YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

İLGİYE YÖNELİK YAKLAŞIMLA YAZILIM GELİŞTİRME

Bilgisayar Müh. Oytun KURTAR

FBE Bilgisayar Mühendisliği Anabilim Dalı Bilgisayar Mühendisliği Programında Hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı : Prof. Dr. Oya KALIPSIZ

İSTANBUL, 2007

İÇİNDEKİLER

KISALTMA LİSTESİ

AOP	Aspect Oriented Programming
AOSD	Aspect Oriented Software Development
DAO	Data Access Object
IDE	Integrated Developer Environment
FTS	Fatura Takip Sistemi
GOF	Gang of Four
İYP	İlgiye Yönelik Programlama
İYYG	İlgiye Yönelik Yazılım Geliştirme
NGO	New Generation Orbis
NYP	Nesneye Yönelik Programlama
UML	Unified Modelling Language

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖNSÖZ

Yüksek Lisans öğrenimim ve tez sürecim boyunca yardımlarını ve değerli katkılarını benden esirgemeyen Prof. Dr. Oya Kalıpsız' a teşekkürü bir borç bilirim.

Yaşamım boyunca her konuda benden sevgi, destek ve güvenlerini eksik etmeyen aileme teşekkür ederim.

Tez geliştirme sürecinde ve günlük yaşamda yardımlarını ve desteklerini eksik etmeyen tüm arkadaşlarıma teşekkür ederim.

Temmuz 2007

Oytun Kurtar

ÖZET

Nesneye Yönelik Programlama (NYP), yazılım mühendisliği için önemli bir dönüm noktasıdır. Çünkü NYP, günlük yaşantımızda karşılaştığımız problem çözme mantığına benzer olarak, problemleri nesne modeli olarak ele alabilmemizi sağlayan bir programlama mekanizması sunar. Fakat NYP teknikleri, yazılım sistemlerindeki artan ihtiyaçlar ve karmaşıklaşan problemler karşısında bazı gereksinimleri karşılamakta yetersiz kalmaya başlamıştır.

Bu çalışmada NYP tekniklerinin eksik kaldıkları yerlerde kullanılabilecek yeni teknikleri içeren bir programlama yaklaşımı üzerinde durulacaktır. Bu yeni yaklaşım İlgiye Yönelik Programlama (İYP) adı ile anılmaktadır. İYP ile birlikte yazılım dünyasına yeni terimler eklenmiştir. En önemli kullanım alanı sistem işlevleri ile aynı anda çalışması gereken ve sürekli tekrarlanmak durumunda olan kod bloklarının ayrıştırılması ve bunların özel yapılar ile uygulamaya eklenmesidir. Sistem işlevleri ile çakışan işlevlere "enine kesen ilgi", bunların ayrıştırılmasından oluşan yeni yapılara ise "ilgi" adı verilmektedir.

Çalışma süresince, bazı tasarımların neden gerçek kod içerisinde uygulanmasının zor olduğu ve karmaşıklığa neden olduğu incelenmiş ve bunu gidermek için kullanılan İlgiye Yönelik Programlama (İYP) teknikleri üzerinde durulmuştur. Bu programlama yaklaşımında yeni bir sınıf tipi olan ilgiler kullanılarak karmaşıklıktan uzak, yeniden kullanılabilir ve bakımı kolay programlar geliştirilmesi hedeflenmektedir. Yine, çalışma süresince İYP teknikleri ve getirdikleri yenilikler üzerinde durulmuş, yazılım dünyasında getirdikleri kurallar birçok problemin çözümünde kullanılan tasarım kalıplarının İYP kullanım alanlarına değinilmiş, performansları açısından İYP araçlarının karşılaştırılmaları yapılmıştır. Geliştirilen örnek bir uygulamada çalışma süresince anlatılan tasarım ve geliştirme adımları izlenmiş ve sonuçlar belirtilmiştir.

Anahtar kelimeler: İlgiye Yönelik Program Geliştirme (İYPG), Enine Kesen İlgiler, UML, GOF Tasarım Kalıpları, İYP araç ve çatıları

ABSTRACT

Object Oriented Programming (OOP) is a very important milestone in Software Engineering, since it provides a programming mechanism to handle problems as an object model , similar to our daily life problem solving mechanism. But in some cases, OOP techniques remain insufficient due to increasing needs and the complexity of the software systems.

In this work, insufficient parts of the OOP techniques will be discussed and the approach called Aspect Oriented Programming (AOP) will be introduced. There are many terms specific to AOP introduced in software world. Fundamental usage area of AOP programming technique is preventing the code tangling because of the system based concerns overlapping the main functionality of the system and the unnecessary repeating code blocks. The term used to describe the system based concerns overlapping the main functionality of the system is known as a "crosscutting concern". Modularization of the crosscutting concerns requires new special classes to be written to add new functionality to the system. These special classes are also called aspects.

During this work, reasons of difficulty of applying some design types in actual code and reasons of the results in complexity is examined, and focused on the AOP techniques to overcome these problems. This programming approach provides a system having less complexity, reusable parts and easy to maintain, using a specific type also the fundamental of AOP called aspect. Design patterns which provide solutions to many individiual software problems and different AOP approaches were examined and compared with respect to their performance criterias. The design and development steps are followed with a tutorial and results were discussed.

Keywords: Aspect Oriented Software Development (AOSD), Crosscutting Concerns, UML, GOF Design Patterns, different AOP tools and frameworks.

1. GİRİŞ

İhtiyaçları sürekli artan ve karmaşıklaşan iş dünyasındaki birçok firma, bu gereksinimlerini yazılım ürünleri ile karşılamayı talep etmekte, süreçlerini bu uygulamalar ile takip ederek verimliliklerini arttırmayı hedeflemektedir. Bu nedenle geliştirilen yazılımların, sürekli artan ihtiyaçları karşılayabilmeleri gerekmekte ve bunların, yeni eklenebilecek işlevlere açık olmaları beklenmektedir. Bu sebeplerden dolayı kolay adapte edilebilir, düşük maliyetli ve değişiklik yönetimi kolay yapılabilen uygulamaların geliştirilmesine ihtiyaç duyulmaktadır.

Günümüzde geliştirilen uygulamaların büyük bir kısmı, özellikle kurumsal uygulamalar, tek bir dosya içerisine yazılmış tek bir modülden oluşmuş yapıda değildirler. Bunlar, sistem gereksinimlerini karşılamak için oluşturulmuş modül koleksiyonlarının bir arada çalışmasından oluşmaktadırlar. Teorik olarak yazılımcılar bazı soyut fonksiyonları içeren modüller oluşturabilirler. Böylece bir güvenlik modülü veya bir kayıt olma modülü gibi ortak modüller için ortak kütüphaneler oluşturulabilir ve benzer uygulamalarda kullanılabilir.

Nesneye Yönelik araç ve dillerin temeli modülerlik olmasına rağmen bazı uygulamalarda modüleritenin yeterince sağlanamadığı görülmektedir. Program modülleri karmaşık hedefler içeren alt modüllere ayrıştırılmaktadır. Aynı hedefleri içeren kodların modüller içerisinde tekrarlanması ise kod karmaşasına yol açmakta ve uygulamaların yönetilebilirliğinin azalmasına yol açmaktadır (Örneğin uygulama işlevlerinden sonra günlük tutulması gibi). Bu durumda günlük tutma ile ilgili kodlar modüller içerisine dağıtılmalıdır. Ortak kodlardan herhangi birinde yapılacak olan bir değişiklik sadece o modülü etkileyeceğinden aynı değişikliğin bütün modüllere uygulanması gerekmektedir. Bu, kod karmaşasına yol açmaktadır. Böylece yazılan uygulamaların hata ayıklaması, yeniden yapılandırılması, dokümante edilmesi ve desteğinin verilmesi zorlaşmaktadır. İlgiye Yönelik Programlama'nın (İYP) hedefi bu ve benzeri yazılım geliştirme problemlerine çözüm getirmektir. Bu yeni programlama yaklaşımı, dağıtık işlevselliği merkezileştiren modüller olan ilgilerin oluşturulması üzerine kurulmuştur (Gradecki ve Lesiecki, 2003).

İYP teorisinin pratiğe dönüştürülmesi Java dünyasında ilk olarak AspectJ programlama dili ile gerçekleşmiştir. AspectJ, Java diline uzantı olarak geliştirilmiş açık kaynak kodlu ve Java dili ile tamamen uyumlu bir programlama dilidir. AspectJ ile geliştirilmiş ilgiler, Java sınıfları ile birlikte çalışarak geniş kapsamlı uygulamaları meydana getirirler (Gradecki ve Lesiecki, 2003).

İYP yaklaşımının kullanılması ile daha az karmaşık ve daha kısa kod yazılması, uygulamanın gelişim ve bakım sürecinin yönetilebilir olması, hata ayıklama ve yeniden yapılandırma işlemlerinin kolaylaştırılması, yeniden kullanılabilir yazılım bileşenlerinin ve kütüphanelerinin oluşturulması hedeflenmektedir.

Yazılım geliştirmenin yaygınlaşmaya başlamasıyla, gerçek dünya problemlerinin modellenebilmesi amacıyla daha gelişmiş ve etkin teknikler üzerinde çalışılmaya başlandı. İlk yıllarda genel yöntem, tüm işlevselliğin küçük fonksiyonlara ayrılıp yazılmasıydı. Bu uzun kod satırları ile sonuçlanmaktaydı. Bu yöntem çok fazla uygulandı fakat büyük projelerde yönetiminin çok zor olmasından dolayı çok etkin bir yöntem olmadığı anlaşıldı. Bu sebeplerden dolayı yeni arayışlara ihtiyaç duyuldu. Nesneye Yönelik Programlama yaklaşımı ile sistem nesneler topluluğu olarak ele alınmaya başlamıştır. Bunların özel (private) nesneler olarak tanımlanıp yazılan metodlarla erişim hakları kontrol edilebilmekteydi. Nesneye Yönelik Programlama mantığı teoride sistemin nesnelere bölünerek modüleritenin arttırılması esasına dayanır. Fakat sisteme eklenecek dağıtık bir fonksiyonelitede sistem kodlarının güncellenmesi gerekmektedir. Bu sebepten dolayı da kod karmaşası kaçınılmaz hale gelir. Problemin nesnelere bölünmesi işlevsellik açısından büyük bir gelişimdir fakat bu ayrıştırmanın nesneler içerisinde de yapılabilmesi gerekmektedir. İlgiye Yönelik Programlama (İYP) daha az karmaşık ve gereksiz işlevsellik içermeyen iyi özetlenmiş nesneler oluşturulmasına olanak sunan bir yöntemdir. Bu özelliği ile İYP, NYP'nin eksik kalan özelliklerinin yerini doldurmaktadır (Irwin vd., 1997).

Çalışmanın birinci bölümdeki girişten sonra, ikinci bölümünde İlgiye Yönelik Programlama (İYP)'ye giriş ve programlama dillerinin evrimi ile İYP'nin gerekliliği incelenecek, üçüncü bölümünde ise İYP temel kavramları üzerinde durulacaktır. Dördüncü bölümde ilgilerin UML kullanılarak tasarımları ve geliştirmeleri üzerinde durulacaktır. Beşinci bölüm, çeşitli İYP araçlarının İYP yaklaşımlarını uygulama biçimlerini ve ilgili kriterlerle karşılaştırılmalarını içermektedir. Altıncı bölümde NYP uygulamalarında sıkça kullanılan GOF tasarım kalıplarının İYP yaklaşımı ile uygulanmaları incelenmiştir. Yedinci bölümde, çalışma boyunca anlatılan İYP yaklaşımının uygulanması bir örnek ile gerçeklenecek, sekizinci bölümde geliştirilmiş uygulama üzerinde İYP tekniklerinin özellikleri tartışılacaktır. Son bölümde ise genel olarak çalışma boyunca değinilen İYP tekniğinin getirdiği avantajlar ve dezavantajlar üzerinde durulacak ve çalışma sonlandırılacaktır.

2. İLGİYE YÖNELİK PROGRAMLAMA

Bu bölümde, uygulamalar geliştirilirken Nesneye Yönelik Programlama (NYP) yaklaşımında eksik kalan özelliklerin tamamlanması ve etkin, kolay adapte edilebilir, bakımı kolay yapılabilen uygulamalar geliştirilebilmesine imkan veren yeni teknikler sunan bir teknoloji olan İlgiye Yönelik Programlama (İYP) üzerinde durulacaktır. Bölüm içerisinde öncelikle yazılım dillerinin evriminden bahsedilecek, karşılaşılan problemlere değinilecek, daha sonra da İYP'nin karşılaşılan problemlere getirdiği çözümler üzerinde durulacak ve gerekliliğinden bahsedilecektir.

2.1 Yazılım Programlama Dilleri Evrimi

Yazılım projelerinde, dönemsel ihtiyaçlara ve projelerin büyüklüklerine göre programlama tekniklerinde bazı evrimsel değişiklikler meydana gelmiştir. Programlama dillerinin evrimi bakımından 4 sınıftan bahsedilebilir [1].

Makine dilleri (1.kuşak): Bilgisayar biliminin ilk yıllarında makine düzeyi kodlama ile programlar yazılmaktaydı. Burada programcılar yazmakta oldukları programın gereksinimlerinden çok makine dili komut kümeleri üzerinde düşünmek zorunda kalıyorlardı. Çünkü bu diller sayılardan (1 veya 0) oluşan, sadece belirli bir makinenin anlayabileceği , hata yapmaya çok açık ve kodlaması zor olan dillerdir.

Assembly dilleri (2.kuşak): Makine dilinden sonra Assembly programlama dönemi başladı. Bu dönemde, makine dilindeki 1 ve 0'lardan oluşan komutlara karşılık gelen akılda kalıcı ADD, LOAD gibi İngilizce komutlardan oluşan alt seviye bir programlama dili kullanılmaya başlandı. Assembly dili ile yazılmış bir program, bir çevirici (translator) ile makine diline çevrilerek çalıştırılır. Basit komutlardan oluşmasına karşın bir işlemin yapılması fazla kod yazımını gerektirir.

Yüksek seviyeli diller (3.kuşak) - Yapısal Diller - Problemler güçleştikçe makine ve assembly dili komutlarının yetersiz kalmaya başlamasıyla kullanılan makinenin dilinden soyutlanılmasına imkan veren daha üst düzey diller kullanılmaya başlandı. Böylece yapısal programlama temelleri atıldı. Bu yaklaşımla birlikte, ele alınan problemlerin yordamlar düzeyinde ayrıştırılmasına imkan sağlanmış oldu. Bu programlama yaklaşımı ile tek bir dosya içerisinde problemler çeşitli işlevleri yerine getirebilen yordamlarla daha basit ve okunabilir hale gelmiştir. Çünkü İngilizceye benzer, aritmetik işlemlerde kullandığımız işaretleri kullanan bu dillerle, daha hızlı ve kolay program yazılabilmektedir. Bu programlama yaklaşımı ile ilgili birçok dil geliştirilmiştir. C, Pascal bu gruptadır.

Çok yüksek seviyeli diller (4.kuşak): Karmaşıklaşan problemler ve artan ihtiyaçlar farklı çözüm arayışlarını da beraberinde getirdi. Nesneye Yönelik Programlama ile bir sisteme birbiri ile birlikte çalışan nesneler kümesi şeklinde bakabilme olanağı sunulmasıyla gerçek dünya problemleri, günlük hayatımızdaki gibi ele alınıp çözülebilir hale geldi. Gerçekleştirim detayları arayüzler yardımıyla gizlendi. Kalıtım ile bir sınıftan onun özelliklerini birebir içeren alt sınıflar türetildi ve çok şekillilik ile ilişkili kavramlar için ortak bir davranış ve arayüz sağlandı.

Yazılım geliştirme yöntemlerindeki bu evrimler ile birlikte, problemlerin çözümünde birçok yeni yöntem ele alınmıştır. Her yeni yöntemle gelen çözümler, problemlerin ve beklentilerin daha da artmasına sebep olmuştur. Aynı deyiş bunun tam tersi için de geçerlidir. Karmaşıklaşan sistem gereksinimleri yeni yöntem arayışlarını da beraberinde getirmiştir.

Günümüzde çoğu projenin gerçekleştirilmesi için Nesneye Yönelik Programlama yöntemi tercih edilmektedir. Fakat büyük, karmaşık ve birçok modülden oluşan sistemlerde Nesneye Yönelik Programlama çözümleri de yetersiz kalmaktadır. Bu yöntemin eksikliklerini tamamlamak ve sistemi nesneler dışında işlevsel olarak da ayrıştırmaya yönelik bir çözüm sunan İlgiye Yönelik Programlama, yeni nesil programlama metodolojisi olmaya adaydır. Günümüzdeki yazılım projeleri birçok işlevsel modül ve bu modüller üzerine dağılmış "enine kesen ilgi"lerden oluşmaktadır. İlgiye Yönelik Programlama bu modüller ve ilgilerin birbirlerinden ayrıştırılması ve modülerliğin arttırılması konusunda çözümler sunan bir programlama yöntemi olarak geliştirilmiştir.

2.2 İlgi Nedir?

İlgi, özel bir amaç veya kavram anlamına gelmektedir. Teknik terim olarak ifade edilirse; ilgi bir sistemin kendinden beklenen işlevselliği ve bu işlevsellik yanında çalışması gereken sistemsel yordamlardır. İlgiler, yazılım geliştiriciler tarafından ele alınması gereken işlevsel veya işlevsel olmayan kalite hedefli sistem bileşenleridir [2].

Teknoloji terimiyle bir sistem:

- Çekirdek-Düzeyi İlgiler
- Sistem-Düzeyi İlgiler'den oluşmaktadır

Örnek olarak verilirse, bir kredi kartı sisteminde:
Çekirdek Düzeyi İlgi : Ödemeler, Alışveriş vs.
Sistem Düzeyi İlgi: Yetki Kontrolü, Güvenlik, Günlük Tutma'dır.

2.3 Enine Kesen İlgiler

Günümüzde geliştirilmekte olan tüm uygulamalar yukarıda bahsedildiği gibi 2 farklı tip gereksinim kümesinden oluşmaktadır. Bunlardan ilki sistemden beklenen işlevsellik, diğeri ise arka planda çalışması gereken işlevselliktir. Örneğin bir bankamatik uygulamasında Para Çekme, Para Yatırma, Fatura Ödeme ve Havale Yapma gibi işlemler, sistemin işlevsel özellikleridir. Yani problemin ana öğeleridir. Bunun yanında işlem saati, yatırılan veya çekilen para miktarını tutan Günlük Tutma mekanizmaları ve fonksiyonları ile Güvenlik, Yetki Kontrolü, Performans fonksiyonları, sistemin ana fonksiyonları ile birlikte çalışması gereken sistemsel işlevlerdendir.

Birçok ilgi, sistemin temel fonksiyonelitesini etkileme eğilimindedir. Yukarıda bahsedilen Bankamatik uygulaması göz önünde bulundurulursa, Para Çekme, Para Yatırma gibi işlevleri gerçekleştirmeden önce bir Kimlik Doğrulama veya Sisteme Kaydolma işleminin başarı ile gerçekleşmesi gerekmektedir. Bunlar gibi sistemin temel fonksiyonları ile bütünleşik çalışan ve bu temel fonksiyonların işleyişini etkileyen ilgilere Enine Kesen İlgiler (Crosscutting Concerns) adı verilir. Enine kesen ilgiler, sistemin diğer fonksiyonları ile birlikte çalışmak durumunda olduğundan bunların modüller içine yayılması, projenin ileri safhalarında kontrol edilebilirliğin zorlaşmasına hatta imkansız hale gelmesine neden olabilmektedir.

Enine kesen ilgiler uygulama geliştirme sürecinin tüm fazlarında da etkin rol oynarlar.

Tasarım Fazında, tasarımcıların içlerine düştükleri Alt Tasarım (Under Design) ve Üst Tasarım (Over Design) ikileminin çözümünde rol oynar. Alt Tasarım yaklaşımında gelecekte karşılaşılacak olası enine kesen ilgiler göz önünde bulundurulmadan sistem tasarımı gerçeklenir ve sistemin genişletilmesi yüksek maliyetlere yol açabilir. Diğer yandan Üst Tasarım yaklaşımında ise gelecekteki olası eklentiler desteklenir.

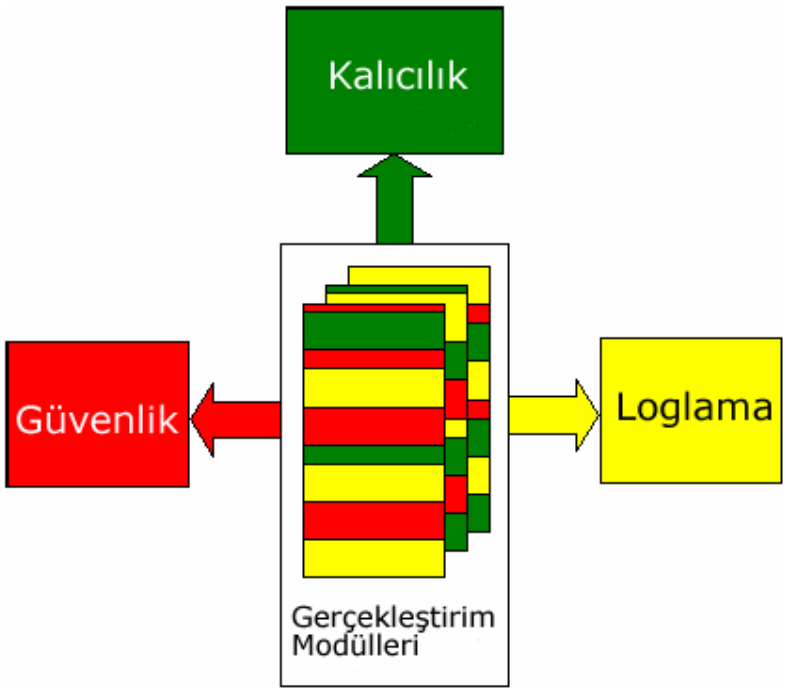
Uygulama Geliştirme Fazında enine kesen ilgiler, karmaşık ve anlaşılmaz kodlara, gereksiz performans kayıplarına, tasarlanan sistemle geliştirilen uygulama kodlarının birbirinden farklı olmasına yol açmaktadırlar.

Bakım ve Gelişim Fazlarında enine kesen ilgiler, sistemin yapısında geniş çapta değişikliklere yol açabilmektedirler (Bodkin ve Laddad, 2004).

2.4 Sistemin İlgilere Ayrıştırılması

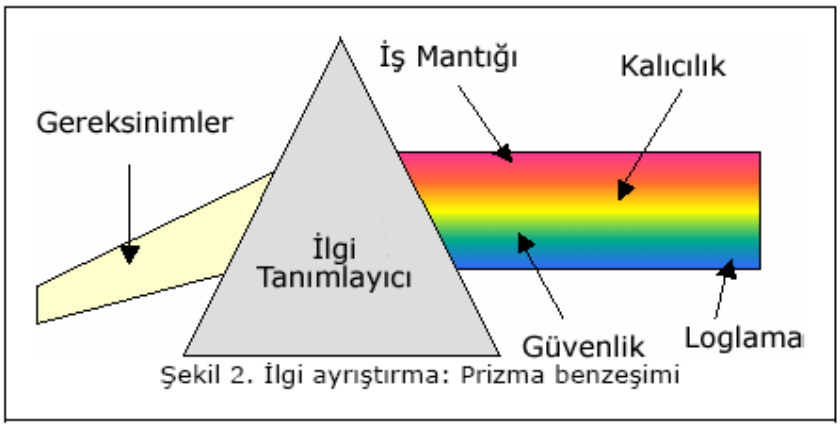
Günümüzde, projelerin artan karmaşıklığı ve fonksiyonelitesi yeni çözümlerin arayışını da beraberinde getirmiştir. Sistemin ana işlevlerinin yanında çalışması gereken birçok sistem ilgisi bir arada uygun bir şekilde çalışabilmeli, gerektiğinde modifiye edilebilmeli ve sistemin

bakımının kolay olması sağlanabilmelidir. Bir sistem, iş mantığının yanı sıra performans, güvenlik, günlük tutma, hata ayıklama, veri bütünlüğü ve kalıcılığı gibi birçok ilginin entegre şekilde çalışmasından meydana gelir. Şekil 2.1'de bazı enine kesen ilgilerin, gerçekleştirilmiş sistem modülleri üzerine dağılımları görülmektedir.



Şekil 2.1 Enine kesen ilgilerin, temel modüller üzerine dağılımı [2]

Karmaşık projeler tasarım ve kodlama aşamasına geçilmeden önce çok iyi analiz edilmeli, gereksinimler tespit edilmeli ve bu gereksinimler doğrultusunda modüllere ayrıştırılmalıdır. İlgiye Yönelik Programlamada, sistemin gerçekleştirimi Şekil 2.2'de görülmektedir.



Şekil 2.2 Gereksinimlerin İYP'de ayrıştırılma süreci [2]

Şekil 2.2'de de görüldüğü gibi, sistem gereksinimleri mantıksal bir ilgi tanımlayıcı mekanizmasından geçirilerek, iş mantığı (sistemden beklenen fonksiyonalite) ile bu iş mantığını enine kesen ilgiler olmak üzere 2 gruba ayrıştırılır.

2.5 Enine Kesen İlgilerden Kaynaklanan Problemler

Bir yazılım sistemi, o sistemin asıl işlevselliğini sağlayan ilgiler ile bu işlevsellikle birlikte çalışması gereken ilgilerden (enine kesen ilgiler) oluşmaktadır. Bu ifadeye göre, sistemin esas işlevselliğini sağlayan ilgilere Çekirdek-Modül Düzeyi İlgiler, esas işlevselliği yanında çalışması gereken ilgilere ise Sistem İlgileri adı verilmektedir. Sistem İlgilerinin diğer ilgiler üzerine uygulanması sistemin kontrol edilebilirliğini zorlaştırır hatta ileri sürümlerde içinden çıkılamaz bir hale getirebilir.

2.5.1 Kod Karıştırma

Yazılımcılar, çoğunlukla sistemin tüm ilgilerini bir arada düşünerek uygulama geliştirme eğilimindedirler. Bir sistemin herhangi bir modülünün gerçekleştiriminde performans, yetki kontrolü güvenlik gibi kriterler tümleşik düşünülerek o modülün işlevselliği içinde bu gibi kontroller de yapıldığında kod karmaşası meydana gelebilir.

2.5.2 Kod Dağıtma

Enine Kesen ilgiler sistemin genel birçok modülünü etkilediğinden dolayı, modüller içerisine ayrı ayrı gömülen bu enine kesen ilgiler, yönetimi zor hale getirir. Örneğin bir günlük tutma veya yetki mekanizmasında herhangi bir değişiklik yapılması gerektiğinde, bu ilgilerin kullanıldığı tüm yerlerde aynı değişiklik yapılmalıdır. Böylece modüller içine dağılmış olan ilgilerin tespit edilmesi ve gerekli düzenlemenin bulunan yerlerde de yapılması gerekmektedir.

2.5.3 Yazılım Sistemlerine Etkileri

Enine Kesen İlgilerin, yazılım sistemlerinin gelişimindeki olumsuz etkileri aşağıda belirtilmiştir:

Zayıf izlenebilirlik: Birçok ilgiyi bir arada gerçekleştirmek, sistem ilgileri ile asıl fonksiyonelitenin birbirine karışmasına neden olur ve bu ikisi arasındaki ayrımın yapılması güçleşir.

Daha düşük üretkenlik: Birçok ilgiyi aynı anda gerçekleştirmek, geliştiricinin dikkatini ana ilgiden daha önemsiz olan ilgilere doğru kaydırır ve bu da uygulamanın üretkenliğini azaltır.

Daha az yeniden kullanılabilir kod: Bir modülde birden çok ilgi gerçekleştiriliyor ise, benzer

işlevselliklere ihtiyaç duyan diğer sistemler modülü kullanamayabilirler.

Aynı anda birden çok ilgiyi hedef almak, bu ilgilerden birinin yada birkaçının gerçekleştirilmesinin yeterince etkin yapılamamasına neden olabilir.

Sınırlı bir bakış ve kısıtlanmış kaynaklar çoğu zaman sadece mevcut ilgiye hitap eden tasarımlar üretir. Olası gereksinimlere hitap edebilmek, mevcut uygulama üzerinde yeniden çalışılmasını gerektirecektir. Ve eğer uygulama modüler değilse, bu bir çok modülün değişmesini gerektirecektir. Birçok alt sistem üzerinde değişiklik yapmak tutarsızlıklara neden olabilir. Ayrıca bu uygulama değişikliklerinin hataya sebep olup olmadığının anlaşılabilmesi için kapsamlı bir testin de yapılması gerekmektedir.

2.6 Neden İYP?

İlgiye Yönelik Programlama, sistemin kontrol edilebilirliğinin sağlanması ve enine kesen ilgilerin yarattıkları problemlerin en aza indirilmesi amacı ile sistemi çeşitli ilgilere ayrıştırma prensibini temel alır. Bu ilgilere ayrıştırma işlemi, yeni bir yazılım geliştirme yaklaşımına temel teşkil etmektedir.

İlk adımda yapılması gereken sistem modülleri üzerine serpiştirilecek olan enine kesen ilgilerin tespit edilmesi ve İlgiye Yönelik Programlama İlgileri (Aspects) şeklinde tanımlanmasıdır. Daha sonra tanımlanan bu ilgiler, sistemin ana modülleriyle birlikte çalışabilir şekilde birleştirilir ve bütünleşik sistem meydana gelir. Temel mantık, enine kesen ilgilerin bağımsız varlıklar (İYP İlgileri) şeklinde ifade edilebilirliğinin sağlanmasıdır.

Günümüzdeki kurumsal yazılım projeleri, yapıları gereği birçok modülden oluşmaktadırlar. Sistemin asıl fonksiyonelitesini oluşturan ilgiler ile bunlarla birlikte çalışan ve bu ilgileri enine kesen ilgiler birbirlerinden ayrıştırılmalıdır. İYP, bu enine kesen ilgilerin modülerize edilmesini kolaylaştıran yeni yöntemler sunmaktadır.

3. İLGİYE YÖNELİK PROGRAMLAMA TEMELLERİ

Karmaşık yazılım projelerinin geliştirilmesini ve yönetilmesini kolaylaştırabilmek için sistem gereksinimlerinin çok iyi modülerize edilmesinden sonra bunların uygun bir şekilde birleştirilmesi ve sistemden beklenenleri karşılaması gerekmektedir. İlgiye Yönelik Programlama bu probleme çözüm sunan yöntemlerden biridir.

3.1 İlgiye Yönelik Programlama Süreci

İlgiye Yönelik Programlama ile sistemin gerçeklenmesi, temel 3 fazdan meydana gelmektedir. Bunlar, İlgisel Ayrıştırma, İlgilerin Gerçekleştirimi, İlgisel Yeniden Birleştirme'dir. Bu adımlar ayrıntılı olarak Şekil 3.1'de görülmektedir.

3.1.1 İlgisel Ayrıştırma

İYP'nin temel prensiplerinden biri sistemin analizinin çok detaylı biçimde yapılıp, işlevselliğin etkin bir şekilde modülerize edilmesidir. Gereksinimleri, enine kesen ilgiler ve genel ilgiler olarak tanımlayabilmek için ayrıştırma adımıdır. Modül-düzeyi ilgiler, sistem-düzeyi enine kesen ilgilerden ayrılır. Örneğin daha önce bahsedilen Bankamatik modülünde üç ilgi tanımlanabilir. Bunlar: Banka İşlemleri, Günlük Tutma ve Yetki Kontrolüdür.

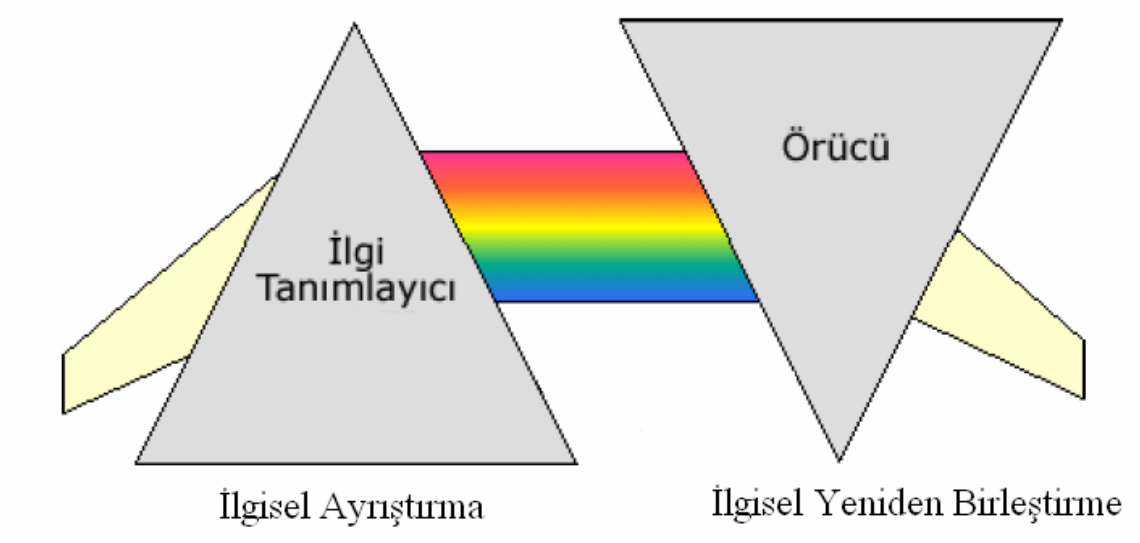
3.1.2 İlgi Gerçekleştirimi

Gereksinim aşamasında ayrıştırılmış her ilgi ayrı ayrı gerçeklenir. Enine kesen ilgilerin uygulama kodları ile birlikte yazılmaması ve ayrı sınıflar halinde gerçekleştirimi, uygulamanın işlevsel kodlarının karmaşıklaşmasını önlerken, aynı zamanda kodlarda meydana gelecek bir değişikliğin uygulamaya adapte edilmesini de kolaylaşmaktadır. İYP'yi destekleyen tüm araçlarda ilgi gerçekleştirim standartları bulunmaktadır. Bunların tek ortak yanları uygulama kodları ile birlikte çalışacak olan ilgilerin tanımlanmasına ve uygulamanın gereksinimlerinin karşılanmasına imkan vermeleridir. Enine kesen ilgilerin uygulama kodları ile birlikte çalışabilmesini sağlayan mekanizma İlgi Dokuma mekanizmasıdır. Bankamatik örneği için, banka işlemleri birimi, günlük tutma birimi ve yetki kontrolü birimi gerçekleştirimleri yapılabilir. Çünkü bu işlevler, uygulamanın diğer işlevleri ile birlikte çalışması gereken enine kesen ilgilerdir.

3.1.3 İlgisel Birleştirme

Bu adımda bir ilgi bütünleştiricisi modülerizasyon birimlerini (İYP İlgilerini) oluşturarak yeniden birleştirme kurallarını belirler. Yeniden birleştirme işlemi, örme (*weaving*) veya

birleşleştirme olarak da bilinir. Sistem ilgilerini birleştirmek için bu yeniden birleştirme kuralları kullanır. Bankamatik örneğinde, İYP ilgilerini, her bankamatik işleminin günlüğünün tutulması ve işlemlerden önce yetkilerin kontrol edilmesi olarak belirtebiliriz.



Şekil 3.1 İYP uygulama gerçekleştirim süreci [2]

3.2 İYP Temel Kavramlar

İlgiye Yönelik Programlama yaklaşımına giriş yapılabilmesi için, öncelikle bu yaklaşımla birlikte gelen bazı kavramlar üzerinde durulması gerekmektedir. Bu kavramlar, İlgiye Yönelik Programlama'nın temelini oluştururlar. Üzerinde durulması gereken kavramlar bu yaklaşımın anlaşılması için gereklidir.

3.2.1 Birleşme Noktaları (Join Points)

Bir sistemdeki Birleşme Noktaları ana modül içinde tanımlanan, bu modül içerisinde bir başka kodun (örneğin bir enine kesen ilginin) icra etmeye başlaması gereken yeri belirten iyi tanımlanmış yerler olmalıdır. Bu iyi tanımlanmış yerler, metod çağrıları (Method call), hata yakalayıcılar (Exception Handler), günlük tutma kodları vb. olabilir. Birleşme Noktalarının İYP süreçlerindeki en önemli rolü, bu noktaların uygulamaya yeni eklenecek olan işlevselliğin uygulamayı hangi noktalarda etkileyeceğini belirlemesidir. Bu nedenle gereksinimleri iyi ayrıştırılmış olan bir sistemde enine kesen ilgilerin sistem ilgileri ile çakışma noktalarının tespiti, birleşme noktalarının belirlenmesini sağlar.

Bankamatik uygulaması ele alındığında, yapılacak olan tüm işlemlerin kayıtlarının tutulması gerekmektedir. Bir işlem; örneğin para çekme işlemi yapılmadan önceki bakiyeyi kaydedecek

bir günlük tutma kodu bu işlemden önce ve sonra çalışmalıdır. Burada para çekme bir birleşme noktasıdır. Yani para çekme kodu ile ayrı bir yerde yazılmış olan günlük tutma kodunun birleştirileceğini gösteren noktadır. Bu durumda, birleşme noktaları, çok iyi tanımlanmış ilgi icra noktaları olarak tanımlanabilir.

3.2.2 İcra Noktaları (Pointcuts) :

İcra noktaları, birleşme noktaları ile belirlenmiş olan metod çağrılarının sisteme yeni eklenecek ilgiler ile uyumlu olarak çalışmasının sağlanabilmesi amacıyla geliştirilmiştir. Bu yapılar, icra edilecek olan ilgi kodunun ana modülün hangi metoduna gelindiğinde başlayacağını tanımlamaktadır. Fakat burada enine kesen ilgi kodunun, birleşme noktasında belirtilen metodun hangi safhasında çalışacağı bilgisi bulunmamaktadır. Bu nedenle enine kesen ilgilerin tanımlanmış olduğu ilgi sınıflarında, ana işlevin bulunduğu sınıfta tanımlanan birleşme noktalarının icra zamanları belirtilmelidir. Bunlara İlgiye Yönelik Programlama kavramları çerçevesinde icra noktaları (pointcuts) adı verilir.

İcra noktaları 3 çeşittir:

- *Önce (Before) :* Belirtilen enine kesen ilgi kodunun, birleşme noktası olarak belirtilen metoddan önce çalışması gerektiğini belirtir. Bu durumda, önce enine kesen ilgi kodu, daha sonra da birleşme noktası olarak belirtilen kod icra edilir.
- Esnasında (Around) : Belirtilen enine kesen ilgi kodunun, birleşme noktası olarak belirtilen metodun yerine çalışması gerektiğini belirtir. Bu durumda mevcut ana işlevin bulunduğu kod değiştirilmeden çalışması gereken işlev değiştirilmiş olur. Enine kesen ilgi kodu çalıştıktan sonra birleşme noktasındaki metodun çalışıp çalışmaması koşula bağlı olarak belirlenir.
- *Sonra (After)* : Belirtilen enine kesen ilgi kodunun, birleşme noktası olarak belirtilen metoddan sonra çalışması gerektiğini belirtir. Bu durumda ise önce birleşme noktası olarak belirtilen kod, daha sonra da enine kesen ilgi kodu icra eder.

3.2.3 İcra Edilecek Kod (Advice)

Bir birleşme noktasına gelindiğinde, enine kesen ilgi kodu içerisinde tanımlanan icra noktasında, işletilecek olan kodu ifade etmek için kullanılır. Bunun için bankamatik uygulaması düşünüldüğünde Para Çekme işlevi gerçekleştirilmeden önce çağrılması gereken günlük tutma kodu örnek olarak verilebilir.

İYP uygulamalarının işleyişinde bir birleşme noktasına gelindiğinde, o birleşme noktasını etkileyecek olan icra noktasında tanımlanmış kod olarak tanımı yapılabilen icra edecek kod (advice), sadece ve sadece kendisini tetikleyecek olan birleşme noktasının icrası ile sistem işleyişine dahil olur. İcra edilecek kodun yapısı, standart Java metodları yapısına benzer. Aralarındaki temel farklardan biri, icra edilecek kod metodları kısıtlı değer döndürebilirler.

İcra edilecek kodlar, birleşme noktalarına parametresiz veya program içerisindeki belli parametreleri alarak onlar üzerinde işlem yapabilirler. Ana program değişkenleri ve metodları üzerinde çeşitli işlemler gerçekleştirebilen bu kodlar, ilgiler içerisinde tanımlanmış olan yeni metod ve değişkenler üzerinde de değişiklik yapabilirler.

3.2.4 İYP İlgileri (Aspects)

İlgiye Yönelik Programlama temelinin, bir sistemi temel gereksinimler ve sistem gereksinimleri olmak üzere ayrıştırılması olduğu belirtilmişti. Bu programlama metodolojisinde, ayrıştırılmış tüm enine kesen işlevlerin bulunduğu birimlere İYP ilgisi (aspect) adı verilmektedir. Bu ilgilerin tamamı java sınıfları yapısındadırlar. Bu nedenle bu ilgilerden, sistemin ana sınıfları ile birlikte çalışan ilgi sınıfları olarak bahsedilecektir.

3.2.5 Ek Tip Tanımlamaları (Inter-Type Declarations)

İlgiye Yönelik Programlamanın temeli olan ilgiler, standart olarak daha önce bahsedilen birleşme noktalarından, onlara erişim için kullanılan icra noktalarından ve icra noktasına ulaşıldığında çalıştırılacak olan ilgi kodundan oluşmaktadır. Bu programlama yaklaşımı, aynı zamanda mevcut sınıfların genişletilmesi, yeni tip ve metodların da eklenebilmesi için kullanılmaktadır. Ek Tip Tanımlamaları, İYP için önemli bir kavramdır çünkü sınıflara esnek tip tanımlama özelliği eklemektedir. Çeşitli ek tip tanımları mevcuttur.

Bunlardan bazıları:

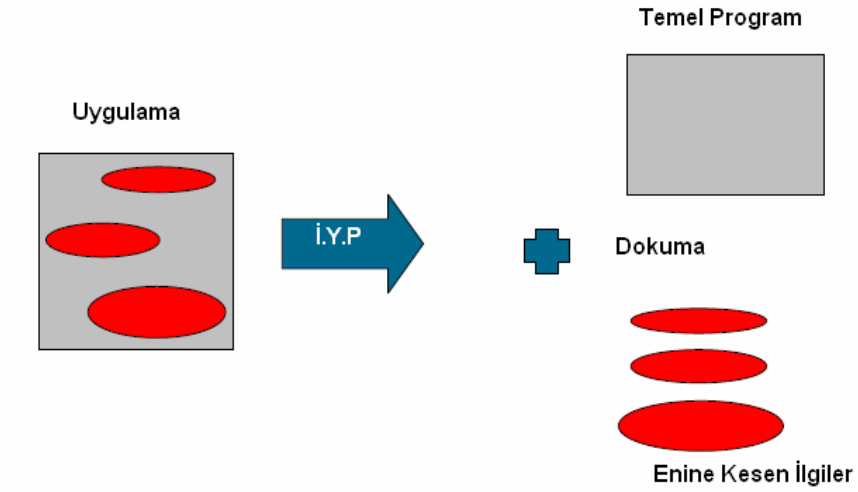
- Üye ekleme (Sınıf, metod, alan) veya tip (diğer ilgiler dahil)
- Arayüzlere üye ekleme
- Mevcut tipleri genişleten yeni tipler tanımlama
- Özel derleme hataları veya uyarıları yazma

İYP mantığında dinamik enine kesen ilgiler mantığı mevcuttur. Ek tip tanımlamaları, standart Java sınıflarını dışardan değiştirme özelliği sağlasa da sonuç Java tip tanımlamaları ile hemen hemen aynıdır. Diğer bir deyişle, standart Javadaki metod ve tip tanımlamaları için geçerli kurallar, ilgiler için de geçerlidir. Metodların üzerine yazma, tipleri genişletme, vb.

3.2.6 Dokuma (Weaving)

İYP'de, dokuyucu adı verilen mekanizma ayrıştırılmış halde bulunan ilgileri bir araya toplar. Diğer bir deyişle farklı çalışma mantığı olan ilgileri, çalışmaları gerektikleri noktaları belirleyerek birleştirir ve gereksinimleri karşılayan sistemin meydana gelmesini sağlar. Bu durumda, sistem ana fonksiyonları ile enine kesen ilgilerin birlikte çalışmaları sağlanır ve sistemin temel işlevlerinde yapılacak herhangi bir değişikliğin yönetilmesi, temel kodu

değiştirmeden sadece ilgilerin eklenmesi veya değiştirilmesi ile sağlanmış olur. Yapılan değişiklik o metodun kullanıldığı her yerde geçerli olacağından kod karmaşasının da önüne geçilmiş olur. Şekil 3.2'de İYP yazılım geliştirme aşamasında dokuma işleminin rolü görülmektedir.



Şekil 3.2 İYP yazılım geliştirme sürecinde dokuma işlemi

Enine kesen ilgiler ile sistem işlevlerini bir araya getiren dokuma işleminin, pratikte birkaç uygulama biçimi bulunmaktadır:

Derleme Zamanı (Compile-Time): İlgi kodları ve temel kodlar, derlenmeden önce bir araya getirilip dokunurlar (AspectJ çalışma prensibi).

Birleşme Zamanı (Link-Time): İlgi kodları ve temel kodlar, byte koda çevrildikten sonra dokunurlar.

Load Time (Yükleme Zamanı): İlgi kodları ve temel kodlar, byte koduna çevrilip classloader (sınıf yükleyicisi) tarafından yüklenirken dokunurlar.

Run-Time (Çalışma Zamanı): Sanal Makine (Virtual Machine), birleşme noktalarını (joinpoints) tespit ederek ilgili metodun yeri geldiğinde çalıştırılmasından sorumludur.

İYP'de iki tip dokuma yaklaşımı bulunmaktadır : Statik ve Dinamik Dokuma Yaklaşımları.

3.2.6.1 Statik Dokuma Yaklaşımı

Statik dokuma, uygulama kaynak kodlarının, seçilmiş birleşme noktalarında yeni ilgi sınıfları eklenerek değiştirilmesidir. Bu yaklaşımda ilgi kodları sınıflar içerisine eklenmektedir. Bunun sonucu olarak da entegre edilmiş sistemin icra hızı, geleneksel yöntemlerle geliştirilen uygulamaların hızı ile karşılaştırılabilir olmaktadır. Statik dokumanın dezavantajı, dokunmuş olan kodda ilgiye özel ifadelerin ayırt edilmesi zordur.

Statik dokuma işleminde derleme zamanı kısıtları önem kazanmaktadır. Standart uygulama ile geliştirilmiş olan ilgi kodları derleme zamanında birleştirilip, yeni işlevselliğe sahip olan yeni bir uygulama oluşturulur. Bu dokuma tipinde, ilgi dokuyucusu ile standart kod derleyicisi bir arada çalışmaktadır. Derleme işleminden geçirilmiş entegre kodun, çalışma zamanına kadar entegrasyon işlemleri tamamlanmış olacağından, çalışma zamanı performansı dokuma işleminden etkilenmeyecektir (Forgáč ve Kollár, 2007).

3.2.6.2 Dinamik Dokuma Yaklaşımı

Dinamik Dokumada, uygulama kodları ile ilgiler yükleme zamanı veya çalışma zamanında birleştirilirler. Bu yaklaşımda uygulamanın işleyişi, ilgiler yardımıyla, çalışma zamanında değiştirilebilir. Dinamik Dokuma işleminin avantajı, kod yazılmasına gerek kalmadan gerekli ilgilerin aktif veya pasif yapılabilmesine olanak sunmasıdır. Bu durumda sistem durdurulmadan yeni işlevler eklenebilmektedir. Bu işlemin avantajlarının yanında bazı dezavantajları da bulunmaktadır. Çalışma zamanında dokuma işlemi, sistemin performansını azaltabilmektedir. Bu dokuma tipinin diğer bir dezavantajı ise, sisteme zarar verebilecek ilgilerin de kolaylıkla sisteme entegre edilebilmesidir. Bu nedenle bu tip sistemlerde güvenlik ve performans göz önünde bulundurulmalıdır (Forgáč ve Kollár, 2007).

3.3 Bir İlgi Örneğinin İncelenmesi

Daha önceki bölümlerde örnek olarak verilen bankamatik uygulaması için ilgi sınıf kodu Şekil 3.3'te görülmektedir.

```
import banking.*;

public aspect Authentication {

    public pointcut authenticationRequired(Account account):
        execution(public * Account.*(..)) && this(account);

    before(Account account): authenticationRequired(account) {
        authenticate(account);
    }
```

Şekil 3.3 Bankamatik uygulaması için kullanılan yetkilendirme ilgisi

Şekil 3.3'te görülen bankamatik ilgisi üzerinden yola çıkarak bir ilginin bölümleri incelendiğinde;

İlgi sınıfları yapısının, standart Java sınıfları ile benzeştiği ve bir sınıf adı içerdiği (Şekil

3.3'teki ilgi yetkilendirmesidir).

Public pointcut ile başlayan bölüm, ilgi içerisindeki icra noktasının yerini belirtir. (Şekil 3.3'teki ilgiye ait icra zamanı yetkilendirmenin gerekliği olduğu yerlerdir).

Execution(public * Account.*(..) && this(account)) bölümü, Account nesnesi içerisindeki tüm metodlara erişimde, yazılmış olan ilgi kodunun icra edeceğini belirtir. Execution ile başlayan satırda belirtilmiş olan metodlar ana modüllerdeki birleşme noktalarına karşılık gelirler.

Before(Account account) bölümü ise bir icra noktasına gelindiğinde, icra edecek ilgi kodunun, birleşme noktasında belirtilen metodun hangi kısmında çalışacağını belirtir (Şekil 3.3'te before ifadesi kullanılmıştır. Bunun anlamı, ilgi kodunun Account nesnesindeki metodlardan önce icra edeceğidir).

3.4 İYP'nin Temel Uygulama Alanları

İlgiye Yönelik Programlamanın birçok kullanım alanı mevcuttur. Bunlardan en önemlileri, günlük tutma, güvenlik ve iş akışı uygulamalarıdır.

3.4.1 Günlük Tutma (Loglama)

Günlük Tutma, herkes tarafından bilindiği gibi başta kurumsal projeler olmak üzere hemen her projede yapılan işlemlerin zamanını ve kim tarafından yapıldığının bilgisinin saklanması işlemidir. Birçok uygulamada çok kritik önerme taşıyan bu işlem neredeyse uygulamalar üzerindeki tüm metodlarla birlikte çalışmalıdır. Enine kesen ilgi olarak en kolay örneklenebilecek işlem budur. Günümüzde birçok günlük tutma aracı bulunmasına rağmen ilgiye yönelik programlama günlük tutma işlemi için idealdir. Çünkü log4j gibi günlük tutma araçlarında, kod içerisinde özel yerlerde günlük tutma ile ilgi komutlar yazmak durumunda kalınabilmekte, bu da enine kesen ilgi kodunun işlevler içine yayılmasına sebebiyet vermektedir. Örneğin bir alışveriş kartı uygulaması düşünüldüğünde bu sistem için şu sınıflar tanımlanabilir:

Urun.java : Sepete eklenebilecek olan tüm ürünlerin kodlarının ve özelliklerini tanımlar

Sepet.java : Ürünlerin eklenip çıkartılabileceği ve fiyatların hesaplandığı alışveriş listesi

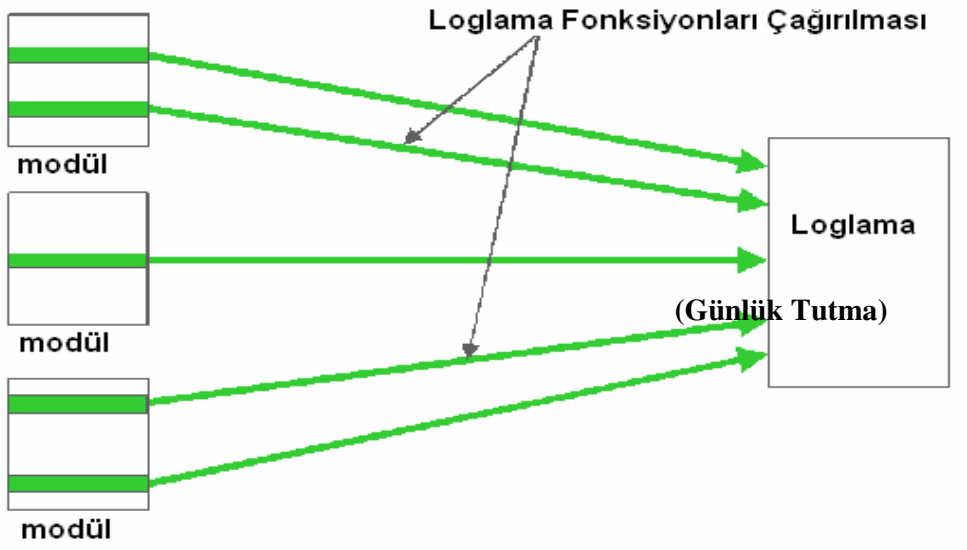
Stok.java : Ürünlerin stoktaki miktarlarını tutar.

AlısverisKartiYoneticisi.java : sepete ürün eklenmesiyle stoktaki o ürünün miktarını azaltıp, sepetin miktarını arttırır. Sepetten eleman çıkartılmasıyla da stoktaki o ürünün miktarını arttırıp, sepetteki miktarını azaltır.

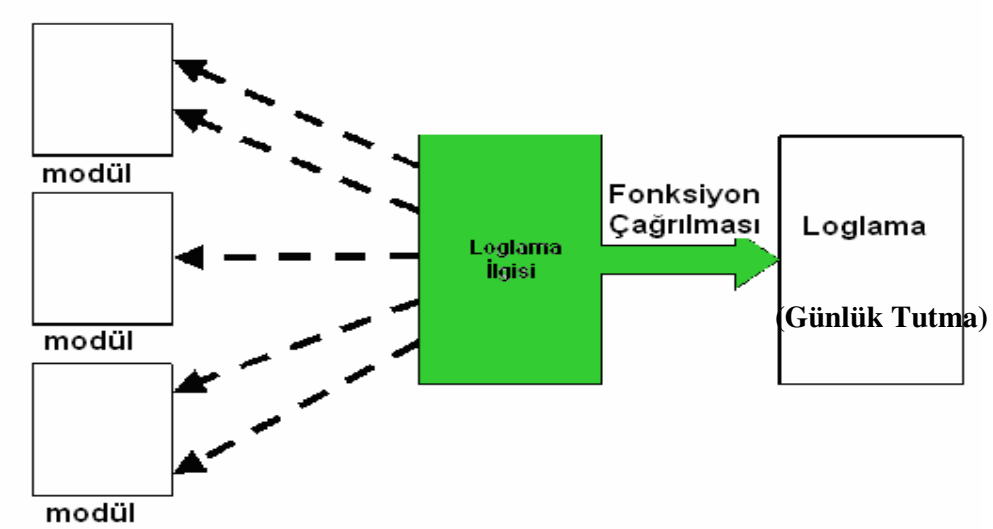
Böyle bir sistemin geleneksel yollarla günlük tutulmasında yukarıdaki sınıflardaki tüm metodların içerisine günlük tutma kodu yazılması gerekmektedir. Örneğin sepete bir ürün eklerken stoktaki miktarını azaltan, ürünlerin kodlarını getiren, fiyatlarını hesaplayan tüm bu metodların içine günlük tutma kodları eklenmelidir.

İlgiye Yönelik Programlama ile günlük tutmada ise birleşme noktası olarak tüm metodların tanımlandığı bir "sonra" icra noktası tanımlayarak, icra edilecek her işlevden sonra günlük tutma mekanizmasının çalıştırılması sağlanabilir.

İki günlük tutma yöntemin karşılaştırılması Şekil 3.4 ve Şekil 3.5'de görülmektedir.



Şekil 3.4 Geleneksel yöntemlerle günlük tutma



Şekil 3.5 İlgiye yönelik programlama ile günlük tutma işlemi

Güvenlik

İlgiye Yönelik Programlamanın uygulama alanlarından bir diğeri de uygulama güvenliğidir. Tıpkı günlük tutma gibi, bir uygulamanın güvenliği de uygulama içerisindeki pek çok modülü etkileme eğilimindedir. Bu nedenle, güvenlik de bir sistemin enine kesen ilgileri arasında gösterilebilir. İlgiye Yönelik Programlama, yazılım geliştiricilere, temel uygulama ile güvenlik uygulamasını ayrı ayrı yazma imkanı sunar (Joosen vd., 2002).

İlgiye Yönelik Programlamanın güvenlik alanına uygulanmasıyla;

- Güvenliğin kritik olduğu yerlerde hata kontrol edilmesinde ve yakalanmasında,
- Fonksiyonların icrasının başlangıç ve bitimlerinde bellek yığın taşması olup olmadığının tespitinde,
- Güvenlikle ilgili günlük tutma tutulmasında,
- Veritabanı kayıt kilitleri (lock) ekleyen kodların üretilmesi ve böylece üzerinde çalışılan kayda erişimin engellenmesinde,
- Bir programdaki öncelikli bölümlerin belirlenmesi ve bu önceliklerin uygun yerlerde çağrılıp, geri döndürülmesinde kullanılabilir.

Eğer bir uygulama güvenlik kavramlarından soyutlanırsa, o uygulama kolaylıkla çökmeye maruz kalır (Bloch vd., 2001).

3.4.2 İş Akışları

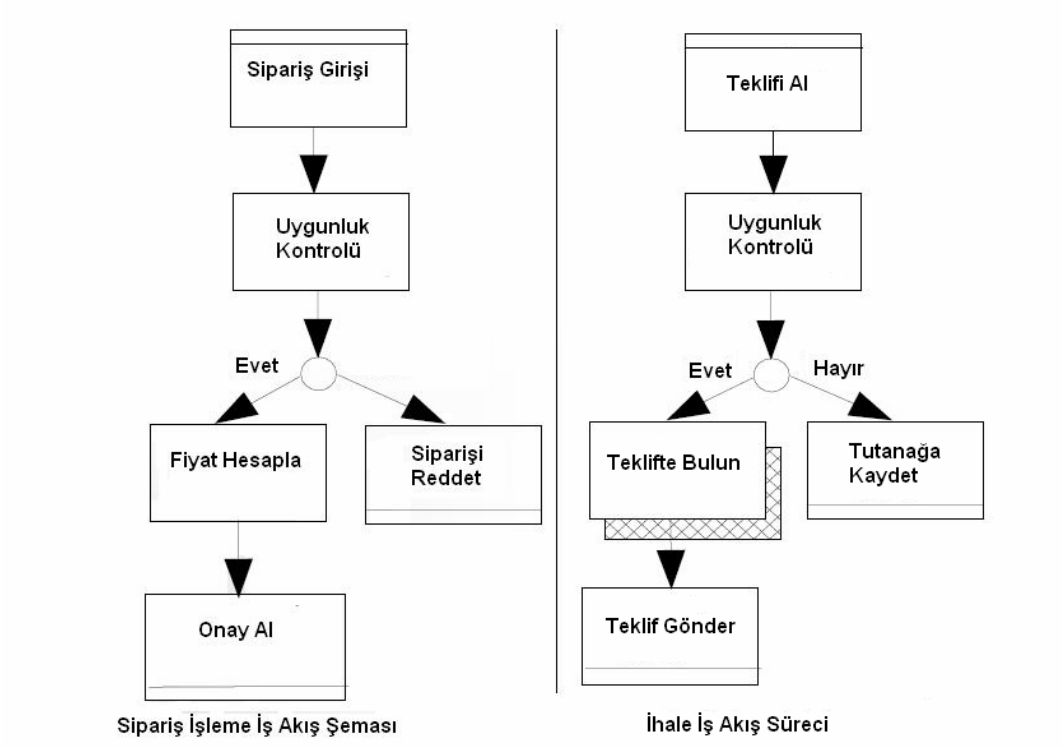
İş Akış yönetim sistemleri, belirli işlerin hangi sıralarla yapılacaklarının tanımlı olduğu otomatize edilmiş sistemlerdir. İlgiye Yönelik programlamanın iş akış sistemlerine uygulanmasıyla, bu sistemlerin daha esnek ve daha modüler yapıda olmaları sağlanabilir.

İş akış yönetim sistemlerinde, işlemlerin başlangıç ve termin tarihlerinin verilmesi, otomatik olarak hangi işlemleri tetikleyeceği gibi kavramlar tanımlanmalıdır.

Burada en önemli kavramlar, iş akış işlem aktivitesi ile geçiş verileridir. Her işlem, kendisi içerisinde birim olarak tanımlanmış aktivitelerden oluşur. Aktiviteler birbirlerine iş akış kontrol koşulları ile bağlıdırlar. Geçiş, bir aktivitenin tamamlanıp kontrolün diğer bir aktiviteye geçmesi olarak tanımlanır. Her aktivitede, gelecek olan aktivitelerin nasıl ilişkili oldukları tanımlıdır. Geçişlerde ise 3 tanım bulunmaktadır. Bunlar, yönlendiren aktivite, yönlendirilen aktivite, geçişin yapıldığı kontrol koşullarıdır.

Diğer tüm sistemlerde olduğu gibi bu iş akış sistemlerinde de güvenlik, günlük tutma, performans, yetkilendirme ve veri tutarlığı gibi sistem ilgilerinin de gerçekleştirilmesi gerekmektedir. Bu gibi enine kesen ilgilerin tanımlanması için İYP yaklaşımının kullanılması mantıklıdır. Çünkü bu sistemler müşteri odaklı ve sıkça modifiye edilebilecek nitelikteki sistemlerdir. İş mantığı ile kalite, sunumunu bir arada barındırırlar. İlgiye Yönelik iş akışı tanımlama ile işlemlerin geçiş esnekleri rahatlıkla sağlanabilir.

Şekil 3.6'da basit iki iş akış süreci görülmektedir (Charfi ve Mezini, 2006).

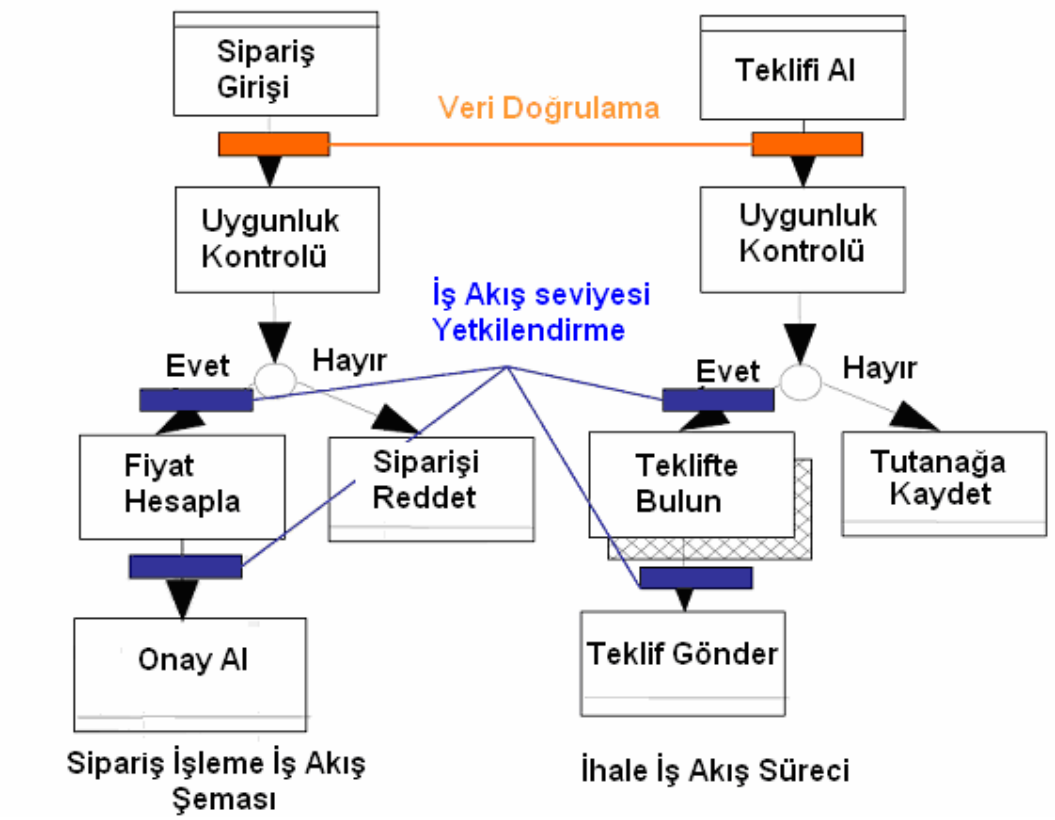


Şekil 3.6 Örnek iş akış diyagramları

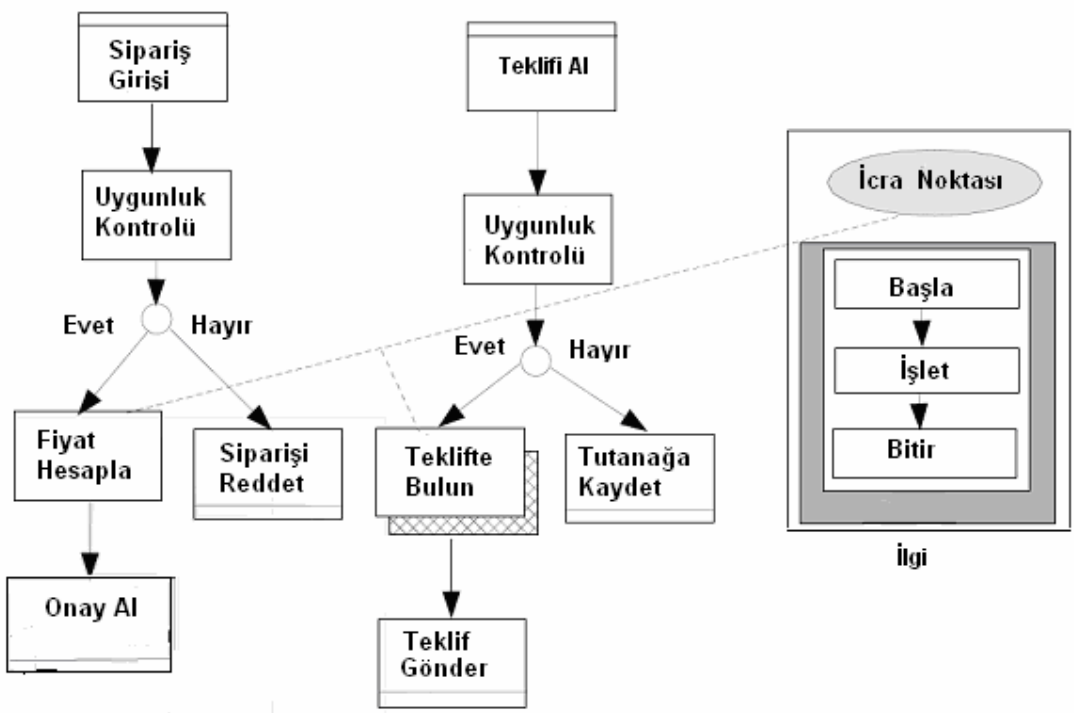
İş Akış diyagramlarının genel özellikleri aşağıdaki gibi listelenebilir:

- İş Akış değişiklikleri birinci sınıf varlıklar olarak ifade edilmezler. Asıl işlevsellik enine kesen ilgilerle birlikte ele alındığından, asıl işleve odaklanmak güçleşir.
- Aktiviteleri, değişkenleri ve uygulama tanımlamalarını içeren bir modül bulunmamaktadır. Bunlar, tüm süreçler içine yayılmış durumdadırlar.
- Değişim isteklerini anlamak ve yönetmek zordur.
- Geçici değişiklikler yapmak çok karmaşık bir işlemdir.

İş Akış süreçlerinin iyi modülerize edilmesi, dinamik istekler karşısında desteğinin daha kolay olmasını sağlayacaktır. Bu bağlamda, İYP metodolojisi, iş akış süreçlerinin modülerizasyonunda kullanılabilir. Şekil 3.7'de, Şekil 3.6'da verilen iş akış diyagramı örneği üzerindeki enine kesen ilgiler gösterilmiştir. Şekil 3.8'de ise belirlenmiş olan enine kesen ilgilerin icra edecekleri icra noktaları görülmektedir.



Şekil 3.7 İş akış sürecinde enine kesen ilgilerin tespit edilmesi



Şekil 3.8 İş akış ilgi örneği

4. İYP SÜREÇLERİNDE UML KULLANIMI

Enine kesen ilgiler, yazılım yaşam döngüsü boyunca yayılmış ve birlikte çalışması gereken fonksiyonları temsil etmektedirler. Bunların etkin ayrıştırılması yazılım geliştirme ve bakım sürecini yönetilebilir hale getirmektedir. İYP bu fikri temel alan bir yazılım geliştirme yaklaşımıdır.

UML (Unified Modelling Language), yazılım mimarilerinin yaratılması ve dokümante edilmesi için kullanılan standart grafiksel bir dildir. Birçok metod ve teoriye kaynak teşkil edebilecek fikirler ve kavramlar içermektedir. UML, birçok diyagram tipi, modelleme elemanı, gösterim şekli içeren sayısız modelleme tekniği sunmaktadır. Bu teknikler farklı karakteristiklerdeki yazılım projelerinin modellenmesi için farklı yollar sunar. UML'i oluşturan anahtar kavramlar model iyileştirmesi, ek mekanizmaların tanımlanması ve kısıtların belirtilebildiği bir dil olmasıdır. Nesneye Yönelik Programlama yaklaşımı ile geliştirilen birçok uygulamanın modellemesi ve süreçlerinin yönetiminde kullanılan UML'in İlgiye Yönelik Programlama süreçlerine de dahil edilmesi kolay yönetilebilen uygulamalar geliştirilmesine katkıda bulunacaktır. Bu bölümde İYP süreçlerinde (Analiz, Tasarım, Modelleme ve Geliştirme) UML dili kullanımı incelenecektir.

4.1 UML ile İlgiye Yönelik Gereksinimlerin Belirlenmesi

Günümüzde İlgiye Yönelik Programlama'nın kullanımı genelde uygulama seviyesinde olmaktadır. Bu kısımda İlgiye Yönelik Ayrıştırma işlemine gereksinim mühendisliği bazında odaklanılacaktır. Gereksinim aşamasında Enine Kesen İlgilerin ayrıştırılmasında UML'den yararlanılacaktır. UML temelli İlgiye Yönelik Gereksinim Mühendisliğinin 2 fazlı etkisi vardır. İlk olarak geniş çapta yayılması gereken ilgilerin avantaj ve dezavantajlarının yazılım geliştirme sürecinin erken fazlarında tespit edilmesini sağlar. Böylece tüm ilgili kullanıcıların kullanım destekleri tasarlanmış olmaktadır. İkinci bir avantajı ise, NYP'de alışılagelmiş bir kullanımı olan UML'in İYP süreçlerine adapte edilebilmesi kolay olmaktadır (Araújo vd., 2002).

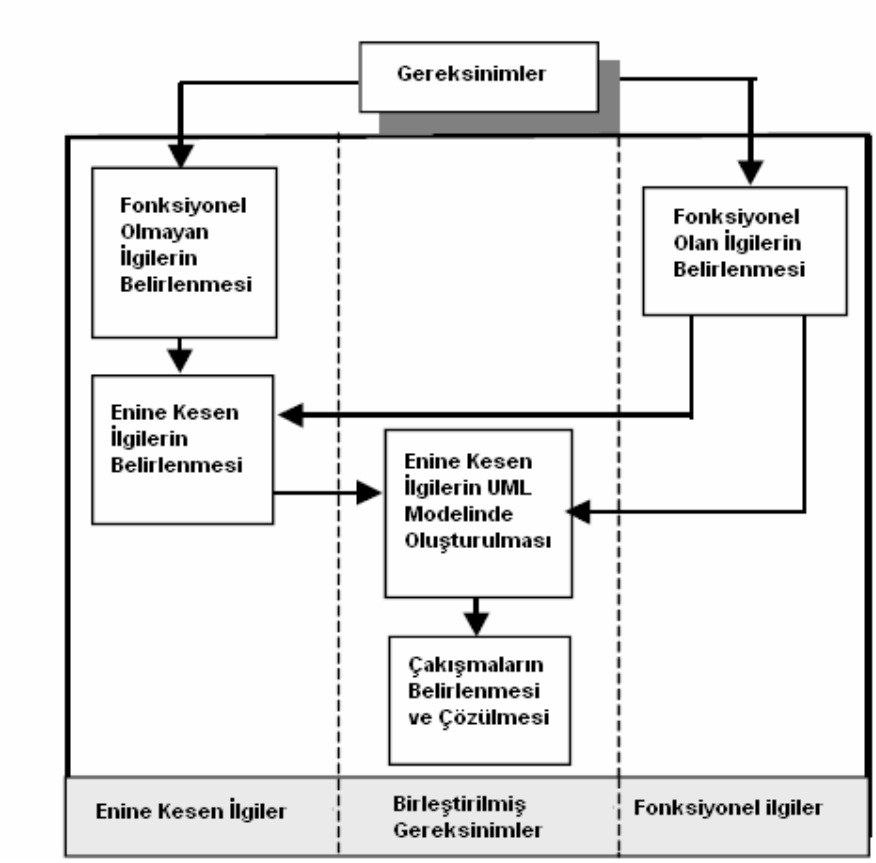
Enine kesen ilgilerin ayrıştırılması, yazılım geliştirme sürecinin gereksinim aşamasından uygulama aşmasına kadar devam etmesi öngörülen temel bir yazılım geliştirme prensibidir. Buradaki esas düşünce, belli bir zamanda sadece sistemin bir özelliğinin yönetilmesidir. Ayrıştırılması gereken ilgiler, diğer ilgilerle aynı anda çalışması gereken ilgilerdir. Bunların kolaylıkla yönetilebilmesi için çok iyi ayrıştırılmaları gerekmektedir. Bazı ilgiler (güvenlik, cevap süresi, yetkilendirme) gereksinimlerin belirlenmesi aşamasında tespit edilip

ayrıştırılabilirler. İYP'de gereksinim mühendisliği kapsamında ele alınacak ilgiler bu tip enine kesen ilgilerdir. Bazı ilgiler ise, seçilen teknolojinin kısıtlarına bağlı olarak tasarım ve uygulama aşamasında tespit edilip ayrıştırılabilirler.

İYP Gereksinim mühendisliği bazında ilgilerin ayrıştırılması, sistemin fonksiyonel ilgilerini kısıtlayan ve etkileyen, geniş çaplı ve fonksiyonel olmayan ilgilerin yönetimini ele alır. Gereksinimlerin analizi sırasında, kullanıcıların sistem tanımlarından öncelikle fonksiyonel ve fonksiyonel olmayan istekler ayrıştırılmalıdır. Örneğin bir banka hesabında müşterinin tanımı şöyle olsun: "Banka hesabı para çekilmesini ve geri iadesini desteklemeli ve sistem kullanıcı isteklerine kısa sürede cevap verebilmelidir." Buradaki tanımlamadan kullanıcının sistem fonksiyonelitesi olarak tanımladığı Para Çekilmesi ve Geri İade Edilebilmesi rahatlıkla seçilebilen fonksiyonel ilgilerdir. Fakat dikkatlice incelendiğinde kullanıcının fonksiyonel olmayan Cevap Süresi ile ilgili tanımlaması da bulunmaktadır. Bu nedenle burada iki ana gereksinim bulunmaktadır. Bunlar fonksiyonel ve fonksiyonel olmayan gereksinimlerdir. Çoğu gereksinim mühendisliği yaklaşımı sadece fonksiyonel gereksinimlere yönelmektedir. Fakat kullanıcının sistemden beklediği, sistem işleyişini etkileyen diğer gereksinimler de göz önünde bulundurulmalıdır. Buradaki örnekte cevap süresi bu tip gereksinimlerdendir. Gereksinim mühendisliği aşamasında ilgilerin ayrıştırılmasında temel gösterim olarak UML kullanılabilir. İlgilerin UML ile entegre edilmesi, gösterim gücünü arttırmaktadır (Araújo vd., 2002).

4.1.1 İlgiye Yönelik Gereksinim Yaklaşımı

Bu yaklaşımda amaçlanan, genel İlgiye Yönelik Gereksinim Mühendisliği'nin UML modelleme dili kullanılarak gerçekleştirilmesidir. NYP yaklaşımında sıkça kullanılan bu modelleme dilinin İYP gereksinim sürecinde de kullanılması, sistemin modülerize edilmesini kolaylaştırmakta ve yeniden kullanılabilir bileşen oranını arttırmaktadır. Basit bir İlgiye Yönelik Gereksinim Mühendisliği süreci, Şekil 4.1'de görülmektedir (Araújo vd., 2002). Şekil 4.1'de de görüldüğü gibi analiz edilmiş tüm sistem gereksinimleri önce fonksiyonel ve fonksiyonel olmayan ilgiler olarak ayrıştırılmaktadır. Fonksiyonel Olmayan İlgiler'den, Fonksiyonel Olan İlgileri enine kesen ilgiler ayrılmakta ve bu ilgilerin UML modelinde gerçekleştirilmesi sağlanmaktadır. Daha sonraki aşamada ise çakışmaların belirlenmesi ve bu çakışmalara yol açan problemlerin çözümleri yer almaktadır.



Şekil 4.1 İlgiye yönelik gereksinim mühendisliği

İlgiye Yönelik Gereksinim Mühendisliğinde 3 dikey bölüm tanımlanmıştır (Şekil 4.1).

- **Enine Kesen İlgiler:** Bu kısım, ilk başta fonksiyonel olmayan ilgilerin tespit edilmesi, daha sonra bu ilgilerden hangilerinin enine kesen ilgiler (fonksiyonel ilgiler ile çakışan ilgiler) olduğunun tespit edilmesi aşamalarını içerir (Aday ilgilerin bulunmasına yöneliktir). Bu ilgiler Çizelge 4.1 kullanılarak belirlenmektedir.
- **Fonksiyonel İlgiler :** Bu kısım geleneksel olarak sistem fonksiyonlarını belirlemeyi kapsamaktadır. Belirlenmesinde UML model yaklaşımı kullanılmaktadır.
- **Birleştirilmiş Gereksinimler :** Bu kısım ise, UML ile modellenmiş olan fonksiyonel ilgilerin, fonksiyonel olmayan enine kesen ilgiler ile birleştirilmesi ile başlar. Daha sonra, birleştirme sonucu ortaya çıkan çakışmaların belirlenmesi ve çözülmesi ile devam eder. Modelin birleştirme aşamasını tanımlayabilmek için genellikle çoğu ilgi ayrıştırma işleminde kullanılan kısmen kaplamak (overlapping), yerine geçmek (overriding) ve sarmalamak (wrapping) kavramları benimsenmiştir. Bu kavramlar incelenecek olursa;

Kısmen Kaplamak (Overlapping): İlgi gereksinimleri, etkiledikleri fonksiyonel gereksinimler üzerinde değişiklik yaparlar. Bu durumda ilgi gereksinimlerine, fonksiyonel gereksinimlerden önce de sonra da ihtiyaç duyulabilir.

Yerine Geçmek (Overriding) : İlgi gereksinimleri fonksiyonel gereksinimleri karşılayabilirler. Bu durumda ilgi gereksinimleri tarafından tanımlanmış davranışlar, fonksiyonel davranışların yerine geçer.

Sarmalamak (Wrapping) : İlgi gereksinimleri, fonksiyonel gereksinimleri kapsayabilir. Bu durumda fonksiyonel gereksinimler ile tanımlanmış davranış, ilgi gereksinimleri ile tanımlanmış davranış tarafından sarmalanır (Araújo vd., 2002).

Çizelge 4.1 Enine kesen ilgilerin tariflenmesi

Enine Kesen İlgi:	<Adı>
Tanım:	<Çalışma tanımlaması>
Öncelik	<Yüksek, Orta, Düşük>
Gereksinimler Listesi	<İlgiyi tanımlayan gereksinimler>
Model Listesi	<İlgi tarafından etkilenen UML modelleri>

4.1.2 İlgiye Yönelik Gereksinim Yaklaşımının Uygulanması

Bu bölümde, bir önceki bölümde anlatılan İlgiye Yönelik Gereksinim Yaklaşımlarının bir örnek üzerinde uygulanması incelenecektir. Burada incelenecek sistem Portekiz Otoyolu ağının bir örneğidir (Araújo vd., 2002).

4.1.2.1 Gereksinimlerin Belirlenmesi

Yol trafik ücretlendirme sisteminde, yetkilenmiş araçlar gişe girişlerinde otomatik olarak ücretlendirilmektedir. Bu girişler yeşil şerit adı verilen özel şeritlerde bulunmaktadır. Sürücüler araçlarına "Gizmo" adı verilen bir aygıt kurmak zorundadırlar. İzin verilen araçların kayıt işlemleri sahibinin kişisel bilgisi, banka hesap numarası ve araç detayları ile yapılabilmektedir. Gizmo, kullanıcıya bir ATM kullanılarak aktif edilmek üzere gönderilir.

Gizmo, gişe girişlerindeki sensörler tarafından okunur. Okunan bilgi sistem tarafından kaydedilir ve şahsi hesabı borçlandırmak amacıyla kullanılır.

Eğer izin verilmiş bir araç, yeşil şeride girerse bir yeşil ışık yanmakta ve borçlanılan miktar görüntülenmektedir. Eğer izin verilmemiş bir araç gişeye girerse sarı bir ışık yanmakta ve bir kamera otomatik olarak aracın plakasını çekmektedir. Otoyollarda ödenen otoyola giriş çıkış ücretleri, aracın tipine ve gidilen yolun uzunluğuna bağlı olarak farklılıklar göstermektedir.

4.1.2.2 Fonksiyonel Olmayan İlgilerin Belirlenmesi

Bir önceki bölümde gereksinimleri tanımlanan problemde iki tip gereksinim bulunmaktadır. Bunlar genel olarak tüm uygulamalarda bulunan fonksiyonel ve fonksiyonel olmayan gereksinimlerdir.

Fonksiyonel olmayan gereksinimlerin belirlenmesi için kullanıcı ile sistem kısıtlarının netleştirilmesi gerekmektedir. Örnek sistemdeki fonksiyonel olmayan ilgiler, bir aracın gişeyi kullanırken sistemin tepki anı ve cevap verme süresidir. Gereksinimlerin analizinde bu süre, izin verilen hız limitinin ve gişeyi oluşturan elemanların (Araçları tespit eden sensörler, ışıklar, kamera vs.) arasındaki uzaklığın bir fonksiyonu olarak hesaplanacak şekilde belirtilebilir. Bunlar tüm bileşenlerin fiziksel olarak konumları hakkında fikir vermektedir. Örneğin eğer aracın plakası ve sürücü fotoğraflanmak isteniyorsa, kamera sadece plakanın görüntülenmek istendiği aracın arka kısmının fotoğraflandığı yerden farklı bir yere yerleştirilmelidir. Ayrıca araç gişeden ayrılmadan yeşil veya sarı ışığın yakılması da çok önemlidir. Aynı şekilde, borçlanılan miktarın da kullanıcı gişeden ayrılmadan kullanıcının görebileceği bir zamanda görüntülenebilmelidir. Yani sistem gereksinimleri incelendiğinde, gişenin sürücünün yakılan ışığı ve borçlanılan miktarı görebileceği hızda cevap vermesi gerekmektedir. Burada göz önüne alınması gereken fonksiyonel olmayan enine kesen ilgi cevap süresidir (Response time).

Gişe cevap süresi şu gereksinim adımları ile tanımlanabilir:

G1: Araç gişeye girdiği anda gişedeki sensörler araç tanımlayıcısını okumalıdır (t1 anı).

G2: Gişeyi kullanan yetkisiz araçların fotoğrafları çekilir (t2 anı)

G3: Araç gişeye girdiği an sistem ışık yakmalıdır (t3 anı)

G4: İzin verilmiş bir araç gişeden geçerken borçlanılan miktar görüntülenmelidir (t4 anı)

Diğer fonksiyonel olmayan gereksinimler ise güvenlik, çok kullanıcılı sistem, uyumluluk doğruluk, yasal öğeler olarak tanımlanabilir (Araújo vd., 2002).

4.1.2.3 Fonksiyonel İlgilerin Belirlenmesi

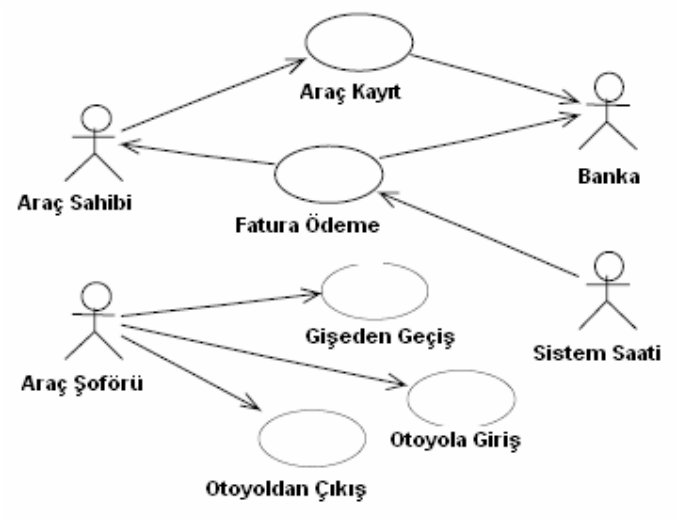
Fonksiyonel Gereksinimlerin belirlenmesi için UML durum diyagramları kullanılmaktadır. Sistemin gereksinimleri analiz edilirken öncelikle aktörler şu şekilde belirlenebilir:

- **Araç sahibi :** Aracın kaydettirilmesinden sorumludur.
- **Araç Sürücüsü :** Aracın yönetiminden ve gizmonun kurulumundan sorumludur.
- **Banka :** Araç sahibinin hesabını temsil eden bir varlık.
- **Sistem saati :** Aylık olarak borçların hesaplanmasını tetikleyen sistem iç saati.

Yukarıda listelenen aktörler tarafından gereksinim duyulan senaryolar şunlardır:

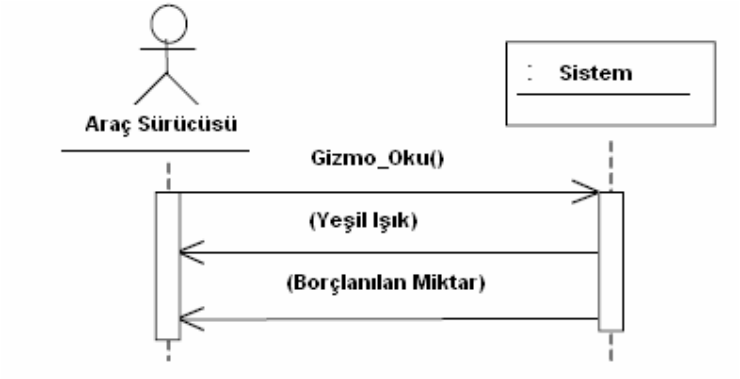
- **Aracın Kaydettirilmesi:** Aracın ve sahibinin kaydettirilmesi ve hesabın bunlarla ilişkilendirilmesi için banka ile iletişime geçilmesi.
- **Gişeden Geçme:** Araçların gişeye girme anında araçta bulunan Gizmo cihazının okunması ve kayıtlı olup olmadığının tespitinin yapılması işlemlerini kapsar. Eğer Gizmo kayıtlı ise gişede yeşil ışık yanmakta ve sürücüye borçlanılan miktar gösterilmektedir. Eğer kayıtlı değilse sarı ışık yanmakta ve fotoğraf çekilmektedir.
- **Otoyola Giriş:** Gizmo kontrol edilir, ışık yakılır ve geçiş kaydedilir. Eğer Gizmo kayıtlı değilse resim çekilir.
- **Otoyoldan Çıkış:** Gizmo kontrol edilir, eğer aracın girişi varsa gidilen yol hesaplanarak ödenmesi gereken ücret gösterilir ve geçiş kaydedilir. Eğer Gizmo kayıtlı değilse veya aracın gişe girişi yoksa sarı ışık yakılır ve fotoğraf çekilir.
- **Faturanın Ödenmesi :** Her araç için tüm geçişlerin aylık toplamları hesaplanır. Araç sahibinin hesabına yansıtılır.

Şekil 4.2'de yol trafik sistemi senaryoları ve aktörlerinden oluşan durum diyagramı gösterilmektedir (Araújo vd., 2002).



Şekil 4.2 Yol trafik sistemi durum diyagramı

Durum diyagramları, önceden tanımlanmış olan aktörler ve senaryolar kullanılarak hazırlanmaktadırlar. Daha sonra her senaryo ardışıl (sequence) diyagramları kullanılarak detaylandırılırlar. Gişe Geçiş, Otoyola Giriş, Otoyoldan Çıkış senaryoları için en az 2 diyagramla tanımlama yapılabilir. İlki izin verilmiş araçlar için ikincisi ise izin verilmemiş araçlar için tanımlanabilir. Şekil 4.3, ana senaryo için bir ardışıl diyagramında sistem içindeki tüm nesnelerin etkileşimini göstermektedir.



Şekil 4.3 İzin verilmiş aracın gişeden geçmesi ardışıl diyagramı

Diyagramda (Şekil 4.3), Yol Trafik Ücretlendirme Sistemi Gizmo cihazını okumakta ve izin verilmiş bir araç sistemden geçerken yeşil lamba yakılmakta ve sürücüye ödemesi gereken miktar gösterilmektedir (Araújo vd., 2002).

4.1.2.4 Enine Kesen İlgilerin Belirlenmesi ve Tarifi:

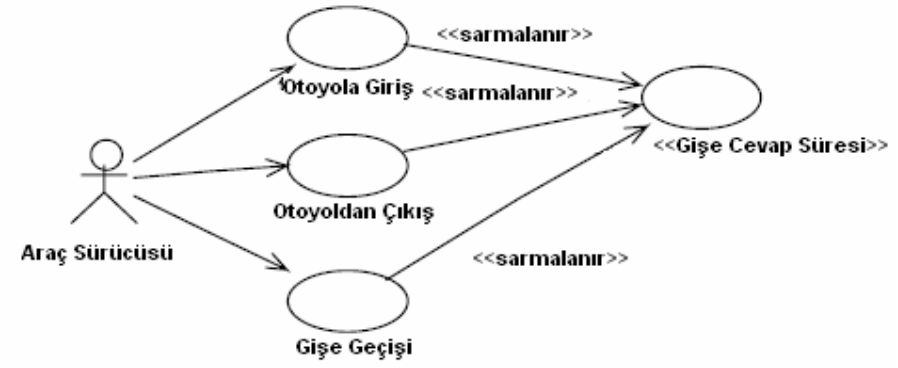
Fonksiyonel olmayan bir ilgi, eğer bir veya birden fazla durum diyagramını etkiliyor ise bu ilgi bir enine kesen ilgidir. Bu sistemde aracın sisteme girdiği andan itibaren sistemin cevap verme süresi bir enine kesen ilgi olarak tarif edilebilir. Bu fonksiyonel olmayan enine kesen ilgi, Gişeden Geçiş, Otoyola Giriş ve Otoyoldan Çıkış senaryolarını etkilemektedir. Bundan dolayı gişe cevap süresi bir enine kesen ilgidir. Bu enine kesen ilginin tarifi Çizelge 4.2'de görülmektedir (Araújo vd., 2002).

Çizelge 4.2 Gişe girişi cevap süresi

Enine Kesen İlgi	Gişe Girişi Cevap Süresi
Tanım	Sistem, araç gişeden ayrılmadan cevap verebilmelidir.
Öncelik	Yüksek
Gereksinim Listesi	G1,G2,G3,G4
Model Listesi	Senaryolar: Gişe Geçişi, Otoyola Giriş, Otoyoldan Çıkış

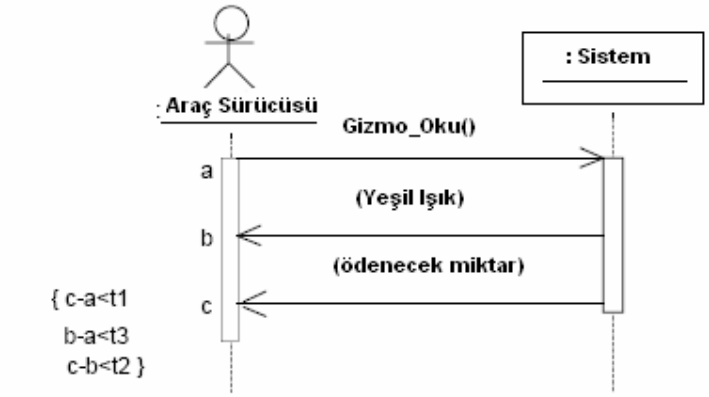
4.1.2.5 Enine Kesen İlgilerin UML Modelleri ile Birleştirilmesi

Fonksiyonel ve enine kesen gereksinimlerin birleştirilmesinde başlıca 2 kriter mevcuttur. Bunlardan biri bütünlük diğeri ise yeterliliktir. Bütünlük ile sistemden beklenen tüm gereksinimlerin sağlandığı garanti edilir. Yeterlilik ile de ilgi içerisindeki tüm gereksinimlerin birleştirme işleminde rolleri belirlenir. Ele alınan örnekte, enine kesen ilgi olarak belirlenen cevap süresinin, fonksiyonel gereksinimler ile birleştirilmesi ele alınacaktır. Durum Diyagramı için özel bir tip tanımlaması yapılmıştır. <<Gişe Cevap Süresi>> (Şekil 4.4)



Şekil 4.4 Enine kesen ilginin, sistem fonksiyonları ile birleştirilmiş UML gösterimi

Gişe Cevap Süresi ile sistemin birleştirilmiş gereksinimlerinin, gerçek zaman kısıtları da dahil edilerek oluşturulmuş ardışıl diyagramı Şekil 4.5'te görülmektedir (Araújo vd., 2002).



Şekil 4.5 İlgi ile birleştirilerek oluşturulmuş ardışıl diyagramı

Şekil 4.5'te görülen ardışıl diyagramı içerisine, olay belirleyicileri dahil edilmiştir (a,b ve c). Bunlar gönderilen mesajlara başlangıç noktası teşkil etmektedirler. Bu olay belirleyiciler daha sonra, olaylar arasındaki göreceli zamanların ifade edilmesi amacıyla zaman işaretleme ifadelerine eklenmiştir.

Bu ifadeler kaşlı parantezler içerisinde {} zamanlama kısıtlarını belirlemek amacıyla kullanılmışlardır. Birçok kısıtın olduğu bu gibi durumların ifadesinde aynı zaman işaretleme ifadesi içerisinde (kaşlı parantezler) çoklu zamanlama kısıtları tanımlanabilir (Şekil 4.5).

4.1.2.6 Çakışmaların Belirlenmesi ve Çözülmesi

Enine Kesen İlgilerin, fonksiyonel gereksinimler ile birleştirilmesi tutarsızlıklar oluşmasına sebep olabilir. Bu, enine kesen ilgiler, uygulamanın aynı metoduna etki ettiklerinde sıkça karşılaşılan bir durumdur. Örneğin Gişe Girişini etkileyen tespit edilmiş 2 enine kesen ilgi, "Cevap Süresi" ve "Güvenlik" 'tir. Bu iki enine kesen ilginin, sistemin aynı fonksiyonel ilgi kümesi ile birleştirilmesi bir çakışmaya neden olabilmektedir. Çünkü bu 2 enine kesen ilgi birbiri ile ters etkileşmektedir. Böyle durumlarda enine kesen ilgiler, önceliklerine göre sisteme dahil edilmelidirler. Örnek problemde öncelikle yapılması gereken, tüm enine kesen ilgilerin birbirleriyle etkileşimlerinin ve önceliklerinin belirlenmesidir. Tespit edilen etkileşimler negatif veya pozitif olabilir. Eğer iki enine kesen ilgi aynı fonksiyonel gereksinim seti üzerinde etkili ise ve birbiri ile negatif etkileşiyorlarsa, çakışma oluşması kaçınılmazdır. Sistemin tamamını veya bir kısmını etkileyecek bu tip çakışmaların çözülmesi için gereksinimi bildiren kullanıcılarla tekrar gereksinim görüşmesi yapılmalıdır. Çelişen gereksinimler sisteme dahil edildiğinde veya edilmediğinde ne gibi dezavantaj ve avantajların ortaya çıkabileceği analiz edilmelidir. Bu analiz çalışmalarından sonra ilgilerin öncelikleri tespit edilmeli ve sistem tasarımı fazına geçilmelidir (Araújo vd., 2002).

4.2 İlgiye Yönelik Modelleme için UML kullanımı

İlgiye Yönelik Programlama yaklaşımı, uygulama içerisindeki enine kesen ilgilerin ayrıştırılması ve programın modülerleştirilmesi esaslarına dayanmaktadır. Bu bağlamda İYP'nin hedefi, NYP'nin eksik kaldığı yerlerde kod karmaşasına ve kod dağılımına sebep olmadan onun eksiklerini tamamlamaktır. İYP'de modelleme yapılabilmesi için:

- Hangi işlevlerin, hangi işlevleri nerelerde kestiği adreslenmelidir.
- Enine kesen ilgileri, diğer ilgilerden ayırmak için bir yöntem belirlenmeli (Gereksinim Aşamasında) ve sistem İYP ilgilerine ayrıştırılmalıdır (Kandé vd., 2002).

İYP, yazılım geliştiricilere sistemlerini bir modelleme tekniği ile ifade etmelerinde geniş bir destek sunamamaktadır. Bunun yanında, NYP'nin eksiklerini tamamlayan bir programlama yaklaşımı olduğundan dolayı, NYP'nin modellemesinde kullanılan UML (Unified Modelling Language), İYP yaklaşımı için de kullanılabilir (Sapir vd., 2002).

UML, yazılım sistemlerinin özelleştirilebildiği, görüntülenebildiği, yapılandırılabildiği ve dokümante edilebildiği grafiksel bir modelleme dilidir. UML nesne modelinin anlamsal kısmını tanımlayan, nesne yapıları arasındaki ilişkilerin de görülüp yönetilebildiği grafiksel bir gösterim dilidir. Genişletilebilir bir dil olmasından dolayı İYP projelerinin modellenmesinde de kullanılmaktadır (Sapir vd., 2002).

4.2.1 UML Kullanılarak İlgi Geliştirme Metodolojisi

İlgiye Yönelik Programlamada, ilgilerin uygulama geliştirme fazından ayrıştırılması, yazılım geliştirme sürecinin yönetilebilmesi açısından önemlidir. Bu yaklaşıma göre ilgi, geliştirilmekte olan uygulamayı enine kesen modülerize edilmiş bir yapıdır. İlgi kodları uygulama kodları ile birleştirilmez veya uygulama kodlarının mantıksal yapısını değiştirmez. Uygulama kodları sabit, modüler yapısını korur. İlgilerin kendi geliştirme süreleri vardır ve tanımlı İYP süreci ilgi geliştirmeyi de kapsamaktadır (Şekil 4.1).

Uygulama Geliştirmenin analiz ve tasarım fazlarında, standart uygulamanın ve ilgilerin geleneksel UML diyagramları oluşturulur. İlgi UML diyagramları, ilgilerin uygulama ile ilişkilerini göstermek için tasarlanır (Sapir vd., 2002).

İlgi kullanım kısıtları, uygulama fazında kullanılmaktadır. Kısıtların etkileri, çalışma ve derleme zamanlarında ortaya çıkmaktadır. Bir kod üreteci ilgi modelini, uygulama nesne diyagramını ve kullanım kısıtlarını baz alarak ilgi dosyalarını oluşturur. İki tip ilgi dosyası mevcuttur :

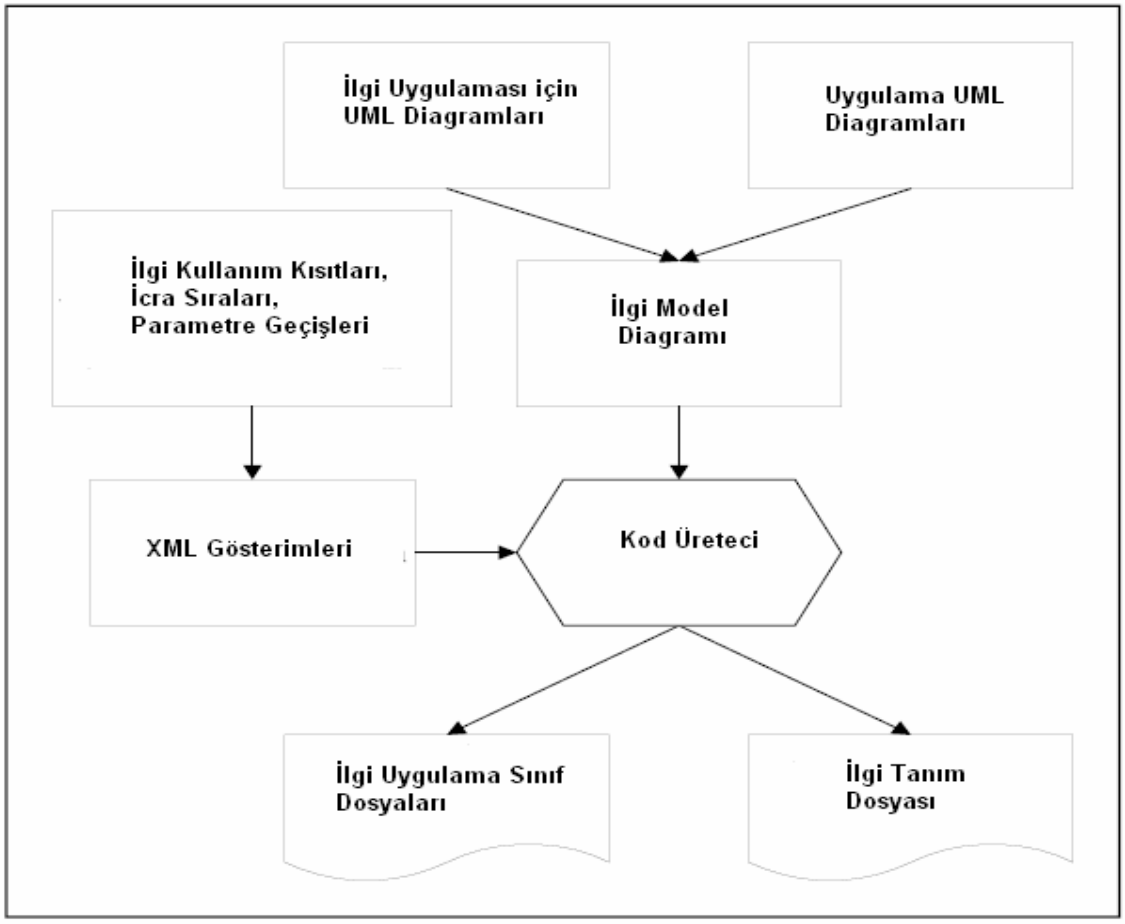
- UML Diyagramları kullanılarak oluşturulan İlgi Sınıf Dosyaları.
- Kullanım Kısıtları ve İlgi Nesne Model Diyagramından oluşan İlgi Tanım Dosyası.

Şekil 4.5'te görülebilen İlgi Geliştirme Metodolojisine göre öncelikle İlgi Uygulaması için ve Standart Uygulama için UML diyagramları oluşturulur. Bu iki diyagram birleştirilerek, İlgi Model Diyagramı meydana gelir. İlgi Kullanım kısıtları (ilgilerin tipleri), parametreleri ve icra sıraları belirlenir. Bu değerlerden XML dosyaları oluşturulur. Oluşturulan XML dosyaları ve İlgi Model Diyagramı, Kod Üretecine girdi oluştururlar. Kod üretecinin görevi ise ilgi dosyalarını oluşturmaktır.

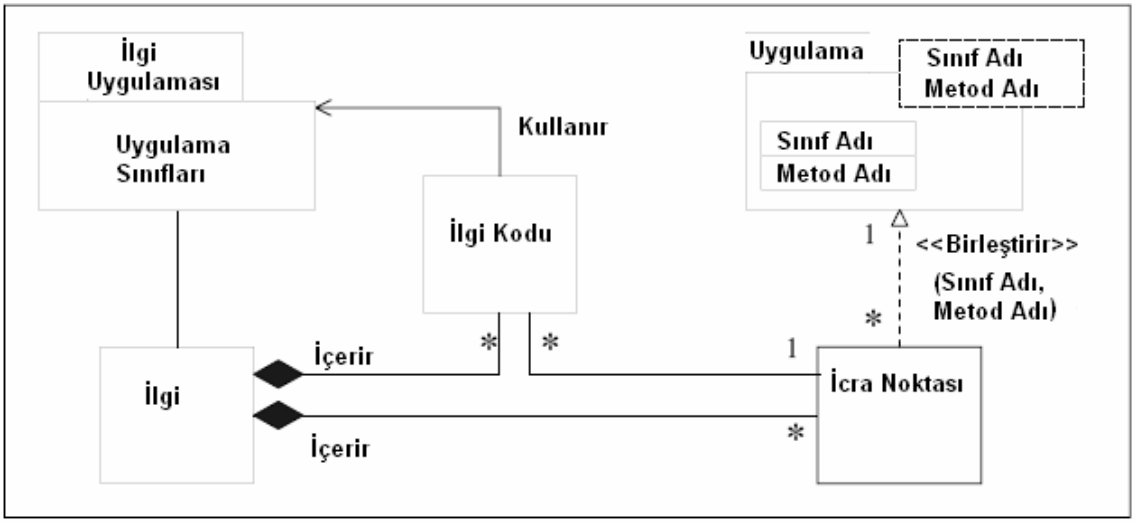
4.2.2 İlgi Analiz ve Tasarımı

İYP ile bir uygulama geliştirilmeden önce, gereksinim analizi aşamasında belirlenen gereksinimlerin fonksiyonel ve fonksiyonel olmayan ilgiler şeklinde ayrıştırılmaları gerekmektedir. Bu ayrıştırma işleminden sonra sistemden beklenen asıl işlevsellik, Standart Uygulama, bu standart uygulamadaki bazı işlevleri enine kesen ilgilerden oluşan yapıya ise İlgi Uygulaması adı verilir. Şekil 4.2'de Standart Uygulama UML Diyagramı ile İlgi

Uygulaması UML Diyagramı birleştirilerek oluşturulan İlgi Model Diyagramı görülmektedir.



Şekil 4.6 İlgi geliştirme metodolojisi



Şekil 4.7 İlgi model diyagramı

Şekil 4.7'de görülen İlgi Model Diyagramının içeriği aşağıdaki gibidir:

- Uygulamada, UML şablonu formatında bir UML paket bileşeni bulunmaktadır (Bu bileşen Şekil 4.1'de gösterilen İlgi UML Diyagramlarını temsil etmektedir).
- Uygulamaya geçirilen parametreler Metod Adı ve Sınıf Adı'dır.
- Uygulama parametreleri gerçek değerlerle bağlı olmalıdır. Burada <<birleştirir>> ilişkisi ile bu bağımlılık sağlanmaktadır. Parametreler ise bağlantı okunun yanında parantez içinde gösterilmektedir (Metod Adı, Sınıf Adı).
- İlgi isimli bir sınıf, icra noktası ve icra edilecek ilgi kodunu içeren ilgi tanımlamasını gösterir. Her icra noktası sıfır veya daha fazla icra kodu içerir ve her ilgi kodu bir icra noktası ile ilişkili olmalıdır.
- Bir ilgi sınıfı, ilgi uygulama sınıfları ile ilişkilidir.

Bu yaklaşıma (UML ile modelleme) göre, bir ilgi ile uygulama arasındaki etkileşim ilginin icra ettiği yer ve zamanda gerçekleşmektedir. İlgi Model Diyagramı, ilginin çalışma yerini gösterirken, UML ardışıl diyagramı ise ilginin çalışma zamanını (uygulama kodundan önce, sonra veya aynı zamanda) göstermektedir.

4.2.3 Analiz ve Tasarım Fazlarında İlgi Kullanım Kısıtları

İlgi Kullanım Kısıtların (İKK) belirlenmesi, bir ilginin analiz ve tasarımının tamamlanması için gerekli olan ek bir adımdır.

- **Sabit Kısıtı:** Sabit Kısıtı, herhangi bir çalışma zamanı verisini veya uygulamanın statik yapısını değiştirmeyen ilgi tanımlamaları için kullanılır. Sabit kısıtı kullanılarak tanımlanmış bir ilgi, "around" ilgisi kullanarak herhangi bir metodu ezemez, çalışma zamanında "set" komutunu kullanarak herhangi bir veride değişiklik yapamaz veya herhangi bir uygulama metodunu çağıramaz. Bir Sabit ilgisi, birleşme noktalarındaki parametreleri okuyabilir bu nedenle loglama, sistem izleme gibi ilgilerin tanımlanmasında kullanılması uygundur.
- **Yegane Kısıtı:** Uygulamayı enine kesen ve sadece bir kopyasının yaratılması gerekli ilgileri göstermek için kullanılır. İlginin çalışan kopyası, uygulama süresince kullanılabilir durumdadır. Yegane tasarım kalıbında (bir sınıfın sadece bir kopyasının olduğu ve uygulama içinde ortak bir erişim noktası olduğunu belirten tasarım kalıbı) olduğu gibi, bu kısıt da bir ilgi sınıfı için merkezi bir işlem ve analiz noktası belirlemek için yararlıdır.
- **Zaman Limiti Kısıtı:** İlgi seviyesinde, bir ilgi uygulamasının çalışma zamanı üst limitini, İcra Noktası seviyesinde ise (örneğin bir icra noktasına erişip ilgi kodunu çalıştırırken) ilgi icra zamanının üst limitini belirtir. Bu kısıt gerçek zamanlı uygulamalar için kullanışlıdır.
- **Zamanlama Kısıtı:** Zamanlama Kısıtı ilgi icra zamanını tanımlar. Bir ilgi, herhangi bir icra noktasından önce, sonra veya o icra noktasına paralel çalışmaya başlayabilir. Bu çalışma zamanları sırasıyla "Önce","Sonra" ve "Esnasında" icra zamanlarına denk gelirler. İlginin paralel icrası, zamanın kritik olduğu ve ilgi icra zamanından etkilenmemesi gereken durumlarda kullanılır.

- **Birleştirme Kısıtı:** İlgi'nin çalışma yerini tanımlar. Uygulama içerisindeki enine kesen ilgilerden etkilenen sınıfların ve metodların listelerini belirler. İcra Noktası seviyesinde tanımlanır (Sapir vd., 2002).

4.3 İlgiye Yönelik Programlama Fazında UML Kullanımı

İlgiye Yönelik Programlama Yaklaşımı, Nesneye Yönelik programlama Yaklaşımının eksiklerini tamamlayan bir yazılım geliştirme yöntemi olduğundan dolayı, NYP'de kullanılan UML modeli, İYP'de de kullanılabilmelidir. İYP yaklaşımında UML modellemesi incelenmeden önce standart NYP'deki UML modelinin incelenmesi daha yararlı olacaktır (Kandé vd., 2002).

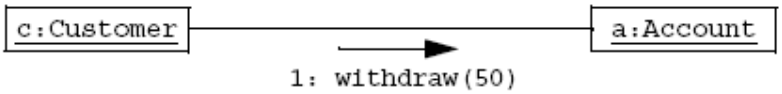
4.3.1 Nesneye Yönelik Yaklaşımla UML Kullanımı

Birçok yazılım projesi ile örneklenebileceği gibi, Nesneye Yönelik Programlama Yaklaşımı, UML dili tarafından desteklenmektedir. Örneğin, basit bir banka sistemi düşünüldüğünde Hesap ve Müşteri sınıfları incelenirse ilgili kodlar aşağıdaki şekildedir:

```
public class Account {
    private int balance = 0;
    public void withdraw (int amount) {...};
    public void deposit (int amount) {...};
    public int getBalance() {...};
  }
  public class Customer {
    public String name;
    // bazı metodlar içinde (a bir hesap nesnesidir)
    a.withdraw(50);
  }
```

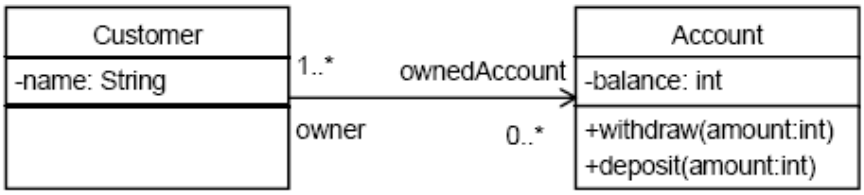
Yukarıda örneği görülen, Java ile yazılmış kodların UML ile modellenmesinde 2 tip diyagram kullanılabilir. Bunlar İşbirliği Diyagramı ve Sınıf Diyagramı'dır. Karmaşık sistemler, tasarlanan görevleri yerine getirmek için ortaklaşa çalışan birçok nesneden oluşmaktadır. UML içindeki İşbirliği Diyagramları temel olarak bu nesneler arasındaki ilişkiye yoğunlaşırlar. Şekil 4,8'de, yukarıdaki kodlara ait İşbirliği Diyagramı görülmektedir. Bu diyagramda bir Müşteri (Customer) nesnesinin Hesap (Account) nesnesi ile etkileşimi

görülmektedir. İşbirliği diyagramları genel olarak nesnelerin etkileşimleri ve birlikte çalışmaları hakkında fikir sahibi olunmasını sağlarlar. Nesneler arasındaki mesajları ve nesnelerin karakteristik özelliklerini gösterirler. Örnekte c nesnesi, a nesnesinin Para Çekme (withDraw) metodunu çağırmaktadır (Kandé vd., 2002).



Şekil 4.8 : UML işbirliği diyagramı

İşbirliği diyagramı örneğinde görüldüğü gibi, bu diyagramlarda ayrıntılı olarak nesnelerin nasıl haberleştiği ve birbirlerine nasıl bağlı oldukları bilgileri bulunmamaktadır. Bu tip bilgiler UML modelinde Ardışıl Diyagramları ile gösterilebilirler. Bununla birlikte, yazılım tasarım fazında bu tip ayrıntıların (nesnelerin bağlantı tipleri ve aralarındaki ilişkiler) belirtildiği diyagramlardan olan sınıf diyagramları da kullanılabilir. Şekil 4.9'da örnek kodların (Hesap ve Müşteri) sınıf diyagramları görülmektedir.



Şekil 4.9 UML sınıf diyagramı

Şekil 4.8'de ilişkili nesnelerin birbiri ile hangi metodlar üzerinden haberleştiği görülmektedir. Şekil 4.9'da görülen UML sınıf diyagramı ile de bağlantının her iki ucuna rol isimleri verilebilmesine olanak sağlanmıştır. Banka sistemleri geliştikçe, gereksinimler de artabilir veya değişebilir. Yazılım geliştiricilerden aşağıdaki gereksinimleri karşılayacak çeşitli değişikliklerin ve eklemelerin yapılması istenebilir. Örneğin, "Hesap nesnesine yapılan her erişimin günlüğü tutulmalıdır", "hesaba erişen müşterinin adı ve erişim tipi sisteme kaydedilmelidir" gibi. Burada belirtilen günlük tutma işlemi tipik bir enine kesen ilgidir. Standart bir nesneye dayalı programlama modelinde bu enine kesen ilginin gösterimi kolay değildir. Çünkü bu işlem, sistemin genelindeki hesap işlemlerinin geçtiği tüm metodlara yayılmaktadır. Bu tarz bir enine kesen ilginin sisteme eklenmesinin bir yolu da İlgiye Yönelik Yazılım Geliştirme yaklaşımıdır (Kandé vd., 2002).

4.3.2 İlgiye Yönelik Yaklaşımla UML Kullanımı

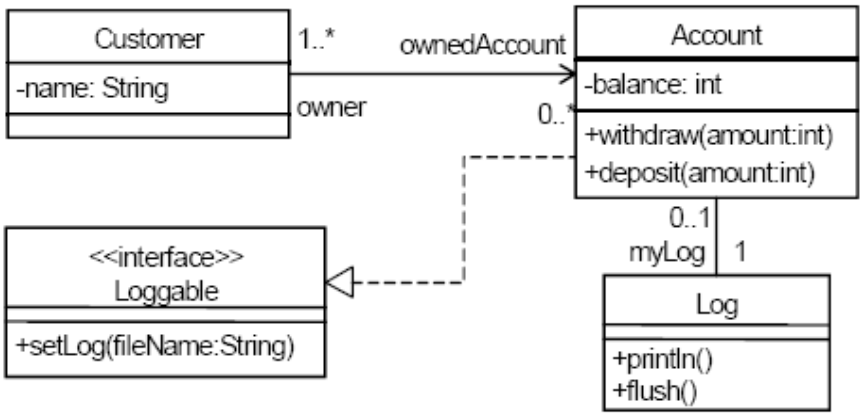
Bu bölümde, bir önceki bölümde incelenen NYP yaklaşımı ile UML kullanım örneğinden sonra, bu örnek üzerinde yapılacak olan eklemeler ve değişiklikler için İYP kullanılacaktır. İYP'nin temel öğelerinden biri olan Birleşme Noktaları, genelde Metod Çağrıları şeklinde tanımlanmaktadır. İcra Noktası tanımlayıcısı ise bir veya birden fazla birleşme noktasının icra etmesine olanak sağlar. İcra noktaları tarafından çağrılan metodlar içinde metod belirteçleri kullanılmaktadır. Bir icra noktası tanımlaması şu şekildedir :

pointcut isim : call(metod_Belirteci)

İcra Noktası yapısında, birleşme noktası ile belirlenmiş olan metodların belli parametre değerleriyle aynı icra noktasının tetiklenmesi ve gerekli ilgi kodunun icra ettirilmesi sağlanabilmektedir. Örnekte Günlük Tutma İlgisinin, Hesap nesnesi üzerinde yapılan tüm işlemlerden sonra icra etmesini sağlayacak ilgi kodu mevcuttur.

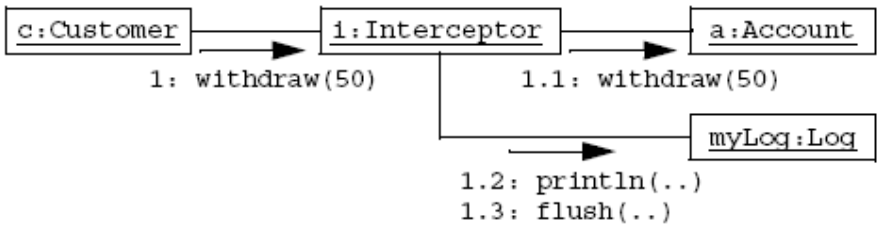
```
public aspect Logging {

private PrintWriter Account.myLog;

public void Account.setLog(String fileName) {

        myLog = new Log(fileName);

        myLog.println("This is the logfile for account " + this);

}

pointcut MethodCall(Customer c, Account a) :

       call (public * Account.*(..))&& this(c) && target(a);

after (Customer c, Account a) : MethodCall(c, a) {

       a.myLog.println(c + " called " + thisJoinPoint.getSignature().getName());

        a.myLog.flush();

    }

}
```

Günlük Tutma İlgisi, Hesap sınıfı içerisine setLog() adında yeni bir metod tanımlamıştır. Bu yeni metod (İç Tip Tanımlama Örneği), Hesap nesnesinin, günlük tutma bilgilerini bir metin dosyasına yazan PrintWriter metodu ile birleştirmiştir. Log referansı ise myLog özelliğinde tutulmuştur. Bu değişikliğin gösterildiği UML sınıf diyagramı Şekil 4.10'da görülmektedir (Kandé vd., 2002).



Şekil 4.10 Hesap günlük tutma ilgisi sınıf diyagramı

Sisteme günlük tutma ilgisi eklendikten sonraki nesnelerin birbirleriyle etkileşimini gösteren, Şekil 4.8'deki İşbirliği Diyagramının genişletilmiş hali olan İşbirliği Diyagramı Şekil 4.11'de görülmektedir (Kandé vd., 2002).



Şekil 4.11 İşbirlik diyagramı

Kesişen metod çağrıları, UML işbirlik diyagramları ile modellenirken kesişen nesneler arasındaki iletişimin sağlanması için Kesişme ara sınıfları kullanılmaktadır. Kesişme nesnesi(i) Müşteri ve Hesap nesneleri arasına yerleştirilmiştir. Bu nesnenin yeni eklenen Günlük Tutma metodunu çağırabilme özelliğinin yanında:

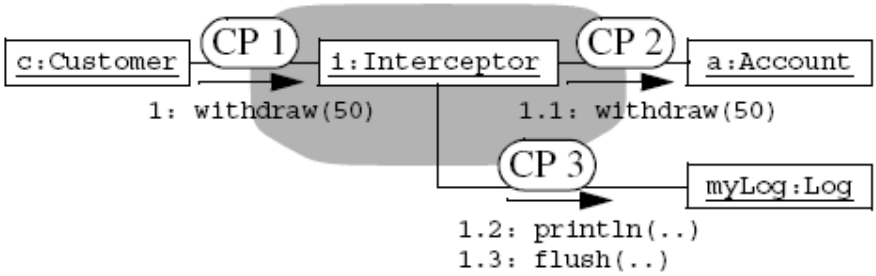
- Müşteri nesnesi tarafından çağrılan Hesap nesnesinin tüm metodlarını içeren bir arayüz sunması
- Kesişmiş metod çağrılarını Hesap nesnesine yönlendirecek bir mekanizma sağlaması

gerekmektedir.

Hesap nesnesini direk çağırmak yerine, Müşteri nesnesi Kesişme nesnesini çağırmaktadır (Şekil 4.11). Örnek uygulamada günlük tutma işlemi için "sonra" icra tipi kullanılmaktadır. withdraw(50) mesajı önce Hesap nesnesine yönlendirilmektedir. Bu metod çalıştırıldıktan sonra Günlük Tutma metodu çalıştırılmakta ve sistem gereksinimi karşılanmaktadır.

4.3.3 UML'in İlgiye Yönelik programlama için Genişletilmesi

İlgiye Yönelik Yazılım Geliştirme ile geliştirilmiş sistemler, enine kesen ilgilerin sistemin fonksiyonel ilgileri ile entegre çalışmasından oluşmaktadır. Bu programlama yaklaşımında belli çalışma koşullarında belli bir anda belli bir işin yapılması esastır. Bu gereksinimin sağlanabilmesi, sistemin değişik bakış açılarından anlaşılmasını ve modellenmesi ihtiyacını ortaya çıkarır. Şu anda kullanılan modelleme tekniklerinin eksikliklerini gidermek amacıyla geliştirilmiş ilgiler, programlama dillerinde UML model elemanları olarak tanımlanabilmesi gerekmektedir. Örnek olarak daha önce bahsedilen günlük tutma uygulaması verilebilir. Günlük Tutma ilgisinin tek bir modül içerisinde tanımlanıp gösterilebilmesi amacıyla ara bir birim olan Kesişme nesnesi tanımlanmıştır. Bu nesnenin diyagramdaki temel amacı Müşteri ve Hesap nesneleri arasındaki etkileşimin gösterilebilmesidir (Kandé vd., 2002).



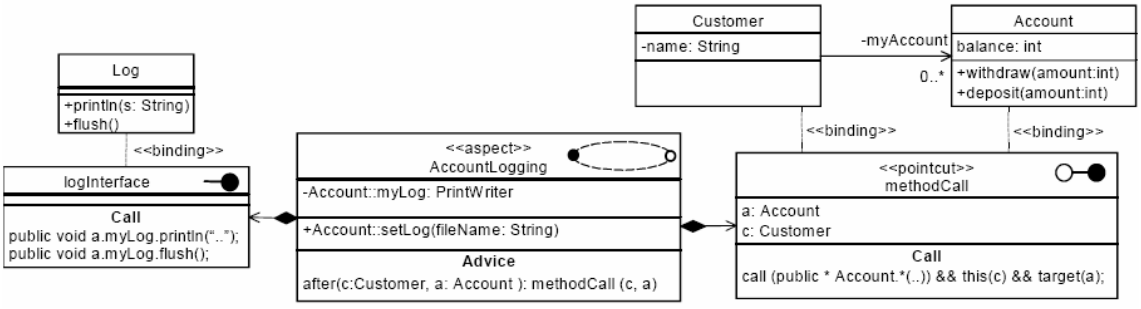
Şekil 4.12 Bağlantı noktaları tespiti

Şekil 4.12'de gri arkaplan ile gösterilen Kesişme nesnesi Günlük Tutma ilgisinin davranışlarını ve etkileşimini göstermek üzere diyagrama eklenmiştir. Bu aşamada ise gri arkaplanlı kısım birinci sınıf bir eleman olarak varsayılacak ve bu ilgiyi oluşturması gereken elemanlar belirlenecektir.

C, a ve myLog bileşenlerinin ilginin bir elemanı olmadığı açıkça görülmektedir. Çünkü bunlar hesaplamalarını gri bölgenin dışında gerçekleştirmektedirler. Kesişme nesnesinin rolü bu nesneler arasındaki etkileşimin sağlanması ve gerekli yönlendirmelerin yapılmasıdır.

Kesişme nesnesi, ilginin bir parçası olmalıdır. Çünkü bu nesne sistem bileşenleri arasındaki bütünlüğün sağlanmasında görev almaktadır. Bu nesnenin aktif olduğu noktalar birleşme noktaları (CP1, CP2, CP3) ile gösterilen noktalardır. Kesişme nesnesi yönünden bakılırsa CP1 gelen yöndeki bir bağlantı noktasıdır. Kontrol akışı ilgiye dışarıdan gelmektedir. CP2 ve CP3 ise giden yöndeki bağlantı noktalarıdır. Günlük Tutma işleminin gerçekleşebilmesi için ilgi, Günlük Tutma (Log) sınıfının bir kopyasına ihtiyaç duyar. Çünkü günlük tutma işleminden

beklenen kayıt ve günlük tutma metodları bu sınıf içerisinde bulunmaktadır. Bu işlem CP3 bağlantı noktası ile gerçekleştirilmektedir. Bağlantı noktalarının gösteriminin açıkça yapılabilmesi için ilgiler tek başına bir anlam ifade edecek şekilde tanımlanabilmeli ve işbirlik diyagramı içerisinde gösterilebilmelidir. Bunlar enine kesen ilgilerin diğer fonksiyonel ilgilerden ayrıştırılmasını belirler nitelikte olmalıdırlar (Kandé vd., 2002).



Şekil 4.13 İşbirliği diyagramı ile ilgiye yönelik günlük tutma ilgisinin gösterimi

Şekil 4.13'de Günlük Tutma ilgisinin gösterimi için 4 tip eleman kullanılmıştır. İlginin kendisi, birleşme noktaları, UML sınıfları ve birleştirme (binding) ilişkisi. İlginin kendisi, birleşme noktalarında yapılacak olan aksiyonları belirtmek üzere, kesikli çizgili bir ovalle işaretlenmiştir. Örneğin şekilde ilk iki eleman ilginin günlük tutma ile ilgili özelliklerini göstermektedir. Bunlardan biri özellik diğeri ise bir metoddur. Bunlar çalışma zamanında Hesap nesnesi ile entegre edilmelidirler. Çünkü her hesap hareketinin günlüğünün tutulması beklenmektedir.

İcra edilecek kod (Advice) kısmı ise hesap nesnesinin ilgili metodu çağrıldıktan sonra günlük tutma metodunun icra edeceğini bildirmektedir. Yine Şekil 4.13'de gösterilen bağlantı noktaları içi beyaz ve siyah olan dairelerle gösterilmektedir. Beyaz daireler, gelen bağlantı noktalarını siyah daireler ise çıkan bağlantı noktalarını göstermek için kullanılır. Bağlantı noktası, ilgiler içerisindeki icra noktalarına karşılık gelmektedir. Fakat diyagramda nesnelerin etkileşim noktaları olarak gösterilmektedir. Birleşme noktaları bir ilginin arayüzü ve sınıflarını oluşturmak için kullanılır. Örnekte gösterilen birleşme noktaları, logInterface ile methodCall 'dur.

LogInterface birleşme noktası Şekil 4.12'de görüldüğü gibi çıkış bağlantı noktasıdır. Bunun anlamı, ilgi çalışma zamanında bunu kullanmaktadır. Bu aynı zamanda ilginin çalışması için dışarıdan nelere gereksinim duyduğunun bilgisini de içerir. Uygulamada bu bir Java arayüzüne veya soyut bir sınıfa karşılık gelmektedir. Özellikleri (değişkenleri veya metodları) Call kısmının bulunduğu bölümde gösterilmiştir (Şekil 4.13).

MethodCall birleşme noktası ise birbirine ters çalışan iki bağlantı noktasını gruplayan özel bir bağlantı noktasıdır (CP1 ve CP2). Uygulamadaki kullanımında direk olarak bir icra noktasına karşılık gelmektedir. İcra noktasının tanımı da Call bölümünde görülmektedir (Şekil 4.13). Dokuma zamanında ilgiler tarafından gereksinim duyulan bağlantı noktaları nesnelere bağlanır. Birleşme (binding) ilişkisi, ilginin hangi nesnelere bağlanabileceğini göstermektedir. Verilen günlük tutma örneğinde ilginin, Müşteri(Customer) , Hesap (Account) ve Günlük Tutma(Log) nesnelerine bağlanması gerektiği görülmektedir. Bu nedenle örnekte Günlük Tutma ilgisi dokuma anında müşteri c sınıfını hesap a ve myLog adı verilmiş olan Günlük Tutma nesnesine bağlamaktadır (Kandé vd., 2002).

5. YAZILIM PROJELERİNDE İYP ARAÇLARININ KULLANIMI

İlgiye Yönelik Programlama, kurumsal uygulamalarda geniş kullanım alanına sahip olmayan fakat yaygınlaşmaya başlayan bir mühendislik yaklaşımdır. İlgiye Yönelik Programlama metodolojisini destekleyen birçok araç mevcuttur. Fakat hangi aracın geliştirilecek olan uygulamalar için en uygun olduğunun seçimi konusunda bazı sıkıntılarla karşılaşılmaktadır. Gerek uygulamanın çalışma zamanı performansı, gerekse geliştirme zaman maliyetleri göz önünde bulundurularak geliştirme yapılacak aracın seçimi çok önemlidir. İlgiye Yönelik Programlama araçlarının çeşitli yönlerden kıyaslanması da projelerde kullanılacak uygun aracın seçilebilmesi açısından çok önemlidir. Çünkü bu süreç, tüm yazılım geliştirme yaşam döngüsünü etkilemektedir.

İlgiye Yönelik Programlama yaklaşımı tüm yazılım geliştirme dilleri için uygundur fakat Java en fazla kullanım alanı bulan platformdur. Bu nedenle, çalışma süresince Java dilini kullanan İYP araçları incelenecek ve bunların çeşitli yönlerden karşılaştırılmaları yapılacaktır.

Günümüzde en çok kullanılan İYP araçları AspectJ, AspectWerkz, JBoss AOP ve Spring AOP'dir. Bu araçların her biri Java kod yazım stilini temel alan açık kaynak kodlu araçlardır. Bu araçların İYP uygulama yöntemleri de farklı olduğundan dolayı, yapılacak olan karşılaştırma aslında farklı İYP yaklaşımlarının karşılaştırılması ve projelerle adaptasyonudur.

5.1 İYP Araçları Giriş

Günümüzde en çok kullanılan açık kaynak kodlu İYP araçlarının ilk sürümlerinin çıkış tarihleri aşağıda listelenmektedir:

- **AspectJ** -- 2001 yılında AOP Xerox PARC İYP ekibi tarafından ilk sürümü çıkarılmıştır. Şu an "eclipse.org" web sayfasından erişilebilmekte ve IBM tarafından desteklenmektedir. Günümüzdeki en son sürümü : 1.5.3
- **AspectWerkz** – 2002 yılında ilk sürümü çıkarılmış ve BEA Systems tarafından desteklenmektedir. Günümüzdeki en son sürümü : 2.0RC3.
- **JBoss AOP** – 2004 yılında JBoss Uygulama sunucu çatısına eklenti olarak geliştirilmiş ve ilk sürümü çıkarılmıştır. Günümüzdeki en son sürümü: 1.5.5
- **Spring AOP** -- 2004 yılında Spring çatısına bir eklenti olarak geliştirilmiştir. Günümüzdeki en son sürümü : 1.2.6

5.1.1 Birleşme Noktaları Yaklaşımı

Tüm İYP araçlarının uygulama geliştirme mantığında, birleşme noktası modeli ve uygulamanın icra noktalarının açıkça tanımlandığı program mekanizması bulunur. Araçların İYP birleşme noktası modelini uygulama biçimlerinde benzerlikler olsa da, her bir aracın İYP uygulama mekanizma yaklaşımları aralarındaki farkların anlaşılması uygulama geliştirme maliyetleri ve çalışma zamanı performansı açısından önemlidir.

İYP araçları, yetkilendirme ve günlük tutma gibi enine kesen ilgilerin ayrıştırılması temeli üzerine kurulmuşlardır. Nesneye Yönelik Programlama ile böyle bir uygulama geliştirildiğinde enine kesen ilgiler kod içerisine yayılmak durumunda kalmaktadır. Bu şekilde bir yayılım, programların yönetilebilirliği ve gelişimi açısından zorluk yaratmaktadır.

Tüm İYP araçlarında enine kesen ilgilerin icra noktalarının belirlendiği birleşme noktası mekanizması bulunmaktadır. Birleşme noktası ana program ile ilginin program içinde kesiştikleri yerdir. Statik birleşme noktaları ilgilerin bir sınıf içerisinde yeni üyeler olarak tanımlanmasına olanak sunar. Dinamik birleşme noktaları ise programın işleyişi esnasında icra edilecek ilgi kodunun başlangıç noktasını belirtir. Örneğin standart Java uygulamaları metod çağrıları veya alan değer atamaları gibi birleşme noktalarını icra ederler.

5.1.2 İcra Noktaları, İcra Edilecek Kodlar ve İç Tip Tanımlamaları

İYP araçları birleşme noktaları ile belirtilen özel noktalara gelindiğinde, ilgili metodun icrasının durdurulmasını ve icra noktasında belirtilmiş ilgi kodunun çalıştırılmasını sağlayan mekanizmalar bulundururlar. Bu mekanizmaya icra noktası mekanizması denir. Buna ek olarak bu araçlarda iç tip tanımlama mekanizması da bulunur. Bu mekanizma, mevcut sınıflarda, bu sınıfların yapıları değiştirilmeden, var olan tipler üzerinde ek üyeler tanımlama olanağı sağlar. İcra noktaları, icra edilecek kodlar ve iç tip tanımları birlikte İYP dillerinin enine kesen ilgi problemine çözüm getirmesini sağlarlar. Bir "ilgi" standart java alan ve metodlarına ek olarak bu 3 üyeyi de içeren iyi modülerize edilmiş bir yapıdır. Bu bölümde İYP yaklaşımının değişik dillerde, değişik yaklaşımlarla uygulamaları incelenecek ve karşılaştırmaları yapılacaktır. Her yaklaşımın temelinde birleşme noktalarının erişim, birleştirme, isimlendirme ve soyutlamaları incelenmektedir.

Erişim: İYP yaklaşımı kullanılarak geliştirilmiş bir uygulamada, birleşme noktası olarak belirlenmiş metodlara gelindiğinde icra noktası ile belirtilen ilginin tetiklenmesini sağlayan durumdur.

Birleştirme: İYP yaklaşımında icra noktası desteği, basit icra noktalarının daha karmaşık olanlar içine toplanmasını sağlar. Kontrol akışı şeklinde olan metodlarda bu yaklaşım kullanılır. Bir metod içerisinde birden fazla ilgili metod çağrısı durumunda, icra noktaları birden fazla birleşme noktasının bir icra noktası içinde toplanmasına olanak sunar.

İsimlendirme: İcra noktalarının isimlendirilmesi, okunabilirlik ve birleştirmeyi kolaylaştırır.

Soyutlama: İYP dillerindeki soyutlama desteği, uygulamadan bağımsız ortak kütüphanelerin oluşturulmasına imkan verir (Kersten, 2005).

5.2 İlgi Yazım Kıyaslamaları

İlgiye Yönelik Programlama araçları, birleşme noktaları, icra edilecek kodlar, bu kodların icra edeceği yerler ve iç tip tanımlarından oluşan mekanizmalardır. İYP'nin uygulandığı araçların dayandığı temel ilke de budur. Bu araçlar arasındaki en büyük farklılık ilgi tanımlamalarının yapılması ve uygulama içerisinde uygulanmasıdır.

İYP araçlarının 3 değişik ilgi tanımlama yaklaşımı bulunmaktadır, bunlar Java benzeri kodlar, dipnotlar ve XML'dir. Bazı uygulama geliştiriciler, standart java diliyle yazılmış olan ilgileri, programlama dillerini genişleten yazılımların yüklenmesini gerektiren ilgilere tercih ederken, diğer bir kısım uygulama geliştiriciler ise dipnotlar ve XML yapıları ile geliştirilen ilgileri, icra noktaları ile çalışmanın zorluklarına tercih etmektedirler.

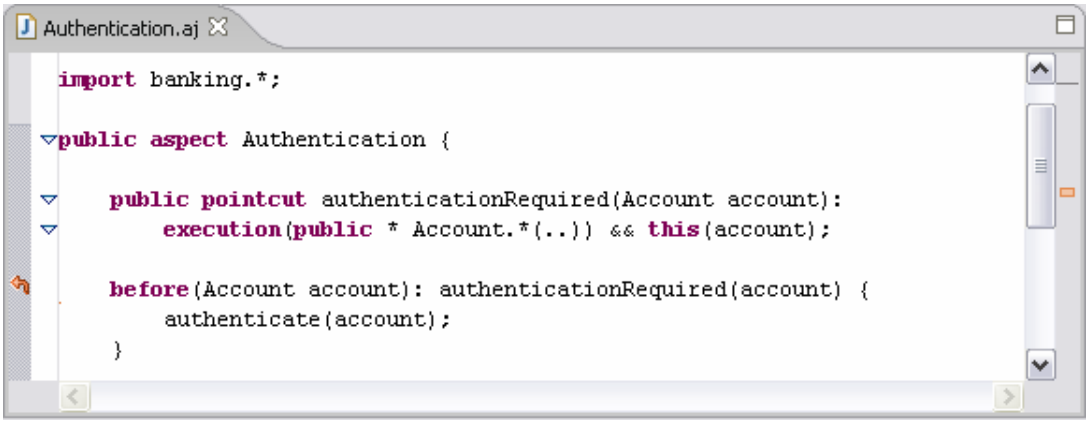
Tüm araçların ilgi tanımlama yaklaşımları arasındaki farkların rahatça görülebilmesi için ortak bir örnek üzerinde inceleme yapılması daha uygundur. Daha önceki bölümlerde örneği verilen bankamatik uygulamasındaki Hesap (Account) sınıfına eklenecek "Yetkilendirme politikası"'nı ele aldığımızda, bu araçların ilgi tanımlama yaklaşımları karşılaştırmalı olarak incelenebilir.

Nesneye Yönelik Programlama yaklaşımındaki yetkilendirme örneklerinde, yetkilendirme ile ilgili kodların yetkilendirme gerektiren tüm sınıfların içerisine yayıldığı görülmektedir. İlgiye Yönelik Programlama uygulamasında ise yazılacak olan bir "Yetkilendirme" ilgisi, mevcut uygulama sınıflarındaki kodları değiştirmeden o kodlarla birlikte çalışarak gerekli özelliği uygulamaya katmaktadır. İlgi, tanımlandığı araçtan bağımsız olarak aşağıdaki bileşenleri içermelidir :

- İcra Noktası: "banking.Account" sınıfındaki tüm genel metodların icralarını kapsar.
- Hesap sınıfının yetkilendirildiğine dair bir belirteç
- İcra Edilecek Kod: Birleşme Noktalarının, İcra Noktası ile belirtilmiş bölümüne gelince uygulamaya eklenecek olan işlevsellik kodunu içeren yapıdır. İncelenen örnekte uygulamaya eklenecek olan Yetkilendirme kodu icra edilecek olan koddur.

5.2.1 AspectJ ile İlgi Geliştirme

AspectJ, Java programlama dilinin ilgi özellikleri ile genişletilmesiyle ortaya çıkmış bir İYP dilidir. Bu dilde ilgiler tanımlanırken kendi yazım diline özel tanımlama komutları kullanılır. Standart alan ve metodlara ek olarak İlgi tanımları, İcra Noktaları ve İcra Edilecek Kodları içerir. Örnekteki icra noktası (Şekil 5.1) tüm genel metodları belirtmek için kendine özel ifadeler kullanmaktadır. Hesap nesnesine erişim, icra noktasını belirten "pointcut" anahtar kelimesi ile sağlanmaktadır. Buradaki ifade "Hesap" nesnesinin tüm metodlarını kapsayan bir etkiye sahiptir. İcra edecek kodun yapısı normal bir java metodunun yapısına çok benzerdir. Bu kod, yetkilendirme kodu içerebilir veya farklı bir metodu çağırabilir. AspectJ kullanılarak tanımlanmış Yetkilendirme ilgisi Şekil 5.1'de görülmektedir.



Şekil 5.1 AspectJ ile geliştirilmiş yetkilendirme ilgisi [3]

Bir AspectJ uygulaması geliştirmek, standart bir Java uygulaması geliştirmeye çok benzerdir. Uygulama yapılandırması, tüm projeyi, standart java kodları ve ilgi kodları ile birleştiren AspectJ derleyicisinin çağrılmasından ibarettir. AspectJ projesinin icra etmesi, standart java projesinin icra etmesinden biraz farklıdır. Çünkü "aspectjrt.jar" kütüphanesinin projenin sınıf yoluna (classpath) eklenmiş olması gerekmektedir. AspectJ aracı kullanılarak uygulama geliştirilmesi ve geliştirilmiş olan ilgilerin standart uygulama ile birleştirilebilmesi için dokuyucu mekanizmayı içeren ve birleştirilmiş uygulamanın derlenmesini sağlayan AspectJ derleyicisinin geliştirme ortamına yüklenmesi gerekmektedir.

5.2.2 AspectWerkz ile İlgi Geliştirme

AspectWerkz ile İYP uygulaması geliştirilirken enine kesen ilgiler standart java sınıfları olarak ifade edilirler. AspectWerkz'i AspectJ'den ayıran en temel fark budur. AspectJ dışındaki diğer araçlar ile uygulama geliştirilirken standart java yazım dili kullanılmaktadır. Bu nedenle ilave bir derleyici kurulumuna gerek kalmamaktadır. AspectWerkz ile İYP uygulaması iki farklı şekilde geliştirilebilir. Bunlardan ilki Java 1.5 ile gelen dipnot stilidir. Diğeri ise XML konfigürasyon dosyaları kullanılarak yeni işlevlerin eklenmesidir. Bu yapıda, ilgilerin ayrı bir XML dosyasında tanımlanması söz konusudur. Örneği incelenmekte olan Yetkilendirme ilgisinin AspectWerkz ile geliştirilmesi Şekil 5.2'de görülmektedir.

Authentication.java

```
import org.codehaus.aspectwerkz.definition.Pointcut;

public class Authentication {
    @Expression("execution(public * banking.Account.*(..))")
    Pointcut authenticationRequired() {
        return null;
    }

    @Before("this(account) && authenticationRequired")
    public void before(Account account) throws Throwable {
        authenticate(account);
    }
```

aop.xml

```
<!DOCTYPE aspectwerkz PUBLIC "-//AspectWerkz//DTD//EN" "http://aspectwerkz.
<aspectwerkz>
    <system id="banking.example">
        <aspect class="banking.Authentication"/>
    </system>
</aspectwerkz>
```

Şekil 5.2 AspectWerkz ile yetkilendirme ilgisinin geliştirilmesi [3]

Burada, "İcra edilecek olan kod" standart bir java sınıf tanımıdır. Geleneksel olarak farklı bir sınıf tipi olması beklenirken, açık bir şekilde kod içerisinde çağrılmaz, bir icra noktasına gelindiğinde otomatik olarak icra eder. AspectWerkz'de icra noktaları tanımlamaları "Pointcut" metodlarına ilişkilendirilmiş metin değerlerdir veya kendi başına ayrı bir XML dosyasında bulunurlar. Bunun sonucu olarak icra noktaları için bir içeri alma (import) mekanizması bulunmaz ve tüm tip tanımlamaları açık bir şekilde yapılır.

Bir AspectWerkz uygulaması geliştirmek, standart bir java yapılandırma işlemi gerektirir. AspectWerkz programını çalıştırmak için AspectWerkz kütüphanelerinin proje içerisine dahil edilmesi veya bunların yerlerinin proje içerisinde belirtilmesi gerekir. "aop.xml" dosyası sisteme dahil edilecek ilgilerin belirlenmesini sağlar.

5.2.3 JBoss AOP ile İlgi Geliştirme

JBoss AOP, XML tabanlı ilgi tanımlama mekanizması içeren bir İYP dilidir. Şekil 5.3'te örnek uygulama için geliştirilen yetkilendirme sınıfının JBoss AOP ile gerçekleştirimi görülmektedir. Bu İYP aracı, AspectWerkz dilinde olduğu gibi dipnot şeklinde ilgi tanımlamalarını da desteklemektedir.

Authentication.java

```
import org.jboss.aop.joinpoint.MethodInvocation;
import org.jboss.aspects.asynchronous.aspects.jboss.TraceThreadAspect;

public class Authentication extends TraceThreadAspect {

    public Object beforeAuth(MethodInvocation method) throws Throwable {
        authenticate((Account)method.getTargetObject());
        return method.invokeNext();
    }
```

jboss-aop.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<aop>
    <aspect class="banking.Authentication" scope="PER_VM"/>
    <bind pointcut="execution(public void banking.Account-&gt;*(..))">
        <advice aspect="banking.Authentication" name="beforeAuth"/>
    </bind>
</aop>
```

Şekil 5.3 JBoss AOP ile yetkilendirme ilgisinin geliştirilmesi [3]

XML stilinde ilgi geliştirmede icra noktası ve birleşme noktası tanımlamalarının hepsi XML içinde yapılmaktadır. İcra edilecek kod JBoss AOP tarafından standart java metodları şeklinde uygulanmaktadır. "Hesap" sınıfını açıkça birleştirmektense JBoss AOP, bu nesnenin o an icra etmekte olan metodlarına erişir.

5.2.4 Spring AOP ile İlgi Geliştirme

Önceki bölümlerde anlatılan diğer araçlara kıyasla, Spring AOP ile ilgi geliştirilmesinde XML'in rolü daha büyüktür. Bu araç, Spring çatısı tarafından çağrılan özel parametreler

içeren bir araçtır. XML'de "accountBean" tanımlaması yapılmaktadır (Şekil 5.4). Bu tanımlama Spring çatısının "Hesap" nesnesine erişimini sağlamaktadır. Bu XML yapısı içerisinde icra edecek kod durdurucusu, kodu çalıştıracak yapı ve ilgiyi gerçekleyecek "önce" icra zamanı kodu bulunmaktadır.

Spring AOP ilgilerinin yapılandırılma, icra etme ve ayarlama prosedürleri JBoss AOP ile aynıdır fakat Spring AOP, ilgileri Spring çatısının pratik ve kısa çalışma zamanı yapılandırmasına dayanır ve bu araçta ayrıca bir yapıya ihtiyaç duyulmaz.

Yetkilendirme ilgisinin Spring AOP aracı ile geliştirilmiş örneği Şekil 5.4'te görülmektedir.

Authentication.java

```
import java.lang.reflect.Method;
import org.springframework.aop.MethodBeforeAdvice;

public class Authentication implements MethodBeforeAdvice {

    public void before(Method m, Object[] args, Object target)
        throws Throwable {
        authenticate((IAccount)target);
    }
```

springconfig.xml

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE beans PUBLIC  "-//SPRING//D
<beans>
    <bean id="accountbean" class="org.springframework.aop.framework.ProxyFa
        <property name="proxyInterfaces">           <!-- CONFIG -->
            <value>banking.IAccount</value>
        </property>
        <property name="target">
            <ref local="beanTarget"/>
        </property>
        <property name="interceptorNames">
            <list><value>authenticationAdvisor</value></list>
        </property>
    </bean>
    <bean id="beanTarget" class="banking.Account"/> <!-- CLASS  -->
    <bean id="authenticationAdvisor" class="org.springframework.aop.support
        <property name="advice">                    <!-- ADVISOR-->
            <ref local="authenticationBeforeAdvice"/>
        </property>
        <property name="pattern"><value>.*</value></property>
    </bean>                                         <!-- ADVICE -->
    <bean id="authenticationBeforeAdvice" class="banking.Authentication"/>
</beans>
```

Şekil 5.4 Spring AOP ile yetkilendirme ilgisinin geliştirilmesi [3]

5.2.5 Yazımsal Farklılıklar ve Stilin Belirlenmesi

Önceki bölümlerde, tüm dillerde ilgi tanımlamalarının nasıl yapıldığı incelemiştir. Buradan çıkarılacak sonuç, İYP araçları arasındaki en büyük farklılık ilgi yazımlarından kaynaklanmaktadır.

AspectJ, ilgilerin tümüyle kod içerisinde tanımlanmasına imkan veren Java diline uzantı olarak geliştirilmiş bir dildir.

AspectWerkz ve JBoss AOP, kod içerisindeki dipnotlarla ve ayrı XML dosyaları ile ilgilerin tanımlanmasına olanak sunan dillerdir.

Spring AOP'de ise ilgilerin ayarları tümüyle XML dosyasında tanımlanmaktadır.

Araçların ilgi tanımlama yöntemleri incelendiğinde, İYP'de ilgilerin 3 farklı yaklaşım ile gerçekleştirilebildiği sonucu ortaya çıkmaktadır.

Her yaklaşımın geliştirilecek uygulamaya göre avantajları ve dezavantajları vardır. Bu nedenle gelitirilecek uygulamanın gereksinimleri ve özellikleri detaylı analiz edilmeden geliştirme yapılacak araca karar verilmemelidir.

Hangi İYP aracı kullanılacağından bağımsız olarak, geliştirilen icra edilecek ilgi kodları ile çalışmak, temelde standart Java kodu ile çalışmaktır. Araçlar arasındaki farklılıklar, icra noktalarının tanımlandığı yerlerde ortaya çıkmaktadır.

XML stilinde çalışılırken (Spring AOP'de olduğu gibi). XML içerisinde bir icra noktasında bir değişiklik yapma, icra edilecek kodu içeren Java dosyasından, ilgili icra noktası tanımının olduğu XML dosyasına gitmeyi gerektirmektedir. Bu işlem, tüm ilgilerin uygulama dışındaki bir dosyadan (XML) yönetilebilmesi avantajını sağlarken, birçok ilgi içeren büyük bir sistemde java dosyaları ile XML dosyası arasında gidip gelmek problemlere ve karmaşıklıklara yol açabilmektedir.

Dipnotların kullanıldığı stil tercih edildiğinde (AspectWerkz veya JBoss AOP araçlarında olduğu gibi) icra noktalarının XML'den Java sınıfları içerisine dipnot olarak eklenmesi imkanı vardır. Bu yapı, icra edilecek ilgi ve icra noktaları üzerinde eşzamanlı olarak kolaylıkla çalışılabilme olanağı sunar.

AspectJ stili tercih edilirse, icra noktası ve icra edilecek olan kodlar üzerinde çalışmanın standart java kodu ile çalışmaktan farklı olmadığı görülür. Bir Java kodundan beklenen herşey (örneğin içeri alma) icra noktaları için de çalışacaktır.

Çizelge 5.1 İYP araçlarının bileşenler yönünden karşılaştırılması [3]

<table>
<tr><th></th><th>İlgi Tanımlamaları</th><th>İç Tip Tanımlamaları</th><th>İlgi Kodları</th><th>İcra Noktaları</th><th>Statik Yazım</th><th>Konfigürasyon</th></tr>
<tr><td>AspectJ</td><td colspan="3">Kod</td><td></td><td>Hata/Uyarı tanımlama</td><td>.lst içerik listesi</td></tr>
<tr><td>Aspect Werkz</td><td colspan="2" rowspan="2">Dipnot veya XML</td><td rowspan="3">Java metodu</td><td rowspan="3">Metin Değeri</td><td rowspan="3">-</td><td>aop.xml</td></tr>
<tr><td>JBoss AOP</td><td>jboss-aop.xml</td></tr>
<tr><td>Spring AOP</td><td colspan="2">XML</td><td>springconfig.xml</td></tr>
</table>

İYP araçları ile çalışılırken ilgi tanımlama stilleri geniş bir etkiye sahiptir. Örneğin XML dosyası içinde ilgi tanımlanmasını destekleyen araçlar, aynı XML dosyasında tanımlanan ilgilerin sisteme nasıl etki edeceklerinin de tanımlanmasına imkan verir. Uygulamaların geliştirilmesinde ilgi kodları ve icra noktaları tanımlamalarının yanında, iç tip tanımlamaları da seçilmiş olan aracın İYP geliştirme stilinden etkilenir. AspectJ aracı ile iç tip tanımlamaları, normal java sınıflarına metod eklemek veya yeni tip tanımlamaktan farklı değildir. Diğer yaklaşımlarda ise yeni metodlar, dipnotlar veya XML'ler ile yeni bir sınıf dahil edilerek veya tanımlanmış olan yeni elemanlar miras alınarak dahil edilirler.

İYP yaklaşımı ile bir uygulama geliştirilmeye başlanmadan önce seçilecek olan araçların göz önünde bulundurulması gereken özelliklerinde biri de seçilecek olan aracın ilgi geliştime stilidir. Daha önceden de üzerinde durulduğu gibi İYP'de 3 tip yazım stili bulunmaktadır. Her bir stilin geliştirilecek uygulamaya göre avantaj ve dezavantajları mevcuttur.

5.3 İYP Araçlarında Anlamsal Benzerlikler

İncelenmekte olan araçlarda, ilgi tanımlamalarının görünür yazımsal farklılıkları olmasına karşın İYP yaklaşımları, temelde anlamsal olarak benzerdir. Her araç anlamsal olarak benzer birleşme noktası, icra noktası ve icra edecek kod kavramlarını barındırır. Çizelge 5.2 her bir yaklaşımın anlamsal olarak karşılaştırılmasını göstermektedir. Yaklaşımlarda küçük bir takım farklılıklar görülmektedir. Fakat temelde hepsi ortak amaçlarda birleşmektedirler. Araçlar arasındaki bu benzeşmeler bir avantaj olarak değerlendirilebilir, çünkü bir İYP diline adapte olunduktan sonra, diğer diller de kolaylıkla öğrenilip uygulanabilir.

Bir İYP aracının birleşme noktası modelinin anlamlılığı, kullanılabilir birleşme noktalarının boyu ve bunların nasıl eşlendiğini belirler. Her İYP aracı birleşme noktalarını eşlemek için bir veya birden fazla icra noktası içerir (Örneğin metod çağrıları). Bazı icra noktaları sadece özel bir tip birleşme noktalarını belirtmek için kullanılır. Diğer icra noktaları ise ortak özelliklere sahip olan her tip birleşme noktası ile eşlenebilir (Örneğin bir iş akışı içindeki tüm birleşme noktaları). Birleşme Noktası tipleri ve bunlara özel icra noktaları aşağıdaki gibi gruplandırılabilir:

- Uyandırma: Metodların veya diğer kod elemanlarının ne zaman çağrılacağı ve çalıştırılacağını işaret eder.
- Başlatma: Tiplerin ve nesnelerin başlatılmasını işaret eder
- Erişim: Alanların okunup yazıldığı zamana işaret eder.
- İstisna İşleme: Hataların ve istisnaların oluştuğu veya ele alındığı zaman işaret eder.

İcra Noktaları ise ;

- Kontrol Akışı: Belirli program kontrol akışları içindeki birleşme noktaları
- Kapsama: Belirli sınıflar veya metodlar içerinde bulunan kodda bulunan birleşme noktalarına karşılık gelir**.**
- Koşulluluk: Belirlenmiş bir koşulun gerçekleşmesi durumundaki icra noktaları kavramlarını içerir.

Çizelge 5.2 Anlamsal açıdan İYP araçlarının karşılaştırılması [3]

	Uyandırma	**Başlangıç**	**Erişim**	**İstisna İşleme**	**Kontrol Akışı**	**Kapsama**	**Koşulluluk**
AspectJ	{metod, konstrüktör, ilgi) x {çağrı, icra)	Örnek, statik, ön inceleme	Alan Get/Set	Kod	Kontrol Akışı	Kod içerisinde	if
Aspect Werkz		Örnek, statik				Kod içerisinde, metodu veya çağrısı bulunur	-
JBoss AOP		Örnek		İlgi	(Belirlenmiş bir yığın çağrısı)	Kod içerisinde, metod, alan veya çağrısı bulunur	Dinamik Kontrol akışı
Spring AOP	Metod icrası	-	-	İlgi	Kontrol Akışı	-	Özel bir icra noktası

5.4 Dil Mekanizmaları

Çizelge 5.3 İYP araçlarının dil mekanizmaları yönünden daha detaylı bir biçimde karşılaştırılmasını göstermektedir. Buradaki baz alınan karşılaştırma kriterleri, icra noktası eşleme (birleşme noktaları ile icra noktalarının eşleştirilmesi), icra noktası birleştirmesi (icra noktasında tanımlanan metodların aynı ifade içerisinde birleştirilmesini belirten durumlar), ilgi biçimleri (ilgilerin icra noktalarında tanımlanma biçimleri ve icra yerleri belirtimi), dinamik içerik (birleşme noktalarının statik veya dinamik olarak tanımlanma durumları), ilgi icra başlatımları (uygulama içinde programın işleyişinin durdurulup ilgilerin icra ettirildiği mekanizmalar), uygulamaya eklenmiş olan ilgilerin genişletilebilme destekleridir.

Çizelge 5.3 İYP araçlarının anlamsal yönden karşılaştırılması [3]

<table>
<tr><th></th><th>İcra Noktası Eşleme</th><th>İcra Noktası Birleştirme</th><th>İlgi Biçimleri</th><th>Dinamik İçerik</th><th>Başlatılma</th><th>Genişletilebilirlik</th></tr>
<tr><td>AspectJ</td><td rowspan="2">Metod ismi, alt tipler, dipnot stili</td><td rowspan="3">&&, ||, !</td><td rowspan="2">önce, sonra, esnasında</td><td rowspan="3">this, target, args, (static olarak yazılır)</td><td>Sanal Makine, Hedef, kopya</td><td>Soyut icra noktaları</td></tr>
<tr><td>Aspect Werkz</td><td>Sanal Makine, Sınıf, kopya</td><td rowspan="3">Üzerine yazma, Ilgi bağlantıları</td></tr>
<tr><td>JBoss AOP</td><td>Metod ismi, kopyası, dipnot stili</td><td>Esnasında</td><td rowspan="2">Dinamik erişim</td><td>Sanal Makine, Sınıf, kopya, Birleşme Noktası</td></tr>
<tr><td>Spring AOP</td><td>Düzenli İfadeler</td><td>&&, ||</td><td>önce, sonra, esasında istisna işleme</td><td>Sınıf, kopya</td></tr>
</table>

5.4.1 İcra Noktası Eşleme ve Birleştirme

AspectJ, AspectWerkz ve JBoss AOP metod imzaları ile icra noktası eşleme imkanı sunarlar. AspectJ ve AspectWerkz birden çok tip için alt tiplerin de ifade edilebildiği basit bir yapı sunar (Account örneğindeki tüm alt tiplerin kapsandığı birleşme noktası vb.). İcra Noktası birleştirme işlemleri araca özel olarak desteklenmektedir.

5.4.2 Birleşme Noktası Bağlamı

AspectJ ve AspectWerkz'de dinamik birleşme noktalarına, standart Java dilindeki metod parametreleri gibi tanımlanmış olan belirleyici ve bağlayıcı icra noktaları parametreleri tarafından erişilir (Şekil 5.1 ve Şekil 5.2). JBoss AOP ve Spring AOP'de, birleşme noktalarına dinamik olarak erişim sağlanır. Bunun yanında bu araçlarda statik parametre yazılm maliyetleri hesaba katılmalıdır.

5.4.3 Başlatılma

Tüm araçlarda ilgi sınıfları cinsinden sınıfların yaratılması "per" ifadeleri ile kontrol edilmektedir. Beklendiği gibi, Spring AOP'de fazla ek kod yazma gerektirmeden ilgiler başlatılabilmektedir. Çalışma zamanında ilgilerin aktif veya pasif yapılıp anında test edilebilmesi bu aracın en büyük avantajlarındandır. Diğer araçlardaki temel farklılıklar ise AspectJ ilgi başlatılmasını her kontrol akışıyla, AspectWerkz her program kopyasıyla, JBoss AOP ise her farklı birleşme noktasıyla gerçekleştirir. Araçlardan hangisinin daha kullanışlı olduğunu geliştirilecek olan uygulamanın ihtiyaçları belirler.

5.4.4 Genişletilebilirlik

İlgi genişletilebilirliği, ilgi kütüphanelerinin yayılımını ve daha sonradan bazı uygulamalarda kullanılabilir olması anlamına gelmektedir. Örneğin bir ilgi kütüphanesi, bir uygulama günlüğünün tutulması için tüm lojik ve gerekli altyapıyı sağlayabilir. Hazırlanan kütüphaneyi belirli bir projeye adapte edebilmek için o kütüphanede kullanılan icra noktaları uygulamaya özel olarak birleşme noktalarına eklenmelidir. AspectJ, soyut ilgiler ve somut icra kodları ile genişlemeyi desteklemektedir. AspectWerkz ve JBoss AOP bu yaklaşımdan tamamen farklı bir mekanizmaya sahiptir. Bunlar soyut icra noktası mekanizması bulundurmazlar. Genişleme, ilginin alt sınıflara bölünmesi ve XML veya dipnotlar ile birleştirilen yeni icra kodları tanımlama ile gerçekleştirilir. AspectWerkz ve JBoss AOP dillerindeki (icra noktası ve icra kodlarının) bu açık birleştirme işlemi, bu dillerdeki ilgilerin alt sınıflara ihtiyaç duyulmadan yeni sistemlere kolaylıkla uyarlanabileceğini göstermektedir.

5.5 Derleme ve Entegrasyon

İYP araçları arasındaki en belirgin farklılıklar ilgilerin tanımlanması gibi yazımsal farklılıklardır. Bu farklar yanında ilgilerin yapılandırması, derlenmesi, yazım kontrolleri gibi alanlarda da farklılıklar bulunmaktadır. Çizelge 5.4'te bu farklar görülmektedir.

5.5.1 İlgilerin Yapılandırılması

İYP araçlarının bir projeye adapte edilmesi sırasında dikkat edilmesi gereken en önemli nokta, araçların IDE desteğine erişim, Ant kullanarak yapılandırma gözönüne alınmadan, geliştirme ortamıyla ne kadar entegre olabildiğidir. İYP araçları arasındaki temel fark yapılandırma ortamı entegrasyonuna gelince dil uzantısı kullanıp kullanmadıklarıdır. AspectJ Java diline uzantı olarak geliştirilmiş bir kod stili kullanırken, diğer 3 yaklaşım standart Java , XML ve dipnot tabanlı kombinasyonu kullanmaktadır. Entegrasyon açısından bakıldığında AspectJ'nin Java diline uzantı olması ilgi tanımlamalarının da standart Java sınıfları gibi basit formu olması yararını getirmektedir. Diğer yandan Java diline uzantı olarak AspectJ yetenekleri kazandırılması nedeniyle aspectJ kullanılarak hazırlanmış olan tüm projeler Java dilinin kullanıldığı tüm araçlara bu uzantıların yüklenmesi gerekmektedir.

Yapılandırma ortamı ile bağlantılı olarak yaklaşımlar arasındaki en temel fark, diğer araçlar standart Java derleyicisi kullanırken, AspectJ java diline uzantı olduğundan dolayı kendi derleyicisini barındırmak zorundadır. AspectJ derleyicisi Eclipse Java derleyicisinin bir uzantısıdır ve Eclipse'den bağımsız olarak komut isteminden çalıştırılabilmelidir. Aynı zamanda bu derleyici Eclipse 3. parti bileşenleri ile ve diğer IDE'lerle birlikte de kullanılabilir. AspectJ derleyicisi .java ve .aj olarak tanımlanmış Java ve aspectj kodlarını derler ve sade Java bytekodu üretir. Yeni bir derleyici gereksiniminin dezavantajları olsa da kritik bir yarar noktası olan statik icra noktası kontrol mekanizması sağlar.

Çizelge 5.4 İYP araçları geliştirme ortamı entegrasyonları [3]

<table>
<tr><th></th><th>Kod</th><th>Derleyici</th><th>Kontrol Etme</th><th>Dokuma</th><th>Yayım</th><th>İcra</th></tr>
<tr><td>AspectJ</td><td>Dosya Uzantısı .java, or .aj</td><td>Genişletilmiş AspectJ Derleyicisi</td><td>Tamamen Statik Kontrol</td><td>Derleme ve Yükleme Zamanı Bytecode Üretme,</td><td>Sabit Yayım</td><td>Standart Java Programı İcrası</td></tr>
<tr><td>Aspect Werkz</td><td rowspan="3">Standart .java, .xml</td><td>Java Derleyicisi,
Çıktı işleyicisi</td><td rowspan="2">Kısmi Statik Kontrol,</td><td rowspan="2">Çalışma Zamanı Ele alma</td><td rowspan="3">Dinamik Yayım</td><td rowspan="3">Uygulama Çatısı tarafından çağrılan ve Yönetilen İcra</td></tr>
<tr><td>JBoss AOP</td></tr>
<tr><td>Spring AOP</td><td>Java Derleyicisi</td><td>-</td></tr>
</table>

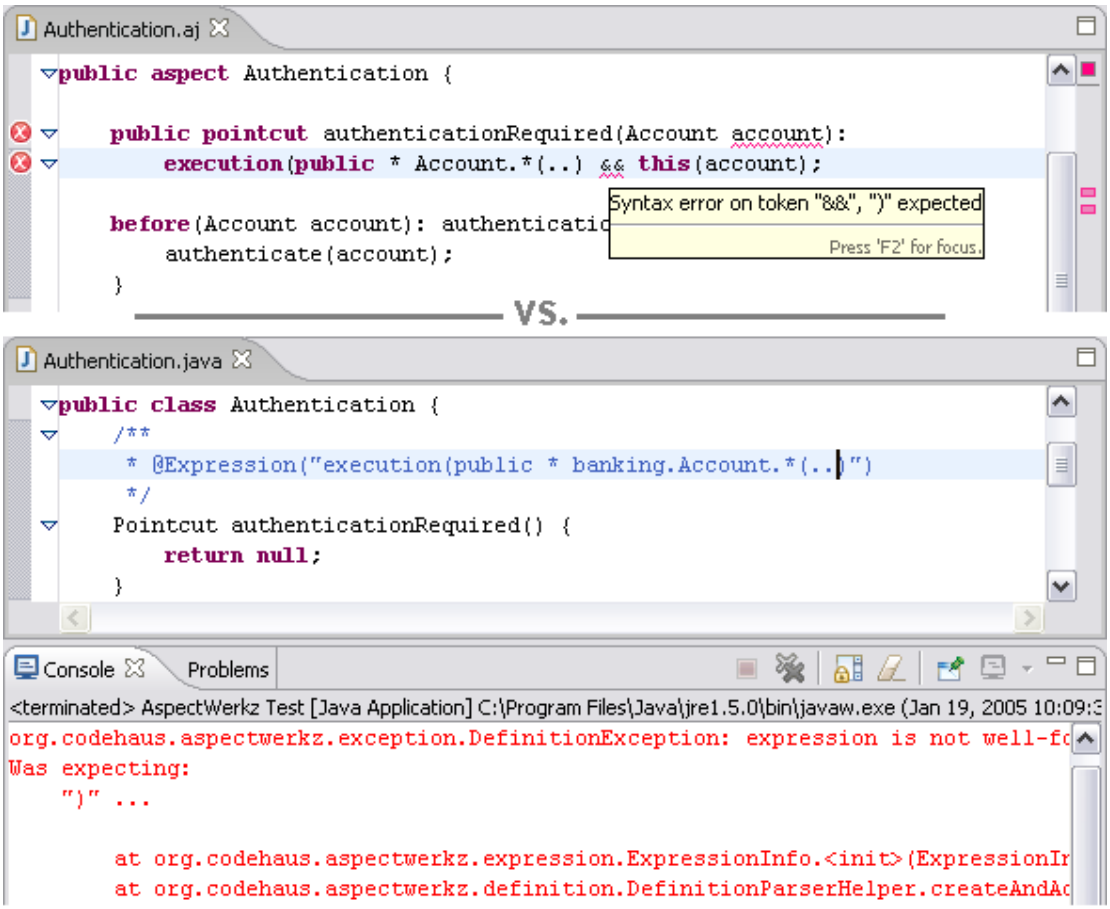
5.5.2 İcra Noktalarının Statik Kontrol Edilmesi

Uygulama Geliştiriciler sınıflar ile çalışırlarken genelde statik kontrol etme mekanizmasını destekleyen IDE'lere yönelirler. Bunun anlamı, sınıf isimlerinin ve metod imzalarının yanlış yazılma ihtimalini ortadan kaldırmak amacıyla hataların kullanıcıya anında bildirilmesidir. Burada derleyici sözdizimsel hataları tespit eder ve geliştiriciye hataların yerini bildirir. Statik kontrol mekanizması olmayan geliştirme ortamlarında sözdizimsel hatalar çalışma zamanına kadar tespit edilemezler. AspectJ, ilgi tanımlamaları için de tamamen statik bir kontrol mekanizmasına sahiptir ve ilgiler içindeki icra noktalarında refere edilen kodların yazımındaki herhangi bir hata kullanıcılara geliştirme anında bildirilir. Bu özellik, Java diline uzantı olarak geliştirilmiş olan yazılım ile desteklenir. Diğer İYP araçları dereceli olarak ilgi tanımlamaları içerisinde kontrol mekanizması barındırmaktadırlar fakat hiçbirinde statik icra noktası kontrol mekanizması bulunmaz. Çünkü bu yaklaşımlar dipnot veya XML yapısı kullanmaktadırlar. Tipik Java kullanıcıları için AspectJ dışındaki diğer yaklaşımlarda XML dosyaları içerisinde tanımlamalar yapıldığından hataların çalışma zamanında ayıklanması maliyetini ortaya çıkarmaktadırlar. Statik kontrol mekanizmasının bulunduğu sistemlerde kolaylıkla yakalanabilecek olan bir yazım hatası, bu kontrol mekanizmasının olmadığı durumlarda büyük hata ayıklama maliyetlerine sebep olabilmektedir.

AspectJ derleyicisi ile İYP uygulaması geliştirilirken tüm standart java dili statik kontrol mekanizmasının avantajlarına sahip olunur. Bu mekanizmanın olmadığı yaklaşımlarda, icra noktalarının belirtildiği kodların yazımında çok dikkatli olunması gerekmektedir. Bu

yaklaşımlar kullanılırken uygulama geliştiren kişinin çalışma zamanı hata ayıklama ve çözmeye alışması gerekmektedir. Bu işlem çoğunlukla maliyetli ve zordur.

Şekil 5.5'de iki İYP aracının, icra noktası tanımındaki bir sağ parantez eksikliği hatasını tespit etmesi arasındaki fark görülmektedir. Üst kısımda, AspectJ'nin hatayı statik kontrol ile gösterme örneği, alt kısımda ise AspectWerkz'in çalışma zamanında hatayı tespit etme yöntemleri görülmektedir.



Şekil 5.5 AspectJ statik kontrol mekanizması [3]

AspectJ derleyicisi, ilgi kodu yazılırken hatayı yakalar, hatanın kaynağını ve ilgili yeri direkt olarak kullanıcıya gösterir. AspectWerkz kullanılırken ise çalışma zamanına kadar hata tespit edilemez. Buradan statik kontrol mekanizması olmayan araçlarda hata yakalamanın uzun ve daha zor bir süreç olduğu sonucuna varılabilir. Fakat daha genel ve daha fazla zaman harcamaya neden olan problemler, icra noktalarının isimlerinin yazımındaki hatalardan kaynaklanmaktadır. Statik kontrol mekanizması olmayan sistemlerde İYP çatısı hiçbir ilgi çalıştıramaz ve başarısızlığa uğrar. Neyin yanlış gittiğini anlamak için öncelikle icra noktalarının sözdizimsel olarak kontrol edilmesi, hata yakalama maliyetini ve süresini

kısaltmaktadır.

5.5.3 Dil Uzantısı ile Geliştirme Ortamı Entegrasyon Avantaj ve Dezavantajları

Java dili uzantısı olarak İYP mekanizmasının uygulandığı araçlarda (AspectJ) birtakım avantaj ve dezavantajlar vardır.

- Geliştirilmiş uygulamanın farklı platformlarda çalıştırılabilmesi için gerekli yazılımın o ortama da yüklenmesi gerekmektedir.
- Geliştirilmiş olan ilgilerin sisteme eklenebilmesi için geliştirilmiş bir derleyici gerektirir.

+ Java derleyicisi tabanlı derleme mekanizması sayesinde ilgilerin normal java sınıfları gibi statik kontrol mekanizması ile kontrol edilme kolaylığı.

+ İcra noktalarının yazılması ve hata ayıklanmasındaki kolaylık

5.5.4 Dokuma ve Performans

Nesneye Yönelik geliştirilmiş olan uygulamaların farklı mekanizmalarda derlenip çalıştırılabilmesine rağmen İYP araçları ilgilerin geliştirilmesi ve çalıştırılabilmesi için farklı gereksinimlere ihtiyaç duymaktadırlar. Bir ilgi dokuyucu bir icra noktasına gelindiğinde icra edilecek olan kodun otomatik olarak çağrılmasını sağlayan bir dokuma işlemi gerçekleştirir. Dokuyucular İYP kodlarını kaynak kodu veya ikili formda girdi olarak alırlar. İlgi dokunmasının büyük ölçüde dokuma anından kaynaklanan performans ve ölçülebilirlik kısıtları vardır.

AspectJ ve AspectWerkz, hem geliştirme zamanı hem de yükleme zamanı dokumayı destekler. Buna rağmen AspectJ geliştirme zamanı, AspectWerkz ise yükleme zamanı dokuma mekanizması üzerine odaklanmaktadır.

JBoss AOP ve Spring AOP, dinamik birleşme noktaları kullanırlar ve çalışma zamanı dokuma işlemine odaklanırlar. Bu mekanizmalarda program durdurucular ilgileri aktive ederler. Java Sanal Makinesi teknolojisi de çalışma zamanı dokuma mekanizmasını destekleyecek şekilde genişletilebilir. Çalışma zamanı durdurma ve ilgi kodunu icra ettirme mekanizmasının en temel yararı, ilgilerin program işleyişinde yayımına imkan sağlamasıdır (Dinamik yayım). Bunun anlamı ilgilerin çalışma zamanının aktif veya pasif olabilmesidir. Çalışma zamanı yayım özelliği JBoss AOP'nin en temel özelliğidir. Bu özellikle JBoss AOP, ilgilerin aktif veya pasif olarak işaretlenebileceği uygulama sunucusu yönetim konsolu sunmaktadır. Bu özellik Spring AOP'de de bulunmaktadır. AspectWerkz yükleme ve geliştirme zamanı dokuma mekanizması da çalışma zamanı yayımı destekleyecek şekilde genişletilebilir. AspectJ kullanılırken ilgilerin aktif veya pasif olarak işaretlenmesi ilgi veya icra noktasındaki

"if" ifadeleri ile mümkündür. Dinamik İYP bazı zamanlarda çalışma zamanı yayım olarak da tanımlanabilmektedir.

5.5.5 Performans Kriterleri

İlgiler ile yazılmış bir uygulama kodunun çalışması ile enine kesen ilgilerin nesneye yönelik programlama kodunun içerisine yayılmış bir şekilde çalışması, performans yönünden hemen hemen aynıdır. Çalışma zamanının uzaktan veya veritabanı komutları ile belirlendiği çoğu kurumsal uygulamada, İYP araçlarından herhangi birinin kullanılmasından kaynaklanacak maliyet göze alınabilir. Araçların performans açısından karşılaştırılabilmesi için de çeşitli kriterler mevcuttur.

AspectJ ile uygulama geliştirilirken, bellek kullanımı ve zaman açısından derleme maliyetleri diğer araçlara göre daha fazladır. Çünkü bu araçta ilgi planlamalarının çoğu derleme zamanında gerçekleştirilir. Bu maliyetler özellikle büyük projelerde problemlere yol açacak boyutlara ulaşabilir, ilgiye yönelik bağımlılıklardan dolayı, bir icra noktasında yapılan bir değişiklik sonucunda projenin büyük bir kısmının yeniden derlenmesi gerekebilmektedir. Fakat bu aynı zamanda çalışma zamanı yapılacak işleri en aza indirger. Çünkü çalışma zamanına sadece ilgilerin eşleştirilme işlemi kalmaktadır. Buna karşın JBoss AOP ve Spring AOP yaklaşımlarında ise çalışma zamanında birçok işlem yapılmaktadır. Bunun sonucu olarak AspectJ 'de ilgi icra etme işlemi AspectWerkz, JBoss AOP ve Spring AOP'ye göre daha hızlı gerçekleşmektedir. Bunun yanında AspectJ en fazla geliştirme zamanı maliyetine sahiptir, AspectWerkz ikinci sırada, JBoss AOP 3. sırada ve Spring AOP ise son sırada yer almaktadır.

5.6 İlgi Yeniden Kullanımı

Uygulama geliştiriciler, çeşitli uygulamalar tarafından sağlanmış olan kütüphaneleri kullanabildikleri gibi İYP ile geliştirdikleri uygulamalarda da çeşitli ilgi kütüphaneleri yaratabilirler. İlgilerin yazılımı sırasında icra edecek ilgi kodu ilginin davranışını belirlerken, icra noktası ise ilginin uygulamaya entegre edileceği yeri belirler. İcra noktaları geliştirilecek olan uygulamalara özel olduklarından ilgilerin yeniden kullanılabilecek olan kısımları icra edilecek olan kodlardır.

AspectJ ile geliştirilmiş olan ilgiler miras alma ve soyut icra noktası (arayüz) içerdiklerinden dolayı yeniden kullanılırlığa imkan verirler. Soyut icra noktaları, diğer herhangi bir icra noktası gibi icra edecek olan ilgi kodunu tanımlamak için kullanılırlar. Bunlar metod isimleri ve imzalarından oluşurlar. Miras alma mekanizması ile de ilgi sınıflarının geliştirilmesi sağlanabilmektedir.

JBoss AOP ve AspectWerkz metod durdurucular ve icra noktalarının ayrı ayrı yazılmasından oluşmaktadır (sırasıyla Java ve XML dosyalarına). Metod durdurucular kolaylıkla yeniden kullanılabilirken, icra noktaları uygulamaya göre özelleştirilmelidir.

Spring AOP yaklaşımında ise ilgilerin konfigürasyon mekanizması farklı uygulamalar için yeniden yapılandırmayı gerektirir.

İlgilerin yeniden kullanımı için farklı yaklaşımlardaki araçlarda, farklı avantaj ve dezavantajlar mevcuttur. Standart uygulama özelliklerini içeren (örneğin miras alma) araçlarda ilgilerin yeniden kullanılabilirliği ön planda iken, konfigürasyon dosyaları ile konfigüre edilebilen araçlarda ise ilgilerin uygulamaya adapte edilebilirlik özelliği ön plandadır. En etkin yeniden kullanılabilir ilgi kütüphanelerinin oluşturulabilmesi, yukarıda bahsedilen iki yaklaşımın birden uygulandığı araçlarda olacaktır (Foundations of AOP for J2EE development).

5.7 İYP Araçlarının Seçimi

İYP araçlarının özellikleri incelendikten sonra, hangi aracın geliştirilecek olan projeler için uygun olduğunun seçimi yapılabilir. Bu araçlar temelde İYP yaklaşımını uygulamalarına rağmen, farklı İYP uygulama biçimleri projelerin geliştirme sürelerini, maliyetlerini ve performanslarını etkilemektedirler. Bu nedenle bu bölümde, her bir aracın projelerdeki avantaj ve dezavantajlarının bir özeti bulunmaktadır.

5.7.1 AspectJ

- Java diline uzantı olarak geliştirilen bu araç dil uzantılarının, yeni bir derleyicinin ve ilgili diğer araçların kurulumunu gerektirir.

- Kütüphane Eksikliği

+ İlgi Tanımlamalarının basit olması ve statik icra noktası kontrol mekanizması

+ En gelişmiş IDE entegrasyonu

+ Geniş dokümantasyon desteği

5.7.2 AspectWerkz

- AspectJ'ye göre daha karmaşık ilgi ve icra noktası tanımlama

- Statik icra noktası kontrolü eksikliği

- Kütüphane Eksikliği

+ Dil Uzantıları ve yeni derleyici kurulumu gerektirmez.

+ İlgilerin dinamik yayımına imkan verir.

5.7.3 JBoss AOP

- İcra Noktalarının statik kontrolünün eksikliği

- Gelişmiş IDE desteği eksikliği

+ Zengin ilgi kütüphaneleri, zengin JBoss kütüphaneleri ile entegre edilmiştir.

+ IDE desteği sayesinde XML yazımı kolaydır.

+ İlgilerin dinamik olarak çalışma zamanında yayımına imkan verir.

5.7.4 Spring AOP

- IDE desteği eksikliği

+ Birleşme Noktası modeli öğrenmesi ve yazımı kolaydır.

+ Spring çatı entegrasyonu sayesinde Spring uygulamalarına kolaylıkla adapte edilebilir.

+ Uygulama sunucuları arasında, geliştirilmiş ilgi kütüphanelerini taşınma kolaylığı

6. İYP VE GOF (Gang of Four) TASARIM KALIPLARI

Yazılım mühendisliğinde tasarım kalıbı, sıkça karşılaşılan problemlere karşı geliştirilmiş genel uygulanabilen çözüm olarak tanımlanabilir. Bir tasarım kalıbı direkt olarak koda dönüştürülebilecek tamamlanmış bir tasarım değildir. Farklı durumlarda problemlerin çözümü için kullanılabilen bir tanımlama veya çözüm şablonudur.

Nesneye Yönelik Program geliştirmede sıkça kullanılan tasarım kalıpları, sadece belirli bir dil veya kısmi bir programlama yaklaşımı için tasarlanmamışlardır. Tasarım kalıpları yazılım sistemlerinin yeniden kullanılabilirliğine katkıda bulunmak ve sıkça karşılaşılan problemlere çözüm getirmek amacıyla geliştirilmişlerdir. Belirli problemlere etkin çözümler getiren tasarım kalıpları bazen uygulamalar içerisinde enine kesen ilgiler problemine sebep olabilmektedirler. İYP'de tasarım kalıplarının uygulanmasının amacı, bu kalıpların uygulanmasındaki enine kesen ilgiler problemine çözüm getirmektir (Pawlak vd., 2005).

6.1 Tasarım Kalıplarının İlgiye Yönelik Programlama ile Uygulanması

Günümüzde tasarım kalıplarının İYP ile uygulanması alanında birçok araştırmalar yapılmaktadır. Tasarım kalıplarının İYP ile uygulanması ile elde edilecek avantajlar aşağıda listelenmektedir:

Yerellik: Yeni işlevsellik kazandıran kodlar ilgiler içerisinde tanımlanır, böylece uygulamaların kodlarında değişiklik yapılmasına gerek kalmaz ve modülerizasyon sağlanmış olur.

Yeniden Kullanılabilirlik: İşlevsellik kodunun ilgiler içerisinde uygulanması büyük oranda soyutlama sağlar. Bu özellik sayesinde tanımlanan arayüzler aracılığıyla yeniden kullanılabilirlik oranı artar.

Birleştirme Şeffaflığı: Uygulamanın bütününde karmaşıklığa sebep olmadan birçok tasarım kalıbının bir nesneye uygulanabilmesi sağlanır.

Eklenip - Ayrılabilirlik: Tasarım kalıpları kullanılarak geliştirilen bir uygulamada uygulanmış tasarım kalıbı uygulamanın genelini etkilemez. İlgiye Yönelik Programlama ile bir parametre değişimi tasarım kalıbının aktif veya pasif yapılabilmesini sağlar.

Bu bölümde NYP'de sık kullanılan bazı tasarım kalıplarının İYP ile uygulanmasına değinilecektir.

6.2 Yegane Tasarım Kalıbı ve İYP

Yegane Tasarım Kalıbı, uygulama yaşam döngüsü boyunca sadece bir kez yaratılması istenen nesneler için kullanılmaktadır. Bu kalıp ile uygulama, uygulanan nesnenin bir kez yaratıldığını garanti eder. Böylece gereksiz bellek israfı ve bazı kritik işlemlerin sadece bir kere yapıldığı garanti edilebilir. Uygulanması genelde statik olarak nesnelerin yaratılmasına ve bellekte tutulması ile yapılmaktadır. Bu kalıbın başlıca 2 kullanım alanı mevcuttur:

- Uygulama içerisinde sadece bir kopyasının yaratılması gereken nesneler, örneğin uygulama içerisindeki evrensel (global) değişkenler için kullanılır. Çünkü bu tip nesnelerin birden fazla kez yaratılması çalışma zamanı hatalarına sebep olabilmektedir.
- Bellek ve kaynakların kullanımını kontrol altında tutmak.

Bu probleme çözüm, tek kopyasının yaratılması istenen sınıfın içerisine yaratılacak olan nesnenin referansının tutan statik bir özellik koymak ve eğer bu referans boş (null) olmayan bir nesneyi gösteriyorsa o nesnenin yeni bir kopyanın yaratılmasına izin verilmemesidir.

Yegane Tasarım Kalıbının java ile uygulanma örneği aşağıda görülmektedir:

```
public class MySingleton {
        private static MySingleton instance = null;
        public static MySingleton getInstance() {
                if (instance==null) {
                instance = new MySingleton();
                }
        return instance;
        }
}
```

MySingleton sınıfı içerisinde tanımlanmış olan getInstance() metodu, uygulamada sınıfın aynı kopyasını kullanıldığından emin olunmasını sağlar. Bu kullanımın uygunsuzluğu, sınıfın yeni bir kopyasının "yeni" (new) yapılandırıcısı kullanılarak uygulamadaki diğer sınıflar tarafından direkt olarak başlatılamamasına sebep olmaktadır. MySingleton nesnesinin bir kopyası sadece MySingleton sınıfını uygulayan nesneler tarafından yaratılabilmektedir. İYP yaklaşımı bu problemi çözmek için kullanılabilir. İYP yaklaşımının bu problemi çözmek için kullandığı teknik, "yeni" yapılandırıcısının davranışını simüle eden bir ilgi tanımlamaktır. Yapılandırıcı çağrıldığında ilgi program işleyişini durdurur ve yeni metodu çağrılarak nesnenin bir kopyasının yaratılması sağlanmış olur.

Bir tasarım kalıbında uygulamanın diğer sınıflarına gore "Şeffaflık" kavramı sadece Yegane Kalıbı ile sınırlı değildir. Genel yapıları itibari ile tasarım kalıplarının uygulama içerisinde etkileşim içinde bulundukları sınıflar üzerinde büyük etkileri vardır. Uygulama kodları incelenirken Yegane Tasarım Kalıbının tespit edilmesi diğer tasarım kalıplarının tespit

edilmesine gore daha basittir. Yegane Tasarım Kalıbı da dahil tüm tasarım kalıplarının ortak hedefi kodu karmaşıklaştıracak yapıda olan problemlere etkin çözümler üretmektir.

6.3 Gözlemci Tasarım Kalıbı ve İYP

Bazı durumlarda uygulama geliştiriciler uygulama içindeki nesnelerin durum değişimlerini ve nesneler tarafından üretilen durumları kontrol altına almak isterler. Bu gibi durumlara örnek olarak bir metin yazım editörü ile yazılan bir metin üzerinde değişiklik olduğu zaman kaydetme işlevinin aktif olması verilebilir. Gözlemci Tasarım Kalıbı, "özne"ler olarak adlandırılan diğer nesnelerin durumlarının değişmesi koşulunda gözlemci nesnelerin bilgilendirilmesini sağlar.

Nesneye Yönelik Programlama yaklaşımında iki arayüz üzerine odaklanılır. Bunlar "Gözlemci" ve "Özne" arayüzleridir. Özne arayüzü tüm gözlenen sınıflar tarafından uygulanır ve gözlemci sınıfları kaydetme, çıkarma işlemleri için metodlar sunar. Gözlemci arayüzü ise gözlemci nesneler tarafından uygulanır ve durum değişiklikleri hakkında bilgilendirmelerin yapılması için gerekli metodları içerir.

İYP ile gözlemci kalıbının kullanılması ile sunulan çözüm, gözlemcilerin yönetiminin ve durum değişiklerinin algılanmasının ilgiler aracılığıyla ele alınmasıdır. İki aşamalı olarak geliştirilecek uygulamada ilk aşama kodun genel kısımlarını içeren soyut bir ilgi geliştirilmesidir. Bu ilgi tüm gözlemciler için yeniden kullanılabilir. İkinci aşamada ise bu ilgi, gerçek gözlemcilerin ihtiyaçlarını karşılayacak şekilde genişletilir.

İlginin genel kısmının uygulanmasında başlıca iki metod bulunur. Bunlardan bir tanesi gözlemcileri kaydetmek diğeri ise bunları uygulamadan çıkarmak içindir. Tanımlanan soyut metodlar içerisinde tanımlanan icra noktaları gözlemlenecek olan nesneleri belirtirler. Bu icra noktaları durumları değişecek olan nesnelerin gözlemlenerek durum değişikliklerinin algılanması ile görevlidirler.

İlginin gözlemci kısmının uygulanmasında ise tanımlanacak olan sınıf gözlemci arayüzünü içerir. Bu sınıf içerisinde gözlemlenecek, durumu değişen nesnelerin tanımlamaları yapılır. Bu sınıflar içerisinde herhangi bir durum değişikliği olduğunda ilk aşamada tanımlanmış olan ilgi sınıfı çağrılacak ve gerekli kodların icra etmesi sağlanacaktır.

6.4 Komut Tasarım Kalıbı ve İYP

Nesneye Yönelik Yaklaşım, sınıflar içerisinde bulunan ve veriler üzerinde değişiklikler yapan işlemlerin modülerizasyonu için çeşitli teknikler sunmaktadır. Bu işlemler, sınıflar içerisindeki metodlarda tanımlanırlar. Bir sınıfın yaratılan tüm kopyaları aynı işlemleri

kullanırlar. Bu işlemler değişmezler; herbiri yaratıldığı sınıf kopyaları sonlandırılmadığı sürece sabit kalırlar. Bu sabitlik, işlemler için güvenilir fakat katı kurallıdır. Komut tasarım kalıbı, işlemlerin sınıflardan bağımsız olarak tanımlanmasına imkan verir. Birçok farklı komut tipi sınıflardaki gereksinimlere tanımlanabilir ve ilgili sınıflara atanabilir.

Gözlemci Kalıbında olduğu gibi, Komut kalıbının da İYP yaklaşımında uygulanması 2 aşamadan meydana gelmektedir. Bunlardan ilki genel komut ilgisinin tanımlanması, ikincisi ise gerçek komut sınıfının geliştirilmesidir.

İlginin genel kısmında, icra edilecek komutun değerinin belirlendiği bir metod tanımı yapılır. Komutlar, tek bir icra metodunun tanımlandığı Komut arayüzünü uygularlar. Bu metod komutun eriştiği nesneyi parametre olarak alır. İYP ile bu kalıbın uygulanmasında tanımlanacak olan icra noktaları, komutun çalıştırılacağı sınıfların belirlenmesinde ve komutun çalışmasının tetikleneceği durumların belirlenmesinde kullanılır.

İkinci aşamada tanımlanacak olan gerçek komut tasarımında ise genel ilgi içerisindeki icra noktalarında kullanılan metodlar tanımlanır.

6.5 Sorumluluk Zinciri Tasarım Kalıbı

Bir önceki bölümde açıklanan Komut Tasarım Kalıbı, nesneler tarafından icra ettirilen komutların tanımlanmasına imkan verir. Sorumluluklar Zinciri Tasarım Kalıbı, komut kalıbını genişletilmişi olarak tanımlanabilir, şöyle ki bu kalıp birden fazla komutun bir zincir içerisinde gruplanmasına olanak sunar. Zincir icra ettiğinde zincir içerisindeki tüm elemanlar ziyaret edilir. Bir eleman ziyaret edildiğinde bir komut icra edebilir veya hiçbir işlem yapılmayabilir.

Nesneye Yönelik uygulamalarda Sorumluluk Zinciri Tasarım Kalıbı "Eğitici" (Handler) arayüzünü uygular. Zincir içerisindeki her bir komut bu arayüzü uygulamalıdır. Bu arayüz iki başlıca iki metod içerir. Bunlardan ilki, zincir içerinde icra etmekte olan komutun varisinin belirlenmesinde, diğeri ise bu metodun icrasında kullanılır.

Sorumluluklar Zinciri Tasarım Kalıbı'nın İYP'de uygulanması ilgi önceliği prensibine dayanır. Öncelikle zincir içerisinde tanımlanmış olan tüm komutlar tarafından ortak olarak kullanılan kodu uygulamak için soyut bir ilgi tanımlanır. Bu soyut ilgi, zincir içerisindeki tüm komutlar için genişletilir. Zincir içerisindeki komutların sıralaması için ilgi önceliği kavramı kullanılmaktadır.

Sorumluluklar Zinciri Tasarım Kalıbı'nın uygulanmasındaki ilk aşama olan genel ilgi tanımlama kısmında zincir icrasını tetikleyecek olan icra noktaları tanımlanmaktadır.

İkinci aşama olan gerçek sorumluluklar zinciri sınıfının tanımlanmasında, genel ilginin icra noktalarında belirlenen metodların tanımlamaları yapılır. Bu aşamada en önemli nokta öncelik kavramının tanımlanmasıdır. Bu tanımlama ile zincir içerisindeki komut icra sırasının belirlenmesinde rol oynar.

6.6 Vekil Tasarım Kalıbı

Birçok durumda, bir program içerisindeki nesneler doğrudan yerel metodlar ile iletişim kuramayabilirler. Örneğin dağıtık bir uygulama içerisindeki nesnelerin bir ağ ile bölündüğü düşünülürse, ağ içerisinde sunucu istemci arasındaki iletişimler bir vekil tarafından sağlanır. Vekil nesnesi, diğer nesneler adına temsilcilik görevini üstlenen bir nesnedir. Genel bir vekil nesnesi, tüm diğer nesneler için temsilci görevi yapar. Bunun tam tersine özel bir vekil, temsil ettiği nesne tiplerinin vekilliğini de yapabilir.

Vekil nesneleri tarafından gerçekleştirilen işlemler, tasarım kalıpları alanı dışında yer almaktadır. En basit vekil nesnesi, gelen komutları işlem yapmadan ana sınıfa ileten nesnedir.

İYP'de Vekil Kalıbının uygulanması iki aşamalıdır:

Genel ilgi tasarımı, icra edilecek olan komutları temsil edecek icra noktalarının tanımlanmasını içerir. Örneğin bir sisteme kayıt olma uygulaması düşünüldüğünde sisteme giriş eyleminde bulunulduğunda, şifre ve kullanıcı adı ekranına yönlendirilme işlemi gibi.

Bu örnekteki icra noktası ve icra noktasında tanımlanan sisteme giriş metodunun belirlenmesi, ilginin tanımıdır.

Sisteme giriş metodunun tanımlandığı kısım ise geliştirme aşamasının ikinci fazı olan vekil sınıfı tanımlamasıdır.

7. FATURA TAKİP SİSTEMİ UYGULAMASI

Tez çalışması süresince anlatılan İYP yaklaşımının bir örnek üzerinde incelenmesinin, bu yaklaşım ile birlikte gelen avantajlar ve dezavantajların görülmesi açısından faydalı olacağı düşünülmüştür. Uygulamada öncelikle gereksinimler analiz edilecek ve işlevsel gereksinimler ile enine kesen ilgiler ayrıştırılacaktır. Daha sonra uygulamanın nesneye dayalı diyagramları çizilecek ve enine kesen ilgiler eklenerek aradaki farklar, İYP ile sistemin gerçeklenmesi incelenecektir. Sistem gereksinimlerine göre uygun araç seçilecek, ilgilerin içeriklerine ve sisteme etkilerine göre tasarım kalıpları ile gerçeklemeleri sağlanacaktır. Tasarım Kalıplarındaki enine kesen ilgiler problemi de göz önünde bulundurularak bu ilgiler için belirlenen tasarım kalıpları açıklanacak ve bu tasarım kalıplarının İYP ile uygulanmasıyla elde edilen sınıf diyagramları oluşturulacaktır. Belirlenmiş bu proje tasarımı ve planına göre de uygulama gerçekleştirilecektir.

İYP'nin uygulanması için gerçekleştirilecek olan proje bir Fatura Takip Uygulamasıdır. Gereksinimlerin analizi ile başlayacak uygulama gerçekleştirim sürecinde enine kesen ilgilerin tespiti, açıklanması ve ilgiye yönelik gerçekleştirim süreci ele alınacaktır.

7.1 Fatura Takip Sistemi Gereksinimlerin Belirlenmesi

İlgiye Yönelik yaklaşım ile geliştirilecek olan FTS uygulamasının gereksinim analizi süreci sonunda çıkarılan gereksinimleri aşağıda listelenmektedir.

G1: Her satış için fatura talebi oluşturulur. Mevcut uygulama tarafından gönderilecek satış bilgileri ile fatura talep kaydı ve taksit bilgileri birlikte tanımlanır.

G2: Oluşturulan fatura talepleri muhasebe onayını bekler. Satış aşamasında oluşturulan fatura talepleri onaysız kayıtlardır.

G3: Onaylanmamış fatura kayıtları Muhasebe tarafından listelenir ve muhasebe onayından geçtikten sonra muhasebe kayıtlarına esas teşkil edecek kayıt statüsüne geçerler. Böylece onaylanan kayıtlar fatura tarihi ve fatura numarası alır. Ayrıca dövize endeksli satışlarda muhasebe tarafından faturaya esas döviz kuru girilir.

G4: Onaylanan fatura kayıtları için belirli bir desende veri çıkışı yapılır. Bu nedenle uygulamadan belirli bir desende xml veri çıkışı yapılabilmelidir.

G5: Reddedilen fatura kayıtları satış temsilcisi tarafından silinir ya da düzenlenir ve tekrar onaya sunulur. Muhasebe tarafından reddedilen fatura taleplerinin satış tarafından görüntülenerek tekrar gözden geçirilmesi gerekir. Muhasebe tarafından rededilen fatura talep

formları e-posta ile talebi yapan satış temsilcisi ve diğer tanımlı alıcı listesine gönderilir. Güncellenen kayıtlar, yeniden muhasebe tarafından onay bekler.

G6: Fatura Kayıtları üzerinde yapılan tüm işlemler sonucunda ilgili kişilere bilgilendirme

e-postası gönderilir. Örneğin bu işlemler bir fatura kaydı oluşturma, taksit güncelleme, fatura onaylama, fatura iptali ve fatura reddi olabilir.

G7: Tüm kayıtlar YTL ve döviz olmak üzere 2 para birimi cinsinden saklanacaktır. YTL esaslı satışlarda döviz ve YTL değerleri eşit olacaktır. Satışların ne cinsten yapıldığı sisteme tariflenecek olup döviz cinsinden kapatılan faturaların YTL karşılığı muhasebe ve satış gelirleri açısından mutabakat sağlamak için kullanılacaktır.

G8: Fatura kurundaki değişiklikler taksit tutarlarını güncelleyecektir. Satışlar sabit kurlar üzerinden yapılırken kesilmiş olan faturalardaki kur değeri değiştirilemeyecektir. Bunun tam tersine, ileriki bir tarif referans verilerek oluşturulmuş olan faturalardaki kur değerleri, muhasebe onayı sırasında taksitlerle birlikte güncellenecektir.

G9: Ödeme işlemi : YTL borca yapılan döviz ödemeler muhasebe tarafından ödeme girişi sırasında girilecek kurdan YTL'ye dönülecektir. USD borca YTL yapılan ödemeler ise ödeme tarihindeki TCMB döviz satış kuru kullanılarak YTL'ye çevrilmek sureti ile kaydedilecektir.

G10: Taksitin Ödendi işlemi: Tam kapatılmış yani ödeme ile eşlenmiş, bakiyesi 0 olan fatura taksitleri ödendi olarak işaretlenirler. Burada satış temsilcisi, gelen ödemeye bakarak taksitleri yeniden yapılandırabilir ve ona göre ödemenin ilgili faturayı kapatmasını sağlayabilir.

G11: Tüm veritabanı işlemleri (okuma, yazma, kayıt vb.) tek bir ara nesne tarafından gerçekleştirilecektir. Aynı anda birden fazla veri kaydının yazılmaması için veritabanı ile iletişimde bulunulan tek bir ara nesne yaratılacak ve uygulama ile veritabanı arasındaki tüm işlemler bu ara nesnenin sağladığı metodlar ile gerçekleştirilecektir.

G12: Tüm işlemlerin günlüğü, kullanıcıları, yapılan işlem ve işlemi yapan kişiler olmak üzere veritabanındaki bir Günlük tablosunda saklanacaktır.

7.2 Fonksiyonel İlgilerin Belirlenmesi:

7.2.1 Aktörlerin Belirlenmesi:

Fatura Takip sisteminin İYP kullanılarak yeniden düzenlenecek olan kısmında aktif rol oynayan aktörler şöyle sıralanabilir:

Satış Temsilcisi: Satış yapma ve yapılan satışla ilgili fatura talep kaydı oluşturma, reddedilmiş faturaları listeleme ve üzerinde güncellemeler yapıp onaya gönderme, fatura talep kaydı silme

vb. işşlemlerden sorumlu sistem kullanıcısıdır.

Muhasebe: Satış Temsilcileri tarafından girilmiş taslak durumdaki onaylanmamış fatura taleplerini listeleme, uygun olanları onaylama, uygun olmayanları reddetme, onaylanmış faturaları iptal etme vb. işlemlerden sorumlu sistem kullanıcısıdır.

Koordinatör: Tüm kullanıcıların yetkilerine sahip olan ve gerektiğinde kayıtların kontrol edilip iptal edilmesi ve onaylanmasından sorumlu en üst seviye sistem kullanıcısıdır.

Müşteri: Satış Temsilcisi tarafından üyelik paketi veya reklam satışı yapılan firmanın sistem tarafından tanımıdır.

NGO Satış Modülü: İçinde Fatura Takip Sistemini de olan tüm ürünlerin satış sürecini barındıran sistem modülüdür.

7.2.2 Senaryoların Belirlenmesi

Fatura Talep Kaydı Oluşturma:

Hedef olarak belirlenmiş müşteri firmalara satış yapıldıktan sonra satış temsilcileri tarafından yapılan satışla ilgili fatura talep kaydı oluşturulur.

Fatura Talep Listeleme: Satış Temsilcileri tarafından girilmiş olan fatura talep kayıtlarının yetkili kullanıcılar tarafından görüntülenmesidir. Onaylanmamış fatura talepleri satış temsilcileri tarafından değiştirilebilir, durumları ise onaylanmış veya reddedilmiş statüsüne muhasebe tarafından getirilebilir. Reddedilmiş fatura kayıtları satış temsilcileri tarafından listelenebilir ve değiştirilebilir. Güncellenmiş fatura kayıtları yeniden muhasebe onayına gönderilir. Onaylanmış fatura kayıtları muhasebe tarafından, tüm fatura talep kayıtları da koordinatörler tarafından listelenip değiştirilebilir.

Fatura Talep Kaydı Silme: Satış Temsilcileri tarafından oluşturulmuş fatura talep kayıtları muhasebe onayında reddedildiğinde, Reddedilmiş Fatura Talepleri menü adımı altında listelenirler. Bu kayıtlar satış temsilcileri tarafından güncellenip yeniden muhasebe onayına gönderilebilecekleri gibi satış temsilcileri tarafından silinebilirler.

Fatura Talep Kaydı Onaylama: Satış Temsilcileri tarafından oluşturulan veya güncellenen fatura taleplerinden uygun bulunanların Muhasebe tarafından onaylanmasıdır.

Fatura Talep Kaydı Reddedilmesi: Satış Temsilcileri tarafından oluşturulan veya güncellenen fatura taleplerinden uygun bulunmayanların Muhasebe tarafından reddedilmesidir.

Fatura Veri Çıkışı: Onaylanan fatura talep bilgilerinin NGO içinde veya diğer programlarda kullanılabilmesi için XML veri deseninde veri çıkışı yapılacaktır. Kayıt deseni belirlendikten

sonra oluşacak dosya fatura talebi için girilmiş tüm bilgileri ve taksit bilgilerini içerecektir.

Faturanın Kalıcı Olarak İşaretlenmesi: Muhasebe tarafından onaylanmış olan faturalar ile satış süreci sonlandırılması.

Ödeme Girişi: Satış yapılan firmalar (müşteriler) tarafından fatura talep kaydı ile belirtilen taksit tutarlarının ödemesinin muhasebeye yapılmasıdır.

Ödeme Taksit Eşleme: Müşteriler tarafından girilmiş olan ödemelerin fatura taksitleri eşleştirilmesi ve taksitlerin ödendi olarak işaretlenmesidir.

Eşlenmiş Ödeme Taksit Bilgisini Silme: Müşteriler tarafından girilmiş olan ödemelerin, eşlenmiş olan faturalardan iptal edilmesi ve taksitlerin ödenmedi durumuna getirilmesidir.

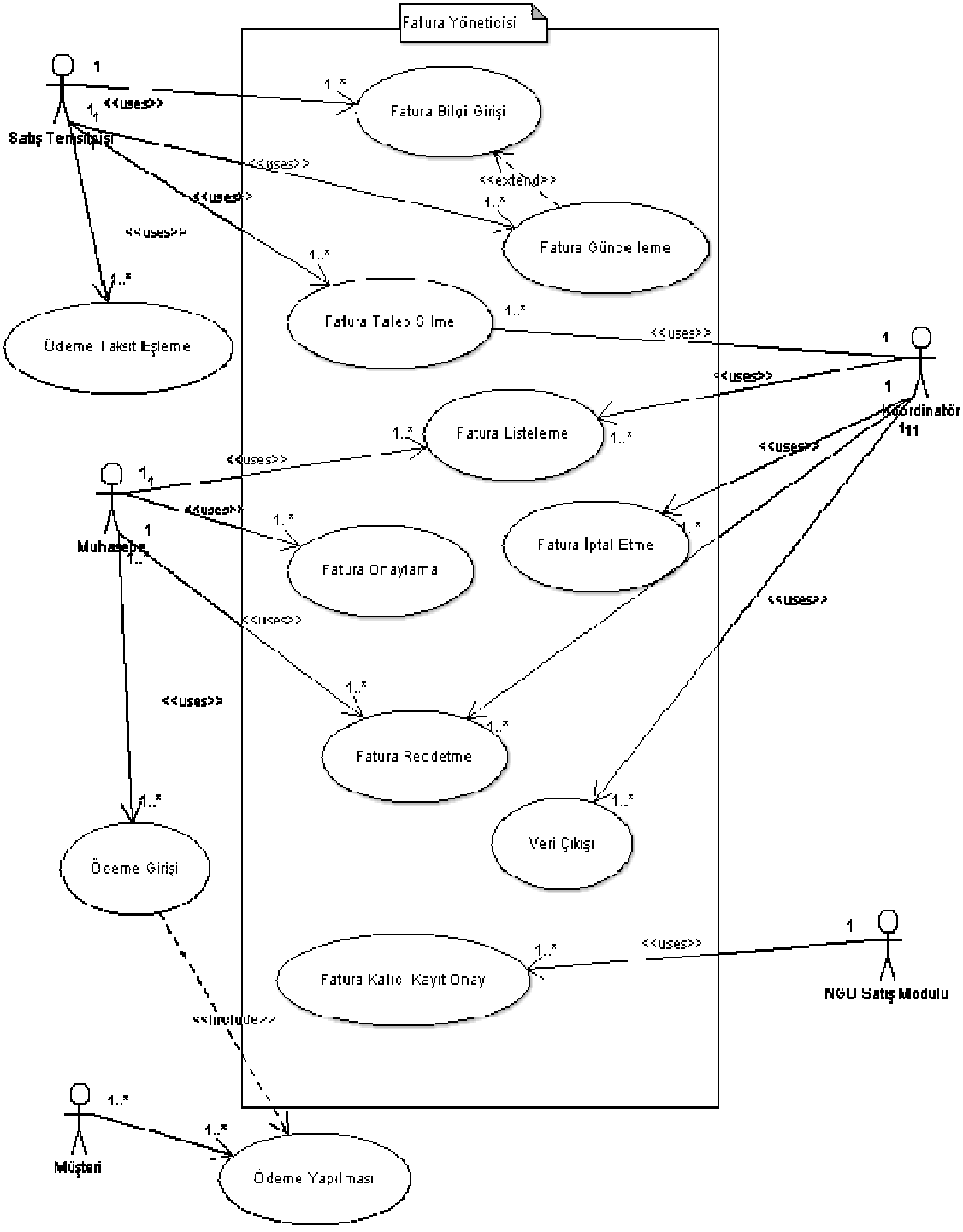
Taksit Düzenleme: Fatura tutarının taksitlendirilmiş miktarlarının yeniden düzenlenmesidir.

7.2.3 Aktörler ve Aktivitelerden oluşan Kullanım Durum Diyagramları

Fatura Takip Sistemi uygulamasında veritabanı detay işlemlerinin işlevsel kodlarda ayrıştırılması böylece kod karmaşasının önlenmesi esas alınmıştır. Bu nedenle ileriki bölümlerde detaylı bir şekilde anlatılacağı gibi bir Veri Erişim Nesnesi (Data Access Object - DAO) kullanılmıştır. Sistemde başlı başına iki alt modül bulunmaktadır. Bunlar Fatura ve Ödeme modülleridir. Fatura'ya ve Ödeme'ye ilişkin işlevler ilgili modül adı ile başlayan (Fatura Yöneticisi ve Ödeme Yöneticisi) ara nesneler vasıtası ile veritabanı ile ilişkili DAO nesnesi ile etkileşirler. Burada asıl işlevi yapan başlıca nesneler bu ara nesnelerdir. (Fatura ve Ödeme Yöneticileri). Bu ara nesnelerin Kullanım Durum Diyagramları Şekil 7.1 ve Şekil 7.2'de görülmektedir.

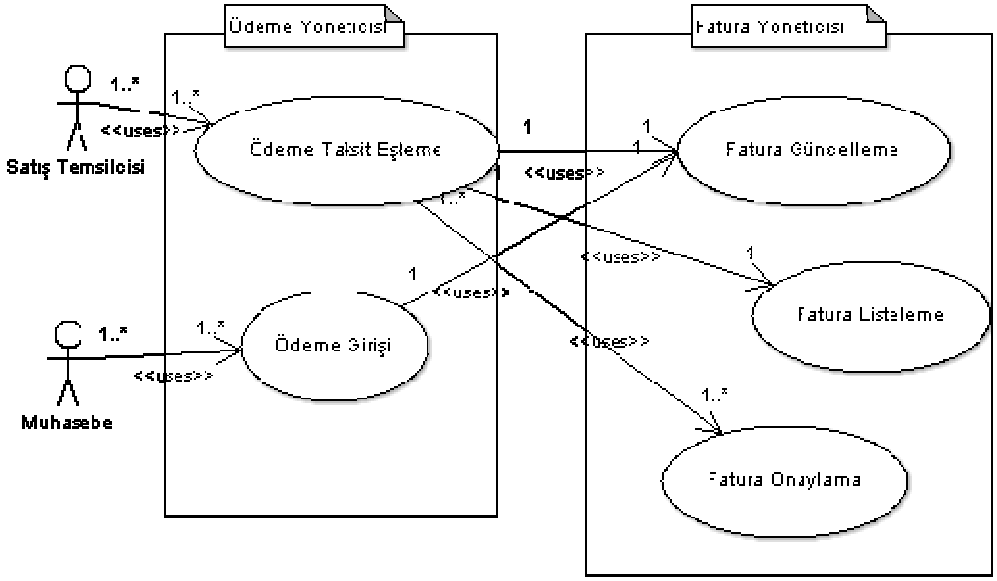
Bu bölümde aynı zamanda FTS uygulamasının daha iyi anlaşılabilmesi için, belirlenmiş olan senaryoların süreç akış diyagramları da verilmiş, bu sayede uygulamanın işlevselliğinin daha net ortaya konulması hedeflenmiştir.

FaturaYoneticisi: Faturalamalar ile ilgili işlevleri barındıran ara metoddur.



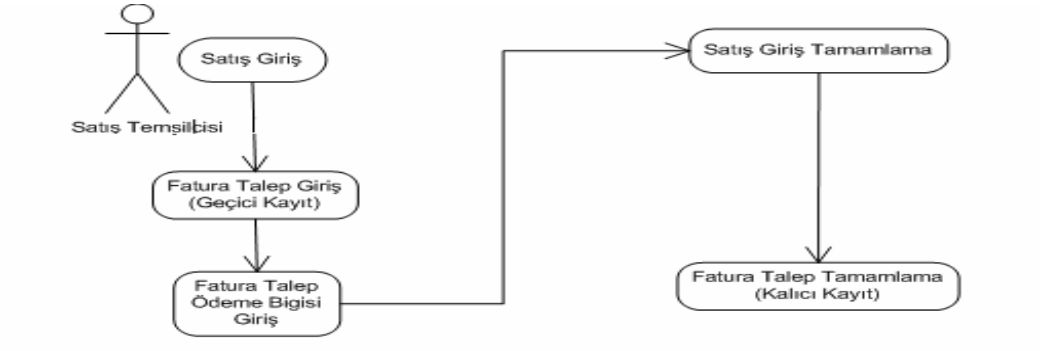
Şekil 7.1 FaturaYoneticisi FTS UML kullanım durum diyagramı

OdemeYoneticisi: Ödemeler ile ilgili işlevleri barındıran ara metoddur.

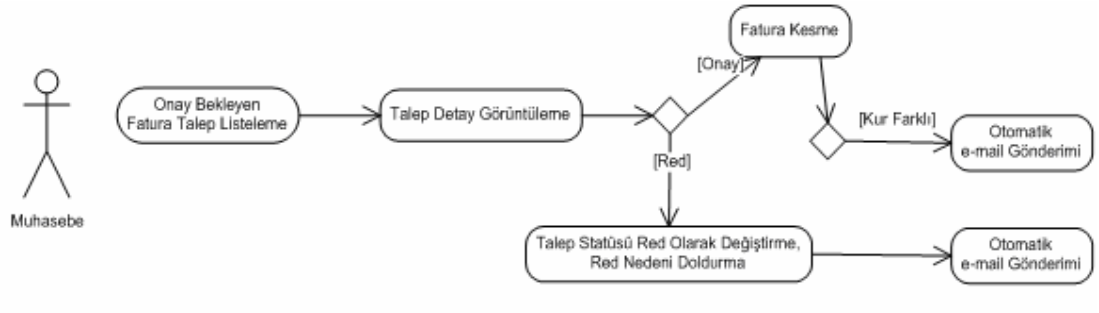


Şekil 7.2 OdemeYoneticisi FTS UML kullanım durum diyagramı

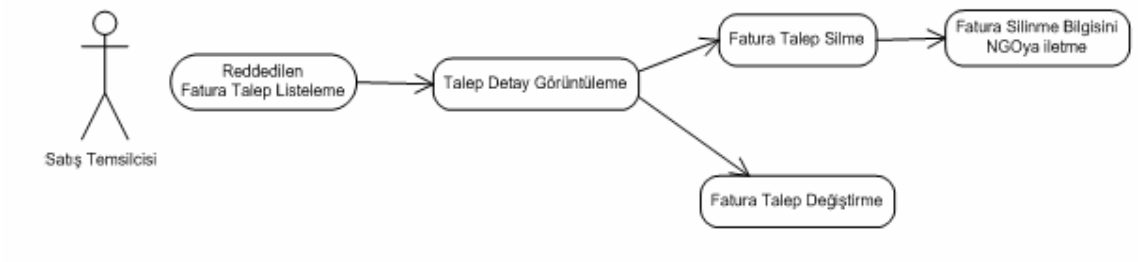
Kullanım durum diyagramlarında görülen senaryoların akış şemaları aşağıdaki gibidir:



Şekil 7.3 Fatura talep giriş akış diyagramı

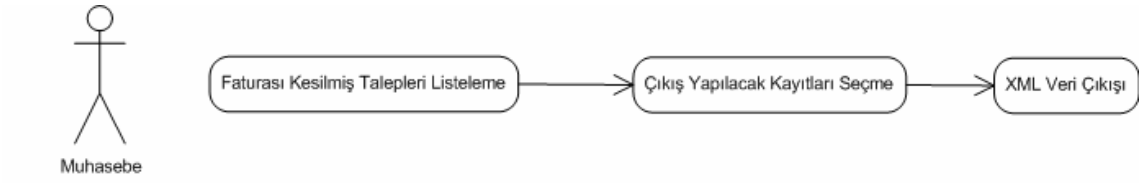


Şekil 7.4 Onay bekleyen faturalar akış diyagramı

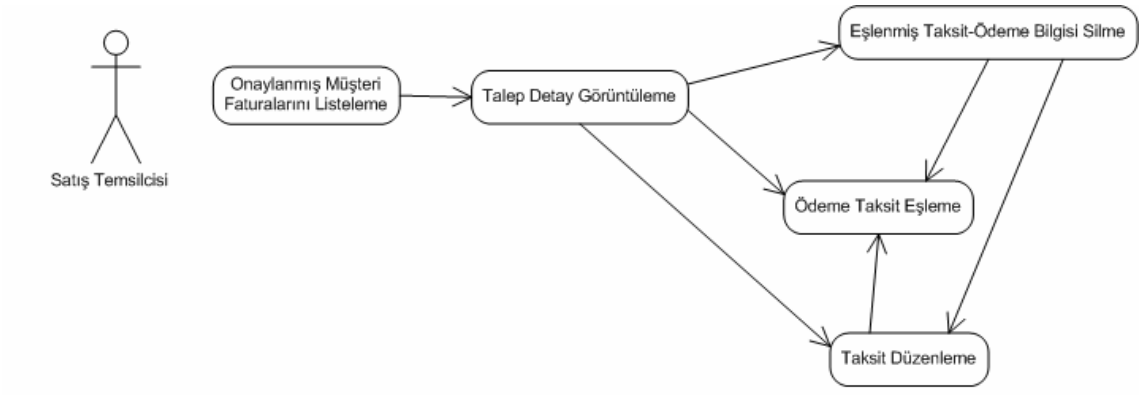


Şekil 7.5 Reddedilen faturalar akış diyagramı

Şekil 7.6 Onaylanmış faturalar akış diyagramı



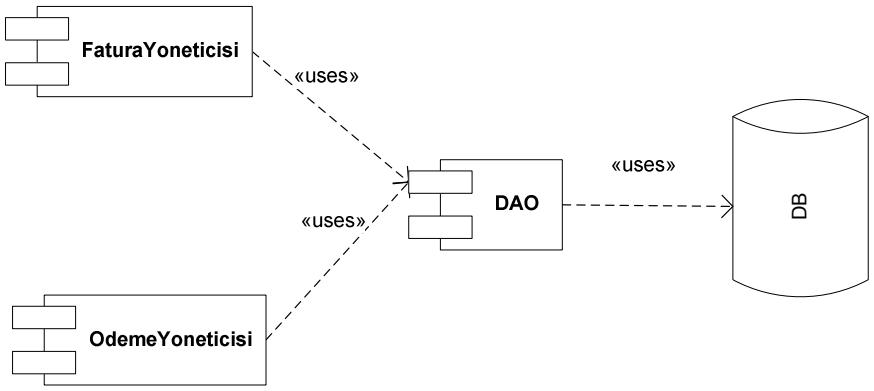
Şekil 7.7 XML veri çıkışı akış diyagramı



Şekil 7.8 Ödeme taksit eşleme

Sistemde veritabanı işlemlerinin uygulama kodlarından bağımsız ara nesneler aracılığıyla yapılması tasarlanmıştır. Böylece veritabanı kayıt okuma, silme, yazma gibi işlemlerin gereksiz detaylarından arınmış bir şekilde uygulama geliştirilmesi esas alınacaktır. Tüm veritabanı temel işlemlerinin yapıldığı bir alt sınıf olan DAO (Data Access Object) sınıfı oluşturulmuş ve yaratılacak olan FaturaYoneticisi sınıfının bu sınıf ile bağlantı kurarak

gerekli işlemlerin gerçekleştirilmesi sağlanacaktır. Genel yapının teknik olarak gösterimi Şekil 7.9'da görülmektedir.



Şekil 7.9 Veritabanı nesnelerine erişim teknik diyagramı

7.3 Senaryoların Ardışıl Diyagramları

Bu bölümde, FTS süreçlerinin ve işlevlerinin daha iyi ifade edilebilmesi amacıyla ardışıl diyagramları yer almaktadır. Örnek uygulamada öncelikle nesneye yönelik tasarım ile işlevsel gereksinimler ele alınacak, daha sonraki bölümde ise ilgilerin tanımlanması ve diyagramlara eklenmesiyle sistem son halini alacaktır. Bu bölümde ele alınacak temel ardışıl diyagramları şöyle sıralanabilir:

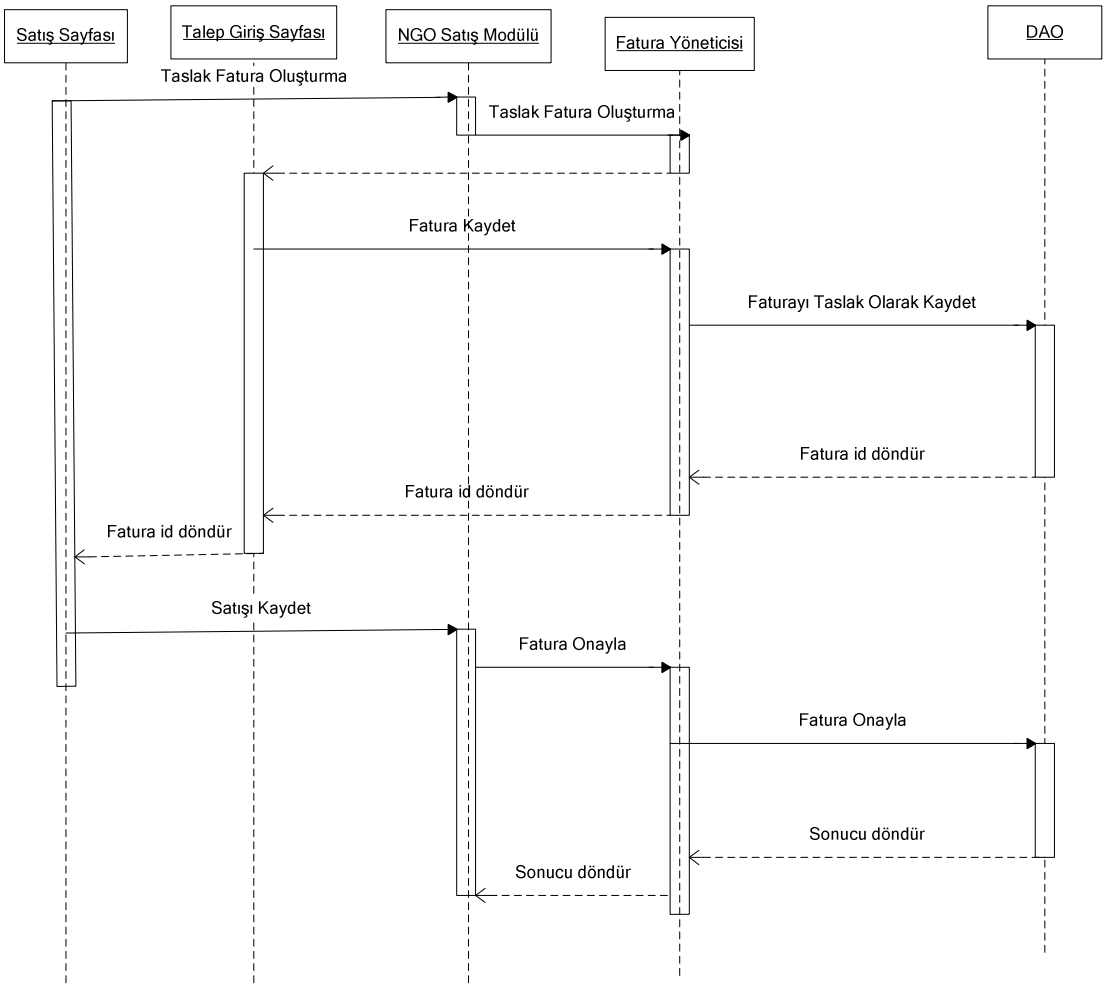
Fatura Talep Ardışıl Diyagramı: Satış sürecinin ayrılmaz bir parçası olan fatura oluşturma, öncelikle taslak fatura talebi düzenlenmesi ile başlar. Fatura talep formu taslak halinde kaydedildikten sonra sistem tarafından otomatik olarak bir fatura numarası oluşturulur. Sürecin sonlanabilmesi, oluşturulmuş olan taslak faturanın satış temsilcisi tarafından onayının alınması gerekmektedir. Onay verilip fatura muhasebe onayı bekleyen duruma geçtiğinde satış süreci tamamlanır (Şekil 7.10).

Fatura Onay/Red Ardışıl Diyagramı : Onay bekleyen durumdaki faturalar muhasebe tarafından listelenebilir ve incelendikten sonra muhasebe tarafından onaylanabilir veya red sebebi girilerek reddedilir (Şekil 7.11).

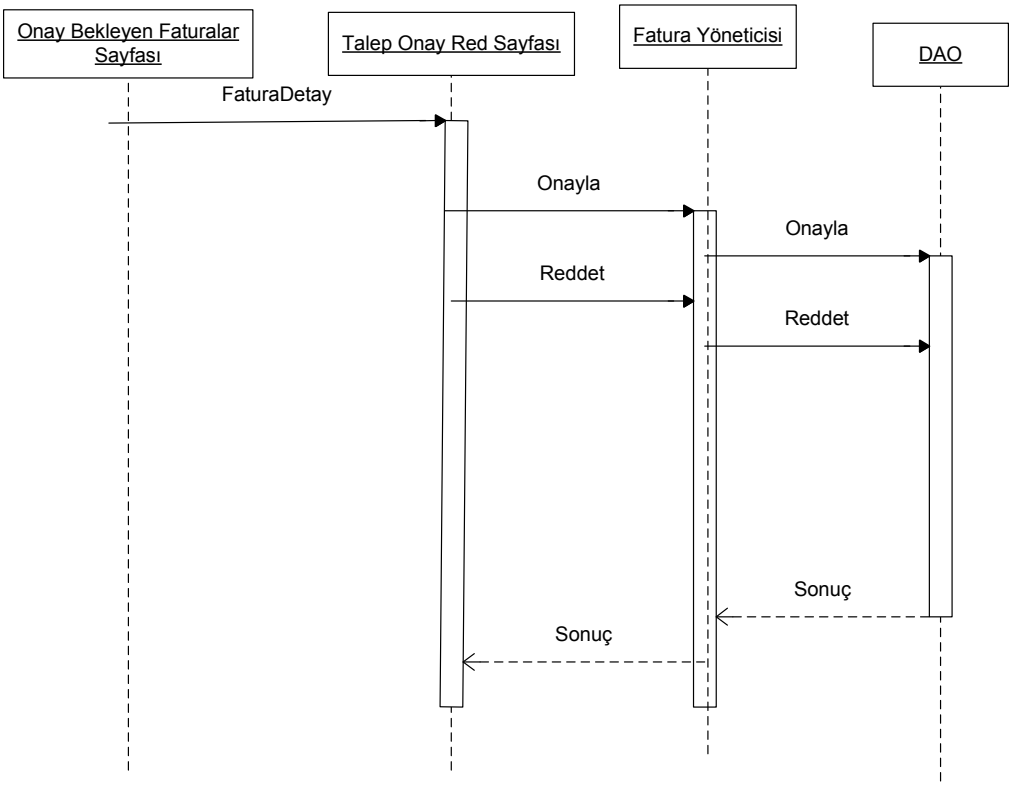
XML Veri Çıkışı Ardışıl Diyagramı : Onaylanmış fatura taleplerinden seçilen faturalar daha önceden belirlenmiş XML formatına dönüştürülür ve içerikleri bu formatta farklı uygulamalarda kullanılabilir (Şekil 7.12).

Ödeme Giriş Ardışıl Diyagramı: Ödeme giriş ekranından, müşteriler tarafından yapılmış ödemeler muhasebe tarafından sisteme girilir (Şekil 7.13).

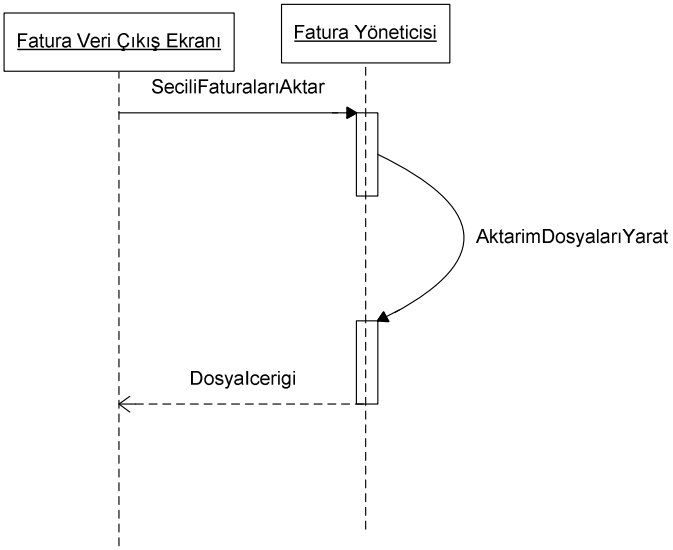
Ödeme Taksit Eşleme Ardışıl Diyagramı: Ödeme Taksit Eşleme ekranında seçilmiş olan fatura ile ilgili taksitlerin ödendi ödenmedi bilgileri ve yapılan ödemeler alt alta listelenirler. Yapılan ödemeler ile taksit tutarlarının eşleştirilmesi bu ekran ile gerçekleştirilir. Seçilen öceme ile seçilen taksit(ler) tutarları uyuşuyor ise birbirleri ile eşleştirilebilirler ve böylece ilgili taksit(ler) ödendi olarak işaretlenir. Burada yapılan ödemeler ile fatura taksitleri eşleştirilebileceği gibi eşlenmiş olan taksit-ödeme çiftleri de iptal edilebilir. Böylece taksit tutarları yeniden ödenmedi statüsüne geçerler. Bu ekranda aynı zamanda taksit tutarların yeniden düzenlenmesi işlevi de gerçekleştirilebilir (Şekil 7.14).



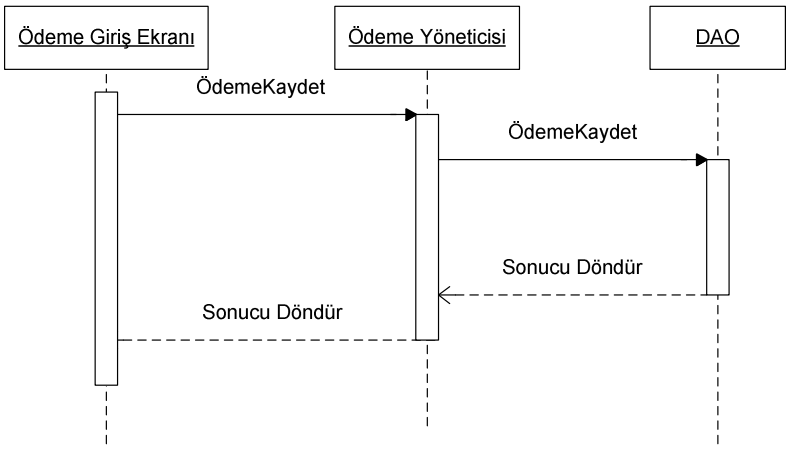
Şekil 7.10 Fatura talep ardışıl diyagramı



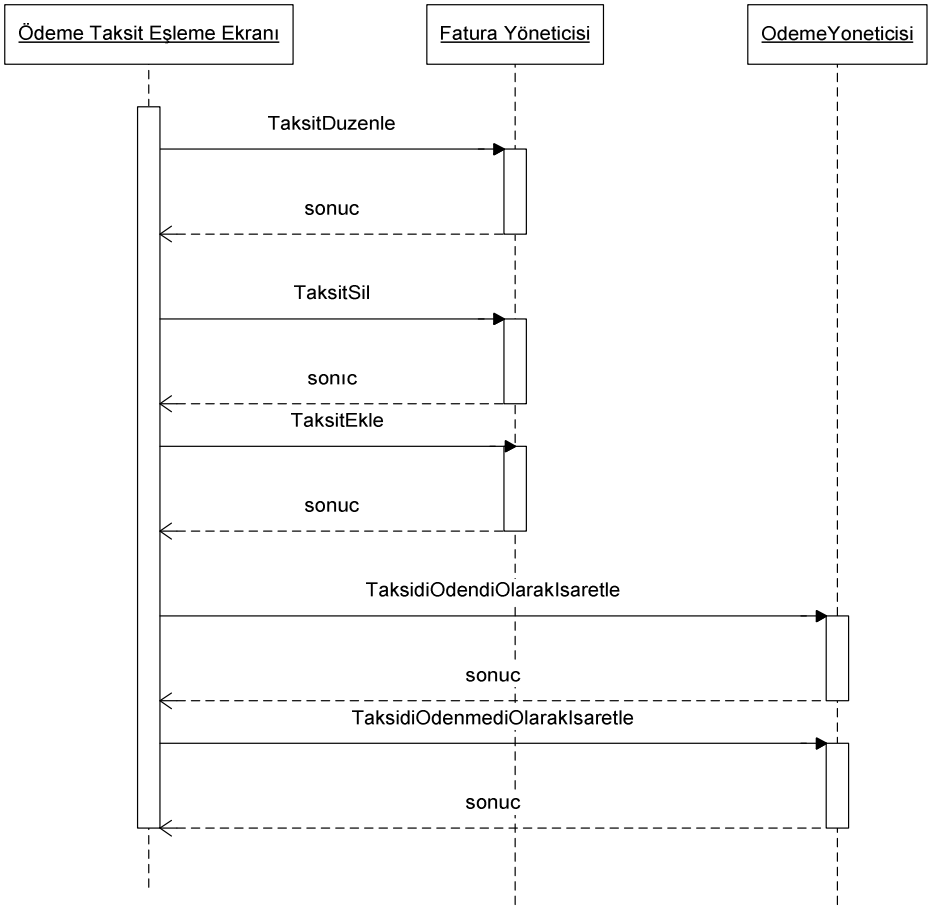
Şekil 7.11 Fatura onay/red ardışıl diyagramı



Şekil 7.12 XML veri çıkışı



Şekil 7.13 Ödeme giriş ardışıl diyagramı



Şekil 7.14 Ödeme taksit eşleme ardışıl diyagramı

7.4 İlgilerin Tariflenmesi

Gereksinimlerin Analizi sırasında geliştirilecek olan sistemle ilgili tüm isterler listelenmiştir. Bu isterlerin bir kısmı sistemden beklenen esas fonksiyonalite, diğer bir kısmı ise geliştirilecek olan tüm fonksiyonaliteyi etkileyecek olan ilgilerdir. Bölümün ilk kısmında belirtilen "Gereksinimler" göz önünde bulundurularak tasarlanmış ilgiler, Çizelge 7.1, Çizelge 7.2 ve Çizelge 7.3'te görülmektedir.

Bu ilgiler, gereksinim analizi sırasında çıkarılmış olan gereksinimlerden, sistemden beklenen esas işlevselliği enine kesen ve esas işlevsellikle birlikte arka planda çalışması gereken fonksiyonelliktir.

Çizelge 7.1 Mail Gönderme

Enine Kesen İlgi	Otomatik E-posta Gönderimi
Tanım	Sistem, fatura talebi oluşturma, onaylama, iptal, reddetme ve silme durumlarında tanımlı kullanıcı listesine e-posta göndermelidir.
Öncelik	Yüksek
Gereksinim Listesi	G1,G2,G3,G5,G6,G8
Model Listesi	Senaryolar: Fatura Talebi Oluşturma, Fatura Talebi Silme, Fatura Talebi Onaylama, Fatura Talebi Reddetme ve Fatura Talebi İptal Etme

Çizelge 7.2 Veritabanı İşlemleri

Enine Kesen İlgi	Veritabanı İşlemleri
Tanım	Sistemin, tüm veritabanı kayıt, okuma ve silme işlerini esas modüllerden soyutlanmış bir ara katman tarafından gerçeklemesi
Öncelik	Yüksek
Gereksinim Listesi	G1, G2, G5, G6, G7, G8, G9, G10, G11, G12
Model Listesi	Senaryolar: Fatura Talebi Oluşturma, Fatura Talebi Silme, Fatura Talebi Onaylama, Fatura Talebi Reddetme ve Fatura Talebi İptal Etme, Ödeme Girişleri, Fatura Taksit Eşleme, Taksit Güncelleme

Çizelge 7.3 Günlük Tutma Sistemi

Enine Kesen İlgi	Günlük Tutma
Tanım	Sistem, tüm işlemlerden sonra işlemi yapan kişi, işlem yapılan tarih ve yapılan işlem bilgilerini kaydetmelidir.
Öncelik	Yüksek
Gereksinim Listesi	G1, G2, G3, G4, G5, G6, G8, G9, G10, G11, G12
Model Listesi	Senaryolar: Fatura Talebi Oluşturma, Fatura Talebi Silme, Fatura Talebi Onaylama, Fatura Talebi Reddetme ve Fatura Talebi İptal Etme, Ödeme Girişleri, Fatura Taksit Eşleme, Taksit Güncelleme, Fatura Taksit Eşleme

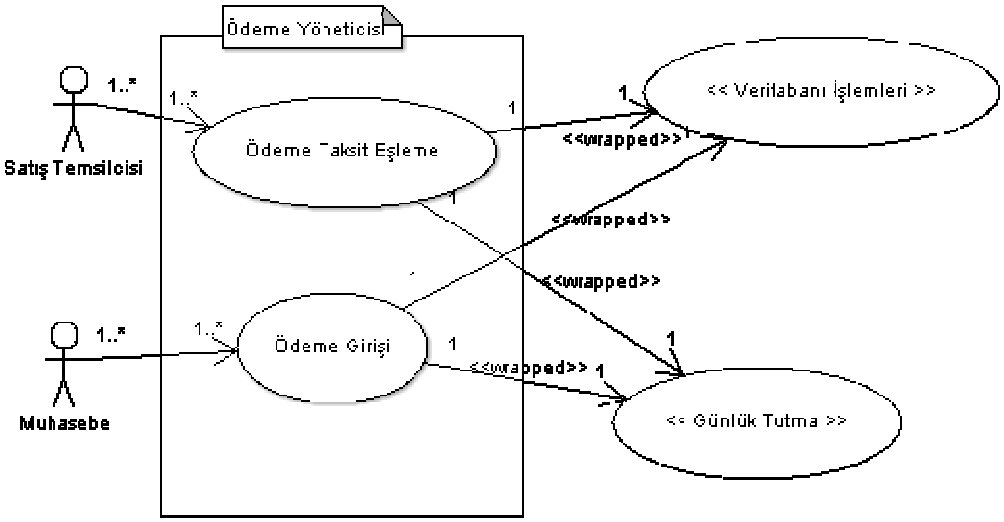
7.5 İlgilerin Kullanım Durum Diyagramları ile Birleştirilmesi

Gereksinim Analizi sırasında ayrıştırılmış olan işlevlerin ilgiler ile birleştirilmesinden sonra oluşturulan UML diyagramları bu alt bölümde incelenecektir.

FaturaYoneticisi ara nesnesinin ilgiler ile bütünleştirilmiş Kullanım Durum Diyagramı Şekil

7.15'te görülmektedir.

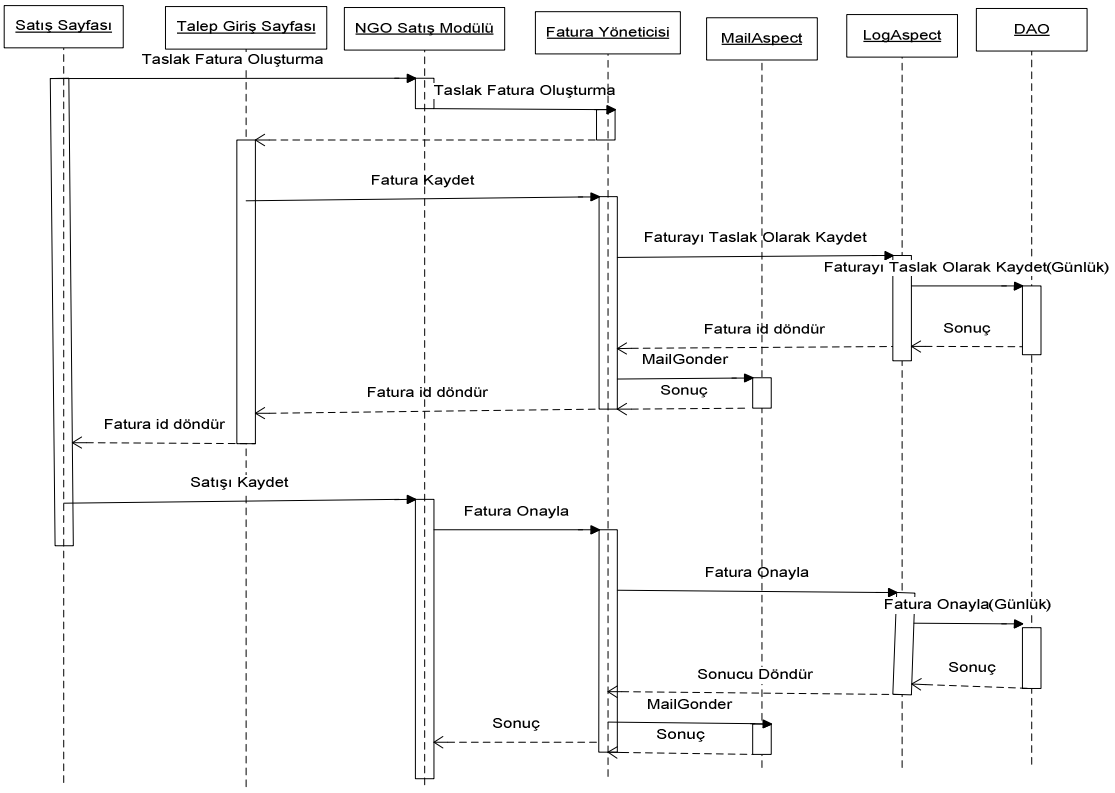
Şekil 7.15 İlgiler ile birleştirilmiş FaturaYoneticisi kullanım durum diyagramı



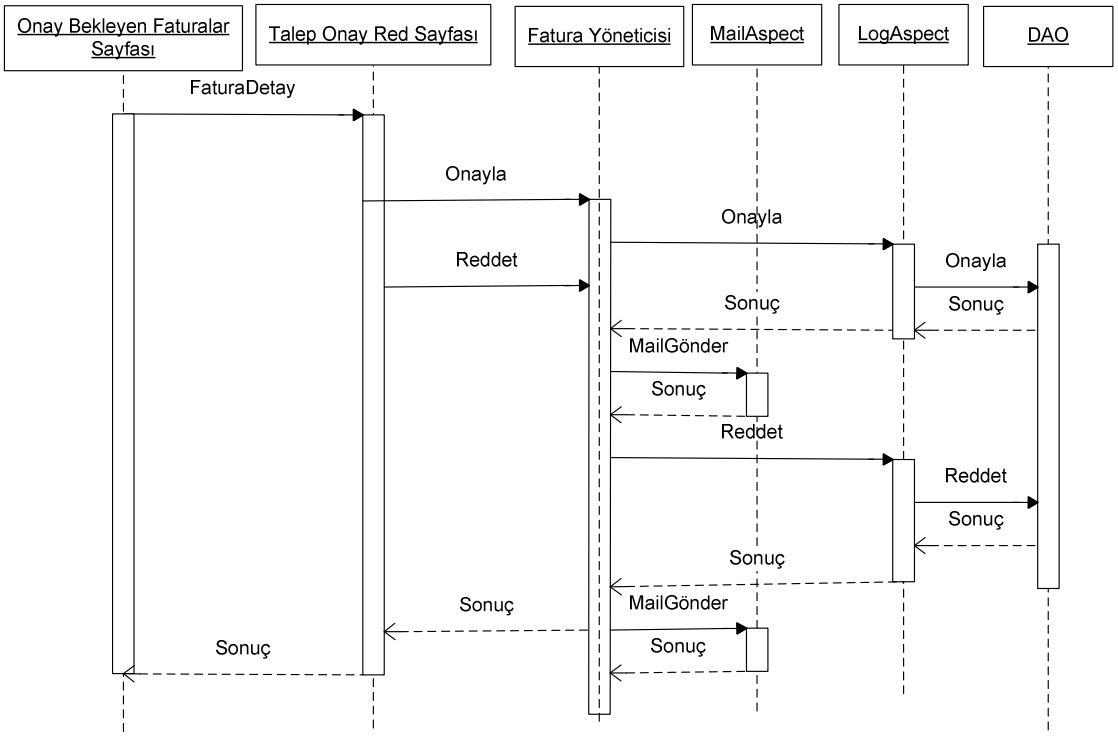
Şekil 7.16 İlgiler ile birleştirilmiş OdemeYoneticisi kullanım durum diyagramı

7.6 İlgilerin Ardışıl Diyagramları ile Birleştirilmesi

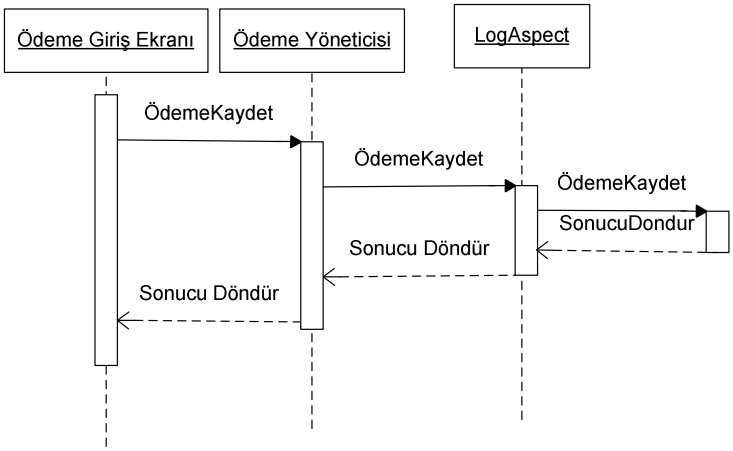
Bu bölümde, bölüm 7.3'de Nesneye Yönelik Yaklaşım ile çizilmiş olan ardışıl diyagramların ilgiler ile entegre edilmiş sürümleri yer almaktadır.



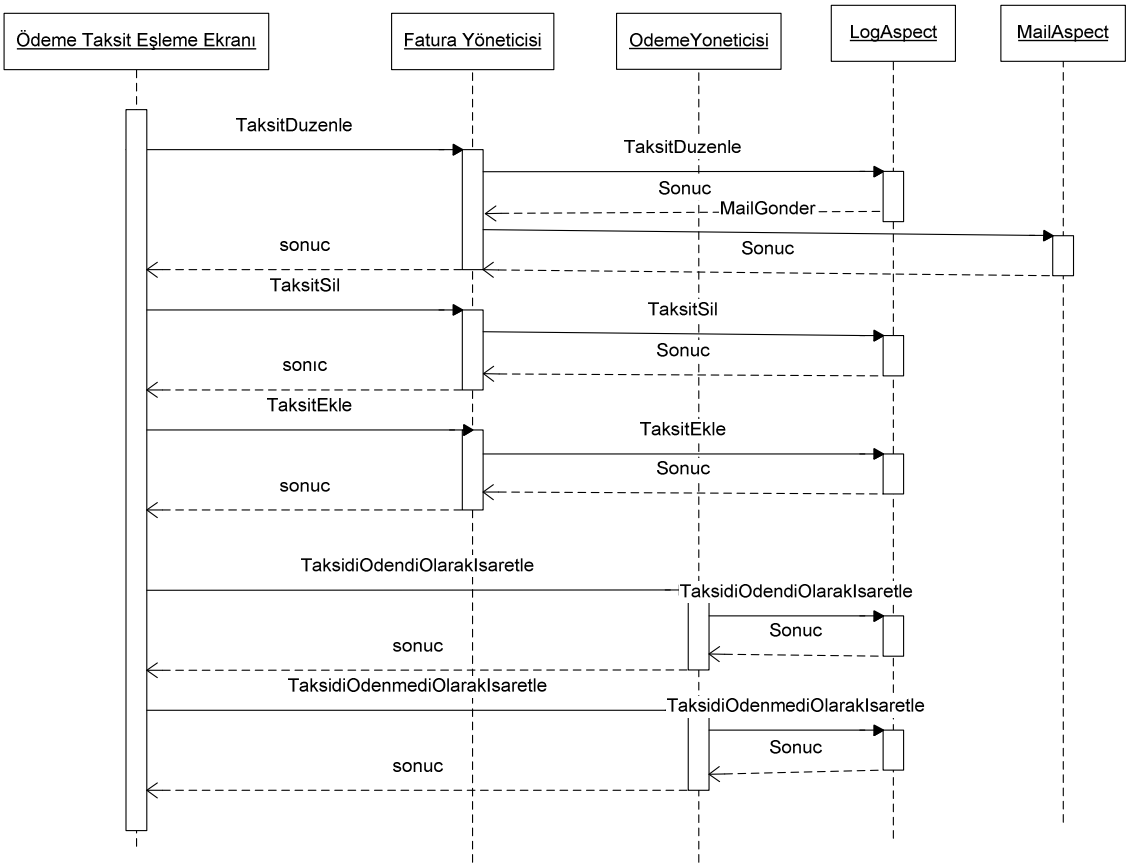
Şekil 7.17 Fatura Talep Kaydı ilgiler ile birleştirilmiş ardışıl diyagramı



Şekil 7.18 İlgiler ile birleştirilmiş fatura onay/red ardışıl diyagramı



Şekil 7.19 İlgiler ile birleştirilmiş ödeme kaydetme ardışıl diyagramı



Şekil 7.20 İlgiler ile birleştirilmiş ödeme taksit eşleme ardışıl diyagramı

7.7 İlgilerin Tasarım Kalıpları ile Uygulanması

Gereksinimlerin analizi esnasında belirlenmiş olan sistem ilgileri standart IYP teknikleri kullanılarak uygulanabilecekleri gibi, getirdiği çözümlerin etkileri kanıtlanmış olan tasarım kalıpları ile de gerçekleştirilebilir. Java dilinde uygulanan tasarım kalıplarından bazıları problemlere getirdikleri özel çözümlerin yanı sıra bazen kendileri de enine kesen ilgi problemlerine yol açmaktadırlar. Bu nedenle, belirli problemlere özel çözümler getirmiş olan tasarım kalıplarının da İYP ile uygulanması etkin ve karmaşıklıktan uzak bir yazılım geliştirmeyi sağlamaktadır.

Yeniden geliştirilmekte olan FTS uygulamasında, enine kesen ilgiler olarak belirlenmiş başlıca ilgiler:

- Fatura işlemlerinden sonra otomatik e-posta gönderimi
- Veritabanı işlemlerinin uygulamadan soyutlanması
- Sisteme yapılan tüm girişlerin günlüğünün tutulması.

Bölüm 6'da üzerinde durulan tasarım kalıplarının İYP uygulamaları ve kullanım alanları göz önünde bulundurularak uygulamada kullanılacak olan ilgilerin hangi tasarım kalıpları ile uygulanacağı tasarlanabilir.

7.7.1 Otomatik E-posta Gönderme

FTS uygulamasında fatura talep kayıtları ile ilgili tüm durum değişiklikleri hakkında tüm tanımlı kullanıcı listesine e-posta gönderilmesi talep edilmektedir. Standart Java uygulaması ile geliştirildiğinde tüm fatura işlemlerinin gerçekleştiği kodların altında e-posta gönderim ile ilgili kodların da yer alması gerekmektedir. Bu da kod karmaşasına sebep olabileceği gibi, e-posta gönderim kodlarıyla ilgili yapılacak bir değişikliğinde tüm bu kodların geçtiği yerlerde kod düzeltmesi yapmayı gerektirir. Bu nedenle e-posta gönderim işlemi, programın genelini etkileyen bir enine kesen ilgi olarak belirlenmiştir. E-posta gönderme ilgisinin tasarım kalıplarıyla gerçekleştirilmesi kod yayılımı ve dağıtımını engelleyecektir. Bunun için 6. bölümde bahsedilen tasarım kalıplarından Gözlemci Tasarım Kalıbı uygun bulunmuştur. Gözlemci tasarım kalıbı, gözlenen nesnelerin durum değişikliklerinin algılanıp gerekli işlemlerin yapılmasını sağlamaktadır. Bu yapı temel alınarak fatura taleplerinin durum değişikliklerinin algılanarak otomatik e-posta gönderiminin tetiklenmesi, gözlemci tasarım kalıbı ile gerçeklenmiştir. Gözlemci Tasarım Kalıbının yapısı itibari ile uygulama içerisinde gözlemci ve gözlenen nesnelerin olması gerekmektedir. E-posta gönderim esnasında gözlenen nesne "Fatura" nesneleri gözlemleyen nesne ise MailGonderim sınıfı ise gözlemci konumundadır. Bu durumda Fatura nesnelerinde meydana gelen bir durum değişikliği, MailGonderim nesnesindeki e-posta oluşturma ve oluşturulan e-postayı gönderme metodlarını tetikleyecek ve enine kesen ilgi kodunun çalıştırılması sağlanacaktır.

7.7.2 Veritabanı İşlemleri

Uygulamada faturaların ve ödemelerin sisteme kaydettirilmesi, kaydedilmiş faturaların listelenmesi, durumlarının değiştirilmesi, ödeme işlemlerinin yapılması, günlük kayıtlarının tutulması gibi veritabanından bilgi okuma, veritabanına bilgi yazma gibi işlemlerin gerçekleştirilmesi bir enine kesen ilgi olarak tasarlanmıştır. Bu enine kesen ilgi kodlarının uygulama içerisindeki kodlarla birlikte yazılıp kod yayılımına ve karmaşasına sebebiyet vermemesi için işlevsel sınıflardan soyutlanmaları gerekmektedir. Uygulamada, veritabanı bütünlüğü için eklenmiş olan Hibernate çatısı kod stili ile oluşturulmuş bir DAO ara sınıfı kullanılmaktadır. Bu ara sınıf uygulama işlevleri ile direk olarak bağlantılı olan OdemeYoneticisi ve FaturaYoneticisi sınıfları ile iletişim halindedir. (Şekil 1.9) DAO arasınıfı içerisinde verilerin kaydedilmesi, okunması ve çeşitli sql komutlarının kullanılan

hibernate çatısına uygun yazılmış olan metodları mevcuttur. Bu ara sınıfla iletişim halinde bulunan FaturaYoneticisi ve OdemeYoneticisi sınıfları tüm uygulamanın veritabanı işlemleri ile ilişkili tek bağlantı noktaları olmalarından ötürü bu sınıfların yaratılmış tek bir kopyaları olması tüm işlemlerin bu kopyalar üzerinden gerçekleştirilmesi veritabanı bütünlüğü ve bellek kullanımı açısından gerek şarttır. Bir sınıfın tek bir kopyasının yaratılmasının istendiği durumlarda kullanılan Yegane Tasarım Kalıbı, Veritabanı İşlemleri ilgisi için kullanılabilir. Bu tasarım kalıbının İYP yaklaşımıyla uygulanmasıyla tüm veritabanı işlemi gerektiren sınıflar tarafından (eğer yaratılmamışsa) FaturaYoneticisi veya OdemeYoneticisi sınıflarının bir kopyasının yaratılabilmesi sağlanacaktır.

7.7.3 Günlük Tutma

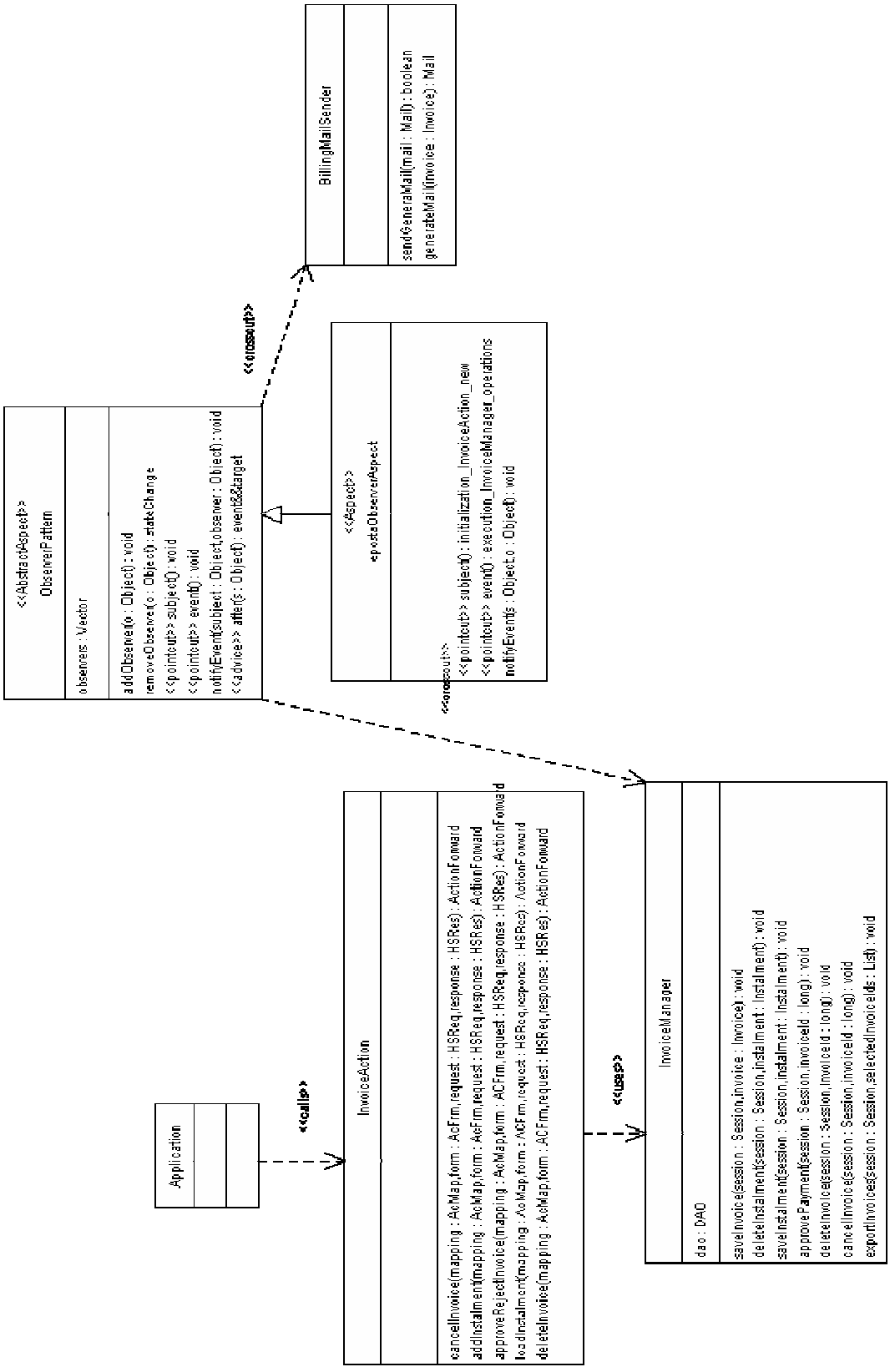
Uygulamaya eklenecek olan günlük tutma işlevi sistem kullanıcıları tarafından yapılacak olan tüm işlemlerin kullanıcı ve tarih bazında kayıtlarının tutulmasını kapsamaktadır. Yapılacak olan işlemlerin sonunda günlüğünün tutulması, günlük tutma kodunun tüm metodlar içerisine gömülmesini gerektirir. Bu özelliğinden dolayı günlük tutma işlevi de sistemden bir enine kesen ilgi olarak tanımlanmıştır. Tüm işlem komutlarından sonra tanımlanması gereken bir ilgi olan günlük tutma, işletilecek olan uygulama komutlarının kaydedilmesi içerdiğinden dolayı tasarım kalıpları bölümünde anlatılan Komut Tasarım Kalıbı kullanılarak gerçeklenebilir. Komut Tasarım Kalıbı'nın İYP ile uygulanması ile işletilecek olan tüm komutların tek bir icra noktası ile gerçeklenmesi sağlanabilir. Böylece uygulama, tasarım kalıbının enine kesen ilgi özelliğinden de yalıtılmış olur. Yaratılacak olan günlük tutma sınıfı, sistem tarafından gerçekleştirilen temel işlemlerin tanımlanmış oldukları FaturaYonetici ve OdemeYoneticisi sınıflarına uygulanacak ve bu sınıflar içerisindeki tüm metodlar ile birlikte çalışması sağlanacaktır.

7.8 İlgilerin İYP Tasarım Kalıpları ile Uygulanmaları ve Sınıf Diyagramları:

Çizelge 1.1, Çizelge 1.2 ve Çizelge 1.3'le tariflenen sistem enine kesen ilgilerinin, uygulama koduna dönüştürülmeden önce teknik tasarımın yapılması ve sınıf diyagramlarının çizilmesi hem tasarımın anlaşılması hem de uygulamaya geçirilmesi açısından kolaylık sağlamaktadır.

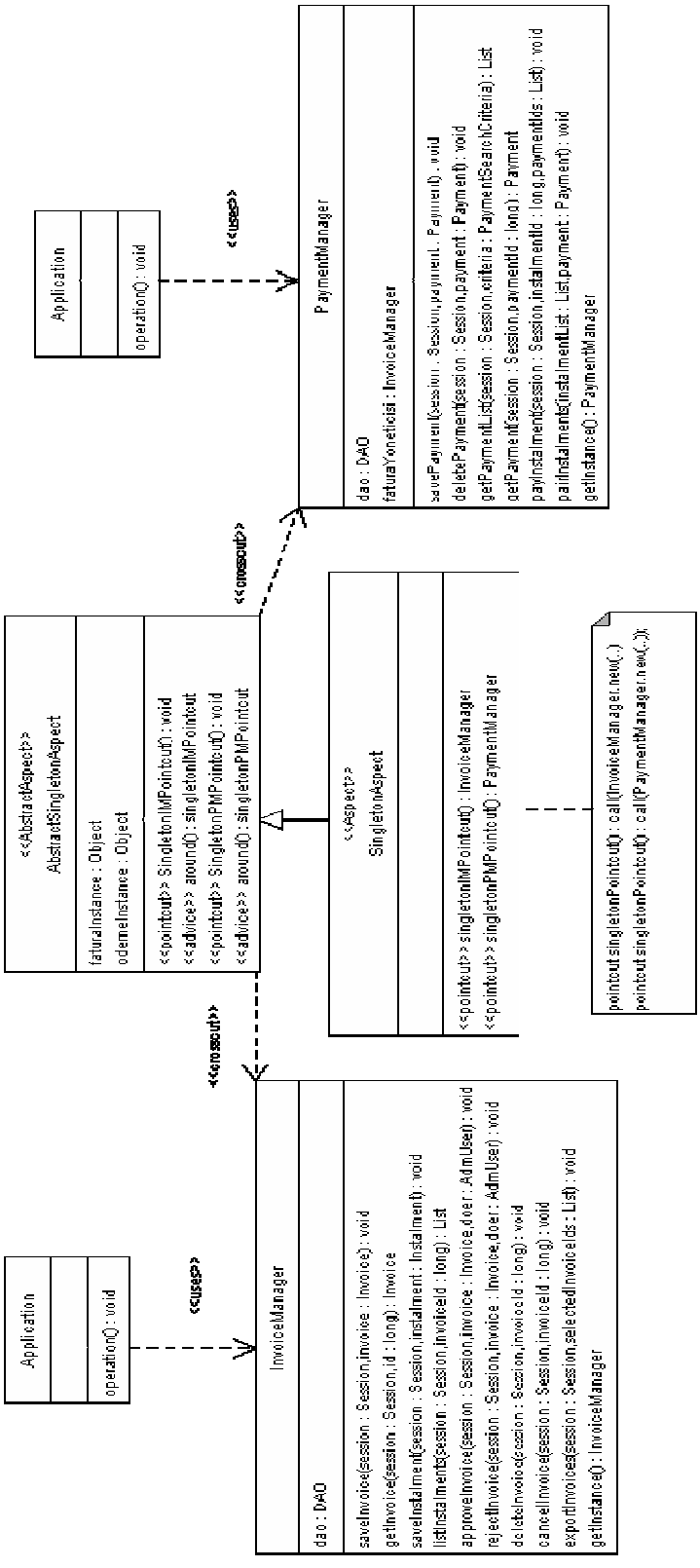
İlgiler için belirlenmiş olan tasarım kalıplarının İYP uygulamaları bu kalıpların getirdikleri çözümler içerisinde oluşabilecek enine kesen ilgiler problemini çözmek için gereklidir. Tasarım Kalıplarının İYP ile gerçekleştirimleri Şekil 1.23, Şekil 1.24 ve Şekil 1.25'te görülmektedir.

7.8.1 Otomatik E-posta Gönderim İlgisi



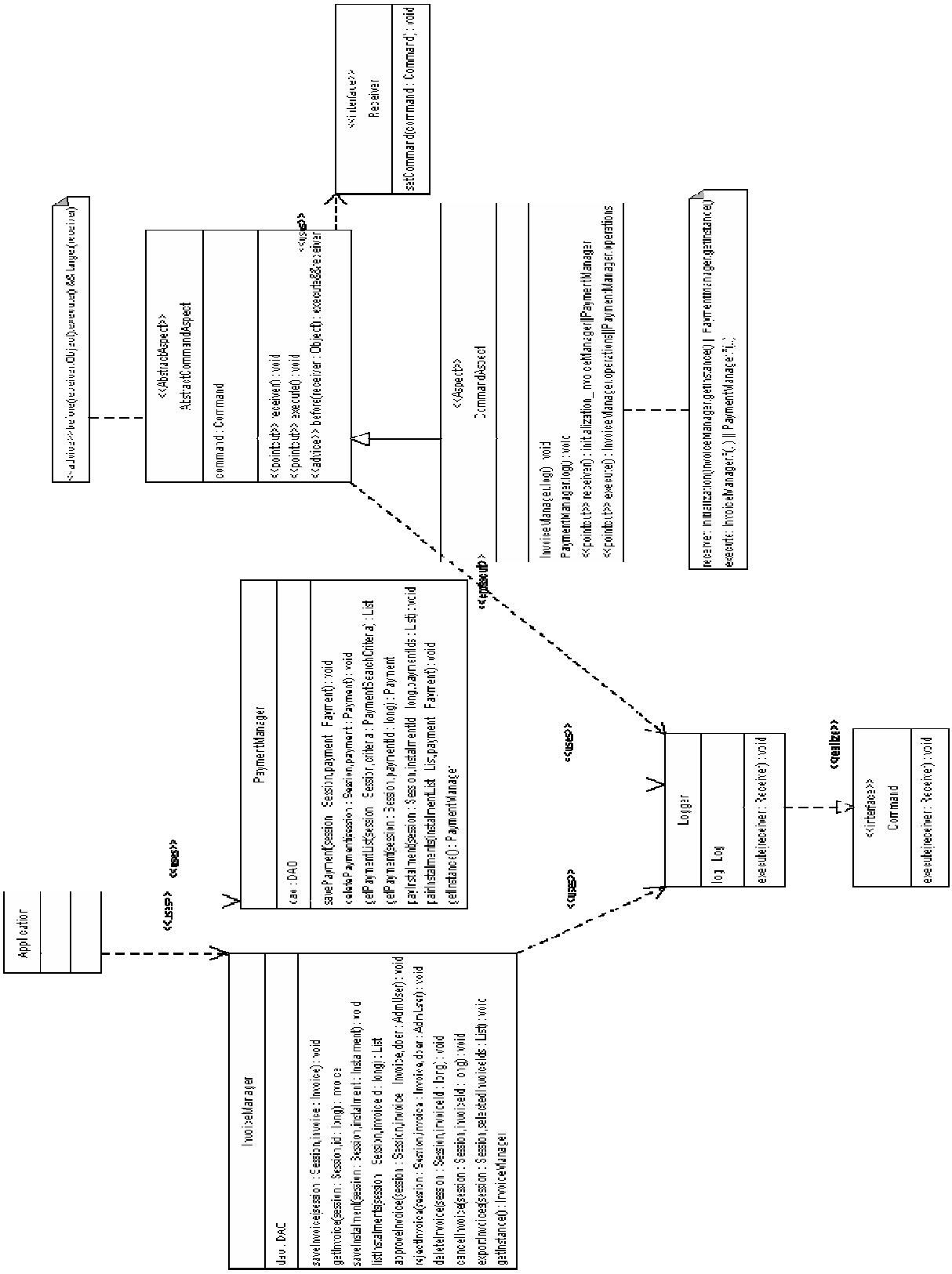
Şekil 7.21 Otomatik e-posta gönderim ilgisi sınıf diyagramı

7.8.2 Veritabanı İşlemleri İlgisi



Şekil 7.22 Veritabanı işlemleri ilgisi sınıf diyagramı

7.8.3 Günlük Tutma İlgisi



Şekil 7.23 Günlük tutma ilgisi

7.9 Uygulama için İYP Aracı Seçimi

Tasarımı önceki bölümlerde adım adım yapılmış olan Fatura Takip Sistemi (FTS) uygulamasının İYP ile gerçeklenmesi için Bölüm 5'te tüm yönleriyle ele alınmış olan araçlardan hangisinin kullanılacağının tespitinde, uygulamanın başlıca özellikleri dikkate alınmalı ve bu özellikler ışığında en etkin geliştirme yapılabilecek ortam seçilmelidir.

- Uygulamanın geliştirme süresinin kısalığı esastır. Çünkü geliştirim süresi arttığında getirdiği maliyet de bağlantılı olarak artmaktadır. Uygulama geliştirim süresi kısalığı, geliştirilen ortamın sağladığı kolaylıklarla doğru orantılı olduğundan seçilecek olan derleyicinin kod yazım özellikleri birinci derecede önem kazanmaktadır. Bu nedenle ilgi yazımı ve hata ayıklaması için gereksiz zaman kaybı maliyeti arttıracaktır.
- Uygulamanın özelliği gereği, taleplere en hızlı şekilde cevap verilebilmeli ve talep edilen işlemler gerçekleştirilebilmelidir. Satış sürecinin tamamlanabilmesi ve onayların verilebilmesi için bu gerek şart bir koşuldur. Enine kesen ilgilerin sistemi en az ölçüde yavaşlatması beklenmektedir. Bu da seçilecek olan ortamın çalışma zamanı hızına bağlıdır. Yani çalışma zamanı maliyeti fazla olmamalıdır.
- Uygulama daha önceden nesneye yönelik yaklaşım ile Eclipse ortamında geliştirildiğinden dolayı, İYP yaklaşımına adapte edilebilmesi kolay olmalıdır.
- Zaman kısıtı ve öğrenme detayları açısından ilgi geliştirim kolaylığı (standart java metodlarına benzer ilgi geliştirme) ve dokümantasyon imkanının olması araç seçiminde önemli rol oynayacaktır.

Uygulamanın bahsedilen bu genel özellikleri göz önünde bulundurularak geliştirme aşamasında java diline entegre edilebilecek 3. parti bileşene sahip olan AspectJ kullanılacaktır. AspectJ barındırdığı statik kontrol mekanizması ve çalışma zamanı en az kaynak harcaması nedeniyle uygulama için en uygun araç olarak görülmüştür.

7.10 İYP'nin Tasarım Kalıplarına Uygulanmasının Yararları

İYP Kullanımının en temel faydası üzerinde çalışılan tasarım kalıbı kodunun yerelleştirilme özelliği kazanmasıdır. Bunun anlamı tasarım kalıbının çoğu durumda tek bir ilgi veya birbiri ile bağlantılı bir çift ilgi tarafından gerçeklenebilmesidir. Kodun tamamının tek bir lokasyondan görülebilmesi birçok pratik fayda sağlar:

- Kodu gören bir geliştirici kalıbı rahatça anlayabilir ve ayırt edebilir.
- Kodun değiştirilmesi gerektiğinde tek bir yerde değişiklik yapılması yeterli olur.
- Tasarım kalıbını içeren ilgiler kalıbın ismi ile adlandırılabilir. Böylece yazılım geliştiriciler için uygulama daha anlamlı bir hal alır.
- İYP'nin tasarım kalıpları üzerindeki yararları, bazı tasarım kalıplarının kod seviyesinde yeniden kullanılabilirliğini sağlamaktadır.
- Kodun yeniden kullanılabilirliğinin sağlanması, oluşturulan soyut ilgi sınıfları ile tasarım kalıbının birçok uygulamada kullanılanılabilmenin sağlanmasıdır. Tasarım kalıplarının uygulandığı sınıfların kodlarında herhangi bir değişiklik yapılmadığından dolayı bu sınıflar tasarım kalıpları kullanılmayan uygulamalarda da kullanılabilmektedir.

8. İYP VE GOF TASARIM KALIPLARI KULLANIM ETKİLERİ

Bu bölümde kullanılan tasarım kalıplarının ve İYP yaklaşımının, geliştirilen FTS uygulamasını nasıl etkilediği anlatılacaktır. Yeni tekniklerin uygulanmasında göz önünde bulundurulacak kriterler 6. Bölümde anlatılan Yerellik, Yeniden Kullanılabilirlik, Birleştirme Şeffaflığı ve Eklenip Ayrılabilirlik'tir. Burada incelenecek kriterler dışında, İYP yaklaşımı ile birlikte gelen karmaşıklıktan uzak kod yazımı, sistemin modülerizasyonunun etkin bir şekilde yapılabilmesi, bakım yapılabilirlik ve uygulama kodlarının değiştirilmeden yeni işlevlerin eklenebilir olması diğer incelenebilecek alt kriterler arasında yer almaktadır.

8.1 Yerellik

FTS Uygulaması boyunca geliştirilmiş olan tüm enine kesen ilgi kodları İYP yaklaşımı kullanılarak, problemin çeşidine uygun bir tasarım kalıbı ile gerçeklenmiştir. NYP yaklaşımı ile geliştirilmiş uygulamaya eklenecek yeni bir işlevselliğin, varolan uygulama kodları içerisine serpiştirilmesi gerekirken, İYP ile gerçekleştirimde yeni eklenecek olan işlevsellik kodu ayrı bir sınıf (ilgi) oluşturularak sisteme eklenmektedir. Bu da yeni işlevselliğin yerelleştirilmesini sağlamaktadır. Uygulama süresince kullanılan Yegane, Gözlemci ve Komut Tasarım Kalıpları öncelikle bir soyut ilgi daha sonra o soyut ilgiyi uygulayan bir ilgi ile gerçekleştirilmişlerdir. Böylece hem kalıbın genel özellikleri ve kuralları soyut ilgide tanımlanmış ve gerçek ilgiyi kullanan uygulamadan soyutlanmıştır, hem de aynı kalıp kullanılarak başka bir işlevsellik gerçeklenmek istendiğinde bu yerellenmiş soyut ilgi kullanılarak yeni eklenecek işlevsellik ilgisi sisteme adapte edilebilmiştir.

FTS Uygulamasında kullanılan Yegane Tasarım Kalıbı, Gözlemci Tasarım Kalıbı ve Komut Tasarım Kalıpları uygulanmasını gerektiren yeni bir işlevsellik eklenmek istendiğinde soyut ilgileri uygulayan yeni bir ilgi yazılması yeterli olacaktır. Bu da hem yazılım geliştirme maliyetini azaltmakta hem de süresini kısaltmakta, aynı zamanda kod karmaşasını da engellemektedir. Nesneye Yönelik olarak gerçekleştirilen kalıplarda bir tasarım kalıbını uygulayan işlevsellik her seferinde tasarım kalıbı kuralını gerçekleyen kodun yazılmasını gerektirmektedir, böylece kod karmaşasına ve yayılımına neden olabilmektedir.

8.2 Yeniden Kullanılabilirlik

FTS Uygulamasında Tasarım Kalıpları ve İYP kullanımı iki yönden yeniden kullanılabilirlik sağlamaktadır:

- İlgi yeniden kullanılabilirliği
- Uygulama sınıfları yeniden kullanılabilirliği

İlgi Yeniden Kullanılabilirliği: Uygulama süresince geliştirilmiş olan tüm ilgiler bir soyut ilgi ve o ilgiyi gerçekleyen bir somut ilgiden oluşmaktadır. Enine Kesen İlgiler problemi için tasarım kalıplarının kullanılmasının sebebi benzer problemlere etkin çözümler getirmesi ve maliyetin azaltılmasıdır. Tasarım Kalıplarının uygulanması sırasında karşılaşılan enine kesen ilgiler problemi içinse bu kalıpların İYP ile uygulanması çözümü getirilmiştir. Örneğin gözlemci tasarım kalıbının uygulandığı mail gönderim uygulamasının NYP gözlemci tasarım kalıbı ile gerçeklendiği düşünülürse, e-posta gönderiminin gerektiği her sınıf kodu içerisine gözlemci kodlarının eklenmesi ve e-posta gönderiminin tetiklenmesi gerekmektedir. İYP ile gerçekleştirimde ise e-posta gönderimini tetikleyecek kod, tasarım ilgisinin içerisinde gerekli metodlardan sonra icra etmek üzere tanımlanır. Uygulamada tasarım kalıplarının soyut ilgiler kullanılarak gerçekleştirilmesi, aynı tasarım kurallarını kullanacak yeni ilgiler geliştirilmek istendiğinde, geliştirilmiş olan bu soyut ilgilerin uygulanarak sisteme eklenmesini sağlayacaktır. Böylece soyut ilgiler yeniden kullanılabilir olmaktadır.

Uygulama Sınıfları Yeniden Kullanılabilirliği: FTS Uygulaması için daha önceden geliştirilmiş olan temel ara sınıflar olan FaturaYoneticisi ve OdemeYoneticisi sınıfları içerisindeki metodlara eklenecek olan yeni işlevsellikler (e-posta gönderimi veya günlük tutulması vb.) İYP kullanılmadan gerçeklendiği takdirde, eklenecekleri sınıfın kodlarında değişiklik yapılmasını gerektirecektir. İlgiler kullanılarak eklenen işlevsellik uygulamanın sınıfları ve metodlarında değişiklik yapılmasını gerektirmediğinden, bu sınıflar ve metodlar farklı uygulamalarda da kullanılabilmektedir.

8.3 Birleştirme Şeffaflığı

İYP yaklaşımı kullanılarak uygulanan tasarım kalıpları uygulama kodlarında herhangi bir değişikliğe neden olmamaktadır. Örneğin metodlardan sonra çalıştırılması gereken günlük tutma kodları ilgiler içerisinde gerçeklendiğinden dolayı daha önceden geliştirilmiş olan uygulama sınıfları ve metodlarında herhangi bir değişiklik yapılmaz. Tasarım Kalıplarının uygulanmasında sınıflar yeni eklenebilecek arayüzler ile ilgiler içerinde genişletilebilir fakat kodlar değişmezler. Bu da uygulama kodları ile ilgilerin birleştirme şeffaflığı olarak tanımlanmaktadır. Uygulama sınıflarının ilgiler tarafından değiştirilmemesi, kod karmaşasını önler ve uygulamada değişiklik yapılmadan yeni işlevselliklerin eklenmesini sağlar. Örneğin Günlük Tutma ilgisine eklenen Command ve Receiver arayüzleri uygulama içerisinde kullanılmış fakat mevcut sınıfların yapısını değiştirmemiştir.

8.4 Eklenip Ayrılabilirlik:

FTS Uygulaması için geliştirilmiş olan ilgiler Birleştirme Şeffaflığı özelliği sayesinde uygulama kodları içerine yayılmazlar ve ayrı sınıflar halinde tanımlanırlar. Uygulamanın bütününde işlevselliğin eklenmesi ve ayrılması sadece ve sadece ilgi kodunun değiştirilmesine bağlıdır. Bunun anlamı da ilgiler ile uygulama kodları birbirlerine az bağımlıdırlar. Bu az bağımlılık uygulama kodlarına yeni kodların entegre edilebilmesini ve istenmeyen işlevselliğin uygulamadan ayrıştırılmasını sağlar. Örneğin uygulamaya yeni eklenecek olan bir metod için yazılmış olan günlük tutma veya e-posta gönderme ilgisinin de aktif olmasını istediğimizde gerekli kalıpları uygulayan ilgilere, yeni eklenen metodların da yazılması bu işlevselliğin eklenebilmesi için yeterli olacaktır. Aynı şekilde İYP ile uygulanan tasarım kalıpları ilgileri içerisinde yapılan herhangi bir değişiklik, ilgi kurallarını tanımlayan soyut ilgilerde herhangi bir değişiklik yapılmasını gerektirmeyecektir. FTS Uygulamasında gerçekleştirilmiş olan Yegane Tasarım Kalıbı göz önüne alınırsa FaturaYoneticisi ve OdemeYoneticisi gibi tek olarak yaratılması istenen sınıfların başlarına o sınıfın daha önce yaratılıp yaratılmadığının kontrol edildiğine dair kodların yazılması gerekecektir. İYP ile gerçekleştirildiğinde ise bu tasarım kuralı soyut ilgi içerisinde tanımlanmıştır. Somut ilgide ise yeni eklenecek olan tüm metodların imzalarının tanımlanması yeterli olacaktır. Böylece tasarım kuralı kodu tekrarlanmamış olacak ve küçük bir değişiklikle yeni eklenecek olan sınıfın bu tasarım kalıbını uygulaması sağlanmış olacaktır. Aynı durum Gözlemci ve Komut tasarım kalıpları için de geçerlidir.

8.5 Diğer Kriterler

Karmaşıklıktan Uzak Kod Yazımı: Yerellik, Birleştirme Şeffaflığı ve Yeniden Kullanılabilirlik özellikleri sayesinde yazılan uygulama kodları ve enine kesen ilgi kodlarının birbirine karıştırılmaması sonucunda yazılan kodlar anlaşılabilir ve karmaşıklıktan uzak olmaktadır. NYP ile geliştirilmiş olan FTS uygulamasına enine kesen ilgilerin eklenmesi sırasında uygulama kodları değiştirilmemiş fakat ilgiler ile genişletilmiştir.

Sistemin Modülerizasyonu: Siteme eklenecek olan işlevsellik iyi modülerize edildiği ve tüm modülleri etkileyen enine kesen ilgilerin tespit edilmesi sistemin modülerizasyonun arttırmış, karmaşıklıktan uzaklaştırmıştır. FaturaYonetici ve OdemeYoneticisi sınıflarındaki pek çok metodu etkileyen ilgiler sisteme kolaylıkla adapte edilebilmiştir.

Bakım yapılabilirlik: Yazılan uygulamanın iyi modülerize edilmiş ve anlaşılabilir olması sistemde yapılması muhtemel değişikliklerin kolaylıkla uyarlanabilmesini beraberinde getirmiş, maliyeti azaltmıştır.

9. SONUÇLAR

Bu tez çalışmasında, günümüzde sürekli artan yazılım ihtiyaçlarını karşılamak için NYP yaklaşımını temel alarak onun eksiklerini tamamlamayı hedef alan kullanım alanı genişlemekte olan İYP yazılım geliştirme yaklaşımı çeşitli yönleriyle incelenmiş, birçok problemin çözümünde kullanılan GOF tasarım kalıplarının bu yeni yazılım geliştirme yaklaşımı ile gerçekleştirimi üzerinde durulmuştur. İncelenen bu yeni yaklaşım, örnek Fatura Takip Sistemi üzerindeki enine kesen ilgilerin gerçekleştiriminde kullanılmıştır. Kullanılan yöntemin sisteme kattığı avantajlar ve dezavantajlar incelenmiştir.

Çalışmada sistem gereksinimleri, işlevsel ilgiler ve enine kesen ilgiler olmak üzere iki ayrı alanda ele alınmıştır. Böylece enine kesen ilgilerin yayılımından kaynaklanan kod karmaşasının engelenmesi hedeflenmiştir. İYP yaklaşımı ile, NYP yaklaşımında karşılaşılan problemlerin çözülmesi hedeflendiğinden dolayı, NYP tasarım fazlarında kullanılan UML modelleme aracından faydalanılmıştır. Gereksinim ve Tasarım sürecinde UML kullanımı sistemin modellenmesini kolaylaştırmış, kodlama aşamasına gelinmeden önce sistemin modülerizasyonun iyi bir şekilde yapılmasını sağlamıştır.

Gereksinim Analizinin ikinci adımı olarak gerçekleştirilen sistem ilgilerinin ayrıştırılması ile belirlenmiş olan Enine Kesen ilgiler, İYP ilgileri olarak tasarlanmışlardır. Belirlenmiş bu ilgiler GOF Kalıpları ve İYP tekniklerinin bir arada uygulanmasıyla gerçeklenmiştir. GOF Kalıplarına İYP tekniğinin uygulanmasıyla kalıpların gerçeklenmesi sırasında meydana gelen gereksiz kod tekrarlaması ve arayüz gerçeklemesinden kaynaklanan kod yeniden yazımı engellenmiştir. Tasarım kalıplarının gerçeklenmesi aşamasında her tasarım kalıbının kuralı soyut bir ilgide tanımlanmış, daha sonra tanımlanan gerçek bir ilgide bu soyut ilgide tanımlanmış olan kural mevcut ilgi verileriyle gerçeklenmiştir. İlgi kuralının soyut ilgi ile tanımlanması, sisteme daha sonradan eklenecek olan aynı kuralın kullanımını gerektiren farklı ilgiler olması durumunda tasarım kalıbının yeniden kullanılabilirliğini sağlamıştır.

Sistemin İYP tekniği kullanılarak uygulanması ile mevcut uygulama kodları değiştirilmeden yeni işlevlerin eklenmesi sağlanmıştır. Bu nedenle mevcut uygulama kodları farklı uygulamalarda da kullanılabilmektedir. İlgilerin sisteme eklenip ayrılması için mevcut ilgi kodunun eklenip çıkarılması yeterlidir. Uygulamanın kodlarında bir değişiklik yapılmasına gerek kalmamaktadır. Uygulamanın geliştirilmesi esnasında her yeni eklenen özellik ilgi sınıfları şeklinde eklendiğinen dolayı sistemin bakım yapılabilirliği de kolaylaşmıştır.

Çalışmanın gerçekleştirimi esnasında karşılaşılan en büyük güçlük tasarım kalıplarının İYP

uygulamalarının pratikte çok az kullanılmış olmasıdır. Kurumsal uygulama geliştirme ortamlarında da İYP yaygın olarak kullanılmamakta ve daha çok akademik düzeyde incelemeleri ve araştırmaları yapılmaktadır. Bu nedenle kurumsal düzeydeki uygulamaların geliştirilmesinde kullanılabilmesi için pratik bilgi, zaman-maliyet hesaplamalarının ve derleme-çalışma zamanı performans planlamalarının proje özellikleri bazında değerlendirilerek yapılması gerekmektedir.

Belirli alanlardaki uygulamalara temel sağlayacak, genişletilebilir ve bakımı yapılabilir, kolay yönetilebilir ve zaman maliyet planlarına uygun bir yapının oluşturulması zor bir süreçtir. Bu çalışmada anlatılan GOF kalıpları ve İYP yaklaşımının kullanımı, yazılım geliştirme süreci sonunda istenilen özelliklere sahip uygulamalar geliştirilmesine yardımcı olacak, NYP yaklaşımı ile karşılaşılan birçok probleme çözüm olacaktır. Bu konulardaki akademik araştırmaların genişletilmesine destek olmak ve İYP kullanımının yaygınlaştırılması bu tez çalışmasının hedeflerinden biridir. Birçok yönden İYP yaklaşımını inceleyen Türkçe kaynakların azlığı, bu tez çalışmasının yapılmasının diğer bir amacıdır.

KAYNAKLAR

Araújo, J., Moreira, A., Brito, I. ve Rashid A., (2002), "Aspect-Oriented Requirements with UML", Workshop on Aspect-Oriented Modelling with UML, October 2002, Dresden, Germany.

Bodkin, R. ve Laddad, R., (2004), "Zen and The Art Of Aspect Oriented Programming, Aspects can greatly simplify design and maintanence of complex systems" , Linux Magazine.

Charfi, A. ve Mezini, M., (2005), "Application of Aspect-Oriented Programming to Workflows: The case of Web Service Composition with AO4BPEL", Software Techonology Group, March 2005, Sousse, Tunisia.

Charfi, A. ve Mezini, M., (2006), "Aspect-Oriented Workflow Languages", Talk @ coopIS 2006, 3 Nov. 2006, Montpellier-France.

Çetin, S., (2006), "Aspects And Java", Cybersoft.

Düzgün, H., "İlgiye Yönelik Programlama ve AspectJ", Hacettepe Üniversitesi Bilgisayar Mühendisliği.

Gradecki, J.D. ve Lesiecki, N., (2003), Mastering AspectJ Aspect Oriented Programming in Java, Wiley Publishing, USA.

Kande, M., Kienzle J. ve Strohmeier H., (2002a), "From AOP to UML - a bottom-up approach", Proceedings of the AOM with UML workshop at AOSD, 2002.

Kande, M., Kienzle J. ve Strohmeier H., (2002b), "From AOP to UML: Towards an Aspect-Oriented Architectural Modeling Approach", Proceedings of the AOM with UML workshop at AOSD, 2002.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J.M. ve Irwin J., (1997), "Aspect-Oriented Programming", European Conference on Object-Oriented Programming (ECOOP), June 1997, Finland.

Laddad, R., (2003), AspectJ in Action: Practical Aspect-Oriented Programming, Manning Publications, USA.

Pawlak, R., Seinturier, L., ve Retaillé, J.P, (2005), Foundations of AOP for J2EE Development, Apress, USA.

Rumbaugh, J., Jacobson, I. ve Booch, G., (1999), The Unified Modelling Language Reference, Addison Willey Longman, USA.

Sapir, N., (2002), "Extending UML with Aspect Usage Constraints in the Analysis and Design Phases".

Viega, J., Bloch J.T. ve Chandra, P., (2001), "Applying Aspect Oriented Programming to Security", Cutter IT Journal, 2001.

Win, B., Joosen, W. ve Piessens, F., (2002), "Developing Secure Applications through Aspect-Oriented Programming", 31 Oct., 2002.

İnternet Kaynakları

[1] www.wikipedia.org

[2] www.javaworld.com

[3] http://www.ibm.com/developerworks

ÖZGEÇMİŞ

Doğum tarihi	07.12.1982	
Doğum yeri	İstanbul	
Lise	1997-2000	Körfez Oruç Reis Anadolu Lisesi / Fen Bilimleri
Lisans	1999-2004	İstanbul Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü
Yüksek Lisans	2004-2007	Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Yüksek Lisans Programı

Çalıştığı kurum(lar)

2004 -2005	Pasifik Bilgi Sistemleri / Yazılım Mühendisi
2005 -2006	Logo Yazılım / Yazılım Mühendisi
11.2006 -	Eczacıbaşı Bilişim / Yazılım Mühendisi