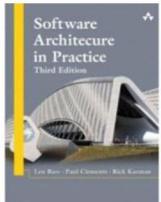


Patterns and Tactics

3-1

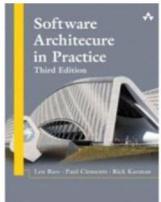
Kalıplar ve Taktikler

© Len Bass, Paul Clements, Rick Kazman,
distributed under Creative Commons
Attribution License



Chapter Outline

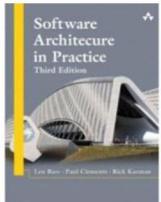
- What is a Pattern
- Overview
- Pattern Catalogue
 - Module patterns
 - Component and Connector Patterns
 - Allocation Patterns
- Relation Between Tactics and Patterns
- Using tactics together
- Summary



What is a Pattern_1?

An architectural pattern establishes a relationship between:

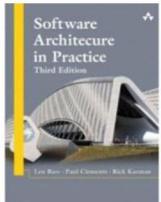
- A *context*. A recurring, common situation in the world that gives rise to a problem.
- A *problem*. The problem, **appropriately generalized**, that arises in the given context.
- A *solution*. A successful architectural resolution to the problem, **appropriately abstracted**.



Kalıp Nedir_1?

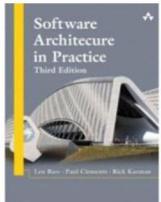
Bir mimari kalıp şunların arasında ilişki kurar:

- *Bir bağlam*. Problem çıkan bir dünyada tekrarlayan genel genel bir durum
- *Bir problem*. Veri bağlamda yer alan **uygun şekilde genelleştirilmiş** problem.
- *Bir çözüm*. Problemin, **uygun şekilde soyutlanmış** başarılı bir mimarisi.



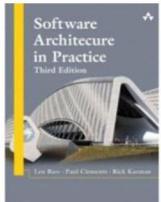
What is a Pattern_2?

- A *solution*. A successful architectural resolution to the problem, **appropriately abstracted**. The solution for a pattern is determined and described by:
 - A set of **element types** (for example, **data repositories, processes, and objects**)
 - A set of **interaction mechanisms or connectors** (for example, **method calls, events, or message bus**)
 - A **topological layout of the components**
 - A set of **semantic constraints** covering **topology, element behavior, and interaction mechanisms**



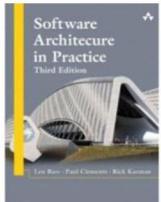
Kalıp Nedir_2?

- *Bir Çözüm.* Problemin, uygun şekilde soyutlanmış başarılı bir mimarisi. Bir kalıp için çözüm sunlar tarafından belirlenir ve tanımlanır:
 - **Unsur tipleri**kümesi (ör. Veri depoları, süreçler ve nesneler)
 - **Etkileşim mekanizmaları ve bağlayanları**kümesi (ör. Metod çağrısı, olaylar, mesaj yolu)
 - **Bileşenlerin topolojik deseni**
 - Topolojiyi, eleman davranışını ve etkileşim mekanizmalarını kapsayan **anlamsal kısıtlar**kümesi



Overview_1

- useful and widely used patterns.
- This catalog is not meant to be exhaustive
- it is meant to be representative.
- patterns of runtime elements (such as broker or client-server)
- design-time elements (such as layers).
- For each pattern we list the *context*, *problem*, and *solution*. As part of the solution, we briefly describe the *elements*, *relations*, and *constraints* of each pattern.



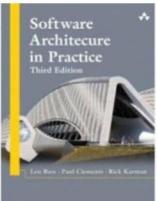
Genel Bakış_1

- Faydalı ve yaygın olarak kullanılan kalıplar.
- Bu katalogun kapsamlı olması amaçlanmamıştır
 - Temsili olması amaçlanmıştır.
- koşma zamanı öğelerinin kalıpları (aracı veya istemci-sunucu gibi)
- tasarım zamanı öğeleri (tabakalar/(katmanlar gibi)).
- Her kalıp için *bağlamı, sorunu ve çözümü* listeliyoruz. Çözümün bir parçası olarak, her kalının unsurlarını, ilişkilerini ve kısıtlarını kısaca açıklıyoruz.



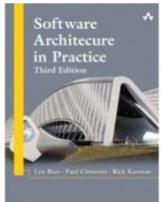
Overview_2

- Applying a pattern is not an all-or-nothing proposition.
- in practice architects may choose to violate them in small ways when there is a good design tradeoff to be had
- Patterns can be categorized by the dominant type of elements that they show:
 - module patterns show modules,
 - component-and-connector (C&C) patterns show components and connectors, and
 - allocation patterns show a combination of software elements (modules, components, connectors) and non-software elements



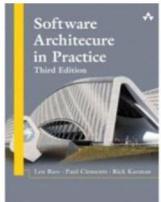
Genel Bakış_2

- Bir kalıp uygulaması “ya hep ya hiç” önerisi değildir.
- Uygulamada mimarlar, iyi bir tasarım ödeşimi olduğunda bunları küçük şekillerde ihlal etmeyi seçebilirler.
- Kalıplar, gösterdikleri baskın öğe türlerine göre kategorize edilebilir:
 - modül kalıpları modüllerini gösterir,
 - bileşen ve bağlayan (C&C) kalıpları bileşenleri ve bağlayıcıları gösterir ve
 - tahsis kalıpları, yazılım öğelerinin (modüller, bileşenler, bağlayıcılar) ve yazılım dışı öğelerin bir kombinasyonunu gösterir



Layered Pattern_1

- **Context:** All complex systems experience the need to develop and evolve portions of the system independently. For this reason the developers of the system need a clear and well-documented separation of concerns, so that modules of the system may be independently developed and maintained.



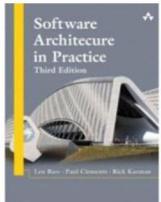
Layered Pattern_1

- **Bağlam:**
- Bütün karmaşık (girift) sistemler bir **sistemin parçalarını** bağımsız olarak geliştirmek ve evriltmek ihtiyacını yaşıar.
- Bu sebeple sistem geliştiricilerinin açık ve iyi dökümantte edilmiş **ilgi ayrımlına** ihtiyaç duyarlar. Öyle ki sistem modülleri **bağımsız** olarak geliştirilebil sin ve idame edilebil sin.



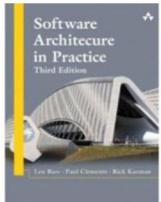
Layered Pattern_2

- **Problem:**
- The software needs to be segmented in such a way that the **modules** can be developed and evolved **separately**
- with little interaction among the parts,
- supporting **portability, modifiability, and reuse.**



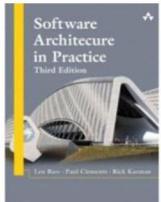
Layered Pattern_2

- **Problem:** Yazılım, öyle bir şekilde bölümlenmeye ihtiyaç duyar ki modüller,
 - taşınabilirliği,
 - değiştirilebilirliği ve
 - tekrar kullanımı destekleyerek
- parçalar arasında zayıf bir etkileşim ile geliştirilebilsin ve evriltilebilsin.



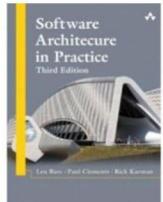
Layered Pattern_3

- **Solution:** To achieve this **separation of concerns**, the layered pattern divides the software into units called **layers**.
- Each **layer** is a grouping of modules that offers a cohesive set of services.
- The usage must be **unidirectional**.
- Layers completely partition a set of software, and each partition is exposed through a public interface.

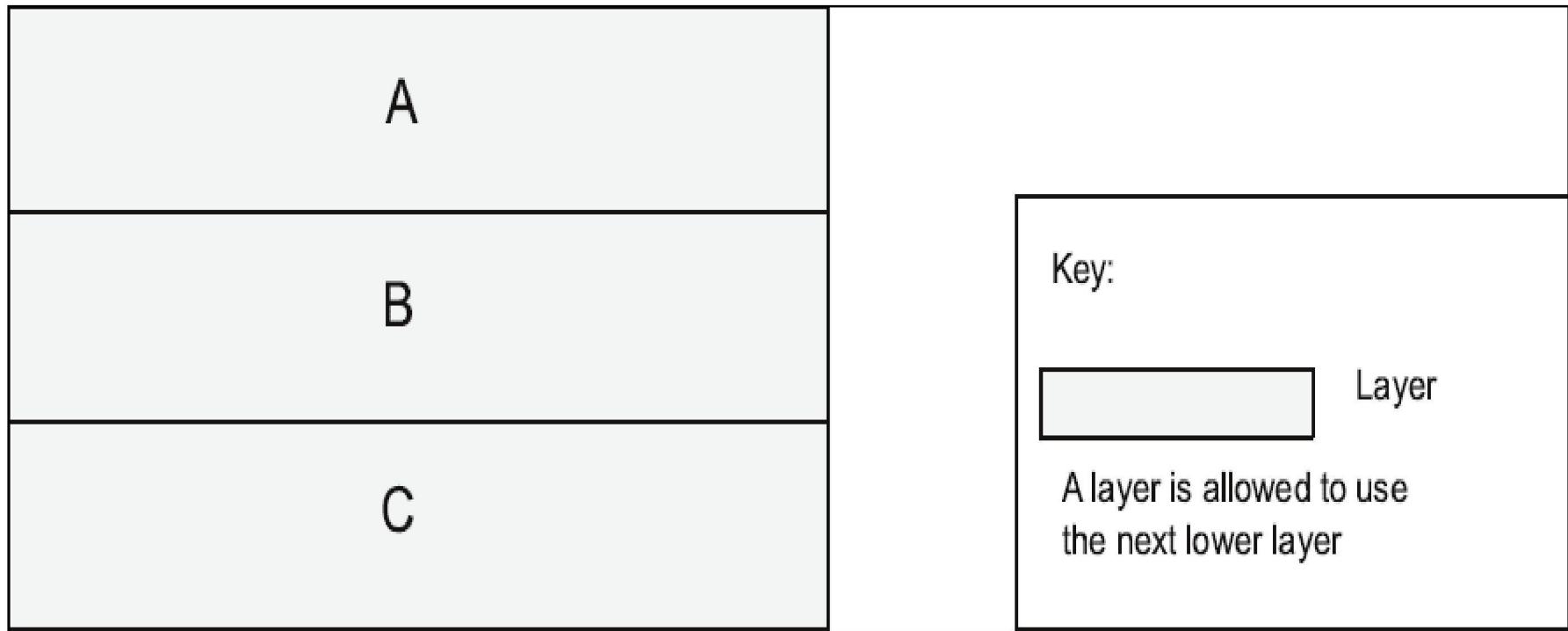


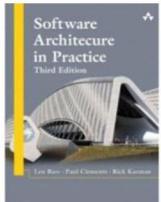
Layered Pattern_3

- Çözüm:
- Bu **ilgi ayrımlına** ulaşmak için tabakalı(katmanlı) kalıp, yazılımı **tabakalara** böler.
- Her tabaka sıkı hizmetler kümesini veren bir **modüller gruplamasıdır**.
- Bu kullanım **tek yönlü** olmalıdır.
- Tabakalar tamamen bir yazılım kümesi bütünüdür ve her bölüm bir genel arayüz ile ortaya çıkar.



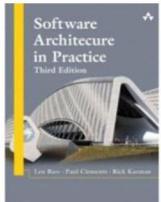
Layered Pattern Example





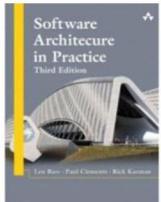
Layered Pattern Solution_1

- **Overview:**
 - The layered pattern defines layers (groupings of modules that offer a cohesive set of services) and a unidirectional *allowed-to-use* relation among the layers.
- **Genel Bakış:**
 - Tabakalı kalıp **tabakalar** (sıkı hizmetler kümesini veren bir modüller grublaması) ve tabakalar arasında tek yönlü ‘**kullanma izinli**’ ilişkisini tanımlar.



Layered Pattern Solution_2

- Elements:
- *Layer*, a kind of module. The description of a layer should define what modules the layer contains.
- Relations: *Allowed to use*. The design should define what the layer usage rules are and any allowable exceptions.



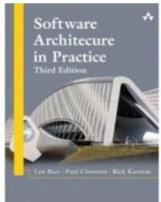
Layered Pattern Solution_2

- **Unsurlar:**

- *Tabaka* bir çeşit modüldür. Tabaka tanımı içерdiği modülleri tanımlamalıdır.

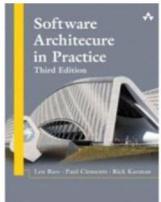
- **İlişkiler:**

- *Kullanma izinli.* **Tasarım** tabaka kullanım kurallarını ve müsaade edilen istisnaları tanımlamalıdır.



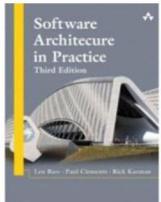
Layered Pattern Solution_3

- Constraints:
 - Every piece of software is allocated to exactly one layer.
 - There are at least two layers (but usually there are three or more).
 - The *allowed-to-use* relations should not be circular (i.e., a lower layer cannot use a layer above).



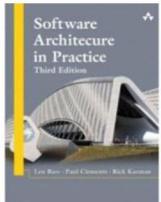
Layered Pattern Solution_3

- **Kısıtlar:**
 - Her yazılım parçası tam olarak bir tabakaya tahsis edilir.
 - En az iki tabaka olmalıdır (genellikle üç veya daha fazla).
 - *'Kullanım izinli'* ilişkisi dairevi olmamalıdır (ör. Alt bir tabaka bir üstteki tabakayı kullanamaz).



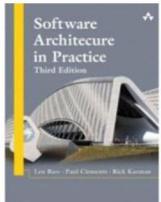
Layered Pattern Solution_4

- **Weaknesses:**
 - The addition of layers adds up-front cost and complexity to a system.
 - Layers contribute a performance penalty.
- **Zayıf yönleri:**
 - Tabakaların eklenmesi, bir sisteme önyüz maliyeti ve karmaşıklık ekler.
 - Tabakalar performans kaybını artırır.



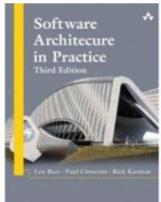
Layered Pattern

- Separation of concerns among components.
- Layers of isolation.
 - The layers of isolation concept means that changes made in one layer of the architecture generally don't impact or affect components in other layers
 - The layers of isolation concept also means that each layer is independent of the other layers, thereby having little or no knowledge of the inner workings of other layers in the architecture.

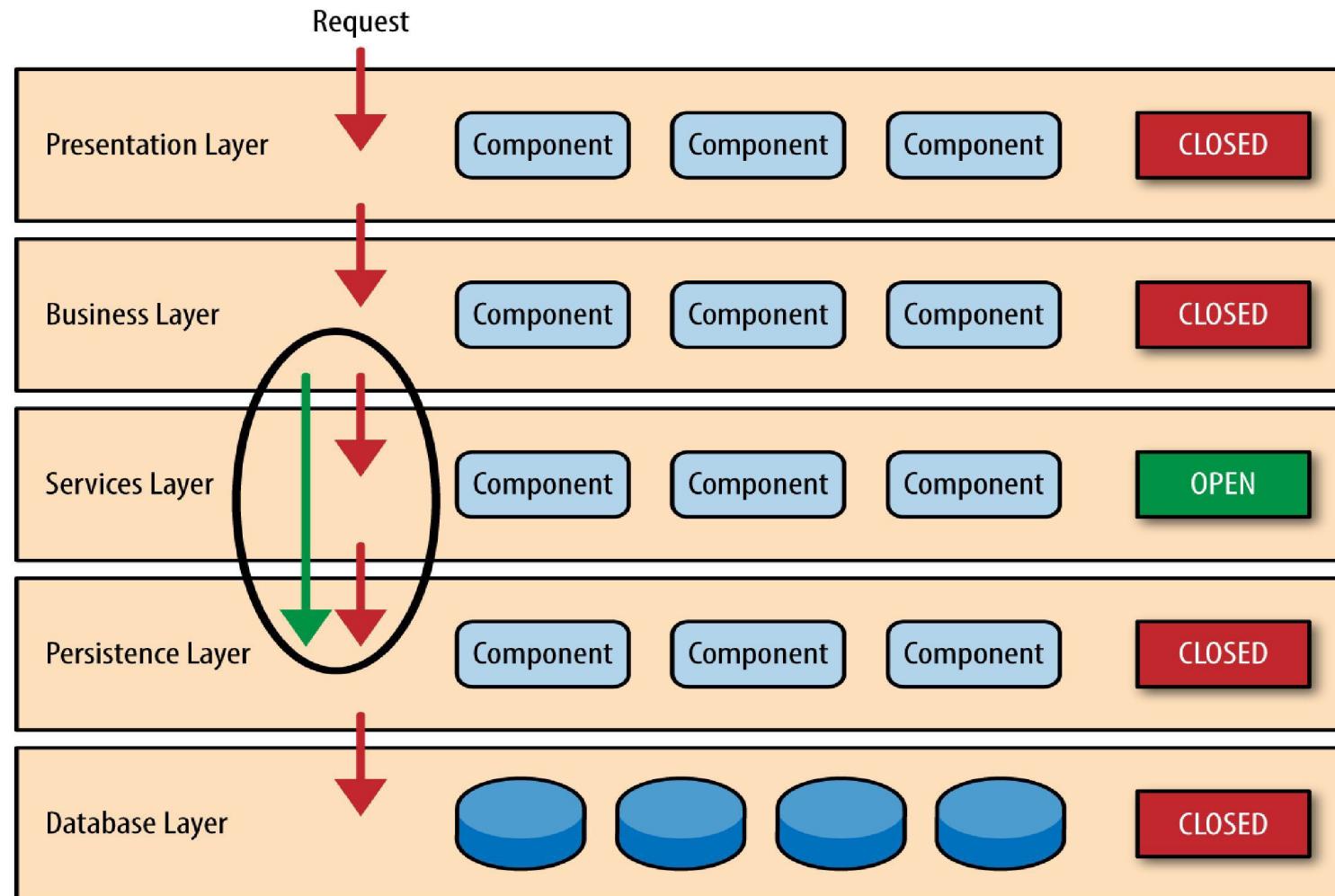


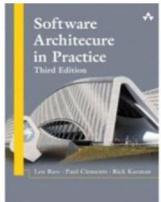
Tabakalı Kalıp

- Bileşenler arasında **ilgi ayrımı**.
- **Yalıtım tabakaları**
 - Yalıtım tabakaları kavramı mimarinin bir katmanında yapılan değişikliklerin genellikle diğer katmanlardaki bileşenleri etkilememesi demektir.
 - Yalıtım tabakaları kavramı aynı zamanda bir katmanın diğer katmanlardan bağımsız olması ve böylece mimarideki diğer katmanların iç çalışmasını bilmemesi veya çok az bilmesi demektir.

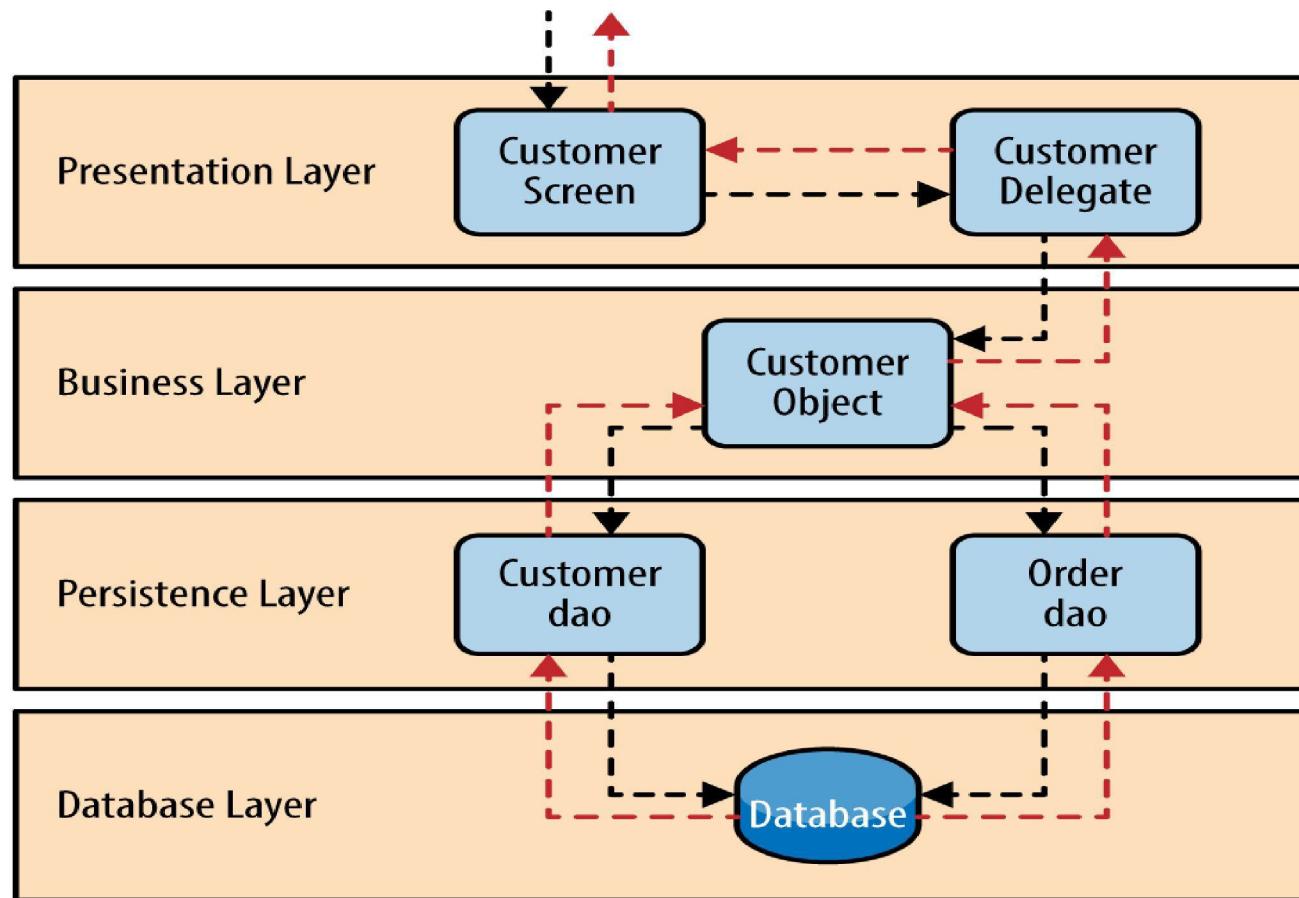


Layered Pattern





Layered Pattern Example

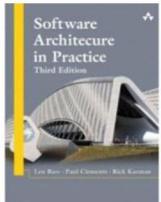


dao: data access object



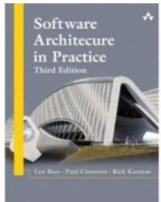
Model-View-Controller Pattern_1

- **Context:** User interface software is typically the most frequently modified portion of an interactive application. Users often wish to look at data from different perspectives, such as a bar graph or a pie chart. These representations should both reflect the current state of the data.



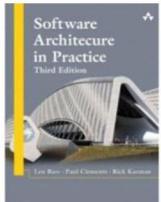
Model-Görünüm-Denetleyici Kalıbı_1

- **Bağlam:**
- Kullanıcı arayüzü yazılımı, etkileşimli bir uygulamanın tipik olarak en sık değiştirilen kısmıdır.
- Kullanıcılar sıkılıkla verilere çubuk grafik veya pasta grafik gibi farklı perspektiflerden bakmak ister. Bu temsillerin her ikisi de verilerin mevcut durumunu yansıtmalıdır.



Model-View-Controller Pattern_2

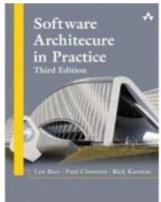
- Problem:
- How can user **interface functionality** be kept separate from **application functionality** and yet still be **responsive to user input**, or to changes in the underlying application's data?
- And how can multiple views of the user interface be created, maintained, and coordinated when the underlying **application data changes**?



Model-Görünüm-Denetleyici Kalıbı_2

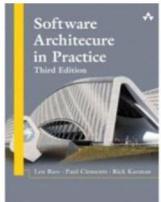
- **Problem:**

- Kullanıcı arabirimini işlevselliği, **uygulama işlevselliğinden** nasıl ayrı tutulabilir ve
- hatta kullanıcı girişine veya **temel uygulama verilerindeki değişikliklere** nasıl cevap verebilir ?
- Ve **temel uygulama verileri değiştiğinde**, kullanıcı arayüzünün çoklu görünümleri nasıl oluşturulabilir, idame edilebilir ve koordine edilebilir?



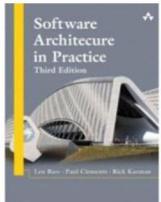
Model-View-Controller Pattern_3

- **Solution:** The model-view-controller (MVC) pattern **separates application functionality** into three kinds of components:
 - A **model**, which contains the application's data
 - A **view**, which displays some portion of the underlying data and interacts with the user
 - A **controller**, which mediates between the model and the view and manages the **notifications of state changes**

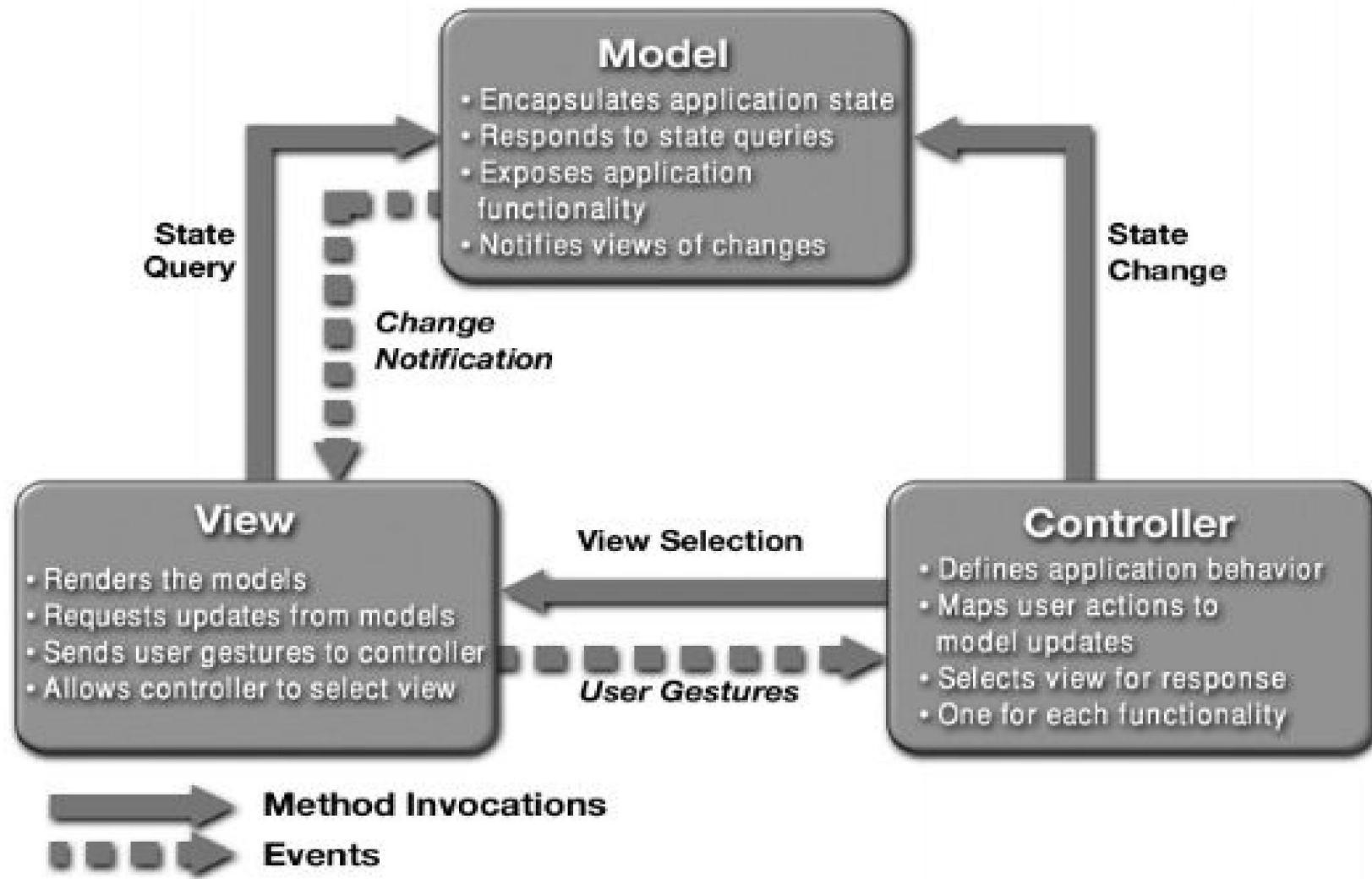


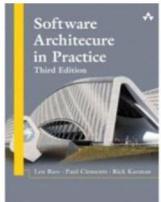
Model-Görünüm-Denetleyici Kalıbü_3

- **Çözüm:**
- Model Görünüm Denetleyici (MVC) modeli, uygulama işlevsellliğini üç tür bileşene ayırır:
 - Uygulama verilerini içeren **bir model**
 - Temel verilerin bir kısmını görüntüleyen ve kullanıcıyla etkileşime giren bir **görünüm**
 - Model ve görünüm arasında aracılık eden ve durum değişiklerinin bildirimlerini yöneten bir **denetleyici**



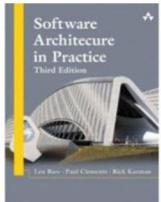
MVC Example





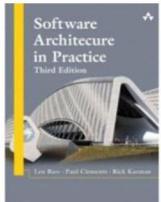
MVC Example- 1

- Model
 - encapsulates application state
 - responds to state queries
 - exposes application functionality
 - modifies views of changes
- View
 - renders the models
 - requests updates from models
 - sends user gestures to controller
 - allows controller to select view
- Controller
 - defines application behaviour
 - maps user actions to model updates selects view for response
 - one for each functionality



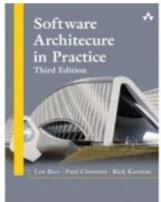
MVC Örneği- 1

- Model
 - uygulama durumunu sarmalar (kapsüller)
 - durum sorgularına karşılık verir
 - uygulama işlevsellliğini ortaya çıkarır
 - değişikliklerin görünümlerini değiştirir
- View
 - modelleri işler
 - modellerden güncelleme talep eder
 - kullanıcı jestlerini denetleyiciye gönderir
 - denetleyicinin görünümü seçmesine imkân verir
- Controller
 - uygulama davranışını tanımlar
 - kullanıcı eylemlerini model güncellemelerine map eder, tepki için view seçer
 - her işlev için bir tane



MVC Çözümü - 1

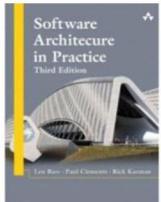
- **Overview:**
 - The MVC pattern breaks system functionality into three components: a model, a view, and a controller that mediates between the model and the view.
- **Genel Bakış:**
 - MVC kalımı, sistem işlevsellliğini üç bileşene ayırır: model, görünüm ve model ile görünüm arasında aracılık eden bir denetleyici.



MVC Solution - 2

- Elements:

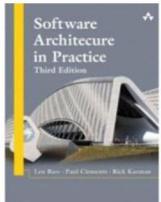
- The *model* is a representation of the application data or state, and it contains (or provides an interface to) application logic.
- The *view* is a user interface component that either produces a representation of the model for the user or allows for some form of user input, or both.
- The *controller* manages the interaction between the model and the view, translating user actions into changes to the model or changes to the view.



MVC Çözümü - 2

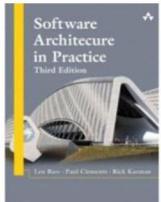
- **Unsurlar:**

- **Model**, uygulama verilerinin veya durumunun bir temsilidir ve uygulama mantığını içerir (veya buna bir arayüz sağlar).
- **Görünüm**, kullanıcı için modelin bir temsilini üreten veya bazı kullanıcı girdi biçimlerine veya her ikisine imkân veren bir kullanıcı arayüzü bileşenidir.
- **Denetleyici**, kullanıcı eylemlerini modeldeki değişikliklere veya görünümdeki değişikliklere dönüştürerek model ve görünüm arasındaki etkileşimi yönetir.



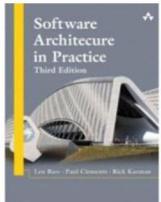
MVC Çözümü - 3

- **Relations:**
- The '*notifies*' relation connects instances of model, view, and controller, notifying elements of relevant state changes.
- **İlişkiler:**
- '*Bildirir*' ilişkisi, model, görünüm ve denetleyici oluşlarını birbirine bağlayıp ilgili durum değişiklik öğelerini bildirir.



MVC Solution - 4

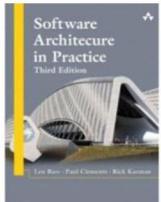
- Constraints:
 - There must be at least one instance each of model, view, and controller.
 - The model component should not interact directly with the controller.



MVC Çözümü - 4

- **Kısıtlar:**

- Model, görünüm ve denetleyicinin her birinin en az bir oluşu olmalıdır.
- Model bileşeni, denetleyici ile doğrudan etkileşime girmemelidir.



MVC Çözümü - 5

- **Weaknesses:**

- The complexity may not be worth it for simple user interfaces.
- The model, view, and controller abstractions may not be good fits for some user interface toolkits.

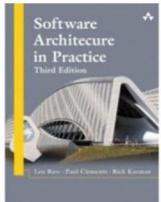
- **Zayıf yönler:**

- Katlanılan karmaşıklık basit kullanıcı arayüzleri için değimeyebilir.
- Model, görünüm ve denetleyici soyutlamaları, bazı kullanıcı arayüzü araç takımları için uygun olmayabilir.



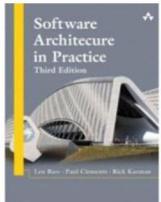
Client-Server Pattern_1

- **Context:** There are shared resources and services that large numbers of distributed clients wish to access, and for which we wish to control access or quality of service.



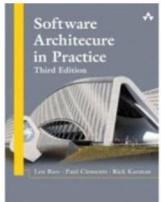
İstemci-Sunucu Kalıbı_1

- **Bağlam:**
- There are shared resources and services that large numbers of distributed clients wish to access, and for which we wish to control access or quality of service.



Client-Server Pattern_2

- **Problem:** By managing a set of shared resources and services, we can promote modifiability and reuse, by factoring out common services and having to modify these in a single location, or a small number of locations. We want to improve scalability and availability by centralizing the control of these resources and services, while distributing the resources themselves across multiple physical servers.



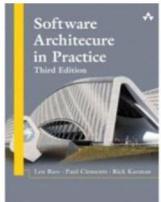
İstemci-Sunucu Kalıbı _2

- **Problem:**
- Ortak kaynaklar ve hizmetler kümесини yönetirken, genel hizmetleri hesaba katmakla ve bu değişiklikleri **tek veya az sayıda yerde yapmakla , yeniden kullanımı teşvik** edebiliriz.
- Kaynakları kendisini birden çok sunucuya (fizikî) dağıtırken, bu kaynakların ve hizmetlerin kontrolünü merkezileştirek
 - Ölçeklenebilirliği ve erişilebilirliği geliştirmek isteriz



Client-Server Pattern _3

- **Solution:**
- Clients interact by requesting services of servers, which provide a set of services.
- Some components may act as both clients and servers.
- There may be one central server or multiple distributed ones.

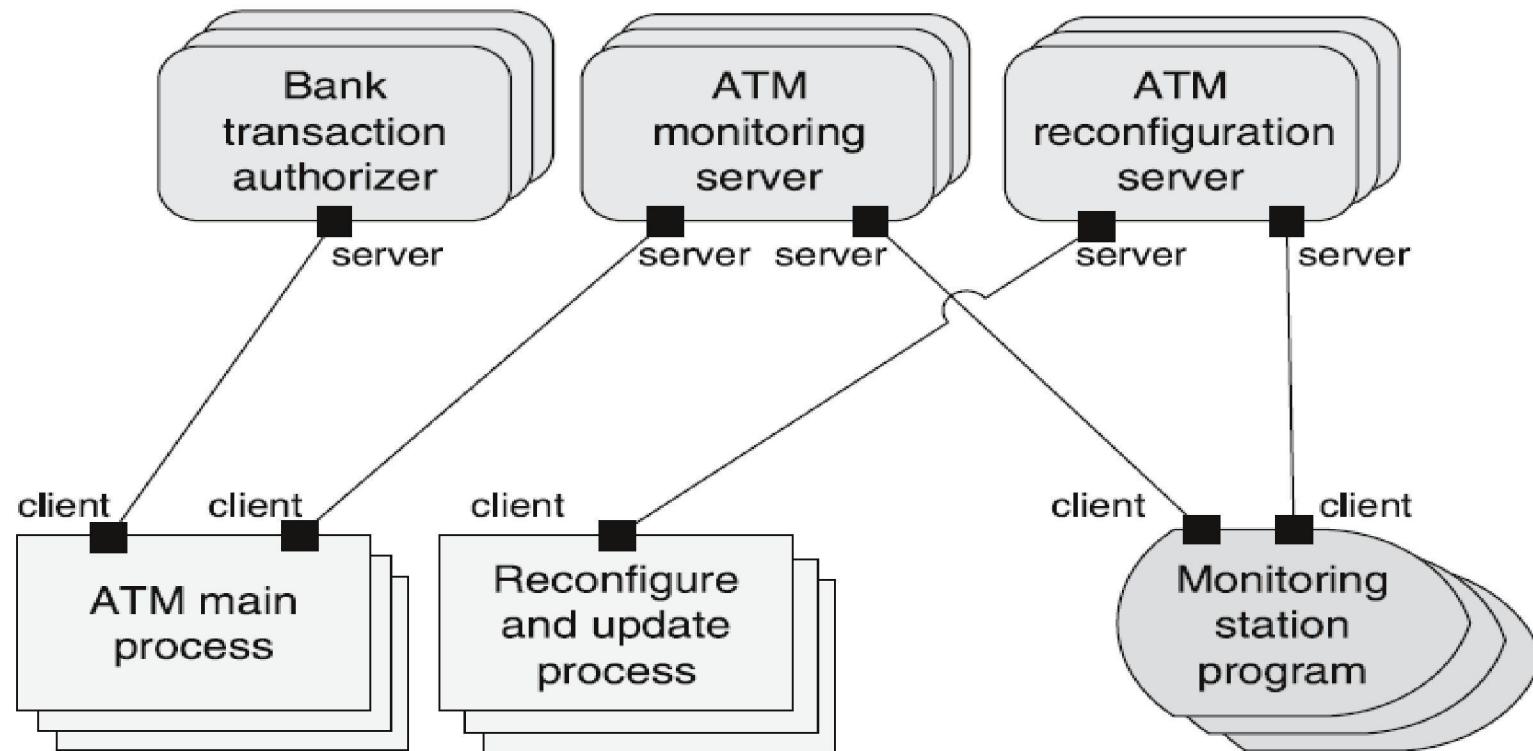


İstemci-Sunucu Kalıbı_3

- **Çözüm:**
- İstemciler bir dizi hizmet sağlayan sunuculardan hizmet talebiyle etkileşimde bulunurlar.
- Bazı bileşenler hem istemci hem de sunucu gibi davranabilirler.
- Tek bir merkezî veya birden çok dağıtık sunucu olabilir.



Client-Server Example



Key:



Client



Server

TCP socket connector with
client and server ports



FTX server
daemon



ATM OS/2
client process

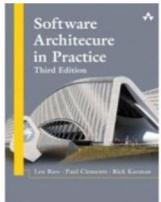


Windows
application



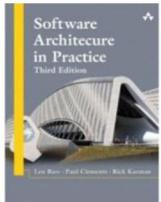
Client-Server Solution - 1

- **Overview:**
- Clients initiate interactions with servers, invoking services as needed from those servers and waiting for the results of those requests.



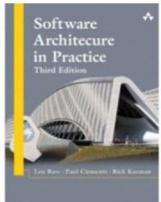
İstemci-Sunucu Çözümü- 1

- **Genel Bakış:**
- İstemciler sunucularla ihtiyaç duydukları hizmetleri isteyerek ve bu taleplerin sonucunu bekleyerek etkileşimi başlatırlar.



Client-Server Solution - 2

- Elements:
 - *Client*, a component that invokes services of a server component. Clients have ports that describe the services they require.
 - *Server*: a component that provides services to clients. Servers have ports that describe the services they provide.



İstemci-Sunucu Çözümü - 2

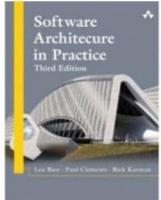
- **Unsurlar:**

- *İstemci* sunucuya hizmet almak için başvuran bileşendir. İstemciler istedikleri hizmetleri tarif eden portlara sahiptir.
- *Sunucu* istemcilere hizmetleri sağlayan bileşendir. Sunucular sağladıkları hizmetleri tarif eden portlara sahiptir.



Client-Server Solution - 3

- *Request/reply connector:*
- A data connector employing a request/reply protocol, used by a client to invoke services on a server.
- Important characteristics include whether the calls are local or remote, and whether data is encrypted.



İstemci-Sunucu Çözümü - 3

- *Talep/cevap bağlayıcısı:*
- Bir istemci tarafından bir sunucuya hizmet başvurusu için kullanılan ve talep/cevap protokolü barındıran bir veri bağlayıcısıdır.
- Önemli özellikleri çağrıların yerel veya uzak olup olmaması ve verinin şifreli olup olmamasıdır.



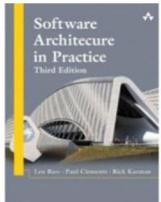
İstemci-Sunucu Çözümü - 4

- Relations: The '*attachment*' relation associates clients with servers.
- İlişkiler:
- '*Bağ'(rabıta)* ilişkisi istemcileri sunucular ile eşleştirir.



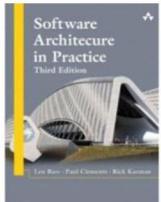
Client-Server Solution- 5

- Constraints:
 - Clients are connected to servers through request/reply connectors.
 - Server components can be clients to other servers.



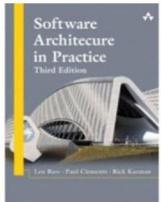
İstemci-Sunucu Çözümü - 5

- Kısıtlar:
 - İstemciler sunuculara talep/cevap bağlayıcıları ile bağlanırlar.
 - Sunucu bileşenleri diğer sunucuların istemcileri olabilir.



Client-Server Solution- 6

- Weaknesses:
 - Server can be a performance bottleneck.
 - Server can be a single point of failure.
 - Decisions about where to locate functionality (in the client or in the server) are often complex and costly to change after a system has been built.



İstemci-Sunucu Çözümü - 6

- Zayıf yönleri:
 - Sunucu performans darboğazına girebilir.
 - Sunucu arızanın tek merkezi olabilir.
 - İşlevselligin hangi tarafa (sunucuya mı istemciye mi) yerleştirileceğine dair kararlar sıkılıkla karmaşıktır ve sistem kuruluktan sonra değiştirilmesi maliyetlidir.