YZM 3017
Yazılım Tasarımı ve Mimarisi

Prof. Dr. Hamdi Tolga KAHRAMAN
Arş. Gör. M. Hakan BOZKURT
Arş. Gör. Sefa ARAS

Yazılım Tasarımı ve Mimarisi

UML (Unified Modeling Language)

Uml Nedir?

- UML yazılım sisteminin önemli bileşenlerini tanımlamayı, tasarlamayı ve dokümantasyonunu sağlayan grafiksel bir modelleme dilidir
- Yazılım geliştirme sürecindeki tüm katılımcıların (kullanıcı, iş çözümleyici, sistem çözümleyici, tasarımcı, programcı,...) gözüyle modellenmesine olanak sağlar.
- UML gösterimi nesneye dayalı yazılım mühendisliğine dayanır
- Grady Booch , James Rumbaugh ve Ivar Jacobson tarafından geliştirilmiştir.

Yazılım Tasarımı ve Mimarisi

UML (Unified Modeling Language)

Uml Faydaları

- Yazılım sistemi herhangi bir kod yazmadan önce profesyonelce tasarlanır ve dokümantasyonu yapılır
- Yeniden kullanılabilir kod parçaları kolaylıkla ayırt edilir ve en yüksek verimle kodlanır
- Daha düşük geliştirme maliyeti
- Tasarımdaki mantıksal boşluklar tasarım çizimlerinde kolaylıkla saptanabilir
- Daha az sürpriz – yazılımlar beklendiğimiz şekilde davranırlar
- Tüm tasarım kararları kod yazmadan verilir
- UML "resmin tamamını" görmemizi sağlar
- Daha etkin ve performanslı kod yazımı açısından faydalıdır.
- Sistemde değişiklik yapmayı kolaylaştırır.
- Ortak çalışılan projelerde programcıların iletişimi daha kolay hale gelir. Çünkü UML ile programımızı parçalara ayırdık ve parçalar arasında bir ilişki kurduk.
- Tasarım aşaması düzgün yapıldıysa tekrar kullanılabilen kodların sayısı artacaktır. Buda program geliştirme maliyetini büyük ölçüde düşürecektir.

Yazılım Tasarımı ve Mimarisi

UML MODELLERİ VE İLİŞKİLİ DİYAGRAMLAR

- UML de bir sisteme farklı açılardan yaklaşılıp bunun sonucunda yapılan modellemeyi dökümante edebilecek farklı diyagramlar söz konusudur. Aşağıda UML diyagramlarına ait bir kategorik sınıflandırma yapılmaya çalışılmıştır.

- **Dinamik Davranışsal (Behavioural) Modelleme:** *Söz konusu sistemin davranışlarını konu almaktadır.*
  - Sequence (Etkileşim) Diyagramları
  - Communication (İletişim) Diyagramları
  - State (Durum) Diyagramları
  - Activity (Faaliyet) Diyagramları
  - Timing (Zamanlama) Diyagramları (Real Time Sistemlerde kullanılır.)
- **Statik, Yapısal (Structural) Modelleme:** *Sistemin yapısını ortaya koyar. Davranıştan ziyade yapısal özelliklere odaklanır.*
  - Class (sınıf) Diyagramları
  - Object (Nesne) Diyagramları
  - Deployment (Dağıtım) Diyagramları
  - Composite Structure (Bileşke Yapı) Diyagramları
  - Component (Bileşen) Diyagramları
- **İşlevsel (Functional) Modelleme:** *Söz konusu sistemin yapabildiği işleri ve bunları sistem dışı varlıkların nasıl kullandığını anlatır.*
  - Use Case (Kullanım Senaryosu) Diyagramları.

Yazılım Tasarımı ve Mimarisi

UML MODELLERİ VE İLİŞKİLİ DİYAGRAMLAR

- UML, nesneler arasında ilişki kurmak için bir takım grafiksel elemanlara sahiptir. Bu elemanları kullanarak diyagramlar oluşturulur. UML temel olarak aşağıdaki diyagram türlerini kapsar:

- **Sınıf (Class) Diyagramları:** Sınıf, aynı işlevlere, aynı ilişkilere ve aynı anlama sahip nesneler topluluğunun ortak tanımıdır. Sınıflar yazılımın durağan yapısının tanımlanmasında kullanılırlar.
- **Nesne (Object) Diyagramları:** Nesne, sınıfın bir örneğidir. Bu tür diyagramlarda sınıfın yerine her bir sınıftan oluşturulmuş nesneler yer alır.
- **Durum (State) Diyagramları:** Gerçek nesnelerin herhangi bir zaman içindeki durumunu ve durumunun zaman içinde nasıl bir değişim gösterdiğini modelleyen diyagramlardır. Genel olarak durum diyagramları tüm nesneler için değil yalnızca karmaşık olan, davranışı, kendine gönderilen iletilerin yanısıra o an içinde bulunduğu duruma göre de farklılık gösteren nesneler için oluşturulur.
- **Ardıl Etkileşim (Sequence) Diyagramları:** Sınıf ve nesne diyagramları durağan bilgiyi modeller. Ancak gerçek zamanlı sistemlerde zaman içinde değişen durumlar bu diyagramlarla gösterilemez. Bu tür zamanla değişen durumları belirtmek için nesnelerin birbirleriyle zamana bağlı olarak haberleşmelerini ele alan ardıl etkileşim diyagramları kullanılır.

Yazılım Tasarımı ve Mimarisi

UML MODELLERİ VE İLİŞKİLİ DİYAGRAMLAR

- **Etkinlik (Activity) Diyagramları:** Bir nesnenin durumu zamanla kullanıcı tarafından ya da nesnenin kendi içsel işlevleri tarafından değiştirilebilir. Bu değişim sırası etkinlik diyagramlarıyla gösterilir.
- **Kullanıcı Senaryosu (Use Case) Diyagramları:** Programın davranışının bir kullanıcı gözüyle incelenmesi kullanıcı senaryosu diyagramlarıyla yapılır. Gerçek dünyada insanların kullanacağı bir sistemde bu diyagramlar büyük önem taşırlar.
- **İş Birliği (Collaboration)Diyagramları:** Bir çok parçadan oluşan projelerde bir işin amacına uygun şekilde çalışabilmesi için bütün parçaların işlevlerini eksiksiz yerine getirmesi gerekir. Bu parçalar arasındaki ilişki iş birliği diyagramlarıyla gösterilir.
- **Bileşen (Component) Diyagramları:** Özellikle çok sayıda kişinin çalıştığı büyük çaplı projelerde, projeyi bileşenlerine ayırmak gerekmektedir. Sistemin doğru modellenmesiyle bileşenlerin ayrı ayrı çalışması sağlanmalıdır. Bu tür modellemeler bileşen diyagramlarıyla yapılmaktadır.
- **Dağıtım (Deployment) Diyagramları:** Dağıtım diyagramları yazılımın nasıl dağıtılacağının planlandığı aşamada kullanılırlar. Sistemin fiziksel incelemesini yapmaktadırlar. Bilgisayarlar arasındaki bağlantılar, programın kurulacağı makinalar, sistemdeki ağ ve yazıcı bağlantıları gibi her türlü detay dağıtım diyagramlarında gösterilir.
- **Paket (Package) Diyagramları:** Paket diyagramları, büyük yazılımlarda sistemi oluşturan alt yazılımlar veya etkileşimde bulunulan yan sistemler olduğu durumlarda sistemler arası etkileşimi gösteren kısaca sistem mimarisinin paket yönünü özetleyen bir diyagramdır.

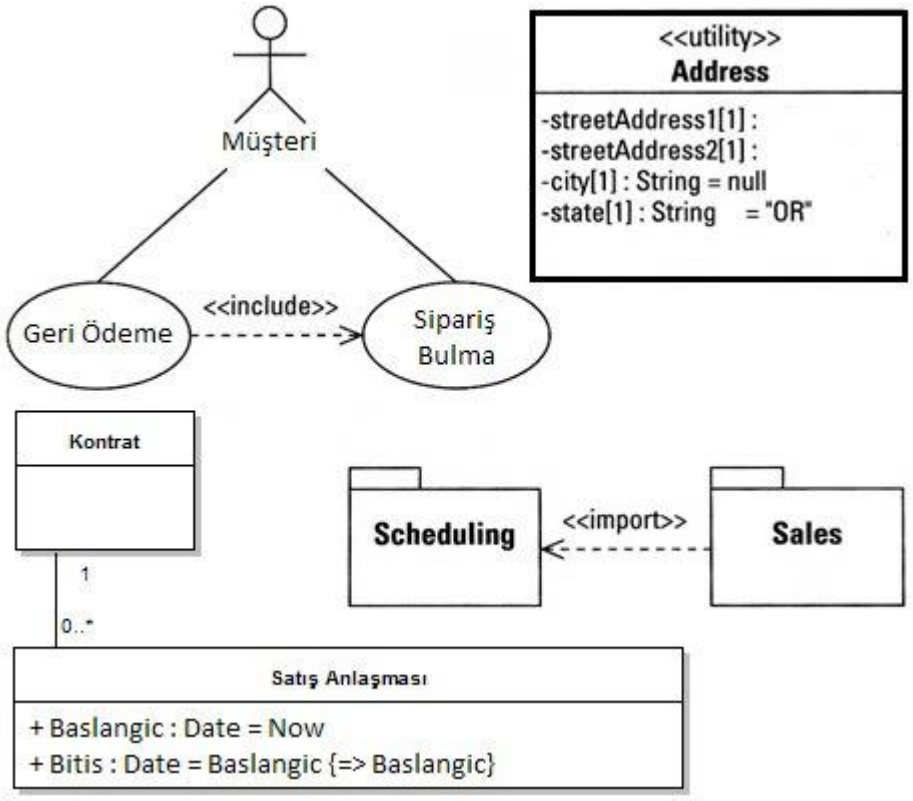
Yazılım Tasarımı ve Mimarisi

STEREOTYPE'lar: UML de sınıfların özel durumlarını belirtirler. <<stereotype>> şeklinde ifade edilirler. **<<interface>>** ya da **<<delegate>>** gibi bazıları ön tanımlıdır.

CONSTRAİNT (Kısıtlamalar) Mevcut notasyona yine farklı ve özel anlamlar kazandıran eklentilerdir. {constrsaint} şeklinde gösterilirler. Örneğin bir sınıfın abstract olduğunu ifadesi {abstract} bir constraint' dir.
{frozen} bir değişkenin asla değişmeyeceği anlamına gelir. {read only}' den farklıdır. Örneğin bir kişinin yaşı {read only} olsa da değişebilir. Ancak doğum tarihi asla değişmez {frozen}. {frozen} UML' de ön tanımlı bir kısıtlamadır. {read only} ön tanımlı değildir. Ayrıca constraint' ler ile semantik bir koşul ya da kısıtlama da ifade edebilir..

Örneğin telefon numarası bilgisinde en fazla 20 karakter olabileceği ya da sadece sayı ve + ile – işaretlerinin bulunabileceği bir kısıtlama bu yolla oluşturulabilir Sağ tarafta Kontrat ve Satış anlaşması sınıflarında buna benzer bir constraint kullanılmıştır.
OCL ise **Object Constraint Language** anlamında olup constraint'lerdeki karmaşık ifadelerin yazımını sağlar.
Örnekler : <<entity>>, <<exception>>, <<enum>>, <<union>>, <<servlet>>, <<webservice>>, {abstract}, {concreate}, {read only} gibi.

TAGGED VALUE yani etiketlenmiş değer ise bir tür anahtar-değer çifti olup anahtarlara özel tag denilmektedir. Kullanım amaçları herhangi bir model elemanına ilave özellik kazandırmaktır.
Örneğin;
{Geliştirici = 'XXXX", Versiyon = ''2.5"}
{Hız = ''2 GHZ", Çekirdek = ''4", Tur=''64 bit"}



Yazılım Tasarımı ve Mimarisi

USE CASE DİYAGRAMLARI

- Daha çok analiz aşamasında aktivite diyagramları ile birlikte kullanılan use case diyagramları aktörler, use caseler ve aralarındaki ilişkilerden yola çıkılarak oluşturulan çizimlerdir. Bu çizimler sistemin fonksiyonel (operasyonel) gereksinimlerini sisteme dışarıdan bakarak net bir biçimde tanımlamaya ve sistemin kullanıcılarına neleri sunduğunu ortaya çıkartmaya yardımcı olur.

- Use case diyagramının oluşturulması kapsamın belirlenmesinde fayda sağlar böylece yazılım ekininin iş yükünün baştan netleştirilmesinde yararlıdır.

- Use case i sistemde yapılan bir işin tanımı olarak belirtebiliriz. Örneğin veritabanına bir kullanıcı eklenmesi veya satış raporlarının alınması gibi.

- Use caselerin belirlenmesindeki yöntem “Sistemde yapılması gerekli işlemler nelerdir” sorusuna uygun cevabın bulunmasıdır.

- bir parçası olamazlar. Aktörler sistemin tamamı ya da bir kısmı ile etkileşime geçen varlıklardır.

Yazılım Tasarımı ve Mimarisi

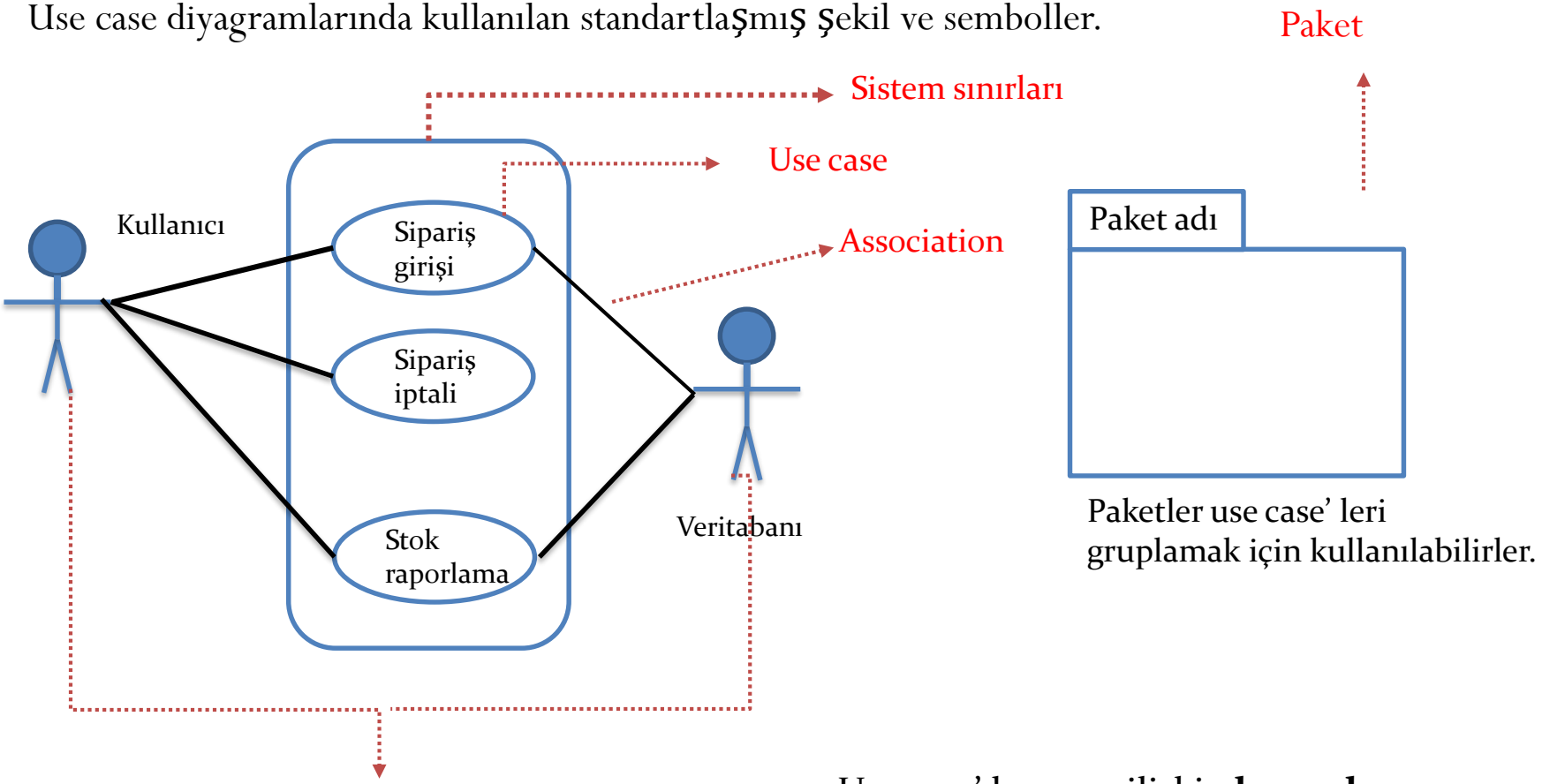
USE CASE DİYAGRAMLARI

- Sistemde yapılacak işler nelerdir – Use Case
- Bu sistemi kimler ya da neler kullanacak ? – Aktörler (Örn. Kullanıcı, veritabanı, harici sistemler ya da donanımlar.
- Aktörler ve use case ler arası ilişki çokludur.
- Bir aktör birden fazla use case i kullanabileceği gibi bir use case i de birden fazla aktör kullanabilir.
- Aktörler genelde yetki düzeylerine veya türlerine göre sınıflandırılırlar
  1. Esas aktörler : Ana sistem fonksiyonunu kullanan aktörler
  2. İkincil aktörler : Yönetim ve bakım gibi ikincil düzeydeki işleri yapan aktörler
  3. Harici Donanımlar
  4. Diğer Sistemler: Etkileşimde bulunulan diğer sistemler.
- Aktörler sistemin bir parçası olamazlar. Aktörler sistemin tamamı ya da bir kısmı ile etkileşime geçen varlıklardır.

Yazılım Tasarımı ve Mimarisi

Use Case Diyagramları

Use case diyagramlarında kullanılan standartlaşmış şekil ve semboller.

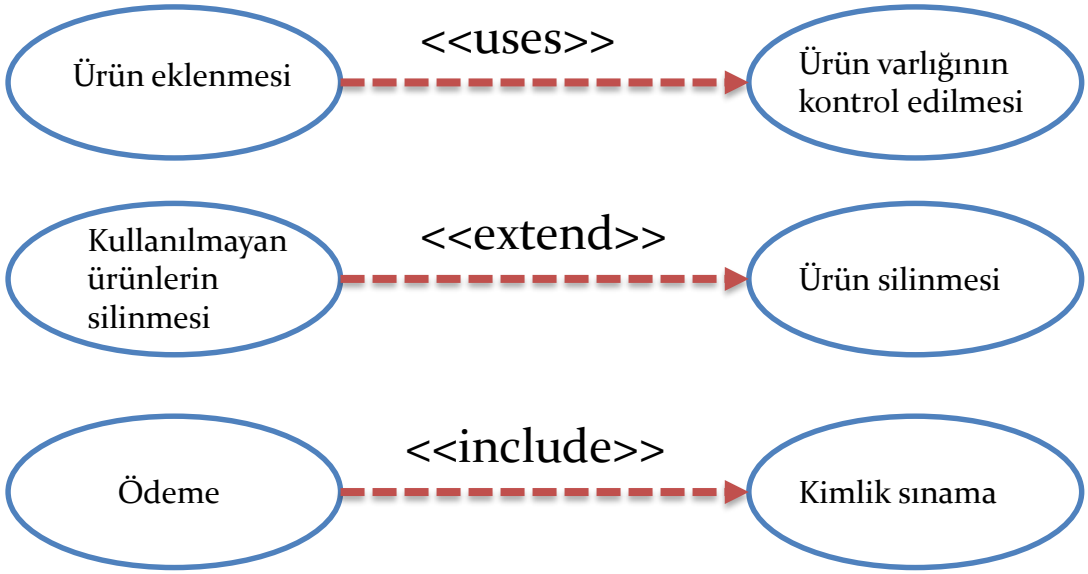


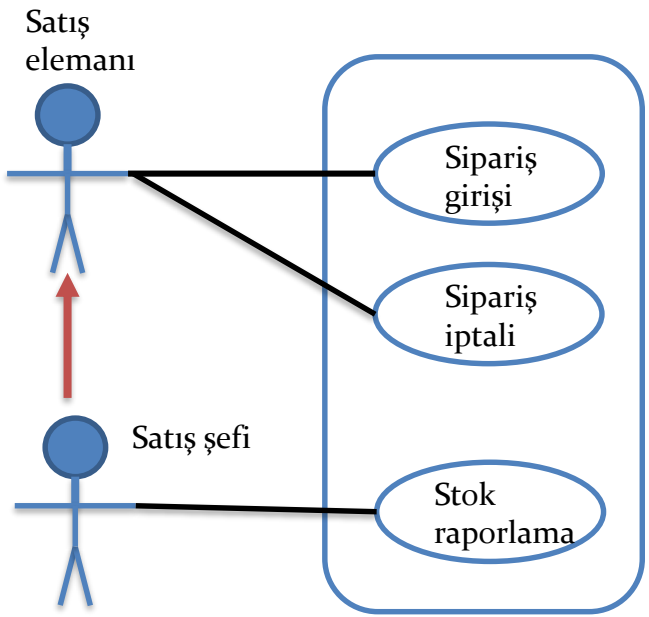
Paketler use case' leri gruplamak için kullanılabilirler.

- Use case' ler arası ilişki: **dependency** (bağımlılık). Kesik çizgi ile ifade edilir.
- Use case ile aktör arası ilişki: **association**. Düz çizgi ile ifade edilir.

USE CASE' LER ARASINDAKİ İLİŞKİLER

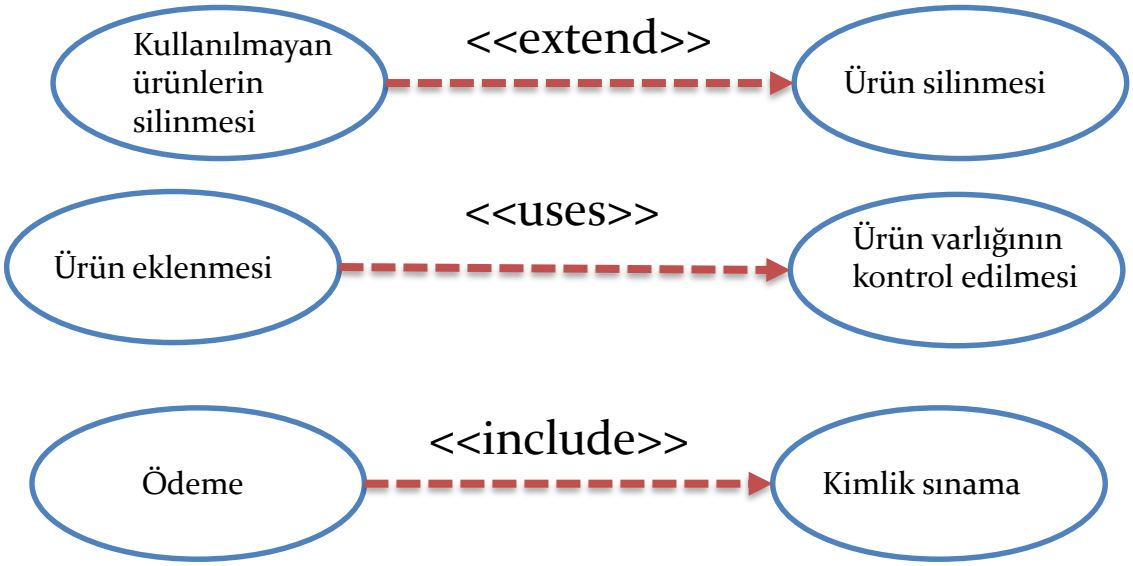
Modellenen sistemlerde birbirinin varyasyonu ya da birbirini içeren use case' ler bulunabilmektedir. Bu tür durumlarda use case' ler arası ilişki kurulabilir. Bu amaçla <<*uses*>>, <<*extend*>> ve <<*include*>> stereotype' ları kullanılabilir.

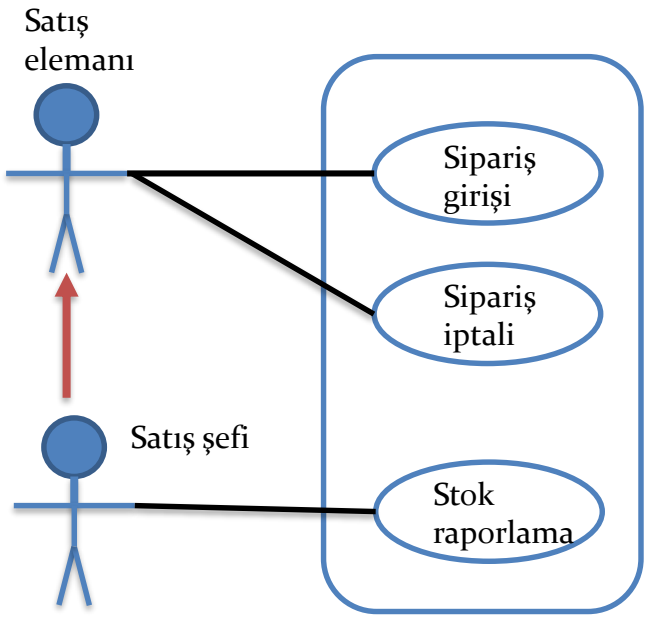




Yazılım Tasarımı ve Mimarisi

USE CASE' LER ARASINDAKİ İLİŞKİLER



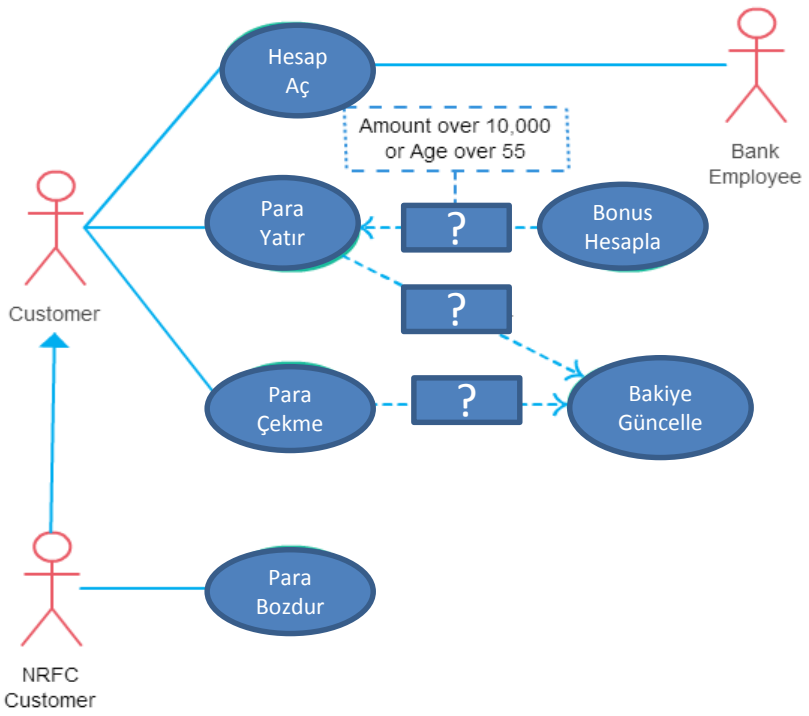


<<extend>> stereotype 'ı varyasyonel işlemlerin modellenmesinde kullanılır. Örneğin veritabanındaki kullanılmayan ürünlerin silinmesi işlemi bir ürünün silinmesinin farklı bir varyasyonudur.

<<include>> veya **<<uses>>** Bir use case' in diğerini içermesi veya işlemin içinde en az bir kez kullanılması. Örneğin ödeme işlemi kendi içerisinde kullanıcı kimliğinin sınanmasını içermesi. *(Son versiyon UML de <<include>> ve <<uses>> aynı anlamda kullanılmaktadır.)*

Aktörler arasında ilişki olabilir, aktörler birbirinden türeyebilir. Örneğin Sipariş girişi ve sipariş iptali yapabilen satış elemanı kullanılarak; satış elemanının yapabildiklerine ek olarak stok raporlama da yapabilen satış şefi türetilebilir. Kırmızı ok; satış şefinin, satış elemanından türetildiğini gösterir.

Yazılım Tasarımı ve Mimarisi



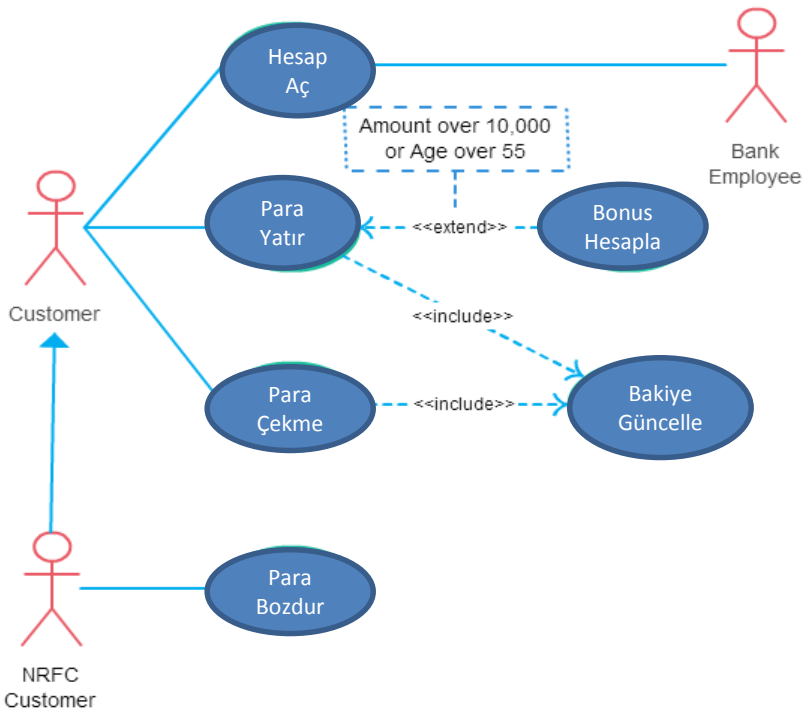
Opsiyonel Fonksiyonlar veya Ek Fonksiyonlar

- Bazen bir fonksiyonun duruma göre tetiklenmesi gerekebilir. Bu durumda <<extend>> ilişkisi, ilgili kural eklenerek kullanılabilir. Yukarıdaki örnekte bonus hesaplanması (calculate bonus) sadece verilen koşul sağlandığında aktifleşir (tetiklenir).

- **Örnek:** Yukarıdaki sistemdeki ilişkileri açıklayınız.

1. Aktörler arası ilişkiler.
2. Use case' ler arası ilişkiler.
3. Aktörler ve use case' ler arası ilişkiler.

Yazılım Tasarımı ve Mimarisi



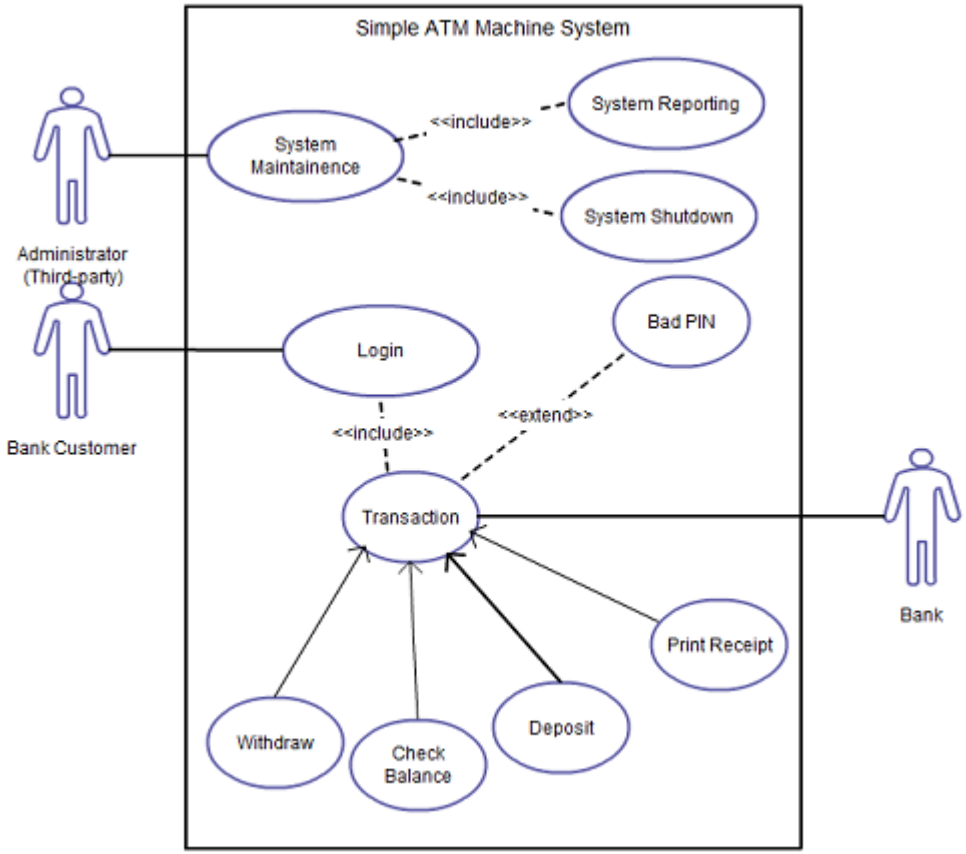
Opsiyonel Fonksiyonlar veya Ek Fonksiyonlar

- Bazen bir fonksiyonun duruma göre tetiklenmesi gerekebilir. Bu durumda <<extend>> ilişkisi, ilgili kural eklenerek kullanılabilir. Yukarıdaki örnekte bonus hesaplanması (calculate bonus) sadece verilen koşul sağlandığında aktifleşir (tetiklenir).

- **Örnek:** Yukarıdaki sistemdeki ilişkileri açıklayınız.

1. Aktörler arası ilişkiler.
2. Use case' ler arası ilişkiler.
3. Aktörler ve use case' ler arası ilişkiler.

Yazılım Tasarımı ve Mimarisi



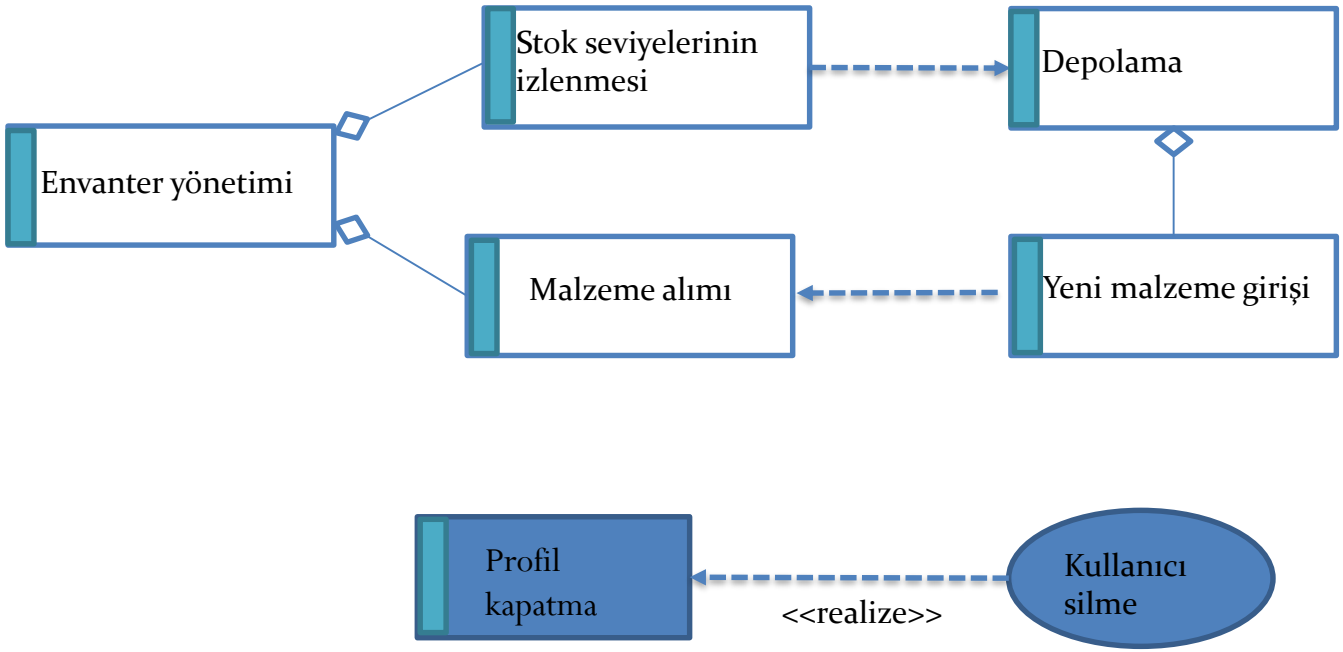
- **Örnek:** Yukarıdaki sistemi ve ilişkileri yorumlayınız.

Yazılım Tasarımı ve Mimarisi

Use Case Anlatımları

- Görsel olarak temsili use case' ler ile yapılan sistemlerin use case anlatımları (narrative) kullanılarak detaylı ancak teknik olmayan bir dille anlatılması gerekmektedir.
- Use case anlatımlarında sistemde yapılacak işler anlatılır, işin teknik olarak nasıl yapıldığı anlatılmaz.
- Örnek use case anlatımı
  - Sipariş Girişi:
  - Kullanıcı listeden seçtiği ürünün ID bilgisini kullanarak bir sipariş formu doldurur.
  - Sipariş formuna girilmesi gereken bilgiler ;
  - Müşteri isim ve soyismi
  - Adres
  - Teslimat tarihi
  - Ödeme türü bilgileridir.
  - Müşteri nakit ödeme yerine kredi kartı ile ödemeyi seçerse kart no ve cvc bilgilerini de girmek zorundadır. Söz konusu bilgiler veri tabanında ilgili tablolara kaydedilmelidir

Yazılım Tasarımı ve Mimarisi



Gereksinim Modelleme

Gereksinimler sol kenarlarında farklı renkte bir dikdörtgen daha olan ve içinde gereksinimin adı yazılı bir dikdörtgen ile sembolize edilir. Gereksinimlerin kendi aralarında ve/veya use case' ler ile aralarında ilişkiler olabilir.

Yazılım Tasarımı ve Mimarisi

Aktivite diyagramları

Aktivite diyagramları use case diyagramlarının tamamlayıcısı gibidir. Zira use case diyagramlarında sistemin neler yapabildiği ifade edilirken aktivite diyagramları sayesinde bu işlerin hangi aşamalardan geçilerek, nasıl yapıldığı ortaya konulur. Bu nedenle analiz aşamasında use case diyagramlarından hemen sonra çizilmelidir.

Aktivite diyagramlarında temelde yapılan şey; belirli bir iş sürecine dahil olan aksiyonlar arasındaki geçişleri göstermektedir. Bu anlamda oklar (flow veya path) aksiyonlar arasındaki geçişleri ifade eder. Şüphesiz bu geçişler belirli koşullara bağlı olabilir. Bu durumda yine baklava sembolü ile gerekli karar(desicion) ve kalıtımlar(merge) gösterilebilir.

Yazılım Tasarımı ve Mimarisi

AKTİVİTE DİYAGRAMI

Başlangıç düğümü

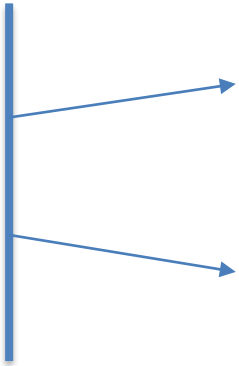
Aktivite sonu

Akış sonu

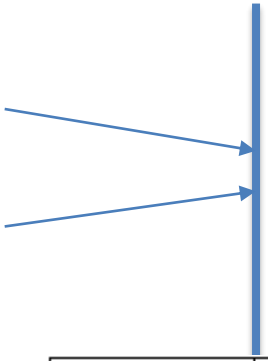
Karar yada Birleştirme düğümü (Aynı gösterime sahipler.)

Aksiyon

Akış

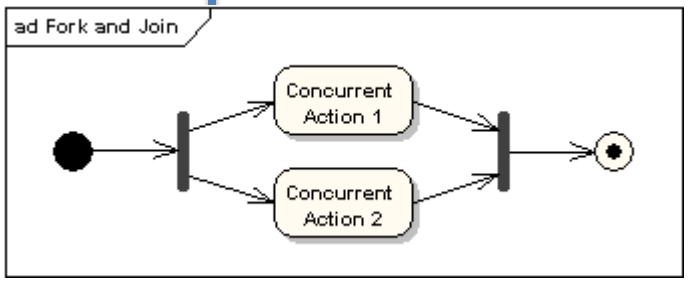


Çatallanma

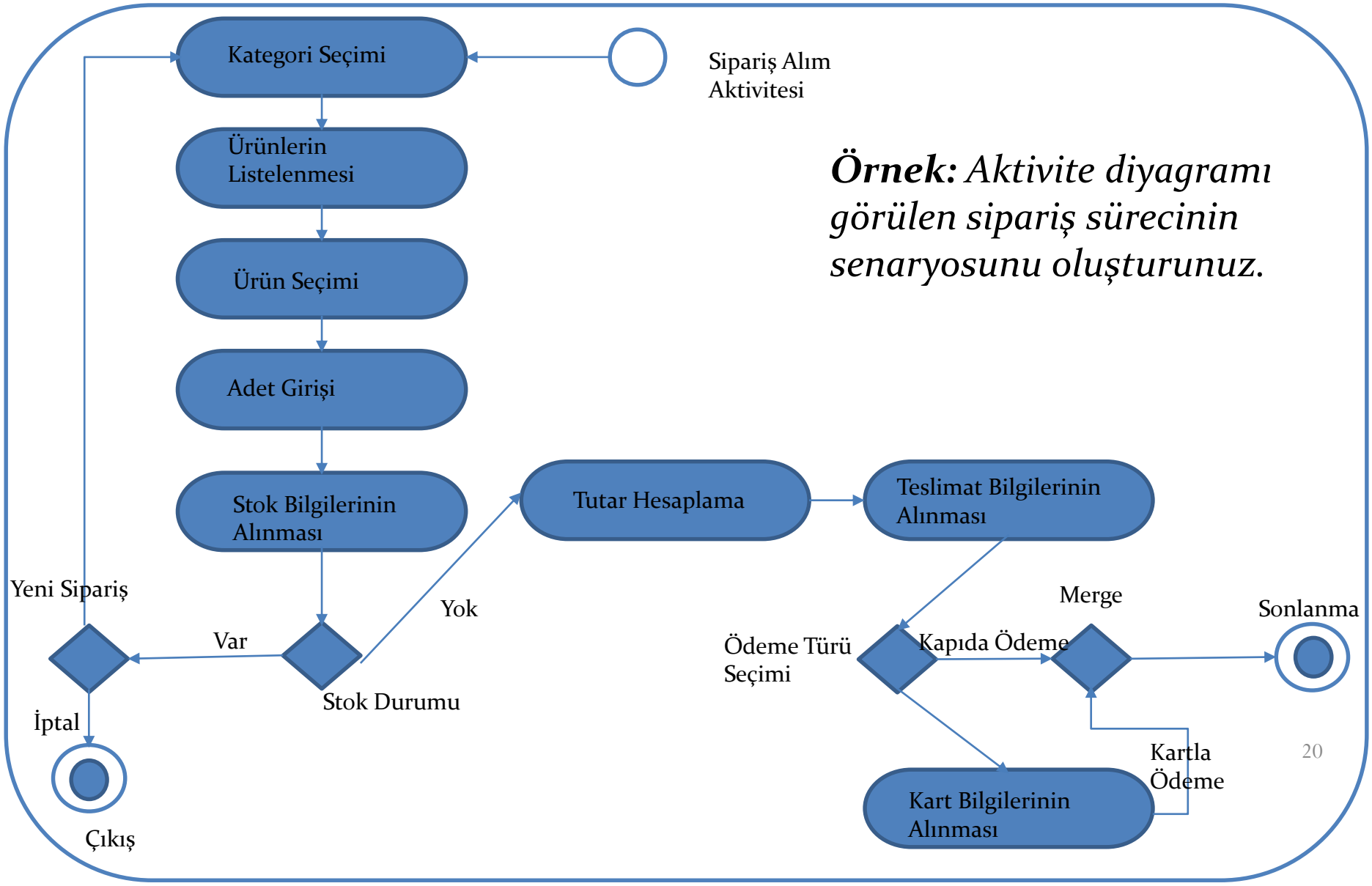


Birleşme

Çatallanma (Fork) ve ***Birleşme (Join)*** Eşzamanlı çalışan işlemleri göstermek için kullanılır.
Sağdaki yapıda bunun örneği görülmektedir.

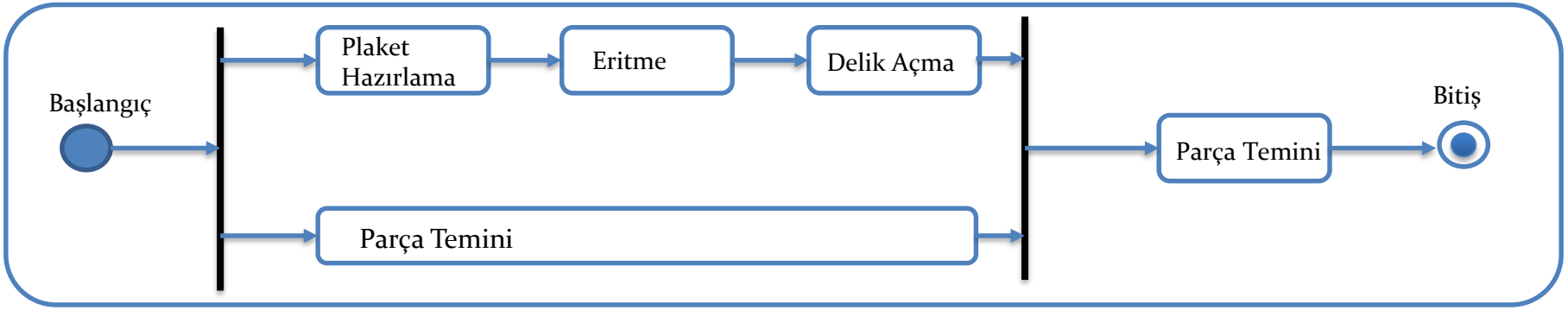


Yazılım Tasarımı ve Mimarisi



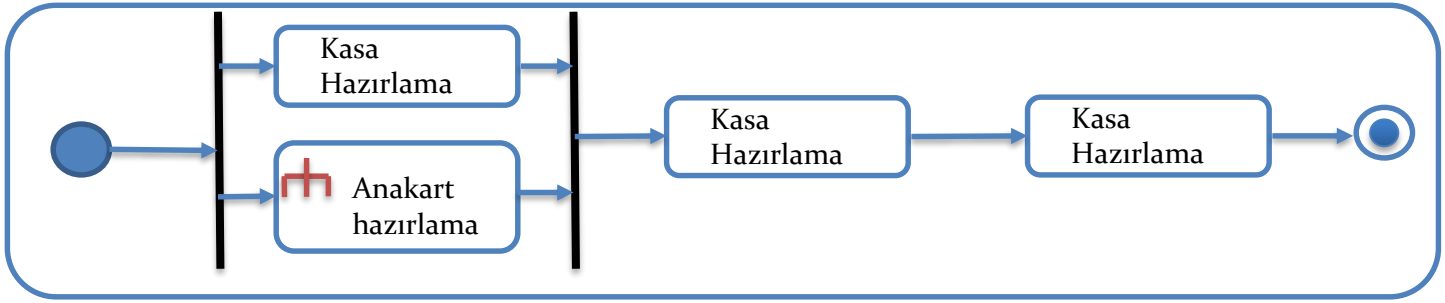
Örnek: *Aktivite diyagramı görülen sipariş sürecinin senaryosunu oluşturunuz.*

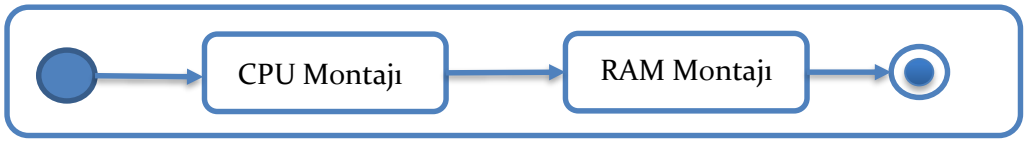
Yazılım Tasarımı ve Mimarisi



Yukarıdaki örnekte elektronik baskı devresinin hazırlanış aktivitesinde çatallanma(fork) ve birleşme(join) işlemi görülmektedir. Plaket hazırlama, eritme, delik açma ile parça temini paralel yürütülebilir.

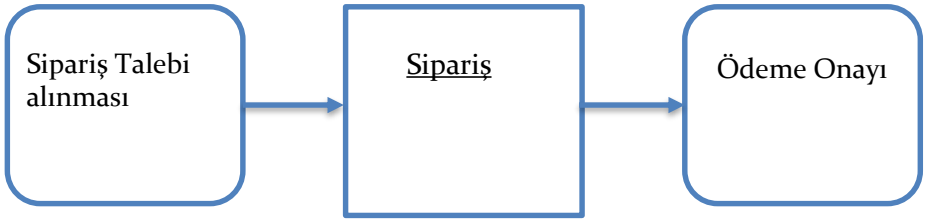
Bazen karmaşık bir aktivite diyagramı başka bir aktivite diyagramı ile detaylandırılabilir. Bu durumda aşağıdaki örnekte olduğu gibi topraklama işaretine benzer bir işaretle işaretlenir.





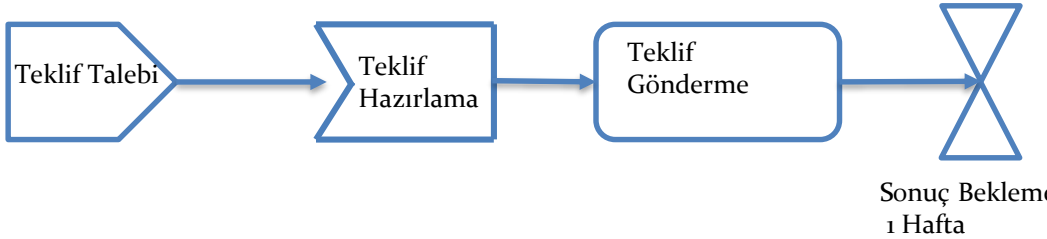
NOTLAR

Yazılım Tasarımı ve Mimarisi

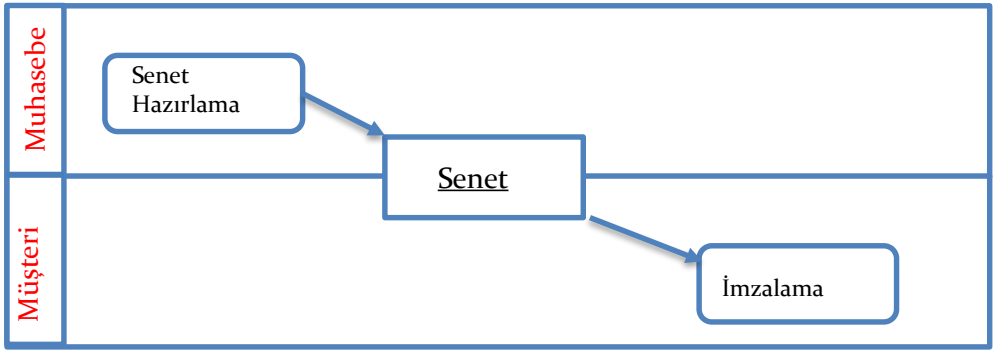


Aktivite diyagramlarında nesneler de önemli bir unsurdur. Nesneler fiziksel bir iş olarak düşünülebilir. Örneğin bir sipariş onaylama sürecini ele alırsak siparişe dair bir bilgi nesne gibi düşünülebilir. Nesneler aktiviteye ait aksiyonlar arasında bilgi akışında kullanılabilir ve diyagramda dikdörtgen sembolü ile gösterilir.

Bazı durumlarda nesneler aksiyonların girdi ve çıktılarını içerebilir. Buna iğne(pin) gösterimi denilmektedir. İlgili aksiyonun kenarına iliştirilen bir dikdörtgen ile gösterilir.

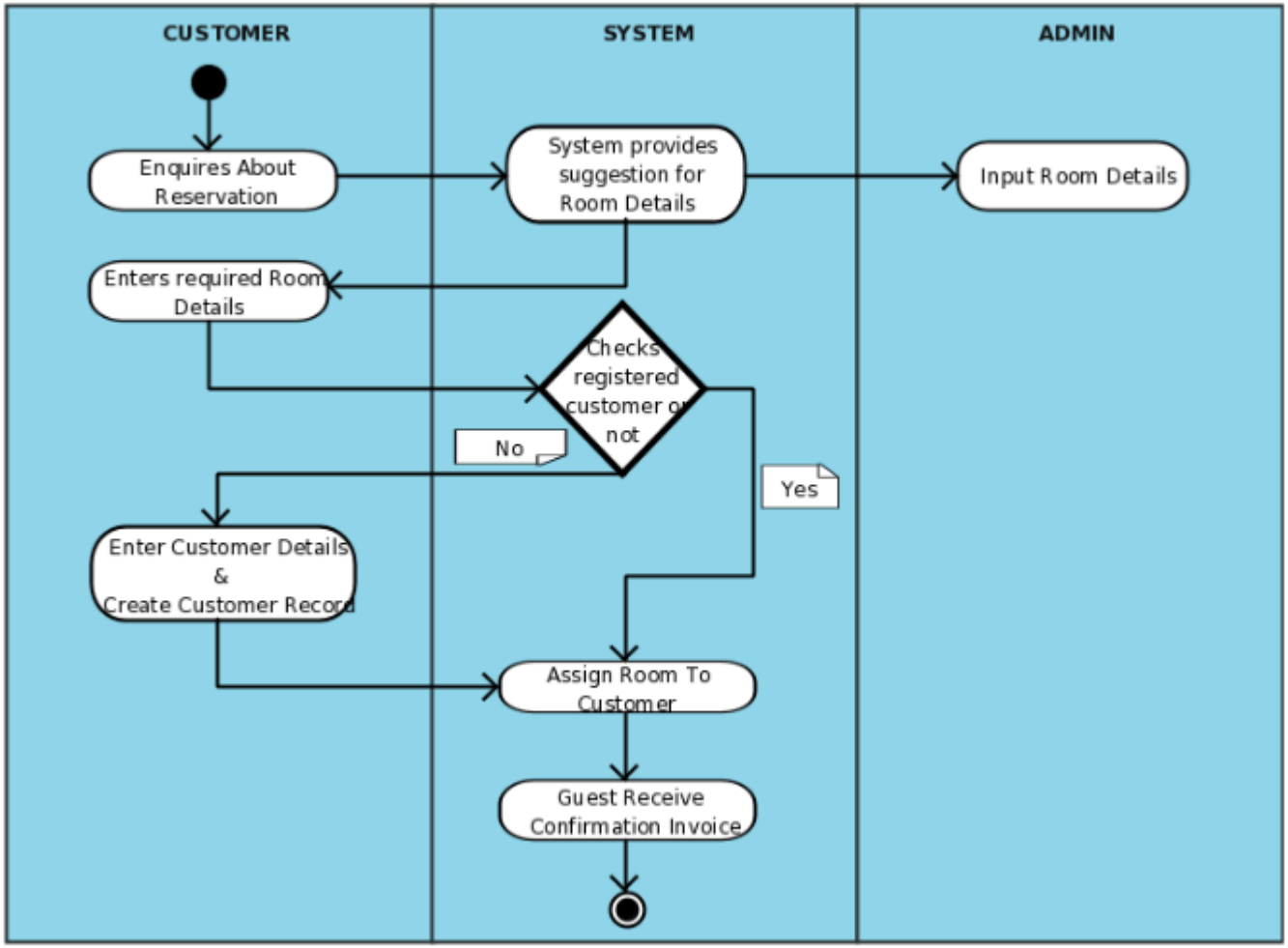


Aktivite diyagramında zaman kavramı gösterilebilir. Kum saati şeklindeki işaretle bekleme süresi ifade edilebilir.



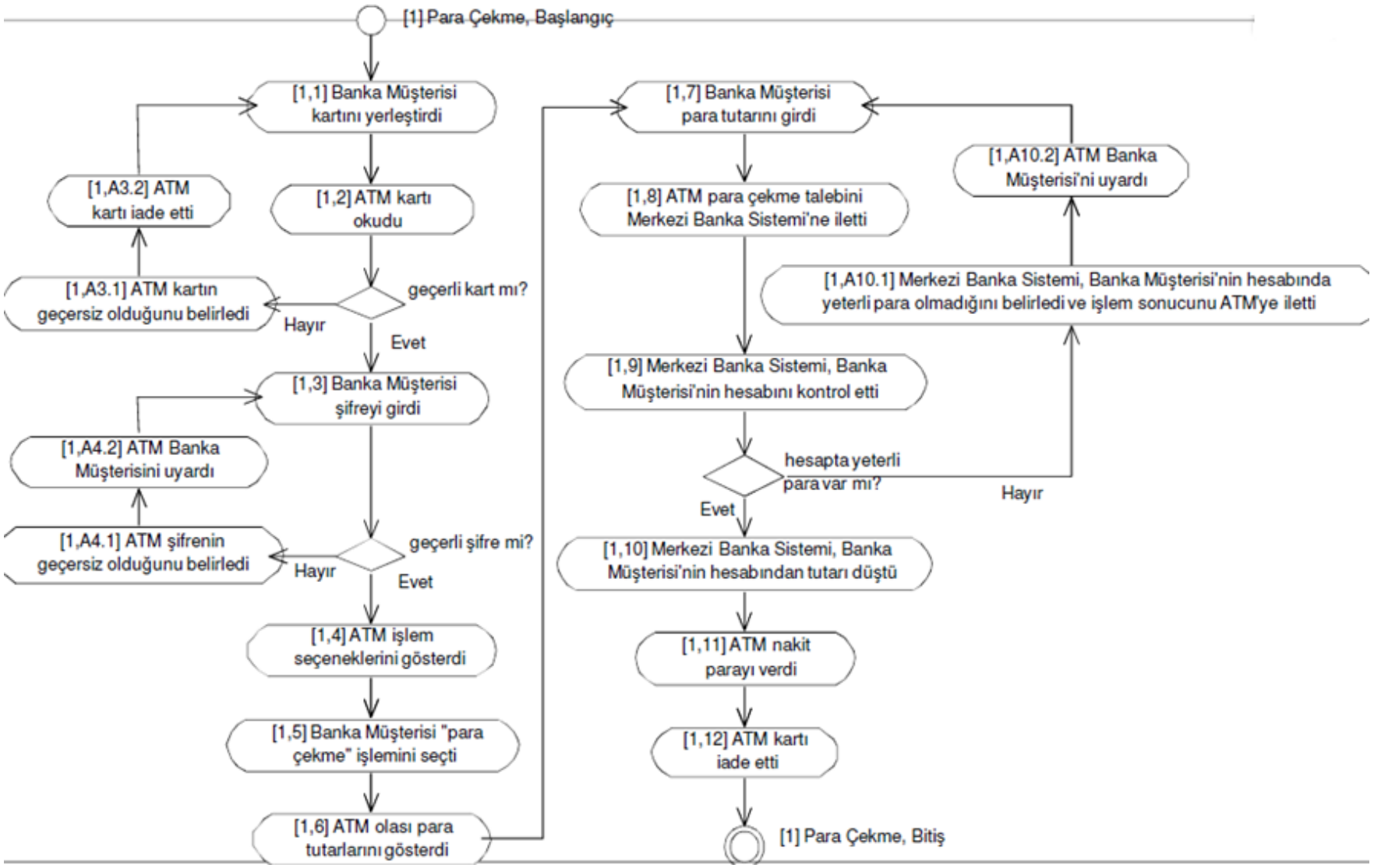
Aktivite diyagramlarında bölümler (partition) kullanarak diyagramların mantıksal anlamda daha iyi organize edilmesi sağlanabilir. Bölümlere yazılımsal anlamda bir sınıf ya da sistemdeki bir kullanıcı gibi anlamlar yüklemek mümkündür. Sol tarafta Muhasebe ve Müşteri olmak üzere iki adet bölüm oluşturulmuş ve ilgili aksiyonlar bu bölümlere yerleştirilmiştir.

Yazılım Tasarımı ve Mimarisi



Örnek: Otel rezervasyon sistemi için oluşturulmuş aktivite diyagramı

Yazılım Tasarımı ve Mimarisi

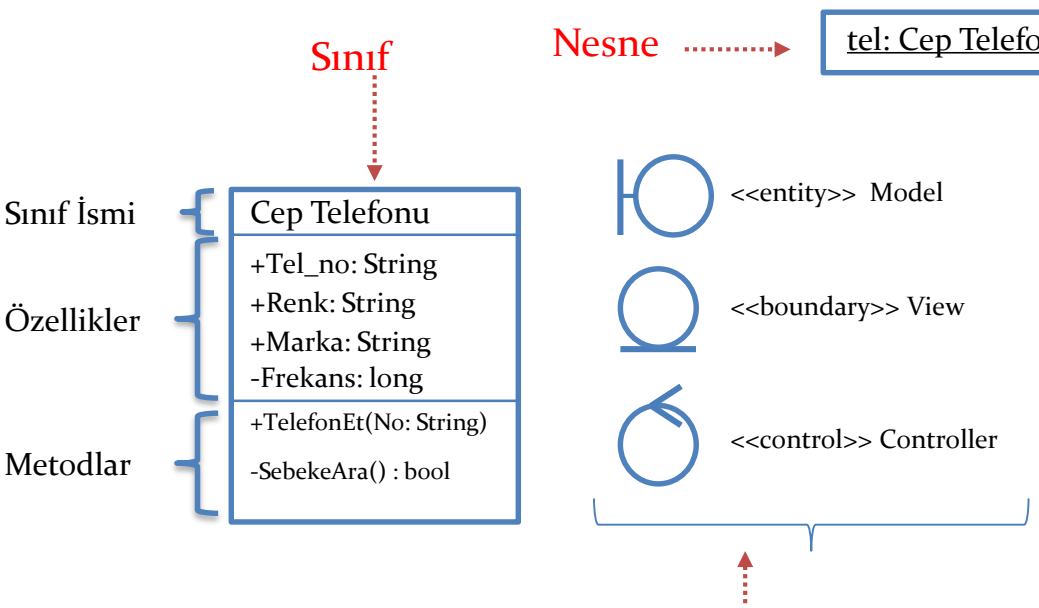


Yazılım Tasarımı ve Mimarisi

SINIF VE NESNE DİYAGRAMLARI

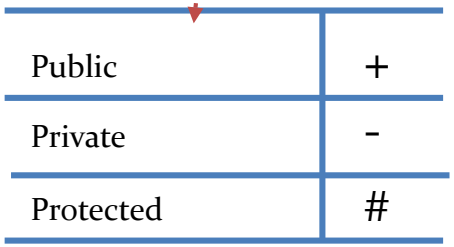
Sınıf Diyagramları UML 'in en sık kullanılan diyagram türü olup Nesne Yönelimli Analiz, Tasarım ve Programlama 'da esas teşkil eden sınıfları en iyi şekilde temsil etmeyi amaçlar. Sistemin statik yapısını ortaya koymaya yararlar.

- Sınıflar nesne tabanlı programlama mantığından yola çıkarak tasarlanmıştır.
- Sınıf diyagramları bir sistem içerisindeki nesne tiplerini ve birbirleri ile olan ilişkileri tanımlamak için kullanılırlar.
- Sınıf Diyagramları UML 'in en sık kullanılan diyagram türüdür.



Bazı özel nesneler örneğin MVC' deki *Model-View-Controller* gibi nesneler hazır stereotype' lar ile özel sembollerle de gösterilebilir. Bunlar **stereotyped** nesneler diye anılırlar.

Sınıftaki özellik ve metodlar için erişilebilirlikler (public,private, vb.); sınıf içerisinde aşağıdaki işaretlerle belirtilir



Not: Pratikte genellikle nesnelerin sadece isimleri yazılmakta türleri ise basitleştirme adına verilmeyebilmektedirler. Bu tip nesnelere ise **anonim nesne** denilir.

Sınıf Diyagramı Gösterimleri

Veri elemanlarının yani özelliklerin (attribute) gösterimi

```
Erişilebilirlik Değişken_İsmi: Tür[=İlk Değer]
```

Örnek:

```
+ id: int = 0
- m_hesapHareketleri: HashTable
# m_Bakiye: long
+ Kayitlar: DataRow [0 ..5]
```
(Dizi ya da koleksiyon biçimindeki veri elemanlarının gösterimi)

Fonksiyonların yani metodların gösterimi

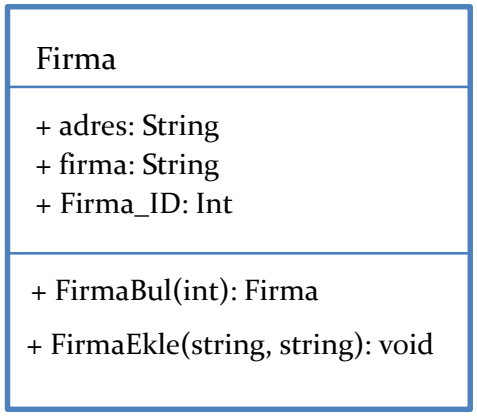
```
Erişilebilirlik Fonksiyon_İsmi (parametre_listesi): Geri_Dönüş
```

Örnek:

```
- MoveLastRecord(): DataRow
+ DisplayPoint(x: int, y:int)
```

Not: Sınıf diyagramlarında sanal (virtual) fonksiyonların isimleri *italik* yazılırken, statik üyeler altı çizili yazılarak vurgulanabilirler.

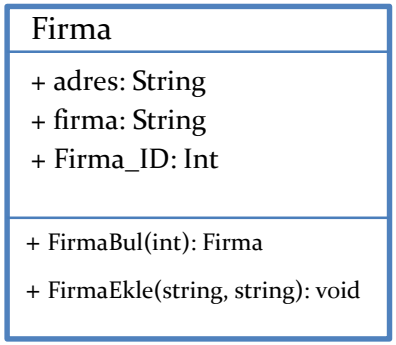
Örnek: Aşağıda sınıf diyagramı görülen Firma isimli sınıfın C# dilinde implementasyonunu yapınız.



CEVABI GÖSTER

Yazılım Tasarımı ve Mimarisi

Cevap: Aşağıda sınıf diyagramı görülen Firma isimli sınıfın C# dilinde implementasyonunu yapınız.




```
namespace Ornek
{
    public class Firma
   {
            private int Firma_ID;
            public String firma;
            public String adres;

            public Firma(){
            }

            ~Firma(){
            }

            public void FirmaEkle(String adres, String firma){
            }

            Public Firma_Bul(int id){
            }
      }
}
```

Yazılım Tasarımı ve Mimarisi

Nesneler Arası İlişkiler

Sınıflar ve/veya nesneler arasında var olabilecek ilişkiler UML' de şu şekilde tanımlanmıştır.

- Association
- Aggregation
- Compozition
- Generalization/Specialization
- Dependency
- Usage
- Realization

ASSOCİATİON (Referans veya Birliktelik)

- İlişki bağlantıları bir sınıf diyagramdaki en genel bağlantıdır. İlişki sınıf'ın örnek (instance)'leri arasındaki bağlantıları gösterir. Örneğin Sipariş sınıfı Müşteri sınıfı ile ilişkilidir.
- Aşağıdaki kod örneğinde de Automobile ve Person sınıflarında bu tarz bir ilişki bulunmaktadır. Automobile sınıfındaki driver isimli, Person sınıfı türündeki car isimli veri elemanı bu sınıflara ait yaratılacak olası nesne örneklerinde association ilişkisinin oluşmasına yol açar.

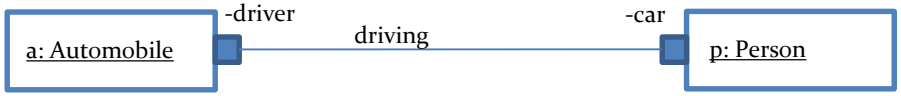
```
class Automobile{ private Person driver; }
class Person { private Automobile car; }

Automobile a = new Automobile();
Person p = new Person ();

a.Driver = p;
p.Driver = a;
```

Yazılım Tasarımı ve Mimarisi

Association ilişkisi iki nesne arasına çizilen düz bir çizgi ile ifade edilir. Çoğunlukla bu çizginin üzerine ilişkiye verilen isim yazılmaktadır. Ayrıca ilişki içerisindeki davranışları ya da üstlendikleri roller de rol isimleri ile ifade edilebilmektedir. Gerçekte rol isimleri diğer nesneye ait referansı taşıyan veri elemanının adı hatta erişim belirteci kullanılarak konulur. Çokluk ilişkisi bu ilişkiye dahil olabilecek nesne sayısını temsil eder. Örnekte Sipariş nesnesi sadece bir Müşteriye ilişkilendirilebilirken bir Müşteri birden fazla Sipariş ile ilişkilendirilebilir.



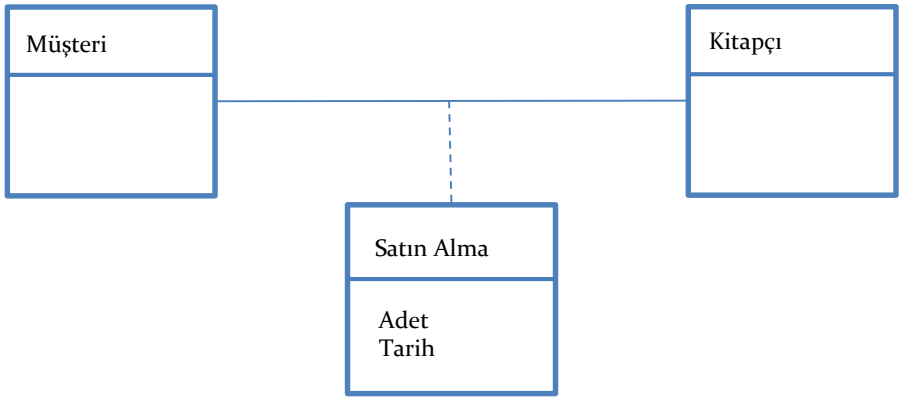
Çokluk ilişkisi (Multiplicity) bu ilişkiye dahil olabilecek nesne sayısını temsil eder. Örnekte Sipariş nesnesi sadece bir Müşteriye ilişkilendirilebilirken bir Müşteri birden fazla Sipariş ile ilişkilendirilebilir.

Çokluk ilişkisi gösterimleri(Multiplicity)

1 :	*1 adet*	*m..n :*	*2 değer arasında*	*0..1 :*	*Var ya da yok*
** :*	*Belirsiz çokluk*	*1..* :*	*1 tane ya da sonsuz sayıda*	*0..* :*	*0 tane ya da sonsuz sayıda*

Sınıflar arasındaki tek yönlü ilişkiyi gösterir. Ok işareti ile gösterilir. Örneğin Sipariş sınıfında (siparişin hangi müşteriye ait olduğunu ifade etmek amacıyla) Müşteri türünde bir referans tutuluyor ise soldaki gibi olmaktadır.

Yazılım Tasarımı ve Mimarisi



Sınıflar arasındaki ilişkinin bir çizgiyle belirtebilecek kadar basit olmadığı durumlarda **ilişki sınıfları (association class)** kullanılır. Müşteri ile Kitapçı sınıfı arasında "satın alma" ilişkisi vardır. Fakat müşteri satın alırken Ücret ödemek zorundadır. Ücret sınıfı ile satın alma ilişkisi kesikli çizgi ile birleştirilir.

Bir nesnenin kendi kendini referanse etmesine **self (reflexive) association** denilir. Bu durum kendi üzerine çizilmiş bir ilişki çizgisi ile gösterilir.

Anahtar değer çifti kullanan sistemlerde anahtar(key) varlığı sebebiyle **nitelikli (qualified) association** diye anılan bu notasyonda nesnenin yanına iliştirilen bir dikdörtgen içine niteleyici anahtar (qualifier) yazılabilir. Örnekte Urun_ID qualifier yani niteleyici konumundadır.

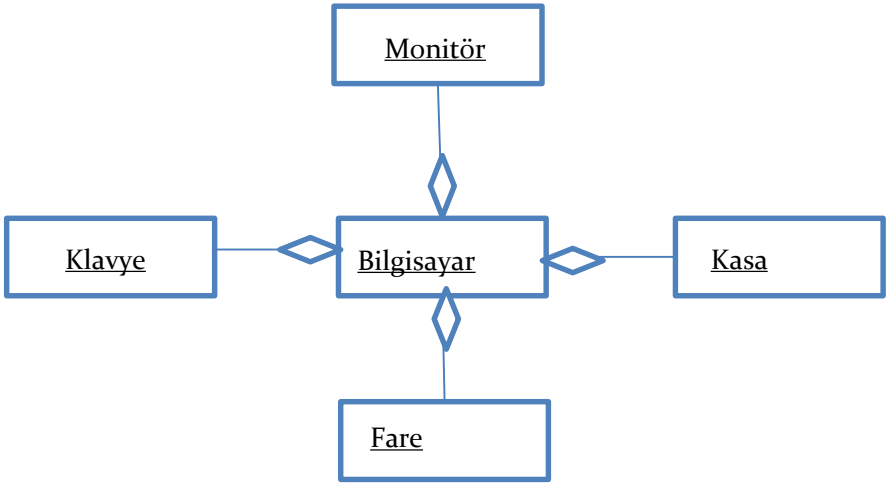
Yazılım Tasarımı ve Mimarisi

AGGREGATIONS (İçerme)

İçerme bağıntısı, iki sınıf arasındaki "sahiptir" veya "içerir" türünden bağıntıları modellemekte kullanılır. Bu bağıntıda bir sınıfın nesnesi, diğer sınıfın nesnesi tarafından sahiplenilmektedir. Örneğin, kumandanın tuş takımı, pil ve ışık lambası gibi parça elemanları vardır ve her bir parça kendi başına işlevsel bir bütünlük taşır. Diğer bir örnek olarak bilgisayar ele alınırsa; kasa, monitör, klavye ve fare bilgisayarı oluşturan parçalardır. Bu örnekte de aggregation ilişkisi bulunduğu görülmektedir.

UML' de nesneler arası aggregation ilişkisini göstermek üzere ucunda içi boş karo şekli olan çizgiler kullanılır.

Aggeration ilişkilerde de çokluk ilişkisi *(multiplicity)* bulunabilmektedir.



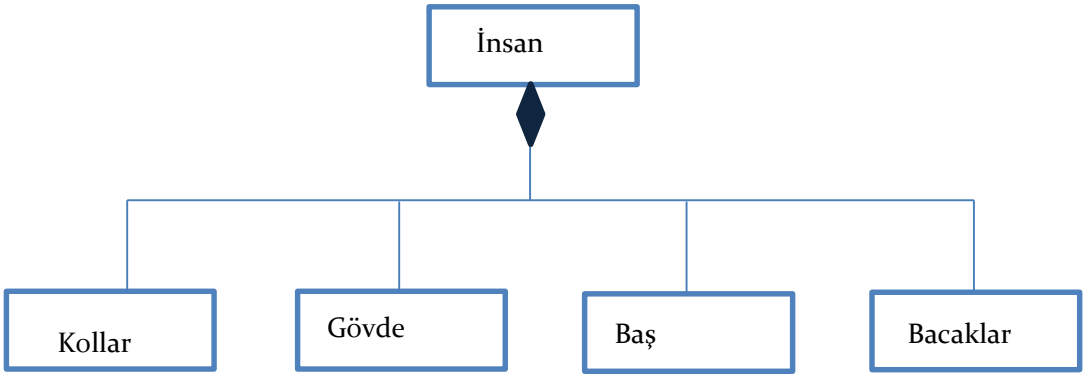
Yazılım Tasarımı ve Mimarisi

COMPOSITION (Bağımlı İyelik)

Bu türde parça ile bütün arasında adeta organik bir bütünlük bulunmaktadır. Composition ilişkisinde parçayla bütün birbirinden bağımsız bulunamaz. Örneğin İnsan ile kolları, bacakları vb. organları arasında composition ilişkisi bulunmaktadır.

Bir kitap ile kitabın bölümleri arasında da böyle bir ilişki vardır. Composition ilişkisinde parça ve bütün birbirinden bağımsız şekilde bulunmaz. Yani parça ile bütün birbirinden ayrı düşünülebiliyorsa aggregation ilişkisi, düşünülemiyor ise composition ilişkisi vardır denilebilir. Composition ilişkisi düz bir çizginin ucundaki dolu karo şekli ile gösterilir.

Aggeration ilişkilerde de çokluk ilişkisi *(multiplicity)* bulunabilmektedir.



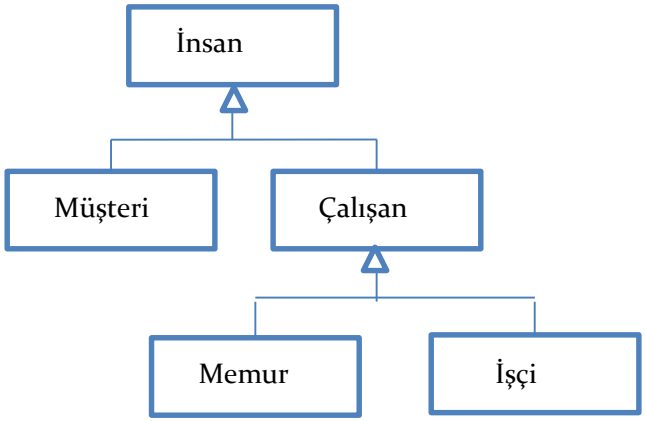
Yazılım Tasarımı ve Mimarisi

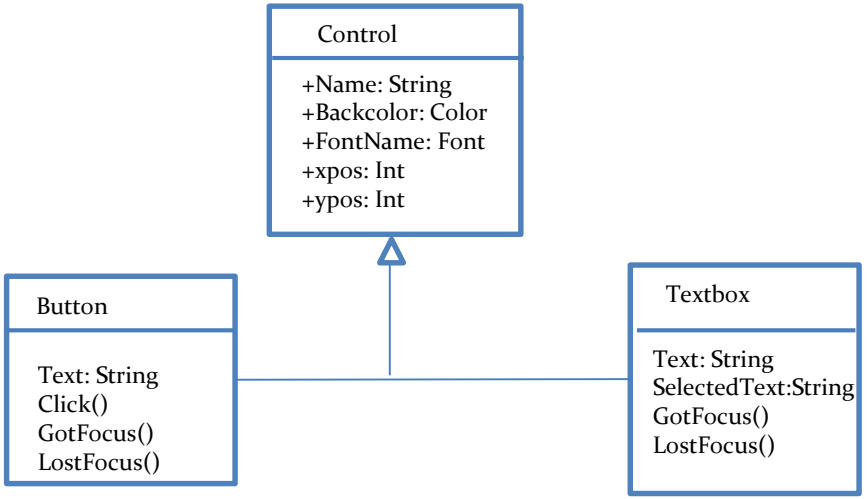
GENERALİZATİON (Genelleme)

Nesne tabanlı programlama tekniğinin en önemli parçası türetme (inheritance) dir. Türetme yoluyla bir sınıf başka bir sınıfın var olan özelliklerini alarak, o sınıf türünden başka bir nesneymiş gibi kullanılabilir. Bir sınıfın işlevleri türetme yoluyla genişletilecekse, türetmenin yapılacağı sınıfa taban sınıf (base class), türetilmiş olan sınıfa da türemiş sınıf (derived class) denir.

UML' de taban ve türemiş sınıflar arası ilişki generalization (genelleme) olarak adlandırılmıştır.

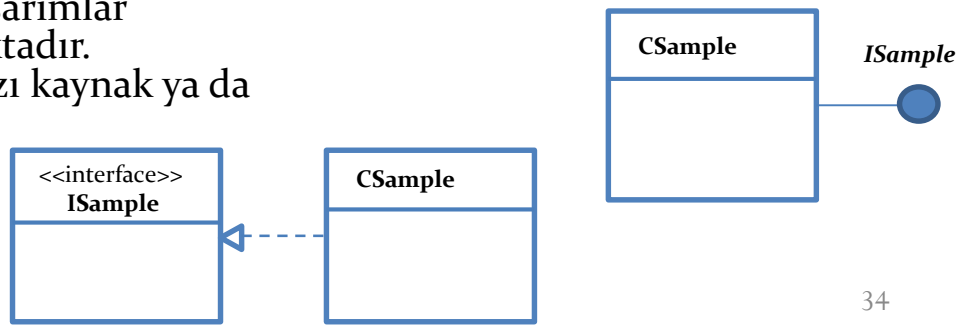
UML' de generalization içi boş bir ok sembolü ile gösterilir.





REALİZATİON (Gerçekleme)

UML' de sınıftan interface' e doğru çizilen bir ok ile gösterilen bu ilişki özellikle C# ve JAVA dillerinde polimforfik tasarımlar yapabilme amacıyla yoğun bir biçimde kullanılmaktadır. Gösteriminde kesikli çizgi ucunda ok kullanılır. Bazı kaynak ya da araçlarda bu ilişki lolipop sembolü ile gösterilir.

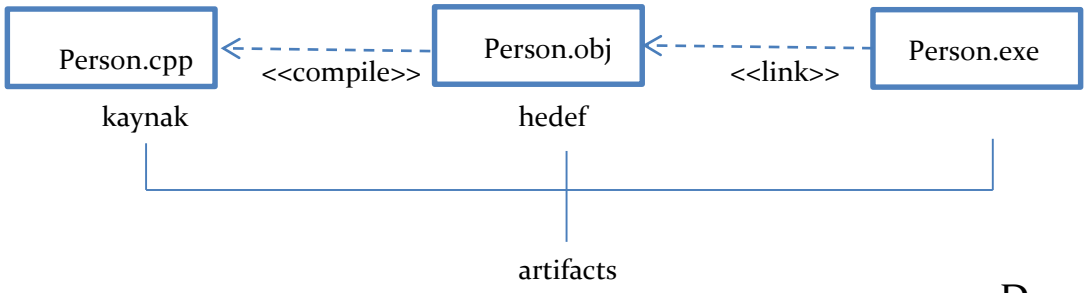


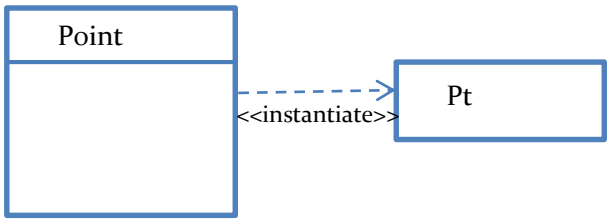
Yazılım Tasarımı ve Mimarisi

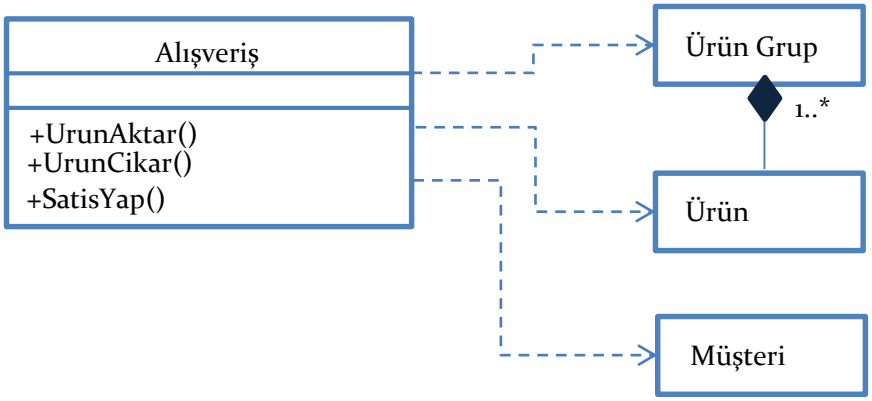
DEPENDENCY (Bağımlılık)

Dependency ilişkisinin dayandığı temel fikir; kaynak nesne üzerinde yapılan herhangi bir değişimin hedef durumundaki nesneye de yansıtmaktadır.

Bağımlılık mutlaka nesneler arasında olacak diye bir kural yoktur. Bir sınıf ile nesne arasında olabileceği gibi UML' de artifact diye anılan kaynak kod dosyası veya derlenmiş dosya gibi unsurlar arasında da bağımlılık gelişebilir. Bağımlılık ilişkisi, bağımlı sınıftan bağımsız sınıfa doğru kesikli çizgi ile ifade edilir. Örneğin aşağıdaki diyagramda bu durum izlenebilir.







Dependency ilişkisinin farklı stereotype' larla ifade edilen çeşitli türleri vardır. Yukarıdaki örnekteki <<instantiate>> gibi. Bunlardan ön tanımlıları bulunduğu gibi kullanıcı tarafından da oluşturulabilir.

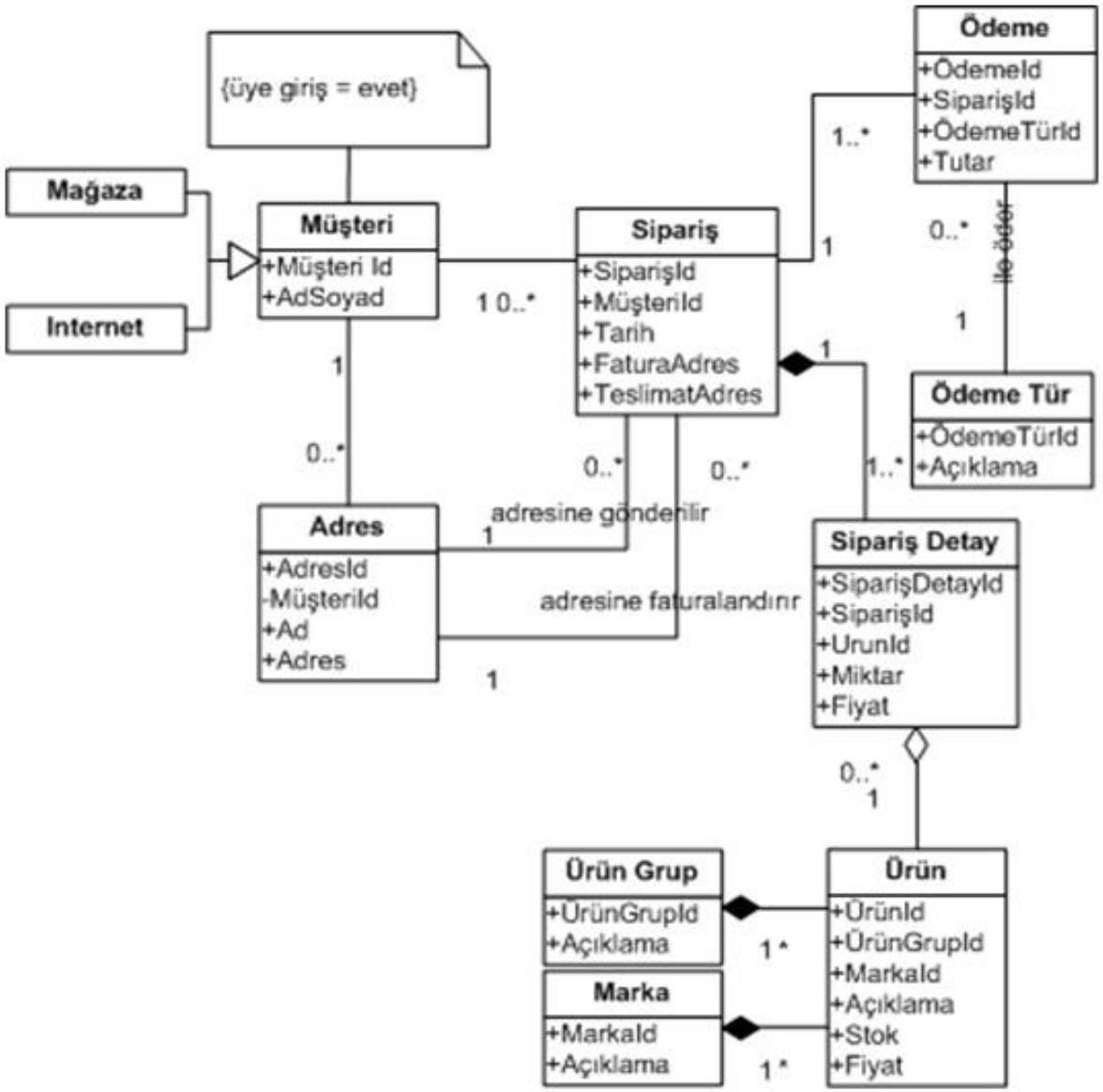
Bazı ön tanımlı stereotype' lar

<<trace>> , <<refinement>>, <<usage>> veya <<use>>, <<bind>>, <<call>>, <<derive>>, <<send>>, <<friend>>

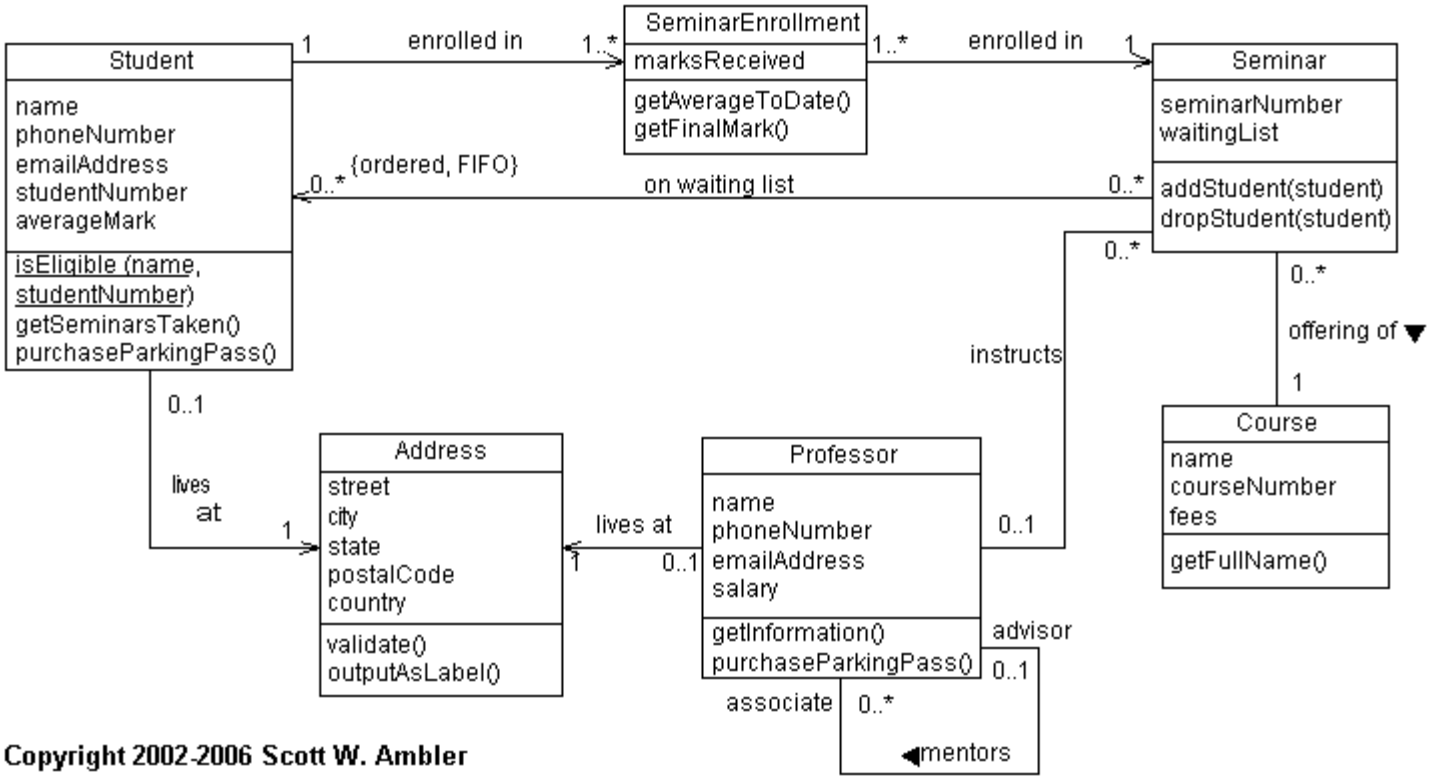
Yazılım Tasarımı ve Mimarisi

Sınıf Diyagram Örnek:

Bir e-ticaret uygulamasında sipariş



Sınıf Diyagram Örnek: Ders bilgi sistemi



STATE (DURUM) DİYAGRAMLARI

Sistem veya nesnelerin farklı durumlar arasında belirli koşullar ve dış etkenlere bağlı olarak geçişler olabilir. State diyagramları da bu sistem ya da nesnelerin durumlarını ve bu durumlar arası geçişleri gösterir. Özellikle event' lere (olay) sahip nesnelerin dinamik davranışlarının ortaya konulmasında bu diyagramlardan yararlanılır.

Terminoloji

Initial State (İlk Durum): Yaşam döngüsünün ilk eylemi ya da başlama noktasını ifade eden elemandır. İçi dolu yuvarlak ile gösterilir. Sözde durum (*pseudo state*) olarak da adlandırılır. Sözde durum denilmesinin sebebi değişkeni veya herhangi bir eyleminin olmayışıdır.

State (Durum): Nesnenin ya da sistemin x anındaki durumunu ifade etmek için kullanılır. Köşeleri yuvarlatılmış dikdörtgenler ile gösterilir.

Transition (Geçiş): Nesnenin bir durumdan diğer bir duruma geçişini ifade eder. Ok sembolu ile gösterilir. Bazı durumlarda bu geçişe neyin sebep olduğu okun üzerine yazılır. Geçişe sebep olan şeyin programatik olarak bir anlamı varsa (örn. Bir fonksiyonun çalışması) bu kez söz konusu fonksiyonun ismi geçiş oku üzerine yazılarak belirtilebilir. Bu yazın biçimi için önerilen notasyon şöyledir:
`Tetikleyici Faktör [Koşul] / Eylem` Örn: `Display[id!=null] / Kaydın Görüntülenmesi`

Concurrency (Eş zamanlılık ve Senkronizasyon): *Concurrency* eş zamanlı gerçekleşen olayları temsil eder. Bunun programatik karşılığı *thread* kavramıdır. Bazen paralel gerçekleşen olaylar bir noktada birleşip senkronize olabilirler.

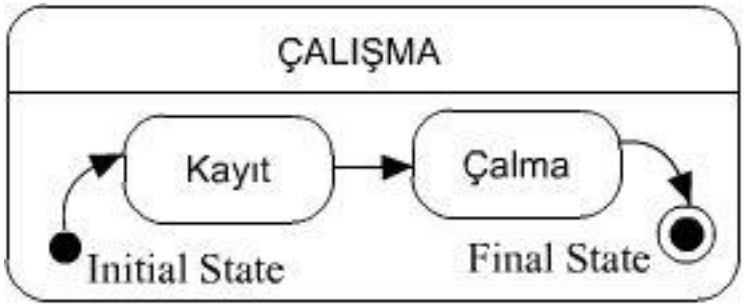
Desicion (Karar) : Mantıksal bir koşula bağlı olarak farklı bir aktivitenin gerçekleştirilmesini sağlar. Baklava (Karo) sembolü ile gösterilir.

Substate (Alt durum): Karmaşık sistemlerdeki bazı durumlar kendi içerisinde başka alt durumları içerebilirler. Böyle durumlar *Compozit State*' ler olarak bilinir. Alt durumlar ardışıl ya da eş zamanlı olabilirler. Bir üst durumun içerisine çizilerek gösterilirler.

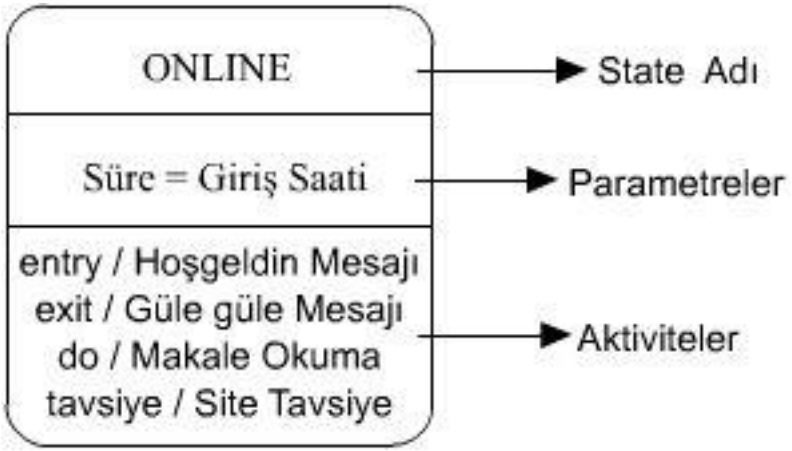
Fork and Join (Çatallama ve birleşma): Kalınca çizilen bir çizgiye gelen ve giden oklar olarak şeklinde sembolize edilirler. *Fork*(Çatal) 1 giriş ve n çıkışlı yapısıyla o noktadan sonra paralel gerçekleşen durumları gösterir. *Join* (birleşme) de ters şekilde n giriş ve 1 çıkışlı yapıdadır.

Guard Condition (Geçiş Koşulu) : Çoğu zaman iki durum arasındaki geçiş için bir event' in (olay) gerçekleşmesinin yanı sıra tanımlı bir koşulun da sağlanması gerekmektedir. Örneğin teyp üzerinden kayıt yapmak için kayıt tuşuna basmanın yanı sıra kasetin de boş olmasının kontör edilmesi gibi. Guard condition' lar köşeli parantez ile ifade edilirler. (Örn `[bakiye > 0 ]`)

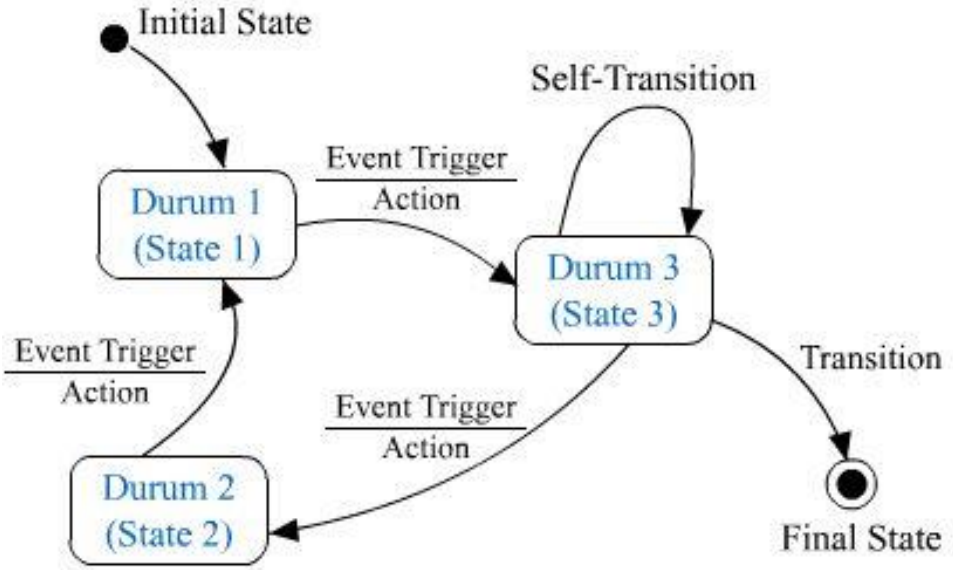
Yazılım Tasarımı ve Mimarisi



Substate (Alt durum) örneği

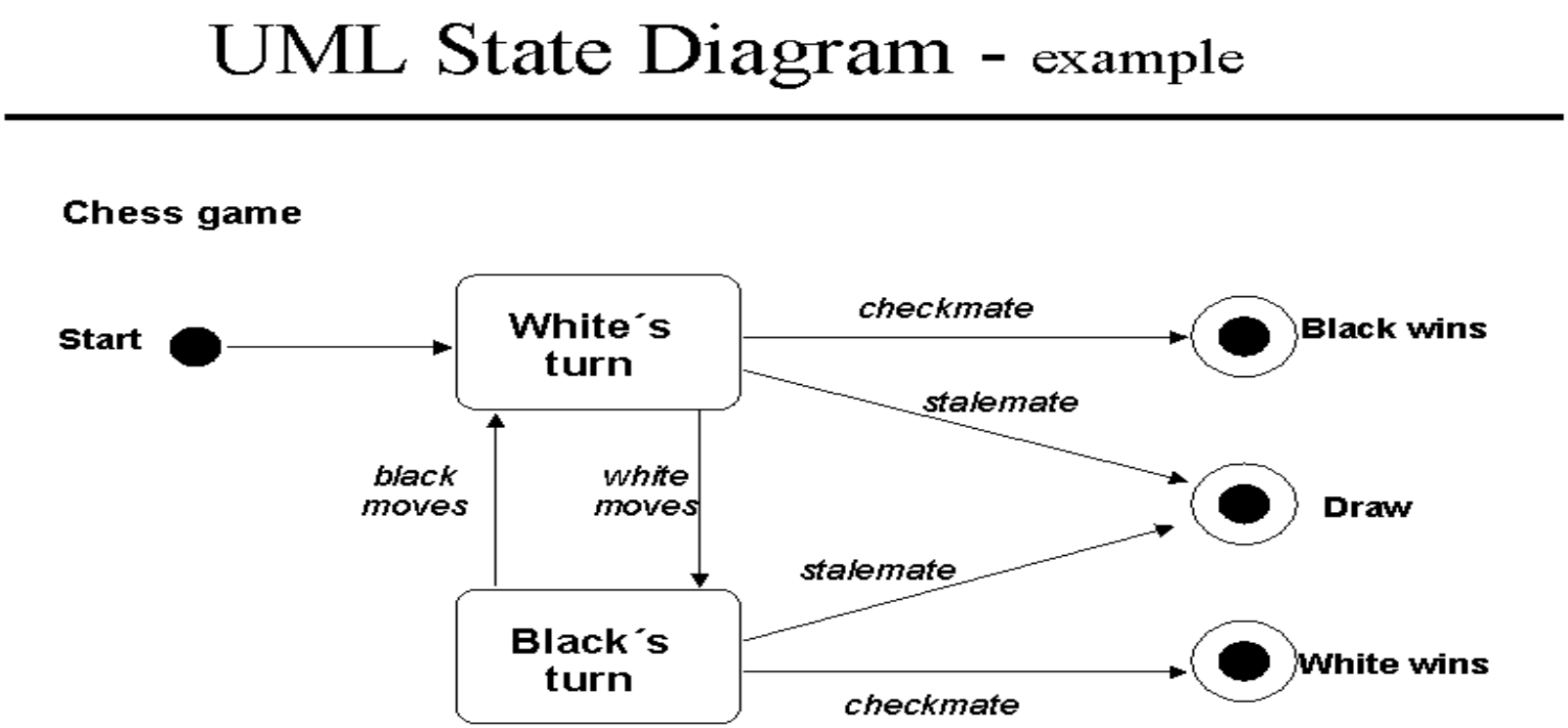


Temel state modeli

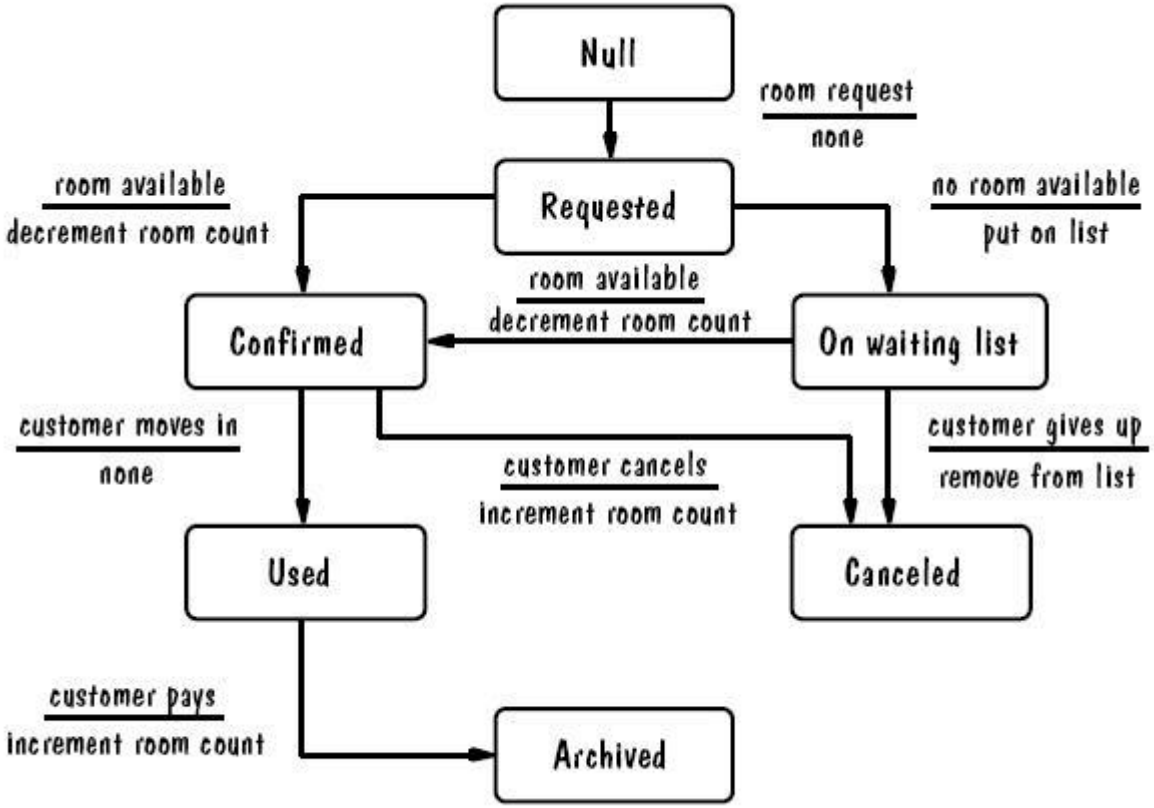


Basit Bir **State Machine** Modeli

State Diyagram Örneği - Satranç



State Diyagram Örneği Otel Rezervasyonu

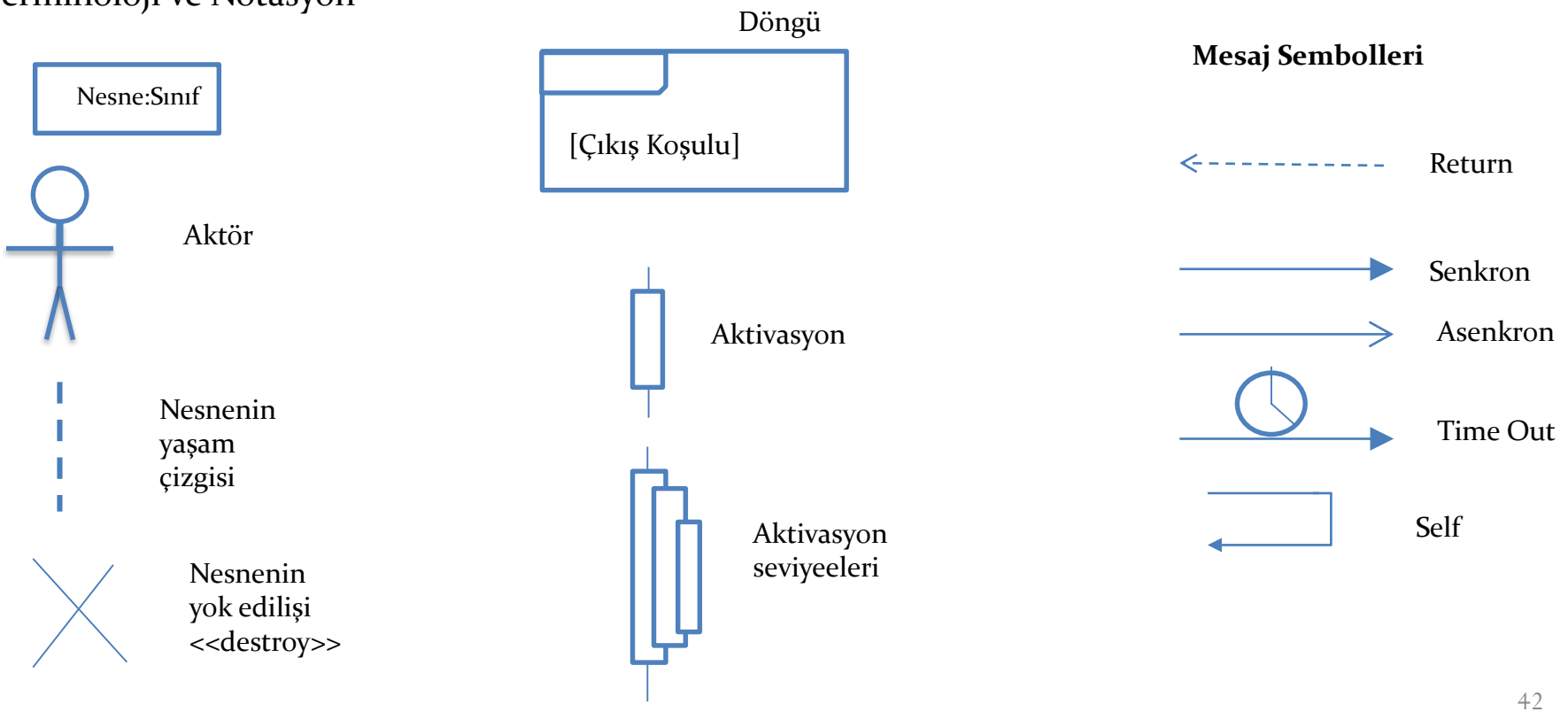


Yazılım Tasarımı ve Mimarisi

SEQUENCE (ETKİLEŞİM) DİYAGRAMLARI

Sequence diyagramları nesneler (katılımcılar) arası belirli bir zaman aralığında gerçekleşen çeşitli etkileşimleri mesaj alışverişleri biçimde dökümante eder. Yani bu diyagram türünde tasarımcı nesnelerin birbirleriyle iletişimine odaklanır. Sequence diyagramlarını iki boyutlu olarak düşünmek gerekir. Bu tür diyagramlarda nesneler soldan sağa yatay olarak sıralanırken, zaman ekseni düşey olarak gösterilir. Zamanın akışı ise yukarıdan aşağı doğrudur.

Terminoloji ve Notasyon



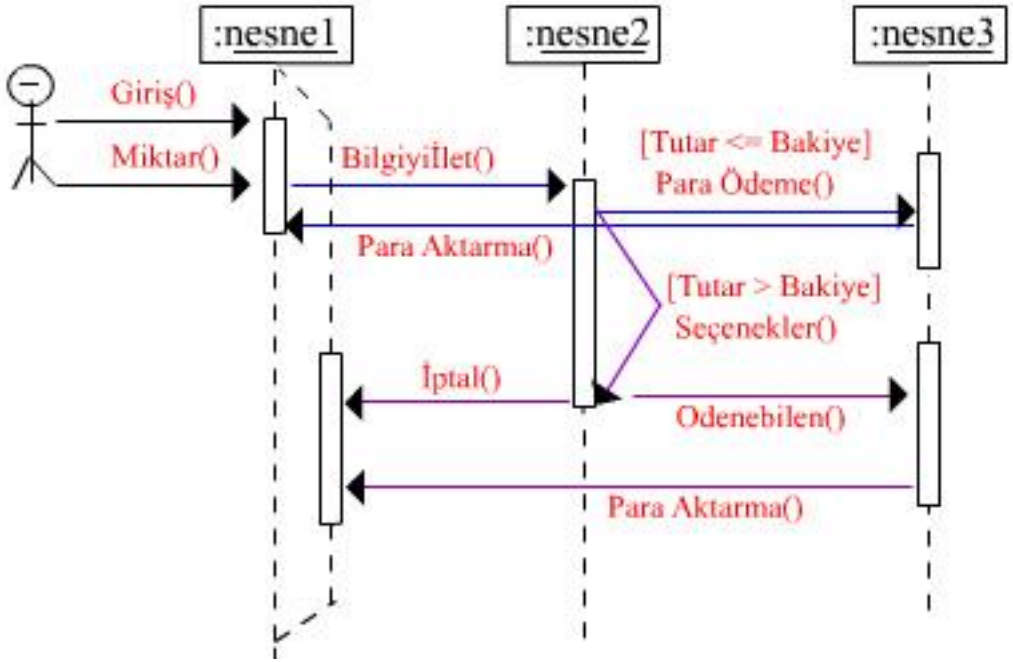
Örnek Sequence Diyagram

Üç nesneli bir ATM makinasının çalışmasını modelleyen "sequence" diyagramını örnek verilsin. Sistemimde ilgilenilen üç nesne bulunmaktadır.

Tuş Takımı ve Para Alma Modülü: Kullanıcının ATM makinası ile haberleştiği arayüzler (Nesne 1)
Hesap Kontrol Modülü : Kullanıcı doğrulama ve bakiye kontrol gibi mantıkların işletildiği birim (Nesne 2)
Para İletme Bölümü : Kullanıcının yani talebinin arayüz yani Nesne1 yardımıyla kullanıcıya sunulması. (Nesne 3)

Diyagram çizilmeden önce iş kuralları yazılsın :

1 - Kullanıcı şifresini yazar.
2 - Ardından çekmek istediği tutarı nesne 1 yardımıyla yazar. (tuş takımı)
3 - Çekilmek istenilen tutar nesne 2 tarafından kontrol edilir.
4 - Eğer bakiye uygun ise Nesne 3 e mesaj gönderilerek bu modüle para aktarımı sağlanır.
5 - Nesne 3, Nesne 1 i yani arayüzü uyararak paranın alınması sağlanır.



Yazılım Tasarımı ve Mimarisi

Örnek:

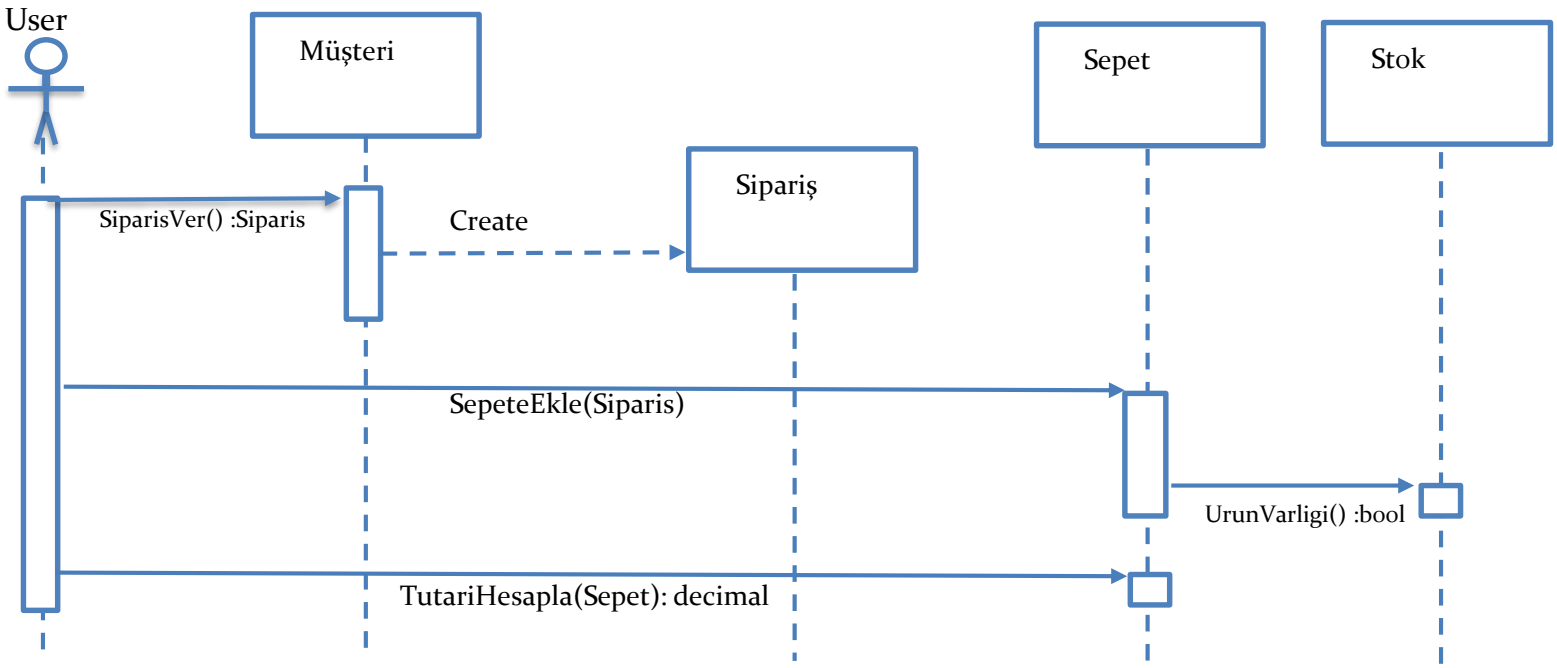
Bir online sipariş sisteminin modellendiği varsayılsın. Bu sistemde müşteri, Sipariş, Sepet ve Stok sınıfları türünde nesneler olabilir.

Müşteri bir sipariş oluşturursa Musteri sınıfına ait SiparisOlustur() fonksiyonunu çağırır ve bu fonksiyondan geriye Siparis turunde bir nesne oluşturularak döner.

Müşteri daha sonra bir listeden seçtiği ürünleri sepete SepeteEkle() fonksiyonu ile ekler.

Şüphesiz bu ürünlerin stokta mevcut olması gerekeceği için her bir ürünün stoktaki varlığı UrunVarligi() isimli fonksiyonlar kontrol edilir.

Son durumda ise siparişlerin toplam tutarı TutarHesapla() fonksiyonu ile hesaplanıp müşteriye bilgi olarak verilir.



Yazılım Tasarımı ve Mimarisi

COMMUNİCATİON (İLETİŞİM) DİYAGRAMLARI

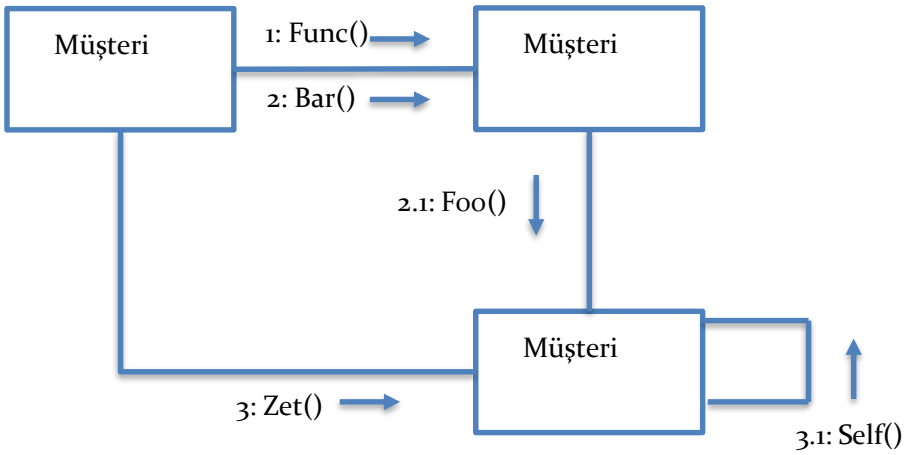
Pratikte bir senaryoyu gerçeklemek amacıyla ortaklaşa çalışan birden çok nesne Communication (iletişim) diyagramlarıyla modellenebilir. Bu diyagram sequence diyagramlarını anımsatsa da zaman kavramının olmayışı ile sequence diyagramlarından ayrılmaktadır. Communication diyagramları ile sistemin gerek dinamik gerekse de statik yapısı ortaya konulabilir.

Communication diyagramlarında nesnelerin iki farklı karakteri olduğundan söz edilir. Şayet bir nesne (örneğin A) diğer bir nesnenin (B' nin) fonksiyonunu çağırıyorsa; A istemci (client) , B ise tedarikçi (supplier) terimleri ile bu nesneler adlandırılır.

Communication diyagramlarında da nesneler arasındaki mesajlaşma senkron veya asenkron olabilir. Ancak bu diyagramlarda farklı olarak birer sıra numarası verilmelidir. Numaralandırmanın mantığı her yeni mesaja 1' den başlayan ve birer birer artan numaralar atamaktır.

Şayet bir mesaj içinde başka bir mesaj gönderilmişse (metot çağrılmışsa) nokta sembolüyle ikinci seviye bir numara atanır. (Aşağıdaki örnekteki gibi.)

Aşağıdaki şekilde x,y ve z nesnelerinin mesajlaşması, ardından da bu olayların akışı görülmektedir.



1. **Func:** x istemci nesnesi y tedarikçi nesnesine ait Func() metodunu senkron çağırmıştır.
2. **Bar:** x istemci nesnesi y tedarikçi nesnesine ait Bar() metodunu senkron çağırmıştır.

   2.1. Foo: Bar kendi içerisinde z nesnesine ait Foo() metodunu çağırmıştır.
3. **Zet:** x nesnesi, z nesnesinin Zet() metodunu çağırmıştır.

   3.1. **Self:** Zet içerisinde yine z' ye ait olan Self() metodu çağrılmıştır.

Yazılım Tasarımı ve Mimarisi

COMPONENT (BİLEŞEN) DİYAGRAMLARI

Component diyagramları, sistemin yazılım bileşenlerini ve birbirleri arasındaki bağlantının nasıl olduğunu gösteren diyagramlardır. Sisteme daha yüksek seviyeden yani bileşenler seviyesinden bakabilmeyi sağlarlar. Bileşenler "alt sistemleri" oluştururlar.

Component diyagramların diğer UML diyagramlarından farkı nedir?

- Component diyagramları sistemin uygulanma perspektifini gösterir.
- Component diyagramı içerisinde yer alan bileşenler; sistemdeki farklı tasarım öğelerinin gruplandırılmasını yansıtır. (Örn: sistemin sınıfları)

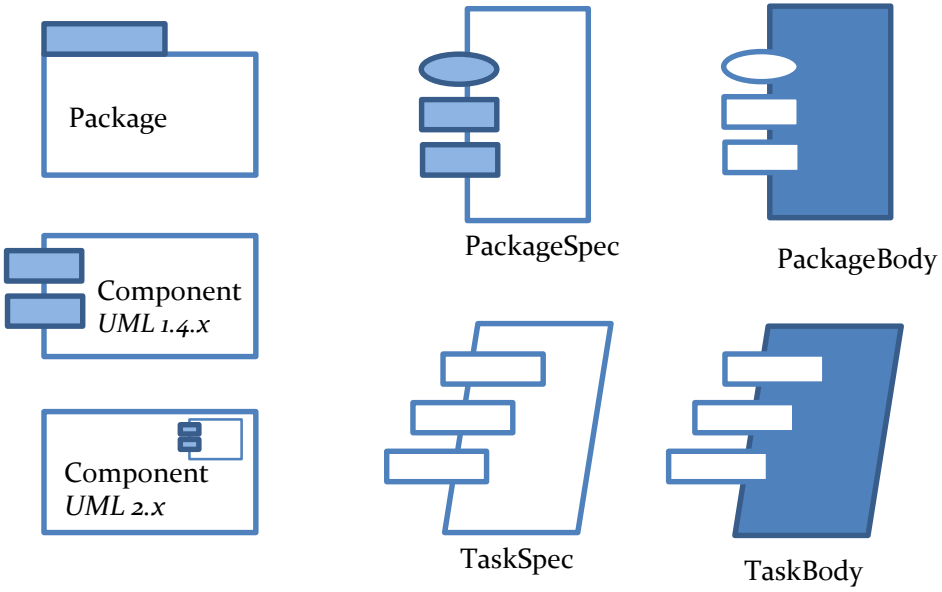
Terminoloji ve Notasyon

Component diyagramlarının ana unsuru bileşenlerdir. Bileşenler genellikle derlenmiş ve binary formdaki dll, jar, ocx gibi modüller biçiminde somutlanabilir.

Alt sıradaki, ***packageSpec*** sembolü ile *C++* dilinde *.h* dosyası kolayca gösterilebilmektedir. Arayüz göstermek için de bu *packageSpec* sembolü kullanılmaktadır. **PackageBody** sembolü ise bu *.h*' e ilişkin *.cpp* dosyasını temsil etmektedir.

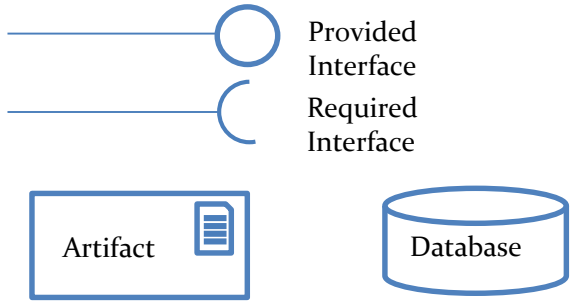
Eğer kendi başına kontrole sahip olan parçacığın *(thread)* arayüzünü göstermek istiyorsanız **TaskSpec** kullanılır, gerçeklemesi ise **TaskBody** içinde gösterilir. *Exe* dosyası, *TaskSpec* olarak *.exe* uzantısı ile bileşen diyagramlarında kullanılabilir.

Veritabanı (**Database**) sembolü ise alt sağda verilmiştir.



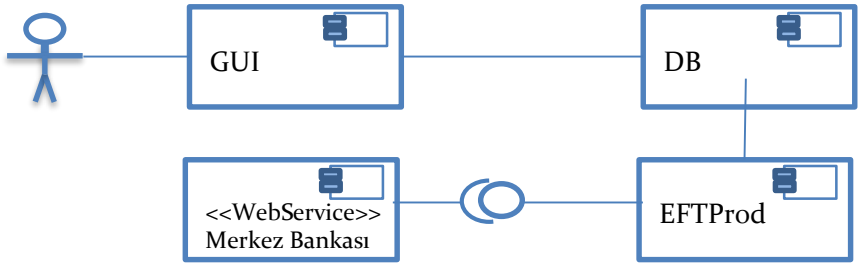
Bileşene ait implementasyon artifact' ler ile gösterilir. UML' de artifact' ler 3 ana kategori altına toplanmıştır.

1. İşletim sistemi, veritabanı, runtime ortamı (.Net Framework, JVM vs...)
2. Kaynak kod dosyaları (.cpp, .cs, .java), derlenmiş dosyalar (.jar, .dll, .class, .obj, .exe ...)
3. Java Bean' ler Servlet' ler , HTML ve XML dökümanları, COM+ ve .NET assembly' leri ve CORBA bileşenleri gibi.

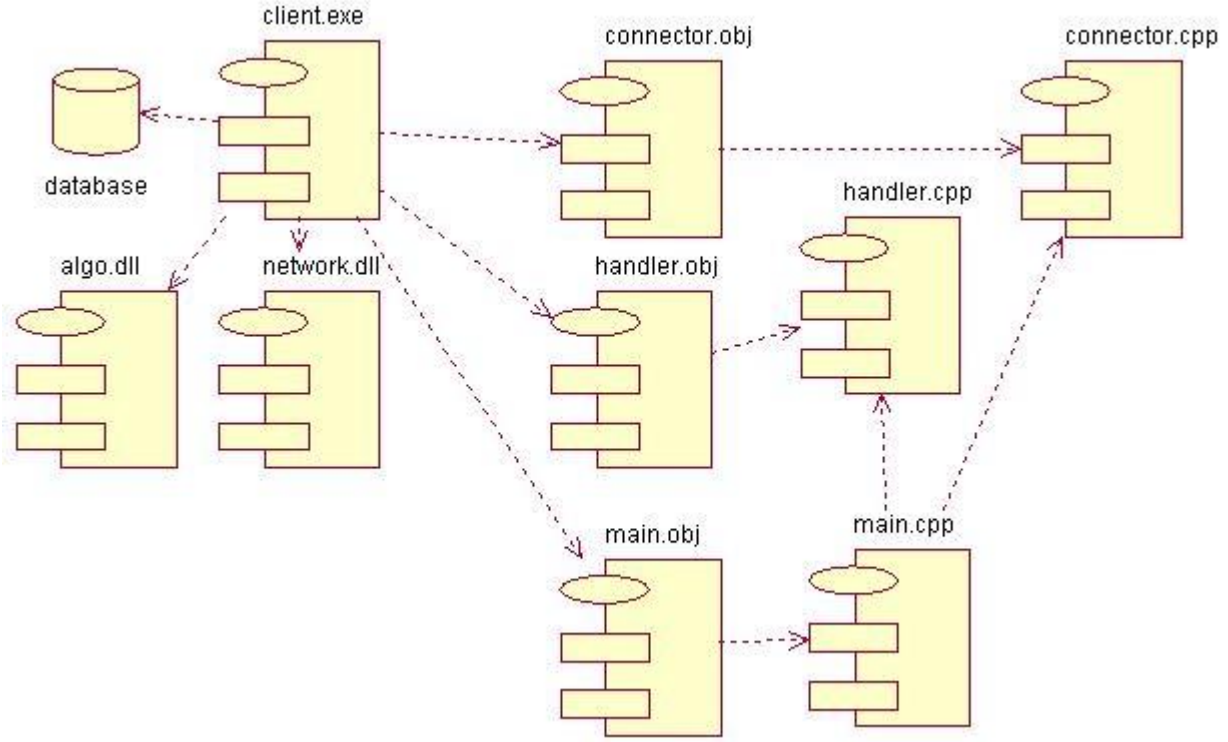


Yazılım Tasarımı ve Mimarisi

Örnek 1: Sağıda görülen örnekte bir bankaya ait EFT sisteminin bileşenleri modellenmiştir. Dikkat edilirse bir web servisi olan MerkezBankası ile EFTProf isimli uygulama servis kontratı diye isimlendirilmiş olan interface' ler aracılığı ile ilişki kurmaktadır.



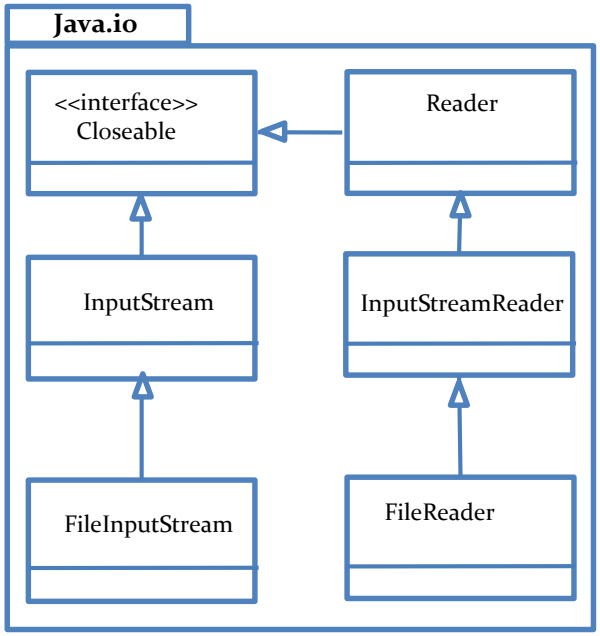
Örnek 2: Sağda 2 tane *.dll* kullanan (*algo.dll* ve *network.dll*) bir "*client.exe*" uygulamasının bileşen diyagramı gösterilmektedir. Bağlantıyı sağlamak için "*connector.cpp*" içindeki arayüzler, istekleri yapmak için "*handler.cpp*" içindeki arayüzler kullanılmaktadır. Şekilde, *.obj* dosyaları ve bu dosyaları oluşturan *.cpp* dosyaları gösterilmektedir. *"main.cpp"* dosyasının içinde, *"connector.cpp"* ve *"handler.cpp"* içindeki bazı metotlar kullanıldığı için bağımlılık oku gösterilmiştir.



Yazılım Tasarımı ve Mimarisi

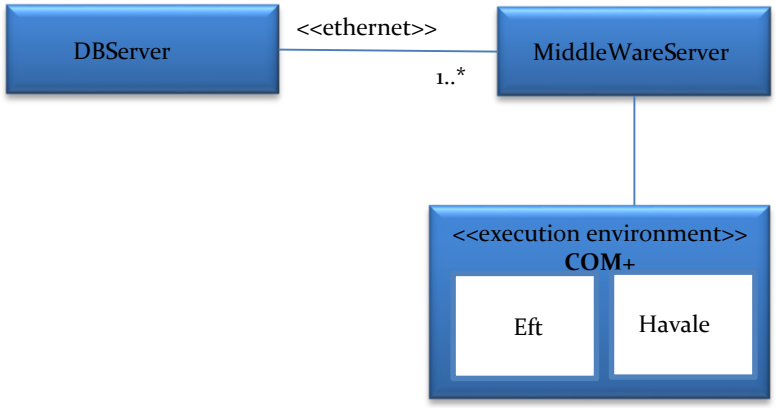
PACKAGE (PAKET) DİYAGRAMLARI

Java' da Package , C++ ve C# ' da namespace olarak bilinen, birbirleriyle ilişkili sınıfların gruplandığı yapılar UML package sembolü ile ifade edilir. Sağıdaki örnekte java.io paketinden bir örnek bulunmaktadır.



DEPLOYMENT (DAĞITIM) DİYAGRAMLARI

Bu diyagramlar modellenen yazılım sistemine ilişkin fiziksel mimari ve topolojiyi en temel unsurları ile (donanım, işletim sistemi, çalıştırma motoru, application server, gibi ...) Deployment diyagramlarında ana sembol; üç boyutlu küp şeklindeki Node (düğüm) sembolüdür. Ayrıca bir node' un tam olarak neyi ifade ettiğini belirten bir stereotype da kullanılabilir.



Yazılım Tasarımı ve Mimarisi

Örnek Sorular:

1. Anlatılan UML diyagramların her birinin nerede kullanıldığına dair bilgi vererek farklarını açıklayınız.

2. Sınıflar ve/veya nesneler arasında var olabilecek ilişkiler UML' de şu şekilde tanımlanmıştır.
   - *Association*
   - *Aggregation*
   - *Compozition*
   - *Generalization/Specialization*
   - *Dependency*
   - *Usage*
   - *Realization*

   Bu ilişkileri kısaca açıklayarak aralarındaki farklardan bahsediniz.

3. Kullanılabilecek bazı özellikleri aşağıdaki gibi olan basit bir kütüphane otomasyonu planlayınız ve bu otomasyon için gerekli
   - *Use case anlatımı,*
   - *Use case diyagramı,*
   - *Activity diyagramı,*
   - *Class diyagramı,*
   - *State diyagramı,*
   - *Sequnce diyagramı ve*
   - *Component diyagramını*

   oluşturunuz.

Kütüphane otomasyonu için kullanılabilecek bazı örnek özellikler aşağıdadır

- *Admin, Kütüphaneci, ve Kullanıcı gibi aktörleri olmalı,*
- *Kullanıcı bilgileri tutulmalı,*
- *Kütüphaneci tarafından Ödünç kitap verilebilmeli, iade alınabilmeli.*
- *Ödünç süresi tutulabilmeli*
- *Süresi geçen kitaplar için kullanıcıya uyarı maili atabilmeli.*

Yazılım Tasarımı ve Mimarisi

KAYNAKLAR

1. Aykut Taşdemir ,UML ve Dizayn Paternleri , Pusula Yayıncılık, 2014
2. Sefer Algan, UML ile State(Durum) Diyagramları, http://www.csharpnedir.com/articles/read/?id=86
3. Univera, UML ve Modelleme bölümleri, http://univera-ng.blogspot.com.tr/
4. Examples of State Transition Diagrams, http://users.csc.calpoly.edu/~jdalbey/SWE/Design/STDexamples.html
5. Yazılım Mühendisliği- Analiz. İçerik Yazılım İster (Gereksinim) Analizi, http://slideplayer.biz.tr/slide/2738446/
6. Ahmet Kaymaz, UML ve UML Diyagramları-II , http://www.ahmetkaymaz.com/2009/09/15/uml-ve-uml-diyagramlari-ii/
7. Creately, Activity Diagram (UML) Examples | Activity Diagram (UML) Templates, http://creately.com/diagram-community/popular/t/activity-diagram
8. Sefer Algan, UML ile "Sequence" Diyagramları, http://www.csharpnedir.com/articles/read/?id=402
9. Scott W. Ambler, A Uml class diagram, http://www.agiledata.org/essays/objectOrientation101.html, 2002-2006
10. Çağatay Çatal, UML ile Bileşen (Component) Diyagramları,http://www.csharpnedir.com/articles/read/?id=482
11. Univera, UML ve Modelleme – Bölüm 10 (Component ve Deployment Diyagramlar), http://univera-ng.blogspot.com.tr/2010/04/uml-ve-modelleme-bolum-10-component-ve.html
12. Aykut Taşdelen, UML'de Stereotype, Constraint, TaggedValue Nedir ?, https://aykuttasdelen.wordpress.com/2011/02/23/umlde-stereotype-constraint-taggedvalue-nedir/