# Özellik Modeli ile Değişkenlik Yönetimi
# Managing Variability with Feature Models

Ender Bulut[1,2][0000−0002−0729−3542] ve Dr. Cevat Şener[1][0000−0002−7263−4534]

[1] Bilgisayar Mühendisliği Bölümü, Orta Doğu Teknik Üniversitesi
bulutender@gmail.com, sener@ceng.metu.edu.tr
[2] TÜBİTAK BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü
ender.bulut@tubitak.gov.tr

**Özet** Bu çalışma kapsamında, nesneye dayalı uygulamalarda özellik modeli ve basit kaynak kod üretme tekniği kullanılarak değişkenliği yönetmeye çalışan bir yaklaşım geliştirildi. Bu yaklaşım pratik bir şekilde nesneye dayalı yapılandırılabilir uygulamalar geliştirebilmeyi sağlar. Bu yöntemle geliştirilen bir uygulama girdi olarak bir alana ait kullanıcı seçimlerini içeren konfigürasyon dosyasını alır ve daha sonra, kendini bu seçimlere göre otomatik olarak yapılandırıp yönetir. Bu yöntemi uygulayabilmek için takip edilmesi gereken temel adımlar vardır. Öncelikle, bir alandaki değişkenlik noktaları ve ortak noktalar belirlenir ve özellik modeli kullanılarak tasarlanır. Daha sonra, bu bildiride sunulan metot için geliştirilmiş olan kod üreticisi ara birimleri, akıllı kaynak kod taslaklarını otomatik olarak üretir. Bir sonraki adımda da geliştiriciler üretilmiş olan kodları, uygulamanın ortak noktalarını ve değişkenliğini göz önünde bulunduran veri yapılarını kullanarak uygulamayı geliştirir. Bu uygulama girdi olarak alacağı kullanıcı seçimlerini içeren bütün konfigürasyonlar için çalışabilecektir. Böylece alandaki değişkenlikler doğrudan doğruya ve otomatik olarak özellik modeli aracılığıyla bu nesneye dayalı uygulamaya yansıtılmış olacaktır.

**Abstract** In this work, an approach that aims to manage variability by using feature model and a simple code generator in object oriented applications is developed. This method provides practically develop configurable object oriented applications. An application that is developed with respect to the method takes a configuration file including user selections in a domain, as an input so that the application can manage and automatically configure itself with user selections. In order to implement this method, there are some main rules that should be obeyed. First of all, variability points and commonalities in a specific domain is described and a feature model is designed with respect to them. Then, the code generator developed for the method presented in this specification automatically generate intelligent source code drafts. In the next step, developers will develop the implementation of the generated code using data structures that take into account the commonalities and variabilities of the application. This application can work for all configurations that contain user selections that will be input. Thus, the variances in the scene will be reflected directly and automatically to the object-oriented application via the feature model.

**Anahtar Kelimeler:** Kod Üretimi · Değişkenlik Yönetimi · Özellik Modeli
**Keywords:** Code Generation · Variability Management · Feature Model

## 1 Giriş

Son yıllarda, yazılım şirketleri daha kullanıcı dostu ve yüksek kaliteli yazılım ürünlerini pratik ve hızlı bir şekilde üretebilmek için yeni metotlar ve teknikler araştırıyor. Bu kapsamda bir sistemdeki yazılım bileşenlerinin yeniden kullanılabilirliğini sağlamak bu şirketler için büyük ölçüde önemli hale geldi. Yazılım sektöründe ve akademik alanda yeniden kullanılabilir ve uyarlanabilir yazılımlar üretebilmek için nesneye dayalı programlama konseptlerinin kullanıldığı farklı yöntemler vardır. Fakat bu yöntemlerin birçoğu alan değişkenliğini yani farklı ürünlerin değişken özelliklerini göz önünde bulundurmaz. Bazıları da sadece yeniden kullanılabilir objeleri tanımlayıp modelleyerek yeniden kullanılabilirlik üzerine yoğunlaşır ve bunların genelde alanın değişkenlik noktalarını belirlemede eksikleri vardır. Bir grup analiz metotları ise bu tanımlama için bazı standart kurallar kullanır ancak bir alandaki benzer uygulamalar arasındaki değişkenlik noktalarını saptama kabiliyetleri yoktur. Bundan dolayı bazı yöntemlerde, belli bir alanda nesneye dayalı uygulamalar geliştirmek için özellik modelleri kullanılmaya başlanmıştır. Bu bildiride kullanılan yaklaşım ise akıllı veri yapıları desenleri ile birlikte kaynak kod taslakları üretmeyi ve böylece kod geliştirme aşamasını pratik bir şekilde hızlandırmayı ve önceden belirlenmiş olan alana göre geliştirilmiş olan nesneye dayalı uygulamaya o alandaki değişkenlikleri yansıtmayı amaçlamıştır. Bu sayede, tasarım ve kod gibi yazılım bileşenlerinin yeniden kullanabilirliğini arttırmak ve aynı anda gereksinim, tasarım değişikliği gibi durumlarda yeniden yazılım geliştirme döngüsü ihtiyacını azaltmak hedeflenmiştir.

Bu yöntemi gerçekleştirebilmek için takip edilmesi gereken bir takım adımlar vardır. İlk adımda, belirli bir alandaki ortak noktalar ve değişkenlik noktaları müşterilerin istekleri ve belirlenmiş tasarım seçimleri göz önünde bulundurularak bir özellik modelinde belirlenir. Daha sonra bu özellik modeli kullanılarak yarı otomatik, geliştiricilere zaman kazandıran ve esnek olan alana özgü kaynak kod taslakları bu yaklaşıma özel tasarlanmış kod üreticisi sayesinde otomatik olarak üretilir. Üretilen bu kodların geliştirilmesine devam edilerek uygulamanın yazılımı tamamlanır. Daha sonra özellik modeli kullanılarak otomatik olarak üretilen değişkenlik yönetimi sınıfları ile bu uygulamanın özellik modelindeki bütün özellik kombinasyonları için çalışması sağlanmış olur. Böylece alandaki değişkenlikler fazladan hiç bir işlem yapmaya gerek kalmadan doğrudan ve otomatik olarak uygulamaya yansıtılmış olur. Bu yöntem sayesinde, uygulamada çeşitli özellik kombinasyonlarının çalışabilirliğini sağlayabilmek için kod değişikliği veya yeniden bir yazılım tasarımı hazırlamaya gerek kalmamış olacaktır.

Literatürde bu bildiride sunulan yönteme benzeyen bazı araştırmalar mevcuttur. Mesela, Zhang ve Jarzabek bir alandaki benzerlik ve değişkenliği yönetebil-

mek için XML tabanlı olan Variant Configuration Language (XVCL) dilini kullanarak yazılım ürün hattı bileşenleri oluşturur [11]. Bu yaklaşıma göre, mümkün olabilecek bütün özellik seçimleri için gerekli kaynak kod parçaları başlangıçta hazır olmalıdır. Yani herhangi bir özellik kombinasyonunda bulunan özellikler için ilgili kaynak kod parçaları seçilerek yazılımın çalışması sağlanacaktır. Fakat bu yöntem yeterince ölçeklenebilir değildir ve ayrıca özellik modelleri büyüdükçe bütün olasılıklar için kaynak kodu hazırlamanın maliyeti çok fazla olacaktır.

AHEAD aracı da özelliğe dayalı programlama yöntemi kullanılarak Batory [4] tarafından geliştirilmiştir. Bu yöntemde özellikler arttırımlı olarak eklenerek basit programlar geliştirilir. Yani özellik modeline eklenen her bir özellik için Jak dosya formatında basit program parçaları tanımlanır. Jak, Java programlama dilinden türetilmiş kod üretmeyi destekleyen bir programlama dilidir. Bu yöntemde birçok nesneye dayalı programlama dili varken sadece Java dil desteği vardır ve ayrıca kullanıcıyı Jak programlama dilini de öğrenmeye mecbur bırakmaktadır. Jak dili bilmeden bu yöntemi uygulamak mümkün olmamaktadır. Bu bildiride sunulan yöntem ise uygulama geliştirme dili dışında fazladan bir programlama dili öğrenmeye gerek kalmadan değişkenlik yönetimini otomatik bir şekilde sağlamayı hedeflemektedir.

Bildirinin içeriği ilerleyen bölümlerde detaylı bir şekilde anlatılmıştır. Özelliğe Dayalı Yazılım Geliştirme bölümü, bildiride geçen konuyla ilgili genel terimler, yazılım geliştirme yaklaşımları ve yaşam döngüsü gibi konularda bilgiler içerir. Üçüncü bölümde ise yazılım geliştirmede özellik modelinin nasıl kullanıldığı hakkında genel çerçeveden bakacak şekilde bilgi verilmiştir. Dördüncü bölümde ise bu bildiride anlatılan yöntemin uygulandığı örnek bir uygulama senaryosu ve bu uygulama için hazırlanan özellik modeli anlatılmaktadır. Bir sonraki beşinci bölümde de bildiride önerilen yöntemin ve özellik modelinin nasıl kullanıldığının detayları adım adım anlatılmaktadır. Altıncı ve son bölüm ise bildiride önerilen yöntemin kısa bir özetini, avantajlarını, dezavantajlarını ve bu yöntemin üstüne ileride yapılabilecek çalışmaları anlatır.

## 2 Özelliğe Dayalı Yazılım Geliştirme

Yeteneğe Dayalı Yazılım Geliştirme (YDYG) [2] yazılım sistemlerine farklı metotlar, diller, tasarım teknikleri ve araçlar ile birlikte yapı, gereksinime uyarlama ve sentezleme sağlayan ve değişken, uyarlanabilir ve geliştirilebilir yazılımlar geliştirmeye yarayan bir paradigmadır. Özellik, YDYG'nin en önemli parçasıdır. Bu yaklaşımda daha süreçlerin en başındayken belirlenen özellikler yazılım yaşam döngüsündeki bütün aşamalarda kullanılır ve bir köprü görevi görür. YDYG temelde yazılım ürün hatları geliştirmek ve alan mühendisliği yapmak için kullanılır. Yazılım ürün hattı ise belli bir market segmentinin veya problemin belirli ihtiyaçlarını karşılayan ortak ve değişken özelliklerin yönetildiği bir grup yazılım sistemleridir.

### 2.1 Özellik

YDYG'nin temel parçası olan Özellik için literatürde yapılmış çok sayıda teknik ve soyut tanımlar bulunmaktadır. Bu teknik tanımlar incelendiğinde özelliklerin tasarım kararlarını simgeleyen ve potansiyel bir konfigürasyon seçeneği sunan yazılım gereksinimlerini karşılamak için uygulanabileceği görülmektedir.

Aşağıda farklı özellik tanımlarının yapıldığı kısımlar görülebilir.

1. "Bir paydaş ihtiyacını karşılamak, bir tasarım kararını uygulamak ve kapsüllemek ve bir konfigürasyon seçeneği sunabilmek için belirli bir programın yapısını genişleten ve değiştiren bir yapı" [3]
2. "İsteğe bağlı veya artan bir işlev birimi" [10]
3. "f= (R, W, S), R, özelliklerin gereksinimlerini karşılamayı; W özelliğin çevresi hakkında aldığı varsayımları ve S, özelliğin spesifikasyonunu ifade ettiği bir üçlüdür." [6]
4. "İşlevsel ve işlevsel olmayan bir dizi gereksinimle belirlenen mantıksal davranış birimi" [5].
5. "Uygulanması, test edilmesi, iletilmesi ve sürdürülmesi gereken ayırt edici tanımlanabilir bir soyutlama" [8].

Yukarıda belirtilen tanımlar da göz önünde bulundurularak, özellik ifadesinin bu bildiride önerilen yöntemde asıl kullanılma amacı ve özelliğe yüklenen tanım; belli bir alan için belirlenen bir özelliğin o alan kapsamında yapılacak olan yazılım sistemindeki belirli gereksinimlere, tasarım kararlarına ve potansiyel yapılandırma seçeneklerine karşılık gelebileceği olmuştur. Bu kapsamda Özellik kavramı için bildiride farklı tipler kullanılmıştır. İlerleyen bölümlerde bu özellik tipleri, tanımları ve nerede ne zaman kullanıldığı hakkında detaylı bilgi verilmiştir.

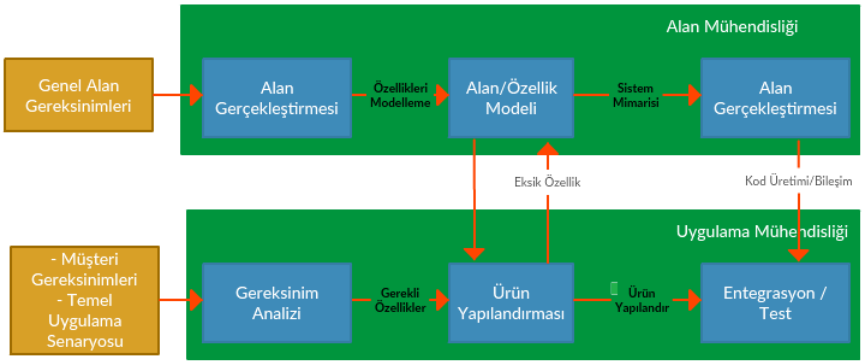
### 2.2 YDYG Yaşam Döngüsü

Ortaklıkları ve değişkenlikleri belirleyip yönetebilmek için özellik parçaları belirlenip oluşturulmalıdır. YDYG yöntemi konfigürasyon seçenekleri sunup yazılım ürünlerinin oluşturulmasını sağlar. Bunu sağlamak için de 4 temel fazın olduğu bir yaşam döngüsü vardır:

1. Alan analizi
2. Alan tasarımı ve spefisikasyonu
3. Alan uygulaması
4. Ürün konfigürasyonu ve üretilmesi

Bu yaşam döngüsündeki ilk 3 adım aslında temel olarak alan mühendisliği kapsamında ele alınır. Ürün konfigürasyonu ve üretilmesi adımı ise uygulama mühendisliği kapsamında düşünülebilir. Alan mühendisliği belirli bir alanda uygulama geliştirmek için kullanılabilecek parçaların geliştirilmesini amaçlar. Fakat bir yazılım ürün hattının ortak ve değişken noktaları çok iyi bir şekilde tanımlanıp gösterilmesi çok önemlidir. Birçok yazılım sistemi aynı alandaki başka

yazılım sistemlerinin farklı varyasyonları olabilir. Şirketler buna benzer durumlarda kazançlarını arttırabilmek için, markete daha hızlı ürün çıkarabilmek için alan mühendisliği kapsamında bu konsept ve geliştirmeleri kullanabilir. Alan mühendisliği 3 temel bölümden oluşur. Bu bölümler Alan Analizi, Alan Tasarımı, Alan Gerçekleştirmesi olarak belirlenmiştir. Bu bölümlere paralel olarak Uygulama Mühendisliği adımları vardır. Şekil 1 de gösterilen Alan Analizi ve Gereksinim Analizi adımları aslında müşteri ihtiyaçlarının tespit edilmesi aşamasıdır. İlk aşamadan sonra Alan ve Uygulama için gerekli olan özellikler tespit edilir. Özellik modeli ile Ürün konfigürasyonu her zaman bir biri ile uyumlu olmalıdır. Bunlar karşılıklı olarak birbirlerini etkiler. Daha sonraki aşama uygulamanın geliştirilmesi ve test edilmesi kısmıdır.




**Şekil 1.** Alan ve Uygulama Mühendisliği [7]

## 3 Yazılım Geliştirmede Özellik Modelinin Kullanılması

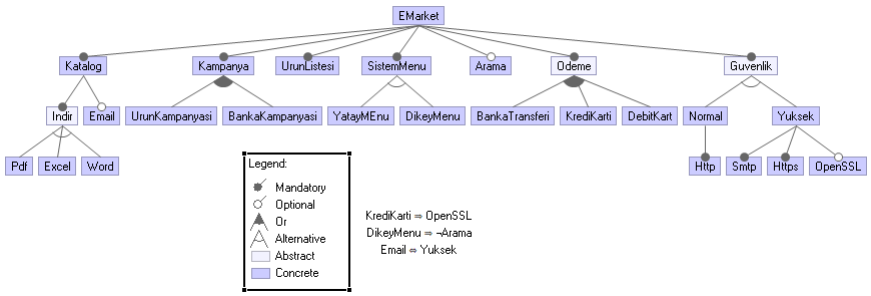
Nesneye dayalı programlama dilinin genel yaşam döngüsü üç temel adımdan oluşur. İlk adım bir yazılım sisteminin gereksinimlerini ve mümkün olabilecek tasarım seçimlerini belirlemektir. Daha sonra bu belirlenmiş olan gereksinim ve tasarım seçimlerinin UML modelleme dili gibi bir araç ile arayüzleri, bileşenleri ve bu bileşenlerin bir birleri ile olan bağlantıları içeren bir modele dönüştürülmesi gelir. Son adımda da yazılım mühendisleri tarafından yazılımın bütün kaynak kodu geliştirilir ve gerçekleştirilir. Bu yaklaşımda ise ilk olarak nesneye dayalı programlama ile geliştirilen uygulamalar için özellik modellerini kullanarak akıllı kaynak kod kalıpları üretmek amaçlanmıştır. Böylece geliştiriciler yazılım geliştirme sürecine başladığında yazılımda gerekli olacak olan başlangıç sınıflarını ve arayüz sınıflarını oluşturmak için zaman harcamamış olacaktır. Bir diğer amaç ise son ürün olarak elde edilen uygulamanın özellik modelinden türetilen bütün konfigürasyon varyasyonları için sistemde hiç bir değişiklik yapmadan çalışabilmesini sağlamaktır. Böyle bir uygulama geliştirmek sadece nes-

neye dayalı programlama mimarilerini kullanarak mümkün değildir. Önerilen yöntemde de otomatik olarak oluşturulan akıllı sınıfların sağladığı veri yapıları bütün yazılım geliştirme sürecinde kullanılır. Buna ek olarak özellik modeli, detaylı bir şekilde analiz edilir ve otomatik olarak değişkenlik yönetimini sağlayacak olan sınıflar üretilir. Böylece üretilen bu sınıfların geliştirilmiş olan uygulamanın bütün kullanıcı seçimlerine göre sistemde hiç bir değişikliğe gerek duymadan uyarlanıp çalışabilmesi hedeflenmiştir.

## 4 Örnek Uygulama: İnternet'ten Alışveriş Alanı

Bu yayında belirtilen yöntemi daha iyi anlatabilmek için örnek bir masa üstü uygulaması olan internetten alış verişi sağlayan bir konsept uygulama geliştirmeye karar verildi. Kompleks bir web tabanlı alış veriş uygulaması yapmak ilerleyen zamanlarda düşünülebilir. Şimdilik daha pratik olması açısından standart masa üstü uygulamasının daha uygun olacağı düşünüldü. Bu örnekte ilk iş olarak alandaki limitler tahmini olarak belirlendi. Bir sonraki aşamada ise alandaki gereksinimler belirlendi ve bu gereksinimler temel alınarak alandaki ortaklıklar ve değişkenlikler alan mühendisliği yapılarak belirlendi. Bunun dışında müşterilerin de ekstra özellikler talep edebileceği düşünülerek ilerlendi. Bu çalışmada, ortaklık ve değişkenlik noktaları başarılı ve doğru bir şekilde tespit edildikten sonra özellik modelinden oluşturulacak farklı tasarım seçimi kümeleri için ayrı ayrı uygulamalar üretmek yerine ortaya sadece bir tane çalışan bir uygulama geliştirip özellik modelinde yapılacak farklı seçim varyasyonlarının hepsi için bu uygulamanın çalışma esnasında yeni özellik seçimlerine göre kendini otomatik olarak adapte etmesi yani bu seçimleri uygulamaya yansıtması hedeflenmiştir.

Bu örnek için alış veriş alanı baz alınarak bazı temel tasarım kararları belirlendi ve bunlara göre de Şekil 2'de gösterilen özellik modeli oluşturuldu. Bu örnek, ilerleyen bölümlerde anlatılan adımlara göre uygulanmıştır ve uygulamanın çalışma esnasında bütün ürün kombinasyonları için doğru bir şekilde çalışabildiği görülmüştür.

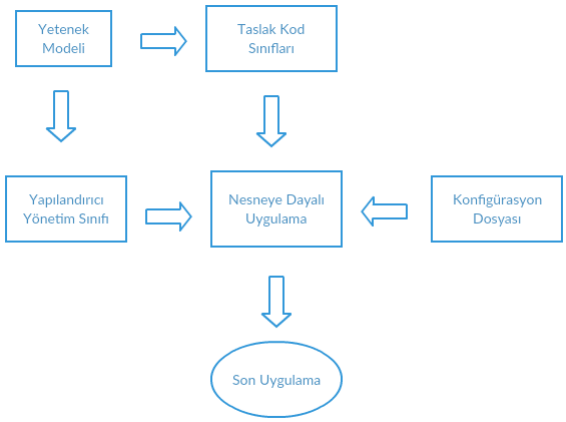



**Şekil 2.** E-Market Özellik Modeli

Bir sonraki bölümde de bu örnek uygulamanın bildirideki yönteme göre nasıl oluşturulduğu, yöntemin adımlarının nasıl uygulandığı, özellik modelinin nasıl oluşturulduğu, bu modele göre kaynak kod üretimi, yazılım geliştirmenin nasıl yapılacağı ve uygulamanın en sonda kullanıcı seçimlerine göre çalışabildiği detaylı bir şekilde anlatılmıştır.

## 5 Önerilen Yöntem

Bu yöntemi daha iyi anlayabilmek için yöntemde izlenmesi gereken adımlar Şekil 3'deki akış diyagramı ile incelenmelidir. Bu diyagramda görüldüğü gibi öncelikle belirlenmiş olan bir alandaki tasarım kararlarına göre ve bir özellik modeli aracı yardımı ile özellik modeli tasarlanır. Bu özellik modeli belirlenen alandaki özelliklerden hangilerinin nasıl ve hangi kurallara uyarak sistemde olabileceğini gösterir. Daha sonra geliştirmiş olduğumuz otomatik kod üreticisi, taslak kodları ve yazılımda kullanılacak olan arayüz sınıflarını oluşturur. Aynı zamanda özellik modelindeki özelliklerin birbirleriyle olan bütün ilişkilerini göz önünde bulundurarak değişkenlik ve konfigürasyon yönetimi yapacak olan sınıflar da otomatik olarak üretilir.




**Şekil 3.** Önerilen Yöntemin Akış Diyagramı

Kod üretiminin detayları bir sonraki alt bölümlerde anlatılacaktır. Akışta görüldüğü gibi bir sonraki adım yazılım geliştirmenin yapıldığı aşamadır. Bu adımda geliştiriciler bu üretilen taslak kodları ve bunların içindeki veri yapılarını kullanarak yazılım geliştirme sürecini devam ettirirler. Yazılım geliştirme süreci sona erdikten sonra nesneye dayalı yapılandırılabilir bir uygulama ortaya çıkmış olur. Akış diyagramında da görülebileceği gibi bu uygulama çalışmaya başlamadan önce özellik modelinden kullanıcı seçimleriyle oluşturulan bir konfigürasyon dosyası uygulamaya girdi olarak verilir. Kullanıcıların seçimlerini içeren bu girdi

değişkenlik yönetimini sağlayan sınıflarda anlamlandırılarak sonuçları uygulamaya otomatik olarak yansıtılır. Bu sınıflar bağımlılık denetimi yöntemiyle uygulamanın kodunda gerekli konfigürasyon işlemlerini çalışma zamanı sırasında yapar ve böylece uygulama girdi olarak gelebilecek bütün kullanıcı seçimi kombinasyonları için çalışabilecektir. Sonuç olarak belirlenmiş olan alandaki değişkenlik noktaları direk ve otomatik olarak nesneye dayalı uygulamalara özellik modeli sayesinde yansıtılmış olacaktır.

### 5.1 Özellik Modeli Dili: FeatureIDE

Belirlenmiş olan bir alanın detaylarını anlamlı bir şekilde yapılandırmak ve gösterebilmek için bu yöntemde bir özellik modeli aracı kullanmak gerekiyordu. Basit, anlaşılır ve kullanıcı dostu bir arayüze sahip olan ve yazılım geliştirme platformu Eclipse için eklentisi bulunan FeatureIDE aracı kullanılmasına karar verildi. Bu araç, var olan eklentiyi iyileştirmek veya geliştirmek için değil de sadece bildirideki yöntemde uygulanması gereken adımlarda gereken özellik modelini oluşturmak ve bu modeli analiz etmek için kullanılmıştır. Bu araç bir özellik konfigürasyonunun özellik modelindeki kurallara göre uygun olup olmadığına karar verebiliyor ve bunu da görsel olarak kullanıcıya iyi bir şekilde gösterebiliyor. Bir özellik modelini kolayca oluşturabilmeyi sağlamasının yanında kolay anlaşılan arayüzü sayesinde kullanıcı özellik modelinden seçimlerini kolayca yapabiliyor. FeatureIDE bir özellik ve ona bağlı olan özellikler arasında çeşitli ilişki türleri kurabilmeyi sağlar. Şekil 2'de bu araç kullanılarak İnternet üzerinden alışveriş alanı için hazırlanmış özellik diyagramını inceleyebilirsiniz. Bu özellik diyagramında görüldüğü gibi bu araçta kullanılan özellik modeli dilinde *Zorunlu*, *Opsiyonel*, *Soyut* özellik türleri ve özellikler arasında da *Ve*, *Veya*, *Alternatif* bağlantı türleri vardır. Ayrıca özellik modelinin alt tarafındaki gibi uygulamaya yönelik kısıtlamalar da tanımlanabilmektedir.

### 5.2 Özellik Modelinin Özyinelemeli Ayrıştırılması

Bu yöntemin ilk temel adımı özellik modelinin bilgilerini taşıyan XML dosyasının özyinelemeli olarak ayrıştırılması ve anlamlandırılmasıdır. Özellik modelindeki bütün özelliklerin kod tarafında anlamlı bir ağaç veri yapısına koyulabilmesi için basit bir derin öncelikli gezinme algoritması kullanıldı. Bu algoritma her ağaçtaki her özelliği derin öncelikli sırayla gezip aşağıdaki işlemleri özyinelemeli olarak gerçekleştirmiştir:

1. Önceden düzenleme işlemini yap
2. For each $i$ ($i$: from 1 to $n$-1) do:
   2.1. Eğer varsa $i$. elemanı ziyaret et
   2.2. Özellikleri ile birlikte bir özellik düğümü oluşturulur.
   2.3. Normal sırayla işlem yap
3. Eğer varsa son ($n$.) çocuğu ziyaret et
4. Çatı düğümünden önce sol ve sağ alt ağaçları işleme sok

n: Çocuk düğümlerin sayısını simgeler. Bu algoritma varlık sınıflarını nesneye dayalı olarak tasarlanmış bir şekilde üretebilmek için özellik modelindeki bütün özellikleri yapılandırır.

### 5.3 Veri Yapıları ile Kaynak Kod Üretimi

Yapılan araştırmalar çerçevesinde nesneye dayalı programlama konseptlerinin bir özellik modelinin kod parçalarına dönüştürülebileceğini ve bu anlamlı parçaların bir alandaki değişkenliği yönetebilmek için kullanılabileceği görüldü. Bu fikirden yola çıkarak bir özellik modelindeki kullanıcı seçimlerine göre otomatik uyarlanabilen uygulamalar geliştirebilmek için bu yöntem oluşturuldu. Bu yöntemin ilk aşamasında kod üretim işlemini gerçekleştirebilmek için *CodeModel Uygulama Ara Birimi* [1] kullanıldı. Bu kütüphane tip-güvenli Java kodları üretilmesini sağlayarak önerilen yönteme katkı sağlamış oldu.

### 5.4 Veri Yapıları İçeriği ile Birlikte Varlık ve Arayüz Sınıfları

Özellik modeline uygun olarak varlık ve arayüz sınıflarını akıllı veri yapılarını içerecek şekilde oluşturmak için özellik modelleme dilindeki özellik tipleri ve özellikler arasındaki bağlantı türlerine göre özellik modeli ile kod arasında bir eşleme yöntemi geliştirildi. Bu özellik tipleri ve bağlantı türlerinin nasıl kullanıldığı ve otomatik olarak üretilen kodların hangi mantığa göre tasarlandığı aşağıda açıklanmıştır.

### 5.5 *Zorunlu* Özellik

Bir özellik modelinden seçim yaparak oluşturulacak olan bütün ürün konfigürasyonlarında olması istenilen özellikler *Zorunlu* özellik tipi olarak özellik modelinde tanımlanır. "B" adındaki zorunlu özelliğe sahip bir özellik "A" adındaki başka özelliğin çocuğu olduğu durumda, nesneye dayalı programlama mantığına göre öncelikle "A" ve "B" adlarında varlık sınıfları oluşturulur. Ayrıca bu iki sınıf arasında *Kompozisyon (Composition)* bağlantısı kurulur. Yani "A" sınıfı "B" sınıfını içinde kullanabilecektir. Uygulama çalışacağı zaman "A" sınıfı türündeki nesnenin "B" sınıfı türündeki özelliği varsayılan olarak oluşturulacaktır.

### 5.6 *Opsiyonel* Özellik

Eğer bir ürün konfigürasyonunda bir özelliğin bulunması opsiyonel olarak sunulmak isteniyorsa bu özellik *Opsiyonel* özelliğinde tanımlanır. "B" adındaki opsiyonel özelliğine sahip bir özellik "A" adındaki başka özelliğin çocuğu olduğu durumda, nesneye dayalı programlama mantığına göre öncelikle "A" ve "B" adlarında varlık sınıfları oluşturulur. Ayrıca bu iki sınıf arasında *Kompozisyon (Composition)* bağlantısı kurulur. Yani "A" sınıfı "B" sınıfını içinde kullanabilecektir. Uygulama çalışacağı zaman da eğer ürün konfigürasyonunda "B" özelliği seçilmişse "A" sınıfı türündeki nesnenin "B" sınıfı türündeki özelliği otomatik olarak oluşturulacaktır; eğer seçilmemişse bu özellik boş olarak kalacaktır ve uygulamanın çalıştığı süre boyunca hiç bir zaman aktif olmayacaktır.

### 5.7 *Ve* Bağlantısı

Bir özellik ve onun çocuk özellikleri arasında *Ve* bağlantısı olduğu durumlarda, bu yöntemde yine *Kompozisyon* bağlantısı olarak kullanılmıştır. Örneğin, "A" özelliğinin "B" ve "C" adında iki alt (çocuk) özelliği olsun. Eğer bu iki özellik "A" özelliğine *Ve* bağlantı türü ile bağlı ise bunun UML sınıf diyagramındaki karşılığında "A", "B" ve "C" varlık sınıfları oluşturulur. "A" sınıfının içinde de "B" ve "C" sınıfı türlerinde özellikler *Kompozisyon* mantığına göre bulunurlar.

### 5.8 *Alternatif* Bağlantısı

Bir özellik ve onun çocuk özellikleri ile arasında *Alternatif* bağlantısı olması ise alt özelliklerden sadece birinin seçilebileceği anlamına gelir ve bu bağlantı türü için buraya kadar anlatılanlardan daha farklı bir yöntem kullanılır. Özellik modelinde bir "A" özelliğinin altında *Alternatif* bağlantısı ile bağlanmış olan "B" ve "C" gibi iki alt özellik olduğu zaman yine her bir özellik için varlık sınıfları oluşturulur. Ayrıca "IA" adında bir arayüz sınıfı oluşturulur ve "B" ve "C" sınıflarının bu arayüzü gerçekleştirmesi sağlanır. *Alternatif* ilişkisi için de *Kompozisyon* mantığı kullanılmıştır. Fakat bu sefer "A" sınıfında "IA" arayüzü sınıfı tipinde bir değişken tanımlanacaktır ve uygulamanın çalışması esnasında ürün konfigürasyon dosyası içinde bulunan kullanıcı seçimine göre bu özelliğin türü belirlenecektir. Böylece kullanıcı özellik seçimini değiştirdiği zaman sistemde herhangi bir değişiklik yapmadan uygulama seçilen özelliği otomatik olarak direk güncelleyecektir.

### 5.9 *Veya* Bağlantısı

Bir özellik ve onun çocuk özellikleri ile arasında *Veya* bağlantısı olması ise alt özelliklerden biri ya da birden fazlasının seçilebileceğini ifade eder. Özellik modelinde bir "A" özelliğinin altında *Alternatif* bağlantısı ile bağlanmış olan "B" ve "C" gibi iki alt özellik olduğu zaman yine her bir özellik için varlık sınıfları oluşturulur. Ve yine "IA" adında bir arayüz sınıfı oluşturulur ve "B" ve "C" sınıflarının bu arayüzü gerçekleştirmesi sağlanır. *Veya* ilişkisi için de *Kompozisyon* mantığı kullanılmıştır. Fakat *Alternatif* bağlantısından farklı olarak "A" sınıfında "IA" arayüzü sınıfı tipinde bir liste değişkeni tanımlanır. Daha sonra uygulamanın çalışması esnasında ürün konfigürasyon dosyası içinde bulunan kullanıcı seçimlerine göre bu listenin elemanları belirlenip oluşturulacaktır. Eğer bu alt özelliklerden biri seçilmezse bu listeye eklenmeyecektir ve uygulama yaşam döngüsü boyunca da kullanılmayacaktır. Yazılım geliştirme esnasında her özellik için sanki uygulamada olacakmış gibi kodlaması yapılacaktır. Böylece kullanıcı özellik seçimini değiştirdiği zaman sistemde herhangi bir değişiklik yapmadan uygulama seçilen özellik grubuna göre otomatik olarak aktif özelliklerin çalışmasını sağlayacaktır.

### 5.10 Soyut Özellik

Bu yöntemde kullanılan özellik modelleme dilinde *Soyut* ve *Somut* olmak üzere iki temel özellik tipi vardır. Yukarıda anlatılan durumlardaki özellikler hep *Somut* özellik olarak kabul edilip değerlendirilmiştir. *Soyut* özellikler ile ilgili yaptığımız araştırmalar sonucunda bu özelliklerin kod tarafında bir karşılığı olmadığını sadece özellik modelini yapılandırmaya yardım ettiği tespit edildi. Bundan dolayı *Soyut* özellikler için kod tarafında herhangi bir varlık sınıfı oluşturulmadı. Sadece soyut özelliğin adını kullanarak bir arayüz sınıfı oluşturup bu soyut özelliğin varsa alt özelliklerinin bu arayüzü gerçekleştirmesi sağlandı. Bundan dolayı bu yöntem kapsamında özellik modeli oluşturulurken kod tarafında varlık sınıfı olarak üretilmesine gerek olmayan sadece mimariyi yapılandırmaya yarayacak olan özelliklerin *Somut* özellik olarak belirlenmesine karar verildi.

## 6 Sonuç

Bu bildiride nesneye dayalı uygulamalarda değişkenliği özellik modeli kullanarak yönetebilmek için tasarlanmış olan bir yaklaşım anlatıldı. Bu yaklaşımın en önemli noktası bir uygulamanın çeşitli konfigürasyonlarını yönetmek için değişkenlik bilgisinin nasıl kullanıldığıdır. Bunu da sağlamak için özellik modelinden akıllı veri yapılarını içeren ve ilerleyen aşamalarda alan değişkenliğini otomatik yönetmeye yarayacak olan kod taslaklarını üreten tasarım desenleri oluşturuldu. Ayrıca bu oluşan kodlardaki veri yapıları yazılımcılara kod geliştirme sürecinde kolaylık sağlayıp, zaman kazanmalarına yardımcı olması amaçlandı. Bu amaca ulaşmak için de özellik modeli temelli yazılım geliştirme tekniği kullanıldı. Bu süreçte FeatureIDE [9] özellik modellerinden akıllı kod taslakları oluşturulması sağlandı. Bu kodlar yazılım sisteminin son halinin genel çerçevesini çizmektedir. Bu şekilde yazılımcılar bu çerçeve kapsamında kod geliştirmesini yapabilirler.

Bu yöntemin dezavantajı ise tersine mühendislik konusunda zayıf olmasıdır. Yani, kod tarafında bir değişiklik yapıldığı zaman bu özellik modeli tarafına yansıtılamayacaktır. Gelecekte yapılabilecek bir çalışma olarak bu kısım üzerinde durulması planlanıyor. Ayrıca bu çalışma kapsamında alan analizi aşamasında alınan tasarım kararlarının iyi ve doğru bir şekilde tanımlanmış olması ve özellik modeline yansıtılmış olması beklenmektedir. Bu özellik modelinin son uygulamadaki ortak noktalar ve değişkenlik noktalarını içermesi gerekmektedir. Böylece yazılım geliştirme süreci düzenli ve doğru bir şekilde ilerleyebilecek ve uygulamanın son hali alandaki değişkenlik noktalarını otomatik olarak yönetebilecektir. Eğer bu başlangıç aşamasındaki özellik modeli oluşturulması adımı eksik veya hatalı yapılırsa bu yöntemin ilerleyen süreçleri için sorunlar ortaya çıkacaktır.

Bu yöntemi ilerletmek için gelecekte bazı çalışmalar yapılabilir. Öncelikle yöntemin daha iyi bir hale getirilebilmesi için büyük çaplı bir alan veya market üzerinde denenmesi daha iyi olacaktır. Böylece yaklaşımın eksik veya zayıf tarafları tespit edilip iyileştirmesi sağlanabilir. Bu yaklaşımın adımlarını uygulayabilmek için özellik modeli kısımları için bir Eclipse eklentisi olan FeatureIDE aracı kullanıldı. Gelecekte yapılabilecek işlerden bir diğeri ise bu yöntemdeki bütün

adımları kolaylıkla uygulayabilmek için Eclipse, IntelliJ IDEA gibi platformlara yeni bir eklenti yapılması olabilir. Böyle bir eklenti süreçlerin uygulanma hızını ve hatasız ilerleyebilmeyi sağlayacaktır. Bir diğer nokta da bu yöntemin Java programlama dili dışında farklı programlama dillerini de kapsayacak şekilde geliştirilmesi olabilir.

## Kaynaklar


1. Codemodel api. from https://codemodel.java.net/nonav/apidocs/com/sun/codemodel/package-summary.html (2014), "last accessed on 18/08/2014"
2. Apel, S., Kästner, C.: An overview of feature-oriented software development (2009)
3. Apel, S., Lengauer, C., Möller, B., Kästner, C.: An algebra for features and feature composition. In: Proceedings of the 12th International Conference on Algebraic Methodology and Software Technology (AMAST). Lecture Notes in Computer Science, vol. 5140, pp. 36–50. Springer-Verlag, Berlin/Heidelberg, Germany (Jul 2008). https://doi.org/http://dx.doi.org/10.1007/978-3-540-79980-1_4, acceptance rate: 47 % (27/58)
4. Batory, D.: A tutorial on feature oriented programming and the ahead tool suite. In: Proceedings of the 2005 International Conference on Generative and Transformational Techniques in Software Engineering. pp. 3–35. GTTSE'05, Springer-Verlag, Berlin, Heidelberg (2006). https://doi.org/10.1007/11877028_1, http://dx.doi.org/10.1007/11877028_1
5. Bosch, J.: Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (2000)
6. Classen, A., Heymans, P., Schobbens, P.Y.: What's in a feature: A requirements engineering perspective. In: Proceedings of the Theory and Practice of Software, 11th International Conference on Fundamental Approaches to Software Engineering. pp. 16–30. FASE'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg (2008), http://dl.acm.org/citation.cfm?id=1792838.1792841
7. Czarnecki, K., Eisenecker, U.W.: Generative Programming: Methods, Tools, and Applications. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (2000)
8. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: Form: A feature-; oriented reuse method with domain-; specific reference architectures. Annals of Software Engineering **5**(1), 143–168 (1998)
9. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: Featureide: An extensible framework for feature-oriented software development. Science of Computer Programming **79**, 70–85 (2014). https://doi.org/http://dx.doi.org/10.1016/j.scico.2012.06.002
10. Zave, P.: Programming methodology. chap. An Experiment in Feature Engineering, pp. 353–377. Springer-Verlag New York, Inc., New York, NY, USA (2003), http://dl.acm.org/citation.cfm?id=766951.766969
11. Zhang, H., Jarzabek, S.: Xvcl: A mechanism for handling variants in software product lines. Sci. Comput. Program. **53**(3), 381–407 (Dec 2004). https://doi.org/10.1016/j.scico.2003.04.007, http://dx.doi.org/10.1016/j.scico.2003.04.007