Yazılım Geliştirmede Sistem Modelleme

Giriş

Modelleme kavramı, tüm bilim dallarında olguların sistem yaklaşımıyla incelenmesi ve tasarlanmasında kullanılan eski bir kavramdır. Gerçek hayatta karşılaşılan sistemler karmaşıktır ve eğitim ve zeka düzeyi ne olursa olsun her insanın algılama ve çözümleme yeteneğinin bir üst sınırı vardır. Gerçek sistemlerin karmaşıklığı, sistemin tek kişi tarafından tüm yönleriyle, bir defada kavranmasını olanaksız kıldığından bu tür sistemlerin parçalar halinde incelenebilmesini ya da tasarlanabilmesini sağlayacak başka yöntemler kullanılması gerekir. Modelleme kavramı, insanlığın sistemlerin karmaşıklığı ile başetmekte kullandığı en eski ve en etkin yöntemdir.

Bir sistemi modelleyen kimse, sistemi değişik açılardan tekrar tekrar inceler. Bu amaçla, sistemin değişik özelliklerinden o anda ilgilendiklerini öne çıkarırken, diğerlerini geriye iten soyut yapılar kurar. Bu soyut yapıların her biri, sistemin ilgilenilen özelliklerinin bir *model*idir. Modelleme yapan kişi, sistemin kendi ilgi alanına giren özelliklerini bu soyut yapı üzerinde inceler, bu özellikler arasındaki bağlantıları keşfeder ve sistemin çalışma biçimini belirleyen kuralları saptar. Modelleme tasarım amacıyla yapılıyorsa, bu aşamalardan sonra tasarımını yapar ve yaptığı tasarımı yine model üzerinde sınar.

Hiçbir model, tanımı gereği, hedef sistemin tüm özelliklerini gerçeğe özdeş biçimde içermez. Her model aslının yalınlaştırılmış bir kopyasıdır ve bu nedenle aslındaki tüm ayrıntıları içinde barındırması olanaklı değildir. Ancak, sistemin o anda incelenen özellikleri ile ilgili ayrıntılardan, sistemin davranışında en belirleyici olanları mutlak olarak içermesi gerekir. Aksi durumda modelden elde edilecek sonuçların yol göstermekten çok yanıltıcı olacağı açıktır. Modellemeyi yapanın soyut düşünme yeteneği, modelin neleri içerip neleri dışarıda bırakması gerektiğine karar verirken yardımcı olur.

Gerçek hayattaki olguları çok farklı biçimlerde soyutlayıp modellemek mümkündür. Tek bir olguya bakıp buna ilişkin çok farklı modeller geliştirmek mümkündür. Modellemede kullanılan bu bakışlar ve yaklaşımların ortaya koyduğu matematik modeller, anolog modeller, ikonik modeller gibi model türleri mevcuttur.

Modeller sistem karmaşıklığını yönetilir boyutlara indirip anlaşılabilirliğini arttırmanın yanısıra tasarımcılar arasında bir iletişim aracı olarak da hizmet görürler. Bir tasarımın temel özelliklerinin açıklanmasıve çeşitli seçeneklerin değerlendirilmesi bir model üzerinde daha etkin biçimde yapılabilir. Bunun yapılabilmesi için modellemede ortak bir gösterim biçiminin, bir başka deyişle ortak bir modelleme dilinin kullanılması zorunludur.

Yazılım sektöründe modelleme için geliştirilmiş EXPRESS, EXPRESS-G, IDEF, UML gibi çeşitli modelleme dilleri mevcut olup, nesne tabanlı sistemlerin incelenmesi ve tasarımında standart olarak kullanılan modelleme dili UML'dir (Unified Modeling Language). Geçmiş deneyimler, düz metin ve simgelerin yanında görsel ögeler de içeren modellerin daha anlaşılabilir olduğunu göstermiştir. Bu nedenle UML özellikle görsel ögeler açısından zengin bir dil olarak tasarlanmıştır. Bu dil, 1994 yılına kadar geliştirilen nesneye yönelik modelleme yöntemlerinden sektörde genel kabul görmüş olanların harmanlanmış bir biçimidir. İlk genel tanımı 1995 Ekim'inde "unified method v08" olarak yayınlanan UML'in bu gün endüstride yaygın olarak kullanılan sürümü Ocak 1997'de OMG'ye (Object Management Group) Standart Önerisi olarak verilen 1.1 sürümüdür.

Yazılımda Sistem Modelleme ve UML

Yazılımda sistem modelleme süreci problemin tanımının yapılması ve bu probleme bir çözüm oluşturulması aşamalarını içerir. Bu iki aşama genel olarak *sistem çözümleme* ve *sistem tasarımı* olarak adlandırılır. Sistem çözümleme ve tasarım aşamalarında sistemin farklı yönlerini ortaya koyan değişik modelleri oluşturlur. Her model sistemin değişik bir açıdan incelenmesini sağlar ve yazılımcının sistemi kavramasında, çözüm seçenekleri oluşturmasında, müşteriyle ve meslekdaşları/proje grup üyeleriyle görüş alışverişinde bulunmasında yardımcı olur. UML modelleri yalnızca yazılım profesyonelleri tarafından kullanılmakla kalmayıp, otomatik kod üreteçleri tarafından da okunabilme özelliğine sahiptir.

Sistem tasarımcısının ilk aşamada yapması gereken çalışma, kullanıcı gereksiniminin belirlenmesidir. Gereksinim belirleme aşaması, bir yazılım projesindeki en kritik aşamadır. Bu aşamada değişik mesleklerden birçok insan bir sistem oluşturmak için biraraya gelir. Grupta temel olarak müşterinin alan uzmanları ve yazılım firmasının teknik elemanları yer alır. Sistemin tasarımını ve gerçekleştirimini üstlenen yazılımcılar genellikle problem alanıyla ilgili olarak o alanın profesyonelleri kadar bilgi ve deneyime sahip değillerdir. Buna karşılık müşterinin de yazılım geliştirme konusundaki bilgisi sınırlıdır. Daha önce bir proje geliştirme ortamında birlikte çalışmamış bu insanlar arasında terminoloji eksikliği de gözönüne alındığında iletişim problemlerinin doğması kaçınılmazdır. Bu belirsizlik ortamında yazılımcının temel görevi uygulamanın temel kavramlarını hızla özümsemek, uygulama için yaşamsal önemi olan konuları diğerlerinden ayırmak ve bir gereksinim belirleme raporunu müşteriye sunmaktır.

UML, sistemlerin incelenmesi ve tasarımı için aşağıdaki modellerin kullanılmasına olanak sağlamaktadır:

1. Kullanıcı Senaryoları Modeli (kullanıcı gereksinmelerinin modellenmesi için kullanılır)
2. Sınıf Modeli (yazılımın durağan yapısının modellenmesi için kullanılır)
3. Etkileşim Modeli (senaryoların ve ileti akışının modellenmesi için kullanılır)
4. Durum Modeli (nesnelerin devingen davranışlarının modellenmesi için kullanılır)
5. Gerçekleştirim Modeli (gerçekleştirime dönük modelleme için kullanılır)
6. Yaygınlaştırma Modeli (bileşenlerin mimari yapı üzerinde nasıl dağıtıldığının modellenmesi için kullanılır)

Yukarıdaki modellerin herbirinin kurulmasında aşağıdaki UML diyagramlarından biri ya da birkaçının çizilmesi gerekir:

1. Kullanıcı Senaryosu Diyagramları (Use-case diagrams)
2. Sınıf Diyagramları (Class diagrams)
3. Nesne Diyagramları (Object diagrams)
4. Ardıl Etkileşim Diyagramları (Sequence diagrams)
5. İşbirliği Diyagramları (Collaboration diagrams)
6. Durum Diyagramları (State diagrams)
7. Etkinlik Diyagramları (Activity Diagrams)
8. Bileşen Diyagramları (Component diagrams)
9. Yaygınlaştırma Diyagramları (Deployment diagrams

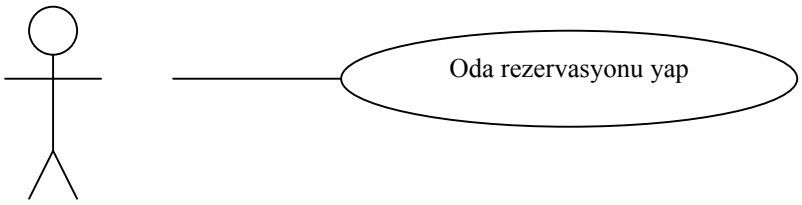
Bu bölümde kullanıcı senaryosu, sınıf, ardıl etkileşim ve durum diyagramlarına değinilecektir.

Nesneye yönelik sistem çözümleme yöntemleri, gereksinim belirleme aşamasında ağırlıklı olarak *kullanıcı senaryoları* (use-case) olarak adlandırılan bir yapıyı kullanır. Kullanıcı senaryoları ve bunların UML içindeki karşılığı olan Kullanıcı Senaryosu Diyagramları aşağıda ele alınmıştır.

Kullanıcı Senaryoları
Kullanıcı senaryoları, kullanıcı-sistem etkileşimini modellemek için kullanılan bir yöntemdir. Bunlar sistemin uç kullanıcı bakış açısından modellenmesini sağlar ve bu nedenle amaç-odaklı (goal-oriented) olarak nitelendirilirler. Kullanıcı senaryolarında uç kullanıcı ve sistem olarak iki katılımcı yer alır. Bunlardan uç kullanıcı aktör olarak adlandırılır. Aktörlerin genellikle insanlar olduğu varsayılır. Kullanıcı senaryoları:

- Bir aktör tarafından uzun zaman aralığına yayılmadan bir oturumda başlatılıp bitirilen,
- Uygulayan aktöre sonunda ölçülebilir bir fayda sağlayan,
- Sistemle etkileşimli olarak yürütülen
- Birbiriyle ilişkili

Kullanıcı ile sistem arasında etkileşimli adımların bütünü olarak tanımlanır. Her kullanıcı kural olarak sistemi kararlı bir durumda devralmalı ve kararlı bir durumda bırakmalıdır. Bir diğer deyişle kullanıcı senaryoları yarım ya da ucu açık olarak tanımlanamazlar. Aşağıda bir kullanıcı senaryosu örneği verilmiştir (Şekil 1):



Şekil 1 UML Kullanıcı Senaryosu Diyagramı

Her kullanıcı senaryosu en azından aşağıdaki bilgileri içermelidir:

- Kullanıcı senaryosunu tanımlayan kullanıcı senaryosu adı ya da kodu
- Kullanıcı senaryosunu başlatan aktör
- Kullanıcı senaryosunun amacının kısa tanımı
- Kullanıcı senaryosuna ilişkin ardışık numaralanmış işlem adımları

Buna göre bir otel odası rezervasyonunun yapılmasına yönelik kullanıcı senaryosu aşağıdaki biçimde yazılabilir:

Kullanıcı Senaryosu Adı : Oda Rezervasyonu Yap (RSV001)
Aktör : Rezervasyon Görevlisi (RG)
Amaç : Bir Otel Odasının Rezervasyonunu Yapmak
Ana Senaryo:

1. RG otel, tarih ve oda tipi bilgilerini girer
2. Sistem fiyat ve boş/dolu bilgilerini gösterir
3. RG odalardan birini seçerek işleme devam eder
4. RG rezervasyonun kimin adına yapıldığı bilgisini girer
5. RG ödeme bilgilerini girer
6. Sistem gereken rezervasyonu yapar ve bir rezervasyon numarası belirler

7. Sistem belirlediği rezervasyon numarasını RG'ne yansıtır

Yardımcı Senaryolar:

3. RG'nin belirttiği otelde istediği tarihte ve türde boş oda yoktur
    3.1. Sistem başka oda türü ve tarih seçenekleri sunar
    3.2. RG seçeneklerden birini seçer
    3.3. Devam (Ana senaryo Adım 4)

Yukarıdaki kullanıcı senaryosunun adımları *Ana Senaryo* ve *Yardımcı Senaryolar* olmak üzere ikiye ayrılmıştır. Ana senaryo, aktörle amacı arasındaki en kısa ve en sorunsuz iş akışının adımlarını içermektedir. Bu senaryo yazılırken hiç bir sorun yaşanmadığı varsayılır. Bir diğer deyişle, aktörü amacına taşıyacak daha kısa ve daha sorunsuz bir yol mevcut değildir. Bu nedenle ana senaryo, " mutlu senaryo", "güneşli gün senaryosu" gibi isimlerle de anılır. Ana senaryonun bir diğer özelliği de işin yapılmasında en sık kullanılan uygulama biçimine karşılık gelmesidir.

Uygulamada sistem çözümleyicilerin zaman zaman ana senaryonun adımlarına yoğunlaşmak yerine öncelikle çıkabilecek aykırı durumları ve sorunları yakalamak gibi sağlıksız bir yaklaşıma sapabildikleri görülmektedir. Yazılımcı, kodlama sırasında yazacağı kodun gerek hacim gerekse karmaşıklık açısından önemli bir bölümünün aykırı durumlarla ilgili olacağını bildiğinden öncelikle bu konulardaki belirsizlikleri gidermeye çalışır. Bu durumda da gereksiz ayrıntı ve felaket öngörüleriyle dolu senaryolar üremeye başlar.

Yardımcı senaryolar yazılırken önce ana senaryonun kaçıncı adımından sapıldığı belirtilir. Bu nedenle yardımcı senaryolar en dış düzeyde ardışık olarak numaralanmazlar; içerlek biçimde ve ana adımın numarasını içerecek şekilde ikili numaralama yapılır. Daha sonra sapmaya neden olan durum yazılır. Bir diğer deyişle yardımcı senaryonun bu bölümünün işletilmesi için gerekli koşul yazılır. Genel olarak bir yardımcı senaryo üç şekilde sonlanabilir: *Ana senaryoya dönüş, Vazgeç, Dur*. Dur komutu, kullanıcı senaryosunu olumlu biçimde sonlandırmak için kullanılır. Bu durumda kullanıcı ana senaryoya dönmeden ancak onun kadar kolay bir yoldan giderek işlemi tamamlamış demektir. Vazgeç komutunda ,ise başarısızlık sözkonusudur ve bir nedenle amaç işlem gerçekleşememiştir. Yardımcı senaryoların kullanılması mutlaka bir sorun olduğu anlamına da gelmez; bir işi yapmanın değişik yöntemleri olabilir.

Yardımcı senaryoların gereksiz yere karmaşıklaşması, kullanıcı senaryosunun hızla bir program kodu görünümüne bürünmesine neden olur. Yarı program görünümündeki çok sayıda iç içe dallanmalar içeren kullanıcı senaryoları alan uzmanlarının kolay değerlendirebileceği yapılar değildir ve alan uzmanı ile sistem çözümleyici (system analyst) arasındaki iletişimi baltalarlar.

Gerçek projelerde kullanıcı senaryosu yazılması sanıldığından zordur. Bu zorluğun nedenleri şöyle sıralanabilir:

- Neyin kullanıcı senaryosu olduğu ve neyin olmadığının saptanması ve iş süreçleri (business process) ilekullanıcı senaryolarının birbirine karıştırılması
- Büyük projelerde sayısı hızla artan kullanıcı senaryolarının yönetilmesindeki güçlük
- Kullanıcı senaryolarının içermesi gereken ayrıntı düzeyinin belirlenmesindeki hatalar

Bu nedenlerin pek çoğu kullanıcı senaryosu kavramının sağlam bir matematiksel temele sahip olmamasından kaynaklanmaktadır. Kullanıcı senaryoları ile ilgili yaygın olarak kabul edilmiş kural ve kavramlar yoktur. Genel prensipler ve deneyimlere göre, her kullanıcı senaryosu sistemin bir kullanıcı NE yapacağını tarif eder; ne

yapılacağının tarifi içinde NASIL sorusunun cevabı yer almaz. Dolayısıyla gerçekleştirime yönelik ayrıntıların kullanıcı senaryosunun içine sızmasına izin verilmez. Sistemin temel yetenekleri ve sınırları bu senaryo kapsamında belirlenirken kullanıcı arayüzünün konuşulmaması gerekir.

Kullanıcı Senaryolarının Yönetimi

Kullanıcı senaryolarının hazırlanmasında ve yönetilmesinde dikkat edilecek noktalar aşağıda tanımlanmıştır:

- Sistemin aktörleri belirlenir. Aktör bir sistemde işin yapılması için talimat veren ya da başvuruda bulunan kiş değildir. Aktör bizzat bilgisayarın tuşlarına basan kişidir. Aktör, bilgisayar programını başlatıp bitiren bir başka program da olabilir.
- Her aktörün sistemi kullanarak NE yapmak istediği saptanır. Aktörün her yapmak istediği iş için ayrı bir kullanıcı senaryosu hazırlanacaktır.
- Aktörün sistemde yapacağı her iş için en çok karşılaşılan durum ve en sık kullanılan iş görme biçimi belirlenerek bir *Ana Senaryo Özeti* yazılır. Senaryo özetlerinde numaralanmış senaryo adımları yer almaz; özetlerin uzunluğu bir iki cümledir.
- Tüm sistem için yazılan senaryo özetleri biraraya getirilerek gözden geçirilir, gereksiz olanlar atılır, eksikler eklenir, birleştirilmesine karar verilenler birleştirilir, parçalanması gerekenler parçalanır.
- Her senaryo için ana senaryonun adımları numaralanarak yazılır. Her adım "aktör şunu yapar, sistem şunu yapar" formatında ifade edilir. "Makbuz kesilir, fiş numarası girilir" biçimindeki ifadeler yanlıştır.
- Ana senaryolar tamamlandıktan sonra yardımcı ve aykırı durum senaryolarına geçilir. Büyük projelerde ana senaryodaki amacı alternatif yollarla gerçekleştiren senaryolara *yardımcı senaryo*, yalnızca sorunlu durumları ele alan senaryolara ise *aykırı durum senaryoları* denir. Yardımcı senaryolar ana senaryodan ve aykırı durum senaryoları da her ikisinden daha kapsamlı ve karmaşıktır. Proje yöneticisi süresel programda bunlara daha fazla zaman ayırmalıdır.
- Kullanıcı senaryoları bu aşamada tekrar gözden geçirilir. Gereken ekleme, çıkarma, düzeltme, birleştirme işleri yapılır. Geliştirilen senaryonun organizasyondaki iş süreçlerine cevap verip vermediğinden emin olunması gereklidir.
- Kullanıcı senaryoları yazılırken önce genişlemesine, sonra derinlemesine ilerlenmelidir.

Genişletilmiş Kullanıcı Senaryosu Şablonu

Çoğu kez, geniş ölçekli projelerde kullanılan kullanıcı senaryosu şablonunun daha değişik bilgiler ve proje yönetimine yardımcı olacak ek alanlar içermesi gerekmektedir. Bu durumda genişletilmiş kullanıcı senaryosu şablonu adı verilen şablonlardan biri kullanılır.

Şablon dört ana bölümden oluşur. "Tanıtım Bilgileri"nde senaryonun adı ve amacının dışında senaryoyu yazan ve onaylayan kişilerin adları, görevleri, senaryonun çalışabilmesi için gereken koşullar, senaryo çalıştıktan sonra sağlanması gereken koşullar, senaryo ile ilgili notlar, kısıtlamalar ve riskler gibi bilgiler yer alır. İkinci bölümde senaryodaki ana, yardımcı ve aykırı durum senaryolarının sadece adları yazılır. Üçüncü bölüm ise ikinci bölümde adı geçen senaryoların numaralanmış işlem

adımları yer alır. Dördüncü bölüm, işlevsel olmayan gereksinimlerin yazıldığı bölümdür. Şablonu detayları aşağıda verilmiştir:

Bölüm 1: Tanıtım Bilgileri

1. Senaryo adı
2. Amaç
3. Senaryoyu yazan takım lideri ve üyeleri
4. Önkoşullar
5. Kararlı durum koşulları
6. Riskler, notlar, kısıtlamalar
7. Senaryoyu tetikleyen olaylar
8. Aktör
9. Yardımcı aktörler

Bölüm 2: Senaryo Adları

1. Ana senaryonun adı
2. Yardımcı senaryoların adları
3. Aykırı durum senaryolarının adları

Bölüm 3: Senaryo Adımları (her bir senaryo için ayrı ayrı)

1. Senaryo adı
2. Senaryoyu tetikleyen olaylar
3. Senaryonun adımları
4. İş kuralları (gerekirse)
5. Rskler, notlar, kısıtlamaar (gerekirse)

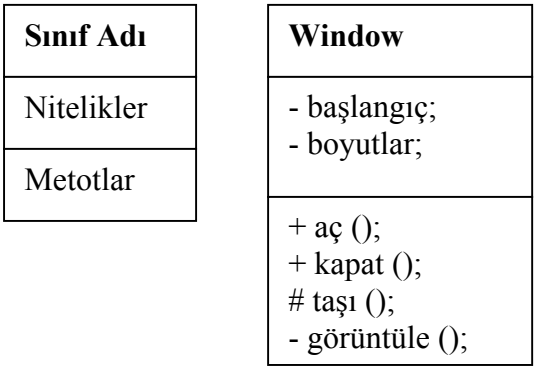
Bölüm 4: İşlevsel Olmayan Gereksinimler

1. Öncelik
2. Performans hedefleri
3. Kullanım sıklığı
4. Kullanıcı arayüzü
5. Konum (dağıtımlı sistemler için, gerekirse)

Sınıf Diyagramları

Sınıf (class), aynı metotları (işlev), aynı ilişkileri ve aynı anlamı paylaşan nesneler topluluğunun ortak tanımıdır. Sınıflar, yazılımın durağan (static) yapısının tanımlanmasında kullanılırlar.

UML, sınıfları görsel olarak üçe bölünmüş dikdörtgenlerle tanımlar. Dikdörtgenin birinci bölümüne sınıfın adı, ikinci bölümüne içerdiği niteliklerin (attributes) adı, üçüncü bölüme de içerdiği metotların (methods) listesi yazılır (Şekil 2).



Şekil 2 UML Sınıf Diyagramının Yapısı ve Bir Örnek

Sınıf diyagramında yer alan nitelik ve metot isimlerinin önünde aşağıda sıralanan bezemeler (adornments) kullanılabilir.

- Özel (-) : Nitelik ya da metota sınıf dışından erişim engellenmiştir (private)
- Korumalı (#) : Nitelik ya da metota erişim sınırlandırılmıştır (protected)
- Genel (+) : Nitelik ya da metot genel kullanıma açıktır (public)

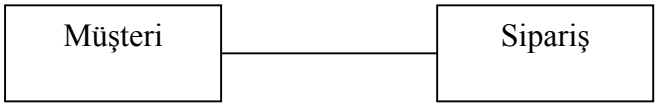
Sınıf diyagramları kendi başlarına son derece yalın yapılardır ve görsel olarak çizilmeleri, ancak aralarındaki ilişkilerle birlikte incelendiklerinde anlamlıdır. UML içerisinde sınıflar arasında 4 değişik tür ilişki tanımlanabilir:

- Bağıntı ilişkisi (Association)
- Genelleme ilişkisi (Generalization)
- Bağımlılık ilişkisi (Dependency)
- Gerçekleştirim ilişkisi (Realization)

Nesneye yönelik sistem çözümleme ve tasarım sırasında nesneler arasındaki ilişkilerin tanımlanmaması düşünülemez. Bilindiği gibi nesneye yönelik programlama, nesnelere ileti göndererek programlama biçiminde de açıklanmaktadır. Bir diğer deyişle nesneye dayalı sistemler, nesneler arasında işbirliği kavramı üzerine oturtulmuştur. İşbirliği ise iletişim gerektirir. Sınıf diyagramlarında, aralarında ilişki bulunmayan sınıflar arasında iletişim öngörülmemiş demektir. Dolayısıyla, sınıflar arası ilişkiler aynı zamanda sınıflar arası iletişim kanallarını da tanımlar. Aşağıda İlişki türleri, anlamları ve gösterimleri açıklanmaktadır.

Bağıntı İlişkisi

Bağıntı ilişkisi, bir sınıfa ait nesnelerin diğer sınıfa ait nesnelere nasıl bağlandığını tanımlar ve sınıflar arasında düz bir çizgi ile gösterilir (Şekil 3).

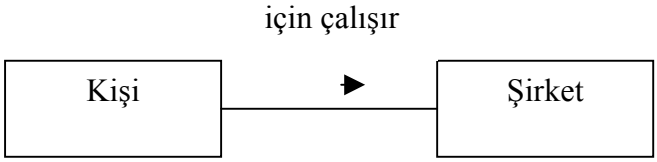


Şekil 3 Bağıntı İlişkisi

Bağıntı türü ilişkiler için tanımlanmış bezeme türleri aşağıda sıralanmıştır:

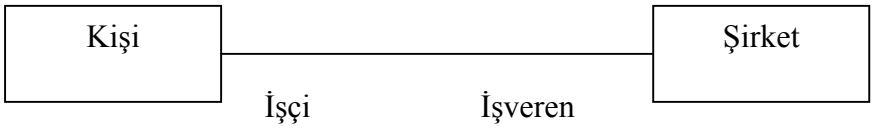
1. Bağıntının adı
2. Sınıfın bağıntıdaki rolü
3. Bağıntının çokluğu

Bağıntı adı, iki sınıf arasındaki ilişkinin küçük bir açıklamasıdır. Bu açıklama yazılırken gerekiyorsa yön bilgisi de içi dolu küçük bir üçgenle ayrıca gösterilebilir (Şekil 4).

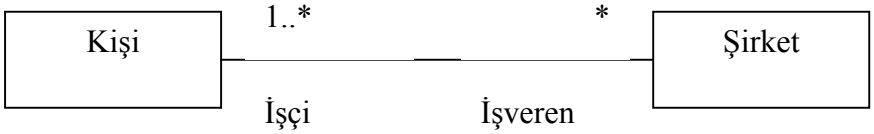


Şekil 4 Bağıntı İlişkisinde Bağıntı Adı ve Yönü

İki sınıfın nesneleri arasındaki ilişkide her sınıfın nesnesinin üstlendiği rolün de belirtilmesi, sınıf diyagramlarının daha kolay anlaşılmasını sağlar (Şekil 5).



Şekil 5 Bağıntı İlişkisinde Roller

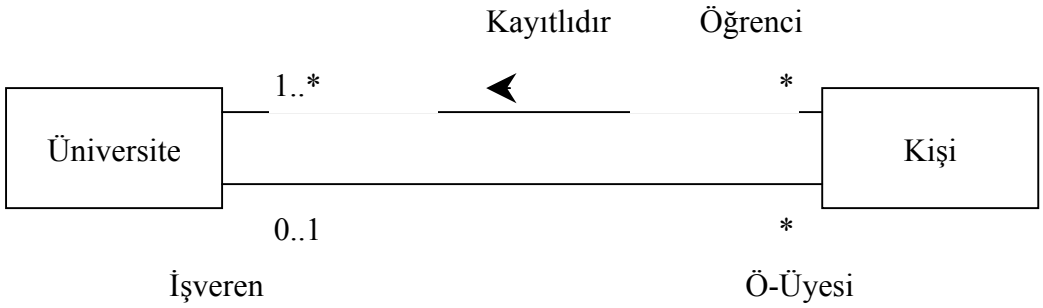


Şekil 6 Bağıntı İlişkisinde Çokluk Gösterimi

Yukarıdaki gösterimde işçi-işveren ilişkisinde bir şirketin en az bir işçisi olduğu (1..*), bir işçinin ise 0 ya da herhangi bir sayıda (*) şirkette işçi olarak çalışmış olabileceği ifade edilmektedir. Kullanılabilecek çokluk bezemeleri aşağıda sıralanmıştır:

Çokluk Bezemesi	Açıklama
1	Yalnızca 1 (herhangi bir sayma sayısı da kullanılabilir)
0..1	Yalnızca 0 ya da 1
M..N	En az M, en çok N
*	0 ya da daha çok
0..*	0 ya da daha çok
1..*	1 ya da daha çok

İki sınıf arasında yalnızca tek bir bağıntı çizilmesi gibi bir kısıt yoktur.

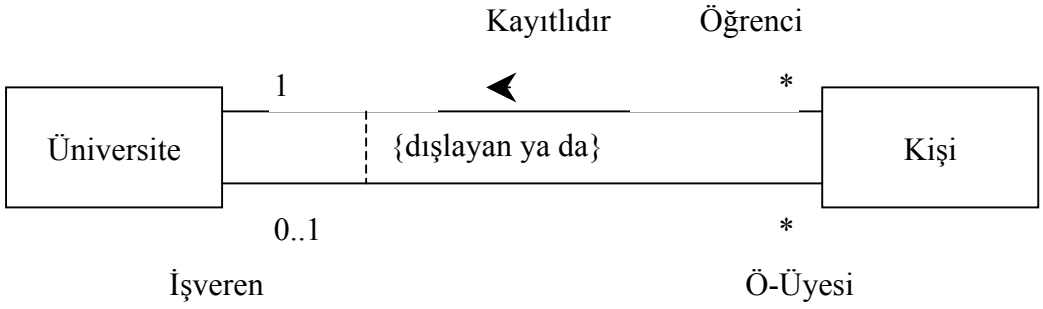


Şekil 7 Üniversite-Kişi Bağıntısında Çoklu Bağıntı Tanımı

Yukarıdaki gösterim, şu şekilde okunmalıdır:

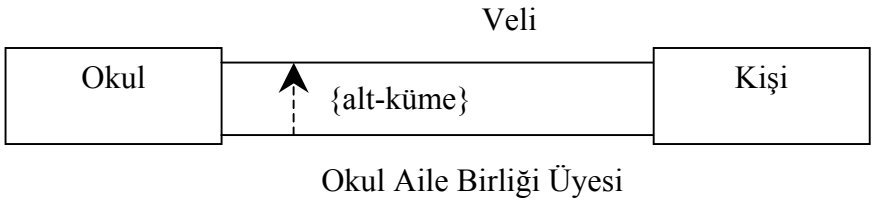
- Kişiler ile üniversite arasında bir ilişki vardır
- Bu ilişkide bazı kişiler öğrenci, bazı kişiler ise öğretim üyesidir. Bir diğer deyişle Kişiler iki gruba ayrılmaktadır
- Üniversite, öğretim üyesinin işverenidir, öğrencilerin işvereni değildir
- Her öğrenci bir üniversiteye kayıtlıdır
- Her öğretim üyesi en çok bir üniversiteye bağlıdır. Hiçbir üniversiteye bağlı olmayan öğretim üyesi de olabilir

İki sınıf arasında birden fazla bağıntı yukarıdaki gibi tanımlandığında bu iki bağıntı arasında bir etkileşim sözkonusu olmaz. Verilen örnekte bir kişi üniversite ile hem öğrenci hem de öğretim üyesi olarak ilişki kurmuş olabilir. Bu istenmiyorsa çizim aşağıdaki şekilde değiştirilmelidir (Şekil 8).



Şekil 8 Üniversite – Kişi Bağıntısı

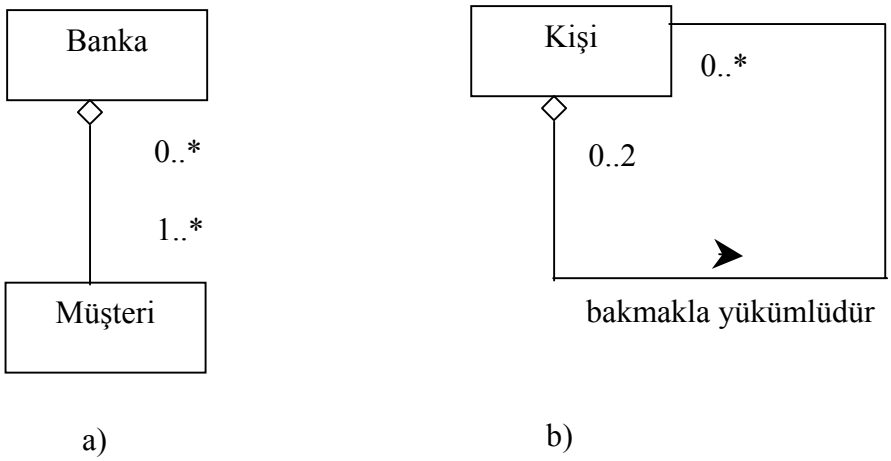
İki bağıntı arasında bir kısıt ya da ilişki tanımlamak için bağıntılar kesikli bir çizgi ile birleştirilir ve bu çizginin yanına { } parantezleri arasına gerekli açıklama yazılır. Üniversite-Kişi bağıntısında kullanılan *dışlayan ya da* (exclusive or) kısıtı, bir kişinin ya öğrenci ya öğretim üyesi olabileceğini, iki rolü birden üstlenmesinin mümkün olmadığını göstermektedir.
Bağıntılar arasında tanımlanan kısıta bağlı olarak bir nesne her iki bağıntıda da yer alabilir (Şekil 9).



Şekil 9 Okul-Veli Bağıntısı

Okul-Veli bağıntısında veli olarak yer alan kişiler arasından bazıları okul aile birliği üyesidir. Okul aile birliği kümesi, veli kümesinin bir altkümesi olarak tanımlandığından, her okul aile birliği üyesi aynı zamanda veli kümesinin de elemanıdır. Bu çizime göre veli olmayan bir kişinin okul aile birliği üyesi olması mümkün değildir.
Yukarıda tanımlanan basit bağıntının dışında *içerim* (aggregation) ve *oluşum* (composition) bağıntıları olarak adlandırılan iki bağıntı türü daha vardır.

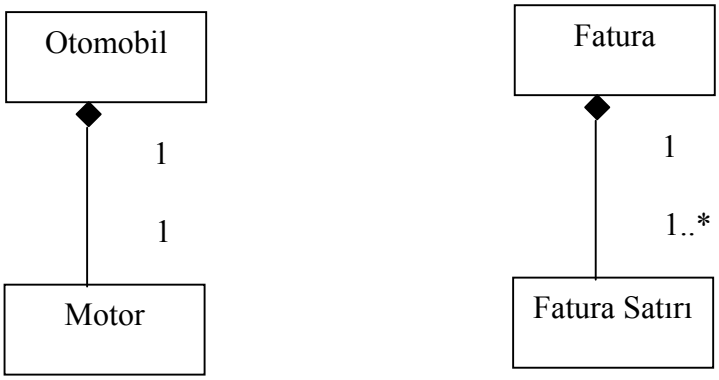
İçerim Bağıntısı (Aggregation)
İçerim bağıntısı, iki sınıf arasındaki "sahiptir" veya "içerir" türünden bağıntıları modellemekte kullanılır. Bu bağıntıda bir sınıfın nesnesi, diğer sınıfın nesnesi tarafından sahiplenilmektedir. İçerim bağıntısı, içeren (sahiplenen) sınıf tarafına yerleştirilen boş bir eşkenar dörtgen bezemesi ile gösterilir (Şekil 10).



Şekil 10 İçerim Bağıntıları

Yukarıdaki çizimde iki içerim örneği yer almakta olup, birincisinde bankanın müşterilere sahip olduğu gerçeği modellenmiştir. İkincisinde ise ebeveynlerin bakmakla yükümlü oldukları çocukları sahiplendikleri görülmektedir.

Oluşum Bağıntısı (Composition)
Oluşum bağıntısı tüm bağıntılar içinde en güçlü olanıdır ve parça-bütün ilişkilerini modellemekte kullanılır. Bu tür bağıntılar "bütün" tarafındaki sınıfa bitişik olarak yerleştirilen içi dolu eşkenar dörtgen ile gösterilir (Şekil 11).



a) b)

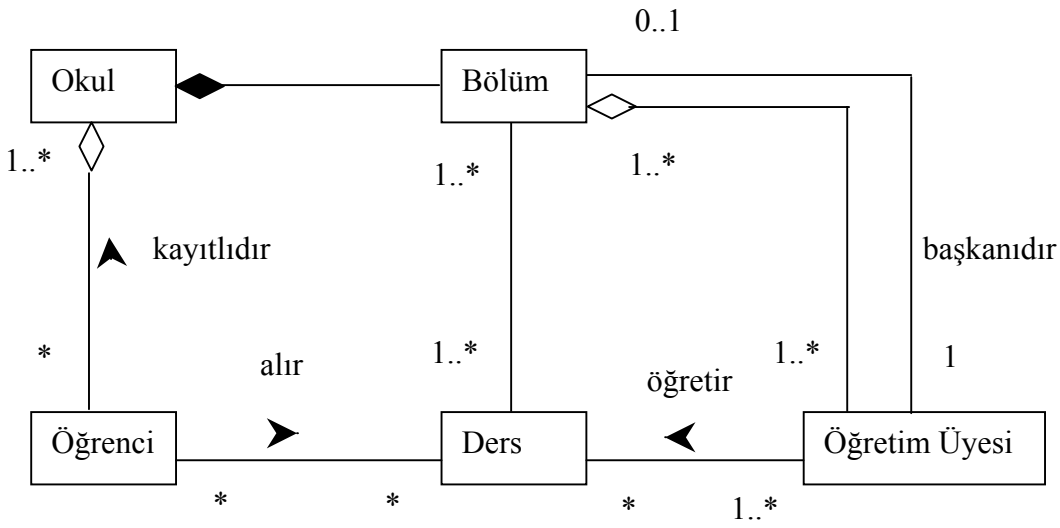
Şekil 11 Oluşum Bağıntıları

Uygulamada oluşum ve içerim bağıntılarının kimi durumlarda birbirine karıştırıldığı görülmektedir. Bunun nedeni, oluşum bağıntısının aslında içerim bağıntısının yalnızca daha güçlü bir biçimi olmasıdır. Bir başka deyişle her oluşum bağıntısı aynı zamanda bir içerim bağıntısıdır. Belirleyici fark ise iki bağıntı türünün nesneleri birbirine bağlama gücü arasındaki farktır. Buna göre iki bağıntı türü arasındaki farklar aşağıda sıralanmıştır:

- Oluşum bağıntısında her *parça* aynı anda sadece tek bir *bütüne* ait olabilir. Her motor tek bir otomobil tarafından kullanılabilir ya da her fatura satırı sadece tek bir faturaya ait olabilir. Bu nedenle oluşum bağıntısında *bütün* tarafı *çokluk değeri* daima 1 olmak zorundadır. İçerim bağıntısında ise bir parçanın birden fazla bütün tarafından paylaşımı söz konusu olabilir. Örneğin bir çocuğa 0, 1 ya da 2 ebeveyn bakmakla yükümlüdür; bir müşteri birden fazla bankanın müşterisi olabilir ya da bir danışman birden fazla firmanın danışmanı olabilir.
- Oluşum ilişkisinde *parçanın* yaşam süresi doğrudan *bütünün* yaşam süresine bağlıdır. *Bütün* nesnesi yaratılmadan *parça* nesnesi yaratılamaz ve *bütün* tarafındaki nesne silindiğinde *parça* nesneler de onunla birlikte silinmelidir. Örneğin fatura nesnesi silindiğinde o faturaya ilişkin satırlara karşılık gelen nesnelerin varlıklarını sürdürmeleri anlamsız olur. Ancak banka nesnesi silindiğinde o bankanın içerdiği müşteri nesnelerinin otomatik olarak silinmesi gerekmez. Büyük olasılıkla bir banka müşterisinin birden fazla bankada hesabı vardır ve banka nesnelerinin yaşam süreleri ile müşteri nesnelerinin yaşam süreleri arasında belirleyici bir bağlantı yoktur.

Oluşum ve içerim bağıntıları ile basit bağıntılar arasındaki farklar daha çok gerçekleştirime yönelik noktalarda önemlidir. Aralarında oluşum bağıntısı tanımlanmış nesnelere karşılık gelen veritabanı tablolarında, ana kaydı içeren tablodan bir kayıt silindiğinde, ayrıntı kayıt tablosundan da ilgili tüm kayıtların otomatik olarak silinmesini sağlayan veritabanı yordamlarının ya da kısıtlarının yazılması gerektiğinin modelde gösterilmesi durumu buna örnektir. Benzer biçimde, oluşum bağıntısı ana kayıt yaratılmadan ayrıntı kayıtların yaratılmasını engeller. Modellemenin ilk aşamalarında gerçekleştirime yönelik bu tür ayrıntılar önemsizdir ve yalnızca basit bağıntılar kullanılır. Model olgunlaşıp, gerçekleştirime yaklaşıldıkça

basit bağıntı, içerim bağıntısı ve oluşum bağıntısı ayırımına gidilir. Aşağıda çeşitli bağıntı türlerinin bir arada kullanıldığı bir örnek yer almaktadır (Şekil 12).



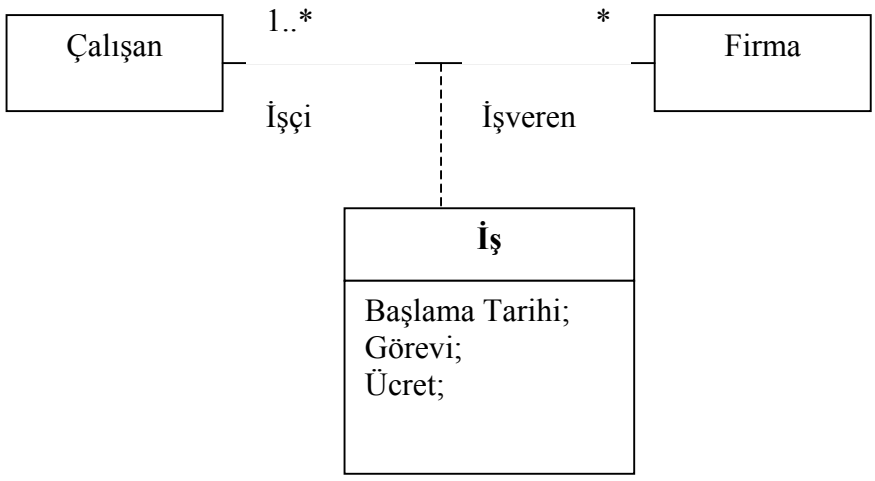
Şekil 12 Bağıntı Örneği

Yukarıdaki bağıntıda aşağıdaki ilişkiler yer almaktadır:

- Her okulun sıfır ya da daha fazla öğrencisi vardır
- Her okulda bir ya da daha fazla bölüm vardır
- Her bölüm bir tek okula bağlıdır
- Her bölüm bir ya da daha fazla sayıda ders açmaktadır
- Her bölüm bir ya da daha fazla öğretim üyesine sahiptir
- Her bölümün öğretim üyelerinden seçilmiş bir başkanı vardır
- Her öğretim üyesi bir ya da daha fazla bölümde görev yapmaktadır
- Her öğretim üyesi sıfır ya da daha fazla sayıda ders vermektedir
- Her ders bir ya da daha çok bölüm tarafından açılır
- Her dersi sıfır ya da daha fazla sayıda öğrenci almaktadır
- Her derse en az bir tane öğretim üyesi atanmıştır
- Her öğrenci bir ya da daha fazla sayıda okula kayıtlıdır
- Her öğrenci sıfır ya da daha fazla sayıda ders alabilir
- Her öğretim üyesi ya tek bir bölümün başkanıdır ya da hiç bir bölümün başkanı değildir. Bir diğer deyişle, bir öğretim üyesi birden fazla bölümün başkanı olamaz ve her bölümün bir tek başkanı vardır.

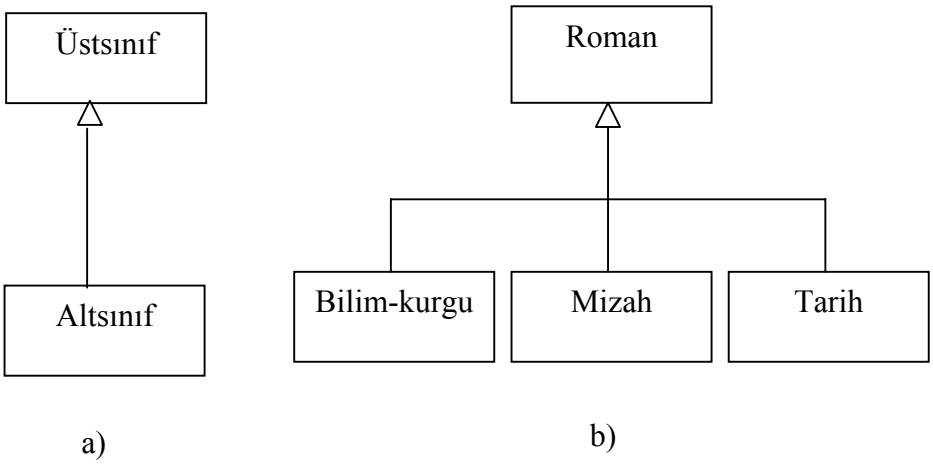
Sınıf diyagramları, sınıflar arası bağıntıları da içerecek biçimde çizildiklerinde son derece güçlü ve yoğun bilgi içeren bir gösterim biçimi oluştururlar. Öyle ki yalnızca yukarıdaki çizime bakarak ortaya çıkacak yazılımın yeteneklerinin sınırlarını algılayabilmek olanaklıdır. Örnek olarak “bir bölümün öğrencilerinin aldığı farklı derslerin listelenmesi” isteğini ele alalım. Yukarıdaki çizime göre bu isteği yerine getirecek bir yazılım gerçekleştirilmesi olanaksızdır. Çünkü çizimde “bölüm” ile öğrenci ilişkisi kurulmadığından “bölüm öğrencisi” kavramı yer almamaktadır. Öğrenci, okulun öğrencisidir ve bölümle hiç bir üyelik ilişkisi yoktur.

Bağıntı Sınıfları (Association Classes)
İki sınıfın nesneleri arasında bir bağıntı kurulduğunda her iki sınıfa da ait olmayan ancak bağıntının kendisine ait nitelikler ortaya çıkabilir. Örneğin, çalışan-firma bağıntısında çalışanın işe başlama tarihi, görevi, ücreti gibi nitelikler ancak çalışan o firmada işe girmişse tutulması anlamlı niteliklerdir. Arada hiç bir iş ilişkis iyokken firma ya da çalışan sınıfına ait nesnelerin bu nitelikleri barındırması gerekmez. Bağıntının kendisine ilişkin nitelikler bağıntı nesnelerinde tutulur. Bağıntı sınıfları, bağıntıya kesikli çizgi ile bağlı olarak gösterilir (Şekil 13).



Şekil 13 Bağıntı Sınıfı

Genelleme İlişkisi
Bu ilişki, bir nesnenin bir diğerinin özel bir türü olduğu gerçeğini modellemek için kullanılır. Bu ilişkide nesneler genelden özele doğru sıradüzensel bir dizilim içinde yerleştirilir. Örneğin, bilim-kurgu, mizah ve tarih romanları, roman genel kategorisi içinde yer alır. Buna karşılık roman da şiir ve öykü gibi edebi eserler kategorisinin bir elemanıdır. Genelleme ilişkisi sıradüzensel bir yapı oluşturduğundan sınıflar arasında *üstsınıf* (superclass) ve *altsınıf* (subclass) ayırımının yapılmasını gerektirir. Altsınıflar, içi boş bir ok aracılığıyla üst sınıfa bağlanırlar (Şekil 14).

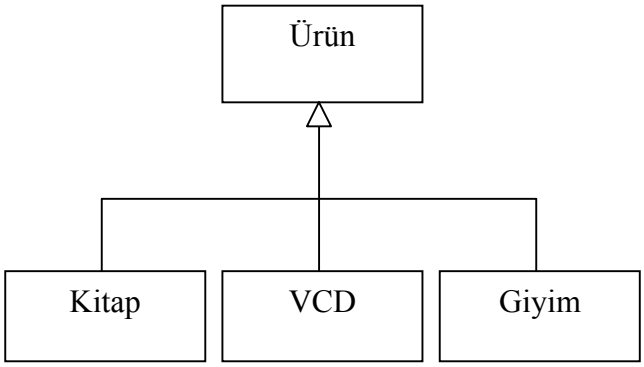


Şekil 14 Genelleme İlişkisi

Genelleme ilişkilerinin belirlenmesi temelde sınıflama (classification) kavramına dayalıdır. Sınıflandırma kavramı, nesnelerin ortak özelliklerinin belirlenmesine dayanır. Bu nedenle farklı bakışlar farklı sınıflandırmalar ortaya koyabilir. Bunlar içinde en anlamlı olan, amaca en fazla hizmet edendir ve bunun ortaya konması her zaman o kadar kolay olmayabilir. Programcı kullandığı nesnelerin temel özelliklerini saptayarak;

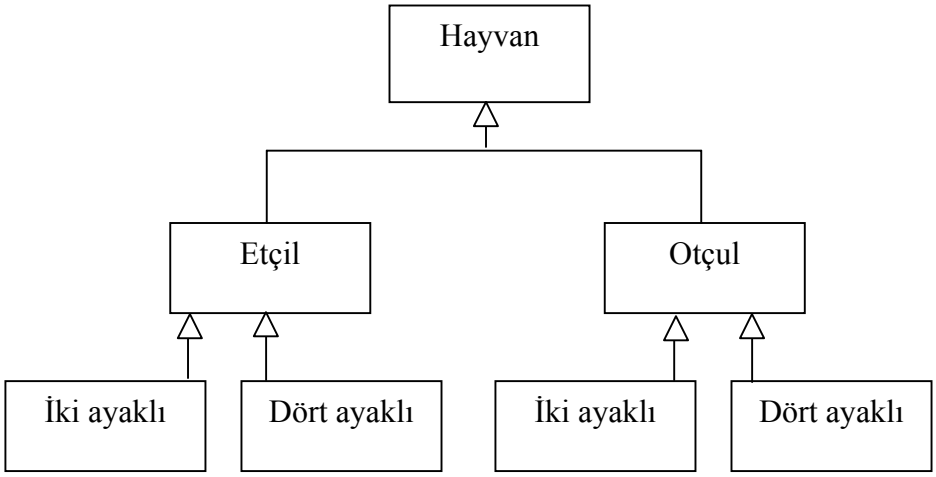
- altsınıflardan önce üstsınıflara yönelik metotları belirler, sınıfın tümüne yönelik genel amaçlı kod kesiminin neler içermesi gerektiğini saptar. Üstsınıf metotları iyi düzenlendiğinde altsınıflara yapılacak ekleme ve değişikliklerden etkilenmezler
- Sistemdeki sınıfların sayısında ve içeriğinde gelecekte gerekli olacak değişiklikler için önceden belirlenmiş ekleme ve değiştirme noktaları hazırlar.

Bu durum aşağıdaki örnek üzerinde açıklanmıştır (Şekil 15):



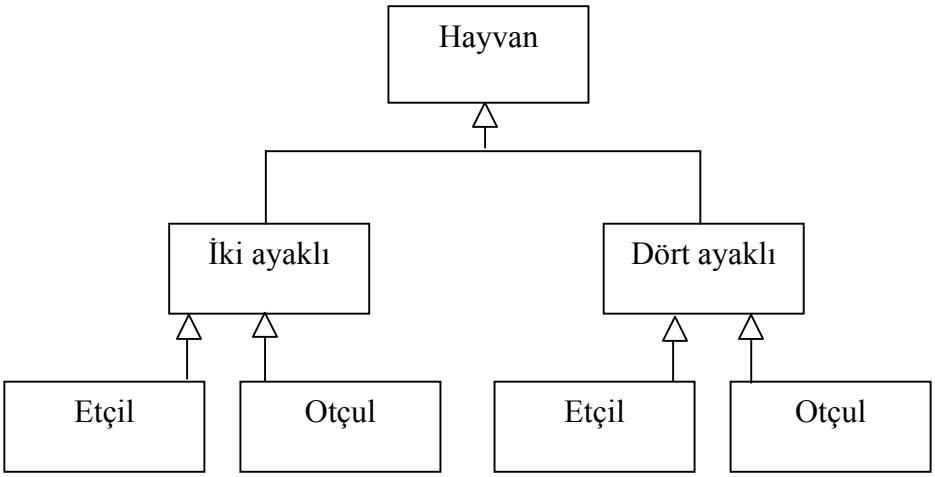
Şekil 15 Genelleme İlişkisi Örneği

Yukarıdaki örnekte bir mağazada satılan ürünler görülmektedir. Üç farklı ürün grubu öngörülmüştür ve her grubun kendine özgü nitelikleri vardır. Kitapların yazar adları ve sayfa sayıları, VCD filmlerin uzunlukları, yönetmen ve oyuncu listeleri, giyim ürünlerinin bedenleri ve renkleri vardır. Ancak temel bir ürün kavramından yola çıkılarak, belirli uygulamaların ürünler arası farklardan etkilenmeyecek biçimde yazılması sağlanabilir. Örneğin muhasebe açısından yalnızca ürünün satış bedeli önemlidir. Bu nedenle, muhasebe için gereken metotlar altsınıflar yerine üstsınıflar içine yerleştirilir ve tüm ürünler tarafından ortaklaşa kullanılır. Muhasebe uygulamasında yalnızca ürün sınıfı kullanılır ve diğer alt ürün sınıf adları yer almaz. Böylece mağaza gelecekte meyve-sebze satma kararı verdiğinde ürün sınıfına yeni bir altsınıf eklenecek eolmakla birlikte muhasebe uygulamasında bir değişiklik gerekmeyecektir.
Sınıflandırmanın kavramsal zorluğunun bir nedeni de aynı nesnelerin bakış açısına ve kullanım amacına bağlı olarak pek çok değişik biçimde sınıflandırılabilmesidir (Şekil 16 - 17).



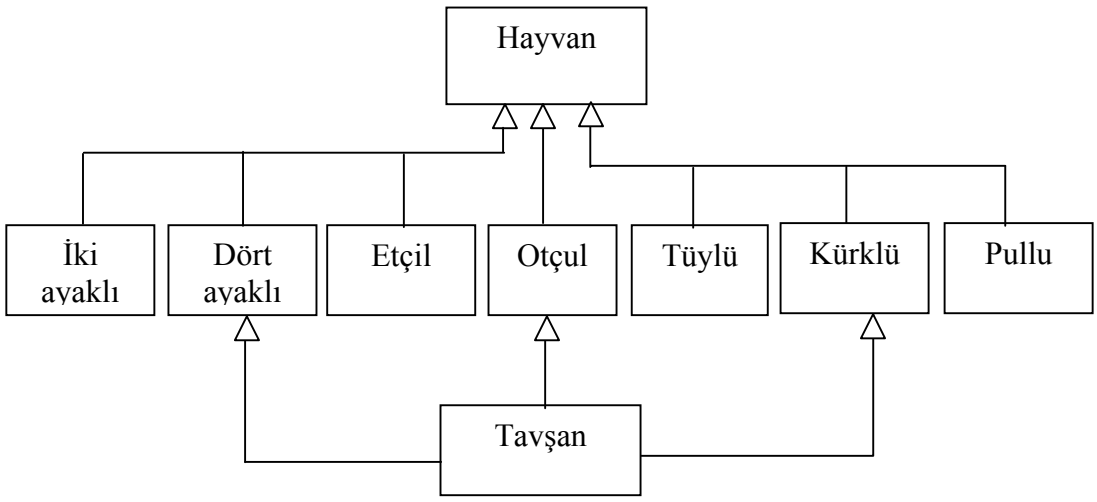
Şekil 16 Genelleme İlişkisi

Genelleme ilişkisi nesneye yönelik programlama dillerinde kalıtım mekanizması ile gerçekleşir. Dolayısıyla iki ve dört ayaklı sınıflarına ilişkin kodun tekrarlanması gerekir. Ayak sayısı ve beslenme biçimi kıstasları farklı sırada uygulansaydı aşağıdaki sıradüzeni elde edilecekti.



Şekil 17 Genelleme İlişkisi

Bu sınıflamaya hayvanların dış görünüm özelliklerinin de eklendiğini varsayarsak (tüylü, kürklü, pullu vs) şemanın ne kadar genişleyeceği görülebilir. Bu soruna çözüm getirmek üzere çoklu kalıtım (multiple inheritance) kavramı geliştirilmiştir. Bu kavram, bir sınıfın birden fazla üstsınıfa bağlanabilmesini sağlar. Şekil 18'de çoklu kalıtım tekniği kullanılmış ve Tavşan sınıfı üç ayrı sınıftan türetilmiştir.

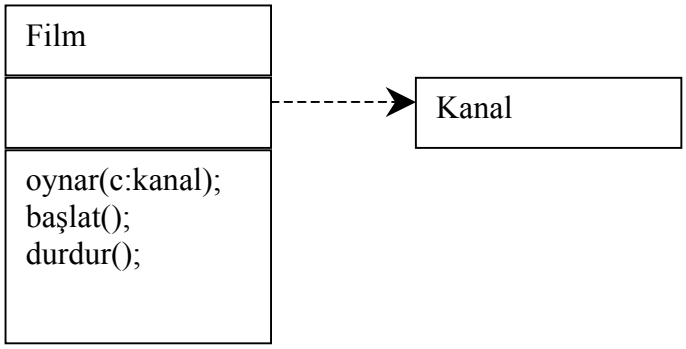


Şekil 19 Çoklu Kalıtım

Çoklu kalıtım, programlama dillerinde desteklenmesi zor ve ve kullanımı karmaşık bir mekanizmadır. Bu nedenle Java dahil nesneye yönelik kimi programlama dilleri bu özelliği desteklemezler. Bu dillerde yukarıdaki yapının gerçekleştirimi özel teknikler gerektirir.

Bağımlılık İlişkisi

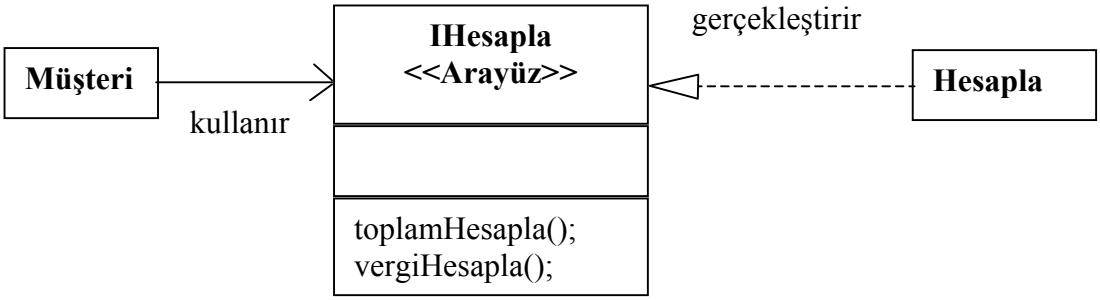
Bağımlılık ilişkisi, "kullanır" ilişkisinin modellemesinde kullanılır. Kullanır ilişkisinde *bağımlı* (dependent) ve *bağımsız* (independent) ayırımı söz konusudur. Eğer bir sınıf, diğer sınıf metotlarından en az birisinde parametre olarak kullanılıyorsa iki sınıf arasında bağımlılık ilişkisi vardır (Şekil 20). Bağımlılık ilişkisi, bağımlı sınıftan bağımsız sınıfa doğru kesikli çizgi ile çizilen bir okla gösterilir.



Şekil 20 Bağımlılık İlişkisi

Verilen örnekte film sınıfı hangi kanalda gösterimde olduğuna ilişkin bir metota sahiptir. Bu nedenle *Film* sınıfı, *Kanal* sınıfına bağımlıdır. *Kanal* sınıfında yapılacak bir değişiklik *Film* sınıfında değişiklik yapılmasını gerektirirken tersi söz konusu değildir.

Gerçekleştirim İlişkisi
Başka kullanım alanları da olmakla birlikte gerçekleştirim ilişkisi en çok kullanıcı arayüzlerinin (user interface) modellenmesinde kullanılır. Arayüz, tanımı gereği yalnızca metot adlarını ve bunların parametrelerin içerir. Metotların gerçekleştirimi ise genellikle farklı sınıflar içerisine yerleştirilir. Program yazarken, yalnızca arayüzlerin kullanılması ve arayüzü gerçekleştiren sınıfın diğer sınıflardan ayrı tutulması, yazılımın geliştirilmesi ve bakımında (maintenance) önemli kolaylık sağlar. Gerçekleştirim ilişkisi kesikli bir çizginin ucuna yerleştirilen içi boş bir üçgenle gösterilir (Şekil 21).

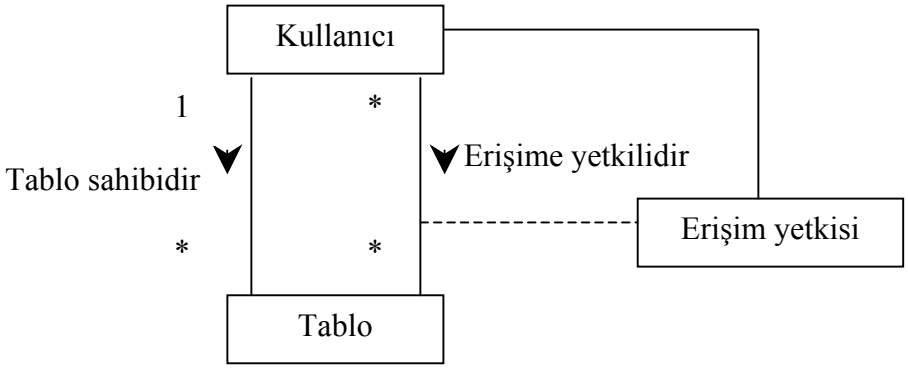


Şekil 21 Gerçekleştirim İlişkisi

Yukarıdaki örnekte *IHesapla* adı verilen ve diğer sınıfların kullanımına açılmış bir arayüz sınıfı tanımlanmıştır. Bu sunufta yer alan metotların gerçekleştirimi ise *Hesapla* sınıfı tarafından üstlenilmiş durumdadır. İyi yapılan bir tasarımda tüm sınıflar yalnızca arayüzde yer alan metotları kullanmalı, doğrudan Hesapla sınıfının metotlarını çağırmamalıdır. Böylece *Hesapla* sınıfının gerçekleştirmi, uygulamanın kalanını etkilemeksizin istendiği gibi değiştirilebilir. Arayüzlerin bir diğer yararı, arayüzün gerçekleştirimini yapan sınıfın kaynak kodunun (source code), sınıfı kullanan diğer programcılar tarafından erişilmesi gereğini ortadan kaldırmasıdır. Bu nedenle arayüz sınıfları, kod gizliliğini sağlamak isteyen yazılım firmaları tarafından yaygın biçimde kullanılmaktadır.

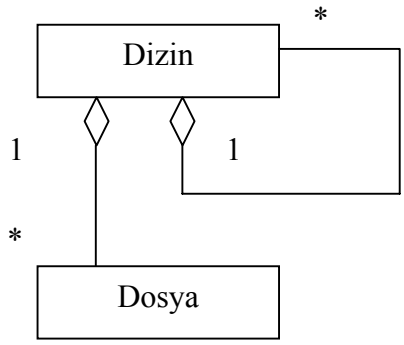
Örnekler

Örnek 1:
Örnek veritabanında her veritabanı tablosu tek bir kullanıcı tarafından sahiplenilmiştir. Bir kullanıcı birden fazla tabloya sahip olabilir. Her tablo 0 ya da daha çok kullanıcı tarafından kullanılabilir. Bir kullanıcının bir tabloyu kullanabilmesi için ya o tablonun sahibi olması ya da tablonun sahibi olan kullanıcının kendisine erişim yetkisi vermiş olması gerekir. Bu ilişkiler iki adet *sınıf* (Kullanıcı ve Tablo) ve bir *bağıntı sınıfı* (Erişim Yetkisi) kullanılarak Şekil 22'de modellenmiştir.



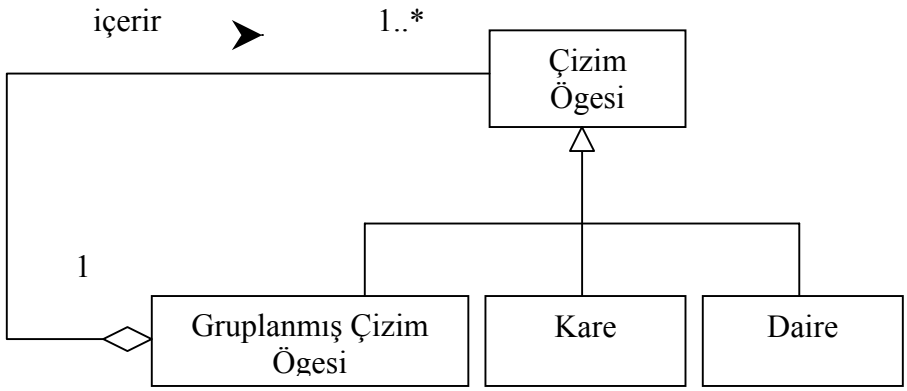
Şekil 22 Veritabanı Tablolarına Erişim Yetkisi Modeli

Örnek 2
Bilgisayarlarda kullanılan kütük yönetim (file management) sistemleri, dosyaları dizinler (folder/directory) içinde toplarlar. Her dizin içinde dosyaların yanısıra başka dizinler de (altdizinler) yer alabilir. Dizin-Dosya ilişkisine ilişkin modelleme Şekil 23'de görülmektedir.



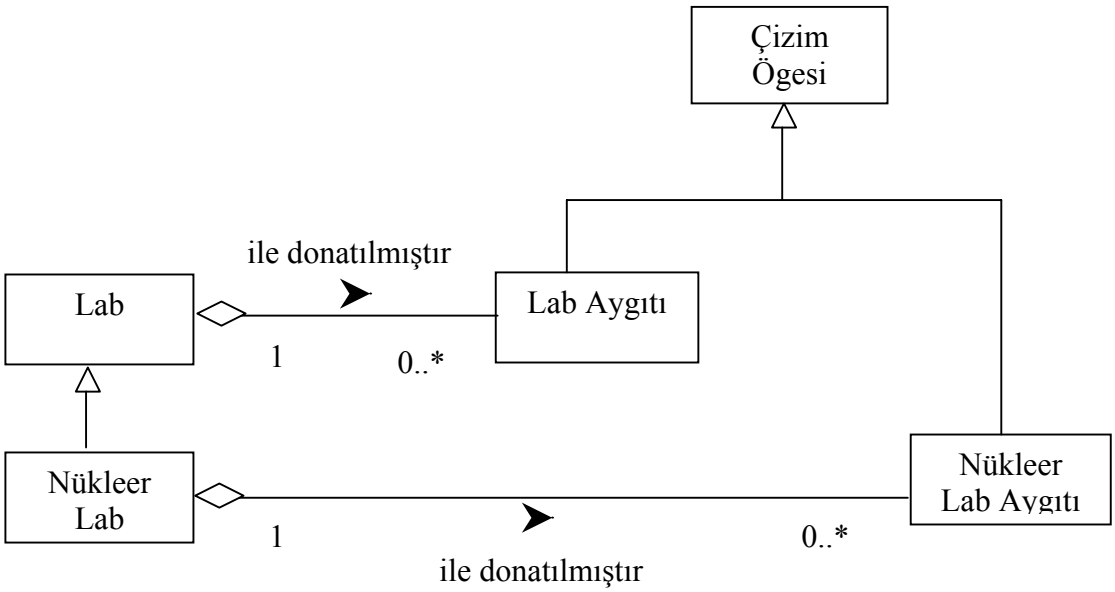
Şekil 23 Dizin – Dosya İlişkisi Modeli

Örnek 3
Bilgisayarda çizim yapılmasını sağlayacak bir program tasarlanmaktadır. Her çizimin değişik sayıda çizim ögelerinden oluşacağı varsayılmaktadır. Her çizim ögesi kare ya da daire şekillerinden biri olmak zorundadır. Ancak istenen sayıda kare ya da daire bir grup içinde toplanarak taşıma, silme, kopyalama, büyütme ve küçültme gibi işlemlerde tek bir nesne olarak kullanılabilmektedir (Şekil 24). Şekilde üstsınıf ile altsınıflar arasında yalnızca genelleme değil, aynı zamanda içerim ilişkisi de olduğu görülmektedir. Altsınıfın üst sınıfı içermesi durumu *geriye doğru içerim*, üstsınıfın altsınıfı içermesi durumu *ileriye doğru içerim* olarak adlandırılır. İleriye ve geriye doğru içerim ilişkileri özyineli (recursive) yapılar oluşturur ve bu nedenle ürün ağaçlarının ve iletişim ağı ilingilerinin (topology) modellenmesinde yoğun biçimde kullanılırlar.



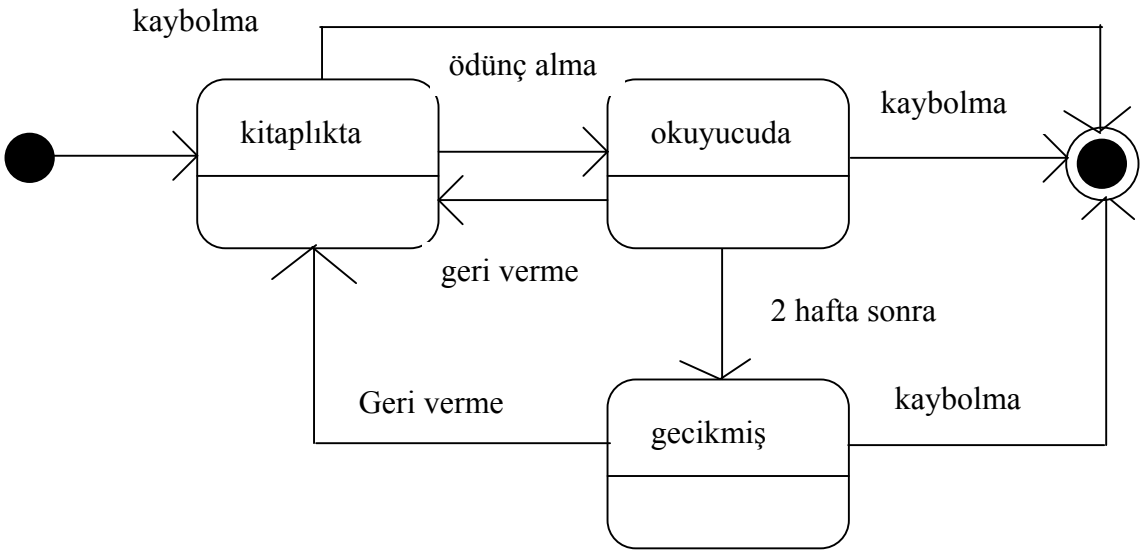
Şekil 24 Geriye Doğru İçerim İlişkisi ve Genelleme İlişkisi

Örnek 4
Bir kurumda çeşitli laboratuvarlar vardır ve bu laboratuvarlara çeşitli aygıtlar yerleştirilmiştir. Laboratuvarlardan bazıları nükleer niteliktedir ve nükleer aygıtlar yalnızca nükleer lab'larda kullanılabilmektedir. Lab – Aygıt sınıf diyagramı Şekil 25'de verilmiştir.



Şekil 25 Lab – Aygıt Sınıf Diyagramı

Durum Diyagramları
Bu diyagramlar bir nesnenin durumunun zaman içinde nasıl bir değişim gösterdiğini modellemek için kullanılır. Gerçek zamanlı sistemlerde ya da birim işlem mantığı içerisinde çalışan nesnelerde olduğu gibi, nesnenin davranışı kendisine gönderilen iletilerin yanısıra o an içinde bulunduğu duruma göre de farklılık gösteriyorsa gerçekleştirimden önce nesnenin durum diyagramının çizilmesi yararlı olur. Bu nedenle durum diyagramları projede bulunan tüm nesneler için değil yalnızca karmaşık nesneler için çizilir. Şekil 26'da bir durum diyagramı örneği görülmektedir.



Şekil 26 Kitap Nesnesi Durum Diyagramı

Verilen örnekte kütüphane sistemi içinde kullanılan bir kitap nesnesinin içinde bulunabileceği durumlar gösterilmiştir. Sistem açısından kitap nesnesinin yaratılması kitabın satınalınmasıyla başlar. Satın alınan kitap kitaplığa girdikten sonra okurlarca ödünç alınabilir. Kitabı ödünç alan okur iki hafta içinde geri getirmezse kitap gecikmiş duruma geçer. Kitap herhangi bir durumdayken kaybolursa kütüphane sistemi açısından varlığı sona erer.
Kitabın içinde bulunduğu durum, kitap nesnesi üzerinde yapılabilecek işlemleri belirlemesi açısından önemlidir. Örneğin okuyucuya verilen kitap geri dönmeden tekrar ödünç işlemi yapılamaz. Kütüphane gecikmiş bir kitabı okuyucudan istemek üzere uyarı mektubu gönderebilir. Ancak kütüphanede olan ya da iki haftasını doldurmamış kitapların kendileri ile ilgili uyarı mektubu gönderilmesine izin vermemeleri gerekir. Sonuç olarak durum diyagramları hem gerçekleştirime hem de uygulamanın mantık ve işleyişin tutarlılığına ilişkin önemli bilgiler içerirler.
Uygulamada sistem içinde yer alan pek çok nesneden ancak küçük bir kısmının bu tür davranış zenginliğine sahip olabildikleri görülmektedir. Durum diyagramları yalnızca bu tür nesneler için çizilmelidir.
Durum diyagramlarında başlangıç durumu içi dolu, bitiş durumu ise içi boş bir daire içindeki dolu bir daire ile gösterilir. Durumlar kenarları yuvarlatılmış dörtgenler ile gösterilir ve içine durumun adı yazılır. Durumlar arası geçişler okla gösterilir, geçişe neden olan olay okun üzerine yazılır.