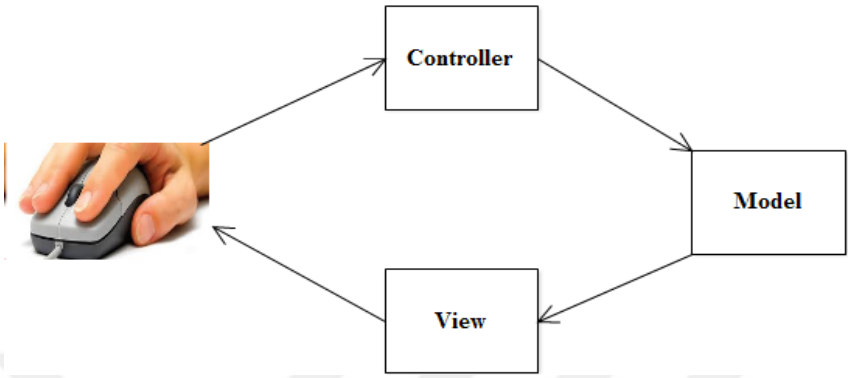


Şekil 3. MVC'nin döngüsel yapısı

MVC'nin çalışma mantığı Şekil 3 ve Şekil 4'de gösterilmiştir. Model, view tarafından görüntülenen veriyi ifade eder. Örneğin, bir checkbox bileşenin on/off durum bilgisi, ya da bir textfield bileşeninin metin verisi gibi. View ihtiyaç duyduğu veriye model üzerinden erişir ve bu veriyi kullanarak kullanıcı ara yüzü ekranda görüntüleme işlemini gerçekleştirir. Controller ise kullanıcı girişinden eventler ile model üzerinde değişikliğe gidilmesini sağlar. Model'deki değişiklik de modifikasyonlar vasıtası ile view tarafından algılanarak ekrana yansıtılır. Güncel pek çok dokümanda MVC'nin amacı olarak "iş mantığının kullanıcı ara yüzü kodundan ayrılması" olarak anlatılır. Bu sayede view katmanında herhangi bir değişiklik yapılmak istenirse, bunu iş mantığında herhangi bir probleme veya değişikliğe yol açmadan kolaylıkla yapabileceği vurgulanmıştır.

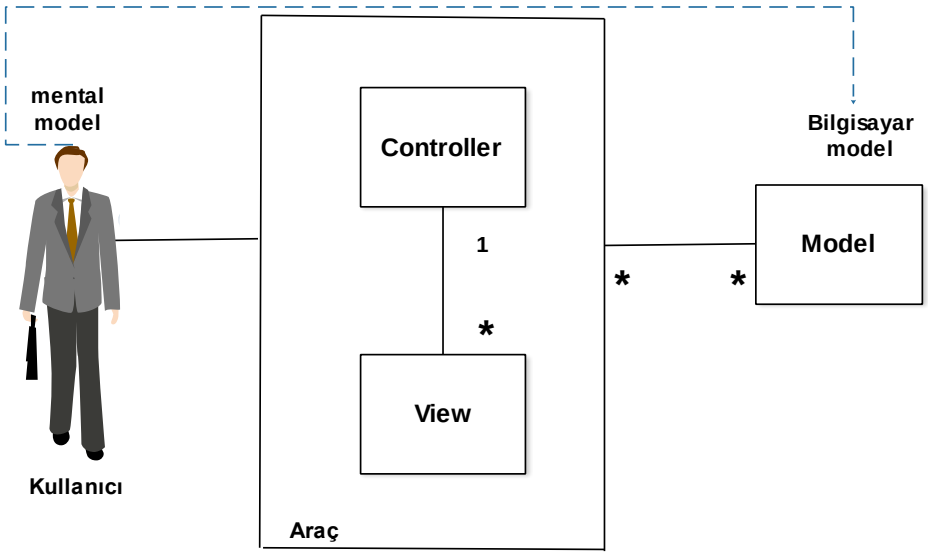
MVC'nin mucidi Reenskaug, MVC'yi anlattığı makalesinde asıl amacının Şekil 5'te görüldüğü üzere kullanıcıların zihinlerindeki mental model ile bilgisayar

sistemlerindeki sayısal model arasındaki boşluğu dolduran genel bir çözüm oluşturmak olduğunu vurgulamıştır.



Şekil 4. MVC'nin döngüsel yapısı

Bu çözüm ile etki alanı (domain) verisi, başka bir deyişle model doğrudan kullanıcı tarafından erişilebilir, incelenebilir ve güncellenebilir hale gelecektir [23].



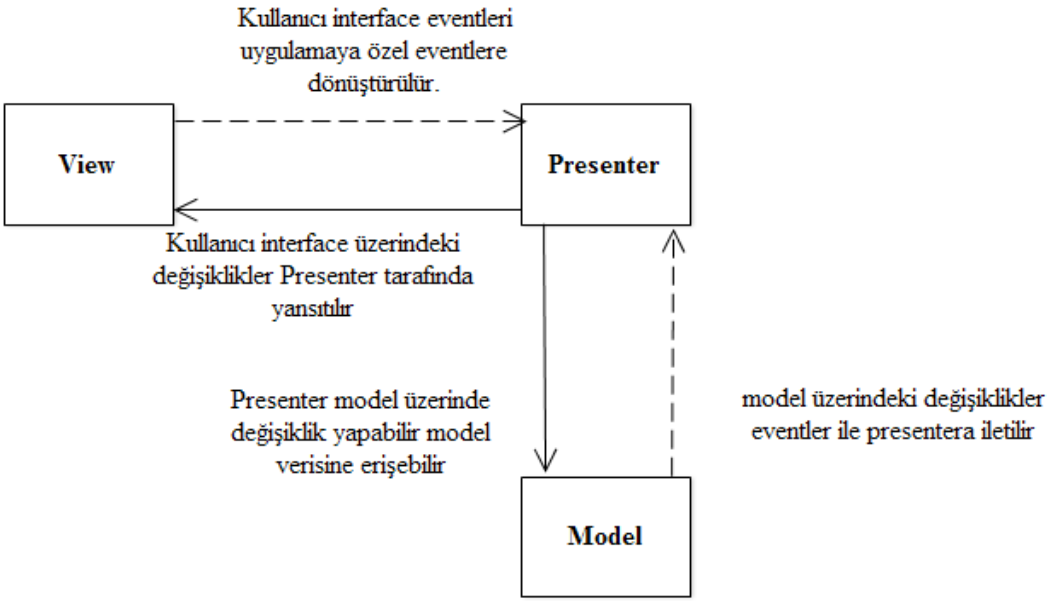
Şekil 5. MVC'nin alternatif döngüsel yapısı

Uygulamayı modüler bir yapıya büründürmek ve farklı görevleri farklı katmanlara ayrıştırmak MVC için ilk hedef olmamıştır. Model, controller ve view bölümleri çözüm içerisinde vardır, ancak bunlar yukarıda bahsettiğimiz asıl amaca yönelik olarak şekillenen kısımlardır. Orijinal MVC makalesinde "Seperation of Concern" bir amaç değil sonuçtur [24].

Uygulamanın modüler biçimde geliştirilebilmesi ve katmanların diğer katmanlardan bağımsız biçimde görevlerini yerine getirebilmesi amacı ile MVC deseni üzerinde bir uyarlamaya gidilmesi söz konusudur. Bunun temel nedenlerinden birisi de view içindeki uygulama ile ilgili kodun ve iş mantığının genellikle iç içe girmeleridir. İki katmanı birbirlerinden daha net biçimde ayıracak bir yapıya ihtiyaç vardır [25].

2.2.2.2. MVP (Model-View-Presenter)

MVP'nin temeli Şekil 6'da gösterildiği gibi view katmanı içerisinde yer alan GUI kodunu katmanı içerisinden çıkararak ayrı bir presenter sınıfına taşımaktır. Böylece uygulama ile ilgili kod GUI oluşturulması ve ekranda görüntülemesi işlemlerinden bağımsız biçimde çalıştırılarak test edilebilmektedir.



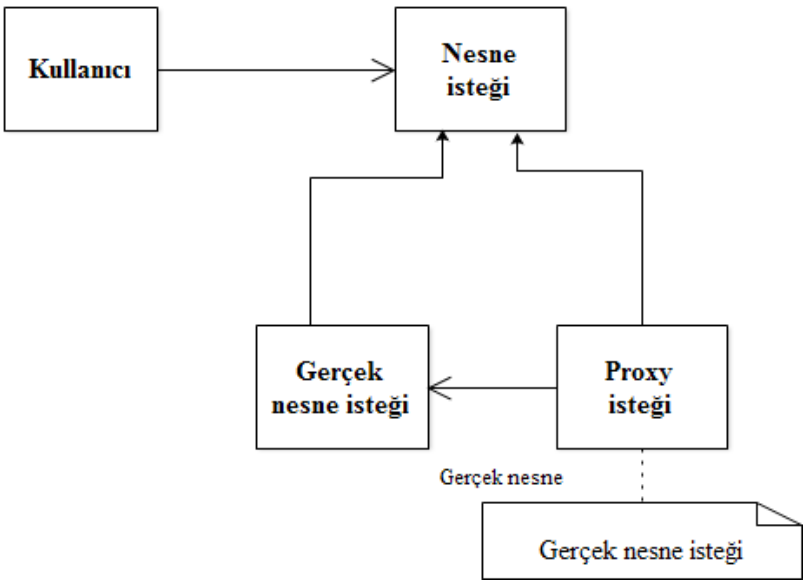
Şekil 6. MVP'nin döngüsel yapısı

Presenter view tarafından kullanıcı girişini elde ederek ilgili iş mantığını yürütmesi için işi model katmanına havale eder. Model tarafında işletilen davranış sonucu model üzerinde bir takım durum değişiklikleri olacaktır. Bu durum değişiklikleri yine presenter'a event'ler vasıtası ile haberdar edilmektedir. Presenter'da bu durum değişikliklerini uygun metotları kullanarak view tarafına yansıtır.

MVP ile "seperation of concern" hedefi daha kolay biçimde hayata geçirilebilir olmaktadır. Ayrıca uygulamaya ait davranışın da view'dan bağımsız biçimde kolay biçimde test edilebilir hale gelmesi sağlanmaktadır. MVP, yazılım ekiplerinin büyük bir uygulamayı fonksiyonel olarak gruplara ayırarak aynı anda birden fazla grubun beraber çalışarak geliştirmelerine yardımcı olacak bir mimarisel altyapı sunmaktadır [26].

2.2.2.3. Proxy

Proxy tasarım deseninin çalışma mantığı Şekil 7'de gösterilmiştir. Proxy, karmaşık veya oluşturulması zaman alan bir nesneyi daha basit bir nesne ile temsil etmek gerektiğinde kullanılır [27]. Eğer nesnenin oluşturulması zahmetli ise Proxy size, bu oluşturma işlemini, asıl nesne gerekinceye kadar erteleyebilmenizi sağlar.



Şekil 7. Proxy'nin döngüsel yapısı

Proxy genellikle temsil ettiği nesne ile aynı metotlara sahiptir ve nesne yüklendiği zaman asıl nesneye ulaşmamızı sağlayan metodu da bulundurmaktadır.

Proxy deseninin yararlı olabileceği birkaç durum vardır. Büyük bir nesnenin sisteme yüklenmesi zaman alan işlemdir. Uzun süren bir hesaplama işlemi sürerken ara sıra sonuçları göstermek. Network'un yoğun zamanlarında uzaktan erişime sahip (remote) bir bilgisayarda bulunan bir nesnenin network üzerinden alınması yavaş olabileceği durumlarda veya nesneye erişim haklarının kısıtlı olduğu durumlarda Proxy kullanıcı için erişim hakları doğrulaması yapabilir [28].

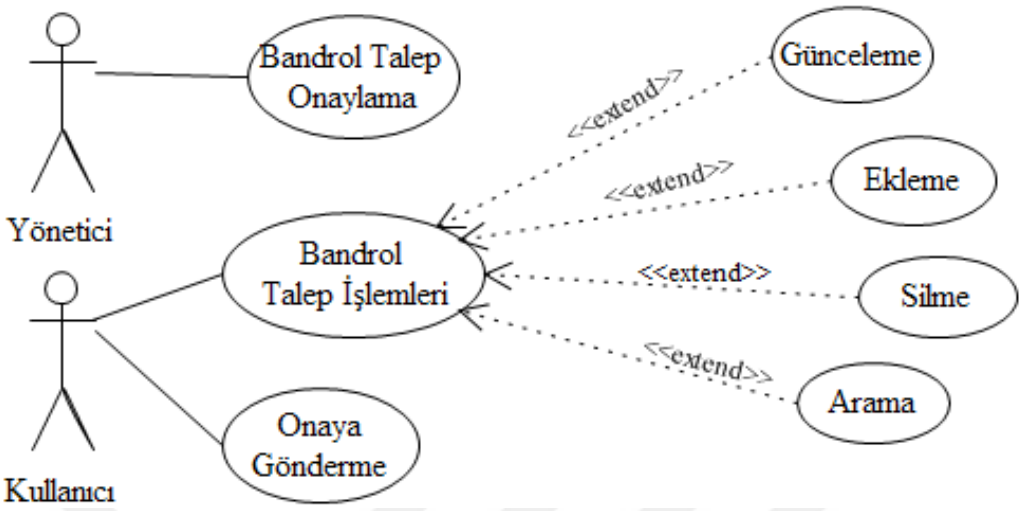
2.2.3. Tasarım Aşaması

UML, yazılım projelerinin tanımlanması ve tasarlanması amacıyla geliştirilmiş bir modelleme dilidir.

Gereksinim analiz dokümanı ve tasarım deseni belirlendikten sonra UML kullanılarak kullanım senaryoları, aktivite (aktivity), dizge (sequence) ve sınıf (class) diyagramları çizilmiştir. Bu aşamadan sonra tasarım deseni kullanılarak yazılan yazılım kodlaması yapılmıştır.

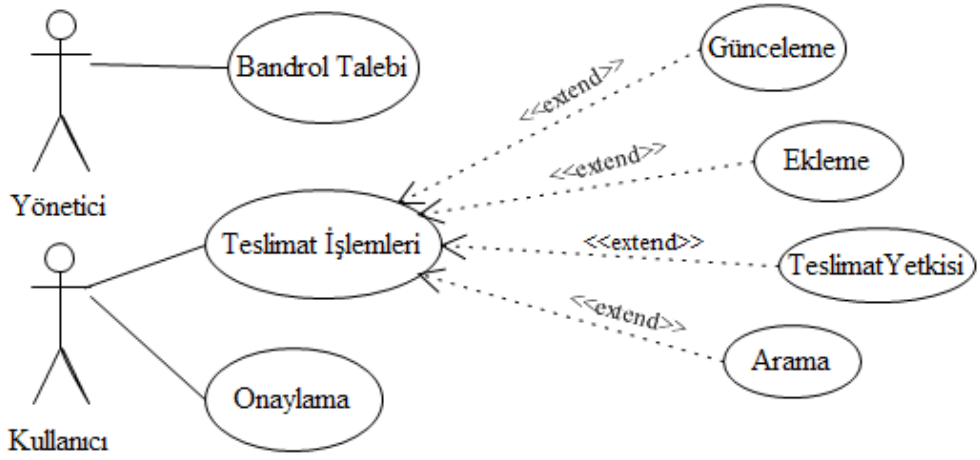
2.2.3.1. Kullanım Senaryoları ve İşlevsel Gereksinimler

Bu aşamada projenin kullanım senaryoları belirlenmiştir. Bir kullanıcı ve bir sistem arasındaki etkileşimi anlatan senaryolara kullanım senaryoları denir [29]. UML şemalarından Kullanım Senaryoları Diyagramları kullanılarak, sistemin kapsamında gerçekleştirilecek işlemler tanımlanacaktır.



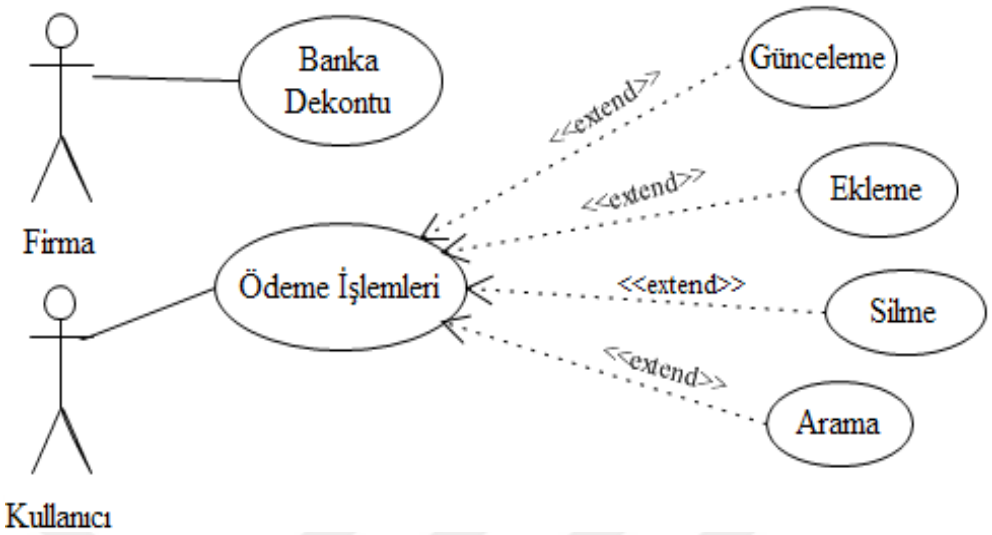
Şekil 8. Bandrol talep işlemleri kullanım senaryosu

Firmalar TRT'ye gelerek bandrol talebinde bulunurlar. Kullanıcı firmanın bandrol talebini sisteme girerek gerekli alanları doldurur. Sonra yöneticisine onaya sunar. Bu aşamada gerekli ekleme-silme-güncelleme işlemleri varsa kullanıcı bu işlemleri gerçekleştirmektedir.



Şekil 9. Teslimat işlemleri kullanım senaryosu

Bandrol talep işlemleri onaylanan firma ilgili kullanıcıya gelerek gerekli olan belgeleri gösterip kendisine ait olan bandrolleri teslim alır. Bu aşamada gerekli ekleme-silme-güncelleme işlemleri varsa kullanıcı bu işlemleri gerçekleştirmektedir.

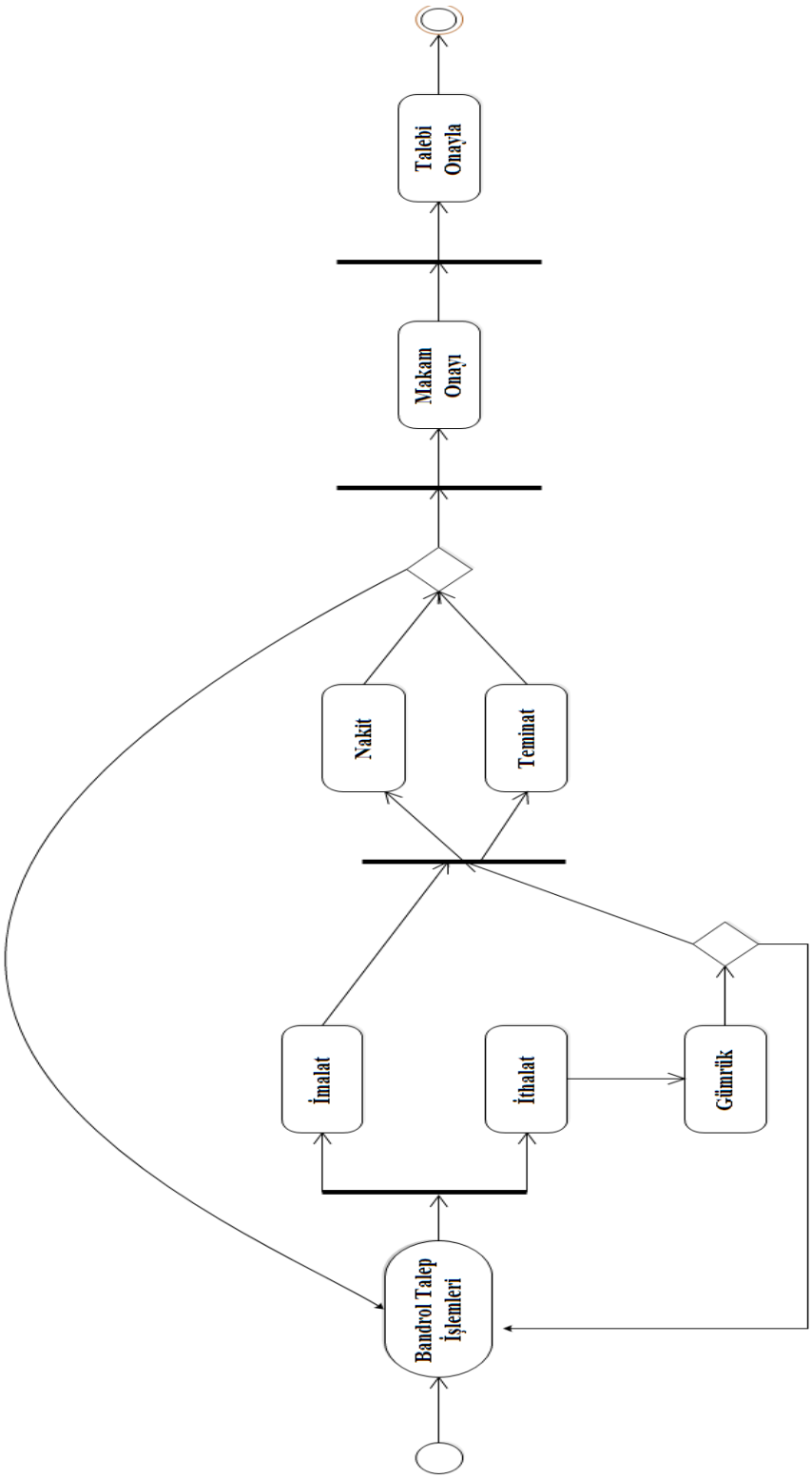


Şekil 10. Ödeme işlemleri kullanım senaryosu

Firmalar bandrol talep işlemlerinde bulunurken bandrolün hesaplanan ücretini TRT'nin ilgili hesabına ödemede bulunurlar. Firma ilgili dekontu TRT'ye getirir. Kullanıcı da firmanın getirmiş olduğu dekonttaki bilgileri sisteme girmektedir. Projede kullanılan bandrol talep işlemleri kullanım senaryosu şekil 8'de, teslimat işlemleri kullanım senaryosu şekil 9'da ve ödeme işlemleri kullanım senaryosu Şekil 10'da gösterilmektedir.

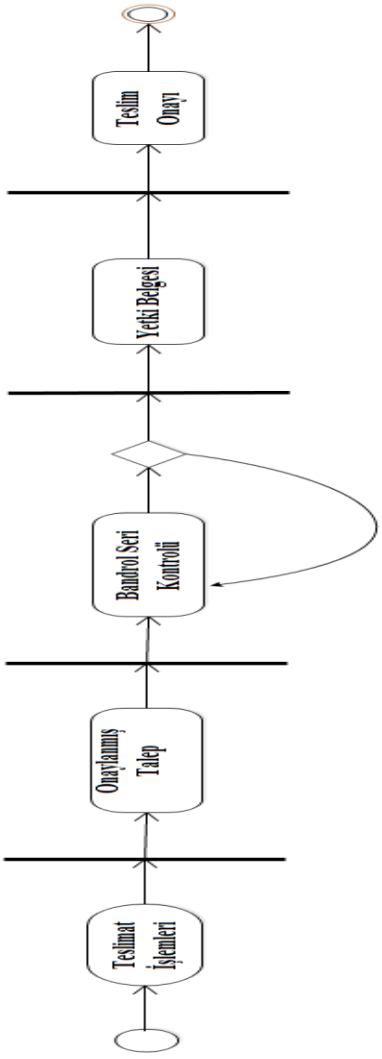
2.2.3.2. Aktivite Diyagramı

Faaliyetler arası kontrol akışını modelleyerek sistemin dinamik yapısını gösteren diyagramlardır [30].



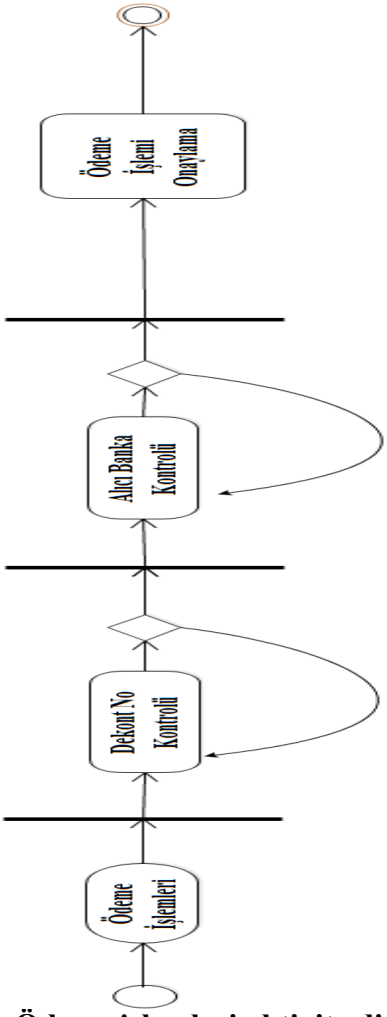
Şekil 11. Bandrol talep işlemleri aktivite diyagramı

Bandrol talep işlemlerinin dinamik yapısı Şekil 11'de gösterilmiştir. Talep işlemleri yapılırken ilk önce talebin türü seçilir. Seçilen talep türü ithalat ise gümrük bakanlığına bağlı GUVAS sisteminden ilgili talebin beyanname numarası kontrolü yapılmaktadır. Sonra ödeme türü seçilir. Seçilen ödeme türüne göre firmaya ait nakit ve teminat hesabındaki bakiye kontrolünü yapılır. Son aşamada ise ilgili talep makam onayına sunulmaktadır.



Şekil 12. Teslimat işlemleri aktivite diyagramı

Bandrol teslimat işlemlerinin dinamik yapısı Şekil 12'de gösterilmiştir. Bandrol teslimat işlemi yapılırken ilk koşul firmanın ilgili talebinin onaylanmış olması gerekmektedir. Kullanıcı firmaya teslim edeceği bandrollerin seri no aralıklarını seçer. Seçilen bandrol seri numara aralıklarının kontrolünü yapılır. Firma tarafından kullanıcıya teslimatta bulunacak kişinin bilgilerini önceden verilmesi gerekmektedir. Kullanıcı teslimatı alacak firma yetkilisinin bilgilerini kontrol eder. Kullanıcı teslimat tutanağı onaylatılarak ilgili bandrolleri firma yetkilisine teslim etmektedir.

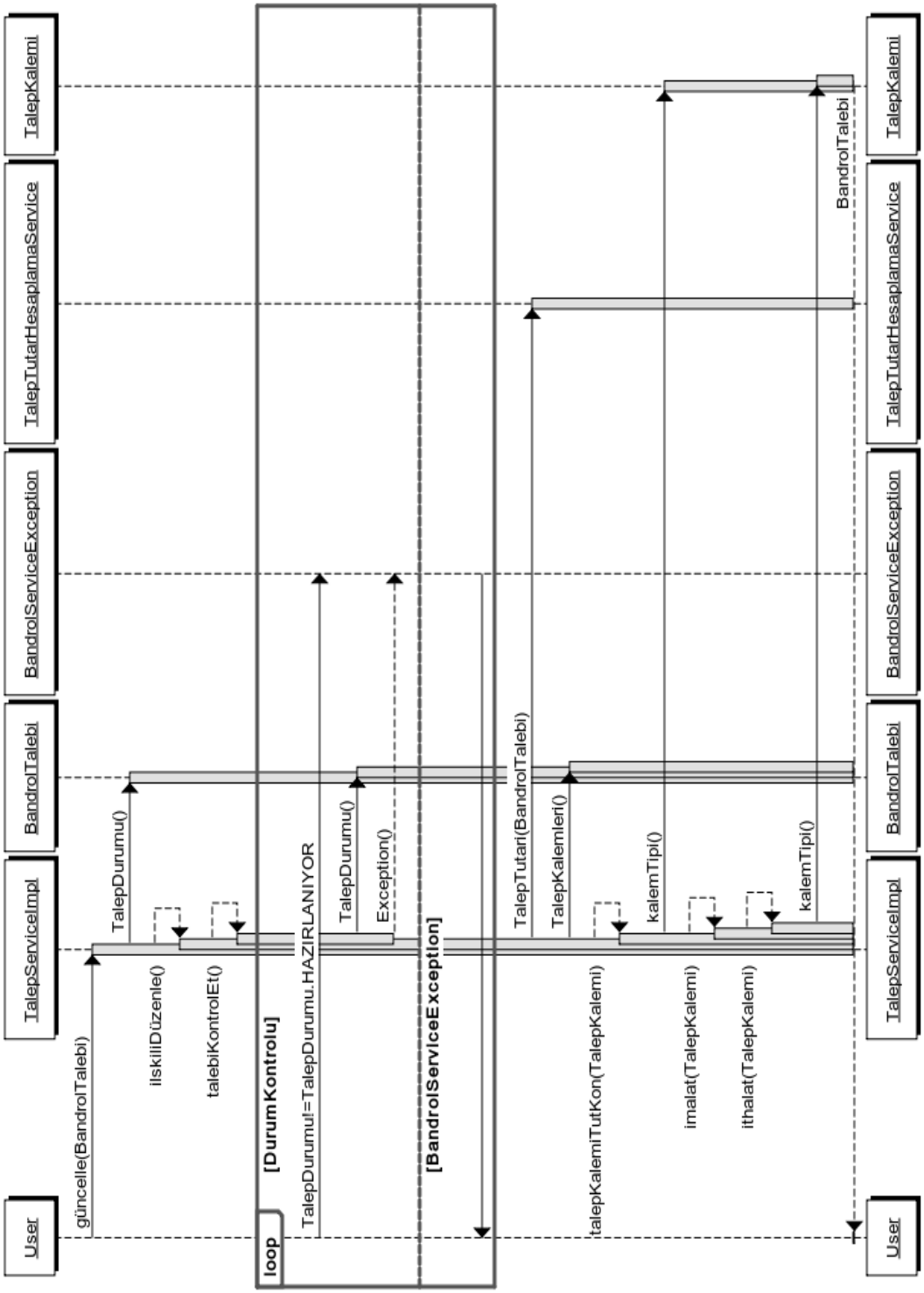


Şekil 13. **Ödeme işlemleri aktivite diyagramı**

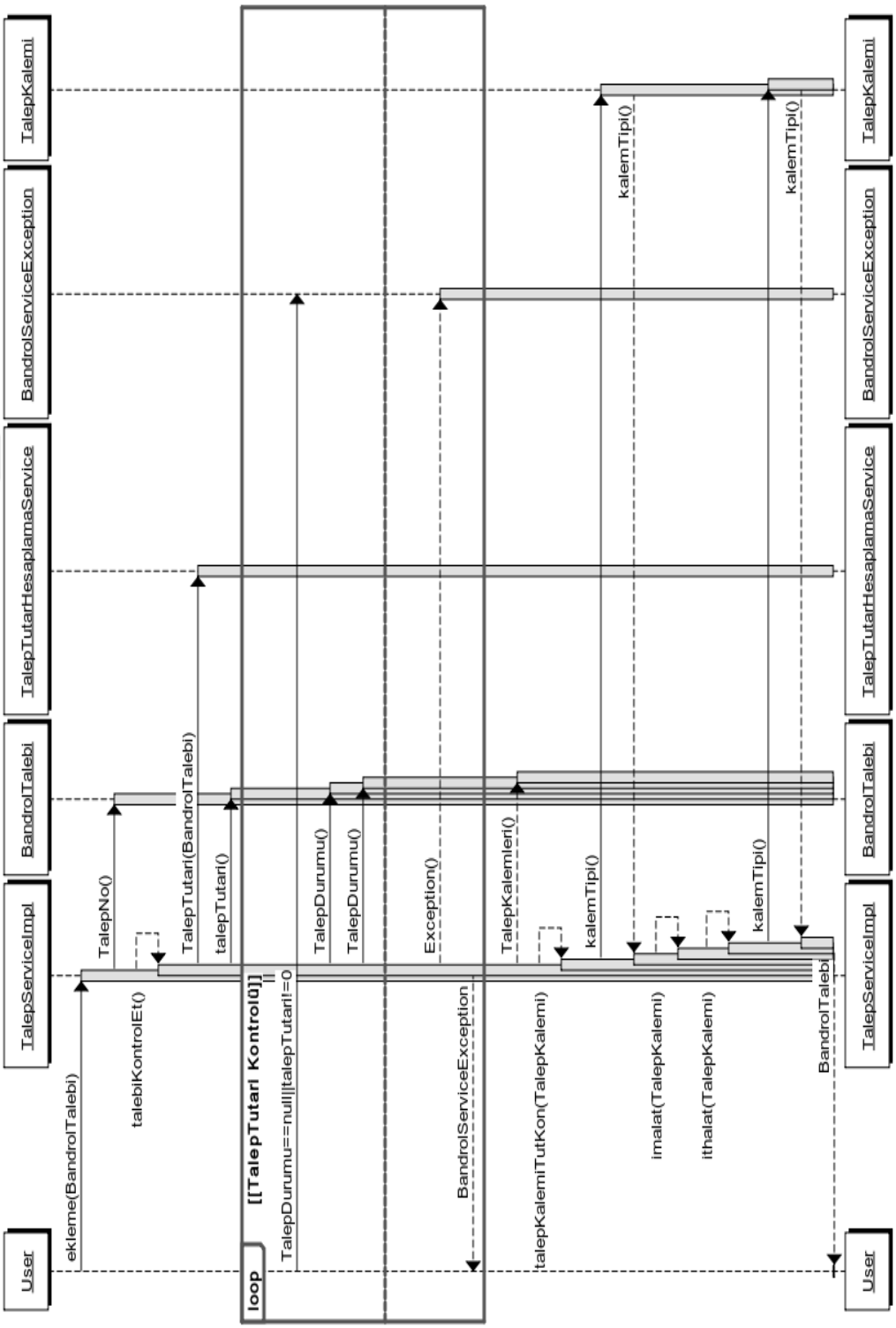
Ödeme işlemlerinin dinamik yapısı Şekil 13’te gösterilmiştir. Firma bandrol ücretini bankaya yatırdıktan sonra bandrol talep işlemi yapabilmektedir. Firma yetkilisi yatırmış olduğu dekontu kullanıcıya teslim etmektedir. Kullanıcı da firma yetkilisinin getirmiş olduğu dekontu kontrol etmektedir. Kullanıcı dekontla ilgili kontrolleri yaptıktan sonra onaylama işlemini yapmaktadır.

2.2.3.3. Dizge Diyagramı

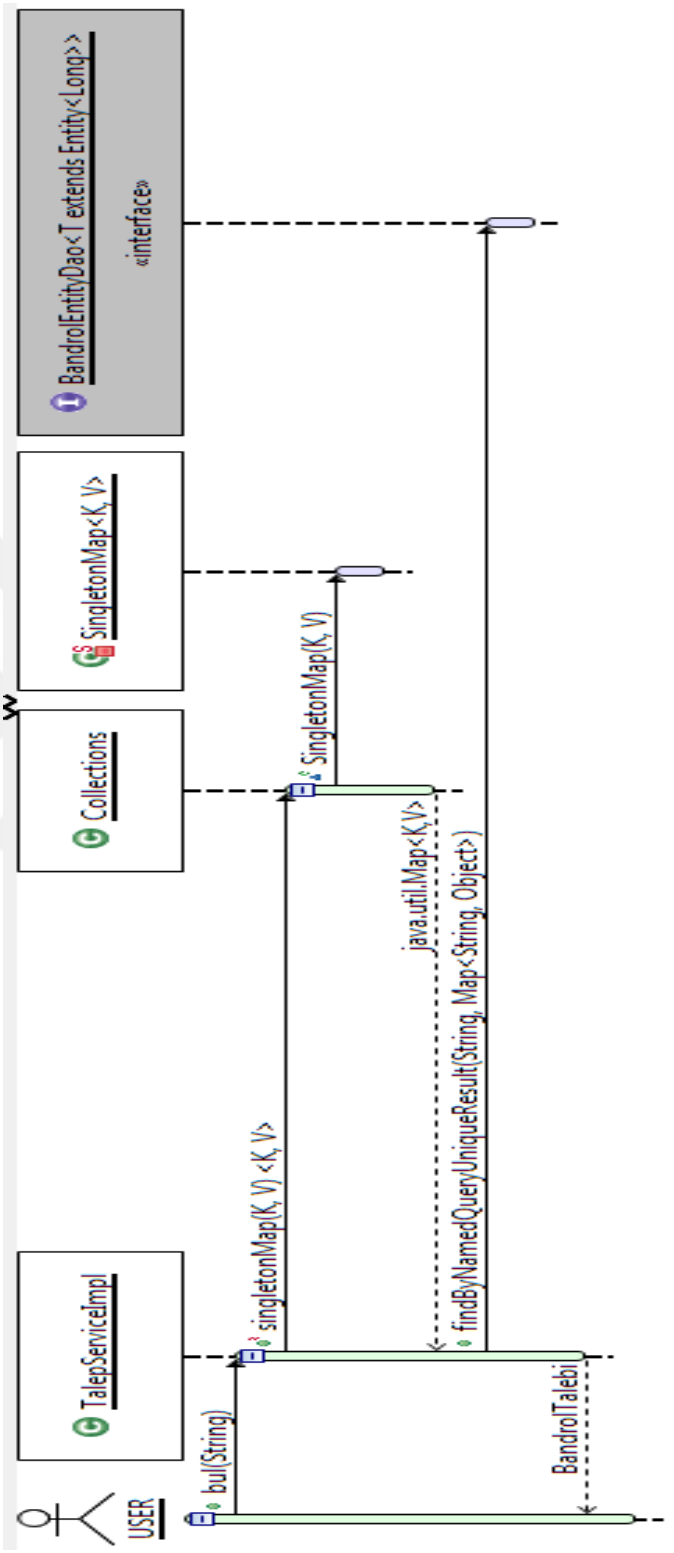
UML şemalarından Dizge Diyagramları kullanılarak, sistemin kapsamında gerçekleştirilecek işlemler tanımlanmıştır. Dizge diyagramları, yazılımdaki nesneler ya da bileşenler arasındaki mesaj akışının olaylarını ve hareketlerini dizge şekilde modellemek için kullanılan diyagramlardır [31]. Sistemin nesneler arasındaki dinamik davranışını tarif etmektedir. Dizge diyagramları çizerken sınıflardaki metotlar göz önünde bulunduruldu. [Sayfa 25-33] ("Şekil14-Şekil25 arasındaki dizge diyagram şekilleri") kullanım senaryoları göz önünde bulundurularak çizilmiştir.



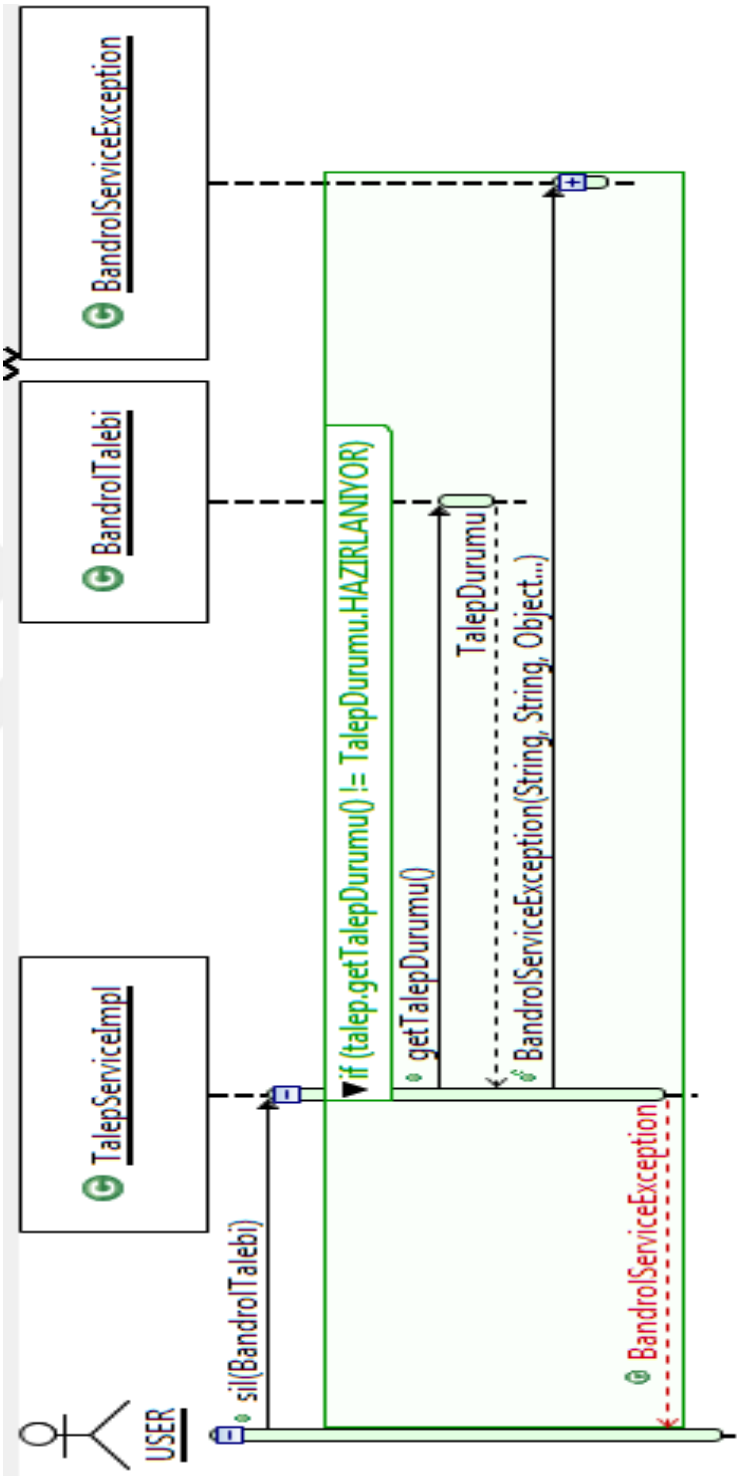
Şekil 14. Bandrol talep işlemleri güncelleme senaryosu



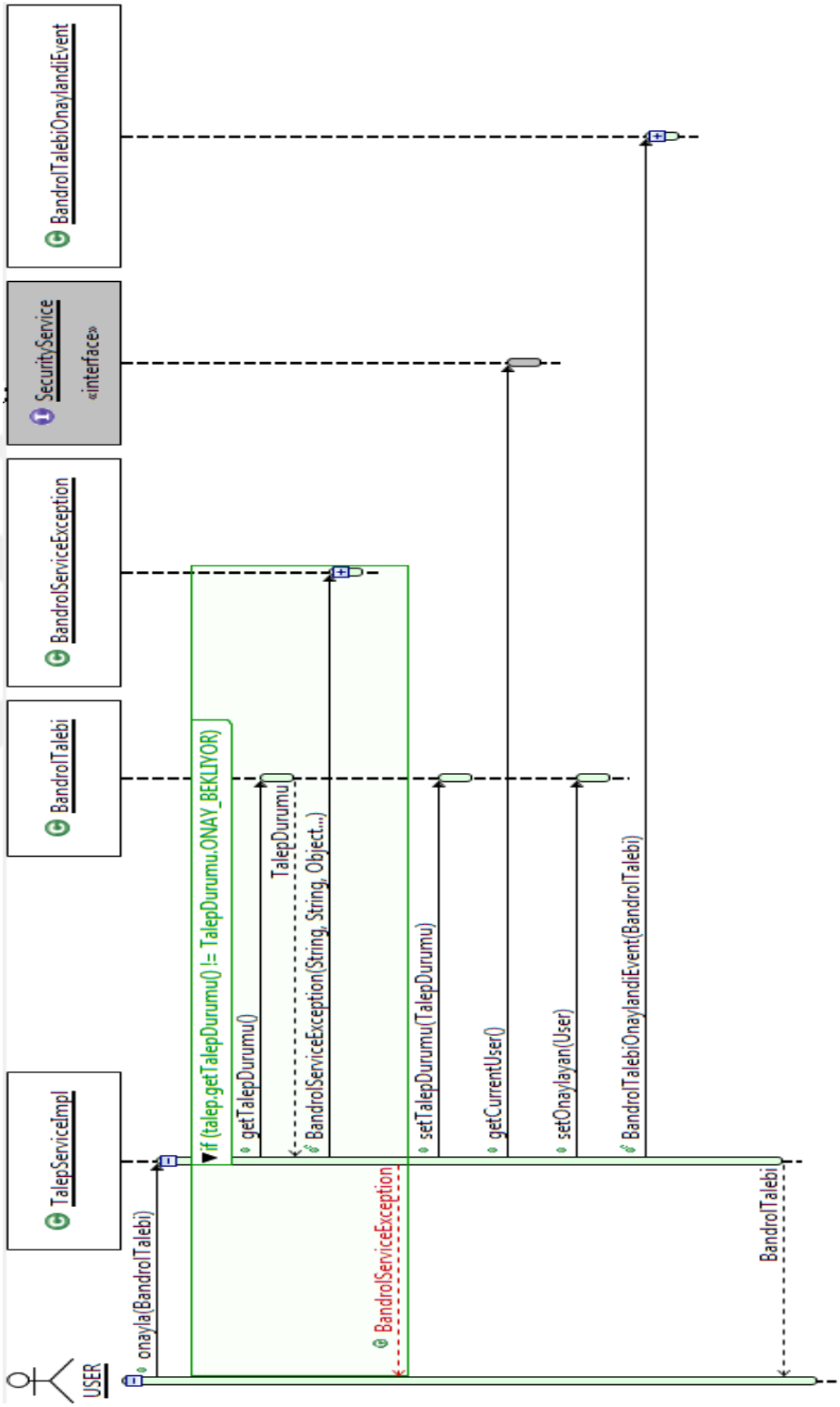
Şekil 15. Bandrol talep işlemleri ekleme senaryosu



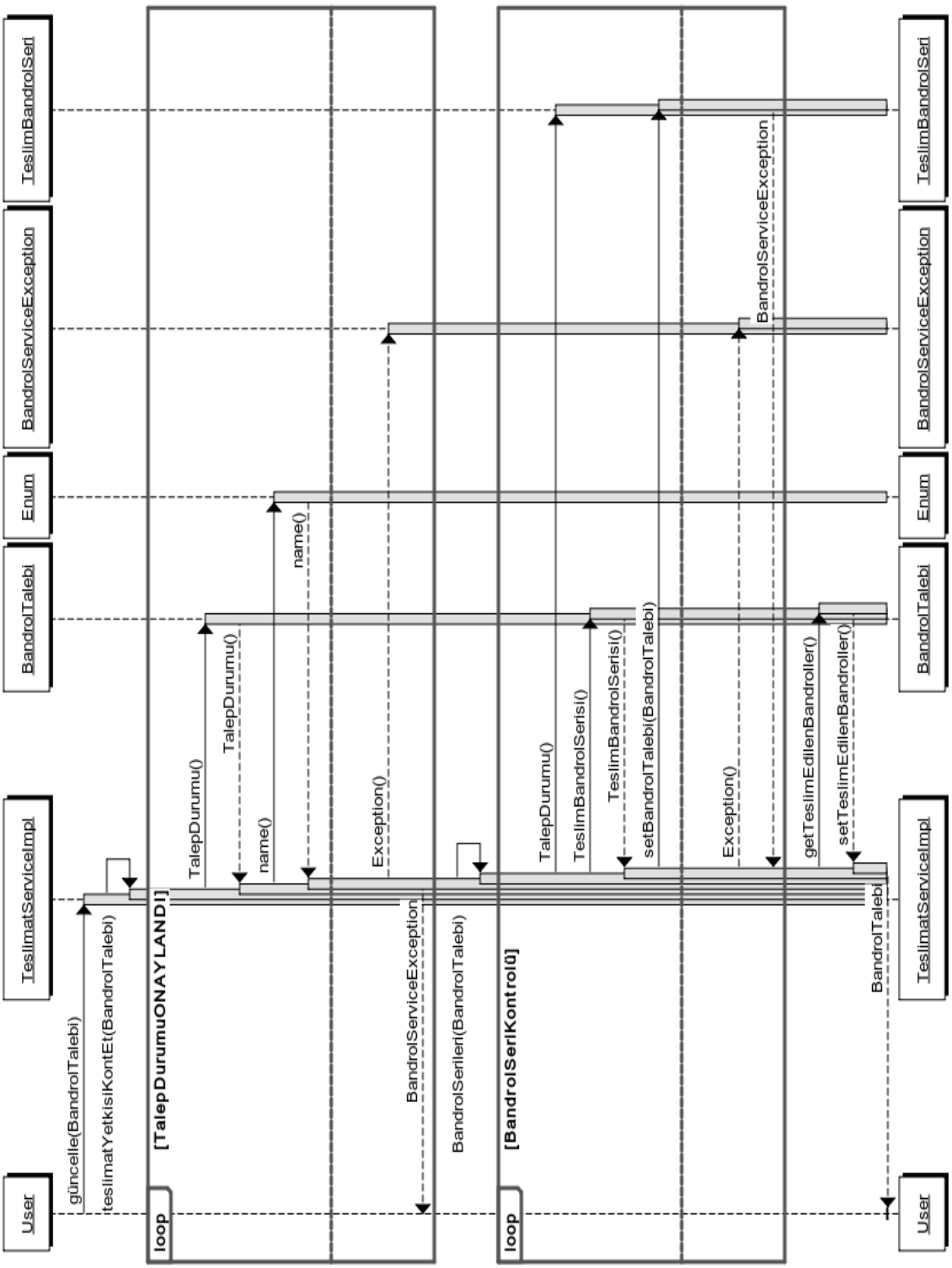
Şekil 16. Bandrol talep işlemleri arama senaryosu



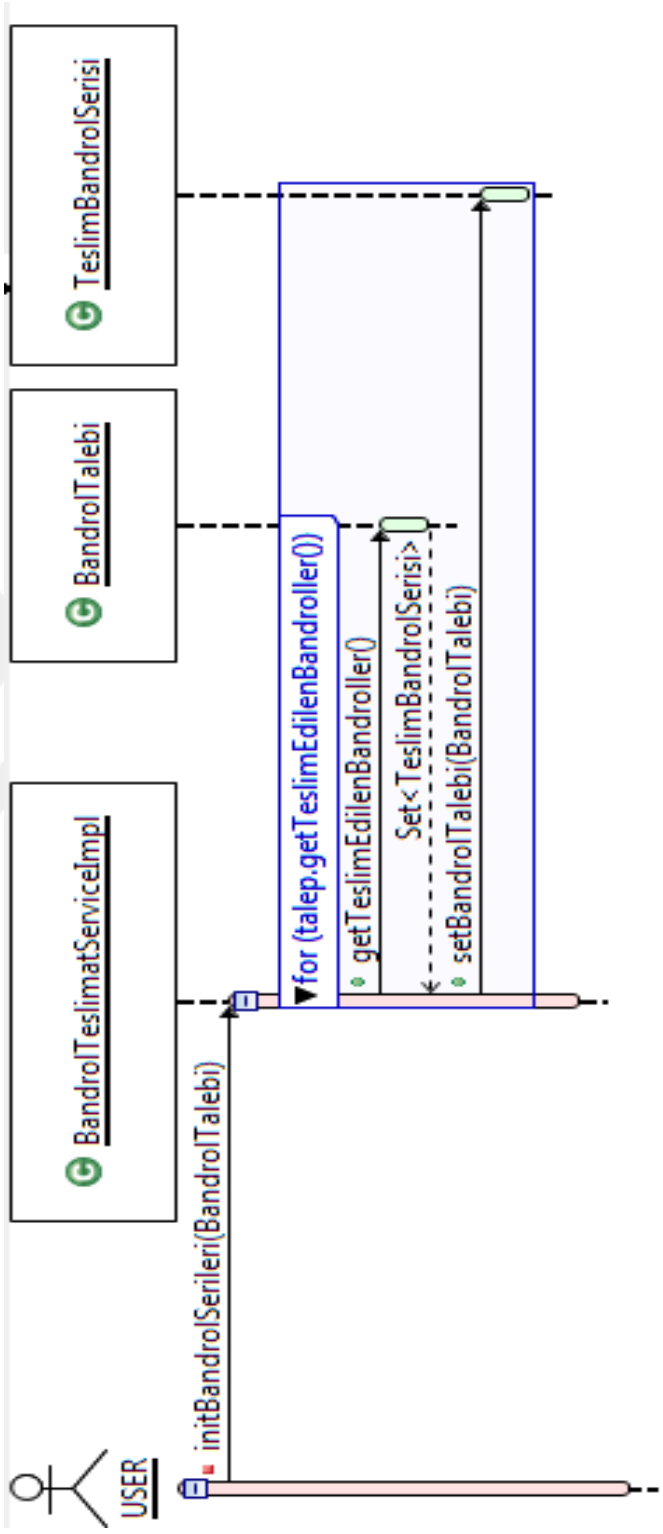
Şekil 17. Bandrol talep işlemleri silme senaryosu



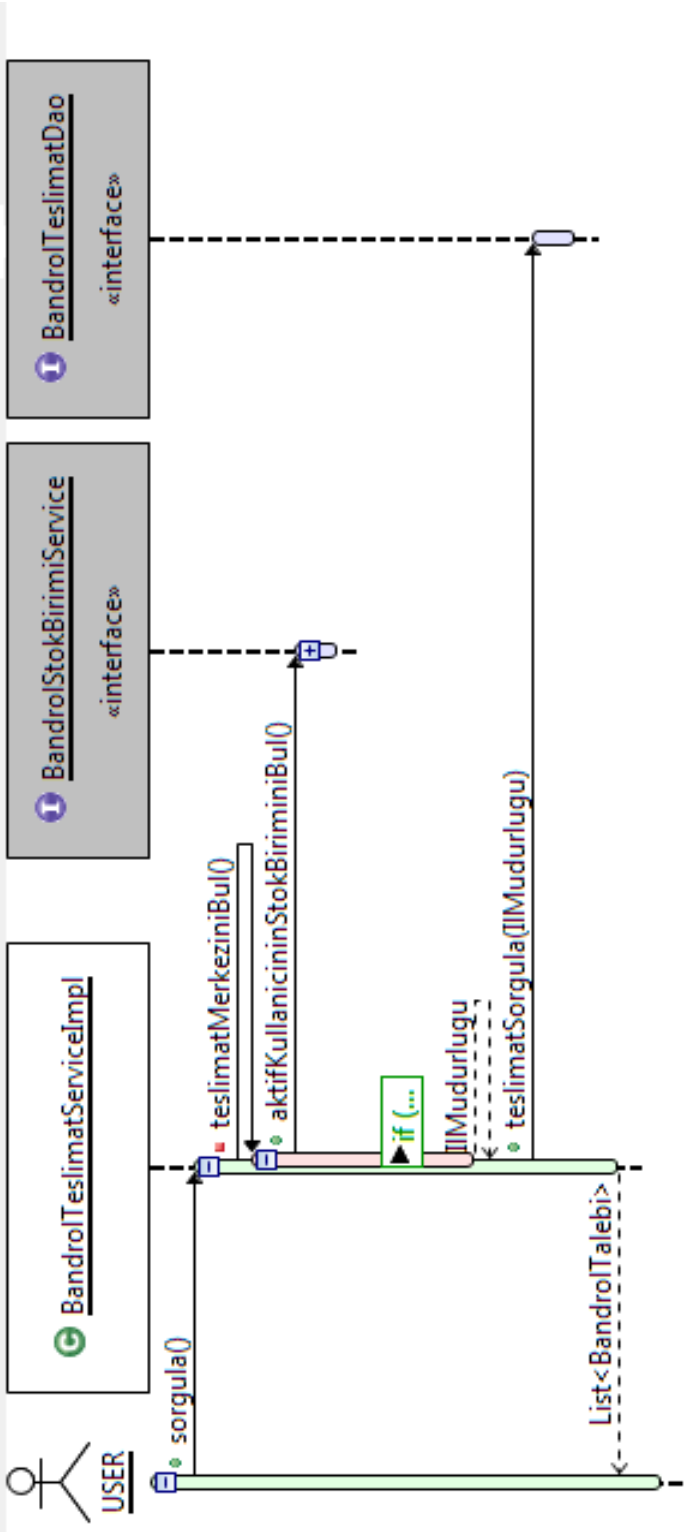
Şekil 18. Bandrol talep işlemleri onaylama senaryosu



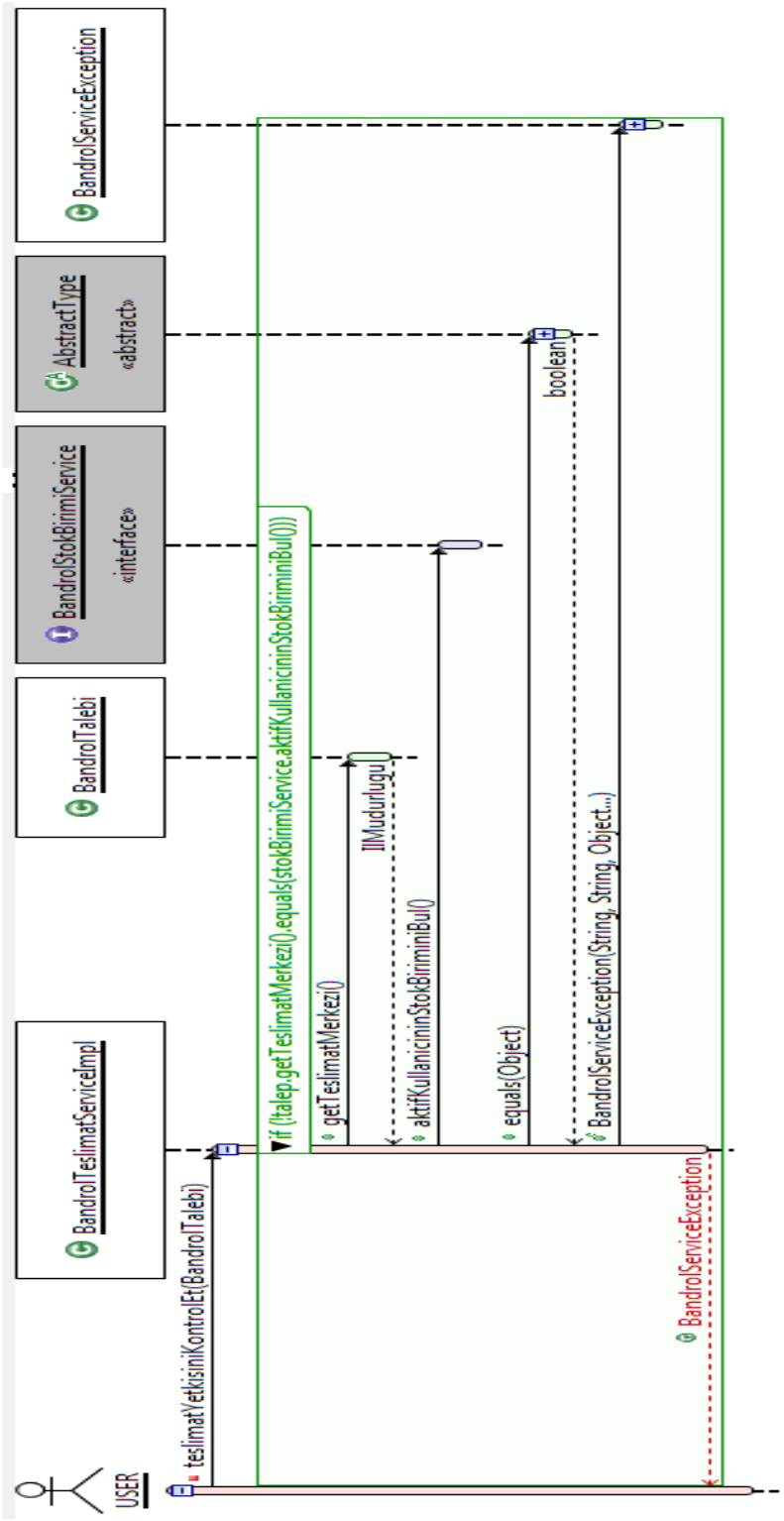
Şekil 19. Teslimat işlemleri güncelleme senaryosu



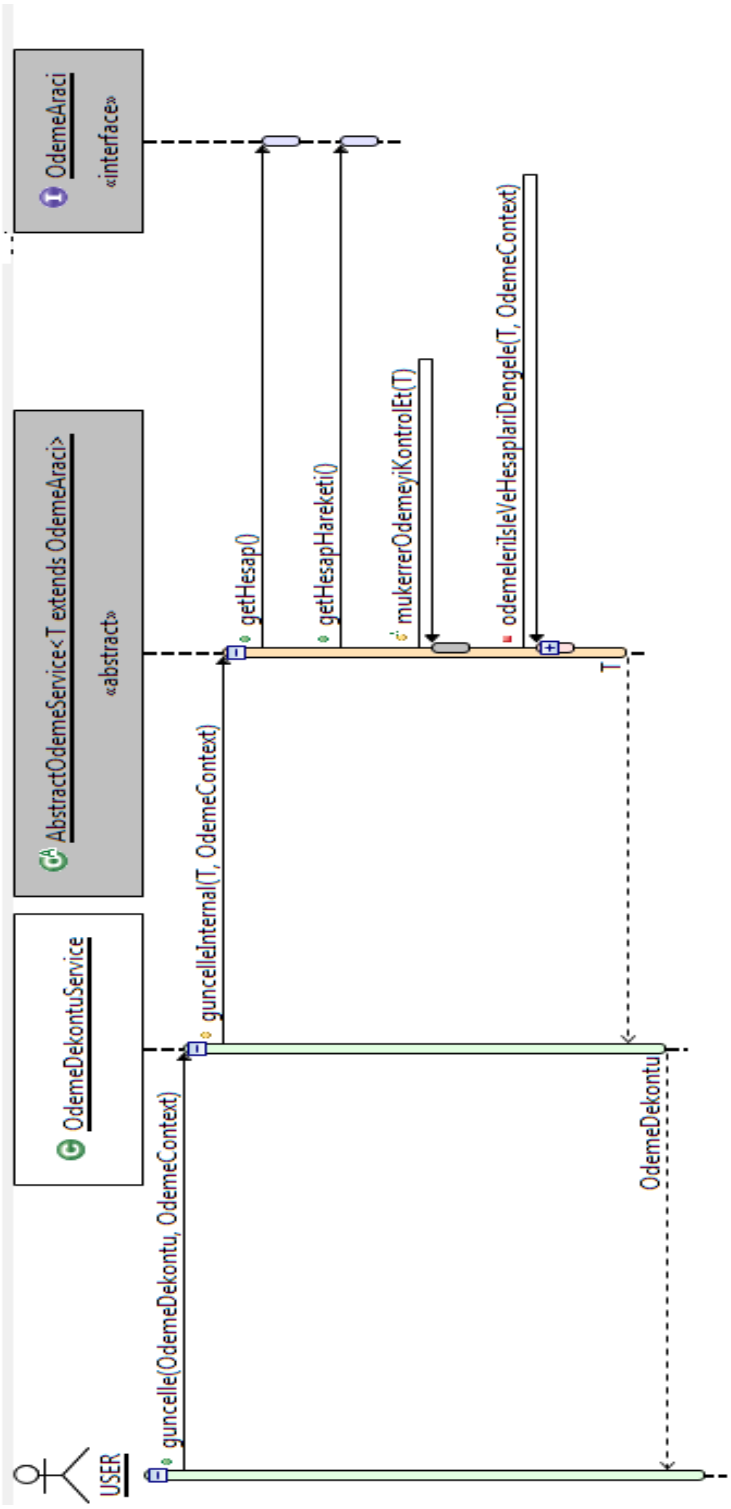
Şekil 20. Teslimat işlemleri ekleme senaryosu



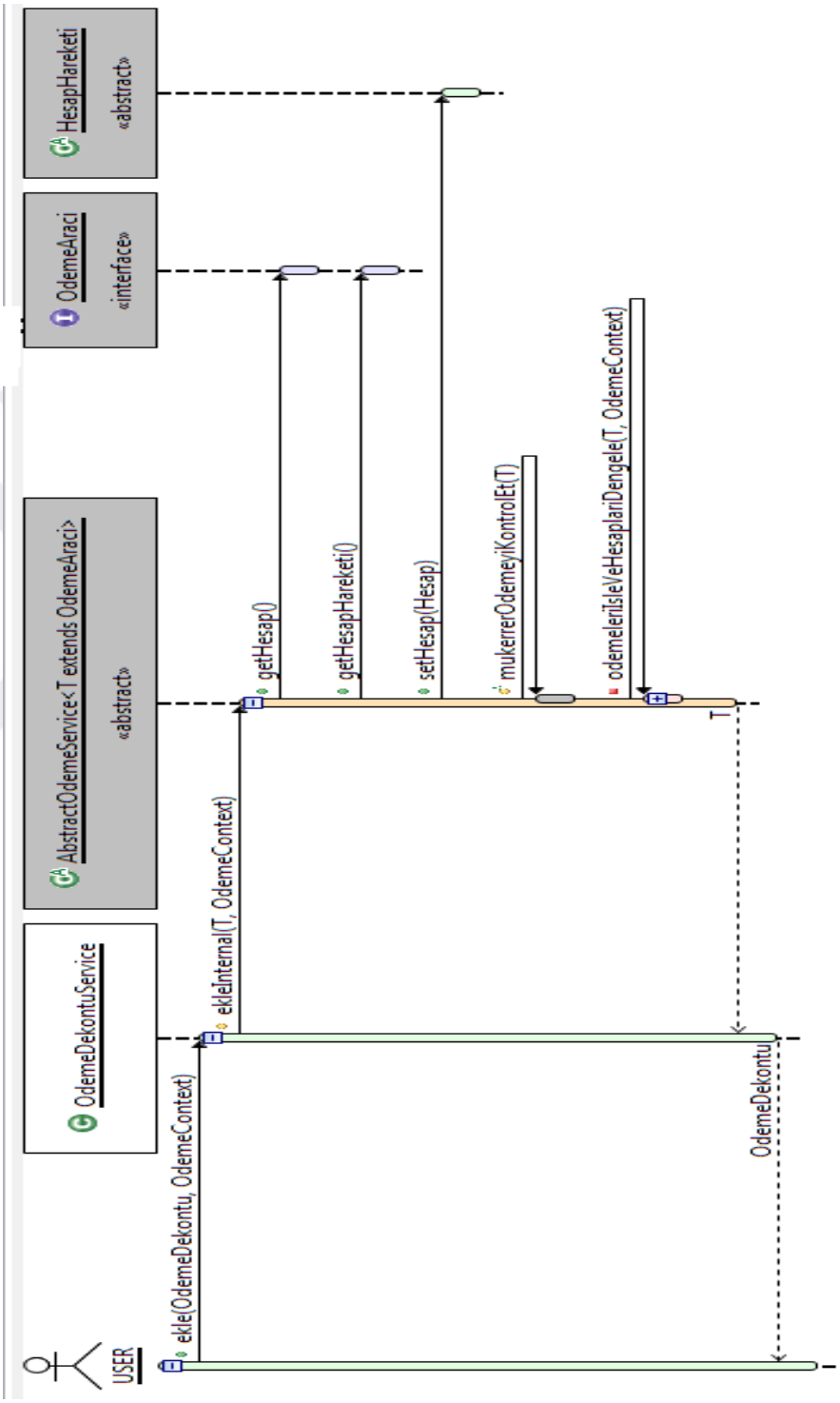
Şekil 21. Teslimat işlemleri arama senaryosu



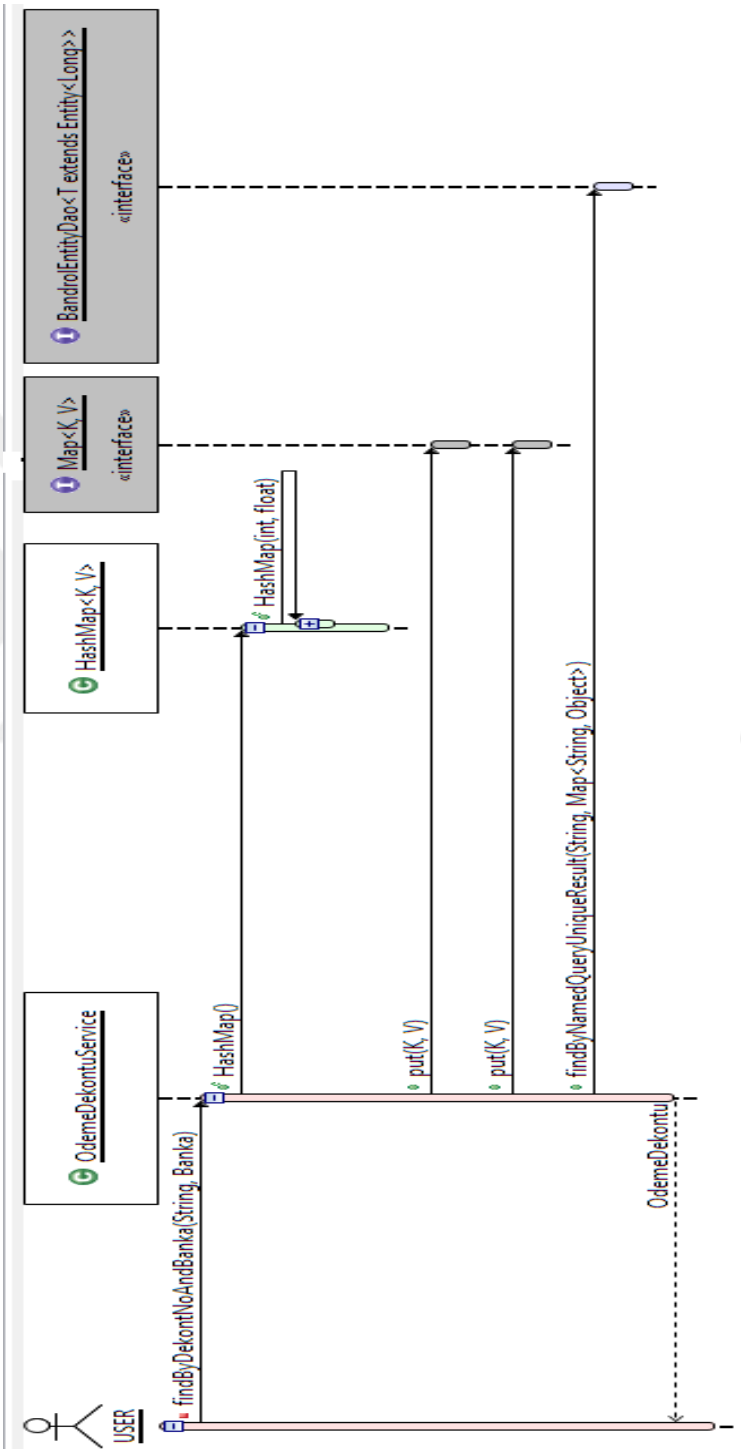
Şekil 22. Teslimat işlemlerindeki teslimat yetkisi senaryosu



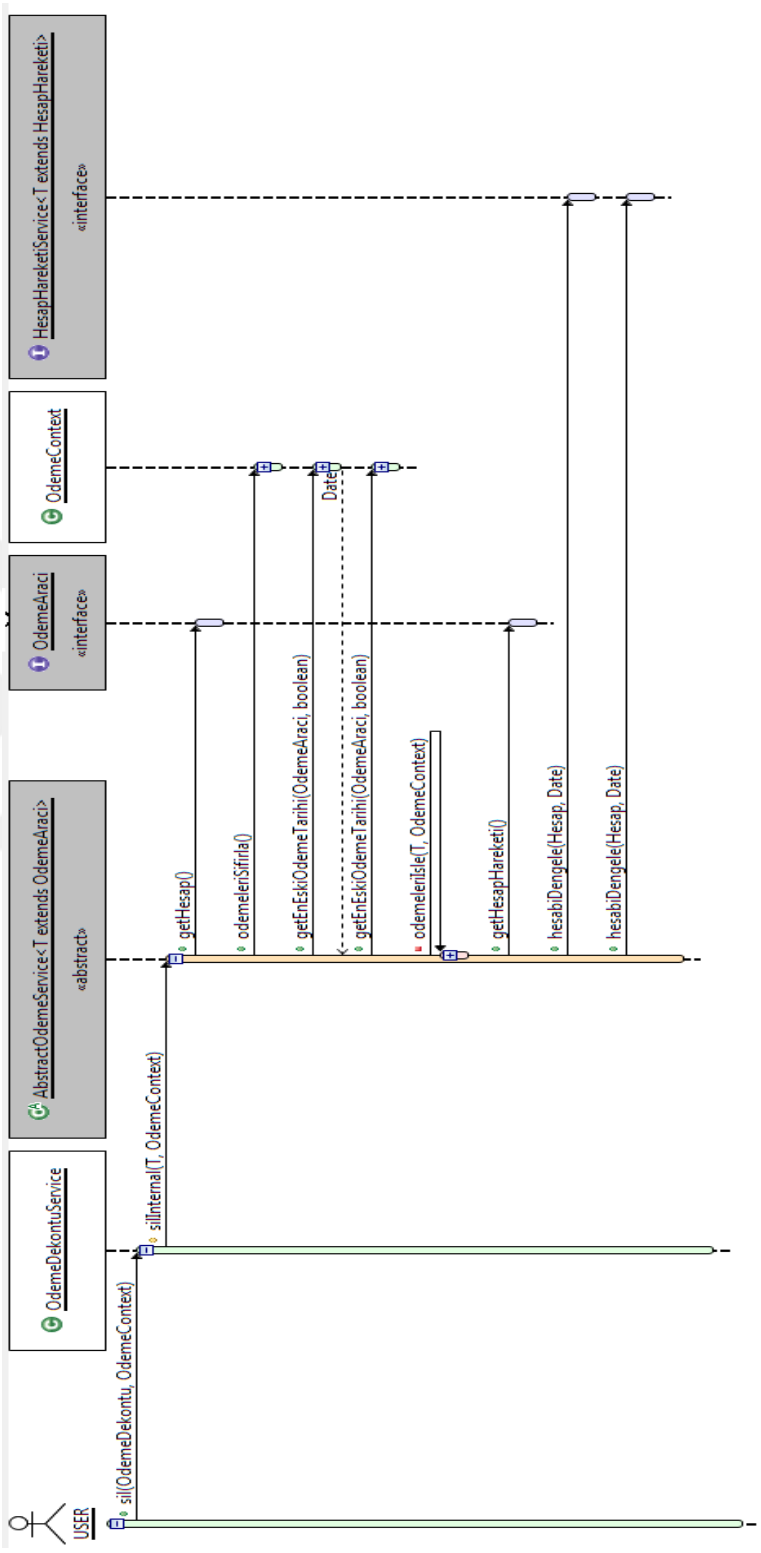
Şekil 23. Ödeme işlemleri güncelleme senaryosu



Şekil 24. Ödeme işlemleri ekleme senaryosu



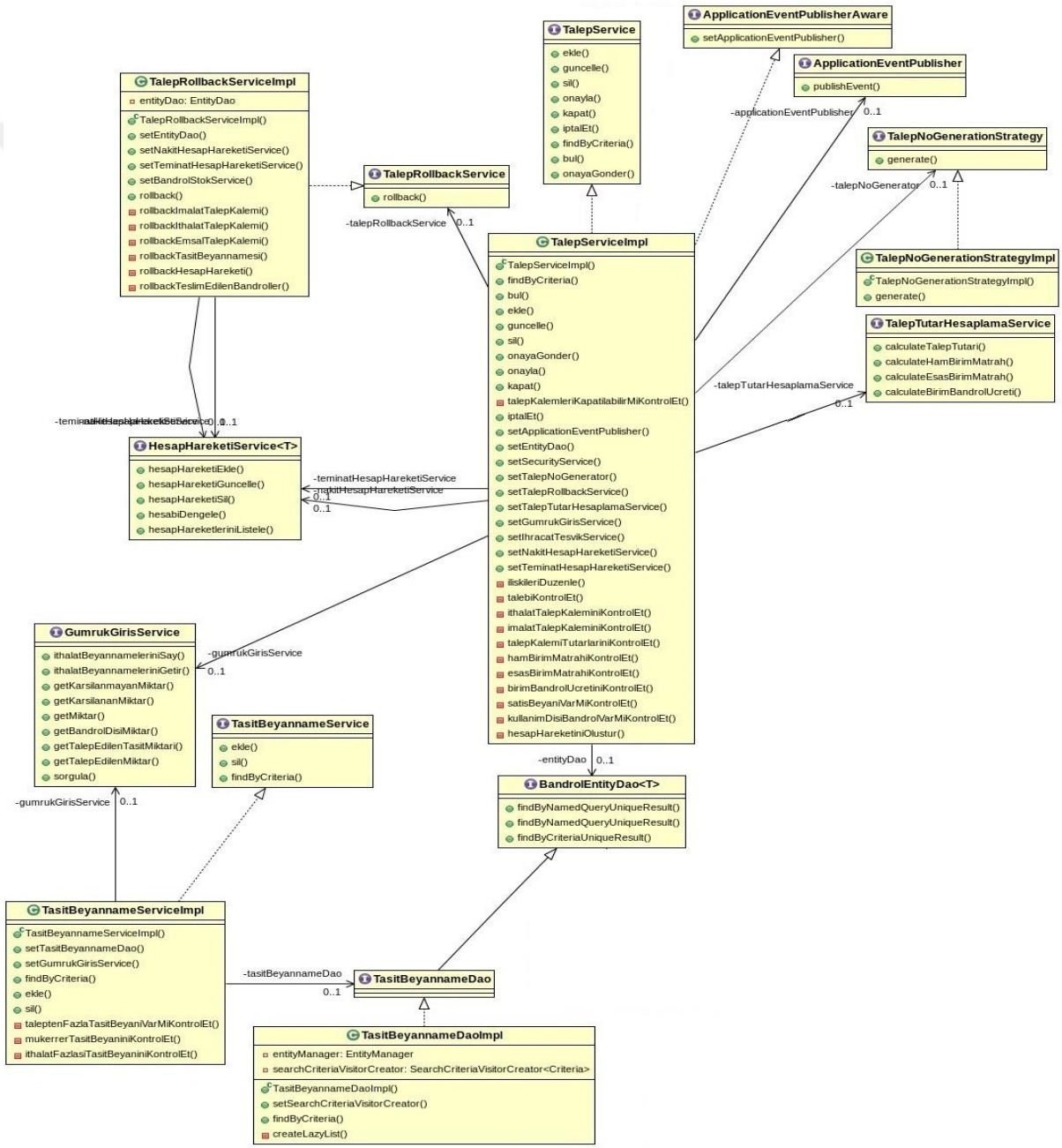
Şekil 25. Ödeme işlemleri arama senaryosu



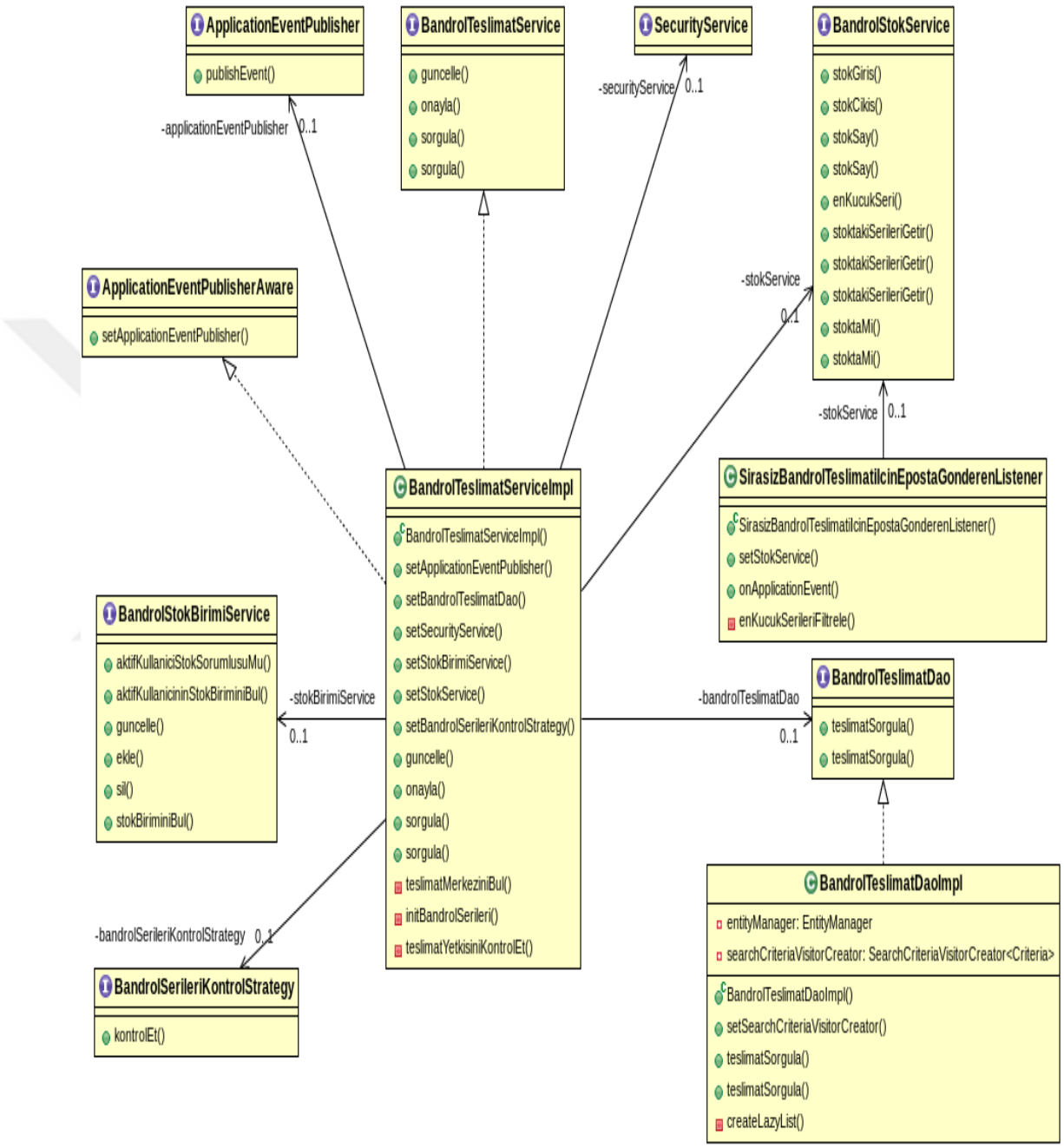
Şekil 25. Ödeme işlemleri silme senaryosu

2.2.3.4.Sınıf Diyagramı

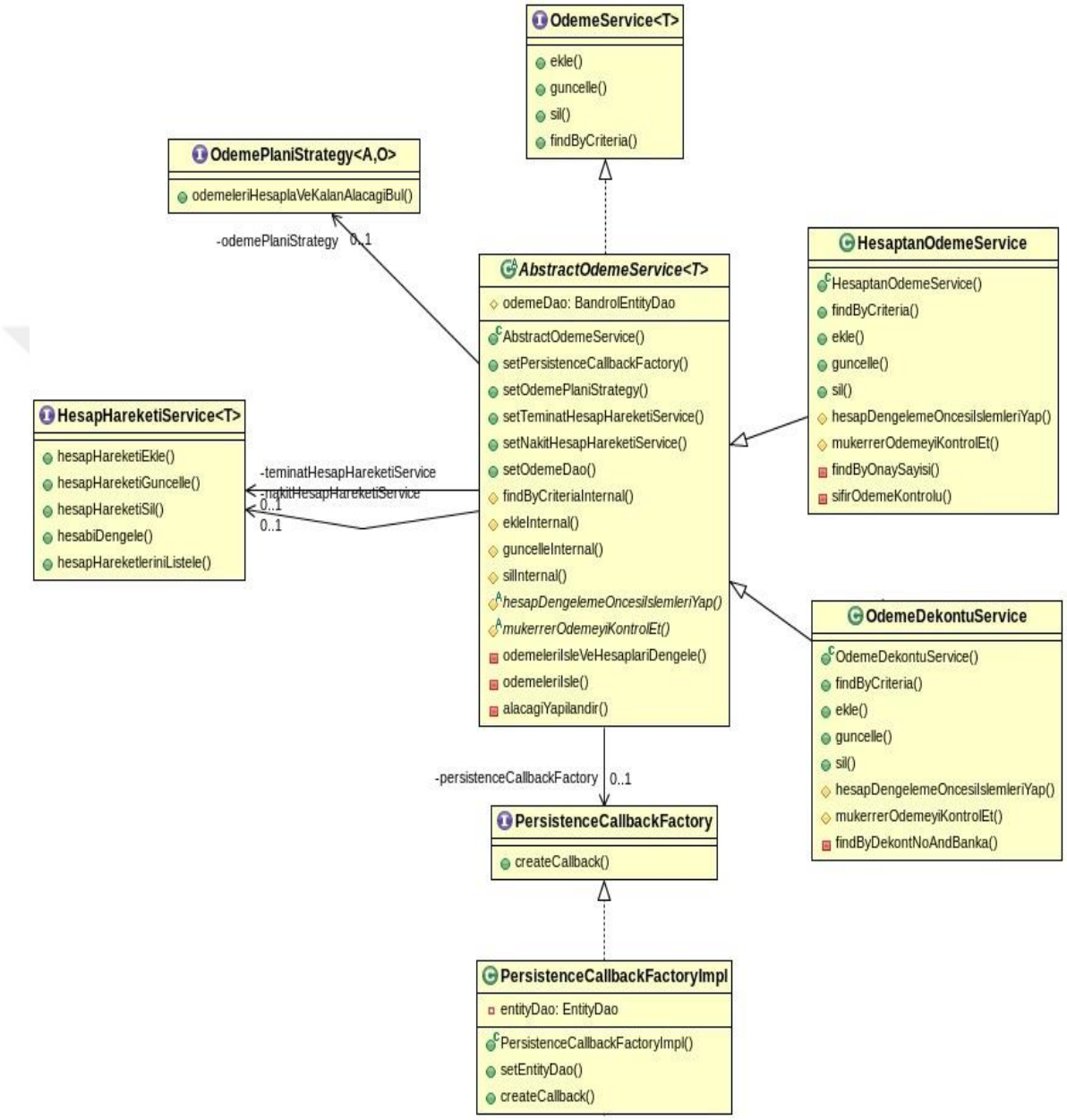
Sınıf diyagramları, sistemdeki sınıfların, özelliklerini ve birbirileri arasındaki ilişkileri gösterir. Sistem içerindeki nesneler tasvir edilir [32].



Şekil 26. Bandrol talep işlemleri sınıf diyagramı



Şekil 27. Teslimat işlemleri sınıf diyagramı



Şekil 28. Ödeme işlemleri sınıf diyagramı

2.2.3.5. Yazılımın Statik Mimarisi

Tasarım desenleri kullanılarak geliştirilen yazılımda MVC, MVP ve Proxy desenleri kullanılmıştır. Bu aşamada, geliştirilen bir yazılımda tasarım desenlerinin nasıl kullanıldığı tarif edilmiştir.

Servis katmanında MVP tasarım deseni kullanılmıştır. MVP, service katmanıyla view katmanı arasındaki iletişimi sağlamaktadır. Örneğin BandrolTalebi sınıfında talepno, talepdurumu, beyantipi, ödemesekli, taleptutarı gibi alanlar bulunmaktadır. Kullanıcı ara yüz üzerinden beyantipi olarak imalat ve ithalatı seçebilmektedir. Kullanıcı beyan tipini imalat seçerse presenterlar view katmanında imalatla ilgili alanları, eğer ithalat seçerse view katmanında ithalat ile alanları görüntülemektedir. View katmanından service katmanına iletilen bütün işlemleri MVP yapısındaki presenter tarafından yapılmaktadır. Service katmanın yapılan bütün işleri presenter yapısındaki eventler sağlamaktadır. Şekil 26, Şekil 27 ve Şekil 28'da görüldüğü gibi service katmanındaki bütün işlemler eventler sayesinde yapılmaktadır.

AbstractOdemeService, HesaptanOdemeService ve OdemeDekontuService işlem yönetimi (transaction) Proxy tasarım deseni kullanılarak düzenlenmiştir. OdemeService interfacesine ait yetki aktarma (delagate) nesnesinin metotları aynı zamanda proxy deseni içinde kullanılmaktadır. Söz konusu metotlar çağrılmadan önce işlem yönetimi açılır. Açılmış çağrı sonrasında işlem yönetimi kayıt (commit) etmiştir. Eğer yetki aktarım metodu içerisinde herhangi bir hata (exception) olursa işlem yönetiminin geri alması (rollback) sağlanmıştır. Benzer şekilde metot sürelerinin ölçümünde de (Monitoring) bu tasarım deseninden faydalanılmıştır. Bu işlem Proxy tasarım deseni ile yapılabileceği gibi AOP yapısındaki byte-code manipülasyon yöntemiyle de sağlanabilmektedir. Proxy deseni ayrıca entitynin temiz kalmasını sağlar. Örneğin kullanıcı bandrol talep işlemini yaparken hata oluşması durumunda entitynin kirli (dirty) kalmaması için geri alma yapılır. Hata olmadığı zaman ise kaydedilerek entity temiz bırakılır.

2.2.4. Test Aşaması

İki projeyi karşılaştırabilmek için çeşitli testler yapılmıştır. Bu testler kullanıcı sayısı ve deneme sayısı değiştirilerek yapılmıştır. Test işlemi için iki araç kullanılmıştır. Performans testlerinin yapılma amacı iki projeye yapılan girdilerden alınan dönütlerle, olması gereken sonuçları karşılaştırmaktır. Test aşamasında aşağıdaki iki araç kullanılmıştır.

2.2.4.1. Jmeter

Test ve performans ölçümü yapabilen, ölçümünü yaptığı sonuçları tablo ve grafik halinde sunabilen bir yazılımdır [33]. Geliştirilen bir yazılımın ürün olarak en önemli özelliklerinden biri de hız ve performans olarak ölçümlerinin yapılabilmiş olmasıdır [34]. Performans ölçümlerinin sonucunda uygulamanın kaç kullanıcılı bir yapıyı kaldırabildiği, darboğazların oluştuğu web sayfaları, işlem hacmi, yoğun zamanlarda kullandığı işlemci, hafıza gibi birçok değerli veriye sahip olabilmeyi sağlar.

2.2.4.2. Speedy Framework

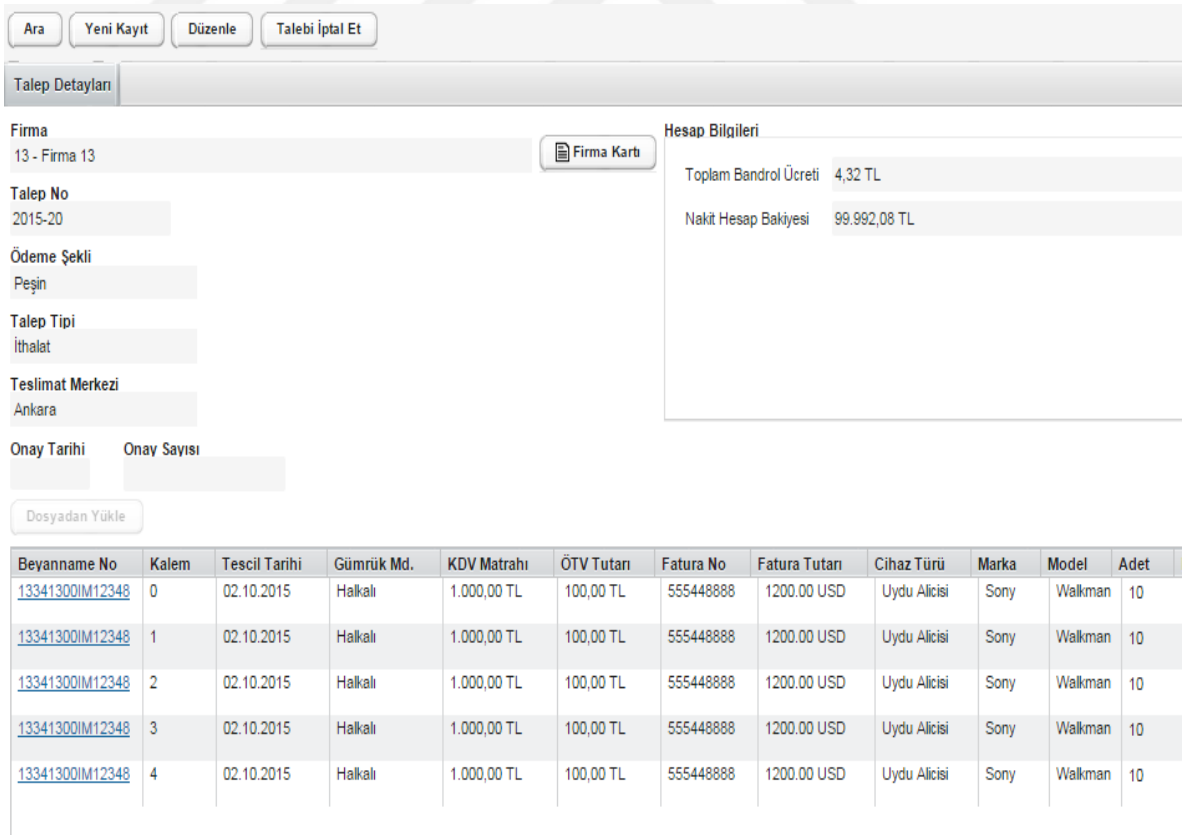
Web tabanlı kurumsal uygulamalar geliştirmek için kullanılan model güdümlü yazılım geliştirmeye yönelik bir framework'tur. Speedy Framework, birçok altyapısal servisin yanı sıra çeşitli katmanlarda performans ölçümü yapmaya yönelik mekanizmalar sunmaktadır. Speedy-Monitoring modülü sayesinde request, event veya metot düzeyinde performans ölçümü yapılabilmektedir. Tasarım deseni kullanılarak geliştirilen yazılım speedy framework ile geliştirilmiş olup performans ölçümlerinde adı geçen araçlardan yararlanılmıştır.

ÜÇÜNCÜ BÖLÜM
İKİ UYGULAMANIN KARŞILAŞTIRILMASI

Bu bölümde geliştirilen yazılımın kullanıcı ara yüzleri hakkında bilgi verilmiştir. Uygulamalar işlem sayısı, ortalama süre, minimum süre ve maksimum süre alanları gözetilerek performans değerlerinin ölçülmesi ve zaman bakımından iki yazılımın karşılaştırılmıştır. Elde edilen veriler analiz edilmiştir.

3.1. GELİŞTİRİLEN YAZILIMIN KULLANICI ARA YÜZLERİ

3.1.1. Bandrol Talep İşlemleri Arayüzü

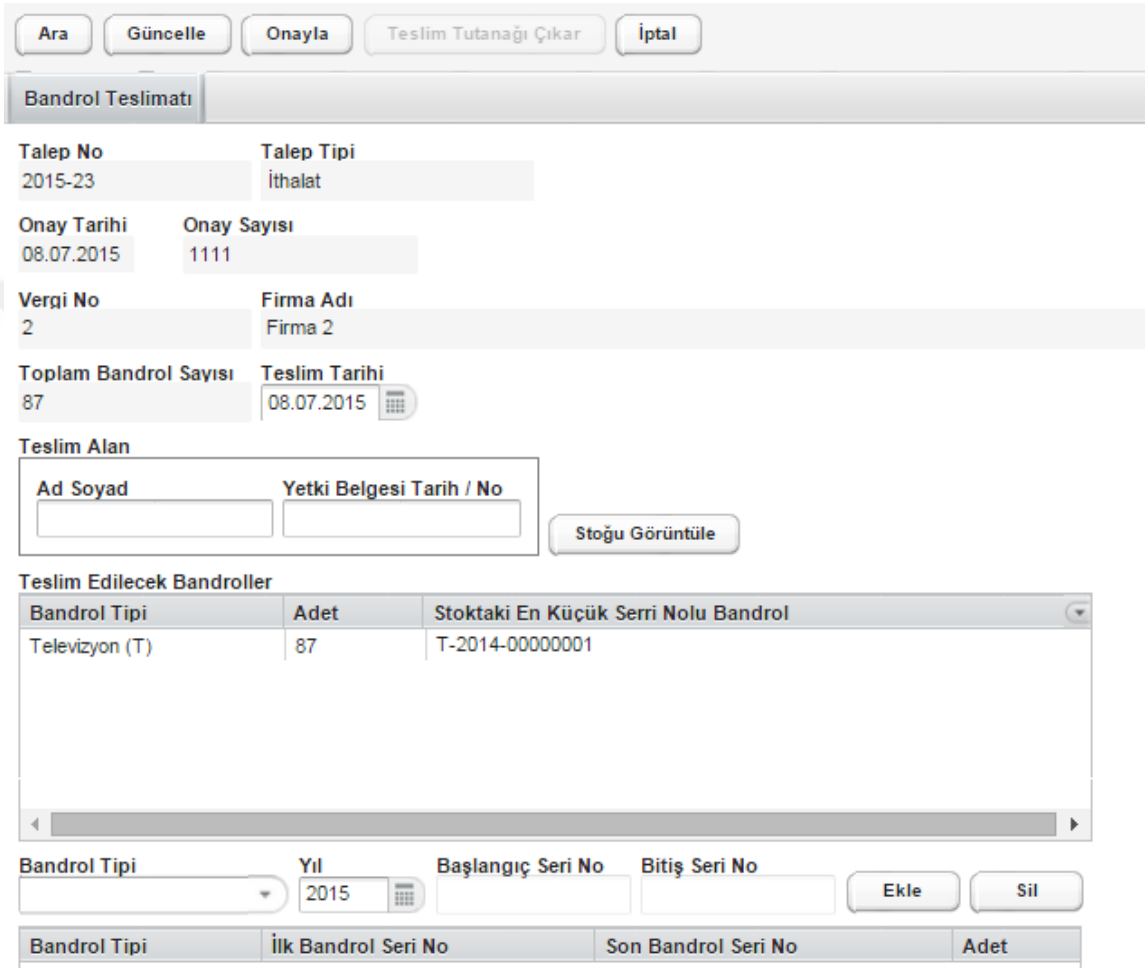


Şekil 29. Bandrol talep işlemleri kullanıcı ekranı

3093 Sayılı TRT Gelirleri Kanunu'na göre, TV ve radyo yayınlarını alabilme özelliği gösteren cihazları imal veya ithal eden firmalar, bu cihazları satmadan önce her bir cihaz için bandrol almakla yükümlüdürler. Firmaların bandrol talepleri bu ekran üzerinden sisteme girilir ve takip edilir. Talep işlemleri ekranına menüden Bandrol Yönetimi ▸ Talep İşlemleri seçilerek ulaşılır.

- Bandrol almak isteyen firma gerekli evraklar ile TRT'ye başvurduğunda, ilgili firmadan sorumlu kullanıcı Şekil 29'daki ekrandan *Yeni Kayıt* butonuna tıklayarak işlemi başlatır.
- Kullanıcı bandrol talebi yapan firmayı firma seçim bileşeninden seçer.
- Firma seçildiğinde firmanın nakit ve teminat hesap bakiyeleri Hesap Bilgileri panelinde görüntülenir.
- Kullanıcı firmanın talep için tercih ettiği ödeme şeklini belirtir. Ödeme şekli *peşin* ya da *teminatlı* olabilir. Peşin alımlar firmanın nakit bakiyesinden karşılanırken teminatlı alımlar için teminat hesabına bandrol ücreti kadar blokaj konur. Bu blokaj, daha sonra bandrollerin satış beyanları tahakkuk ettirilerek hesaplanan alacaklar ödendikçe kaldırılacaktır.
- Kullanıcı talep tipini seçer. Talep tipi *ithalat* veya *imalat* olabilir.
- Kullanıcı bandrollerin teslim alınacağı il müdürlüğünü seçer.
- Kullanıcı *Kaydet* butonuna tıklayarak talebi sisteme kaydeder.

3.1.2. Teslimat İşlemleri Arayüzü



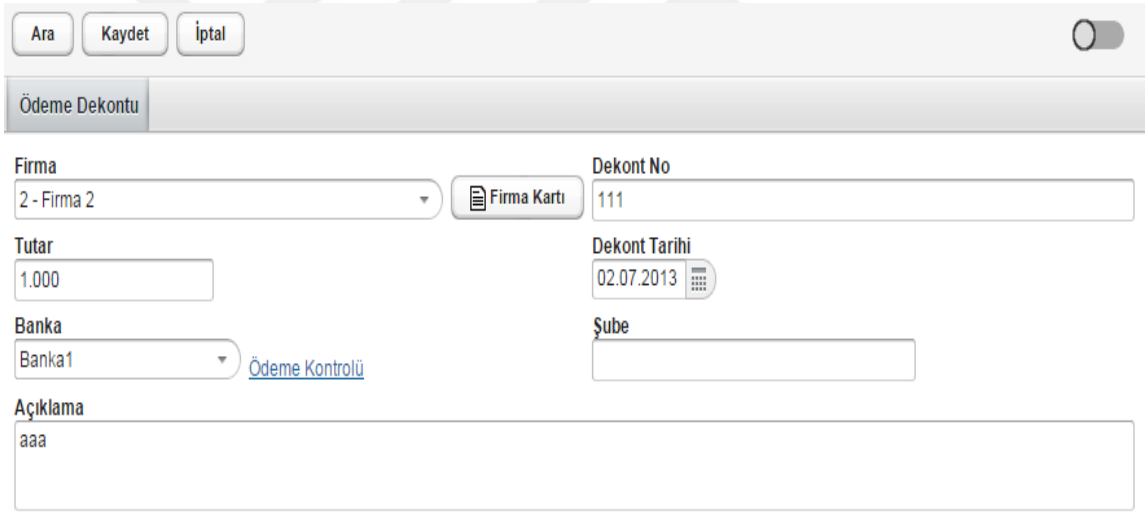
Şekil 30. Teslimat işlemleri kullanıcı ekranı

Teslimatı yapacak kullanıcı Şekil 30'daki "Teslimat İşlemleri" ekranına gelerek:

- *Ara* tuşuna tıklar ve açılan sorgu penceresinde teslimatını yapacağı talebi bulmaya yönelik sorgu parametrelerini girer.
- Sol tarafta, sorgu sonucunda listelenen taleplerden uygun olanı seçer.
- *Düzenle* tuşunu tıklayarak düzenleme moduna geçer.

- Teslim tarihini girer.
- Teslim alan kişinin adını, soyadını ve yetki belgesinin tarihini ve nosunu girer.
- Teslim edeceği bandrollerin tipini, başlangıç ve bitiş sıra nolarını girer.
- Düzenlemelerini tamamladıktan sonra *Onayla* tuşuna tıklayarak teslimat kaydını onaylar.

3.1.3. Ödeme İşlemleri Arayüzü



Şekil 31. Ödeme işlemleri kullanıcı ekranı

Firmaların nakit ödemelerinin takibi ödeme işlemleri arayüzünden yapılmaktadır. Kullanıcı dekont girişi yaparken Şekil 31'deki adımları izlemektedir.

Kullanıcı Ödeme dekontu ekranından ödemeyi yapan firmayı seçer. Ödemeye ilişkin dekontun tutarını, numarasını, tarihini, hangi bankanın hangi şubesine yatırıldığının açıklamasını girer.

Önceden kaydedilmiş ödeme dekontlarını aramak için kullanıcı *ara* tuşuna tıklayarak sorgu kriterlerini girebileceği arama penceresini açar. Bu pencerede yer alan

arama kriterlerinden uygun olanlarını seçerek arama işlemini başlatır. Sistem, arama kriterlerine uyan ödeme dekontlarını soldaki listede görüntüler.

3.2. PERFORMANS ANALİZİNDE KULLANILAN ARAÇLAR

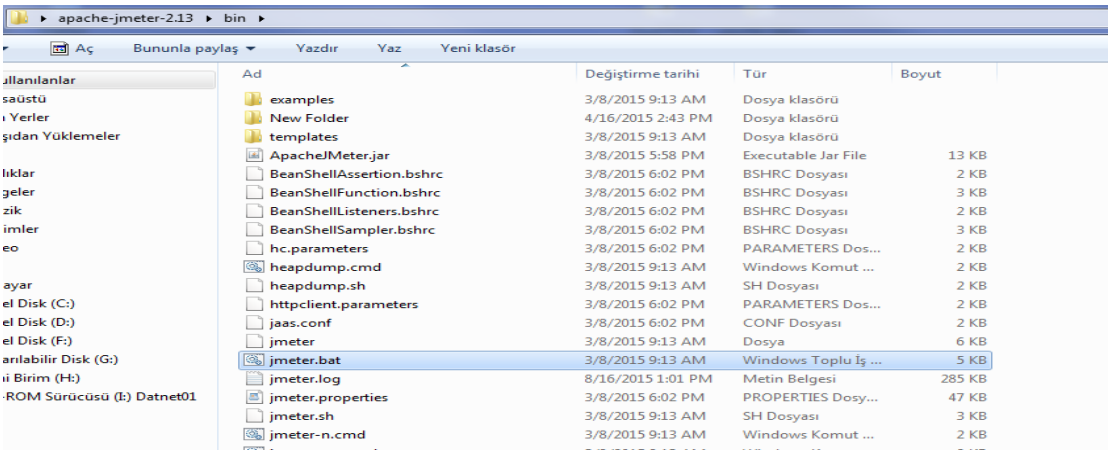
Bu aşamada performans testleri için kullanılan araçlar tanıtılmıştır. Araçların nasıl kullanıldığı gösterilmiştir. Performans testleri karşılaştırmak için iki araç kullanılmıştır.

- Jmeter
- Speedy Framework

3.2.1. Jmeter

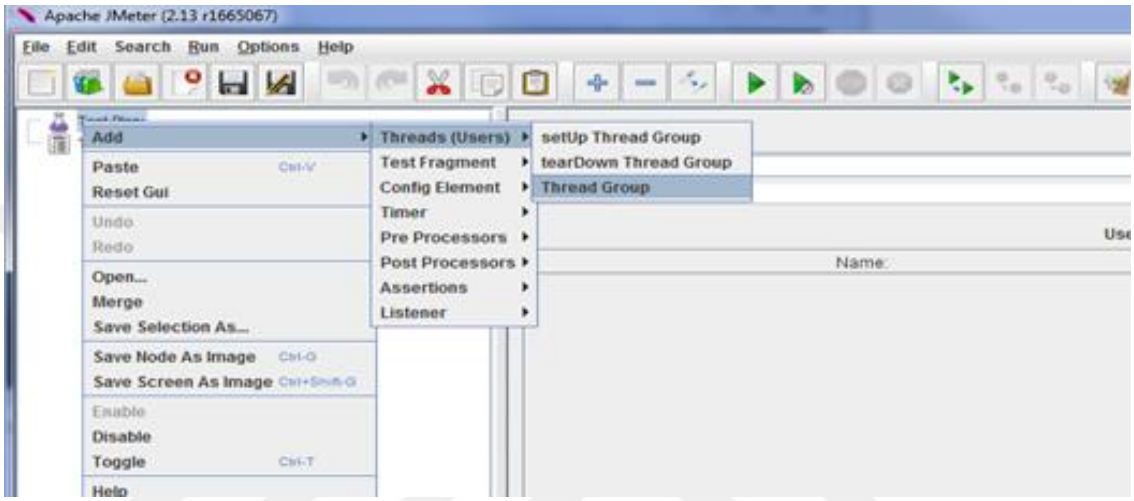
Jmeter test ve performans ölçümü yapabilen, ölçümünü yaptığı sonuçları tablo ve grafik halinde sunabilen bir yazılımdır. Bu çalışmada tasarım deseni kullanılarak geliştirilen yazılımla tasarım deseni kullanılmadan geliştirilen yazılımın performans farkını ortaya koymak için kullanılmıştır.

Test senaryosu hazırlamak için bin klasörü altındaki Şekil 32'deki jmeter.bat dosyası çalıştırılır.



Şekil 32. Jmeter .bat dosya dizini

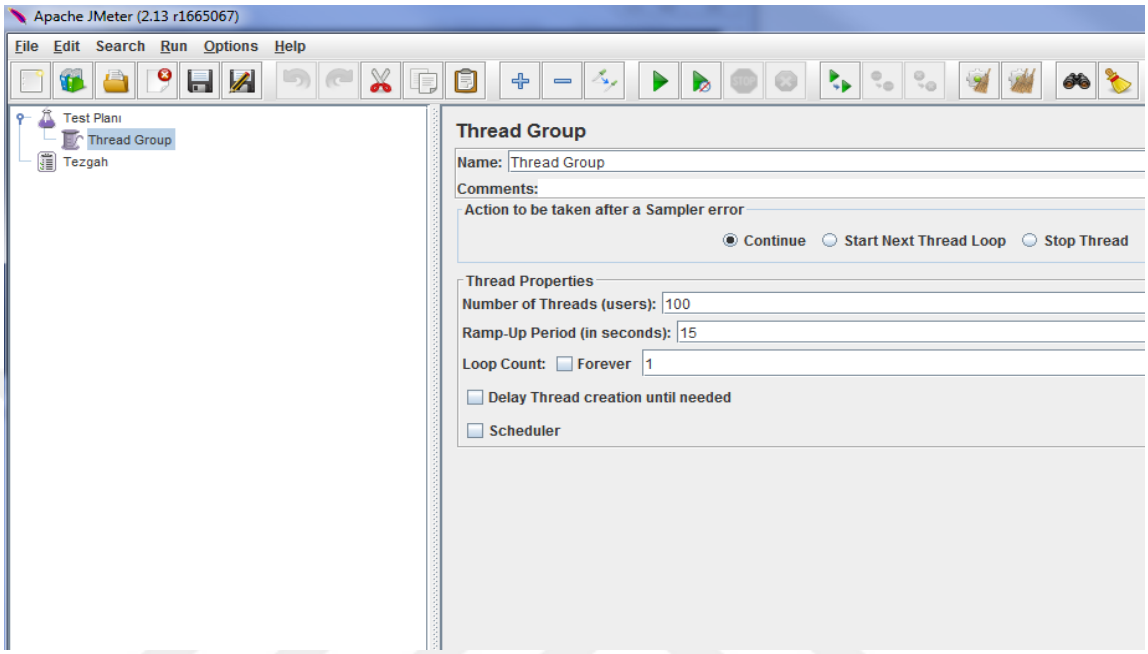
Jmeter.bat dosyası çalıştırıldıktan sonra test planı ekranı gelmektedir. Kullanıcıları belirlemek için Şekil 33'teki Thread Group eklenir.



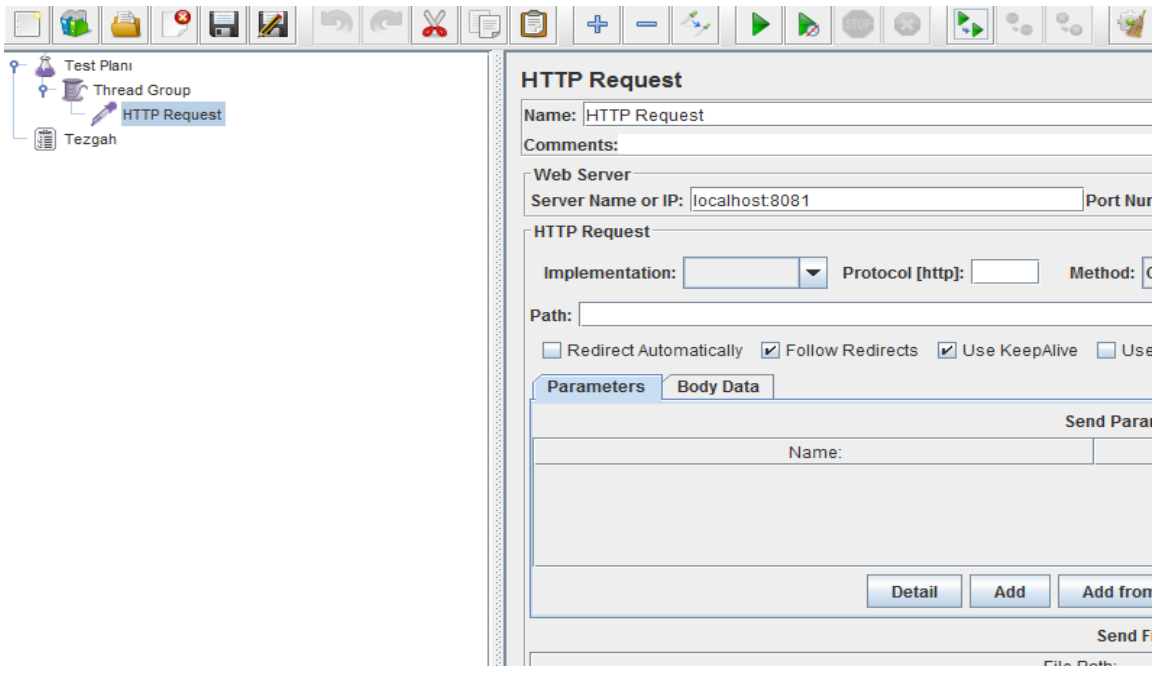
Şekil 33. Thread Group ekleme görünümü

Thread Group ekranında kullanıcı sayısını ve kullanıcı yükünün ne kadar sürede artarak devam edeceği belirtilir. Şekil 33'te Thread Group nasıl kullanıldığı gösterilmiştir

Test edilecek URL'yi belirlemek için sampler menüsünden, HTTP (Hyper text transfer protocol) Request eklenir. Şekil 35'de uygulamanın çalıştığı serverin ipsinin nasıl tanımlandığı gösterilmiştir.



Şekil 34. Thread Group görünümü

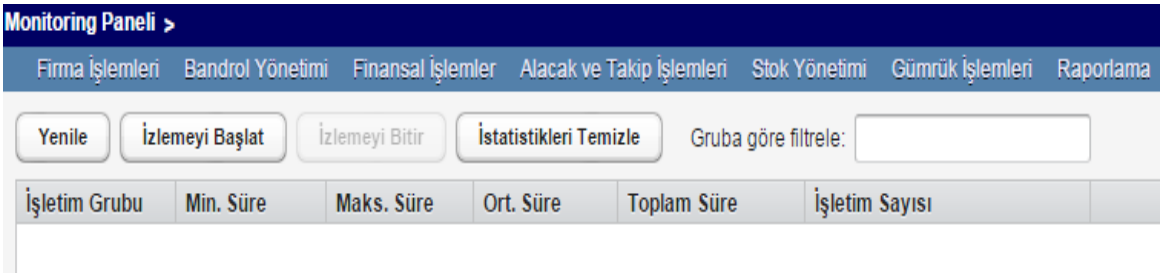


Şekil 35. HTTP Request ekleme görünümü

3.2.2. Speedy Framework

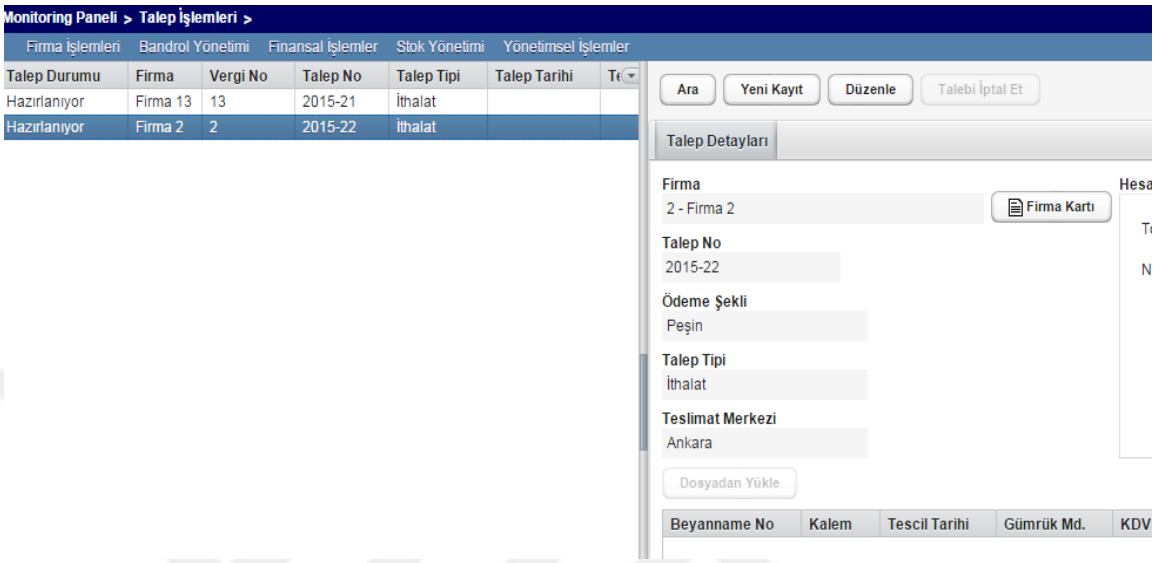
Web tabanlı kurumsal uygulamalar geliştirmek için kullanılan model güdümlü yazılım geliştirmeye yönelik bir framework'tur. Speedy Framework, birçok altyapısal servisin yanı sıra çeşitli katmanlarda performans ölçümü yapmaya yönelik mekanizmalar sunmaktadır. Speedy-Monitoring modülü sayesinde request, event veya metot düzeyinde performans ölçümü yapılabilmektedir.

Speedy Framework'u tasarım deseni kullanılarak geliştirilen yazılımda çalıştırılmıştır. Fakat tasarım deseni kullanılmadan geliştirilen yazılımın yeni entegrasyonlara cevap vermemesi nedeni ile yazılımın kodu üzerinde değişik yapılmasına rağmen Speedy Framework çalıştırılamamıştır. Speedy framework tasarım deseni kullanılarak geliştirilen yazılımda eventlerde geçen süre için kullanılmıştır. Jmeter'ın tasarım deseni kullanılarak geliştirilen yazılım için elde ettiği verilerin kontrolü için kullanılmıştır. Şekil 36'daki monitöring panelinde izlemeyi başlat butonuna basarak performans aracı başlatılmıştır.



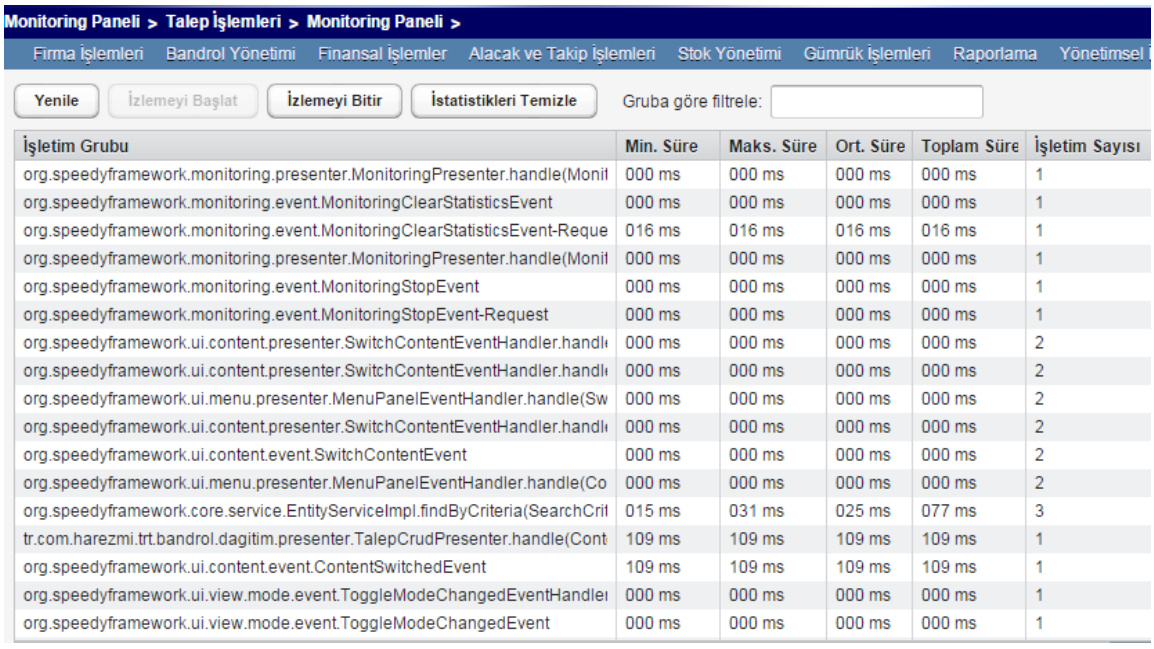
Şekil 36. Monitöring ekranı

Şekil 37'de test yapmak isteğimiz bölüm seçilmiştir.



Şekil 37. Monitöring test başlatma ekranı

Bu işlemleri yaptıktan sonra tekrar Şekil 38'deki monitoring panelinde *izlemeyi bitir* butonuna tıklanır. Böylece kıyaslama işleminde kullanacağımız veri elde edilmiş olmaktadır.



Şekil 38. Monitöring test sonucu ekranı

3.3. İKİ UYGULAMANIN KARŞILAŞTIRLMASI

İki uygulama Jmeter ve Speedy Framework kullanılarak; belirlenen üç tane kullanım senaryosunun; bir, on, yirmi, elli, yüz, yüz kırk ve bin tane kullanıcı tarafından aynı işlem yapılarak karşılaştırılmıştır. Karşılaştırma sonucundaki minimum süre, maksimum süre, ortalama süre, toplam süre ve işlem sayısı gibi değerler görüntülenmiştir.

3.3.1. Jmeter Kullanılarak Yapılan Ölçümler

Tablodaki alanlar aşağıdaki gibi tanımlanmıştır.

- Kullanıcı Sayısı: Testte kullanılan kullanıcı sayısıdır.
- İşlem Sayısı: Test işleminde kullanılan metot sayısıdır.
- Ortalama: Test işleminde isteklere verilen sürenin ortalamasıdır.
- Med: Testin başlangıcı ile bitimi arasında eşit uzaklıkta olan süredir.
- Min: URL istekleri için geçen en kısa süredir.
- Max: URL istekleri için geçen en uzun süredir.

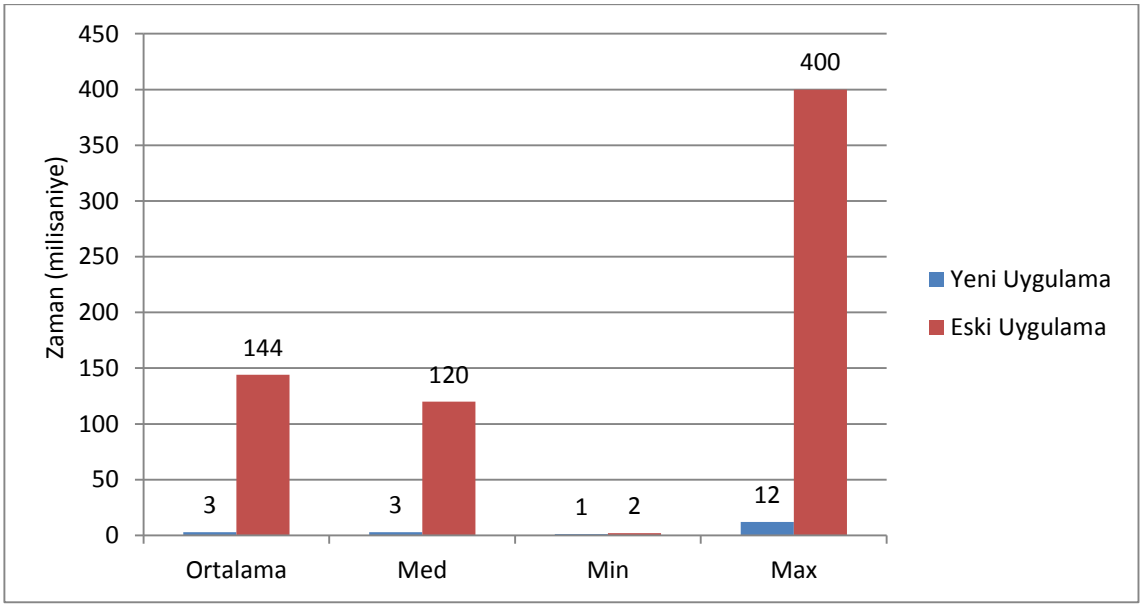
Tablo 1, Tablo 2, Tablo 3 ve Tablo 4'te eski ve yeni uygulamalar farklı kullanım senaryolarındaki kullanıcı sayısı ve işlem sayısı alanlarına göre karşılaştırılmıştır.

Yeni uygulama: Tasarım deseni kullanılarak geliştirilen yazılım

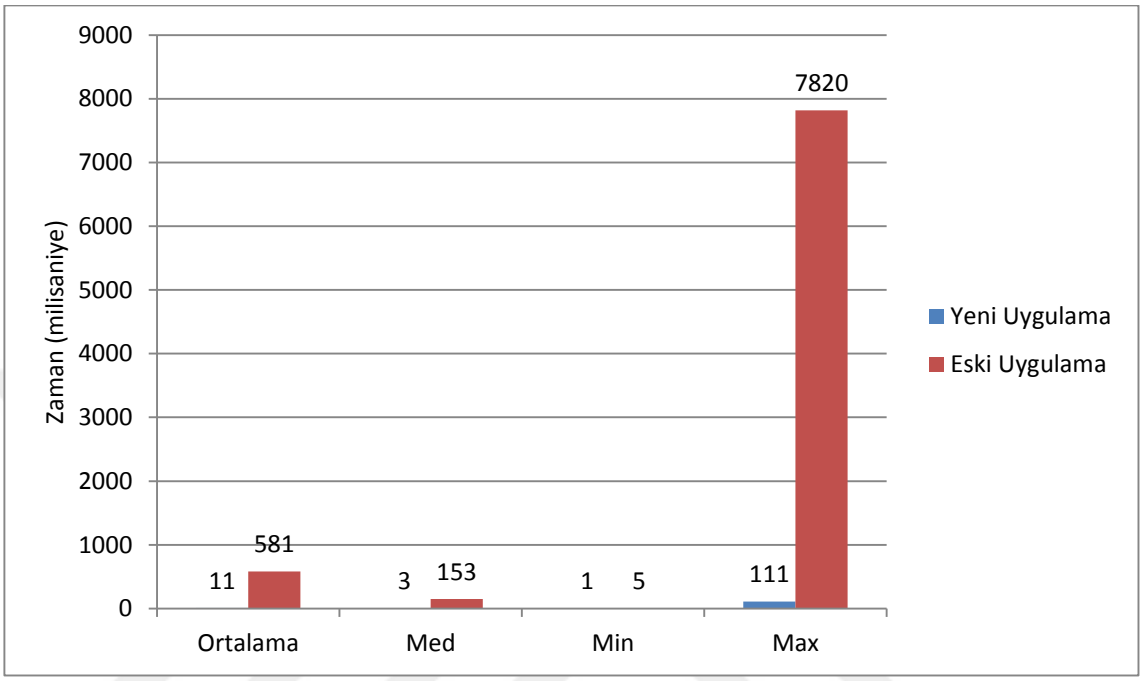
Eski uygulama: Tasarım deseni kullanılmadan geliştirilen yazılım

Tablo 1. Bandrol talep kullanım senaryosu performans analiz verileri

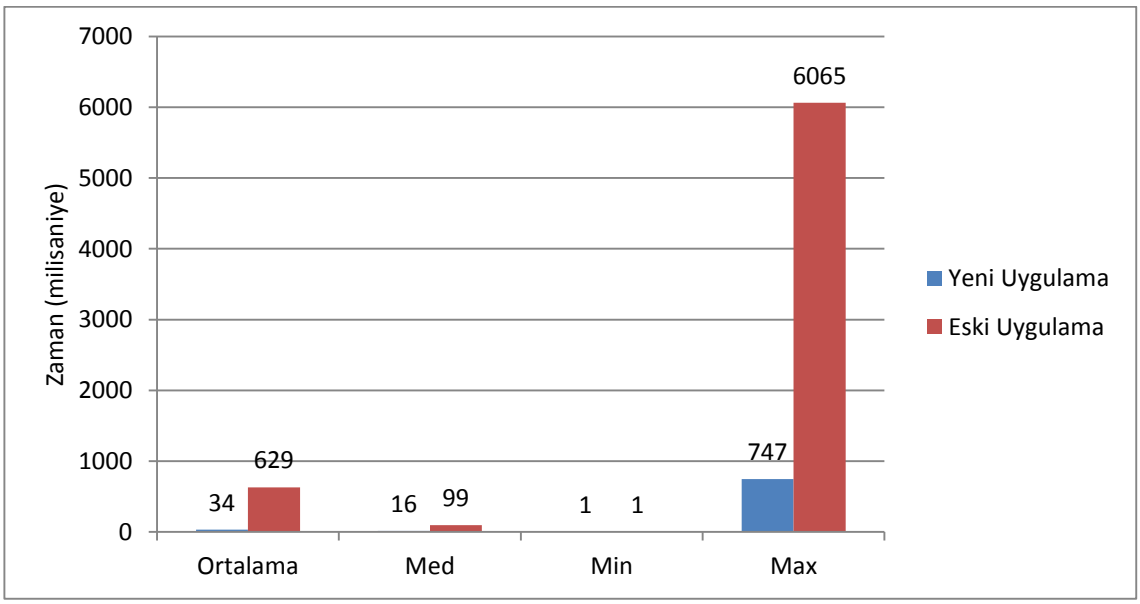
Usecase	Kullanıcı Sayısı	İşlemSayısı
Yeni Uygulama	1	30
Eski Uygulama	1	9
Yeni Uygulama	10	240
Eski Uygulama	10	100
Yeni Uygulama	20	480
Eski Uygulama	20	280
Yeni Uygulama	50	1200
Eski Uygulama	50	250
Yeni Uygulama	100	2200
Eski Uygulama	100	900
Yeni Uygulama	140	3840
Eski Uygulama	140	900
Yeni Uygulama	1000	42000
Eski Uygulama	1000	10000



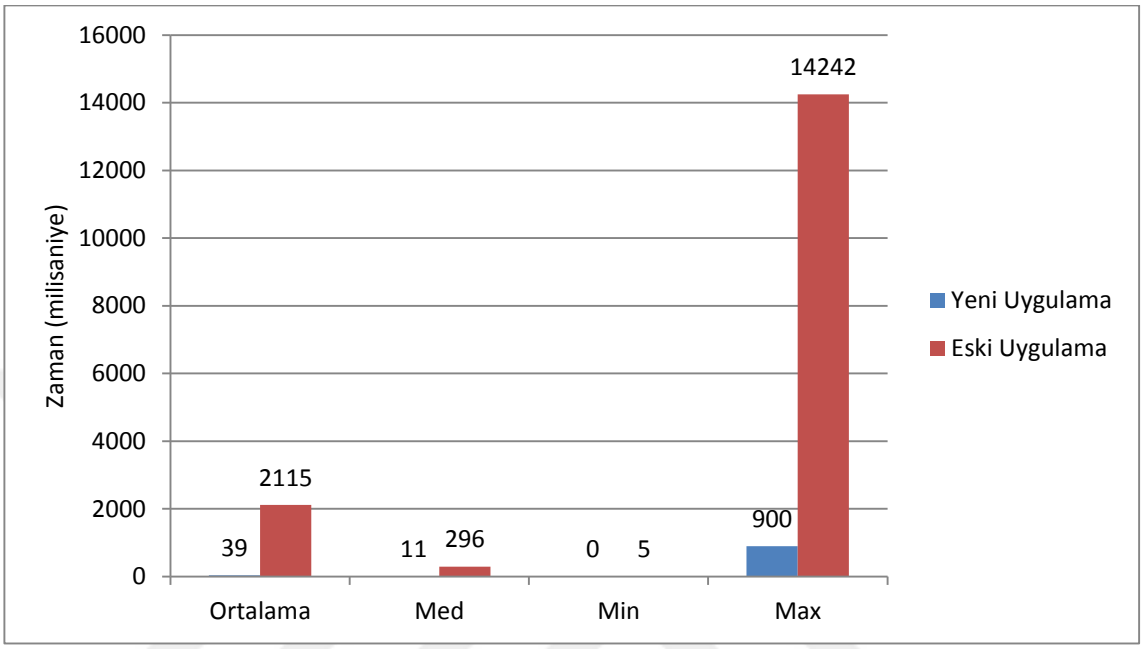
Şekil 39. Bandrol talep işlemleri kullanım senaryosunda 1 kullanıcı kullanılarak elde edilen çalışma zamanı



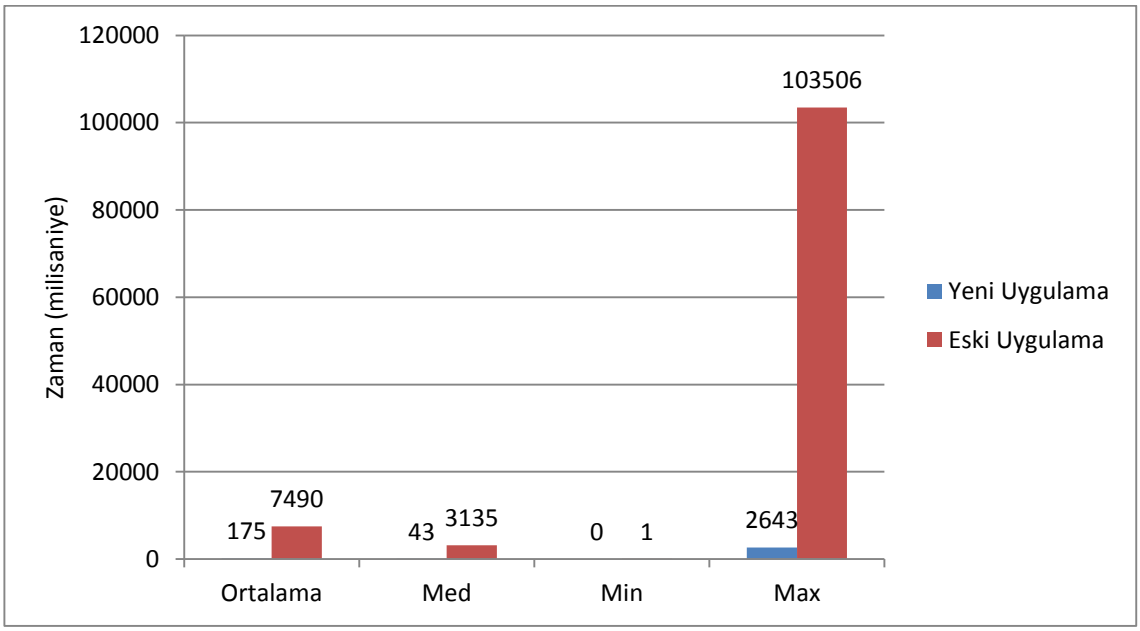
Şekil 40. Bandrol talep işlemleri kullanım senaryosunda 10 kullanıcı kullanılarak elde edilen çalışma zamanı



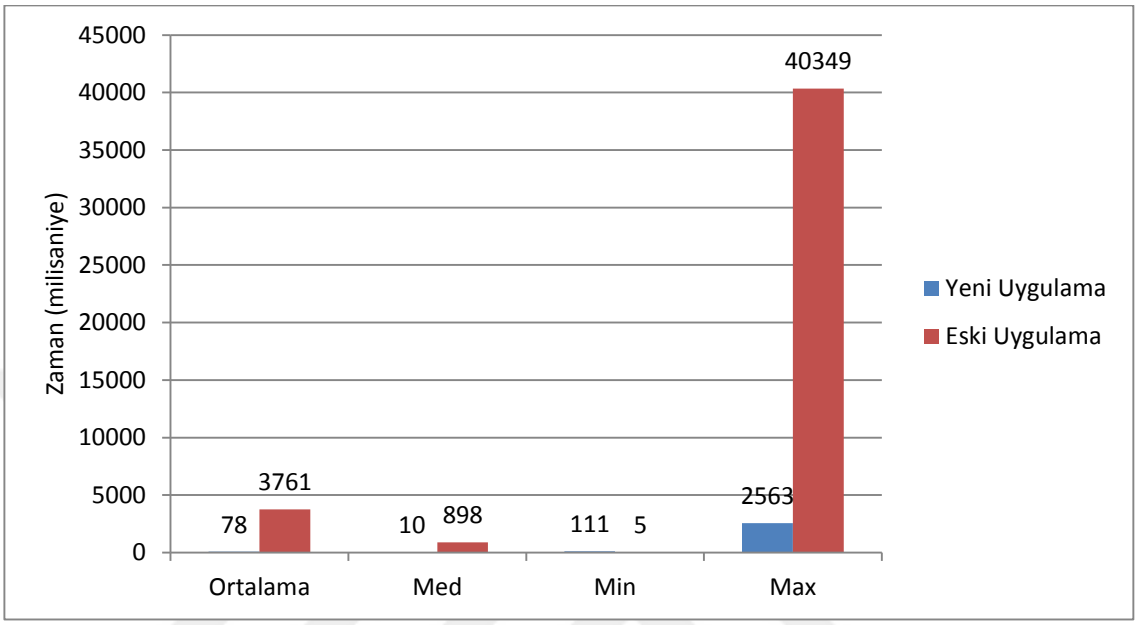
Şekil 41. Bandrol talep işlemleri kullanım senaryosunda 20 kullanıcı kullanılarak elde edilen çalışma zamanı



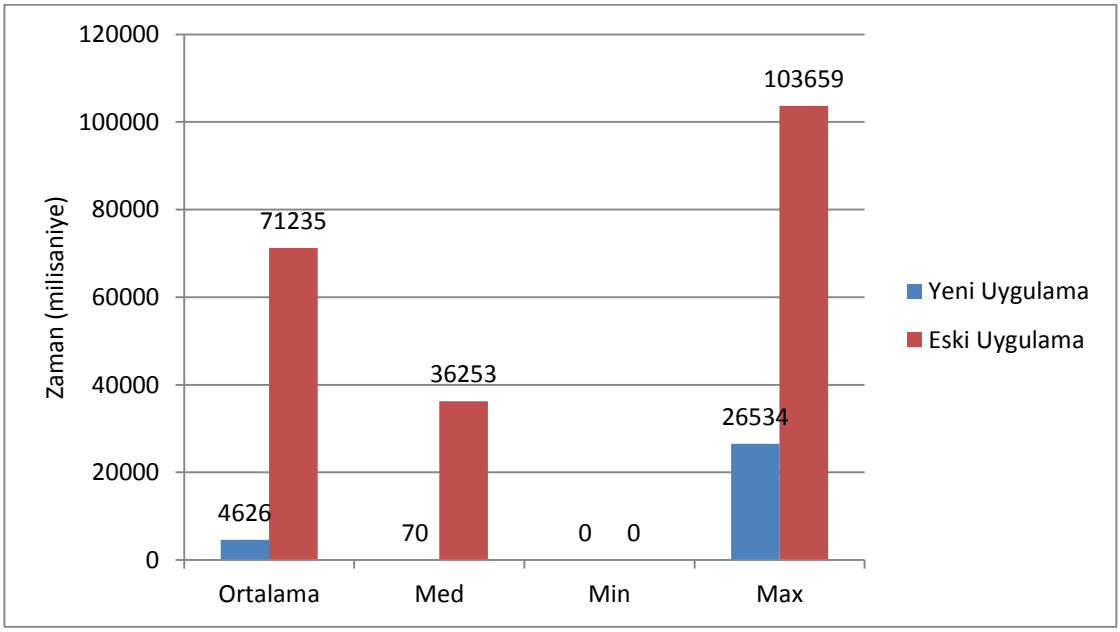
Şekil 42. Bandrol talep işlemleri kullanım senaryosunda 50 kullanıcı kullanılarak elde edilen çalışma zamanı



Şekil 43. Bandrol talep işlemleri kullanım senaryosunda 100 kullanıcı kullanılarak elde edilen çalışma zamanı



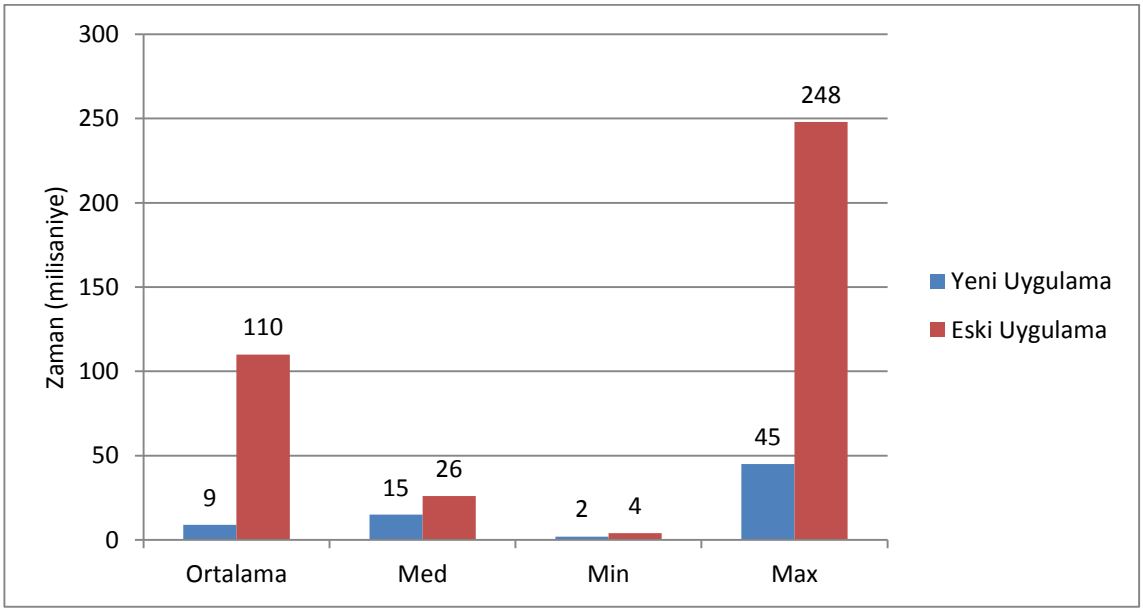
Şekil 44. Bandrol talep işlemleri kullanım senaryosunda 140 kullanıcı kullanılarak elde edilen çalışma zamanı



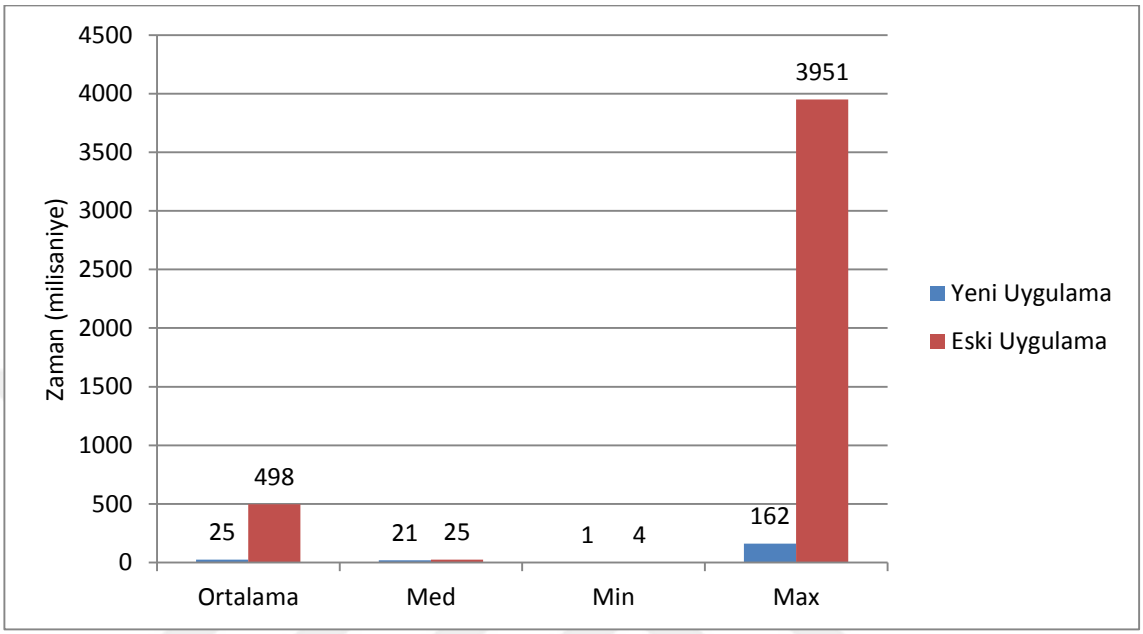
Şekil 45. Bandrol talep işlemleri kullanım senaryosunda 1000 kullanıcı kullanılarak elde edilen çalışma zamanı

Tablo 2. Teslimat işlemleri kullanım senaryosu performans analiz verileri

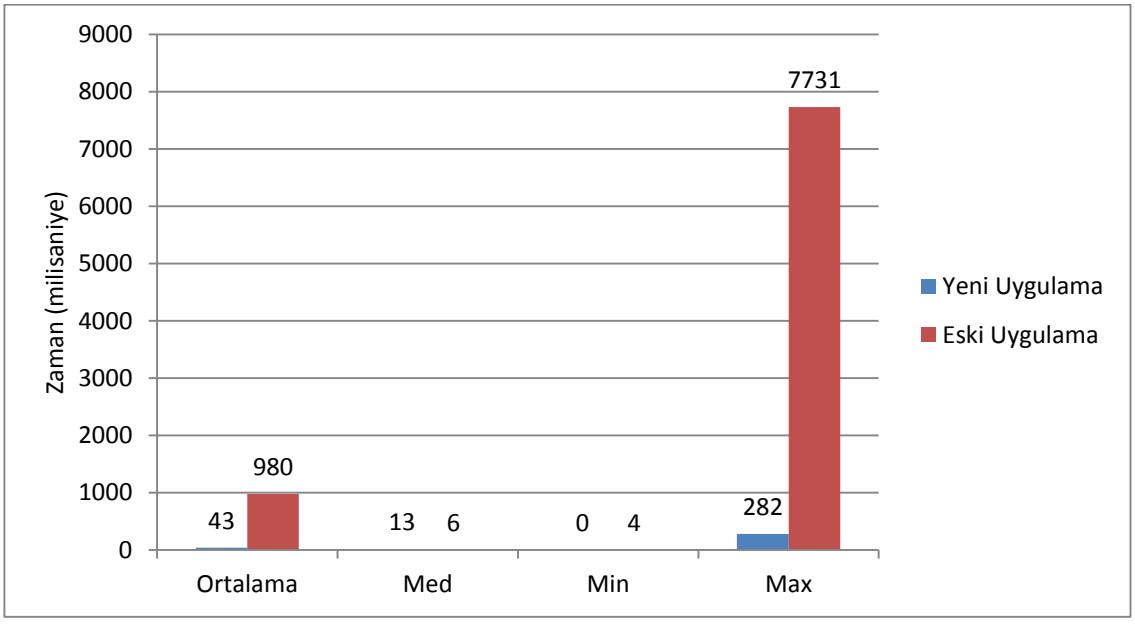
Usecase	Kullanıcı Sayısı	İşlemSayısı
Yeni Uygulama	1	15
Eski Uygulama	1	7
Yeni Uygulama	10	160
Eski Uygulama	10	50
Yeni Uygulama	20	280
Eski Uygulama	20	100
Yeni Uygulama	50	650
Eski Uygulama	50	250
Yeni Uygulama	100	1400
Eski Uygulama	100	500
Yeni Uygulama	140	1820
Eski Uygulama	140	700
Yeni Uygulama	1000	13000
Eski Uygulama	1000	8000



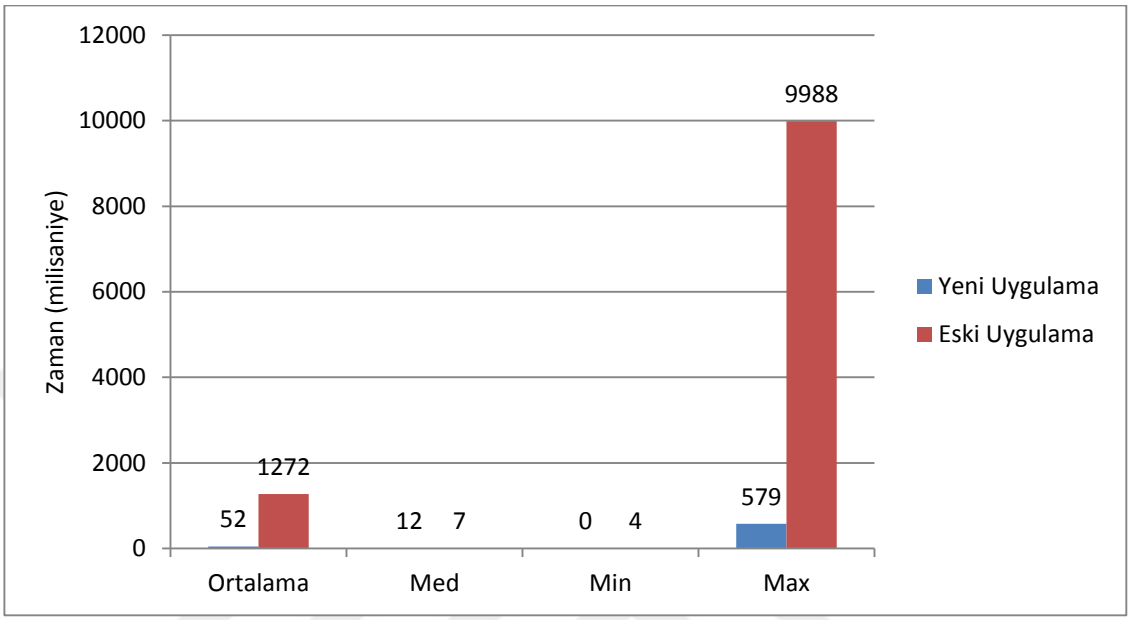
Şekil 46. Teslimat işlemleri kullanım senaryosunda 1 kullanıcı kullanılarak elde edilen çalışma zamanı



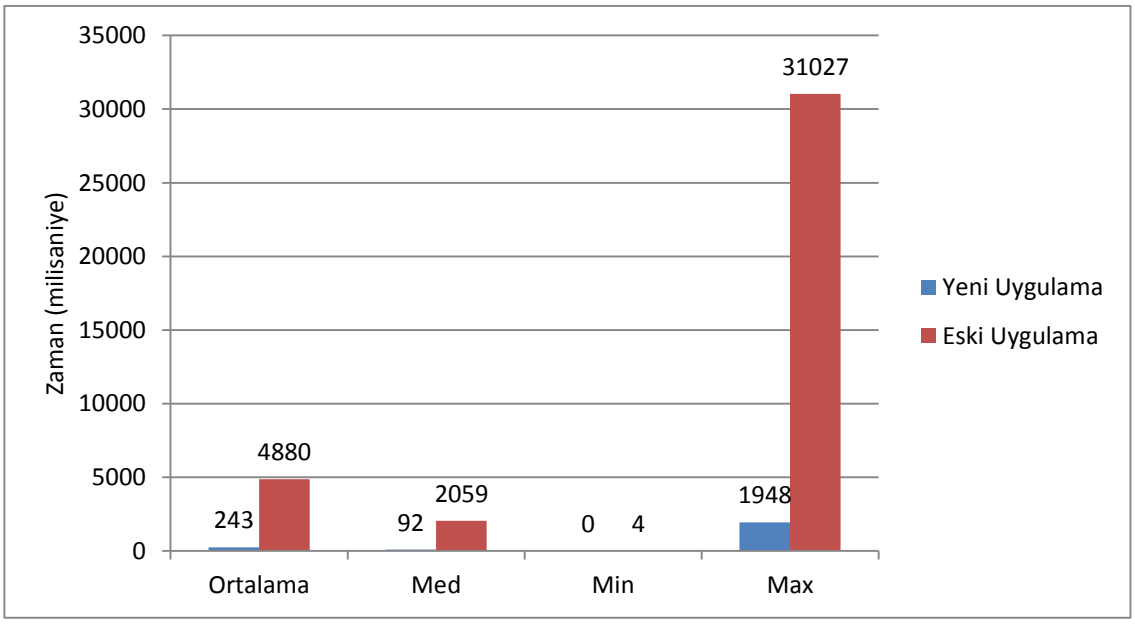
Şekil 47. Teslimat işlemleri kullanım senaryosunda 10 kullanıcı kullanılarak elde edilen çalışma zamanı



Şekil 48. Teslimat işlemleri kullanım senaryosunda 20 kullanıcı kullanılarak elde edilen çalışma zamanı



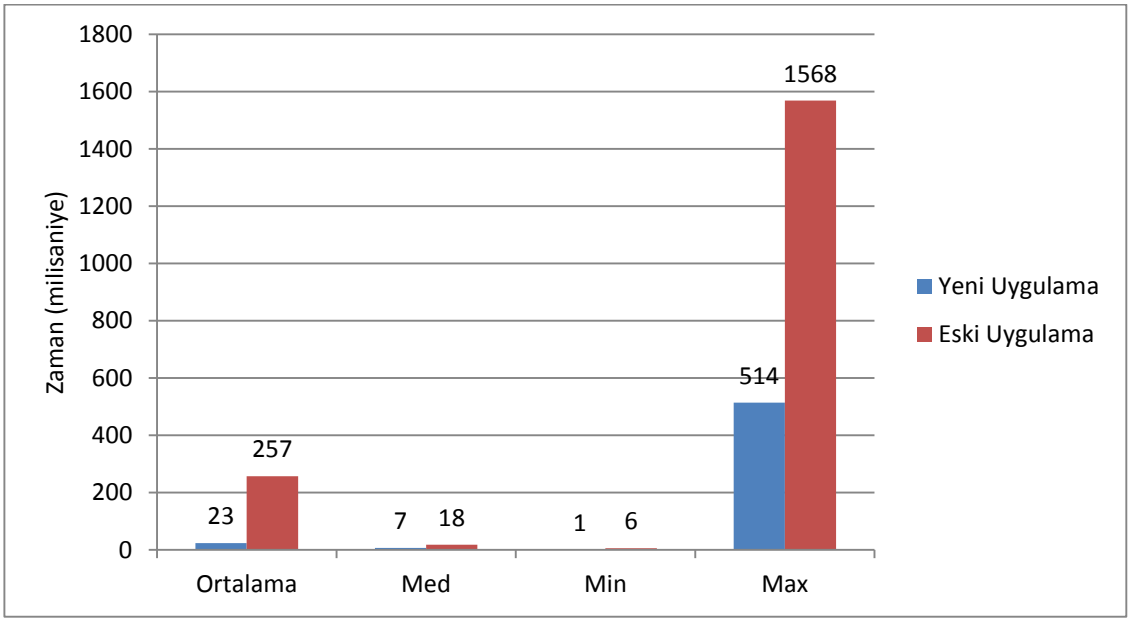
Şekil 49. Teslimat işlemleri kullanım senaryosunda 50 kullanıcı kullanılarak elde edilen çalışma zamanı



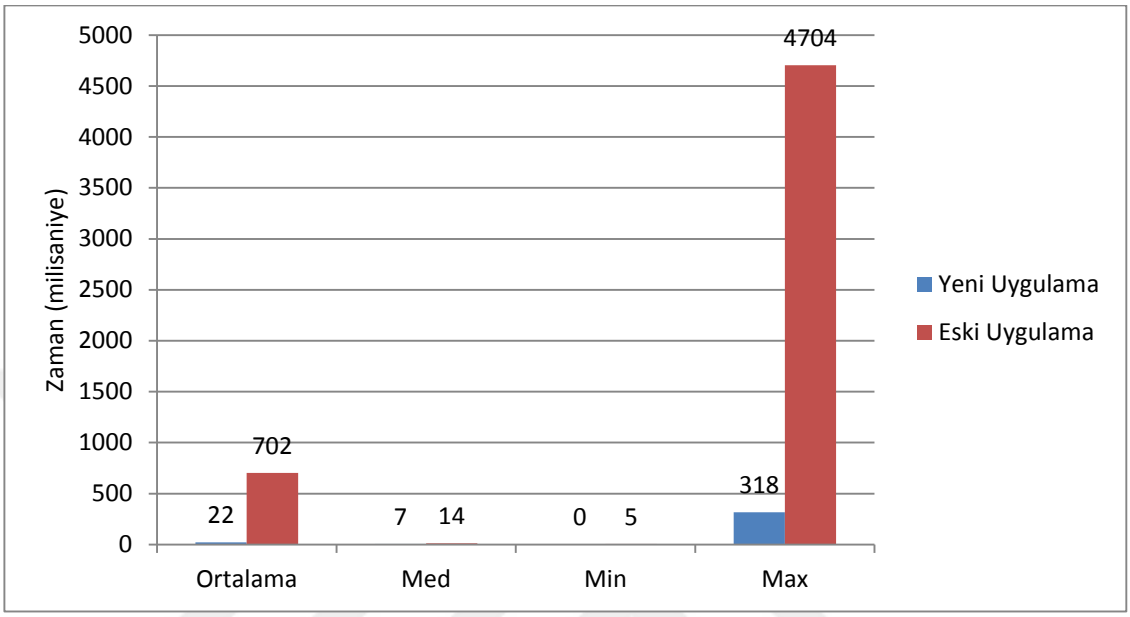
Şekil 50. Teslimat işlemleri kullanım senaryosunda 100 kullanıcı kullanılarak elde edilen çalışma zamanı

Tablo 3. Ödeme işlemleri kullanım senaryosu performans analiz verileri

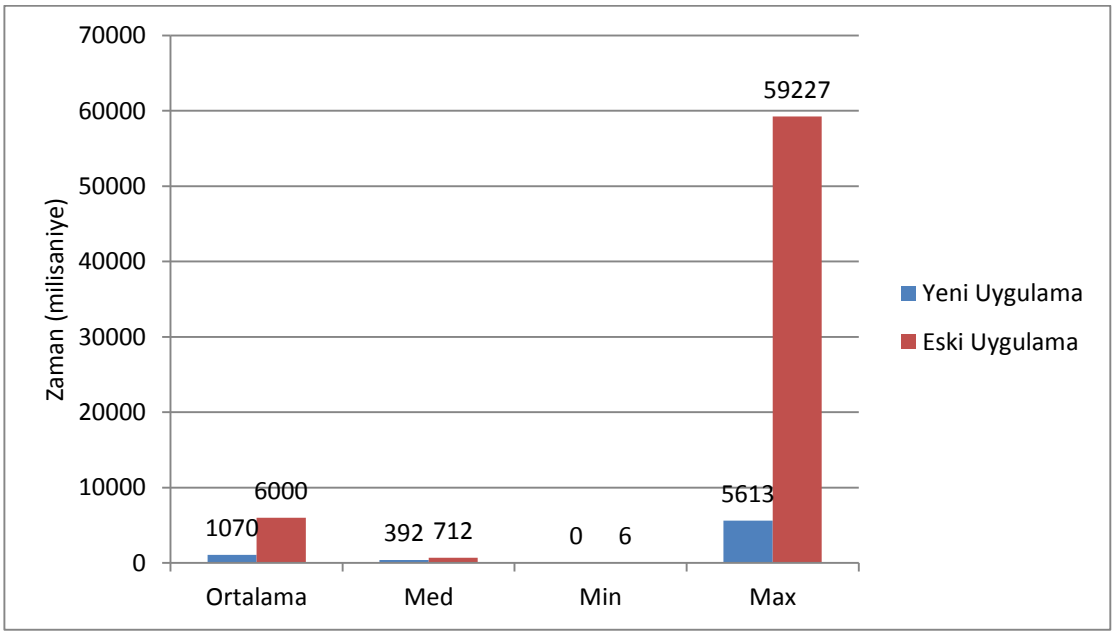
Usecase	Kullanıcı Sayısı	İşlemSayısı
Yeni Uygulama	1	11
Eski Uygulama	1	3
Yeni Uygulama	10	220
Eski Uygulama	10	30
Yeni Uygulama	20	260
Eski Uygulama	20	60
Yeni Uygulama	50	650
Eski Uygulama	50	150
Yeni Uygulama	100	1400
Eski Uygulama	100	300
Yeni Uygulama	140	1820
Eski Uygulama	140	420
Yeni Uygulama	1000	14000
Eski Uygulama	1000	3000



Şekil 51. Ödeme işlemleri kullanım senaryosunda 10 kullanıcı kullanılarak elde edilen çalışma zamanı



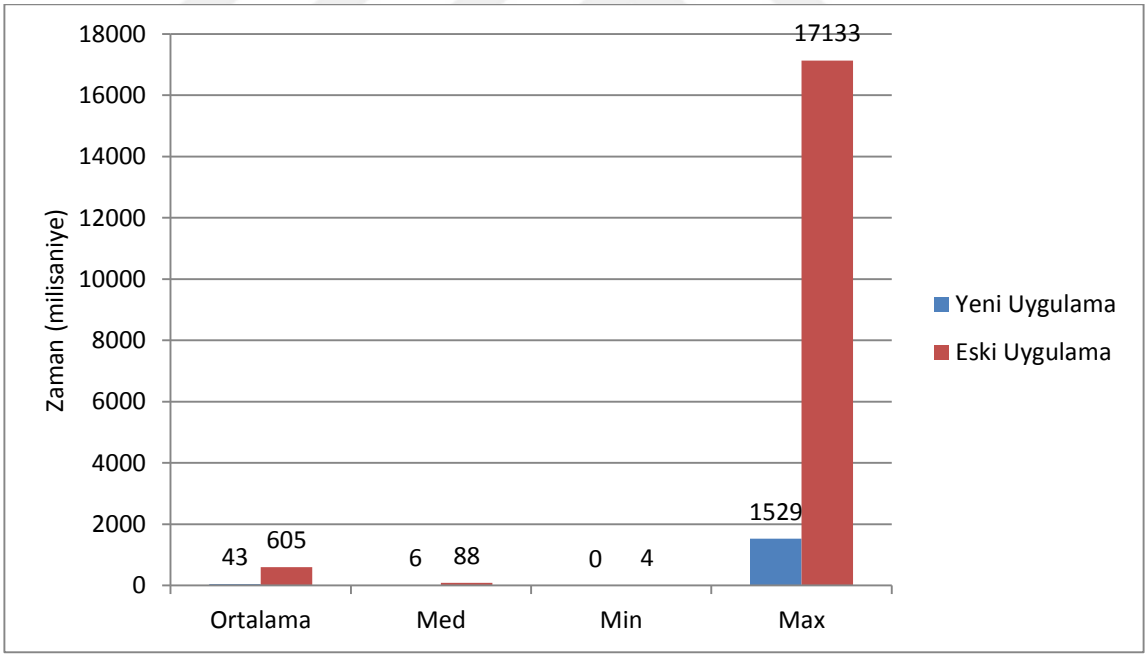
Şekil 52. Ödeme işlemleri kullanım senaryosunda 50 kullanıcı kullanılarak elde edilen çalışma zamanı



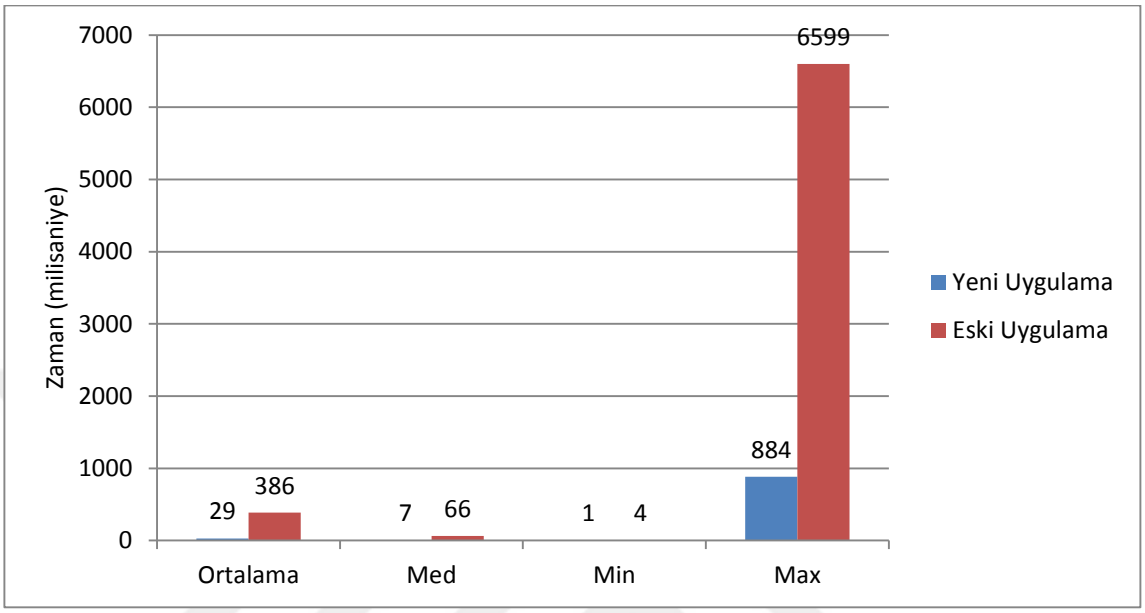
Şekil 53. Ödeme işlemleri kullanım senaryosunda 1000 kullanıcı kullanılarak elde edilen çalışma zamanı

Tablo 4. Üç kullanım senaryosunun beraber olduğu performans analiz verileri

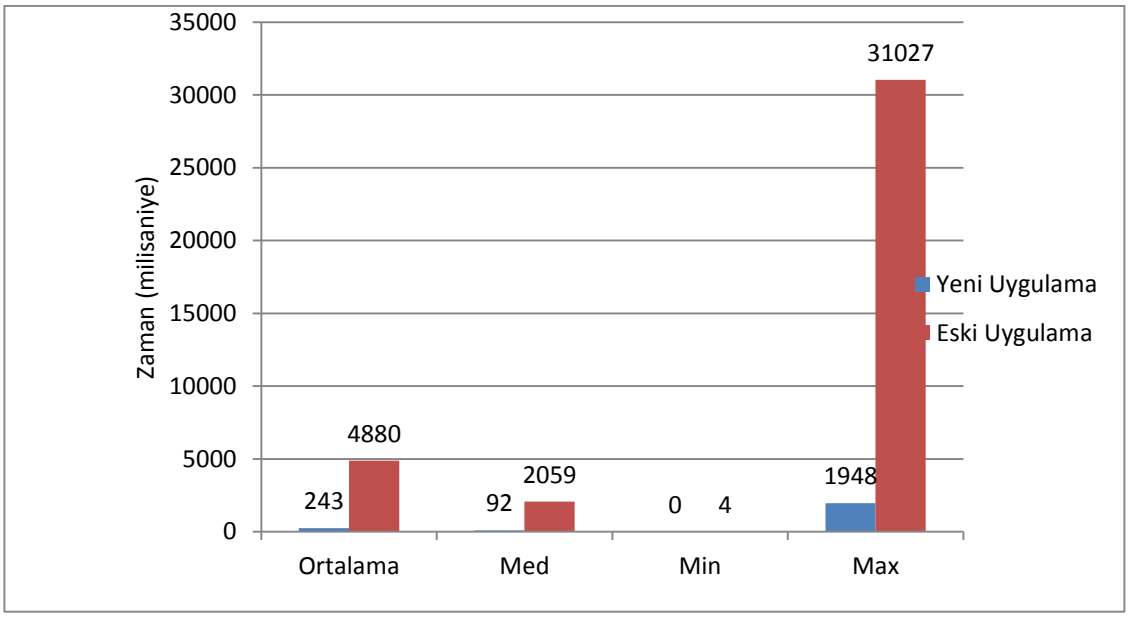
Usecase	Kullanıcı Sayısı	İşlemSayısı
Yeni Uygulama	1	36
Eski Uygulama	1	13
Yeni Uygulama	10	500
Eski Uygulama	10	130
Yeni Uygulama	40	2000
Eski Uygulama	40	520



Şekil 54. Üç kullanım senaryosunda aynı anda 1 kullanıcı kullanılarak elde edilen çalışma zamanı



Şekil 55. Üç kullanım senaryosunda aynı anda 10 kullanıcı kullanılarak elde edilen çalışma zamanı



Şekil 56. Üç kullanım senaryosunda aynı anda 40 kullanıcı kullanılarak elde edilen çalışma zamanı

Tasarım deseni kullanarak geliştirilen yazılımı, tasarım deseni kullanılmadan geliştirilen yazılımla karşılaştırmak için üç kulanım senaryosu kullanılmıştır. Bu üç kullanım senaryosunda kullanıcı sayısı değiştirilerek yapılan testlerde çalışma zamanı alanı üzerinden iki yazılım karşılaştırılmıştır.

Bandrol talep işlemleri senaryosu ("Şekil 39-45 arasındaki şekillerde") farklı kullanıcı sayıları (1, 10, 20, 50, 100, 140, 1000) kullanılarak test edilmiştir. Bu testler sonucu elde edilen ortalama ve ortanca çalışma zamanına göre tasarım deseni kullanarak geliştirilen yazılımın, tasarım deseni kullanılmadan geliştirilen yazılıma göre performans ölçümlerinin daha iyi olduğu tespit edilmiştir. Ayrıca minimum ve maksimum alanlarının çalışma zamanları karşılaştırıldığında ise minimum alanında fark olmamasına rağmen maksimum alanındaki fark oldukça fazladır. Bu fark uzun süren işlemlerde tasarım deseni kullanılarak geliştirilen yazılımın, tasarım deseni kullanılmadan geliştirilen yazılıma göre performansının çok iyi olduğunu göstermiştir.

Teslimat işlemleri senaryosu ("Şekil 46-50 arasındaki şekillerde") farklı kullanıcı sayıları (1, 10, 50, 100) kullanılarak test edilmiştir. Bu testler sonucu elde edilen ortalama ve ortanca çalışma zamanına göre tasarım deseni kullanarak geliştirilen yazılımın, tasarım deseni kullanılmadan geliştirilen yazılıma göre performans ölçümlerinin daha iyi olduğu tespit edilmiştir. Ayrıca minimum ve maksimum alanlarının çalışma zamanları karşılaştırılmasında ise minimum alanında fark olmamasına rağmen maksimum alanındaki oldukça fazladır. Bu fark uzun süren işlemlerde tasarım deseni kullanılarak geliştirilen yazılımın, tasarım deseni kullanılmadan geliştirilen yazılıma göre performansının çok iyi olduğunu göstermiştir.

Ödeme işlemleri senaryosu ("Şekil 51-53 arasındaki şekillerde") farklı kullanıcı sayıları (10, 50, 1000) kullanılarak test edilmiştir. Bu testler sonucu elde edilen ortalama ve ortanca çalışma zamanına göre tasarım deseni kullanarak geliştirilen yazılımın, tasarım deseni kullanılmadan geliştirilen yazılıma göre performans ölçümlerinin daha iyi olduğu tespit edilmiştir.

Üç kullanım senaryosunun aynı anda ("Şekil 54-56 arasındaki şekillerde") farklı kullanıcı sayılarında (1, 10, 40) kullanıldığı testler de yapılmıştır. Bu testler sonucu elde edilen ortalama ve ortanca çalışma zamanına göre tasarım deseni kullanarak geliştirilen yazılımın, tasarım deseni kullanılmadan geliştirilen yazılıma göre performans ölçümlerinin daha iyi olduğu tespit edilmiştir.

Tasarım deseni kullanarak geliştirilen yazılımın, tasarım deseni kullanılmadan geliştirilen yazılıma göre, ("Şekil 39-56 arasındaki şekillerde") ortalama alanları göz önünde bulunarak incelendiğinde üç kullanım senaryosunda da performans ölçümlerinin daha iyi olduğu tespit edilmiştir.

("Tablo 1-4 arasındaki tablolarda") işlem sayısı alanı göz önünde bulunarak incelendiğinde üç kullanım senaryosunda da uygulamadaki kullanıcı sayısı arttırıldığında projedeki geri bildirimlerin daha hızlı olduğu ve kullanıcıların işlem yaparken bekleme sürelerinin daha az olduğu tespit edilmiştir.

Test sayısı, kullanıcı sayısı veya senaryolar değiştirilse bile iki yazılım arasındaki performans ölçümlerinde büyük çapta bir değişiklik olmadığı ve yukarıdaki verilere yakın sonuçlar elde edildiği görülmüştür. Bu performans ölçümlerinde yazılımların farklı teknolojiler kullanılarak geliştirildiği göz önünde bulundurulmalıdır. Fakat kullanılan farklı teknolojilerin direk olarak yazılımların performansına etki etmediği varsayılmıştır. Bu varsayımın nedeni teknolojilerin kullanıcı ekranlarının tasarımlarıyla ilgili olmasıdır.

DÖRDÜNCÜ BÖLÜM
SONUÇ

4.1. PERFORMANS ANALİZİ

Tasarım deseni kullanarak geliştirilen yazılımda ve tasarım deseni kullanılmadan geliştirilen yazılımda yapılan testlerin sonucunda iki yazılımın, farklı senaryolarda işlem sayısı, ortalama süre, ortanca süre, minimum süre ve maksimum süre alanları gözetilerek performans değerlerinin ölçülmesi ve zaman bakımından iki yazılımın karşılaştırılması sağlamıştır. Bu verilere göre tasarım deseni kullanılarak geliştirilen yazılımın yapılan işlemlere çok kısa zamanda cevap vermesi, işlem sayısının fazla olmasına rağmen işlem başına düşen zamanın daha az olduğu görülmüştür. Bu verilerin analizi neticesinde tasarım deseni kullanılarak geliştirilen bir yazılımın performans ölçümlerinin daha iyi olduğu sonucuna varılmıştır.

4.2. KODA BAĞIMLILIĞIN KARŞILAŞTIRILMASI

Bir projeye başlarken projenin gereksinimlerin değişmesinden şikâyet etmek anlamsızdır. Yapılması gereken gereksinimlerdeki değişmelerin önüne geçmekten çok, değişiklikleri daha kolay biçimde ele alıp projede uygulamamızı sağlayacak bir sistem tasarımına sahip olmaya çalışmaktır. Tasarım deseni kullanılmadan geliştirilen yazılımda sonradan ihtiyaç duyulan bazı entegrasyonlar ortaya çıkmıştır. Bu entegrasyonları gerçekleştirmede büyük sorunlar yaşanmıştır. Ancak tasarım deseniyle kullanılarak geliştirilen yazılımda ise hiç sorun yaşamadan entegrasyonlar yapılmıştır.

Tasarım deseni kullanılarak geliştirilen yazılımın, değişen gereksinimlere göre yeniden düzenlenmesi daha kolay olmaktadır. İlk uygulamanın devreden çıkarılarak daha yeni bir uygulama ile değiştirilmesinin başlıca sebeplerinden biri de zaman içerisinde ortaya çıkan değişiklik taleplerinin yerine getirilmesinde karşılaşılan sıkıntılardır. Zaman içerisinde ortaya çıkan yeni gereksinimleri uygulamaya yansıtmak zaman ve efor açısından değerlendirildiğinde oldukça zor olmaktadır. Tasarım deseni

kullanılarak geliştirilen yazılımda ise zaman içerisinde ortaya çıkan birçok gereksinim kolaylıkla yazılıma yansıtılmıştır. Uygulamanın modüler bir yapıda oluşu ve modüller arasındaki bağımlılıkların net bir şekilde tanımlanmış olması bu değişikliklerin yapılmasını oldukça kolaylaştırmıştır. Örneğin tasarım deseni kullanılarak geliştirilen yazılımda, başlangıçta talep edilmeyen ve sonradan ortaya çıkan bir gereksinim olan teslimat esnasında sırasız bir şekilde bandrol dağıtımı senaryosunun önüne geçmek amacıyla hatalı teslimatların tespit edilip yetkili kişilere e-posta yoluyla bildirilmesi gereksinimi, event mekanizması sayesinde yazılan basit bir listener ile kolayca gerçekleştirilmiştir. Ayrıca bu kabiliyetin parametrik bir biçimde açılıp kapatılması yoluyla mevcut işleyiş etkilenmeden devreye alınabilmiştir. Bu örnek tasarım deseni kullanılarak geliştirilen yazılımların kodsal olarak her türlü değişikliğe açık olduğunu ve kod üzerinde değişikliklerin kolayca yapılarak yeni entegrasyonların yazılıma uygulanabildiğini göstermiştir.

KAYNAKÇA

[1] Freeman, E., Freeman, E., Bates, B. ve Sierra, K. (2004). *Head First Design Patterns*, O'Reilly.

[2] Meyer, B. (1997). *Object-Oriented Software Construction*, Prentice Hall.

[3] Bennett, S., McRobb, S. ve Farmer, R. (2010). *Object-Oriented Systems Analysis and Design Using UML*, McGraw-Hill.

[4] Freeman, A. (2015). *Pro Design Patterns in C#*, Apress.

[5] Mills, B. (2009). *Practical Formal Software Engineering: Wanting the Software You Get*, Cambridge University Press.

[6] Nicholson, J., Eden, Amnon H., Gasparis, E. ve Kazman, R. (2014). "Automated verification of design patterns: A case study", *Science of Computer Programming*, 80, 211–222.

[7] Chu, Chu W., Lu, Wei C., Shiu, Peng C. ve He, X. (2000). "Pattern-based software reengineering: a case study", *Journal of Software Maintenance: Research and Practice*, 12, 121–141.

[8] Martin, Robert C. ve Martin, M. (2007). *Agile Priniciples, Patterns, and Practices in C#*, Prentice Hall.

[9] Laporti, V., Borges, Marcos R. ve Braganholo, V. (2009). "Athena: A collaborative approach to requirements elicitation", *Computers in Industry*, 60, 367–380.

[10] Virginia, N. (2013). "A Design Patterns Perspective on Data Structures", *Acta Universitatis Apulensis*, 34, 335-354.

[11] Gamma, E., Helm, R., Johnson R. ve Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley.

[12] Some, Stephane S. (2006). "Supporting use case based requirements engineering", *Information and Software Technology*, 48, 43–58.

[13] Bevis, T., (2012). *Java Design Pattern Essentials*, Ability Firs.

[14] Cooper, James W. (2000). *Java Design Patterns: A Tutorial*, Addison Wesley.

[15] Noborikawa, M. (2003). Refactoring Software Using Design Patterns, Alındığı tarih: 01.09.2015, adres: http://www.cs.uni.edu/~wallingf/miscellaneous/student-papers/noborikawa-paper.pdf.

[16] Gamma, E., Helm, R., Johnson, R. ve Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley.

[17] Parreiras, Fernando S., Staab, S. ve Winter, A. (2008). "Improving Design Patterns by Description Logics: A Use Case with Abstract Factory and Strategy", *Johannes-gutenberg-University.*

[18] Zhang, C. ve Budgen, D. (2012). "What Do We Know about the Effectiveness of Software Design Patterns?", *IEEE Transactions on Software Engineering*, 38, 1213-1231.

[19] Al-Ahmad, W. (2006). "Object-Oriented Design Patterns for Detailed Design", *Journal of 0bject Technology*, 5, 155-169.

[20] Shi, N. ve Olsson, Ronald A. (2006). "Reverse Engineering of Design Patterns from Java Source Code", *Automated Software Engineering*, 21, 123-134.

[21] Mani, N., Dorina , C. ve Woodside , M. (2011). "Studying the Impact of Design Patterns on the Performance Analysis of Service Oriented Architecture", *Software Engineering and Advanced Applications*, 37, 12-19.

[22] Schmidt, Douglas C. (1996). "Experience Using Design Patterns to Develop Reuseable Object-Oriented Communication Software", Washington University.

[23] Reenskaug, T. (2003). "The Model-View-Controller (MVC) Its Past and Present", University of Oslo.

[24] Reenskaug, T. (1978). "The Model-View-Controller (MVC)", Xerox Parc.

[25] Potel, M. (1996). "MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java", Taligent.

[26] Fowler, M. (2002). *Patterns of Enterprise Application Architecture*, Addison Wesley.

[27] Alan Shalloway, A. ve Trott, James R. (2007). *Design Patterns Explained: A New Perspective on Object-Oriented Design*, Addison Wesley.

[28] Eckel, B. (2006). *Thinking in Patterns with Java*, Prentice Hall.

[29] Fowler, M. (2004). *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Addison Wesley.

[30] Larman, C. (2008). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development,* Dorling Kindersley.

[31] Dennis, A., Wixom, Barbara H. ve Tegarden, D. (2015). *Systems Analysis and Design: An Object-Oriented Approach with UML,* Wiley.

[32] Lethbridge, Timothy C. ve Laganiere, R. (2001). *Object-Oriented Software Engineering: Practical Software Development Using UML and Java*, McGraw-Hill.

[33] Nevedrov, D. (2006). Using JMeter to Performance Test Web Services, Alındığı tarih: 01.09.2015, adres: http://loadstorm.com/files/Using-JMeter-to-Performance-Test-Web-Services.pdf

[34] JMeter. Alındığı tarih: 01.09.2015, adres: http://jmeter.apache.org.