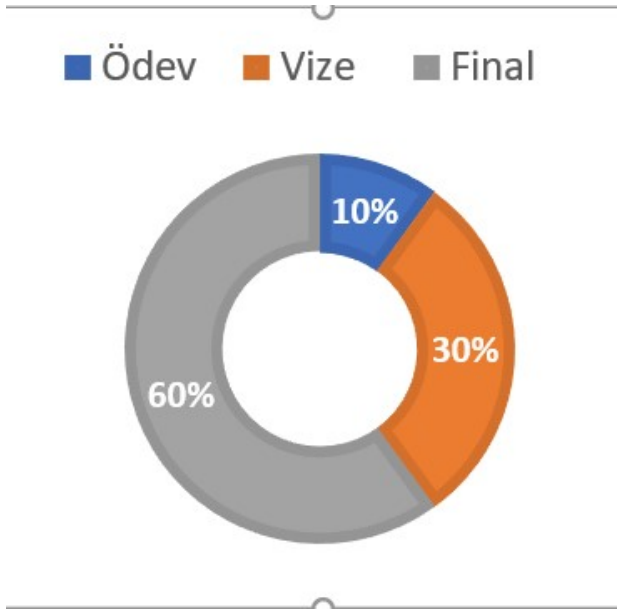
Yazılım Mühendisliği
1906003082015

Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

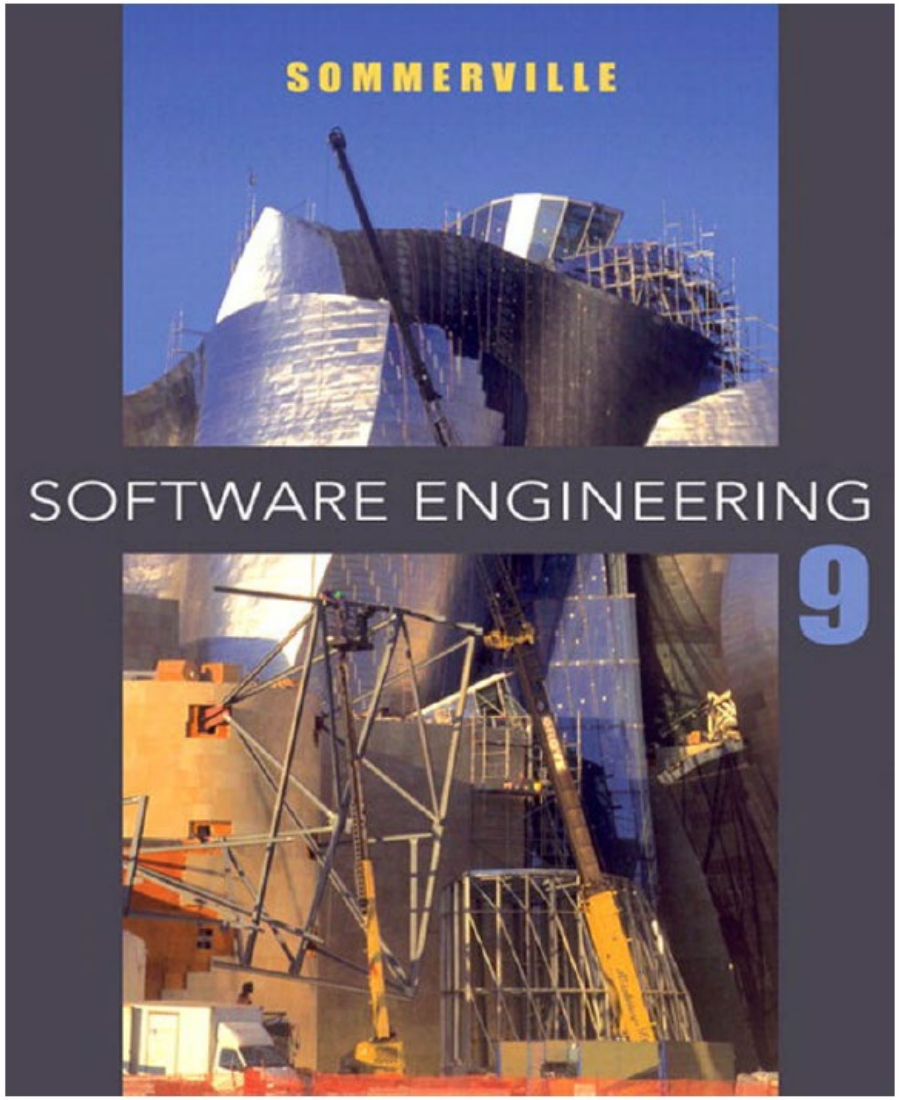
Giriş

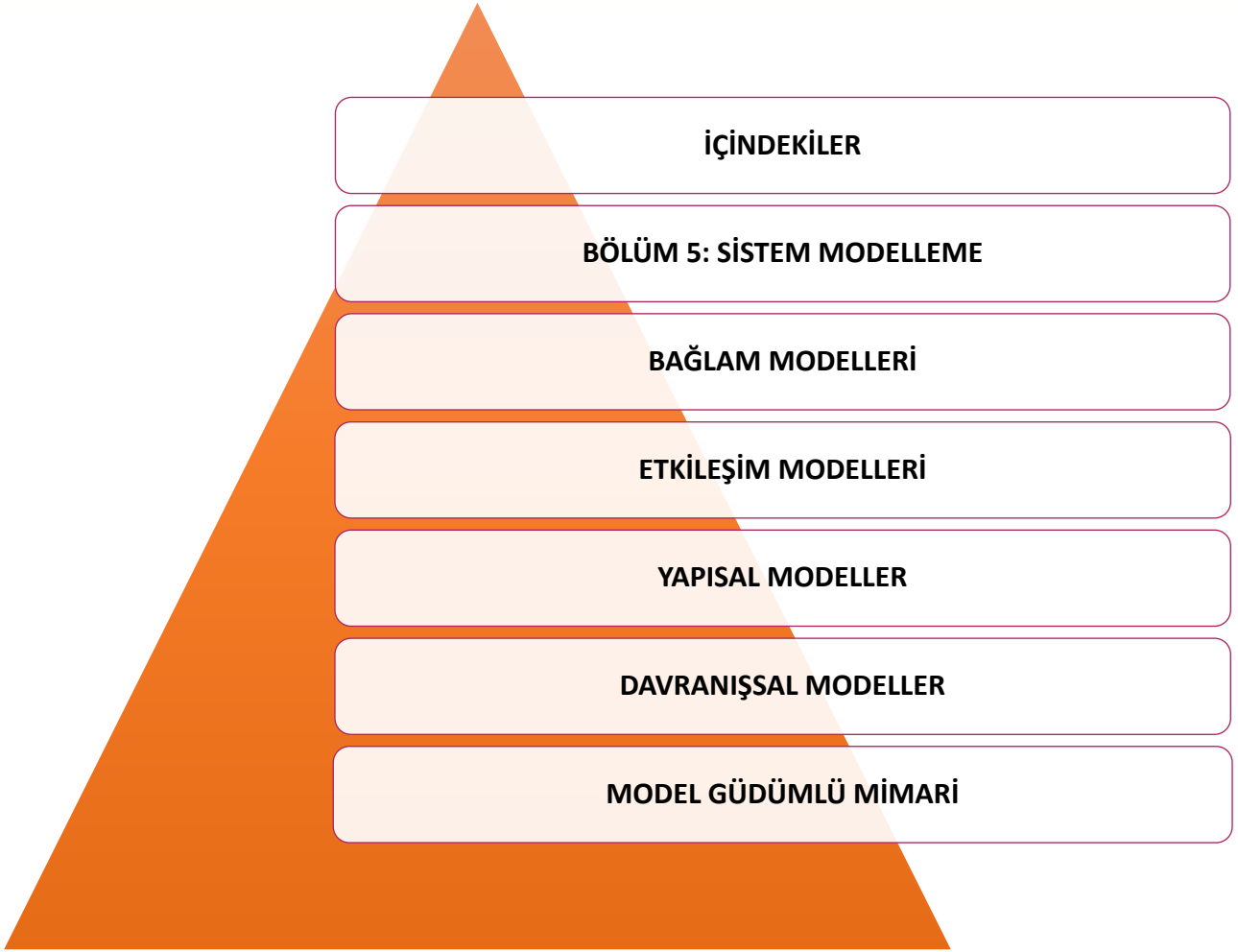
Ders Günü ve Saati:
Salı: 09:15-13:00
Devam zorunluluğu %70



HAFTA	KONULAR
Hafta 1	Yazılım Mühendisliğine Giriş
Hafta 2	Yazılım Geliştirme Süreç Modelleri
Hafta 3	Yazılım Gereksinim Mühendisliği
Hafta 4	Yazılım Mimarisi
Hafta 5	Nesneye Yönelik Analiz ve Tasarım
Hafta 6	Laboratuar Çalışması: UML Modelleme Araçları
Hafta 7	Yazılım Test Teknikleri
Hafta 8	Ara Sınav
Hafta 9	Yazılım Kalite Yönetimi
Hafta 10	Yazılım Bakımı - Yeniden Kullanımı ve Konfigürasyon Yönetimi
Hafta 11	Yazılım Proje Yönetimi (Yazılım Ölçümü ve Yazılım Proje Maliyet Tahmin Yöntemleri)
Hafta 12	Yazılım Proje Yönetimi (Yazılım Risk Yönetimi)
Hafta 13	Çevik Yazılım Geliştirme Süreç Modelleri
Hafta 14	Yazılım Süreci İyileştirme, Yeterlilik Modeli (CMM)

Kaynaklar




İÇİNDEKİLER
BÖLÜM 5: SİSTEM MODELLEME
BAĞLAM MODELLERİ
ETKİLEŞİM MODELLERİ
YAPISAL MODELLER
DAVRANIŞSAL MODELLER
MODEL GÜDÜMLÜ MİMARİ

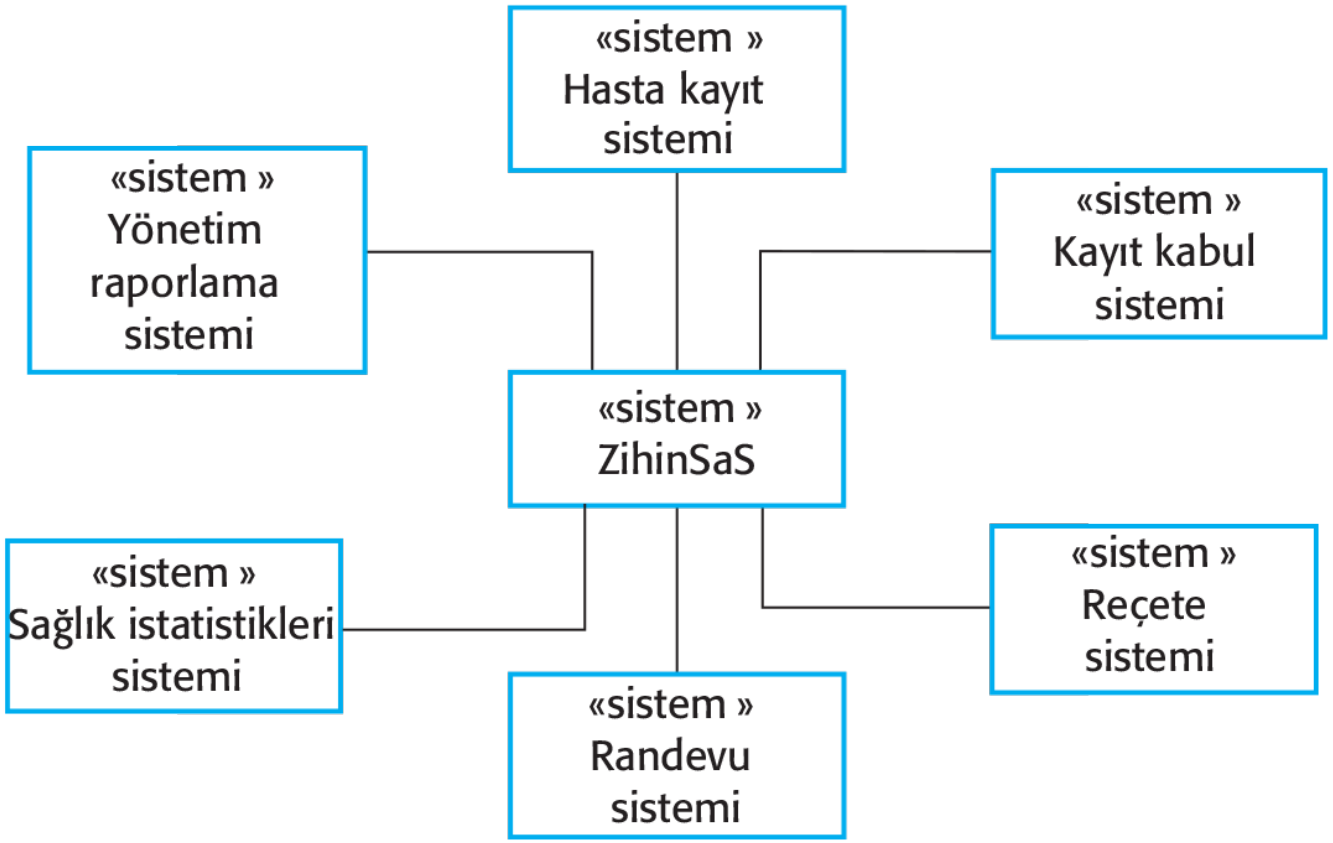
5.BÖLÜM

SİSTEM MODELLEME

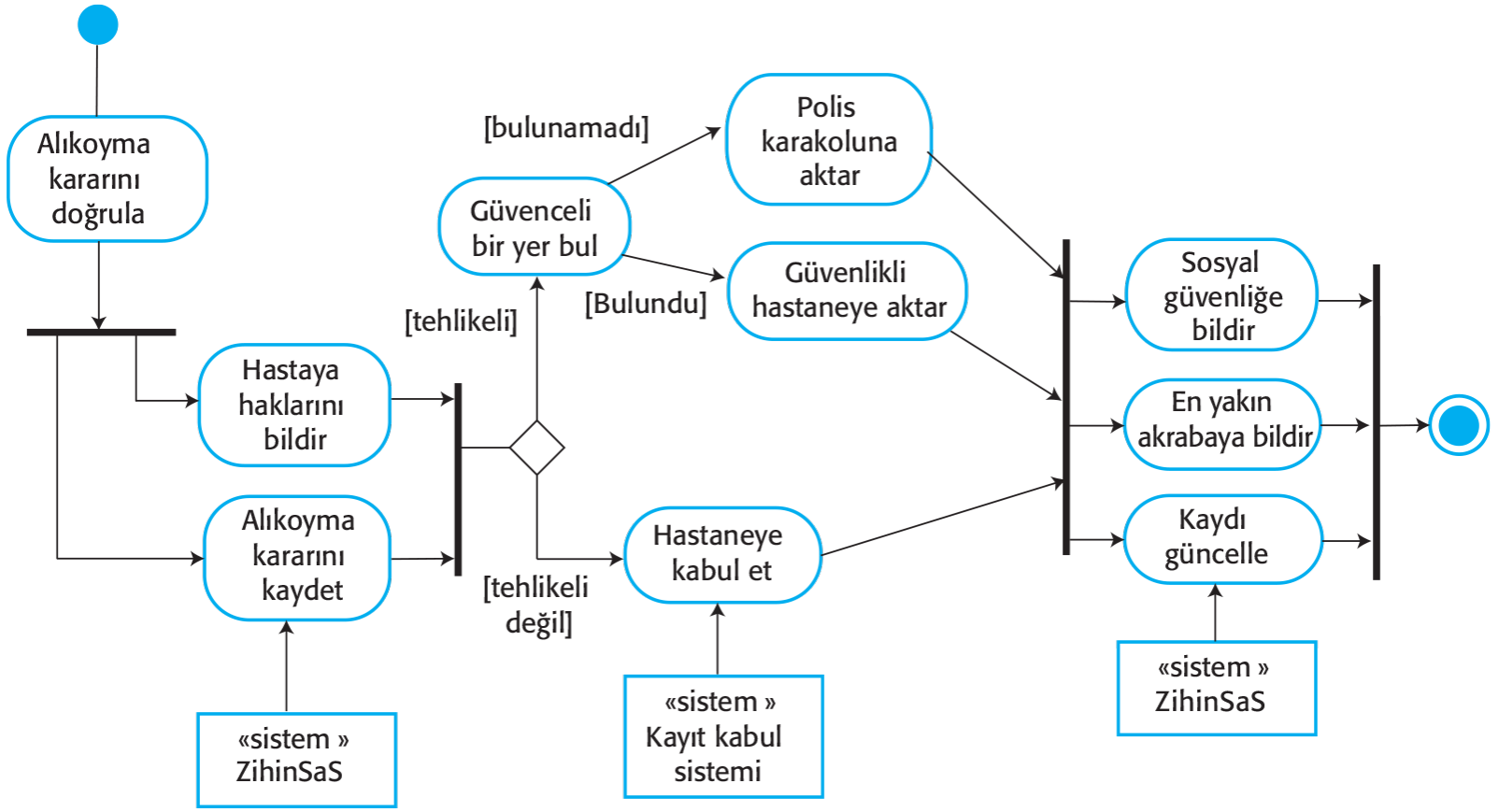
BAĞLAM MODELLERİ

Sistem tanımlamanın erken bir safhasında sistemin sınırlarına karar vermeniz gerekir ki bu neyin geliştirilmekte olan sistemin içinde kaldığına, neyin ise dışarıda bırakılacağına karar vermektir. Bu iş, paydaşlarla birlikte sistemin hangi işlevleri içereceğine karar vermek ve sistemin operasyonel çevresindeki işleme ve çalışma biçimlerini belirlemek ile ilgilidir.

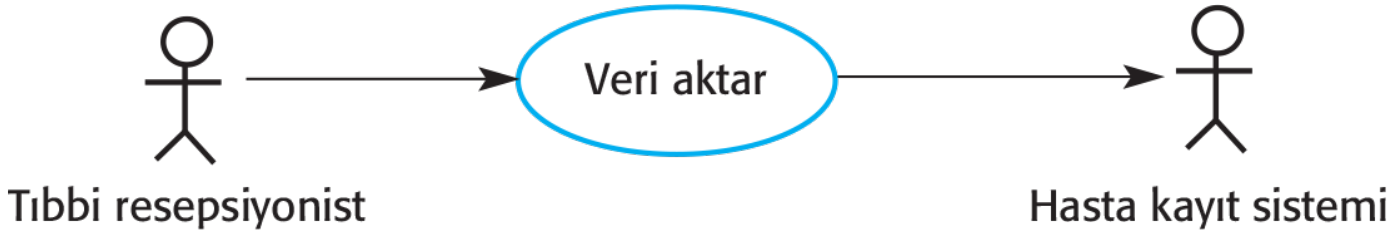
Bazı iş süreçleri için otomasyon desteği vermek isterken bazı başka süreçleri elle yapmak veya başka sistemlerle desteklemek isteyebilirsiniz. Sistemin işlevlerinin bazı var olan sistemlerle çakışmasını kontrol etmek ve yeni işlevlerin nerede gerçekleştirilmesi gerektiğine karar vermelisiniz. Maliyeti ve sistem gereksinimleri ile tasarımı anlamak için gereken zamanı azaltmak için bu kararlar sürecin başlarında alınmalıdır.



Şekil 5.1 ZihinSaS sisteminin bağlamı



Şekil 5.2 İstemsiz alıkoymanın süreç modeli



Şekil 5.3 Veri aktar kullanım durumu

ETKİLEŞİM MODELLERİ

Tüm sistemler bir tür etkileşim gerektirir. Etkileşim, kullanıcı girdi ve çıktılarını içeren kullanıcı etkileşimi olmanın yanı sıra geliştirilmekte olan sistem ile ortamdaki diğer sistemler arasında ve nihayet sistemin bileşenleri arasında yer alan etkileşimdir.

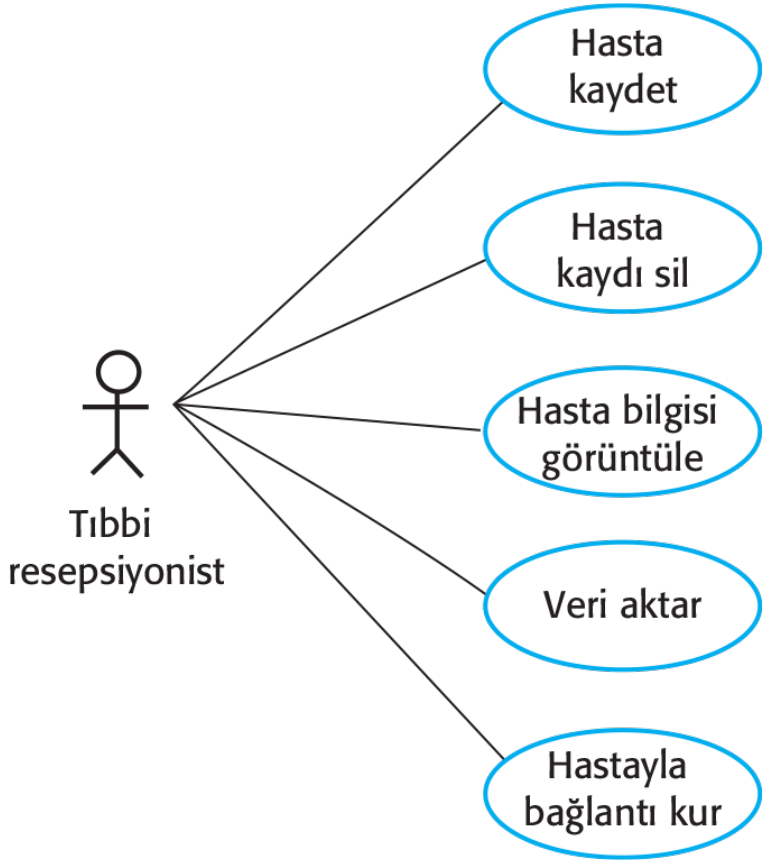
Kullanıcı etkileşimi modelleme kullanıcı gereksinimlerini belirlemeyi sağlaması nedeniyle önemlidir. Sistemler arası etkileşimi modellemek iletişim problemlerini göz önüne getirir. Bu kısımda etkileşim modellemeye yönelik birbiriyle ilintili iki yaklaşım tartışılacaktır.

1. Kullanım durumu modelleme: Genellikle sistem ile dış faktörler arasındaki (insan kullanıcı veya başka bir sistem) etkileşimin modellenmesidir.

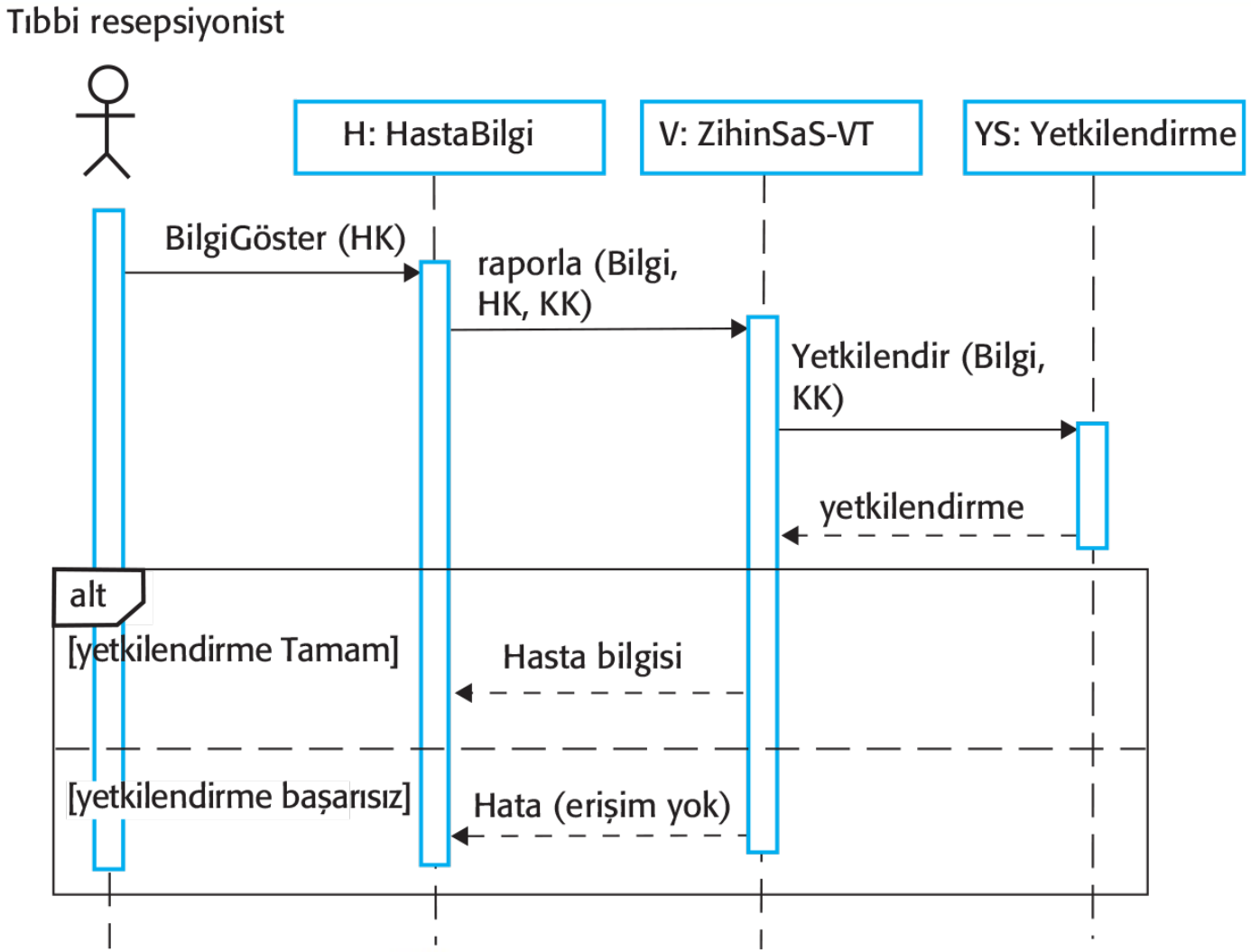
2. Sıra diyagramları: Sistem bileşenleri arasındaki (dış faktörler dâhil) etkileşimin modellenmesidir.

ZihinSaS sistemi: Veri aktar	
Aktörler	Tıbbi resepsiyonist, Hasta kayıt sistemi (HKS)
Betimleme	Bir resepsiyonist ZihinSaS sisteminden bir sağlık otoritesince yürütülen bir hasta kayıt sistemine veri aktarabilir. Aktarılan bilgi güncellenmiş kişisel bilgiler olabilir (adres,telefon numarası vs.) veya hastaya ilişkin teşhis ve tedavinin bir özeti olabilir.
Veri	Hastanın kişisel bilgileri, tedavi özeti
Uyaran	Tıbbi resepsiyonist tarafından verilen kullanıcı komutu
Tepki	HKS'nin güncellendiğinin teyidi
Yorumlar	Resepsiyonist hasta bilgisine ve HKS'ye erişmek için gerekli güvenlik izinlerine sahip olmalıdır.

Şekil 5.4 Veri aktar kullanım durumunun tablo gösterimi



Şekil 5.5 "Tıbbi resepsiyonist" içeren kullanım durumları

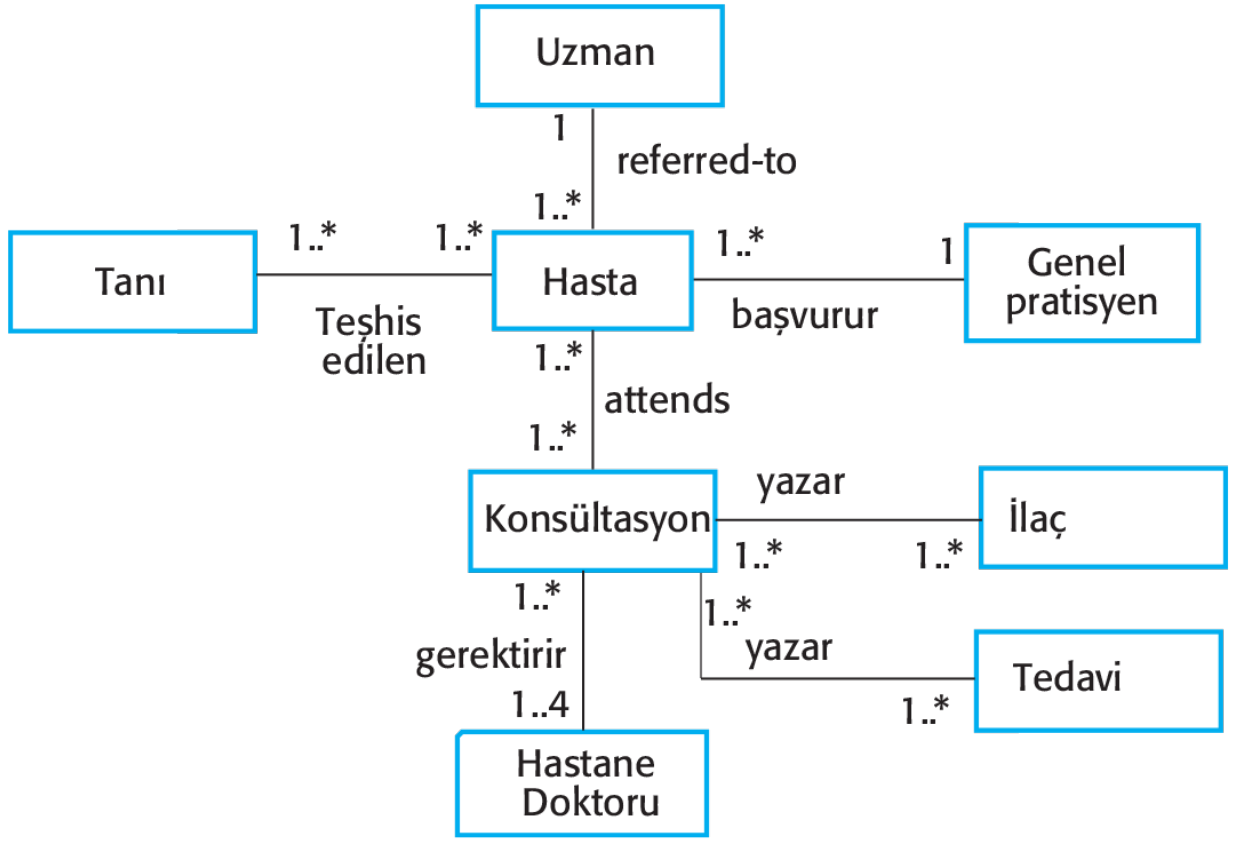


Şekil 5.6 "Hasta bilgisi görüntüle" için sıra diyagramı

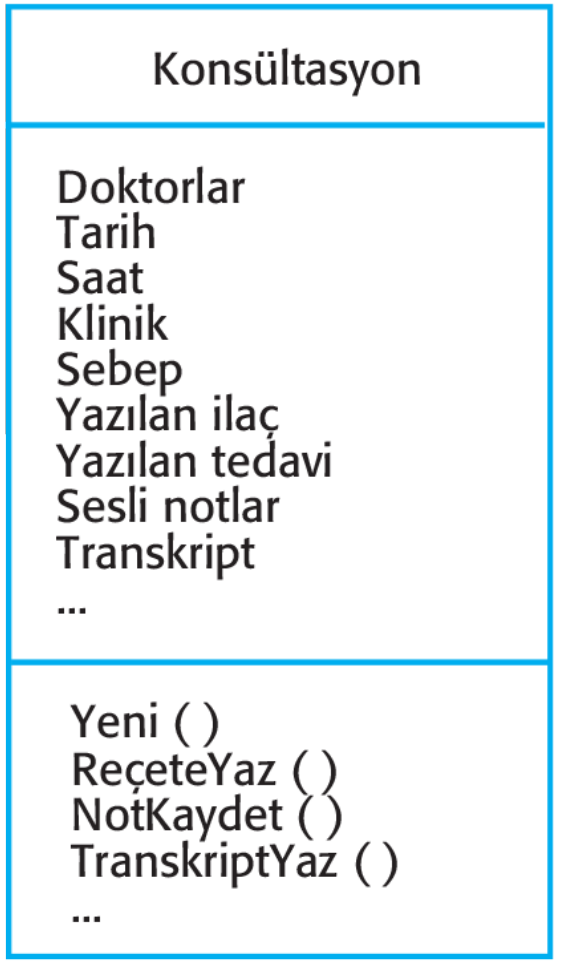
YAPISAL MODELLER

- Yapısal modeller bir sistemin organizasyonunu o sistemi oluşturan bileşenler ve bileşenler arasındaki ilişkiler cinsinden gösterirler.
- Yapısal modeller sistem tasarımını gösteren statik modeller olabildikleri gibi sistemin çalışma esnasındaki organizasyonunu gösteren dinamik modeller de olabilir.
- Bu ikisi aynı şey değildir; bir sistemin etkileşimli iş parçacıkları biçimindeki dinamik organizasyonu sistem bileşenlerini içeren statik modelden çok farklı olabilir.
- Bir sistemin yapısal modellerini, sistem mimarisini tasarlarken ve tartışırken yaratırız. Bunlar tüm sistem mimarisinin modelleri olabilirler veya sistem içindeki nesneler ve bunların ilişkilerini içeren daha detaylı modeller olabilirler.

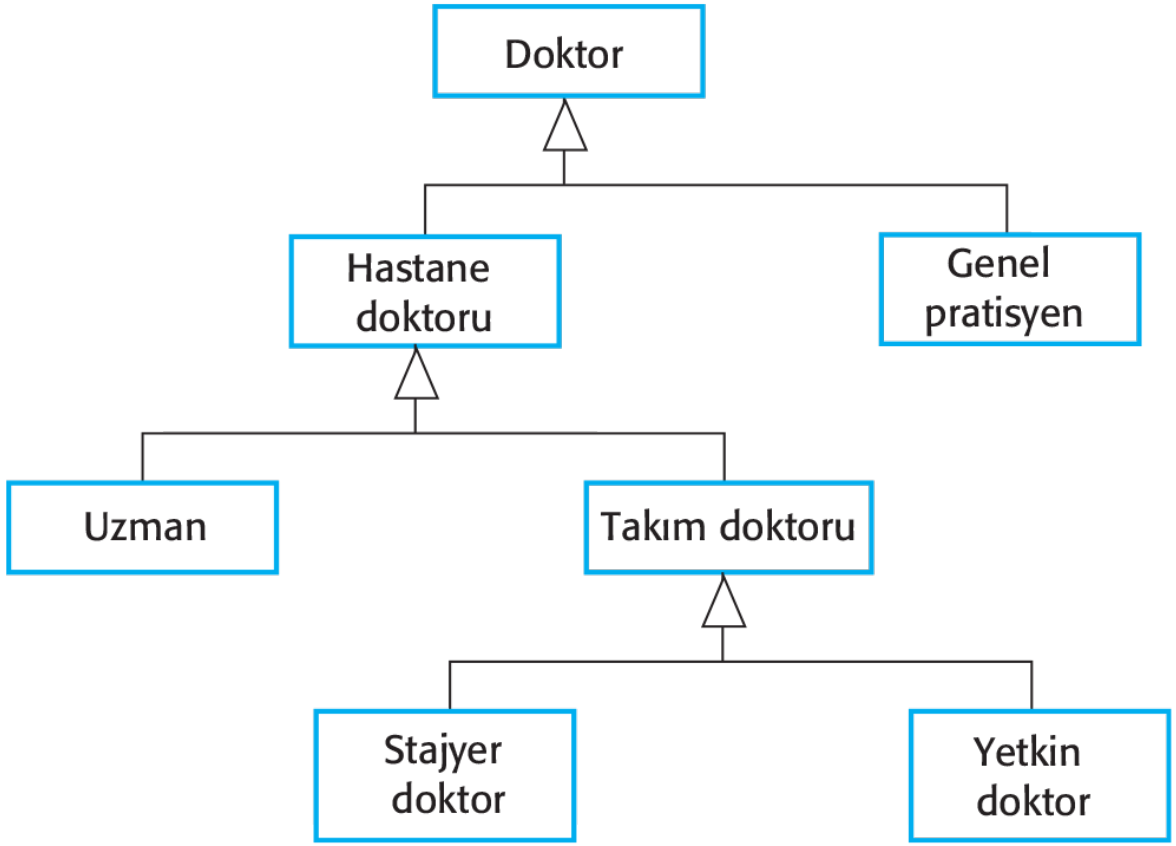
Şekil 5.8 UML Sınıfları ve bağlar



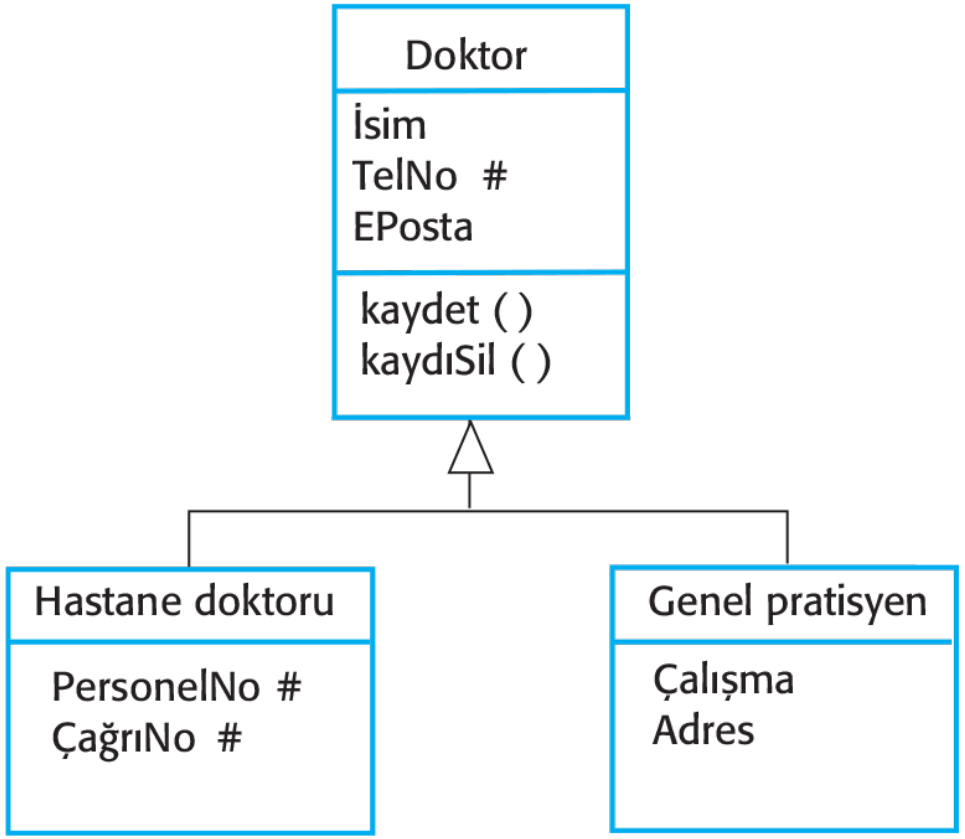
Şekil 5.9 ZihinSaS sistemindeki sınıflar ve bağlar



Şekil 5.10 Bir Konsültasyon sınıfı



Şekil 5.11 Bir genelleştirme hiyerarşisi



Şekil 5.12
Detaylandırılmış bir genelleştirme hiyerarşisi

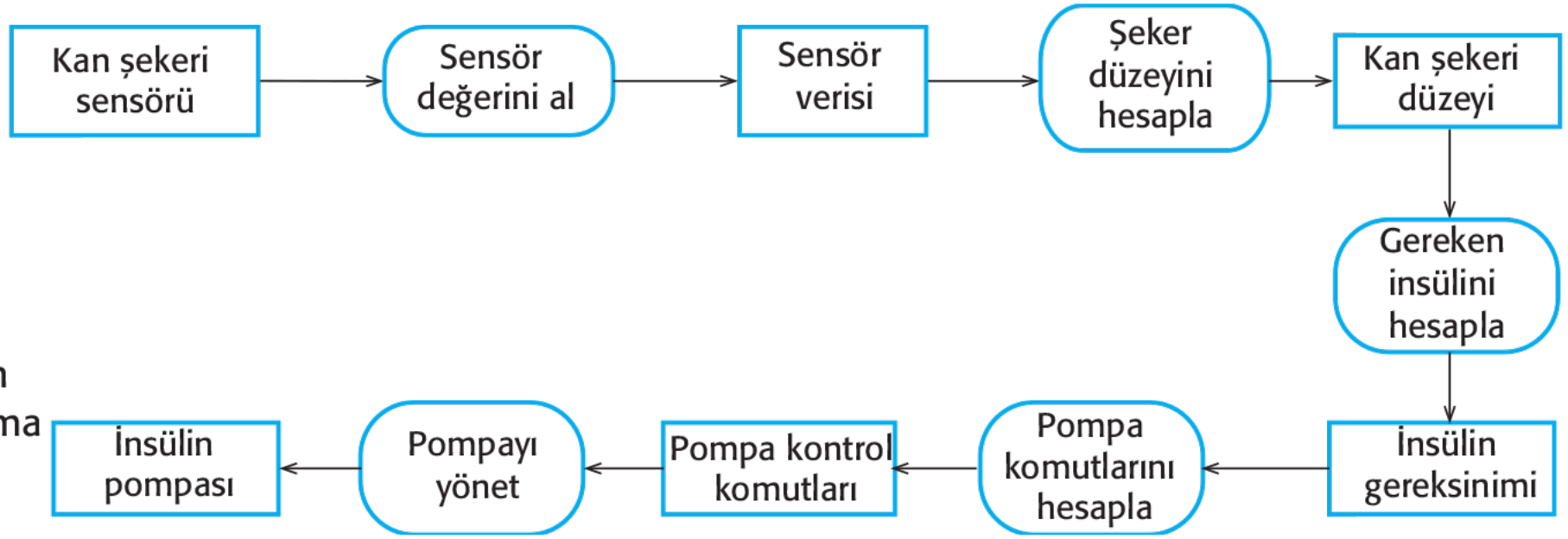
DAVRANIŞSAL MODELLER

Davranışsal modeller bir sistemin işleyişi sırasındaki dinamik davranışının modelleridir. Sistem, çevresinden gelen bir uyarana tepki verdiğinde ne olduğunu veya ne olması gerektiğini gösterirler. Bu uyaranlar veri veya olaylar olabilir:

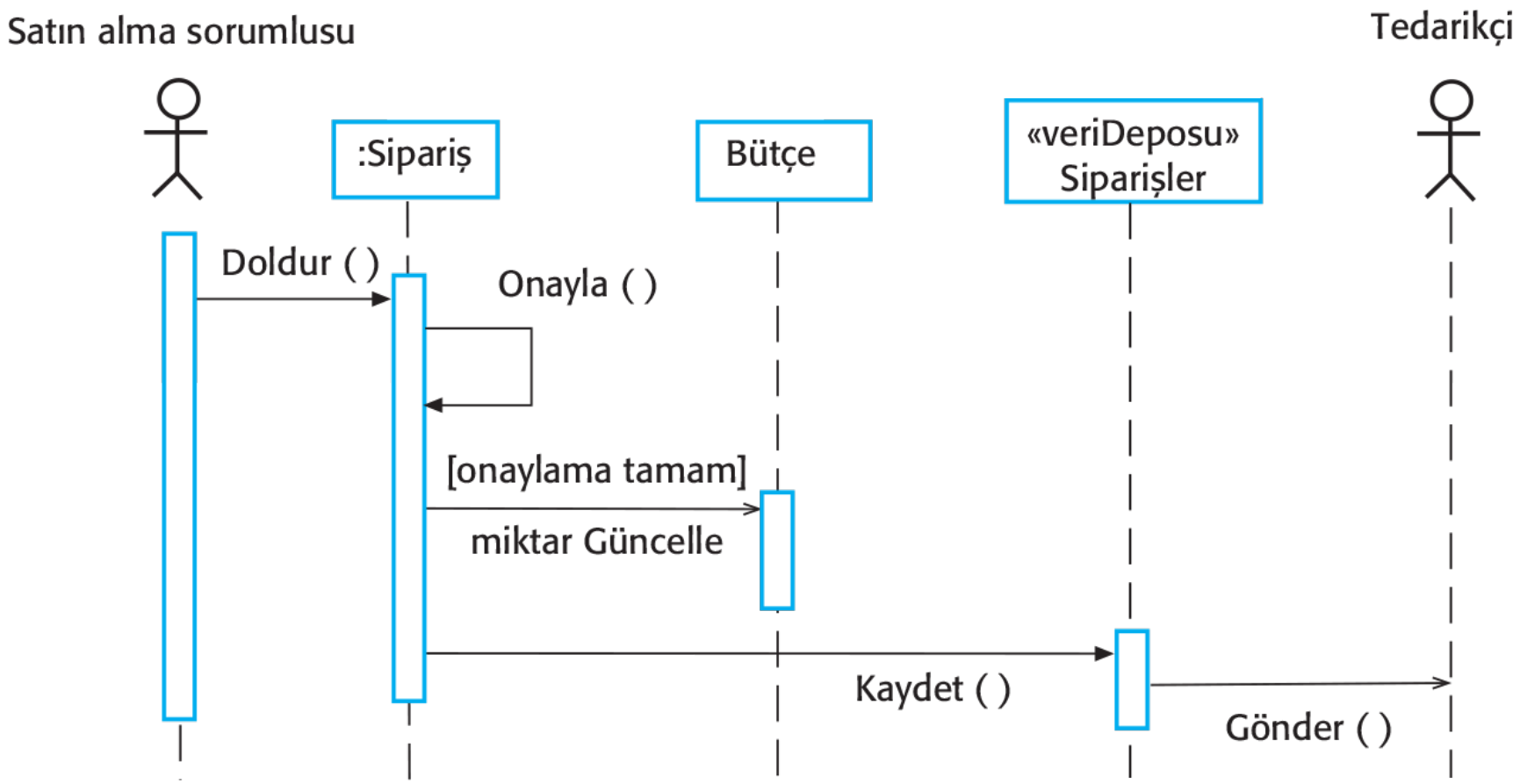
1. Sistem tarafından işlenecek olan veri erişilebilir hale gelir. Verinin mevcudiyeti işlemi tetikler.

2. Sistemin işleyişini tetikleyecek bir olay olur. Olaylar kendilerine bağlı verilere de sahip olabilirler fakat buna her zaman rastlanmaz.

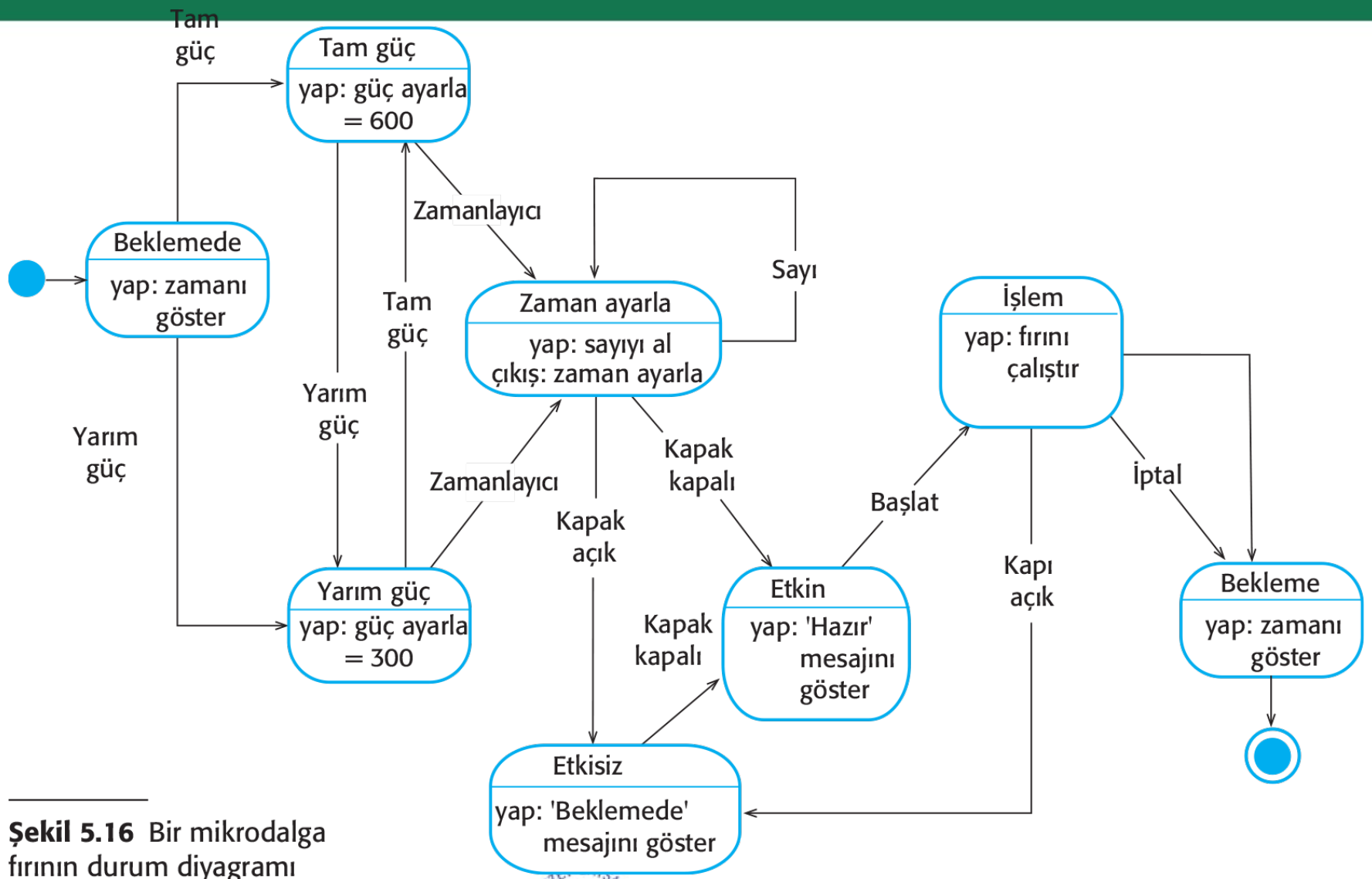
Pek çok iş sistemi veri tarafından güdülen veri-işleme sistemleridir. Göreceli az sayıda dış olay işleme gereksinimi ile birlikte genellikle sisteme girilen veri tarafından kontrol edilirler. İşlevleri veri üzerinde bir dizi etkinlik sonrasında çıktı üretmeyi içerir.



Şekil 5.14 İnsülin pompasının çalışma biçimini gösteren etkinlik modeli



Şekil 5.15 Sipariş işleme



Şekil 5.16 Bir mikrodalga fırının durum diyagramı

Durum	Açıklama
Bekleme	Fırın, girdi beklemekte. Gösterge paneli saati gösteriyor.
Yarım güç	Fırın 300 watt'a ayarlı. Gösterge paneli "Yarım güç" gösteriyor.
Tam güç	Fırın 600 watt'a ayarlı. Gösterge paneli "Tam güç" gösteriyor.
Zaman ayarı	Pişirme zamanı kullanıcının girdiği değere ayarlanır. Gösterge paneli seçilen pişirme zamanını gösterir ve ayarlama sırasında güncellenir.
Engelli	Fırının çalışması güvenlik için engellendi. İç fırın ışığı açık. Gösterge panelinde "Hazır değil" yazıyor.
Etkin	Fırın etkin halde. İç fırın ışığı kapalı. Gösterge panelinde "Pişirmeye hazır" yazıyor.
Pişirme	Fırın pişirme konumunda. İç fırın ışığı açık. Gösterge paneli zamanlayıcının geri sayımını gösteriyor. Pişirme bittiğinde 5 saniye boyunca zil çalar. Fırın ışığı yanar. Zil çalarken gösterge panelinde "Pişirme tamamlandı" yazar.

Uyaran	Açıklama
Yarım güç	Kullanıcı yarım güç düğmesine bastı
Tam güç	Kullanıcı tam güç düğmesine bastı
Zamanlayıcı	Kullanıcı zamanlayıcı düğmelerinden birine bastı
Sayı	Kullanıcı sayısal bir tuşa bastı
Kapak açık	Fırın kapağı düğmesi kapalı değil
Kapak kapalı	Fırın kapağı düğmesi kapalı
Başlat	Kullanıcı Başlat düğmesine bastı
İptal	Kullanıcı İptal düğmesine bastı.

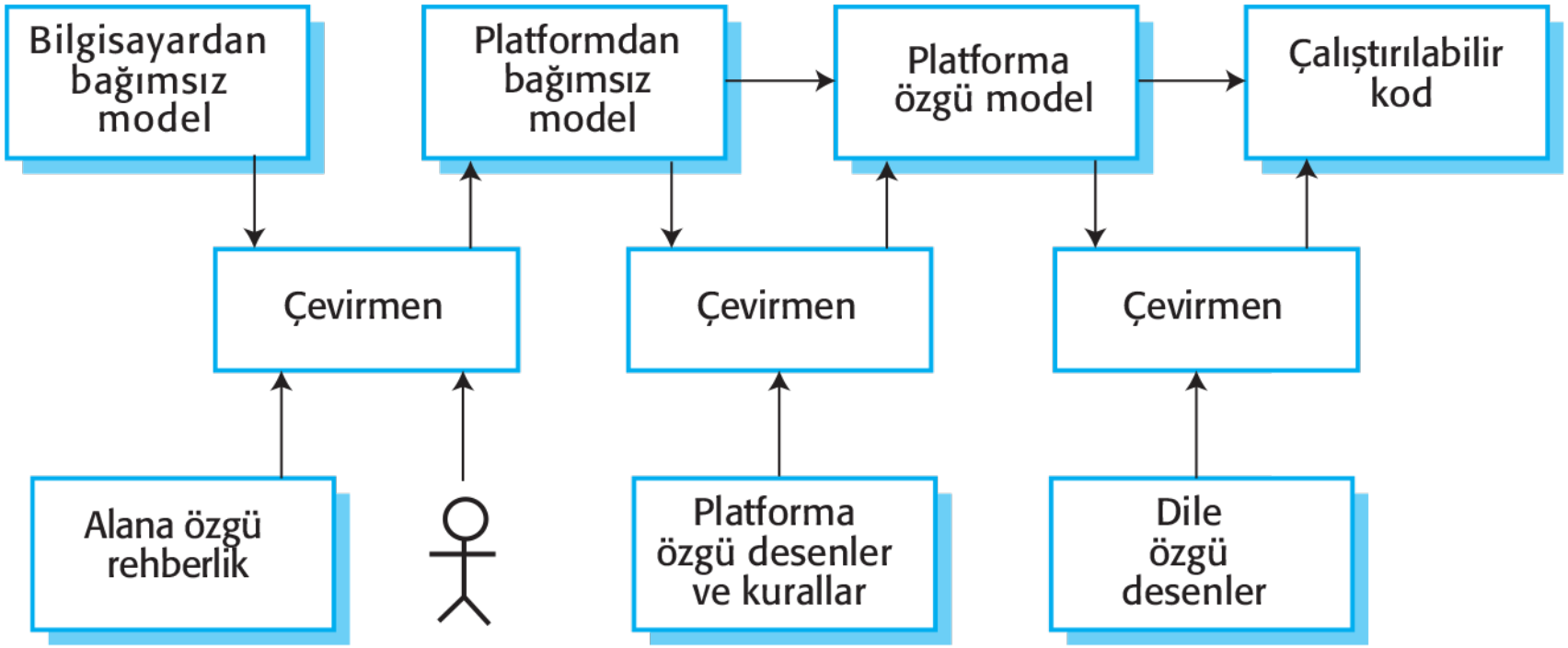
Şekil 5.18 Mikrodalga fırın için durumlar ve uyaranlar.

MODEL GÜDÜMLÜ MİMARİ

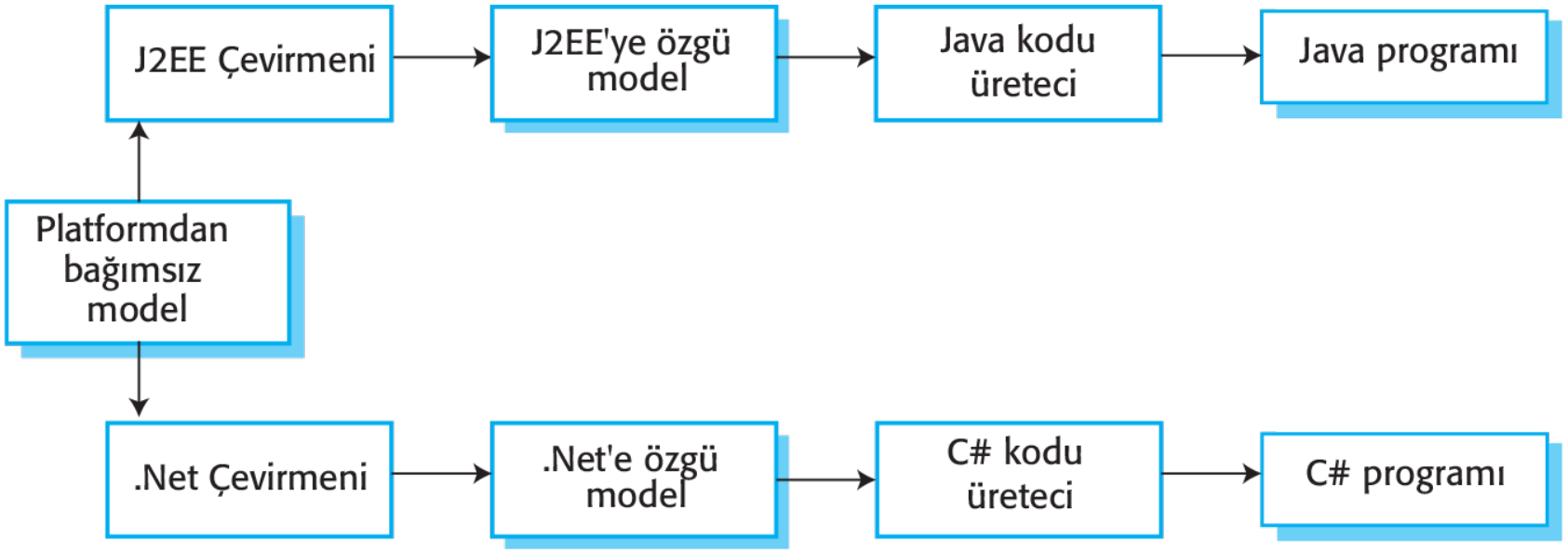
Model güdümlü mimari, UML modellerinin bir alt kümesini kullanarak bir sistemi betimleyen, model odaklı bir yazılım tasarım ve gerçekleştirim yaklaşımıdır. Burada farklı soyutlama derecelerindeki modeller yaratılır. Prensip olarak yüksek düzeyli platformdan bağımsız bir modelden insan eli değmeden çalışan bir program oluşturmak mümkündür.

MDA metodu üç çeşit soyut sistem modelinin üretilmesini tavsiye eder:

1. Bir bilgisayardan bağımsız model (BBM)
2. Bir platformdan bağımsız model (PBM)
3. Bir Platforma özgü model (PÖM)



Şekil 5.19 MDA dönüşümleri



Şekil 5.20 Çoğul platforma özgü modeller

Executable UML

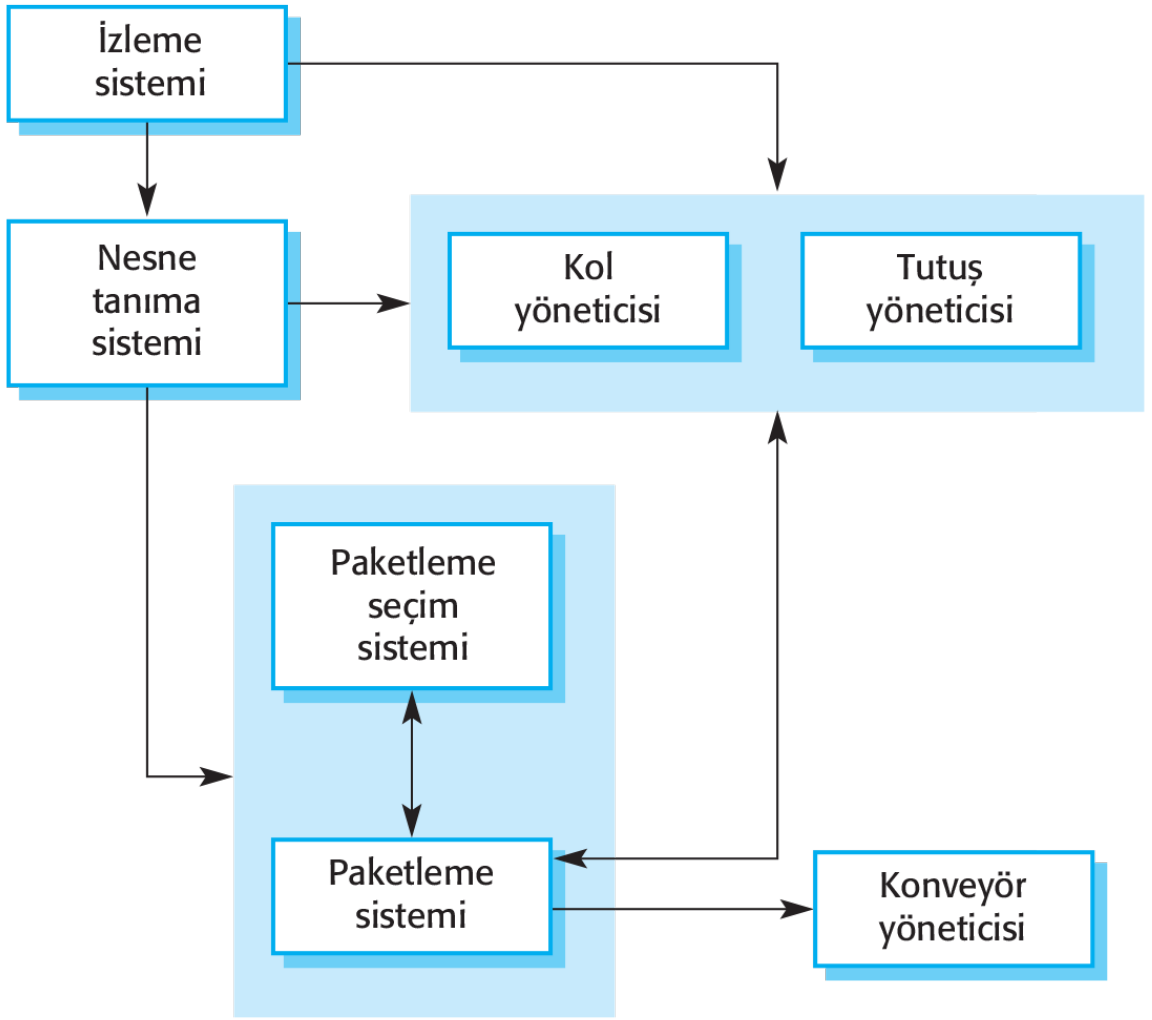
Model güdümlü mühendisliğin arkasındaki asıl fikir modellerden koda tam otomatik dönüşümün mümkün olabilmesidir. Buna ulaşabilmek için şekilsel modelleri, anlamları açık biçimde tanımlanmış ve çalıştırılabilir koda derlenebilecek biçimde kurabilmek gereklidir. Ayrıca şekilsel modellere bu modellerde tanımlanan işlemlerin nasıl gerçekleştirildiği hakkında bilgiler eklemek için bir yönteme ihtiyaç vardır. Bu, UML 2'nin çalıştırılabilir UML veya xUML adı verilen bir alt kümesini kullanarak mümkündür (Mellor ve Balcer 2002).


İÇİNDEKİLER
BÖLÜM 6: MİMARİ TASARIM
MİMARİ TASARIM KARARLARI
MİMARİ GÖRÜNÜMLER
MİMARİ DESENLER
UYGULAMA MİMARİLERİ
- Hareket İşleme Sistemleri
- Bilgi Sistemleri
- Dil İşleme Sistemleri

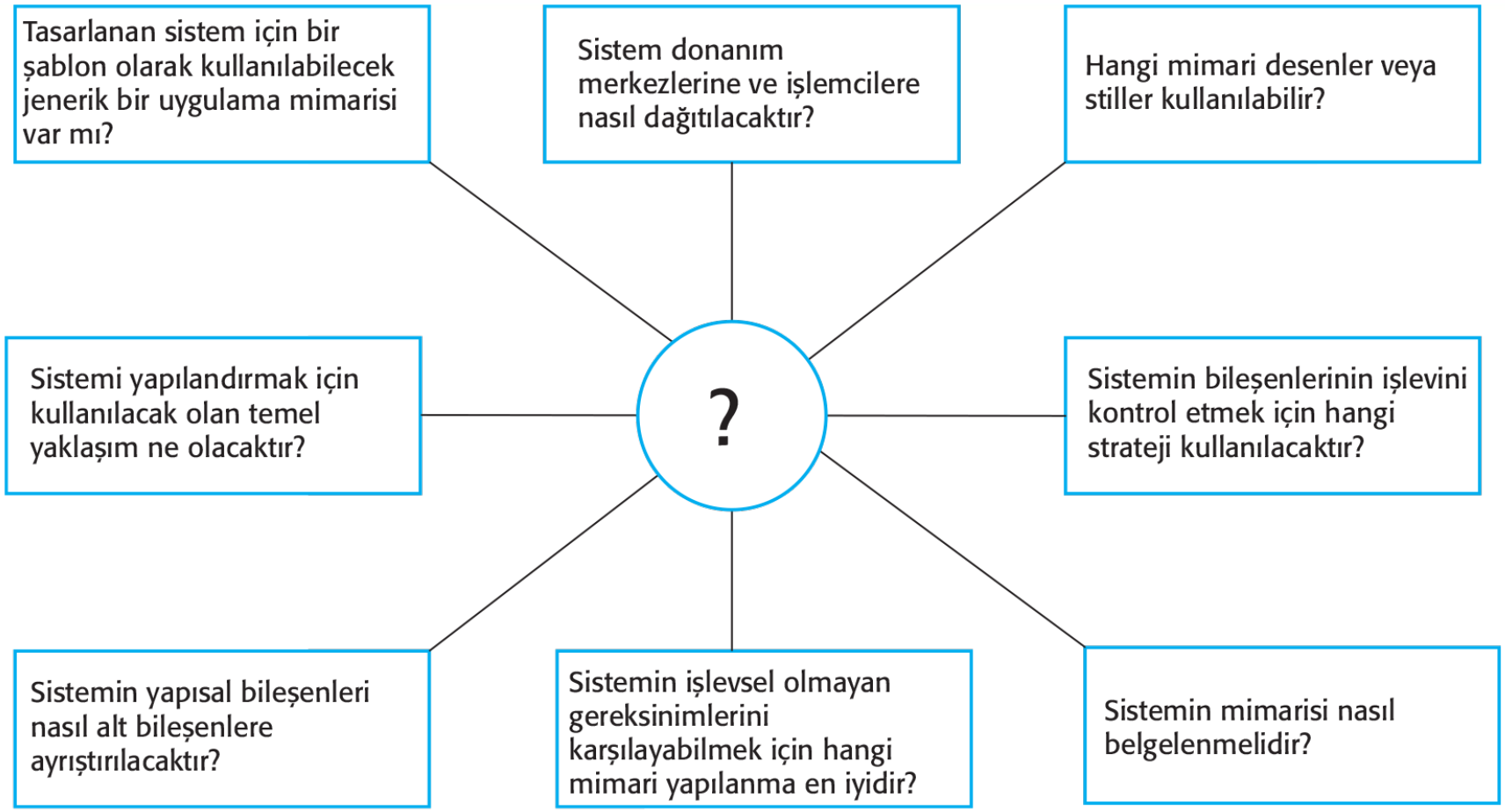
6.BÖLÜM
MİMARİ TASARIM

- Mimari tasarım bir yazılım sisteminin nasıl yapılandırılması gerektiğini anlamak ve sistemin genel yapısını tasarlamaktır. Bölüm 2'de açıkladığım yazılım geliştirme sürecinde yazılım tasarımı aşamasının ilk basamağı mimari tasarımdır.

- Bu basamak tasarım ile gereksinim mühendisliği arasındaki kritik bağlantıdır, çünkü sistemin ana yapısal bileşenlerini ve bu bileşenler arasındaki ilişkileri belirlemektedir. Mimari tasarım sürecinin çıktısı bir sistemin haberleşen bileşenler olarak nasıl yapılandığını anlatan bir mimari modeldir.



Şekil 6.1 Bir paketleme robotu yönetim sisteminin mimarisi



Şekil 6.2 Mimari tasarım kararları

MİMARİ TASARIM KARARLARI

- Mimari tasarım bir sistemin fonksiyonel olan ve olmayan gereksinimlerini karşılayan bir sistem organizasyonunu tasarladığınız yaratıcı bir süreçtir. Kalıplaşmış bir mimari tasarım süreci yoktur.

- Süreç, geliştirilen sisteme, sistem mimarının geçmişine, tecrübesine ve sistemin kendine özgü gereksinimlerine bağlı olarak şekillenmektedir.

- Sonuç olarak kanımca mimari tasarımı bir etkinlikler dizisi olarak görmek yerine alınması gereken bir kararlar serisi olarak görmek en doğrusudur.

- Mimari tasarım sürecinde sistem mimarları sistemi ve geliştirme sürecini ciddi olarak etkileyen birçok yapısal karar almak zorundadırlar. Bilgilerine ve tecrübelerine dayanarak Şekil 6.2'de gösterilen sorulara cevap aramak durumunda kalırlar.

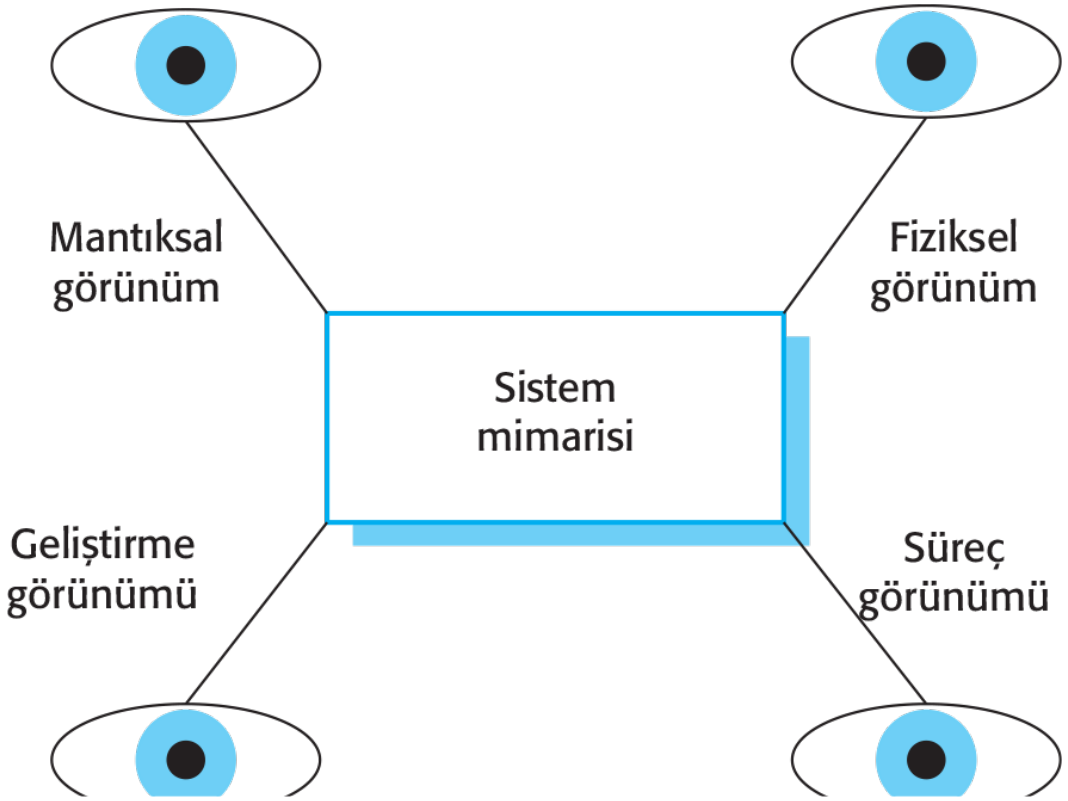
MİMARİ GÖRÜNÜMLER

Bu bölümün girişinde bir yazılım sisteminin mimari modelinin, yazılım gereksinimleri veya tasarımı konulu bir tartışmayı doğru noktaya odaklamak için kullanılabileceğini açıklamıştım. Buna alternatif olarak bir tasarımı sistemin daha detaylı bir tasarımına ve gerçekleştirimine taban oluşturabilecek biçimde belgelemek amacıyla da kullanılabilir. Bu bölümde her iki kullanım ile de ilgili iki konuya değineceğim:

1. Bir sistemin mimarisini tasarlarken ve belgelerken hangi görünümler veya perspektifler kullanışlıdır?

2. Mimari modelleri ifade etmek için hangi gösterim biçimleri kullanılmalıdır?

Bir şekilsel model bir sistemin yalnızca bir tek görünümünü veya perspektifini gösterebilir; bu nedenle tek bir diyagramda bir sistemin mimarisine ilişkin bütün bilgilerin gösterilmesi imkânsızdır. Bir sistemin nasıl modüllere ayrıştığını, çalışma zamanı süreçlerin nasıl etkileştiklerini veya sistem bileşenlerinin bir ağ üzerinde farklı dağılma biçimlerini gösterebilir.



Şekil 6.3 Mimari görünümler

MİMARİ DESENLER

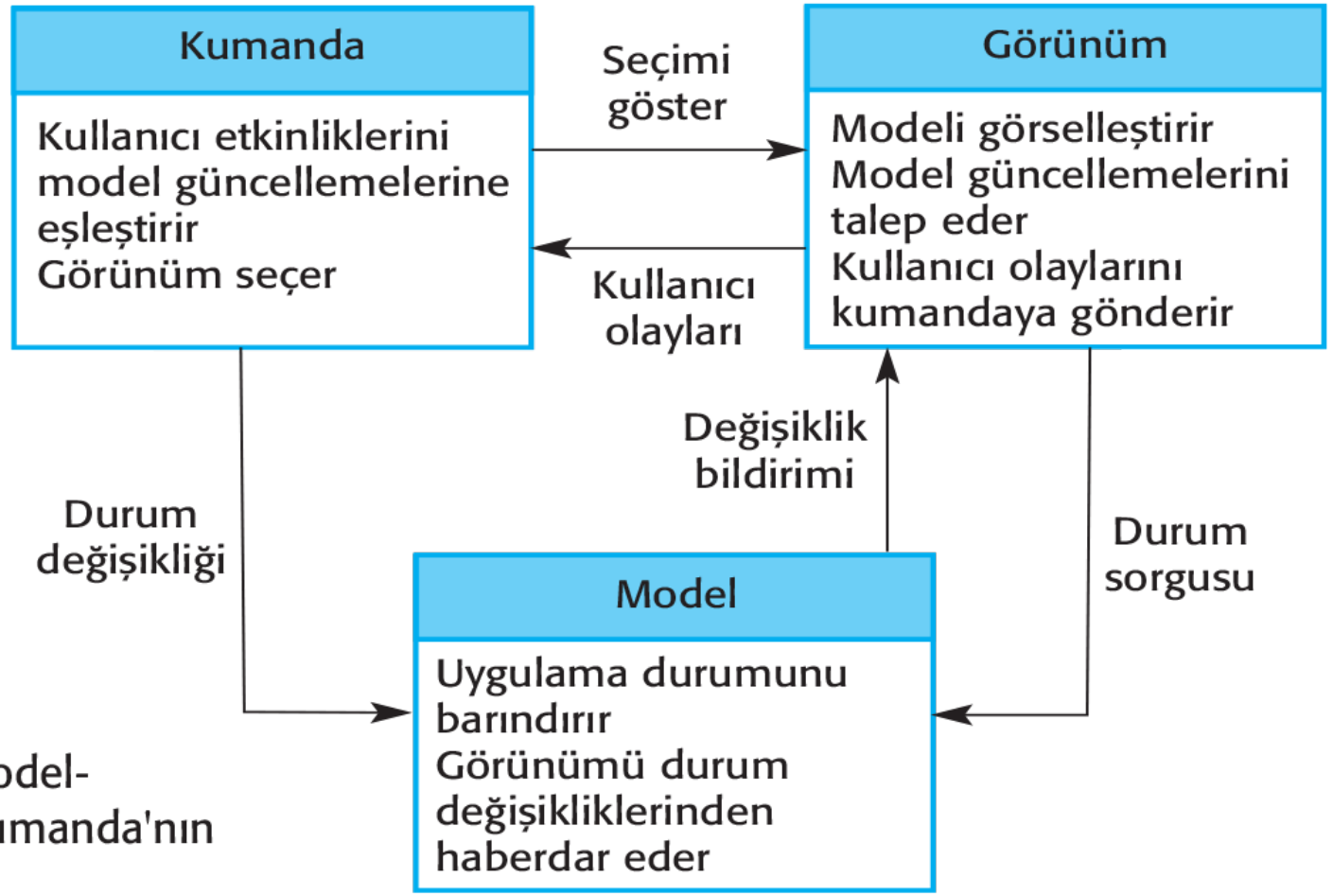
Desen fikri; yazılım sistemleri hakkındaki bilgiyi sunmanın, paylaşmanın ve yeniden kullanmanın bir yolu olarak, yazılım mühendisliğinin birçok alanında benimsenmiştir. Bu akımı tetikleyen nesneye yönelik tasarım desenleri konusunda yazılmış bir kitap.

Bu kitap aynı zamanda bir dizi farklı desen tipinin ortaya çıkması için işaret fişeği işlevi görmüştür. Bunlar arasında organizasyonel tasarım için olan desenler, kullanılabilirlik desenleri, iş birlikli etkileşim desenleri, ve yapılandırma yönetimi desenleri vardır.

Mimari desenler 1990'larda "mimari stiller" adı altında öne sürülmüştür. 1996 ile 2007 arasında desen tabanlı yazılım mimarisi konulu çok detaylı beş ciltlik bir el kitabı serisi yayımlanmıştır.

İsim	Model-Görünüm-Kumanda (MGK) [MVC (Model-View-Controller)]
Tanım	Sunum ve etkileşimi sistem verisinden ayırır. Sistem birlikte çalışan üç mantıksal bileşenden oluşur. Model bileşeni sistem verilerini ve o verilerle bağlantılı işlemleri yönetir. Görünüm bileşeni verinin kullanıcıya nasıl sunulacağını tanımlar ve yönetir. Kumanda bileşeni kullanıcı etkileşimini yönetir (ör. tuşlara basmak veya fare tıklamaları vs.) ve etkileşimleri Görünüm ve Model'e aktarır. Bakınız Şekil 6.5.
Örnek	Şekil 6.6 MGK kullanılarak yapılandırılmış bir Web-tabanlı uygulamanın mimarisini göstermektedir.
Kullanım alanı	Veriyi kullanmak ve onunla etkileşmek için birden fazla yol olunca kullanılır. Etkileşim ve sunuma dair gelecekteki gereksinimler bilinemediğinde de kullanılır.
Avantajları	Verinin ve temsil biçiminin birbirinden bağımsız olarak değişebilmesine imkan sağlar. Aynı verinin farklı sunuş biçimlerini birlikte destekleyerek biri üzerinde yapılan bir değişikliğin aynı anda hepsinde gösterilmesini sağlar.
Dezavantajlar	Veri modeli ve etkileşimlerin basit olduğu durumda dahi ek kod ve kod karmaşıklığına neden olabilir.

Şekil 6.4 Model-Görünüm-Kumanda (MGK) deseni



Şekil 6.5 Model-Görünüm-Kumanda'nın yapılanması

İsim	Katmanlı mimari
Tanım	Her katmanda ilgili fonksiyonellik olmak üzere sistemi katmanlar halinde düzenler. Her bir katman, üstündeki katmana servisler sunar böylece en alt katmanlarda tüm sistem çapında kullanılma ihtimali olan çekirdek servisler temsil edilir. Bkz. Şekil 6.8
Örnek	Tüm konuların okullarda öğrenilmesini destekleyebilecek bir sayısal öğrenme sisteminin katmanlı bir modeli.
Kullanım alanı	Var olan sistemlerin üzerine yeni imkanlar inşa edilirken: geliştirme her biri farklı bir fonksiyonellik katmanından sorumlu çok sayıda takım arasında dağıtılmışken; çok katmanlı güvenlik için bir gereksinim varken.
Avantajları	Arayüz korunduğu sürece katmanların bütün olarak değiştirilebilmesini sağlar. Sistemin güvenilirliğini arttırmak için her bir katmanda yedeklenmiş imkanlar (örn. kimlik doğrulama) sunulabilir.
Dezavantajları	Pratikte katmanlar arasında temiz bir ayrışma genellikle zordur, ve üst düzeyli bir katman bir alt katmanın aracılığı olmaksızın daha alt katmanlarla etkileşmek zorunda kalabilir. Ayrıca bir servis talebinin her bir katmanda tekrar yorumlanması performans problemine yol açabilir.

Şekil 6.7 Katmanlı Mimari deseni

Şekil 6.10 Ambar deseni

İsim	Ambar
Tanım	Sistemdeki bütün veriler, tüm sistem bileşenlerine açık merkezi bir ambarda tutulur. Bileşenler doğrudan etkileşime girmezler. Tüm etkileşim ambar üzerinden yapılır.
Örnek	Şekil 6.11 sistem tasarımı bilgisini barındıran bir ambar içeren bir EGO örneğidir. Her yazılım aracı diğer araçlar tarafından da erişilir olacak bilgiler üretir.
Kullanım alanı	Bu deseni uzun süre saklamanız gereken büyük miktarda bilgi olduğu zaman kullanmalısınız. Verinin ambara eklenmesinin herhangi bir işlevi veya aracı tetiklediği veri güdümlü sistemlerde de kullanabilirsiniz.
Avantajları	Bileşenler bağımsız olabilirler; diğer bileşenlerin varlığımdan dahi haberdar olmayabilirler. Bir bileşen tarafından yapılan değişiklikler diğer bileşenlere yansıtılabilir. Tüm veri sanki aynı yerdeymişcesine tutarlı bir biçimde yönetilir. (Örneğin, yedeklemeler aynı anda yapılır.)
Dezavantajları	Ambardaki tek bir hata tüm bir sistemi etkiler. Her iletişimin ambar üzerinden yapılması verimsiz olabilir. Ambarı çok sayıda bilgisayar üzerinde dağıtmak zor olabilir.

İsim	İstemci-sunucu
Tanım	İstemci-sunucu mimarisinde sistem bir servisler kümesidir ve her bir servis ayrı bir sunucu tarafından sağlanır. İstemciler bu servislerin kullanıcılarıdır ve servislerden yararlanabilmek için sunuculara ulaşırlar.
Örnek	Şekil 6.13 istemci-sunucu sistemi olarak yapılandırılmış bir film ve video/DVD kütüphanesi içermektedir.
Kullanım alanı	Paylaşılan veri tabanında yer alan veriye çeşitli noktalardan erişilmesi gerektiğinde kullanılır. Sunucular kopyalanarak çoğaltılabildiğinden sistem üzerindeki yükün değişken olduğu zamanlarda da kullanılabilir.
Avantajları	Bu modelin temel faydası sunucuların bir ağ üzerinde dağıtılabilmesini sağlamasıdır. Genel işlevler (ör. yazıcı servisi) tüm istemciler için erişilir olabilir ve tüm sunucular tarafından gerçekleştirilmesi gerekli değildir.
Dezavantajları	Her bir servis tekil bir kritik noktadır ve bu nedenle servis engelleme saldırıları ve sunucu hatalarına karşı hassastır. Sadece sistemin kendisine değil aynı zamanda ağa bağlı olan performansı kestirmek zor olabilir. Sunucular farklı kurumlara ait olduklarında yönetimleri problemli olabilir.

Şekil 6.12 İstemci-Sunucu deseni

İsim	Boru ve süzgeç
Tanım	Sistem içindeki veri işleme düzeni öyle düzenlenmiştir ki her bir işlemci bileşeni (süzgeç) ayrıktır ve tek bir veri dönüşümü gerçekleştirir. Veri işlenmesi için bir bileşenden diğerine (boru içindeymiş gibi) akar.
Örnek	Şekil 6.15 faturaları işlemek için kullanılan bir boru ve süzgeç sistemidir.
Kullanım alanı	Yaygın olarak girdilerin, karşılık gelen çıktıları üretmek için ayrık aşamalarda işlendiği (hem yığın hem hareket tabanlı) veri işleme sistemlerinde kullanılır.
Avantajları	Anlaşılması kolaydır ve dönüşümlerin yeniden kullanımını destekler. İş akışı stili birçok iş sürecinin yapısına uygundur. Bir dönüşüm ekleyerek evrimleşmesi dolaysızdır. Sıralı veya eş zamanlı çalışan bir sistem olarak gerçekleştirilebilir.
Dezavantajları	Haberleşen dönüşümler arasında veri transferi karara bağlanmalıdır. Her bir dönüşüm üzerinde anlaşılmış biçime girdisini çözümlemek çıktısını ise sentezlemek zorundadır. Bu sistemin maliyetini arttırır ve bu uyumsuz veri yapılarını kullanan mimari bileşenleri kullanmanın imkansız olduğu anlamına gelebilir.

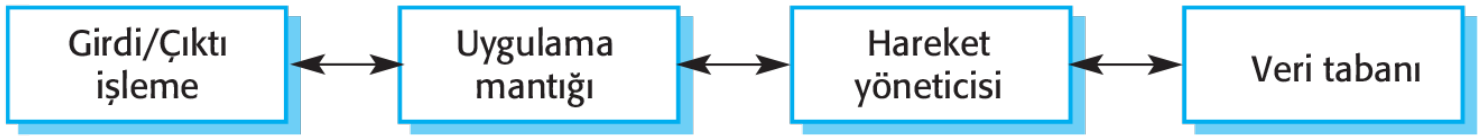
Şekil 6.14 Boru ve Süzgeç deseni

UYGULAMA MİMARİLERİ

Uygulama sistemleri iş dünyasına ilişkin veya kurumsal bir ihtiyacı karşılamayı amaçlarlar. Tüm işlerin ortak bazı yönleri vardır; insanları işe alırlar, fatura keserler, hesap tutarlar vs. Aynı sektörde faaliyet gösteren meslekler ortak sektöre özel uygulamaları kullanırlar.

Dolayısıyla tüm telefon şirketleri genel iş fonksiyonlarının yanı sıra çağrıları bağlayan ve ölçen, ağı yöneten ve abonelerine fatura çıkaran sistemlere ihtiyaç duyarlar. Sonuç olarak bu işlerde kullanılan uygulama sistemlerinin de birçok ortak yanı vardır.

Bu ortaklıklar belirli tipteki yazılım sisteminin yapısını ve organizasyonunu betimleyen yazılım mimarilerinin geliştirilmesine öncülük etmiştir. Uygulama mimarileri belirli bir sistem sınıfının esas özelliklerini paketler.

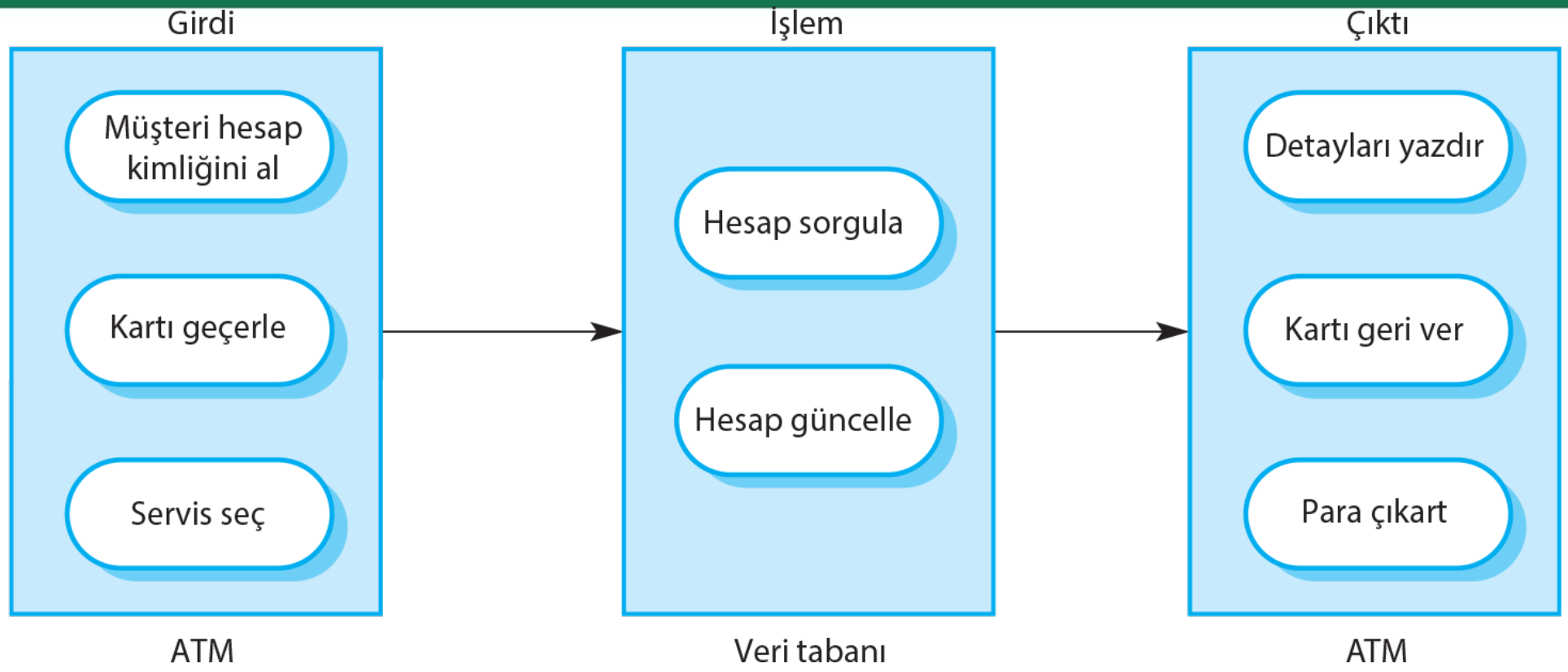


Şekil 6.16
Hareket işleme uygulamalarının yapısı

Hareket İşleme Sistemleri

Hareket işleme sistemleri bir veri tabanından bilgi taleplerini veya veri tabanı üzerinde güncelleme taleplerini işlemek üzere tasarlanmış sistemlerdir. Kullanıcı perspektifinden bir hareketi, belirli bir amacı, örneğin “Londra’dan Paris’e uçuşların zamanlarını bulmak” gibi bir amacı sağlayan tutarlı bir işlemler dizisidir.

Kullanıcı hareketi veri tabanının değiştirilmesini gerektirmedikçe bunu teknik olarak bir veri tabanı hareketi olarak paketlemek gerekmeyebilir. Bir kullanıcı hareketinin gerçek bir örneği bir müşterinin ATM aracılığıyla hesabından para çekme talebidir.

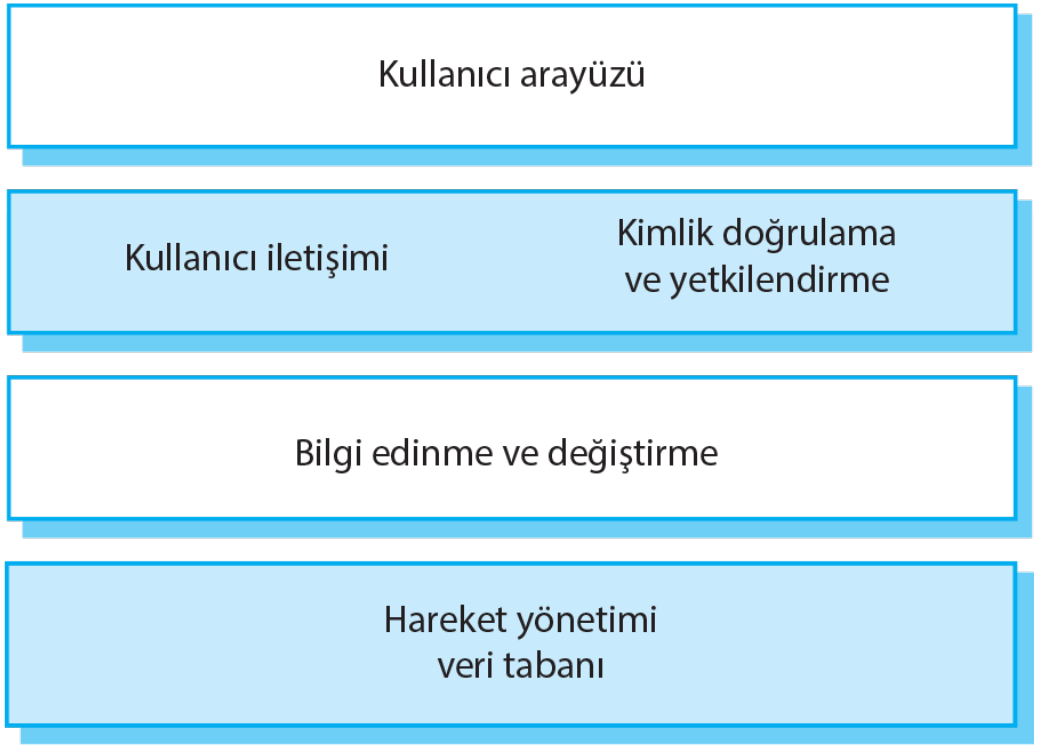


Şekil 6.17 Bir ATM sisteminin yazılım mimarisi

Bilgi Sistemleri

- Paylaşılan bir veri tabanı ile etkileşimi gerektiren bütün sistemler hareket tabanlı bilgi sistemleri olarak görülebilirler.
- Bir bilgi sistemi; bir kütüphane kataloğu, bir uçuş tarifesi veya bir hastanedeki hasta kayıtları gibi büyük bir bilgi tabanına kontrollü bir biçimde erişilmesine izin verir.
- Bilgi sistemleri hemen her zaman kullanıcı arayüzünün bir Web tarayıcısı olarak gerçekleştirildiği Web tabanlı sistemlerdir. Şekil 6.18 bir bilgi sisteminin çok genel bir modelidir.

Şekil 6.18 Katmanlı bilgi sistemi mimarisi

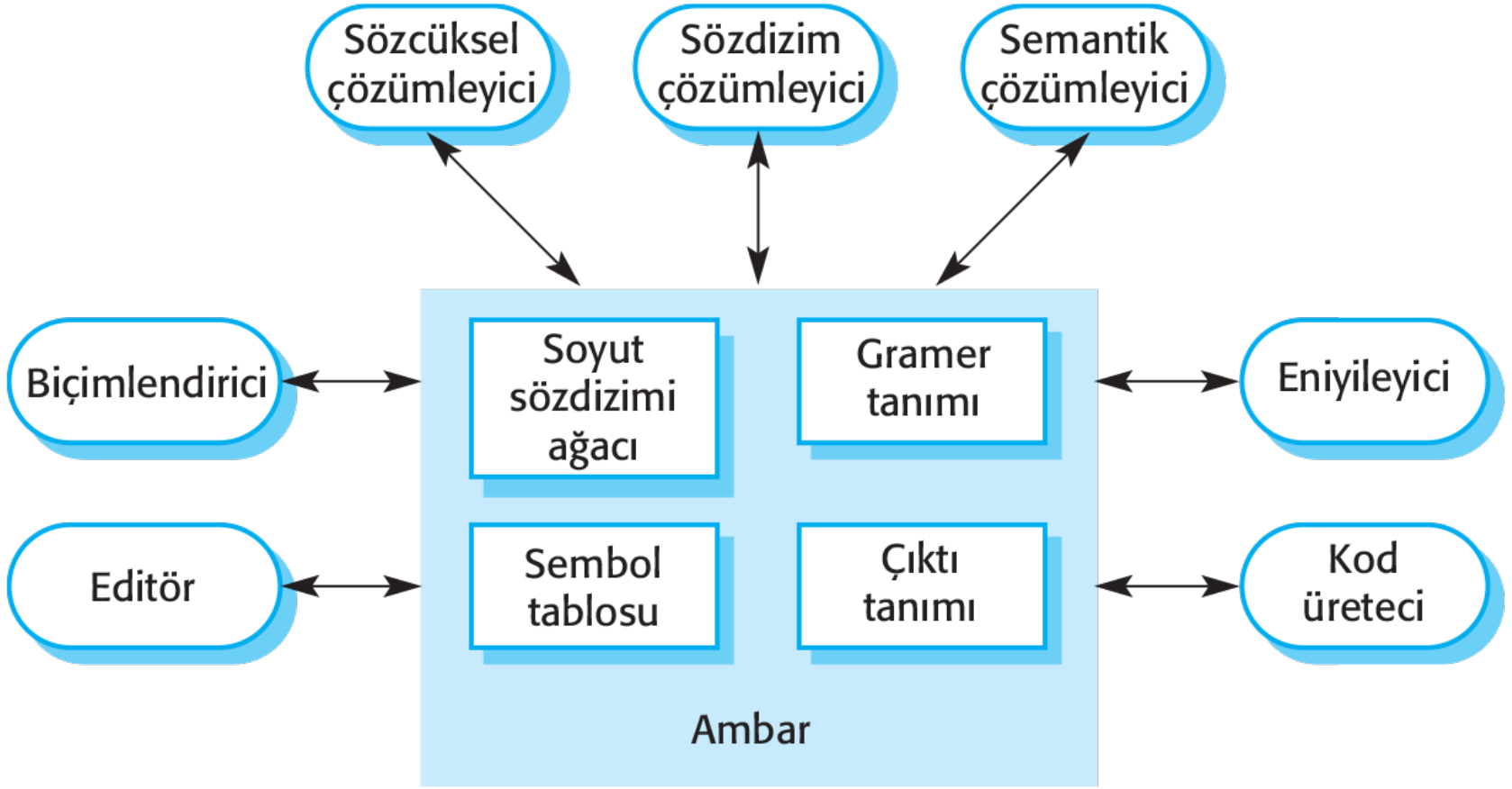


Dil İşleme Sistemleri

Dil işleme sistemleri bir dili alternatif bir gösterim biçimine çevirir ve programlama dilleri söz konusu ise aynı zamanda sonuçta oluşan kodu çalıştırabilir de. Derleyiciler bir programlama dilini makine koduna çevirirler.

Diğer dil işleme sistemleri XML veri gösterimini bir veri tabanını sorgulayan komutlara veya alternatif bir XML temsiline çevirebilir. Doğal dil işleme sistemleri bir doğal dili diğerine çevirebilir; örneğin Fransızcayı Norveççeye.

Şekil 6.20'de bir programlama dili için bir dil işleme sistemine ait mümkün olan bir mimari gösterilmiştir. Kaynak dil komutları çalıştırılacak olan programı tanımlar ve bir çevirmen bunları soyut bir makinenin komutlarına çevirir.



Şekil 6.21 Bir dil işleme sistemi için bir ambar mimarisi

ANALİZ

İÇERİK

Bu bölümde,

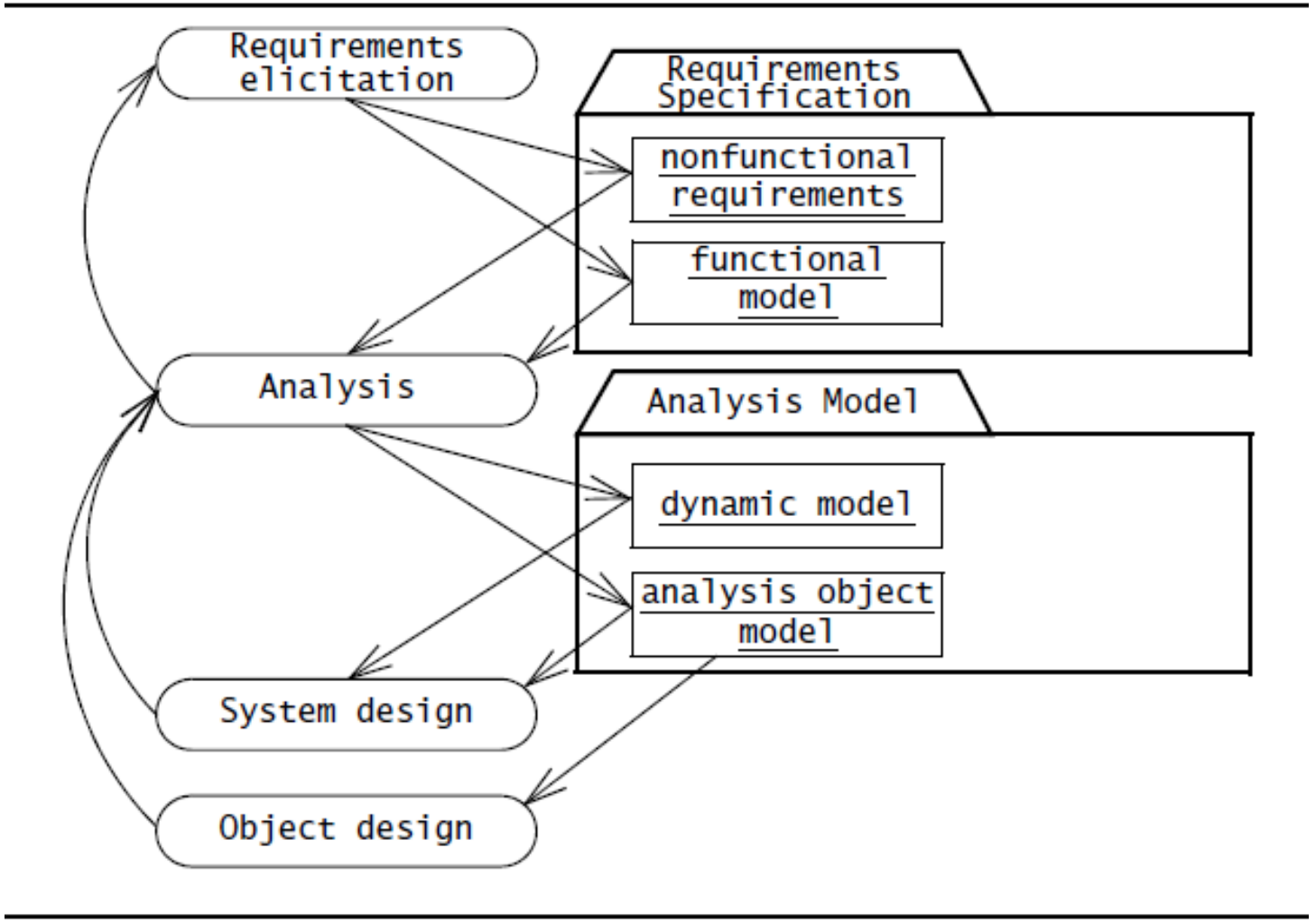
fonksiyonel modelin düzeltilmesi ile nesne ve dinamik modellerin üretilmesini göreceğiz.

Bu işlemler model analizinin daha detaylı ve daha bütün bir tanımının yapılmasını sağlar.

ANALİZ

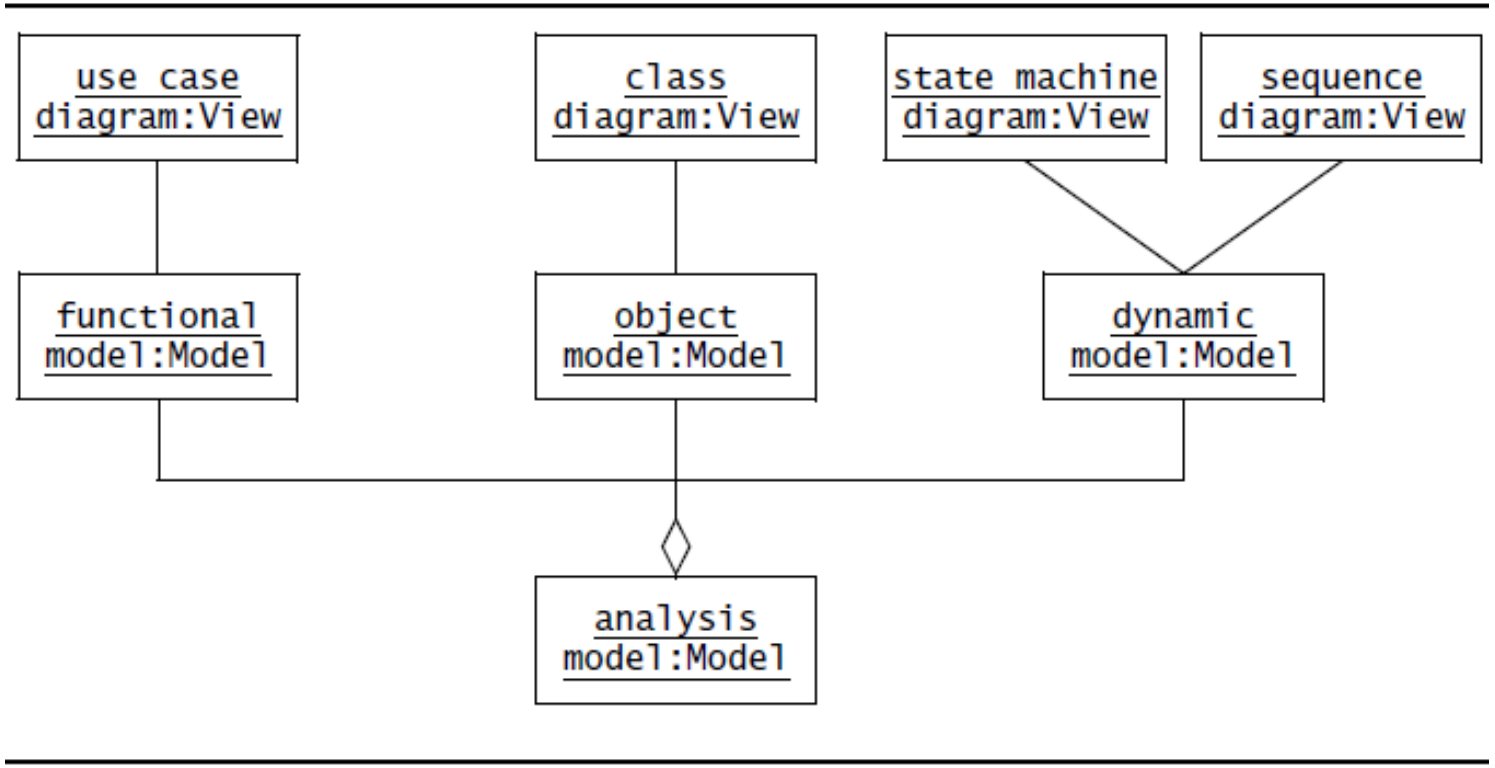
- Analiz, doğru, eksiksiz, tutarlı ve doğrulanabilir olan analiz modeli olarak adlandırılan sistemin bir modelinin üretilmesine odaklanır.
- Analiz, geliştiricilerin kullanıcılardan gelen gereksinimleri yapılandırmaya ve resmileştirmeye odaklandığı koşullardaki gereksinimlerden farklıdır.
- Analiz modeli, kullanıcılar ve müşteri için anlaşılabilir olmayabileceği için, geliştiricilerin, analiz sırasında elde edilen bilgileri yansıtacak şekilde gereksinim şartnamesini güncellemeleri ve ardından müşteriyle ve kullanıcılarla olan değişiklikleri gözden geçirmeleri gerekmektedir.
- Sonunda, gereksinimler, müşteri ve kullanıcılar tarafından anlaşılabilir olmalıdır.

ANALİZ


Requirements elicitation
Requirements Specification
nonfunctional requirements
functional model
Analysis
Analysis Model
dynamic model
analysis object model
System design
Object design

ANALİZ

- Analiz modeli üç ayrı modelden oluşmaktadır: kullanım durumları ve senaryolar ile temsil edilen **fonksiyonel model**, sınıf ve nesne diyagramları ile temsil edilen **analiz nesne modeli** ve durum makinesi ve dizi diyagramları (state machine and sequence diagrams) ile temsil edilen **dinamik model**

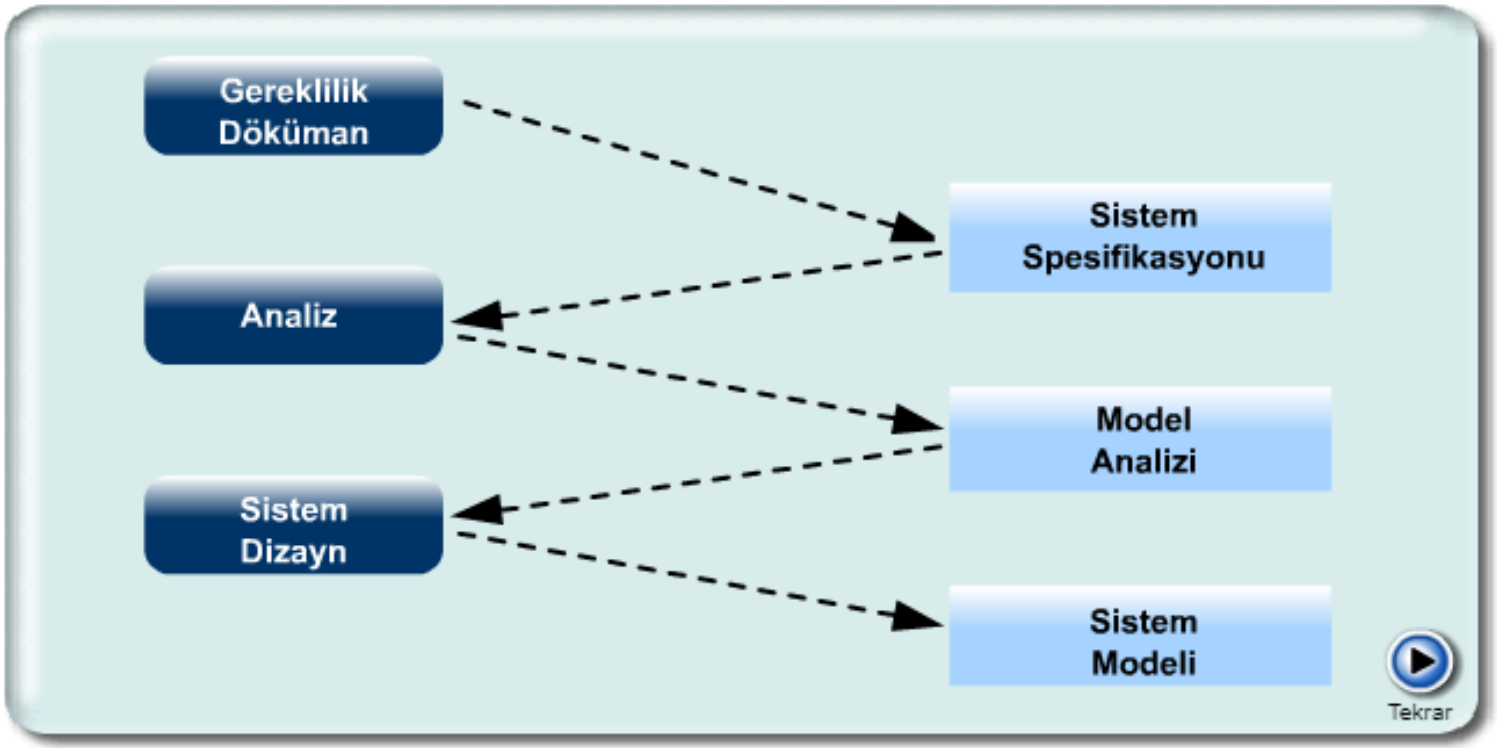


1.ANALİZ

- Nesne temelli analizde, programcılar uygulama alanını tarif eden bir model inşa ederler. Örneğin, bir saatin çalışma modeli, saatin zamanı nasıl gösterdiğini tarif etmelidir. (Saat, artık yılları göz önüne alıyor mu? Haftanın günlerini ekrana yazıyor mu? Ayın dönemleri hakkında bir bilgi var mı?)
- Sonraki adımda Model Analizi, aktörleri ve sistemin modeli nasıl değiştirdiğini de anlatmak üzere genişletilir. (Saatin sahibi saati nasıl sıfırlar? Saatin sahibi haftanın günlerini nasıl ilk haline getirir?)
- Bu işlemlerin sonucunda meydana çıkan Model Analizi, fonksiyonel olmayan gerekliliklerle bir araya getirilerek Sistemin Mimarisi hazırlanır.

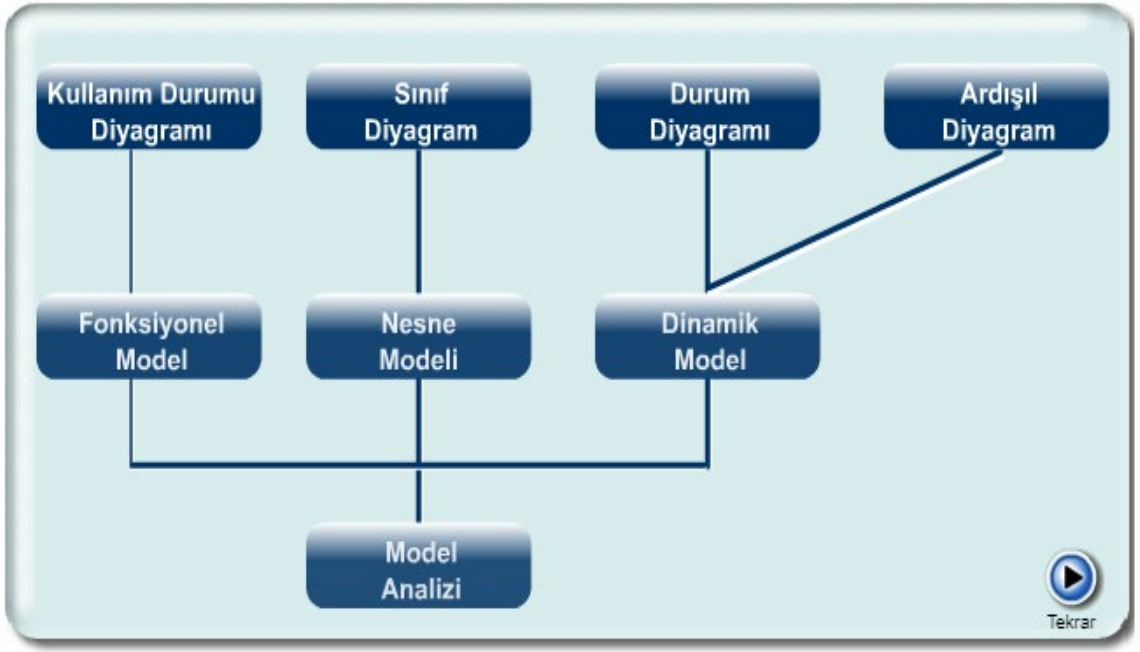
1. 1.İsterlik Dökümanı ve Analiz

- Analiz, İsterlik Dökümanından şu açıdan farklıdır. İsterlik dökümanında programcılar kullanıcıyla birlikte meydana çıkardıkları gereklilikleri yapılandırırlar.
- Diğer taraftan model analizi son kullanıcılar ve müşteriler tarafından anlaşılmak zorunda değildir. Model Analizi, programcıların İsterliklerin meydana çıkarılması aşamasında sistem tanımını geliştirmelerine yarar.



1.3. Varlık, Kenar ve Kontrol Nesneleri (Entity, Boundary and Control Objects)

- Model analizi üç bağımsız modelden meydana gelir:
  - Fonksiyonel model
  - Nesne model analizi
  - Dinamik model
- Fonksiyonel model değişik senaryolar ve Durumlarla ifade edilir, Nesne model analizi sınıf ve nesne diyagramlarıyla gösterilir, Dinamik model ise işlev diyagramlarıyla ifade edilir.



- Model Analizi, Fonksiyonel Model, Nesne Modeli ve Dinamik Modelden meydana gelir.
- İsterlerin belirlenmesi aşamasında Senaryolar üretilmişti. Bu bölümde, fonksiyonel modelin düzeltilmesi; nesne ve dinamik modellerin üretilmesini göreceğiz. Bu işlemler model analizinin daha detaylı ve daha bütün bir tanımını sağlar.

1.2. Model Analizi

- Nesne model analizi varlık (entity), sınır (boundary) ve kontrol (control) nesnelerinden oluşur.
  - Varlık Nesneleri, sistem tarafından izlenen bilgileri,
  - Kenar nesneleri, aktörler ve sistem arasındaki ilişkileri,
  - Kontrol nesneleri, sistem tarafından desteklenen ve kullanıcı tarafından yapılan görevleri gösterir
- Sistemi varlık, kenar ve kontrol nesneleri ile göstermek,
  - Programcıların, farklı fakat ilglili kavramları ayırt etmesine yardımcı olur.
  - Daha küçük ve daha özel nesnelerin oluşturulmasına yardımcı olur.

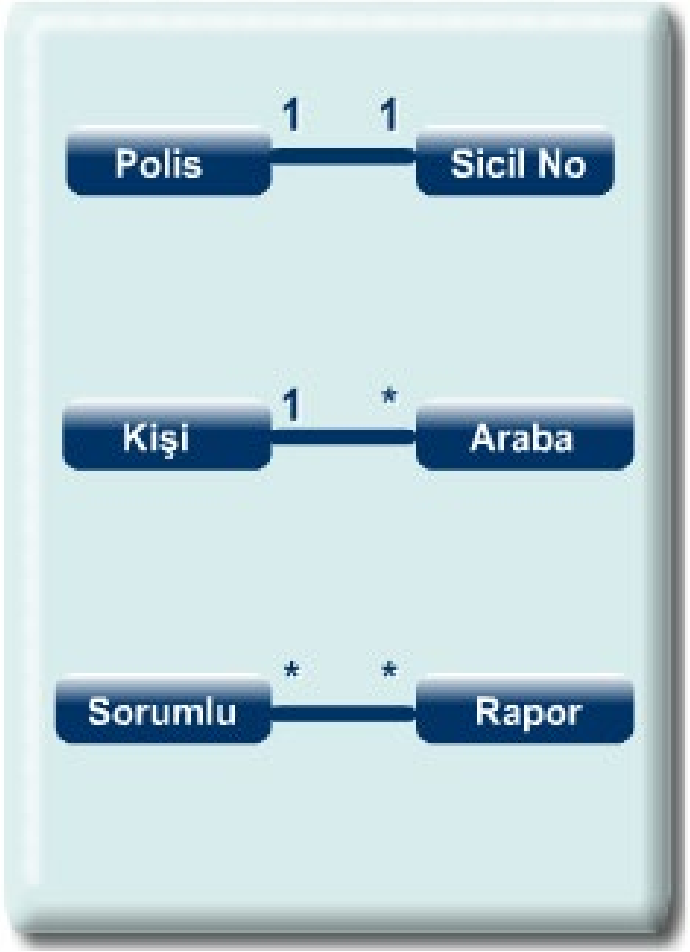
UML üç tip nesne tabanlı modelleme yöntemini destekler. *<<control>>* ön eki TarihDeğiştir nesnesine eklenebilir. Ayrıca isimlendirme kurallarını da uygulayabiliriz. Örneğin kenar nesnelerinin isimlerinin sonuna *Boundary* eki, kontrol nesnelerinin sonuna da *Control* eki konabilir.

1.4. İlişkinin Değerliliği

- UML'den de bildiğimiz gibi her ilişki *değerlilik* olarak isimlendirilen değeri tamsayı olan bir sayı ile işaretlenebilir. Bir ilişkinin değerliliği o ilişkinin orijini olan sınıfın bağlandığı nesneye olan bağların sayısıdır.

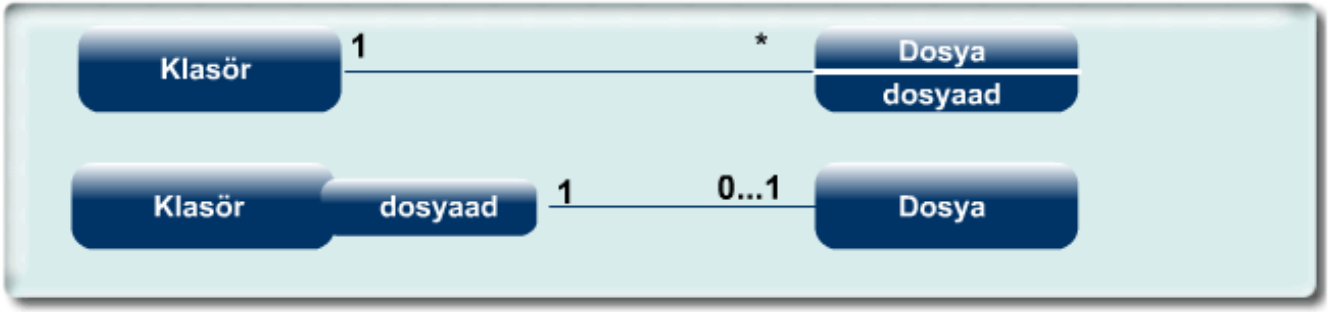
- UML de bir ilişkinin değerliği herhangi bir sayı olabilir. Fakat pratikte 3 durumdan bir tanesi ile karşılaşırız.

- İlişkilere değerlilikler vermek uygulama alanından alabileceğimiz bilgi miktarını artırır. İlişkinin değerliliğinin verilmesi, uygulama alanını tasarlamak için hangi örneklerin gerektiği belirlendiğinde önemli hale gelir.



1.5. İndirgenmiş İlişkiler

- İndirgeme bir ilişkinin değerliliğini anahtarlar kullanarak düşürmektir. 0...1 veya 1 değerlilikli ilişkiler, 0...n veya 1...n değerlilikli ilişkilere göre daha kolay anlaşılır. *Bir-çok*ilişki durumunda, ilişkinin *çok* tarafındaki nesneler bir diğerinden isim kullanılarak ayırt edilebilirler.
- Örneğin hiyerarşik bir dosya yapısında, her dosya bir klasörün içindedir. Her dosya bir klasörün içinde, diğer dosyalardan o dosyanın adı kullanılarak ayrılabilir. İndirgeme olmadan, Klasör ve Dosya arasındaki ilişkinin Klasör tarafında 1 değerliliği, Dosya tarafında ise 1...n değerliliği vardır. Dosya tarafında ilişkinin değerliliğini, dosyaadı özelliğini anahtar olarak kullanarak azaltırız.

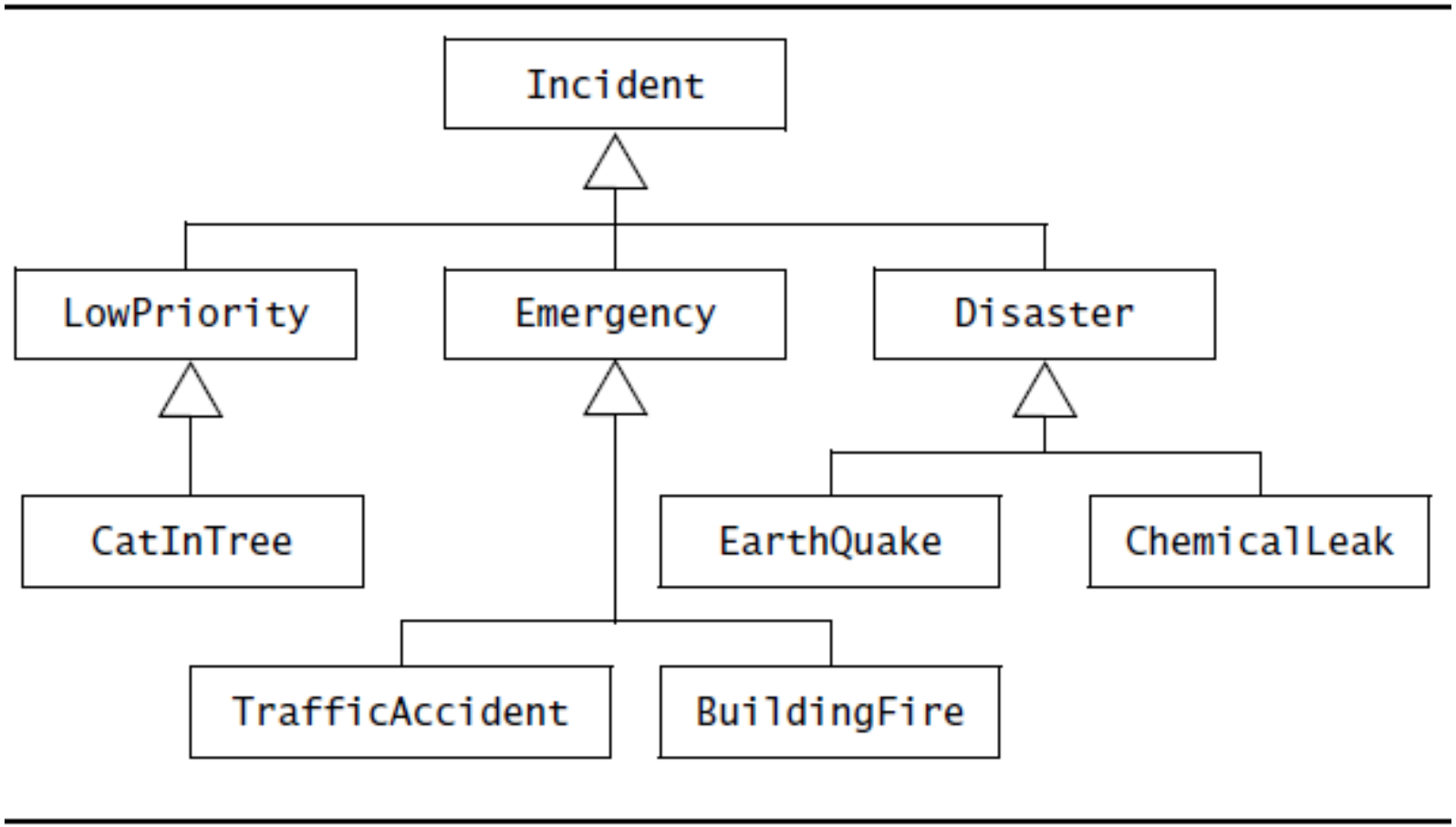


- İlişkinin değerliliğini düşürmek her zaman istenir. Değerlilik azaldıkça model daha anlaşılır ve daha az durumun gözönüne alınmasını sağlar.

1.6. Genelleştirme

- *Genelleştirme*, kavramları hiyerarşik olarak düzenlememizi olası kılar. Hiyerarşinin tepesinde genel bir kavram ve en altında ise en özelleşmiş kavramlar yer alır. Hiyerarşinin ortasında değişik miktarlarda seviye olabilir. Bu seviyeler az çok genel kavramları anlatır.
- Nesneye dayalı bir programlama dilinde, *miras* sık sık kullanılan bir tekniktir. Bir Çocuk sınıfı Ebeveyn sınıfından *miras* bir sınıfsa, Ebeveyn sınıfın tüm özellikleri ve metotları otomatik olarak Çocuk sınıfında da kullanılabilir olacaktır. Çocuk sınıfı ek metotlarla Ebeveyn sınıfın metotlarının üzerine yazabilir. *Genelleştirme*de olduğu gibi hiyerarşinin üst taraflarındaki sınıflar daha genel iken, alt kısımlarındaki kavramlar daha özelleşmiş kavramlardır.
- *Genelleştirme* ve *miras* kavramları birbirine yakın kavramlar olmakla beraber aynı değildirler. Miras özellikleri ve metotları bir yeniden kullanabilme yöntemidir. Bu yapılırken miras tekniğinde yer alan sınıfların bir *genelleştirme* bağlantıları olmak zorunda değildir.

1.6. Genelleştirme



2. Analiz Aktiviteleri

- Bu bölümde İsterlik Dökümanı'nın oluşturulması sırasında ürettiğimiz Durum ve Senaryolar'ı bir Model Analizi'ne dönüştürecek aktiviteleri göreceğiz.

Analiz aktiviteleri,

- Varlık nesnelerini belirleme
- Kenar nesnelerini belirleme
- Kontrol nesnelerini belirleme
- Durumları nesnelere eşleme
- Nesneler arasında ilişkiler belirleme
- Nesnelerin özelliklerini belirleme
- Net olmayan davranışları şemalarla modelleme
- Genelleştirmeleri modelleme
- Analiz modelini gözden geçirme

2.1. Varlık Nesnelerini Belirleme

- Model analizinin temelini modele katkıda bulunan varlık nesneleri oluşturur. Sistem spesifikasyonundan doğal dil analizi ile nesneleri, özellikleri ve ilişkileri belirlemek bilinen bir yöntemdir (Abbot 1983). Konuşmada geçen özneler, sıfatlar ve fiilimsiler nesneleri, işlemleri ve sınıfları belirlemede etkilidir.
- Doğal dil analizi kullanıcının terminolojisine dayanması açısından avantajlıdır. Fakat aynı şekilde birçok uygulamada sınırlar taşır.
- Doğal dil kesinlikten uzaktır
- Doğal dilde olması gereken sınıf sayısından daha fazla isim vardır.
- Programcılar nesneler belirlendikten hemen sonra onların isimlerini, özelliklerini ve sorumluluklarını tanımlarlar. Her nesnenin isminin farklı olması bir standart olmuştur.
- Nesneleri kısaca tanımlamak yanlış anlamaları önlemek için önemlidir; fakat detaylar üzerinde fazla durmamak gerekir. Nesneler, özellikleri ve sorumlulukları rahat anlaşılabilir olduğu sürece tanımlanabilir. Detaylandırmayı uzatmamanın sebebi bu nesneler üzerinde sık sık gidip gelineceğindendir.
- Varlık nesnelerinin belirlenmesi sistemin tam bir tanımının oluşturulmasında küçük bir adımdır.

2.1. Varlık Nesnelerini Belirleme

Varlık nesnelerini tanımlamak için sezgisel yaklaşım

- • Kullanım durumunu anlamak için geliştiricilerin veya kullanıcıların açıklığa kavuşturulması gereken şartlar
- • Kullanım durumlarında tekrarlayan isimler (ör., Olay)
- • Sistemin izlemesi gereken gerçek dünya kuruluşları (örn. FieldOfficer, Dispatcher, Resource)
- • Sistemin izlemesi gereken gerçek dünyadaki aktiviteler (örn. EmergencyOperationsPlan)
- • Veri kaynakları veya lavabolar (ör. Yazıcı).

2.2. Kenar Nesnelerini Belirleme

- Sınır nesneleri, aktörleri ile sistem arayüzünü temsil eder. Her kullanım durumunda, her aktör en az bir sınır nesnesiyle etkileşime girer. Sınır nesnesi, bilgiyi aktörden toplar ve onu hem varlık hem de kontrol nesneleri tarafından kullanılabilecek bir forma dönüştürür.
- Kenar nesneleri kullanıcı arayüzünü yüzeysel olarak modeller. Kullanıcı arayüzünün görüntüsel özelliklerini detaylandırmazlar. Örneğin *düğme* veya *menü parçası* çok detaylı olabilecekı nesnelerdir. Kullanıcı arayüzü programcılar tarafından önce tasarlanır daha sonra kullanıcı testlerinden geçerek değişirler. Her görüntü değişikliğinde model analizini değiştirmek hem zaman kaybettirici hem de fazla bir şey kazandırmayan bir yöntemdir.

2.2. Kenar Nesnelerini Belirleme

Sınır nesnelerini tanımlamak için sezgisel yöntemler

- • Kullanıcının kullanım durumunu başlatması gereken kullanıcı arayüzü kontrollerini tanımlayın (ör. ReportEmergencyButton).
- • Kullanıcının sisteme veri girmesi için gereken formları tanımlayın (ör. EmergencyReportForm).
- • Sistemin kullanıcıya cevap vermek için kullandığı bildirimleri ve mesajları belirleyin (örn. Onaylama Bilgisi).
- • Bir kullanım durumunda birden fazla aktör söz konusu olduğunda, değerlendirilen kullanıcı arayüzüne başvurmak için aktör terminallerini (örn. DispatcherStation) tanımlayın.
- • Arayüzün görsel yönlerini sınır nesnelerle modellemeyin (kullanıcı maketleri bunun için daha uygundur).
- • Son kullanıcı arayüzlerini tanımlamak için daima son kullanıcı şartlarını kullanın; Çözüm veya uygulama alanlarındaki terimleri kullanmayın.

2.3. Kontrol Nesnelerini Belirleme

- Kontrol nesneleri varlık ve kenar nesneleri arasındaki koordinasyonu sağlamaktan sorumludur. Kontrol nesnelerinin gerçek dünyada tam bir karşılıkları yoktur. Çoğu zaman bir durumla bir kontrol nesnesi arasında sıkı bir ilişki vardır.
- Bir kontrol nesnesi durumun başında yaratılır ve durum sonunda varlığı sona erer. Sorumluluğu kenar nesnelerinden bilgi toplamak ve bunu varlık nesnelerine iletmektir. Örneğin, dağınık bir sistemde kontrol nesnelerinin görevi formların sıralanmasınının, geri alma ve tarih sıralarının tutulmasını sağlamaktır.

2.3. Kontrol Nesnelerini Belirleme

- Kontrol nesnelerini tanımlamak için sezgisel yaklaşım
  - • Kullanım durumu başına bir kontrol nesnesini tanımlayın.
  - • Kullanım durumunda aktör başına bir kontrol nesnesini tanımlayın.
  - • Bir kontrol nesnesinin ömrü, kullanım durumunun kapsamını veya bir kullanıcı oturumunun kapsamını kapsamalıdır. Bir kontrol nesnesi aktivasyonunun başlangıcını ve sonunu belirlemek zorsa, ilgili kullanım durumu muhtemelen iyi tanımlanmış giriş ve çıkış koşullarına sahip değildir.

2.4. Nesneler Arası İnteraksiyonları Modelleme: Ardışıl Diyagramlar

- Ardışıl Diyagramlar durumları nesnelere bağlar. Bir Durumun veya Senaryonun davranış biçimini katılan nesneler arasında dağıtır. Ardışıl Diyagramlar yardımıyla kullanıcı ile haberleşilmez fakat Ardışıl Diyagramlar bize unuttuğumuz nesneler ve sistem tanımında geri kalan alanları gösterir.
- Bir Ardışıl Diyagram'ın sütunları o durumda görev alan nesneleri gösterir. En soldaki sütun durumu başlatan aktörü gösterir. Satırlarda gösterilen oklar, nesneler arasında gönderilen mesajları gösterir. Zaman yukarıdan aşağıya doğru ilerler. Diyagramda görülen dikdörtgenler başka mesajların üretilebileceği yerlerdir. Dikdörtgenlerin uzunluğu işlemin uzunluğu ile doğru orantılıdır.

2.4. Nesneler Arası İnteraksiyonları Modelleme: Ardışıl Diyagramlar

- Genel olarak, Ardışıl Diyagramlar'ın ikinci sütunu aktörün durumu başlatmak için kullandığı kenar nesnesini gösterir. Üçüncü sütun ise durumun kalanını kontrol eden kontrol nesnesidir. Buradan itibaren, kontrol nesnesi gerekli gördüğü diğer kenar nesnelerini üretir ve diğer kenar nesneleri ile gerekirse interaksiyona girebilir.
- Ardışıl Diyagramlar'ı tasarlayarak, sadece nesneler arasındaki interaksiyonun sırasını modellemekle kalmıyor, durumun davranışını da dağıtmış oluyoruz. Diğer bir şekilde söylersek, her nesneye bir dizi işlem olarak sorumluluklar yüklüyoruz. Bu işlemler aynı nesnenin çalıştığı diğer durumlarda da kullanılabilir. Burada önemli nokta, aynı işlem birden fazla durumda görülüyorsa her durumdaki tanımının aynı olmasıdır.

2.4. Nesneler Arası İnteraksiyonları Modelleme: Ardışıl Diyagramlar

- Durumlar arasında işlemleri ortaklaşa kullanmak programcıların sistem tanımındaki gereksiz noktaları ayıklamasını sağlar ve sistem tanımının tutarlılığını artırır. Anlaşılabilirlik, her zaman sistemin gereksiz noktalarının ayıklanmasından önce gelir.
- Analizde, Ardışıl Diyagramlar yeni katılan nesneleri ve eksik davranışları anlamamıza yarar. Yüksek seviyede davranışa odaklanırız. Ardışıl Diyagramlar'ı tasarlamak zaman alıcı olsa da, iyi bir Ardışıl Diyagram tasarlanmasına harcanan zaman kesinlikle kaybedilmemiş bir zamandır.

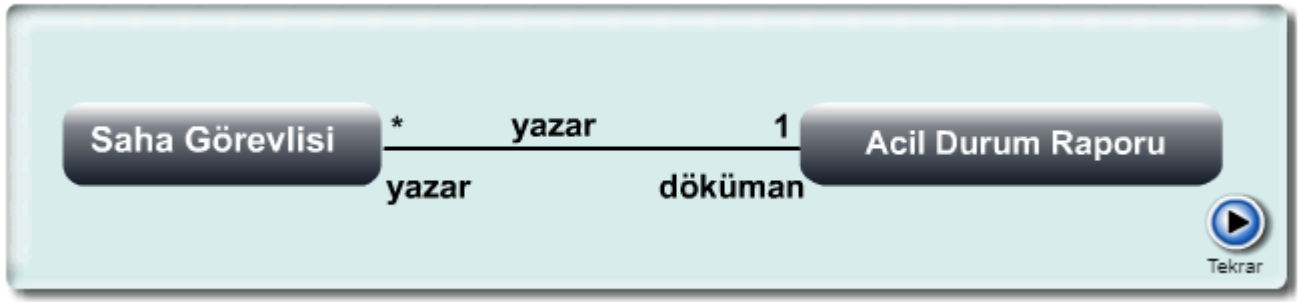
2.4. Nesneler Arası İnteraksiyonları Modelleme: Ardışıl Diyagramlar

Dizi diyagramlarının çizilmesi için sezgisel yaklaşım

- • İlk sütun, kullanım durumunu başlatan oyuncuya karşılık gelmelidir.
- • İkinci sütun bir sınır nesnesi olmalıdır (aktör, kullanım durumunu başlatmak için kullanılır).
- • Üçüncü sütun, kullanım durumunun kalanını yöneten kontrol nesnesi olmalıdır.
- • Kontrol nesneleri, kullanım durumlarını başlatan sınır nesneleri tarafından oluşturulur.
- • Sınır nesneleri kontrol nesneleri tarafından oluşturulur.
- • Varlık nesnelerine kontrol ve sınır nesneleri tarafından erişilir.
- • Varlık nesneleri asla sınırlara veya kontrol nesnelerine erişemez; Bu, varlık nesnelerinin kullanım durumları arasında paylaşılmasını kolaylaştırır.

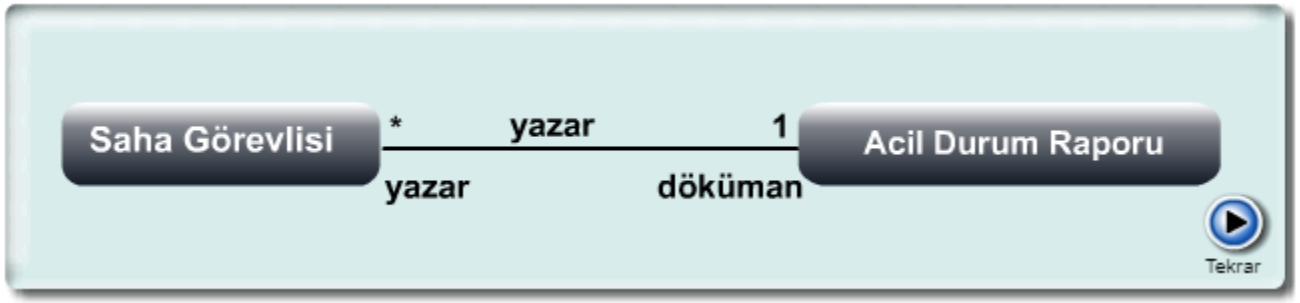
2.5. İlişkileri Belirleme

- Ardışıl Diyagramlar, programcıların nesnelerin zaman içindeki interaksiyonunu göstermek için bir yöntem olduğu gibi, Sınıf Diyagramları ise nesneler arasındaki uzaysal bağlantıyı tanımlar.
- Bir ilişki iki veya daha fazla sınıf arasında tanımlanır. Örneğin SahaGörevlisi bir AcilDurumRaporu yazar. İlişkileri belirlemenin iki avantajı vardır. Birinci olarak, nesneler arasındaki ilişkileri tam olarak tanımladığı için model analizinin anlaşılabilirliğini artırır. İkinci olarak, programcının bağlantılar ile ilgili olarak kenar durumlarını keşfetmesini sağlar. Kenar durumları modelde mutlaka anlaşılması gereken noktalardır.



2.5. İlişkileri Belirleme

- Örneğin, biz bir çok AcilDurumRaporu'nun sadece bir SahaGörevlisi tarafından yazılabileceğini biliriz. Ama, acaba sistem AcilDurumRapor'larının birden fazla kişi tarafından yazılabilmesini destekleyecek midir? Hiç kimsenin yazmadığı AcilDurumRapor'ları olabilir mi? Bu tip sorular alan bigisi kullanılarak araştırılması gereken sorulardır.



2.5. İlişkileri Belirleme

- İlişkilerin belli başlı üç özelliği vardır:
  - İsim: İki sınıf arasındaki ilişkiyi tanımlamak için verilir. İlişki isimleri opsiyoneldir ve iki ilişki ismi aynı olabilir.
  - Rol: Her iki noktada da bir rol tanımlı olmalıdır. Bu rol ilişkiye göre sınıfların fonksiyonlarını belirleyecektir.
  - Değerlilik: Her iki noktada da olası durum sayısını belirler.
- Varlık nesneleri arasındaki ilişkiler en önemli ilişkilerdir. Bunun sebebi uygulama alanı hakkında en çok bilgi vermeleridir. İlişkileri belirlerken en sık kullanılan yöntem, fiilleri ve fiilimsileri incelemektir. Fakat tüm fiil ve fiilimsiler incelenir, tümü için ilişkiler çıkarılırsa, sistemde çok fazla ilişki oluşabilir. Bu da modelin karmaşık olmasına ve gereksiz bilginin depolanmasına neden olur.
- Model analizi belirlendikten sonra programcılar, her sınıfın üzerinden geçmeli ve hangi aktörlerle, hangi bağlamda belirlendiğini anlamalıdırlar.

2.6. Öznitelikleri Belirleme

- Özenitelikler her nesnenin kendine özgü nitelikleridir. Örneğin bir AcilDurumRaporu'nun, bir aciliyet tipi, bir yer bilgisi, ve bir tanım bilgisi vardır. Bu bilgiler SahaGörevlisi tarafından bir acil durum bildirildiğinde girilir ve sistem tarafından sürekli takip edilir. Nesnelerin öznitelikleri belirlendiğinde, sadece sistemle alakalı olanlarına dikkat edilmesi gerekir.
- Örneğin her SahaGörevlisi'nin acil durumlarla alakası olmayan bir SSN numarası vardır. Buna karşılık SahaGörevlileri'ni birbirinden ayıran bir yakanumarası vardır.

2.6. Öznitelikleri Belirleme

- Nesneler tarafından gösterilen nitelikler öznitelik olarak belirlenmezler. Programcılar öznitelikleri belirlemeden önce olabilecek tüm nesneleri ve onların arasında olabilecek tüm ilişkileri tanımlamalıdırlar. Bu şekilde nitelik olmayan özellikleri sistem tanımına yerleştirmemiş olurlar.
- Her özniteliğin bir ismi,tanımı ve tipi vardır:
  - **İsim:** Nesne içinde onu belirleyen bir ismi vardır. Örneğin AcilDurumRaporu'nun bir raportipi ve bir aciliyetdurumu özniteliği olabilir. Raportipi özniteliği doldurulan raporun tipini, aciliyetdurumu ise olayın aciliyetinin ne olduğunu gösterir.
  - **Tanım:** Kısa bir tanım.
  - **Tip:** Özelliğin alabileceği olası değerleri belirleyen bir veri tipi. Örneğin AcilDurumRaporu'nun tanım özelliğinin tipi katar (string)'dır.

2.6. Öznitelikleri Belirleme

Öznitelikleri belirlemek için öneriler:

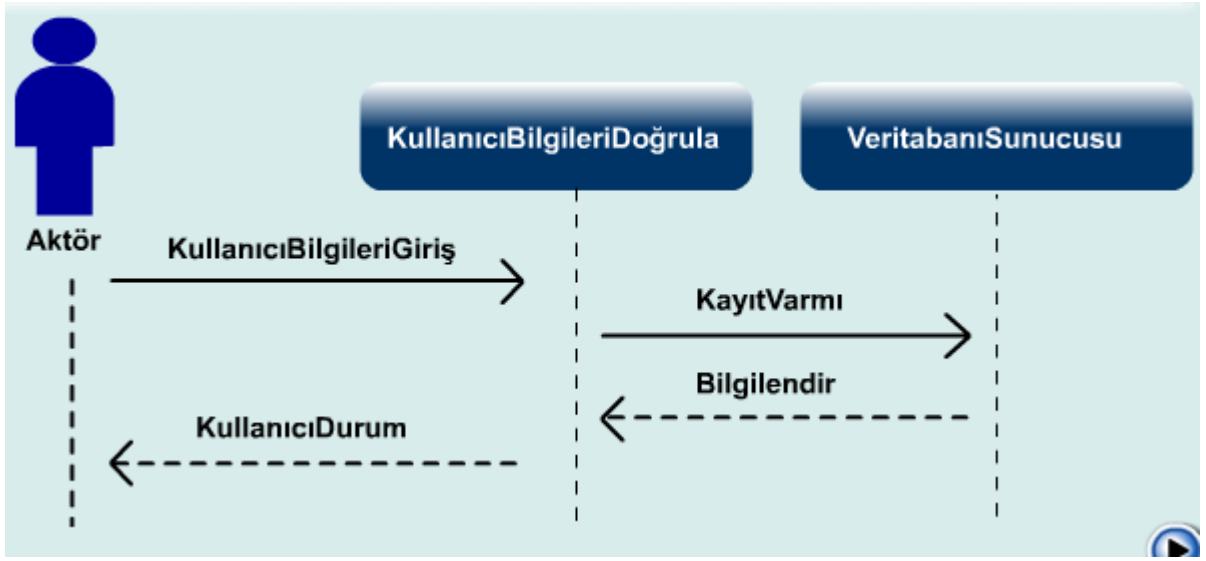
- **Sahip olma şeklindeki (var) kelimeleri inceleyin.**
- **Her özelliği tanımlayın.**
- **Bir özelliği bir nesne olarak belirlemeyin, bunun yerine ilişkileri kullanın.**
- **Nesne yapısı kararlı hale gelmeden detaylar üzerinde durmayın.**

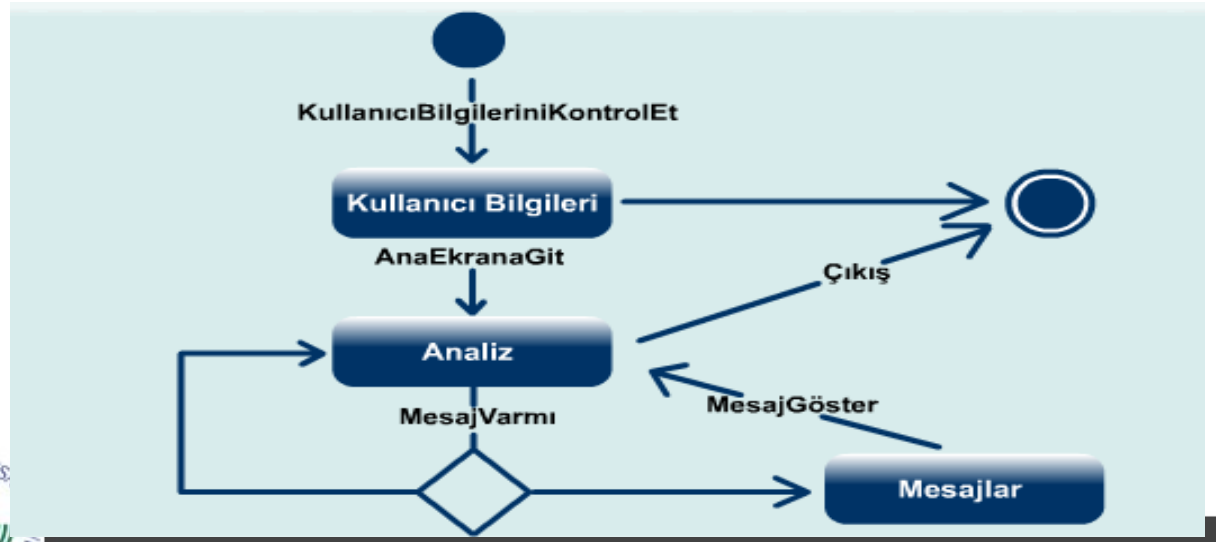
- Dikkat edilirse görülür ki, öznitelikler nesne modelinin en az kararlı taraflarıdır. Sistemin tasarlanması devam ettiği sürece sık sık yeni öznitelikler eklenir veya silinir. Eklenen öznitelikler yeni işlevler ekledikleri sürece üzerinde durulmalı, daha önemsiz öznitelikler sistemin akışı içinde belirlenmelidir.

2.7. Basit Olmayan Davranışlı Nesneleri Modelleme

- Ardışıl Diyagramlar'ı nesneler arasında davranışları dağıtmak ve işlemleri belirlemek için kullanırız. Ardışıl Diyagramlar, sistemin davranışını tek bir durumun bakış açısından gösterir.

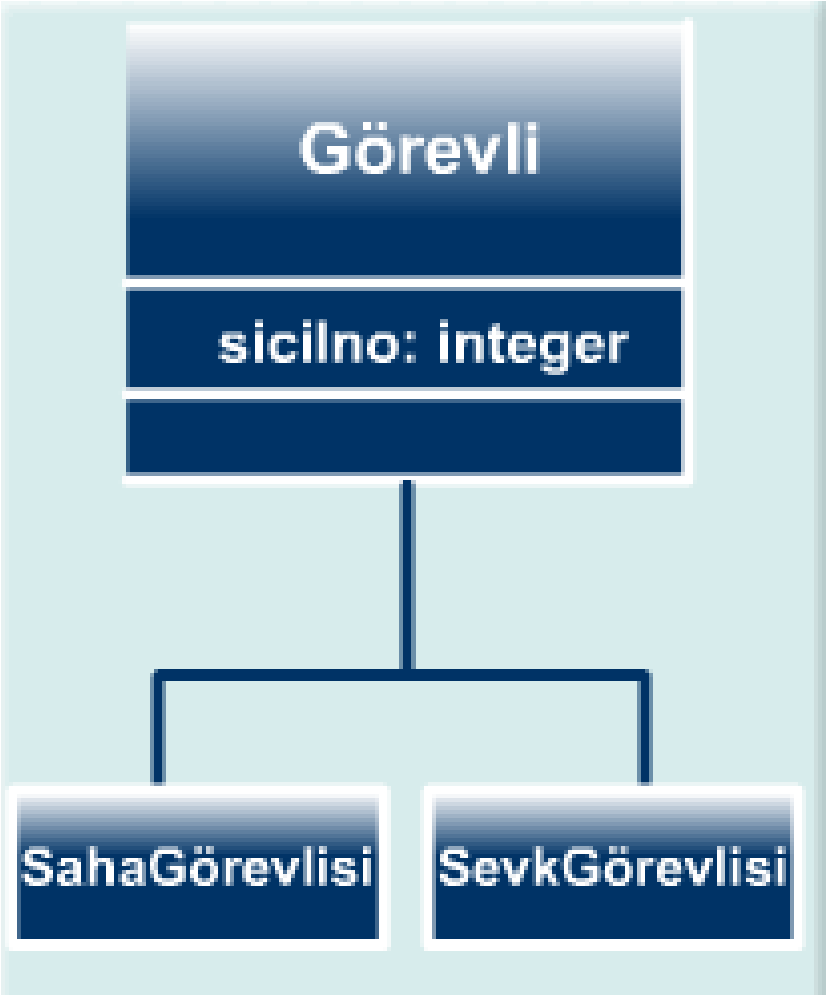
- Durum Diyagramları (Statechart) ise, tek bir nesnenin bakış açısından davranışları bize gösterir. Tek bir nesnenin bakış açısını yakalamak, programcıya, unutulmuş durumları belirlemesinde ve nesnenin davranışının daha formal tanımını yapabilmesinde olanak sağlar. Durum Diyagramları'nın her sınıf için yapılması gerekmemektedir.





2.8. Nesneler Arasındaki Genelleştirme İlişkilerinin Modellenmesi

- Genelleştirme, Model Analizi'nde gereksiz unsurların atılmasında kullanılır. Eğer iki veya daha fazla sınıfın ortak özellikleri veya davranışları varsa, benzerlikler bir üst sınıfa aktarılır.
- Örneğin, SahaGörevlisi ve SevkGörevlisi'nin aynı şehir içinde ayırt edilebilmeleri için birer sicilnumaraları vardır. Hem SahaGörevlisi hem de SevkGörevlisi aslında farklı görevlere atanmış birer Görevli'dir.
- Bu benzerliği modellemek için, SahaGörevlisi ve SevkGörevlisi sınıflarından soyut bir Görevli sınıfı ortaya çıkarılır.



2.9. Model Analizini Gözden Geçirme

- Model Analizi yavaş yavaş ve aşama aşama inşaa edilir. Model Analizi, ilk geçişte nadiren doğru ve tam ortaya çıkar. Programcıların Model Analizi üzerine dizayn ve uygulama yapabilmesi için, müşteri ve kullanıcılarla birden fazla konuşma yapması gerekebilir. Örneğin, analiz sırasında unutulan bir şey yüzünden yeni bir durum eklenebilir veya var olan durumun genişletilmesi gerekebilir.
- Analiz Modeli kararlı hale geldiğinde, önce programcılar sonra müşteri tarafından gözden geçirilir. Gözden geçirmeyle doğru, tam, tutarlı ve gerçekçi bir sistem spesifikasyonunun yapılıp yapılmadığı anlaşılabilir.

3. Analizi Yönetmek

- Bu bölümde, birden çok takımlı bir projede analiz aktivitelerini yönetmeyle ilgili konulara değineceğiz.
- Böyle bir projede gereksinimlerin yönetilmesindeki birincil zorluk, çok fazla kaynak kullanırken tutarlılığı korumaktır.
- Sonuç olarak, gereksinim analizi dokümanı, tek bir kişiye anlaşılabilir olan tek bir tutarlı sistemi tanımlamalıdır.
- İlk olarak analiz sonuçlarını belgelemek için kullanılabilecek bir belge şablonunu tanımlanır.
- Ardından, analize yönelik rol atamasını açıklanır.
- Daha sonra analiz sırasında iletişim sorunlarını ele alınır. Ardından, gereksinimlerin yinelemeli ve artımlı doğasıyla ilgili yönetim sorunlarını ele alıbır.

3. Analizi Yönetmek

3.1. Analizi Dökümante Etme

- İsterlik analizi ve analiz aktiviteleri İsterler Analizi Dökümanında (RAD) anlatılır. Analiz sırasında RAD dökumanındaki 1-3.5.2 arasındaki bölümlerde yapılanlar gözden geçirilir ve gerekirse yeni fonksiyonlar keşfedilebilir. Esas iş ise, nesne analizi modelinin yazılmasıdır (RAD'daki bölüm 3.5.3-3.5.4).
- 3.5.3 bölümünde, nesne modelleri kısmında tüm nesneler detaylandırılır, özellikleri, durumlar ve Ardışıl Diyagramlar düşünülerek gösterilir. Sınıf Diyagramları yardımıyla nesneler arasındaki ilişkiler çizilir.

Requirements Analysis Document

1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Scope of the system
   - 1.3 Objectives and success criteria of the project
   - 1.4 Definitions, acronyms, and abbreviations
   - 1.5 References
   - 1.6 Overview
2. Current system
3. Proposed system
   - 3.1 Overview
   - 3.2 Functional requirements
   - 3.3 Nonfunctional requirements
     - 3.3.1 Usability
     - 3.3.2 Reliability
     - 3.3.3 Performance
     - 3.3.4 Supportability
     - 3.3.5 Implementation
     - 3.3.6 Interface
     - 3.3.7 Packaging
     - 3.3.8 Legal
   - 3.4 System models
     - 3.4.1 Scenarios
     - 3.4.2 Use case model
     - *3.4.3 Object model*
     - *3.4.4 Dynamic model*
     - 3.4.5 User interface—navigational paths and screen mock-ups
4. Glossary

3. Analizi Yönetmek

3.1. Analizi Dökümante Etme

- 3.5.4 bölümünde, dinamik modellerde Durum Diyagramları (Statechart) ve Ardışıl Diyagramlar anlatılır. Durum Modeli yüzünden bu kısım gereksiz gibi olmuşsa da, daha karmaşık davranışlar bu kısımda detaylandırılabilir.

Requirements Analysis Document

1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Scope of the system
   - 1.3 Objectives and success criteria of the project
   - 1.4 Definitions, acronyms, and abbreviations
   - 1.5 References
   - 1.6 Overview
2. Current system
3. Proposed system
   - 3.1 Overview
   - 3.2 Functional requirements
   - 3.3 Nonfunctional requirements
     - 3.3.1 Usability
     - 3.3.2 Reliability
     - 3.3.3 Performance
     - 3.3.4 Supportability
     - 3.3.5 Implementation
     - 3.3.6 Interface
     - 3.3.7 Packaging
     - 3.3.8 Legal
   - 3.4 System models
     - 3.4.1 Scenarios
     - 3.4.2 Use case model
     - *3.4.3 Object model*
     - *3.4.4 Dynamic model*
     - 3.4.5 User interface—navigational paths and screen mock-ups
4. Glossary

3. Analizi Yönetmek

3.2. Sorumlulukları Verme

- Analiz değişik alanlardan gelen birçok kişinin katılımı ile yapılır. Hedef kullanıcı uygulama alanı bilgisini verir. Müşteri projeyi mali açıdan destekler ve projenin kullanıcı tarafını koordine eder. Analist uygulama alanı bilgisini kavrar ve formalize eder. Programcılar uygulanabilirlik ve tutar konusunda bilgi verirler. Proje yöneticisi programcı tarafındaki çalışmayı koordine eder.
- Büyük sistemlerde, hem kullanıcılar, hem analizciler hem de programcıların sayısı bir hayli kalabalık olabilir, bu da sistemin birleştirilme aşamasını daha zor kılar. Bu zor aşamayı atlatabilmek için görevler tam olarak tanımlanmalıdır.
- Üç tip rol vardır: Bilginin üretilmesi, Birleştirme ve Gözden Geçirme.

3. Analizi Yönetmek

3.3. Analiz Hakkında Haberleşme

Gereksinim anlalizi dökümanı hazırlanması ve analizi aşamasındaki en büyük problemlerden birisi bilgiyi iletmektir.

- Kullanıcılar, müşteriler ve programcılar değişik kültürlerden ve değişik alt yapılardan gelmişlerdir.
- Kullanıcılar, müşteriler ve proje yöneticilerinin sistemden beklentileri farklıdır.
- İsterler analizi dokümanı, genelde yeni bir projenin ve dolayısı ile zaman zaman yeni bir takımın işe başlamasını gerektirir.

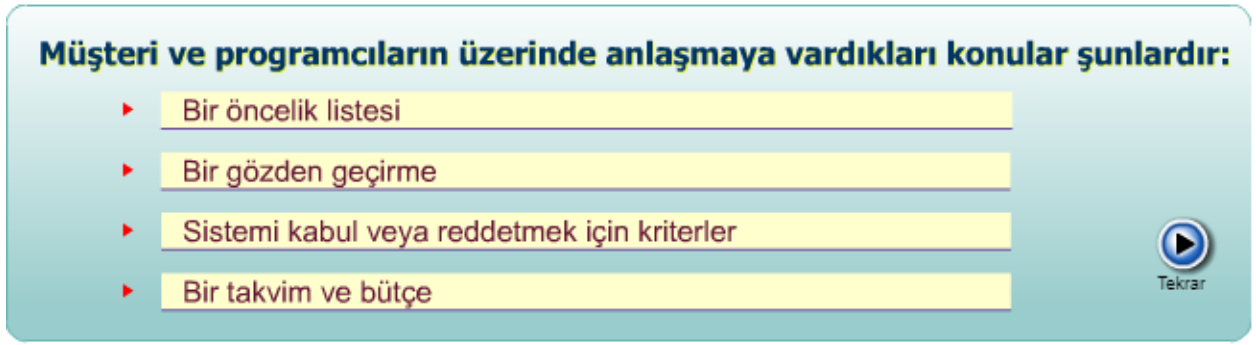
Bütün bu sorunları tam olarak çözen bir yöntem henüz bulunamamıştır. Fakat aşağıda birkaç öneri verilmektedir:

- Sınırları tam olarak belirleyin. Takımlar arasında gerekirse gizli iç toplantılar düzenlenmeli ve bunu sağlamak için bir veri tabanı kullanılabilmelidir.
- Amaçlar ve başarı kriterleri tam olarak belirlenmeli ki müşteri ve proje yöneticisi taraflarında bir sorun çıkmasın.
- Tüm çalışanları bir odaya toplayıp odada beyin jimnastiği yapmak, hem proje için hem de insan ilişkileri için faydalıdır.

3. Analizi Yönetmek

3.4. Müşteriyle anlaşma

- Müşteriyle anlaşma, model analizinin kabul edildiği anlamına gelir. Müşteri ve programcılar tek bir düşünce (problem çözümü) etrafında birleşirler. Ayrıca aşağıda belirtilen konular üzerinde anlaşırlar.



3. Analizi Yönetmek

3.4. Müşteriyle anlaşma

- Bir öncelik listesi üzerinde uzlaşma programcılar için önemlidir. Proje birden fazla parça halinde teslim edilecekse önce hangi parçaların önemli olduğu bu yöntemle belirlenebilir. Sistem tek bir parça halinde devredilecek ise de fonksiyonları öncelik sırasına dizmek müşterinin kendisi için önemli parçaları bildirmesi ve uygulamanın ne üzerinde durularak yapılacağının bilinmesi için önemlidir.
- Gözden geçirme, müşteri ve programcının anlaşmadan sonraki değişikliklerin nasıl halledileceğinin belirlendiği aşamadır. Tabiki çeşitli sebeplerden dolayı isterler değişecektir.
- Sistemin kabulü veya reddi için kriterler müşterinin korunmasına yöneliktir. Müşterinin programcının yaptığı değişiklikleri kabul etme veya etmemesi için bir temel ihtiyacı vardır.