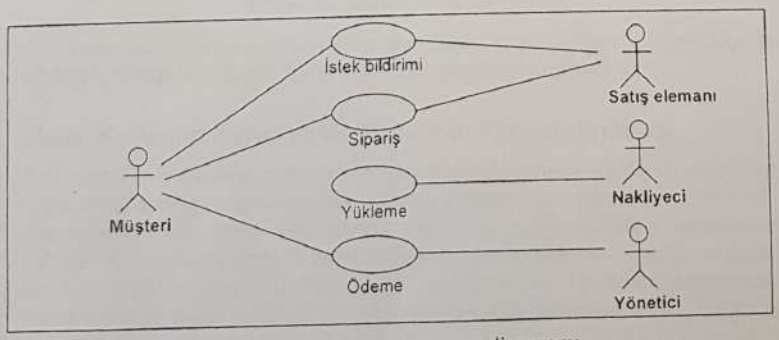
5. Kullanım Senaryoları

3. Hafta
1. Grup
Başlangıç

UML'in çözümleme yöntemi kullanım senaryolarıdır (use cases). Bir kullanım senaryosu, sistemin iç yapısını göstermeden kullanıcıya bir sonuç değeri veren, sistemin belirli bir işlevidir. Sistem bir kara kutu olarak ele alınarak sistemin yerine getirmesi gereken sorumluluklar tanımlanır. Genel olarak müşteriyle görüşülerek belirlenirler. Kullanım senaryolarının birer amacı vardır. Var olan, gözlemlenebilir ve ölçülebilir bir değeri kullanıcıya verirler. Senaryo, anlamlı bir sonuca ulaşmak için aktör ile sistem arasında gerçekleşen olaylar zinciridir. Olası senaryoların tamamı ayrı ayrı tanımlanır. Şekil-C.2, kullanım senaryosu diyagramına örnek olarak aktörleri ve kullanım senaryoları ile olan ilişkileri göstermektedir.



Şekil-C.2. Kullanım senaryosu diyagramı.

UML ile yazılım isterleri çözümlemesi kullanım senaryolarını içeren çeşitli etkinliklerle yürütülür. Şimdi bu etkinlikleri görelim.

5.1 Aktörlerin ve Kullanım Senaryolarının Bulunması

Bu etkinlik içinde gerçek hayatta karşılığı kullanıcı olan en az bir aktör tanımlanır. Daha fazla sayıdaki aktörler arasında çok az ilişki vardır. Kullanıcının sistemden bekledikleri kullanım senaryoları olarak tanımlanır. Bazen kullanım senaryolarının kapsamlarına karar verilmesinde ve ilişkide olduğu diğer kullanım senaryolarının belirlenmesinde güçlük çekilir. Bu durumda müşteri ile birlikte modelin üzerinden birkaç kez geçilerek iyileştirme sağlanır. Kullanım senaryolarının belirlenmesinde çok genel olarak temel kurallar şunlar kullanılır:

- Bir kullanım senaryosu mutlaka bir aktör tarafından başlatılmalıdır.
- Herbir kullanım senaryosu başlatıcı aktöre belirli bir amaca ulaşması için gerekli bir sonuç döndürmelidir.
- Kullanım senaryosu tam olmalı, açık kısımlar kalmamalıdır.
- Kullanım senaryosu kontrol edilebilir bir büyüklüğü aşmamalıdır.

Her aktörün sistemdeki rolü kısaca tanımlanır. Kısıtlamalar ve özel durumlar ayrıntılarıyla anlatılır. Her kullanım senaryosunun yerine getirmesi gereken işlevsellik tanımlanır, eylemler özetlenir. Kullanım senaryolarının aktörlerle nasıl ilişkilendirildiği bir bütün olarak ele alınarak model hazırlanır.

5.2 Kullanım Senaryolarının Detaylandırılması

Her kullanım senaryosunda bulunan olay akışları ayrıntıyla anlatılır. Kullanım senaryosunun nasıl başladığı, sonlandığı ve aktörlerle etkileşimi tanımlanır. Sonuçta sistem davranışı metinsel bir anlatıma dönüştürülmüş olur.

Bir kullanım senaryosu içinde başlangıç, bitiş ve çeşitli ara durumlar (state) bulunabilir. Bunların arasından en çok kullanılanı ve aktöre en uygun değeri döndüren bir tanesi "ana yol" olarak seçilir. Ana yol, bir eylemler dizisi halinde tanımlanır. Ana yoldan ayrılmaya sebep olan bir olay aykırı durum (exception) olarak nitelendirilir. Aykırı durumlar geri kazanılır ya da geri kazanılamaz olabilir. Eğer aykırı durumların açıklaması kısa ise ana yol tanımlaması sırasında açıklanır. Daha uzun ise bu açıklama ayrı bir kısımda yapılır.

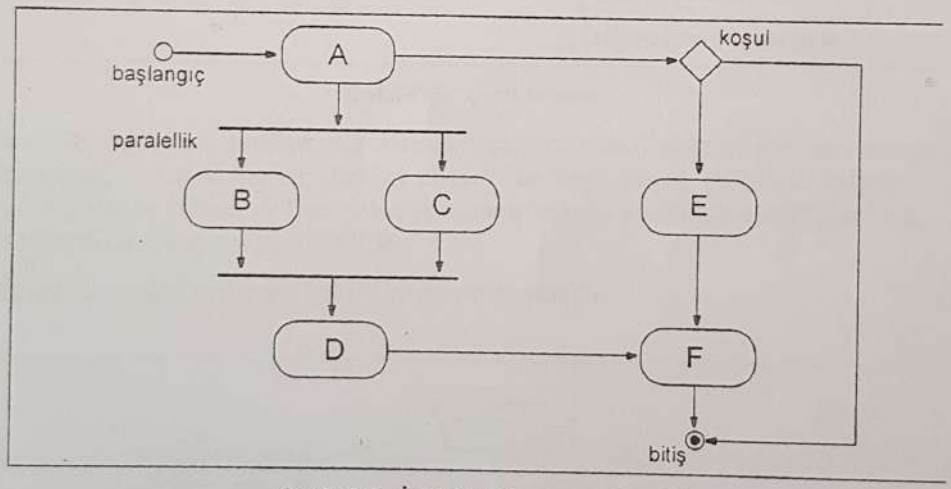
Bir kullanım senaryosunun ana hatları şu şekildedir:

- *Önsöz:* Baştaki bu kısımda, kullanıcılar, ilgililer ve sistemden beklentileri olan unsurlar tanımlanır.
- *Ön koşullar* (precondition) : Belirli bir senaryoyu oluşturan olayların başlaması için sağlanması gereken koşullar ve başlangıç durumu tanımlanır.
- *Ana yol* (basic course) : Etkinliklerin gerçekleşme sırasına göre başlangıçtan bitişe kadar tüm durumların nasıl ve ne zaman oluşacağı açıklanır. Koşullar ve

dallanmalar hariç çalışma şekli adım adım anlatılır.

- *Son koşullar* (postcondition) : Senaryolar tamamlandığında sistemin ulaşacağı olası sonlanma durumları anlatılır.
- *Aykırı durumlar* (exception) : Ana yol dışında olup bir soruna sebep olan sıradışı durumlarla sistemde hata oluştuğunda yapılacaklar anlatılır.
- *Özel isterler* : İşlevselliğe ait olmayan hız, güvenilirlik, kullanım kolaylığı gibi isterler anlatılır.
- *Teknik beklentiler* : Kullanıcının öngördüğü donanım özellikleri anlatılır.

Kullanım senaryolarını ayrıntılı bir şekilde tanımlamak için İşleklik Diyagramları kullanılır. Bu diyagramlar sistemin dinamik yanlarını modellemeye ve bir işlemin a şını tanımlamaya yararlar. İş akışı, geçişler, paralel durumlar, dallanmalar, başlan ve bitişler gösterilir. İşleklik Diyagramları, tüm sistemi bir bütün halinde göster için kullanılabileceği gibi, bir altsistemin, bir işlemin, hatta bir sınıfın dinamik y larını modellemek için de kullanılabilir. Bir örnek Şekil-C.3 te verilmektedir.



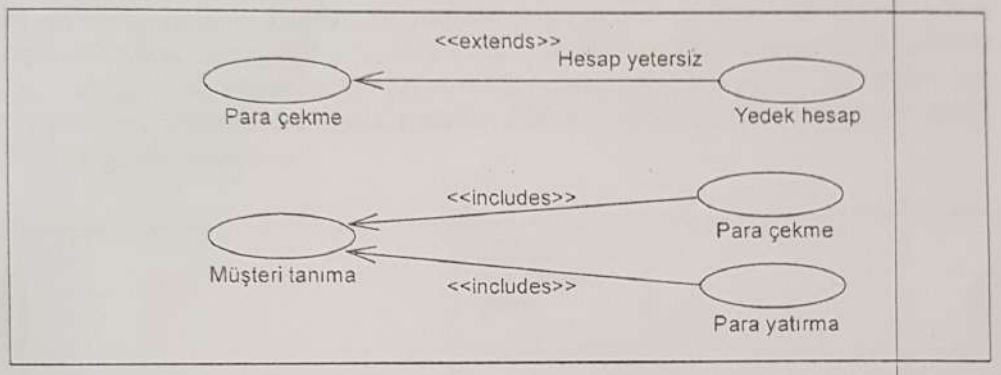
Şekil-C.3. İşleklik Diyagramı örneği.

Kullanım senaryolarında yer alan sistem durumları Durum Çizelgeleri ile açıklanır. B çizelgeler hangi koşullarda, hangi durumlara geçileceğini gösterirler.

5.3 Kullanım Senaryosu Modelinin Yapılandırılması

Kullanım senaryoları arasında ortak olarak kullanılan eylemler, açıklayıcı ve seçeneğ bağlı tanımlamalar çıkartılarak yapılandırma sağlanır. Tüm model içinde böyle bi değerlendirme yapılarak sadeleştirme yapılır ve anlaşılabilirliğin artırılmasın çalışılır. Kullanım senaryoları arasında geçerli olan *genişletme* (extend) ve *içerm* (include) şeklinde iki tip ilişki bulunabilir. Genişletme, temel kullanım senaryosunu davranışının bir başka kullanım senaryosu davranışı ile tamamlanmasıdır. İçerme is

bir kullanım senaryosunun davranışının bir başka kullanım senaryosu içinde bulunmasıdır. Gösterim şekli Şekil-C.4 te verilmektedir:



Şekil-C.4. Genişletme ve içerme.

Geliştirme etkinliklerinde planlama yapılmasına yardımcı olmak üzere kullanım senaryoları arasında önceliklendirme yapılır. Kullanıcı ve geliştiricinin beraberce yaptığı gözden geçirmelerle de modelin doğrulaması (verification) ve geçerlemesi (validation) sağlanır.

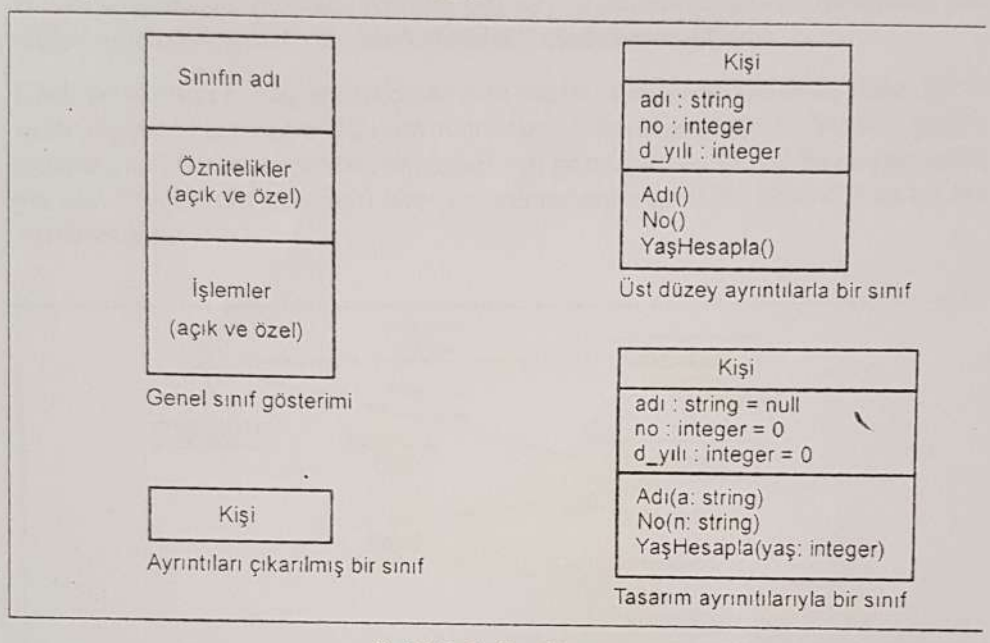
6. UML İle Tasarım

Yazılım geliştirmenin gerçekleştirmeye yönelik en önemli yatırımı tasarım aşamasıdır. Nesneye yönelik çözümleme ve tasarım arasında açık bir ayrım bulunmamaktadır. Onun yerine gerçekleştirme ayrıntılarına yoğunlaşılır. UML ile yapılan yazılım isterleri çözümlemesi sonunda ortaya Yazılım İsterleri Belirtimi çıkar. Kullanım Senaryoları Diyagramları ve Etkinlik Diyagramları bu aşamada hazırlanır. Çözümleme sırasında oluşturulan UML modelleri tasarım sırasında daha da zenginleştirilerek kullanılır.

Tasarımı üst düzey ve ayrıntılı tasarım olarak ikiye ayırmak oldukça yaygındır. Yazılım İsterleri Belirtimi temel alınarak nesne yapısı, modelleme, sistem mimarisi ve veri yapıları tanımları oluşturulur. Sınıf Diyagramları, Ardıllık Diyagramları ve İşbirliği Diyagramları kullanılarak yapısal tasarım ve davranış tasarımı yapılır. Tasarım aşaması sonunda, bir araç ile desteklenen UML modelleri desteğiyle Yazılım Tasarım Tanımlaması hazırlanır.

6.1. Yapısal Tasarım

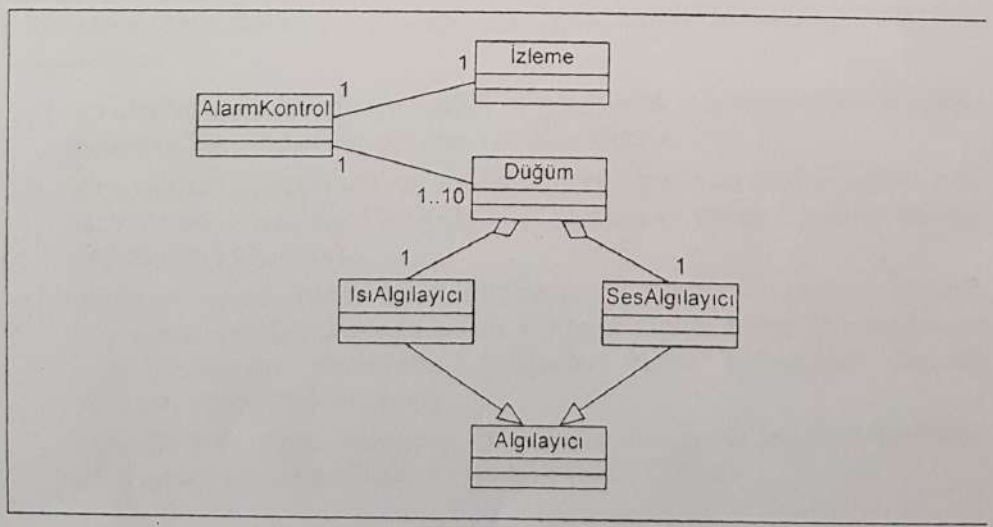
Yapısal tasarımın amacı uygulama alanındaki temel kavramlara karşılık düşen uygun sınıfları, tanımlanan işlevleri yerine getirebilmek için bu sınıflar arasındaki ilişkileri belirlemek ve sistemin sınıf yapısını oluşturmaktır. Yapısal tasarımı gerçekleştirmek için sınıf tanımları ve sınıf yapıları görsel olarak yapılır. Sınıf yapısının gösterim örnekleri Şekil-C.5 te verilmektedir:



Şekil-C.5. Sınıf tanımı.

Sınıf Diyagramları, sınıfları ve onların birbirleriyle olan ilişkilerini gösteren durağa bir model olarak kullanılır. Sınıflar birbirlerine bağlı olarak gösterilir. Paketler v arayüzlerin de kullanılır. Tüm sistemde bulunan sınıflar mantıksal olarak gruplanara birden fazla diyagramla gösterilirler.

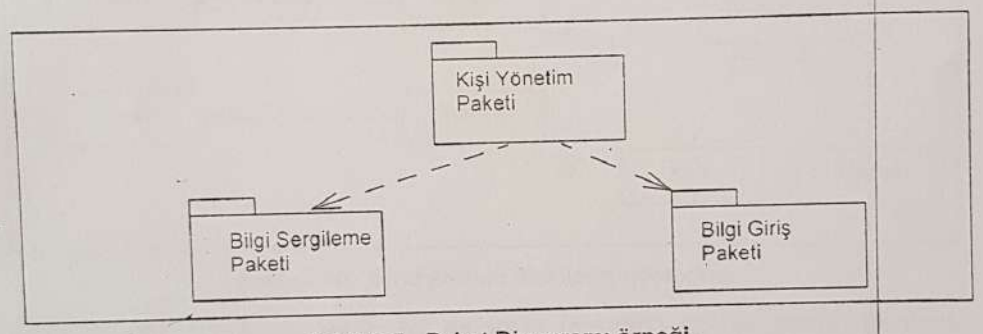
Örnek bir Sınıf Diyagramı Şekil-C.6 da verilmektedir:



Şekil-C.6. Sınıf Diyagramı örneği.

Örnekteki `IsıAlgılayıcı` ve `SesAlgılayıcı` sınıflar `Algılayıcı` sınıfının bir altsınıfıdır. Bu iki sınıf `Düğüm` sınıfının bir parçasıdır. `AlarmKontrol` birer tane `İzleme` ve 10 tane `Düğüm` nesnesine sahiptir.

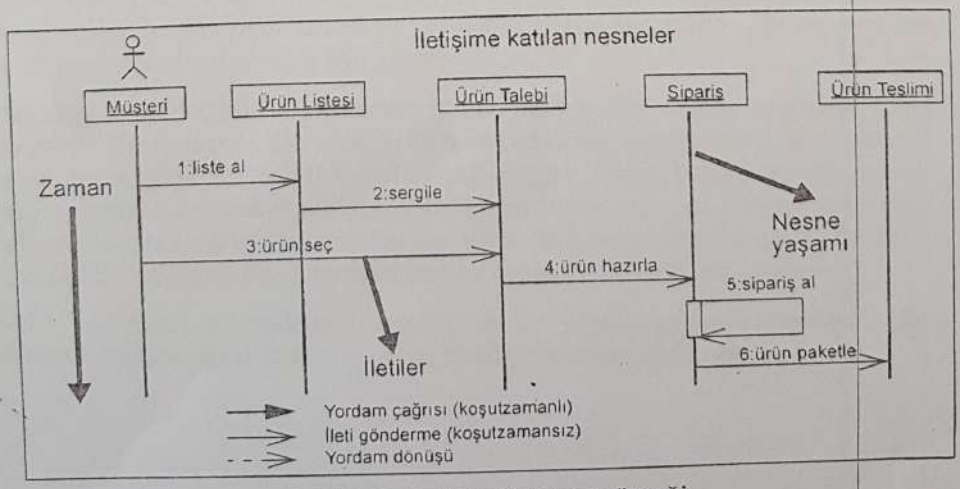
Sınıf diyagramlarını gruplayarak *paket*ler (package) oluşturulur. Her paketin içinde işlevsel olarak birbiriyle ilgili sınıflar bulunur. Paketlerin birbirleriyle olan ilişkileri de aynı diyagramda gösterilir. Bir paket sınıfları, arayüzleri, bileşenleri, işbirliğini, kullanım senaryolarını, hatta başka paketleri içerebilir. Basit bir Paket Diyagramı Şekil-C.7 de gösterilmektedir:



Şekil-C.7. Paket Diyagramı örneği.

6.2. Davranış Tasarımı

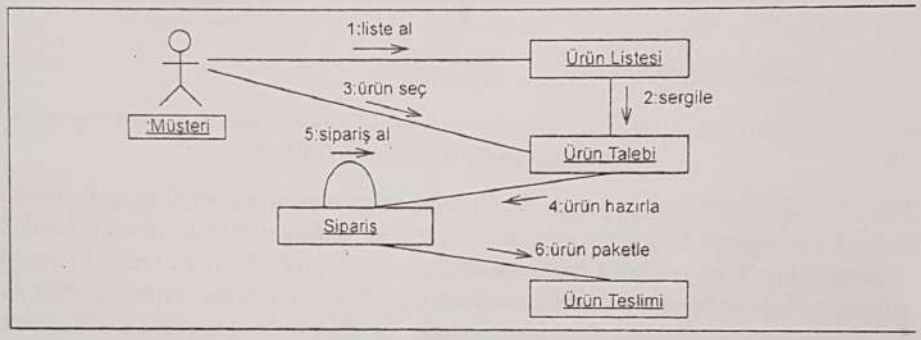
Sistemin dinamik yapısı davranış tasarımı ile belirlenir. Davranış, bir yapısal tanımlama olan işbirliği (collaboration) ile, iletişim tanımı da etkileşim (interaction) ile belirlenir. Genellikle gerçek zamanlı ve karmaşık senaryoların tanımlanmasında kullanılan bu diyagramların bir örneği ve kullanılan simgeler Şekil-C.8 de gösterilmektedir.



Şekil-C.8. Ardıllık Diyagramı örneği.

Ardıllık Diyagramı ve İşbirliği Diyagramı ile sistem davranışları modellenir. İletiler zaman sıralamasını gösteren Ardıllık Diyagramı nesnelerin yaşam sürelerini, yürütükleri eylemleri, genel yürütme kontrolünü modellemeye yarar.

UML de İşbirliği Diyagramları nesnelerin yapısal sıradüzeni üzerinde durur. Bir nenenin diğerine nasıl bağlandığı sıra numaralarıyla beraber gösterilir. Yordam çağrıla eşzamansız iletişim ve yordam dönüşleri ayrı gösterim şekilleriyle bu di-yagramlar yer alır. Etkileşimin sırası ayrı birer sıra numarasıyla belirtilir. Şekil-C.9 da bir örn verilmektedir:



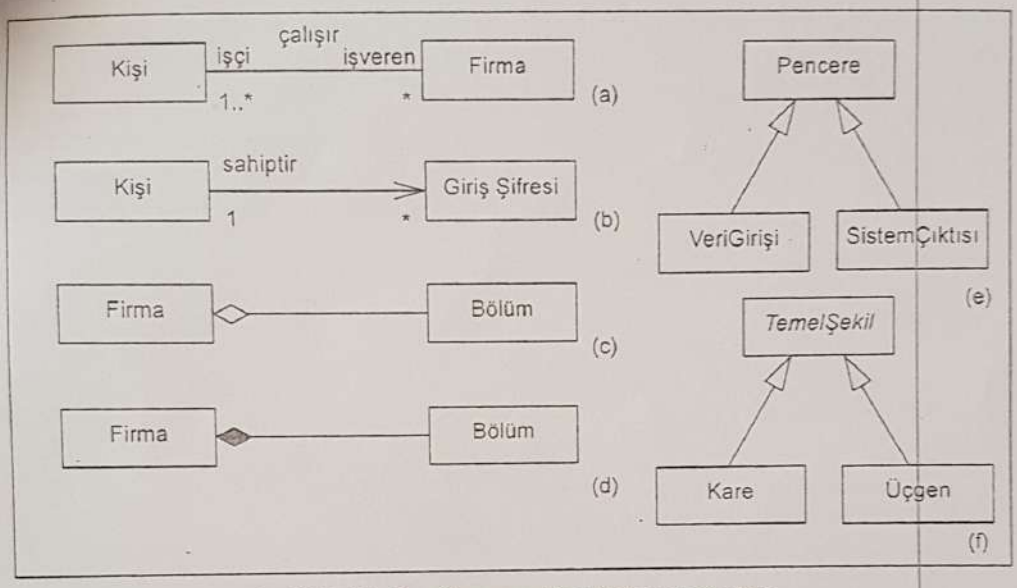
Şekil-C.9. İşbirliği Diyagramı örneği.

6.3. Ayrıntılı Yapısal Tasarım

Yapısal tasarımı daha ayrıntılı hale getirmek üzere sınıf tanımları, ilişkiler ve sistem mimarisi yeniden elden geçirilir. Sınıf tanımlarında yer alan öznitelikler ve yordamlarla ilgili bilgiler seçilen programlama dilinin özelliklerine göre görünürlük ve soyutlama gibi ilkelere bağlı kalınarak ayrıntıyla tanımlanır. Özniteliklerin türleri, alabilecekleri sınır değerleri ve ilk değerler belirtilir. Sınıflar arasında aşağıdaki ilişkiler tanımlanır:

- *Birliktelik* (association), rol, çokluk ve görünürlük bilgilerini içeren bir ilişki tanımlar. Tek yönlü veya çift yönlü olabilir (Şekil-C.10-a, b).
- *Kümeleşme* (aggregation), bir sınıfın sahibinin başka bir sınıf olduğunu gösterir. Aradaki ilişki zayıf bir ilişki olup, yaratma ve yoketme konusu belirgin değildir (Şekil-C.10-c).
- *Birleşme* (composition), bir sınıfın başka bir sınıftan oluştuğunu gösterir. Kümeleşmenin daha kuvvetli halidir. Birleşme ilişkisi, bileşenin yaratılma ve yokedilmesinden sorumludur. Bileşenler başka birleşmeler arasında paylaşılamazlar (Şekil-C.10-d).
- *Genelleştirme* (generalization), bir sınıfın bir başka sınıftan özellikler alındığını, yani kalıtım ilişkisini tanımlar (Şekil-C.10-e).

8



Şekil-C.10. Sınıflararası ilişkilerin gösterimi.

Genelleştirme sıradüzeni içinde bir de soyut (abstract) sınıflar vardır. Bunların doğrudan yaratımları bulunmaz, mutlaka kalıtımla kullanılmaları gerekir. Bu tür sınıfların adları eğik harflerle (*TemelŞekil* sınıfı) yazılır (Şekil-C.10-f).

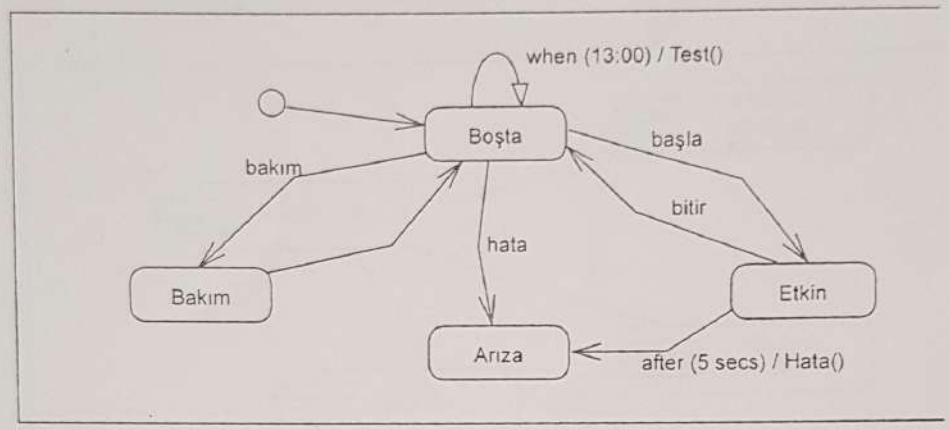
6.4. Ayrıntılı Davranış Tasarımı

Sistemdeki sınıfların davranışlarını ayrıntılı bir şekilde tasarlamak üzere nesne grupları bağlamında için Ardıllık, İşbirliği, ve Etkinlik Diyagramları, tek bir nesne bağlamında da Durum ve Etkinlik Diyagramları kullanılır.

Ayrıntılı tasarım için daha önce yapılmış olan Ardıllık ve İşbirliği Diyagramları biraz daha genişletilir. Tasarımın temelinde yer alan sınıfların davranışları, durum geçişleri ve etkileşimleri daha ayrıntılı hale getirilir.

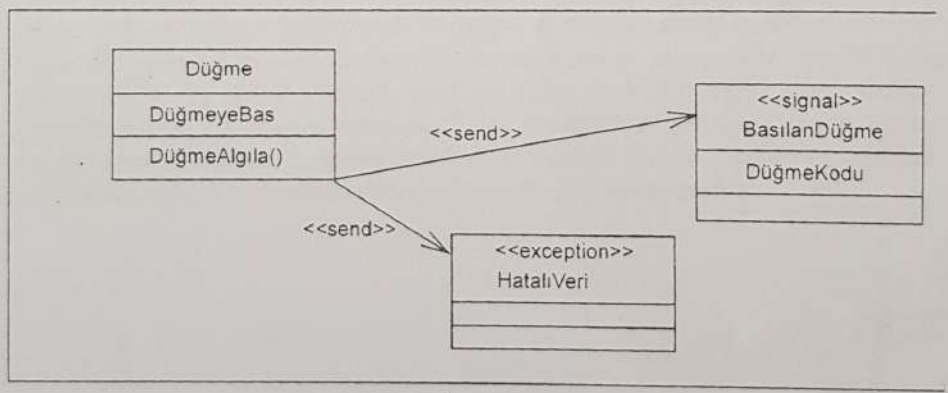
Davranış modellemesinin en önemli unsurlarından biri durum makinesi (state machine) oluşturmaktır. Bu amaçla kullanılan Durum Diyagramları, bir nesnenin yaşam süreci boyunca belirli koşulları sağladığında içinde bulunacağı durumu ve yerine getirmesi gereken etkinlikleri gösterir. Her durumun bir adı, bir giriş ve bir çıkış noktası, ile yapılacak işlerin tanımlanması vardır. Durumdan duruma geçişlerin de bir kaynağı, bir tetikleme olayı, koruma koşulu, eylem ve hedef durum bulunur.

Şekil-C.11 örnek bir makine sisteminin durum diyagramını göstermektedir. Bu sistemde zamana dayalı olaylar da diyagramda gösterilebilmektedir.



Şekil-C.11. Durum Diyagramı örneği.

Durum değişikliğine yol açan olaylar arasında bir işaret, iç durumdaki bir değişiklil belirli bir zaman değerine ulaşma veya dışarıdan gelen bir çağrı bulunabilir. Olayları belirli süreleri ve kalıcılıkları yoktur; mutlaka meydana geldikleri anda yakalanmala ve yanıtlanmaları gereklidir. UML, bir yordamın başka bir nesneye işaret gönde mesini modellemeye yardımcı olacak gösterime sahiptir. Şekil-C.12, *stereotip* adı ve rilen gösterimle işaret gönderilişine ve aykırı durum oluştuğunda kotarılması içi hatanın yönlendirilmesine örnek vermektedir.

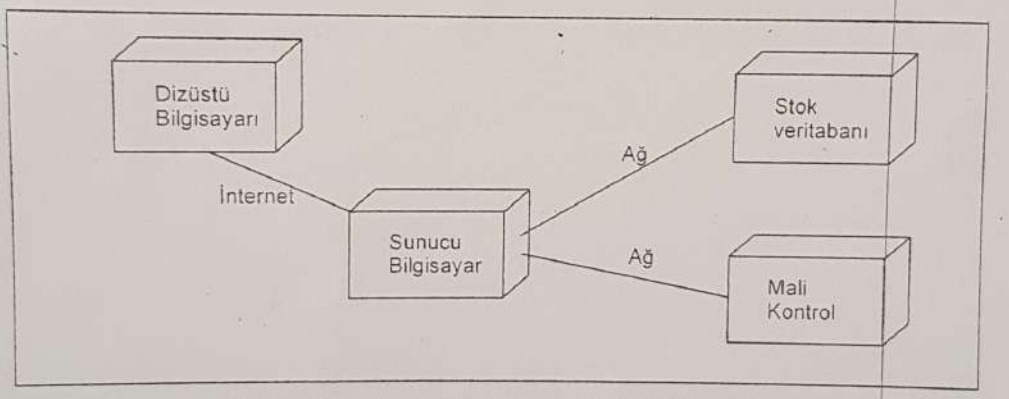


Şekil-C.12. İşaret gönderme.

Ayrıntılı Sistem Mimarisi

Sistem mimari tasarımının amacı genel sistem sıradüzenini bileşenlere ayrıştırarak tanımlamaktır. Sistem paketleri daha ayrıntılı hale getirilir, temel tasarım sınıfları ya ayrı paketlere konur, ya da aynı paketlerin içine yerleştirilir. Paketlerin ne tür kod parçalarına dönüştürüleceği, uygulama yazılımının tek işlemci üzerinde mi koşacağı yoksa dağıtık mı olacağı belirlenir.

Bir uygulamanın en basit şekli yürütülen ana programın ve tüm dosyaların aynı işlemci üzerinde bulunmasıdır. Dağıtık yapıdaki bazı sistemler iki katmanlı (two-tiered) olarak tanımlanır ve kullanıcı-sunucu modeline uyar. Bazı sistemler de üç katmanlı (three-tiered) olarak tanımlanır, kullanıcı arayüzü, iş yapan kısım ve saklama kısmı ayrı ayrı makinelerde bulunabilir. Bu yapılar göre belirlenen sistemin fiziksel mimarisi Konuşlandırma Diyagramı ile modellenir. Şekil-C.13, İnternet üzerinden satış gerçekleştiren bir örnek sistemi göstermektedir:



Şekil-C.13. Konuşlandırma Diyagramı örneği.

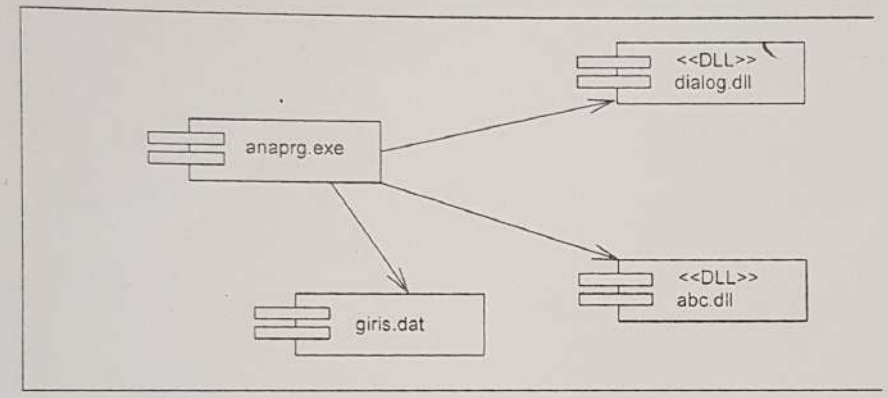
7. Gerçekleştirim

UML ile yapılan çözümleme ve tasarımın ardından bir nesneye yönelik programlama dili kullanılarak gerçekleştirim (implementation), yani kodlama yapılır. Amaç, tasarım iş akışlarından geçerek modellerle tanımlanan sistemi gerçek hale getirmektir. Bu bir yazılım olduğu için asıl hedef yürütülebilir kodun oluşturulmasıdır. Kodu oluşturan ayrıntılı bir bileşen mimarisi tanımlanır ve geliştirilen bu bileşenlerin birim testleri yapılır.

Tasarımda yer alan sınıflar belirli dosyalara yerleştirilir ve kodlanır. Sınıf diyagramlarında yer alan yordam belirtimleri artık yordam bedenleri halinde kodlanır. Algoritmalar koda dönüştürülür. Her sınıf için birim testi (kara kutu veya saydam kutu testleri) yapılır.

Bileşenler (component), bir bilgisayar üzerinde yer alan yürütülebilir kod, dinamik kütüphane, tablo veya dosya gibi fiziksel varlıklardır. Herbirinin arayüzleri ve belirli adları vardır. UML de bileşenler ve aralarındaki ilişkiler Bileşen Diyagramları ile gösterilirler.

Şekil-C.14 bu diyagrama bir örnek vermektedir:



Şekil-C.14. Bileşen Diyagramı örneği.

Belgelendirme

Geliştirme sürecinin başında ne tür bir belgelendirme yapılacağının belirlenme şarttır. Birçok belgelendirme standardı çözümleme ve tasarım aşamalarının sonunc birer belge çıkmasını bekler, fakat kodlama sonrasında bir belgeye gereksinim duy maz. UML ile yapılan geliştirmede bu belgelere neler konacağı önceden belirlenme lidir. Belgelerin biçimleri, içerikleri, içlerine konacak diyagramlar üzerinde kara verilmelidir. Bu işlerin kodlama bittikten sonra yapılabileceğine karar verilmes aslında, hiç yapılmamasına neden olur. UML diyagramları belirli bir yazılım paketi il desteklendiği takdirde, bu ortamda saklamak da bir yöntemdir. O takdirde, düzenleşi sistemine kaynak kodla birlikte diyagram dosyaları da konmalıdır.