Yazılım Testi ve Proje Yönetimi

11.03.2024

Yıl içi projesinin görsel çözümlemelerinin UML ile gerçekleştirilmesi

Component (Bileşen) Diyagramı

Yıl içi Projesinde Çalışmanın İçeriği olarak hazırlanacak

- Statik bir diyagramdır. Diğer bir ifade ile problemin üst düzey çözümünde yapısal (structural) modelleme gerçekleştirir.
- Büyük bir sistemin basitleştirilmiş bir görünümüdür.

Amacı:

Sınıflardan oluşan grupların bileşenler şeklinde sınıflandırılmasıdır.

Böylece:

Kodun değiştirilebilmesi

Yeniden kullanımı sağlanır.

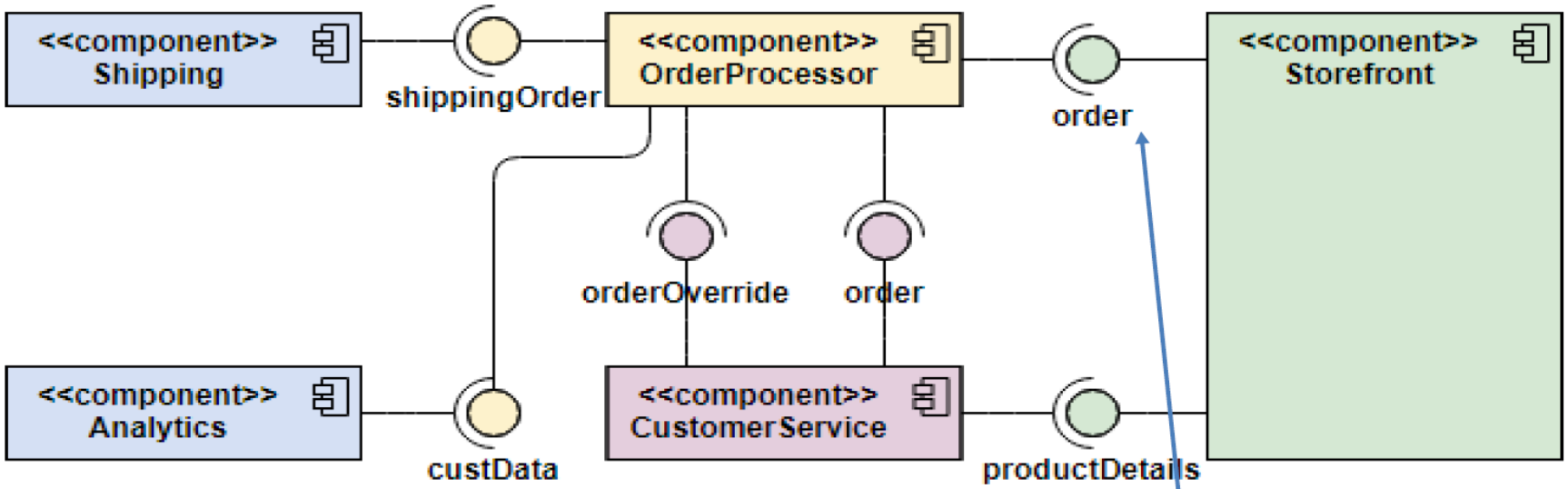
Özetle:

Component diyagramı ile bileşenlerin nasıl oluştuğu ve sistemle etkileşimleri ifade edilir.

UML ile Betimlenmesi ne anlama Gelir?

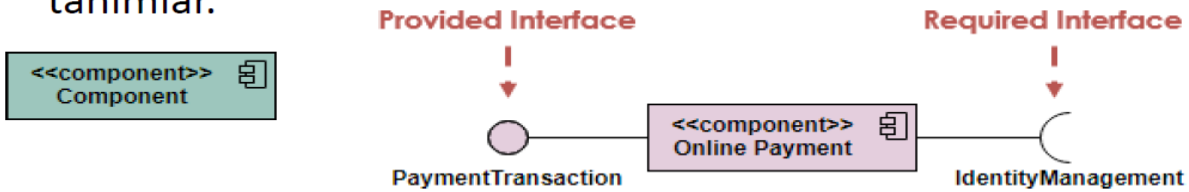
- Bir bileşen (component) diyagramının temel amacı, bir sistemin bileşenleri arasındaki yapısal ilişkileri göstermektir.
- UML'de bileşenler (components), benzer bir amaca hizmet etmek üzere sınıflandırılmış **yazılım nesnelerinden** oluşur.
- Bileşenler (components), bir veya daha fazla arabirimin sağlandığı bir sistem veya alt sistem içindeki **bağımsız, kapsüllenmiş** (encapsulated) birimlerdir.
- Bir sınıf grubunun (birden fazla sınıfın komposizyonu) **bileşen** (component) olarak sınıflandırılmasının amacı, bileşenler değiştirilip yeniden kullanıldığında tüm sistemin **daha modüler bir yapıda** çalışacağıdır.
- Bileşenler (components), bir **bileşenin kapsüllenmesini** sağlar.
  - Sadece bileşenin arabirimler aracılığıyla etkileşimde bulunduğu araçlar görüntülenir.

Basit bir Bileşen (Component) Diyagramı

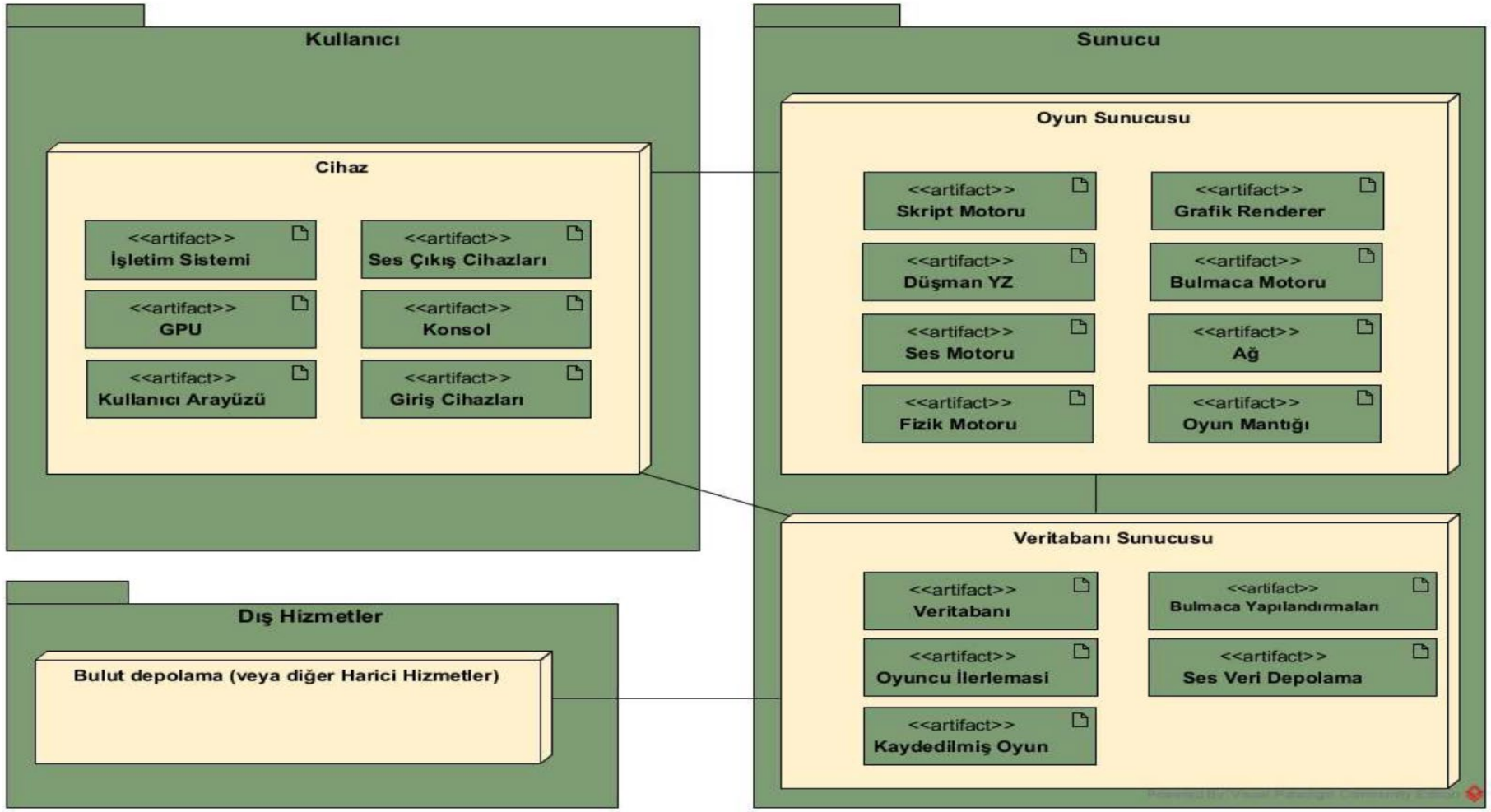


Components can be "wired" together using to form subsystems, with the use of a ball-and-socket joint.

Provided ve Required arayüzler, "belirli bir arayüz uygulandığında sınıflar tarafından sağlanması gereken bir dizi public öznitelikler (attributes) ve metotlar (operations) tanımlar.



Örnek Yazılım Projesinin /Oyun Uygulamasının Deployment Diyagramı



Nesneye Yönelik Tasarımın bir Özelliği: Kapsülleme (Encapsulation)

- Bir programın temeli hedefi, bir modelinin tasarlanmasıyla çözüme ulaşılmasıdır.
- Nesne yönelimli programlama ise, programlama hatalarını azaltır ve kodun **yeniden kullanımını** destekler.
  - Python, nesne yönelimli bir dildir.
  - Python'da tanımlanan **nesneler**
    - ✓ Identity /Kimlik Her nesne farklı olmalı ve bu test edilebilir olmalıdır.

      «is» ve «is not» testleri bu amacı sağlar.

      State / Durum Her nesne bir duruma sahip olmalıdır (içermelidir). **fields** ve **instance variables** (örnek değişkenler)gibi **attributes** (öznitelikler) bu amacı oluşturur.

      Behaviour / Davranış Her nesne durumunu aktifleştirmelidir. **methods** (yöntemler) bu amacı gerçekleştirir.

is / is not problem

Tedarik sorunu ile ilgili tanımlamalar içeren problem

Problem: Supply Shortage	
IS	**IS NOT**
• What - What is the problem? • Where - It is from overseas supplier's end. • When - It is a problem from last two months. • How Big - It is a small issue if solved quickly.	• What - What is not the problem? • Where - It is not from the domestic supplier's end. • When- It is not the typical issue. • How Big - It is not a big problem.

Python Dili ve Nesneye Yönelik Özellikleri

- Class based object creation /Sınıf tabanlı nesne oluşturma
  - Sınıflar, nesneleri oluşturan şablonlardır.
  - Nesneler, ilişkili davranışa sahip veri yapılarıdır.
- Inheritance with Polymorphism /Polimorfizm ile Kalıtım
  - Python, tekli ve çoklu kalıtımı destekler.
  - Tüm Python örneklerinde yöntemler çoklu şekilcidir (polymorphic).
  - Tüm yöntemler alt sınıflar tarafından geçersiz kılınabilir (overridden)
- Encapsulation with data hiding / Veri Gizleme ile Kapsülleme
  - Python, özniteliklerin (attributes) gizlenmesine izin verir.
  - Gizlendiğinde, sınıf dışından özniteliklere yalnızca sınıfın yöntemleri aracılığıyla erişilebilir .
  - Sınıflar, verileri değiştirmek için yöntemlerin implementasyonunu gerçekleştirir.

Bu slayt sadece bilgilendirme amaçlıdır.

Bir Python uygulaması

```
class MyClass
attr1 = 10                  #class attributes
attr2 = "hello "
def method1(self):
print MyClass.attr1     #reference the class attribute
def method2(self):
print MyClass.attr2     #reference the class attribute
def method3(self, text):
self.text = text            #instance attribute
print text, self.text       #print my argument and my attribute
method4 = method3    #make an alias for method3
```

- Sınıfın bir özniteliğinin (attributes) tüm örnekleri (instances) MyClass.attr1 olarak yazılır.
- Örnek özniteliklerin örneklerine (instances) yapılan tüm referanslar, self değişkeni ile yapılır (self.text).
- Sınıfın dışında, sınıf özniteliklerine yapılan tüm erişimler sınıf adıyla, MyClass .attr1 veya sınıfın bir örneğiyle x.attr1 yapılabilir.
- Sınıfın dışında, örnek değişkenlere yapılan tüm başvurular, sınıfın bir örneğiyle x.text yazılır.

Sınıf /Nesne Tanımlamaları

Bu slayt sadece bilgilendirme amaçlıdır

```
class name (superclasses):    #sınıf tanımı
assignment
. .
function
. .

x = MyClass()    #sınıf örneğinin oluşturulması
x.attr1 = 1      # sınıf örneğine  bir  «attribute» ataması
x.attr2 = 2
……
x.attrN = n
```

Miras Özelliğinin Öneminin Python ile Gösterimi

- Python hem tekli ,hem de çoklu kalıtımı destekler.
  - Tekli kalıtım, yalnızca bir üst sınıf ile, çoklu kalıtım ise birden fazla üst sınıf ile gerçekleşir.
- Bir üst sınıftaki herhangi bir öznitelik veya yöntem aynı zamanda herhangi bir alt sınıftadır.
  - Öznitelik veya yöntem gizli olmadığı sürece sınıfın kendisi tarafından kullanılabilir.
- Bir alt sınıfın herhangi bir örneği, bir üst sınıfın örneğinin kullanılabildiği her yerde kullanılabilir;
  - Bu bir polimorfizm örneğidir.
- Tüm bu bu özellikler, yeniden kullanım ve genişletme (extend) kolaylığı sağlar.

```
class Class1: pass              #no inheritance
class Class2: pass
class Class3(Class1): pass      #single inheritance
class Class4(Class3, Class2): pass    #multiple inheritance
```

Deployment Diyagram

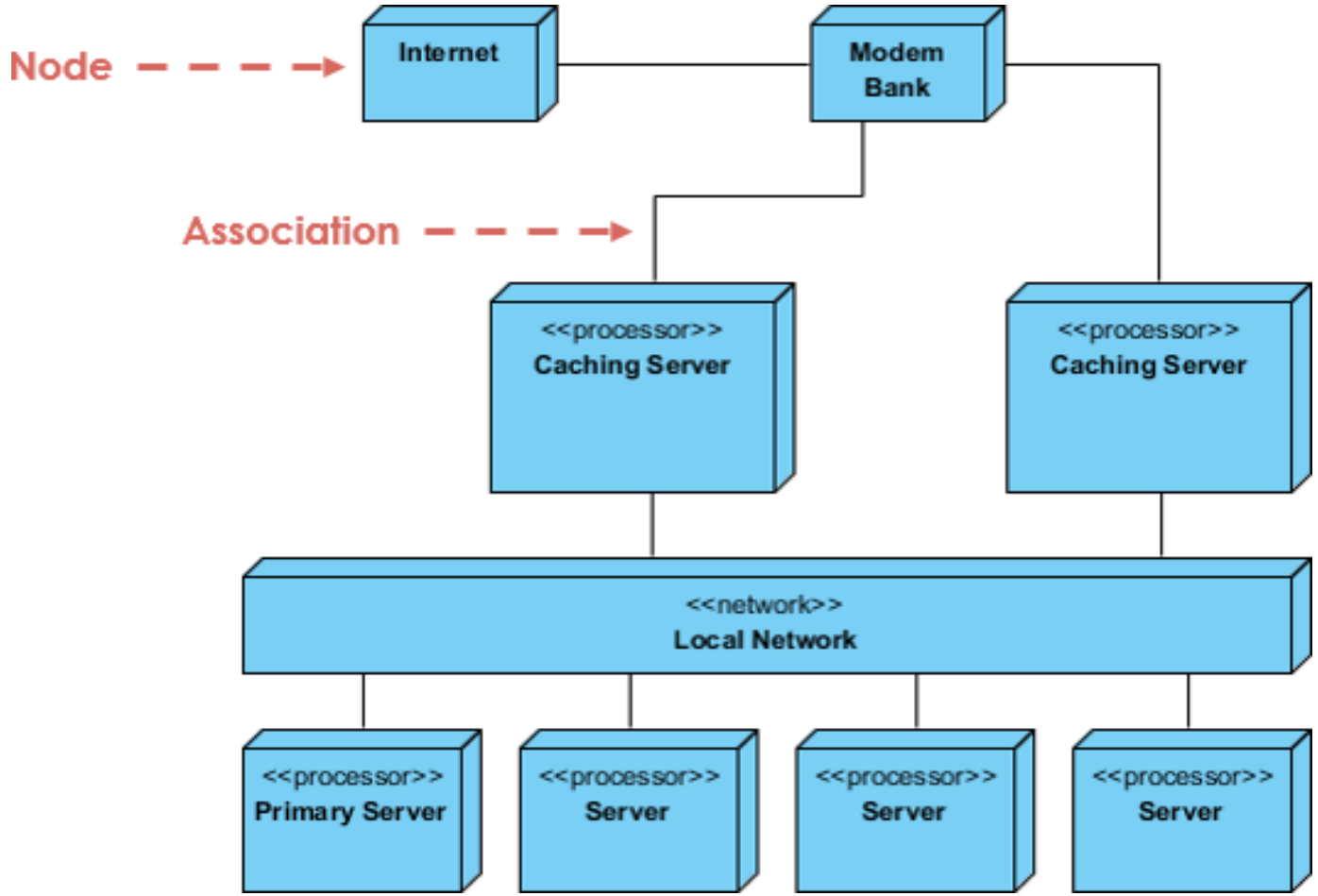
Yıl içi Projesinde Yazılım Mimarisi olarak hazırlanacak

Yeni eklenecek bir sistem mevcut sistemlerden (önceden geliştirilmiş) hangileriyle etkileşimde olacaktır ?

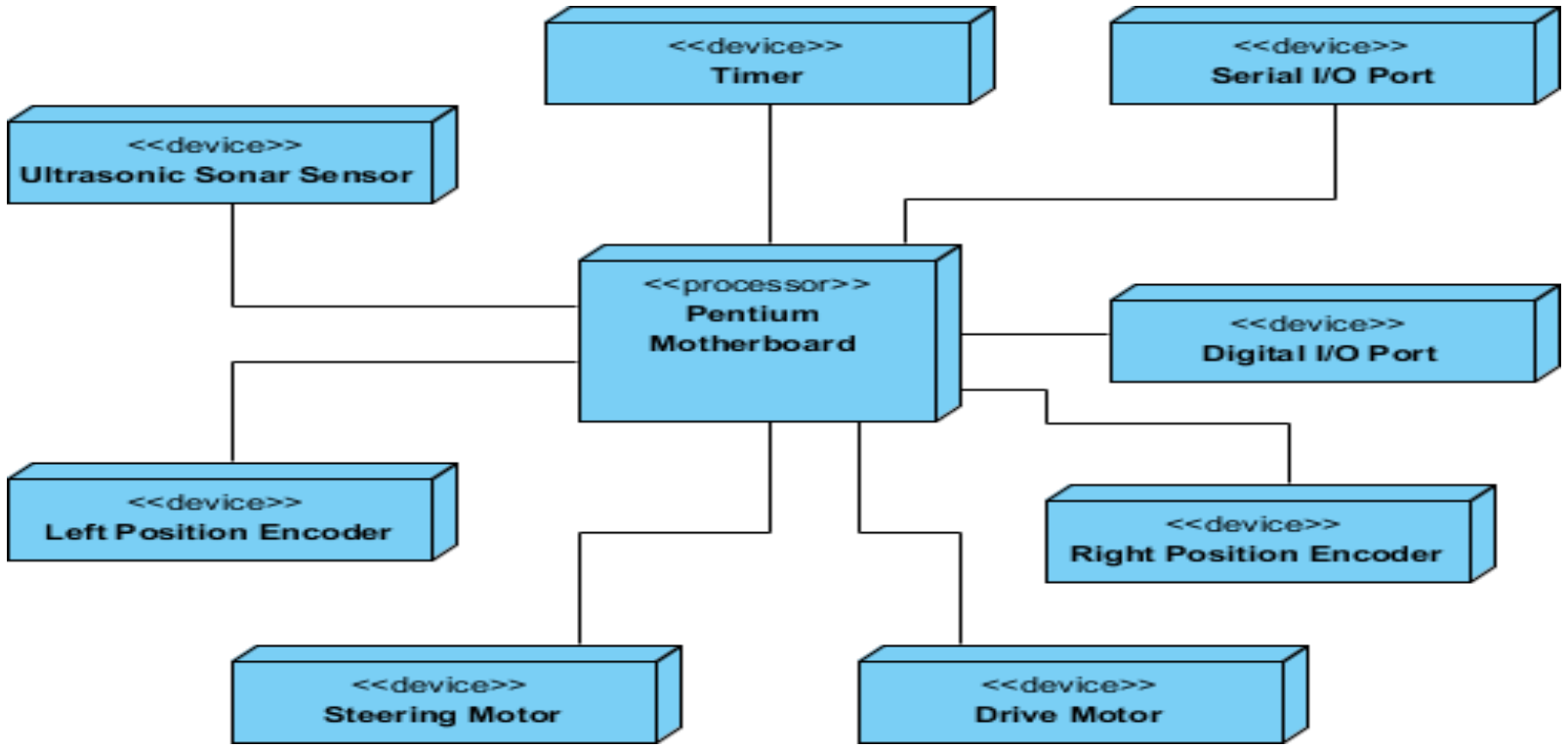
- Sistemin sağlamlığı (**robust**) ölçüsü nedir? Örneğin, sistemin çökmesi (**failure)** durumunda yedek donanım stratejisi nedir?
- Sisteme kimler ve neler bağlanacak veya sistemle etkileşime girecektir? Bunu nasıl yapacaklar?
- Sistem, işletim sistemi ve iletişim şekli ve protokollerle birlikte hangi ara yazılımı (middleware) kullanacaktır?
- Kullanıcılar hangi donanım ve yazılımlarla doğrudan etkileşime girecekler? (PC'ler, ağ bilgisayarları, tarayıcılar vb.)
- Dağıtımdan sonra (after deployed) sistem nasıl izlenecektir?
- Sistemin ne kadar güvenli olmalıdır? (bir güvenlik duvarı, fiziksel olarak güvenli bir donanım ihtiyacı vb.)

Deployment Diyagramın Amacı

- Sistemin çalışma zamanı (run time) yapısı gösterilir.
- Sistemin implementasyonu sırasında kullanılacak farklı donanım öğeleri arasındaki bağlantıları içerir.
- Fiziksel donanım öğeleri ve aralarındaki iletişim yolları modellenir.
- Bir sistemin mimarisini planlamak için kullanılabilir.
- Yazılım bileşenlerinin veya düğümlerin dağıtımı belgelenebilir.


Node
Association
Internet
Modem Bank
<<processor>>
Caching Server
<<processor>>
Caching Server
<<network>>
Local Network
<<processor>>
Primary Server
<<processor>>
Server
<<processor>>
Server
<<processor>>
Server

Gömülü Sistem için Deployment Diyagram Örneği



Use Case Diyagramı (Dinamik Diyagram)

- use case bir sistemin ya da sistemin bir parçasının davranışını betimler. Bu bağlamda değişimleri içeren bir dizi eylemin (action) tanımlaması yapılır.
  - Böylece sistem, herhangi bir aktöre (actor) ait olan gözlemlenebilir bir sonucun değerini verecektir.

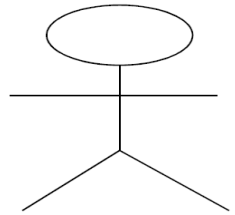
The UML User Guide (Booch, 1999)

- Aktör bir sistem, alt sistem veya sınıfla etkileşime giren dışsal (external) bir kişi, bir süreç veya şeydir.
  - Bir aktör, dışsal kullanıcıların sistemle olan etkileşimlerini karakterize eder.

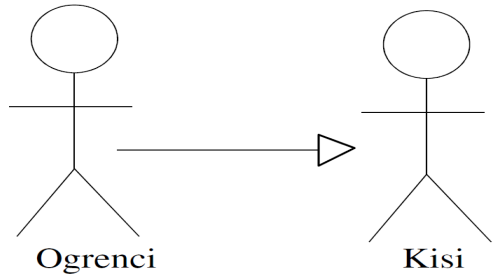
UML Reference Manual (1999, Rumbaugh)

Use case Diyagramının Görsel Betimlemesi

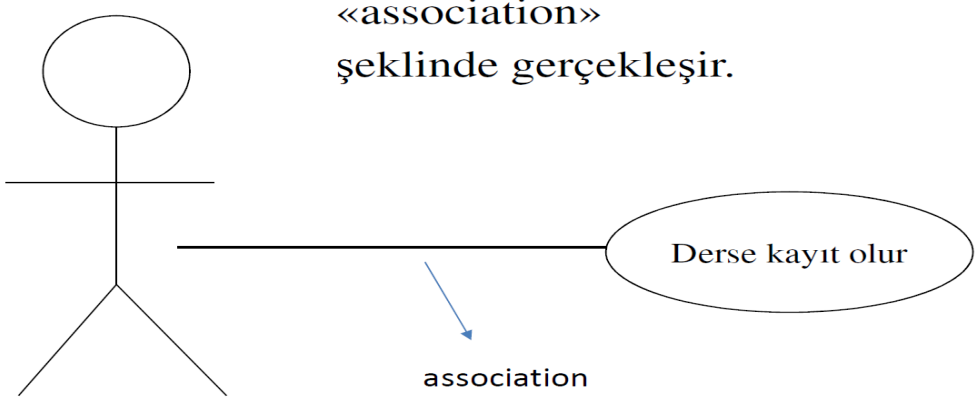
- Herhangi bir «use case», use case diyagramında **elips** ile gösterilir.



- Bir «use case» her zaman ismi ile etiketlenir
- Her aktör bir veya daha fazla «use case» ile ilişkili olabilir
- Bir aktörün , başka bir aktör ile genelleştirme (generalization)ilişkisi olabilir



Aktörler ile use case bağlantısı «association» şeklinde gerçekleşir.



«Use Case» Tanımlamaları

- Bir «use case» olası işlevlerini tanımlarken aktörünün aşağıdaki soruları cevaplaması beklenir.
  - Bunlar ne (what) sorularıdır.
    - ✓Aktör ne gerçekleştirmek istemektedir?
    - ✓Aktörün neler yapabilme kapasitesi vardır?
    - ✓Aktörün temel görevi (task) nedir?
    - ✓Aktörün sistemden alması gereken bilgiler nedir?
    - ✓Aktör sisteme ne bilgiler sağlar?

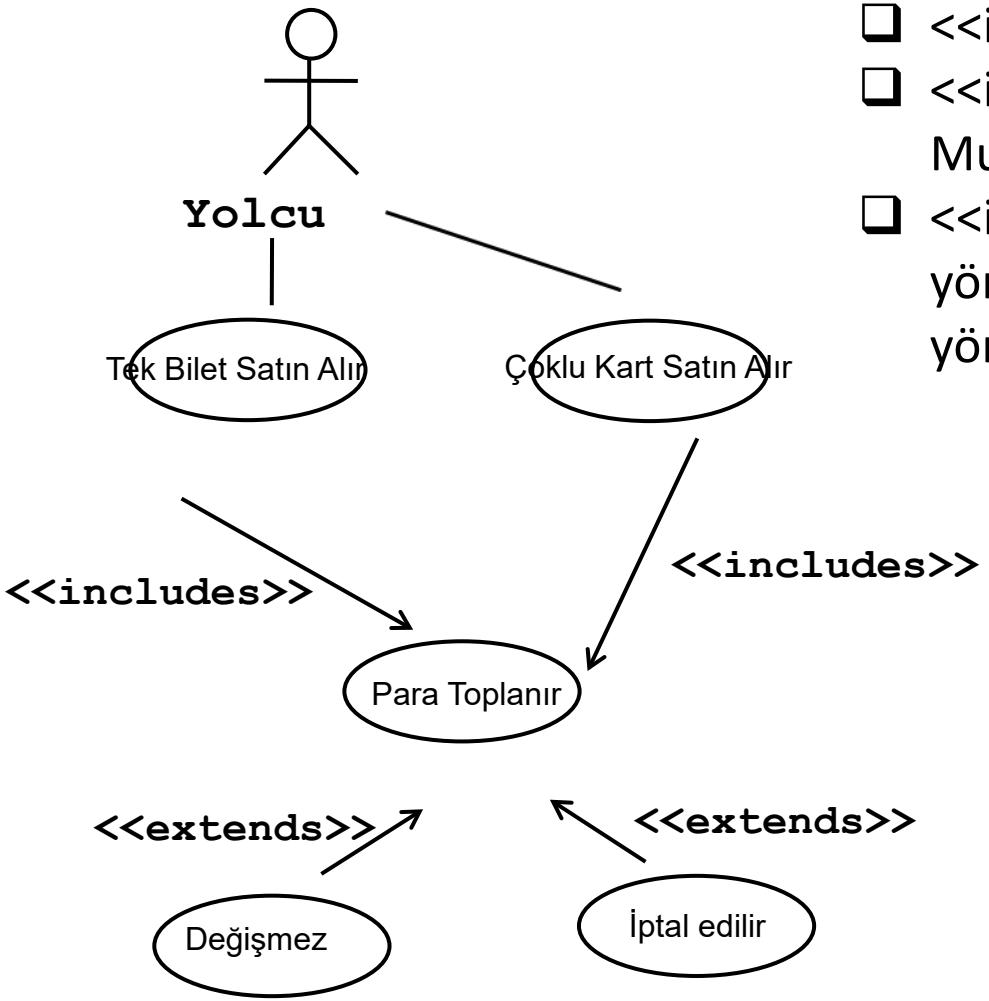
Aktör ve «Use Case» arasındaki etkileşim

- Bir aktör canlı ya da cansız bir varlık olarak pek çok «use case» ile iletişimde olabilir.
- Bir «use case» bir dizi işlevini gerçekleştirirken bir veya daha fazla aktör ile iletişimde olabilir.
- İki «use case» hem aynı varlık (tek aktör) ile ilişkilenip hem de birbiri ile iletişimde bulunamaz.
  - Çünkü her bir «use case» varlığı (aktörü) kendi kullanmaktadır.

Ama:

- İki <<use case>> birbirleri ile iletişimde ise, include ya da extend ilişkisindeki use case KESİNLİKLE aktörle bağıntılı olamaz. Zira, aktörle ilişkiyi diğer <<use case>> (temel use case) sağlar.

<<includes>> ve <<extends>> İlişkileri Örneği



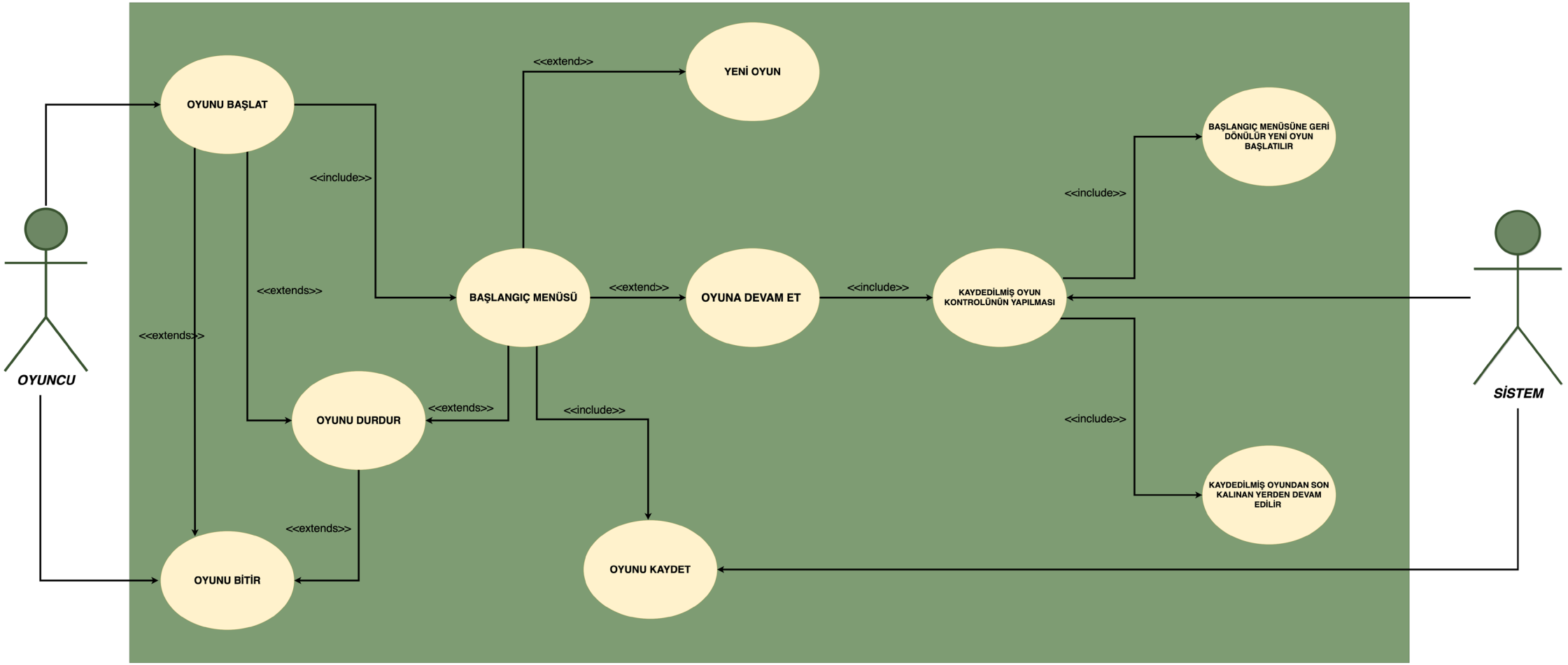
- <<includes>> ilişkisi «use case» in davranışını simgeler.
- <<includes>> davranışı yeniden kullanımdır (reuse) . Mutlaka içerilmelidir. Hariç tutulamaz.
- <<includes>> ilişkisinin yönü, kullanılan use case yönündedir. Yani, «extends» ilişkisinin yönünün tersi yönde işaretlenir.

- Kural dışı (exceptional) veya nadir (seldom) karşılaşılan ilişkiler <<extends>> ilişkisidir.
- Kural dışı olay akışları temel <<use case>> akışının açıklaştırılmasında kullanılır.
- Olası (exceptional) akışları betimleyen «use case» ler birden fazla olası «use case» e genişleyebilir.
- <<extends>> okunun yönü temel use case yönüne doğrudur.

include / extend ilişkileri arasındaki fark

	İnclude use case	Extend use case
Bu «use case» seçmeli midir?	No	Yes
«Temel use case» bu «use case» olmadan işlevini tamamlar mı?	No	Yes
Bu «use case» in çalışması koşula bağlı mıdır?	No	Yes
Bu «use case» «temel use case» in davranışını değiştirir mi?	No	Yes

Örnek Yazılım Projesinin /Oyun Uygulamasının Use Case Diyagramı: Başlangıç Menusu Tasarımı



https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

Use Case Diyagramı: Başlangıç Menusu Tasarımı

Use Case Numara:	**1**
Use Case İsmi	*Başlangıç Menüsü Kullanımı*
Ön Koşul	*Oyuncunun oyunu başlattığı varsayılır*
Son Koşul	*Oyuncu Ana menü ekranını kullanarak oyununu başarılı bir şekilde başlatmıştır*
Temel yol	*1. Kullanıcı Oyunu başlat butonuna tıklar* *2. Kullanıcı karşısına çıkar ara yüzden yeni oyun seçeneğini seçer.* *3. Oyun yazılımı oyunu başlatır.*
Alternatif Yol	*1. Kullanıcı Oyunu Başlat butonuna tıklar* *2. Oyuna devam et butonuna tıklar* *3. Oyun yazılımı kaydedilmiş bir oyun var mı kontrolü yapar* *4. Oyun yazılımı kaydedilmiş bir oyun dosyası bulur ise* *5. Oyunu son kalan yerden devam ettirir*
Alternatif Yol:	*1. Kullanıcı Oyunu Başlat butonuna tıklar* *2. Oyuna devam et butonuna tıklar* *3. Oyun yazılımı kaydedilmiş bir oyun var mı kontrolü yapar* *4. Oyun yazılımı kaydedilmiş bir oyun dosyası bulamaz ise* *5. Oyun yazılımı başlangıç menüsüne geri döndürür.*
Alternatif Yol:	*1. Oyuncu kaydet butonuna tıklayarak oyunu sonlandırır*
Alternatif Yol:	*1. Kullanıcı oyunu başlat butonuna tıklar* *2. Kullanıcı oyunu durdur butonuna tıklar* *3. Kullanıcı oyunu bitir butonuna tıklar*