T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

EĞİTİM ÖĞRETİM BİLGİ SİSTEMLERİNDE TASARIM KALIPLARI KULLANIMI

YÜKSEK LİSANS TEZİ

Bilg.Müh. Serkan DARGA

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ

Tez Danışmanı : Yrd. Doç. Dr. Hayrettin EVİRGEN

Ocak 2012

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

EĞİTİM ÖĞRETİM BİLGİ SİSTEMLERİNDE TASARIM KALIPLARI KULLANIMI

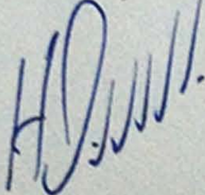
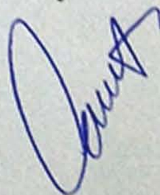
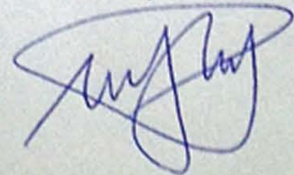
YÜKSEK LİSANS TEZİ

Bilg.Müh. Serkan DARGA

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ

Bu tez 05/01/2012 tarihinde aşağıdaki jüri tarafından Oybirliği ile kabul edilmiştir.

Yrd.Doç.Dr. Hayrettin EVİRGEN	Doç.Dr. Cemil ÖZ	Yrd.Doç.Dr. Mustafa TURAN
Jüri Başkanı	**Üye**	**Üye**

ÖNSÖZ

Yüksek lisans öğrenimim ve tez çalışmam boyunca gösterdiği yakın ilgi ve katkılarından dolayı değerli hocam Yrd. Doç. Dr. Hayrettin EVİRGEN'e teşekkür ederim.

Çalışma boyunca verdikleri destek nedeni ile başta Özkan CANAY, Metin VARAN, Ali DURDU ve Ahmet ŞANSLI olmak üzere tüm SAÜ Bilgisayar Araştırma ve Uygulama Merkezi çalışanlarına teşekkür ederim.

Çalışmalarımın sonuna kadar gösterdikleri sabır ve manevi desteklerinden dolayı anneme ve eşime teşekkür ederim

İÇİNDEKİLER

ÖNSÖZ.. ii

İÇİNDEKİLER.. iii

SİMGELER VE KISALTMALAR LİSTESİ.. vi

ŞEKİLLER LİSTESİ.. vii

TABLOLAR LİSTESİ.. ix

ÖZET.. x

SUMMARY.. xi

BÖLÜM 1.

GİRİŞ.. 1

1.1. Tez İçeriği ve Tezin Organizasyonu.. 5

BÖLÜM 2.

BOLOGNA SÜRECİ VE EĞİTİM ÖĞRETİM BİLGİ SİSTEMLERİ........... 7

2.1. Bologna Süreci.. 7

2.2. Bologna Süreci Eylem Başlıklarının Türkiye Yüksek Öğretiminde Uygulanması Çalışmaları.. 8

2.2.1. AKTS sisteminin uygulanması çalışmaları........................... 9

2.2.2. Ulusal yeterlilikler çerçevesinin tanımlanması çalışmaları.... 10

2.2.3. Ulusal yeterlilikler çerçevesi kapsamında üniversitelerde yapılan çalışmalar.. 11

2.3. Eğitim Öğretim Bilgi Sistemi Yazılım Çerçevesi Tasarımı............ 12

2.3.1. Eğitim öğretim bilgi sistemi yazılımlarının sahip olması gereken özellikler.. 12

BÖLÜM 3.
TASARIM PRENSİPLERİ VE TASARIM KALIPLARI.............................. 14
3.1. Kötü Tasarım Belirtileri.. 14
3.1.1. Genişlemeye kapalı olmak.. 14
3.1.2. Kırılganlık... 15
3.1.3. Taşınamazlık... 15
3.1.4. Akışkanlığın az olması... 15
3.2. Nesneye Yönelik Tasarımın Temel Prensipleri............................... 15
3.2.1. Açık kapalı prensibi.. 16
3.2.2. Liskov ayrışabilme prensibi.. 17
3.2.3. Bağımlılığın azaltılması prensibi... 17
3.2.4. Tek sorumluluk prensibi.. 18
3.2.5. Arayüz ayırma prensibi... 19
3.3. Tasarım Kalıplarına Genel Bakış.. 19
3.4. Tasarım Kalıbı Nedir.. 20
3.5. Tasarım Kalıbı Dokümanı.. 21
3.6. Tasarım Kalıplarının Sınıflandırılması... 22
3.6.1. Oluşturucu tasarım kalıpları.. 23
3.6.1.1. Soyut fabrika tasarım kalıbı..................................... 24
3.6.1.2. Yapıcı tasarım kalıbı.. 25
3.6.1.3. Fabrika metodu tasarım kalıbı................................. 26
3.6.1.4. Prototip tasarım kalıbı.. 26
3.6.1.5. Tekil tasarım kalıbı.. 27
3.6.2. Yapısal tasarım kalıpları... 28
3.6.2.1. Adaptör tasarım kalıbı.. 28
3.6.2.2. Köprü tasarım kalıbı.. 29
3.6.2.3. Bileşik tasarım kalıbı... 30
3.6.2.4. Dekoratör tasarım kalıbı.. 31
3.6.2.5. Önyüz tasarım kalıbı.. 32
3.6.2.6. Sineksiklet tasarım kalıbı... 33
3.6.2.7. Vekil tasarım kalıbı.. 34
3.6.3. Davranışsal tasarım kalıpları... 35
3.6.3.1. Sorumluluklar zinciri tasarım kalıbı........................ 36

3.6.3.2. Komut tasarım kalıbı... 37
3.6.3.3. Yorumlayıcı tasarım kalıbı....................................... 37
3.6.3.4. Tekrarlayıcı tasarım kalıbı....................................... 38
3.6.3.5. Arabulucu tasarım kalıbı.. 39
3.6.3.6. Hatırlayıcı tasarım kalıbı.. 40
3.6.3.7. Gözlemci tasarım kalıbı.. 41
3.6.3.8. Durum tasarım kalıbı.. 42
3.6.3.9. Strateji tasarım kalıbı.. 43
3.6.3.10. Kalıp yordamı tasarım kalıbı.................................. 44
3.6.3.11. Ziyaretçi tasarım kalıbı... 45

BÖLÜM 4.
EOBS UYGULAMASI... 46
4.1. EOBS Çerçeve Yazılımının Amacı.. 47
4.2. EOBS Çerçeve Yazılımı Mimarisi.. 48
4.3. Bölüm / Program Bilgileri Modülünde Kullanılan Tasarım Kalıpları.. 50
4.4. Ders Bilgileri Modülünde Kullanılan Tasarım Kalıpları................. 52
4.5. Uygulamanın Genel İşleyişinde Kullanılan Tasarım Kalıpları........ 54
4.5.1. Kullanıcı doğrulama işlemleri.. 55
4.5.2. Menü yapısının oluşturulması... 56
4.5.3. Veritabanı işlemleri.. 58
4.6. Kullanılan Tasarım Kalıplarının EOBS Çerçeve Yazılımına Etkileri.. 62

BÖLÜM 5.
SONUÇLAR VE ÖNERİLER... 65

KAYNAKLAR…………………………………………………………….. 67
ÖZGEÇMİŞ……………………………………………………………….. 70

SİMGELER VE KISALTMALAR LİSTESİ

AKTS	: Avrupa Kredi Transfer Sistemi
CBO	: Nesneler arası bağımlılık
DIT	: Miras alma ağacının derinliği
ECTS	: European Credit Transfer System
EOBS	: Eğitim Öğretim Bilgi Sistemi
GOF	: Gangs of Four
LDAP	: Lightweight Directory Access Protocol
NOC	: Bir sınıftan doğrudan türetilen sınıf sayısı
OOPSLA	: Object-Oriented Programming, Systems, Languages and Applications
PLOP	: Pattern Languages of Programs
POP	: Post Office Protocol
QF-EHEA	: Qualifications Framework for European Higher Education Area
SQL	: Structured Query Language
UML	: Unified Modeling Language
UYÇ	: Ulusal Yeterlilikler Çerçevesi
XML	: Extensible Markup Language

ŞEKİLLER LİSTESİ

Şekil 2.1.	Bologna Süreci İlişkileri..	11
Şekil 3.1.	Baglan Sınıfının Değişikliğe Kapalı Olduğunu Gösteren UML Şeması..	16
Şekil 3.2.	Liskov Ayrışabilme Prensibi UML Şeması.................................	17
Şekil 3.3.	Bağımlılığın Azaltılması Prensibi UML Şeması...........................	18
Şekil 3.4.	Arayüz Ayırma Prensibi UML Şeması..	19
Şekil 3.5.	Soyut Fabrika Tasarım Kalıbı UML Şeması................................	24
Şekil 3.6.	Yapıcı Tasarım Kalıbı UML Şeması..	25
Şekil 3.7.	Fabrika Metodu Tasarım Kalıbı UML Şeması..............................	26
Şekil 3.8.	Prototip Tasarım Kalıbı UML Şeması..	27
Şekil 3.9.	Tekil Tasarım Kalıbı UML Şeması...	28
Şekil 3.10.	Adaptör Tasarım Kalıbı UML Şeması..	29
Şekil 3.11.	Köprü Tasarım Kalıbı UML Şeması...	30
Şekil 3.12.	Bileşik Tasarım Kalıbı UML Şeması..	31
Şekil 3.13.	Dekoratör Tasarım Kalıbı UML Şeması.......................................	32
Şekil 3.14.	Önyüz Tasarım Kalıbı UML Şeması..	33
Şekil 3.15.	Sineksiklet Tasarım Kalıbı UML Şeması.....................................	34
Şekil 3.16.	Vekil Tasarım Kalıbı UML Şeması..	35
Şekil 3.17.	Sorumluluklar Zinciri Tasarım Kalıbı UML Şeması....................	36
Şekil 3.18.	Komut Tasarım Kalıbı UML Şeması..	37
Şekil 3.19.	Yorumlayıcı Tasarım Kalıbı UML Şeması..................................	38
Şekil 3.20.	Tekrarlayıcı Tasarım Kalıbı UML Şeması...................................	39
Şekil 3.21.	Arabulucu Tasarım Kalıbı UML Şeması......................................	40
Şekil 3.22.	Hatırlayıcı Tasarım Kalıbı UML Şeması......................................	41
Şekil 3.23.	Gözlemci Tasarım Kalıbı UML Şeması.......................................	42
Şekil 3.24.	Durum Tasarım Kalıbı UML Şeması..	42

Şekil 3.25. Strateji Tasarım Kalıbı UML Şeması... 43
Şekil 3.26. Kalıp Yordamı Tasarım Kalıbı UML Şeması.............................. 44
Şekil 3.27. Ziyaretçi Tasarım Kalıbı UML Şeması.. 45
Şekil 4.1. EOBS Çerçeve Yazılımında Kullanılan Temel Modüller............. 49
Şekil 4.2. Bölüm/Pregram Modülünde Kullanılan Bileşik ve Strateji Tasarım Kalıbı UML Şeması.. 51
Şekil 4.3. Ders Modülünde Kullanılan Strateji Tasarım Kalıbı UML Şeması... 52
Şekil 4.4. Ders Modülünde Kullanılan Gözlemci Tasarım Kalıbı UML Şeması... 53
Şekil 4.5. Ders Modülünde Kullanılan Vekil Tasarım Kalıbı UML Şeması... 54
Şekil 4.6. Doğrulama İşleminde Kullanılan Fabrika Metodu Tasarım Kalıbı UML Şeması... 56
Şekil 4.7. Menülerde Kullanılan Bileşik ve Strateji Tasarım Kalıpları UML Şeması... 57
Şekil 4.8. Veritabanı Bağlantısı İçin Kullanılan Tekil Tasarım Kalıbı UML Şeması... 59
Şekil 4.9. .NET Framework İçerisinde Bulunan Connection Sınıflarının UML Şeması... 60
Şekil 4.10. .NET Framework İçerisinde Bulunan Factory Sınıflarının UML Şeması... 61
Şekil 4.11. Modüller Arası İlişkiler……... 62

TABLOLAR LİSTESİ

Tablo 3.1. Tasarım Kalıplarının Sınıflandırılması... 23
Tablo 4.1. Modüller Arası Bağımlılıklar... 63

ÖZET

Anahtar kelimeler: Tasarım prensipleri, Tasarım kalıpları, Yazılım çerçeveleri, Eğitim öğretim bilgi sistemi

Bu çalışmada Bologna süreci kapsamında eğitim ve öğretimlerini düzenlemek isteyen üniversitelerin ihtiyaç duyacağı bilgi sistemleri için geliştirilecek yazılımlara temel oluşturacak bir yazılım çerçevesinin, tasarım prensipleri ve tasarım kalıpları kullanılarak geliştirilmesi incelenmiştir. Web tabanlı olarak geliştirilen Eğitim Öğretim Bilgi Sistemi Çerçeve yazılımı ile üniversitelerin eğitim-öğretim süreçlerinin tanımlı, şeffaf ve sürekli geliştirilebilir bir çerçeveye taşınması hedeflenmektedir. Geliştirilen bilgi sistemi aynı zamanda, üniversitenin diğer bilgi sistemleri ile entegre çalışan ve eğitim-öğretim süreçlerini geliştirmeye yönelik servis tabanlı hizmetler üreten özel bir çerçeve yazılımıdır.

Bir yazılım çerçevesinin sahip olması gereken nitelikler arasında yer alan taşınabilir ve geliştirilebilir olma gibi özellikleri sağlamada tasarım kalıpları önemli bir role sahiptir.

Bu çalışmada, geliştirilen çerçeve yazılımında uygulanan tasarım kalıpları ve uygulamaya kazandırdığı avantajlar incelenmiştir. Geliştirilen uygulamadan yola çıkarak tasarım kalıpları kullanmanın tekrar kullanılabilir, genişletilebilir, bakımı yapılabilir, kolay okunabilir ve kaliteli yazılım ürünleri oluşturmada önemli bir role sahip olduğu vurgulanmak istenmektedir.

DESIGN PATTERNS USAGE IN EDUCATION AND TRAINING INFORMATION SYSTEM

SUMMARY

Key Words: Design Principles, Design Patterns, Software Frameworks, Education and Training Information System

In this study, it is aimed developing software systems for universities which want to edit training and teaching procedures compatible with Bologna Process. Developed software framework is mentioned to be a basis for upgrading the scope of the information systems by using design principles and design patterns. Software Framework of Education and Training Information System was developed as a web based application. This property brings education and training processes into defined, transparent, continuously developable category. Developed information system is a service based specific software framework that integrated with other information systems in university.

Portable and continuously developable qualifications are requirement of a software framework development process. Design patterns have an important role in providing these qualifications.

In this study, it was also investigated the advantages of design patterns which were applied and implemented to developed software framework. By the route of developed application, roles of design patterns were discussed in context of creating qualified software product that comprises expandibility, maintainability, and easy readability features.

BÖLÜM 1. GİRİŞ

21. yüzyılda bilişim teknolojileri, tüm dünyada finans, iş ve eğlence sektörlerinin yanı sıra, eğitim alanında da bilgi toplumunu meydana getirme yolunda etkin bir şekilde kullanılmaktadır. Eğitim alanında ortaya çıkan ihtiyaçlara çözüm olarak önerilen yöntemlerin uygulanması sırasında bilişim teknolojilerinin kullanılması, eğitim sistemlerinin ortak bir kalite standardına sahip olmasına, eğitim sisteminin öğrenciye kazandıracağı yeterliliklerin öngörülebilir ve planlanabilir olmasına katkı sağlayacaktır.

Bilişim teknolojileri ve eğitim alanında yaşanan gelişmeler, her alanda olduğu gibi yükseköğretim alanında da eğitim sistemlerinin ve programlarının gözden geçirilmesini ve geliştirilmesini gündeme getirmiştir. Bunun sonucu olarak yaşam boyu öğrenme, öğrenci merkezli eğitim, çıktıya dayalı eğitim gibi her geçen gün önem kazanan yeni kavramlar ortaya çıkmıştır. Eğitim öğretim süreçlerini bu kavramramlar çerçevesinde düzenlemek isteyen yüksek öğretim kurumlarının, geçekleştirecekleri düzenlemeleri etkin ve hızlı bir şekilde yapabilmeleri için, diğer iş süreçlerinde olduğu gibi, bu alanda da bilişim teknolojilerini kullanma ihtiyaçları vardır.

Bologna Süreci, Avrupa Birliği'nin 1999 yılında yayınladığı "Bologna Bildirgesi" ile başlayan bir yükseköğretim reformu girişimidir. Hedef 2012 yılına kadar Avrupa Ülkeleri'nde kendi içinde uyumlu, birbirlerini karşılıklı olarak anlayan, tamamlayan ve rekabet gücü yüksek bir "Avrupa Yüksek Öğretim Alanı" oluşturulmasıdır [1]. Bu amaçla eğitim öğretim süreçlerinin güncellenmesi, tanımlı, şeffaf ve sürekli geliştirilebilir bir çerçeveye taşınması gerekmektedir. Bu amacı gerçekleştirmek için, yükseköğretim kurumları tarafından kabul gören ve uygulanabilen, ulusal ve uluslararası paydaşlarca tanınan derecelerin verilebileceği bir yeterlilik derecelendirme sistemini bünyesinde barındıran, aynı zamanda üniversitelerin diğer

bilgi sistemleri ile entegre çalışan ve eğitim öğretim süreçlerini geliştirmeye yönelik internet tabanlı bilgi sistemlerine ihtiyaç vardır [2].

İnternet tabanlı bilgi sistemleri, mimarileri tasarlanırken kaynak tüketimlerinin makul sınırlar içerisinde oluşturulması, mimarinin genişletilebilir bir yapıya sahip olması ve tasarlanan yazılım mimarisinin belli standartlar ölçüsünde olması gerekmektedir. Tasarlanan yazılım mimarisinin geliştirilen her uygulama için geliştiriciler tarafından tekrar tekrar öğrenilmesi, tasarlanması, geliştirilmesi ve bakımının yapılması kaynakların verimsiz kullanılması anlamına gelecektir [3]. Kaynakların verimsiz kullanılması problemini çözmenin bir yolu, ilgili programların ortak özelliklerini içinde barındıran ve uygulamalara temel bir çatı belirleyen yazılım çerçevelerinin kullanılmasıdır [4].

Üniversitelerin eğitim öğretim süreçleri Bologna Süreci kapsamında belirlendiğinden, bu sürece uygun eğitim öğretim programlarını belirlemek isteyen üniversitelerin yapacağı çalışmalar, ortak pek çok nokta barındırmaktadır. Bu ortak noktaları bünyesinde barındıran bir çerçeve yazılımı kullanılarak, tüm yükseköğretim kurumları, kendileri için en başından bir eğitim öğretim bilgi sistemi (EOBS) yazılımı geliştirme durumunda kalmayacaklardır. Her üniversitenin kendi EOBS yazılımını kendisinin geliştirmesi, yazılım mühendisliğinin temel kavramlarından olan “tekrar kullanılabilirlik” ve “taşınabilirlik” felsefesine uymamakta ve kaynakların aynı işlemler için tekrar tekrar harcanmasına sebep olmaktadır. Bu nedenle üniversitelerin eğitim öğretimlerini güncellemeleri için hazırlanacak projelere temel oluşturacak bir çerçeve yazılımı geliştirmek, gereksiz kaynak kullanımının önüne geçecektir.

Yazılım çerçevelerinin uygulama geliştirme ve bakımını yapma işlemlerinde verimli bir şekilde kullanılabilmesi için çerçevelerin kolay kullanılabilir, geliştirilebilir, bakım yapılabilir ve sağlam bir yapıya sahip olması gerekmektedir [5]. Çerçevelerin pek çok uygulamaya temel oluşturacak olması, çerçeve tasarlama ve geliştirme işini zorlaştırmaktadır. Çünkü çerçeveler, çerçeve temelinde geliştirilen uygulama özelinde belli bir seviyeye kadar geliştirilebilmelidir. Bu durumda çerçeve tasarımının esnek bir yapıya sahip olması gerekmektedir [6].

Çerçevelerin uzun ömürlü, yazılım geliştiriciler tarafından tercih edilir ve kullanılabilir olması önemlidir. Bahsedilen mimari özelliklerdeki tasarımlara sahip çerçevelerin geliştirilmesinde, geliştiricilerin tecrübeleri oldukça önemlidir. Ancak her uygulama geliştirme sürecinde, istenilen tecrübeye sahip insanların bulunması da zor bir iştir [7]. Bu durumda karşımıza tecrübeli tasarımcılar tarafından geliştirilmiş ve benzer problemlerin çözümünde kabul görmüş yapıların kullanımına olanak sağlayan tasarım kalıpları çıkmaktadır. Tasarım kalıpları kullanılarak istenilen tecrübeye sahip olmayan geliştiriciler de belli başlı tasarım prensiplerine uyarak uygulama geliştirme sürecine dahil olabileceklerdir.

Tasarım kalıpları, yazılım tasarımında ortaya çıkan benzer problemlerin genel çözümleridirler. Tasarım kalıpları belirli tasarım problemlerinin çözümü için esnek ve geliştirilebilir yapılar önerir [8]. Tasarım kalıpları, kullanılmış ve başarıya ulaşmış tasarım ve mimarilerin yeniden kullanılabilmesini sağlarlar. Bu şekilde üretilen sistemin de yeniden kullanılabilirliği artmaktadır. Zaten kullanılmış ve başarıya ulaşmış tasarımları kullanmak, uygulamaya özel problemlerin çözümü için iyi bir başlangıç yapmayı sağlar ve düşülebilecek potansiyel hataları engeller. Geliştiriciler ve tasarımcılar zaten çözülmüş problemlerle tekrar tekrar uğraşmazlar. Bunun yerine başarıya ulaşmış iyi tasarımları yeniden kullanarak kendi sistemleri için özelleştirirler [9]. Çerçeve geliştirme sürecinde tasarım kalıplarının kullanımı, istenilen tecrübeye sahip çalışanların bulunması probleminin aşılmasında etkili olacaktır.

Tasarım kalıplarının temelleri, mimar Christopher Alexander'ın 1970’li yılların sonlarında başlattığı çalışmalara dayanmaktadır. Alexander, desenlerin belgelenmesi için temel kabul edilen örnekler ile ilgili 1977'de Desen Dili : Kentler, Binalar, Yapılar [10] ve 1979'da Ebedi Yapım Yöntemi [11] kitaplarını yayınlamıştır. Bu kitaplarda mimari desen örneklerinin yanı sıra, bu desenlerin nasıl belgeleneceği de konu edilmiştir.

1987'deki uluslararası Nesneye Yönelik Programlama, Sistemler, Diller ve Uygulamalar (OOPSLA) konferansına kadar tasarım kalıplarıyla ilgili bir çalışma ortaya çıkmamıştır. Bu tarihten sonra ise Grady Booch, Richard Helm, Erich Gamma

ve Kent Beck başta olmak üzere tasarım kalıpları ile ilgili birçok makale ve sunum yayınlanmıştır.

1987 yılında, Ward Cunningham ve Kent Beck, bir kullanıcı ara yüzünün tasarımı ve SmallTalk üzerine çalışıyordu. Bu amaçla yeni SmallTalk programcılarına yol göstermesi amacıyla beş küçük desen dili geliştirmeye ve Alexander'ın fikirlerinden yararlanmaya karar verdiler. Elde edilen sonuçlar "Using Pattern Languages for Object-Oriented Programs" adlı makalede yazılıp OOPSLA'87 Orlando'da sunuldu [12]. Kısa bir sure sonra Jim Coplien, C++ deyimleri kataloğu hazırlamış ve 1991 yılında Advanced C++ Programming Styles and Idioms adlı bir kitap yazmıştır [13]. 1990 yılından 1992 yılına kadar Gang of For (GoF) ekibinin üyeleri Erich Gamma, Richard Helin, Ralph Johnson, ve John Vlissides çeşitli desen katalogları hazırlamışlardır.

OOPSLA'91 görüşmelerinde çok sayıda tasarım kalıbının tartışması yapılmıştır. Bu tartışmalarda Jim Coplien, Doug Lea, Desmond D'Souza, Norm Kerth, Wolfgang Pree gibi saygın model uzmanları katılmıştır. 1993 yılı Ağustos ayında Colorado şehrinde Kent Beck ve Grady Booch sponsorluğunda Hillside Group ilk toplantısını yapmıştır. Daha sonra desen toplantısı OOPSLA'93 yapılmış ve Nisan 1994'te Hillside Group ilk PloP konferansını planlamak üzere toplanmıştır.

Kısa bir süre sonra 1994'de Erich Gamma, Richard Helm, Ralph Johnson ve John Vlissides tarafından yayınlanan "Tasarım Kalıpları : Tekrar Kullanılabilir Nesneye Yönelik Yazılımın Temelleri" kitabı tasarım kalıplarının yazılımda kullanılmasında dönüm noktası olmuştur [8].

Bir tasarım kalıbının belirli bir bağlam içerisinde tekrar eden somut bir biçim için soyutlama biçimindeki tanımlaması, literatürdeki diğer tanımlamalardan daha geneldir. Tasarım kalıpları için günümüzdeki en yaygın tanımlama olan "Bir bağlam içerisinde tekrar eden problem için bir çözüm" tanımlaması tasarım içerisindeki problemlerin çözümüne de uygun bir tanımlamadır [14]. Alexander, ayrıca şu tanımlamayı yapmıştır :

"Her bir tasarım kalıbı belirli bir bağlam, bir problem ve bir çözüm arasında bir ilişkiyi gösteren üç bölümlü bir kuraldır [14]."

Bu tanımdan yola çıkılarak bir desenin belirli bir bağlam içerisinde tekrar eden zorlamalar ve bu zorlamaları çözen bir biçim arasındaki ilişki olduğunu söylemek mümkündür. Daha sonra Alexander özel bir bağlam içerisinde tekrar eden bir problemin çözümü olan, sadece mimari için değil, yazılım tasarımı için de bir desen tanımlamıştır [14]. Somut desenler, sınıfların ve nesnelerin kullanımını gösterir, bununla birlikte belirli bağlamlar içerisinde tekrar eden problemlere çözümler sunar. Bu tanım, günümüzde desenlerin en yaygın biçimidir ve Beck ve Johnson [15], Schmidt [16], Buschmann [17] ve diğer birçok araştırmacı tarafından kabul edilmiştir.

1.1. Tez İçeriği ve Tezin Organizasyonu

Bu çalışmada Bologna Süreci'ne uygun olarak geliştirilen Eğitim Öğretim Bilgi Sistemi (EOBS) çerçeve yazılımı ile üniversitelerin eğitim-öğretim süreçlerinin tanımlı, şeffaf ve sürekli geliştirilebilir bir çerçeveye taşınması hedeflenmektedir. Bologna Süreci eylem başlıkları arasında ifade edilen Avrupa Kredi Transfer Sisteminin (European Credit Transfer System, ECTS) uygulanması, ulusal ve uluslararası paydaşlarca tanınan, kolay anlaşılır ve birbirleriyle karşılaştırılabilir ve ilişkilendirilebilen derecelerin verilebileceği bir yeterlilik derecelendirme sistemi çerçevesinde yükseköğretim diploma veya derecelerinin oluşturulması ve sürece dahil olan ülkelerin ulusal yeterlilikler çerçevelerini bir web bilgi sistemi dahilinde oluşturmalarını sağlayabilecek, aynı zamanda üniversitelerin diğer bilgi sistemleri ile entegre çalışan ve eğitim-öğretim süreçlerini geliştirmeye yönelik web üzerinden servisler veren özel bir çerçeve yazılımının, tasarım kalıplarından faydalanarak, esnek ve bakımı yapılabilir şekilde geliştirilmesi amaçlanmıştır.

Uygulama geliştirme aracı olarak Microsoft Visual Studio 2008 ortamında ASP.NET, veritabanı yönetim sistemi olarak da Microsoft SQL Server 2008 kullanılmıştır.

Çalışmanın birinci bölümündeki girişten sonra, ikinci bölümünde Bologna Süreci ve Bologna Süreci kapsamında gelişitirilen eğitim öğretim bilgi sistemleri açıklanmıştır. Üçüncü bölümde tasarım prensipleri ve tasarım kalıpları konusunda incelemelerin sonuçları bulunmaktadır. Dördüncü bölümde tasarım prensipleri ve tasarım kalıpları temelinde geliştirilen EOBS çerçeve yazılımı incelenmiştir. Beşinci bölümde sonuçlar ve öneriler yer almaktadır.

BÖLÜM 2. BOLOGNA SÜRECİ VE EĞİTİM ÖĞRETİM BİLGİ SİSTEMLERİ

2.1. Bologna Süreci

Eğitim alanında, özellikle yükseköğretim kurumlarında uygulanan çeşitli eğitim-öğretim sistemleri ve standartları, farklı okullardan mezun olan öğrencilerin farklı bilgi, beceri ve yeterlilikte olmasına sebep olmaktadır. Bu da yüksek öğretim kurumları arasında tanınma, öğrenci hareketliliği ve bilgi paylaşımı konularında sorunlara yol açmaktadır.

Bilgi Avrupa'sı; vatandaşlarına yeni bin yılın getirdiği zorluklarla başa çıkabilecek yeterlilik ile aynı sosyal ve kültürel alana mensup olma ve ortak değerler bilinci verebilen bir Avrupa vatandaşlığı nosyonunu zenginleştirmek ve güçlendirmek için vazgeçilmez bir unsur ve toplum ve insan gelişimi için yeri doldurulamaz bir faktör olarak kabul edilmektedir. Demokratik, barışçı ve istikrarlı toplumların gelişmesi ve güçlendirilmesinde, özellikle de Güneydoğu Avrupa ülkelerinin durumu düşünüldüğünde, eğitim ve eğitimde işbirliğinin, uluslararası düzeyde bir öneme sahip olduğu kabul edilmektedir [18].

Bu düşünceler ışığında imzalanan 25 Mayıs 1998 tarihli Sorbon Deklarasyonu, Avrupa kültürel boyutlarını oluşturmada üniversitelerin rolünü vurgulamaktadır. Bunun yanı sıra söz konusu deklarasyonda; vatandaşların hareketliliğini ve istihdamını teşvik etmek ve kıtanın genel gelişimini desteklemek için bir Avrupa Yükseköğretim Alanı oluşturulmasının öneminin altı çizilmiştir. 1999 yılında 29 Avrupa ülkesinin yükseköğretimden sorumlu Bakanları tarafından Bologna Bildirisi'nin imzalanması ile başlayan süreç, bugün ülkemizin de dahil olduğu 47 üyeye ulaşarak, her ülkenin özgür iradeleri ile katıldıkları bir oluşum halini almıştır.
Bologna Süreci, Avrupa Birliği ülkeleri yükseköğretim kurumlarını yeterlikler çerçevesinde değerlendirmeyi ve Avrupa genelinde ortak bir kalite anlayışı

oluşturmayı hedeflemektedir. Bologna süreci eylem başlıkları arasında ifade edilen Avrupa Kredi Transfer Sisteminin (European Credit Transfer System, ECTS) uygulanması, kolay anlaşılır ve birbirleriyle karşılaştırılabilir yükseköğretim diploma veya derecelerinin oluşturulması ve sürece dahil olan ülkelerin ulusal yeterlilikler çerçevelerini tanımlama süreçleri ve bu süreçleri oluşturmada yapılması gereken çalışmalar, alt çalışma grupları oluşturularak belirlenmeye devam etmektedir. Ülkelerin, ulusal yeterlilikler çerçevesi (UYÇ) çalışmalarını, bu aşamaları takip ederek bir takvim doğrultusunda sürdürmeleri ve en geç 2012 yılına kadar tamamlamaları hedeflenmektedir [19].

2.2. Bologna Süreci Eylem Başlıklarının Türkiye Yükseköğretiminde Uygulanması Çalışmaları

Bologna süreci kapsamında Türkiye'de Yüksek Öğretim Kurumu bünyesinde konunun uzmanlarından oluşan bir çalışma grubu oluşturulmuştur. Bu çalışma grubu Bologna başlıklarının Türkiye Yükseköğretimi'nde uygulanması hususunda 4 temel faaliyet alanı belirlemiştir. Bunlar;

- Derece ve Öğrenim Sürelerinin Tanınması
- Yükseköğretim Yeterlilikler Çerçevesi
- Kalite
- Sosyal Boyut

şeklindedir.

Bu faaliyet alanları ile varılmak istenen sürecin temel taşları konumundaki

- Avrupa Kredi Transfer Sisteminin (European Credit Transfer System, ECTS) uygulanması
- Program öğrenme çıktılarının oluşturulması
- Sürece dahil olan ülkelerin ulusal yeterlilikler çerçevelerinin tanımlanması

başlıklarının gerçeklenerek, ayrı bir faaliyet alanı olarak belirlenen kalite güvencesi kapsamında bu çalışmaların sürekli olarak güncellenmesinin yapılmasıdır. Bu

bağlamda öncelikle bu dört eylem başlığının içeriği anlatılacak ve bu içerikler dahilinde söz konusu başlıkların Türkiye yükseköğretiminde uygulanması çalışmaları ele alınacaktır.

2.2.1. AKTS sisteminin uygulanması çalışmaları

Bologna sürecinden önce Avrupa'da yükseköğretimde hareketlilik ve tanınmanın önündeki en önemli engel, sistem ve standartların farklılığı olmuştur. Bologna süreciyle birlikte, öğrencilerin çeşitli yükseköğretim kurumlarından almış oldukları eğitimlerin, Avrupa'daki diğer yükseköğretim kurumları tarafından da tanınması ile ilgili sorunlara çözüm getirmek üzere Avrupa Kredi Transfer Sistemi, AKTS (European Credit Transfer System, ECTS) geliştirilmiştir. AKTS, öğrenci merkezli, öğrencinin iş yüküne dayalı bir kredi sistemidir. AKTS kredisi, öğrencinin bir dersi başarıyla tamamlayabilmesi için yapması gereken çalışmaların tümünü (teorik ders, uygulama, seminer, bireysel çalışma, sınavlar, ödevler vb.) ifade eden bir birimdir. Başlangıçta kredi transferi için düşünülen bu sistem, kurumsal, bölgesel, ulusal ve Avrupa seviyesinde kredi toplama sistemi olarak da kullanılmaktadır [19].

Bologna sürecinin Türkiye'de uygulanması aşamasında, AKTS en önemli çalışma alanlarından birisidir. Son yıllarda, Türkiye'deki birçok üniversite kendi kredi ve notlandırma sistemlerini AKTS prensiplerine uyumlaştırma çalışmalarını yoğunlaştırmış durumdadırlar. Bu kapsamda, üniversitelerde hem AKTS'nin uygulanışından hem de öğrencilerin AKTS hakkında bilgilendirilmesinden sorumlu olmak üzere AKTS kurum koordinatörleri ve her fakülte veya bölüm için ayrı AKTS bölüm koordinatörleri belirlenmiştir. Aynı zamanda, Bologna Ulusal Takımı, Yükseköğretim Kurulu ile birlikte, özellikle yeni kurulmuş olan üniversitelerdeki akademisyenleri AKTS'nin doğru şekilde hesaplanması konusunda bilgilendirmek amacıyla ulusal veya bölgesel düzeyde toplantılar, çalıştaylar düzenleyerek üniversitelerdeki AKTS çalışmalarını hızlandırmaktadırlar.

2.2.2. Ulusal yeterlilikler çerçevesinin tanımlanması çalışmaları

Bologna sürecinde öngörülen yapıda Avrupa Yükseköğretim Alanı Yeterlilikler Çerçevesi’nin amacı,

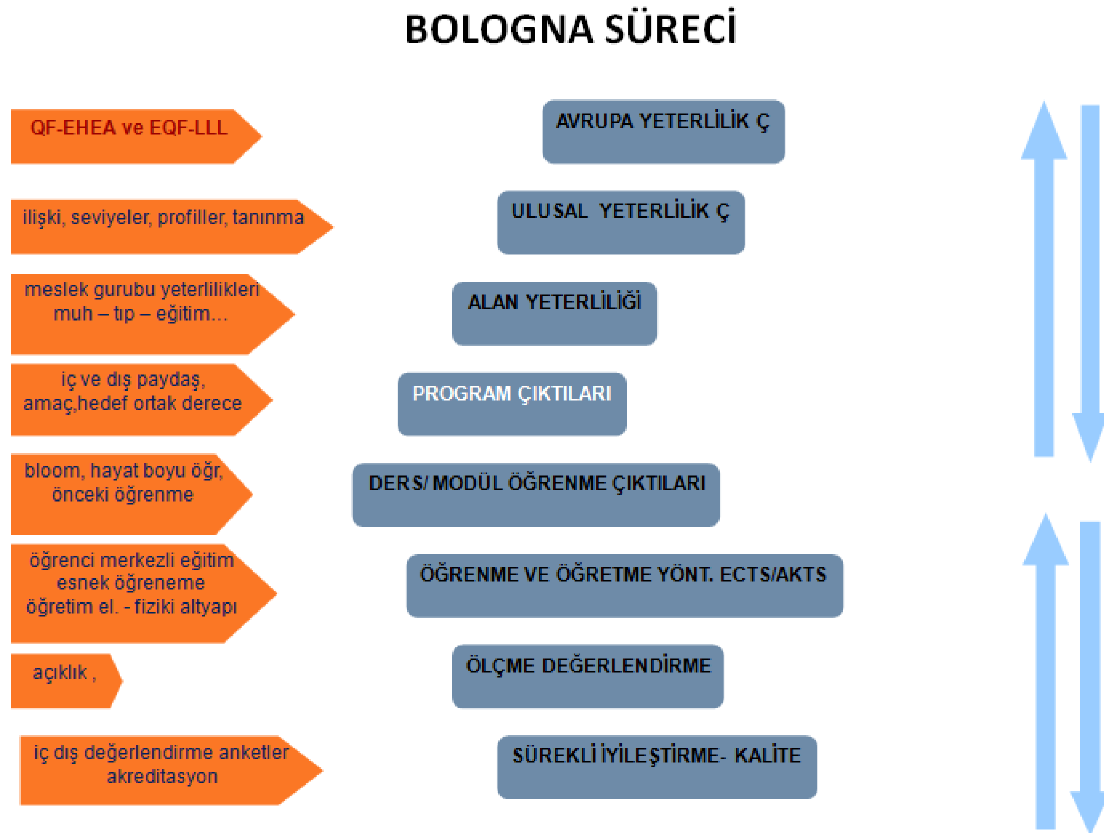
- Yükseköğretim sistemleri arasında uluslararası ilişkilendirmeyi sağlamak (saydamlık),
- Yükseköğretim sistemlerinin birbirini tanımasını kolaylaştırmak (tanınma)
- Öğrenenlerin ve mezunların hareketliliğini artırmak (hareketlilik)

olarak tanımlanmaktadır [18].

Türkiye'de yükseköğretimde Ulusal Yeterlilikler Çerçevesi (UYÇ) oluşturulmasına yönelik ilk çalışmalar, Bologna sürecinde 2005 yılında Bergen'de gerçekleştirilen ve ulusal yeterlilikler çerçevelerinin oluşturulmasını karara bağlayan bakanlar zirvesi sonrasında başlatılmıştır. Yükseköğretim Kurulu bünyesinde kurulan Yükseköğretim Yeterlilikler Komisyonu tarafından yürütülen çalışmalar ile birlikte ağırlıklı olarak Avrupa Yükseköğretim Alanı için Yeterlilikler Çerçevesi (Qualifications Framework for European Higher Education Area - QF-EHEA) düzey tanımlayıcılarını kullanarak UYÇ'yi yükseköğretimin her düzeyi (önlisans, lisans, yüksek lisans ve doktora) sonunda asgari olarak kazanılması gereken bilgi, beceri ve yetkinliklere göre belirlenmiştir. Yapılan yeterlilikler çerçevesi tasarımında, kapsanacak düzeyler (cycles, levels), her bir düzeyde mevcut eğitim-öğretim program profillerinin belirlenmesi (profiles), her bir düzey ve profil için verilen derecelerin (diplomaların) türü (award types), düzeylerin tanımlanması için kullanılacak öğrenme çıktıları ve genel düzey tanımlayıcıları (learning outcomes, output descriptors, QF-EHEA or EQF-LLL level descriptors) ve her bir düzey için gerekli kredi ve iş yükleri (credits and workload) tanımlanmıştır. Bu kapsamda öğrenme çıktıları ile ifade edilen "Türkiye Yükseköğretim Yeterlilikler Çerçevesi"nin ilk taslak çalışmasını ilgili paydaşların görüşlerine ve katkılarına sunmuştur [19].

2.2.3. Ulusal yeterlilikler çerçevesi kapsamında üniversitelerde yapılan çalışmalar

Son yıllarda Türkiye'deki üniversitelerde Bologna Süreci çerçevesinde yoğun altyapı, bilgilendirme ve uyum çalışmaları yapılmaktadır. Bu çalışmaların en önemli aşamalarından birini, mevcut eğitim öğretim programlarının gelişmelere paralel olarak güncelleştirilmesi oluşturmaktadır.



Şekil 2.1. Bologna Süreci İlişkileri [18]

Şekil 2.1'de görülen Bologna süreci ilişkilerinden "Program Çıktıları" ve altındaki adımlar doğrudan üniversitelerde yürütülmesi gereken çalışmalar olarak karşımıza çıkmaktadır. Bu kapsamda her program için amaç ve hedef belirleme, yeterliklerin ve öğrenme çıktılarının belirlenmesi, ders planlarının oluşturulması ve sürdürülebilir gelişme adımlarının belirlenmesi gibi konularda bütün üniversiteler kendi stratejilerini belirlemekte ve bu yönde çalışmalarını sürdürmektedir.

2.3. Eğitim Öğretim Bilgi Sistemi Yazılım Çerçevesi Tasarımı

Çalışmanın bu bölümünde, Bologna Süreci eylem başlıkları arasında yeralan; Avrupa Kredi Transfer Sisteminin (European Credit Transfer System, ECTS) uygulanması, program öğrenme çıktılarının oluşturulması ve yeterlilik çerçevelerinin tanımlanması gibi süreçlerin, bir web tabanlı eğitim öğretim bilgi sistemi (EOBS) üzerinden oluşturulmalarını sağlayabilecek şekilde yazılım çerçevesinde olması gereken özellikler çıkarılarak tasarımın hangi temel üzerine oturtulması gerektiği anlatılacaktır.

2.3.1. Eğitim öğretim bilgi sistemi yazılımlarının sahip olması gereken özellikler

Hazırlanacak EOBS yazılım çerçevesi Şekil-2.1 de gösterilen Bologna Süreçlerinin uygulanması kapsamında program yeterliliklerinin belirlenmesi, ders içeriklerinin hazırlanması, ders planı ve AKTS kredilerinin hesaplanması gibi sisteme özel yapıları barındırmasının yanı sıra, web tabanlı bir bilgi sisteminde olması gereken, doğrulama (authentication), yetkilendirme (authorization), işlem kayıtları tutma (logging), hata yakalama (error handling) ve raporlama (error reporting), kullanıcı dostu arayüz (user-friendly interface) sunulması, sağlıklı ve düzenli bir hesap yönetimi (account administration) gibi özellikleri de gerçekleştirmelidir. Doğrulama, yetkilendirme ve raporlamaların yapıldığı modüller üniversitenin diğer bilgi sistemleri ile entegre çalışabilecek şekilde esnek bir mimaride ve çok dilli olarak tasarlanmalıdır.

Eğitim Öğretim Bilgi Sistemi yazılımları, program ve ders bilgileri olmak üzere iki temel süreç üzerine inşa edilmelidir. Programlar ve dersler birbirleriyle ilişkili nesneler olacaklarından bu iki temel yapı ve aralarındaki ilişkileri temsil eden yapılar bir ağaç yapısı şeklinde ifade edilebilmelidir. Bir diğer deyişle programlar; programların yeterlilikleri, programlarda okutulan dersler, derslerin öğrenme çıktıları ve öğrenme çıktılarının program yeterliliklerine katkılarını gösterecek yapılar hiyerarşik bir düzende menü ağaçları veya başka görsel kontroller kullanılarak kullanıcıya sunulmalıdır.

Program bazlı işlemler, programın amaç ve hedefleri, öğrenme çıktıları, eğitim öğretim metodları, ders planı ve AKTS kredileri, ders-program çıktıları ilişkileri, ders kategori listesi, alınacak derece, kabul koşulları, istihdam olanakları, mezuniyet koşulları, ölçme ve değerlendirme ve öğrencilere uygulanan anketler gibi alt modülleri bünyesinde barındırmalıdır.

Programın Ders Planı modülünden ders bilgilerine erişilebilmeli, bir programa ait derse erişildiğinde, dersin tanımı, akışı, program çıktılarına katkısı ve AKTS iş yükü gibi alt modüller içerisinde dersin detaylı bilgileri görüntülenebilmelidir.

Program yeterlilikleri, programın amaç ve hedefleri, öğrenme çıktıları gibi program bazlı genel işlemler, ve ders içeriği, dersin akışı, ders planı ve AKTS kredileri gibi ders bazlı işlemler genel erişime açık olurken, ders içeriklerinin ve program güncellemelerinin yapılması, kişiye özel sayfaların gösterilmesi, doküman eklenmesi ve ilgili derse ait dokümanlara erişilebilmesi gibi işlemler belirli kullanıcı grup ve yetkileri doğrultusunda yapılmalıdır.

Program ve ders güncelleme çalışmaları, önceden belirlenmiş takvime uygun olarak bir çevrim dahilinde yapılacağından program ve ders bazlı işlemlerle ilgili modüller üzerinde yapılan bütün veri giriş işlemleri, önceden tanımlanan yetkiler çerçevesinde ilgili birim sorumlularınca yapılmalıdır. Yazılım çerçevesi, bu esnekliği sağlayacak şekilde kullanıcı yetkilerini modüler bazlı tanımlamaya imkan tanırken, aynı zamanda belli bir zaman aralığına bağlı olarak da etkinleştirilebilecek şekilde tasarlanmalıdır [2].

BÖLÜM 3. TASARIM PRENSİPLERİ VE TASARIM KALIPLARI

Yazılım tasarımı, tasarım prensipleri ve tasarım kalıpları adı verilen örgülerin kullanılması ile karmaşık programlama yapısını basite indirgeyen, benzer programlama süreçlerinin bir kez kullanılması ile birden çok kez kullanımını ortadan kaldıran, özetleme yapan ve etkinlik sağlayan bir süreçtir.

İyi bir yazılım tasarımının amacı, yazılım sistemine esneklik katmak, sistemin bakılabilirliğini ve geliştirilebilirliğini sağlamaktır. İyi bir tasarımı gerçekleştirmek için uyulması gereken belli başlı tasarım prensipleri vardır. Bu tasarım prensipleri kötü tasarıma neden olabilecek 4 ana unsuru ortadan kaldırmak amacıyla geliştirilmiştir. Robert C. Martin kötü tasarım belirtilerini açıklamıştır [20].

3.1. Kötü Tasarım Belirtileri

Kötü tasarım belirtileri bağımlılık seviyesi yüksek sınıflarda görülür. Sınıf tasarımları eğer birbirlerine sıkı sıkıya bağlı ise, bu tasarım esneklikten, sağlamlıktan ve taşınabilirlikten uzaktır. Bundan dolayı iyi bir tasarımı gerçekleştirebilmek için mümkün olduğu kadar sınıfların birbirlerine olan bağımlılıklarını azaltmak gerekir.

3.1.1. Genişlemeye kapalı olmak

Genişlemeye kapalı olma (Rigidity), yazılım üzerinde küçük bir değişiklik yapılmasının bile zor olması anlamına gelir. Çoğu değişiklik veya eklenti birçok modülde zincirleme değişikliklerin yapılmasını zorunlu kılar. Böyle bir yazılımda yapılacak değişiklikler için gereken süreyi hesaplamak zordur. Bu problemin esas sebebi, modüller arasında sıkı bağımlı bir yapının olmasıdır [20].

3.1.2. Kırılganlık

Kırılganlık (Fragility), genişlemeye kapalı olma problemi ile yakından ilgilidir. Kırılganlık yazılımın, bir yerinde bir değişiklik yapıldığında başka birçok yerinden bozulmasıdır. Genellikle bozulmalar değişiklik yapılan yer ile kavramsal olarak hiç ilgisi olmayan yerlerde oluşur. Kırılganlığın arttığı durumlarda bozulma olasılığı zamanla artar. Bu türlü yazılımların bakımının yapılması olanaksızdır. Yapılan her değişiklik, durumu daha da kötüleştirir. Yapılan değişiklik çözümden daha fazla, yeni sorunlara yol açar [20].

3.1.3. Taşınamazlık

Taşınamazlık (Immobility), yazılım sistemlerinin tekrar kullanılamayacak biçimde tasarlanması problemidir. Bir modülün veya bir kod parçasının başka projelerde tekrar kullanılması gerektiğinde ortaya çıkar. İhtiyaç duyulan modül genellikle bir çok başka modüle bağımlı durumdadır. Yazılımın bu kısmının istenmeyen diğer bölümlerden ayrılması çok zor ve risklidir. Bu durumda birbirlerine çok benzeyen modüllerin bile tekrar yazılması gerekir. Bu problemin ortaya çıkmasındaki ana sebep hazırlanmış olan bir modülün aslında birden fazla küçük modüller şeklinde hazırlanması gerekliliğidir [20].

3.1.4. Akışkanlığın az olması

Bir değişiklik isteğinin karşılanması için birden fazla çözüm yöntemi ortaya çıkabilir. Bazı yöntemler yazılım tasarımının, bazıları ise önceden yazılmış olan kaynak kodun değişmesini önerir. Eğer yazılım tasarımı, gelen değişiklik istekleri karşısında bozuluyorsa tasarım hatalıdır. İyi tasarlanmış yazılım ürünlerinde, değişikliklerin orjinal tasarımı değiştirmeden yapılabilmesi daha kolaydır [20].

3.2. Nesneye Yönelik Tasarımın Temel Prensipleri

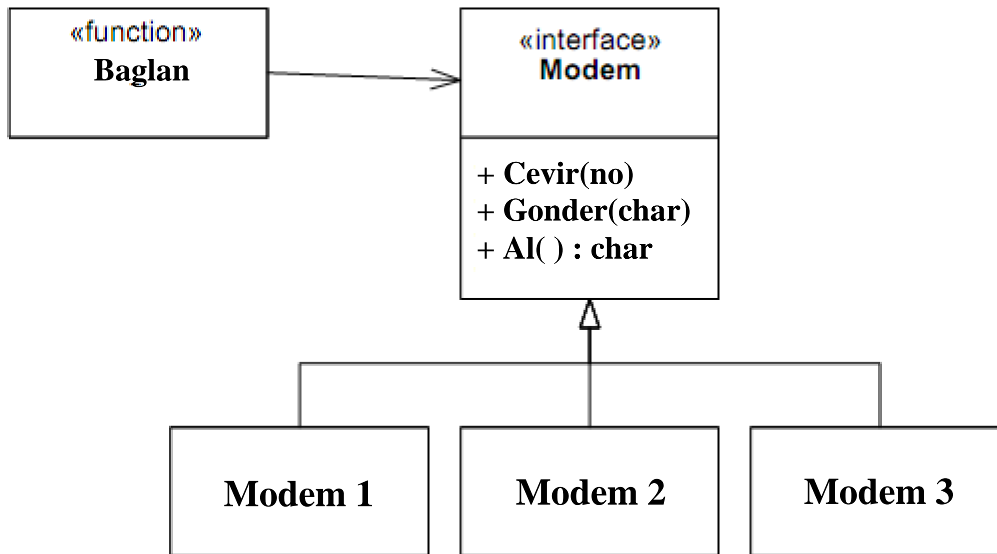
Nesneye dayalı programlama ile geliştirilen yazılımlar oldukça karmaşık mimari tasarımlardan oluşurlar. Tecrübeli yazılımcılar, bu karmaşık yapılardaki tasarımları oluştururken ileride geliştirilebilme ihtimalini her zaman göz önünde bulundurarak

esnek yapılar hazırlarlar. Bu esnek yapıların oluşturulmasında da bazı prensiplere uyulur. Bunlar, nesneye yönelik programlamanın temel prensipleridir [21].

3.2.1. Açık kapalı prensibi

Açık Kapalı prensibinin (Open Closed Principle) temel amacı, geliştirilen yazılım sisteminin genişlemeye açık fakat değişikliğe kapalı olmasını sağlamaktır. Bir diğer ifadeyle geliştirilen yazılım sistemine bir eklenti yapılmak istendiğinde önceden yazılmış kodlarda herhangi bir değişiklik yapılmaması gerektiğini vurgular. Sadece sisteme eklenecek yeni modüle ait kodlamalar yapılır ve sistemin diğer modülleri bu yeni modülün eklenmesinden etkilenmezler. Bunu sağlamak için soyut sınıf ve arayüzlerden yararlanılır [22].

Örneğin Şekil 3.1'de gösterilen UML Şemasında Baglan sınıfı sadece Modem arayüzü ile bağlantı kurmaktadır. Bu sayede sistem, Modem arayüzünü kullanan yeni modem sınıflarıyla genişletilmek istendiğinde Baglan sınıfına ait fonksiyonlarda herhangi bir değişiklik yapmaya gerek kalmaz.

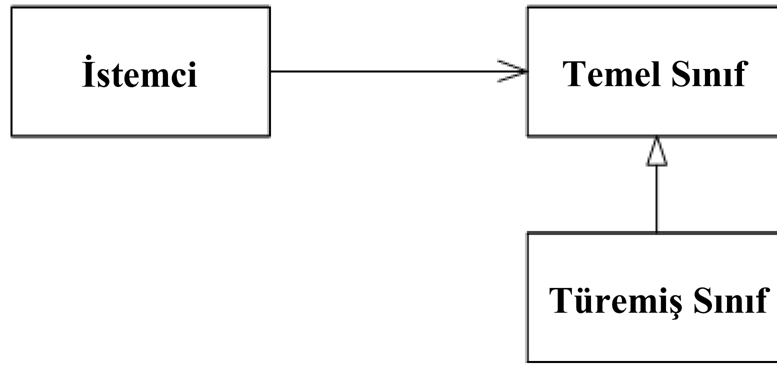


Şekil 3.1. Baglan sınıfının değişikliğe kapalı olduğunu gösteren UML şeması [20]

3.2.2. Liskov ayrışabilme prensibi

Liskov Ayrışabilme Prensibi (Liskov Substitution Principle), türemiş sınıfların, türedikleri temel sınıfların yerini herhangi bir problem çıkmadan alabilmesi prensibidir. Yani temel sınıftan türemiş bir nesneyi kullanan bir fonksiyon, temel sınıf yerine bu temel sınıftan türemiş bir sınıfın nesne örneğini de aynı yerde kullanabilmelidir [23].

Şekil 3.2'de gösterilen UML Şemasında, İstemci sınıfında kullanılan Temel sınıfa ait bir fonksiyon veya özelliğin yerini, Temel sınıftan türemiş, Türemiş sınıfına ait aynı fonksiyon veya özelliğin alabileceği gösterilmektedir.



Şekil 3.2. Liskov Ayrışabilme Prensibi UML şeması [20]

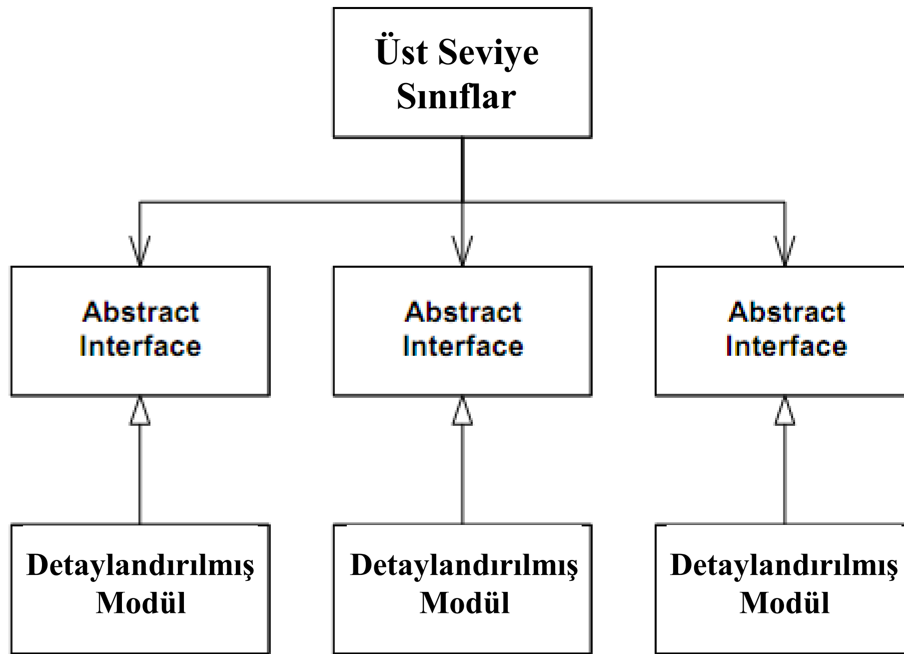
3.2.3. Bağımlılığın azaltılması prensibi

Bağımlılık, herhangi bir sınıfın, başka bir sınıfa ait nesne örneğini, bir fonksiyonunu veya özelliğini kendi içerisinde kullanması anlamına gelir. Bağımlılığın yüksek olduğu sınıflarda bir değişiklik yapılması gerektiğinde bağımlı sınıflar içerisinde de bir dizi değişiklik yapmak gerekebilir. Bu da sistemin modülerliğinin azalmasına sebep olur.

Bağımlılığın azaltılması (Dependency Inversion Principle) prensibine göre; bir yazılım sistemi tasarımında, sınıflar arasındaki ilişkilerin gerçek sınıflardan türemiş nesneler ile değil, ilgili arayüzler veya soyut sınıflar kullanılarak gerçeklenmesi gerekir. Prensip, yüksek seviye sınıfların, düşük seviye sınıflara direkt olarak bağımlı

olmaması gerektiğini belirtir. Bunu sağlamak için de kullanıcı ile sınıflar arasındaki ilişkinin olabildiğince soyutlanmış sınıflar ve arayüzler aracılığıyla yapılmasını önerir [24].

Şekil 3.3'te üst seviyedeki modüller onları uygulayan detaylarla pek ilgilenmezler. Detayları içeren modüller başka modüllerin kendilerine bağımlı olması yerine kendileri soyut sınıflara bağımlıdır. Bu sayede somut sınıflara bağımlılık azaltılmıştır.



Şekil 3.3. Bağımlılığın Azaltılması Prensibi UML şeması [20]

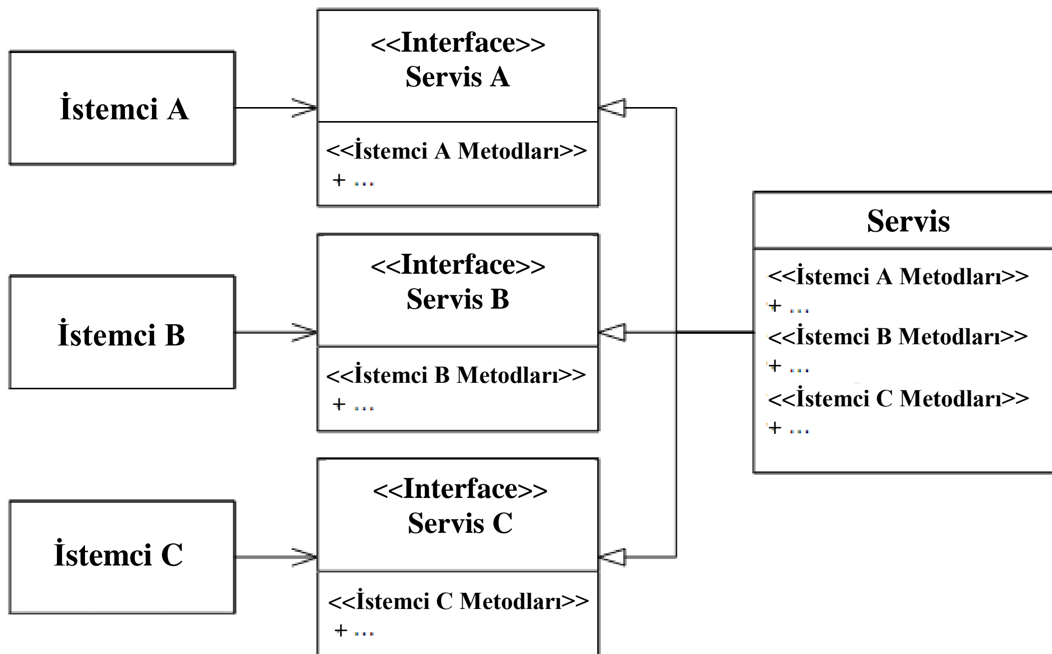
3.2.4. Tek sorumluluk prensibi

Tek sorumluluk (Single Responsibility Principle) prensibine göre bir sınıf sadece tek bir sorumluluğu yerine getirmelidir. Birbiriyle ilişkisi olmayan bir çok görev ve özelliğin bir sınıfa ait metodlar içerisinde gerçeklenmemesi, görev ve sorumlulukların ayrı sınıflara verilmesi anlamına gelir. Her sınıfın sorumluluğu farklı olduğu zaman, ihtiyaçların değişmesi durumunda sadece ilgili sınıf içerisinde değişiklik yapmak yeterli olacaktır. Sınıfların birden fazla sorumluluğunun olması bağımlılığın artmasına neden olur [25]

3.2.5. Arayüz ayırma prensibi

Arayüz Ayırma Prensibi (Interface Segregation Principle), kullanılacak metodların ve özelliklerin gruplandırılarak her biri ayrı işlevi tanımlayan farklı arayüzlere bölünmesi prensibidir. Sonuç olarak, bir sınıfın kullanmadığı metod ve özellikler sınıfın içeriğine alınmamış olur [25].

Şekil 3.4'te gösterilen UML Şemasına göre her istemcinin gereksinim duyduğu metotlar kendisine ait arayüzde yazılır. Bu arayüzler Servis sınıfından kalıtım alır ve metotların uygulaması kendi içerisinde yapılır. Eğer istemci A'nın arabirimi değiştirilirse istemci B ve istemci C bu değişiklikten etkilenmez.



Şekil 3.4. Arayüz Ayırma Prensibi UML şeması [20]

3.3. Tasarım Kalıplarına Genel Bakış

Yazılım tasarımı sırasında kullanılan ve yukarıda anlatılan temel tasarım prensiplerini destekleyen daha özel yapılar da geliştirilmiştir. Bu yapılara Tasarım Kalıpları ismi verilmiştir.

Tasarım kalıpları genel olarak karşımıza tekrar tekrar çıkan problemlerin temeline bir çözüm sağlayan yöntemlerdir. Tasarım kalıplarının temel felsefesi, tüm tasarım işlemlerinde kullanılan tekrar kullanılabilirlik mantığına dayanır [21].

3.4. Tasarım Kalıbı Nedir

Bir yazılım mühendisliği kavramı olarak tasarım kalıbı, yazılım tasarımında ortaya çıkan benzer problemlerin genel çözümleridirler. Tasarım kalıpları genel bir tasarım şablonudur. Tamamen bitmiş ve doğrudan kodlanabilen tasarımlar değillerdir. Ortaya koymuş oldukları genel çözüm yolları uygulamaya göre şekillendirilerek ve değiştirilerek kullanılırlar [26].

Tasarım kalıpları nesneler ya da sınıflar arasındaki ilişkileri ve sorumlulukları belirtirler. Uygulamaya yönelik gösterimleri ve ilişkileri barındırmazlar. Problemin genel çözüm yoluna ilişkin tasarımsal gösterimleri içerirler. Algoritmik ifadeler de tasarım kalıpları içerisinde yer almazlar. Çünkü algoritmalar, işlemsel problemleri çözerler, tasarım problemlerinin bir parçası değillerdir; gerçekleştirim tabanlıdırlar [8].

Tasarım kalıpları genel tasarım çözümlerinin probleme özel olmayacak şekilde dökümante edilmiş halleridir. Bu dökümanlar geliştiricilerin yazılım ilişkilerini iyi tanımlanmış ve anlaşılmış yöntemlerle kurmalarını sağlar. Ayrıca tasarım kalıpları kodun okunabilirliğini ve düzenliliğini sağlarlar. Tasarımcılar için ortak bir dil oluşturmaktadırlar [26].

Tasarım kalıpları, kullanılmış ve başarıya ulaşmış tasarım ve mimarilerin yeniden kullanılabilmesini sağlarlar. Bu sayede üretilen sistemin yeniden kullanılabilirliği artmaktadır. Zaten kullanılmış ve başarıya ulaşmış tasarımları kullanmak, uygulamaya özel problemlerin çözümü için iyi bir başlangıç yapmayı sağlar ve düşülebilecek potansiyel hataları engeller. Geliştiriciler ve tasarımcılar zaten çözülmüş problemlerle tekrar tekrar uğraşmazlar. Bunun yerine başarıya ulaşmış iyi tasarımları yeniden kullanarak kendi sistemleri için özelleştirirler [27].

Tasarım kalıpları, yeni gereksinimleri önceden sezerek sistemi genişleyebilir olarak tasarlamak için kullanılırlar. Sistem değişimlere ve genişlemeye uygun tasarlanmadıysa, ileride ortaya çıkan gereksinimler, sistemin belki de baştan yazılmasına ve test edilmesine neden olacaktır. Tasarım kalıpları, sistemin daha rahat, hızlı ve sağlam bir değişebilirlik özelliğine sahip olmasını sağlarlar [8].

Tasarım kalıpları, probleme ve probleme özgü tasarıma, yüksek seviye bir bakış açısı ile yaklaşılmasını sağlarlar. Bu sayede probleme özgü detaylarla, zamanı geldiğinde ilgilenilir. Tasarımın başlangıcında detaylarla zaman kaybedilmesi engellenmiş olur. Bu sayede sistem daha sağlıklı bir biçimde gelişir, detayların sistemde oluşturabileceği bağımlılıklar mümkün olduğunca azaltılır [8].

Tasarım kalıplarının bir yazılım sistemine katabileceği özellikler aşağıdaki gibi özetlenebilir [23] :

- Benzer problemlerde var olan ve kalitesi kanıtlanmış çözümleri ve tasarımları kullanmak
- Yazılım geliştirme takımları arasında ortak bir terminoloji ve dil kurmak
- Problem çözümleri için daha yüksek seviyeden bir bakış açısı ile yaklaşmayı sağlamak
- Sadece doğru sonuçlar üreten bir sisteme değil, aynı zamanda doğru ve sağlam bir tasarıma da sahip olmak
- Kodun değişebilirliğini ve yeniden kullanılabilirliğini sağlamak

3.5. Tasarım Kalıbı Dokümanı

Tasarım kalıbı dökümanı, tasarım kalıbı hakkında yeterli derecede bilgi veren, hangi kapsamda kullanıldığını ortaya koyan ve önerilen çözüm yolunun sunulduğu dökümandır. Tasarım kalıbı dokümanı aşağıdaki bölümleri içermektedir [8] :

- Kalıp İsmi ve Sınıflandırması: Her kalıbın kendini en iyi şekilde ifade eden ve bir kelimeyle, yaptığı işi anlatan bir ismi vardır. Ayrıca kalıbın, hangi sınıfa girdiği de burada belirtilmektedir.

- Amaç: Kalıbın amacını belirtir, çözdüğü problemi ortaya koyar.
- Kalıbın Diğer İsimleri: Bir tasarım kalıbı birden fazla isme sahip olabilir. Kalıbın diğer isimleri bu bölümde belirtilir.
- Motivasyon: Bu bölüm problemi içeren bir senaryo ve bu senaryo dahilinde problemin nasıl çözüldüğünü gösterir.
- Uygulanabilirlik: Bu bölüm kalıbın hangi durumlarda kullanılabileceğini belirtir.
- Yapı: Kalıbın UML gösterimi burada belirtilir. Sınıf diyagramları ve sınıflar arasındaki ilişkiler belirtilir.
- Katılımcılar: Kalıpta kullanılan sınıf ve nesnelere ve bunların tasarımdaki rollerine burada yer verilir.
- İşbirlikleri: Kalıpta kullanılan sınıflar ve nesnelerin birbirleri ile olan ilişkilerine burada yer verilir.
- Sonuçlar: Kalıbın kullanımının sonuçlarını, avantaj ve dezavantajlarını belirtir.
- Gerçekleştirim: Bu bölüm, kalıbın gerçekleştiriminde kullanılacak teknikleri ve yöntemleri ortaya koymaktadır.
- Örnek kod: Nesneye dayalı bir programlama dilinde kalıbın uygulanışı bir örnek üzerinde gösterilir.
- Bilinen Kullanımlar: Bu bölüm, kalıbın var olan gerçek sistemlerde kullanımına örnekler verir.
- İlişkili Kalıplar: Bu kalıpla ilişkisi ve benzerliği olan diğer kalıplar belirtilir.

3.6. Tasarım Kalıplarının Sınıflandırılması

Tasarım kalıpları çözmüş oldukları tasarım problemine göre üç sınıfta incelenebilirler [8] :

Tablo 3.1. Tasarım Kalıplarının Sınıflandırılması

Oluşturucu	Yapısal	Davranışsal
– Soyut Fabrika (AbstractFactory) – Yapıcı (Builder) – Fabrika Metodu (Factory Method) – Prototip (Prototype) – Tekil (Singleton)	– Adaptör (Adapter) – Köprü (Bridge) – Bileşik (Composite) – Dekoratör (Decorator) – Önyüz (Facade) – Sineksiklet (Flyweight) – Vekil (Proxy)	– Sorumluluklar Zinciri (Chain of Responsibility) – Komut (Command) – Yorumlayıcı (Interpreter) – Tekrarlayıcı (Iterator) – Arabulucu (Mediator) – Hatırlayıcı (Memento) – Gözlemci (Observer) – Durum (State) – Strateji (Strategy) – Kalıp Yordamı (Template Method) – Ziyaretçi (Visitor)

- Oluşturucu Tasarım Kalıpları : Sistemdeki nesnelerin somut sınıfları belirterek nasıl oluşturulması gerektiğini tanımlarlar.
- Yapısal Tasarım Kalıpları: Nesneler ve sınıflar arasındaki kalıtım ve içerme ilişkilerini belirtirler
- Davranışsal Tasarım Kalıpları: Nesnelerin sorumluluklarını ve aralarındaki etkileşimleri belirtirler.

3.6.1. Oluşturucu tasarım kalıpları

Oluşturucu kalıplar, nesne oluşturma işini ve bu işlemdeki karmaşıklığı istemciden soyutlamak için kullanılmaktadır. Ancak bu tanım sadece kullanım kolaylığını çağrıştırmaktadır. Oysa oluşturucu kalıplar, istemcinin uygulama kodunu yazarken somut nesneler kullanmak yerine arayüzler ve soyut sınıflar kullanarak daha esnek yapılarla çalışabilmesini de sağlamaktadır.

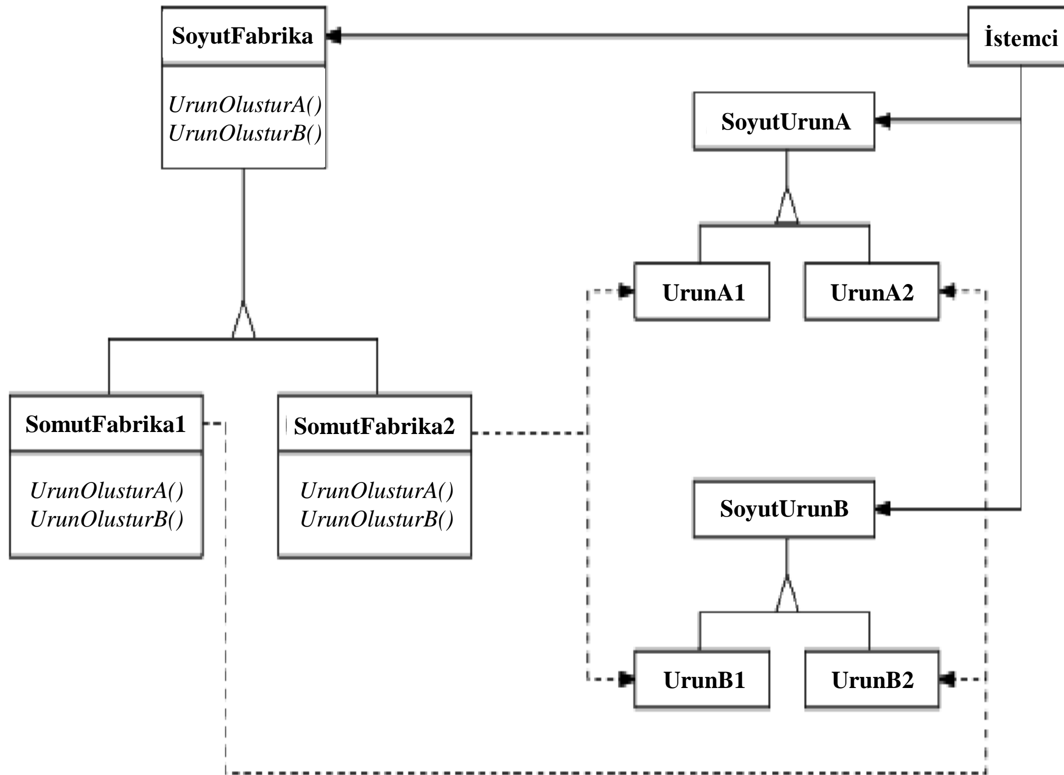
Bu sınıftaki tasarım kalıpları aşağıda listelenmektedir:

- Soyut Fabrika (AbstractFactory)

- Yapıcı (Builder)
- Fabrika Metodu (Factory Method)
- Prototip (Prototype)
- Tekil (Singleton)

3.6.1.1. Soyut fabrika tasarım kalıbı

Soyut Fabrika (Abstract Factory) Tasarım Kalıbı, istemcinin ihtiyacı olan ve aralarında ilişkiler olan nesnelerin üretilmesinden soyut fabrikaların sorumlu olması mantığına dayanır. İstemciler ihtiyaç duydukları ürünlerin tiplerine göre farklı fabrikaları seçip kullanabilirler. İstemcinin ihtiyacı olan nesnler ve bu nesneler arasındaki ilişkiler soyut düzeyde gerçekleştirildiğinden nesne üretimi tamamen istemciden soyutlanmıştır. Soyut Fabrika tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



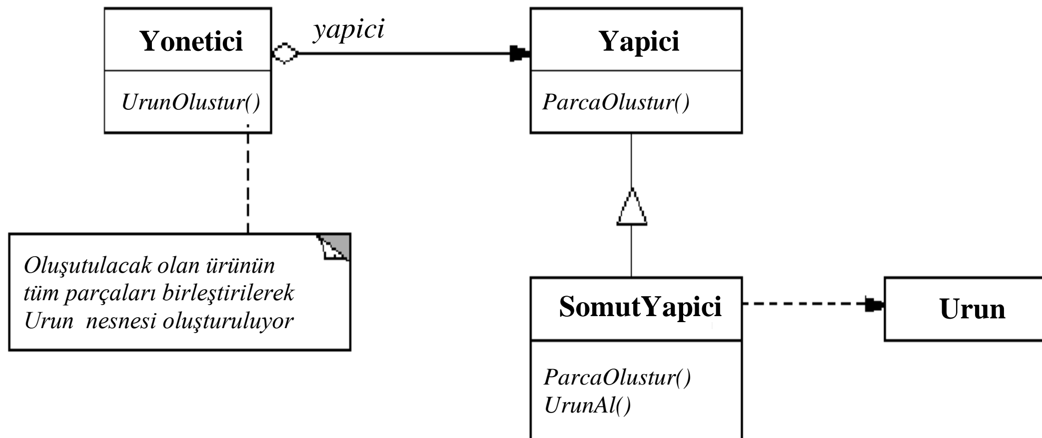
Şekil 3.5. Soyut Fabrika Tasarım Kalıbı UML Şeması [8]

3.6.1.2. Yapıcı tasarım kalıbı

Yapıcı (Builder) tasarım kalıbı, karmaşık yapıdaki nesnelerin oluşturulmasında, istemcinin sadece nesne tipini belirterek üretimi gerçekleştirebilmesini sağlamak için kullanılmaktadır. Bu kalıpta, istemcinin kullanmak istediği gerçek ürünün birden fazla sunumunun olabileceği göz önüne alınır. Bu farklı sunumların üretimi ise Builder adı verilen nesnelerin sorumluluğu altındadır. Dolayısıyla Yapıcı kalıbından yararlanılarak asıl ürünün farklı sunumlarının elde edilebilmesi için gerekli olan karmaşık üretim süreçleri, istemciden tamamen soyutlanabilir.

Yapıcı kalıbının önemli olan özelliklerinden birisi de Soyut Fabrika tasarım kalıbı ile çok benzer yapıda olmasıdır. Ancak arada bazı farklılıklar da vardır. Herşeyden önce Soyut Fabrika kalıbına göre, fabrikanın metodları kendi nesnelerinin üretiminden doğrudan sorumludur.

Yapıcı tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :

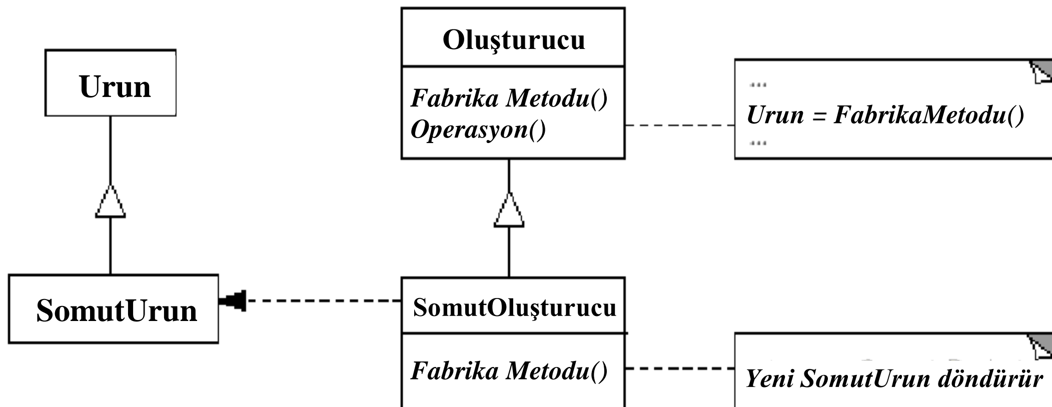


Şekil 3.6. Yapıcı Tasarım Kalıbı UML Şeması [8]

3.6.1.3. Fabrika metodu tasarım kalıbı

Fabrika Metodu (Factory Method) tasarım kalıbının temel amacı, istemcinin ihtiyacı olan bir nesnenin üretiminin sorumluluğunu bir metot yardımıyla yapmaktır. Yani nesnelerin oluşturulması için bir arayüz sağlar. Ancak hangi nesnenin oluşturulacağı kararını bu arayüzü gerçekleştiren altsınıflara bırakır. Nesne oluşturma işini sistemden ayırır ve aldığı parametreye göre nesneleri oluşturarak kullanıcıya geri döner. Bu sayede istemci oluşturulacak nesnenin nasıl oluşturulduğuyla ilgilenmez. Böylece oluşturulacak sınıflara bir bağımlılığı da kalmaz. İstemci sadece fabrika sınıfına hangi nesneyi oluşturmak istediğini bildirir ve bu fabrika sınıfı içerisindeki fabrika metodu, istemcinin ihtiyacı olan nesneyi oluşturarak istemciye geri döndürür.

Fabrika Metodu tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir:



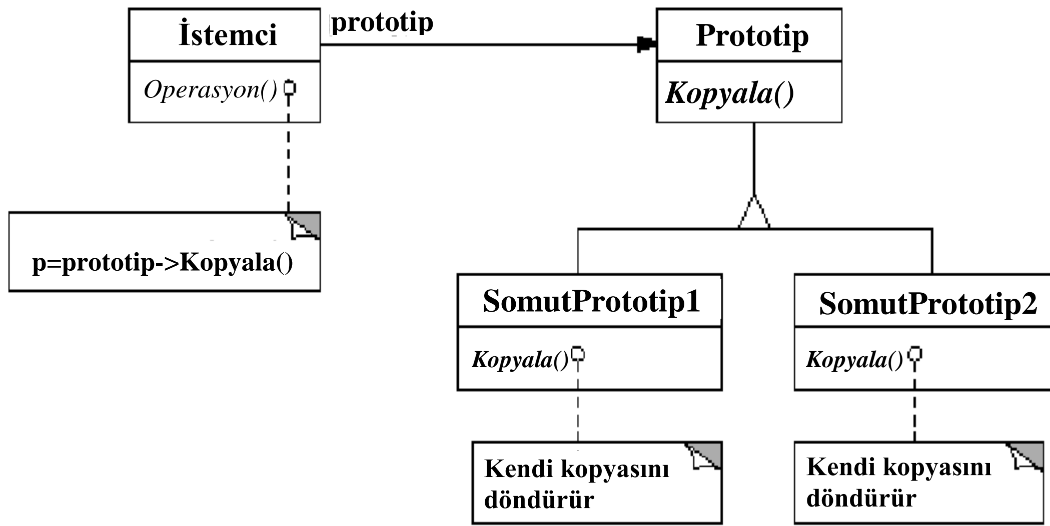
Şekil 3.7. Fabrika Metodu Tasarım Kalıbı UML Şeması [8]

3.6.1.4. Prototip tasarım kalıbı

Üretimi maliyetli olan nesneler söz konusu olduğunda ve “new” operatörü ile oluşan maaliyetten kaçınmak istendiğinde, klon nesnelerin üretilmesi yolu tercih edilebilir. Bu durum zaman içerisinde pek çok nesne yönelimli projede ortaya çıkınca haliyle kalıplaşmış ve bir desen haline gelmiştir.

Söz gelimi bir oyun sahnesinin tekrar eden nesne üretimlerinde, oluşturulma maaliyetlerinin azaltılmasına etki edebilir. Öyle ki oyun sahnesi içerisindeki sabit olan pek çok yapının nesnel olarak ifadesi sırasında bu maliyetler oldukça yükselmektedir. Ya da finansal veriler üzerine analiz gerçekleştiren bir sistemde, aynı veri kümesinden hareket edeceğimiz durumlarda, veriyi içeren nesne üretimlerinin maaliyetleri en aza indirgenebilir. Senaryolar çoğaltılabilir ancak özünde, nesne oluşturulma maaliyetleri vardır.

Prototip tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :

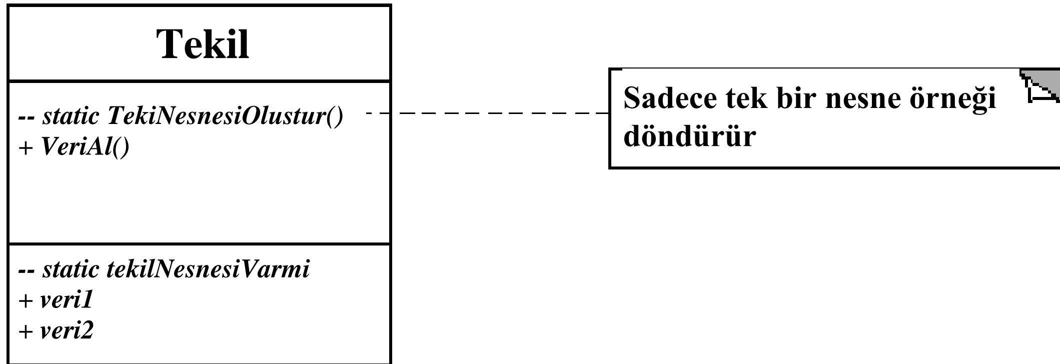


Şekil 3.8. Prototip Tasarım Kalıbı UML Şeması [8]

3.6.1.5. Tekil tasarım kalıbı

Tekil (Singleton) tasarım kalıbının temel amacı, bir sınıfa ait bir nesne örneğinin, uygulamanın çalışma zamanında sadece bir tane olmasının garanti altına alınmasıdır. Diğer sınıflar, bu sınıfa ait bir nesne örneğini kullanmak istediklerinde her zaman bu ilk üretilen nesne referansı geri döndürülecektir. Nesne örneğinin bir tane olmasının garanti altına alınmasından yine aynı sınıf sorumludur.

Tekil tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



Şekil 3.9. Tekil Tasarım Kalıbı UML Şeması [8]

3.6.2. Yapısal tasarım kalıpları

Sınıfların ve nesnelerin daha büyük yapılar oluşturabilmek için nasıl bir araya getirileceği ile ilgili problemlerin çözümleri Yapısal Kalıplar başlığı altında incelenmektedir. Nesne veya sınıfların birleştirilmesi ile belirli bir işi yapan sınıfın işlevi çalışma zamanında değiştirilebilir, işleve yeni özellikler kazandırılabilir.

Bu sınıftaki tasarım kalıpları aşağıda listelenmektedir:

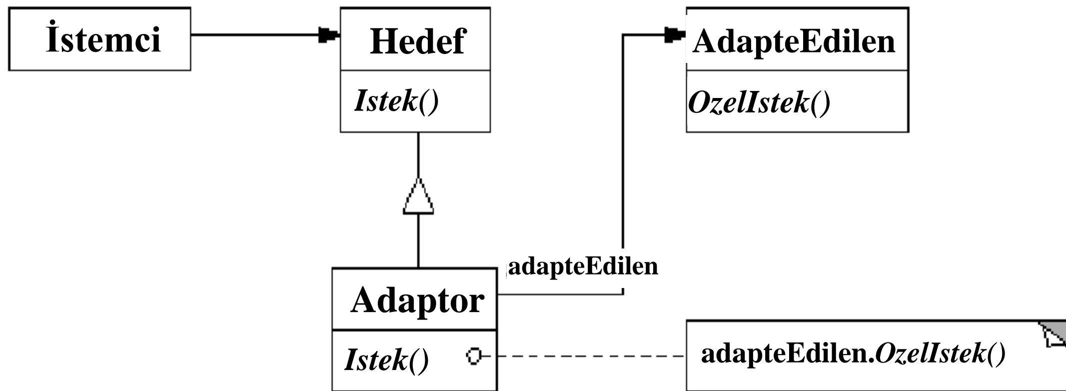
- Adaptör (Adapter)
- Köprü (Bridge)
- Bileşik (Composite)
- Dekoratör (Decorator)
- Önyüz (Facade)
- Sineksiklet (Flyweight)
- Vekil (Proxy)

3.6.2.1. Adaptör tasarım kalıbı

Yabancı bir sistem parçasının, varolan sisteme adapte edilebilmesini ve o sistem içerinde kullanılabilmesini sağlayan bir tasarım kalıbıdır. Yabancı sistemin kendisine has olan değişken, özellik ve metot gibi yapılarının kendi sistemimize uygun hale

getirilerek kullanılmasına olanak tanır. Adaptör tasarım kalıbı ile faklı arayüze sahip olduğu için birlikte çalışamayacak gibi görünen sınıfların birlikte çalışması sağlanırken, sınıfların var olan tanımları üzerinde bir değişiklik gerçekleşmez.

Adaptör tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



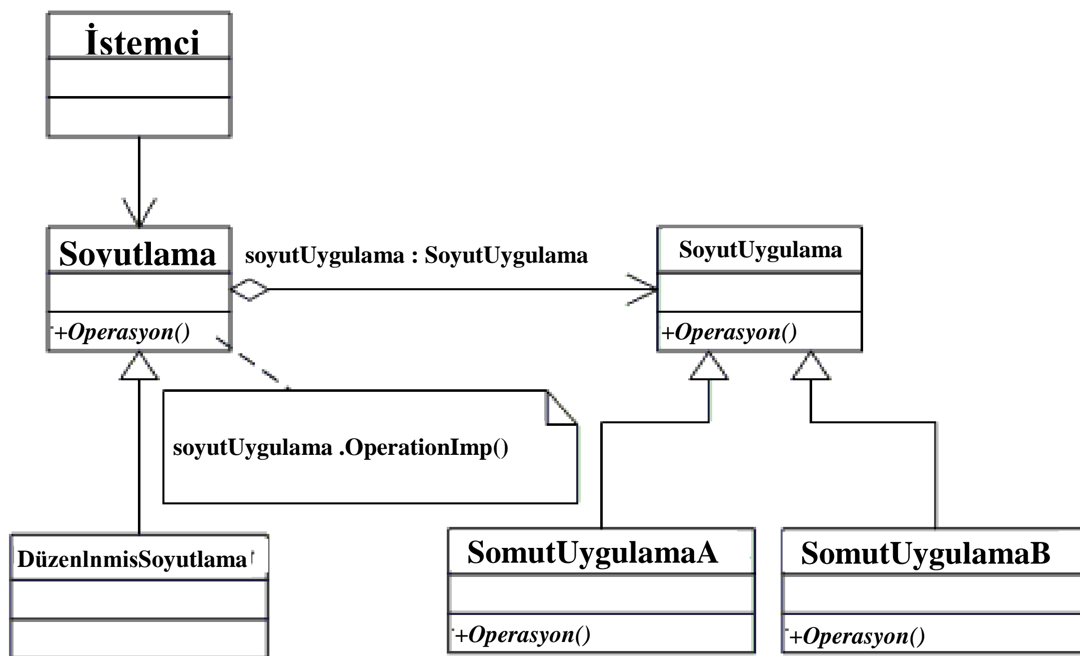
Şekil 3.10. Adaptör Tasarım Kalıbı UML Şeması [8]

3.6.2.2. Köprü tasarım kalıbı

Köprü (Bridge) tasarım kalıbı, soyutlama ile gerçekleştirmeyi ayrı sınıf hiyerarşisi içinde ayırmak için kullanılır. Sınıflara daha fazla bir soyutlama ve genişleme imkanı tanır. Desende hem soyutlama kısmının, hem de gerçekleştirme kısmının bir üst sınıfı bulunur. Bu üst sınıfların altındaysa belirli bir sınıf hiyerarşisi bulunur. Bu iki hiyerarşi de birbirlerine bağlıdır. Köprü tasarım kalıbı, iki kısım arasında köprü gibi bir yapı olarak duran bu bağdan ismini almıştır. Soyutlama kısmında, sistemin daha üst düzey işlemleri bulunur. Gerçekleştirme kısmında ise, bu soyutlama kısmındaki üst düzey işlemlere bağlı daha basit ve bu üst düzey işlemleri detaylandıran işlemler bulunur. Sistemin gerçekleştirme ve soyutlama kısımlarının birbirlerinden bağımsız olarak alt sınıflarla genişlemesine imkan vererek gerçekleştirme kısmını istemciden tamamen soyutlar [28].

Köprü tasarım kalıbı, bir modelleme yapılırken oluşan soyut oluşumlar ve bu oluşumlara ait uygulamaları birbirinden ayırır. Bu sayede yazılımcı sınıf hiyerarşilerini daha esnek bir hale getirebilir. Sınıf hiyerarşilerinin daha esnek bir hale getirilmesi modellemede bulunan bir üst sınıfın içinde barındırdığı soyut oluşumların bir arayüz sınıfına taşınmasına olanak sağlar. Böylece yazılımcı, alt sınıfların herhangi bir uygulamayı kullanabilmesini sağlar.

Köprü tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



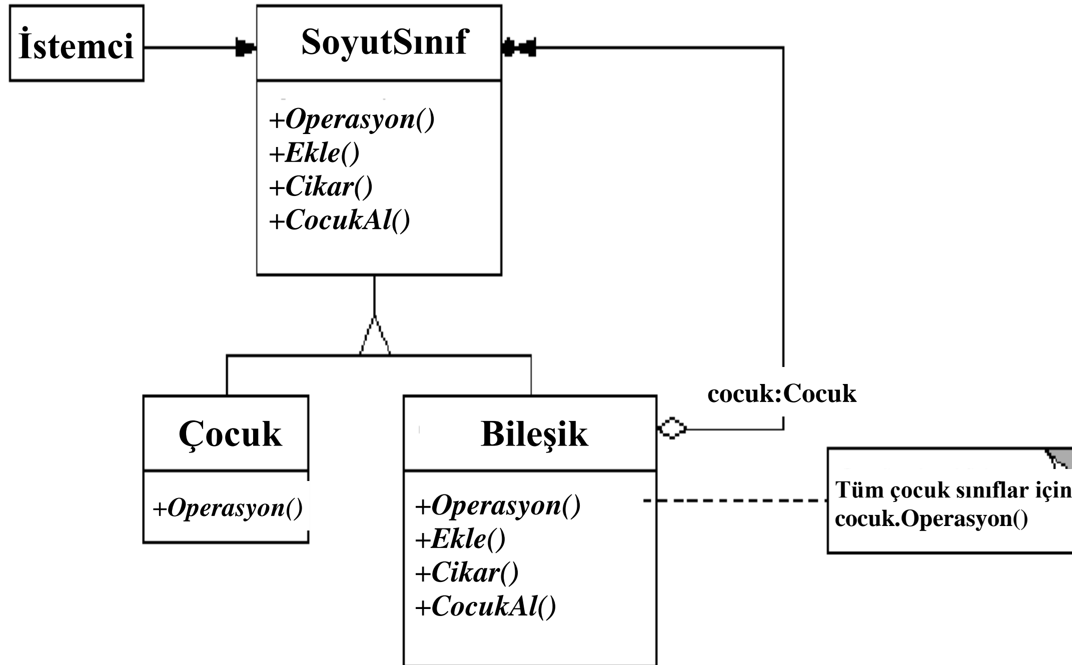
Şekil 3.11. Köprü Tasarım Kalıbı UML Şeması [8]

3.6.2.3. Bileşik tasarım kalıbı

Bileşik (Composite) tasarım kalıbının amacı, belli bir ağaç hiyerarşik yapısına sahip nesneleri bu hiyerarşiye uygun olarak ele alabilmektir. Hiyerarşik yapı söz konusu olduğundan işlemler yukarıdan aşağıya doğru sırayla uygulanır. Bu kalıbı en iyi açıklayan örnek askeriyedeki hiyerarşik yapıdır. Düşük rütbeli askerler daha üst rütbeli başka bir askerin komutasındadır. Hatta bu üst rütbeli asker de yine daha üst rütbeli başka bir askerin komutasındadır. Bu yapının bütün üyeleri askerlerdir.

Yukarıdan verilen bir emir, sırayla alt rütbelere doğru iletilir. Bir komutanın verdiği emir, onun altındaki bütün askerleri ilgilendirir. Zincir böyle sürer gider. Bu şekilde içiçe yapılar bulunduran nesneler Bileşik tasarım kalıbıyla ifade edilebilir.

Bileşik tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



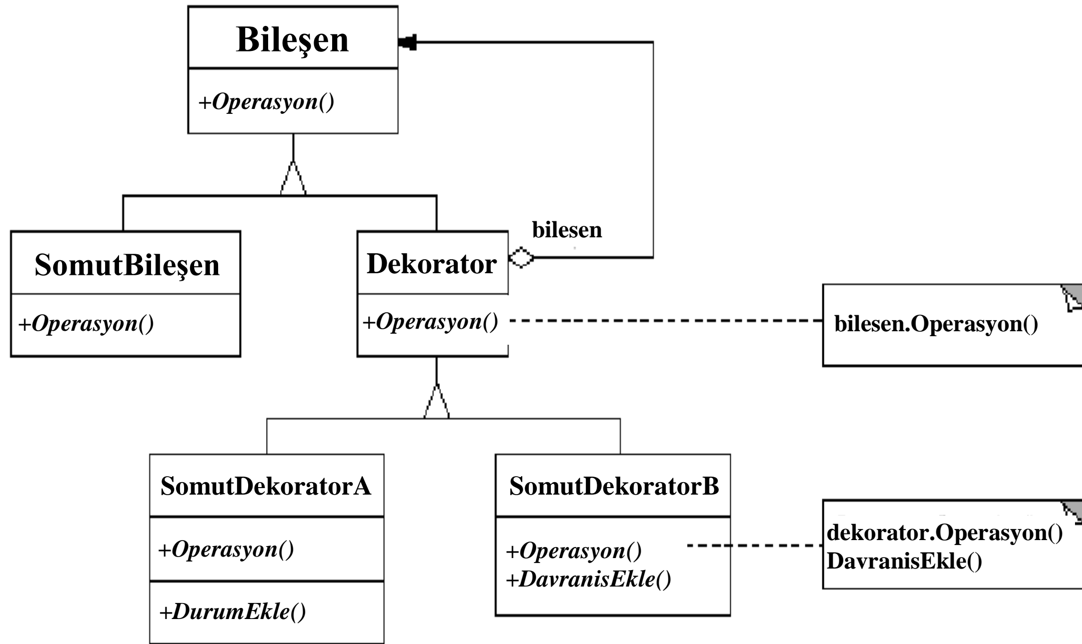
Şekil 3.12. Bileşik Tasarım Kalıbı UML Şeması [8]

3.6.2.4. Dekoratör tasarım kalıbı

Bir nesneye dinamik olarak yeni sorumlulukların eklenmesi ve hatta var olanların çıkartılması amacıyla kullanılır. Bir açıdan bakıldığında nesneyi kendisinden türeyen alt sınıflar ile genişletmek yerine kullanılabilen alternatif bir yaklaşım olarak düşünülebilir.

Dekoratör tasarım kalıbı bize bir nesneye yeni özellikler eklemek için, türetmeyi kullanarak yeni sınıf oluşturmadan nesneye yeni özellikler eklememize olanak sağlar. Bu işlemi türetme yerine kompozisyon kullanarak yapar.

Dekoratör tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



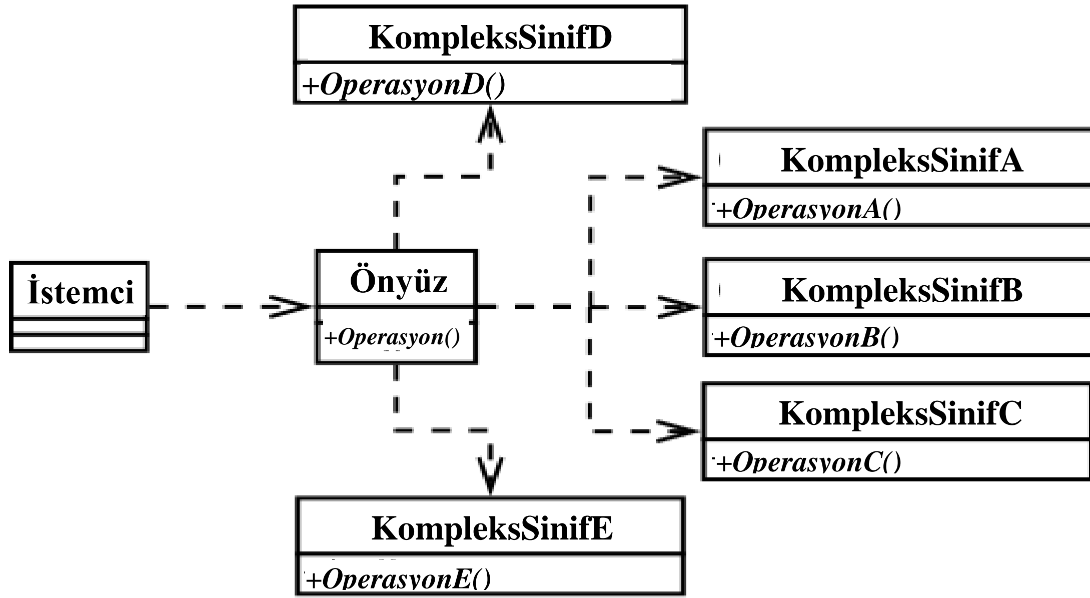
Şekil 3.13. Dekoratör Tasarım Kalıbı UML Şeması [8]

3.6.2.5. Önyüz tasarım kalıbı

Önyüz (Facade) kalıbının temel amacı, bir alt sistemin yapısını oluşturan sınıfları istemciden soyutlayarak kullanımını daha kolay hale getirmektir. Nesne ilişkileri fazla olan ve çok sayıda sınıf barındıran sınıf kütüphanelerinde oldukça kullanışlıdır.

Önyüz kalıbı, var olan karmaşık bir sisteme daha kolay ve kullanışlı yeni bir yüz katmak veya bir amaca yönelik olarak; karmaşık sistemdeki sınıfların üyelerini ve metotlarını kendi içinde barındırarak, kullanıcıdan soyutlamak için kullanılır. Bu sayede kullanıcı karmaşık sistemin karmaşık sınıf ve bu sınıflara ait metotlarından soyutlanarak daha basit bir yöntemle işini gerçekleştirebilir.

Önyüz tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



Şekil 3.14. Önyüz Tasarım Kalıbı UML Şeması [29]

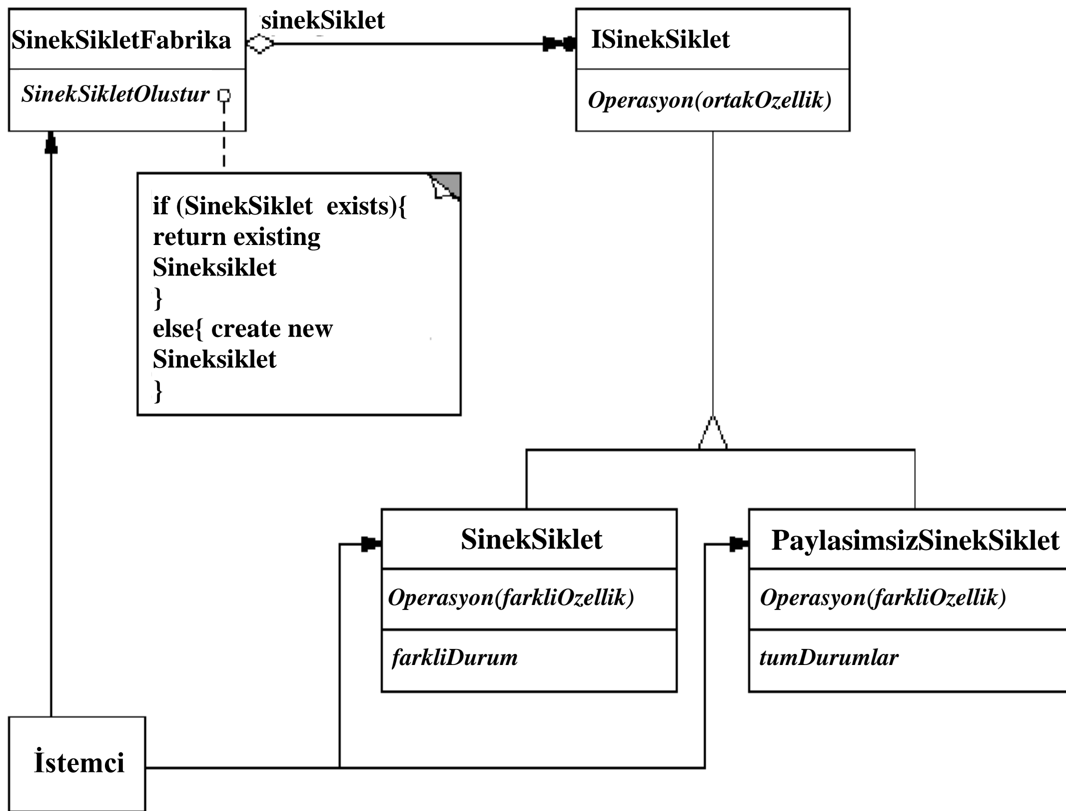
3.6.2.6. Sineksiklet tasarım kalıbı

Sineksiklet (Flyweight)**,** bellek tüketimini optimize etmek amacıyla kullanılan bir tasarım kalıbıdır. Burada önemli olan nokta, bellek tüketiminin çok fazla sayıda nesnenin bir arada ele alınması sırasında ortaya çıkmasıdır. Buna göre söz konusu nesnelerin ortak olan, paylaşılabilen içerikleri ve bunların dışında kendilerine has durumları olduğu takdirde, nesne üretimlerini sürekli tekrar ettirmektense basit bir havuz içerisinden tedarik ettirmek, uygulamanın harcadığı bellek alanlarının optimize edilmesi için yeterli olacaktır. Bu açıdan bakıldığında kalıbın, paylaşımlı nesneleri efektif olarak kullanabilmek üzerine odaklandığını söyleyebiliriz.

Örnek olarak bir oyun sahnesinde yer alan çok sayıda nesne ve bu nesnelerin çok sayıda ve sürekli tekrar eden sayısız örneklerinin uygulama alanında değerlendirildiğini göz önüne alalım. Bu nesnelerin tekrar tekrar kullanılmasının oyun sahnesine getirdiği bellek yükünü hafifletmek amacıyla, söz konusu nesneler, birer Sineksiklet tip haline getirilebilir. Sonrasında ise bu nesnelerin oyun sahnesindeki yüklerini dengeleyecek, bir başka deyişle havuzu oluşturup istemciye

sunacak bir fabrika tipi (FlyWeight Factory) tasarlanabilir. Bu sayede bu nesnelerin tekrar tekrar oluştrurulması yerine ortak özellikleri havuzdan alınarak oluşturulabilir.

Sineksiklet tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :

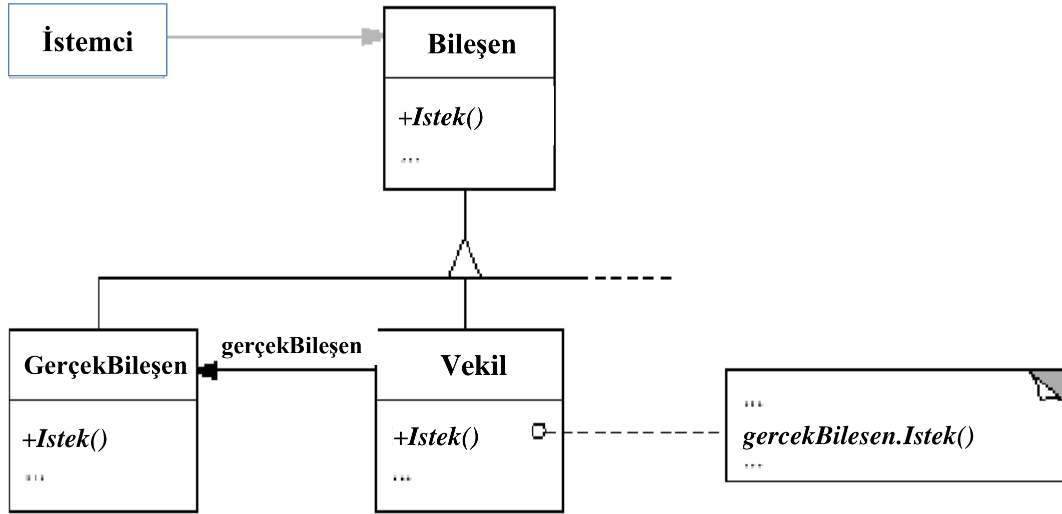


Şekil 3.15. Sineksiklet Tasarım Kalıbı UML Şeması [8]

3.6.2.7. Vekil tasarım kalıbı

Vekil (Proxy) tasarım kalıbının amacı, oluşturulmaları zaman alıcı ve sistem kaynaklarını zorlayan nesnelerin oluşturulmasına aracılık etmektir. Vekil nesneler vekil oldukları nesnelerin tüm metotlarına sahiptirler ve istemci ile vekil olunan nesne arasında aracılık yaparlar. Vekil nesnesi, istemciden gelen tüm istekleri asıl nesneye iletir. Bunun amacı asıl nesneye, gerek olmadığı sürece erişimleri engellemektir. Böylece asıl nesneye dışarıdan erişimler kontrol altına alınmış olur.

Vekil tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



Şekil 3.16. Vekil Tasarım Kalıbı UML Şeması [8]

3.6.3. Davranışsal tasarım kalıpları

Bu sınıftaki kalıplar nesnelere işlevsel sorumluluklarını atayarak nesne grupları arasındaki iletişimin tanımlanmasında kullanılır ve daha karmaşık programlarda akış kontrolünü sağlarlar. Nesneler arasındaki en temel iletişim bir nesnenin başka bir nesnenin metodunu çağırmasıdır. Bu temel yöntem nesneler arasında sıkı bağımlı bir yapının oluşmasına neden olur. Bu bağımlılık sınıf sayısı arttığında bakım maliyetlerini de oldukça arttıracaktır.

Bu sınıftaki tasarım kalıpları aşağıda listelenmektedir:

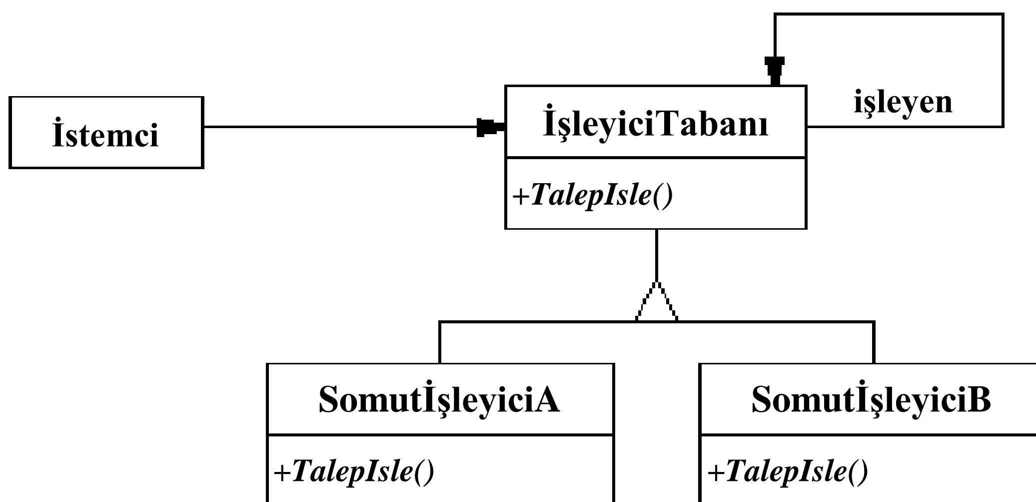
- Sorumluluklar Zinciri (Chain of Responsibility)
- Komut (Command)
- Yorumlayıcı (Interpreter)
- Tekrarlayıcı (Iterator)
- Arabulucu (Mediator)
- Hatırlayıcı (Memento)

- Gözlemci (Observer)
- Durum (State)
- Strateji (Strategy)
- Kalıp Yordamı (Template Method)
- Ziyaretçi (Visitor)

3.6.3.1. Sorumluluklar zinciri tasarım kalıbı

Ortak bir mesaj veya talebin, birbirlerine zayıf bir şekilde bağlanmış nesneler arasında gezdirilmesi ve bu zincir içerisinde asıl sorumlu olan nesne tarafından ele alınması gerektiği durumlarda kullanılmaktadır.

Sorumluluklar Zinciri (Chain of Responsibility) tasarım kalıbına göre, mesajı işleyecek olan asıl nesne örnekleri hayali bir zincir şeklinde dizilmektedir. İstemci, işlenmesini istediği bilgiyi bu zincirin en başında yer alan nesneye gönderir. Zincir içerisinde yer alan nesne örnekleri de söz konusu içeriği asıl işleneceği yere kadar gönderirler. Bir başka deyişle bir akıştan söz etmemiz mümkündür. Zincire atılan her mesaj, zincire dahil olan tüm nesneler tarafından ele alınabilir veya bir sonrakine gönderilebilir. Sorumluluklar Zinciri tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :

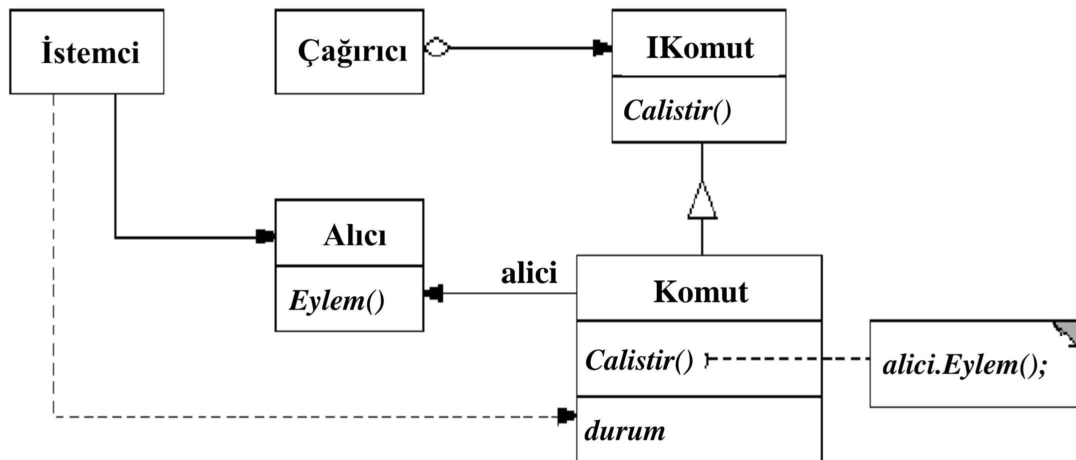


Şekil 3.17. Sorumluluklar Zinciri Tasarım Kalıbı UML Şeması [8]

3.6.3.2. Komut tasarım kalıbı

Komut (Command) tasarım kalıbında, nesneler bir işlevi ve bu işlev için gerekli değişiklikleri içerirler. Dışarıdan bu nesneler tetiklenerek bazı işlemler gerçekleştirilir. Bu tasarım kalıbıyla, işlemi tetikleyecek nesneler ile işlemi yapan nesneler birbirlerinden ayrılmış olur. İşlemi yapacak nesneler birden fazla olabilir ve bunlar koleksiyonlarda saklanabilir. Bu nesnelerin hepsi birden sırayla çalıştırılabilir, böylece istenen fonksiyon, birden fazla işlevin çalışmasıyla gerçekleştirilmiş olur. Yeni işlevler istendiğinde, yeni komut nesneleri de kolayca eklenebilir.

Komut tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



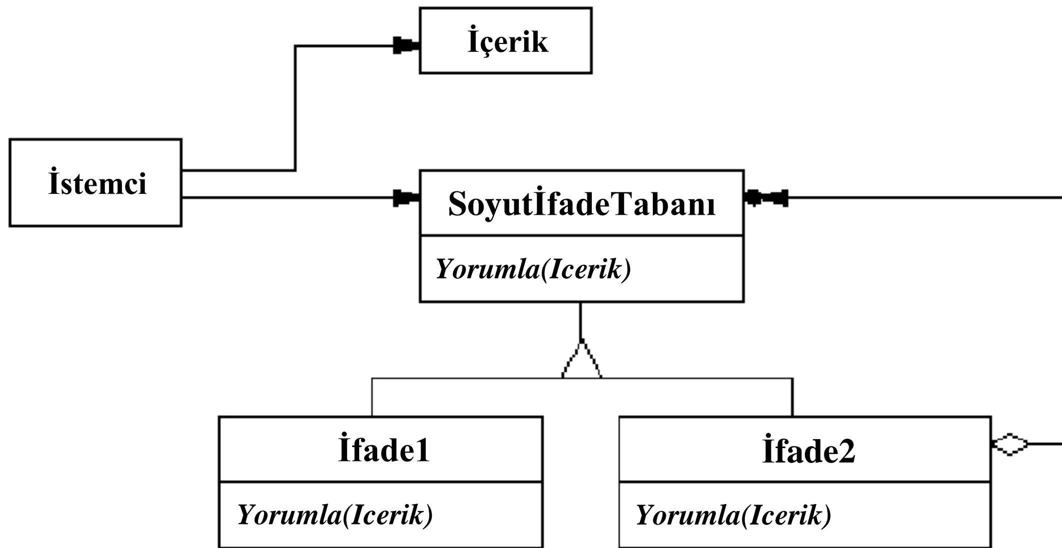
Şekil 3.18. Komut Tasarım Kalıbı UML Şeması [8]

3.6.3.3. Yorumlayıcı tasarım kalıbı

Yorumlayıcı (Interpreter) tasarım kalıbındaki amaç, özelleşmiş bir bilgisayar dilinin yorumlanmasının gerçekleştirilmesidir. Ana fikir, bu özelleşmiş dilin her bir sembolü için ayrı bir sınıf yaratmaktır. Bu tasarım kalıbı, veritabanı yönetim sistemlerindeki SQL tarzı dillerin yorumlanmasında kullanılabilir. Bir başka örnek olarak, bilgisayar

ağlarındaki iletişim protokollerindeki özel dillerin çözümlenmesinde kullanılabilir [21].

Yorumlayıcı tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



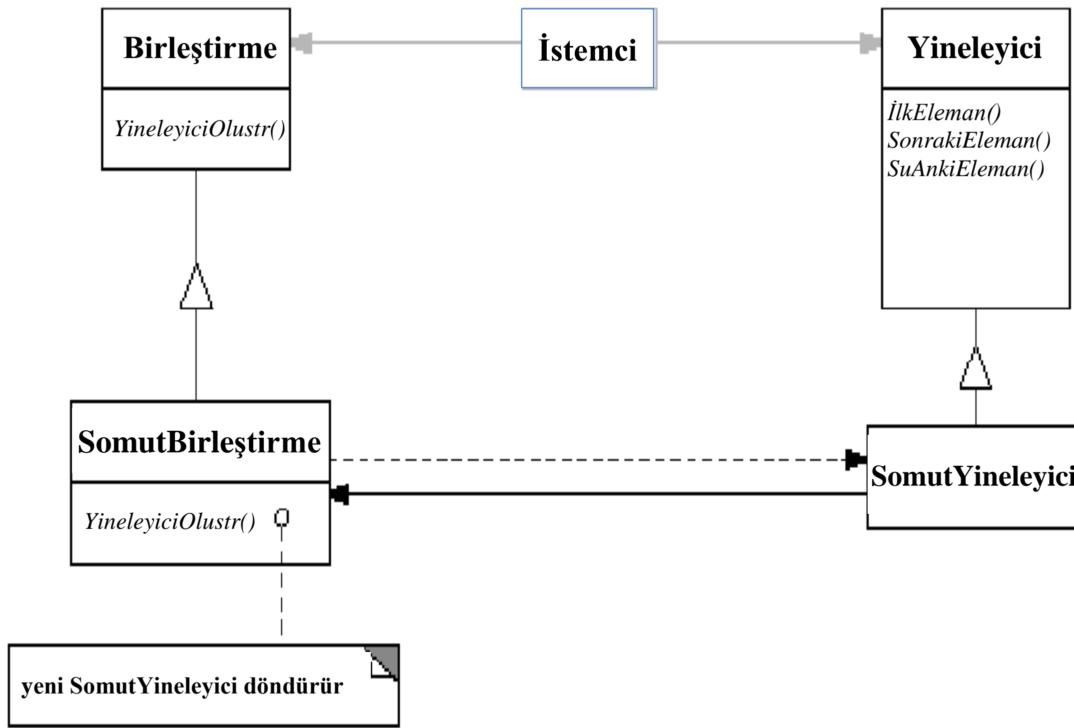
Şekil 3.19. Yorumlayıcı Tasarım Kalıbı UML Şeması [8]

3.6.3.4. Tekrarlayıcı tasarım kalıbı

Tekrarlayıcı (Iterator) tasarım kalıbı ile bir listede yer alan nesnelerin sırasıyla, listenin yapısını ve çalışma tarzının, uygulamanın diğer kısımları ile olan bağlantılarını en aza indirmek için uygulamadan soyutlama amaçlı kullanılabilmektedir [21].

Tekrarlayıcı tasarım kalıbı; birleşik bir nesnenin bileşenlerine, nesnenin esas ifadesinin gösterimini açığa çıkarmadan sırayla erişebilmeyi sağlar. Bir listenin yapısının ve çalışma tarzının uygulamanın diğer kısımları ile olan bağlantılarını en aza indirmek için; listede yer alan nesnelerin, sırasıyla uygulamadan soyutlanması amacıyla kullanılır [21].

Tekrarlayıcı tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



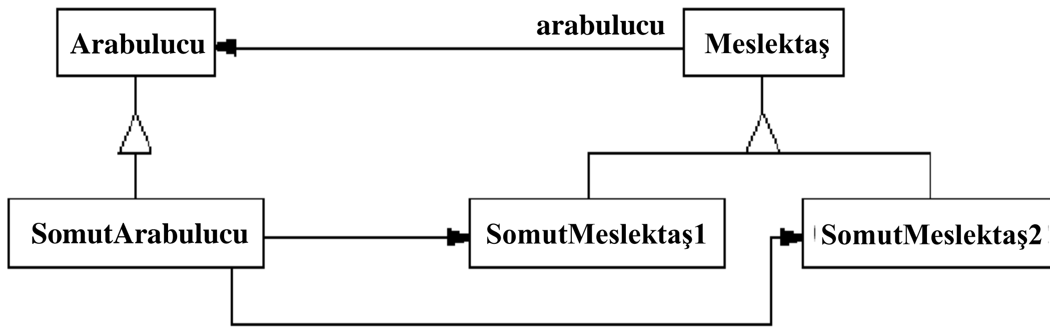
Şekil 3.20. Tekrarlayıcı Tasarım Kalıbı UML Şeması [8]

3.6.3.5. Arabulucu tasarım kalıbı

Arabulucu (Mediator) tasarım kalıbının kullanım amacı, nesne kümelerinin birbirleriyle nasıl haberleşeceğini düzenleyen bir ara nesnenin kullanılmasıdır.

Yazılım dünyasında bu konuya ilişkin verilebilecek en güzel örneklerden birisi de sohbet uygulamalarıdır. Bir sohbet uygulamasına dahil olan katılımcıların her birinin birbirleriyle iletişim kurarak konuşması, zaman içerisinde ağ yükünü arttıracak ve yönetilemez hale getirecektir. Bunun yerine katılımcılar arasındaki iletişimin yönetimini sağlayacak bir merkezin olması önerilir. Katılımcılar isteklerini arabulucu nesneye iletirler ve sonuçlardan haberdar edilirler. Sohbet uygulaması göz önüne alındığında Mediator aslında mesajlaşma işlemini üstlenen sunucu uygulama olarak düşünülebilir.

Arabulucu tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



Şekil 3.21. Arabulucu Tasarım Kalıbı UML Şeması [8]

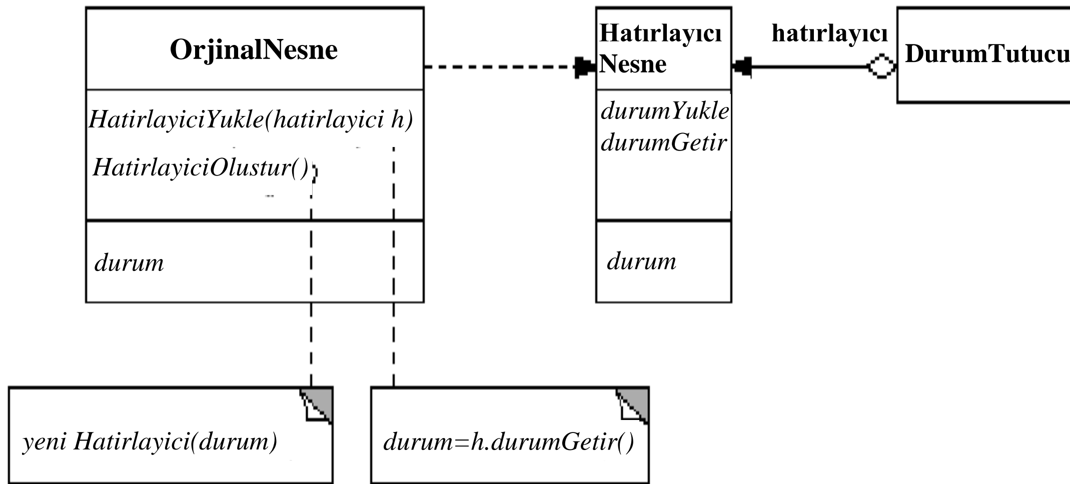
3.6.3.6. Hatırlayıcı tasarım kalıbı

Hatırlayıcı (Memento) tasarım kalıbı, temel olarak bir nesnenin daha önceki durumunun saklanması ve istenildiğinde tekrardan elde edilmesi üzerine tasarlanmış bir kalıptır. Nesnelere, dahili durumları için geri alma işlemi yeteneğininin kazandırılmasını sağlar.

Bu kalıpta durumu korunmak istenen nesnenin birebir kendisini veya en azından saklanmak istenen özelliklerini tutan kopyası yer alır (Hatırlayıcı). Diğer taraftan hatırlayıcı nesnesini oluşturan, bir başka deyişle kaydeden ya da var olduğu son durumu yükleyerek geri getiren işlevlere sahip olan asıl tipimiz yer almaktadır (Orijinal Nesne).

Bu temel tiplerin yanında, saklanan memento nesnesinin güvenli bir şekilde korunmasını sağlayacak bakıcı bir tip de yer almaktadır (Durum Tutucu).

Hatırlayıcı tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



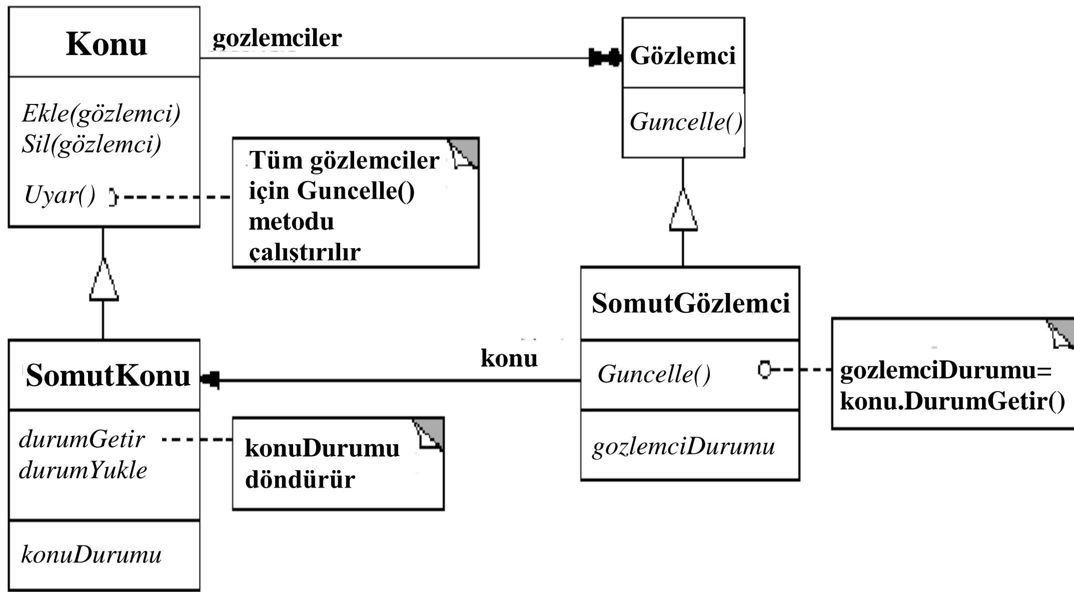
Şekil 3.22. Hatırlayıcı Tasarım Kalıbı UML Şeması [8]

3.6.3.7. Gözlemci tasarım kalıbı

Gözlemci (Observer) tasarım kalıbı, bir nesnenin kendi durumunda olabilecek değişikliklerden kendisine bağlı olan diğer abone nesnelerin haberdar edilmesi gereken durumlarda kullanılır. Bu nedenle izlenmeye değer bir konu ve bununla ilişkili nesneler söz konusudur. Ayrıca konuyu takip edecek olan tipler de bulunmaktadır.

Bir stok takip sisteminde, stok hareketlerinde olan değişimlerin bayilere bildirimesi, haber ajanslarının, kendilerine bağlı olan bölümlere yeni başlıklar geldikçe bilgilendirmede bulunmaları gibi senaryolarda kullanılabilir.

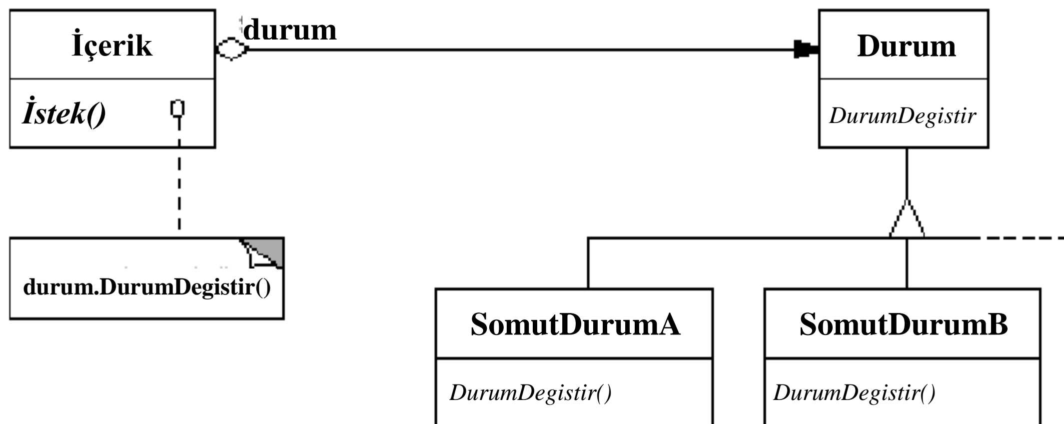
Gözlemci tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



Şekil 3.23. Gözlemci Tasarım Kalıbı UML Şeması [8]

3.6.3.8. Durum tasarım kalıbı

Nesnenin durumu değiştiğinde, davranışı da değişiyorsa, yani nesneler farklı durumlarda, farklı davranışlar gösteriyorsa, Durum (State) tasarım kalıbı kullanılabilir**.** Bu tasarım kalıbının kullanılması, nesnelerin durumlarına bağlı değişen davranışlarının karmaşık "if-else" veya "switch" ifadeleriyle kontrol edilmesini önler. Durum tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



Şekil 3.24. Durum Tasarım Kalıbı UML Şeması [8]

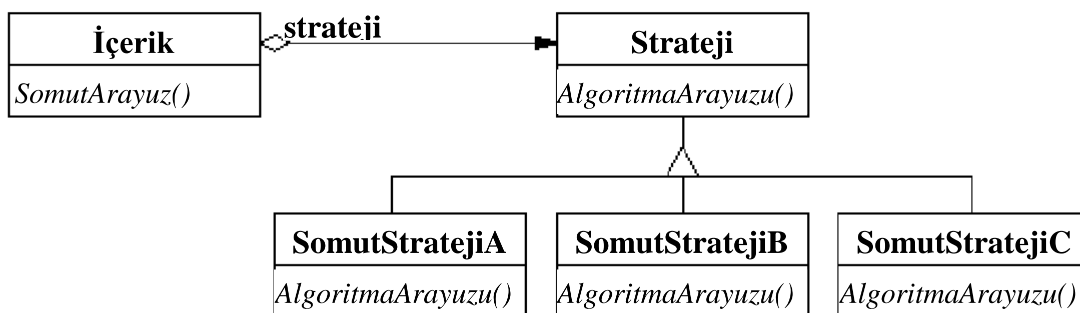
3.6.3.9. Strateji tasarım kalıbı

Strateji tasarım kalıbı temel olarak, bir nesnenin herhangi bir operasyonu gerçekleştirmek için kullanabileceği farklı algoritmaları içeren farklı tipleri kendi içerisinde ele alarak kullanması yerine, kullanmak istediği fonksiyonelliğin nasıl uygulandığını bilmesine gerek kalmaksızın sadece seçerek, çalışma zamanında yürütmesine olanak tanımaktadır.

Bir işlemi yapabilmek için birden fazla yöntem veya algoritma mevcut olabilir. Yerine göre bir yöntemi seçip, uygulamak için Strateji tasarım kalıbı kullanılır. Her yöntem bir sınıf içinde implemente edilir.

Örneğin bir string katarının şifrelenmesi ve şifresinin çözülmesi farklı algoritmalar vasıtasıyla yapılabilir. Bu durumda ilk akla gelen içerik tipi içerisinde söz konusu şifreleme seçeneklerini ele almaktır. Bu da bir sürü “if” veya “switch” ile olayı kontrol altına almak anlamına gelebilir. Veriyi yeni bir algoritmaya göre şifrelemek istediğimizde, içerik tipinin üzerinde kod değişikliği yapmamız gerekecektir. Halbuki içerik tipinin, çalışma zamanında kullanıldığı yerde, sadece şifreleme algoritmasını seçmesinin sağlanması bunun önüne geçebilir. Dikkat etmemiz gereken noktalardan birisi de, içerik tipinin şifreleme algoritmasının nasıl yapıldığının bilmesine gerek olmayışıdır.

Strateji tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :

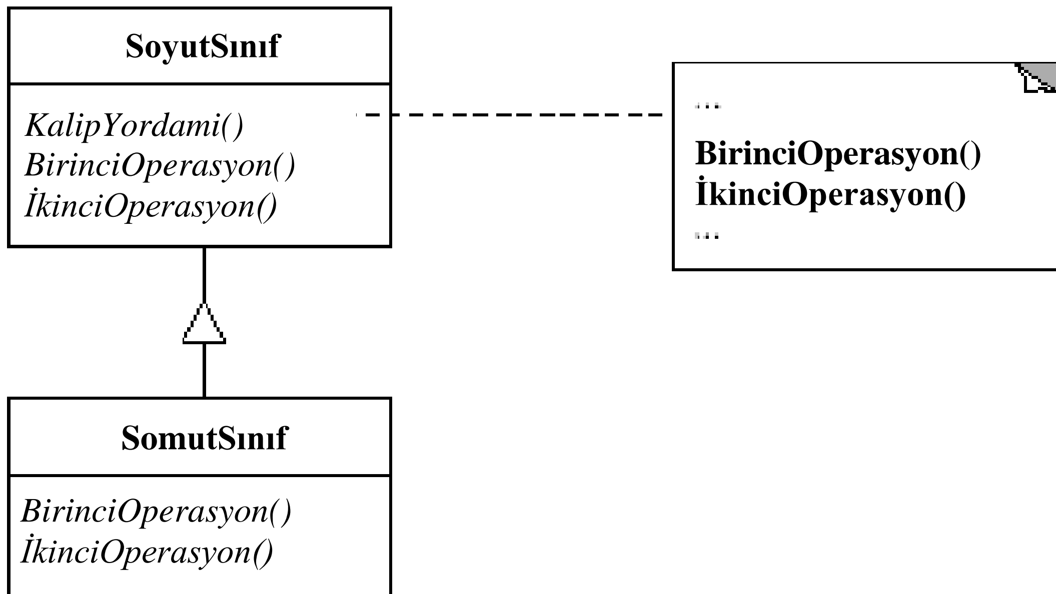


Şekil 3.25. Strateji Tasarım Kalıbı UML Şeması [8]

3.6.3.10. Kalıp Yordamı tasarım kalıbı

Kalıp yordamı (Template Method), sınıf hiyerarşisinde üst sınıfta yer alır. Bu yordam gerekli yöntemin adımlarını belirler. Bu yöntemin hangi adımlardan oluşması gerektiğini ortaya koyarken, alt sınıfların uyması gereken şablonu ortaya koyar. Alt sınıflar detayları kendileri belirler. Fakat uyulması gereken yapı üst sınıf tarafından, kalıp yordamı ile belirlenir. Bu tasarım kalıbı sayesinde alt sınıflarda yapılacak kod tekrarlarından kaçınılır. Alt sınıfların ortak kodları, üst sınıfta tek bir yerde toplanır. Bu ortak kısımda bir değişikliğe ihtiyaç duyulduğunda, bu tek noktada yapılır. Kısaca, üst sınıfta yer alan kalıp yordamı, tüm alt sınıfların ihtiyaç duyduğu ortak adımları barındırır, bunlar için bir kalıp oluşturur ve detayları alt sınıflara bırakır.

Kalıp Yordamı tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :

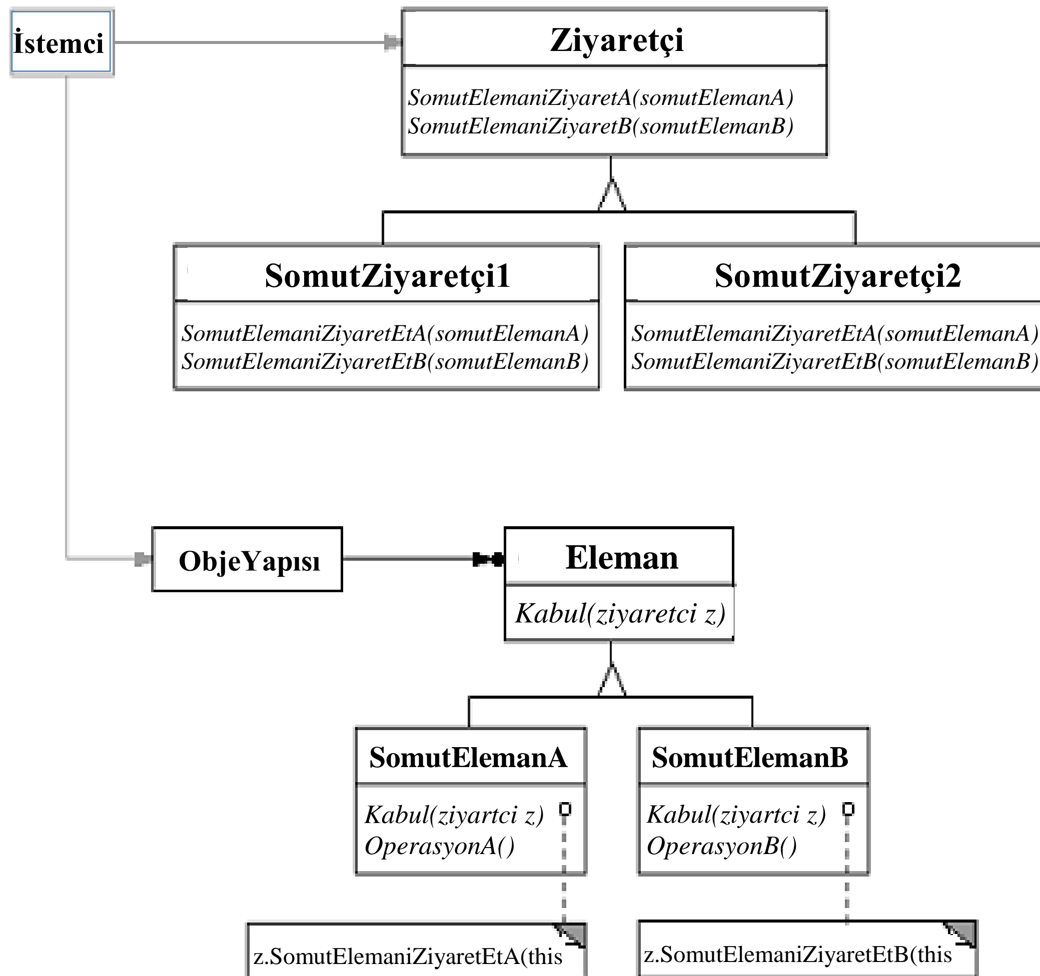


Şekil 3.26. Kalıp Yordamı Tasarım Kalıbı UML Şeması [8]

3.6.3.11. Ziyaretçi tasarım kalıbı

Ziyaretçi (Visitor) tasarım kalıbı, çok sayıda ve farklı tipteki nesneler üzerinde işlem yapabilmek amacıyla kullanılır. İşlem yapılacak nesnelerde herhangi bir değişiklik yapılmaz. İşlemi ziyaretçi nesneleri yapar. Eğer sisteme yeni nesneler eklenmiyor, fakat sık sık yeni işlemlerin eklenmesi gerekiyorsa bu tasarım kalıbı kullanılabilir. Bu tasarım kalıbının kullanılmasıyla, yapılacak işlemle ilgili kodlar merkezi bir nesnede toplanır.

Ziyaretçi tasarım kalıbının UML Şeması aşağıdaki şekilde gösterilmektedir :



Şekil 3.27. Ziyaretçi Tasarım Kalıbı UML Şeması [8]

BÖLÜM 4. EOBS UYGULAMASI

Üniversitelerin eğitim öğretim süreçleri Bologna Süreci kapsamında belirlendiğinden, bu sürece uygun eğitim öğretim programlarını belirlemek isteyen üniversitelerin yapacağı çalışmalar, ortak pek çok nokta barındırmaktadır. Bu ortak noktaları bünyesinde barındıran bir çerçeve yazılımı sayesinde, tüm yükseköğretim kurumları kendileri için en başından bir eğitim öğretim bilgi sistemi (EOBS) yazılımı geliştirme durumunda kalmayacaklardır. Her üniversitenin kendi EOBS yazılımını kendisinin geliştirmesi, yazılım mühendisliğinin temel kavramlarından olan "tekrar kullanılabilirlik" ve "taşınabilirlik" felsefesine uymamakta ve kaynakların aynı işlemler için tekrar tekrar harcanmasına sebep olmaktadır. Bu nedenle üniversitelerin eğitim öğretimlerini güncellemeleri için hazırlanacak projelere temel oluşturacak bir çerçeve yazılımı geliştirmek, gereksiz kaynak kullanımının önüne geçecektir.

Bologna Süreci'nin sürekli güncellenen ve yeni eklemeler yapılan bir süreç olması ve üniversitelerin özgünlüklerinin veya farklılıklarının da ortaya konmasına imkan sağlaması, eğitim öğretim bilgi sistemini de buna göre tasarlamayı gerektirmektedir. Belli bir seviyeye kadar geliştirilmiş ve temel özellikleri bünyesinde barındıran EOBS çerçeve yazılımı, istendiğinde üniversitelerin ihtiyaçları doğrultusunda genişleyebilir olmalıdır. Bu genişlemeye imkan verecek yapıların oluşturulmasında tasarım kalıpları önemli bir role sahiptir.

Çalışmanın bu bölümünde üniversitelerin, Bologna Süreci kapsamında eğitim öğretimlerini güncellemeleri için geliştirilen EOBS çerçeve yazılımının mimarisi açıklanmış ve tasarım kalıplarının kullanımı incelenmiştir. Tasarım kalıpları kullanmanın EOBS çerçeve yazılımının geliştirilmesi sürecinde sağladığı avantajlara değinilmiş ve ileride olabilecek güncellemelere nasıl imkan verdiği anlatılmıştır.

4.1. EOBS Çerçeve Yazılımının Amacı

Bologna Süreci kapsamında, yükseköğretim kurumlarındaki eğitim-öğretim programlarının öğrenme çıktıları, yüksek öğretim ulusal yeterlilikleri çerçevesi ve temel alan yeterliliklerine bağlı olarak tanımlanması gerekmektedir. Bu çerçevede yükseköğretim kurumlarında yapılması önerilen çalışmalar aşağıda verilmiştir [30].

- Üniversitelerin misyon, vizyon ve nasıl bir mezun profilinin amaçlandığını gösteren, programın hedef ve amaçlarını yazmaları,
- Mezunların ne tür yeterliliklere sahip olmaları gerektiğinin açıklandığı, program çıktılarını belirlemeleri,
- Eğitim-öğretim planlarında yer alan her bir ders için öğrenme çıktılarını yazmaları,
- Ders öğrenme çıktıları ile program çıktılarını ilişkilendirmeleri,
- Ders öğrenme çıktılarını göz önüne alarak ders öğretim planlarını hazırlamaları,
- Ders öğrenme çıktılarına ulaşabilmek için gerekli iş yükü ve AKTS kredilerini hesaplamaları,
- Her programın, programlarını tanıtıcı bilgiyi ve ders öğretim planlarını hazırlamaları

EOBS yazılımı, yukarıda maddeler halinde verilen; üniversitelerin Bologna Süreci kapsamında yapması gereken çalışmaların ortak bir platformdan yapılmasına imkan veren bir otomasyon sistemidir. Bu sistemin amacı, programlarla ilgili tanıtım yapılabilmesine, program çıktıları ve ders öğrenme çıktılarının hazırlanmasına, öğrenme çıktıları ile program çıktılarının ilişkilendirilmesine, programın eğitim planı ve programa ait ders kataloğunun oluşturulmasına, ders öğretim planlarının ve haftalık ders akışlarının hazırlanmasına, dersle ilgili kaynak ve materyallerin paylaşılabilmesine, web üzerinden imkan sağlamaktır.

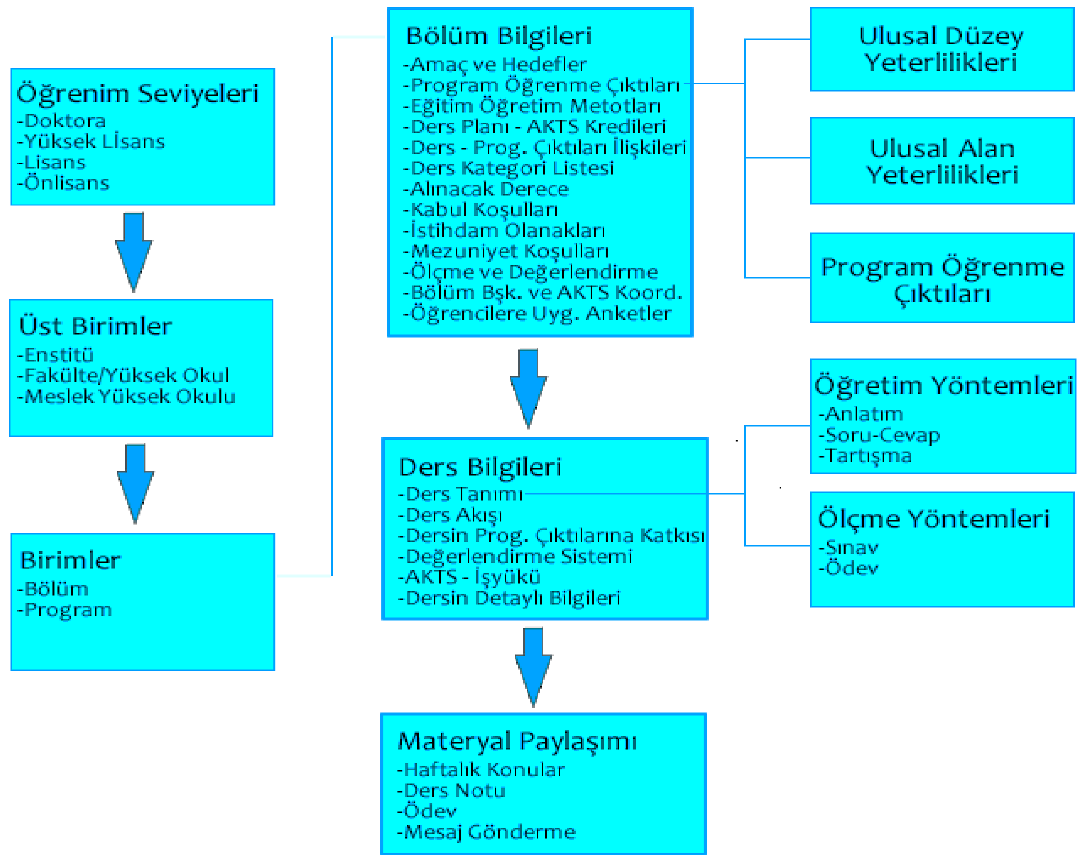
4.2. EOBS Çerçeve Yazılımı Mimarisi

EOBS çerçeve yazılımı, temel olarak program ve ders bilgileri olmak üzere iki ana yapı üzerine inşa edilmiştir. Program Bilgileri modülünde bölüm ve programların tanımlanması, amaç ve hedeflerinin girilmesi ve program yeterliliklerinin belirlenmesi gibi işlemler yer alırken, Ders Bilgileri modülünde ise ders içeriklerinin hazırlanması, ders planı ve AKTS kredilerinin hesaplanması ve dersle ilgili kaynak ve materyallerin paylaşılması gibi sisteme özel işlemlere imkan tanınmaktadır.

EOBS çerçeve yazılımı, yukarıda örnekleri verilen sisteme özel yapıları barındırmasının dışında, web tabanlı bir sistemde olması gereken doğrulama, yetkilendirme, işlem kaydı tutma, raporlama güvenli bir hesap yönetimi gibi işlevleri de gerçekleştirmektedir.

EOBS çerçeve yazılımda temel olarak bulunan modüller aşağıda gösterilmektedir :

- Bölüm / Program bilgileri modülü
- Ders bilgileri modülü
- Yetkilendirme modülü
- İşlem kaydı tutma modülü
- Raporlama modülü



Şekil 4.1. EOBS Çerçeve Yazılımında Kullanılan Temel Modüller

Yukarıda belirtilen modüller, birbirinden bağımsız olarak sorumlu oldukları işlemleri yerine getirseler de sahip oldukları sınıflar ve işlevler açısından birbirlerini etkilemekte ve birbirlerinin sınıf ve metodlarını kullanabilmektedirler. Örneğin ders çıktılarının program yeterliliklerine katkısının düzenlenmesi ve gösterilmesi bu ilişkilerin bir sonucudur.

Tasarım kalıpları, tasarımın yeniden kullanılabilirliğine dayanmaktadır. Ancak bir sistemde kullanılan tasarım kalıpları, sistemin belli özelliklerini iyileştirirken, diğer bazı özelliklerine ise olumsuz yönde etki edebilmektedir. Bu nedenle bir sistem tasarlanırken kullanılacak tasarım kalıplarının ihtiyaca ne derece katkı sağladığına ve getirdiği avantajların ve dezavantajların neler olduğuna dikkat edilmelidir.

Nesneye dayalı programlama paradigması çerçevesinde C# programlama dili kullanılarak geliştirilen EOBS çerçeve yazılımında, yukarıda bahsedilen modüller arasındaki ilişkilerin sağlanmasını kolaylaştırmak ve uygulamanın sınıf hiyerarşisinin bir noktadan sonra karmaşık ve anlaşılması zor bir hale gelmesine meydan vermemek için, sistemde bulunan sınıflar ve nesneler en uygun tasarım kalıpları kullanılarak geliştirilmiştir.

4.3. Bölüm / Program Bilgileri Modülünde Kullanılan Tasarım Kalıpları

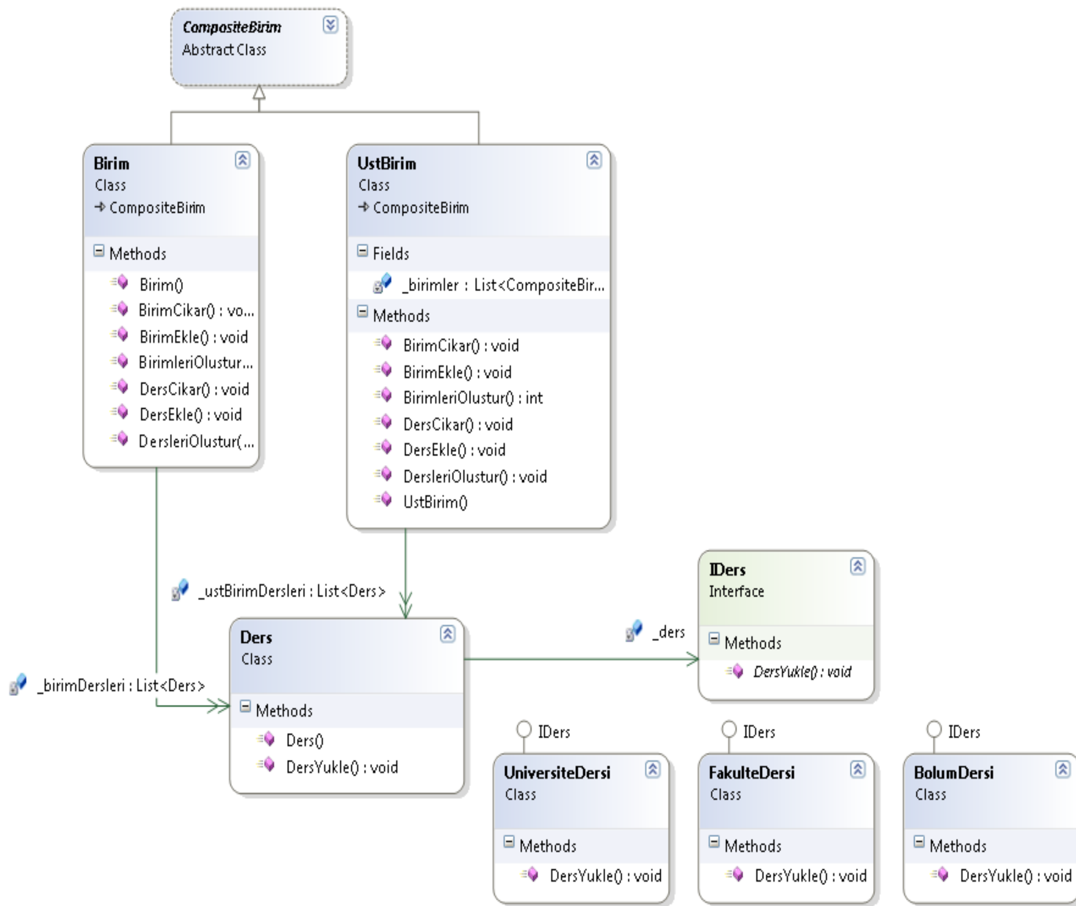
Üniversitelerin eğitim öğretim sistemlerini düzenlemek ve güncellemek için geliştirecekleri yazılımlarda bulunması gereken ortak bileşenlerden biri bölüm ve program bilgilerinin tanımlanması, bu programlara ait amaç, hedef ve yeterlilik gibi bilgilerin düzenlenmesini sağlayan yapılardır.

Üniversitelerdeki bölüm ve programları temsil eden birimler hiyerarşik bir yapıda bulunmaktadır. Örneğin bir fakültenin altında bölümler ve bölümlerin altında da ana bilim dalları vardır. Bu tip hiyerarşik yapıların yazılımda temsil edilmesi gerektiği durumlarda Bileşik (Composite) tasarım kalıbı kullanmak oldukça yerinde olacaktır. Bileşik tasarım kalıbı sayesinde sistemde bölüm ve programları temsil eden hiyerarşik yapıların kolay bir şekilde gerçeklenmesinin yanısıra bu hiyerarşik yapıya yeni birimler eklenmesi de oldukça kolay ve genişleyebilir bir yapıya kavuşmaktadır. Üniversiteler kendi eğitim birimlerinin hiyerarşik yapısını sisteme kolayca ekleyebilmektedirler.

Bölüm ve programları temsil eden yapılar hiyerarşik açıdan birbirlerine benzeseler de öğretim seviyelerinin ön lisans, lisans, yüksek lisans ve doktora olmasına göre yüksekokul, fakülte, enstitü gibi farklı birimleri temsil edebilecek yapıda olması gerekmektedir. Bu farklı birimlerin sahip olacağı özellikler de değişmektedir. Örneğin bu birimler altındaki hiyerarşik yapılar, öğrenim süreleri, AKTS iş yükü ve mezuniyet şartları gibi bilgiler birbirinden farklı olduğu gibi, bu birimlerde gösterilen dersler de farklı özelliklere sahip olabilmektedir. Tasarım kalıplarının kullanılmadığı yöntemde, fakülte veya bölüm gibi farklı seviyelerde okutulan derslerin farklı özelliklere sahip olmasından dolayı her ders tipi için somut bir sınıf tanımlamak ve

bu dersin okutulduğu birimlere ait sınıfların da bu derslerden haberdar edilmesi gerekmektedir. Bu da sınıflar arası sıkı bağlı bir yapının oluşmasına ve programın taşınamaz olmasına sebep olacaktır. Fakat bir nesnenin farklı özelliklere sahip sunumlarının olduğu durumlarda kullanılan Strateji tasarım kalıbı sayesinde, Birim ve UstBirim sınıflarının IDers arayüzünü uygulayan somut ders sınıflarıyla bağlantısı koparılmıştır. Birim sınıfları IDers arayüzünü kendi içerisinde barındıran bir sınıf sayesinde, hangi ders tipine ait nesne örneğinin kullanıldığıyla ilgilenmemektedir. Bu da birimlere ait sınıflarla bu birimlerde okutulan derslere ait sınıflar arasında sıkı bağlı bir yapının oluşmasına engel olmaktadır. Bu yapı birimlere bağlı olabilecek diğer sınıflar için de kullanılabilir.

Şekil 4.2'de birimlere ait sınıflar ile derslere ait sınıflar arasındaki ilişkiler gösterilmektedir.



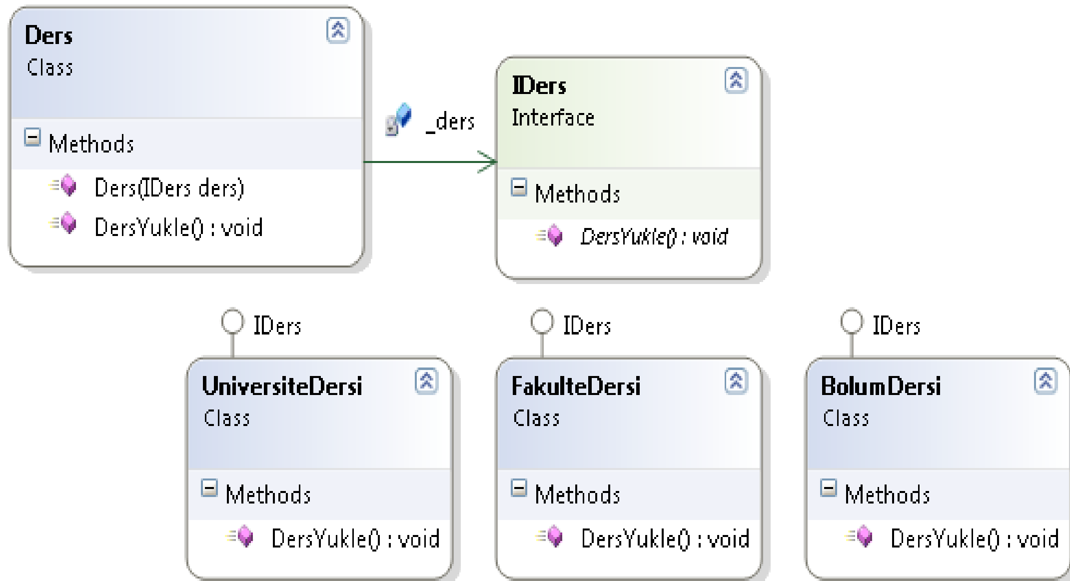
Şekil 4.2. Bölüm/Program Modülünde Kullanılan Bileşik ve Strateji Tasarım Kalıbı

4.4. Ders Bilgileri Modülünde Kullanılan Tasarım Kalıpları

Üniversitelerdeki eğitim öğretim bilgi sistemlerinde bulunması gereken temel yapılardan bir tanesi de ders içeriklerinin girilmesi, dersin program çıktılarına katkısı, AKTS iş yükü, dersin doküman ve kaynaklarının paylaşılması gibi işlemlerin gerçekleştirilmesini sağlayacak ders bilgileri modülüdür.

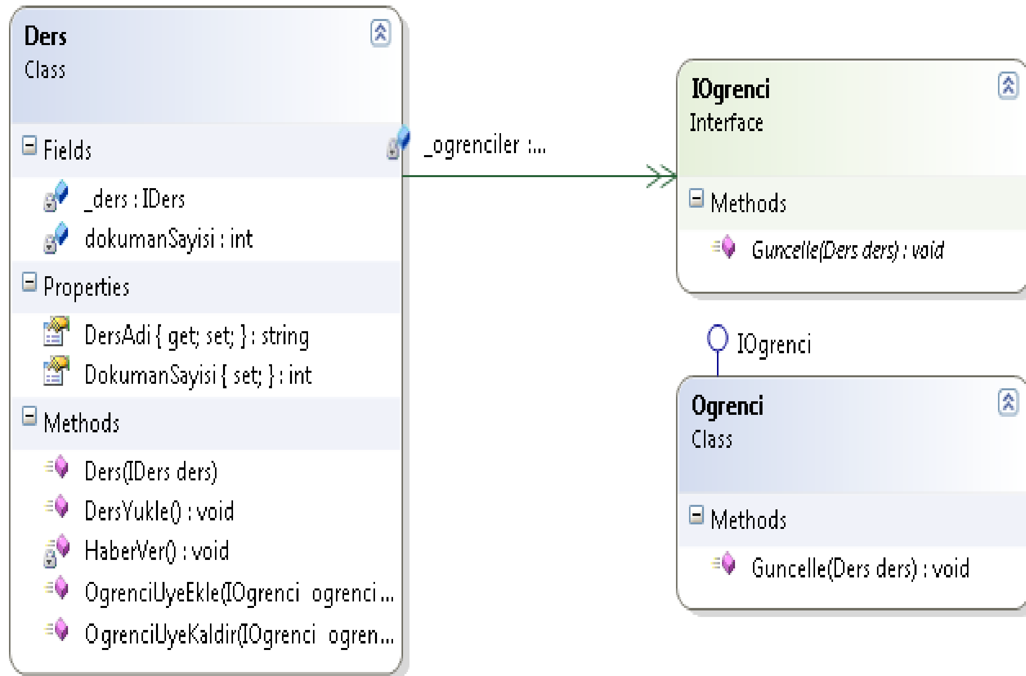
Üniversitelerde gösterilen dersler; üniversitenin tüm bölümlerinde okutulan dersler, belirli bir fakültedeki tüm bölümlerde okutulan dersler ve sadece bölüme özel dersler olmak üzere üç sınıfta toplanmaktadır. Aynı zamanda derslerin zorunlu veya seçmeli olması gibi durumlar da söz konusudur.

Ders bilgilerinin gösterildiği arayüz farklı ders tiplerinden etkilenmemelidir. Bu durumun sağlanabilmesi için ders bilgilerini gösteren arayüz kendisine verilen ders nesnesinin hangi sınıfın bir nesne örneği olduğunu bilmemelidir. Bu arayüzün bilmesi gereken tek şey kendisine verilen ders nesnesinin IDers arayüzünü uygulamış olduğudur. Uygulamada bunu sağlamak için de Strateji tasarım kalıbı kullanılmıştır.



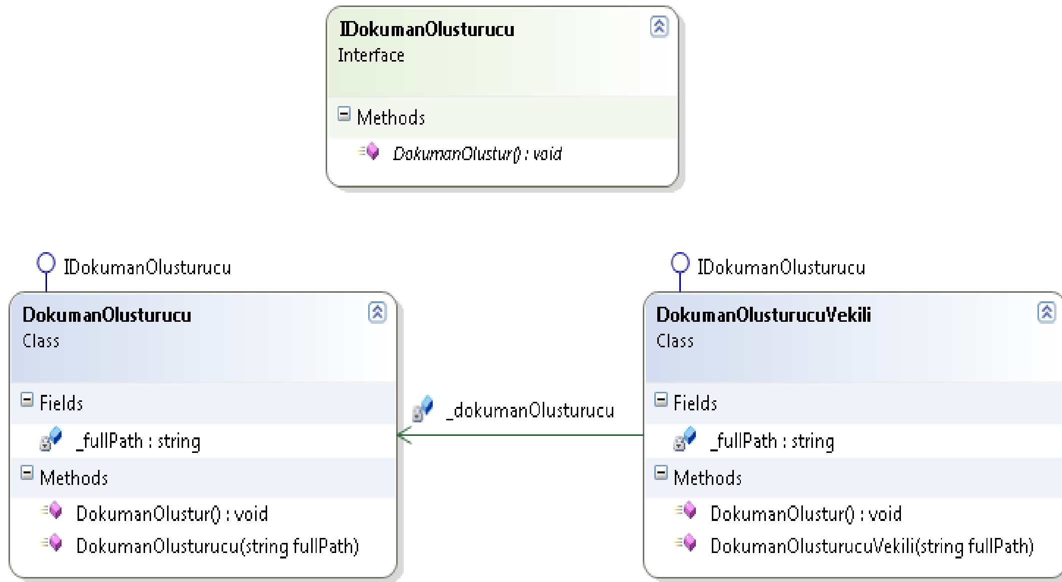
Şekil 4.3. Ders Modülünde Kullanılan Strateji Tasarım Kalıbı

Dersin kaynakları ve materyallerinin dersi veren öğretim üyesi tarafından haftalık olarak sisteme yüklenmesi dersin akışı ve işleyişi açısından önemlidir. Bu sayede öğrenciler dersle ilgili materyal ve kaynaklara sistem üzerinden erişebilmekte ve bu materyalleri kendi bilgisayarlarına indirebilmektedir. Aynı zamanda sisteme yüklenen kaynak ve materyallerin dersi alan öğrencilere bildirilmesi, sisteme düzenli olarak giriş yapmayan öğrencilerin de yüklenen materyalden haberdar edilmesini sağlayacaktır. Bu bilgilendirme işlemi için Gözlemci (Observer) tasarım kalıbı kullanılmıştır. Gözlemci tasarım kalıbı, sisteme üye olan nesnelerin, sistemin herhangi bir yerinde olabilecek bir değişiklikten haberdar edilmesi esasına dayanır [8]. Burada materyal yüklenecek dersi alan öğrenciler o dersin doğal üyesidirler. Sisteme bir materyal eklenmesi durumunda Gözlemci tasarım kalıbı sayesinde sistem önceden tanımlanan herhangi bir yöntemle –mail yoluyla veya sistem içerisinde geliştirilebilecek mesajlaşma modülüyle olabilir- otomatik olarak dersi alan öğrenciye bilgilendirme mesajı gönderebilecektir.



Şekil 4.4. Ders Modülünde Kullanılan Gözlemci Tasarım Kalıbı

Aynı zamanda sisteme yüklenen ders materyalleri, sistem üzerinden de görüntülenebilir. Bu materyaller video, resim veya sunuları da içeren büyük boyutlu dosyalar olabilir. Bu durumda bu dosyaların sürekli sisteme yüklenmesi performans açısından büyük sorunlara yol açabilir. Binlerce kullanıcısı olan web tabanlı bir sistemde böyle bir durum sistem kaynaklarının tüketilmesine ve sistemin kullanıcıdan gelen isteklere cevap verememesine neden olabilir. Bu problemin çözümü için de Vekil (Proxy) tasarım kalıbı kullanılmıştır. Proxy tasarım kalıbının amacı, oluşturulmaları zaman alan ve sistem kaynaklarını zorlayan nesnelerin oluşturulmalarına aracılık etmektir [8]. Sistem, oluşturulacak nesneyi proxy nesne sayesinde önbelleğinde tutarak daha önce oluşturulmuş nesneyi kullanıcıya gösterir. Bu şekilde sistem kaynakları zorlanmamış olur.



Şekil 4.5. Ders Modülünde Kullanılan Vekil Tasarım Kalıbı

4.5. Uygulamanın Genel İşleyişinde Kullanılan Tasarım Kalıpları

Bu kısımda, EOBS yazılımından bağımsız, web tabanlı bir bilgi sisteminde olması gereken; kullanıcı doğrulama, menü yapısının oluşturulması, veritabanı işlemleri için gerekli sınıfların düzenlenmesi gibi diğer uygulamalarda da kullanılabilecek yapılarda kullanılan tasarım kalıpları incelenmiştir.

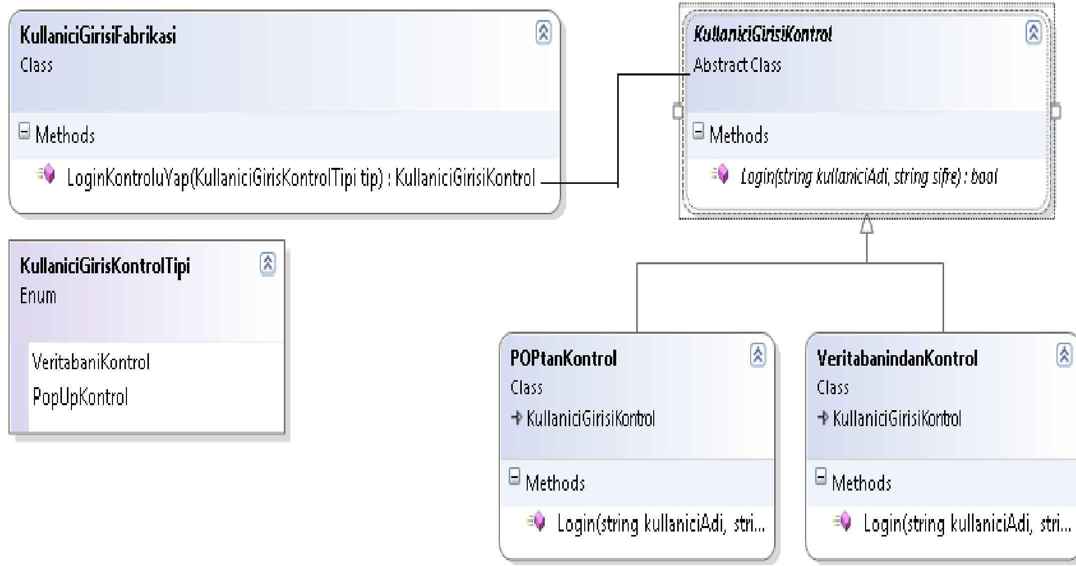
4.5.1. Kullanıcı doğrulama işlemleri

Web tabanlı eğitim öğretim bilgi sistemlerinde genel erişime açık kısımların haricinde kişiye özel sayfaların gösterilmesi, doküman eklenebilmesi veya ilgili derse ait kaynaklara erişilebilmesi gibi işlemleri gerçekleştirilebilmek için belli yetkilere sahip olan kullanıcılar tarafından erişilebilen kısımlar da bulunmak durumundadır. Bahsedilen yetkilerin neler olduğunun belirlenmesinden önce kullanıcının sisteme giriş yetkisinin olup olmadığının bilinmesi gerekmektedir. Bu işleme doğrulama işlemi denmektedir.

Doğrulama, bir istemcinin kim olduğunun belirlenmesidir. Bu işlem normal olarak bir kullanıcı adı ve parola kontrolü ile gerçekleştirilir. Yetkilendirme ise doğrulama aşamasını geçmiş bir kullanıcının sistemde bulunan kaynaklara erişebilme seviyesinin belirlenmesidir [31].

Doğrulama işlemi yapabilmek için sisteme giriş yetkisi olan kullanıcıların kullanıcı adı ve şifre bilgisinin bir yerlerde tutuluyor olması gerekmektedir. Bu işlem için farklı alternatifler söz konusu olabilmektedir. Örneğin bu işlem, bir veritabanı yönetim sisteminde tutulan kayıtların kontrolü şeklinde olabileceği gibi bir mail sunucusunda tutulan mail hesaplarının pop kontrolü ile sorgulanması veya bir LDAP kontrolü şeklinde de olabilir. Doğrulama işlemi için üniversitelerin diğer otomasyon sistemlerinin kullandıkları yöntem önemlidir. Bu yüzden EOBS çerçeve yazılımının, farklı doğrulama işlemlerinin de sisteme entegre edilebilmesine izin vermesi gerekmektedir. Bir başka deyişle, sistemde herhangi bir değişiklik yapmadan sadece sisteme dahil edilecek doğrulama sisteminin, gerektirdiği sınıf ve yapıları kodlamak yeterli olmalıdır.

Uygulamada farklı doğrulama sistemlerinin sisteme dahil edilebilmesine olanak sağlamak amacıyla Fabrika Metodu tasarım kalıbı kullanılmıştır. Fabrika Metodu tasarım kalıbına göre, kullanıcının doğrulama işleminin nasıl yapıldığını bilmesine gerek yoktur. Kullanıcı bu görevi, doğrulama işlemini gerçekleştirecek sınıfın bir metoduna havale etmiştir. Bu sayede kullanıcının sadece bu metoda doğrulama işleminin türünü, parametre olarak göndermesi yeterli olacaktır.



Şekil 4.6. Doğrulama İşleminde Kullanılan Fabrika Metodu Tasarım Kalıbı UML Şeması

Şekil 4.6'daki diyagramda kullanıcı doğrulama işleminin türünü belirleyen sınıflar KullaniciGirisiKontrol adlı soyut sınıftan türetilmişlerdir. Burada ata sınıfta bulunan Login metodu alt sınıflarda doğrulama işleminin türüne göre çok biçimliliği destekleyecek şekilde yeniden yazılmaktadır. Kullanıcı, doğrulama işlemini gerçekleştirmek için, doğrulama işleminin türünü KullaniciGirisiFabrikasi sınıfının LoginKontroluYap metoduna parametre olarak geçmek suretiyle gerçekleştirmektedir. Bu sayede hem PoptanKontrol ve VeritabanindanKontrol adlı sınıflar kullanıcıdan soyutlanarak kullanıcının doğrulama işlemi için karmaşık işlemler yapmasının önüne geçilmiş hem de sistem, yeni bir doğrulama işlemi türü eklenebilmesi açısından daha esnek hale getirilmiştir.

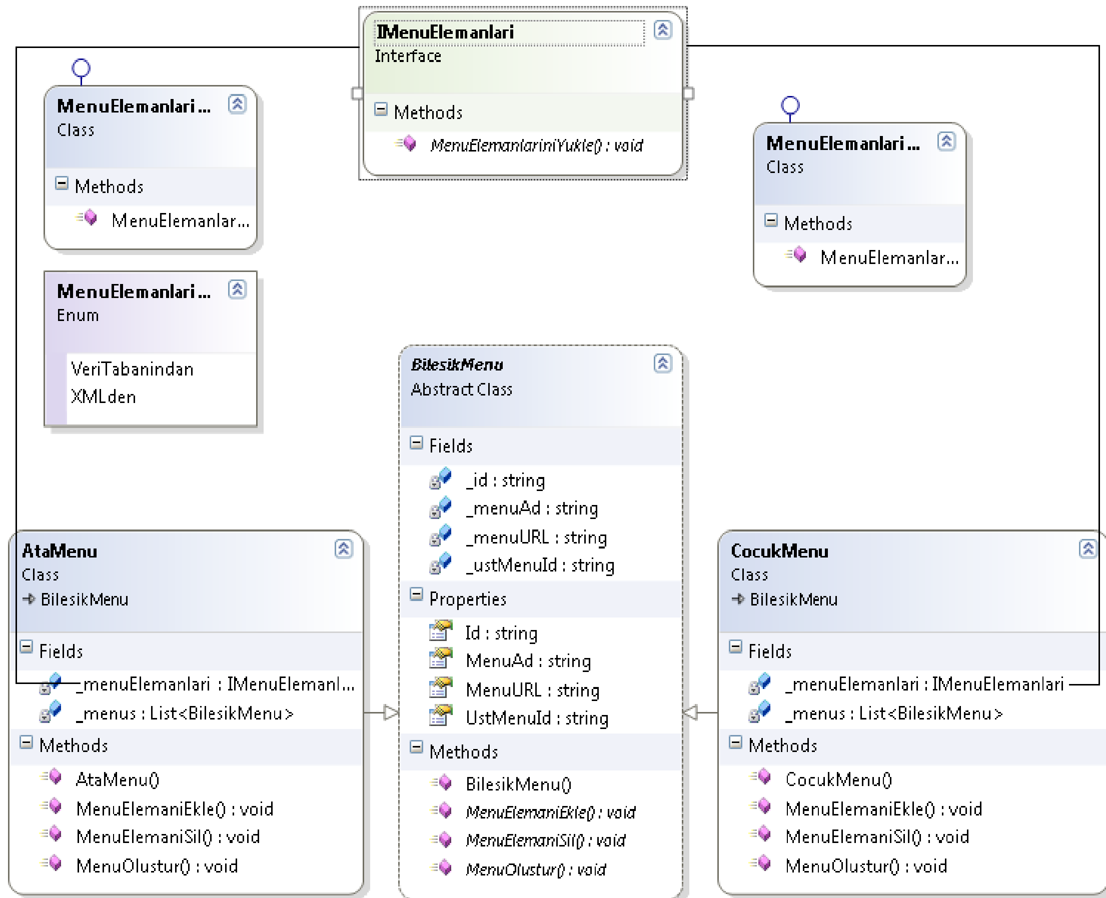
4.5.2. Menü yapısının oluşturulması

Web tabanlı bilgi sistemlerinde, kullanıcıların sayfalar arasında gezinmelerine imkan sağlamak için kullanılan en önemli kontrollerden birisi menü kontrolüdür. Sayfalar arasındaki hiyerarşinin temsil edilmesinde kullanılan menülerin statik bir yapıda

olması, sisteme ileride yeni sayfalar veya özelliklerin eklenmesi durumunda güncellemeyi zorlaştıracak, bu da sistemin esnekliğini azaltacaktır.

EOBS çerçeve yazılımında kullanılan menü elemanlarının görünüm adlarının ve yönlendirecekleri sayfa bilgisinin farklı veri kaynaklarından alınması gerekebilir. Menü yapısını tutan bilgiler, bir veritabanından veya bir XML dosyasından alınabileceği gibi sonradan sisteme eklenecek farklı bir konumdan da çekilebilir. Uygulamada menü elemanlarının dinamik olarak hangi kaynaktan alınacağının belirlenmesi için Fabrika Metodu tasarım kalıbı kullanılmıştır.

Menüler aynı zamanda alt menüler de barındırdığı için hiyerarşik bir yapı söz konusudur. Bu hiyerarşik yapının temsil edilmesi ve sisteme sonradan eklenebilecek yeni sayfalara yönlendirme yapacak menü elemanlarının da dinamik bir şekilde sisteme entegre edilebilmesi için Bileşik tasarım kalıbından yararlanılmıştır.



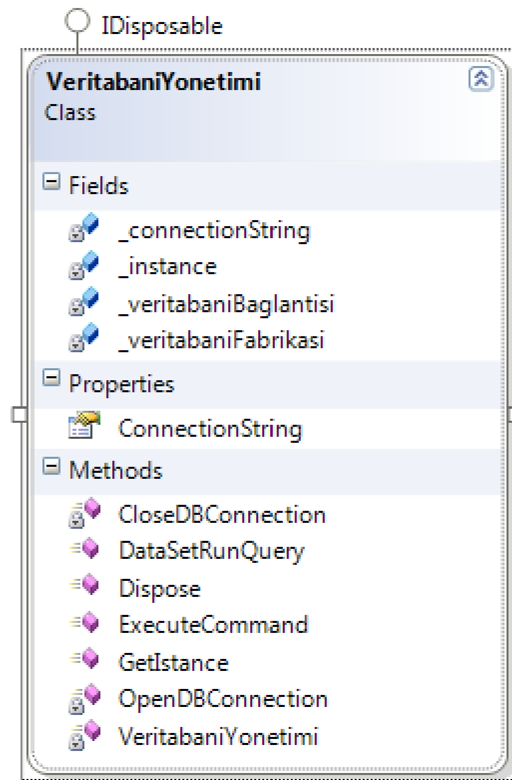
Şekil 4.7. Menülerde Kullanılan Bileşik ve Strateji Tasarım Kalıpları UML Şeması

Şekil 4.7'deki diyagramda menüler arasındaki hiyerarşik yapıyı temsil eden sınıflar, BilesikMenu soyut sınıfından türeyen AtaMenu ve CocukMenu sınıflarıdır. BilesikMenu sınıfının MenuOlustur metodu, kendi içersinde alt menülerin de MenuOlustur metodunu çağırdığından dolayı menü hiyerarşisindeki herhangi bir menünün oluşturulmasıyla menü altındaki tüm alt menüler de oluşturulmuş olacaktır.

Uygulamada menü elemanları yüklenirken kullanılacak veri kaynağı bir veritabanı yönetim sistemi veya bir XML dosyası olarak düşünülmüştür. Fakat farklı veri kaynaklarının kullanılmak istenmesi durumunda IMenuElemanlari ara yüzünü uygulayacak bir sınıfın sisteme eklenmesi yeterli olacaktır. Çünkü AtaMenu ve CocukMenu sınıflarının, menü bilgilerinin hangi kaynaktan geldiğini düzenleyen sınıflarla bir bağımlılığı yoktur. Diyagramdan da anlaşılacağı gibi AtaMenu ve CocukMenu sınıfları IMenuElemanlari arayüzüne ait bir nesne örneğini kendi içersinde kullanmaktadır. Bu sayede Menu sınıflarında bir değişiklik yapmadan sisteme yeni veri kaynakları eklemek mümkün olmaktadır.

4.5.3. Veritabanı işlemleri

Uygulamanın tamamında kullanılacak, veritabanıyla ilgili işlemleri kullanıcıdan soyutlayan ve veritabanı işlemlerini tek bir merkezden yapmaya olanak sağlayan VeritabaniYonetimi adlı bir sınıf oluşturulmuştur. Bu sınıf içerisinde ilk olarak, veritabanı bağlantısının sistemde sadece bir kez oluşturulmasını garanti altına alarak sistem kaynaklarının gereksiz kullanımına engel olmak ve sistemde veritabanına yapılan bir bağlantı varken yeni bir bağlantının açılmasına engel olmak için Tekil tasarım kalıbı kullanılmıştır.



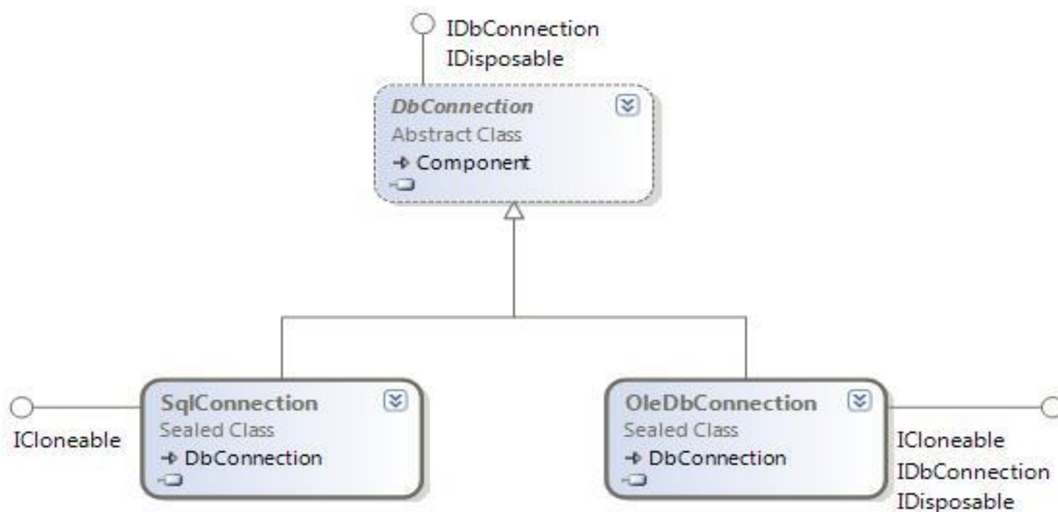
Şekil 4.8. Veritabanı Bağlantısı İçin Kullanılan Tekil Tasarım Kalıbı UML Şeması

Şekil 4.8'deki sınıf yapısında önemli olan nokta, VeritabaniYonetimi sınıfının yapıcı metodunun "private" anahtar kelimesi ile belirtilmiş olmasıdır. Bu durumda VeritabaniYonetimi sınıfı, dışarıdan "new" anahtar kelimesiyle oluşturulamayacaktır. Bu sınıfın bir nesne örneğini kullanmak isteyen başka bir sınıf, VeritabaniYonetimi sınıfının GetInstance metodunu çağırmak durumdadır. GetInstance metodu içerisinde _instance değişkeninin içeriğine bakılarak sistemde daha önce kullanılan bir VeritabaniYonetimi nesnesinin olup olmadığı kontrol edilir. Eğer böyle bir nesne yoksa yeni bir VeritabaniYonetimi nesnesi oluşturularak istemci nesneye döndürülür. Eğer zaten daha önceden kullanılan bir VeritabaniYonetimi nesnesi varsa bu nesne, GetInstance metodunun dönüş değeri olarak kullanıcıya gönderilir. Bu sayede veritabanı bağlantısının tek olması garanti altına alınmış olur.

Üniversitelerin sahip oldukları farklı otomasyon sistemlerinin verileri işlemek için kullandıkları veritabanı yönetim sistemleri farklı olabilmektedir. Dahası kullanılan bu veritabanı yönetim sistemleri, ihtiyaçlar doğrultusunda değiştirilebilecektir. Bu

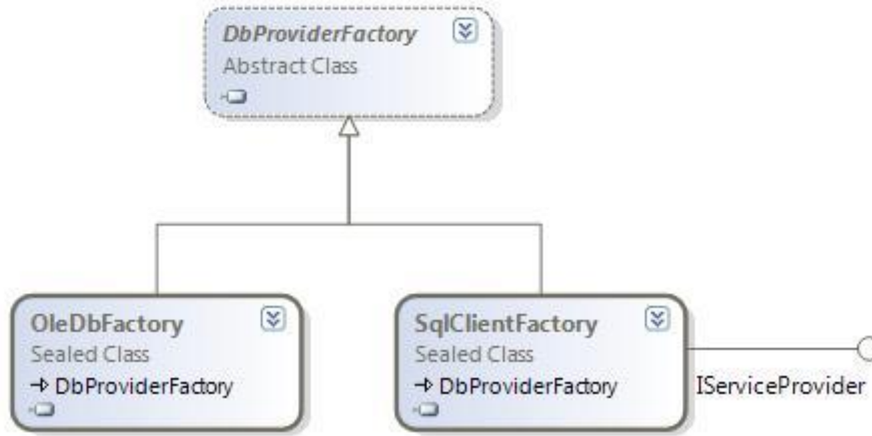
durumda EOBS sisteminin farklı veritabanı altyapılarına uygun olarak güncellenmesi, veritabanı işlemlerinin yapıldığı sınıflardaki kodların tekrar yazılması durumu ortaya çıkabilir. Böyle bir durum sistemin taşınabilir olmasının önünde büyük bir engeldir. Geliştirilen EOBS uygulamasının, bir veritabanı yönetim sistemine uygun olarak geliştirilmesi, bir başka veritabanı yönetim sistemini kullanan üniversitede kullanılabilmesi için tekrar yazılmasını gerektirecektir. Bu da sistemin tercih edilirliğini azaltacaktır. Öyleyse sistemin kullanılabilecek tüm veritabanı yönetim sistemlerine destek verebilecek yapıda tasarlanması gerekmektedir. Hatta ileride piyasaya çıkabilecek herhangi bir veritabanı yönetim sistemi de sisteme tanıtılabilmelidir. Bu duruma çözüm olarak .NET Framework'ün bize sunduğu IDbConnection ve IDbCommand arayüzlerini kullanan sınıflardan yararlanabiliriz. Bu sınıfların mimarileri oluşturulurken Soyut Fabrika tasarım kalıbı kullanılmıştır.

Soyut fabrika tasarım kalıbında ürünler ve bu ürünlerin üretilmesinden sorumlu fabrikalar vardır. .NET Framework içerisinde farklı veritabanlarında işlem yapmaya imkan sağlamak için soyut fabrika tasarım kalıbına göre tasarlanmış sınıflar mevcuttur. Bu tasarıma göre Connection, Command, DataAdapter gibi nesneler ürün sınıflarını temsil ederken DbProviderFactory soyut sınıfından türeyen OracleFactory, SqlClientFactory gibi sınıflar ürünlerin üretilmesinden sorumlu fabrika sınıflarını temsil etmektedir. Kullanıcının yapması gereken sadece sistemin kullanacağı veritabanı altyapısına göre uygun fabrika sınıfını seçmektir.



Şekil 4.9. .NET Framework İçerisinde Bulunan Connection Sınıflarının UML Şeması

Şekil 4.9’ da, örnek ürün sınıfı olarak DbConnection sınıf Şeması gösterilmektedir. Seçilecek fabrika sınıfına göre buradaki ürünlerden, yani bağlantı nesnelerinden, uygun olanı üretilerek kullanıcıya döndürülecektir.



Şekil 4.10. .NET Framework İçerisinde Bulunan Factory Sınıflarının UML Şeması

Şekil 4.10’da ilgili veritabanı yönetim sistemine ait ürünlerin üretilmesini gerçekleştiren DbProviderFactory sınıfının UML Şeması görülmektedir. Burada seçilen fabrika nesnesi, veritabanına uygun olan nesneleri oluşturarak kullanıcıya döndürmektedir. Örneğin SqlServer veritabanına bağlanmak için, SqlClientFactory sınıfını kullanarak nesneleri üretip SqlServer veritabanında ihtiyaç duyduğumuz tüm veritabanı işlemlerini gerçekleştirebiliriz. Daha sonra veritabanını değiştirmek istersek bu durumda yapmamız gereken tek şey fabrika sınıfının değiştirilmesi olacaktır. Bu da sadece bir konfigürasyon dosyasında tutulacak veritabanı yönetim sistemi bilgisinin değiştirilmesidir.

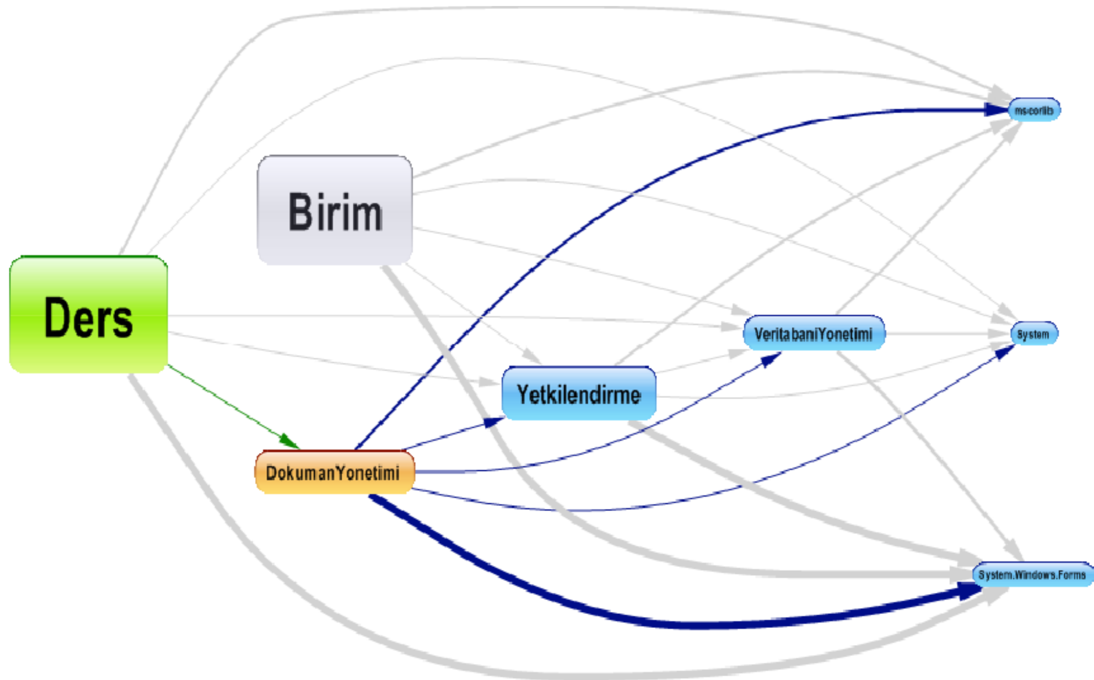
VeritabanıYonetimi sınıfında ayrıca, geliştiricileri rutin ve karmaşık veritabanı işlemlerinden kurtarabilmek ve daha basit yapılarla veritabanı bağlantılarını ve sorgularını gerçekleştirebilmek amacıyla Önyüz tasarım kalıbı kullanılmıştır. VeritabaniYonetimi sınıfı aynı zamanda bir önyüz sınıfıdır. Veritabanı bağlantısını kurmak, sonlandırmak, veritabanındaki kayıtları sorgulamak gibi işlemlerde bu sınıfa ait OpenDbConnection, CloseDbConnection, ExecuteCommand gibi metotları kullanılmak suretiyle rutin ve karmaşık işleri tekrar tekrar yapmaya gerek kalmamaktadır.

4.6. Kullanılan Tasarım Kalıplarınını EOBS Çerçeve Yazılımına Etkileri

Bu bölümde EOBS çerçeve yazılımında kullanılan tasarım kalıplarının tekrar kullanılabilirlik, modülerlik ve genişletilebilirlik, açısından sistemi nasıl etkilediği anlatılacaktır.

Nesne tabanlı bir uygulamanın sağlamlığını ve kalitesini ölçebilmek için çeşitli metrikler kullanılabilir. Bu çalışmada Chidamber ve Kemerer ölçüt kümesinden faydalanarak nesneler arası bağımlılık (CBO) [32] parametresine göre uygulama test edilmiştir.

Şekil 4.11' de görüldüğü gibi Ders ve Birim modülleri arasında herhangi bir bağımlılık söz konusu değildir. Bu da Ders ve Birim modüllerinin üniversitenin farklı uygulamalarında bağımsız olarak kullanılabileceği anlamına gelmektedir.



Şekil 4.11. Modüller Arası İlişkiler

Tablo 4.1'de modüller arası bağımlılık değerleri (Afferent Coupling) görülmektedir. Tablodan da anlaşılacağı üzere, VeritabaniYonetimi modülü, veritabanıyla ilgili işlemlere ihtiyaç duyan tüm sınıflarda sıkça kullanıldığından dolayı bağımlılığı

yüksektir. Fakat diğer modüller arasında sıkı bağlı bir yapı söz konusu değildir. Bu da sistemin genişlemeye açık, değişikliklere kapalı olmasına katkı sağlamıştır.

Tablo 4.1. Modüller Arası Bağımlılıklar

Assemblies	# lines of code	# IL instruction	# Types	# Abstract Types	# lines of comment	% Comment	% Coverage	Afferent Coupling	Efferent Coupling	Relational Cohesion	Instability	Abstractness	Distance
VeritabaniYonetimi v1.0.0.0	21	115	5	0	63	75	-	4	24	0.4	0.86	0	0.1
Yetkilendirme v1.0.0.0	27	142	4	0	66	70	-	3	31	0.5	0.91	0	0.06
Birim v1.0.0.0	28	144	4	0	66	70	-	0	32	0.5	1	0	0
DokumanYonetimi v1.0.0.0	28	144	4	0	66	70	-	1	32	0.5	0.97	0	0.02
Ders v1.0.0.0	29	146	4	0	66	69	-	0	33	0.5	1	0	0

Uygulamada, sınıfların birbirlerine olan bağımlılıkları olabildiğince azaltılmaya çalışılmıştır. Buradaki birinci amaç, her modülün sadece kendi görevlerini yapabilmesi ve modüllerin birbirlerinden bağımsız olarak kullanılabilmesini sağlamaktır. Ancak bahsedilen modüler yapıyı sağlamak için tüm modülleri birbirinden ayırmaya çalışmanın olumsuz etkileri de olabilir. Burada önemli olan bağımlılığı azaltırken diğer ölçüt parametrelerine olumsuz yansıyacak özellikleri de hesaba katmak gerektiğidir.

Uygulamada kullanılan yapıların bazıları üniversitelerde kullanılan diğer otomasyon sistemlerinde de kullanılabilir. Örneğin üniversitedeki birimlerin hiyerarşik yapısını temsil eden yapılar diğer otomasyon sistemlerinde de ihtiyaç olabilecek yapılardır. Burada kodların tekrar kullanılabilirliği ölçütü karşımıza çıkmaktadır. Bileşik kalıbının kullanılarak hiyerarşik yapıların oluşturulması sayesinde tekrar kullanılabilir yapılar oluşturulabilmiştir. Bu durum, örneğin birim/program modülündeki sınıfların diğer modüllerdeki sınıflarla sıkı bağımlı olmamasının bir sonucudur.

Uygulamanın kullanılabilir ve geliştirilebilir olmasındaki ölçütlerden biri de basitlik ölçütüdür. Basitliğin sağlanması uygulama üzerinden geliştirmeler yapan kişilerin nesne oluşturma ve nesneler arasındaki ilişkileri tanımlama işlemlerinden

olabildiğince soyutlanması sayesinde olacaktır. Uygulamada kullanılan Önyüz tasarım kalıbı sayesinde sınıfları kullanacak uygulama geliştiricilerinin, karmaşık sınıfların oluşturma süreciyle ilgili detaylarla ilgilenmelerine gerek kalmayacaktır.

BÖLÜM 5. SONUÇLAR VE ÖNERİLER

Bu çalışmada üniversitelerin eğitim öğretim süreçlerini düzenlemeye yönelik yazılım ihtiyaçlarını karşılayacak bir eğitim öğretim bilgi sisteminin geliştirilmesinde kullanılacak tasarım kalıplarının, sistemin tekrar kullanılabilmesi, taşınabilmesi ve genişletilebilmesine sağladığı katkıların önemi üzerinde durulmuştur. Genel bir eğitim öğretim bilgi sistemi çerçeve yazılımının sahip olması gereken standartlar incelenmiş, üniversitelerin değişen ihtiyaçları doğrultusunda yazılımın genişlemeye imkan veren bir yapıda tasarlanması, tasarım kalıpları kullanılarak sağlanmıştır.

Geliştirilen uygulamanın, yazılım ölçütlerini değerlendiren uygulama programları aracılığıyla test edilmesi sonucunda, ortaya konan çerçevenin tasarım kalıplarının kullanılması sonucu Açık Kapalı Prensibine [24] uygun olduğu görülmüştür. Geliştirilen çerçevede sınıflar arası bağımlılığın az olması sebebiyle uygulamaya eklenecek yeni yapıların, var olan yapılarda bir değişiklik yapma ihtiyacını ortadan kaldırdığı izlenmiştir. Bu yaklaşım kullanılarak uygulama geliştirme ve bakım maliyetlerinin azaltılacağı gösterilmiştir.

Geliştirilen yazılım çerçevesi genişletilebilirlik, tekrar kullanılabilirlik, modülerlik gibi uygulama geliştiricilere kolaylık sağlayacak ölçütler açısından değerlendirilmiştir. Ancak, uygulamanın, kullanıcı açısından önemli olan performans, hızlı cevap verme gibi başka özelliklerinin de incelenmesi gerekmektedir. Yukarıda bahsedilen ölçütleri sağlayabilmek için kullanılan tasarım kalıplarında asıl somut sınıfların haricinde, daha fazla soyut sınıf ve ara yüz kullanmak gerekmiştir. Bu yaklaşım uygulamanın derlenme ve çalışma süresini uzatabilir. Bu da performans açısından olumsuz sonuçlar doğurabilir. Bu performans kayıpları günümüzde kullanıcıların sahip oldukları bilgisayar sistemlerinin kapasiteleri düşünüldüğünde hesaba katılmayabilir. Ancak, uygulamaların çalışma

performansının önemli olduğu durumlarda tasarım kalıpları kullanılmasının, sistem üzerindeki olumsuz etkileri incelenmelidir.

Sonuç olarak, herhangi bir alanda geliştirilecek uygulamalara temel sağlayacak çerçeve yazılımlarının, genişletilebilir ve bakımı yapılabilir ölçütlere uygun olarak geliştirilmesi gerekir. Bu süreçte, tasarım kalıplarının etkin bir şekilde kullanılması, istenilen özelliklere sahip uygulamalar geliştirme amacına önemli katkılar sağlayabilir.

KAYNAKLAR

[1] ÖNDEROĞLU,S., AB Eğitim ve Gençlik Programları Kurum Koordinatörü Bologna Rehberi, 2010

[2] EVİRGEN, H., VARAN, M., CANAY, Ö., DARGA, S., Bologna Süreci Kapsamında Eğitim Öğretim Bilgi Sistemi, International Educational Technology Conference (IETC2011), İstanbul Üniversitesi, 2011

[3] MORISIO, M., ROMANO, D., MOISO, C., Framework Based Software Development: Invesigating the Learning Effect, Sixth International Software Metrics Symposium Proceedings 1999, pp 260-268.

[4] VLADIMIR, L., SYLVIA, I., Towards Developmet and Use od In-house Component Framework: Results and Expectations, Proceedings of 31st EUROMICRO 45 Conference on Software Engineering and Advanced Applications (EUROMICRO- EAA'05) 2005, pp 12-17.

[5] SRINIVASAN, S., Design Patterns in Object-Oriented Frameworks, Computer Issue 1999, 32(2), pp 24-32.

[6] KRAJNC, A., HERICKO, M., Classification of Object-Oriented Frameworks EUROCON 2003, Computer as a Tool. The IEEE Region 8(2), pp 57-61.

[7] HORASAN, R., Tasarım Kalıpları Kullanarak Çerçeve Geliştirme, Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 2007

[8] GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J., Elements of Reusable Object-Oriented Software, Addison-Wesley, 1998.

[9] DOUGLASS, B.P., Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems; Addison-Wesley, 332p, 2002.

[10] ALEXANDER, C., ISHIKAWA, S., SILVERSTEIN, M., JACOBSON, M., FIKSDAHL, I., ANGEL, S., A Pattern Language: Towns, Buildings, Construction. New York: Oxford University Press, 1977.

[11] ALEXANDER, C., The timeless way of building. New York: Oxford University Press, 1979.

[12] BECK, K., CUNNIGHAM, W., Using Patterns languages for Object-Oriented Programs.OOPSLA Workshop on Specification and Design for Object-Oriented Programming, 1987.

[13] COPLIEN, J.O., Advanced C++ Programming Styles and Idioms, 1991.

[14] ALEXANDER, C., BLACK, G., MIYOKO, T., The Mary Rose Museum. New York: Oxford University Press, 1995.

[15] BECK, K., JOHNSON, R., Patterns Generate Architectures. ECOOP '94, LNCS 821, Conference Pro-ceedings pp. 139-149. Berlin, Heidelberg: Springer-Verlag, 1994.

[16] SCHMIDT, D. C., Experience Using Design Patterns to Develop Reusable Object-Oriented Communication Software, Communications of the ACM Special Issue on Object-Oriented Experiences, 38(10), 1995.

[17] BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P., STAL, M., Pattern-Oriented Software Archi-tecture : A System of Patterns. Wiley Sons Ltd, 1996.

[18] EDİNSEL, K., Bologna Sürecinin Türkiye’de Uygulanması: Bologna Uzmanları Ulusal Takımı Projesi 2007-2008 Sonuç Raporu, 2008.

[19] OKTİK, Ş., Yükseköğretim Yeterlilikler Çerçevesi, T.C. Yükseköğretim Kurulu Yükseköğretim Yeterlilikler Komisyonu Raporu, 2007.

[20] MARTIN, R.C., Designing Object Oriented Applications using UML, 2d. ed., Prentice Hall, 1999.

[21] ITU/BIDB, İstanbul Teknik Üniversitesi Bilgi İşlem Daire Başkanlığı Desstek Sayfası, http://www.bidb.itu.edu.tr/?d=369, 2011

[22] MEYER, B. Object Oriented Software Construction, Prentice Hall, 1988.

[23] Liskov B.H., Zilles, S.N., Programming with Abstract Data Types, Computation Structures Group, Memo No 99, MIT, Project MAC, Cambridge Mass, 1974.

[24] MARTIN, R.C., Agile Software Development: Principles, Patterns, and Practices, 2002.

[25] SHALLOWAY, A., TROTT, J., Design Patterns Explained: a New Perspective on Object-Oriented Design, Boston, MA:Addison-Wesley, Inc, 2002.

[26] YILDIRIM, K.S., Gömülü Sistemler İçin Tasarım Desenleri Kullanarak Nesneye Yönelik, Gerçek Zamanlı Bir Mikroçekirdek Tasarımı, p 18, Yüksek Lisans Tezi, Ege Üniversitesi, İzmir, 2006

[27] DOUGLASS, B.P., Real-Time Design Patterns: Robust Scalable Architecture For Real Time Systems, Addison Wesley, p 332, 2002.

[28] http://tasarimdesenleri.com/bridge.html, 2011

[29] http://thisblog.runsfreesoftware/?q=Facade+Design+Pattern+UML+Class+ Diyagram, 2011

[30] ERGÜN, K., Program Çıktıları, Ders Öğrenme Çıktıları ve Program Tanıtım Kılavuzu, Ağrı İbrahim Çeçen Üniversitesi, Haziran 2010

[31] CHEN, X., Developing Application Frameworks in .NET, Apress, USA, 2004.

[32] CHIDAMBER, S., KEMERER C., A Metrics Suite For Object Oriented Design, IEEETrasactions on Software Engineering, 20:476-493, 1994.

ÖZGEÇMİŞ

Serkan DARGA, 29.08.1980'de İstanbul' da doğdu. İlk, orta ve lise eğitimini Sakarya'da tamamladı. 2000 yılında girdiği Sakarya Üniversitesi Bilgisayar Mühendisliği bölümünden 2005 yılında mezun oldu. Aynı yıl Sakarya Üniversitesi Bilgi İşlem Dairesi Başkanlığı bünyesinde uzman olarak göreve başladı. 2007 yılından bu yana Sakarya Üniversitesi Bilgisayar Araştırma Uygulama Merkezi'nde yazılım projelerinde görev almaktadır.