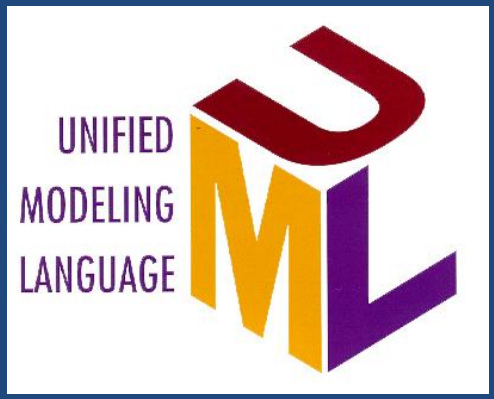


NESNEYE YÖNELİK PROGRAMLAMA
Unified Modelling Language (UML)
Bütünleşik Modelleme Dili

Özlem AYDIN

Trakya Üniversitesi
Bilgisayar Mühendisliği Bölümü

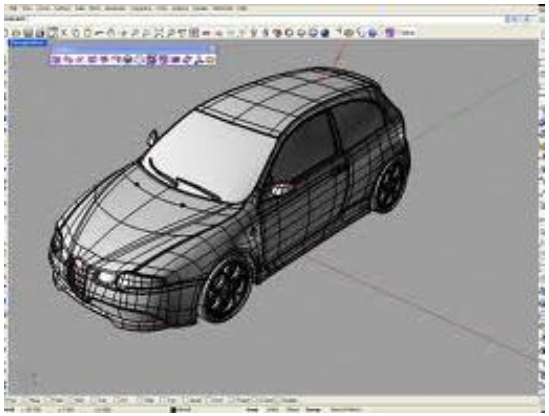
MODEL NEDİR?

- Model, gerçek dünyadaki bir olayın veya sistemin soyutlanması, basitleştirilmesi ve kavramlaştırılmasıdır.

- Model, olayı veya sistemi tanımlamaya başka bir deyişle bir örnek türetmeye yardımcı olur.

- Modeller gerçek dünyadaki örneklerinin yerini alamazlar, ancak gerçek olay veya sistemin karmaşık yapısının anlaşılabilir parçalara indirgenmesinde yararlı olurlar.

MODELLEME NEDİR?



- Modelleme bir sistemi incelemek üzere o sistemin basit bir örneği yapılması anlamına gelir. Bu örnek gerçek sistemin yardımcısı ve basitleştirilmiş bir şeklidir.
- Modelleme sistemlerin karmaşıklığını çözümlemede kullanılan en eski ve en etkin yöntemdir.

MODELLEME NEDİR?

- Bir sistem modellenirken farklı bakış açılarıyla tekrar tekrar incelenir.
- Bu inceleme sırasında modellemeyi yapan kimse sistemin özelliklerinden o anda ilgilendiklerini öne çıkarırken diğerlerini göz ardı edebilir.
- Sonuçta oluşan bu soyut yapı sistemin ilgilenilen özelliklerinin bir modeli olur.
- Hiçbir model gerçek sistemin özelliklerini tümüyle içermez.

Yazılımda sistemin modellenmesi

- Yazılım projelerinde yer alan proje yöneticileri, müşteriler, analistler, tasarımcılar, programcılar, testçiler ve teknik yazarlardan her birinin eğitim düzeyleri ve alt yapıları farklıdır.

- Eğer bir sistem, tüm proje ekibinin anlayabileceği ortak bir dille modellenirse, çok karmaşık anlatımlar basitleşebilir ve aralarındaki iletişim çeşitli diyagramlarla maksimum düzeyde tutulabilir.

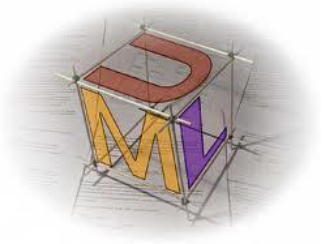
Yazılımda sistemin modellenmesi

- Bir yazılım sistemin modellenmesi süreci aşamaları:
  - ✓ Sistem analizi
  - ✓Sistem tasarımı
- Bu aşamalarda sistemin farklı yönlerini ortaya koyan modeller oluşturulur.
- Her model sistemin farklı açılardan incelenmesini sağlar ve yazılımcının sistemi kavramasında, çözüm seçenekleri oluşturmasında, müşteriyle ve proje ekibiyle fikir alışverişinde bulunmasında yardımcı olur.

Yazılımda sistemin modellenmesi

- Yazılım sektöründe modelleme için geliştirilmiş çeşitli diller mevcuttur.
- Nesneye yönelik sistemlerin analiz ve tasarımında standart olarak kullanılan modelleme dili UML'dir.

UML NEDİR?



- UML, yazılımın modellenmesi ve planlanması için kullanılan standart bir dildir.
- UML yazılım mühendisliğinde nesneye yönelik sistemleri modellemede kullanılan açık standart olmuş bir görsel modelleme dilidir.
- Bir program ya da yazılım geliştirme dili değildir.
- Yazılım geliştirmenin analizden bakıma kadar tüm aşamalarında ekipler ve bireyler arasındaki iletişimin düzgün yürütülmesi için kullanılmaktadır.
- Yazılımın yaşam döngüsü içinde farklı görev gruplarının projeye ve sisteme farklı bakış açıları vardır. Bundan dolayı UML çeşitli bakış açılarını ifade eden diyagramlar içermektedir.
- Çok zengin bir dil olmasından dolayı, Yazılım Mühendisliği'nin bir çok yönden ihtiyaçlarını karşılamaktadır.

UML'NİN GELİŞİM SÜRECİ

1989-1994 yılları yazılım mühendisliğinde metot savaşları olarak bilinen bir dönemdir. Sistemleri modellemek için kullanılan birçok modelleme dili vardı. 90'ların ortalarına doğru öne çıkan 3 yöntem vardır:

- **Booch**
  Yaratıcısı Grady Booch'dur. Tasarım ve gerçekleştirimde mükemmel.

- **OMT (Object Modelling Technology - Nesne Modelleme Teknolojisi)**
  Yaratıcısı Jim Rumbaugh. Analiz ve veri yoğunluğu çok olan sistemler için uygun.

- **OOSE (Object Oriented Software Engineering - Nesneye Yönelik Yazılım Mühendisliği)**
  Yaratıcısı Ivar Jacobson. Use-Case adı verilen güçlü bir teknik içeriyordu.

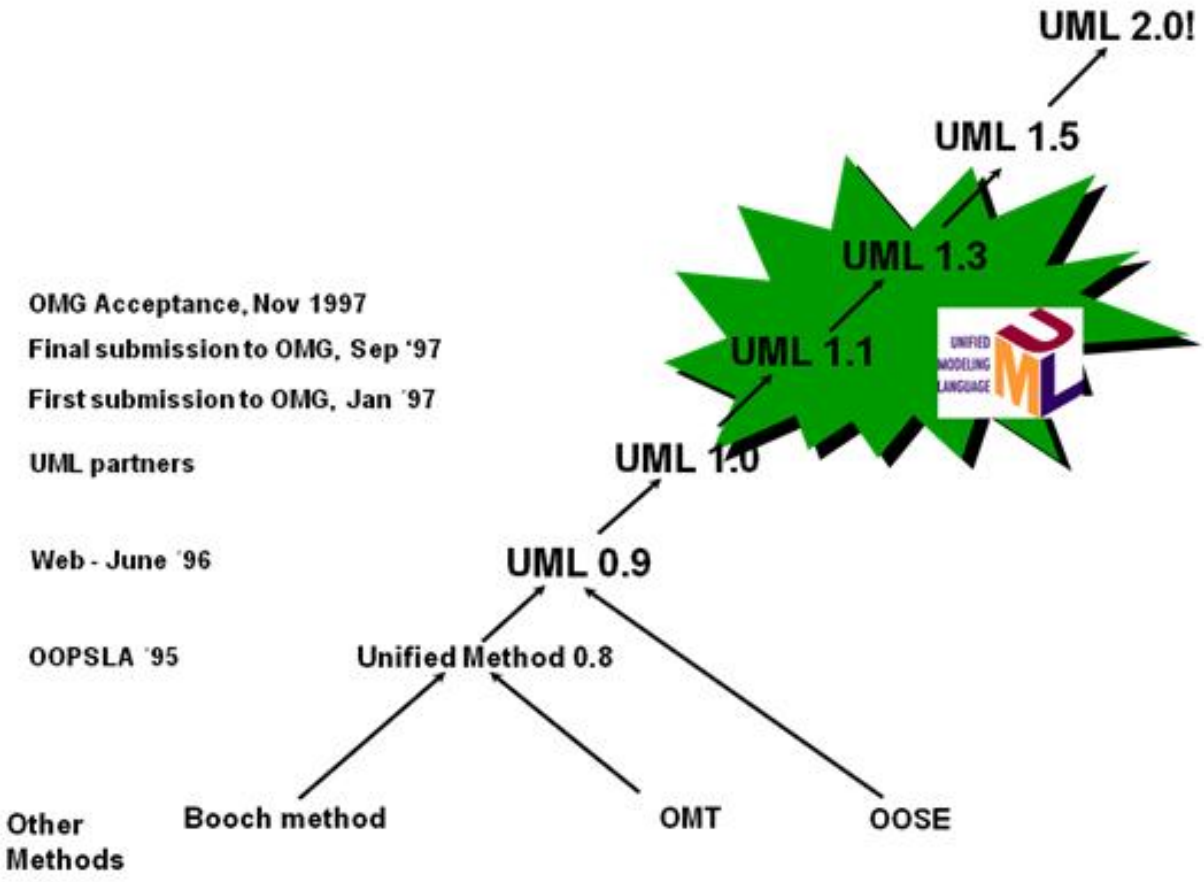
UML'NİN GELİŞİM SÜRECİ

- 1994 yılında Grady Booch (Booch) ve Jim Rumbaugh (OMT) Rational firmasının çatısı altında sahip oldukları iki yöntemi birleştirecek bir yöntem yaratmak için çalışmaya başladılar.

- Firmaya 1995 yılında Ivar Jacobson'ın (OOSE) da katılmasıyla, 3 Amigolar olarak bilinen grup, kendi yöntemlerinin güçlü yönlerini birleştirip bir sistem modelleme dili olarak UML'i geliştirdiler.

- 1997'de OMG (Object Management Group), UML'yi sahiplendi ve açık standart olarak geliştirmeye başladı.

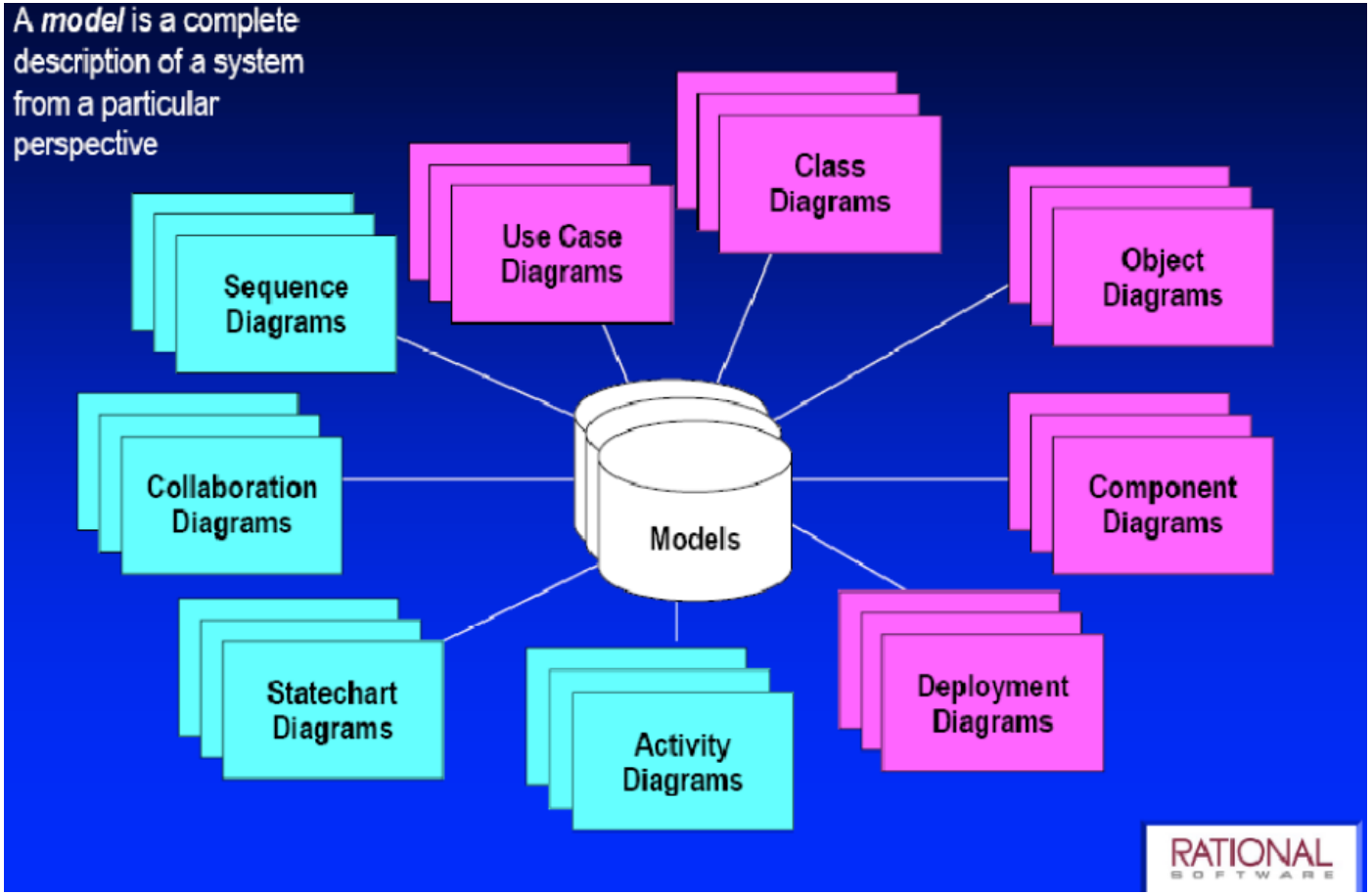
UML'NİN TARİHİ



UML'YE NEDEN GEREK VAR?

- Hataların kolaylıkla fark edilip en düşük seviyeye indirgenmesi.(Risk, zaman, maliyet)
- Yazılım üretiminde başarı oranının düşük olması.
- Yazılımda paylaşım önemlidir. Tüm ekibin aynı dili konuşabilmesi gerekmektedir.
- Sistemin tamamını basit bir dille ve görsellikle görebilmek ve tasarlayabilmek gerekli.
- Modellenmiş ve dokümante edilmiş bir yazılımın tanıtımının kolay olması.
- Yazılım kalitesini arttırma.

UML DİYAGRAMLARI



UML DİYAGRAMLARI

- UML, modelleme için değişik diyagramlar kullanır. Diyagramlar, bir sistem modelini kısmen tarif eden grafiklerdir.
- UML 2.0, 3 bölümde incelenen 13 farklı diyagram içerir.
  - **Yapısal diyagramlarda** modellenen sistemde nelerin var olması gerektiği vurgulanır.
  - **Davranış diyagramlarında** modellenen sistemde nelerin meydana gelmesi gerektiğini belirtir.
  - Davranış diyagramlarının bir alt kümesi olan **Etkileşim diyagramlarında** ise modellenen sistemdeki elemanlar arasındaki veri ve komut akışı gösterilir.

DAVRANIŞ DİYAGRAMLARI

- Kullanıcı Senaryosu (Use-Case) diyagramı
- Durum (Statechart) diyagramı
- Etkinlik (Activity) diyagramı

YAPISAL DİYAGRAMLAR

- Sınıf (Class) diyagramı
- Nesne (Object) diyagramı
- Bileşen (Component) diyagramı
- Paket (Package) diyagramı
- Dağılım (Deployment) diyagramı
- Birleşik Yapı (Composite Structure) diyagramı

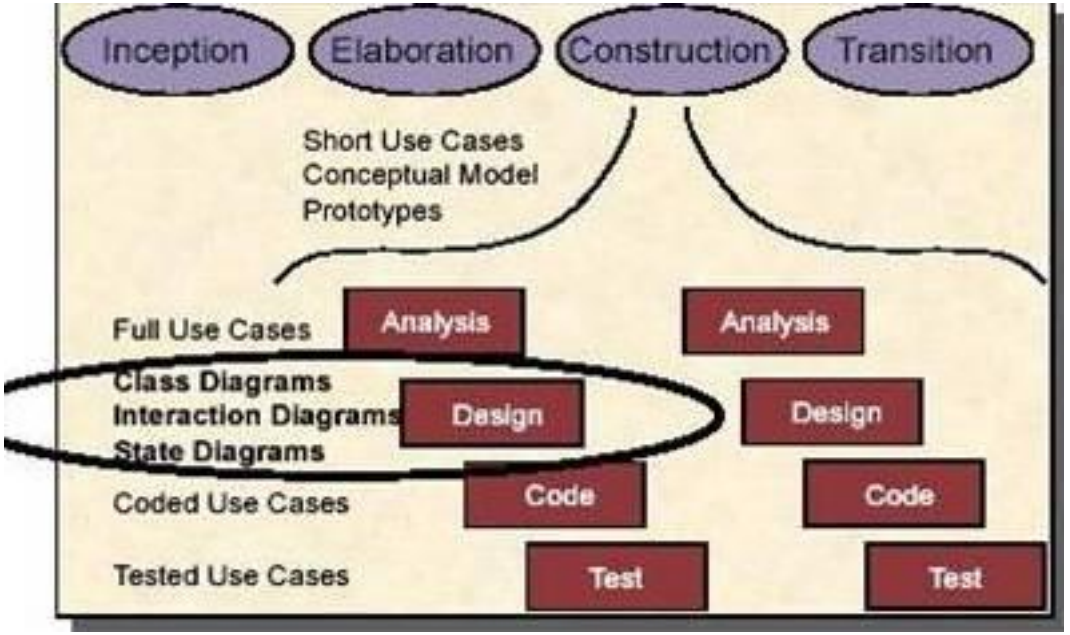
ETKİLEŞİM DİYAGRAMLARI

- Sıralama (Sequence) diyagramı
- İletişim (Communication) diyagramı
- Etkileşime Bakış (Interaction Overview) diyagramı
- Zaman Akış (Timing) diyagramı

USE CASE DİYAGRAMLARI

- Analiz aşamasında Use Case Diyagramları kullanılır.
- Tasarım aşamasında ise modellerin 3 tipi ortaya konulur.

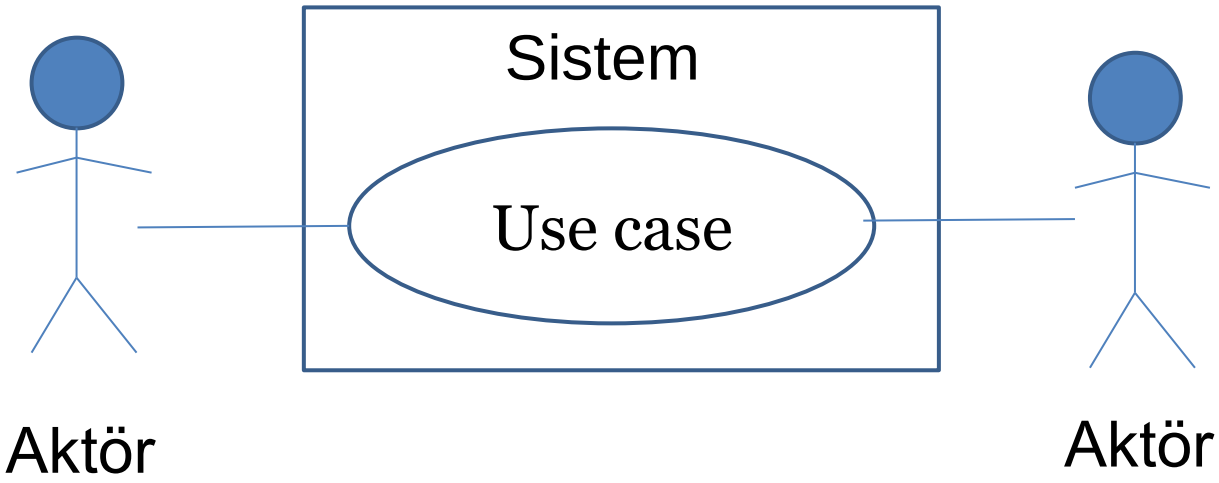
1. Sınıf Diyagramları
2. Durum Diyagramları
3. Etkileşim Diyagramları



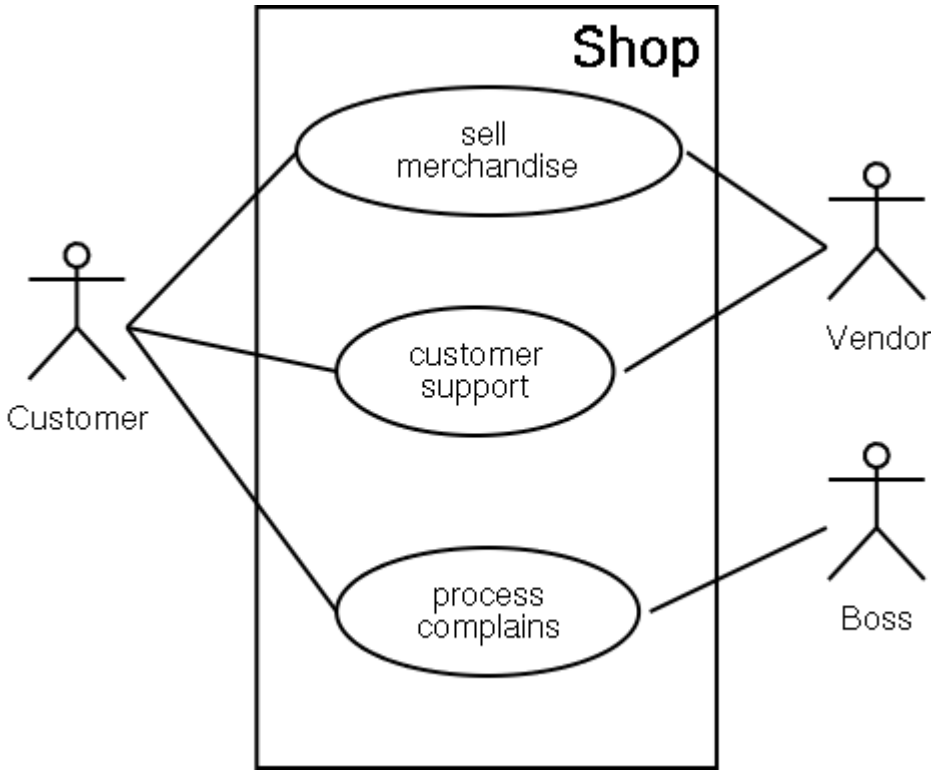
USE CASE DİYAGRAMLARI

- Sistemin çok basit bir şekilde modellenmesini ve işlerin detayının (senaryonun) metin olarak anlatılmasını içerir.
- Aktörden gelen bazı isteklere karşı sistemin yaptığı aktiviteleri gösterir.
- Amaç
  - ✓ Sistemin içeriğini belirtmek.
  - ✓ Sistemin gereksinimlerini elde etmek.
  - ✓ Sistemin mimarisini geçerli kılmak.
- Analistler ve uzmanlar tarafından geliştirilir.

Basit bir use case diyagramı yapısı

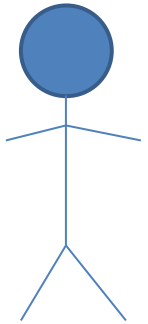


Basit bir use case diyagramı örneği



USE CASE DİYAGRAMLARI BİLEŞENLERİ

- Aktör



- Sistemin kullanıcılarıdır.
- Aktörler genelde belirli bir rol ifade ederler.
- Diğer aktörlerle bağlantılı olabilirler bu bağlantı bir ok ile gösterilir.
- Sistem sınırları dışında gösterilir.

USE CASE DİYAGRAMLARI BİLEŞENLERİ

Use case

- Sistemin destekleyeceği işler.
- Sistem fonksiyonelliğinin büyük bir parçasını gösterir.
- Diğer bir use case ile genişletilebilir.
- Diğer bir use case içerebilir.
- Sistem sınırları içinde gösterilir.

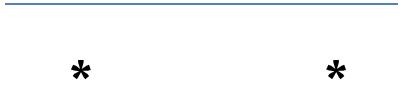
USE CASE DİYAGRAMLARI BİLEŞENLERİ

Sistem sınırı

- İçerisinde sistemin ismi yazılıdır.
- Sistemin kapsamını gösterir.

Bağıntı ilişkisi

- Aktör ve use case'ler arasındaki bağıntıyı gösteren çizgidir.



USE CASE DİYAGRAMLARI BİLEŞENLERİ

Inclusion (içerme) ilişkisi

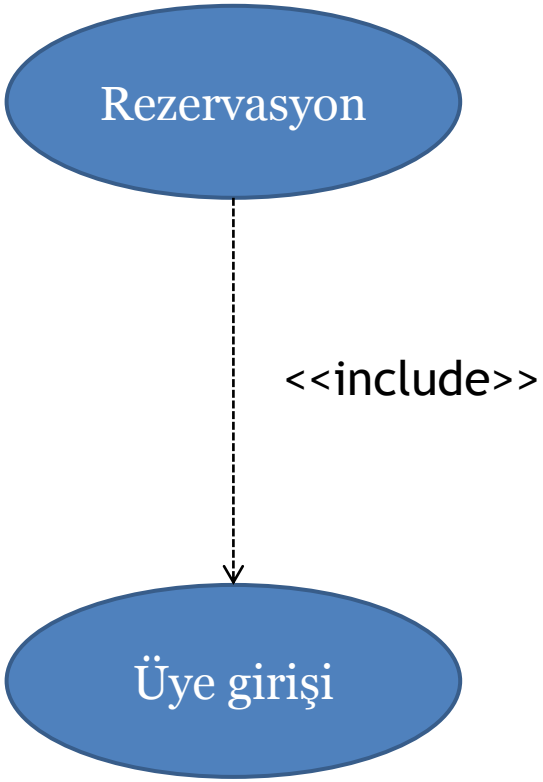
- Bu metotla bir use case içindeki adımlardan birini başka bir use case içinde kullanabiliriz.
- Bir “use-case”in diğerinin davranışını içermesi.
- Inclusion yöntemini kullanmak için <<include>> şeklindeki bir ifade kullanılır.
- Kullanmak istediğimiz use case 'ler arasına çektiğimiz noktalı çizginin üzerine <<include>> yazısını yazarız.

<<include>>

<----------

USE CASE DİYAGRAMLARI BİLEŞENLERİ

Inclusion (içerme) ilişkisi- bir örnek



USE CASE DİYAGRAMLARI BİLEŞENLERİ

Extension (eklenti) ilişkisi

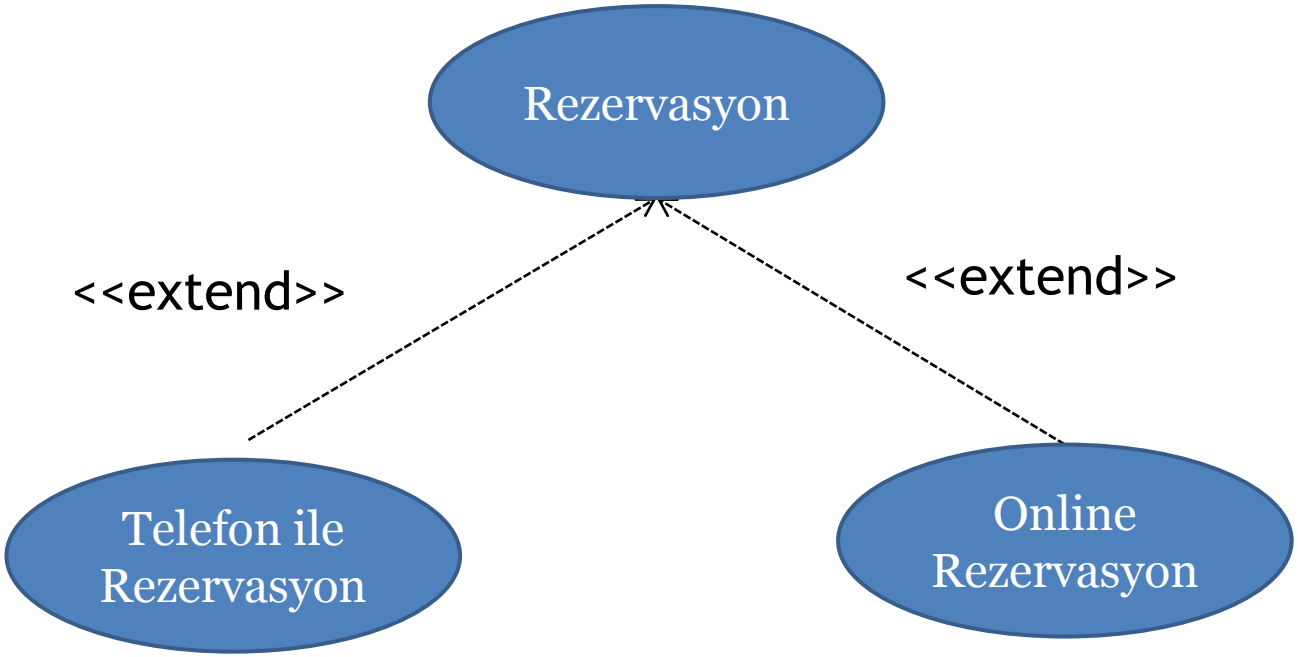
- Bu metodla varolan bir Use Case'e yeni adımlar ekleyerek yeni use case'ler yaratılır.
- Inclusion'da olduğu gibi extension'ları göstermek için yine use case'ler arasına noktalı çizgiler konur ve üzerine <<extend>> ibaresi yazılır.

<<extend>>

<----------

USE CASE DİYAGRAMLARI BİLEŞENLERİ

Extension (eklenti) ilişkisi-bir örnek

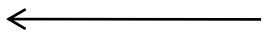


USE CASE DİYAGRAMLARI BİLEŞENLERİ

Genelleme ilişkisi:

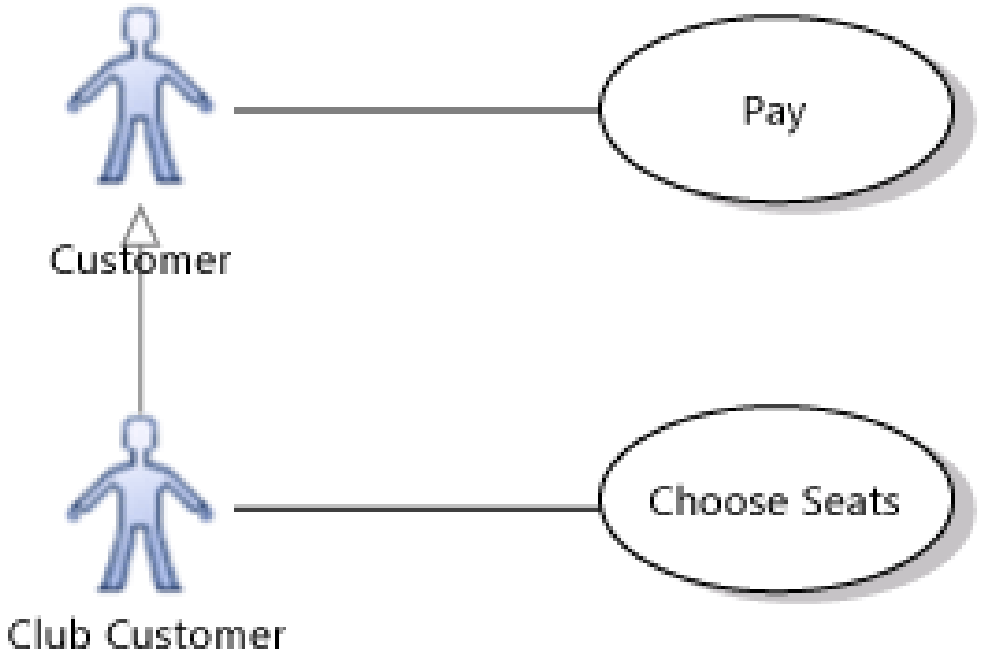
- İki “use-case” veya iki aktör arasındaki kalıtım ilişkisidir. Yani özelleşmiş use case ile daha genel use case arasındaki ilişkidir.

- Özelleşmiş use case’den temel use case’e doğru bir ok ile gösterilir.



USE CASE DİYAGRAMLARI BİLEŞENLERİ

Genelleme ilişkisi:



USE CASE DİYAGRAMI OLUŞTURMADA YÖNTEM

1. Aktörleri ve “use-case”leri belirle.

Amaç: Sisteminin aktörlerini ve “use-case”lerini belirlemek ve üst seviye “use-case” modelini oluşturmak.

- Aktörler belirlenir
- “Use-case”ler belirlenir
- Her aktör ve “use case” kısaca tanımlanır
- Üst seviye “use-case” modeli tanımlanır

2. “Use-case”leri detaylandır.

Amaç: Belirlenen tüm “use-case”lerin is akışlarını detaylı olarak tanımlamak.

- Ana akıs tanımlanır
- Alternatif akıslar tanımlanır

USE CASE DİYAGRAMI OLUŞTURMADA YÖNTEM

3. “Use-case” modelini yapılandır

Amaç: Oluşturulan use case modelini ortak noktaları en aza indirecek şekilde yapılandırmak.

- Gereken yerlerde “extend” ve “include” iliskileri kullanılabilir
- Yapılandırılan “use-case” modeli, is süreçlerini referans alınarak değerlendirilir.

4. Kullanıcı arayüzlerini tanımla

Amaç: Use case tanımları esas alınarak kullanıcı arayüzlerini üst seviyeli olarak tanımlamak.

- Kağıt üzerinde çizim yapılabilir
- Arayüz prototipleme aracı kullanılabilir

Bir örnek: ATM uygulaması

Bir bankanın ATM cihazı için yazılım geliştirilecektir.

ATM, banka kartı olan müşterilerin hesaplarından para çekmelerine, hesaplarına para yatırmalarına ve hesapları arasında para transferi yapmalarına olanak sağlayacaktır.

ATM, banka müşterisi ve hesapları ile ilgili bilgileri, gerektiğinde merkezi banka sisteminden alacaktır.

Bir örnek: ATM uygulaması

ATM uygulama yazılımının kullanıcıları:

- Banka müşterisi
- Merkezi Banka Sistemi

} Aktörler

Bir örnek: ATM uygulaması

Belirlenen aktörler ATM'den ne istiyorlar?

- Aktör: Banka müşterisi
  - **Para çekme**
  - **Para yatırma**
  - **Para transferi**

- Aktör: Merkezi Banka Sistemi
  - **Günlük özet alma**

Bir örnek: ATM uygulaması

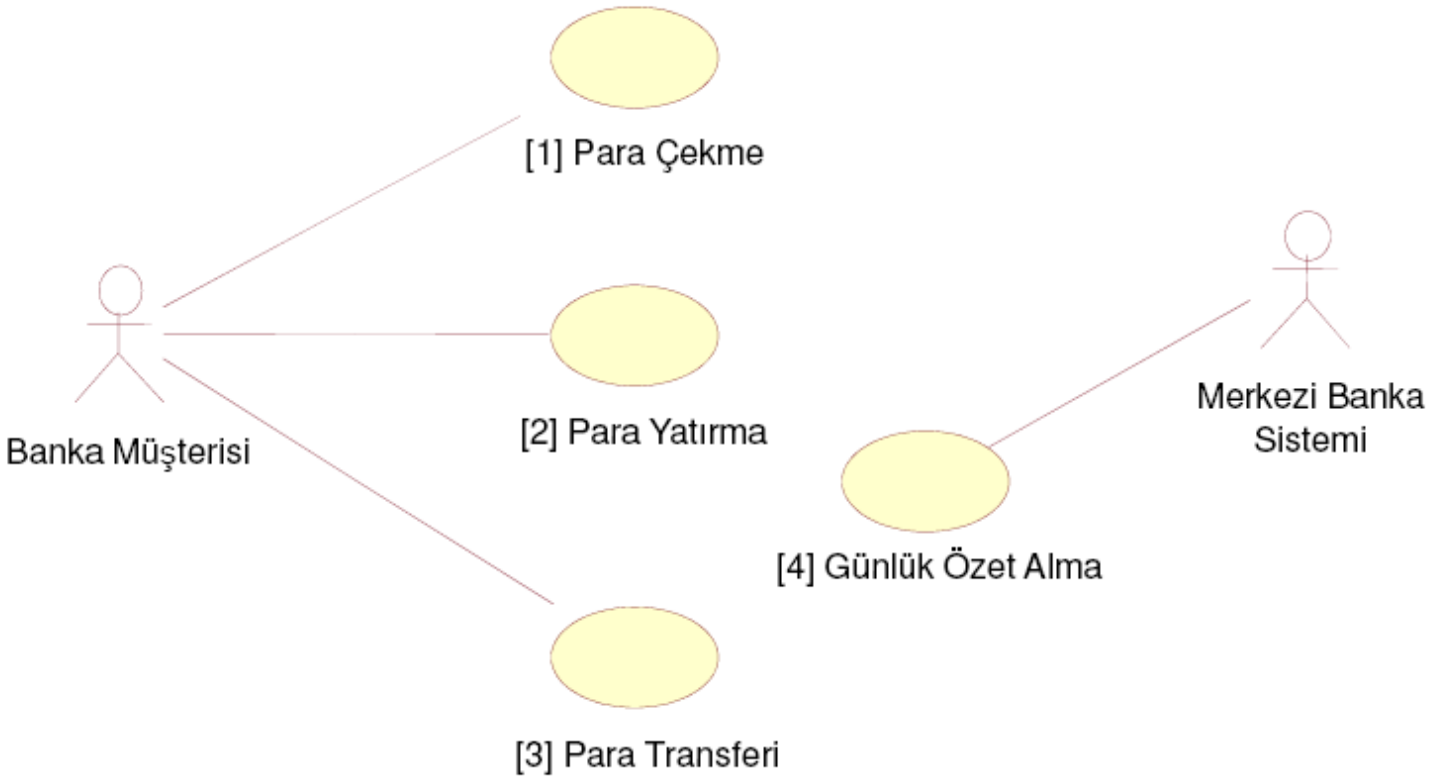
- Aktör: Banka müşterisi

  Bankada hesabı ve banka kartı olan, ATM'den işlem yapma hakkı olan kişidir.

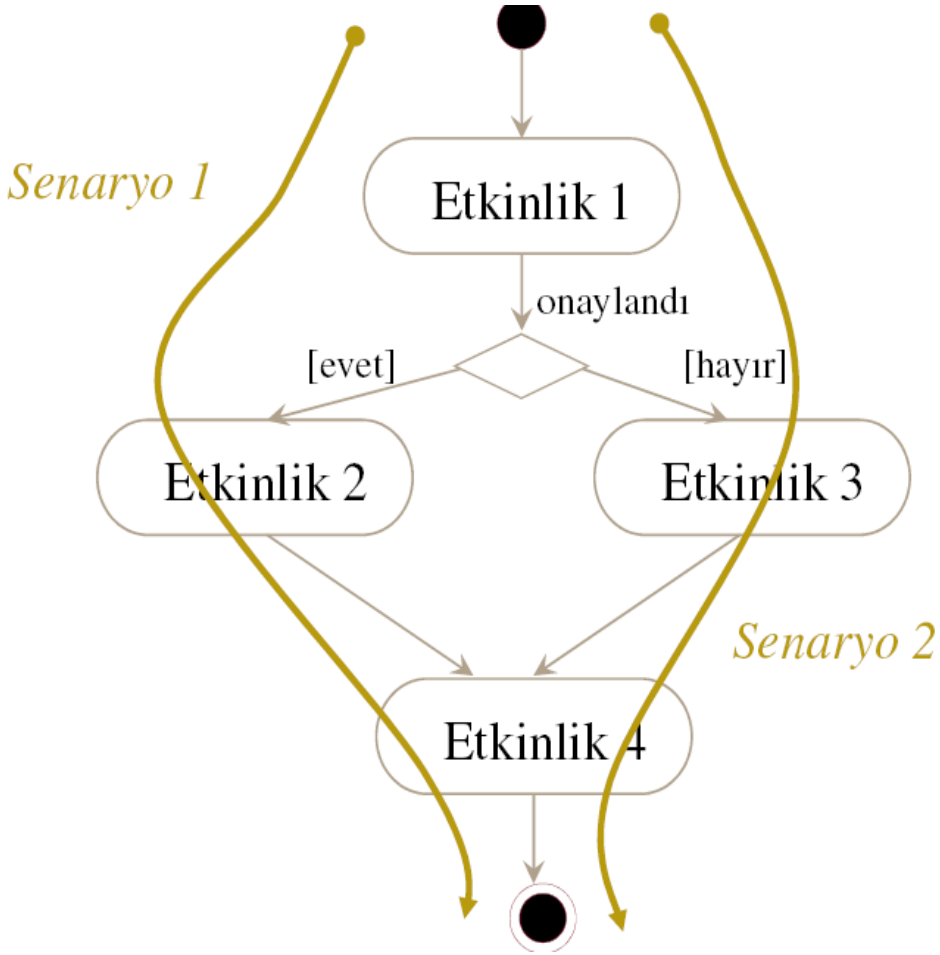
- Use case: Para çekme

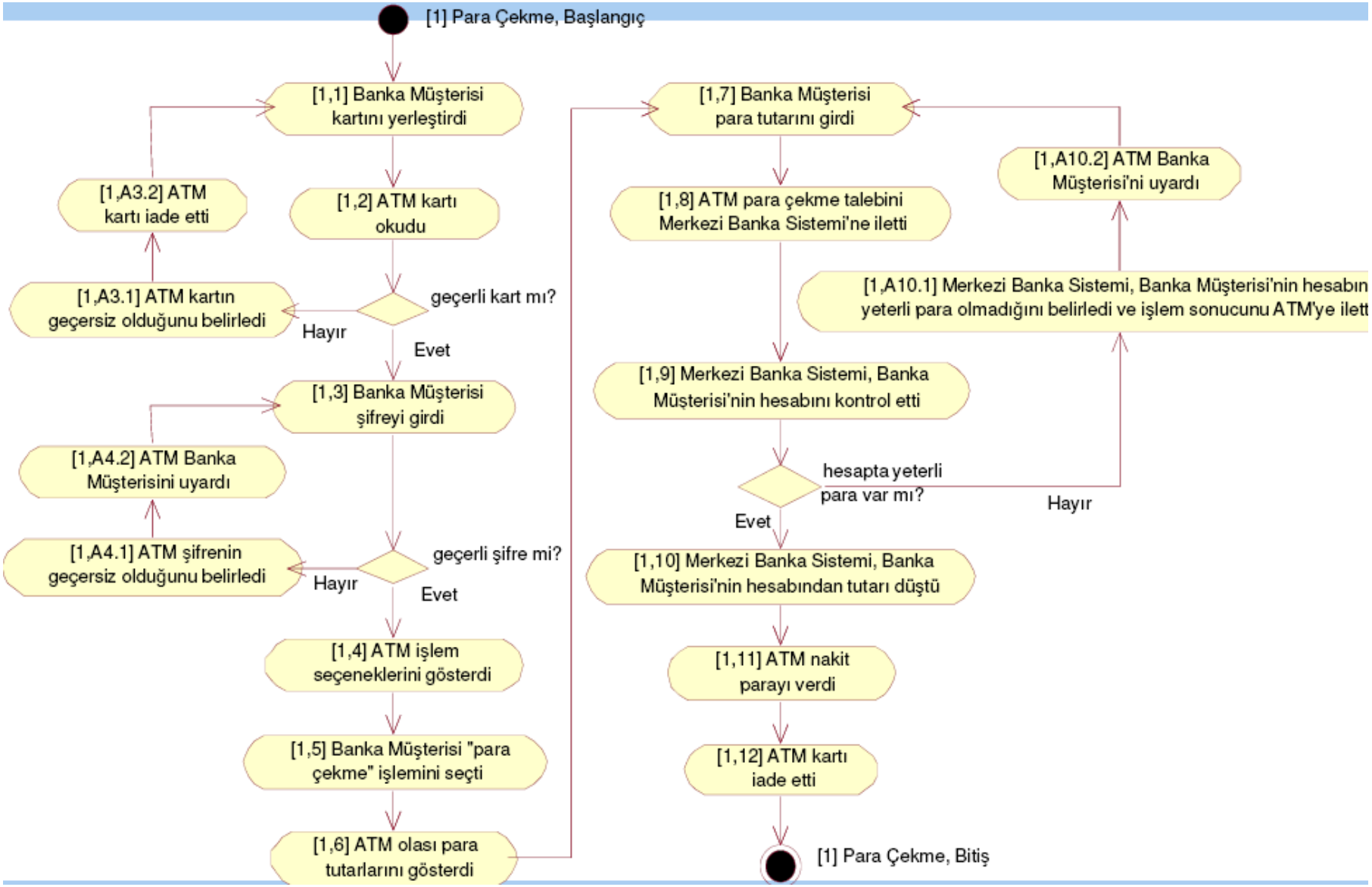
  Banka müşterisinin nasıl para çekeceğini tanımlar. Para çekme işlemi sırasında banka müşterisinin istediği tutarı belirtmesi ve hesabında bu tutarın mevcut olması gerekir.

Bir örnek: ATM uygulaması

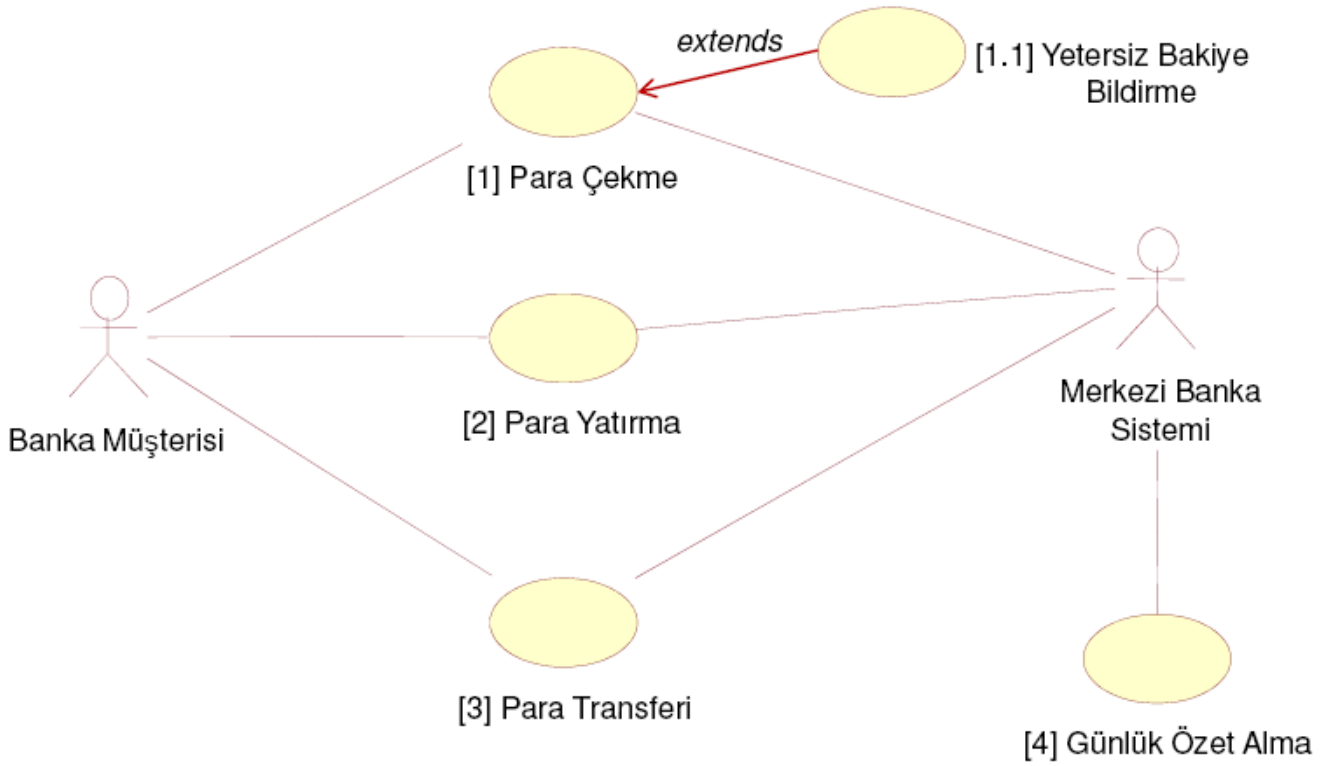


Bir örnek: ATM uygulaması

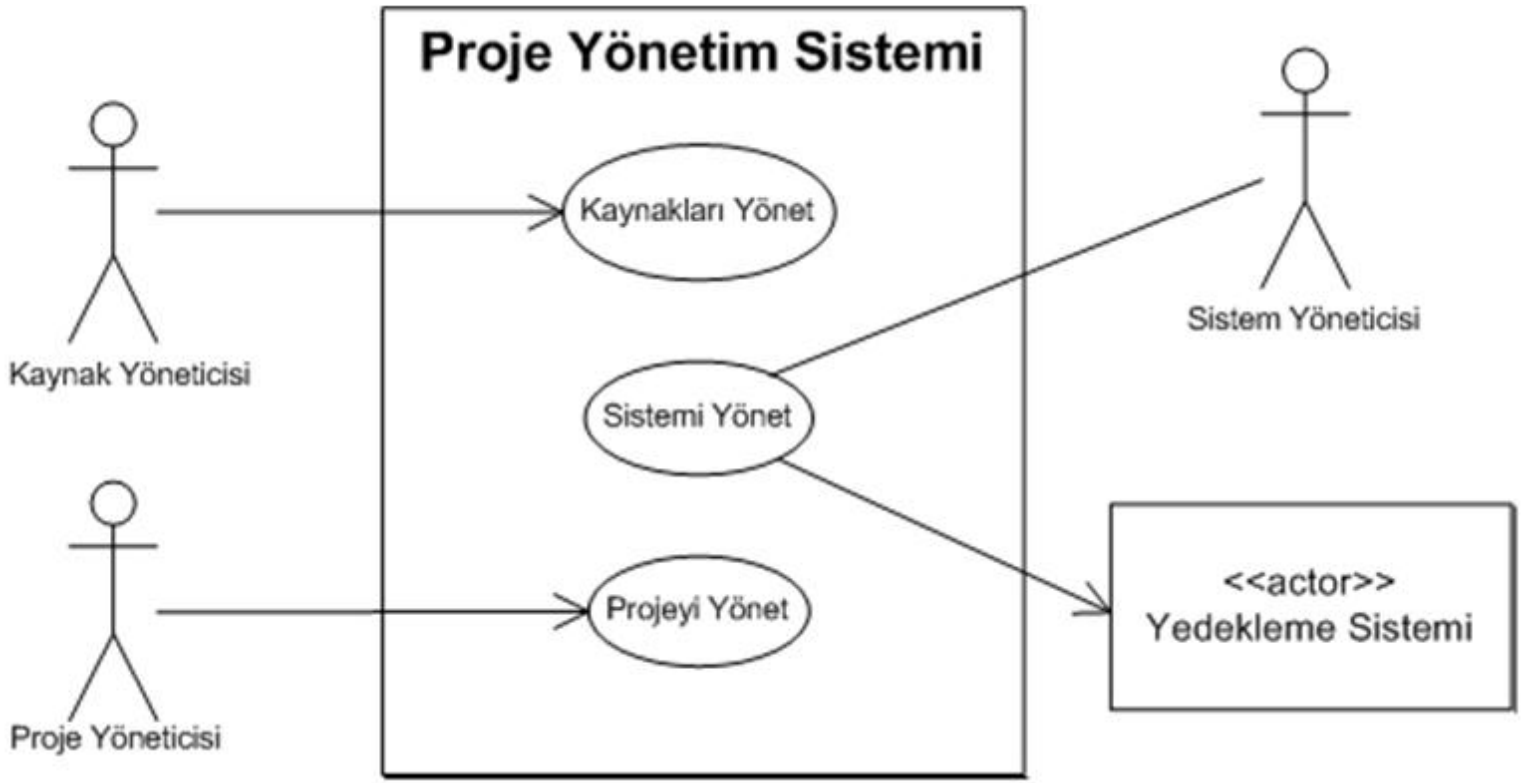



[1] Para Çekme, Başlangıç
[1,1] Banka Müşterisi kartını yerleştirdi
[1,2] ATM kartı okudu
geçerli kart mı?
Hayır
[1,A3.1] ATM kartın geçersiz olduğunu belirledi
[1,A3.2] ATM kartı iade etti
Evet
[1,3] Banka Müşterisi şifreyi girdi
geçerli şifre mi?
Hayır
[1,A4.1] ATM şifrenin geçersiz olduğunu belirledi
[1,A4.2] ATM Banka Müşterisini uyardı
Evet
[1,4] ATM işlem seçeneklerini gösterdi
[1,5] Banka Müşterisi "para çekme" işlemini seçti
[1,6] ATM olası para tutarlarını gösterdi
[1,7] Banka Müşterisi para tutarını girdi
[1,8] ATM para çekme talebini Merkezi Banka Sistemi'ne iletti
[1,9] Merkezi Banka Sistemi, Banka Müşterisi'nin hesabını kontrol etti
hesapta yeterli para var mı?
Hayır
[1,A10.1] Merkezi Banka Sistemi, Banka Müşterisi'nin hesabın yeterli para olmadığını belirledi ve işlem sonucunu ATM'ye ilett
[1,A10.2] ATM Banka Müşterisi'ni uyardı
Evet
[1,10] Merkezi Banka Sistemi, Banka Müşterisi'nin hesabından tutarı düştü
[1,11] ATM nakit parayı verdi
[1,12] ATM kartı iade etti
[1] Para Çekme, Bitiş

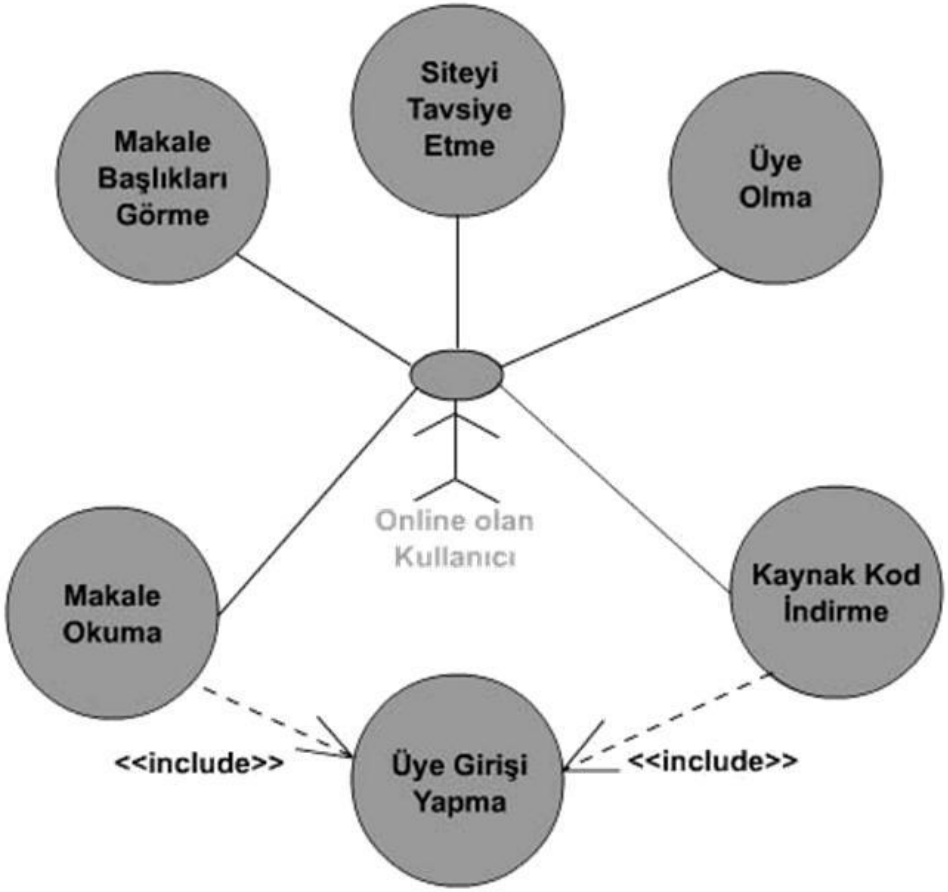
Bir örnek: ATM uygulaması



Bir use case diyagramı örneği



Bir use case diyagramı örneği



Bir web sayfasına gelen bir kullanıcının neler yapabileceğini use case diyagramlarıyla göstermeye çalışalım. Siteye gelen bir kullanıcı kayıtsız şartsız makale başlıklarını görebilmektedir. Online olan kullanıcı Siteyi tavsiye edebilir, siteye üye olabilir,kitapları inceleyebilir. Ancak makale okuması ve kaynak kod indirebilmesi için siteye üye girişi yapmalıdır. Makale okuması ve kaynak kod indirebilmesi için gereken şart siteye üye olmaktır.Siteye bağlanan bir kullanıcının site üzerindeki hareketlerini belirtir diyagram bu şekilde oluşturulabilir.

UML'NİN AVANTAJLARI-1

- Kodlama kolaylığı sağlar. UML ile uygulamanızın tasarımı analiz aşamasında yapıldığı için, modellemeniz bittikten hemen sonra kod yazmaya başlayabilirsiniz.
- Kullanılan tekrar kod sayısı ayırt edilebilir bu sayede verim sağlanır.
- Mantıksal hataların minimum seviyeye düşürülmesini sağlar. Bütün sistem tasarlandığı için oluşabilecek hataların düzeltilmesi de daha kolaydır.
- Geliştirme maliyetinin düşmesini sağlar.

UML'NİN AVANTAJLARI-2

- UML diyagramları ile yazılım tamamını görebileceğimiz için verimli bellek kullanımı sağlanabilir.
- Karmaşık sistemlerde değişiklik yapmayı kolaylaştırır.
- UML ile dokümanlaştırılmış kodları düzenlemek daha az zaman alacaktır.
- UML diyagramlarını kullanan yazılımcılar aynı dili konuşacaklarından kolay iletişim sağlanır. Ayrıca müşteriler ve teknik sorumlular diyagramlar üzerinden kolaylıkla iletişim kurabilirler.

Kaynaklar

- Martin Fowler, Kendall Scott, “Rafine UML”, Alfa Yayınları, 2003.
- http://en.wikipedia.org/wiki/Unified_Modeling_Language.
- http://web.itu.edu.tr/~kanoglu/crs-iscpm-systemmodeling.pdf
- ftp://ftp.cs.hacettepe.edu.tr/pub/dersler/BBS6XX/BBS651_YM/ders%20notlari/hafta-05%20(4%20kasim)/BBS-651-DN04.pdf