YMT 312-Yazılım Tasarım Ve Mimarisi

Mimari Stilleri

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bölüm-12

Bu Haftaki Konular

GENEL BAKIŞ

KISIM 1 – Yazılım Tasarımında Kalıplar

KISIM 2 – Mimari Stilleri

KISIM 1

Yazılım Tasarımında Kalıplar
(Patterns in Software Design)

Amaçlar

- Tasarım kalıplarının neden önemli olduğunu açıklamak
- Yazılım tasarım kalıplarını tanımlamak
- Ayrıntılanışına bağlı olarak tasarım kalıplarının bir sınıflandırmasını sunmak
- Tasarım kalıbı kataloglarını tartışmak

İçerik

- Tasarım kalıpları (design patterns) neden önemli?
- Christopher Alexander
- Yazılım tasarım kalıpları
- Kalıpların ayrıntılanışı
- Kalıp katalogları

Neden Tasarım Kalıpları?

Uzman tasarımcılar acemilerden farklı davranır — uzmanlar, acemilerin bilmediği ne biliyor olabilir?

Diğer şeylerin dışında, uzmanlar yeni problemlere uyguladıkları, geçmiş deneyimlerden gelen başarılı tasarım kalıpları bilgisine sahiptir.

Kalıpların Avantajları

- **İletişimi arttırmak** (Promoting communication)—Kalıp isimleri ve avantaj/dezavantajlarının bilinmesi tasarımcılar arasındaki iletişimi hızlandırır
- **Dokümantasyonu kolaylaştırmak** (Streamlining documentation)—Kalıpların yapılarını ve davranışlarını ayrıntılandırmaya gerek yoktur
- **Verimliliği arttırmak** (Increasing efficiency)—Kalıplara yönelik araç desteği yazılım geliştirmeyi hızlandırır
- Yeniden kullanımı desteklemek (Supporting reuse)—Kalıplar ve onların implementasyonları geniş biçimde yeniden kullanılabilir
- **Yeni fikirler sağlamak** (Providing ideas)—Kalıplar bir tasarımın ya da iyileştirmenin başlangıç noktası olabilir.

Bina Mimarisinde Tasarım Kalıpları

- 1970'lerde mimar Christopher Alexander bina tasarımında tasarım kalıplarına bağlı yeni bir yaklaşım öne sürdü.
- Alexander diyordu ki:
  - Kalıplar insan anatomisi, psikolojisi, fizyolojisi, sosyolojisi, ve politikasının doğal sonuçlarıdır;
  - Büyük mimariler daima kalıplara dayalı gerçekleştirilmiştir, fakat bunlar üzerinde sistematik olarak çalışılmamıştır; ve
  - Kalıplar bilindiği ve anlaşıldığı taktirde herkes büyük yapılar oluşturulabilir.

Alexander'ın Kalıplarından Örnekler

Yayılmış İşyerleri (Scattered Work)—Yasalar ve teşvikler kullanılarak iş yerlerinin şehrin her yerine dağılmasının sağlanması.

4-Kat Sınırı (Four-Story Limit)—Şehirdeki binaların çoğunun en fazla 4 kat yüksekliğinde olması.

Güneye Bakan Dış Mekanlar (South Facing Outdoors)—Binaların daima dış mekanların kuzeyine inşa edilmesi.

Sıcak Renkler (Warm Colors)—Odadaki ışık ve duvar renklerinin sarı-yeşil ton elde edilecek şekilde seçilmesi.

Alexander’ın Çalışmasının Sonuçları

- Alexander’ın çalışması, kendisinin de itiraf ettiği gibi, bina mimarları arasında yaygın kabul görmedi, ve profesyonel olmayanların bile büyük mimariler tasarlamasına yönelik çabaları da genellikle başarısız oldu.
- Fakat Alexander, tasarım kalıpları kullanılarak üretim yapılması konusunda yazılım geliştirme topluluklarına esin kaynağı oldu.

Tasarım Kalıpları

Bir **kalıp/desen** (pattern) taklit edilmek için önerilen bir modeldir. Bir **yazılım tasarım kalıbı** (software design pattern) bir yazılım tasarım probleminin çözümünde kullanılacak, taklit edilmek için önerilen bir modeldir.

Tasarım Kalıplarının Ayrıntılanışı

- **Mimari stilleri veya kalıpları** (Architectural styles or patterns) bütün bir sistem ve alt-sistemler içindir.
- **Tasarım kalıpları** (Design patterns) etkileşen çeşitli fonksiyonlar veya sınıflar içerirler.
- **Veri yapıları & algoritmalar** (Data structures & algorithms) alt-düzey kalıplardır.
- **Programlama deyimleri** (Idioms) spesifik programlama dilleri ile bir takım işleri yapmanın yollarıdır.

Kalıp Katalogları

- Tasarım kalıplarının öneminin anlaşılması, kalıp kataloglarının oluşturulmasına yol açtı.

- Bunlar bina mimarisinde veya iç mimaride kullanılan kalıp kitaplarına ya da mühendislikte kullanılan el kitaplarına benzer kalıp kataloglarıdır.

- Bu ders kapsamında yazılım tasarım kalıplarının küçük bir bölümüne değineceğiz.

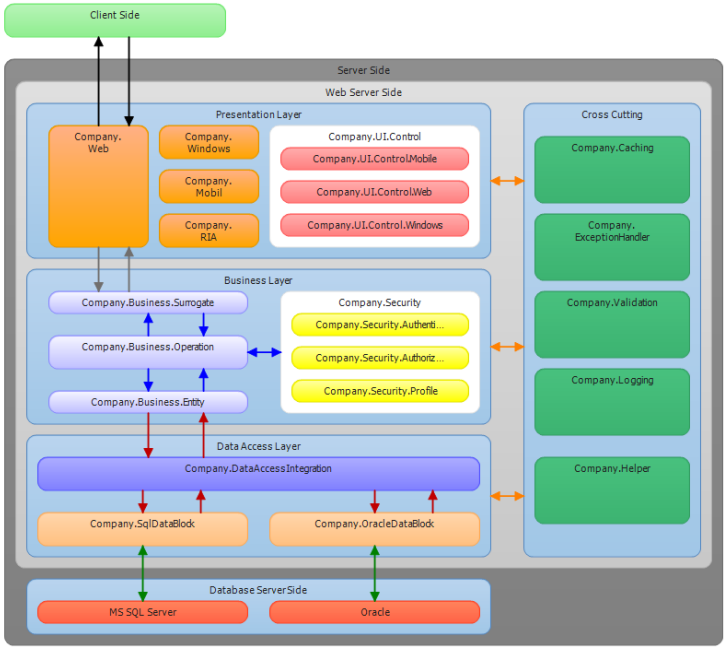
Özet

- Tasarım kalıpları, bir alandaki uzmanlığı ve deneyimi barındıran, hem acemilere hem de uzmanlara büyük kolaylıklar sağlayan şablonlardır.

- Kalıplar çeşitli ayrıntı düzeylerinde olabilir, örneğin mimari stilleri, orta-düzey tasarım kalıpları, veri yapıları ve algoritmalar, ve programlama deyimleri.

KISIM 2

Mimari Stilleri
(Architectural Styles)



Amaçlar

- Mimari tasarım ve mimari stilinin tanımlarını gözden geçirmek
- Çeşitli önemli mimari stillerini sunmak:
  - Katmanlı (Layered)
  - Kanal-ve-Filtre (Pipe-and-Filter)
  - Paylaşımlı-Veri (Shared-Data)
  - Olay-Güdümlü (Event-Driven)
  - Model-View-Controller
  - Hybrid

İçerik

- Yazılım mimarisi ve mimari stilleri
- Katmanlı (Layered) stil
- Kanal-ve-Filtre (Pipe-and-Filter) stili
- Paylaşılan-Veri (Shared-Data) stili
- Olay-Güdümlü (Event-Driven) stili
- Model-View-Controller stili
- Hibrit mimariler

Yazılım Mimarisi

Bir **yazılım mimarisi** (software architecture) bir programın büyük bileşenlerinden, onların sorumluluk ve özelliklerinden, ve aralarındaki ilişki ve etkileşimlerden oluşan yapısıdır (structure).

Yazılım Mimarisi Konuları

- Sistem fonksiyonlarının kabaca ayrıştırılması (decomposition)
- İşlevlerin bileşenlere tahsis edilmesi
- Bileşen arabirimleri (interfaces)
- Bileşenler arasındaki iletişim ve etkileşim
- Bileşen ve sistem özellikleri: kapasite, verim, ve kısıtlar (capacity, throughput, constraints)
- Yaygın tasarım stillerinin yeniden kullanımı

Mimari Stilleri (Architectural Styles)

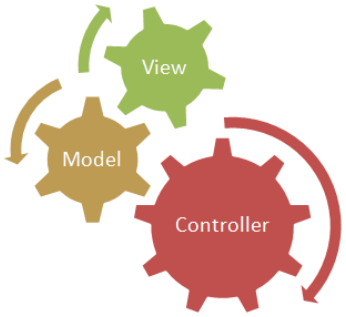
Bir **mimari stili** (architectural style) bir program veya sistemi oluşturan tipler ve bunların etkileşimlerinin bir paradigmasıdır.

Katmanlı Mimariler (Layered Architecture)

- Program bir dizi katmana veya gruba ayrıştırılır.

- Bir katman altındaki katman(lar)dan hizmet alır ve üzerindeki katman(lar)a hizmet verir.

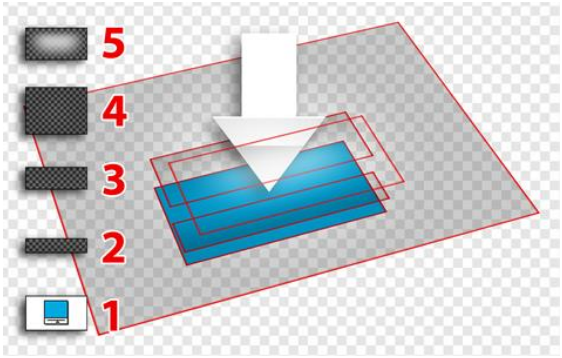
- Tüm mimari stilleri içerisinde katmanlı stil en çok kullanılan stildir.



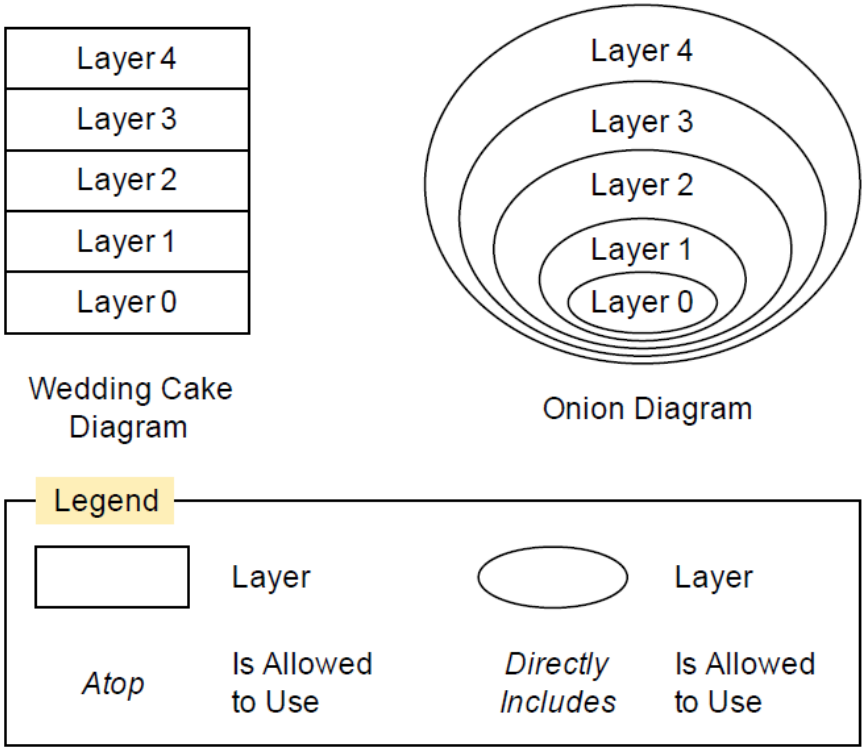
Katman Kısıtları

Statik yapı (Static structure)—Yazılım, her bir katman iyi tanımlanmış bir arabirim üzerinden yüksek uyuma (cohesion) sahip bir dizi servisi sağlayacak şekilde bölümlenir.

Dinamik yapı (Dynamic structure)—Her bir katman yalnızca hemen altındaki katmanı kullanabilir **(Strict Layered)** veya altındaki tüm katmanları kullanabilir **(Relaxed Layered).**



Katmanların Gösterimi



Katmanların Oluşturulması

Soyutlama düzeyleri

- Örneğin: Ağ iletişim katmanları

Sanal makinalar

- Örneğin: İşletim sistemleri, yorumlayıcılar (interpreters)

Bilgi saklama, decoupling, vb.

- Örneğin: Kullanıcı arayüzü katmanı, sanal cihaz (cihaz sürücüsü) katmanı

Katmanlı Stilin Avantajları

- Katmanlar son derece yüksek uyumludur (cohesive) ve bilgi saklamayı arttırır.
- Katmanlar, üzerlerindeki katmanlarla yüksek derecede bağlı değildir (loose coupling), bu sayede toplam bağlılık azaltılmış olur.
- Katmanlar programı parçalara ayırmayı kolaylaştırır, karmaşıklığı azaltır.
- Katmanları değiştirmek veya onarmak, veya tüm katmanı başkasıyla değiştirmek oldukça kolaydır.
- Bir katmana işlevsellik ekleyerek katmanı genişletmek kolaydır.
- Katmanlar genellikle yeniden kullanım için çok uygundur.

Katmanlı Stilin Dezavantajları

- Herşeyi bir çok katman içinden geçirmek sistemi karmaşık hale getirebilir ve performansı düşürebilir.
- Bir çok katman içinden hata ayıklamak genellikle zordur.
- Doğru katmanları oluşturmak zor olabilir.
- Önceden öngörülemeyen bazı işlevsellikleri sağlayabilmek için katmanlar arası kısıtların ihlal edilmesi zorunlu olabilir.

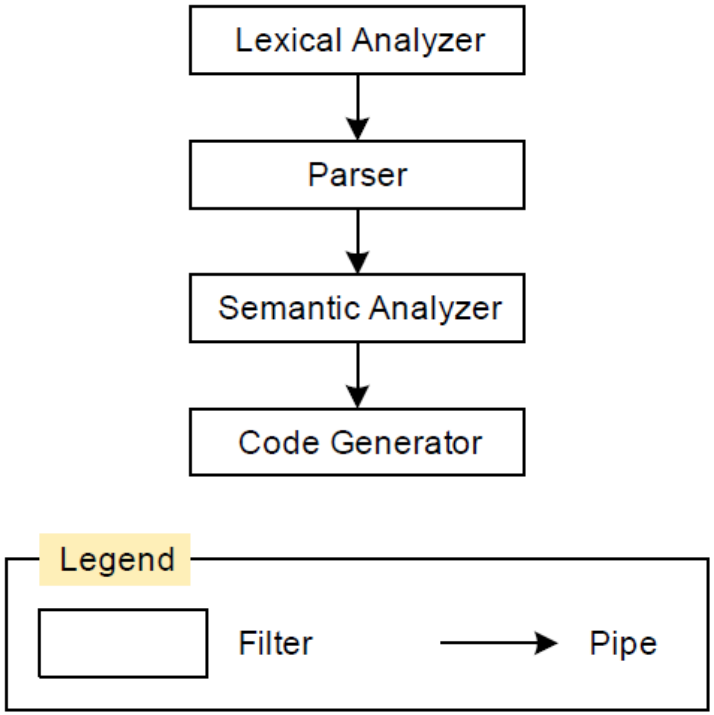
Kanal-ve-Filtre (Pipe-and-Filter) Stili

- Bir **filtre** (filter) bir girdi akışını (input stream) bir çıktı akışına (output stream) dönüştüren bir program bileşenidir.

- Bir kanal (pipe) bir akışı taşıyan aracı bir mecradır.

- **Pipe-and-Filter** stili, program bileşenlerinin kanallar ile birleştirilmiş filtrelerden oluştuğu dinamik bir modeldir.

Kanal-ve-Filtre Örneği



Kanal-ve-Filtre Karakteristikleri

- Kanallar (Pipes) genellikle izole edilmiştir ve yalnızca veri akışları üzerinden iletişim kurarlar. Bu sayede yazılmaları, test edilmeleri, yeniden kullanılmaları, ve başkasıyla değiştirilmeleri kolaydır.
- Filtreler eşzamanlı (concurrent) olarak çalışabilir.
- Kanalların filtreleri senkronize etmesi gerekir
- Pipe-and-filter topolojileri acyclic graph olmalıdır.
- Zamanlama ve dead-lock sorunları yaşanmaması için
- Basit ve doğrusal bir düzenlemesine örnek olarak **pipeline** verilebilir.

Kanal-ve-Filtre Avantajları

- Filtreler kolayca modifiye edilebilir ve bir başkasıyla değiştirilebilir.
- Filtreler çok az bir çaba ile yeniden düzenlenebilir, bu sayede benzer programlar geliştirmek kolaydır.
- Filtreler son derece yüksek yeniden kullanılabilirlik sunarlar.
- Eşzamanlılık desteklenir ve implemente etmesi nispeten kolaydır.

Kanal-ve-Filtre Dezavantajları

- Filtreler yalnızca kanallar üzerinden haberleşir, dolayısıyla onları koordine etmek zor olabilir.
- Filtreler genellikle basit veri akışları (data stream) üzerinde çalışırlar, dolayısıyla boşa giden veri dönüşüm eforuna neden olabilirler.
- Hata yakalaması/yönetimi zordur.
- Eşzamanlılıktan elde edilen kazanç yanıltıcı olabilir.

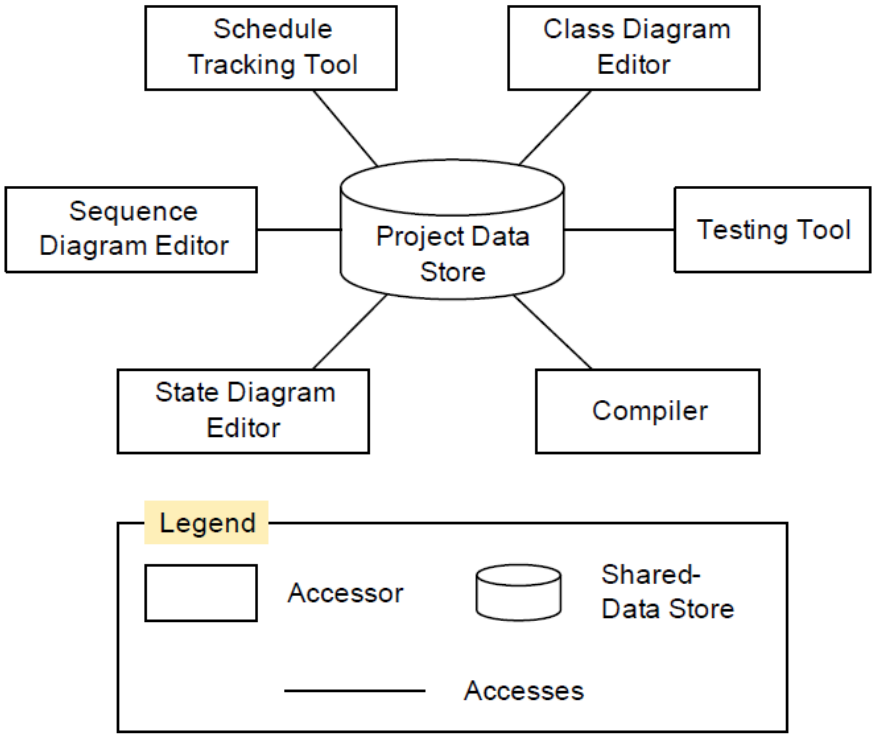
Paylaşılan-Veri (Shared-Data) Stili

- Bir veya daha fazla **paylaşılan-veri deposu** (shared-data stores) yalnızca bu veri depoları üzerinden iletişim kuran bir veya daha fazla **paylaşılan-veri erişimcisi** (shared-data accessors) tarafından kullanılır.

- İki türü vardır:
  - **Karatahta stili** (Blackboard style)—Paylaşılan-veri depoları depolarda değişim olduğunda erişimcileri bilgilendirir.
  - **Depo stili** (Repository style)—Paylaşılan-veri depoları pasiftir ve erişimciler tarafından manipüle edilir.

- Bu yalnızca dinamik bir modeldir.

Paylaşılan-Veri Stili Örneği



Paylaşılan-Veri Stili Avantajları

- Paylaşılan-veri erişimcileri yalnızca paylaşılan-veri depoları üzerinden haberleşir, dolayısıyla bunları değiştirmek, çıkartmak, ya da eklemek kolaydır.

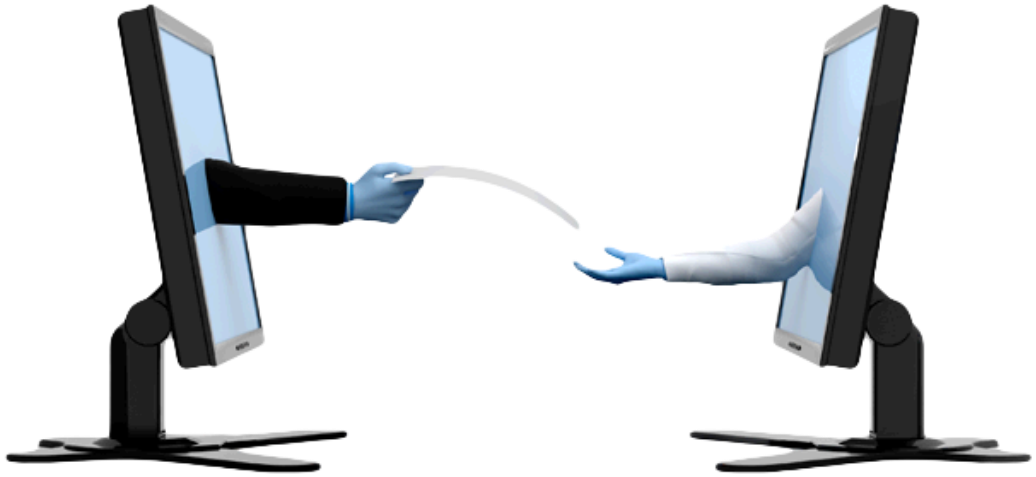
- Erişimci bağımsızlığı, sistemin dayanıklılığını ve hata toleransını arttırır.

- Tüm veriyi paylaşılan-veri deposuna yerleştirmek güvenliği ve denetimi kolaylaştırır.

Paylaşılan-Veri Stili Dezavantajları

- Tüm verinin paylaşımlı-veri deposu üzerinden kullanılmasını zorlamak performansı düşürebilir.

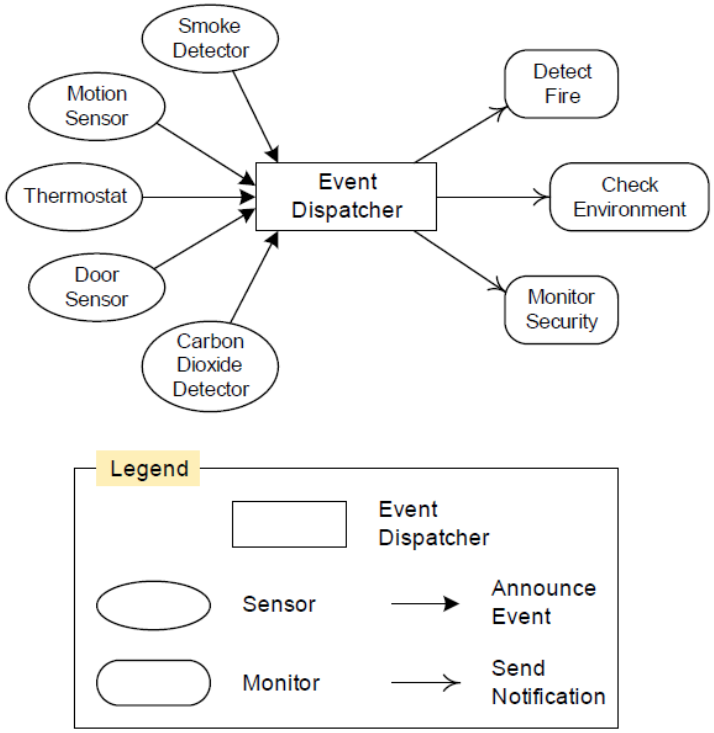
- Paylaşılan-veri deposunda sorun oluşursa, bundan tüm program olumsuz etkilenir.



Olay-Güdümlü (Event-Driven) Stili

- Ayrıca **üstü kapalı çağırma** (Implicit Invocation) stili olarak da bilinir.
- Bir **olay** (event) dikkate değer herhangi bir oluşum/etkinliktir.
- Bir **olay dağıtım memuru** (event dispatcher) olayı duyuran ve olaydan haberdar edilen bileşenler arasında aracılık eder.
- Bu yalnızca dinamik bir modeldir.

Olay-Güdümlü Stili Örneği



Çeşitli Varyasyonlar

- Olaylar yalnızca bildirim şeklinde olabilir ya da veri taşıyabilir.

- Olay hareket memuru tarafından bazı olaylar manipüle edilebilir ya da gözardı edilebilir.

- Olaylar senkron ya da asenkron biçimde yönetilebilir.

Olay-Güdümlü Stili Avantajları

- Bileşenleri eklemek veya çıkartmak kolaydır.

- Bileşenler bağlı değildir (decoupled), bu sayede son derece yüksek yeniden kullanılabilirliğe ve değiştirilebilirliğe sahiptirler.

- Bu stil kullanılarak oluşturulan sistemler genelde sağlamdırlar ve hata toleransları yüksektir.

Olay-Güdümlü Stili Dezavantajları

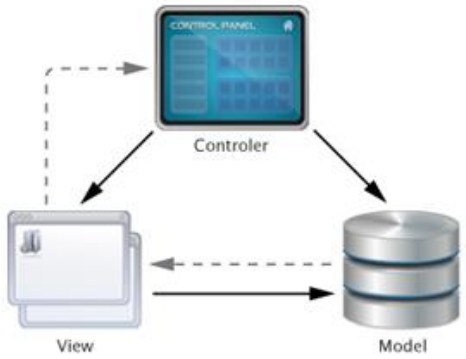
- Olay hareket memuru tarafından yönetildiğinde bileşenlerin etkileşiminde tuhaflıklar olabilir.

- Olayların sıralamasında ve zamanlamasında herhangi bir garanti verilemez, bu da doğru programlar yazmayı zorlaştırabilir.

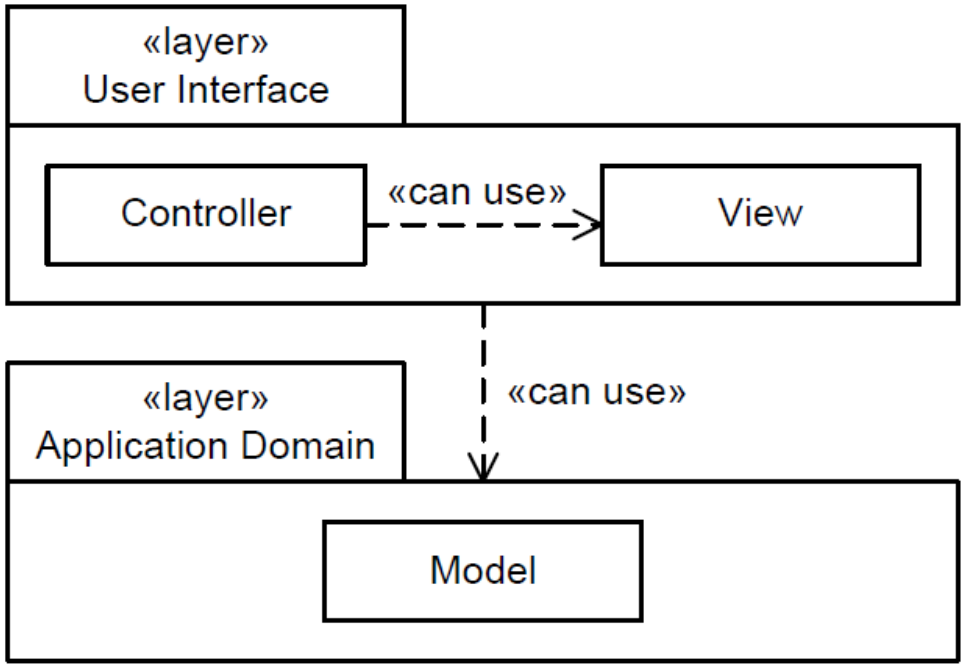
- Olay trafiği çok değişken olabilir, bundan dolayı performans hedeflerine ulaşmak zor olabilir.

Model-View-Controller (MVC) Stili

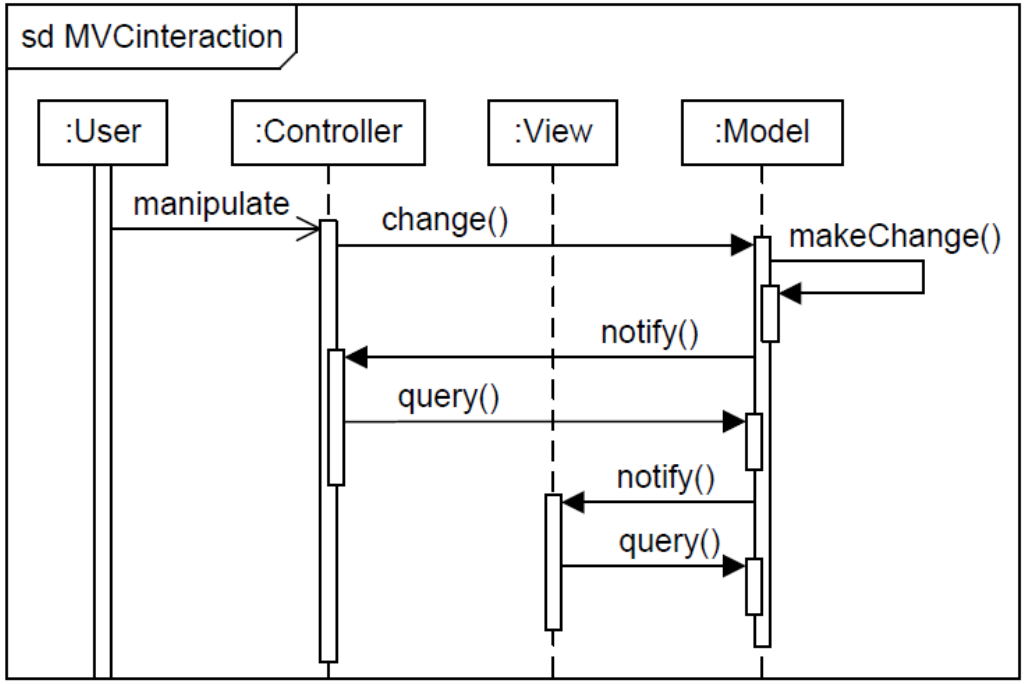
- Bu stil kullanıcı arayüzü ile problem-alanı bileşenleri arasındaki ilişkilerin nasıl kurulacağını modeller.
- **Model**—Kullanıcı arayüzünden tamamen bağımsız bir şekilde programın amaçlarına ulaşmak amacıyla veri ve operasyonlara sahip bir problem-alanı bileşenidir.
- **View**—Bir veri gösterim bileşenidir.
- **Controller**—Kullanıcı girdilerini alan ve bunlara karşı aksiyonda bulunan bir bileşendir



MVC Statik Yapısı



MVC Davranışı



MVC Avantajları

- Model bileşenini hiç etkilemeden View ve Controller bileşenleri eklenebilir, çıkartılabilir, veya değiştirilebilir.

- Programın çalışması sırasında View bileşenleri eklenebilir veya değiştirilebilir.

- Çalışma zamanında (runtime) bile kullanıcı arayüzü bileşenleri değiştirilebilir.

MVC Dezavantajları

- View ve Controller bileşenlerini ayırmak genellikle zordur.

- Sık güncellemeler (yani bileşenler arasındaki etkileşimler) veri gösterimini yavaşlatabilir ve kullanıcı arayüzünde performans sıkıntısı oluşturulabilir.

- MVC stili kullanıcı arayüzü bileşenlerini model bileşenlerine çok fazla bağımlı hale getirebilir.

Hibrit Mimariler

- Çoğu sistem çeşitli mimari stillerini, genellikle farklı soyutlama seviyelerinde, bir arada içerir.
  - Sistemin genelinde Katmanlı stil kullanılırken katmanların birinde Olay-Güdümlü, diğerinde ise Paylaşımlı-Veri stili kullanılabilir.
  - Sistemin genelinde Pipe-and-Filter stili kullanılırken, filtre bileşenleri Katmanlı stilde oluşturulmuş olabilir.

Özet

- Katmanlı stilinde, program bileşenleri katmanlara ayrılmıştır ve her bir katman yalnızca altındaki katman(lar)ı kullanacak şekilde sınırlandırılmıştır.
- Pipe-and-Filter stilinde bileşenler kanallarla (pipe) bağlanmış filtreler (filter) şeklindedir.
- Paylaşılan-Veri stilinde bileşenler bir veya daha fazla paylaşılan-veri erişimcisi tarafından manipüle edilen bir veya daha fazla paylaşılan-veri deposu olarak modellenir.
- Olay-Güdümlü stilinde program bileşenleri bir olay hareket memuruna kaydolurlar, ve olay bildirimleri ve olaydan haberdar olmalar bu hareket memuru aracılığıyla gerçekleşir.
- Model-View-Controller stilinde kullanıcı arayüzü View ve Controller bileşenleri problem-alanı Model bileşenleriyle haberleşirler.
- Mimariler genellikle farklı soyutlama seviyelerinde birden çok stilden oluşurlar. Bu tür mimarilere hibrit mimariler adı verilir.

Kaynaklar

"Software Engineering A Practitioner's Approach" (7th. Ed.), Roger S. Pressman, 2013.

"Software Engineering" (8th. Ed.), Ian Sommerville, 2007.

"Guide to the Software Engineering Body of Knowledge", 2004.

" Yazılım Mühendisliğine Giriş", TBİL-211, Dr. Ali Arifoğlu.

"Yazılım Mühendisliği" (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.

Kalıpsiz, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.

Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (http://www.buzluca.info/dersler.html)

Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.

Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

http://www.cclub.metu.edu.tr/bergi_yeni/e-bergi/2008/Ekim/Cevik-Modelleme-ve-Cevik-Yazilim-Gelistirme

http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2011/ch6_6d_sk

http://dsdmofagilemethodology.wikidot.com/

http://caglarkaya.piquestion.com/2014/07/01/244/