T.C.
MALTEPE ÜNİVERSİTESİ

Bilgisayar Mühendisliği Bölümü

KULLANIM-VAKALARI ile ASPECT YÖNELİMLİ YAZILIM GELİŞTİRME

C. Kerem ERSOY
031402208

Prof. Dr.
Kemal Köymen

Mart, 2007
Maltepe/İstanbul

T.C.
MALTEPE ÜNİVERSİTESİ

Bilgisayar Mühendisliği Bölümü

KULLANIM-VAKALARI ile ASPECT YÖNELİMLİ YAZILIM GELİŞTİRME

C. Kerem ERSOY
031402208

Prof. Dr.
Kemal Köymen

Mart, 2007
Maltepe/İstanbul

ÖZET

Kullanım-vakaları ile Aspect Yönelimli Yazılım Geliştirme isimli bu çalışma Bilgisayar Mühendisliği Yüksek Lisans tezi olarak Maltepe Üniversitesi Fen Bilimleri Enstitüsünde hazırlanmıştır.

Yazılım sistemlerinde amaç isterlerin karşılanmasıdır. İdeal olan her bir isterin ayrı bir modülde karşılanması ve diğerlerinden bağımsız olarak geliştirilmesidir. İsterler, sistemde gerçekleştirildikleri takdirde "ilgi" adını alırlar. Bazı ilgilerin gerçekleştirimi için tek bir modül yeterli olsa da, genellikle birden çok modülü etkilerler. Bu ilgiler "çapraz-kesen ilgiler" olarak adlandırılır.

Etkili bir ilgi ayrımı için isterlerin belirlenmesinden başlayarak her aşamada ilgilerin ayrı tutulması gerekmektedir. Gerçekleştirimde ayırım Aspect Yönelimli Programlama ile sağlanabilse de, aspectlerin tasarım modelinde belirlenip, temsil edilebilmeleri için yeni bir gösterim elemanına ihtiyacımız vardır.

Kullanım-vakaları sadece isterleri belirleme yöntemi değildir. Bütün yazılım yaşam çevrimini yönlendiren bir yazılım mühendisliği yöntemidir. Geleneksel diller ilgi ayırımına destek vermediklerinden, kullanım-vakası tekniğinde de ilgilerle ilgili yeterli destek yoktur. İlgilerin ayrı tutulabilmesi için her bir tasarıma özgü olan parçaların diğerlerinden ayrı tutulması gerekmektedir. Bunun için "kullanım-vakası kesitlerini" kullanırız.

Esnek bir mimari, performans ve güvenlik gibi sistemin genelini etkileyen ilgileri karşılamalı ve her bir parçasının hangi ister ya da kullanım-vakasına karşılık geldiği anlaşılabilmelidir. Bu kullanım-vakası kesitleri tarafından yönlendirilen aspectler ile sağlanabilir.

Bu çalışma 2007 yılında hazırlanmış olup, 61 sayfadan oluşmaktadır.

Anahtar kelimeler: Aspect, kullanım-vakası, modülerlik, ister

ABSTRACT

This research titled Aspect-Oriented Programming with Use-cases was completed as a graduate thesis for Computer Engineering at Science Institute of Maltepe University.

Software systems are built to meet the requirements. Ideally each requirement should be implemented in a separate module and developed independently. When a requirement is implemented in a software system, it usually effects more than one module. These are called “crosscutting concerns”.

For an effective concern separation, separation should be preserved throughout the whole software life cycle. Although separation can be accomplished with aspect-oriented programming in the coding phase, we need a new display element in the design phase to represent aspects.

Use-cases direct the whole software life cycle. However they do not have enough support for aspect-orientation as conventional languages do not support concern separation. To keep concerns separate, parts specific to each concern must be kept separate. We use “use-case slices” for this purpose.

Resilient architectures must meet general purpose concerns like performance and security while each part can be mapped to its requirement or use-case. This is accomplished with use-case oriented aspects.

This research was written in 2007 and contains 61 pages.

Keywords: Aspect, use-case, modularity, requirement

İÇİNDEKİLER

ŞEKİL TABLOSU

KOD TABLOSU

1. GİRİŞ ve AMAÇ

Toplam yazılım maliyetinin çoğunluğunu bakım aşaması oluşturmaktadır [1]. Seksenli yıllarda %50 civarında olan bu oran, iki binli yıllarda %90' ın üzerine çıkmıştır [2]. Maliyetleri asgari düzeye indirmek için yazılım çevriminin her aşamasında sistemin modülerliğinin korunması gerekmektedir. Bu amaçla kullanılan Aspect Yönelimli Programlama (AYP) son yıllarda popülerlik kazanmıştır.

AYP sadece gerçekleştirim aşaması ile ilgilenir. Etkili bir modülerlik için, bu yeni yaklaşımın bütün yazılım yaşam çevrimini kapsayan bir yöntembilim ile desteklenmesi gerekmektedir. Bu çalışma kullanım-vakası güdümlü geliştirme ile aspect yönelimli modüler yapıların oluşturulmasını anlatmaktadır.

Takip edilen anlatım düzeni şu şekildedir: Modüler yapılardan bahsedildikten sonra, yazılım isterleri sisteme uygulandığında karşılaşılan sorunlar üzerinde durulmuş ve AYP' nin getirdiği yeni modülerlik birimi incelenmiştir. Daha sonra kullanım-vakası güdümlü geliştirme ile AYP' nin bütünleştirilmesi ve bunun için gereken yeni elemanlar anlatılmıştır. Son olarak, kullanım-vakası yöntembilimi ile aspect yönelimi uygulanarak esnek mimarilerin geliştirilmesi üzerinde durulmuştur.

2. MODÜLER YAPILAR

Karışık bir problem ile karşılaşıldığında insan beyninin başa çıkabilmesi için, problem alt parçalara ayrılır. Her bir parçanın belirli bir rolü, sorumluluğu ve amacı vardır. Parçalar teker teker çözüldükten sonra bir araya toplanarak sistemin tamamı elde edilir.

Sistemin diğer kısımlarından bağımsız olarak geliştirilebilen her bir alt parçası *modül* olarak adlandırılır. *Modüler yapı*, modüllerden oluşan ve her bir modülü birimsel olarak başka modüllerle değiştirilebilen yapılar için kullanılır [3]. Her bir modül diğerlerini etkilemeden sisteme eklenip çıkartılabilir.

Sistemin modülerliği uyum ve bağlaşım ölçümleriyle derecelendirilir. Uyum, bir modüldeki yöntem ve veri yapılarının birbirleriyle olan ilişkilerini ölçer. Eğer yöntemler arasında amaç birliği yoksa ya da yöntemler kendi içlerinde çok sayıda farklı faaliyetlerde bulunuyor ise modülün uyumu düşer. Düşük uyumun sonuçları şunlardır [4]:

1. Anlaşılabilirlik azalır.

2. Bakılabilirlik azalır. (Sürekli olarak değişikliklerden etkilenir.)

3. Yeniden kullanılırlık zorlaşır. (Bir sürü gereksiz yöntem mevcuttur.)

Bağlaşım, modüllerin birbirlerine olan bağımlılığını ölçer. Bir modülün içinden başka bir modülün kullanımı çoğaldıkça iki modül arasındaki bağımlılık artar. Yüksek bağlaşımın sonuçları şunlardır [5]:

1. Anlaşılabilirlik azalır.

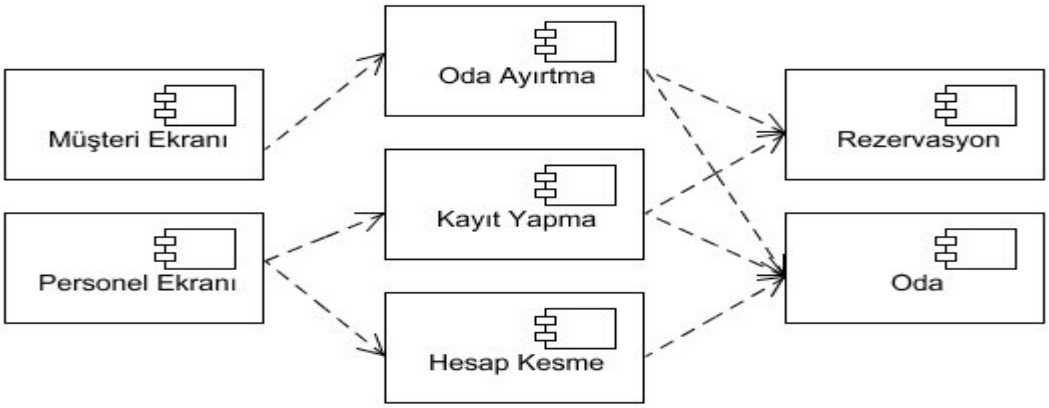
2. Bir modüldeki değişiklik başka modüllerde değişikliğe yol açar.

3. Tekrar kullanılırlık ve sınanabilirlik düşer. (Bağımlı olunan modüllerde dahil edilmelidirler.)

Yüksek uyum ve düşük bağlaşım birbirleriyle uyumluluk göstermektedir. Modül üyelerinin ilişkisi arttıkça, modüller arasındaki bağlaşım azalmaktadır.

Şekil 2.1' de modüler bir yapı örneği gösterilmiştir. Müşteri ve personel ekranı sisteme veri girilmesi ve bu verinin kullanıcılara sunumu ile ilgilenmektedir. OdaAyırtma, Kayıt Yapma ve Hesap Kesme modülleri iş mantığını idare etmekte, Rezervasyon ve Oda modülleri ise verilerin bir veri deposunda saklanmasıyla ilgilenmektedir.

Geleneksel yöntemlerde modülerlik birimi olarak bileşen ya da sınıf kullanılır. Bileşenler (ya da sınıflar) nesnelerin özellik ve davranışlarını bir arada tutar. Böylece bir değişiklik olduğunda sadece ilgili kısımlar etkilenir. Örneğin Odanın özelliğinde bir değişiklik olduğunda, sadece bu özelliği kullanan yöntemlerde değişiklik olur. Ya da personel ekranının görüntüsünde değişiklik istendiğinde sadece Personel Ekranı bileşeni değiştirilir. Yeni bir öznitelik eklenmek istendiğinde ise yeni bir bileşen eklenir ve mevcut bileşenlerde değişiklik yapılır.



Şekil 2.1. Otel Yönetim Sistemi bileşen diyagramı.

Modüler yapıların oluşturulması için sırasıyla isterler bulunur, sistemi oluşturacak modüller belirlenir ve isterler modüllerle eşlenir. Genellikle ister sayısı, modül sayısından çok daha fazladır.

3. İSTERLER

İsterler, işlevsel ve işlevsiz olarak ikiye ayrılırlar [6]. *İşlevsel isterler,* sistemin gerçekleştirmesi gereken öznitelikleri; *işlevsiz isterler* ise sistemin sahip olması gereken özellikleri tanımlar. Birkaç örnek verirsek:

1. Bir müşteri otel odası için rezervasyon yaratabilir.
2. Resepsiyon personeli müşterileri kaydedebilir ya da hesabını kesebilir.
3. Bütün odaların dolu olması durumunda müşteri bekleme listesine eklenecektir.
4. Denetim amacıyla, sistemdeki işlem bilgiler hakkında bilgi tutulmalıdır.
5. Kolaylık olsun diye, sistem kullanıcı tercihlerini varsayılan olarak kullanacaktır.
6. Kayıtlara erişim en fazla iki saniye sürmelidir.

1, 2 ve 3. ifadeler işlevsel, 4, 5 ve 6. ifadeler ise işlevsiz isterleri belirtmektedir. İşlevsiz isterler, genellikle bir altyapı ihtiyacına işaret etmektedirler. Örneğin 4. ifade günlük tutma altyapısı, 6. ifade ise önbellek altyapısına ihtiyacı ortaya çıkarmaktadır.

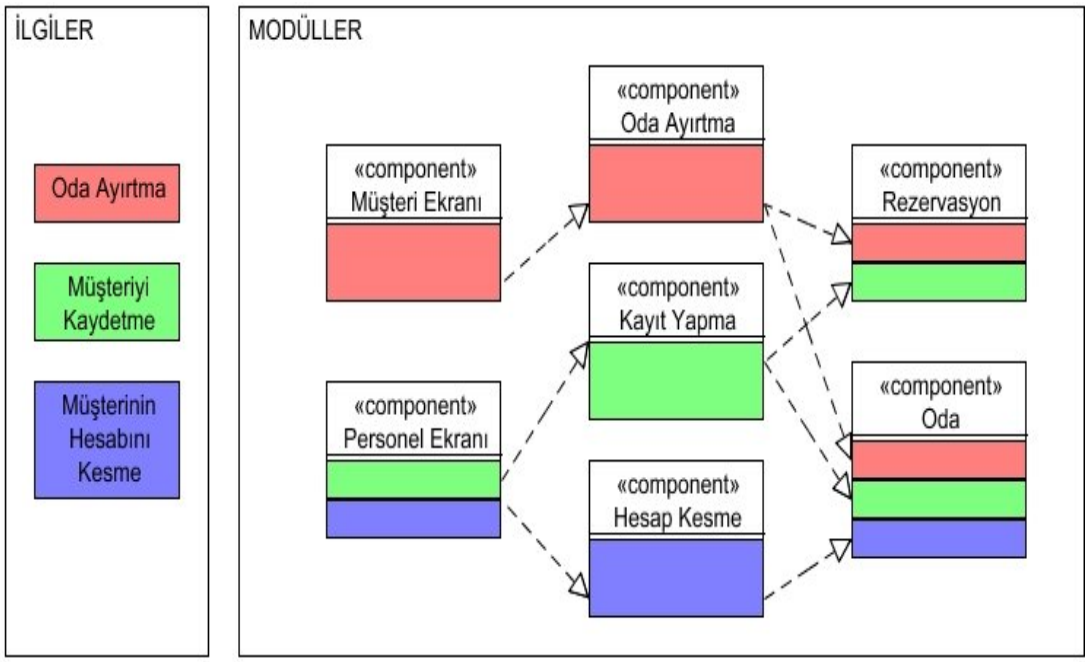
Yazılım sistemlerinde amaç isterlerin karşılanmasıdır. İdeal olan her bir isterin ayrı bir modülde karşılanması ve diğerlerinden bağımsız olarak geliştirilmesidir [7]. Fakat kullanılan geleneksel yöntemler yetersiz kalmaktadır. Sınıflar, nesneleri soyutlamak için yaratılmışlardır ve bir sınıfın kendi başına bir isteri gerçekleştirmesi mümkün olmamaktadır.

4. ÇAPRAZ-KESEN İLGİLER

İsterler, sistemde gerçekleştirildikleri takdirde *ilgi* adını alırlar [8]. Bazı ilgilerin gerçekleştirimi için tek bir modül yeterli olsa da, genellikle birden çok modülü etkilerler. Bu ilgiler *çapraz-kesen ilgiler* olarak adlandırılır. Başlıca iki tipi vardır:

4.1. Eş düzey ilgileri:

Bu ilgiler işlevsel isterlere karşılık gelmektedirler. Varolmak için birbirlerine ihtiyacı olmayan ilgiler, aynı sisteme konulduklarında aşağıdaki şekilde görülen örtüşme meydana gelmektedir.



Şekil 4.1. İlgilerin sisteme yerleştirimi sırasında oluşan karışma ve saçılma.

Bu örtüşme istenmeyen sonuçlar doğurur:

a) Karışma:

Bazı modüller değişik ilgilerin parçalarını içermektedirler. Mesela Oda modülü, üç ilginin kısımlarını içermektedir. Bu durum modülün uyumunu düşürmektedir.

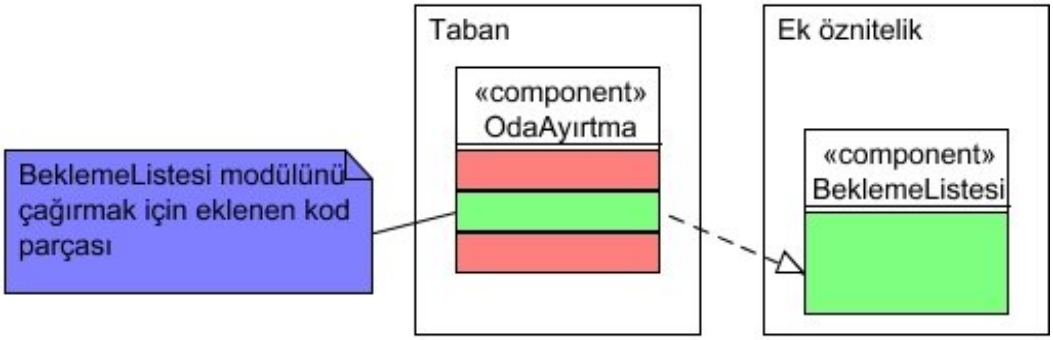
b) Saçılma:

Bir ilgiyi gerçekleyen kod parçacıkları birçok modül arasında yayılmışlardır. Örneğin müşteriyi kaydetme ilgisi dört modül arasında yayılmıştır. Bu modüller arasındaki bağlaşımı yükseltmektedir. Eğer ilgiyi gerçekleştiren tasarımda herhangi bir değişiklik olursa, ilginin ilişkili olduğu bütün modülleri değiştirmemiz gerekmektedir. Dahada önemlisi, sistemin çalışmasını anlamak zorlaşmaktadır. Mesela bir isterin gerçekleştiriminin izlenmesi için değişik modüllerin kaynak kodları okunmak zorundadır. Anlaşılabilirliği az olan sistemlerde bakım ve iyileştirme işlemleri zorlaşmaktadır.

4.2. Uzantı ilgileri:

Bazı ilgiler diğer ilgilere ek öznitelikler katmak için kullanılırlar. Bu tip ilgilerin iki ister türüne de karşılık gelen örnekleri vardır. Mesela Otel Yönetimi Sisteminde, OdaAyırtma modülü için Bekleme Listesi ek öz niteliği bulunmaktadır. Eğer boş oda kalmamışsa, sistem müşteriyi bekleme listesine koyar. Ya da sistemde belirli noktalarda işlemler hakkında günlük tutulması gerekmektedir. Birinci örnek işlevsel, ikinci örnek ise işlevsiz bir istere karşılık gelmektedir. Tipleri farklı olsa da sistemi aynı şekilde etkilerler.

Uzantı ilgileri bir taban üzerine tanımlanırlar. Bu ilgiler taban ilgilerine ek fonksiyonlar katsalar da aslında taban ilgisinden bağımsız ilgilerdir yani sisteme eklenmeseler de taban ilgisi bütünlüğünü kaybetmez. Ek öz niteliklerin ayrı tutulmasıyla taban ilgisinin anlaşılır kılınması hedeflenmektedir.



Şekil 4.2. Taban modülüne ek öznitelik eklenmesi.

Yukarıdaki şekilde OdaAyırtma modülüne Bekleme Listesi ek öz niteliğinin eklenmesi modellenmektedir. Eklentinin çağrılması için OdaAyırtma modülünde belirli bir noktaya ek modülü çağırmak üzere bir kod parçası eklenir. Bu noktaya *uzantı noktası* adı verilir. Başka ek öznitelikler eklenmek istediğinde de aynı şekilde başka uzantı noktalarına kod parçacıkları eklenir.

Eklenen kod parçacıkları modülün uyumunu düşürse de esas önemli problem, bütün uzantı noktalarının önceden belirlenememesidir. İhtiyacımız olan yazılım yaşam çevriminin herhangi bir anında, istediğimiz noktaları uzantı noktası olarak kullanabilmektir.

Geleneksel modülerlik yöntemleri günümüzün karışık sistemleri için yetersiz kalmaktadır. İlgi gerçekleştirimleri ayrı tutulamadığı için sistemin anlaşılması,

bakımı ve genişletilmesi her yeni eklenti yada iyileştirme ile dahada zorlaşmaktadır.

5. YENİ MODÜLERLİK BİRİMİ

Aspect Yönelimli Programlama(AYP) çapraz-kesen ilgilerin ayrılmasına yönelik bir takım yöntemlerden oluşur. Araştırmalar uzun zamandır süregelmesine rağmen, geniş çaplı tanınması 1997 yılında, Xerox Parc da çalışan Gregor Kiczales ve takımının yaptığı sunumla başlamıştır [9]. Bu takım aynı zamanda ilk ve en yaygın AYP dili olan AspectJ' yide geliştirmiştir. AspectJ, Java dilini temel alır ve ilgilerin ayrılmasına yönelik bazı yapılar ekler. AspectJ' in dışında başka AYP dilleri de mevcuttur: Örneğin C dili için AspectC, Python dili için Pythius, Java için JAC… [10].

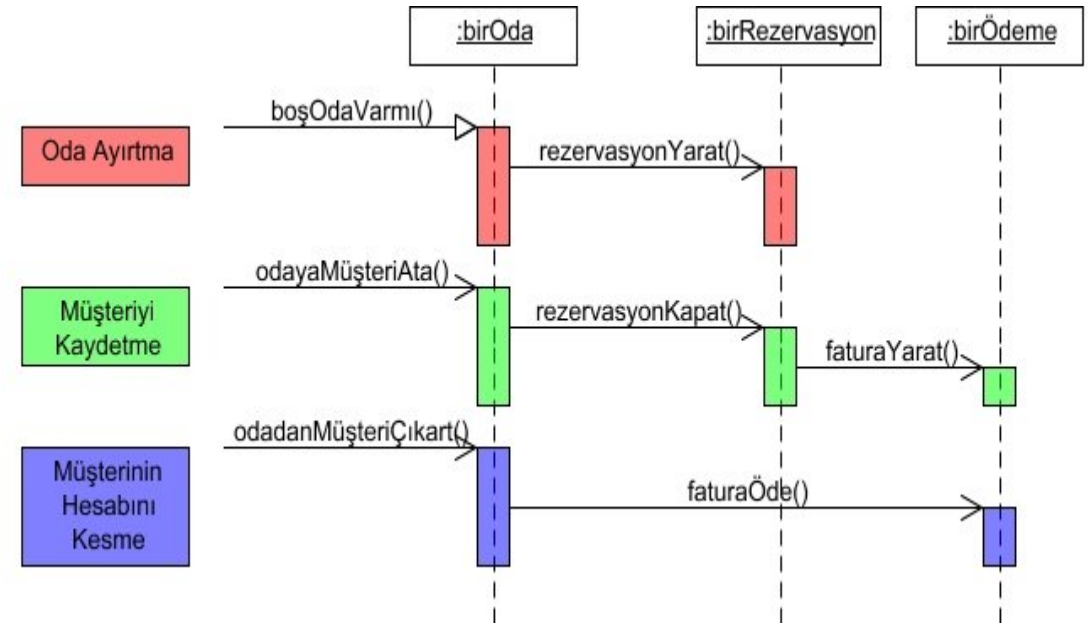
**Tiparası bildirimler,* mevcut modüllere yeni öznitelikler (özellik, yöntem, ilişki) eklenmesini sağlar.

**Tavsiyeler,* mevcut işlemlere nokta-kesimleri tarafından belirlenen uzantı noktalarında eklemeler yapmayı sağlar.

**Aspectler,* tiparası bildirimleri ve tavsiyeleri düzenlemeye yarıyan modülerlik birimleridir.

Bu yeni modülerlik birimi ilgi ayrımı için yeni bir yaklaşım getirmektedir.

5.1. Eş düzey ilgilerinin ayrılması



Şekil 5.1. İlgilerin modülleri kesmesini gösteren dizi diyagramı.

Şekil 5.1' de sistemdeki ilgilerin belirli modüller üzerinde değişik yöntemler gerektirdiği görülmektedir. Bunun yol açtığı karışma ve çapraz-kesmeyi tiparası bildirimleri kullanarak engelleyebiliriz. Bu sayede her bir eş düzey ilginin özellikleri diğerlerinden ayrı tutulur.

```
public aspect MüşteriyiKaydetme {
   public void Oda.odayaMüşteriAta() {
      // odaya müşteri atayan kod
   }
   public void Rezervasyon.rezervasyonKapat() {
      // rezervasyon kapatan kod
   }
   public void Ödeme.faturaYarat() {
      // fatura yaratan kod
   }
}
```

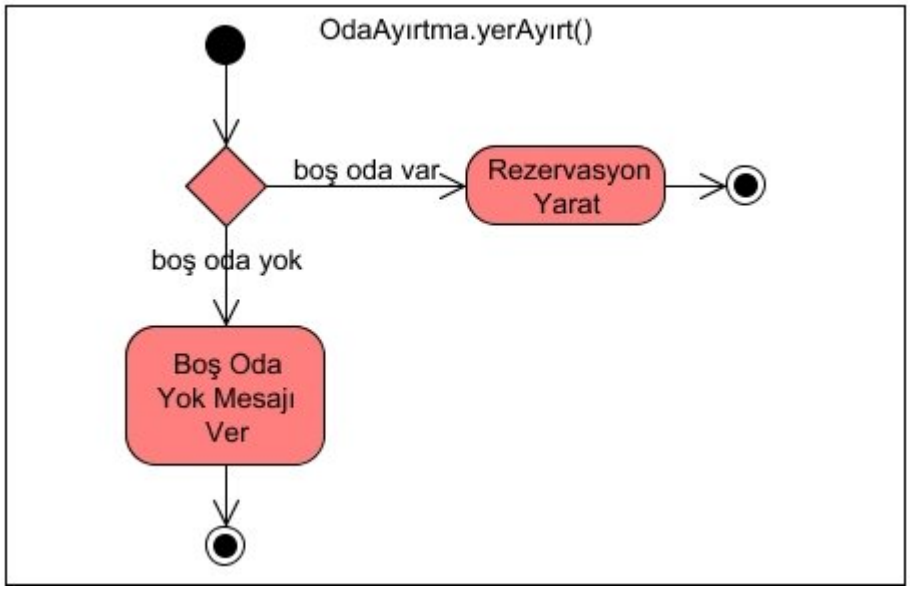
Kod 5.1. Müşteri Kaydetme ilgisinin aspect ile gerçekleştirimi.

Kod örneğinde Müşteriyi Kaydetme isimli yeni bir aspect yaratılmış ve tiparası bildirimler eklenmiş. Bir tiparası bildirimin başlıca iki kısmı bulunmaktadır. İlk kısmı mevcut bir sınıf ismi, ikinci kısmı eklenen metod ismidir. Örneğin Oda.odayaMüşteriAta() bildirimi odayaMüşteriAta() yöntemini Oda sınıfına eklemektedir.

Burada önemli nokta, yöntem sınıfın bir parçası olmasına rağmen, sınıfın dışında tanımlanmıştır. Böylece eş düzey ilgilerini ayrı tutacak şekilde bir kod düzenlemesi yapmak mümkün kılınmıştır.

5.2. Uzantı ilgilerinin ayrılması

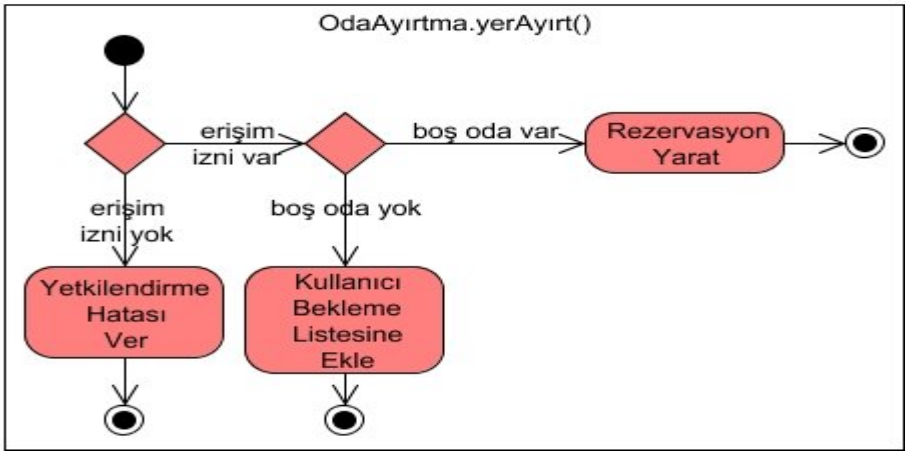
OdaAyırtma modülündeki yerAyırt() yöntemini düşünelim. Aşağıdaki şekilde bu yöntemin eklenti yapılmadan önceki etkinlik diyagramı gözükmektedir. Buna göre kullanıcı belirli tarihler arasında bir oda rezervasyonu yapmak istemektedir. Boş oda yok ise sistem, kullanıcıya boş oda yok mesajını verir. Eğer var ise rezervasyon yaratılır.



Şekil 5.2. Yer Ayırtma etkinlik diyagramı.

İşleme şu işlevleri ekleyelim:

- Kullanıcının kimliği doğrulanmamış ise yetkilendirme hatası oluşacak.
- Boş oda yok ise kullanıcılar bekleme listesine eklenecek



Şekil 5.3. Yer Ayırtmaya ek işlevler eklendikten sonraki etkinlik diyagramı.

Eklentiler doğrudan yer ayırtma ile ilgili olmadıklarından esas işlevi gerçekleştiren kodun anlaşılmasını zorlaştırmaktadır. Tavsiyeleri kullanarak bu durumu engelleyebiliriz.

```
public class OdaAyırtma {
   public final void yerAyırt throws OdaYokHatası {
      if (birOda.boşOdaVarmı()) {
         birRezervasyon.rezervasyonYarat();
      }else{
         throw new OdaYokHatası();
      }
   }
}
```

Kod 5.2. Yer Ayırtma yöntemi.

```
1. public aspect BeklemeListesi {
2.   pointcut rezervasyonYaparken() :
3.          execution(void OdaAyırtma.yerAyırt()) ;
4.
5.   after() throwing (OdaYokHatası e) : rezervasyonYaparken() {
6.      // kullanıcıyı bekleme listesine ekleyen kod
7.   }
8. }
```

Kod 5.3. BeklemeListesi aspecti.

Kod 5.3' de verilen aspectin 2 ve 3. satırlarında OdaAyırtma modülünün yerAyırt() yönteminin yürütümüne karşılık gelen rezervasyonYaparken nokta-kesimi tanımlanmıştır. 5, 6 ve 7. satırlarda ise bu nokta-kesiminin gösterdiği yöntemde OdaYokHatası oluştuğu zaman çalışacak tavsiye tanımlanmıştır. Tavsiye kodu kullanıcıyı bekleme listesine ekleyecektir.

Kod 5.4' de verilen aspect, OdaAyırtma modülünde herhangi bir işlem bilgi yapıldığında yetkilendirme kontrolü yapacaktır. Eğer kullanıcı yetkilendirilmemişse yöntem yürütülmeyecektir.

```
public aspect YetkilendirmeHatası {
   pointcut işlemGerçekleşirken():
      call(void OdaAyırtma.*(..));

   void around() : işlemGerçekleşirken() {
      if (erişimİzniVarmı()) {
         proceed();
      }
}}
```

Kod 5.4. YetkilendirmeHatası aspecti.

Aspect sayesinde mevcut yönteme yeni işlevler eklenmiş ve esas işlevin anlaşılması zorlaşmamıştır. BeklemeListesi ve YetkilendirmeHatası ilgileri OdaAyırtma ilgisinden ayrı tutulmuş olup, ilgi gerçekleştirimleri ayrı aspectlerde yapılmıştır.

AYP yazılım yaşam çevriminin sadece gerçekleştirim aşaması ile ilgilenir. Etkili bir ilgi ayrımı için isterlerin belirlenmesinden başlayarak her aşamada ilgilerin ayrı tutulması gerekmektedir.

6. KULLANIM-VAKASI DİYAGRAMLARI

Kullanım-vakaları sadece isterleri belirleme yöntemi değildir. Bütün yazılım yaşam çevrimini yönlendiren bir yazılım mühendisliği yöntemidir.

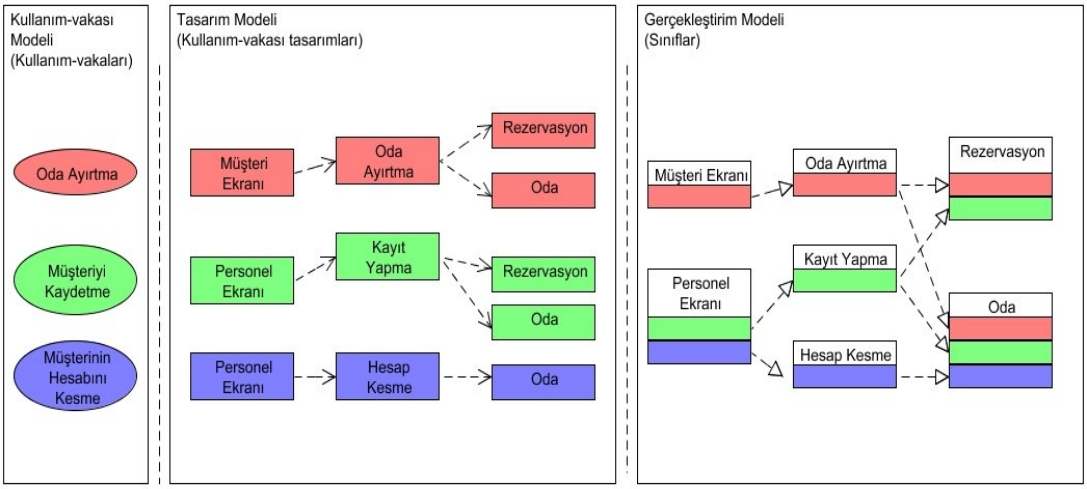
Kullanım-vakası güdümlü geliştirme, yazılım geliştirmenin model güdümlü olduğunu varsayar. Temelde kullanım-vakası modeli, tasarım modeli, gerçekleştirim modeli sırası izlenir ve her yinelemede şu faaliyetler gerçekleştirilir.

1)Kullanım-vakalarını bul ve her birini belirt.

2)Her bir kullanım-vakasını tasarla.

3)Her bir sınıfı gerçekleştir.

Kullanım-vakası modelinde dış perspektiften tanımlanan sistemin, tasarım modelinde iç yapısı gösterildikten sonra gerçekleştirim modelinde kodlaması yapılır. Sistemin iç yapısı tanımlanırken etkileşim diyagramları kullanılarak, hangi sınıfların katıldığı, nasıl etkileştikleri ve her bir sınıfın kullanım-vakasını gerçekleştirmek için hangi sorumlulukları aldığı gösterilir. (özellik, yöntem ve ilişkiler) Daha sonra her sınıfın sorumlulukları bir araya toplanarak sistemin gerçekleştirimi yapılır.

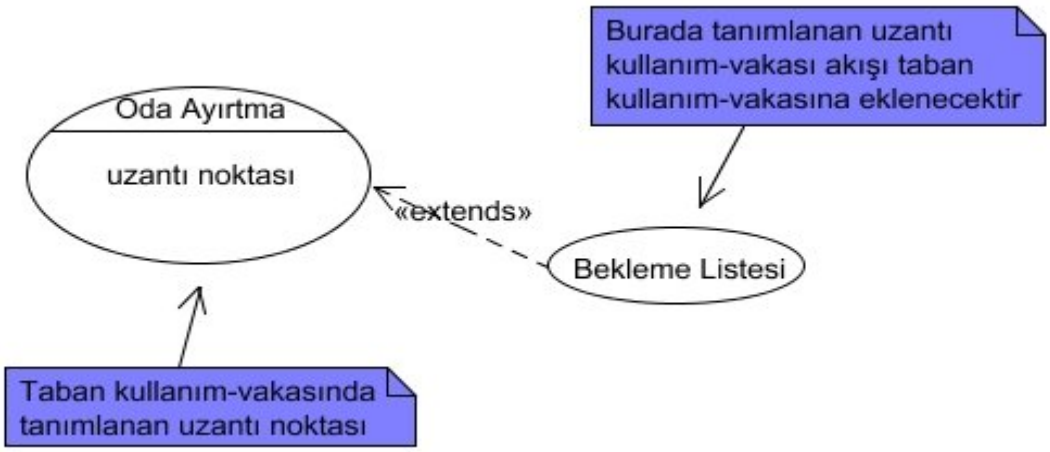


Şekil 6.1. Kullanım-vakası güdümlü geliştirmedeki modeller.

Geleneksel diller ilgi ayırımına destek vermediklerinden, kullanım-vakası tekniğinde de ilgilerle ilgili yeterli destek yoktur. Özellikle şu iki problem öne çıkmaktadır:

1) Şekil 6.1' de gözüktüğü gibi, isterlerin belirlenmesi aşamasında ayrı olan kullanım-vakaları, tasarım ve gerçekleştirimde bunu koruyamamışlardır. Bir kullanım-vakası tasarımı birden çok sınıfı etkilemekte (karışma) ve bir sınıf birden çok kullanım-vakası gerçekleştirimi içermektedir (saçılma).

2) Şekil 6.2' de OdaAyırtma modülüne bekleme listesi işlevinin eklenmesi modellenmektedir. Uzantı kullanım-vakası, uzantı noktasında taban kullanım-vakasına eklenecek bir eylem dizisi içermektedir. Bu eylemler dizisine *uzantı kullanım-vakası akışı* adı verilir. *Uzantı noktası*, kullanım-vakasının yürütümündeki herhangi bir noktadır.



Şekil 6.2. Uzantı kullanım-vakası.

Uzantı kullanım-vakalarını modele ekleyebilmek için taban kullanım-vakasında uzantı-noktasının belirtilmiş olması gerekmektedir. Fakat bu uzantıları ayrı tutma düşüncesiyle uyuşmamaktadır.

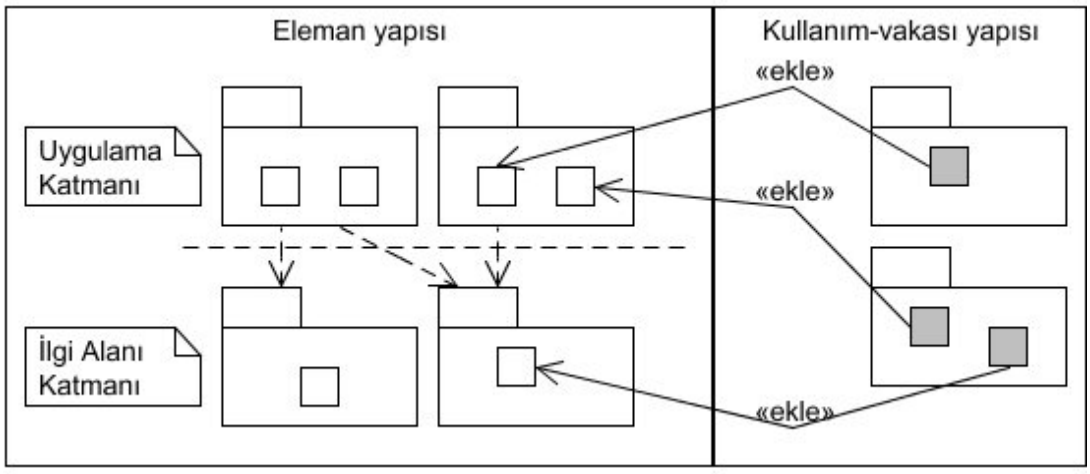
Gerçekleştirimde ayırım AYP ile sağlanabilse de, yukarıda gördüğümüz durumlardan ötürü aspectlerin tasarım modelinde belirlenip, temsil edilebilmeleri için yeni bir gösterim elemanına ihtiyacımız vardır.

7. KULLANIM-VAKASI KESİTLERİ

İlgilerin ayrı tutulabilmesi için her bir tasarıma özgü olan parçaların diğerlerinden ayrı tutulması gerekmektedir. Bunun için kullanım-vakası kesitlerini kullanırız. Kullanım-vakası kesitleriyle birlikte tasarım modeli iki yapıdan oluşur:

1)Sistemdeki elemanları tanımlayan *eleman yapısı.*

2)Elemanların içeriğini belirleyen kullanım-vakası kesitlerinden oluşan *kullanım-vakası yapısı.*

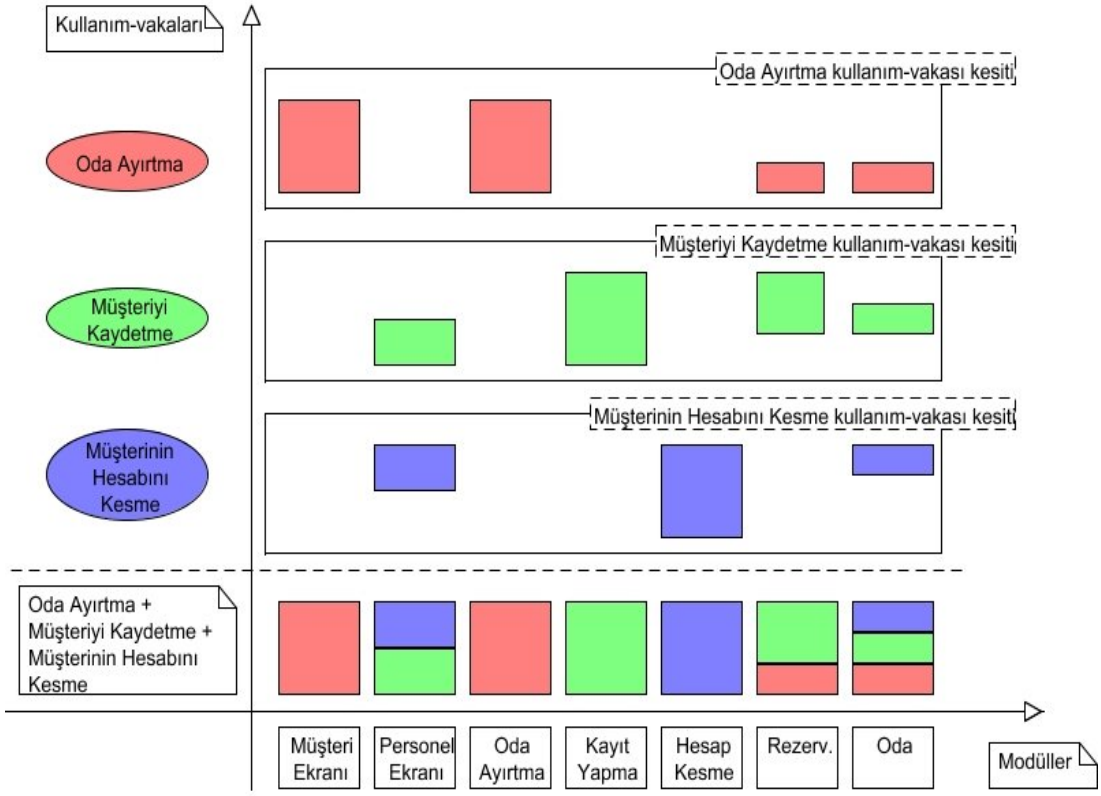


Şekil 7.1. Tasarım modeli yapısı.

Eleman yapısı katman, paket ve sınıflara göre düzenlenir ve elemanların modeldeki yerini tanımlamak için kullanılır. Elemanların içi kullanım-vakası kesitleri tarafından birleştirme sırasında doldurulacaktır.

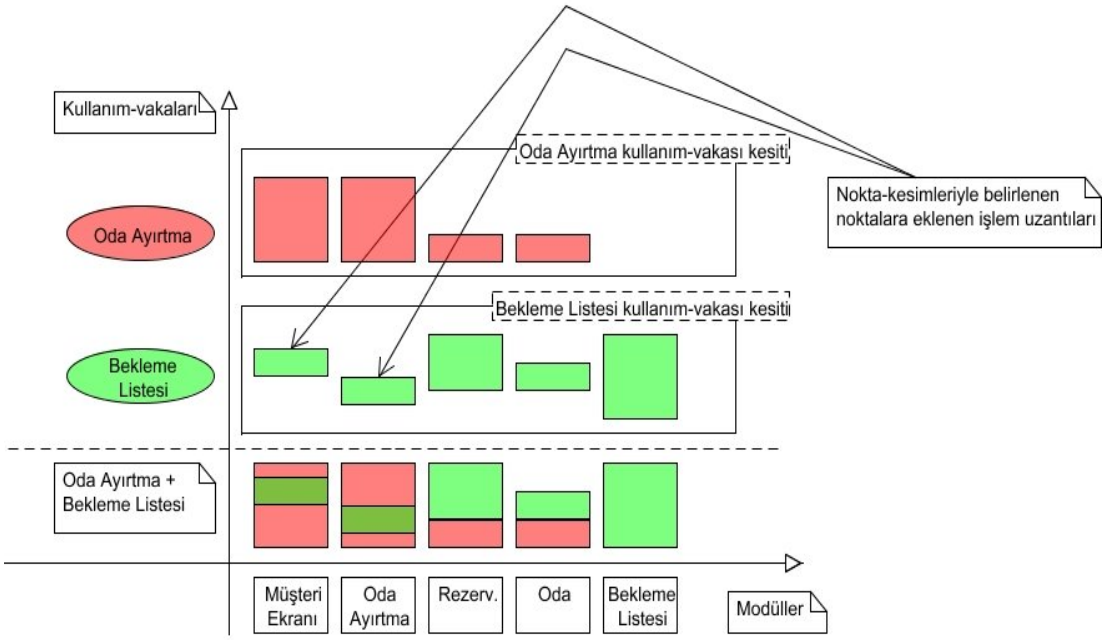
7.1. Eş düzey ilgilerinin ayrı tutulması

Aşağıdaki şekilde yatay eksen sistemdeki sınıfları tanımlayan eleman yapısını , dikey eksen ise gerçekleştirimi yapılan kullanım-vakalarını göstermektedir. Her bir yatay satır, o satıra ait kullanım-vakasının gerçekleştirimi için gereken sınıf parçalarından (sınıf uzantıları) oluşan kullanım-vakası kesitini belirtmektedir. Kullanım-vakası kesitleri bir araya getirilerek sistemi oluşturan sınıfların son hali elde edilebilir.



Şekil 7.2. Eş düzey ilgilerinin kullanım-vakası kesitlerine ayrılması.

7.2. Uzantı ilgilerinin ayrı tutulması



Şekil 7.3. Uzantı ilgilerinin kullanım-vakası kesitlerine ayrılması.

Şekil 7.3 iki tane işlem uzantısı içermektedir. Kullanım-vakası kesitleri birleştirilerek sınıfların bütünü elde edilebilir. Fakat uzantının eklendiği sırada mevcut yöntemin ne yaptığı belirtilmelidir.

8. KULLANIM-VAKASI TASARIMLARI

8.1. Eş düzey kullanım-vakalarının tasarlanması

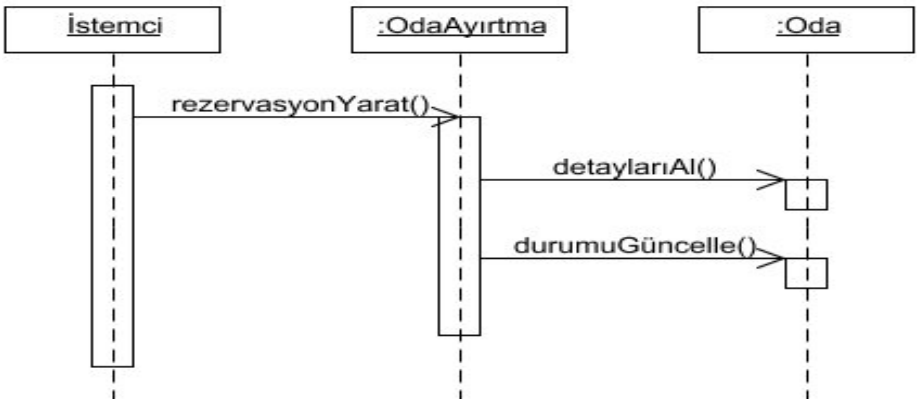
8.1.1. İşbirlikler

Kullanım-vakası tasarımları işbirlikleri ile modellenir. İşbirliği, bir görevi yerine getirmek için birlikte çalışan sınıfların rol ve sorumluluklarını tanımlar. Bunun için kullanım-vakası belirtimi incelenerek, ihtiyaç duyulan sınıflar bulunur. Örneğin OdaAyırtma tasarımında iki sınıfa ihtiyaç duyulmaktadır: OdaAyırtma ve Oda sınıfı.

- OdaAyırtma sınıfı kontrol birimi görevini yerine getirmekte ve diğer sınıfları kontrol etmektedir.

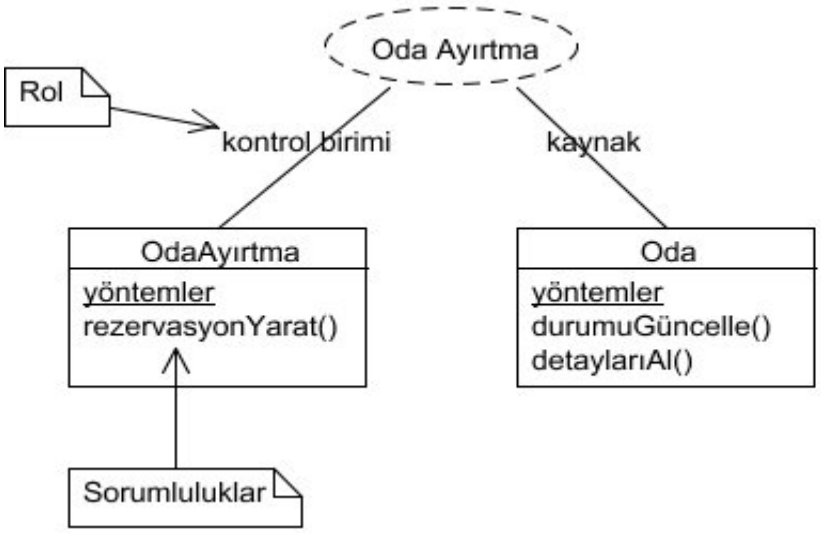
- Oda sınıfı ise ayırtılabilen kaynak rolündedir. Odanın durumu hakkında bilgiyi almak ve güncellemekle sorumludur.

Katılan sınıflar belirlendikten sonra, bu sınıflara görev dağılımı yapılır. Kullanım-vakası belirtiminin adımları üzerinden geçilerek sınıfların birbirlerini nasıl çağırdıkları saptanır. Bu şekil 8.1' deki gibi bir dizi diyagramı ile gösterilebilir.



Şekil 8.1. OdaAyırtma kullanım-vakası dizi diyagramı.

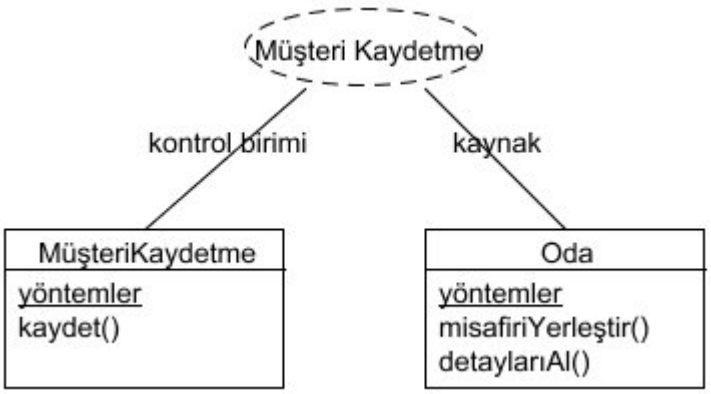
Sınıfların sorumlulukları belirlendikten sonra işbirliği ile gösterimleri yapılır.



Şekil 8.2. OdaAyırtma kullanım-vakası işbirliği.

8.1.2. Kullanım-vakası tasarımlarının örtüşmesi

Aşağıda işbirliği diyagramı gösterilen Müşteri Kaydetme kullanım-vakasını düşünelim:



Şekil 8.3. Müşteri Kaydetme işbirliği.

Oda sınıfındaki detaylarıAl() yöntemi hem OdaAyırtma hem de MüşteriKaydetme sınıflarında kullanılmıştır. Paylaşılan üyelerin kullanılması kullanım-vakası kesitleri arasında bağımlılığa neden olacaktır ve buda istenmemektedir. Bu yüzden ortak üyeler kullanım-vakası kesitlerinin dışında gösterilmelidirler.

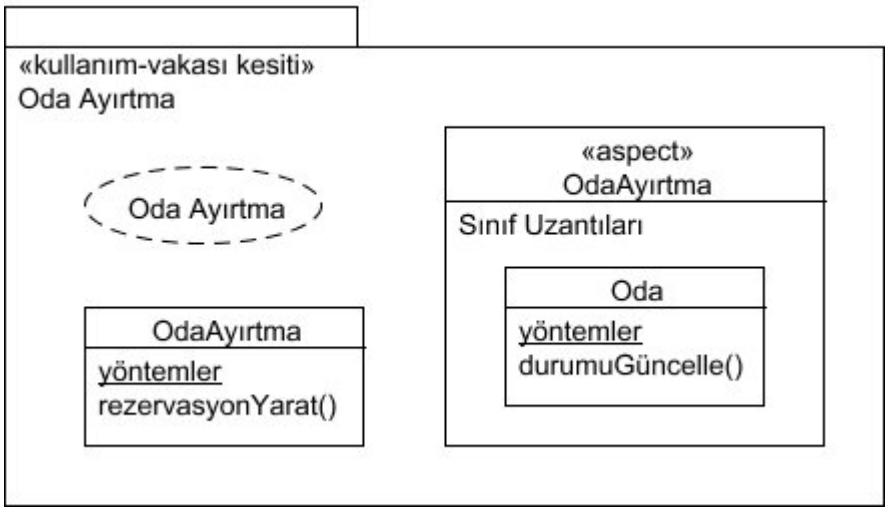
8.1.3. Tasarımların ayrı tutulması

Tasarım modelinde modülerlik birimi olarak kullanım-vakası kesitleri kullanılır. Bir kullanım-vakası kesiti şunları içerir:

1. İşbirliği.

2. Kullanım-vakasına özgü sınıflar.

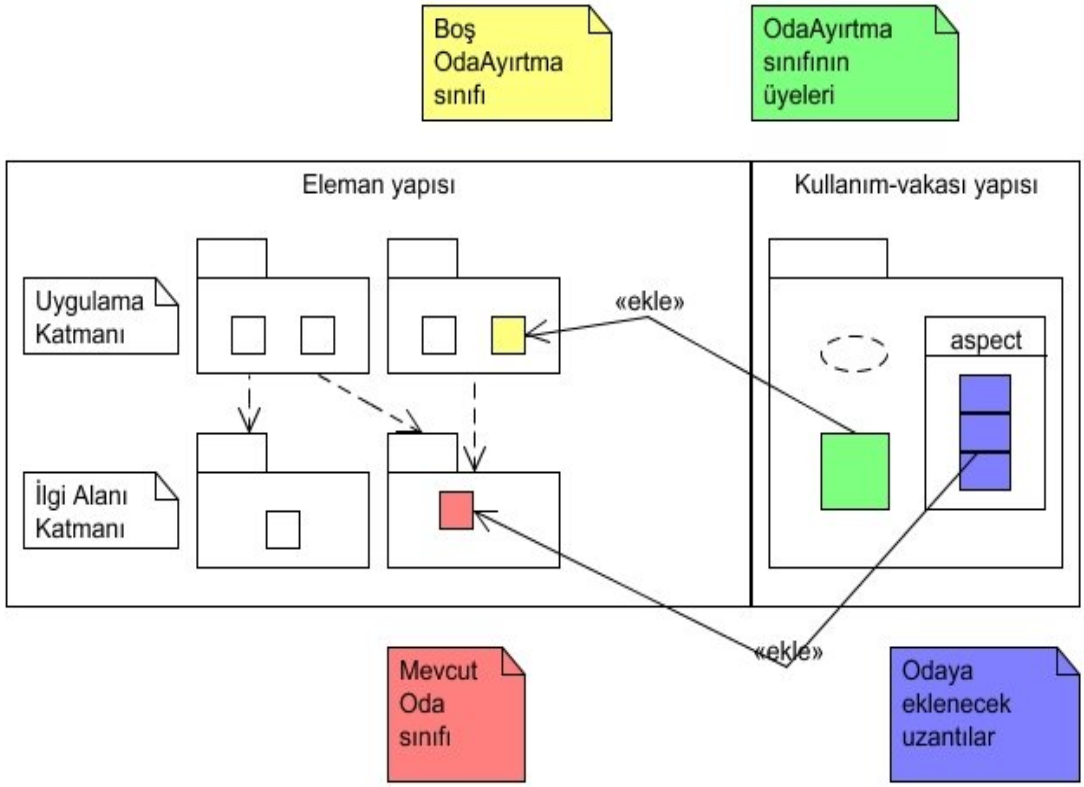
3. Kullanım-vakasına özgü sınıf uzantıları.



Şekil 8.4. OdaAyırtma kullanım-vakası kesiti.

OdaAyırtma kullanım-vakası kesitindeki işbirliği, tasarım için kullanılan etkileşim ve sınıf diyagramlarını simgelemektedir. OdaAyırtma sınıfı ve Oda sınıfındaki durumuGüncelle() sınıf uzantısı sadece OdaAyırtma kullanım-vakasına özgüdür.

OdaAyırtma sınıfı kullanım-vakası kesiti sisteme eklenirken oluşturulacaktır. Oda sınıfı ise sistemde birden fazla yerde kullanıldığından zaten daha önceden sisteme eklenmiştir ve içinde başka yöntemler mevcuttur.



Şekil 8.5. Kullanım-vakası kesitinin sisteme eklenmesi.

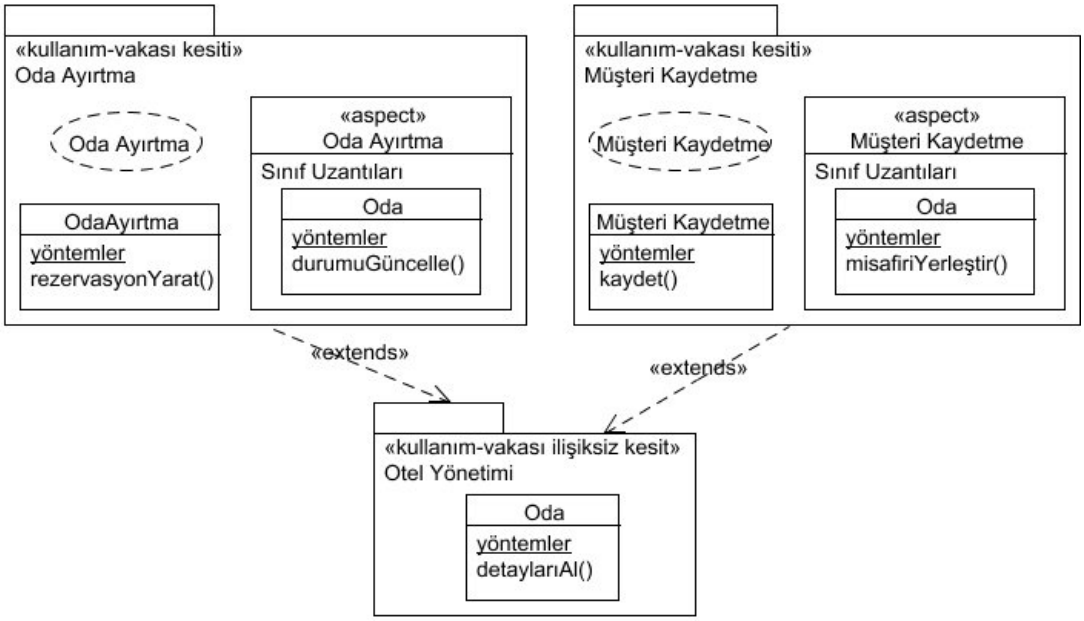
Sınıf uzantıları AYP' deki tiparası bildirim ve tavsiyeler kullanılarak sınıflara eklenir.

```
1. package app.müşteri;
2. import ilgialanı.oda.Oda;
3. public aspect OdaAyırtma {
4.     public void Oda.durumuGüncelle() {
5.         // oda durumunu güncelleyen kod
6.     }
7. }
```

Kod 8.1. Sınıf uzantılarının sisteme eklenmesi.

8.1.4. Kullanım-vakası ilişiksiz kesitler

Oda sınıfı birkaç tasarım arasında paylaşıldığından herhangi bir kullanım-vakasına özgü değildir. Bu yüzden kullanım-vakası ilişiksiz kesitler ile gösteririz.

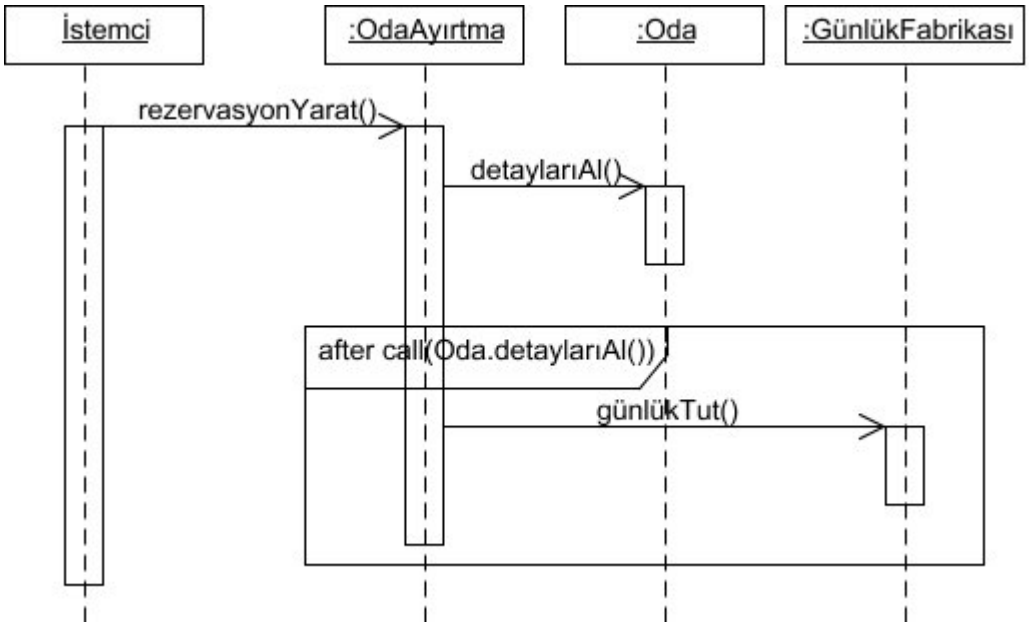


Şekil 8.6. Kullanım-vakası ilişiksiz kesitlerin kullanımı.

Kullanım-vakası ilişiksiz kesitler tabanı tanımlamaktadırlar ve mevcut sınıflara bir eklenti eklemezler. Bu kesitler genellikle bir ilgi alanını temsil ederler ve sistemdeki ortak sınıfları tanımlarlar. Ortak sınıflar aspectlerin yaptığı eklentiler ile son hallerini alırlar.

8.2. Uzantı kullanım-vakalarının tasarlanması

OdaAyırtma kullanım-vakasında her bir oda türüne gelen taleplerin sayısı tutulmak istenmektedir. Bu amaçla Oda sınıfındaki detaylarıAl() yöntemi çağrıldıktan sonra GünlükFabrikasındaki günlükTut() yöntemi yürütülecektir. Aşağıdaki şekilde uzantının hangi noktada kullanım-vakasına ekleneceği gösterilmektedir.



Şekil 8.7. OdaAyırtma kullanım-vakasına günlük tutma işlevinin eklenmesi.

8.2.1. İşlem uzantıları

İşlem uzantısı bildiriminin genel yapısı şu şekildedir:

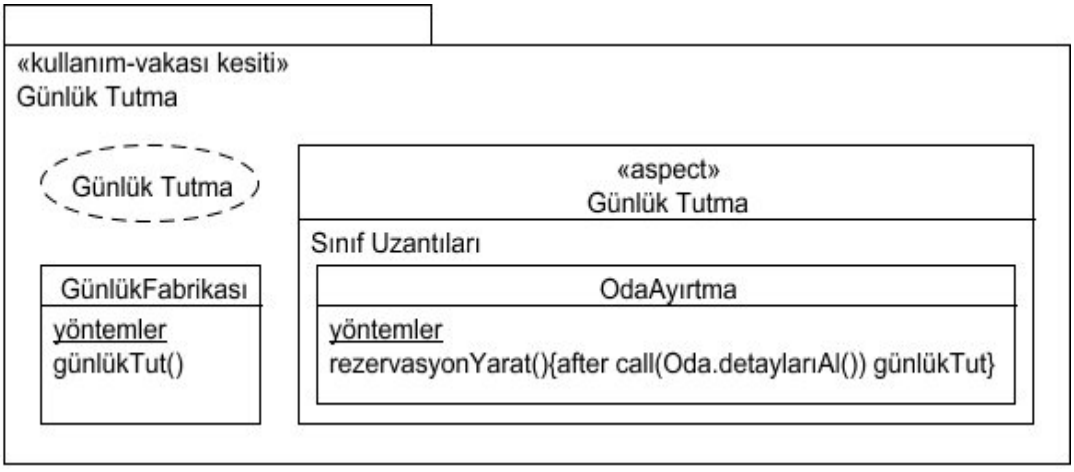
yapısal bağlam + davranışsal bağlam + işlem uzantısı.

1. Yapısal bağlam:

Eleman yapısında işlem uzantısının ekleneceği yeri belirtir. Mesela hangi paket, sınıf ya da yöntemde gibi. Örneğimizde eklenti OdaAyırtma sınıfının rezervasyonYarat() yönteminde çalışacaktır.

2. Davranışsal bağlam:

İşlemin hangi noktasında işlem uzantısının devreye girdiğini belirler. Örneğimizde eklenti Oda sınıfının detaylarıAl() yöntemi çağrıldıktan sonra çalışacaktır.



Şekil 8.8. Günlük Tutma kullanım-vakası kesiti.

Şekilde görüldüğü gibi işlem uzantısı, sınıf uzantısının içinde yer almaktadır. Bu sayede uzantının yapısal bağlamı daha iyi anlaşılmaktadır.

Şekil 8.8' deki Günlük Tutma aspectinin gerçekleştirimi Kod 8.2' de gösterilmiştir:

```
1. package altyapı.günlük;
2. import app.müşteri.OdaAyırtma;
3. import domain.oda.Oda;
4.
5. public aspect GünlükTutma{
6.     after() : withincode(void OdaAyırtma.rezervasyonYarat())
7.                     &&  call(void Oda.detaylarıAl())  {
8.         // günlük tutan kod
9.     }
10. }
```

Kod 8.2. GünlükTutma aspecti.

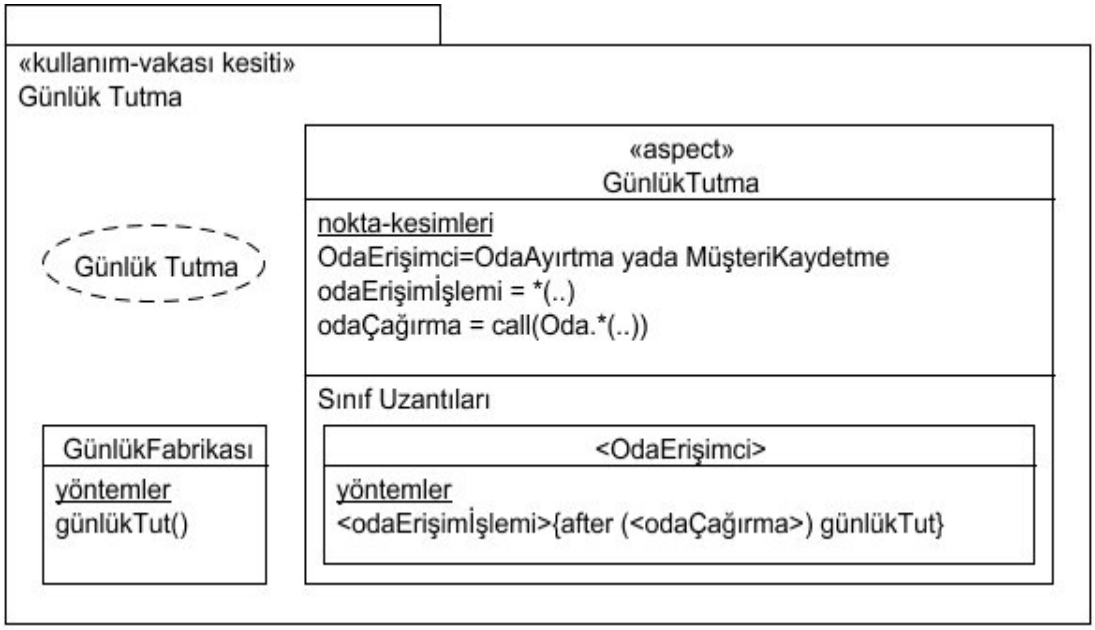
6, 7, 8 ve 9. satırlarda nokta-kesimlerinin yakaladığı birleşme-noktalarından sonra çalışacak olan tavsiye tanımlanmıştır. Nokta-kesimi belirtimi aslında && imgesi ile ayrılmış iki ayrı nokta-kesiminden oluşmaktadır. İmgeden önceki kısım işlem uzantısının yapısal, sonraki kısım ise davranışsal bağlamını ifade etmektedir.

8.2.2. Nokta-kesimlerinin parametrelenmesi

Aspectlerde bağlam belirten kısımların değişikliklere olan hassasiyetini azaltmak için, bu ifadelerin yerine parametre kullanırız. Bu sayede ifadeler sınıf uzantılarının dışına çıkartılmış olur.

Ayrıca bir işlem uzantısını birden çok tasarımdaki birden çok uzantı noktasında ya da aynı tasarımdaki birden çok uzantı noktasında yürütmek isteyebiliriz. Bu gibi durumlarda nokta-kesimi tanımlamalarını joker karakter ve düzenli ifadeler ile parametreleriz.

Günlük tutma örneğinde, sadece OdaAyırtma sınıfındaki rezervasyonYarat() yöntemi içinde Oda sınıfındaki detaylarıAl() yöntemi çağrıldıktan sonra, günlük tutulmaktaydı. Halbuki birden çok sınıftaki herhangi bir yöntem, Oda sınıfındaki herhangi bir yöntemi çağırabilir.



Şekil 8.9. Nokta-kesimlerinin parametrelenmesi.

Kesitte <OdaErişimci>, <odaErişimİşlemi> ve <odaÇağırma> isimleriyle üç tane parametre tanımlanmıştır. Parametrelerin tanımlamaları nokta-kesimleri bölümünde yapılmıştır.

Şekilde hiyerarşik bir yapısal oluşum vardır. Günlük tutma aspecti, <odaÇağırma> yapan <odaErişimİşlemi> ni içeren <OdaErişimci> yi barındırmaktadır. Her bir parametre kapsadığı parametre için yapısal bağlamı belirlemektedir.

Örneğin odaÇağırma nokta-kesimi kendi başına Oda sınıfındaki herhangi bir yöntem talebi için geçerlidir. Fakat aynı zamanda bu çağrıların OdaAyırtma ya da MüşteriKaydetme sınıfındaki herhangi bir yöntemden gelmesi gerekmektedir.

Şekil 8.9' daki GünlükTutma aspectinin gerçekleştirimi Kod 8.3' de gösterilmiştir:

```
public aspect GünlükTutma{
        pointcut OdaErişimci():
                within(OdaAyırtma) || within(MüşteriKaydetme);

        pointcut odaErişimİşlemi() :
                OdaErişimci() && withincode(* *(..));

        pointcut odaÇağırma() :
                odaErişimİşlemi() && call(* Oda.*(..));

        after() : odaÇağırma()  {
                // günlük tutan kod
        }
}
```

Kod 8.3. Parametrelerin aspect içinde gerçekleştirimi.

AspectJ de sınıf uzantıları diye bir kavram olmadığından, hiyerarşik yapının oluşturulabilmesi için "mantıksal ve işletmeni" kullanılmaktadır. Bu yüzden nokta-kesimi ifadeleri uzamaktadır. "odaÇağırma" nokta-kesimi kısaltılarak şu şekilde de ifade edilebilir:

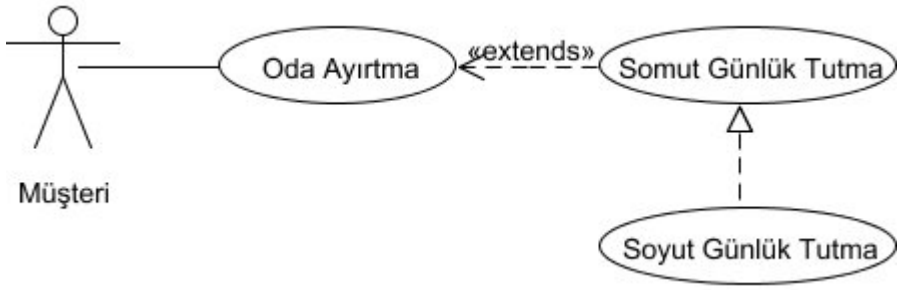
```
pointcut odaÇağırma():
        (within(OdaAyırtma)||within(MüşteriKaydetme))
        && call(* Oda.*(..));
```

Kod 8.4. Nokta-kesimi ifadesinin kısaltılması.

Fakat ifadeler kısaldıkça, nokta-kesimlerinin hangi noktaları gösterdiğini anlamak zorlaşmaktadır. Bu yüzden yapısal ve davranışsal bağlamları vurgulayan bir şekilde ifadelerin yazılması okunabilirliği yükseltecektir.

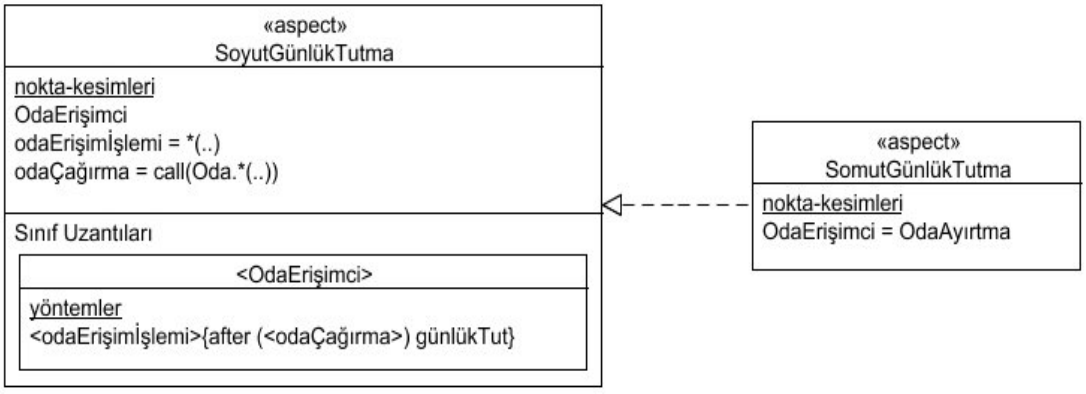
8.2.3. Tasarımların genelleştirilmesi

Günlük tutma uzantısının davranışı çeşitli kullanım-vakaları için aynı olsa da, her bir kullanım-vakası için farklı uzantı noktaları gerekmektedir. Bu amaçla uzantı noktaları, soyut olarak tanımlanır. Soyut günlük tutma aspecti bir kullanım-vakasına uygulanmak istendiğinde, nokta-kesimleri tanımlanarak somut bir hale getirilir.



Şekil 8.10. Günlük Tutma uzantısının genelleştirilmesi.

Eğer soyut ve somut kullanım-vakalarını tasarlarsak, karşılık gelen aspectleri buluruz:



Şekil 8.11. GünlükTutma aspectinin genelleştirilmesi.

SoyutGünlükTutma aspectinde tanımsız bırakılan OdaErişimci nokta-kesimi, SomutGünlükTutma aspectinde OdaAyırtma olarak tanımlanır. Aspectlerin gerçekleştirimi şu şekilde yapılır:

```
public aspect SomutGünlükTutma extends SoyutGünlükTutma {

        pointcut OdaErişimci():
                within(OdaAyırtma) ;
}
```

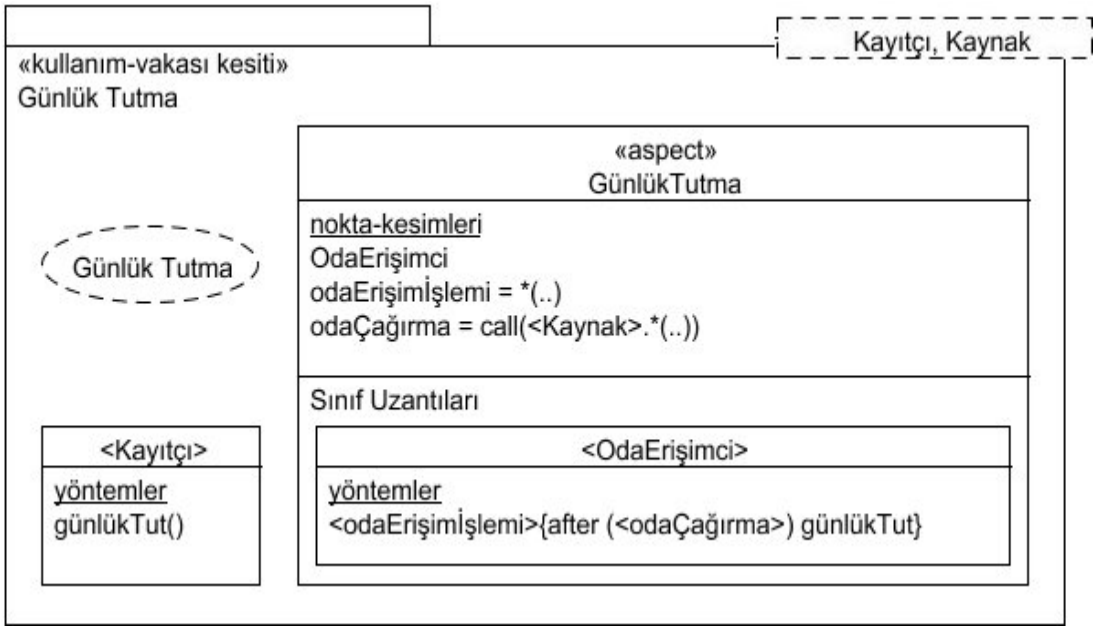
Kod 8.5. SomutGünlükTutma aspectinin gerçekleştirimi.

```
public abstract aspect SoyutGünlükTutma {

        abstract pointcut OdaErişimci();

        pointcut odaErişimİşlemi() :
                OdaErişimci() && withincode(* *(..)) ;

        pointcut odaÇağırma() :
                odaErişimİşlemi() && call(* Oda.*(..)) ;

        after() : odaÇağırma() {
                // günlük tutan kod
        }
}
```

Kod 8.6. SoyutGünlükTutma aspectinin gerçekleştirimi.

8.2.4. Kullanım-vakası kesitlerinin şablon olarak kullanılması

Aspectleri birden çok uzantı noktasına uygulamanın diğer bir yolu da, şablonların kullanımıdır. Bu kullanım C++ ya da Java programlama dilindeki şablonlarla benzeşmektedir. Bu sayede AYP' nin sadece mevcut sınıflara uygulanabilme sınırlaması kalkar.



Şekil 8.12. Günlük tutma şablonu.

oda Erişim İşlemi ve oda Çağırma parametreleri nokta-kesimleri ile tanımlanmıştır. Oda Erişimci parametresinin tanımı ise somut alt sınıfa bırakılmıştır. Ayrıca iki tane kullanım-vakası kesit parametresi tanımlanmıştır: Kaynak ve kayıtçı.

Şablon uygulandığı zaman parametrelerin yerini verilecek değerler alacaktır. Bizim örneğimizde kaynağın yerini Oda, kayıtçının yerini ise Günlük Fabrikası sınıfı alacaktır.

9. ESNEK MİMARİLERİN GELİŞTİRİLMESİ

9.1. Esnek Mimariler

Esnek bir mimari, performans ve güvenlik gibi sistemin genelini etkileyen ilgileri karşılamalı ve her bir parçasının hangi ister ya da kullanım-vakasına karşılık geldiği anlaşılabilmelidir. Sınıflar sadece tanımlı olan rolleri yönünde sorumluluklar içermelidir ve sınıflar arasında sorumluluk örtüşmesi olmamalıdır.

Sistemin bir kısmındaki değişiklik, başka kısımları etkilememelidir. Eğer mimariyi genişletmek gerekiyor ise, etki asgari olmalı ve daha önce çalışmakta olan her şey çalışmaya devam etmelidir. Aspect yönelimi ile farklı ilgiler etkili bir şekilde ayrılabilir.

9.1.1. İşlevsel İsterlerin Ayrılması

Bir sistem birçok işlevsel isteri karşılamalıdır. Eşanlı geliştirme için ise, farklı işlevsel isterlere karşılık gelen ilgiler ayrı tutulmalıdır. Bu ayırım sistemin anlaşılır ve kolay bakımlı olmasını sağlar. İşlevsel isterler, kullanıcıların sistemle neler yapabilecekleriyle ilgili olduklarından, kullanım-vakaları ile etkili bir şekilde ayrılırlar. Fakat bu ayırımın gerçekleştirime kadar gitmesi için kullanım-vakasına, uygulamaya ve ilgi alanına özgü olan kısımlar ile cinsine özgü olan kısımların belirlenmesi gerekmektedir. Bu, sınıfların katmanlara, paketlere, sınıf uzantılarına ve kesitlere ayrılmasına taban teşkil eder.

Bazı işlevsel isterler diğer isterlerin uzantısı olarak tanımlanır. Her bir iterasyonda veya sürümde sisteme ek fonksiyonlar olarak eklenirler. Eklenen uzantılar mevcut olanlardan farklı ilgileri gerçekleştirmelidirler aksi takdirde yamadan farkları kalmaz.

9.1.2. İşlevsiz İsterlerin Ayrılması

Sistemler güvenlik, güvenilirlik ve performans gibi bazı işlevsiz isterleri karşılamalıdırlar. Bunlar bazı altyapı düzenekleri tarafından sağlanırlar. Örneğin güvenlik işlevsiz isteri için yetkilendirme, kimlik doğrulama ve şifreleme altyapı mekanizmaları, performans için önbellek ve yük dengelemesi mekanizmaları gibi.

Genellikle birden çok noktada kullanılırlar. Yeniden kullanılırlıklarını arttırmak için nokta-kesimleri parametrik olarak tanımlanır. Bu sayede işlevsel isterler ile işlevsiz isterler birbirlerinden ayrılmış olurlar.

9.1.3. Platforma Özgü Kısımların Ayrılması

Günümüzde sistemler sunum, iş ve bütünleşme mantığını ayıracak şekilde seviyelere ayrılırlar. Geliştirici uygulamanın yapması gerekenler yanında, bu mekanizmaların alt düzey detaylarını da bilmek zorundadır. Bu detaylar sistemin birçok yerinde tekrarlanır ve teknoloji seçimine bağlı olarak değişiklik gösterirler. Bağımlılığın azaltılması için platforma özgü kısımlar parametrelenmiş kullanım-vakası kesitleri

ile gösterilirler. Bu kesitler daha sonra platform bağımsız kullanım-vakası kesitlerinin üzerine bindirilirler. İlgi ayırımı için aspect yöneliminin yanında çeşitli tasarım modelleri de kullanılır.

9.2. Platform Bağımsız Yapılar

9.2.1. Tasarım

Platform bağımsız yapıların tasarlanması için, sisteme ilk önce platform bağımsız bir bakış açısıyla bakılır. Daha sonra platforma özgü kısımlar eklenir. Bunun için sınıflar ve kullanım-vakaları kullanılır. Platform bağımsız yapılar, eleman yapısı ve kullanım-vakası yapısı olmak üzere iki kısımdan oluşur:

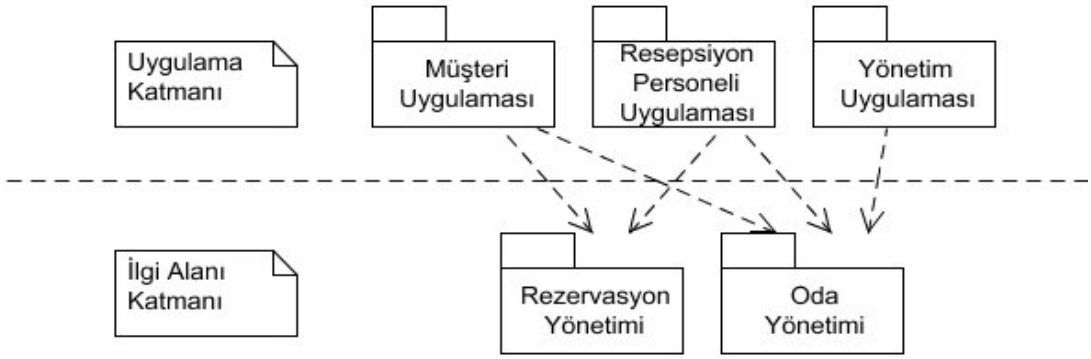
a) Eleman Yapısı

Eleman yapısı, paket ve sınıfların hiyerarşik olarak yapılanmasından oluşur. Her bir eleman benzersiz olarak tanımlanabilir (paket). Aynı soyutlama seviyesinde olan elemanlar bir arada olacak şekilde, sistem katmanlara bölünür. Daha soyut ve tekrar kullanılabilir olan elemanlar daha aşağı katmanlarda, daha somut olanlar ve daha az tekrar kullanılabilir olanlar daha üst katmanlara koyulur. İşlevsel isterlerin ayrıştırılabilmesi için iki katman yetmektedir: Uygulama ve ilgi alanı katmanı.

Uygulama katmanı, kullanım-vakalarındaki iş akışlarını gerçekleyen elemanları içermektedir. Bu katmandaki elemanlar, ilgi alanı katmanındaki elemanları kullanarak kullanım-vakalarını gerçeklerler. Aktör, kullanım-vakası, işlevsel alan gibi ölçütlere göre düzenlenebilirler.

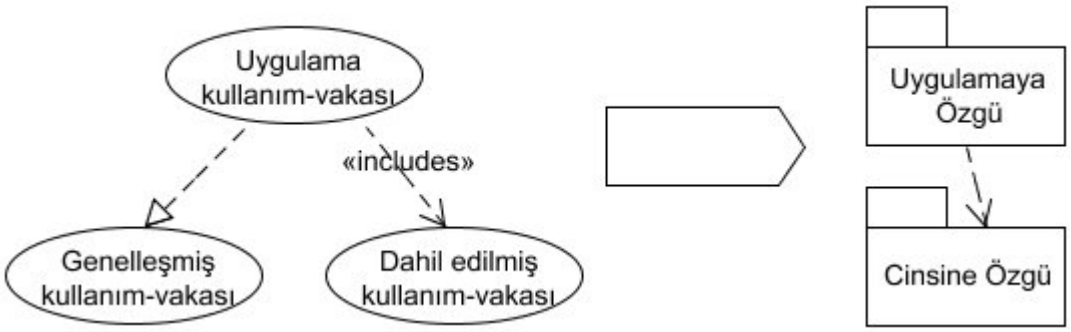
İlgi alanı katmanı, önemli ilgi alanı kavramlarını temsil eden elemanları içerir. Bu elemanlar sistem tarafından idare edilecek ya da işlenecek bilgileri tutarlar.

Aşağıdaki şekilde Otel Yönetim Sisteminin eleman yapısı gösterilmektedir. Uygulama katmanındaki paketler aktörlere göre düzenlenmiştir. İlgi alanı katmanındaki paketler ise birbiriyle alakalı sınıfları bir arada göstermiştir.



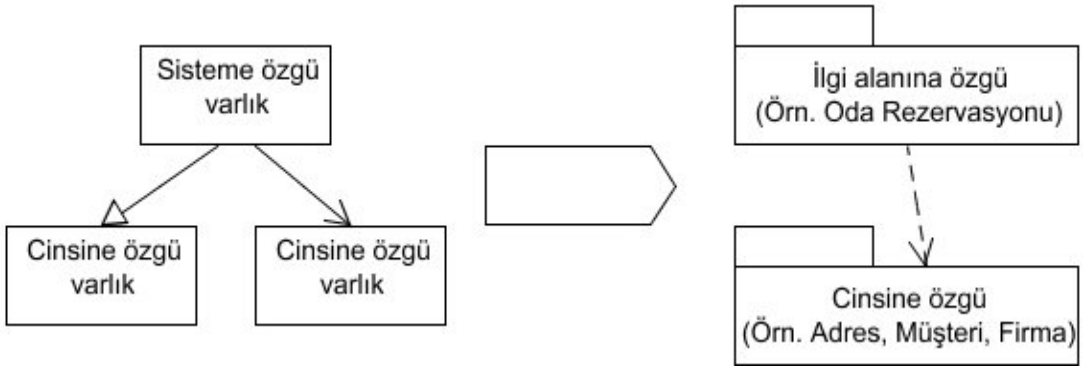
Şekil 9.1. Otel Yönetim Sistemi eleman yapısı.

Şekil 9.1' deki ayırım ufak çapta sistemler için uygundur. Daha büyük sistemler için aşağıdaki yöntemler uygulanabilir.



Şekil 9.2. Uygulama katmanı düzenlenişi.

Burada uygulama katmanı, uygulamaya özgü ve cinsine özgü şeklinde ayrılmıştır. Eş düzey kullanım-vakalarının gerçeklenmesinde yer alan sınıflar uygulamaya özgü paketinde, genelleşmiş veya dahil edilmiş sınıflar ise cinsine özgü paketinde yer almaktadır.

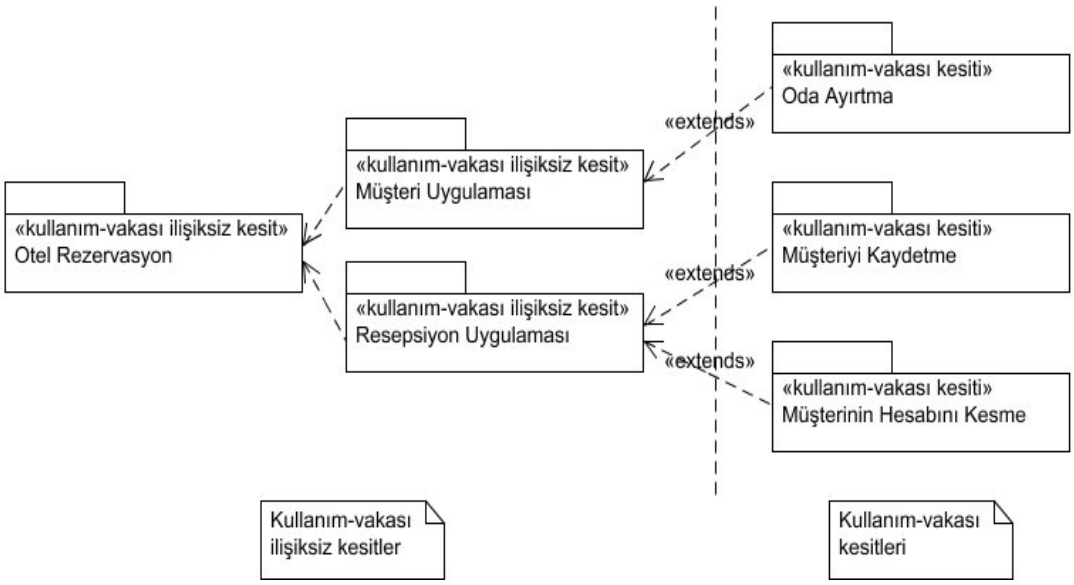


Şekil 9.3. İlgi alanı düzenlenişi.

Bu gösterimin temelinde ilgi alanları arasında ortak olan varlıkların cinsine özgü paketinde toplanması vardır. Örneğin Adres, Müşteri ve Firma birçok ilgi alanında kullanılır. Daha sonra ilgi alanına özgü olan varlıklar bunları özelleştirerek ya da referans olarak kullanırlar.

b) Kullanım-vakası Yapısı

Eleman yapısındaki statik yapı, kullanım-vakası yapısındaki kesitler tarafından doldurulur. İki tür kesit vardır: Kullanım-vakası kesitleri ve kullanım-vakası ilişiksiz kesitler.



Şekil 9.4. Otel Yönetim Sistemi kullanım-vakası yapısı.

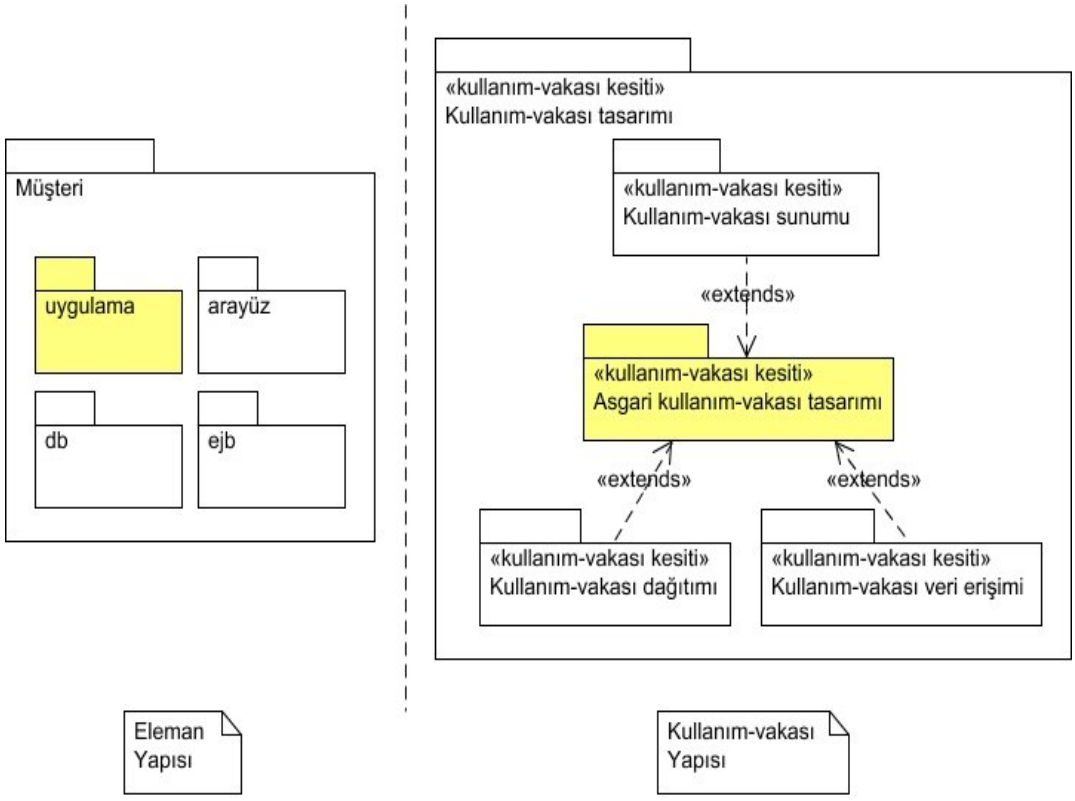
Kullanım-vakası ilişiksiz kesitler, kullanım-vakaları tasarımlarındaki benzeşmeler incelenerek türetilir. Eleman yapısının alt katmanlara karşılık gelir ve paylaşılan kısımları gruplar.

Şekil 8.4' de Otel Rezervasyon ilgi alanı paketlerini, Müşteri ve Resepsiyon Uygulamaları ise uygulama katmanının cinsine özgü paketlerini dolduracaktır.

Kullanım-vakası kesitleri kullanım-vakası modelindeki kullanım-vakalarına birebir karşılık gelirler. Şekilde "OdaAyırtma", "Müşteri Kaydetme" ve "Müşteri Hesabını Kesme" kesitleri gösterilmiştir.

9.2.2. Platforma Özgü Kısımların Bindirilmesi

Kullanıcı arayüzü, dağıtım, veri erişimi gibi platforma özgü kısımlar kullanım-vakası tasarımlarından ayrı tutulur. Böyle tasarımlara *asgari kullanım-vakası tasarımları* denir. Bu sayede platformla ilgili kısımlar sistemin diğer kısımlarından bağımsız olarak geliştirilir.



Şekil 9.5. Platforma özgü kısımların ayrı tutulması.

10. ASPECTLER VE ÖTESİ

Başarılı bir yazılım geliştirme için paydaşların ihtiyaçları değiştikçe, sistemde artımlı olarak gelişebilmelidir. Her bir ister diğerlerinden bağımsız bir modül ile temsil edilebilmeli, değişiklikler ise diğer modülleri etkilememelidir. Bu kullanım-vakası kesitleri tarafından yönlendirilen aspectler ile sağlanır. Kullanım-vakası güdümlü aspect yönelimli sistemlerde anlaşılabilirlik, yeniden kullanılırlık, sınanabilirlik kolaylaşır ve bunların sonucu olarak bakım maliyeti, kalitesi ve süresi çarpıcı bir biçimde gelişir.

REFERANSLAR

[1] K. Erdil, E. Finn, K. Keating, J. Meattle and S. Park, Deborah Yoon, "Software Maintenance As Part of the Software Life Cycle", 2003, (ref. 2006 Kasım), Available HTTP: http://hepguru.com/maintenance/Final_121603_v6.pdf

[2] J.Koskinen, "Software Maintenance Costs", 12 Sept. 2003, (ref. 2006 Kasım), Available HTTP: http://www.cs.jyu.fi/~koskinen/smcosts.htm

[3] Wikipedia, "Module", (ref. 2006 Kasım), Available HTTP: http://en.wikipedia.org/wiki/Module

[4] Wikipedia, "Cohesion", (ref. 2006 Kasım), Available HTTP: http://en.wikipedia.org/wiki/Cohesion_(computer_science)

[5] Wikipedia, "Coupling", (ref. 2006 Kasım), Available HTTP: http://en.wikipedia.org/wiki/Coupling_(computer_science)

[6] K. E. Wiegers, Software Requirements, 2nd Edition ed. , Microsoft Press, 2003.

[7] B. Tekinerdogan, "Aspect-Oriented Software Development", (ref. 2006 Kasım), Available HTTP: http://trese.cs.utwente.nl/taosad/aosd.htm

[8] Wikipedia, "Concern", (ref. 2006 Kasım), Available HTTP: http://en.wikipedia.org/wiki/Concern_(computer_science)

[9] Wikipedia, "Aspect-oriented Programming", (ref. 2006 Kasım), Available HTTP: http://en.wikipedia.org/wiki/Aspect-oriented_programming

[10] R. Laddad, AspectJ in Action, Manning Publications Co., 2003.

[11] I. Jacobson, P. Ng, Aspect-Oriented Software Development with Use Cases, Addison Wesley Professional, 2004.

[12] A. Colyer, A. Clement, G. Harley and M. Webster, Eclipse AspectJ: Aspect-Oriented Programming with AspectJ and the Eclipse AspectJ Development Tools, Addison Wesley Professional, 2004.

[13] IEEE Computer Society, Guide to the Software Engineering, Angela Burgess, 2004.

[14] M. Fowler, UML Distilled, 3rd ed. , Addison Wesley Professional, 2003.

[15] K. Bittner, I. Spence, Use Case Modeling, Addison Wesley Professional, 2002.

[16] G. Övergaard, K. Palmkvist, Use Cases Patterns and Blueprints, Addison Wesley Professional, 2004.

[17] D. Alur, J. Crupi, D. Malks, Core J2EE™ Patterns: Best Practices and Design Strategies, 2nd Edition ed. , Prentice Hall PTR, 2003.

ÖZGEÇMİŞ

Tez yazarı, 1979 yılında doğmuş olup, bilgisayar bilimi ile 1998 yılından beri ilgilenmektedir. 2002 yılından itibaren yazılım sektöründe çalışan yazar, yazılım geliştirme işini daha etkili kılan teknolojilerle ilgilenmektedir. Tek başına nesneye yönelik programlamanın yetersiz olduğuna inanan yazar bu konuda sektöre yönelik araştırmalar yapmaktadır.

EK-A: TERİMLERİN İNGİLİZCE KARŞILIKLARI

TERİMLER	
Türkçe	**İngilizce**
bağlaşım	coupling
aspect yönelimli	aspect oriented
bileşen	component
bütünleşme	integration
cinsine özgü	generic
çapraz-kesen	cross-cutting
davranışsal	behavioral
dizi diyagramı	sequence diagram
düzenli ifade	regular expression
eş düzey	peer
etkileşim diyagramı	interaction diagram
etkinlik diyagramı	activity diagram
gerçekleştirim	implementation
güdümlü	driven
ilgi	concern
ilgi alanı	domain
ister	requirement
işbirliği	collaboration
işlembilgi	transaction
karışma	tangle
kesit	slice
kimlik doğrulama	authentication
kullanım-vakası	use-case
mantıksal ve işletmeni	logical and operator
nokta-kesimi	pointcut
öznitelik	feature
saçılma	scatter

TERİMLER	
Türkçe	**İngilizce**
somut	concrete
soyut	abstract
şablon	template
tasarım	design
tavsiye	advice
tiparası bildirim	intertype declaration
uyum	cohesion
uzantı	extension
yapısal	structural
yazılım yaşam çevrimi	software life cycle
yetkilendirme	authorization
yöntembilim	methodology
yürütmek	execute