Melez Bir Mimari Yaklaşımıyla Yazılım Çercevesi Geliştirme

Osman Karaahmetoğlu[1], Ebru Karaahmetoğlu[2]

okaraahmetoglu@gmail.com[1] , erogluebru@hotmail.com[2]

Özet. Yazılım alanında uygulama geliştirme süreçlerini kolaylaştırmak ve iyileştirmek üzere çeşitli çerçeveler bulunmaktadır. Her bir çerçevenin yazılım süreçlerinde üstün özellikleri olabildiği gibi zayıflıkları da bulunmakta olup, yazılım süreçleri üzerinde olumsuz etkileri olmaktadır. Bazı yazılım çerçeveleri, yazılım süreçlerinde orta katmanda bellek yönetimi, hareket (transaction) yönetimi gibi alanlarda güçlü iken, bazıları ise kullanıcı arabiriminde kullanıcı ile iletişimde ölçeklenebilir, tutarlı kontroller sunmaktadır. Yazılım süreçlerinde optimum beklentiyi sağlamak için bu çerçevelerin etkin özelliklerine kullanarak, kurumsal yazılım mimari ihtiyaçlarını kaşılayacak, bir çerçeve oluşturmak gereklidir. Bu çalışmada orta katmanda bilinen Java tabanlı bir orta katman mimarisi olan Spring çerçeve ve sunucu sayfa tabanlı bir önyüz mimarisi (JSF 2.0)' ın özellikleri birbirleriyle birleştirilerek, yeni bir çerçeve oluşturulmuş ve dinamik teknolojiler, günlük yönetimi, hata yönetimi vb. yöntemlerle genişletilmiştir.

Anahtar kelimeler: çerçeve, kullanıcı arabirimi, orta katman, managed bean, kontrol sınıfı, DAO sınıfları

Developing Software Framework With An Hybrid Architectural Approach

Abstract. On the field of software development, there are different kinds of software framework in order to simplify and optimize the software development processes. Any software framework has not only superior capabilities but also inefficiencies to effect software development processes as positive or negative. While some frameworks are strong in terms of architectural component like memory management, transaction management on middle layer, the others provide reliable and scalable user controls on user interactions. To provide optimum expectation on software development processes, it is need to design a new software development framework by using good features of the frameworks. In this study,it is tried to develop a new software development framework which uses Java based framework, Spring on middle layer, and JSF on user interface. These two technologies are integrated each other and extended by dynamic technologies, log management, exception handling etc.

Key words: framework, user interface, middle layer, managed bean, controller, DAO classes

1 Giriş

Kurumsal uygulamalarda, uygulamanın standartlarını, kalitesini, uygulama yaklaşımını belirleyecek, altyapısal işlemlerde kolaylık sağlayacak, tasarım örüntülerinin nesne yönelimli olarak kullanılmasını kolaylaştıracak, kısacası uygulama teknik çerçevesini belirleyecek yapılara ihtiyaç duyulmaktadır. Böyle yapıların olmamasını, temeli olmayan binalar üretmeye benzetebiliriz. Diğer bir deyişle proje kapsamında her geliştirilen modülde, hatta sayfada, bellek yönetimi, hareket yönetimi, loglama, hata yönetimi ve hatta kullanıcı kontrolleri gibi yapıların mikroservislerden farklı olarak tekrar tekrar geliştirildiği düşünülebilir. Mikroservisler [1] de iş kısmı küçük parçacıklara ayrılmaktadır. Kullanılan ortak bileşenler tekrar tekrar geliştirilmemektedir. Yukarıda bahsedilen tüm olumsuzlukların ışığında, kurumsal bir yazılım mimarisinde, yazılım çerçeve ihtiyacı aşikardır.

Bu noktada nesne yönelimli yazılım mimari bileşenlerinin etkin olarak kullanılmasına olanak sağlayan, uygulama kapsamında kullanılacak ortak bileşenleri uygulamanın kullanımına esnek olarak sunan, farklı tasarım örüntülerinin kullanımına izin veren, bellek yönetimi, hareket yönetimi, hata yönetimi, log yönetimi, veritabanı erişimi ve kullanıcı kontrolleri gibi yapılarla uygulama çerçevesini belirleyen, bir yazılım mimarisi tasarımı, kurumsal uygulamalar için kaçınılmazdır.

Yazılım mimari çerçevelerinin en önemli özelliği, içerisinde proje iş bileşenlerini barındırmamalarıdır. Proje kapsamında, uygulama iş kurallarının geliştirilmesinde gerekli olduğu durumlarda yazılım çerçevesi bileşenleri çağrılır. Böylelikle geliştirilmiş olan yazılım çerçevesi, iş kuralları birbirinden çok farklı uygulamaların geliştirilmesinde kullanılabilir.

Günümüz yazılım mimari yaklaşımlarında kod geliştirme faaliyetlerinin, kolay ve hızlı olmasının yanında, modüler olması da önemlidir. Bu nedenle, özellikle veritabanı uygulamaları için, yazılımcıya mümkün olduğu kadar az iş yükü çıkaracak, sade, kolay anlaşılabilir kodlar üretecek, uygulama mimarileri geliştirmek önemlidir.

Bu çalışmada ön yüzde kararlı, farklı tarayıcılarda kullanıcı kontrollerinin aynı davranışları gösterdiği, kullanıcı kütüphanesinin geniş olduğu ve uygulama ihtiyaçlarına göre özgün kullanıcı kontrolleriyle genişletilebilecek JSF[2] mimarisi ile, orta katmanda hareket yönetimi, bellek yönetimi, hata yönetimi, log yönetimi vb. mimari alanlarında üstün özellikler gösteren Spring mimarisini kullanan, bir yazılım mimari çerçevesi sunulmuştur. Buna ek olarak JPA [3] temelli bir veri nesne modeli (entity) yapısı kurulmuş olup, sanal sınıflarla servis ve veri katmanları ortak bileşen olarak geliştirilerek, uygulama geliştirme sürecinde yazılmcının kod geliştirme faaliyetleri hızlandırılmıştır.

2 Yazılım Mimarisi

Yazılım mimarisi, kurumsal uygulama geliştirme süreçlerine altlık oluşturacak yazılım bileşenleri, bu bileşenlerin uygulama iş bileşenlerinin kullanımına sunulan özellikleri ve birbirleriyle ilişkilerinden oluşan yapılardır [4].

Bir yazılım sisteminin temel çatısı mimari üzerine kurulur. Bu çatı, yazılım sistemi bileşenleri, bileşenlerin kendi arasında ve dış sistemlerle etkileşimleri ve bu çatıyı temel alan uygulama tasarlama ve geliştirme kurallarından oluşur [5].

Yazılım mimarileri tasarım biçimlerine göre aşağıdaki gibi sınıflandırılabilirler [6]:

1. **Katmanlı mimari.** Bileşenler katmanlı olarak tasarlanmışlardır. Bu yapıda bileşenler, komşu katman bileşenleri ile etkileşimde bulunacak şekilde yapılandırılmışlardır.
2. **Kanal ve filtre temelli mimari.** Bu mimaride bileşenler, kanallarla konuşan filtreler gibi işlev görürler.
3. **Paylaşılan veri mimarisi.** Bu mimari yaklaşımında bileşenler, paylaşılan veri erişimcisi tarafından erişilen, paylaşılan veri deposu olarak tasarlanırlar.
4. **Olay güdümlü mimari.** Bu mimari yaklaşımında bileşenler, olay yöneticisi nin, olay bildirimleri doğrultusunda işlev görürler.
5. **Tanım-Görünüm-Yönetici (Model-View-Controller -MVC).** Günümüz web uygulamalarının temel almış olduğu mimari yaklaşımdır. Bu mimari yaklaşımın sağlamış olduğu, soyutlama ile her katman kendi işine odaklanır. Tanım katmanı veri modeli işlemlerini, görünüm katmanı kullanıcı arayüzü işlemlerini, yönetici katmanı ise görünüm katmanı ile tanım katmanı arasında iletişimi sağlar. Yönetici katmanı, iş kurallarını tanım ve görünüm katmanından edinmiş olduğu verilerle çalıştırarak, kullanıcıya gösterilecek biçime çevirdiği veriyi görünüm katmanına, veri saklama ortamlarına kaydedilmesi için de tanım katmanına iletir.

Tablo 1. de bu mimari yaklaşımları kullanarak geliştirilmiş yazılım çerçevelerinin karşılaştırması yapılmıştır [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17].

3 JSF

Java EE [18] mimarisinin bir parçası olan JSF, kullanıcı arabiriminde işlevsellik gösteren kullanıcı kontrolleri ailesinden oluşan bir çerçevedir. JSF' de kullanıcı kontrolleri, sunucu tarafında yorumlanmakta olup, javascript'te karşılaşılabilecek tarayıcı uyumsuzluğu sorunu bulunmamaktadır.

JSF' de kullanıcı kontrollerinin oluşturulmasında, öncelikle kontrol ağacı oluşturulur. Daha sonra, kontrol ağacı HTML sayfasına yorumlanır. Bu iki sürecin birbirinden ayrı olması, kullanıcı tarafında farklı işaretleme (markup) dillerinin kullanılmasına olanak sağlar. Böylelikle farklı ortamlarda, akıllı telefon, tablet, masaüstü bilgisayarlar vb. JSF kullanıcı kontrolleri kullanılabilir [19].

JSF kullanıcı kontrollerinde istemci tarafta gerçekleşen değişiklikler, dinleyici (listener) aracılığıyla, Ajax teknolojileri [20] kullanılarak sunucuya iletilmekte ve bu mesajların işlenmesi sonucu oluşan güncellemeler, kullanıcı kontrollerine bildi-

rilmektedir. Buradan hareketle, JSF'in bir olay güdümlü programlama çerçevesi olduğu söylenebilir..

JSF kullanıcı kontrolleri farklı kullanıcı aletlerini desteklemektedir. JSF ile geliştirilen bir veritabanı uygulamasında, uygulama iş kuralları ve veri modeli nesneleri, uygulama katmanında yer alır [3].

Tablo 1. Yazılım Çerçeveleri Karşılaştırma

	MVC	Modülerlik	Yeniden Kullanım	Ölçeklenebilirlik	Bağımlık Enjeksiyonu Esnek Kullanımı	Gevşek Bağlılık	Basitlik, Anlasılırlık	Öğrenme	Zengin Kütüphane	Güvenlik	Esneklik	Duvarlılık (Responsive)	Etkin Kaynak Yönetimi	Asenkron
Melez Çerçeve	√	√	√	√	√	√	√	√	√	√	√	√	√	√
Spring MVC	√	√	√	√	√	√				√	√		√	√
JSF	√	√	√	√			√	√	√	√		√		
Struts	√	√	√	√						√		√		
Hibernate		√		√			√	√			√			
Play Framework	√		√		√		√	√		√		√	√	√
Vaadin							√	√	√		√	√		
Grails	√	√			√		√	√			√	√		
Django				√			√	√		√		√		
Wicket	√	√	√	√							√	√	√	√
Vert.x							√	√				√	√	√
Entity Framework	√	√	√	√		√	√	√		√		√	√	√
AngularJS	√	√	√		√	√			√	√		√	√	√

4 Sunulan Mimari

Sunulan yazılım mimarisinde, kullanıcı uygulama iletişiminde, JSF teknolojileri kullanılmıştır. Sunucu tarafında yorumlanan JSF kullanıcı kontrollerinin kullanımı ile istemci tarafı, tarayıcı ve kullanıcı aletleri bağımlılığı ortadan kaldırılmıştır. JSF teknolojilerinde tutarlı, ölçeklenebilir, esnekliği kurumsal uygulamalarda denenmiş, zengin içerikli kütüphanelerin geliştirilebilmesi, uygulama geliştirme süreçleri ve geliştirilen üründeki kaliteyi artırmaktadır. JSF çerçevesinin kullanıcı kontrolleri

temelli olması, özellikle veritabanı uygulamalarında orta katmanda iş kuralları ve veri modelleme nesnelerini çerçeveleyerek, hareket yönetimi, bellek yönetimi, log yönetimi, hata yönetimi, güvenlik vb. hizmetleri sağlayacak, bir uygulama katmanına gereksinim oluşturmaktadır.

Bu noktada bilindik, ölçeklenebilir, tutarlı, güvenilir, esnek bir çerçeve olan Spring çerçevesi [21] kullanılmıştır.

Geliştirilen mimaride yazılımcının güvenilir, modüler kaynak kodu, hızlı ve kolay bir şekilde geliştirebilmesi ölçütleri göz önünde tutulmuştur. Veri modelinin tanımlanması, kullanıcı gösterim sayfasının yazılımının yapılması, orta katmandaki bean'lerin, sanal (abstract) bean sınıflarından türetilmesi ve dinamik kod geliştirme teknolojilerinin kullanılması ile orta katmanda mümkün olduğu kadar az kod yazılarak, uygulama geliştirilmesi sağlanmıştır.

Sanal sınıf olarak geliştirilen bir hata yönetim mekanizması, JSF mesajlaşma ve hata yönetim yapısıyla birleştirilmiştir.

Yine uygulama çerçevesine bütünleşik bir veritabanı günlük mekanizması geliştirilmiştir.

Bellek yönetimi, hareket yönetimi, güvenlik yönetimi, Spring çerçeve tarafında, etiketler kullanımı ile sağlanmıştır.

Kullanıcı oturum başlatma mekanizması veritabanındaki bir kullanıcı tablosunda, kullanıcı ve şifre bilgisi tutularak sağlanmıştır.

Uygulama very modeli, veri model nesnesi (entity) JPA teknoloji kullanımıyla veritabanı tablosu ile ilişkilendirilerek oluşturulmuştur. JSF sayfaları ile iletişimi sağlayan managed bean'ler, oluşturulan sanal bir managed bean' den türetilmiştir. Bu sanal managed bean içerisinde, veri modeli nesnesinin dinamik olarak oluşturulması, kullanıcı sayfalarından gelen verilerin, oluşturulan nesneye atanması, iş kuralları ve veritabanı işlemlerini gerçekleştirecek yapıların çağrılması işlemleri yapılır.

Bu işlemler için çağrılan yönetici (Controller) sınıflarında iş kuralı nesneleri oluşturulur; veri modeli nesnelerine değerler atanır ve veritabanı işlemlerini yapacak DAO nesneleri çağrılır. Yukarıda bahsedilen yapıların, tümü sanal sınıflar olup, veri modeli nesnesine göre dinamik olarak oluşturma, değer atama, yöntem çağırma işlemlerini kullanmakta olup, iş kuralları için tekrar tekrar geliştirilmesine gerek bulunmamaktadır.

Raporlama işlemleri, geliştirilen sanal bir rapor sınıfı içerisinde, dinamik kriter atama yöntemleri kullanılan bir rapor arayüzü ile rapor ve kriter sınıfınının, rapor yapısına sağlanması ile Jasper Reports [22] kullanımıyla oluşturulur.

5 Çerçeve Yapıları

5.1 Mamage Bean

Uygulamada istemci sunucu iletişimi managed bean' ler aracılığı ile sağlanır. Web sayfasındaki JSF kullanıcı kontrollerine yapılan veri girişleri, managed bean sınıfında tanımlanmış değişkenler ve nesneler aracılığı ile toplanır. Uygulama çerçevesinde

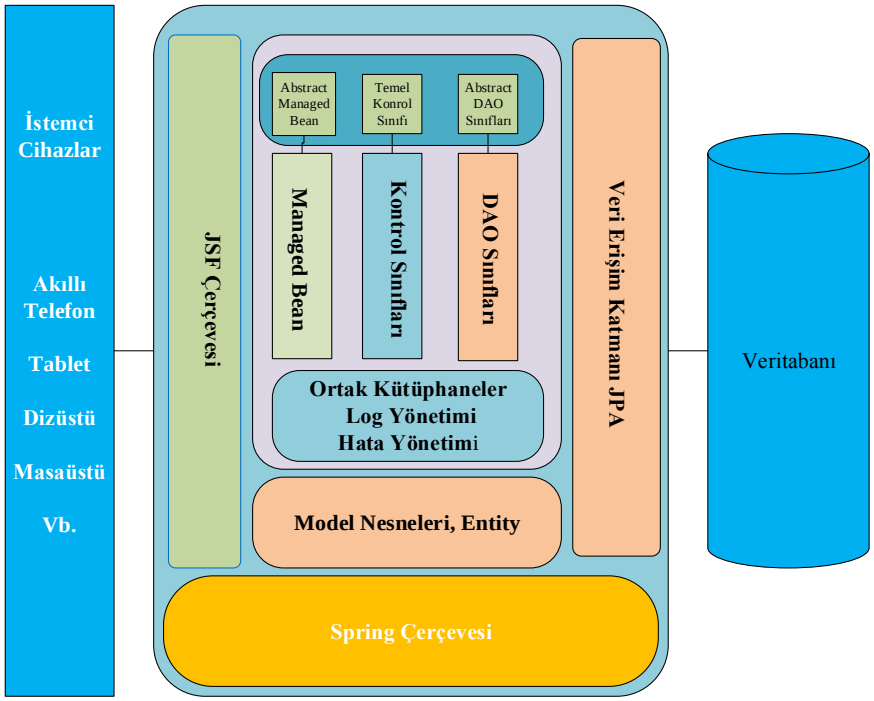
geliştirilen managed bean'ler, sanal bir managed bean' den türetilmekte olup, hiç kod yazmaya gerek olmadan dahi veritabanı işlemleri gerçekleştirilebilir. Bu durum, sanal managed bean üzerinde veri modeli nesnelerinin dinamik olarak, sanal managed bean' in bir özelliği olarak oluşturulabilmesi ve sayfa üzerindeki veri giriş değerlerinin, bu sınıf değişkeninde toplanabilmesiyle sağlanabilmektedir. [2], [3]

Aşağıda örnek pseudo kodları da görülebilecek yapılar, uygulamaya esneklik ve değişik veri modellerine uyarlanabilirlik desteğini sağlamaktadır (bknz. Şekil 3,4,5,6).

Böylelikle Java kodlamada çok tecrübeli olmayan bir yazılımcı dahi, sadece veri modeli nesnesini oluşturup, istemci tarafında JSF sayfasını geliştirerek, çerçevenin sağlamış olduğu kütüphaneler aracılığıyla, Java EE mimarisinde uygulama geliştirmedeki karmaşık teknik detaylarla boğuşmadan, kolay ve hızlı bir şekilde uygulama geliştirebilir.

Zaten bir çerçevenin en önemli faydaları, uygulama geliştirme sürecini bir standarta kavuşturarak, kaliteyi artırmak ve ortak yapıları yazılımcının kullanımına sağlayarak, yazılım sürecini basitleştirmek ve hızlı kod geliştirilmesini sağlamaktır.

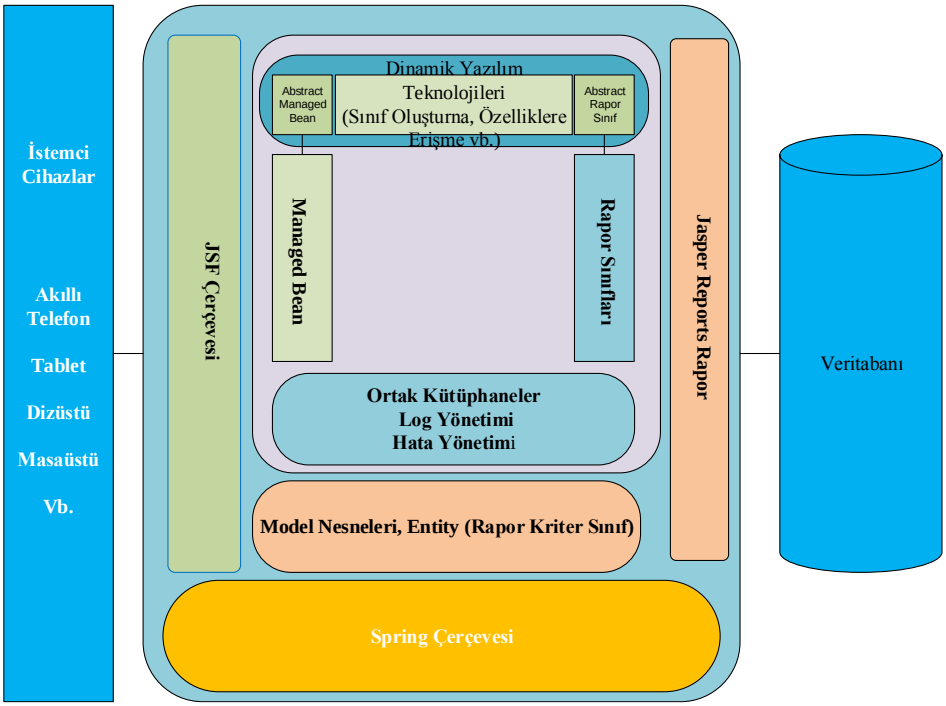
Şekil 1. Yazılım Çerçevesi Sunumu



5.2 Kontrol Sınıfları (Controller)

Bu katmanda JSF ile Spring çerçeve iletişimi sağlanır. Yönetici sınıflar, JPA aracılığıyla veri modeli nesnelerinin veritabanı işlemlerini geliştirmek üzere, DAO sınıflarını çağırırlar.

Şekil 2. Yazılım Çerçevesi Raporlama Yapısı



5.3 DAO Sınıfları

DAO sınıfları yine sanal bir DAO sınıfından türer ve bu DAO sınıfında JPA kullanımı ile veritabanı üzerindeki ekleme, güncelleme, silme ve benzeri işlemleri gerçekleştirirler. Yine bu sanal DAO sınıfı çerçeve tarafından sağlanmış olup, yönetici sınıfları aracılığı ile veritabanı üzerinde işlem yapılacak veri modeli nesnesi parametresi ile çağrılır. Bu sanal DAO sınıfında veri modeli nesnesi ile parametrik işlemler yapılması gereken durumlarda, nesne özelliklerine dinamik olarak ulaşan değer getirme ve atama işlemleri gerçekleştirilmiştir.

5.4 Hata Yönetimi Mekanizması

Uygulama genelinde bir hata yönetim sınıfı uyarlanmış olup, JSF hata mesajları yapısı ile bütünleştirilmiştir.

5.5 Günlük (Log) Yönetimi Mekanizması

Uygulama genelinde seviye bazlı bir günlük yönetimi yapısı geliştirilmiş olup, mesajlar belirlenen biçimde dosya sistemine yazılmaktadır. Sisteme farklı saklama ortamları, geliştirilecek aracı sınıflarla eklenebilir.

5.6 Raporlama Yapısı

Uygulama genelinde raporlamalar Jasper Reports [22] ile geliştirilmektedir. Bu noktada çerçevenin sağladığı özgünlük, rapor parametrelerinin paketlenerek, kullanıcı tarafından talep edilen raporların uygulama katmanında çağrılmasına sağlayacak yapının gerçekleştirilmesidir. Bu noktadaı geliştirilmiş olan AbstractReportBean sınıfı parametre olarak, almış olduğu rapor parametre sınıfı özelliklerine dinamik olarak erişerek, rapor parametrelerini oluşturmakta ve rapor ismi ile birlikte Jasper Reports ile geliştirilen bir raporu çağırmak için gerekli geliştirmeleri içermektedir (bknz. Şekil 2).

6 Yazılım Çerçevesi Geliştirmeleri

Aşağıdaki pseudo koddan da görüldüğü gibi, managed bean'in generik bir yapıdaki AbstractManagedBean sınıfından türetilmesi, yapıcı kısmında türediği sınıfın yapıcısına, veri modeli nesne belirtecinin geçilmesi ve yine üst sanal sınıftaki CreateInstance yöntemi ile, dinamik olarak veri modeli nesnesinin yaratılması işlemlerinin geçeklenmesi, yazılımcının managed bean geliştirmeleri için yeterli olacaktır.

Sayfada yapılan veritabanı işlemleri managed bean'in türediği sanal sınıftan devraldığı, save, update, remove, findAll gibi komutlar aracılığı ile geliştirilmekte olup, aşağıda örnek kod parçaları görülmektedir (bknz. Şekil 3,4.).

Şekil 3. Managed Bean Yapısı

```
ManagedBean :  AbstractManagedBean<VeriModelNesne>
Ctor () :
      Super (VeriModelNesne.class);
      CreateInstance();
```

Şekil 4. Sanal (Abstract) Managed Bean Yapısı

```
Class :  AbstractManagedBean<T>
Ctor (Class<T> entityClass) :
    Yönetici Sınıfı (BaseController) Oluşturulur.
```

```
    Veri modeli nesnesi yönetici sınıfına parametre
    olarak geçilir.
    CreateInstance();

List<T> findAll():
    if (entityList == null){
          entityList =baseController.findAll();
    }
    return  entityList;

List<T> findAll(HashMap<String, Object> parameters):
    if (entityList == null){
        entityList =baseController.findAll(parameters);
    }
    return  entityList;

save():
    baseController.setEntityClass(entityClass);
    baseController.save(entityInstance);

update():
    baseController.update(entityInstance);

remove(T entity):
    baseController.remove(entity);
```

Sanal managed bean' den devir alınan, veritabanı işlemlerine yönelik yöntemler, çerçeve tarafından sağlanan, generik yapıdaki temel yönetici (BaseController) sınıfı aracılığı ile veritabanı işlemlerini gerçekleştirirler. Bu sınıf Spring çerçeve bellek yönetimi, hareket yönetimi mekanizmalarına tabidir (bknz. Şekil 5.).

Şekil 5. Temel Kontrol Sınıfı (BaseController) Yapısı

```
Class :  BaseController <T>
Ctor (Class<T> entityClass) :
    this.entityClass = entityClass;

setEntityClass(Class<T> entityClass) :
    this.entityClass = entityClass;

List<T> findAll():
    dao.setEntityClass(entityClass);
    return dao.findAll();

List<T> findAll(HashMap<String, Object> parameters):
    dao.setEntityClass(entityClass);
    return dao.findAll(parameters);

save(T entity):
    dao.setEntityClass(entityClass);
    dao.create (entity);
```

```
update(T entity):
    dao.setEntityClass(entityClass);
          dao.update (entity);

remove(T entity):
    dao.setEntityClass(entityClass);
    dao.remove (entity);
```

BaseController generic sınıfı, AbsttactDAO sınıfından türeyen DAO sınıfları aracılığı ile veritabanı işlemlerini gerçekleştirir. AbsttactDAO sınıfı veritabanı işlemi yöntemleri ile ilgili tüm işlemlerin gerçeklemelerini içerir.

Bu sınıfta JPA teknolojileri ile geliştirmeler yapılmıştır. Bu sınıflar çerçeve dahilinde yazılımcıların kullanımına açık olup, bu katmanda normal şartlarda ek bir geliştirme gerekmemektedir.

Bahsi geçen sınıfın pseudo kodları aşağıda bulunmaktadır (bknz. Şekil 6.).

Şekil 6. Sanal Veri Erişim Sınıfı (AbstractDAO) Yapısı

```
Class :  AbstractDAO <T>
Ctor (Class<T> entityClass) :
    this.entityClass = entityClass;

setEntityClass(Class<T> entityClass) :
    this.entityClass = entityClass;

getEntityManager():
    return entityManager;

List<T> findAll():
    CriteriaQuery cq = createQuery();  Kriter sorgusu
                                       oluşturulur.
    root = cq.from(entityClass)
    cq.select(root);  Entity nesnesi sorgu alanları
                      listesi oluşturulur.
    cq.where(predicate); Sorgu kriter filtresi eklenir.
    return getEntityManager().createQuery(cq)
                             .getResultList();

List<T> findAll(HashMap<String, Object> parameters):
    CriteriaQuery cq = createQuery();  Kriter sorgusu
                                       oluşturulur.
    root = cq.from(entityClass)
    cq.select(root);  Entity nesnesi sorgu alanları
                      listesi oluşturulur.
    Parametre değerlerinden sorgu kriter belirtimi
    (predicate) yapısı oluşturulur.
    cq.where(predicate); Sorgu kriter filtresi eklenir.
    return getEntityManager().createQuery(cq)
                             .getResultList();

T find(Object id) :
```

```
    return getEntityManager().find(entityClass, id);

create(T entity):
    getEntityManager().persist(entity);

update(T entity):
    getEntityManager().merge(entity);

remove(T entity):
    getEntityManager().remove(entity);
```

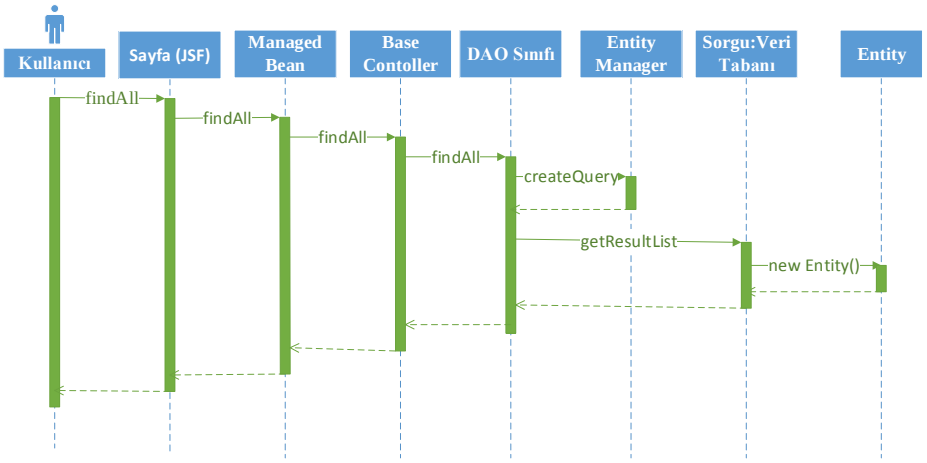
Uygulama genelinde raporlamalar Jasper Reports ile geliştirilmektedir. Çerçevenin rapor sınıfı, AbstractReportBean, parametre olarak almış olduğu, rapor parametre sınıfı özelliklerine, dinamik olarak erişerek rapor parametrelerinin oluşturulması ve rapor ismi ile birlikte, Jasper Reports ile geliştirilen bir raporun çağrılması için gerekli geliştirmeleri içermektedir (bknz. Şekil 7.).

Şekil 7. Sanal Rapor Sınıfı (AbstractReportBean) Yapısı

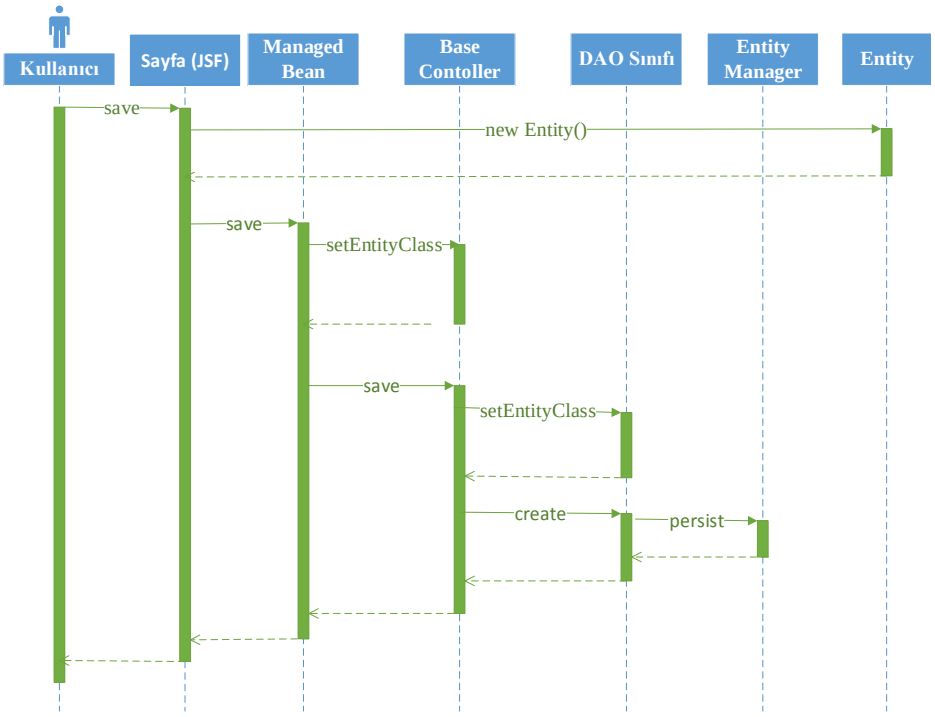
```
Class :  AbstractReportBean <T>
Ctor (Class<T> _reportClass ,T _reportInstance
                           ,String _reportName):
    reportClass = _reportClass;
    reportName = _reportName;
    reportInstance = _reportInstance;
    setExportOption(ExportOption.PDF);

    createParameters():   Rapor   nesnesinden   kriter
    değerlerine alan ismine göre ulaşır ve rapora para-
    metre olarak ekler.
    Map<String,Object> parameters = new HashMap<String,
                                            Object>();
    Field[] fieldsArray = reportInstance.getClass()
                                  .getDeclaredFields();
    for (Field field : fieldsArray) {
     Object value = field.get(reportInstance);
     parameters.put(field.getName(), value.toString());
    }
    return parameters;

execute(): Raporu derler ve çalıştırır.
```

Yazılım çerçevesinin iş bileşenlerinin kullanımına sunmuş olduğu veritabanı işlemlerinin sekans diyagragramları aşağıda gösterilmektedir (bknz. Şekil 8).

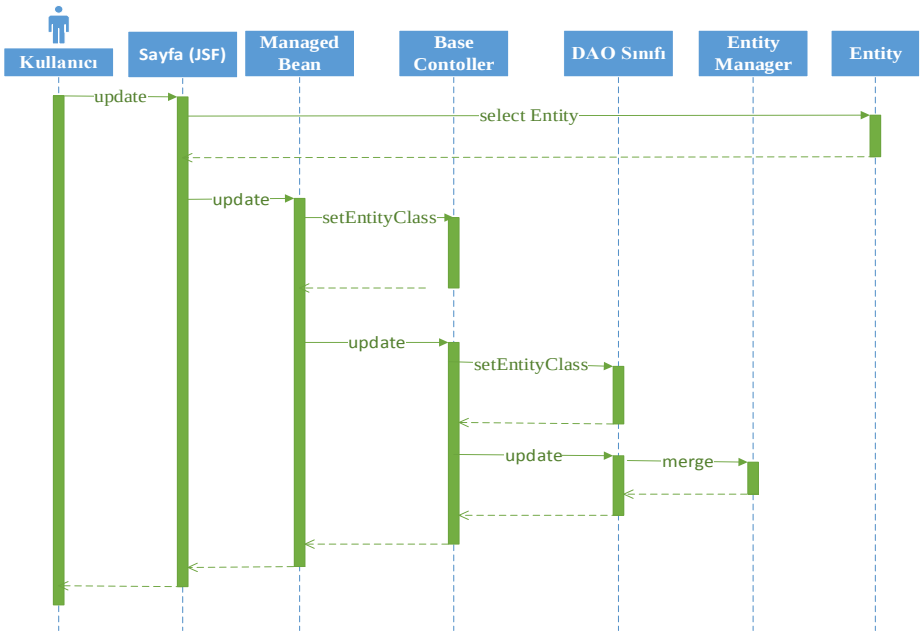
Şekil 8.a. Kayıt Listeleme Yöntemi (findAll) Sekans Diyagramu



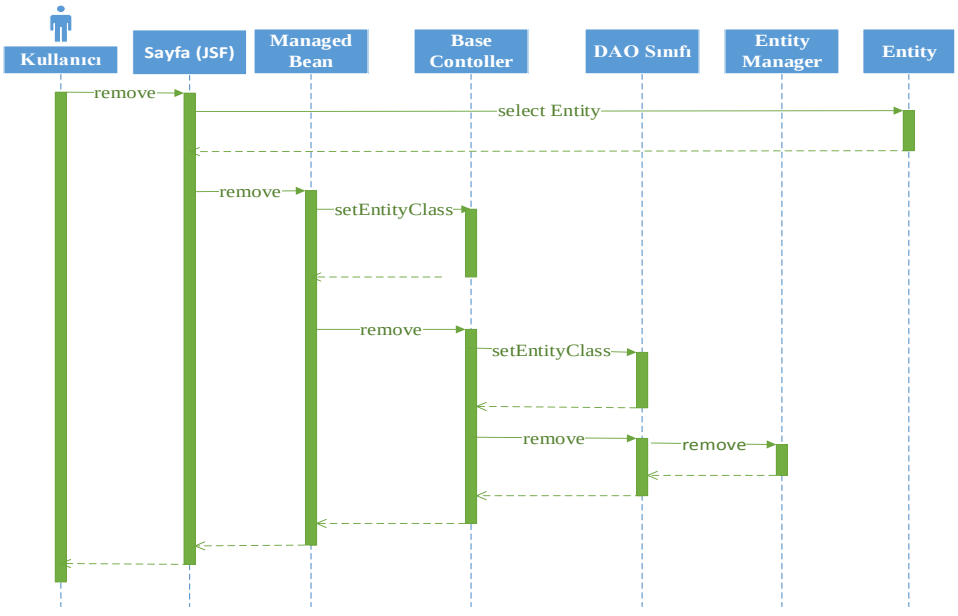
Şekil 8.b Kayıt Ekleme Yöntemi (save) Sekans Diyagramu



Şekil 8.c. Kayıt Güncelleme Yöntemi (update) Sekans Diyagramu

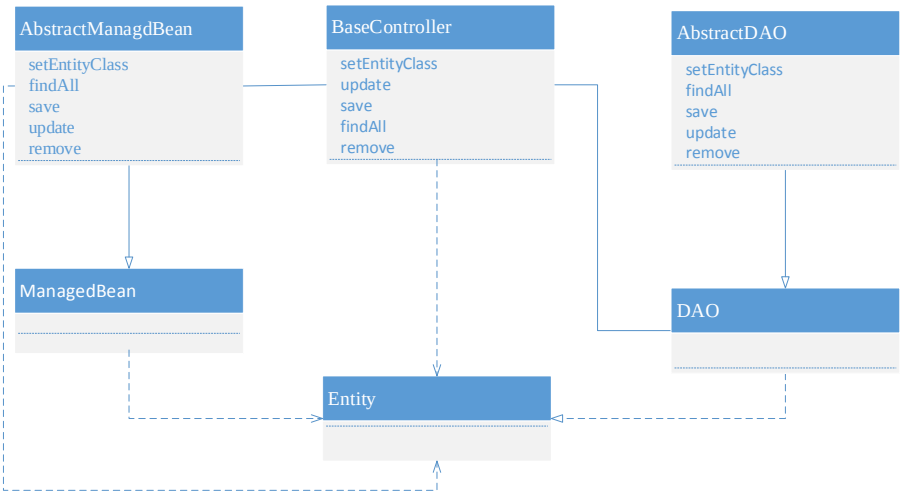


Şekil 8.d. Kayıt Silme Yöntemi (remove) Sekans Diyagramu



Yazılım çerçevesi bileşenlerinin sınıf diyagramı Şekil 9'.da gösterilmektedir.

Şekil 9. Yazılım Çerçevesi Sınıf Diyagramı



7 Sonuç

Kurumsal bir yazılım mimarisinin, gereksinimleri olan modüler, kaliteli, kararlı, ölçeklenebilir, tutarlı, kolay ve hızlı kod geliştirebilme, kolay yaygınlanştırma vb. mimari özellikleri düşünülerek bir yazılım çerçevesi geliştirilmiş olup, bu çalışma ile sunulmaya çalışılmıştır. Bu çalışmada geliştirilen yazılım çerçevesinde, web temelli veri tabanı uygulamaları hedef alınmış olup, orta katmanda Java gibi matematik işlemlerde güçlü bir programlama dili kullanıldığı için, matematiksel işlem yoğunluklu uygulamalarda geliştirilebilir.

Yazılım çerçeveleri genellikle, uygulama süreçlerini standartlaştırmayı ve bu süreçlerde ortak kütüphaneler kullanımıyla, kolay, anlaşılabilir, kaliteli, hızlı kod geliştirmeye hedeflediği için, hazır teknolojilerden faydalanmakla birlikte, çerçeve yaklaşımı özgün olup, iş nesneleri ile iletişimdeki katman yapısı, kullanılan dinamik nesne oluşturma yöntemleri ile uygulama geliştirme süreçlerinde hızlı, anlaşılabilir kod geliştirmenin sağlanması, raporlama araçlarıyla iletişimi sağlayan dinamik yapılarla raporlama süreçlerininin basitleştirilmesi, vb. açılımlar bu yaklaşımla yapılmıştır.

Çerçeve geliştirmelerinin yazılım süreçlerine temel katkısı, kurumsal bir mimaride uygulama geliştirme süreçlerinin karmaşık yapısına destek sağlıyacak yapıların, yetkin ekiplerce geliştirilen kütüphanelerle gerçeklenerek, yazılımcının kullanımına sunulmasıdır. Böylelikle yazılım süreçleri hem kolaylaymış, hem de hızlanmış olur.

Yazılım çerçevesi yaklaşımında lego benzeri bir yaklaşım ön planda tutulmuş olup, bir servis katmanı aracılığıyla javascript temelli bir yaklaşım, çerçeveye eklenebilir. Orta katman Spring çerçevenin sağlamış olduğu bellek yönetimi ve hareket yönetimi gibi faaliyetler geliştirilecek farklı yaklaşımlarla sağlanabilir. Buna ek olarak, verita-

banı erişim katmanında farklı erişim terknolojileri kullanılarak çerçeve yaklaşımı genişletilebilir.

Kaynakça

1. Johannes T. Microservices. IEEE Software, 32(1):116-116, 2015.
2. JSF Reference Site,http://www.javaserverfaces.org/
3. Java Persistence API, https://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html
4 Bass, Klemen, Kazman, Software Architecture in Practice (2 nd edition), Addison-Wesley, 2003.
5 Recommended Practice for Architectural Description of Software-Intensive Systems, ANSI/IEEE Std 1471-2000.
6 https://www.academia.edu/25325573/YAZILIM_TASARIM_VE_MİMARİSİ_-_SOFT WARE _DESIGN_AND_ARCHITECTURE, SIRIUS information technology,Sam Houston State University, Computer science, Graduate Student
7. Struts Framework, https://struts.apache.org/
8. Hibernate, http://hibernate.org/
9. Google Web Toolkit (GWT), https://opensource.google.com/projects/gwt
10. Play Framework, https://www.playframework.com/
11. Vaadin,https://vaadin.com/
12. Grails Framework,https://grails.org/
13. Django, https://www.djangoproject.com/
14. Wicket, https://wicket.apache.org/
15. Vert.X, https://vertx.io/
16. Entity Framework, https://docs.microsoft.com/en-us/ef/
17. AngularJS, https://angularjs.org/
18. JSR-000314 JavaServerTM Faces 2.0, https://jcp.org/aboutJava/communityprocess/ final/jsr314/index.html
19. JSR 366: Java Platform, Enterprise Edition 8 (Java EE 8) Specification, https://jcp.org/en/jsr/detail?id=366
20 Dong, S., Cheng, C., Zhou, Y.: Research on AJAX technology application in web development. In: 2011 International Conference E-Business and E -Government, ICEE (2011)
21. Spring Framework, https://spring.io/.
22. Jasper Reports, https://community.jaspersoft.com/wiki/jaspersoft-community-wiki-0.